Self-verifiable Content using Decentralized Identifiers

Nikos Fotiou and Yannis Thomas and Vasilios A. Siris and George Xylomenos and George C. Polyzos
Mobile Multimedia Laboratory, Department of Informatics
School of Information Sciences and Technology
Athens University of Economics and Business
Patision 76, Athens 10434, Greece
E-mail: {fotiou, thomasi, vsiris, xgeorge, polyzos}@aueb.gr

***Abstract*—We propose a self-sovereign and decentralized naming scheme for Information-Centric Networking (ICN) architectures that makes content items self-verifiable. Our scheme is based on Decentralized Identifiers (DIDs), an emerging identification scheme under standardization by the W3C. With our solution, DIDs are used as content name prefixes and act as the root of trust for the corresponding namespaces. A content item in our scheme includes tamper-proof metadata that can be used to verify its authenticity and integrity based only on the content's name, without relying on a trusted third party. The owner of a DID can authorize content providers to publish items under a sub-space of its namespace, or even fully delegate the management of a sub-space of its namespace to a controlling entity. We implement our scheme for the Named Data Networking (NDN) architecture and show that it removes the need for trusted third parties, enables hassle-free key rotation, and allows joint namespace ownership and namespace management delegation, with negligible computational and space overhead.**

***Index Terms*—ICN, DID, NDN**

I. Introduction

Content naming is the focus of many research efforts, especially in the context of Information-Centric Networking (ICN) [1], where content is directly identified by the network. An important issue in any ICN architecture is how to securely bind a content item to its name, in other words, how the *authenticity* of an item can be verified. Content authenticity verification is of particular importance, as ICN-based architectures are susceptible to *content poisoning* attacks [2], where malicious users pollute the network with fake content. This kind of attack is further amplified due to the native support of caching by ICN: if fake content is cached, it cannot be easily evicted from caches [3]. Traditionally, this challenge is handled with *trust anchors*, such as the digital certificates used in legacy Internet systems, or the *keyLocator* scheme [4] of the Named-Data Networking (NDN) architecture [5]. These trust anchors, however, rely on the existence of trusted third parties (TTPs).

In this paper we propose a content naming scheme which is *self-certifying*, in the sense that the authenticity and the integrity of a content item can be verified based solely on the content name and the content item's metadata, *decentralized*, in the sense that it does not depend on any third party service for functioning, and *self-sovereign*, in the sense that users, independently and autonomously, can generate and name content items in their namespace, authorize others to do so, or even delegate control of (part of) their namespace to others, without any need for TTPs. Content name prefixes in our scheme are 256 bit thumbprints of public keys, generated by the content owner; therefore, prefixes are statistically unique and content name collisions across different content owners are impossible, without requiring a "name registration service" (e.g., see [6], [7]), at the cost of making name prefixes non human-readable.

We achieve these goals by leveraging *Decentralized Identifiers* (DIDs). A DID is a new type of self-administered, globally unique identifier, which is *resolvable* and *cryptographically verifiable* [8]. These properties are achieved by associating a DID with a *DID document*, which includes public keys and auxiliary information that can be used to securely link the DID to its *owner*. DID documents are usually stored in a *registry* [9], which is trusted to implement proper access control and DID document resolution. Our scheme adapts a DID *method* of our own design, known as `did:self` [10] (see Section III-C), which is compatible with the W3C specifications but does not rely on a trusted DID registry, having instead DID owners disseminate DID documents by themselves. We take advantage of this property to include the DID documents in the metadata of the content items, thus making the content items self-verifiable.

In our scheme, DIDs are used as content name *prefixes*; any content items under that prefix belong to a protected *namespace* controlled by the owner of the DID. In this manner, the binding between a content name and the public key of its owner can be achieved without a TTP; this binding is essential for achieving content authenticity [11]. Each protected content item is accompanied by some metadata, which include a DID document that corresponds to the DID of the content owner and an *attestation*, i.e., information for verifying the item's integrity and authenticity. The attestation is protected by a digital signature, which can be validated using a public key specified in the included DID document. By leveraging the properties of `did:self`, any entity can verify the correctness of the DID document associated with a content item, extract the appropriate public key, and verify the attestation's signature. Furthermore, the owner of a DID can authorize a third party to sign attestations under a part of its namespace, or

This preprint has not undergone peer review or any post-submission improvements or corrections. The Version of Record of this article is published in Computer Networks, and is available online at https://doi.org/10.1016/j.comnet.2023.109799

even to fully manage subsets of its namespace. These features enable secure and controlled management of the namespace (see Section III-C).

Our metadata-based approach has significant advantages compared to existing solutions that rely solely on content names to achieve the same properties, e.g., the InterPlanetary File System (IPFS) [12] which uses *content hashes* as names to bind each content item to its name. Those solutions, although simpler, have many drawbacks when it comes to *mutable* content items, since every time an item is modified its hash, and therefore its name, changes completely. We discuss these issues in more detail in Section II.

Compared to public key-based solutions, our DID-based approach facilitates key management, ownership transfer, and joint ownership. For example, in an IoT installation our scheme allows each IoT device to rotate its keys, without affecting the rest of the system; in contrast, in a system based on digital certificates, an IoT device would require an updated certificate for the new key.

Last but not least, in our system content names are location independent, which is significant in architectures such as ICN, where content should not be tied to a network endpoint.

Overall, our paper makes the following contributions:

- We present and analyze our `did:self` DID method for building *self-certifiable* content items, i.e., content items whose authenticity can be verified based solely on their DID-based name prefix.
- We leverage the properties of DIDs to enable content owners to authorize third parties to vouch for the authenticity of content items published under a specific subspace of their DID-based namespace.
- We allow content owners to delegate the management of specific sub-spaces of their DID-based namespace to third parties, while retaining the authority to modify authorizations and delegations.
- We implement our scheme for the NDN ICN architecture and integrate it with NDN's prototype implementation.

The remainder of this paper is organized as follows. We discuss related work in this area in Section II. In Section III we provide an introduction to NDN, present a reference scenario to motivate the goals of our scheme and describe the `did:self` DID method. In Section IV we present the design of our scheme and how it meets our goals. We discuss the implementation of our scheme in Section V and evaluate its performance and security aspects in Section VI. Finally, we outline future work in Section VII.

II. Related work

The most straightforward approach to implement self-verifiable content items is to include a hash of the content in the content's name. This approach has been widely studied even beyond the context of ICN. For example, Fu et al. [13] propose a file system for read-only data where the hash of the content name is also part of the file name. RFC 6920 [14] defines how the hash of a content item can be used as a name. Kuhn et al. [15] define a URI type that includes a cryptographic hash of the content. The *InterPlanetary File System* (IPFS), which is a global-scale p2p network used for file storage and sharing, also uses content names derived from hashing content items [12]. It is no surprise that the same approach has also been considered by many ICN-related efforts. Baugher et al. [16] propose a self-verifying name solution for the Content-Centric Networking (CCN) architecture, that uses a content hash as the content's name for read-only data. Similarly, Won and Nikander [17] propose a naming solution for ICN where content names include the content hash. Safdar et al. [18] propose a naming scheme for vehicular networks based on CCN where the content hash is part of the content name, whereas Arshad et al. [19] propose a similar approach for an ICN-based IoT architecture.

Using a content item's hash as its name is problematic for content items which are often modified, for example, the index page of a website, since whenever an item is modified, it must also be renamed. This has a dual impact: first, a resolution system is needed to allow users to look up the name of the latest version of an item and, second, evicting the old versions of content items from caches and replication points is slow, since from the network's perspective, a new version of an item is just another item. Our system relies on the content item's name prefix and some metadata embedded in the content item to achieve its security properties. Therefore, existing name-based solutions for managing versions of the same item can still be used (e.g., [20]). By allowing these solutions, there is no need for a lookup service to provide the name of the current version of an item, while in-network storage elements can distinguish among various versions of an item and implement smarter eviction mechanisms. Both hash-based approaches and our solution result in non human-readable content names. However, in our solution only a content owner-specific prefix in non-readable: the remainder of the name can be human-readable.

Another approach for implementing self-verifiable content items is to include the public key of the content owner in the content name, and then use this key to sign *metadata* that ensures content integrity and authenticity. This is an approach with considerable history: back in 1998, Mazieres and Kaashoek proposed a decentralized file system where filenames are prefixed by the hash of a public key controlled by the "host" of the file [21].

The DONA architecture was a significant early effort to use self-certified names in ICN [22]. In DONA each content item is identified by a pair of labels, P and L, where P corresponds to the public key of the content owner, and L to a data label. Each content item in DONA includes in its header a digital signature generated using P. A similar approach is followed by Ghodsi et al. [23]. The drawback of these approaches is that the content (and key) owner is the only entity that can sign the metadata, therefore, in order to delegate storage of *mutable* items to third party entities (e.g., a CDN provider), owners have to share their signing key with them. Our solution does not have this limitation, allowing content owners to *authorize* third parties to publish data in a secure and controllable way.

The naming scheme of the NetInf architecture is very close to our system [24]. NetInf uses as part of the content name the hash of a public key P. The private key that corresponds to P

is used to sign a metadata field that includes, among others, the hash of the content item, verifying this way its authenticity and integrity. Furthermore, using a chain of certificates rooted at P, other keys can be authorized to sign the metadata field. Our solution uses the emerging standard of DIDs, instead of the custom certificates of NetInf, and it offers some additional security features, e.g., it allows a delegee to rotate its key without having to receive a new "authorization".

Other similar approaches use alternatives to public keys, in order to combine human readable names with content-based security. For example, Zhang et al. [25] leverage *Identity-Based Encryption* (IBE) to build an ICN system where human readable names are also public keys. A similar approach is proposed by Hamdane et al. [26]. IBE suffers from the so-called *key escrow problem* where a centralized trusted entity (the key generator) must know all private keys. In our solution, all private keys are secret. Similarly, Ramani et al. [27] use *Attribute-Based Encryption* (ABE) to sign content metadata. An advantage of their approach is that it adds a level of privacy, since it is possible to prevent the correlation of two digital signatures, as many entities may share the same attribute. On the other hand, such solutions are not scalable, since all entities must agree on a common set of attributes and trust the same attribute issuing entity. Our solution does not rely on third parties that would limit its scalability.

Many schemes achieve the same goals as our system by using *trust anchoring* instead of self-certifying names. For example, Yu's solution [7] implements a PKI-like approach that relies on trusted *Certificate Authorities* (CAs). Other solutions try to improve this scheme by introducing alternatives to CAs, e.g., "neighbor-based trust" [28], or a blockchain [29]. Nevertheless, the security of these solutions depends on the trust anchoring service. Moreover, these approaches are inflexible when it comes to defining trust relationships. Motivated by the latter limitation and inspired by the work of Blaze et al. [30], Yu et al. [31] proposed the use of "trust schemata" that allow each entity to define "rules" for managing trust relationships. Our solution is similar to [31] but it shifts trust management to the content producer. With our solution, a content producer can define trust management rules, akin to [31], and securely embed them in the content items themselves.

III. Background and System Overview

In this section we provide an introduction to NDN, we outline a reference scenario that is later used to explain our solution's operations and illustrate its capabilities, and, finally, present DIDs and the `did:self` method.

A. An NDN Primer

NDN is probably the most popular ICN architecture, with an active research community, an evolving prototype implementation, and a large network testbed for experimentation. In NDN, everything revolves around content items. Content consumers issue *Interest* messages to request items, which content producers return using *Data* messages; all messages carry the name of a content item. NDN names are hierarchical, but not necessarily human-readable.

All NDN nodes (routers and hosts), maintain three data structures. The *Forwarding Information Base* (FIB) maps content name prefixes to the output port(s), also called *face(s)* in NDN parlance, that should be used to forward Interests towards appropriate data sources; note that faces can lead to both network interfaces (when the Interest must be forwarded to another node) and local applications (when the application can provide the content item). The *Pending Interest Table* (PIT) records the face(s) from which active Interest messages have arrived, i.e., those Interests for which Data messages are still expected; PIT entries are consumed by returning Data messages. Finally, the *Content Store* (CS) is a local cache for content items.

When an Interest is received by an NDN node, the node extracts the content name and checks if the requested content item has been cached in the CS or is locally hosted; if so, the content item is returned in a Data message through the incoming face and the Interest is discarded. Otherwise, the FIB is checked in order to decide where to forward the Interest. If a matching entry is found in the FIB, the Interest is sent through the appropriate face and its incoming face is added to the PIT. Data messages are routed back to the consumers hop-by-hop, using the pointers stored by the Interests in the PITs; when a Data message is received, we look up in the PIT where the corresponding Interest came from, forward the Data through that face, and delete the PIT entry.

NDN supports the cryptographic binding of names to content items, by including in each Data message a signature, covering both the name and the content, as well as a pointer to the key used for signing. Any NDN node can therefore verify the binding between the name and the content item included in a message. However, to verify that the information comes from an authorized source, a node must trust the owner of the public key used for signing.

B. Reference scenario and requirements

To illustrate the goals of our solution, we consider the *smart city platform* depicted in Figure 1, which uses NDN's hierarchical naming scheme. The platform includes a number of IoT devices, each producing content items whose names start with a platform-specific prefix, e.g. "<platform>". An important requirement in this scenario is that the platform owner should be able to *authorize* an IoT device to publish information under a sub-space of the "<platform>" namespace. For example, the drone on the bottom of the figure could be allowed to publish information, but only under the prefix "<platform>/roadB23/traffic". Each device should vouch for the authenticity of the items it produces by signing them with its own private key.

In order to further decentralize management, the platform owner should also be able to delegate the administration of sub-spaces of the root namespace to third parties. For example, the "<platform>/smart-building1" sub-space, could be *delegated* to a building administrator; the building administrator could then *authorize* an energy provider to publish information related to the energy consumption/production of the building, a health service to publish health related alerts, and a building

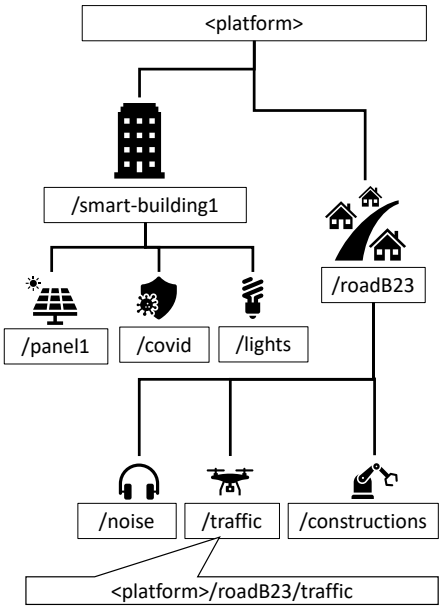

Fig. 1. A smart city platform.

management service to publish information about the status of the building lights. Even though a sub-space has been delegated, the platform owner should retain the right to modify delegations and authorization.

Both publication authorization and delegation of namespace management should fulfill the following additional requirements:

- Content consumers should be able to verify the authenticity of the published content solely by knowing the prefix used by the platform; in other words, that prefix is the trust anchor.
- Authorized content producers should be able to rotate their signing keys without having to "re-new" their authorization. This is important in the smart city scenario, where the producers can be IoT devices with intermittent network connectivity.

C. Decentralized Identifiers and `did:self`

DID-based decentralized identification system associate a *DID* with a *DID document*. The DID is used as a globally unique identifier of the DID owner, whereas the DID document includes information that can be used for interacting securely with the DID owner. More specifically, a DID document includes verification *methods*, usually public keys, as well as verification *relationships* that define the purpose of a verification method, e.g., a public key may be used to authenticate a DID owner, or to verify a digital signature generated by the DID owner. The W3C recommendation for DIDs encourages the specification of DID *methods* by third parties. Each DID method defines its own DID format (which must be in the form of a URI, prefixed with "did:" and followed by the DID method name), as well as how DID documents are resolved. In this paper we use our own `did:self` method, which is included in the W3C registry of DID methods.[1]

[1]https://w3c.github.io/did-spec-registries/#did-methods

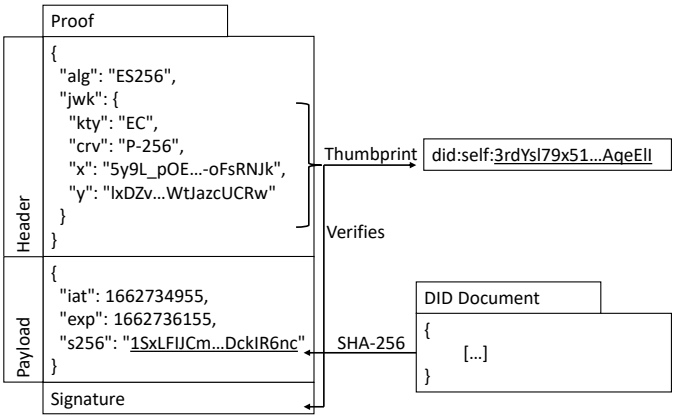

Fig. 2. Components of did:self. A did:self DID is the thumbprint of a public key. The corresponding private key is used to sign a proof that includes the hash of the DID document.

In `did:self` a DID is a thumbprint of a *JSON Web Key* (JWK) [32] prefixed with "`did:self:`". This key is owned and managed by the DID owner. A *DID document* is encoded using JSON and may include any of the DID "properties" included in the DID specifications. The binding between a `did:self` DID and a DID document is achieved using a *proof*, which is a "compact serialization" of a *JSON Web Signature* (JWS) (section 3.1 of [33]) generated by the DID owner. The *header* of such a proof includes the following *claims*:

- `alg`: The algorithm used to generate the signature of the proof (a list of supported algorithms is defined in Section 3 of RFC 7518).
- `jwk`: The public key that corresponds to the private key used to generate the signature of the proof. This key is expressed as a JWK and its thumbprint must match the `did:self` identifier.

The *payload* of the proof includes the following *claims*:

- `iat`: The proof's generation time.
- `exp`: The proof's expiration time.
- `s256`: A SHA-256 hash of the DID document, encoded in the base64url format.

Given a `did:self` identifier, a DID document and its proof, one can verify that the DID document is a valid document for the given identifier as follows:

1) Check that the `did:self` identifier is equal to the thumbprint of the public key in the `jwk` field in the proof's header.
2) Verify that the proof is still valid using the `exp` and `iat` fields.
3) Calculate the SHA-256 hash of the DID document and verify that it is equal to the `s256` field of the proof.
4) Verify the signature of the proof using the `jwk` in the proof's header.

The components of `did:self` (the DID, the DID Document and the proof), and their relationships are illustrated in Figure 2.

IV. System Design

In this section we explain how our scheme operates in detail, outlining its component entities, the operations on them and the rationale for our design.

A. System entities

Our system design includes content *producers* that publish and advertise content items, and content *consumers* that express interest in content items. The names of the content items are hierarchical and unique: two items with the same name *must* be the same item. Namespaces for "self-verifiable" content items are rooted at a `did:self` DID, that is, the DID is the prefix of the namespace; therefore, it is impossible to have naming conflicts across namespaces. These namespaces are owned by the *owner* of the corresponding DID. An owner can publish content items under its namespace, can *authorize* other producers to publish content items under a specific *sub-space* of its namespace, and can *delegate* the administration of a specific sub-space of the namespace to a third party *controller*. A self-verifiable item includes some metadata: using only these metadata, the integrity of the content item and its authenticity can be verified. Furthermore, if a self-verifiable item has been published by an entity other that the content owner, it can be verified whether this entity has been authorized by the content owner.

B. DID document properties

In order to achieve the desired functionality, a DID document in our system includes the following standard fields and properties (see also Figure 3):

- `id`: The `did:self` DID which the document corresponds to (line 2 in Figure 3).
- `verificationMethod`: A list of public keys expressed using the "JsonWebKey2020" encoding [9]. Each entry in the list includes an `id` property, which, prefixed with the DID itself, is used as a unique identifier for this verification method (i.e., for this key). Each entry also includes a `publicKeyJwk` property, which contains the actual public key expressed as a JSON Web Key (JWK) [34]. In the example of Figure 3, the `verificationMethod` is included in lines 3–14. Lines 4–12 define a single public key, identified by "#key1".
- `assertion`: An identifier of a public key defined in a DID document; note that this public key may be defined in *another* DID document. The corresponding private key is used for signing content *attestations* (see the following sub-section). In the example of Figure 3, the `assertion` in line 15 indicates that "#key1", defined in the same document, will be used for verifying content attestations. Note that we use different keys to sign content attestations (the assertion key) and DID document proofs (the key corresponding to the DID itself).

Our system also introduces a new DID document property named `caveats`. This property includes an array of content names, which is used to restrict the *scope* of the corresponding DID document, i.e., the DID document can be used to validate only content items prefixed by a name included in the `caveats` property. In the example of Figure 3, the `caveats` property in lines 16–18 indicates that the DID document can only be used to validate content items prefixed by the `did:self` identifier, followed by the word "photos/" (line 17). Finally a DID document, instead of the `assertion` property, may include the standard property `controller` (not shown in Figure 3), whose value must be a `did:self` DID: the DID document corresponding to the `controller` will then include the appropriate `assertion` properties.

```
1. {
2.   "id": "did:self:3rdYsl...9cs0iUfNAqeElI",
3.   "verificationMethod": [
4.     {
5.       "id": "#key1",
6.       "type": "JsonWebKey2020",
7.       "publicKeyJwk": {
8.         "kty": "EC",
9.         "crv": "P-256",
10.        "x": "5y9L_pOEye...oFsRNJk",
11.        "y": "lxDZvayjRU...JazcUCRw"
12.      }
13.    }
14.  ],
15.  "assertion":"did:self:3rdYsl...9cs0iUfNAqeElI#key1",
16.  "caveats" :[
17.    "photos/"
18.  ]
19. }
```

Fig. 3. A DID document used in our system.

C. Operations

In the remainder of this section, we assume that all operations take place under a protected namespace rooted at a `did:self` DID, referred to as DID_{root}. We will refer to the owner of DID_{root} as $Owner_{root}$.

1) Self-verifiable item creation: A content item becomes self-verifiable by adding to it some *metadata* that include a `did:self` *header* and a signed *attestation*. The `did:self` header is a sequence of DID documents and proofs that (securely) lead to a public key, which is used in an `assertion` verification relationship. This key, referred to as the *assertion key*, is used to verify the signature of the attestation. An attestation is a compact serialization of a JWS that securely binds the content to its name. The payload of an attestation includes the content's *name* and a base64url encoded hash of the content item calculated using SHA-256. This payload may also include other auxilliary information.

Figure 4 provides an example of a self-verifiable content item. The `did:self` header of this item includes a DID document for DID_{root} and the corresponding document proof (omitted for clarity reasons). As the figure shows, the included DID document defines a public key and uses it in the `assertion` verification relationship: this key, therefore, is used to verify the signature of the attestation.

2) Producer authorization: Consider now a producer P that owns a public key Pub_P, for example, the drone in Figure 1. $Owner_{root}$ can authorize P to create self-verifiable

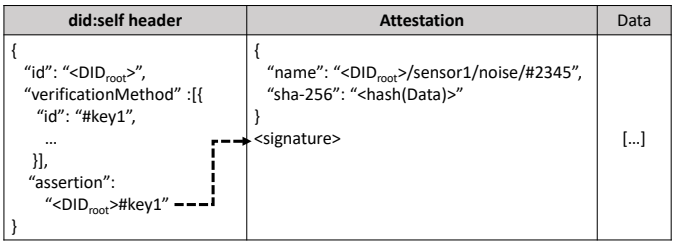

Fig. 4. A self-verifiable content item, generated by $Owner_{root}$. The arrows show which key can be used to verify each signature.

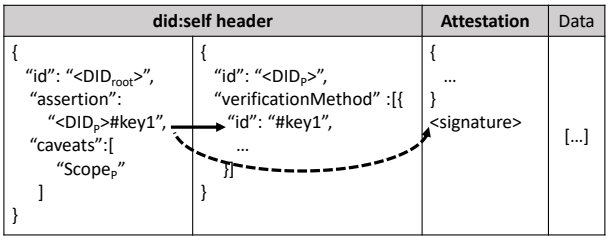

Fig. 5. A self-verifiable content item, generated by an authorized producer. Note that the assertion property of DID_{root} includes the identifier of a key defined in the DID document of DID_P.

content items under $DID_{root}/Scope_P/$, where $Scope_P$ can be any valid prefix. The simplest way to achieve this is by creating a DID document for DID_{root} that includes $Scope_P/$ in its `caveats`, Pub_P in a verification method, and the `id` of Pub_P in the `assertion` verification relationship. Then, $Owner_{root}$ transmits that document and its proof to P, and P uses them in the `did:self` header of its self-verifiable items. The metadata of such an item will be similar to those in Figure 4; the only difference is that the secret key belongs to P and not to $Owner_{root}$. Note that P cannot modify the DID document, as only $Owner_{root}$ can generate a proof for it, therefore P cannot change its signing key.

As an alternative, consider a producer P that owns a `did:self` identifier DID_P and defines in its DID document a public key with identifier $DID_P\#key$. $Owner_{root}$ can then authorize P to create self-verifiable content items under sub-space $DID_{root}/Scope_P/$ by creating a DID document that includes $DID_P\#key$ in the assertion property. To make content items self-verifiable, P has to include in their `did:self` header the DID document generated by $Owner_{root}$, as well as the DID document that corresponds to DID_P (and the corresponding proofs), as shown in Figure 5. The first DID document, generated by $Owner_{root}$, includes in its assertion property a key, which is defined in the second DID document, generated by P.

Although both approaches authorize P to create self-verifiable content items under a sub-space of the $Owner_{root}$ namespace, the former approach authorizes a specific key owned by P, while the latter approach allows P to *change* the public key that corresponds to $DID_P\#key$. Indeed, P can simply generate a new DID document with the new key (but the same identifier) and include that document (and its proof) in the `did:self` header. As a result, P can rotate its signing keys without involving $Owner_{root}$.

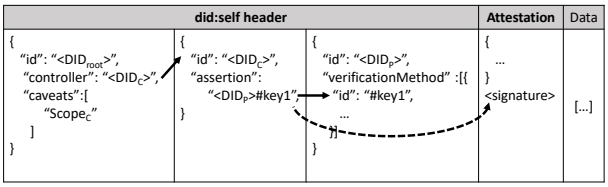

Fig. 6. The `did:self` header of a self-verifiable content item, generated by a producer, authorized by a controller. The owner's DID document indicates the controller, while the controller's DID document points at an assertion key defined in the producer's DID document.

3) Sub-namespace management delegation: Finally, consider an entity C that owns a `did:self` identifier DID_C, for example, the manager of smart-building1 in Figure 1. $Owner_{root}$ can allow C to become a *controller* of a sub-space $DID_{root}/Scope_C/$, where $Scope_C$ can be any valid prefix. This is achieved by including the `controller` property in the DID document of $Owner_{root}$ and setting its value equal to DID_C. A controller can create self-verifiable content items under $DID_{root}/Scope_C/$, similar to an authorized producer. More importantly, though, C can also authorize *other* producers to create such items under this sub-space or a subset of it.

Controller C can then generate valid DID documents for its delegated sub-space under DID_{root} and include them in the `did:self` header of the metadata. Figure 6 provides an example of a `did:self` header created by a producer P authorized by a controller C. Controller C has defined in its DID document an `assertion` property that refers to a public key defined in the DID document of producer P.

4) Content verification: Upon receiving a self-verifiable content item identified by $DID_{root}/suffix$, a consumer can assess its validity by verifying the attestation included in the item's metadata. This is achieved by executing the following steps:

1) Verify that the `name` and `sha-256` fields of the attestation include the correct values.
2) Validate the DID document of DID_{root} using the procedure of Section III-C.
3) If the DID document of DID_{root} includes the `caveats` property, verify that $suffix$ is covered by it.
4) If the DID document of DID_{root} includes the `controller` property, locate the controller's DID document in the `did:self` header and validate it.
5) Extract the `assertion` property included in the DID document of DID_{root} (or in the DID document of the controller).
6) If the public key referenced by the `assertion` property is included in another DID document, locate that document in the `did:self` header and validate it.
7) Verify the signature of the attestation using the public key referenced by the `assertion` property.

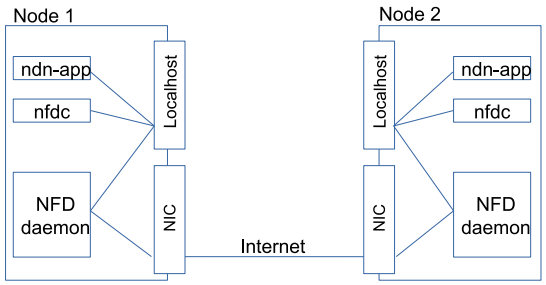

Fig. 7. The NFD daemon and its links with other processes (for a Linux-based node).

V. Implementation

We have implemented a proof of concept of our solution using our Python3-based implementation of `did:self`.[2] As a verification method we are using Ed22519 cryptographic keys. Attestations are signed are serialized using the JWS format by using the JWCrypto library,[3] while the SHA-256 hashes included in the various proofs use Python's hashlib library. We also created a Python3 application that receives as input a content item, the appropriate DID documents and proofs, and the corresponding secret keys, producing as output a "self-verifiable content item." The same application can also verify such items.[4] We next explain how this implementation can be integrated into NDN.

A. Integration with the NDN prototype

The *Named Data Networking Forwarding Daemon* (NFD) is a prototype implementation of the NDN protocol suite, which incorporates the complete content-based functionality of a router [35]. The core element of NFD is the `nfd-daemon`, a process running in the background that listens to predefined sockets for packets originating from local or remote applications. The `nfd-daemon` is controlled by a command line configuration tool, named `nfdc`. The NDN applications generate data plane packets, while `nfdc` sends control plane packets to configure the router's functionality, as shown in Figure 7.

In general, NDN data plane packet handling is defined through *forwarding strategies*; these are sets of rules that govern the selection of the faces (network interfaces or local applications), to push a received packet to. Strategies allow the implementation of diverse content delivery patterns, such as multipath (by forwarding the packet over multiple faces), shortest path routing (by forwarding the packet over the face that reported the minimum latency) and more.

In the context of this work, we focus on the handling of NDN Data packets that include self-verifiable content items. When packets are sent by applications, they also carry some metadata, as an authentication block. By having the first on-path router audit the metadata, we can enforce security early on. The metadata audit mechanism can be implemented through a new forwarding strategy for NDN Data packets, defined as follows:

1) Check if the current node is the first on-path router; if not, then the node forwards the Data packet according to the default forwarding policy.
2) Parse the metadata from the header and check if it is valid for the content name of the Data packet.
   a) If it is not valid, then discard the packet and send a *Negative Acknowledgement* (NACK) packet to the consumer. The NACK notifies the consumer that the Data packet was not valid, and consumes the PIT state on the preceding on-path routers.
   b) If it is valid, then sign the packet with the router's unique certificate to mark the packet as successfully audited, and forward it according to the default forwarding policy.

NACKs can open the door to simple *Denial of Service* (DoS) attacks, if malicious users are allowed to inject NACKs for Interests that traverse their node. This is a trivial way to censor content considered subversive or simply undesirable. Therefore, we assume that NACKs are secured according to the design proposed in [36], where NACKs are signed by the producers or, in our case, the in-network routers.

Although this is a straightforward implementation, it conflicts with two design tenets of NFD. First, NFD defines forwarding strategies at the content name level and introduces an 1-to-1 mapping between the forwarding strategy and the content name at each node; different on-path routers can use different forwarding policies for the same content name. If the audit mechanism is implemented as a forwarding strategy, then the auditing nodes will no longer be able to select alternative forwarding strategies. Second, in NFD, a forwarding strategy can postpone the delivery of a Data packet (by delaying its forwarding), but is not meant to permanently drop a Data packet, since this operation breaks the 1-to-1 Interest-to-Data packet matching. NDN does not endorse dropping Data packets; it suggests that Interest packets should be discarded instead.

To support the utilization of different forwarding strategies in parallel with our solution, we propose implementing it as a module of the core forwarding functionality, thus making it independent from the forwarding strategy. In this approach, the audit algorithm is engaged regardless of the strategy mapped to the name, which only deals with the selection of the outgoing face for the Data packet or the NACK. There are two open issues that remain to be resolved: the definition of the name space that the metadata auditing will be enabled for and the implementation of the metadata check module.

Regarding the application scope of our mechanism, we restrict it to a designated name prefix, namely, *DIDCHECK*. This is critical for the transparent operation of our module in the core of NFD, since the control plane of NFD also follows the Interest-Data communication pattern; without this, Data packets that do not carry a DID, such as control plane packets, would be erroneously rejected. Therefore, each time a Data packet is received, its name is checked; if the designated prefix matches, only then the metadata are audited.

[2]https://github.com/mmlab-aueb/did-self-py
[3]https://jwcrypto.readthedocs.io/en/latest/
[4]https://github.com/mmlab-aueb/did-self-svci.

TABLE I
CRYPTOGRAPHIC OPERATIONS REQUIRED BY OUR SOLUTION AND THEIR PROCESSING OVERHEAD.

Operation	Time (ms)
Generation of a key pair	46
Generation of a DID document and its proof	2.7
Calculation and serialization of a content attestation	0.7 + 7*(size of item in MB)
Verification of a DID document	1.5
Verification of a content attestation	0.2

TABLE II
TIME REQUIRED TO VERIFY A CONTENT ITEM.

Scenario	Time (ms)
Scenario A	1.7 + 7*(size of item in MB)
Scenario A	3.2 + 7*(size of item in MB)
Scenario C	4.7 +7*(size of item in MB)

TABLE III
SIZE (IN BYTES) OF THE DID:SELF HEADER.

Scenario	Size (bytes)
Scenario A	713
Scenario A	1209
Scenario C	1701

Regarding the implementation of the metadata audit module, we consider it too generic to be included in NFD, hence we exploit the Python3 implementation discussed above. The communication of the C++ implementation of NFD and the Python process is based on the *popen* (process open) command[5] offered by the Linux kernel, which allows a C++ process to invoke a Python process and read the output of the latter through a pipe stream. This approach allows a seamless integration with the NFD code, in that only minor modifications are required, and offers sufficient performance, at least in the context of our design.

VI. EVALUATION

We evaluate the performance of our solution using the implementation presented in the previous section and a machine running Ubuntu 18.04 with a 3.1GHz Intel i5 CPU and 2GB of RAM. For our performance evaluation we are considering three scenarios:

- **Scenario A**: The content owner generates the metadata.
- **Scenario B**: A producer authorized by the content owner generates the metadata.
- **Scenario C**: A producer authorized by a controller generates the metadata.

A. Computational overhead

All scenarios involve the creation of (at least) a DID, the corresponding DID document and its proof. In our implementation, DID creation requires the generation of an Ed22519 key pair, while proof generation requires an EdDSA signature. All scenarios also involve the generation of an attestation for the content. Finally, all scenarios require the verification of (at least) a DID document and an attestation. Table I shows the time required (in ms) to perform the operations of our system. We see that most operations are executed in less than 3 ms; the exception is key pair generation which is only rarely needed. The only variable performance metric is the time required to create an attestation, since this involves the calculation of the SHA-256 hash of the file. In our evaluation machine, using the `sha256sum` command, this requires $\approx$ 7 ms per MB.

When it comes to the verification of a content item, in Scenario A, a consumer has to verify one DID document and one attestation, in Scenario B, two DID documents and one attestation, and in Scenario C, three DID documents and one attestation. Table II shows the time required (in ms) for a consumer to verify a content item. In terms of verification overhead, the existing, trust schemata-based approach of NDN [31], has the same overhead as Scenario A, since it requires the verification of a digital certificate, the calculation of the hash of the content item, and the verification of a digital signature.

A content producer has to perform the following tasks: generate an attestation signing key and the corresponding DID document *once*, regardless of the number (and size) of content items, and then, for each content item calculate its SHA-256 hash and sign its attestation. Additionally, every time the generated DID document expires, the producer has to generate a new DID document and generate a fresh proof. The producer does not have to change the attestation signing key whenever a new DID document is created. When the attestation signing key needs to be updated, all signed attestations must be updated as well, without re-calculating the SHA-256 hashes of the content items. In terms of content production overhead, the existing, trust schemata-based approach of NDN [31] has the same overhead as our solution, since it requires the generation of a single digital certificate, the calculation of the SHA-256 hash of each item and the creation of a digital signature per item.

B. Communication overhead

We now calculate the communication overhead introduced by our solution. We only consider the `did:self` header, since NDN already includes a data structure similar to the attestation. For our calculations, we assume that the DID document of the content owner includes the `caveats` property and its value is a single 10 byte string. Table III shows the size of the `did:self` header for each scenario. We see that the maximum size of the `did:self` header is 1701 bytes.

The existing, trust schemata-based approach of NDN, uses smaller packets than our solution, but it requires more messages to verify a content item. Rather than embedding the verification metadata in the content, as in our solution, NDN includes a *name* in the content that can be used to retrieve a certificate from the network, relying on caching to decrease the communication overhead. In order to better understand the trade-offs, we consider an alternative to our solution, which is closer to the solution currently used by NDN. In this alternative, rather than including DID documents in a content item's metadata, we treat the `did:self` header itself as a content item, that can be fetched from the corresponding

[5]https://linux.die.net/man/3/popen

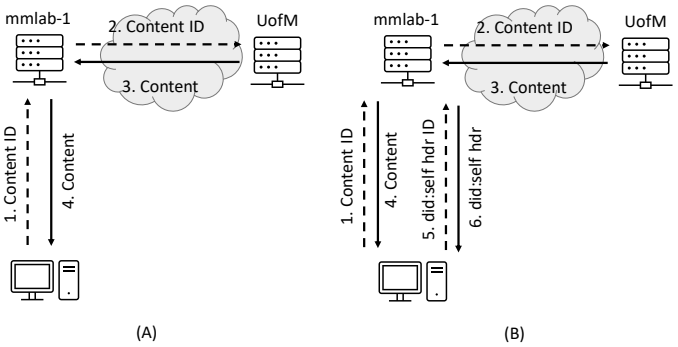

Fig. 8. Content retrieval when the did:self header is embedded in the content item (A), or provided as a standalone item (B). Dotted lines indicate Interest packets and solid lines indicate Data packets.

Producer. Moreover, we assume that the `did:self` header required for validating an item is retrieved using a well-known content identifier of the form <content name>/did-document.

We compare these two options over the NDN testbed,[6] a global shared resource created for research purposes that relies on software routers at several participating institutions, application host nodes, and other devices. We allocated two testbed nodes, one provided by the Athens University of Economics and Business (`mmlab-1`), and another provided by the University of Memphis (`UoFM`). Node `UoFM` acts as an authorized content producer. We assume that a consumer application connects to `mmlab-1` and requests a content item. Moreover, for the second option, we assume that the content item that corresponds to the `did:self` header has already been cached by `mmlab-1`.

Content retrieval using our original solution is illustrated in Figure 8.A. The Interest for the content item (dotted line) arrives to `UoFM`; the latter node responds with the content item itself (solid line), which can be immediately verified. Content retrieval using the alternative approach is illustrated in Figure 8.B. In this case, the received content item does not include the `did:self` header; the consumer application sends a second Interest message and receives a cached version from `mmlab-1`. With the alternative approach we can save 713–1701 bytes (see Table III) per content item in the link between `mmlab-1` and `UoFM`, at the cost of an additional roundtrip between the consumer and `mmlab-1`.

C. Security evaluation

Our solution considers the threat model of [31]. In particular, we consider that threats to data authenticity verification include failed authentication of a legitimate signing key, attackers trying to impersonate content owners or authorized producers, as well as malicious keys. Additionally. we consider adversaries based on the Dolev-Yao model [37], i.e., adversaries that tamper with network packets but cannot break the security of the used cryptographic primitives. Assuming that a consumer knows the DID used as the prefix of a content item's name, our solution achieves the following security goals in this threat model.

[6]https://named-data.net/ndn-testbed/

Content producers are always properly authenticated. Content owners in our solution are authenticated using a public key which is part of the content name. Our solution, therefore, does not depend on any trusted entity to verify the owner's identity. Similarly, authorized producers are authenticated using information included in the content item's metadata. This limits the chance of implementation errors that would result in failing to authenticate an authorized producer.

Attackers cannot impersonate content owners or authorized producers. Assuming that the private key that corresponds to the DID of a content owner or authorized producer is properly secured, then an attacker cannot impersonate them. If an attacker gains access to the private key of a content owner, then the corresponding DID must be changed, which means that content consumers must learn a new content name prefix. On the other hand, updating the DID of an authorized producer (e.g., due to a private key breach) requires only the issuance of new DID documents. Furthermore, old DID documents (e.g., included in cached items) can be invalidated using DID document revocation. We have adapted a revocation scheme designed for *Verifiable Credentials* (VCs), presented in [38]; a similar approach has been proven to be scalable and privacy preserving at an Internet-wide scale [39]. In our scheme, each DID owner maintains a revocation list for all DID documents that it has issued, until they expire, in the form of a bit string. Each DID document issued by the owner corresponds to a position in this list. Revoking a DID document requires setting the corresponding bit in the revocation list to 1. Each DID document includes in its document proof the property "revocationListIndex" which specifies its bit position in the revocation list, and the property "revocationListCredential" that includes a `did:self` DID used as a prefix of the revocation list name. The latest version of a revocation list can be retrieved using a synchronization protocol, such as ChronoSync [20].

Our solution is resilient to malicious keys. A DID document included in the metadata of a content item enforces a mapping between signing keys and content name prefixes (akin to the mapping implemented by trust schemata [31]). Therefore, attackers can provide valid attestation signatures only if they have access to a valid signing key. Our solution protects signing keys by allowing content owners and authorized producers to easily rotate them. In particular, when it comes to authorized producers, by using an assertion key included in a producer's DID document, the owner allows the producer to freely replace that key with a new one without the owner issuing a new DID document. This is in stark contrast to a certificate-based solution, where a new key certificate must be issued whenever a key is replaced. This is particularly useful in use cases such as the smart city scenario presented in Section III-B, where "rotating a key" may be translated in the physical world to "rotating a device." For example, due to the limited lifetime of the battery of a drone, multiple drones can be used one after another to provide the same type of information (e.g., information prefixed with "<smart city DID >/roadB23/traffic"); these drones do not have to share the same assertion key, they only have to be configured with the appropriate DID document. Additionally, using the same key

identifier for the newly generated key allows for the revocation of old ones included, e.g., in cached items; this is achieved by following a simple rule: if two verification methods, belonging to the same DID, have the same identifier, then the verification method defined in the *newest* DID document replaces the other. Finally, by using *caveats* our solution restricts signing keys to specific portions of the namespace, thus achieving the "least privilege" principle.

Our solution is resilient to active attackers. A digest of the content item's data is recorded in the *attestation* part of the metadata, therefore an attacker cannot replace or modify the transmitted item without invalidating its attestation. The inclusion of the content name in an attestation prevents an attacker from sending an Interest for a different item under the same prefix, and then using this (valid) response to replace an item requested by another consumer. Furthermore, a malicious but authorized producer cannot generate responses for content names other than those for which it has been authorized.

D. Discussion

Comparison to TTP and trust schemata approaches. In our solution content name prefixes are used as *trust anchors*: there is an explicit binding between a content item (via its name) and its trust anchor. Alternative solutions rely on *Trusted Third Parties* (TTPs) or on endpoint configuration using *trust schemata* [31] to provide this binding.

A TTP-based solution requires all participating entities to agree on a (set of) TTPs, which in many cases may be cumbersome, or even involve non-negligible monetary cost. This is a problem when security verifications must be performed by in-network nodes, such as routers, as they are harder to configure and update than applications. TTPs are also a significant security threat since they can generate certificates for *any* prefix; a compromised TTP can generate an unlimited number of valid certificates. Finally, in TTP-based solutions it is hard to manage cases where a TTP must rotate its keys. If rotation means that old keys are revoked, then all issued certificates must be re-issued. Additionally, all verification points must be configured with the new keys of the TTP, which may require significant effort, especially if in-network devices must be reconfigured.

Using trust schemata and self-generated certificates, security properties similar to our solution can be achieved. However, a schemata-based solution requires that either all content name prefixes are known, in order for the appropriate rules to be configured, or that a TTP can be used as a trust anchor for prefixes not covered by a rule. Therefore, in scenarios where content names are "discovered" (e.g., in Web-like, or search applications) a schemata-based solution behaves similarly to a TTP-based solution.

The impact of content owner DID loss. In our system, if the private key which is associated with a prefix is breached or lost, then this prefix must change. We recognize that this creates a security concern, but it also has an interesting side-effect: it allows the creation of long-lasting, immutable, archival item collections. In particular, an entity can generate a DID-based prefix, use the private key corresponding to the DID to sign attestations, and then permanently delete it: as long as the signing algorithm remains secure, it will be impossible to modify the generated self-verifiable content items.

Non human-memorable names. The DIDs used in our scheme are neither human-readable, nor human-memorable. Although this limits the usability of our system, we postulate that it improves its security. Firstly, security solutions that rely on memorable content names are susceptible to phishing attacks: there is considerable empirical evidence that users can easily confuse carefully crafted fake URLs with real ones. Secondly, since prefixes are cryptographic public keys randomly generated by users, our solution achieves uniqueness of content names with very high probability, without requiring a content name management system, which would add both overhead and security risks.

VII. Conclusions and future work

We presented a self-certifying naming scheme based on the paradigm of *Decentralized Identifiers* (DIDs). Our scheme supports fast integrity and authenticity verification of content items that use DID-based content name prefixes. The verification processes of our system do not require interaction with third parties; instead all necessary input is prepended in the content items' data in a secure and verifiable way. This approach not only makes our approach fast and lightweight, it also facilitates its deployment.

The DID method leveraged by our system removes the need for a DID document "registry," remaining at the same time compatible with the evolving DID specification. Our solution is also interoperable with other DID systems, including registry-based ones. DIDs are considered in many applications, ranging from IoT deployments to integration with real world identities and credentials. Furthermore, ongoing standardization efforts aim at adding support for DID-based procedures in web browsers, mobile devices, hardware tokens, and other popular systems. Therefore, in an environment where DIDs are widely used, we anticipate that our approach can fuel novel and exciting applications.

Our solution does not limit the type of cryptographic keys used for protecting content authenticity. Similarly, DIDs specifications allow alternative "verification methods" and support many types of proofs. This creates opportunities for combining our solution with contemporary cryptographic systems, including Attribute/Identity-Based encryption, for using other forms of verification methods, which can be linked to the physical world, e.g., biometrics, and for applying zero-knowledge proofs, e.g., for hiding parts of a content item from unauthorized users.

Our solution uses DIDs to protect transmitted data. Nevertheless, a similar approach can be used to filter "advertisements" of content name prefixes. On-going work in this area involves the application of our scheme to edge routers, allowing them to only forward towards the core those routing advertisements originating from authorized producers [40].

Another concept closely related to DIDs is that of *Verifiable Credentials* [41] (VCs). VCs allow an *issuer* to assert one or

more *claims* about a *subject*. A VC includes a set of claims that can be securely verified by a *verifier*. In particular, a verifier can request from the VC holder to create a *verifiable presentation* of its credentials that prove certain claims. We envision that such *proof requests* can be embedded into a DID document, allowing content owners to "express" access control policies that should control the dissemination of their content. These policies can then be evaluated and enforced by third party storage nodes. Additionally, by including VCs in Interest messages, we can limit the impact of "interest flooding attacks," as each router will be able to verify if a consumer is authorized to express interest for a specific content item.

ACKNOWLEDGMENT

The work reported in this paper has been funded at AUEB in part by European Unions Horizon 2020 Research and Innovation Programme through the subgrants "Self-Certifying Names for Named Data Networking" (SCN4NDN) and "Securing Content Delivery and Provenance" (SECOND) of the project NGIatlantic.eu, under grant agreement No 871582, where we collaborated with Prof. Christos Papadopoulos' team at the University of Memphis.

REFERENCES

[1] G. Xylomenos, C. N. Ververidis, V. A. Siris, N. Fotiou, C. Tsilopoulos, X. Vasilakos, K. V. Katsaros, and G. C. Polyzos, "A survey of Information-Centric Networking research," *IEEE Communications Surveys and Tutorials*, vol. 16, no. 2, pp. 1024–1049, 2014.

[2] C. Ghali, G. Tsudik, and E. Uzun, "Needle in a haystack: Mitigating content poisoning in Named-Data Networking," in *NDSS Workshop on Security of Emerging Networking Technologies (SENT)*. Internet Society, 2014, pp. 1–10.

[3] J. Wang, X. Wei, J. Fan, Q. Duan, J. Liu, and Y. Wang, "Request pattern change-based cache pollution attack detection and defense in edge computing," *Digital Communications and Networks*, to appear.

[4] Z. Zhang, Y. Yu, H. Zhang, E. Newberry, S. Mastorakis, Y. Li, A. Afanasyev, and L. Zhang, "An overview of security support in Named Data Networking," *IEEE Communications Magazine*, vol. 56, no. 11, pp. 62–68, 2018.

[5] L. Zhang, A. Afanasyev, J. Burke, V. Jacobson, k. claffy, P. Crowley, C. Papadopoulos, L. Wang, and B. Zhang, "Named Data Networking," *SIGCOMM Computer Communications Review*, vol. 44, no. 3, p. 6673, jul 2014.

[6] P. F. Tehrani, L. Keidel, E. Osterweil, J. H. Schiller, T. C. Schmidt, and M. Wählisch, "NDNSSEC: Namespace management in NDN with DNSSEC," in *ACM Conference on Information-Centric Networking*. New York, NY, USA: ACM, 2019, pp. 171–172.

[7] Y. Yu, "Public key management in Named Data Networking," NDN Consortium, Tech. Rep. NDN-0029, 2015.

[8] A. Hughes, M. Sporny, and D. Reed, Eds., *A Primer for Decentralized Identifiers*. W3C Credentials Community Group, 2019. [Online]. Available: https://w3c-ccg.github.io/did-primer/

[9] O. Steele and M. Sporny, *DID Specification Registries*. W3C Decentralized Identifier Working Group, August 2022. [Online]. Available: https://www.w3.org/TR/did-spec-registries/

[10] N. Fotiou, "did:self method specification," Mobile Multimedia Laboratory, 2021. [Online]. Available: https://github.com/excid-io/did-self

[11] P. F. Tehrani, E. Osterweil, T. C. Schmidt, and M. Wählisch, "SoK: Public key and namespace management in NDN," in *ACM Conference on Information-Centric Networking*. New York, NY, USA: ACM, 2022, pp. 67–79.

[12] Y. Psaras and D. Dias, "The InterPlanetary File System and the Filecoin network," in *International Conference on Dependable Systems and Networks-Supplemental Volume (DSN-S)*. New York, NY, USA: IEEE, 2020, pp. 80–80.

[13] K. Fu, M. F. Kaashoek, and D. Mazieres, "Fast and secure distributed read-only file system," *ACM Transactions on Computer Systems (TOCS)*, vol. 20, no. 1, pp. 1–24, 2002.

[14] S. Farrell, D. Kutscher, C. Dannewitz, B. Ohlman, A. Keranen, and P. Hallam-Baker, "Naming Things with Hashes," Internet Requests for Comments, IETF, RFC 6920, April 2013. [Online]. Available: https://www.rfc-editor.org/rfc/rfc6920.txt

[15] T. Kuhn and M. Dumontier, "Making digital artifacts on the web verifiable and reliable," *IEEE Transactions on Knowledge and Data Engineering*, vol. 27, no. 9, pp. 2390–2400, 2015.

[16] M. Baugher, B. Davie, A. Narayanan, and D. Oran, "Self-verifying names for read-only named data," in *IEEE INFOCOM Workshops*. New York, NY, USA: IEEE, 2012, pp. 274–279.

[17] W. Wong and P. Nikander, "Secure naming in information-centric networks," in *ACM Re-Architecting the Internet Workshop (RE-ARCH)*. New York, NY, USA: ACM, 2010, pp. 1–6.

[18] S. H. Bouk, S. H. Ahmed, and D. Kim, "Hierarchical and hash based naming with compact trie name management scheme for vehicular content centric networks," *Computer Communications*, vol. 71, pp. 73–83, 2015.

[19] S. Arshad, B. Shahzaad, M. A. Azam, J. Loo, S. H. Ahmed, and S. Aslam, "Hierarchical and flat-based hybrid naming scheme in content-centric networks of things," *IEEE Internet of Things Journal*, vol. 5, no. 2, pp. 1070–1080, 2018.

[20] Z. Zhu and A. Afanasyev, "Let's ChronoSync: Decentralized dataset state synchronization in Named Data Networking," in *International Conference on Network Protocols (ICNP)*. New York, NY, USA: IEEE, 2013, pp. 1–10.

[21] D. Mazieres and M. F. Kaashoek, "Escaping the evils of centralized control with self-certifying pathnames," in *ACM SIGOPS European workshop on Support for composing distributed applications*. New York, NY, USA: ACM, 1998, pp. 118–125.

[22] T. Koponen, M. Chawla, B.-G. Chun, A. Ermolinskiy, K. H. Kim, S. Shenker, and I. Stoica, "A data-oriented (and beyond) network architecture," *SIGCOMM Computer Communications Review*, vol. 37, no. 4, pp. 181–192, Aug. 2007.

[23] A. Ghodsi, T. Koponen, J. Rajahalme, P. Sarolahti, and S. Shenker, "Naming in content-oriented architectures," in *ACM Workshop on Information-Centric Networking*. New York, NY, USA: ACM, 2011, pp. 1–6.

[24] C. Dannewitz, J. Golic, B. Ohlman, and B. Ahlgren, "Secure naming for a network of information," in *IEEE INFOCOM*. New York, NY, USA: IEEE, March 2010, pp. 1–6.

[25] X. Zhang, K. Chang, H. Xiong, Y. Wen, G. Shi, and G. Wang, "Towards name-based trust and security for content-centric network," in *IEEE International Conference on Network Protocols (ICNP)*. New York, NY, USA: IEEE, 2011, pp. 1–6.

[26] B. Hamdane, S. G. El Fatmi, and A. Serhrouchni, "A novel name-based security mechanism for information-centric networking," in *IEEE Wireless Communications and Networking Conference (WCNC)*. New York, NY, USA: IEEE, 2014, pp. 2928–2933.

[27] S. K. Ramani, R. Tourani, G. Torres, S. Misra, and A. Afanasyev, "NDN=ABS: Attribute-based signature scheme for Named Data Networking," in *ACM Conference on Information-Centric Networking*. New York, NY, USA: ACM, 2019, pp. 123–133.

[28] R. Li, H. Asaeda, and J. Wu, "DCAuth: Data-centric authentication for secure in-network big-data retrieval," *IEEE Transactions on Network Science and Engineering*, vol. 7, no. 1, pp. 15–27, 2018.

[29] J. Shi, X. Zeng, and R. Han, "A blockchain-based decentralized public key infrastructure for information-centric networks," *Information*, vol. 13, no. 5, p. 264, 2022.

[30] M. Blaze, J. Feigenbaum, and J. Lacy, "Decentralized trust management," in *IEEE Symposium on Security and Privacy*. New York, NY, USA: IEEE, 1996, pp. 164–173.

[31] Y. Yu, A. Afanasyev, D. Clark, k. claffy, V. Jacobson, and L. Zhang, "Schematizing trust in Named Data Networking," in *ACM Conference on Information-Centric Networking*. New York, NY, USA: ACM, 2015, pp. 177–86.

[32] M. Jones and N. Sakimura, "JSON Web Key (JWK) Thumbprint," Internet Requests for Comments, IETF, RFC 7638, September 2015. [Online]. Available: https://tools.ietf.org/html/rfc7638

[33] M. Jones, J. Bradley, and N. Sakimura, "JSON Web Signature (JWS)," Internet Requests for Comments, IETF, RFC 7515, May 2015. [Online]. Available: https://tools.ietf.org/html/rfc7515

[34] M. Jones, "JSON Web Key (JWK)," Internet Requests for Comments, IETF, RFC 7517, May 2015. [Online]. Available: https://tools.ietf.org/html/rfc7517

[35] A. Afanasyev et al., "NFD developer's guide," NDN Consortium, Tech. Rep. NDN-0021, 2021. [Online]. Available: https://named-data.net/publications/techreports/nfd-developer-guide/

[36] A. Compagno, M. Conti, C. Ghali, and G. Tsudik, "To NACK or not to NACK? negative acknowledgments in Information-Centric Networking," in *International Conference on Computer Communication and Networks (ICCCN)*. New York, NY, USA: IEEE, 2015, pp. 1–10.
[37] D. Dolev and A. Yao, "On the security of public key protocols," *IEEE Transactions on Information Theory*, vol. 29, no. 2, pp. 198–208, 1983.
[38] M. Sporny, D. Longley, *Revocation List 2020*. W3C Credentials Community Group, 2020. [Online]. Available: https://w3c-ccg.github.io/vc-status-rl-2020/
[39] T. Smith, L. Dickinson, and K. Seamons, "Let's revoke: Scalable global certificate revocation," in *Network and Distributed System Security Symposium (NDSS)*. New York, NY, USA: Internet Society, 2020, pp. 1–14.
[40] N. Fotiou, Y. Thomas, V. A. Siris, G. Xylomenos, and G. C. Polyzos, "Securing Named Data Networking routing using decentralized identifiers," in *Semantic Addressing and Routing for Future Networks Workshop (SARNET)*. New York, NY, USA: IEEE, 2021, pp. 1–6.
[41] M. Sporny and D. Longley and D. Chadwick, *Verifiable Credentials Data Model*. W3C Verifiable Credentials Working Group, 2019. [Online]. Available: https://www.w3.org/TR/verifiable-claims-data-model/