Secure NDN Packet Encapsulation

Daniel Townley, Young Kim, Fred Douglis
Jesse Elwell, Constantin Serban
Peraton Labs
Basking Ridge, NJ, USA
dtownley,ykim,fdouglis,jelwell,cserban@peratonlabs.com

Lan Wang
University of Memphis
Memphis, TN, USA
lanwang@memphis.edu

Alex Afanasyev
Florida International University
Miami, FL, USA
aa@cs.fiu.edu

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

***Abstract*—Packet encapsulation is a general network technique that provides an essential building block for constructing secure networks. While extensively used in IP networks over the last few decades, secure packet encapsulation remains largely unexplored in the context of Named Data Networking (NDN) networks. NDN represents a radical departure from traditional endpoint-oriented networking by making secured data the centerpiece of communication. This new data-centric design brings both advantages and new challenges for the development of secure packet encapsulation that can preserve essential properties of an NDN network, including in-network data caching and built-in multicast data delivery. In this paper, we first identify the major differences between encapsulation solution designs in IP and NDN, highlighting the ensuing challenges, both inherent and practical. We then present a novel design to achieve secure NDN data packet encapsulation, and showcase an implementation suite that enables efficient fetching of securely encapsulated data.[1]**

***Index Terms*—Named Data Networking, NDN, Tunneling, Encapsulation, Security, Name Encryption**

I. Introduction

Packet tunneling via encapsulation is a well established and widely deployed practice. It can be used to implement virtual links and networks, and to secure IP communications across such connections by encrypting the whole original IP packets under the outer IP header. It is desirable to support packet encapsulation in Named Data Networking (NDN) [1], [2] to provide similar functions. In this paper, we investigate the solution space of NDN packet encapsulation by first identifying the fundamental differences between IP and NDN encapsulations, and then developing an effective solution to enable secure NDN packet encapsulation.

Intuitively, one might think that secure NDN-in-NDN encapsulation could simply follow the common practice for secure tunneling in IP networks. That is, given an NDN packet D, one encrypts D to get the encrypted packet $E(D)$, then encapsulates $E(D)$ in another NDN data packet with a different name. However, in an NDN network, consumers use *interest packets* to request desired data by *names*, and the network returns the named data as *data packets* [2]. An NDN network uses a stateful data plane to support this interest-data exchange by keeping track of the *names* of all received, but yet to be answered, interest packets. This enables request aggregation (hence the returned data packets are multicast to all requesters for the same data), interest loop suppression, and path discovery; naming and securing data directly also enables in-network caching.

However, the above functions bring new challenges to encapsulation. The name carried in an *encapsulating* interest packet that requests an *encapsulated* NDN data packet must (a) uniquely identify the original data packet it requests, to allow NDN routers to use the name carried in the encapsulating interest to look up local caches; (b) be consistently constructed by all legitimate consumers, so that NDN routers can aggregate interests for the same original data packet; and (c) provide necessary context for forwarding and security to reliably retrieve encapsulated/secured data packets over public networks. In addition, an encapsulating interest must reflect the qualifier fields (e.g., "`MustBeFresh`" and "`CanBePrefix`") of the encapsulated interest, to retrieve the original data packet that satisfies all the constraints. Therefore, the encapsulating NDN header has to follow a specific formulation to identify and relate to the encapsulated NDN header, which significantly complicates the encapsulation process.

In this paper, we propose an NDN-in-NDN solution that encrypts the entire NDN data packet including the name, key locator, and content, while allowing retrieval of such data using a specially crafted interest packet that does not reveal meta information about the "real" request or data.[2] We developed two prototype solutions that can be embedded at the application (Face Level Secure Encapsulation, FLSE) and middle box (Appliance Level Secure Encapsulation, ALSE) levels and devised an example symmetric cryptographic key management inspired by IPSec.

The main contributions of this paper are two-fold: identifying the constraints in providing secure encapsulation in NDN networks, and providing a principled approach to satisfying such constraints while maintaining both performance and security. We describe our design goals and limitations, and sketch two different implementations within the same framework. In the rest of this paper, we first identify the particular challenges of providing secure NDN-in-NDN encapsulation (Section II),

[1] This work was supported in part by DARPA. The views, opinions and/or findings expressed are those of the authors and should not be interpreted as representing the official views or policies of the Department of Defense or the U.S. Government. Distribution Statement "A" (Approved for Public Release, Distribution Unlimited)

[2] Note that *timing attacks*, where the adversary uses the mere fact of Interest-Data communication, and *inference-based attacks*, where the adversary correlates packets based on their size and relative timing, are outside the scope of this paper, and we plan to address these issues in our future work.

we report our NDN-in-NDN design and implementation of NDN full packet encapsulation and encryption (Section III and Section IV), and we provide a brief comparison with related work before concluding.

II. Background

A. IP Secure Tunnel Encapsulation

IP encapsulation, e.g., IP-in-IP [3] or GRE [4], provides IP packet encapsulation and datagram delivery between two end points. IP datagrams are "nameless", i.e., they do not have their own identifiers. The destination address of an IP-in-IP packet is used for packet delivery to the endpoint and the "Protocol" field is used for payload delivery to the next protocol module, i.e., the decapsulation endpoint. *IP secure encapsulation*, e.g., IPSec [5], provides similar functionality, except that the IP packet to be tunneled is first encrypted and an authentication header (i.e., ESP header) is added for identification of the decapsulation and decryption endpoint. IPSec is essentially a tunneling protocol between two endpoints A and B: it first encrypts a given IP packet which has a pair of source and destination addresses [X, Y], and then adds a new IP header with a new pair of source, destination addresses [A, B] for the outer header. The addresses [X, Y] may be the same as [A, B], in the case of the *IPSec transport mode*, or they can be entirely independent, as in the case of *IPsec tunnel mode*. The TTL field can either get a new value or is copied from the inner header depending on the tunnel configuration. A great portion of the IPSec environment specification is devoted to the issue of encryption and authentication key configuration and distribution. While these aspects are important and consequential in the design of the IPSec encapsulation and tunneling, these questions are outside the scope of IPSec itself, and in practice they depend on user configuration. The mapping of the inner IP header to the outer IP header is relatively straightforward, in contrast to the NDN case, as we shall see below.

B. NDN Headers and Names

NDN packet headers, and their most important component, the NDN names, have a more complex and sophisticated structure and use than the IP headers and addresses. As opposed to IP's use of addresses for forwarding only, NDN uses names for three purposes: interest packet forwarding, data identification, and data security:

- An interest packet carries the name of the data packet to be retrieved, and routers perform name lookups using longest prefix match similar to IP forwarding lookups, but with more structurally complex names.
- Each name must uniquely identify a data packet, which may be fetched from its producer or found at router caches; multiple consumers must retrieve exactly the same data packet when they query with the same name.
- NDN names are used for security management; consumers perform data authentication based on the name of the data.

NDN names are hierarchical in nature and commonly structured in the following way. The leftmost portion of the name (prefix components) represents information typically devoted to "network de/multiplexing" activities, i.e., information used by NDN forwarding to move an interest packet towards the requested data. The next portion of the name (one or more components) represent information that enables interests to reach the right application process or portion of the NDN application stack. This portion is broadly equivalent to the transport header in an IP packet. Finally, the remaining portion of the name identifies the specific piece of data that is being retrieved.

In addition to the name, an interest packet also carries additional information related to the data being requested. Two most important pieces of the additional information are

- Name match constraint: the "`CanBePrefix`" boolean flag in an interest specifies whether the name of an interest packet can be a prefix of the name of the matching data packet; and
- Timing constraint: the "`InterestLifetime`" field in an interest requires a data match to occur within a specified time period, and the "`MustBeFresh`" boolean flag in an interest specifies whether a data packet can still be a match after the data has reached the end of its freshness time period.

C. Solution Requirements

a) Deployment Requirements: Figure 1 shows the two possible deployments considered by the proposed secure NDN-in-NDN encapsulation. Let us call the name carried in the outer interest *encapsulating name*, and the name of the inner interest (the name of the original data packet) *encapsulated name*. On the left figure, multiple producers and consumers are operating on the same "public" network. Here the goal of the secure NDN-in-NDN encapsulation is to provide privacy to the data packet content as well as to the trailing portion of the name consisting of the application/-transport (data name suffix), while exposing only the leading portion of the name used for handling by the NDN network. As shown in the figure, the original name "`/public/app1 /item1`" is encapsulated and becomes the encapsulating name "`/public/data1`", which provides privacy to the "`app1/item1`" portion of the name. This service is intended to be broadly equivalent to the privacy guarantees of the IPSec Transport mode. While such privacy can be achieved by a carefully designed application namespace, performing this task would be considered burdensome to application developers. Thus, a desired solution is to provide such privacy function in a way that is transparent to applications.

The right figure depicts the function of the secure NDN-in-NDN encapsulation that allows the tunneling of private content over a "public" NDN network.[3] Here the goal of the secure NDN-in-NDN encapsulation is to provide privacy for the entire private NDN packet, including both the name

[3] Here the meaning of "public" network refers to the ability of all the participants of the network to retrieve data based on a commonly-agreed and previously-assigned naming scheme. This is in contrast to a "private" network where the naming scheme is discretionary to the private party.

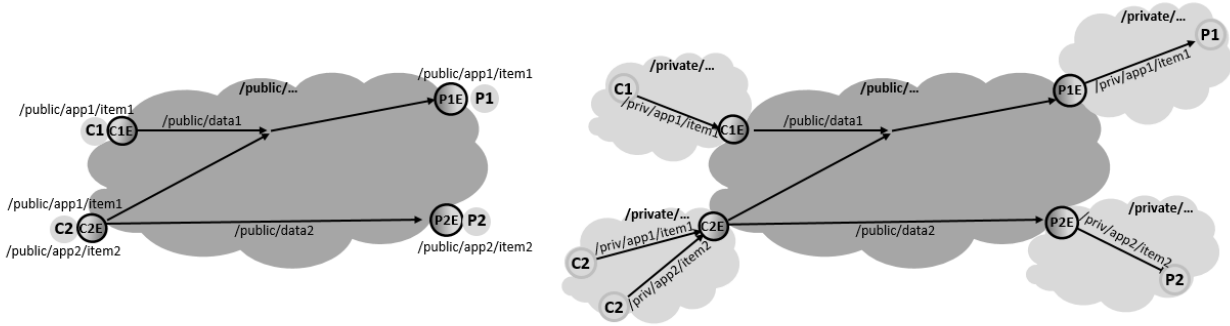

Fig. 1: Deployment Modes for the Secure NDN-in-NDN Encapsulation: Transport Mode (left), Tunnel Mode (right)

and content. As shown in the figure, a consumer residing in a private NDN network retrieves a data packet with the name “`/private/app1/item1`”. Here data is encapsulated at the boundary between the private and public network in a packet with the encapsulating name “`/public/data1`”, hence providing privacy to the entire packet. This service is designed to be broadly equivalent to the IPSec Tunnel mode. In this scheme, packets are forwarded and matched according to the inner private name while in the private network (either producer or consumer side), and handled according to the outer public name while in the public network. Similar to the previous case, the encapsulation should be transparent to the application, the consumers, or the producers, and happen at the boundary between the networks.

b) Functional Requirements: Given that the names serve multiple roles in an NDN network, as discussed in II-B, it is incumbent upon the encapsulating names to fulfill the same requirements with respect to the encapsulated data as follows:

- The name in the outer NDN interest must uniquely identify a piece of data both at the producer and in the network in a consistent manner. Accordingly, a piece of data generated by a producer and encapsulated in an outer NDN header should be retrieved by the encapsulating name from the producer and any network caches, if the corresponding data exists.
- Multiple consumers attempting to retrieve the same piece of data should generate interests with the same encapsulating name. Furthermore, these interests should also match the same data generated by the producer. This situation is depicted in Figure 1 where two consumers issuing the interest “`/private/app1/item1`” will generate interests with the same encapsulating name (“`/public/data1`”), so that the interests can be aggregated in the network; the encapsulating name should also be converted to the same original name at the producer.
- The encapsulating name of the NDN interest packet should carry enough information to allow efficient forwarding of interests to both the producers and to network caches with the data. For example, in Figure 1, the encapsulating names “`/public/data1`” and “`/public/data2`” should enable the NDN network to distinguish between the two prefixes such that “`/public/data1`” interests are forwarded to the encapsulation endpoint *P1E* corresponding to producer *P1*, while “`/public/data2`” interests are forwarded to the encapsulation endpoint *P2E* corresponding to producer *P2*.
- While NDN interests generaly carry the full name of the data to be fetched, some interests may carry a partial name prefix of the data (for a more detailed example, see Section III). Similarly, producers generating NDN data may announce only a partial name prefix for the NDN interests to be directed toward. An encapsulation mechanism should support such data requests by partial prefix both in the network and at the producers.

c) Security Requirements: It is assumed that the inner NDN packet is encrypted and authenticated using standard cryptographic mechanisms; the details of these mechanisms are beyond the scope of this paper. Given the requirement that the encapsulated NDN name must uniquely resolve to an encapsulating NDN name, while at the same time preserving the privacy of the encapsulated name, it follows that this name conversion should be defined as a cryptographic one-way function that prohibits the computation of the encapsulated name based on the encapsulating name without proper authentication and encryption credentials.

Finally, an overall goal of the solution is to provide transparency and allow the secure NDN-in-NDN encapsulation to be applied repeatedly–as long as it can be afforded from a performance and an overhead point-of-view, to provide multiple layers of security, as desired, without constraints imposed by the solution design or implementation. The next section will discuss our design satisfying the above requirements.

III. Solution Design

A. Abstract Model

The most important aspect of the design of the NDN-in-NDN solution is the design of the outer namespace, and its relationship with the encapsulated name. Abstractly speaking, the process of generating the encapsulating namespace represents a mapping function that transforms the inner namespace into an outer namespace using a unique transformation. A *secure* encapsulation is a reverse transformation between an encapsulating namespace into an encapsulated namespace that is cryptographically secured. An NDN namespace can be represented as a tree (or forest) where each node holds a name component that may occur in the namespace. Accordingly,

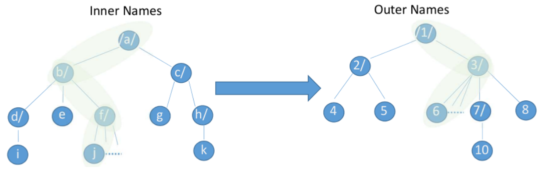

Fig. 2: Encapsulation as a Namespace Tree Mapping

the name of each piece of data is represented by a path between the root node and a leaf node. As an example, Figure 2 shows two such namespaces, one for the encapsulated data (left), and one for the encapsulating data (right). The encapsulation process defines the mapping between the two trees, where each possible path in the left tree uniquely maps to a path in the right tree. In this example, the encapsulated (inner) name "`/a/b/f/j`" is transformed to the encapsulating (outer) name "`/1/3/6`". Similarly, the inner name "`/a/b/e`" could be transformed into outer name "`/1/2/5`". For the NDN encapsulation to function, it is necessary to satisfy the uniqueness constraint: i.e. "`/a/b/f/j`" always maps to "`/1/3/6`" during encapsulation; conversely "`/1/3/6`" uniquely maps to "`/a/b/f/j`" during decapsulation. Note that the depth, structure, or composition of either tree should be flexible enough to accommodate different deployment assumptions and limitations.

B. NDN-in-NDN Encapsulation Approaches

Below we discuss several secure NDN-in-NDN encapsulation approaches that follow directly from this abstract model.

a) Structure-Preserving Secure Encapsulation: A straightforward way to implement secure encapsulation is to separately encrypt each name component in the inner name tree into a ciphertext name component in the outer, encapsulating name tree. Figure 3 shows such encapsulation.

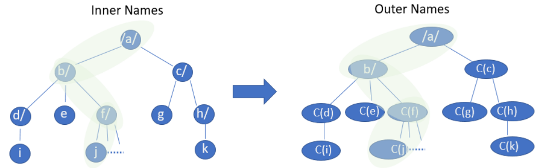

Fig. 3: Per-Component Name Encryption

At the encapsulation time, each component in the inner name, e.g., *f* and *j* get encrypted into ciphertext *C(f)* and *C(j)*, respectively. At decryption time, the reverse process takes place: an encapsulating name is decrypted component by component to retrieve the inner name (see later for details on the type and usage of credentials to perform such operation).

Two optimizations warrant further discussion. First, the above per name component encryption can be applied selectively to part of the hierarchy. The top portion of the name is used for network forwarding activities, if it does not expose privacy information, such components can be mapped to a cleartext outer name, or directly copied from the inner name. As shown in Figure 3, "`/a/b`" components are copied directly into the outer name. This optimization is consistent with the Transport mode deployment discusses in Subsection II-C, where only the trailing part of the name, dedicated to transport and application state, is protected for privacy and confidentiality.

The second optimization refers to the encryption scheme used for each component. Given that the encryption of each name component can yield lengthy ciphertext representation (especially when padding and block encryption is used), a cryptographic keyed hash can be used instead. Since such hashing is a one-way function, the inner name cannot be obtained directly from the hash, making the decapsulation more challenging. The entire encapsulated packet can be decrypted instead, and the hash can be applied to the clear text name for the outer name verification.

While this scheme is straightforward, it has two drawbacks. The first drawback is that it can produce lengthy outer names. Even with the hash-based optimization discussed above, a name consisting of ten components will be mapped to an outer name with 32×10 bytes using a SHA256 hash function, which is arguably excessive for just the name portion, without counting the encapsulated packet carried as payload. The second and more concerning drawback is that per-component encryption exposes the structure of the application namespace, which may allow a direct observation by an adversary of the state machine transition of the application. This reduces the traffic confidentiality of the secure encapsulation.

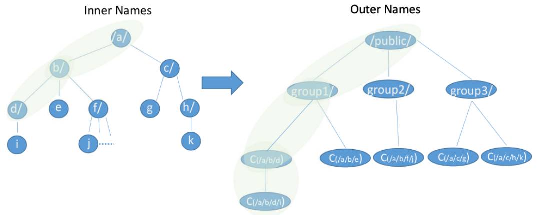

Fig. 4: Privacy-Oriented Name Encryption

b) Privacy-Oriented Secure Encapsulation: Exposing the entire hierarchical structure of the inner name is not only undesirable, but also unnecessary. The rightmost components of the outer names are not expected to be employed in forwarding of interests or delivery of data over the public network, so they are hardly expected to be used as individual, separable, entities. Instead, an entire portion of the inner name can be mapped and encrypted as a single component, hence collapsed into a single name component. This brings about a number of advantages, such as increased efficiency due to shorter outer names, and increased privacy. Figure 4 shows such an approach. On the right-hand side of each name tree, the inner name "`/a/c/h/k`" is encapsulated in the outer name "`/public/group3/C(/a/c/h/k)`", where the entire path is encrypted (or hashed according to the above-described optimization) as a whole. This encapsulating name exhibits a clear-text prefix designed to help its forwarding in the public network. While the savings in outer name length may look minor in this example, for a deep namespace it will be substantial. The encapsulation scheme shown here is equally suitable for NDN-in-NDN Transport mode as well as Tunnel mode. The example is geared towards Tunnel mode operations. Figure 4 shows another important aspect of this design. While name collapsing brings distinct advantages, it also introduces some challenges. Consumers may send interest packets with partial names, i.e., prefixes, to fetch data, and producers may register prefix for serving data. When such prefixes are encrypted as a unit (e.g., "`C(/a/c/h)`"), they will not match any data encapsulated with the encrypted full name (e.g., "`C(/a/c/h/k)`"), hence breaking the functionality. The proposed design introduces an intermediary component that separately encrypts the prefix, and adds it as an outer name component. As shown at the left of Figure 4, the inner name "`/a/b/d/i`" is encapsulated as "`/public/group1/C(/a/b/d)/C(/a/b/d/i)`", such that the prefix is exposed as a separate component. If the consumer issues a prefix-based interest "`/a/b/d`", this will be encapsulated as a "`/public/group1/C(/a/b/d/)`" and it will match any data with such prefix, including "`/public/group1/C(/a/b/d)/C(/a/b/d/i)`". While this approach solves the problem, it assumes that the encapsulation mechanism is aware of the prefixes that the application will use, which can be a limiting factor.

IV. Implementation

We implemented two versions of the Privacy-Oriented design described above: 1) a security library-based implementation, called *Face Level Secure Encapsulation* (FLSE), and 2) an appliance-based implementation, called *Appliance Level Secure Encapsulation* (ALSE). Differences between the implementations are highlighted in Figure 5.

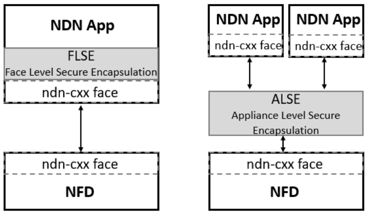

Fig. 5: Implementations of Secure NDN-in-NDN Encapsulation

FLSE has been implemented as an extension to the ndn-cxx, the standard NDN library [6]. It augments the application-level entry point to NDN networking (*Face* API in ndn-cxx) with automatic encapsulation and encryption functionality, allowing applications to use the new functionality by merely switching the compile-time libraries. In this respect, FLSE provides Transport Mode functionality, as expressed interests and published data out of the application are already encapsulated and encrypted. The name is transformed using the Privacy-Oriented design; the NFD to which the application connects via ndn-cxx may place constraints on the allowed published prefix, so this name transformation obeys such constraints.

ALSE has been implemented as a standalone appliance (process/service). This service resides outside of the application and acts as a border forwarder. Multiple applications (or even upstream NFDs) can connect to an ALSE service, which encrypts and encapsulates the traffic arriving on the upstream faces, and forwards the encapsulated traffic on a single downstream face. ALSE is particularly suited to provide secure NDN-in-NDN encapsulation in Tunnel Mode, enabling applications using private namespaces to be deployed over public networks.

In terms of deployment, applications can use either of the two implementations. Additionally, the two implementations do not preclude stacking or recursive usage. For instance, the ALSE can be positioned between an NFD instance and NDN application compiled with FLSE. Conceptually both solutions have similar processing pipelines, but their implementations can (and do) use different packages and security libraries for added diversity and protection.

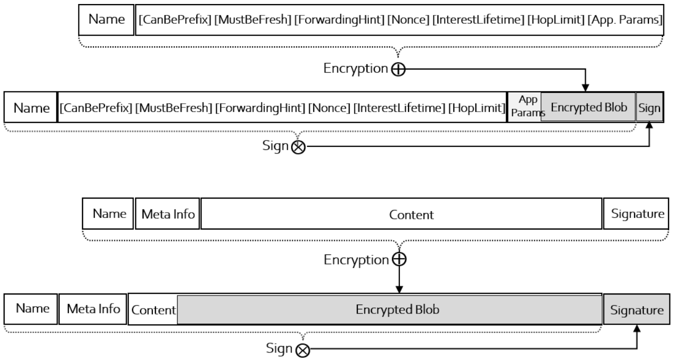
Name
[CanBePrefix] [MustBeFresh] [ForwardingHint] [Nonce] [InterestLifetime] [HopLimit] [App. Params]
Encryption ⊕
Name
[CanBePrefix] [MustBeFresh] [ForwardingHint] [Nonce] [InterestLifetime] [HopLimit]
App Params
Encrypted Blob
Sign
Sign ⊗
Name
Meta Info
Content
Signature
Encryption ⊕
Name
Meta Info
Content
Encrypted Blob
Signature
Sign ⊗

Fig. 6: NDN interest and Data Secure Encapsulation

Our encapsulation implementation follows the NDN packet format specification v0.3 [7].[4] We selected an encapsulation strategy and mapping design highlighted in Figure 6. Specifically, for data packets, we chose to encrypt the complete original data packet using a symmetric cryptography and include it as a payload of the new data packet that has the new mapped name (see Section III), as well as new "`MetaInfo`" (new "EncryptedBlob" content type and, for simplicity, the same value of "`FreshnessPeriod`" if it was present in the original packet), and "`Signature`" fields (generated by ALSE or FLSE according to the desired trust policy).

For each interest packet, we also encrypt the complete original packet and include it as "`ApplicationParameter`" in the new interest, which carries the new encapsulating name and, for simplicity, the copy of all original qualifiers to preserve type and timing of the request (i.e., the name matching and timing constraints described in II-B).

The secure NDN-in-NDN encapsulation requires crypto keys for inner packet encryption, outer packet authentication, and cryptographic name transformation (Section III). To this end we use a symmetric key k_p controlled by the encapsulation producer. Prior to encapsulation, the encapsulation consumer must be able to securely obtain the credentials (including key k_p). In our implementation, a set of per-encapsulator producer credentials including key k_p is associated to Security Parameter Index (SPI), representing a custom *name component* managed by a data producer that enables the decapsulating

[4] In FLSE implementation we made a small deviation from the specification regarding computation of "`ParametersDigest`" component of the name. We plan to resolve this in future versions.

system to select the Security Association (SA) under which a received encapsulated packet will be processed. The SPI can be included in the outer NDN packet header, or it can be derived from the outer name of the received encapsulated NDN packets. The approach is broadly similar to the employment of the SPIs by IPSec (as per RFC 2401). The configuration and distribution of the SAs and associated key material is beyond the scope of this paper. Our scheme uses a pre-shared encapsulation table ET that maps $\langle inner\_pfx, outer\_pfx, SPI\rangle$. Upon interest encapsulation, the consumer matches the expressed interest to the $inner\_pfx$ entry in ET, it uses the $outer\_pfx$ entry for the encapsulated name and the SPI-identified credentials to encrypt the interest.

V. Related Work

Instead of relying on channel-based security (as in TLS/IP), NDN adopts a data-centric security paradigm: all NDN data packets are cryptographically signed and encrypted if needed, hence alleviating many of the security problems that plague IP networks [8]. In addition, Name-based Access Control (NAC and NAC-ABE) [9] encrypts the payload of NDN data packets with automated encryption/decryption key management. NAC supports granular per-namespace access policies, and only authorized data consumers can obtain decryption keys to get access to data. While this design secures the content of NDN data packets, it exposes the original data names in both interest and data packets.

Partridge et al. [10] proposed encapsulating encrypted ICN packets inside other ICN packets that can be safely forwarded over an untrusted network, similar to VPN encapsulation. Different from our proposed solution, this approach explicitly addresses content encryption only and requires that all nodes in the private network share the same security association, as any one of them might be required to respond to an encrypted interest.

Another set of related solutions are based on proxy encryption concept, allowing parties to exchange encrypted data without sharing private keys by using multiple layers of encryption [11]. The use of proxy encryption for content-based networking was first proposed by Chaabane et al. [12], although this work did not implement or evaluate the mechanism described. PrivICN [13] described a detailed implementation of proxy-encrypted ICN using the Elgamal cryptosystem [14], including a performance evaluation and an open-source library. This system applies encryption to both data names and contents, preserves the full benefit of ICN caching by encrypting each name component individually, and also allows for the revocation of user keys. However, the encryption scheme creates large overhead for typical NDN name sizes and reduces the confidentiality of the application flow control. Additionally, the proxy encryption approach relies on a central key management server to distribute credentials, creating a level of persistent centralization that is at odds with the design points of NDN and other ICN systems.

A number of papers have been published on NDN name obfuscation to protect user privacy. Although the majority of the proposed solutions use proxy servers to perform this translation, the proxy-less name encryption and encapsulation mechanisms developed in this work seem to represent the most promising direction [15].

VI. Conclusion

This paper addresses the problem of secure packet encapsulation in NDN networks. While the NDN data centric approach to networking increases the security of the content, improves data robustness, and increases efficiency in group-based communications, this paper shows that the very same properties make secure data encapsulation difficult, especially when considering the introduced overhead and loss in application traffic flow privacy. This novel NDN-in-NDN approach addresses these challenges and demonstrates an implementation that is both practical as well as flexible for a large set of use cases. Rigorous performance evaluation, testing and deployment against diverse NDN-based applications, and key management aspects of the solution are left for future work.

References

[1] V. Jacobson, D. K. Smetters, J. Thronton, M. F. Plass, N. H. Briggs, and R. Braynard, "Network Named Content," *ACM CoNEXT*, 2009.

[2] L. Zhang, A. Afanasyev, J. Burke, V. Jacobson, kc claffy, P. Crowley, C. Papadopoulos, L. Wang, and B. Zhang, "Named Data Networking," *ACM Computer Communication Review*, Jun. 2014.

[3] W. Simpson, "RFC 1853 - IP in IP Tunneling," IETF Network Working Group, RFC, 1995.

[4] S. Hanks, T. Li, D. Farinacci, and P. Traina, "RFC 1701 - generic routing encapsulation (GRE)," IETF Network Working Group, RFC, 1994.

[5] K. S. S. Kent, "RFC 4301 - security architecture for the Internet protocol," IETF Network Working Group, RFC, 2005.

[6] NDN Project Team, "ndn-cxx: Ndn c++ library with experimental extensions," Online: https://docs.named-data.net/ndn-cxx/current/, Last accessed: Feb. 12, 2023.

[7] ——, "NDN packet format specification," Online: https://docs.named-data.net/NDN-packet-spec/0.3/, Last accessed: Feb. 12, 2023.

[8] Y. Yu, A. Afanasyev, D. Clark, K. Claffy, V. Jacobson, and L. Zhang, "Schematizing trust in Named Data Networking," in *Proceedings of 2nd ACM Conference on Information-Centric Networking*, Sep. 2015. [Online]. Available: http://dx.doi.org/10.1145/2810156.2810170

[9] Z. Zhang, Y. Yu, S. K. Ramani, A. Afanasyev, and L. Zhang, "NAC: Automating access control via Named Data," in *IEEE Military Communications Conference (MILCOM)*, 2018, pp. 626–633.

[10] C. Partridge, S. Nelson, and D. Kong, "Realizing a virtual private network using named data networking," in *Proceedings of the 4th ACM Conference on Information-Centric Networking*, 2017, pp. 156–162.

[11] M. Blaze, G. Bleumer, and M. Strauss, "Divertible protocols and atomic proxy cryptography," in *International Conference on the Theory and Applications of Cryptographic Techniques*, 1998, pp. 127–144.

[12] A. Chaabane, E. De Cristofaro, M. A. Kaafar, and E. Uzun, "Privacy in content-oriented networking: Threats and countermeasures," *ACM SIGCOMM Computer Communication Review*, vol. 43, no. 3, pp. 25–33, 2013.

[13] C. Bernardini, S. Marchal, M. R. Asghar, and B. Crispo, "PrivICN: Privacy-preserving content retrieval in information-centric networking," *Computer Networks*, vol. 149, pp. 13–28, 2019.

[14] T. Elgamal, "A public key cryptosystem and a signature scheme based on discrete logarithms," *IEEE Transactions on Information Theory*, vol. 31, no. 4, pp. 469–472, 1985.

[15] Z. Zhang, S. Y. Won, and L. Zhang, "Investigating the design space for name confidentiality in Named Data Networking," in *IEEE Military Communications Conference (MILCOM)*, 2021, pp. 570–576.