# Certificate Management for Cloud-Hosted Digital Twins

Nikos Fotiou*, Chalima Dimitra Nassar Kyriakidou†, Athanasia Maria Papathanasiou†, Iakovos Pittaras†, Yannis Thomas†, George Xylomenos†
*ExcID, Athens, Greece
fotiou@excid.io
†Mobile Multimedia Laboratory
Department of Informatics, School of Information Sciences and Technology
Athens University of Economics and Business, Greece
{dnassar,sissypapathanasiou,pittaras,thomasi,xgeorge}@aueb.gr

***Abstract*—A key enabler for the digitization of physical devices is digital twining technology. A digital twin is a virtual representation of a physical object (or a collection of physical objects) that allows their integration into cyber systems. Digital twins are usually hosted in cloud environments, which provide high availability and resilience to failures. This integration creates new opportunities and enables new capabilities, but it also raises security concerns. In this paper, we design a digital certificate management solution that allows building trust on digital twins independently of their network location. Our solution allows digital twins to securely receive certificates, which can be used to digitally sign data at the application layer. Our scheme does not depend on the certificate infrastructure used to secure the communication between end-users and the (cloud-hosted) digital twins. Our solution is feasible, realistic and resilient against key breaches, with a marginal communication overhead. Finally, our scheme automates the process of certificate issuance for digital twins, thus enabling very fast key and certificate rotation.**

***Index Terms*—Digital Twins, OpenID Connect, TUF, X.509.**

## I. Introduction

As the *Internet of Things* (IoT) becomes an integral component of our life, the *Digital Twin* (DT) paradigm becomes even more important. A DT is a virtual representation of a physical object, system, or asset [1] that allows its integration into cyber systems enabling the "I" in the IoT. In order to provide high availability and resilience to network errors, DTs often operate in a powerful and secure network location, "hosted" by trusted third parties (e.g., cloud providers). There are already providers that offer such services, such as Amazon[1] and Microsoft[2]. For example, in a smart city transit monitoring system, city buses, trams and, even, public electric bikes, can periodically upload their status to a DT hosted in a public cloud, allowing traffic monitoring applications constant access to their status, as well as enabling long-term traffic data mining via machine learning algorithms. While these (complex) systems provide increased flexibility and advanced monitoring capabilities, they create new security and trust challenges and are often the target of cyberattacks [2].

The need for real-time synchronization of information, the open communication environment, and the, possibly sensitive, information exchanged between the DT and its physical counterpart, highlight the need to ensure their secure communication. In this context, it is crucial for end systems twinned with a DT to be able to verify the integrity and authenticity of received messages. This requires establishing robust trust relationships with DTs, an aspect that many existing systems neglect, considering it something that happens "out-of-band". As a result, in many existing systems, the authentication of and interaction with DTs pose significant challenges, as they typically rely on outdated methods, such as hard-coded credentials, static passwords, or simple access control lists installed at the factory, which are hard or impossible to update. These approaches lack the necessary flexibility, scalability, and resilience against the sophisticated attacks which are commonplace against cloud providers.

To address these challenges, in this paper we focus on DT identification and authentication, designing a secure and resilient certificate management mechanism that allows cloud-hosted DTs to obtain short-lived, and easily rotated, certificates for digitally signing data. Our solution achieves the following:

- It removes the need for DTs to store any secrets.
- It is resilient against malicious *Certificate Authorities* (CA), allowing explicit definition of trusted CA keys.
- It enables automated certificate issuance, thus allowing the frequent rotation of signing keys.
- It has minimal overhead and it builds upon existing standards and protocols.

The remainder of the paper is organized as follows. Section II presents background information and related work in this area. Section III details the design of our solution, while Section IV discuses its implementation and evaluation. Finally, Section V concludes the paper and discusses future work.

## II. Background and Related Work

Various definitions have been proposed for DTs, with the most widely-adopted comprising three main components: a *Physical Object* (PO), a *Virtual Object* (VO), and a mapping between them, which may be one-sided or bidirectional, depending on the system [3]. DTs are commonly utilized for continuous monitoring of network infrastructure; often,

[1]https://aws.amazon.com/iot-twinmaker/
[2]https://azure.microsoft.com/en-us/products/digital-twins/

machine learning algorithms analyze historical data patterns to predict future trends and anticipate performance failures. Moreover, DTs are utilized to enhance network optimization efforts by leveraging data-driven analytics to reduce network latency, increase system speed, and address potential errors or malfunctions in physical devices. Finally, DTs can serve as security implementation tools, conducting risk assessments using real-world data without impacting actual devices.

DT technology has been studied in many security-sensitive domains. Lai et al. [4] use DTs in a traffic control system, where they introduce a privacy-preserving protocol. The protocol includes a group signature scheme, which is responsible for authenticating data sources and enabling efficient revocation and privacy protection. This approach ensures the secure storage of data generated from vehicles on cloud service providers, once synchronized with their DTs. In the data sharing phase, the authors employ *Attribute-Based Access Control* (ABAC), in which the parameters of specific sub-policies are stored during their first decryption and re-used when the same access control sub-policy is relevant again, so as to reduce computational overhead.

Liu et al. [5] leverage DTs to construct a cloud scheme for healthcare data. Their framework enables real-time monitoring, diagnosis, and prediction, using data collected from medical devices, which are transmitted across wireless networks to a cloud server, where a DT is responsible for authorizing users and granting them access to the server's medical data.

Patel et al. [6] propose a user-centric approach to enable privacy-preserving authorization in cloud-based DTs, by leveraging *Decentralized Identifiers* (DIDs) and *Verifiable Credentials* (VCs). In particular, they design three protocols: secure access to a DT's data, command execution on the physical object and recovery in case of private key loss. Similarly, Thakur et al. [7] propose a three-factor authentication scheme for cloud-based DT environments and Chen et al. [8] propose a privacy-preserving protocol for mutual authentication between vehicles, with a DT representing the whole network.

Another framework, proposed by Xu et al. [9], introduces an authentication protocol for secure communication between autonomous vehicles and DTs that consists of two parts: intra-twin authentication and inter-twin authentication. For intra-twin authentication, when an autonomous vehicle joins the system, it establishes a connection with the central authority to obtain information about its personal DT, and they authenticate each other using pseudonyms generated by the aforementioned authority. In inter-twin authentication, after the DT authenticates with the vehicle, it communicates with the central authority to obtain a group certificate.

These systems aim to protect data in transit, implementing end-user authentication and access control under the assumption that DTs are secure and trusted. In contrast, our solution does not assume implicitly trusted DTs; instead, it provides mechanisms that allow end-users to reliably determine whether they are interacting with a trusted DT or not.

Our solution relies on X.509 certificates issued to entities that present a valid OpenID identity token. Recent efforts [10], [11] try to remove the need for a CA, using OpenID identity tokens directly as digital certificates. Although these approaches do result in faster processing, they require new tooling for signing and verifying data. Furthermore, OpenID tokens often include additional information that may be sensitive, hence these approaches introduce privacy threats.

Finally, recent efforts [12], [13] attempt to reduce the amount of trust in (automated) CAs that use OpenID identity tokens for issuing certificates. These approaches, however, require modifications to OpenID endpoints. Our solution is compatible with these approaches and can benefit from a set-up where these solutions are deployed at OpenID endpoints.

## III. Design

Our system considers the following (digital) entities (see also Fig. 1): a device *owner* that owns IoT devices, a cloud *provider* that operates a network of *nodes*, a digital twinning *platform* that provides *Digital Twin* (DT) functionality, a *consumer* that accesses DTs, and a *Certificate Authority* (CA) responsible for issuing digital certificates used by DTs to sign data at the application layer. In the smart city transit monitoring system example, the device owner is a company (or multiple companies) operating buses, trams and/or electric bikes, while the consumer is the city agency that monitors transit operations. The cloud provider and the CA are publicly available services, while the digital twinning platform is software (possibly, off the shelf) that implements DTs.

Our design considers the following identities. An owner is identified by a URL, referred to as $Owner_{URL}$. Usually, this URL is the prefix of the (HTTPS) URL used by a consumer to access the DTs of the corresponding owner. Similarly, a CA is identified by a URL referred to as $CA_{URL}$. Finally, each digital twinning platform instance is identified by a unique identifier, referred to as $Instance_{ID}$.

From a high level perspective, the defined entities interact with each other as follows. The digital twinning platform implements an API that is used by IoT devices to update the state of their corresponding DT and by consumers to access the state of a DT, or receive events related to it. Our design is oblivious to this API; our implementation builds on ETSI's NGSI-LD API [14] to provide this functionality. An instance of the digital twinning platform trusted by a device owner is executed in one or more nodes of a cloud provider. This instance obtains a digital certificate by a CA trusted by the device owner and uses that certificate to sign the responses sent to a consumer. Our goal is to enable consumers to verify the authenticity, provenance, and integrity of these responses in a secure and efficient way.

### A. Trust relationships

Our system considers the following trust relationships (see also Fig. 1). Consumers are pre-configured, using out-of-band mechanisms, with the $Owner_{URL}$ of each trusted device owner. We also assume that device owners know the $Instance_{ID}$ of the DT platform instances. An $Instance_{ID}$ can be verified and *attested* by a cloud provider. Examples of identifiers that can be used as an$Instance_{ID}$ are the digest of the platform's binary, a workload identifier (e.g., SPIFFE, Kurbenetes pod identifier), and others. An attestation

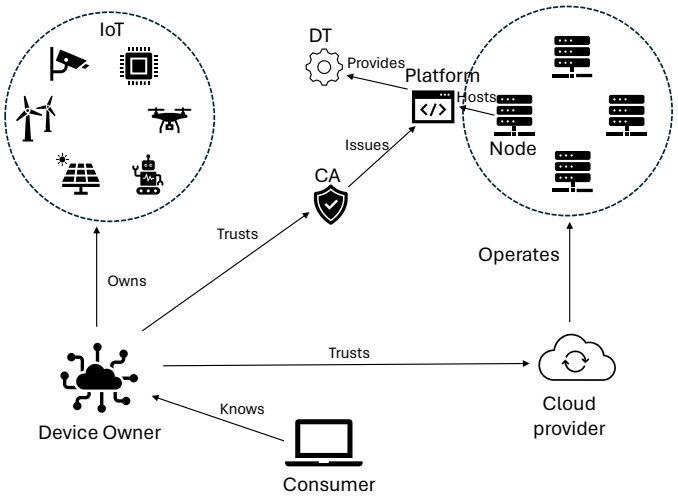


Fig. 1. System entities and their relationships.

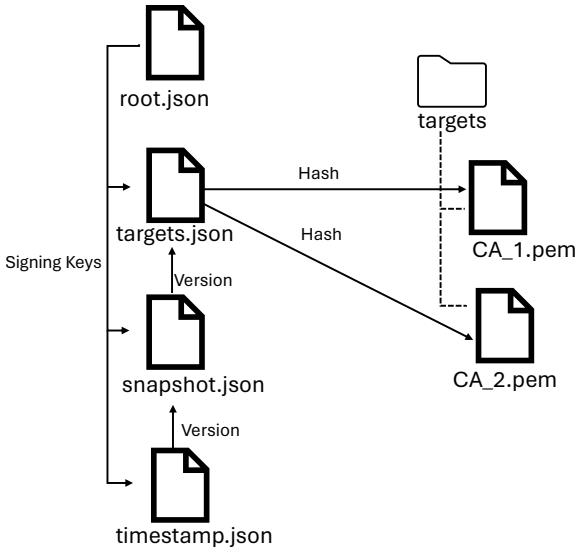


Fig. 2. TUF repository structure.

of an $Instance_{ID}$ is signed by a key controlled by the cloud provider and trusted by the device owner. This allows the device owner to verify that the DT platform instance hosted by the cloud provided is a trusted piece of software.

In the following, we define a mechanism that allows an instance of a DT platform to prove to a consumer that it is authorized to provide DT functionality for the IoT devices of a device owner. This authorization is proved through a digital certificate that binds $Owner_{URL}$ to a public key controlled by that platform instance; the certificate is issued by a CA trusted by the device owner. Our solution allows: a) a consumer to learn the CAs that a device owner trusts, and b) an instance of a platform executed in a cloud provider trusted by the device owner to obtain a digital certificate from a CA that is also trusted by the device owner.

### *B. Trusted CA management*

In our solution we are using *The Update Framework* (TUF) [15] enhanced with [16] to disseminate the public keys of the CAs trusted by a device owner. Note that TUF is a general purpose framework, thus our solution can be extended to allow dissemination of other trusted resources. In TUF, a device owner maintains a *repository* where the public keys of all trusted entities are stored: the files that include these keys are the *target* files. Moreover, this repository includes *metadata* files that protect its state. Using TUF, a consumer can securely access and verify target files, thus learning the public keys of the entities trusted by a device owner.

In TUF, a device owner defines four *roles*: *Root*, *Targets*, *Snapshot*, and *Timestamp*. Each role is associated with a number of keys: the keys of the Root role are securely transmitted using out-of-band mechanisms to consumers. Each role is responsible for the following files (see also Fig. 2):

- The Root role generates and signs the `root.json` metadata file. This file specifies the keys of each other role, as well as the number of keys required to sign each role's metadata file. This file can be versioned. New versions of `root.json` are named `VERSION_NUMBER.root.json` where `VERSION_NUMBER` is an integer that is always increased by one. A consumer downloads the latest version of `root.json` simply by increasing a counter `N` one by one and testing if the file `N.root.json` exists.
- The Target role generates and signs the `targets.json` metadata file. This file lists the hashes and sizes of target files. This file can be also be versioned. The current version of the `targets.json` file is specified in the `snapshot.json` file.
- The Snapshot role generates and signs the `snapshot.json` metadata file. This file lists the version numbers of the `root.json` and `targets.json` files. This file can also be versioned. The current version of `snapshot.json` is specified in the `timestamp.json` file.
- The Timestamp role generates and signs the `timestamp.json` metadata file. This file lists the hash, size, and version of the `snapshot.json` file. This file is *not* versioned. It is frequently re-signed, and has a short expiration date.

A consumer can securely learn the public keys of the CAs that a device owner trusts as follows. Initially, it retrieves the latest version of `root.json` and verifies it using the pre-configured keys of the root role. From there, it extracts and stores the public keys of the other roles. Next, it downloads `timestamp.json`, verifies it, and extracts the current version of `snapshot.json`. Following a similar approach, it downloads the latest versions of `snapshot.json` and then of `targets.json`. After verifying them, it downloads the target files and examines if their hashes match the values included in `targets.json`. Note that consumers that have already performed these steps once, can determine if the target files have been modified by simply downloading the latest `timestamp.json` file.

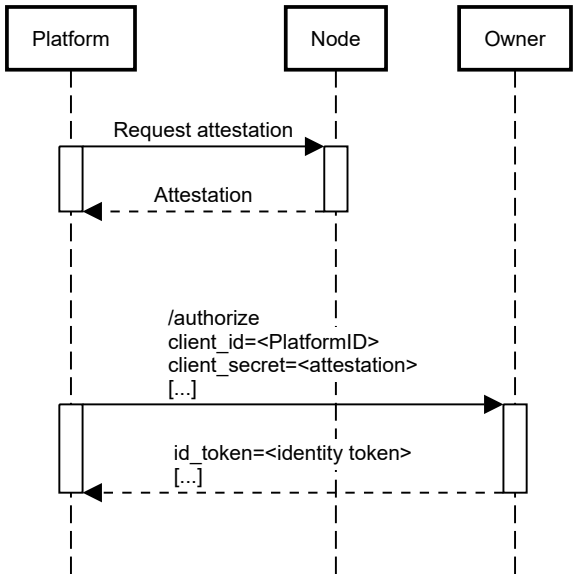


Fig. 3. Platform bootstrapping.

### C. Platform bootstrapping

The purpose of the platform bootstrapping process is to enable a DT platform instance to authenticate itself to a device owner and receive an *identity token*, thus allowing it to operate DTs on behalf of the device owner. As we discussed in the previous section, each device owner knows the $Instance_{ID}$ identifiers of all available platforms. An identity token proves that this specific DT platform instance is trusted by the device owner. Identity tokens are issued using OpenID Connect [17]. Accordingly, a device owner acts as an OpenID Connect *Provider* and implements all necessary HTTP endpoints required by the OpenID Connect Core and OpenID Connect Discovery [18] specifications. The bootstrapping process is implemented as follows.

Initially, a digital twinning platform instance requests from the node where it is hosted an *attestation* that acts as a proof of the platform's identifier ($Instance_{ID}$). This attestation is signed with a key controlled by the cloud provider and trusted by the device owner. Our design is oblivious to the attestation type and the attestation method generation, e.g., an attestation can be generated by the TPM of the node, a Virtual Machine provider, or a container framework. The only requirement of our solution is for the attestation to include a unique identifier that can be used to prevent attestation re-use. Then, the attestation is used as a *client secret* by the platform instance to receive an *identity token* from the device owner using OpenID connect. These steps are illustrated in Fig. 3.

In more detail, the DT platform instance sends the received attestation to the *authorization* endpoint of the device owner. The owner verifies that (a) the attestation includes a trusted $Instance_{ID}$, (b) it has not been re-used, and (c) it has been signed by a trusted cloud provider. Then, it generates the identity token which is an OpenID `id_token` (i.e., a signed JSON Web Token). The claim values that the identity token must include in our solution are given in Table I. The identity token is signed using a *JSON Web Key* (JWK):

TABLE I
CLAIM VALUES OF IDENTITY TOKENS GENERATED IN OUR SYSTEM.

| claim | value |
|---|---|
| `jti` | A unique identifier for preventing token re-use |
| `sub` | $Instance_{ID}$ |
| `iss` | $Owner_{URL}$ |
| `aud` | $CA_{URL}$ |

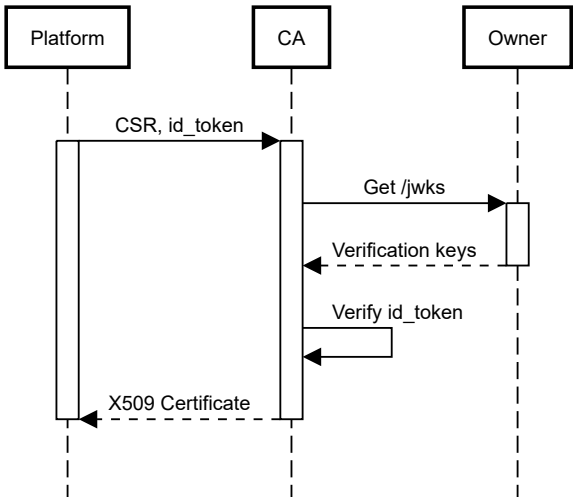


Fig. 4. Certificate issuance.

the corresponding public key can be retrieved from the *jwks* endpoint of the device owner [18]. In our solution, the key used to sign identity tokens is rotated frequently.

### D. Certificate issuance

The purpose of this process is to enable a DT platform instance to receive an X.509 certificate from a CA that binds a public key controlled by that instance to the $Owner_{URL}$ of a device owner. This certificate enables this platform instance to provide DTs that are securely (and verifiably) twinned to the devices of $Owner_{URL}$.

First, the DT platform instance has to obtain a fresh identity token using the previous protocol. Then, it generates a public-private key pair, which will be used for signing data. Finally, it creates a *Certificate Signing Request* (CSR) and sends it to the CA together with the identity token (see also Fig. 4). The CA then verifies that the identity token has not been re-used, includes the correct claim values (as specified in Table I), and has not expired. Then, the CA retrieves from the *jwks* endpoint of the device owner the appropriate verification key and validates the signature of the identity token (the process of locating the *jwks* endpoint is described in the OpenID Connect Discovery specifications [18]). If all the verifications are successful, the CA issues the requested X.509 certificate.

The issued certificate's *subject alternative name* (i.e., who the certificate is issued for) matches the `sub` claim from the identity token. Additionally, the certificate includes an X.509 v3 extension field that matches the provider of the identity token (i.e., the $Owner_{URL}$ included in the `iss` claim of the identity token). These certificates are short-lived in our solu-

tion, to enforce frequent key rotation; in our implementation, a certificate has a lifetime of only 10 min.

### E. Signing and verification

Our solution assumes a system where consumers request data related to an IoT device and these requests are handled by a platform instance that responds based on the status of the corresponding DT. Responses are signed using an ephemeral private-public key pair generated by the platform instance. In order for a signature to be verifiable by a consumer, the generated public key must be included in a valid X.509 certificate issued using the processes described in the previous section. To summarize, the platform instance must obtain a valid identity attestation from the hosting node, exchange it with an identity token from the device owner, and send this token together with a CSR to a trusted CA. Our solution is not bound to any specific digital signature mechanism; our implementation uses JSON Web Signatures, as well as signing schemes that enable selective data disclosure [19].

On the other hand, the only trust relationship that consumers establish is with the device owner. Therefore, upon receiving a signed response, they use their trust relationship with the owner as the "root of trust" and try to verify the digital signature included in the received item. In order to do that, they must first retrieve the public key(s) of the CA(s) trusted by the device owner, using TUF and the approach described in section III-B. Then, they examine if the certificate of the signer (which, in our implementation, is included in the signed item itself) has been issued by a trusted CA. Finally, if all checks are successful, they verify the signature using the latter certificate.

## IV. Implementation and evaluation

Our implementation includes a custom-made platform that implements ETSI's NGSI-LD API. Instances of the digital twinning platforms are executed in a docker container. Platforms are identified using SPIFFE[3] and attestations are generated using SPIRE[4]. We also implemented a custom-made OpenID provider used by device owners. Finally, as a CA we use a local instance of Fulcio[5], which is part of Sigstore [20].[6]

### A. Performance evaluation

Performance-wise, the goal of our system is to enable fast and automated certificate issuance to platform instances. In order to measure the time required to issue a certificate, we considered a scenario where the device owner's OpenID components are hosted by the same cloud provider as the digital twinning platform. On the other hand, we would expect the CA to be a public service located in another network. In order to approximate the time required to receive a certificate from the public CA, we measured the time needed to interact with the public instance of Fulcio.[7] Table II presents a breakdown of the time a DT platform instance spends on the various phases of receiving a certificate, obtained in a desktop PC using an Intel i5 processor and 8GB of RAM, running Ubuntu 22.04. We can see that a certificate is issued in less that 0.5 seconds, even assuming a remote public CA.

TABLE II
Time in ms required for issuing a certificate.

| Process | Time (ms) |
|---|---|
| Attestation issuance | 1 |
| Identity token issuance | 4 |
| Certificate issuance | 390 |

### B. Security evaluation

For the security evaluation of our system, we adapt the threat model of [20]. Specifically, we consider powerful adversaries that are capable of both compromising components in our system and performing man-in-the middle attacks. However, we assume that attackers cannot control the attestation mechanism of the cloud provider and cannot control all root keys used by the device owner to sign the root TUF metadata file. Finally, we assume that an attacker cannot have access to the certificate signing key of the trusted CA. We now discuss the impact of different types of compromises.

*1) Man-in-the-Middle attacks:* An attacker that intercepts the communication between a DT platform instance and a consumer, could replace the transmitted response with a tampered one. However, the consumer will detect this attack, since the signature will be invalid. On the other hand, an attacker that is able to intercept the communication between a DT platform instance and the device owner could use the identity token to request a certificate. In order to mitigate this attack, identity tokens should be bound to a key controlled by the platform instance. This can be achieved by leveraging the nonce-based mechanism used in [21] or by using the OAuth 2.0 DPoP (as specified in RFC 9449).

*2) Platform compromise:* An attacker can compromise a platform instance before execution or during runtime. If a platform is compromised before running, then (and depending on the platform identification attestation method used) it may result in receiving an invalid attestation that will be rejected during the OpenID connect flow with the device owner. On the other hand, a compromised running instance of the platform may be able to receive a valid certificate and create fake signed responses. In that case, responses generated by other instances of the same platform will not be affected.

*3) OpenID provider compromise:* An attacker that controls the OpenID provider of a device owner can issue valid identity tokens to its own (malicious) platforms. If consumers are convinced to use these platforms, then they will receive fake responses. By using solutions such as Certificate Transparency, a device owner can detect that certificates are issued on its behalf for platform identifiers it does not control.

*4) Malicious CA:* An attacker can create a malicious CA, which can be trusted in the Web PKI model, and issue certificates to its (malicious) platforms. However, in our system

[3]https://spiffe.io/

[4]https://spiffe.io/docs/latest/spire-about/

[5]https://github.com/sigstore/fulcio

[6]A proof-of-concept implementation of the entire system is available at https://github.com/mmlab-aueb/certificate-management

[7]https://fulcio.sigstore.dev

consumers are using the TUF metadata in order to learn the CAs that the device owner trusts. Furthermore, and due to the security properties of TUF (detailed in [16]) a device owner can recover even if all metadata files are compromised, as long as its root signing keys remain safe. In our implementation we are using 5 root signing keys which are kept offline.

## V. Conclusions

In this paper we presented a certificate management solution for cloud-hosted Digital Twins (DTs) used to represent IoT devices. Our solution allows DTs to obtain digital certificates in a secure, automated, and fast manner, thus allowing frequent key and certificate rotation. These certificates prove that the DTs are securely bound to IoT devices operated by a trusted device owner. Our solution removes the need for DTs to store secret information, therefore they can be safely hosted in public cloud systems. It also builds on existing protocols and cryptographic primitives, hence it is realistic and feasible.

Future work in this area includes the integration of transparency services (e.g., certificate transparency and software supply chain transparency logs). With these services, issued certificates and metadata about signed "artifacts" are immutably recorded in append-only auditable logs. This not only improves the auditability of the architecture but also enables early detection of key breaches (since attackers are forced to record information in the log). Such services also provide *timestamping* of signatures, enabling signature verification even after signing keys have been rotated. Timestamping is useful in scenarios where a digital signature has to be verified after some time (e.g., to verify the signature of an item retrieved from a cache).

## Acknowledgment

The work reported in this paper has been partly funded by the EU's Horizon 2020 Programme through the subgrant Secure Named Data Sharing (SNDS) (NGISARGASSO-2023-CALL1-9-SNDS) of project NGI SARGASSO (grant agreement No 101092887).

## References

[1] Y. Wang, Z. Su, S. Guo, M. Dai, T. H. Luan, and Y. Liu, "A survey on digital twins: Architecture, enabling technologies, security and privacy, and future prospects," *IEEE Internet of Things Journal*, vol. 10, no. 17, pp. 14 965–14 987, 2023.

[2] T. Miller, A. Staves, S. Maesschalck, M. Sturdee, and B. Green, "Looking back to look forward: Lessons learnt from cyber-attacks on industrial control systems," *International Journal of Critical Infrastructure Protection*, vol. 35, p. 100464, 2021.

[3] Y. Wu, K. Zhang, and Y. Zhang, "Digital twin networks: A survey," *IEEE Internet of Things Journal*, vol. 8, no. 18, pp. 13 789–13 804, 2021.

[4] C. Lai, M. Wang, and D. Zheng, "SPDT: Secure and privacy-preserving scheme for digital twin-based traffic control," in *Proceedings of the IEEE/CIC International Conference on Communications in China (ICCC)*, 2022, pp. 144–149.

[5] Y. Liu, L. Zhang, Y. Yang, L. Zhou, L. Ren, F. Wang, R. Liu, Z. Pang, and M. J. Deen, "A novel cloud-based framework for the elderly healthcare services using digital twin," *IEEE Access*, vol. 7, pp. 49 088–49 101, 2019.

[6] C. Patel, A. Pasikhani, P. Gope, and J. Clark, "User-empowered secure privacy-preserving authentication scheme for digital twin," *Computers & Security*, vol. 140, 2024.

[7] G. Thakur, P. Kumar, S. Jangirala, A. K. Das, Y. Park *et al.*, "An effective privacy-preserving blockchain-assisted security protocol for cloud-based digital twin environment," *IEEE Access*, vol. 11, pp. 26 877–26 892, 2023.

[8] C.-M. Chen, Q. Miao, S. Kumar, and T.-Y. Wu, "Privacy-preserving authentication scheme for digital twin-enabled autonomous vehicle environments," *Transactions on Emerging Telecommunications Technologies*, vol. 34, no. 11, p. e4751, 2023.

[9] J. Xu, C. He, and T. H. Luan, "Efficient authentication for vehicular digital twin communications," in *Proceedings of the IEEE Vehicular Technology Conference (VTC-Fall)*, 2021, pp. 1–5.

[10] F. Baldimtsi, K. K. Chalkias, Y. Ji, J. Lindstrøm, D. Maram, B. Riva, A. Roy, M. Sedaghat, and J. Wang, "zklogin: Privacy-preserving blockchain authentication with existing credentials," *arXiv preprint arXiv:2401.11735*, 2024.

[11] J. Primbs and M. Menth, "Oidc2: Open identity certification with openid connect," *IEEE Open Journal of the Communications Society*, vol. 5, pp. 1880–1898, 2024.

[12] K. Merrill, Z. Newman, S. Torres-Arias, and K. R. Sollins, "Speranza: Usable, privacy-friendly software signing," in *Proceedings of the ACM Conference on Computer and Communications Security (CCS)*, 2023, pp. 3388–3402.

[13] Z. Newman, "Reducing trust in automated certificate authorities via proofs-of-authentication," *arXiv preprint arXiv:2307.08201*, 2023.

[14] Context Information Management (CIM) ETSI ISG, "NGSI-LD API," ETSI, Group Specification CIM-009v161, 2022.

[15] J. Samuel, N. Mathewson, J. Cappos, and R. Dingledine, "Survivable key compromise in software update systems," in *Proceedings of the ACM Conference on Computer and Communications Security (CCS)*, 2010, p. 61–72.

[16] T. K. Kuppusamy, V. Diaz, and J. Cappos, "Mercury: bandwidth-effective prevention of rollback attacks against community repositories," in *Proceedings of the Usenix Annual Technical Conference*, 2017, p. 673–688.

[17] N. Sakimura, J. Bradley, M. Jones, B. de Medeiros, and C. Mortimore, "OpenID Connect Core 1.0 incorporating errata set 2," OpenID Connect WG, Specification, 2023, https://openid.net/specs/openid-connect-core-1_0.html.

[18] N. Sakimura, J. Bradley, M. Jones, and E. Jay, "OpenID Connect Discovery 1.0 incorporating errata set 2," OpenID Connect WG, Specification, 2023, https://openid.net/specs/openid-connect-discovery-1_0.html.

[19] N. Fotiou, Y. Thomas, and G. Xylomenos, "Data integrity protection for data spaces," in *Proceedings of the ACM EUROSEC Workshop*, Athens, Greece, 2024.

[20] Z. Newman, J. S. Meyers, and S. Torres-Arias, "Sigstore: Software signing for everybody," in *Proceedings of the ACM Conference on Computer and Communications Security (CCS)*, 2022, p. 2353–2367.

[21] E. Heilman, L. Mugnier, A. Filippidis, S. Goldberg, S. Lipman, Y. Marcus, M. Milano, S. Premkumar, and C. Unrein, "Openpubkey: Augmenting openid connect with user held signing keys," *Cryptology ePrint Archive*, 2023.