Authentication and Authorization for Content-Centric Routing using W3C DIDs and VCs

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

***Abstract*—Content-Centric Routing is the cornerstone of the standardization efforts related to Named Data Networking (NDN). It enables advanced communication paradigms and facilitates content replication and caching, by allowing routers to exchange information and perform forwarding based on content item identifiers rather than location identifiers (e.g., network addresses). However, since content may originate from multiple locations, malicious nodes can advertise content that they do not host, effectively hijacking the corresponding content name prefixes. In this paper, we leverage two recent W3C recommendations, Decentralized Identifiers (DIDs) and Verifiable Credentials (VCs), to enable routing nodes to independently check whether a content advertisement was generated by an authorized node, in a fully decentralized manner that does not rely on trusted third parties. We implement and evaluate our solution in a system where advertisements are verified by the content routers at the edge of the network, showing that it prevents fake routing advertisements with minimal overhead.**

***Index Terms*—Self-sovereign, edge networks, DID, VC, NDN.**

I. Introduction

Content-Centric Routing is the main building block of many Future Internet standardization efforts that depart from traditional internetworking. Unlike the existing Internet which is based on opaque, location-dependent, endpoint addresses, with Content-Centric Routing the network operates on semantically-rich, location independent, *content* names, giving rise to the *Information-Centric Networking* (ICN) [1] paradigm. The most popular ICN realization is *Named Data Networking* (NDN) [2], which is under standardization in the ICN Research Group (icnrg) of the IRTF.[1] In NDN each piece of content has its own, unique and location-independent, name. This unties content items from a single origin server, thus allowing content to be hosted at multiple servers or caches.

In this paper, we consider a *limited domain* NDN deployment (as defined in RFC 8799 [3]), where a content *owner* wishes to make some content available via an NDN node, referred to as the *publisher*; all of the owner's content items share the same name prefix, which is unique to that owner. The publisher may have a number of servers hosting content, similar to a *Content Distribution Network* (CDN). Publishers in NDN must *announce* to the NDN network the prefixes of the content items that they *host*, thus allowing content *consumers* to become aware of and retrieve the corresponding content. These advertisements first reach an *edge router*, which subsequently disseminates them to the rest of the network.

Since content in NDN can be served by multiple nodes, a malicious publisher can inject content advertisements into the network without previous authorization by the content owner; effectively, this is a prefix hijacking attack, where the malicious publisher pretends to be an authorized source. The problem is how to distinguish between authorized and unauthorized content advertisements. In the following subsections we first explain how NDN solves this problem by relying on trusted third parties, and then present our proposed solution, which is fully decentralized and self-sovereign, i.e., content owners have full control of how their content is advertised.

A. Content-Centric Routing in NDN

Content items in NDN are identified by unique hierarchical *names*, for example `/aueb/mmlab/project`. A *consumer* can retrieve a content item by transmiting an *Interest* packet containing the content's name, which is forwarded by *content routers* (CRs) to the appropriate content *publishers*. The CR uses a lookup table mapping content names to output interface(s), called the *Forwarding Information Base* (FIB), to forward these Interests. FIBs are generated by the *Named-data Link State Routing* (NLSR) protocol [4], in which content hosting nodes advertise *prefixes* of content names; such an advertisement indicates that they can provide any content items under that prefix. The edge CRs propagate these advertisements using NLSR, so as to populate the network's FIBs.

In NDN content items are not tied to a location, hence a CR may receive multiple *valid* prefix advertisements from different network directions. This, however, opens the door to fake advertisements by malicious nodes trying to hijack a prefix. NDN addresses this problem by (optionally) signing prefix advertisements. Edge CRs can verify such signatures by following a chain rooted in a *trust anchor*: a *Trusted Third Party* (TTP) whose certificate is known to all edge CRs [5]. The TTP must issue a certificate for the content owner, who must then issue a certificate for the publisher. The signed content advertisements include "pointers" to these certificates. As a result, an edge CR can verify signed prefix advertisements as follows: it first retrieves these certificates and then it checks whether (a) the owner's certificate was issued by the TTP, (b) the publisher's certificate was issued by the owner and (c) the

[1] https://datatracker.ietf.org/rg/icnrg/about/

advertisement was signed by the publisher. This is similar to how HTTPS works, but instead of certifying the endpoint from where content is downloaded, we certify the content itself; in both cases, the TTP must be universally trusted.

NDN's approach has some limitations that our solution aims to overcome. First, having a universally trusted TTP introduces security risks, especially a TTP that decides whether an entity is entitled to use a content name prefix. Second, the TTP can be a bottleneck, as usually the owner verification process takes some time, preventing real-time certificate issuance. Third, in order for a publisher to rotate its signing key, a new certificate must be issued. Fourth, it is not straightforward to delegate part of the name space, e.g., allow the owner of "/aueb/mmlab" to delegate to another entity the prefix "/aueb/mmlab/project". Finally, certificate revocation is an open issue.

B. Solution overview and contributions

Our solution uses *Decentralized Identifiers* (DIDs) as content name prefixes and publisher identifiers, and *Verifiable Credentials* (VCs) to authorize publishers to advertise content name prefixes. DIDs and VCs are recent W3C recommendations that have attracted the attention of both industry and academia due to their intriguing security and privacy properties. In short, our solution operates as follows. Content owners generate DIDs that are used as *prefixes* of their content names and are associated with a public key. Then, they *issue* a VC to each publisher authorized to "host" their content items; these VCs are signed using the key associated with the corresponding DID. Publishers then sign their routing advertisements using keys included in *their own* DID; these signatures can be verified based on the information included in the VC. Our solution makes the following contributions:

- We remove the need for a TTP, at the cost of non human-readable content name prefixes.
- We allow content owners to generate by themselves content name prefixes that are globally unique and for which they can prove ownership.
- We allow content owners to authorize publishers to advertise (part of) their content name prefixes.
- We allow publishers to rotate their advertisement signing keys without requiring a new VC.

In the remainder of this paper, we first introduce DIDs and VCs and present our overall design in Section II. In Section III we present the implementation and evaluate the proposed system. We discuss related work in Section IV. Finally, we conclude and discuss future work in Section V.

II. System Design

A. Decentralized Identifiers

Decentralized Identifiers (DIDs), standardized by the W3C, are globally unique identifiers that can be resolved with a high availability and be verified with cryptographic means [6]. A DID has the form of a URI and is accompanied by a *DID document*; the DID document can include public keys, authentication protocols, and any other information needed to perform cryptographically-verifiable transactions with the owner of the DID [6]. A DID document can be stored in a *DID registry*, which can implement appropriate security and access control schemes. A registry allows a third party to lookup a DID document based on the DID, providing a *proof* of correctness for the document, such as a digital signature.

The actual contents of DID documents and the way a registry operates are not defined by the standards; instead, these details are left to instantiations of the standard called *DID methods*. Our solution is based on a DID method that we have developed, called *did:self* [7], which does *not* require a registry. Rather, *did:self* based DID documents can be retrieved from a publicly accessible server or directly transmitted to a recipient. Instead of relying on a trusted registry, *did:self* ensures that a DID document can be cryptographically matched to a DID, regardless of how it was retrieved.

A DID conforming to *did:self* is a thumbprint of a JSON Web Key (JWK) [8] prefixed with 'did:self:'. The matching *DID document* is encoded using JSON and may include any of the DID "properties" included in the DID specifications. Our scheme uses the following properties (see also Listing 1):

- `id`: The DID corresponding to the document (line 1).
- `verificationMethod`: one or more public keys expressed in "JsonWebKey2020" notation [9]. Each such key is identified by an *id* (in Listing 1 two such keys are defined in lines 3-15).
- `authentication`: One or more public keys (or key identifiers) used to authenticate prefix advertisements (line 16 states that "#key1" can be used as an authentication key).
- `assertion`: One or more public keys (or key identifiers) used to verify the digital signatures of VCs (line 17 states that "#key2" can be used as an assertion key).

```
1   {
2     "id": "did:self:6varD0Rj...",
3     "verificationMethod'':[{
4           "id": "#key1",
5           "type": "JsonWebKey2020",
6           "publicKeyJwk": {
7             "crv": "Ed25519",
8             "x": "8aJkufAc...",
9             "kty": "OKP"
10          }
11      },{
12          "id": "#key2",
13          "type": "JsonWebKey2020",
14          "publicKeyJwk": { ... }
15      }],
16    "authentication": ["#key1''],
17    "assertion": ["#key2'']
18  }
```

Listing 1. A sample DID document.

To verify the binding between a *did:self* DID and its corresponding DID document, each DID document is associated with a *proof*; this proof is a "compact serialization" of a JSON Web Signature (JWS). The proof's *header* includes the following *claims*:

- `alg`: The algorithm used to generate the proof.
- `jwk`: The JWK needed to verify the proof, whose thumbprint must match the *did:self* identifier.

The proof's *payload* includes the following *claims*:

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

The proof is signed using the private key that corresponds to the public key named in the DID. Given a *did:self* identifier, a DID document, and its proof, we can check whether the DID document matches the given identifier as follows:

1) Check that the identifier in the DID document is equal to the thumbprint of the `jwk` in the proof's header.
2) Check that the SHA-256 hash of the DID document is equal to `s256` in the proof's payload.
3) Check using `exp` that the proof has not expired.
4) Verify the proof using the `jwk` in the proof's header.

B. Verifiable Credentials

A *Verifiable Credential* (VC) [10] allows the *issuer* of the VC to assert a number of *attributes* about the *subject* of the VC. Threfore, a VC must at least include the identities of the issuer and the subject, as well as the asserted attributes; it may also include constraints, such as an expiration time. A VC can be encoded in different ways; in our scheme it is encoded by its issuer as a JSON Web Token (JWT) and embedded in a JSON Web Signature (JWS). Such a VC includes (at least) the following claims (see also Listing 2):

- `jti`: A VC *identifier* unique to the issuer.
- `iss`: The issuer's *did:self* identifier.
- `sub`: The subject's *did:self* identifier.
- `iat`: The VC's time of issue.
- `exp`: The VC's time of expiration.
- `vc`: The assertions of the VC (see Section II-D).

The JWS header may also include the *kid* claim that indicates the key that can used to verify the corresponding signature (see Section II-D). The data model for VCs allows defining different VC *types*, stating the attributes that the VC should include. Our solution uses a new VC type named *authorization*, which includes the *prefixes* that the VC's subject can advertise.

C. System entities

Our scheme involves content *owners*, content *publishers*, and *edge content routers* (see also Fig. 1). Content owners create content items and authorize publishers to store them. A publisher may control multiple *publishing nodes*, as in a CDN, with each node *advertising* the prefixes of the content items that it hosts. Each publishing node can be attached to a different edge content router (CR), using its own secret keys to sign advertisements. Edge CRs verify these advertisements; if verification succeeds, they are forwarded to the core network.

A content owner can generate multiple *did:self* DIDs to be used as content name *prefixes* for separate namespaces. Each content item is identified by a unique hierarchical name prefixed by a *did:self* DID, e.g., "did:self:abc.../aueb/mmlab".

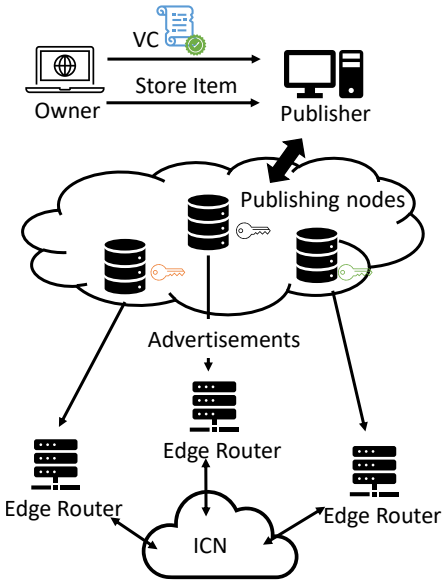

Fig. 1. System overview.

The *did:self* DID used as the root of a content name is called DID_{root}: many content items may share the same DID_{root}. Publishers also generate their own *did:self* DID; the corresponding DID document includes an *authentication* public key, used to authenticate the publisher's prefix advertisements. Finally, each edge CR is assumed to "know" the DIDs of the publishers attached to it, accepting prefix advertisements only from authorized publishers. How this is achieved is out of scope of this work, but it can also be implemented with VCs.

D. Publisher authorization

A content owner can authorize a publisher to advertise a content name prefix starting with DID_{root} by issuing an *authorization* VC. An example of such a VC is given below.

```
{
  "jti": "cred1",
  "iss": "did:self:...",
  "iat": 1650557539,
  "exp": 1681661539,
  "sub": "did:self:...''
  "vc": {
    "credentialSubject": {
      "allowedPrefixes": [
        "/pictures/holidays'',
        "/videos/holidays''
      ]
    }
  }
}
```

Listing 2. A sample authorization VC.

As mentioned above, this VC is included in a JWS. The JWS header should include the *kid* claim, whose value should be equal to the *id* of an *assertion key* defined in the owner's DID; this key can be used to verify the JWS signature. The

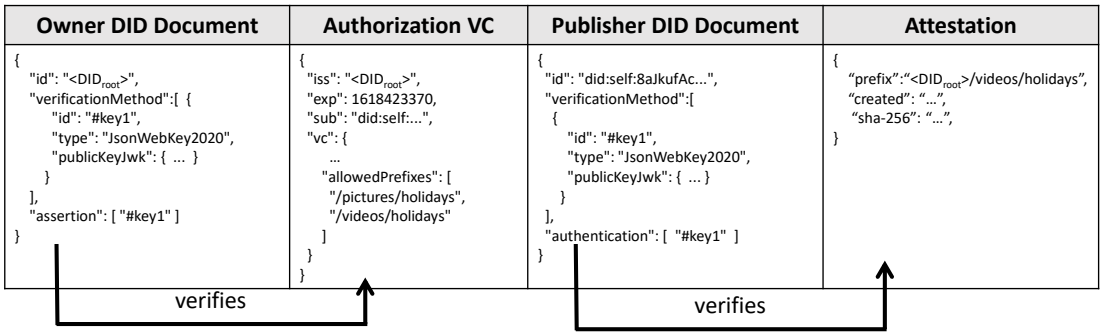

Fig. 2. The header of a content name prefix advertisement. The arrows show which key verifies what.

owner can may employ multiple assertion keys: by looking at the *kid* claim, a verifier can choose a key for verification.

Each owner must create a unique *jti* for each VC it issues. If two VCs, issued by the same owner, have the same *jti*, then the oldest VC is considered invalid. We utilize this property for VC revocation (see Section III-B2).

E. Prefix advertisement

A publisher in NDN must advertise the prefixes it hosts to its attached edge CRs; in our solution, these advertisements are extended with a *header* including (a) the VC issued by the content owner, (b) the DID documents of the content owner and the publisher (and their proofs) and (c) an *attestation*. The attestation is a compact serialization of a JWS, whose payload is a JSON object with the following fields:

- `prefix`: The content prefix being advertised.
- `created`: The time of the attestation's generation.
- `sha-256`: The advertisement payload's SHA-256 hash, encoded using base64url.

The attestation is signed by the publisher and can be verified using an authentication key from the *publisher's* DID document. Similar to VC signatures, the JWS header of an attestation may include the *kid* claim, indicating one of the (possibly many) authentication keys listed in the publisher's DID. An example advertisement header is shown in Fig. 2.

The validity of an advertisement is verified by an edge CR using the following steps (see also Fig. 2):

1) Extract the DID documents of the content owner and publisher and verify them.
2) Verify the signature of the authorization VC using the right assertion key from the owner's DID document.
3) Verify that the publisher's *did:self* identifier is included in the `sub` claim of the authorization VC.
4) Verify the signature of the attestation using the right authentication key from the publisher's DID document.
5) Verify that the sha-256 hash of the advertisement's payload is equal to the `sha-256` field of the attestation.
6) Verify that the advertised prefix is equal to the `prefix` property of the attestation and is included in the `allowedPrefixes` property of the authorization VC.
7) Verify that the attestation is recent using the `created` property of the authorization VC.

TABLE I
EXECUTION TIMES.

Operation	Time (ms)
Generation of Ed22519 key pair	46
DID document and proof generation	2.7
VC generation	2.7
Attestation generation	0.7
DID document verification	1.5
VC verification	1.5
Attestation verification	0.2

With steps 1 to 4 the integrity of the attestation is verified. With step 5 the integrity of the advertisement is verified. With step 6 the authorization of the publisher is verified. Finally, step 7 is used to prevent replay attacks.

III. IMPLEMENTATION AND EVALUATION

A. Computation and communication overhead

We implemented the DID functionality of our solution using the Python3 implementation of *did:self*.[2] The VC functionality was implemented via the JWCrypto library,[3] and the SHA-256 hashes were calculated with Python's hashlib library.

For the evaluation we used Ed22519 keys to create DIDs and EdDSA to sign DID documents. To create a DID, a content owner or a publisher must generate an Ed22519 key pair, a DID document and the corresponding proof. In addition, a content owner must generate an authorization VC and a publisher must sign the "attestation" of the advertisement. Finally, to check an advertisement, an edge CR must verify the DID documents of the owner and the publisher (using the provided proofs), the authorization VC and the attestation of the advertisement. We executed all these operations on a PC running Ubuntu 18.04 with an Intel i5 CPU clocked at 3.1Ghz and 2GB of RAM and show their execution times (in ms) in Table I. Apart from key generation, which is infrequent, all other operations take less than 3 ms, with attestation generation and verification taking less than 1 ms.

The communication overhead of our system is due to the introduction of the advertisement header. Table II shows the sizes of the fields in the advertisement header (in bytes). The values in this table assume that each DID document includes

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

TABLE II
SIZE OF ADVERTISEMENT HEADER FIELDS.

Component	Size (bytes)
DID document and proof	532
Authorization VC	417
Attestation	296

one public key and that the authorization VC includes two content prefixes, each 18 bytes long.

B. Security evaluation

1) Security properties: Our solution is resilient even against active attackers. An advertisement replayed at another edge CR will be rejected, since edge CRs accept advertisements only from specific publisher DIDs. The signed attestation included in the advertisement header protects an advertisement's integrity. Moreover, the attestation timestamp prevents attackers from "replaying" old advertisements to the same edge CR.

2) Impact of key breaches: Our solution relies on the following private keys: the keys that correspond to the DIDs of the content owner and publisher, the assertion keys of the content owner, and the authentication keys of the publisher.

The private key of the owner's DID is the most critical asset in our system, since the breach of this key will result in content owners losing control of the corresponding DID_{root}. We are investigating solutions to this problem, including the use of *recovery keys* combined with revocation mechanisms.

If the private key of the publisher's DID is breached, the publisher must generate a new DID and receive new authentication VCs. The new VCs must have the same *jti* (i.e., VC identifier) as the old ones. However, an attacker that has access to the breached key will be able to use the old VCs until they expire. To prevent this attack, an edge CR can keep track of the *jti* identifiers of non-expired VCs and their corresponding creation time: if a VC with a known *jti* but an older creation time is used, it will be considered invalid.

If the assertion key of the owner is breached, an attacker that has access to the breached key can use it to issue arbitrary VCs, up until the DID document of the owner expires. Since there is no central point where DID documents are stored and there is no revocation mechanism, this issue can be addressed by having owners use the same id for the new key and also have edge CRs store pairs of (owner assertion key id, DID document issuance date); then, if they see a key id included in an older DID document, they will reject it.

If the authentication key of the publisher is breached, the publisher can generate a new one and update its DID document; there is no need to create a new authorization VC. An attacker with access to the breached key can use it to sign attestations for advertisements towards the edge CR to which the publisher is attached. Nevertheless, the attacker cannot modify the publisher's DID document or its proof. Therefore, the edge CR will receive two DID documents for the same DID; since the corresponding proofs include the creation time, the edge CR can mark the old DID document as invalid and keep it in a "rejection list" until it expires.

3) Comparison with NDN: NDN can support similar functionality to our solution through *trust schemata* [5]. In particular, each CR in NDN can be configured with security policies that define trusted public key identifiers per operation. At the same time, each router is configured with trust "anchors" which are used to securely resolve the actual public keys, using a Web-PKI like approach. Trust anchors, therefore, hold the role of a TTP, which creates well-known security concerns.

Another notable difference between the NDN approach and our solution is related to who is in control of defining trust relationships. In our solution, trust relationships are encoded in a VC issued by the content owner, whereas in NDN trust relationships are encoded in a trust schema defined by the CR's administrator. We postulate that our solution offers easier security management (for example, adding or removing a trusted entity in NDN, such as a CDN-like provider, requires re-configuring all routers), and it is less susceptible to human errors (in NDN all routers must be carefully configured with the trust schemata, whereas in our solution they only have to validate a VC and two DID documents).

C. Integration with NDN

The proposed solution has been implemented as an NDN application and has been validated in the NDN testbed.[4] The NDN testbed is a global shared resource created for research purposes that includes software CRs at more than 35 participating institutions, along with application host nodes, and other devices. The testbed is centrally managed and runs a routing protocol that allows communication with every other node. The testbed enables broad areas of research on virtually any type of application and forms a network for real-life evaluation. Testbed nodes are configured with a certificate issued by a testbed-wide Certificate Authority (CA), which is used for signing content name prefix advertisements. We deployed edge nodes directly attached to testbed nodes: publishing nodes interacted with the testbed nodes only through an edge CR. Publishing nodes used our advertisement verification scheme. Upon receiving an advertisement, an edge CR verified the advertisement header and if all verifications were successful the advertisement was removed from the advertisement and replaced with a legacy NDN advertisement header, after which the advertisement was forwarded to the NDN testbed. This procedure made our solution transparent to the testbed nodes.

IV. RELATED WORK

Legacy NDN uses digital *certificates* [11] to achieve the same goals as our solution. However, this approach requires a *trust anchor* that will vouch for the validity of all certificates. The drawbacks of this approach are discussed in detail in Section I.A. Note that, although the naming scheme of our solution is different to NDN's scheme, it is not incompatible: in our solution the DID-based prefix holds the role of the trust anchor, and the DID document holds the role of the certificate.

Our solution relies on using cryptographicly verifiable content name prefixes: this approach has been used by many other

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

systems. For example, in DONA [12] each content item is identified by a pair of labels, namely P and L: in the general case P corresponds to the public key of the data owner, and L to a data label. The InterPlanetary File System (IPFS) uses content hashes as names [13]. Our solution allows multiple signers per content name (prefix) without requiring secret key sharing, as in DONA. Similarly, our solution enables security for both mutable and immutable items, unlike with IPFS.

The naming scheme of the NetInf architecture is very close to our system [14]. NetInf uses the hash of a public key P as part of the content name. The private key corresponding to P is used to sign a metadata field that includes, among others, the hash of the content item, hence verifying the content item's authenticity and integrity. Furthermore, using a chain of certificates rooted at P, additional keys can be authorized to sign the metadata field. In our solution, this "abstract" certificate chain mechanism is realized using the emerging standards of DIDs and VCs. Furthermore, we support non-transitive delegations, i.e., a publisher cannot further delegate content prefix ownership, and fine-grained delegations, i.e., a content prefix owner can delegate a subset of its namespace.

We have previously proposed a DID-only solution to this problem [15], where the DID document of the owner indicated the public key used by the publisher to sign content advertisements; via a *proof* attached to the DID document, this authorization was limited to specific prefixes and edge CRs. This meant that changing the publisher's key or changing publishers, required changing the owner's DID document. In our combined DID–VC solution, the owner's DID document only contains the owner's keys, therefore it does not need to change, as long as these keys are not compromised. The keys from the DID document are used to sign the VCs, which determine the authorized publisher; new VCs can add more publishers. Furthermore, the VCs identify the publisher through its DID, which includes the actual signing keys, hence the publisher can rotate keys without reissuing its VCs.

V. Conclusions and Future Work

We presented a security solution that allows NDN content routers to verify that context prefix announcements originate from authorized nodes. Our solution leverages the recent W3C recommendations for Decentralized Identifiers (DIDs) and Verifiable Credentials (VCs). Our solution does not require a trust anchor and enables trust relationships such as delegation.

Compared to the existing solutions that rely on public keys and digital certificates to offer similar functionality, our solution provides increased security, since cryptographic keys can be easily rotated, and improved security management, since all operations are implemented in a decentralized manner.

We have developed and evaluated a proof-of-concept implementation for NDN that uses the *did:self* DID method. Our solution is compatible with NDN's routing protocols and does not require any modification to the core architecture. The use of *did:self* offers significant advantages, such as support for DID sharing and improved DID management without registries; nevertheless, our solution can be easily adapted to accommodate other DID methods. Future work in this direction includes supporting resilience to content owner DID key breaches, as well as supporting human readable names.

Acknowledgment

The work reported in this paper has been partly funded by the EU's Horizon 2020 Programme through the subgrant Securing Content Delivery and Provenance (SECOND) of project NGIatlantic.eu, under grant agreement No 871582.

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 & Tutorials*, vol. 16, no. 2, pp. 1024–1049, 2014.

[2] 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, pp. 66–73, Jul. 2014.

[3] B. Carpenter and B. Liu, "Limited Domains and Internet Protocols," Internet Requests for Comments, IETF, RFC 8799, July 2020. [Online]. Available: https://tools.ietf.org/html/rfc8799

[4] V. Lehman, A. M. Hoque, Y. Yu, L. Wang, B. Zhang, and L. Zhang. (2016) A secure link state routing protocol for NDN. [Online]. Available: https://named-data.net/wp-content/uploads/2016/01/ndn-0037-1-nlsr.pdf

[5] Y. Yu, A. Afanasyev, D. Clark, k. claffy, V. Jacobson, and L. Zhang, "Schematizing trust in named data networking," in *Proc. of the ACM Conference on Information-Centric Networking (ICN)*. New York, NY, USA: ACM, 2015, p. 177186.

[6] W3C Credentials Community Group. (2020) A Primer for Decentralized Identifiers. [Online]. Available: https://w3c-ccg.github.io/did-primer/

[7] N. Fotiou. (2021) did:self method specification. [Online]. Available: https://github.com/mmlab-aueb/did-self

[8] 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

[9] W3C Credentials Community Group. (2019) DID method registry. [Online]. Available: https://w3c-ccg.github.io/did-method-registry/

[10] Manu Sporny et al., "Verifiable credentials data model v1.1," W3C, W3C Recommendation, 2022. [Online]. Available: https://www.w3.org/TR/verifiable-claims-data-model/

[11] 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.

[12] 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.

[13] 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.

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

[15] N. Fotiou, I. Thomas, V. Siris, G. Xylomenos, and G. Polyzos, "Securing named data networking routing using decentralized identifiers," in *Proc. of the IEEE International Conference on High Performance Switching and Routing (HPSR)*. New York, NY, USA: IEEE, 2021, pp. 1–6.