A New API in Support of NDN Trust Schema

Tianyuan Yu
UCLA
tianyuan@cs.ucla.edu

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

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

Yekta Kocaoğullar
Sabancı University, Istanbul
ykocaogullar@sabanciuniv.edu

Lixia Zhang
UCLA
lixia@cs.ucla.edu

ABSTRACT

The decade-long experiences from developing applications over Named Data Networking (NDN) have taught us the importance of well-designed libraries that offer support to application developers to support data security. NDN trust schema provides a critical component in the NDN security support, however its implementation and support only started receiving significant attention in recent years. This paper first provides a summary of the existing API support for trust schema, then takes a step forward by developing a new trust schema API, named *Envelope*. *Envelope* addresses the application requirements by offering comprehensive trust schema functionalities, an easy-to-write schema language, and an extensible design. To demonstrate the usefulness of *Envelope*, we develop a blog application which uses Envelope to secure its data. Our results show that *Envelope* provides effective trust schema support for applications with acceptable overhead.

KEYWORDS

Named Data Networking (NDN), Trust Schema, API, Software Engineering

ACM Reference Format:
Tianyuan Yu, Xinyu Ma, Hongcheng Xie, Yekta Kocaoğullar, and Lixia Zhang. 2023. A New API in Support of NDN Trust Schema. In *10th ACM Conference on Information-Centric Networking (ACM ICN '23), October 9–10, 2023, Reykjavik, Iceland*. ACM, New York, NY, USA, 9 pages. https://doi.org/10.1145/3623565.3623709

1 INTRODUCTION

Among the multiple architectural advantages provided by Named Data Networking (NDN), the most important one is its ability to build security support intrinsically into the architecture. However, effective and usable security library support are required to enable application development to take this architectural advantage. NDN security libraries need to support three basic functions: (i) installing security parameters including trust anchors, certificates, and trust schemas during each new entity's bootstrapping phase; (ii) storing and managing these security components continuously throughout the entity's life time; and (iii) executing trust schemas during application runtime. In this paper, we focus on providing easy-to-use trust schema support.

The basic concept of trust schema was first introduced in 2015 by Yu et. al. [24]. It proposed to use a structured schema language to define security policies based on the semantic names of data to be communicated, and to sign and validate each NDN data packet following the defined policies. However, the first implementation of trust schema in ndn-cxx [14] offered limited functionality; it can interpret trust schemas written in a language akin to regular expressions, and provides data validation support only. More recent work by Nichols [9, 10] prompted greater attentions to library support for trust schema. Nichols defined a special language, Versatile Security (VerSec), to be used for authoring trust schema, which is implemented in Data-Centric Toolkit (DCT) for home IoT devices [10]. Subsequently, NDNts library [8] offered partial support for using VerSec language in its trust schema implementation.

In this paper, we first summarize the previous efforts in trust schema support. Drawing the lessons from the past, we then design a new trust schema API called *Envelope*. Envelopemakes three contributions to trust schema support. First, it introduces an easy-to-write trust schema language LightVerSec. Second, it provides an abstraction layer that can fetch certificates needed by trust schema execution from multiple sources. Third, it enables applications to customize trust schema execution by defining their own packet validation pipelines.

The remainder of this paper is organized as follows. §2 provides an NDN overview and its trust domain model. §3 states the design goals from a forward looking perspective. §4 introduces our trust schema language LightVerSec. §5 discusses the design of Envelope and how well it can satisfy applications' needs. §6 describes the Envelope implementation and qualitative evaluation which focuses on API effectiveness. We discuss our lessons learned in §7, and conclude the paper in §9.

2 BACKGROUND

In this section, we first clarify several security-related terminologies, and then give a brief introduction to Named-Data Networking (NDN), discuss the model of *trust domain* and the role of trust schemas in NDN security solutions.

2.1 Security, Trust, Validity: Terminology Clarification

The words "valid", "secure", and "trust" have all been used in describing security solutions under different contexts. They express different degrees of trustworthiness and supported by different libraries and systems. For example, TLS uses the word "secure" to mean authentication between the two ends and the content confidentiality of a TCP connection [13]. On the other hand, Bitcoin uses the word "secure" to mean that every transaction is valid *within the specific semantic definition* of Bitcoin [7].

To avoid terminology ambiguities as shown above, we provide the definitions for the following two security-related terms that will be used in the rest of this paper. We use an illustrative example to help clarify these definitions. We pick the blog application from [24] as shown in Figure 1, and give it a name *TechDaily.* TechDaily is a website hosting technical blogs written by multiple authors. The website has one or a few administrators to certify the blog authors, who then publish blog postings following TechDaily's naming convention.

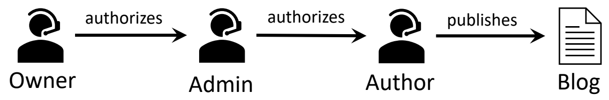

Figure 1: Entities of a simple blog website TechDaily

Authentication: verify whether a piece of data is produced by the claimed producer and not manipulated by an impersonator. NDN mandates that every Data packet carry a signature, so authentication reduces to (i) obtaining the signing key associated with the Data producer and (ii) cryptographically verifying the signature. (i) in turn requires retrieving the signing certificates.

Authorization: a named record (a piece of data) is produced by an intended producer, and accessed by an intended consumer. That is, the application's access control rules specified by users or administrators allows the producer to produce the Data packet. For example, an author's certificate is issued by an admin, and a blog post is written by a valid author.[1]

The design of Envelope specifically focuses on authentication and authorization for data production and access. *Investigating data content is considered out of the scope of our system.* For example, Envelope will not examine whether the blog content is correct or not.

2.2 NDN & Its Basic Security Building Block

In NDN networks [1, 5], applications use semantic names, instead of host addresses, to communicate. All data packets and producers are assigned with semantic names. Data consumers send *Interest* packets containing the desired data names, and the network brings back the matching *Data* packets. Each Data packet is signed by its producer and carries a *KeyLocator* field, which contains the name of the signing key or its corresponding certificate associated with its producer.

The data bits carried in an NDN Data packet can be a segment from a file, a signing key itself (with the resulting signed data being a certificate), or a package of defined security policies. Following the least privilege principle, NDN enforces strict security policies on which key can be used to sign which piece of data. As we explain below, NDN uses trust schema to define and enforce these policies. Given keys and security policies are also named bags of bits, they can all be fetched using their semantic names, in the same way as fetching any other types of data. Therefore, NDN's semantically named and secured data packets offer a basic building block for building secure networked systems. Once the scope of trusted parties is defined, and the security policies governing these parties' interactions are defined, NDN enables simple, secure, and unified communication support for security management. Next we discuss the scope of trust (§2.3) and how to define security poplicies using trust schema (§2.4).

2.3 Trust Domain

There is no global trust in the human society; instead, the society is made of autonomous organizations of various types and sizes, with trust relations of different degrees established between organizations. If cyberspace is a (perhaps extended) reflection of human society, then its trust scope could/should follow similar structures.

The work by Nichols [10] coined the term *trust domain*[2], defined as a collection of named entities under the same administrative control. Each trust domain has a *trust domain controller*, which is the authority to manage security policies for all the entities within its domain. The controller owns a *trust anchor*, which is a self-signed certificate whose name prefix reflects the name of the trust domain. This trust anchor is installed into all the entities within the trust domain during their security bootstrapping process [21].

The controller for a trust domain D_A, $Ctrl_A$, controls the authentication and authorization rules within its domain. These rules are written in the form of *trust schema* [24, 29] which we explain next. The trust schema makes use NDN's semantically meaningful names assigned to all network entities to define the security rules. For instance, the trust schema rule for a name N under D_A's namespace defines which named entity (or entities) is allowed to produce data with N as the name/name prefix. The trust schema of D_A can also define rules to instruct entities in D_A on how to verify the received data produced by the entities from another trust domain D_B.

When a new entity, E_{new}, wants to participate in a trust domain D_A, it goes through the security bootstrapping process first. During this process, the controller of domain D_A, $Ctrl_A$: (i) authenticates E_{new} by its identity obtained through some external existing systems (e.g. an email address, or a device's barcode), then authorizes its participation in the domain D_A; (ii) installs the trust anchor T_A and trust schema into E_{new}; (iii) assigns a name to E_{new}; (iv) issues a certificate to E_{new}. Security bootstrapping ensures that the domain controller $Ctrl_A$ is the sole administrator of domain D_A. Every networked entity belongs to a single trust domain which is administered by a single trust domain controller.

2.4 Trust Schema

All data packets produced within the trust domain must be signed strictly following the trust schema defined for the data name, and all received data must also be verified based on the trust schema

[1]Access control policies restricting the read privilege of consumers are out of the scope of Envelope. In TechDaily's example, every user is allowed to read every post of the author, and access control does not apply here. NAC [28] can be used in applications that require this level of access control.

[2]"Trust zone" was initially used, later changed to trust domain to avoid the confusion with the use of trust zone in hardware area.

defined for the name of received data. A *trust schema* is made of a set of rules that defines the security policies within a trust domain, guiding data producers to select the correct signing key (carried in Data packet's KeyLocator) and consumers to use correct keys to authenticate received packets. As an example, a trust schema determines the legitimate key to sign a given packet by specifying the relationship between the signer's semantic name and the packet's semantic name.

Given that Trust schema is confined to express the relationship between *semantic names*, to use trust schema to enforce security policies requires that all the information needed for decision making must be encoded into data names and/or key names. As an example, if an app wishes to limit the sensing data access to specific time periods, it must encode into the sensing data names the data production time. This restriction makes it easy to design languages to systematically express trust schema rules, and libraries to automate the execution of them. We review the existing trust schema languages and libraries next.

Given trust schema's critical role in enforcing security policies, it must be solely controlled by the domain controller, and be applied to authorize every action in the trust domain. Specific applications or entities in a trust domain may define more locally scoped trust rules for finer control over trust relations, in complement with the trust schema.

2.5 Previous Efforts in Trust Schema Support

ndn-cxx [14] is the first library to provide API support for trust schema. The validator provided in ndn-cxx allows developers to verify the authenticity and integrity of received data packets. To perform its task, the validator checks with the trust schema and fetches the needed certificates from the network to verify the signer is an acceptable entity for signing the received data. ndn-cxx did not implement the data signing support based on trust schema.

VerSec [10], together with its initial implementation in DCT [6], significantly by introducing a domain-specific language to write trust schema. The VerSec language enables application developers to use declarative expressions to define security rules for IoT messages using the Publish/Subscribe (Pub/Sub) communication model, and to specify the corresponding signing chains for these messages. In contrast to ndn-cxx, DCT assumes that all the needed certificates are synchronized among all the home IoT devices, eliminating the need to fetch signer certificates from the network.

Subsequently, the TypeScript library NDNts [8] provided partial support for the VerSec language in its trust schema impementation – it removed the Pub/Sub semantics. NDNts' signing interpreter assumes that the Public Information Base (PIB) [15] stores the already fetched certificates for each application. NDNts's authentication interpreter can then fetch certificates from either the PIB or the network, providing flexibility in certificate retrieval.

The evolution of the aforementioned trust schema implementations demonstrates a continuing progress in API support that is moving closer to meeting the needs of applications. This evolution follows a path that includes:

(1) providing a comprehensive implementation;
(2) introducing a schema language that is expressive and easy to use in writing trust schema;
(3) proposing a general API design for trust schema library.

Envelope takes the next step along the same path to design a new API that addresses the above three application needs in a coherent manner.

3 ENVELOPE: THE DESIGN GOALS

An important observation we made is that, applications have different preferences on certificates storage. In a smart home application [10, 27] where devices usually have limited storage resources, it is a natural choice to store identity certificate and trust anchor in-memory. Applications [3, 4, 12] that are capable to access local file systems prefer storing certificates on local file system (*i.e.* PIB). Besides entity's own certificate, applications also have different choices on retrieving other entities' certificates. Although the most common approach of retrieving producer certificate is to express Interest for it during the data validation time, application [10] in a closed network is more willing to let individuals pre-synchronizing on the certificate set of all entities. A general trust schema API designed for these applications (including the blog application in §2.1) should have a flexible certificate storage design that allows different implementations. Later in §7 we further discuss the security trade-off by enabling this flexibility.

Based on the lessons learned from the past effort in supporting trust schema and our observation on previous application development, we aim the design of our new trust schema API Envelope at satisfying the following goals.

- Supporting an easy-to-use language for application developers to write trust schema.
- Having a flexible certificate storage design that can leverage the existing NDN data storage solutions, both local and remote [16, 23] to offer high certificate availability for the interpreters.

We use TechDaily, as introduced earlier, as an illustrative example to show the utility of Envelope. TechDaily developers write a trust schema to define this trust relationship, and Envelope executes the trust schema during TechDaily's operations. Figure 2 is a block diagram of TechDaily's software architecture, which shows how it configures Envelope, produces and consumes blog data authenticated and authorized by the site's trust schema with Envelope API.

Security Assumptions: In this design, we assume that the operating system is trustworthy and capable of protecting file integrity within the local file system. We also assume the presence of a Trusted Platform Module (TPM) that securely stores private keys and enables applications to sign data with specific keys.

Next, we describe a easy-to-use trust schema language that we developed for Envelopethat can be to specify the security policies based on the trust relationship shown in Figure 2.

4 LIGHTVERSEC

We developed LightVerSec, a modified VerSec [10], to be used as the trust schema language in Envelope. In the following, we introduce LightVerSec from three perspectives: LightVerSec syntax, modifications from VerSec, and name schema tree.

LightVerSec Syntax: LightVerSec employs pattern matching on NDN names to validate packets. To validate a packet, a LightVerSec implementation first searches for a rule in the defined trust schema

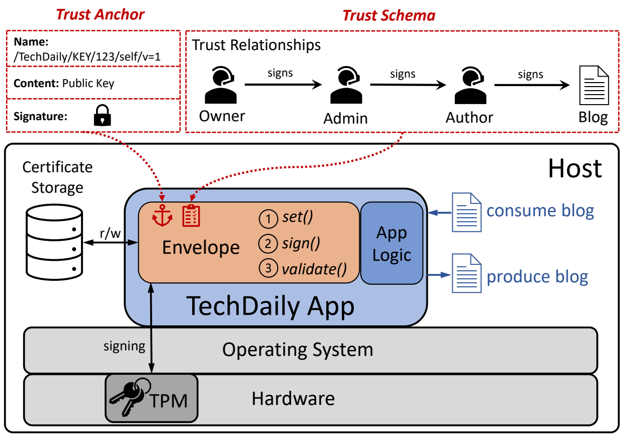

Figure 2: TechDaily's software architecture. The application consumes and produces blog data and realizes schematized trust using the three functions that Envelope provides. We also consider the certificate storage as a universal resource accessible to any NDN application running on the same host.

whose name pattern matches the packet name, and then checks whether the KeyLocator complies with the pattern specified by the rule.

In LightVerSec, a *name pattern* consists of a sequence of name components and pattern variables, separated by slashes. Name components are enclosed in quotes, while patterns are represented as C-style identifiers. For example:

```
"TechDaily"/"admin"/adminID/"KEY"/_
```

is a name pattern consisting of three name components and two pattern variables (adminID and _). A name pattern can only match a name when they have the same length and name components given in the name pattern equals to the components in the name at the corresponding positions. Then, the pattern variables are assigned with corresponding components in the given name. Variables starting with an underscore are considered as temporary variables and not assigned. Name patterns can be named with an identifier starting with a hash. When this identifier occurs in another name pattern, it is replaced by its definition.

A rule is defined in the form of

```
#PacketNamePattern <= #KeyLocatorNamePattern
```

Which means any packet matching "#PacketNamePattern" should have a key locator matching "#KeyLocatorNamePattern", and the pattern variables with same names assigned during the matching must agree. Extra restrictions can be expressed in the form of {var: func()}, where "var" is a pattern variable and "**func**" is a user defined function passed to the LightVerSec implementation at run time.

Figure 3 gives an example of TechDaily's trust schema. In this trust schema, six name patterns and three rules are defined; the three rules are the administrator rule #admin<=#root, the author rule #author<=#admin, and the article rule #article<=#author. In the #article<=#author rule, the "authorID" in the article's name is required to be the same as is in the name of the author's key. A user-defined function named isVersion() is used to restrict

```
#KEY: "KEY"/_/_/_version & {_version: isVersion()}
#site: "TechDaily"
#root: #site/#KEY
#admin: #site/"admin"/adminID/#KEY <= #root
#author: #site/"author"/authorID/#KEY <= #admin
#article: #site/"article"/topic/articleID/authorID/_version & {
      _version: isVersion()} <= #author
```

Figure 3: The policy of the TechDaily website

the last name component of every packet, which simply checks if the component is a valid version number. We provide the formal Backus–Naur Form (BNF) notation of LightVerSec in Appendix A.

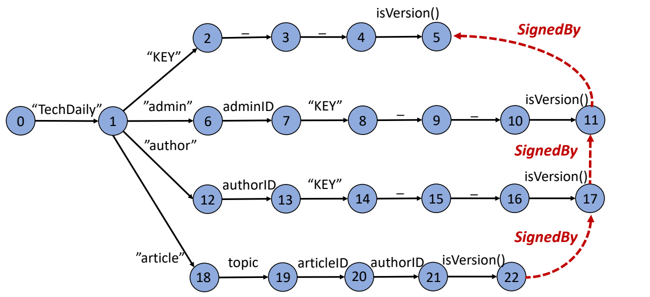

Figure 4: An example name schema tree compiled from the trust schema text

Modifications from VerSec: LightVerSec differs from the original VerSec in the following two aspects:

(1) *Decoupling from Pub/Sub.* VerSec has specific syntax and semantics defined for Pub/Sub, such as publication naming rules [11]. Though the core VerSec language does not require Pub/Sub implementation, it is hard to implement a VerSec compiler without Pub/Sub features. To target more general NDN applications and simplify compiler implementation, LightVerSec removes such Pub/Sub specific support. Internal functions in VerSec are also replaced by more flexible user-defined functions in LightVerSec, which are written by the application developers and passed to the implementation at runtime.
(2) *Focusing on Authenticating Received Packets.* When compiling a trust schema, the VerSec compiler can optionally embed internal information for building publications, such as signers, while LightVerSec does not optimize for Data producers, simplifying compiler implementation.

These modifications do not restrict the usage of the language. That is, all applications supported by VerSec are still supported by LightVerSec. The differences only affect the way to write the schema and use corresponding libraries.

Name Schema Tree: To optimize the execution of the trust schema, LightVerSec compiles the trust schema text into a trie-like data structure called a *name schema tree*. In this data structure, each edge represents a name component or pattern variable, and each leaf node corresponds to a name pattern of a packet. This name schema tree efficiently represents the structure of the trust schema.

When a trust schema is compiled into a name schema tree, rules are attached to the leaf nodes. These rules are represented by pointers that point to the KeyLocator's name patterns. This allows for quick and accurate matching of packet names against the defined trust schema. For example, Figure 4 illustrates the compiled name schema tree for TechDaily's trust schema. It demonstrates how the trust schema is transformed into a hierarchical structure for efficient validation and verification of packet names.

5 THE DESIGN OF ENVELOPE

In this section, we present the software design of Envelope. § 5.1 describes the certificate storage abstraction Box that Envelope relies on. Then § 5.2 discusses the high-level functions Envelope provides. Afterwards, § 5.3 explains how *set* function configures Envelope with trust anchor and trust schema. Finally, we demonstrate how Envelope signs and validates a packet in § 5.4 and § 5.5.

5.1 Certificate Storage Abstraction: Box

The Envelope design goals described in § 3 desire a flexible certificate storage design. Specifically, this certificate storage should have a straightforward yet extensible API, ensuring that: (i) The storage itself does not impose any specific storage model, such as a Relational Database or Key-Value Store. (ii) The design should be compatible with the existing certificate storage solutions, such as KeyChain and the NDN data repository, allowing TechDaily to use the existing software that has been developed by the NDN community. (iii) TechDaily developers can customize its certificate storage solution without modifying Envelope APIs.

To meet these design requirements, Envelope provides an abstraction layer for certificate storage, referred to as the *Box*. The Box abstraction includes the following two APIs:

- $get(prefix, lambda) \rightarrow cert$: When the Box receives a query from TechDaily in the form of a name prefix, its role is to return a matching certificate, if one exists. Optionally, TechDaily can specify a callback function, denoted as *lambda*, to enable certificate filtering. One common use case for filtering is to ensure that the returned certificate has a version number higher than a certain value.
- $put(cert)$: TechDaily has the capability to store a certificate in the storage by providing the complete certificate packet. The Box ensures that it is securely stored for future retrieval and usage.

5.2 Envelope Functions

The core of Envelope API has the following three high-level functions.

- $set(anchor, schema)$: This function is invoked by TechDaily entity immediately after the security bootstrapping process to configure Envelope with the acquired trust anchor and trust schema.
- $sign(box, tpm, usPkt) \rightarrow sPkt$: TechDaily utilizes the *sign* function to sign an NDN packet. This function requires a Box, a TPM (Trusted Platform Module) module, and an unsigned NDN packet as input. It then returns the signed NDN packet to TechDaily. During the signing process, Envelope queries the provided Box to retrieve an appropriate signing certificate. Afterwards, Envelope can access TPM, sign the packet and provide the signed packet back to TechDaily.
- $validate(box, sPkt) \rightarrow bool$: TechDaily utilizes this function to validate a signed packet. The function returns a boolean validation result based on the trust schema. During the validation process, Envelope queries the provided Box to fetch the signing certificate pointed to by the packet's KeyLocator.

Note that Envelope allows the *sign* and *validate* functions to use different Boxes. In a real-world deployment scenario, TechDaily may choose to store owned certificates locally on disk while relying on a remote data repository to provide other certificates along the signing chain. In this case, TechDaily would provide two different Boxes when invoking the two functions, allowing for flexibility and customization.

5.3 Envelope Configuration

When the TechDaily instance initializes, it invokes the *set* method to configure Envelope. In response to this call, Envelope compiles trust schema and stores the trust anchor and the compiled name schema tree on the local file system. These files are saved with a predefined and unique name that is associated with the trust domain name.

By consistently storing the trust anchor and name schema tree in the same location for each domain, Envelope ensures easy access and retrieval of the trust-related information whenever required by the application.

5.4 Data Signing

In this section, we demonstrate the signing workflow where a TechDaily author, Alice, wants to publish a blog with the name "`/TechDaily/article/icn/envelope/alice/v=1`". TechDaily utilizes the *sign* function to secure Alice's blog with trust schema.

In this scenario, Envelope first internally prepares a filter function (as the *lambda* defined in §5.1) that semantically check a certificate's signing legitimacy according to trust schema. Then, Envelope retrieves a suitable certificate using this filter.

Once the appropriate certificate is obtained, Envelope utilizes the corresponding private key stored in the TPM to generate the digital signature. Finally, the *sign* function returns the signed blog to TechDaily.

Certificate Indexing To optimize the signing process and avoid enumerating every certificate in the storage, Envelope utilizes the mappings between the nodes in the name schema tree and the corresponding certificate names.

To achieve this optimization, Envelope provides an auxiliary API called *index*. The *index* method takes a certificate name as input and returns the corresponding node ID in the name schema tree.

This mapping is stored as a *context tuple*, which includes the node ID, the pattern associated with the node, and the certificate name. For example, when Alice obtains an author certificate with the name "`/TechDaily/author/alice/KEY/789/bob/v=1`" from the administrator Bob, TechDaily can request Envelope to index her certificate.

During this indexing process, Envelope matches the certificate name against the name schema tree and identifies Node 17 along with the corresponding name pattern `{authorID:"alice"}`. The

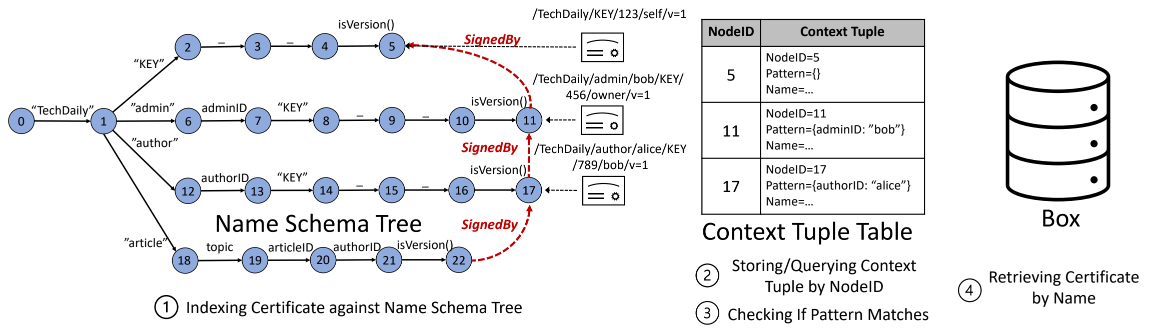
/TechDaily/KEY/123/self/v=1
/TechDaily/admin/bob/KEY/456/owner/v=1
/TechDaily/author/alice/KEY/789/bob/v=1
SignedBy
isVersion()
"KEY"
"TechDaily"
"admin"
adminID
"author"
authorID
"article"
topic
articleID
Name Schema Tree
NodeID
Context Tuple
5
NodeID=5 Pattern={} Name=...
11
NodeID=11 Pattern={adminID: "bob"} Name=...
17
NodeID=17 Pattern={authorID: "alice"} Name=...
Context Tuple Table
Box
1 Indexing Certificate against Name Schema Tree
2 Storing/Querying Context Tuple by NodeID
3 Checking If Pattern Matches
4 Retrieving Certificate by Name

Figure 5: The workflow of retrieving Alice certificate with the help of certificate indexing.

resulting context tuple, containing the node ID, pattern, and certificate name, is then stored in a table for efficient retrieval and reference.

Later when TechDaily needs to sign the packet "`/TechDaily/article/icn/envelope/alice/v=1`", Envelope matches the article name to Node 22 in the name schema tree. According to the name schema tree, Node 22 requires a certificate associated with Node 17, following the name pattern `{authorID:"alice"}`.

By querying the context tuple table, Envelope identifies the certificate "`/TechDaily/author/alice/KEY/789/bob/v=1`", which matches the name pattern `{authorID:"alice"}`. Once the context matching is successful, Envelope invokes the Box's *get* method to retrieve Alice's certificate bytes. It then locates the corresponding key in the Trusted Platform Module (TPM) for signing the packet.

Leveraging the structure of the name schema tree and the prebuilt mappings stored in the context tuple table, this approach significantly improves the efficiency of certificate lookup during the signing process.

5.5 Data Validation

TechDaily needs to *validate* data before consuming Alice blog. In this process, Envelope validate the signing relationships between the author (*i.e.* Alice) and the blog, the administrator (*i.e.* Bob) and the author, and the owner (*i.e.* the website trust anchor owner) and the administrator.

To enable customized data authentication, Envelope abstracts the validation process of each signing relationship as a *Pipeline*. Each Pipeline consists of a list of *Checkers*, which are defined as follows:

$Checker(sPktName, signature) \rightarrow bool$: A Checker accesses the signed packet's name and signature fields and performs specific checks on them. It then returns a boolean checking result.

The Checkers in the Pipeline allow for modular and customizable data validation. Developers can define their own Checkers to perform various checks on the signed packet's name and signature fields, ensuring the data legitimacy. This abstraction provides flexibility in implementing different validation rules based on the defined trust schema.

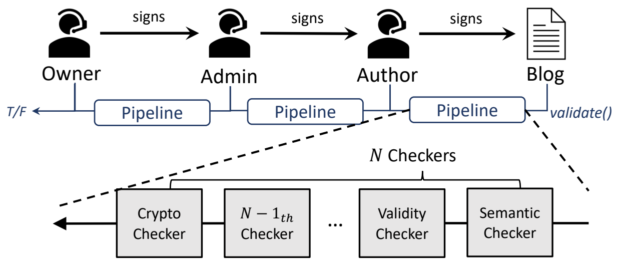

Figure 6: Envelope needs three Pipelines to fully validate Alice blog, and each Pipeline has the same list N Checkers.

Figure 6 illustrates TechDaily's N-Checker Pipeline, which includes a *SemanticChecker* responsible for examining the semantic binding between KeyLocator and the packet name. This is followed by a *ValidityChecker* that rejects expired certificates, along with other necessary Checkers to ensure the legitimacy of the data.

In the design of Envelope, the *CryptoChecker* serves as the final Checker in the Pipeline. As a unique Checker, it attempts to retrieve Alice's certificate from the Box. If the Box cannot provide Alice's certificate, the CryptoChecker tries to fetch the certificate from the network by expressing Interest with the name carried in the blog KeyLocator. Once Alice's certificate is obtained, the CryptoChecker accesses the full packet and employs Alice's public key to verify the signature. If the CryptoChecker succeeds, Envelopecompletes the blog Pipeline and proceeds to the Pipeline for validating Alice's certificate.

TechDaily developers have the flexibility to determine the composition of the Pipeline during software development. With Envelope's provided APIs, developers can design a Pipeline using predefined Checkers such as the SemanticChecker, ValidityChecker, and CryptoChecker, as well as implement their own customized Checkers.

Once the Pipeline is defined in the code, Envelope will execute the same set of Checkers for all subsequent calls to the *validate* function. The same Pipeline logic is repeatedly executed until the trust anchor is reached. If any Checker fails during the validation process, Envelope immediately terminates all subsequent Checkers and returns false. Once all Pipelines are completed, Envelope stores all certificates along the signing chain into the Box using the *put* operation. Finally, Envelope returns true to TechDaily, indicating that the validation process was successful.

Indeed, the Checker and Pipeline abstractions provided by Envelope offer application developers the ability to design their own packet validation logic and also create extensibility for systematically supporting additional security protocols in data validation. For example, certificate revocation checking [22] and long-lived data verification solutions [25] can be implemented as independent Checkers that can be integrated into the TechDaily Pipeline.

6 EVALUATION

This section describes the Envelope implementation and the evaluations that demonstrate the effectiveness of Envelope design.

6.1 Implementation

We have implemented the LightVerSec language as part of the python-ndn library [17] and developed Envelope as an application library based on python-ndn.

To verify the effectiveness of Envelope, we have also created the TechDaily application demoware, which utilizes Envelope to provide trust schema support. The TechDaily trust domain uses the same trust schema discussed in § 4, and it is configured with a validation Pipeline that includes the SemanticChecker, ValidityChecker, and CryptoChecker.

In the TechDaily domain, the domain controller obtains the public keys of administrators and authors through an out-of-band process and manually signs certificates. Before the application logic starts, TechDaily entities are security bootstrapped manually.

TechDaily employs two Boxes to facilitate data signing and validation in Envelope. The *DBBox* provides abstract interfaces to access a local database, while the *RepoBox* serves as the interface for interacting with networked data repositories. After the security bootstrapping process for each entity, the entity indexes all owned certificates and stores them in the DBBox. Whenever a new certificate is issued, the TechDaily domain controller and administrators use the *put* operation to store the new certificates in the RepoBox. Consequently, when a TechDaily entity needs to sign data, Envelope can efficiently retrieve the appropriate signing certificate from the DBBox. During packet validation, the entity utilizes the *get* operation to retrieve certificates from the RepoBox and executes the Pipeline for each signing relationship until the trust anchor is reached.

6.2 Qualitative API Comparison

Table 1 provides a comparison between Envelope and previous works, highlighting the unique features and capabilities of Envelope in contrast to other existing solutions.

Earlier libraries such as ndn-cxx [14], jndn [18], PyNDN [19] and the high-level API [20] based them primarily focused on data validation logic but does not include support for a trust schema or flexible certificate storage design. DCT [10], designed for home IoT applications, uses the VerSec language for writing trust schema. However, it does not allow applications to decide on the storage and retrieval of certificates during data signing and validation. NDNts [8] allows configuring the certificate retrieval location during data validation, but the options are limited to KeyChain and the network. It also lacks support for customizing data validation.

Trust Schema API	Completeness	Domain-Specific Language	Flexible Certificate Storage	Customized Data Validation
ndn-cxx [14]	✗	✗	✗	✗
PyNDN [19]	✗	✗	✗	✗
jndn [18]	✗	✗	✗	✗
DCT [10]	✓	✓	✗	✗
python-ndn [14]	✓*	✓*	✗	✗
NDNts [8]	✓	✓	✓/✗	✗
Envelope	✓	✓	✓	✓

*: LightVerSec Implementation.

Table 1: Comparison of Previous Work.

In contrast, Envelope provides a comprehensive solution that includes a domain-specific language (LightVerSec) for writing trust schema, flexible certificate storage through the Box abstraction, and the ability to customize data validation using the Pipeline and Checker abstractions. Overall, Envelope offers a more flexible solution compared to previous works, addressing the limitations and gaps identified in the existing implementations.

7 LESSONS LEARNED

During our trial-and-error efforts in developing Envelope, the first lesson we learnt is the need for a clearly defined goal of the trust schema support. Through multiple rounds of discussions, we concluded that a trust schema should focus on authentication and authorization only. To keep its implementation simple, the supporting language and libraries are preferably not to cover specific usage patterns.

Another important lesson we learned is the necessity of having a good abstraction for certificate storage. We started with using the DBBox design mentioned in § 6.1. However, we soon realized that this design was not suitable for applications that prefer to store certificates in networked repositories [16] or distributed ledgers [23, 26]. Additionally, running databases may be infeasible in resource-constrained application scenarios [10, 27], due to insufficient memory or the lack of a file system. To address these identified issues, we developed the solution of abstracting certificate storage as an external module that applications can provide based on their available resources or specific constraints. Envelope provides two simple read and write interfaces: *put* and *get*. This abstraction allows applications to make use of their local resources, while avoiding the need for Envelope implementations to reinvent storage solutions.

8 DISCUSSION

Flexibility versus Vulnerabilities: It is arguable that more flexibility might lead to more attack vectors. For example, the certificate storage implemented by a third party could be vulnerable. However, depending on specific application scenarios, a trade-off for felxibility could be preferred to build a practical network. Suppose an enterprise internal network with the following properties [2]:

(1) Security policies are centrally managed by specific administrators.
(2) A handful of local services developed by different parties (e.g., mail services, printers, instant messaging, file sharing, source repositories, relational databases).

In such a scenario, the development of services and the specification of security policies are completely decoupled: administrators typically have no access to source code, and the developers can not predict what policies will be specified. Since services are developed by separated parties, the developers can choose different libraries and security storages to use. A unified, flexible framework to specify security policies is more affordable.

9 CONCLUSION

The trust anchor, certificates, and the trust schema are three fundamental components in the NDN's security solutions. Among the three, the trust schema is uniquely enabled by NDN's use of semantically meaningful names for all types of data. Semantically meaningful names enable one to reason about security in a system's operation, and well defined naming conventions can further simplify the reasoning. As a result, trust schema allows applications to systematically execute security checking operations.

This paper provides an overview of the existing efforts in trust schema support and contributes a new trust schema API, Envelope. The Envelope API addresses the need for comprehensive trust schema functionalities, an easy-to-use schema language (LightVerSec), and an extensible and flexible design. Envelope introduces the abstractions of Pipelines and Checkers to enable customized data validation, and an abstract certificate storage API *Box*, which allows developers to use their preferred solutions for certificate storage. We implemented Envelope and use a blog application to demonstrate its effectiveness and usability. Our results show that Envelope successfully executes trust schema for applications with acceptable performance overhead.

As next step, We plan to further evaluate and test the Envelope API design with more complex application scenarios, making Envelope a useful tool to support the community's efforts in applying NDN to solve most challenging problems facing today's Internet.

A LIGHTVERC FORMAL GRAMMAR

```
TAG_IDENT = CNAME;
RULE_IDENT = "#", CNAME;
FN_IDENT = "$", CNAME;
CNAME = ? C/C++ identifiers ?;
STR = ? C/C++ quoted string ?;

name = ["/"], component, {"/", component};
component = STR
          | TAG_IDENT
          | RULE_IDENT;

definition = RULE_IDENT, ":", def_expr;
def_expr = name, ["&", comp_constraints], ["<=",
     sign_constraints];
sign_constraints = RULE_IDENT, {"|", RULE_IDENT};
comp_constraints = cons_set, {"|", cons_set};
cons_set = "{", cons_term, {",", cons_term}, "}";
cons_term = TAG_IDENT, ":", cons_expr;
cons_expr = cons_option, {"|", cons_option};
cons_option = STR
            | TAG_IDENT
            | FN_IDENT, "(", fn_args, ")";
fn_args = (STR | TAG_IDENT), {",", (STR | TAG_IDENT)};

file_input = {definition};
```

Listing 1: LightVerSec Grammar

REFERENCES

[1] Alexander Afanasyev, Tamer Refaei, Lan Wang, and Lixia Zhang. 2018. A Brief Introduction to Named Data Networking. In *Proc. of MILCOM*.
[2] Martin Casado, Tal Garfinkel, Aditya Akella, Michael J. Freedman, Dan Boneh, and Nick McKeown. 2006. SANE: A Protection Architecture for Enterprise Networks. In *15th USENIX Security Symposium (USENIX Security 06)*. USENIX Association, Vancouver, B.C. Canada. https://www.usenix.org/conference/15th-usenix-security-symposium/sane-protection-architecture-enterprise-networks
[3] Saurab Dulal, Nasir Ali, Adam Robert Thieme, Tianyuan Yu, Siqi Liu, Suravi Regmi, Lixia Zhang, and Lan Wang. 2022. Building a secure mhealth data sharing infrastructure over ndn. In *Proceedings of the 9th ACM Conference on Information-Centric Networking*. 114–124.
[4] Ashlesh Gawande, Jeremy Clark, Damian Coomes, and Lan Wang. 2019. Decentralized and secure multimedia sharing application over named data networking. In *Proceedings of the 6th ACM Conference on Information-Centric Networking*. 19–29.
[5] Van Jacobson, Diana K Smetters, JD Thronton, Michael F Plass, Nicholas H Briggs, and RL Braynard. 2009. Network Named Content. *CoNEXT* (2009).
[6] Pollere LLC. 2023. https://github.com/pollere/DCT. (2023). Accessed: 2023-5-27.
[7] Satoshi Nakamoto. 2009. Bitcoin: A Peer-to-Peer Electronic Cash System. *Cryptography Mailing list at https://metzdowd.com* (03 2009).
[8] NDNts. 2023. https://github.com/yoursunny/NDNts. (2023). Accessed: 2023-5-27.
[9] Kathleen Nichols. 2019. Lessons learned building a secure network measurement framework using basic ndn. In *Proceedings of the 6th ACM Conference on Information-Centric Networking*. 112–122.
[10] Kathleen Nichols. 2021. Trust schemas and ICN: key to secure home IoT. In *Proceedings of the 8th ACM Conference on Information-Centric Networking*. 95–106.
[11] Kathleen Nichols. 2022. The VerSec Trust Schema Compiler. (2022). https://github.com/pollere/DCT/blob/main/tools/compiler/doc/language.pdf
[12] Justin Presley, Xi Wang, Tym Brandel, Xusheng Ai, Proyash Podder, Tianyuan Yu, Varun Patil, Lixia Zhang, Alex Afanasyev, F. Alex Feltus, and Susmit Shannigrahi. 2022. Hydra – A Federated Data Repository over NDN. (2022). arXiv:cs.NI/2211.00919
[13] Eric Rescorla and Tim Dierks. 2008. The Transport Layer Security (TLS) Protocol Version 1.2. RFC 5246. (Aug. 2008). https://doi.org/10.17487/RFC5246
[14] NDN Team. 2023. https://named-data.net/doc/ndn-cxx/current/. (2023). Accessed: 2023-5-27.
[15] NDN Team. 2023. https://redmine.named-data.net/projects/ndn-cxx/wiki/PublicKey_Info_Base. (2023). Accessed: 2023-5-27.
[16] NDN Team. 2023. https://github.com/UCLA-IRL/ndn-python-repo. (2023). Accessed: 2023-5-27.
[17] NDN Team. 2023. https://github.com/named-data/python-ndn. (2023). Accessed: 2023-5-27.
[18] NDN Team. 2023. https://github.com/named-data/jndn. (2023). Accessed: 2023-5-27.
[19] NDN Team. 2023. https://github.com/named-data/PyNDN2. (2023). Accessed: 2023-5-27.
[20] Jeff Thompson, Peter Gusev, and Jeff Burke. 2019. Ndn-cnl: A hierarchical namespace api for named data networking. In *Proceedings of the 6th ACM Conference on Information-Centric Networking*. 30–36.
[21] Tianyuan Yu, Philipp Moll, Zhiyi Zhang, Alexander Afanasyev, and Lixia Zhang. 2021. Enabling Plug-n-Play in Named Data Networking. In *MILCOM 2021-2021 IEEE Military Communications Conference (MILCOM)*. IEEE, 562–569.
[22] Tianyuan Yu, Hongcheng Xie, Siqi Liu, Xinyu Ma, Xiaohua Jia, and Lixia Zhang. 2022. CertRevoke: a certificate revocation framework for named data networking. In *Proceedings of the 9th ACM Conference on Information-Centric Networking*. 80–90.
[23] Tianyuan Yu, Hongcheng Xie, Siqi Liu, Xinyu Ma, Varun Patil, Xiaohua Jia, and Lixia Zhang. 2023. CLedger: A Secure Distributed Certificate Ledger via Named Data. (2023).
[24] Yingdi Yu, Alexander Afanasyev, David Clark, KC Claffy, Van Jacobson, and Lixia Zhang. 2015. Schematizing trust in named data networking. In *Proceedings of the 2nd ACM Conference on Information-Centric Networking*. 177–186.
[25] Yingdi Yu, Alexander Afanasyev, Jan Seedorf, Zhiyi Zhang, and Lixia Zhang. 2017. NDN DeLorean: An authentication system for data archives in named data

networking. In *Proceedings of the 4th ACM Conference on Information-Centric Networking*. 11–21.

[26] Zhiyi Zhang, Vishrant Vasavada, Xinyu Ma, and Lixia Zhang. 2019. Dledger: An iot-friendly private distributed ledger system based on dag. *arXiv preprint arXiv:1902.09031* (2019).

[27] Zhiyi Zhang, Tianyuan Yu, Xinyu Ma, Yu Guan, Philipp Moll, and Lixia Zhang. 2022. Sovereign: Self-contained smart home with data-centric network and security. *IEEE Internet of Things Journal* 9, 15 (2022), 13808–13822.

[28] Zhiyi Zhang, Yingdi Yu, Sanjeev Kaushik Ramani, Alex Afanasyev, and Lixia Zhang. 2018. NAC: Automating access control via Named Data. In *MILCOM 2018-2018 IEEE Military Communications Conference (MILCOM)*. IEEE, 626–633.

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