An Overview of Security Support in Named Data Networking

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

***Abstract*—This paper presents an overview of the security mechanisms in the Named Data Networking (NDN) architecture that have been developed over the past several years. NDN changes the network communication model from the delivery of packets between 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: (i) secures data directly, and (ii) uses name semantics to enable applications to reason about security, and to automate the use of cryptographic keys. In this paper, we introduce NDN's approach to security bootstrapping, data authenticity, integrity, confidentiality, and availability.**

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

I. 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 network layer. This is done by making every Data packet verifiable and, optionally, confidential.

In this paper, 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 is not necessarily familiar with the NDN architecture.

The paper is organized as follows. Section II provides a brief description of the NDN architecture and introduces an example application, which will be used throughout the paper to illustrate the use of various security mechanisms. Section III states the goals of the NDN security design, identifies the major challenges, and introduces the basic supporting components of the solutions. Section IV describes the NDN security bootstrapping process, and Sections V, VI, and VII explain NDN's current solutions to data authenticity, integrity, confidentiality, and availability. Throughout this paper, we aim to explain how NDN enables data to remain secure independent of any underlying communication channel, and how it enables applications to validate received data packets independent of from where they were fetched. Moreover, we illustrate how applications can utilize name semantics to augment the reasoning about which cryptographic keys to use, instead of blindly relying on the "yes-or-no" model provided by third-party certificate services. Section VIII discusses the basic differences between network security solutions in TCP/IP and NDN that result from the two different network architectures; it also identifies remaining issues in NDN's security solutions. Section IX concludes the paper.

We hope that this paper 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.

Zhiyi Zhang, Yingdi Yu, Haitao Zhang, Spyridon Mastorakis, Yanbiao Li, and Lixia Zhang are with the Department of Computer Science, UCLA - e-mail: zhiyi, yingdi, haitao, mastorakis, lybmath, lixia@cs.ucla.edu.

Eric Newberry was with the Department of Computer Science, the University of Arizona at the time of submission. He is now with the Department of Electrical Engineering and Computer Science at the University of Michigan - e-mail: emnewber@umich.edu

Alexander Afanasyev is with the Department of Computer Science, Florida International University - e-mail: aa@cs.fiu.edu

II. Background

A. Named Data Networking

From 10,000 feet, one could view the basic idea of NDN as shifting HTTP's request (for a named data object)-and-response (containing the object) semantics 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 (Figure 1); neither type of packets contains an address. Applications that produce data are called *producers*, while those requesting data are called *consumers*.

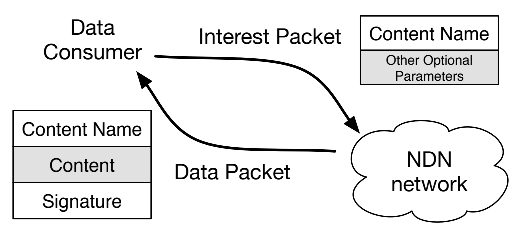

Fig. 1. Interest and Data packet in NDN

In addition to being *network layer* packets, NDN Data packets also differ from HTTP data objects in two other important ways: (i) all NDN Data packets are immutable; when producers change the content of a Data packet, they generate new a packet with a new name to distinguish the different version of the content; and (ii) 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.

Regarding the routing and forwarding of NDN, generally speaking, an NDN network runs routing protocol(s) to propagate the reachability of data name prefixes, similar to how IP networks use routing protocols to propagate IP address prefixes. 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 [2] plane creates a closed feedback loop, enabling routers to make informed Interest forwarding decisions based on observed performance.

B. An Example Application: NDNFit

To aid the reader’s comprehension, we use NDNFit [3], a prototype NDN application for tracking and sharing personal fitness activity, as an illustrative example to explain NDN’s security mechanisms. [1] Because NDNFit handles sensitive personal information, it requires strong data authenticity and confidentiality.

As a typical use case, assume that a data owner “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 also runs an app “Owner” to control the whole system. Figure 2 shows the data and control flow in NDNFit.

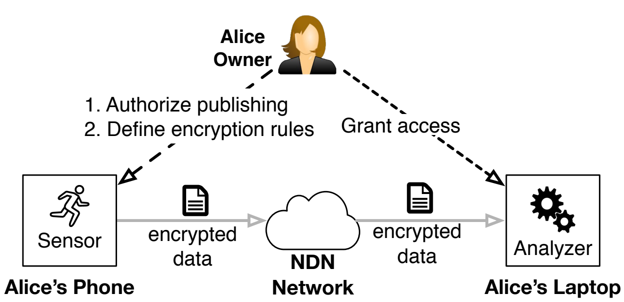

Fig. 2. NDNFit application workflow

To support data authenticity and integrity, NDNFit requires that all data produced by “Sensor” and “Analyzer” be authenticatable, that any data alterations be detectable, and that any fake data created by unauthorized entities be dropped. 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 in Sections IV ~ VII how these objectives can be achieved via NDN’s security mechanisms.

III. An Overview of the NDN Security Design

The NDN security framework is built on public-key cryptography. As described in Section II, NDN secures data directly, enabling applications to achieve data authenticity, integrity, confidentiality, and availability independent of the underlying communication channel 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, as well as automatically enforced by the system itself, minimizing the reliance on manual configuration.

In the rest of this paper, we call applications and all other communication participants in an NDN network *entities*.[2] Each entity possesses one or more names and one or more cryptographic public-private key pairs. An NDN certificate for an entity binds its name and key(s) together by certifying the entity’s ownership of the name and its key(s) – we call each certified name an *identity*. Each entity can also issue certificates for the sub-namespaces it delegates to other entities.

A. Challenges and Solution Directions

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. [3] 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: (i) purchasing certificates from commercial certificate authorities (e.g., TLS certificates used to secure communications with a website), (ii) installing a single global trust anchor (e.g., DNSSEC), or (iii) establishing trust in an ad-hoc manner (e.g., trust on first use, or “TOFU”). NDN utilizes a different trust anchor model. NDN assumes that the authority of each networked system (an organization, a smart home, etc.) 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) [5] in trust anchor establishment.

Providing effective solutions for trust management Effective solutions must enable applications to express their own trust policies and execute these policies automatically. In NDN, entities are able to obtain 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/name prefix, should be used for signature generation and verification. As we will describe in Section V-A, NDN can express users

[1]The NDNFit use case described in this paper is a simplified version of the actual implementation.

[2]An entity can be any administrative unit (such as a country, a university, a company), a neighborhood, 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.

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

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 in a secure, efficient, and *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. 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 said keys, as we explain in Sections V and VI.

B. Basic Components of NDN Security

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

1) 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.

2) Certificates: An NDN certificate signer either signs NDN certificates under its own namespace or signs keys under different namespaces as an endorsement (e.g. in a web of trust [4]). A certificate is also a Data packet that carries public key information and can be fetched like any other data. Certificate names follow the naming convention "`/<prefix>/KEY/<key-id>/<issuer-info>/<version>`", where the "prefix" component represents the owner of the certificate, and the components after "KEY" are the key id, the certificate issuer information, and the certificate version number. For example, a certificate name "`/ndnfit/alice/KEY/001/002/003`" indicates that (i) the certificate owner is "`/ndnfit/alice`"; (ii) the certified key has the id "`001`"; (iii) the certificate signer set the issuer information to "`002`", which could be the signer key's id or some other information defined by the signer; and (iv) the certificate version is "`003`".

3) Trust Policies: Applications define trust policies which specify which entities are trusted for performing what actions, and which key should be used for which data namespace and purpose.

The above three basic components are used in the security mechanisms described in Sections V $\sim$ VII. The next section shows how an entity can obtain these three components from the security bootstrapping process.

IV. Security Bootstrapping in NDN

Security bootstrapping is the process through which an entity obtains its trust anchor and certificate, and learns trust policies. The NDNFit example described in Section II-B 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 paper, we assume that Alice's certificate has a name "`/ndnfit/alice/KEY/001/002/003`", whose meaning is explained in Section III-B.

A. Obtaining Trust Anchors

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

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

B. 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. NDN security offers flexibility to application developers in deciding how to obtain trust anchors. Depending on the system design, an application may obtain certificates from a centralized certificate service (e.g., in cloud-based applications), while a distributed application (e.g., in p2p applications) may obtain the trust anchor certificates from its user.

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. We have developed the NDN certificate management system (NDNCERT) [6] to process such certificate requests automatically.

In our NDNFit use case, the trust anchor, Alice's certificate, resides in an NDNCERT daemon (called an "agent") on her laptop. This agent plays the role of the certificate signer. "Sensor" and "Analyzer" use the NDNCERT protocol to request certificates from this agent, and the agent 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/analyzer/KEY/...`", will be issued to the "Sensor" and "Analyzer" apps, respectively.

C. Learning Trust Policies

To determine which cryptographic key should be used to sign which Data packet, an application needs to obtain and install trust policies after obtaining the trust anchor. In NDN, one's trust polices can be written as a piece of named data that can be retrieved like any other NDN Data packet. After obtaining the trust anchor, an application can fetch and verify the trust polices from trusted sources (e.g., a cloud-based application can learn policies from its central server). Note that there must exist a preconfigured default trust policy, which can be used to validate the Data packets carrying trust policies. A simple default policy could direct 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 "Owner"'s user interfaces – "Owner" can

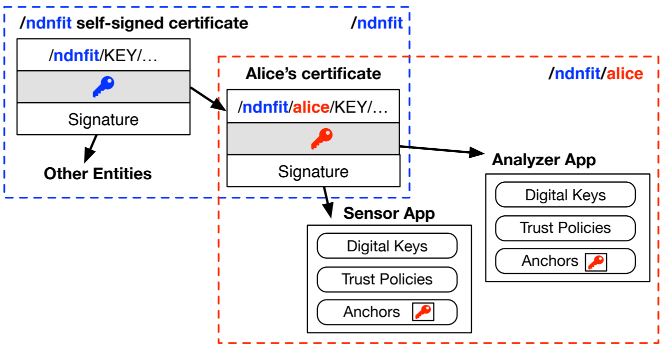

Fig. 3. The cryptographic relationship between the namespaces /ndnfit and /ndnfit/alice, as well as between /ndnfit/alice and its sub-namespaces.

then generate trust policy Data packets. These policy data packets will be signed by Alice's private key. During security bootstrapping, "Analyzer" and "Sensor" fetch the trust policy Data packets and, after verifying the trust policies with the trust anchor (Alice's certificate), the applications can install the policies. As shown in Figure 3, after security bootstrapping, both "Sensor" and "Analyzer" will trust "Owner" and will each have their own trust policies and certificate under "`/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`" (we omit the details of this process here due to the paper length limit).

V. Authenticity and Integrity

In this section, we show how NDN security helps to ensure data authenticity and integrity in an automatic manner. 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 check data authenticity and integrity by verifying the signatures of received Data packets. More importantly, 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 packet by checking whether the piece of data is signed by the right key according to the policies. In this way, trust policies limit the power of each signing key to data with specific names, supporting data authenticity at a fine granularity. For instance, in NDNFit, the key certified in certificate "`/ndnfit/alice/sensor/KEY/...`" is only allowed to sign packets under the prefix "`/ndnfit/alice/sensor`".

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

1) Validation by Trust Polices: Structured data names and key names provide explicit and meaningful contexts for applications, enabling NDN applications to define rules that only accept packets signed by the keys with specific names. More specifically, (i) the data name, (ii) the signing key name, (iii) the relationship between the key name and data name, and (iv) the trust anchor name must follow application-defined rules. We have developed a solution, called *trust schema* [7], to let users and applications express their trust policies in a form that can be directly executed by applications (see Section V-A).

2) Signature Verification: To verify the signature in a Data packet, consumers retrieve the certificate of its producer (identified by the key name in a dedicated section of the Data packet). This certificate recursively points to its signer's certificate and finally arrives at a known 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 of the consumer.

A. Using Trust Schemas to Define Trust Policies

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

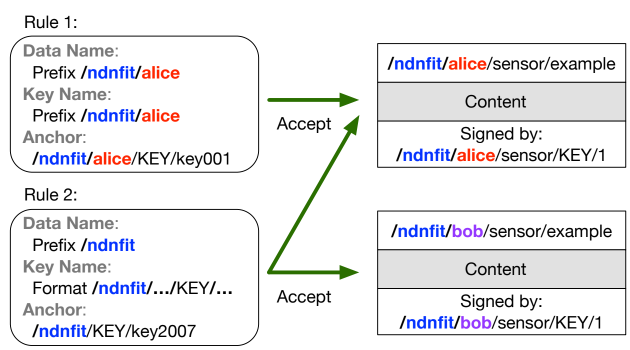

Fig. 4. An example of Trust Schema

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 Figure 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 (i) name prefix is "`/ndnfit/alice`", (ii) signing key name prefix is "`/ndnfit/alice/KEY`", and (iii) certificate chain ends with the trust anchor "`/ndnfit/alice`". Accordingly, only packets signed by Alice and strictly under Alice's prefix are accepted. However, "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. As a consequence, "rule 2" accepts packets produced by either Alice's devices or Bob's devices.

B. Signed Interests

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

VI. Data Confidentiality

NDNFit requires data confidentiality and access control support to protect sensitive user information. NDN's basic approach to data confidentiality is encryption, which requires an automated key management system to enable involved entities to securely fetch the needed encryption and decryption keys.

For point-to-point sessions, key exchange protocols like Diffie-Hellman [8] can derive encryption keys for the session. However, Diffie-Hellman does not apply to constructing encryption keys for multi-party communications, as is the case for NDNFit. 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) [9], which can automate the key distribution process for both point-to-point and multi-party applications. A schematized access control solution [10] has also been proposed to further systemize key management for access control. Below, we use NAC to illustrate the use of naming conventions to automate key management.

A. Name-based Access Control

NDNFit uses NAC to achieve data confidentiality and access control. In our example, Alice is the owner of all Data packets produced under the prefix "`/ndnfit/alice`" and determines who can access her confidential data. NAC explicitly appends each encryption key name to the name of the corresponding Data packet. For instance, a Data packet produced by "Sensor" has the name "`/ndnfit/alice/sensor/example/ENCRYPTED-BY/ndnfit/alice/sensor/CK/001`", where the components after "`ENCRYPTED-BY`" name the encryption key. As mentioned previously, "Analyzer" is authorized by Alice to access "Sensor"'s data under the prefix "`/ndnfit/alice/sensor`". The naming convention is shown in Figure 5 and a simplified data production and encryption process is illustrated below.

1) Key Generation: The "Owner" app will first generate a key pair (KEK, KDK) for key encryption and decryption, respectively. It then produces two Data packets: one carrying KEK in plaintext and the other containing KDK and encrypted using "Analyzer"'s public key. In our example, the KEK packet has the name "`/ndnfit/alice/NAC/sensor/KEK/002`", while the KDK' packet name follows the format "`/ndnfit/alice/KDK/002/ENCRYPTED-BY/ndnfit/alice/analyzer/...`".

2) Data Production: When producing data, "Sensor" first generates a symmetric key called CK (content key) for content encryption; we assume the key id is "001". Then, it fetches "KEK" and uses it to encrypt the CK. Next, it encrypts each data packet it produces with the CK and packs the encrypted content into Data packet with name "`/ndnfit/alice/sensor/example/ENCRYPTED-BY/ndnfit/alice/sensor/CK/001`", and the encrypted symmetric key into another Data packet, which is named "`/ndnfit/alice/sensor/CK/001`".

3) Data Consumption: As shown in Figure 5, when "Analyzer" wants to consume data, it first fetches the Data packet using the name "`/ndnfit/alice/sensor/example`". The returned Data packet name informs "Analyzer" that the content was encrypted using the "CK", and thus "Analyzer" then fetches the corresponding "CK" using an Interest with the name extracted from the content Data name. To decrypt "C-KEY", the consumer sends an Interest to further fetch "KDK" Data packet "`/ndnfit/alice/KDK/002/ENCRYPTED-BY/ndnfit/alice/analyzer/...`". As indicated by the KDK Data name, the fetched "KDK" is encrypted using "Analyzer"'s key. By decrypting the content in the fetched "KDK" Data packet, the application obtains "KDK" and can decrypt the symmetric key and use it to finally decrypt the sensor data.

Interest /<content Name>
Data /<content Name>/**ENCRYPTED-BY**/<CK-prefix>/**CK**/<key-id>

Interest /<CK-prefix>/**CK**/<key-id>
CK Data /<CK-prefix>/**CK**/<key-id>/**ENCRYPTED-BY**/<access-prefix>/**KDK**/<key-id>

Interest /<access-prefix>/**KDK**/<key-id>
KDK Data /<access-prefix>/**KDK**/<key-id>/**ENCRYPTED-BY**/<consumer-name>/**KEY**/<key-id>

Fig. 5. Naming Convention in Name-based Access Control

B. Fine Granularity Access Control

By defining naming conventions, NAC enables fine-grained access control. For instance, an "KEK" name could be "`/ndnfit/alice/NAC/sensor/example/8AM/10AM/KEK/002`", indicating that the key will only be used to protect the example data produced by "Sensor" from 8 AM to 10 AM.

VII. Data and Certificate Availability

A. Improving Data Availability via In-network Storage

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

B. 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 developed the NDN certificate bundle [11] to allow each producer to collect all the certificates 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" combines the certificates needed to verify its data in a certificate bundle. Specifically, the bundle will 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 with a single Interest. [4]

VIII. Discussion

A. Comparison of NDN and TCP/IP Security

The differences between the NDN and TCP/IP security solutions originate from the fact that NDN names data whereas IP names locations.

1) Securing Data vs Securing Channels: In TCP/IP, the basic communication unit is a channel between two IP addresses. Consequently, protocols like IPSec and TLS secure channels (e.g., IP channels or TCP channels). However, (i) protected network channels do not directly translate to data authenticity – the data could have been altered before entering the channel; (ii) data loses cryptographic protection as soon as it leaves the channel; and (iii) when multiple parties communicate, securing the channel between every pair of endpoints can quickly cause scalability and manageability issues. By 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.

2) Establishing Trust using Name Semantics: Existing security solutions lack the means to effectively reason about trust. For instance, current secure communication protocols (e.g., HTTPS, or QUIC) follow a common practice of accepting a signature if it was (in)directly signed by a trusted CA. However, [12] shows that commercial certificate authorities themselves may not be reliable and that signature verification alone is not enough to establish trust. NDN takes a fundamentally different approach to trust establishment. In NDN, (i) entities may utilize local authorities, instead of commercial certificate authorities, as trust anchors; (ii) trust policies are expressed explicitly by using name semantics in a systematic way, allowing applications to reason about security rather than blindly trusting signatures by external CAs; and (iii) naming conventions can facilitate automated key management, thus improving system usability.

B. Remaining Challenges

The development of the NDN architecture has guided the creation of a new network security framework and, at the same time, brought both new opportunities and new challenges [13]. Regarding user privacy [14], on one hand, Interest packets carry 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 protected. Additionally, both the Content Store and Pending Interest Table in an NDN router have the potential to increase the attack surface [15]. The NDN research community is actively investigating ways to mitigate these challenges.

[4]If the bundle size is larger than the maximum packet size, it will be fragmented. The first returned segment will contain the total number of pieces.

IX. Conclusion

In [16], we argued that, by naming and securing data directly, NDN offered intrinsic advantages for securing network communications. Evidence from our efforts to develop NDN security solutions suggests that this is indeed true. Named, secured Data packets (including certificates and trust schemas) can be easily fetched from anywhere and serve as a powerful building block for security solution development. Furthermore, we learned that one can establish well-defined naming conventions to systematically define trust policies using schemas, as well as design name-based access control via encryption. We also learned, the hard way, the importance of automating security operations instead of leaving the burden 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 existing solutions implemented in TCP/IP networks. The development process of the NDN security model has convinced us that, by building a network architecture based upon named data, we can effectively develop exciting new network security solutions.

Acknowledgment

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

References

[1] L. Zhang, A. Afanasyev *et al.*, "Named Data Networking," *ACM SIGCOMM Computer Communication Review*, 2014.

[2] C. Yi, A. Afanasyev, I. Moiseenko, L. Wang, B. Zhang, and L. Zhang, "A case for stateful forwarding plane," *Computer Communications*, vol. 36, no. 7, pp. 779–791, 2013.

[3] H. Zhang, Z. Wang *et al.*, "Sharing mHealth Data via Named Data Networking," in *ICN*, 2016, pp. 142–147.

[4] Y. Yu, A. Afanasyev, Z. Zhu, and L. Zhang, "An endorsement-based key management system for decentralized NDN chat application," NDN, Technical Report NDN-0023, Jul. 2014. [Online]. Available: http://named-data.net/publications/techreports/

[5] R. L. Rivest and B. Lampson, "SDSI–a simple distributed security infrastructure." Crypto, 1996.

[6] Z. Zhang, A. Afanasyev, and L. Zhang, "NDNCERT: universal usable trust management for ndn," in *Proceedings of the 4th ACM Conference on Information-Centric Networking*. ACM, 2017, pp. 178–179.

[7] Y. Yu, A. Afanasyev *et al.*, "Schematizing Trust in Named Data Networking," in *Proceedings of the 2nd International Conference on Information-Centric Networking*. ACM, 2015, pp. 177–186.

[8] M. Mosko, E. Uzun, and C. A. Wood, "Mobile sessions in content-centric networks," in *IFIP Networking*, 2017.

[9] Z. Zhang, Y. Yu, A. Afanasyev, J. Burke, and L. Zhang, "NAC: name-based access control in named data networking," in *Proceedings of the 4th ACM Conference on Information-Centric Networking*. ACM, 2017, pp. 186–187.

[10] C. Marxer and C. Tschudin, "Schematized access control for data cubes and trees," in *Proc. of ACM Conference on Information-Centric Networking*, 2017.

[11] M. Mittal, A. Afanasyev, and L. Zhang, "NDN certificate bundle," NDN, Technical Report NDN-0054, 2017.

[12] C. Cimpanu, "14,766 Let's Encrypt SSL Certificates Issued to PayPal Phishing Sites," [Posted 24-March-2017]. [Online]. Available: https://www.bleepingcomputer.com/news/security/14-766-lets-encrypt-ssl-certificates-issued-to-paypal-phishing-sites/

[13] R. Tourani, S. Misra, T. Mick, and G. Panwar, "Security, privacy, and access control in information-centric networking: A survey," *IEEE Communications Surveys & Tutorials*, 2017.

[14] C. Ghali, G. Tsudik, and C. A. Wood, "When encryption is not enough: privacy attacks in content-centric networking," in *Proceedings of the 4th ACM Conference on Information-Centric Networking*. ACM, 2017, pp. 1–10.

[15] C. Ghali, G. Tsudik, E. Uzun, and C. A. Wood, "Closing the floodgate with stateless content-centric networking," in *Computer Communication and Networks (ICCCN), 2017 26th International Conference on*. IEEE, 2017, pp. 1–10.

[16] L. Zhang *et al.*, "Named data networking (NDN) project," NDN, Technical Report NDN-0001, October 2010.