Securing Named Data Networking routing using Decentralized Identifiers

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

***Abstract*—Named Data Networking (NDN) is a realization of the Information-Centric Networking (ICN) paradigm, where routing is based on content identifiers rather than on network location identifiers. The routing state in NDN can grow exponentially, not only due to the huge number of content identifiers (as opposed to network addresses) but also because it is difficult to detect "fake" routing advertisements. For example, in contrast to IP-based routing, a potentially valid routing entry in NDN can be advertised from multiple network locations, making NDN susceptible to Denial-of-Service attacks at the routing layer. In this paper, we leverage Decentralized Identifiers (DIDs) to build self-verifiable "content advertisements." With our solution, any router can verify that a content advertisement originates from an "authorized" entity, without requiring any trusted third party. We implement our solution and we evaluate it in a scenario where filtering is implemented by the edge routers. We show that our solution reduces fake routing advertisements with minimal computational overhead.**

***Index Terms*—Self-sovereignty, DID, fake advertisements, self-verifiable content advertisements, edge networking.**

I. Introduction

Information-Centric Networking (ICN) has been in the spotlight of many recent research efforts around the world [1], since it promises improved security, efficient information mediation, and improved governance [2]. In this paper, we focus on the predominant realization of the ICN paradigm, *Named Data Networking* (NDN) [3]. We consider the case of a content *owner* wishing to share a piece of content (see Fig. 1). The owner does not interact with the NDN network directly, instead it uses a "hosting service" referred to as the *publisher*. A publisher can be a Web server, an online storage system, or even a Content Distribution Network (CDN). Publishers are responsible for *announcing* content items they *host* to the NDN network, thus allowing content *consumers* to retrieve them. A publisher may want to make the same content available from many different locations where it has points of presence; this is, indeed, the standard operation of CDNs. Content advertisements are initially received by an *edge router*, which is responsible for disseminating them to the rest of the network. We are considering a single network and we are assuming an intra-domain routing protocol. Our goal is to enable edge routers to detect malicious publishers who try to pollute the routing state of the network by advertising content items that they are not authorized to host.

A. Routing in NDN

The NDN architecture includes mechanisms that enable content discovery and delivery based solely on content names, instead of location identifiers. Each content item is identified by a unique *name*. In NDN, a *consumer* expresses her interest for a content item by sending an *Interest* packet. Interests are routed by *content routers* (CRs) towards content *publishers* based on a lookup table, which maps content names to output interface(s), the *Forwarding Information Base* (FIB).

The FIB entries of CRs are populated using a link state routing protocol, known as "Named-data Link State Routing protocol (NLSR)" [4]. Content name prefixes are advertised to edge CRs, which in turn propagate these advertisements using NLSR. NDN (and ICN in general) supports replicating content items in multiple network locations, thus facilitating multihoming and multisourcing, and also supports publisher mobility. Therefore, a CR may receive multiple *valid* content advertisements from multiple locations, and it should update its FIB to include all of them. As a result, it is harder to filter out fake advertisements compared to, for example, a routing protocol which is used to propagate location-dependent identifiers (e.g., network addresses in IP).

NDN tries to solve this problem using digital signatures, by allowing content advertisements to be signed. This enables edge CRs to verify these signatures based on pre-configured rules [5]. In a typical deployment, edge CRs would use these rules to verify a chain of signatures rooted in a pre-installed *trust anchor*. In the example of Fig. 1, the role of the trust anchor is held by a *Trusted Third Party* (TTP), the certificate of which is installed in all edge routers. The TTP has issued a certificate for the content owner, and the owner has issued a certificate for the publisher. The publisher signs content advertisements and includes in the advertisement messages "pointers" to its certificate, as well as to the owner's certificate. Therefore, an edge router validates signed advertisements in two steps: first, it retrieves the certificates of the owner and the publisher and, second, it verifies that the certificate of the publisher has been issued by the owner, and that the certificate of the owner has been issued by the TTP.

B. Solution overview and contributions

In this paper, we design a solution that removes the need for TTPs. In particular, we propose that each content owner issues

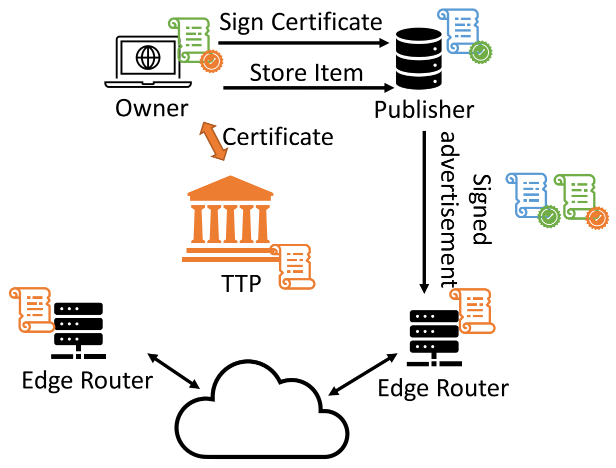


Fig. 1. Trust relationships in legacy NDN routing. Edge routers are configured with the certificate of the TTP. The TTP issues a certificate to the owner and the owner issues a certificate to the publisher. The publisher includes both certificates in the advertisement.

Decentralized Identifiers (DIDs), which are used as content name prefixes. A DID, which is a new identification paradigm that is getting increased attention from academia and industry, can be regarded as an opaque URI, which is associated with public keys and other auxiliary information.

A content owner can prove ownership of a DID and it can also authorize a publisher to advertise a content name prefix that includes the DID, on the owner's behalf. With our solution we make the following contributions:

- We allow publishers to prove that they have been authorized to advertise a particular content name prefix to a specific edge router.
- We enable edge routers to verify these proofs without relying on third parties or requiring edge routers to be pre-configured with information specific to our solution.
- We facilitate the detection of attackers using keys that have been revoked. Furthermore, we limit the impact of publisher key breaches, even when they are not detected.
- We improve content owners' privacy and simplify namespace management.

The remainder of this paper is organized as follows. In Section II we introduce Decentralized Identifiers and we present the design of our system. In Section III we present the implementation of our system, we discuss its performance, it security properties, and we compare it against a TTP-based alternative. Finally, we present our conclusions and discuss future work in Section IV.

II. System Design

A. Decentralized Identifiers and the did:self method

Decentralized Identifiers (DIDs), defined by W3C, are a new type of identifier that is globally unique, resolvable with high availability, and cryptographically verifiable [6]. A DID, which is a simple URI, is associated with a DID *document* that includes public keys, authentication protocols, and service endpoints necessary to bootstrap cryptographically-verifiable interactions with the identified entity [6]. Usually, a DID document is maintained by a DID *registry* which is responsible for implementing proper security and access control mechanisms. Registries allow third parties to lookup DID documents and provide *proofs* of correctness (for example, a proof can be a digital signature generated by the registry).

DID specifications do not dictate the actual contents of a DID document, neither do they define how a registry operates. Instead, these are left as design choices to individual DID instantiations, also referred to as DID *methods*. Our system uses a DID method we devised, *did:self* [7]. A key property of *did:self* is that it does not require any trusted registry; DID documents can be directly transmitted to interested parties or stored in publicly accessible locations. The *did:self* method assures that a DID document can be verified as "correct" even if it is retrieved over an unsecured channel.

A *did:self*-based DID is a base64url encoded Ed22519 public key [8], [9] prefixed with the string 'did:self:'. A *DID document* in *did:self* is a JSON-encoded document that may include any of the DID "properties" defined by the DID specifications. Our solution uses the following properties:

- `id`: The DID which the document concerns.
- `assertion`: An Ed22519 public key expressed using the "JsonWebKey2020" notation [10].

The *assertion* key is used in our system for signing content advertisements. An example of a DID document is included in the following listing, where line 2 includes the DID, and lines 3-10 define the assertion (key), which is in essence the JSON Web Key (JWK) representation [11] of an Ed22519 public key (lines 7-9). Observe that the assertion property includes an "id", the *key identifier*, which is used to build a compromised key detection mechanism in the following section.

```
1    {
2      "id": "did:self:6varD0Rj...",
3      "assertion": {
4        "id": "publisherA.com#key1",
5        "type": "JsonWebKey2020",
6        "publicKeyJwk": {
7          "crv": "Ed25519",
8          "x": "7wJkufDc...",
9          "kty": "OKP"
10       }
11     }
12   }
```

Listing 1. An example of a DID document used in our solution.

Additionally, each DID document is associated with a *proof* which is a "compact serialization" of a JSON Web Signature (JWS) (section 3.1 of [12]). The payload of a proof is a JSON document that includes the following properties:

- `id`: The DID.
- `created`: The date and time when the proof was generated.
- `expires`: An optional expiration time.
- `sha-256`: The base64url encoded hash of the DID document, calculated using SHA-256.

- `caveats`: A list of application layer "restrictions" on the scope of the DID document. In our system, this property is used to control the prefixes that a publisher can advertise and limit the advertisements to one or more content routers.

The signature of the proof is generated using *Edwards-curve Digital Signature Algorithm* (EdDSA) and the private key that corresponds to the DID. The proof is used to validate the binding between a DID document and a *did:self* DID. In particular, given a DID, a DID document, and the document proof, any entity can trivially verify whether the DID document corresponds to the DID by executing the following steps:

1) Verify that the DID is included in the `id` field of the proof.
2) Verify that the digest of the DID document is the same as the `sha-256` field of the proof.
3) Verify that the proof has not expired, if the `expires` field is set.
4) Verify the signature of the proof using the *did:self* DID (recall that a *did:self* DID *is* a public key.)

With our solution, *DID documents and their proofs are integrated into the content item advertisements.*

B. System entities

Our system considers content *owners*, *publishers*, and *edge routers* (see also Fig. 2). Content owners generate content items and pass them to a publisher, while publishers are responsible for advertising the content items they host to the edge routers to which they are attached. Edge routers communicate with the rest of the NDN network using standard, unmodified NDN protocols.

Each content owner can generate an arbitrary number of *did:self* DIDs, which are used as content name *prefixes*. In particular, each content item is uniquely identified by a hierarchical name rooted at a *did:self* DID, e.g., "did:self:abc.../holidays/video1/chunk1". We refer to the *did:self* DID used as the root of a content name as DID_{root}: many content items, that belong to the same owner, may share the same DID_{root}. Moreover, publishers own public-private key-pairs, that are uniquely identified by a key_{id}. Finally, each edge router is identified by a unique $router_{id}$. The format of a $router_{id}$ is deployment specific; our solution only requires that publishers know the identifiers of the edge router(s) to which they are attached (which is the case in typical NDN deployments).

C. Publisher authorization

A content owner can authorize a publisher to advertise a content name prefix rooted at a DID_{root}, to a specific router identified by $router_{id}$. Content advertisements will be secured using the publisher's key-pair key_{id}. This is achieved as follows. The content owner generates a DID document for DID_{root} that includes an assertion with "id" equal to key_{id} and a "publicKeyJwk" equal to the JWK representation of the public key that corresponds to key_{id}. Essentially, this "delegates" publishing rights to the publisher that holds that key_{id}. Moreover, the proof of this document, which is signed using the private key that corresponds to DID_{root}, includes in its `caveats` field the content name prefix which the publisher is authorized to advertise, and the $router_{id}$. Finally, the owner sends the DID document and its proof to the publisher.

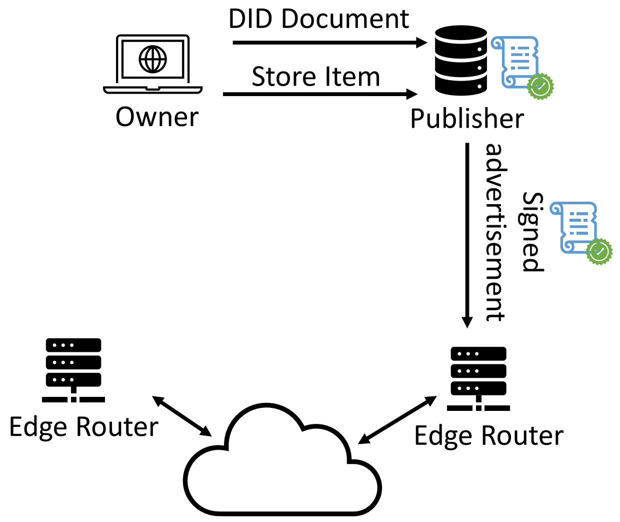


Fig. 2. System entities and interactions. The content owner generates a DID document which is stored in the Publisher. The publisher includes this DID document in its advertisements.

An example of a DID document and its proof is included in the first two columns of the table in Fig. 3. In this example, a publisher is authorized to advertise the prefix "DID_{root}/videos" to router "$router_{id}$".

D. Content name advertisement

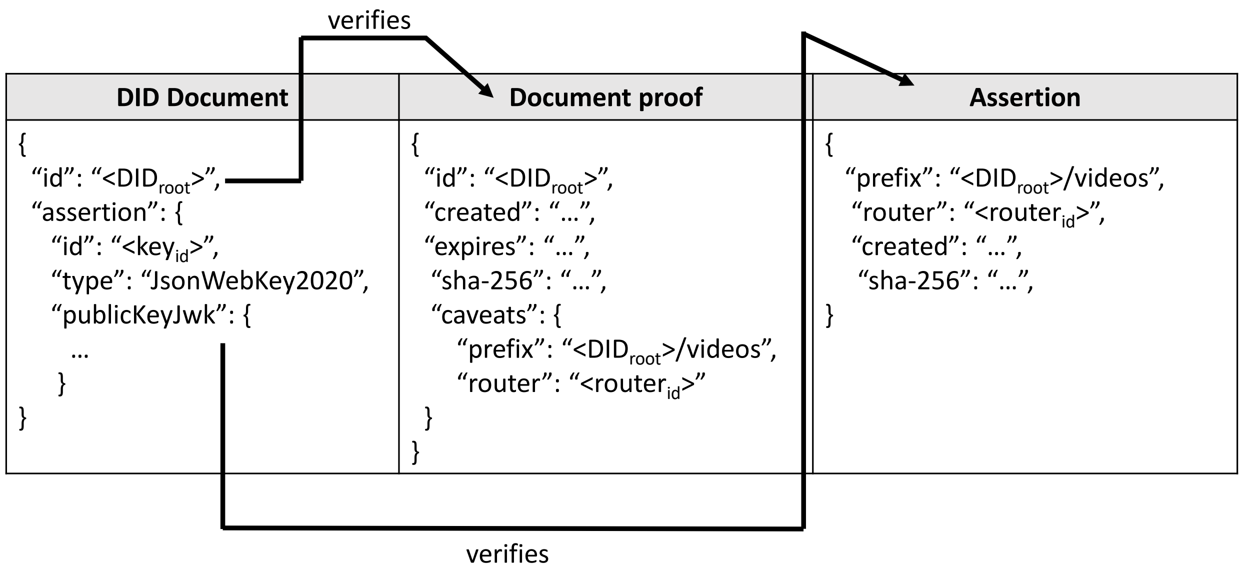
Publishers advertise content name prefixes to edge routers. In our system, advertisements are expanded with a *header* that includes the DID document and the proof received from the content owner, using the procedure described in the previous section, as well as an *assertion*, a compact serialization of a JWS. The payload of an assertion is a JSON document that includes the following properties:

- `prefix`: The advertised prefix.
- `router`: The $router_{id}$ of the edge router.
- `created`: The date and time when the assertion was generated.
- `sha-256`: The base64url encoded hash of the advertisement payload.

The signature of the assertion is generated by the publisher using EdDSA and the private key that corresponds to key_{id}. An example of an assertion is included in the last column of the table in Fig. 3

An edge router can verify the validity of an advertisement by taking advantage of the added header as follows:

1) It extracts the DID document and its proof, and it verifies that this is a valid DID document for DID_{root}.
2) It extracts the assertion key from the DID document and it uses it to verify the signature of the assertion.



```
{
  "id": "<DID_root>",
  "assertion": {
    "id": "<key_id>",
    "type": "JsonWebKey2020",
    "publicKeyJwk": {
      ...
    }
}
```

```
{
  "id": "<DID_root>",
  "created": "...",
  "expires": "...",
  "sha-256": "...",
  "caveats": {
    "prefix": "<DID_root>/videos",
    "router": "<router_id>"
  }
}
```

```
{
  "prefix": "<DID_root>/videos",
  "router": "<router_id>",
  "created": "...",
  "sha-256": "...",
}
```

Fig. 3. A content name prefix advertisement. The signature of the document proof is verified using DID_{root} and the signature of the assertion is verified using key_{id}. For clarity, signatures are omitted from the figure.

3) It calculates the sha-256 hash of the advertisement payload and it verifies that it matches the value included in the `sha-256` field of the assertion.
4) It verifies that the `prefix` property of the assertion is the advertised prefix and that it is included in the `caveats` property of the document proof.
5) It verifies that the `router` property of the assertion includes the correct $router_{id}$ and that $router_{id}$ is included in the `caveats` property of the document proof.
6) It verifies that the assertion is "adequately fresh" using the `created` property.

Steps 1 and 2 are used to verify the integrity of the assertion. With step 3, the edge router verifies the integrity of the advertisement. Step 4 is used to verify that the publisher is authorized to advertise this specific prefix. In step 5, the edge router verifies that it is the intended recipient and that the publisher is authorized to advertise the prefix to that specific edge router. Finally, the router detects a "replayed" advertisement in step 6.

E. Publisher key rotation

In case a key used by a publisher to sign assertions is breached, it must be replaced with a new one, and the publisher must receive new DID documents from the content owner(s). In our system we follow the convention that the new key will have *the same* key_{id} as the replaced one.

An attacker that found out the breached key can use it to sign assertions but cannot modify the corresponding, old, DID document. Therefore, an edge router receiving two *valid* advertisements, which include DID documents that use the same "id" for the assertion property but have different "publicKeyJwk" values, will understand that one of these is not valid. In that case, the edge router will mark the key included in the *oldest* DID document as "forbidden" and it will store it in an internal "key rejection list".

All DID documents that include the same publisher public key should use the same key_{id}, even if they were generated by different content owners. This is easy to achieve, since the publisher passes this information, along with the key, to each owner. As a result, a structure that maps key_{id}s to public keys stored in an edge router will have size proportional to the number of publishers' public keys, regardless of the number of content owners and the number of prefixes.

III. Implementation and Evaluation

A. Performance evaluation

For the evaluation, we have used the Python3 implementation of *did:self*.[1] JWS generation, verification, and serialization are implemented using the JWCrypto library.[2] SHA-256 hashes are calculated using Python's hashlib library.

In our system the following cryptographic operations have to be performed. For the creation of the DID, a content owner has to generate an Ed22519 key pair, a DID document, and the corresponding proof. A publisher has to sign the "assertion" field of the advertisement. Finally, for the advertisement verification, an edge router has to verify the DID document using the provided proof, as well as the signature of the assertion.

Table 1 shows the time required (in ms) to perform the cryptographic operations of our system, as measured in a desktop PC running Ubutnu 18.04, on an Intel i5 CPU, 3.1Ghz with 2GB of RAM. It can be seen that most operations are executed in less than 3 ms.

When it comes to storage overhead, Table 2 shows the size of the various components of the advertisement header

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

TABLE I
CRYPTOGRAPHIC OPERATIONS REQUIRED BY OUR SYSTEM AND THEIR OVERHEAD.

Operation	Time (ms)
Key pair generation	46
DID document and proof generation	2.7
JSON web signature calculation and serialization	0.7
DID document verification	1.5
JSON web signature verification	0.2

TABLE II
SIZE IN BYTES OF THE COMPONENTS OF AN ADVERTISEMENT.

Component	Size (bytes)
DID document	427
Proof	512
Assertion	304

(in bytes). For this measurement we set the size of the variable length fields to the following values: key_{id} = 20 bytes, $router_{id}$ = 20 bytes, `prefix` = 80 bytes and `caveats` = 100 bytes.

B. Security evaluation

1) Security properties: Our solution has intriguing security properties, even against active attackers.

The signed assertion included in the advertisement header protects the integrity of the advertisements. Furthermore, by including $router_{id}$ in the assertion, attackers are prevented from injecting a valid advertisement captured from another link. Similarly, by including a timestamp in an assertion indicating when it was created, prevents attackers from "replaying" "old" advertisements. Of course this requires the clocks of the publishers and the edge routers to be loosely synchronized. Although this is not hard to achieve, since usually publishers and edge routers are one network hop way and they belong to the same network, an alternative approach is each publisher to use an incremental serial number: then, edge routers can store the last value of the serial number of each publisher and reject advertisements that include older serial numbers.

Even if an attacker can access the assertion key of a publisher, he cannot modify the DID documents that the publisher has received: this limits the damage an attacker can make. In particular, for the lifetime of a DID document and providing that the key breach has not been detected, an attacker can send advertisements for a specific prefix to a specific router. This has an impact on the routing state only if the publisher does not host anymore the advertised prefix.

2) Comparison to a TTP-based solution: Our solution removes the need for a TTP. Using a TTP has the following disadvantages:

All entities must agree on the set of TTPs. When a TTP-based solution is used, a TTP must be trusted by all content owners, as well as by the network provider. This can become really challenging, e.g., in cases of content owners that roam from network to network.

A TTP is a significant security threat. In the general case, a TTP can generate certificates for any name prefix, hence a malicious TTP, or an attacker that has access to the private key of a TTP, can generate an arbitrary number of valid certificates.

Security management is harder. In case a TTP has to revoke its signing key then, all generated certificates must be re-issued, and–more importantly–all edge routers must be configured with the new TTP key. Moreover, in case a content owner needs to update her certificate she must be able to prove to the TTP that she is the legitimate owner of the prefix included in the certificate, otherwise an attacker could receive a certificate for a prefix he does not own, hence he will be able to impersonate a valid owner.

On the other hand a TTP can limit the number of prefixes a content owner can use, whereas in our system content owners can generate (by themselves) an arbitrary number of prefixes, thus increasing routing state. In any case, limiting the number of prefixes an owner can use comes with some costs. Firstly, it creates a privacy risk since it becomes easier to track the content items of a specific owner, as well as to filter/censor a specific owner. Secondly, in case there are multiple TTPs they have to be synchronized so that each TTP knows which prefixes are available and which are already assigned to owners. In our system, a content owner not only is allowed to generate as many prefixes she wants (even a prefix per content-item) but it is statistically guaranteed that these prefixes are unique, hence a third party keeping track of the available prefixes is not required.

Another important property of TTP-based systems is that content owners do not necessarily lose control of the prefixes they own if their keys are breached or lost. This is not the case in our system: the security of *did:self* DIDs relies on the ability of content owners to protect the corresponding private keys; if an owner's private key is lost, the attacker has equal rights to the DID (and its namespace) as the owner.

IV. CONCLUSIONS AND FUTURE WORK

In this paper, we presented a solution for protecting routing state in Named Data Networking (NDN). Our solution leverages Decentralized Identifiers (DIDs), it enables content owners to authorize publishers to advertise content prefixes they own, and it allows edge routers to verify the validity of content advertisements. Our solution removes the need for a Trusted Third Party (TTP), while being lightweight, secure, and supporting seamless integration with the NDN edge routers. Furthermore, our solution does not require any configuration state to the edge routers, such as installing keys. The proposed scheme allows content owners to generate by themselves as many statistically unique prefixes they want, simplifying namespace management, while protecting content owners from tracking and censorship. Finally, our solution does not affect routing in the core network, neither does it require modifications to the application layer protocols.

Currently, we have developed an "emulated" edge router that implements our solution, in the context of the project "Self-Certifying Names for Named Data Networking (SCN4NDN)" [13]. We are actively working on integrating the proposed solution in the core NDN libraries.

Although our solution was designed for NDN, we believe that it has applications to other systems that involve similar routing protocols. We have already investigated the use of our solution for protecting mutable content in the Inter-Planetary Files System [14]. Future work in this direction involves the application of our solution in service-oriented architectures, as well as the integration of our constructions to protocols such as the "Cryptographically Generated Addresses" [15].

The presented solution does not exploit all features offered by the *did:self* DID method. *did:self* supports methods to make key rotation easier and add protection "layers" to the private key that corresponds to a DID. Nevertheless, recovering from a key breach without losing control of the corresponding DID still remains an open, unsolved issue.

A *did:self* DID is not human memorable. Although relying on human memorable identifiers to provide security is a bad practice (since users can easily fall victims of "phishing" or "impersonation" attacks) they are valuable from a usability standpoint. Bridging human memorable names with *did:self* DIDs is another open issue under investigation.

Acknowledgment

This work was supported by a contract with the Waterford Institute of Technology under Article 15 of Grant Agreement number 871582 for financial support to third parties of EU H2020 project NGIatlantic.eu, a Research & Innovation Action in the field of Next Generation Internet.

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

[12] 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

of Information-Centric Networking research," *IEEE Communications Surveys Tutorials*, vol. 16, no. 2, pp. 1024–1049, 2014.

[2] D. Trossen, M. Sarela, and K. Sollins, "Arguments for an Information-Centric Internetworking architecture," *SIGCOMM Computer Communications Review*, vol. 40, no. 2, pp. 26–33, Apr. 2010.

[3] 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.

[4] V. Lehman, A. M. Hoque, Y. Yu, L. Wang, B. Zhang, and L. Zhang, "A secure link state routing protocol for NDN," *Tech. Rep. NDN-0037*, 2016.

[5] Y. Yu, A. Afanasyev, D. Clark, k. claffy, V. Jacobson, and L. Zhang, "Schematizing trust in named data networking," in *Proceedings of the 2nd ACM Conference on Information-Centric Networking*. New York, NY, USA: Association for Computing Machinery, 2015, p. 177–186.

[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] S. Josefsson, "The Base16, Base32, and Base64 Data Encodings," Internet Requests for Comments, IETF, RFC 4648, October 2006. [Online]. Available: https://www.rfc-editor.org/rfc/rfc4648.txt

[9] D. J. Bernstein, N. Duif, T. Lange, P. Schwabe, and B.-Y. Yang, "High-speed high-security signatures," *Journal of cryptographic engineering*, vol. 2, no. 2, pp. 77–89, 2012.

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

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

[13] Mobile Multimedia Laboratory. (2021) Self-Certifying Names for Named Data Networking (SCN4NDN) project home page. [Online]. Available: https://mm.aueb.gr/scn4ndn/

[14] N. Fotiou, V. Siris, and G. Polyzos, "Enabling self-verifiable mutable content items in IPFS using Decentralized Identifiers," in *DI2F: Decentralising the Internet with IPFS and Filecoin, IFIP Networking 2021 workshop*, 2021.

[15] T. Aura, "Cryptographically Generated Addresses (CGA)," Internet Request for Comments, IETF, RFC 3972, March 2005. [Online]. Available: https://rfc-editor.org/rfc/rfc3972.txt