

An Overview of Security Support in Named Data Networking

Zhiyi Zhang, Yingdi Yu, Haitao Zhang, Eric Newberry, Spyridon Mastorakis, Yanbiao Li, Alexander Afanasyev, and Lixia Zhang

The authors present an overview of the security mechanisms in the NDN architecture that have been developed over the past several years. NDN changes the network communication model from the delivery of packets to hosts identified by IP addresses to the retrieval of named and secured data packets. Consequently, NDN also fundamentally changes the approaches to network security.

ABSTRACT

This article presents an overview of the security mechanisms in the NDN architecture that have been developed over the past several years. NDN changes the network communication model from the delivery of packets to hosts identified by IP addresses to the retrieval of named and secured data packets. Consequently, NDN also fundamentally changes the approaches to network security. Making named data the centerpiece of the architecture leads to a new security framework that secures data directly, and uses name semantics to enable applications to reason about security and to automate the use of cryptographic keys. In this article, we introduce NDN's approaches to security bootstrapping, data authenticity, confidentiality, and availability.

INTRODUCTION

Named data networking (NDN), a proposed Internet architecture, changes the basic network communication model. Instead of delivering packets to receivers identified by IP addresses, NDN lets consumers request desired data using application-layer names. Naming data enables NDN to secure data directly at the network layer. This is achieved by making the content of every Data packet verifiable and, optionally, confidential.

In this article, we provide an overview of NDN's security framework and illustrate the developed mechanisms with example prototype realizations, showing how all the components in the framework function together. We assume that readers have some basic knowledge of cryptography, but are not necessarily familiar with the NDN architecture.

The article is organized as follows. The following section provides a brief description of the NDN architecture and introduces an example application, which will be used throughout the article to illustrate the use of various security mechanisms. Following that, we state the goals of the NDN security design, identify the major challenges, and introduce the basic supporting components of the solutions. Then we describe the NDN security bootstrapping process, and explain NDN's current solutions to data authenticity,¹ confidentiality, and availability. Throughout this article, we aim to explain how NDN enables data to remain secure independent of underlying communication channels and how it enables

applications to validate received data packets independent of from where they are fetched. Furthermore, we illustrate how applications can utilize name semantics to augment reasoning about which cryptographic keys to use for which content, instead of blindly relying on the "yes-or-no" model provided by third-party certificate services. Then we discuss the basic differences between network security solutions in NDN and TCP/IP that result from the two different network architectures; we also identify remaining issues in NDN's security solution development. The final section concludes the article.

We hope that this article can serve as a guide to NDN security efforts for readers interested in NDN research, as well as a useful demonstration of new approaches to network security that differ from today's common practices.

BACKGROUND

NAMED DATA NETWORKING

From 10,000 feet, one might view the basic idea of NDN as shifting HTTP's request (for a named data object)-and-response (containing the object) semantics at the application layer to the *network layer* [1]. Being a network-layer protocol, NDN's requests/responses work at a network packet granularity — each request, carried in an NDN *Interest* packet containing the name of the requested data, fetches one NDN *Data* packet (Fig. 1); neither type of packet contains an address. Applications that produce data are called *producers*, while those requesting data are called *consumers*.

In addition to being *network layer* packets, NDN *Data* packets also differ from HTTP data objects in two other important ways:

- All NDN *Data* packets are immutable; when a producer changes the content of a *Data* packet, it generates a new packet with a new name to distinguish the different versions of the content.
- Every NDN *Data* packet carries a signature generated using its producer's cryptographic key at the time of data creation, binding its name to its content.

Named, secured data packets provide a basic building block for securing NDN communications.

An NDN network runs routing protocol(s) to propagate the reachability of data names, similar

¹ Data integrity is ensured at the same time as authenticity.

to how IP networks use routing protocols to propagate the reachability of IP addresses. Each NDN router forwards Interest packets according to their names, recording both the interfaces from which Interests are received and the interfaces to which they are forwarded in a pending interest table (PIT). Once an Interest packet reaches a Data packet with a matching name, the Data packet will follow the reverse path of the corresponding Interest to reach the consumer, satisfying the corresponding PIT entry on each router along the way. Data packets can also be cached at routers to serve future requests for the same data. This stateful forwarding plane creates a closed feedback loop, enabling routers to make informed Interest forwarding decisions based on observed performance.

AN EXAMPLE APPLICATION: NDNFit

To aid the reader's comprehension, we use NDNFit [2], a prototype NDN application for tracking and sharing personal fitness activity, as an illustrative example to explain NDN's security mechanisms.² Because NDNFit handles sensitive personal information, it requires strong data authenticity and confidentiality.

As a typical use case, assume that a user "Alice" wants to use NDNFit to record her daily fitness information. Alice runs an app "Sensor" on her mobile phone and an app "Analyzer" on her laptop. "Sensor" collects Alice's daily time-location information, while "Analyzer" produces analytics and visualizations from the data produced by "Sensor." Alice controls the whole system using another app "Owner." Figure 2 shows the data and control flow in NDNFit.

NDNFit requires that all data produced by "Sensor" and "Analyzer" be authenticatable and that any data alterations or data created by unauthorized producers be detectable. Furthermore, to keep her data confidential, Alice only grants "Analyzer" the privilege to access the fitness data produced by "Sensor" — no one else should be able to read this data. We illustrate later how these objectives are achieved via NDN's security mechanisms.

AN OVERVIEW OF THE NDN SECURITY DESIGN

The NDN security framework is built on public-key cryptography. As described previously, NDN secures data directly, enabling applications to achieve data authenticity, confidentiality, and availability independent of underlying communication channels and regardless of whether the data is in transit or at rest (e.g., being cached in the network or stored at end nodes). At the same time, NDN aims to provide highly usable security: to the greatest extent possible, all cryptographic key management and operations should be automated and enforced by the system itself, minimizing the reliance on manual configuration.

In the rest of this article, we call applications and all other communication participants in an NDN network *entities*.³ Each entity owns one or more names. An entity proves its ownership of a name through an NDN certificate, which binds the name and a cryptographic public-private key pair possessed by the entity. We call each certi-

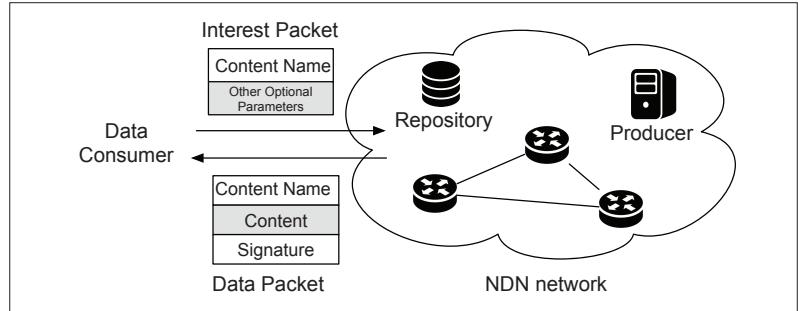


Figure 1. In an NDN network, one Interest packet can fetch one Data packet from its producer, from a data repository, or from a router's cache.

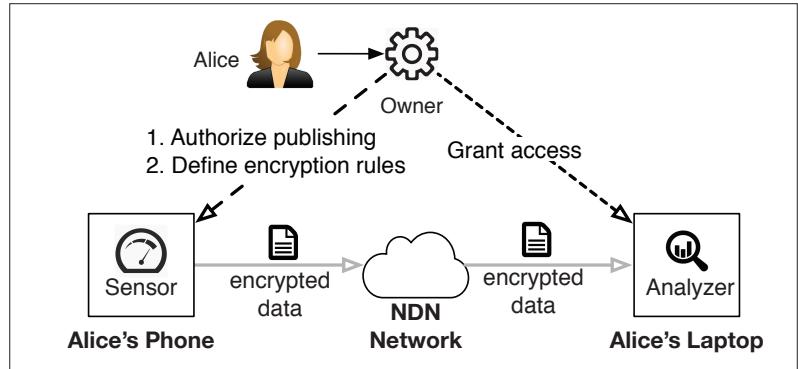


Figure 2. NDNFit application workflow.

fied name an *identity*. Each entity can issue certificates for the sub-namespaces it delegates to other entities.

CHALLENGES AND OVERVIEW OF SOLUTIONS

Utilizing public key cryptography to validate communications requires NDN to address the following three challenges.

Establishing Trust Anchor(s): All cryptographic verifications must terminate at a pre-established trust anchor. After a trust anchor is installed, an entity can verify other entities' signatures by verifying their certificates along the certificate chain to the trust anchor.⁴ Trust anchors are usually installed via out-of-band mechanisms, and the development of these supporting mechanisms depends on the trust anchor model in use. In today's practice, trust anchors are commonly established via the following means:

- Obtaining certificates from commercial certificate authorities (e.g., TLS certificates)
- Installing a single global trust anchor (e.g., DNSSEC)
- Establishing trust in an ad hoc manner (e.g., Trust-On-First-Use, Web-Of-Trust).

NDN utilizes a different trust anchor model. NDN assumes that the authority of each networked system (e.g., an organization, a smart home, or a cloud service provider) establishes its own trust anchor(s) and that all the entities under that authority can discover these trust anchors through local system settings. This trust model resembles that of the Simple Distributed Security Infrastructure (SDSI/SPKI) [4] in trust anchor establishment.

Providing Effective Solutions for Trust Management: Effective solutions must enable applications to express their own trust policies, and the system must be able to execute these policies automatically. In NDN, entities are able to obtain

² The NDNFit use case described in this article is a simplified version of the actual implementation.

³ An entity can be an administrative unit (e.g., a country, a university, a company), a home, a user, a node, or an app process. The task of allocating names to entities is beyond the scope of the NDN design, just like the task of assigning IP addresses is beyond the scope of the TCP/IP design.

⁴ An alternative is to establish trust via a web of trust as described in [3].

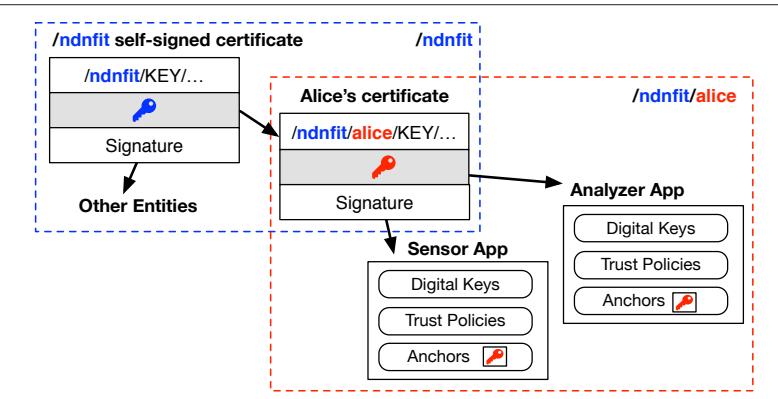


Figure 3. The cryptographic relationship between the namespaces /ndnfit and /ndnfit/alice, and between /ndnfit/alice and its sub-namespaces.

NDN certificates and learn trust policies from trustworthy parties. A certificate enables an entity to generate verifiable signatures for its data and build trust relationships with other entities. The trust policies inform each entity which keys, for a given name or name prefix, should be used for signature generation and verification. As we will describe later, NDN can express users and applications' trust policies by defining the relationships between data names and signing key names.

Providing Usable Key Management Solutions: Signing, verification, encryption, and decryption involve the use of cryptographic keys, requiring mechanisms to assign and deliver the correct keys or certificates to the parties in need in an automatic manner. Taking advantage of its structured, semantically meaningful data names, NDN enables application developers to define naming conventions to systematically construct the names of the cryptographic keys/certificates used for signing, verification, encryption, and decryption. As we explain later, these naming conventions in turn enable individual entities to automatically construct the names of the required cryptographic keys for a given data name and to fetch keys, improving the usability of key management (certificate issuance, certificate provisioning, etc.).

BASIC COMPONENTS OF NDN SECURITY

The NDN security framework makes use of the following three basic components.

Digital Keys: NDN treats cryptographic keys in the same way as any other named data, allowing them to be retrieved using Interest-Data exchanges at the network layer.

Certificates: An NDN certificate represents its issuer's endorsement of the binding between the name and the public key; note that the name of the key is not necessarily under the issuer's namespace, for example, in a web of trust system [3]. A certificate is a Data packet carrying a public key and can be fetched like any other Data packet. The issuer will put its signing key name with other auxiliary information into Data's signature info field. Certificate names follow the naming convention "/<prefix>/KEY/<key-id>/<issuer-info>/<version>," where the "prefix" is the name to which the key is bound, and the components after "KEY" are the key id, the issuer-specified information, and the certificate version number. For example, a certificate name

/ndnfit/alice/KEY/001/N-testbed/002 indicates that:

- The certificate owner is /ndnfit/alice.
- The certified key has the id 001, which identifies an instance of Alice's public key.
- The certificate signer sets the issuer information to N-testbed, which indicates that it is an NDN testbed-issued certificate.
- The certificate version is 002.

Trust Policies: Applications define trust policies that specify which entities are trusted for producing which piece of data, and which key should be used for which data namespace and for what purpose. For example, a trust policy can require that the key used to authenticate data must not be used to sign encryption keys.

The above three basic components are used in the security mechanisms described below. The next section shows how an entity can obtain these three components from the security bootstrapping process.

SECURITY BOOTSTRAPPING IN NDN

Security bootstrapping is the process through which an entity obtains its trust anchor and certificate, and learns trust policies. The NDN-Fit example described earlier must go through security bootstrapping to be properly initialized. In this example, since Alice is the owner of her devices and data, Alice's certificate is set to be the trust anchor. In this article, we assume that Alice's certificate has the name /ndnfit/alice/KEY/001/N-testbed/002, whose meaning is explained above.

OBTAINING TRUST ANCHORS

An entity needs trust anchors to verify other entities' authenticity. Trust anchors are expected to be either pre-configured or securely obtained through some out-of-band means. Following the SDSI model, the NDN security design assumes that different systems establish their own trust anchors and that entities within those systems decide or develop their own means to obtain trust anchors.

In our NDNFit example, we take a simple approach of manually installing Alice's certificate into the "Owner," "Sensor," and "Analyzer" applications.

OBTAINING CERTIFICATES

To generate Data packets with valid names and verifiable signatures, a (producer) application must first obtain a name and a certificate that certifies its ownership of that name. Consumer applications do not need to obtain identity certificates for Data consumption, although they must obtain trust anchors for data verification. Once the trust anchor is obtained, an entity can identify a trustworthy certificate signer by checking its certificate (e.g., a signer's certificate is the trust anchor or endorsed by the trust anchor), then request a certificate for itself. NDN security offers flexibility to application developers in deciding how to obtain certificates. Depending on the system design, a cloud-based application may obtain its certificate from a centralized certificate service, while a distributed application (e.g., P2P applications) may obtain the certificate from its users. We have developed the NDN certificate management sys-

tem (NDNCERT) [5] to process such certificate requests automatically.

In our NDNFit use case, the trust anchor, Alice's certificate, resides in the Owner application on her laptop. Owner plays the role of the certificate signer by invoking application programming interfaces (APIs) provided by NDNCERT. Sensor and Analyzer are instructed to request certificates from Owner using the NDNCERT protocol, and Owner can approve the two apps using customized out-of-band challenges (e.g., Alice may manually check the application's PIN code and approve the corresponding certificate request). Two certificates, `/ndnfit/alice/sensor/KEY/...` and `/ndnfit/alice/analyz-er/KEY/...`, are then issued to the Sensor and Analyzer apps, respectively.

LEARNING TRUST POLICIES

To determine which cryptographic key is legitimate to sign which Data packet when producing new data or verifying received data, an application needs to obtain trust policies after obtaining the trust anchor. As we explain below, NDN apps can define their trust policies using a *trust schema*, which is simply a piece of named content that can be retrieved like any other content. After obtaining the trust anchor, an application can fetch and verify the trust policies from trusted sources. Note that there must be a preconfigured default trust policy, which can be used to validate the Data packets carrying trust policies. A simple default policy may define that Data packets carrying trust policies must be directly signed by a trust anchor with a given name.

In our NDNFit example, Alice can configure the trust policies through the Owner user interface; then Owner produces trust policy Data packets and signs them with Alice's private key. During security bootstrapping, Analyzer and Sensor fetch the trust policy Data packets, verify them using the trust anchor (Alice's certificate), and then save the policies for future use. As shown in Fig. 3, after security bootstrapping, both Sensor and Analyzer will trust Owner, and each will have their own trust policy and certificate under the namespace `/ndnfit/alice`.

The security bootstrapping of Alice's own certificate takes place in a different network system where the trust anchor is `/ndnfit/KEY/...`. Alice learns of this trust anchor and obtains the certificate `/ndnfit/alice/KEY/...` from the authority of the namespace `/ndnfit` via some means defined by NDNFit (we omit the details of this process due to space limitations).

DATA AUTHENTICITY

In this section, we show how NDN security helps ensure data authenticity automatically. To enable this supporting function, users must first define their data acceptance policies.

After obtaining their certificates, the apps Sensor and Analyzer can produce Data packets under their corresponding namespaces and sign them using their corresponding private keys, enabling consumers to authenticate the received Data packets by verifying their signatures. NDN's rich name semantics enable applications to use names to reason about trust and define trust policies. Trust policies help consumers validate a

received Data packet by checking whether the packet is signed by the correct key according to the policies. In this way, trust policies limit the power of each signing key to Data packets with specific names, supporting data authenticity at a fine granularity. For instance, in our example, the key `/ndnfit/alice/sensor/KEY/...` can be limited to sign packets under the prefix `/ndnfit/alice/sensor` only.

The authenticity and integrity of received Data packets (some of them may be certificates) are determined by a combination of the following two factors.

Validation by Trust Policies: Structured data names and key names provide explicit and meaningful contexts for applications, enabling NDN applications to define rules to only accept packets signed by keys with specific names. More specifically, the data name, the signing key name, the relationship between the key name and Data name, and the trust anchor name must follow application-defined rules. We have developed a solution, called *trust schema* [6], to let users and applications express their trust policies in a form that can be directly executed by applications.

Signature Verification: To verify the signature in a received Data packet, a consumer retrieves the certificate of its producer (identified by the key name in a dedicated section of the packet). This certificate recursively points to its signer's certificate and finally arrives at a specified trust anchor. The received data packet is considered valid only if all the certificates in the above chain have valid signatures and satisfy the trust policies.

USING TRUST SCHEMAS TO DEFINE TRUST POLICIES

Trust schemas make use of NDN's naming conventions to enable systematic descriptions of trust policies, namely: how Data packet names should be structured, how packet signing key names should be structured, how the components in a Data packet name should be related to those in its signing key name, and which trust anchor is acceptable.

Upon receiving a Data packet, a consumer application first uses its trust schemas to assess the packet's trustworthiness by examining its certificate chain to the trust anchor – this takes place before any cryptographic signature verification is performed. For instance, as shown in Fig. 4, in addition to Alice (`/ndnfit/alice`), a user named Bob (`/ndnfit/bob`) is also running an NDNFit system. We assume that both Alice's certificates and Bob's certificates are signed by the same trust anchor in the `/ndnfit` namespace. Alice's devices and Bob's devices produce Data packets under their own prefixes, namely `/ndnfit/alice/sensor/example` and `/ndnfit/bob/sensor/example`. Figure 4 shows that there are two trust schemas. Schema "rule 1" accepts Data packets whose name prefix is `/ndnfit/alice`, signing key name prefix is `/ndnfit/alice/KEY`, and certificate chain ends with the trust anchor `/ndnfit/alice`. Accordingly, only packets signed by Alice and strictly under Alice's prefix are accepted. "Rule 2" has a looser requirement: all data packets with the name and key name prefix `/ndnfit` and a certificate chain eventually tracing to the anchor `/ndnfit` can be accepted. Consequently, rule 2 accepts packets

Trust schemas make use of NDN's naming conventions to enable systematic descriptions of trust policies, namely: how Data packet names should be structured, how packet signing key names should be structured, how the components in a Data packet name should be related to those in its signing key name, and which trust anchor is acceptable.

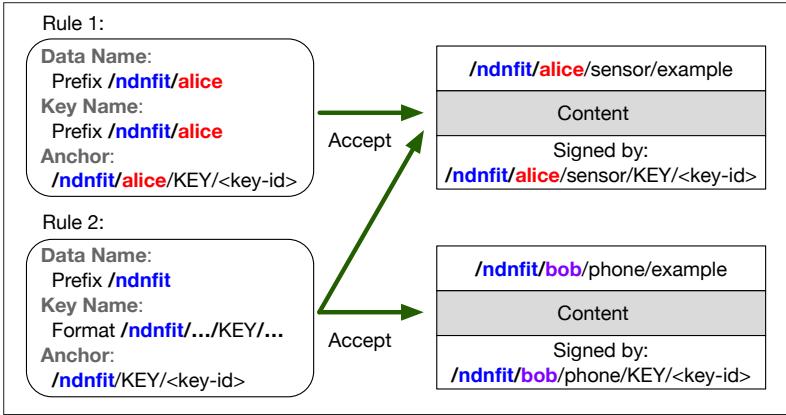


Figure 4. Different trust schemas leading to different authentication result.

produced by either Alice or Bob's devices.

SIGNED INTERESTS

Although Interest packets are not signed by default, an Interest can and should be signed when its use case requires authenticity. For example, in an IoT scenario, when a received Interest packet contains a command, a smart home device needs to authenticate the sender of the Interest before executing the command. Signed Interests enable a controller to actuate IoT devices. The Interest signature validation process is the same as the one used to validate Data packets.

DATA CONFIDENTIALITY

NDNFit requires data confidentiality and access control support to protect sensitive user information. NDN's basic approach to data confidentiality is to use encryption. The Diffie-Hellman key exchange protocol [7] is widely used to automatically derive encryption keys for point-to-point sessions. However, Diffie-Hellman does not apply to constructing encryption keys for multi-party communications, as is the case for NDN applications in general. By taking advantage of structured names that can convey rich semantics, we have developed named-based access control (NAC) and its enhancement with attribute-based encryption (NAC-ABE) [8]. NAC/NAC-ABE automates the key distribution process for both point-to-point and multi-party applications. A schematized access control solution [9] has also been proposed to further systemize key management for access control in NDN networks.

NAME-BASED ACCESS CONTROL

To grant access rights, NAC uses an "access manager" (e.g., an "Owner" app) entity that publishes granular per-namespace access policies in the form of key encryption keys (KEKs, plaintext public keys) and key decryption keys (KDKs, encrypted private keys). NAC explicitly appends an encryption key name to the KDK name with a separator **ENCRYPTED-BY** component; thus, consumers can learn the key names after fetching the encrypted Data packet.

In our NDNFit example, Alice is the owner of all the Data packets produced under the prefix `/ndnfit/alice`. Alice grants access rights to Analyzer to read the data under the prefix `/ndnfit/alice/sensor` produced by the "Sensor" app.

Key Generation: To grant data access to Analyzer, Owner generates a new pair of keys and produces two Data packets: a KEK packet carrying the KEK in plaintext with the name `/ndnfit/alice/NAC/sensor/KEK/<Key-id>` and a KDK packet with the name `/ndnfit/alice/NAC/sensor/KDK/<Key-id>/ENCRYPTED-BY/ndnfit/alice/analyzer/KEY/<Analyzer-Key-id>`, which contains the KDK encrypted using Analyzer's public key.

Data Production: When producing data, Sensor first generates a symmetric content key (CK) for content encryption. Then it encrypts the content with the CK and packs the encrypted content with the name of the corresponding CK, `/ndnfit/alice/sensor/CK/<CK-id>` (in plaintext) into the Data packet named `/ndnfit/alice/sensor/example`. Finally, it fetches the KEK `/ndnfit/alice/NAC/sensor/KEK/<Key-id>` and uses it to encrypt the CK, and publishes this encrypted CK by putting it into another Data packet with the name `/ndnfit/alice/sensor/CK/<CK-id>/ENCRYPTED-BY/ndnfit/alice/NAC/sensor/KEK/<Key-id>`.

Data Consumption: As shown in Fig. 5, Analyzer first fetches the Data packet produced by Sensor, and the returned Data packet conveys that its content was encrypted using the CK. Analyzer extracts the CK name from the content and automatically generates an Interest to fetch the corresponding CK. To decrypt the CK with the corresponding KDK, the consumer follows the naming convention and combines the KEK name extracted from the CK Data name with its own identity to construct the Interest `/ndnfit/alice/NAC/sensor/KDK/<Key-id>/ENCRYPTED-BY/ndnfit/alice/analyzer` and fetches the KDK. Since the fetched KDK is encrypted using Analyzer's key, Analyzer can decrypt the content and get the KDK, use the KDK to decrypt the CK, and finally decrypt the content with the CK.

ACCESS CONTROL GRANULARITY

To control access control granularity, NAC leverages the structured namespace of NDN. For example, the above mentioned policy to give access to the sensor data can add the suffix `step/8am/10am` to the policy namespace (`/ndnfit/alice/NAC/sensor/step/8am/10am`), which will restrict access to only the steps data and only during the specified time interval.

DATA AND CERTIFICATE AVAILABILITY

IMPROVING DATA AVAILABILITY VIA IN-NETWORK STORAGE

Because NDN secures data directly, Data packets can be retrieved from anywhere, including router caches and other storage systems, regardless of whether these cache or storage systems are trustworthy. All forwarders may cache passing Data packets to satisfy future Interests.

CERTIFICATE AVAILABILITY

NDN certificates are carried in Data packets, enabling them to benefit from in-network storage. To further improve the availability of certificates, we also developed the NDN certificate bundle [10] to allow each producer to collect all the cer-

tificates in the certificate chain needed to verify its data and bundle them together, making the whole certificate chain available to consumers in a single package.

In the NDNFit example, the producer, Sensor, would combine the certificates needed to verify its data in a certificate bundle. Specifically, the bundle would contain the application certificate (`/ndnfit/alice/sensor/KEY/...`) and the trust anchor certificate (`/ndnfit/alice/KEY/...`). When a consumer application needs to verify the retrieved data, it can fetch all the needed certificates directly from Sensor.

DISCUSSION

COMPARISON OF NDN AND TCP/IP SECURITY

There are two fundamental differences between the NDN and TCP/IP security solutions which originate from the fact that NDN names data, whereas IP names hosts.

Securing Data vs Securing Channels: In TCP/IP, the basic communication unit is a channel between two processes. Consequently, protocols such as IPSec and TLS secure communication channels (e.g., IP channels or TCP channels). However, data delivered through protected communication channels does not directly translate to data authenticity – the data could have been altered before entering the channel and loses cryptographic protection as soon as it leaves the channel; and when multiple parties communicate, securing the channel between every pair of endpoints can quickly lead to scalability and manageability issues. In contrast, NDN secures data directly, removing any reliance on the security of intermediate communication channels, allowing applications to protect what really matters to them – the data itself.

Establishing Trust Using Name Semantics: Existing certificate authentication solutions lack the means to effectively reason about trust. For instance, current secure communication protocols (e.g., HTTPS, QUIC) follow a common practice of accepting a signature if it is (in)directly signed by a pre-installed certificate authority. However, [11] shows that commercial certificate authorities themselves may not be reliable, and signature verification alone is not enough to establish trust. NDN takes a fundamentally different approach to trust establishment. In NDN, entities may utilize local authorities instead of commercial certificate authorities as trust anchors; trust policies are expressed explicitly through the relationships between semantically meaningful names in a systematic way, allowing applications to reason about security rather than blindly accepting signatures; and naming conventions help facilitate automated key management, thus improving system usability.

REMAINING CHALLENGES

The development of the NDN architecture has guided the creation of a new network security framework, which brings both new opportunities and new challenges [12]. Regarding user privacy [13], on one hand, Interest packets carry requested data names only, without disclosing the consumer's information; on the other hand, Data packet names and signatures may disclose a producer's identity if they are not properly pro-

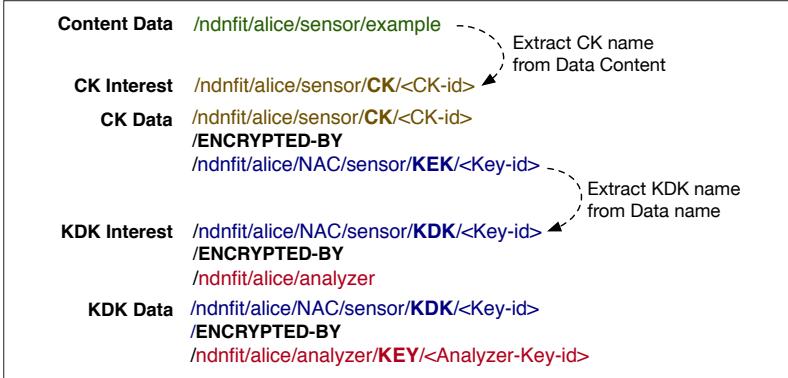


Figure 5. The naming convention used in name-based access control.

ected. Additionally, both the content store and the pending interest table in an NDN router may potentially increase the attack surface [14]. The NDN research community is actively investigating mitigations to these challenges.

CONCLUSION

In [15], we argued that, by naming and securing data directly, NDN offers intrinsic advantages for securing network communications. Evidence from our efforts developing NDN security solutions suggest that this is indeed true. Named, secured Data packets (which can also carry certificates and trust schemas) can easily be fetched from anywhere, serving as a powerful building block for security solution development. Furthermore, we have learned that one can establish well-defined naming conventions to define trust policies in a systematic way, as well as to enable name-based access control via encryption. We also learned, the hard way, the importance of automating security operations instead of leaving the problem to application developers (who would simply make applications work first by leaving security out).

Consequently, NDN secures network communications in a more resilient, intuitive, and less fragmented manner than the solutions in today's TCP/IP networks. The development process of the NDN security model has convinced us that, by building a network architecture based on named data, we can effectively develop exciting new network security solutions.

ACKNOWLEDGMENT

This work was partially supported by the National Science Foundation under awards CNS-1345142, CNS-1345318, CNS-1629009, and CNS-1629922.

REFERENCES

- [1] L. Zhang et al., "Named Data Networking," *ACM SIGCOMM Comp. Commun. Review*, 2014.
- [2] H. Zhang et al., "Sharing mHealth Data via Named Data Networking," *ICN*, 2016, pp. 142–47.
- [3] Y. Yu et al., "An Endorsement-Based Key Management System for Decentralized NDN Chat Application," *NDN*, Tech. Rep. NDN-0023, July 2014; <http://named-data.net/publications/techreports/>.
- [4] R. L. Rivest and B. Lampson, "SDSI – A Simple Distributed Security Infrastructure," *Crypto*, 1996.
- [5] Z. Zhang, A. Afanasyev, and L. Zhang, "NDNCERT: Universal Usable Trust Management for NDN," *Proc. 4th ACM Conf. Information-Centric Networking*, 2017, pp. 178–79.
- [6] Y. Yu et al., "Schematizing Trust in Named Data Networking," *Proc. 2nd ACM Int'l. Conf. Information-Centric Networking*, 2015, pp. 177–86.
- [7] M. Mosko, E. Uzun, and C. A. Wood, "Mobile Sessions in Contentcentric Networks," *IFIP Networking*, 2017.

- [8] Z. Zhang et al., "NAC: Automating Access Control via Named Data," IEEE MILCOM, 2018.
- [9] C. Marxer and C. Tschudin, "Schematized Access Control for Data Cubes and Trees," *Proc. ACM Conf. Information-Centric Networking*, 2017.
- [10] M. Mittal, A. Afanasyev, and L. Zhang, "NDN Certificate Bundle," NDN, Tech. Rep. NDN-0054, 2017.
- [11] C. Cimpanu, "14,766 Let's Encrypt SSL Certificates Issued to PayPal Phishing Sites," posted 24 Mar. 2017; <https://www.bleepingcomputer.com/news/security/14-766-lets-encrypt-ssl-certificates-issued-to-paypal-phishing-sites/>.
- [12] R. Tourani et al., "Security, Privacy, and Access Control in Information-Centric Networking: A Survey," *IEEE Commun. Surveys & Tutorials*, 2017.
- [13] C. Ghali, G. Tsudik, and C. A. Wood, "When Encryption Is Not Enough: Privacy Attacks in Content-Centric Networking," *Proc. 4th ACM Conf. Information-Centric Networking*, 2017, pp. 1–10.
- [14] C. Ghali et al., "Closing the Floodgate with Stateless Content-Centric Networking," *Proc. 2017 IEEE 26th Int'l. Conf. Computer Communication and Networks*, 2017, pp. 1–10.
- [15] L. Zhang et al., "Named Data Networking (NDN) Project," NDN Tech. Rep. NDN-0001, Oct. 2010.

BIOGRAPHIES

ZHIYI ZHANG (zhiyi@cs.ucla.edu) is a Ph.D. candidate in the Computer Science Department at the University of California Los Angeles (UCLA). He received his B.S. in computer science from Nankai University, China. His research focus is on network security, the Internet of Things, and information-centric networking. He has done research in named data networking (NDN) security, including name-based access control, certificate management, and DDoS defense.

YINGDI YU (yingdi@cs.ucla.edu) received his Ph.D. in computer science from UCLA. His research interest is focused on distributed systems and network security. He is currently working as a research scientist at Facebook.

HAITAO ZHANG (haitao@cs.ucla.edu) received his B.E and M.S degrees from the Electronic Engineering Department at Tsing-

hua University in 2011 and 2013. He received his Ph.D. degree from UCLA in 2018. His research interests are Internet architecture and protocols. During his Ph.D. study, he worked on the NDN project. He is now a software engineer at Uber Technologies, Inc.

ERIC NEWBERRY (emnewber@umich.edu) received his B.S. degree in computer science from the University of Arizona in 2018. He is currently a Ph.D. student in computer science and engineering at the University of Michigan. His research interests include computer networks, edge computing, and network security.

SPYRIDON MASTORAKIS (mastorakis@cs.ucla.edu) is a Ph.D. candidate in the Computer Science Department at UCLA. He holds an M.S. degree in computer science from UCLA and a B.S. degree in electrical and computer engineering from the National Technical University of Athens.

YANBIAO LI (lybmath@cs.ucla.edu) is currently a postdoctoral scholar at UCLA. He received his Ph.D. degree in computer science from Hunan University, China, in 2016. His research interests focus on the future Internet architecture, edge computing and the Internet of Things.

ALEXANDER AFANASYEV (aa@cs.fiu.edu) is an assistant professor in the School of Computing and Information Sciences at Florida International University. He received his B.S. and M.S. degrees in computer engineering from Bauman Moscow State Technical University, Russia, and M.S. and Ph.D. degrees in computer science from UCLA. His research focus is on the next-generation Internet architecture as part of the NDN project, and he has done research in multiple fields vital for the success of NDN.

LIXIA ZHANG [F] (lixia@cs.ucla.edu) is a professor in the Computer Science Department at UCLA. She received her Ph.D. from MIT and was a member of the research staff at Xerox PARC before joining UCLA. She is a fellow of ACM, the recipient of an IEEE Internet Award, and the holder of the UCLA Postel Chair in Computer Science. Since 2010 she has been leading the effort on the design and development of Named Data Networking, a new Internet protocol architecture (<http://named-data.net/>).