論文 / 著書情報
Article / Book Information

Title	DIDAuth-IoTFW: Decentralized firmware authentication for smart home IoT devices using verifiable credentials
Authors	W.M.A.B Wijesundara, Joong-sun Lee, Eleni Aloupogianni, Dara Tith, Hiroyuki Suzuki, Takashi Obi
Citation	Internet of Things, Vol. 34, ,
Pub. date	2025, 11
DOI	https://dx.doi.org/10.1016/j.iot.2025.101788
Creative Commons	Information is in the article.

Internet of Things 34 (2025) 101788

Contents lists available at ScienceDirect

Internet of Things

journal homepage: www.elsevier.com/locate/iot

Research article

DIDAuth-IoTFW: Decentralized firmware authentication for smart home IoT devices using verifiable credentials

W.M.A.B. Wijesundara [a,*], Joong-Sun Lee [b,c], Eleni Aloupogianni [a], Dara Tith [a], Hiroyuki Suzuki [d], Takashi Obi [b]

[a] Department of Information and Communications Engineering, Institute of Science Tokyo, Tokyo, Japan
[b] Institute of Innovative Research, Institute of Science Tokyo, Yokohama, Japan
[c] Catholic Kwandong University, Gangwon-do, Republic of Korea
[d] Center for Mathematics and Data Science, Gunma University, Gunma, Japan

ARTICLE INFO

Dataset link: https://github.com/awijesundara/DIDAuth-IoTFW

Keywords:
Decentralized identity
Distributed ledger technologies
Ethereum Layer-2
Arbitrum
Verifiable credentials
Information security
Communication systems
IPFS

ABSTRACT

Rapid proliferation of smart home IoT devices has intensified the demand for secure, scalable, and autonomous firmware authentication mechanisms. Traditional centralized solutions face challenges related to privacy concerns, limited scalability, and vulnerability to single point of failure. In this paper, we propose DIDAuth-IoTFW, a novel decentralized identity and firmware authentication framework that uniquely integrates Ethereum Layer-2 Arbitrum, InterPlanetary File System (IPFS), and W3C-compliant Decentralized Identifiers (DIDs) and Verifiable Credentials (VCs). DIDAuth-IoTFW provides a complete firmware authentication life cycle, from decentralized identity registration to real-time, on-chain verifiable revocation. While enabling autonomous, cryptographic verification directly on resource-constrained IoT devices and ensuring reliable performance even when gateways are compromised or unavailable. Our proof-of-concept implementation on ESP32 and Raspberry Pi achieved complete resistance to replay, forgery, and revocation threats with verification consistently under 1.2 s. Compared to prior work, DIDAuth-IoTFW uniquely combines firmware–VC hash binding, contract binding that prevents cross-registry replay, and device-side enforcement resilient to gateway compromise. Experimental results indicate a robust, privacy-preserving, and scalable alternative to centralized firmware-update pipelines for smart-home IoT.

1. Introduction

Internet of Things (IoT) is being widely applied in diverse fields including agriculture, industry, power and water networks, transportation, smart home, and many others [1]. Among these, introduction of IoT devices in smart homes has resulted in higher levels of convenience and automation [2–4]. Against this backdrop, privacy protection in IoT systems has attracted interest. Privacy and security are two of the most challenging aspects of IoT, as they are closely interwoven with people's personal habits and data [5–7]. Although researchers are paying attention to privacy protection applicable to IoT, cyberphysical systems, and relevant legal frameworks [8], comprehensive research focusing on IoT privacy protection is still limited [9].

Ensuring that devices operate on authentic and uncompromised firmware is fundamental to maintaining the trust and security of smart home environments. However, this remains underexplored. One critical measure to avoid recurring security attacks on IoT

* Correspondence to: Department of Information and Communications Engineering, School of Engineering, Institute of Science Tokyo, Tokyo, Japan.
E-mail addresses: wijesundara.w.aa@m.titech.ac.jp, wijesundara.w.d79f@m.isct.ac.jp (W.M.A.B. Wijesundara), j-lee@asist.iir.isct.ac.jp (J.-S. Lee), ealoupogianni@outlook.com (E. Aloupogianni), darapich.tith@gmail.com (D. Tith), hiroyuki.suzuki@gunma-u.ac.jp (H. Suzuki), obi@isl.titech.ac.jp (T. Obi).

https://doi.org/10.1016/j.iot.2025.101788
Received 19 July 2025; Received in revised form 14 September 2025; Accepted 2 October 2025
Available online 8 October 2025

devices is firmware updates. With the risk of conventional centralized architecture being vulnerable to failures and scalability issues due to the prevalence of large numbers of IoT devices, distributed architecture is seen as a promising alternative [10]. Researchers have turned towards decentralized models that employ collaborative networks for distribution and verification, to overcome the limitations of centralized firmware delivery systems. This shift allows integrating robust distributed technologies into the firmware authentication process [10–12].

Distributed technologies provide enhanced trust and transparency. As opposed to centralized systems, they operate on a network of nodes independent of each other. Each of the nodes contributes to data validation, storage, or transmission. Applications include Ethereum for decentralized computation [13], BitTorrent for file sharing [14], and InterPlanetary File System (IPFS) for content-addressable storage [12], while binaries can also be transmitted in a decentralized manner [11,15,16]. In this context, decentralized identity management has emerged as a promising concept, aligning with the principles of Self-Sovereign Identity (SSI). SSI empowers individuals and devices with control over their own identities, reducing reliance on centralized authorities and enhancing privacy. Distributed Ledger Technologies (DLTs), such as blockchain, have been explored to support decentralized identity frameworks due to their inherent characteristics of immutability, transparency and distributed consensus [17–20]. However, industrial and financial application of blockchain is limited by its low throughput, limited scalability, resource requirements, storage limitations and substantial costs.

In the context of secure binary distribution for IoT, since no single authority redistributes binaries, adversarial attacks that share malicious versions are possible. Therefore, it is necessary to ensure the integrity of the distributed binaries, a non-trivial task [21]. This can be alleviated by schemes that provide a decentralized digital identity management scheme, which ensures the authenticity of smart home IoT firmware. Blockchain technology can be supplemented with Decentralized Identifiers (DIDs) and Verifiable Credentials (VCs) to form a self-sovereign identity framework [22,23]. Importantly, an effective decentralized authentication framework must accommodate diverse devices and follow prevailing standards [24]. The blockchain is able to increase the efficiency of the authentication process and assist the application of sophisticated cryptographic techniques, which in turn improves security, while ensuring privacy [8,25]. Moreover, the integration of secure cryptographic algorithms and innovative identity management protocols, promises both agility and resilience across diverse smart home IoT devices [7,26].

As the global number of connected devices continues to surge, with approximately 17.4 billion short-range IoT devices (such as smart home assistants and wearable fitness trackers) worldwide as of February 2025, [27] implications for implementing a decentralized identity management system become increasingly significant. The capacity for rapid, low-cost transactions becomes essential for effective identity management. These systems not only streamline the authentication processes, but also enhance the overall privacy of users, ensuring that their personal data remains within their control. This self-sovereign identity model addresses prevailing privacy concerns, allowing users to authenticate their devices seamlessly without exposing sensitive information to third parties [28]. An example is Arbitrum, a Layer-2 scaling solution for Ethereum that enhances transaction throughput and reduces costs using optimistic rollups, helping address scalability challenges in decentralized identity management for IoT ecosystems.[29]. However, existing Layer-2 solutions like Arbitrum do not provide built-in support for firmware-specific identity verification and also lack built-in mechanisms for cross vendor interoperability and self-sovereign credential management. Such limitations emphasize the requirement for a customized framework that brings together decentralized identity, verifiable firmware authentication and scalable infrastructure designed for smart home IoT environments.

Therefore, in this paper, we present a novel decentralized architecture for identity and firmware authentication in IoT ecosystems, with the objective of enhancing trust, security and resilience in firmware distribution processes. To address the gaps in Layer-2 solutions, our framework uses Arbitrum in conjunction with IPFS for decentralized storage, introducing a lightweight solution that enables manufacturers and devices to maintain full autonomy over identity and credential life cycle management. By anchoring device identities and verifiable firmware credentials on Arbitrum and offloading bulk metadata to IPFS, our system ensures cost efficiency, avoids vendor lock-in and enables verifiable trust without centralized intermediaries. Furthermore, the architecture adheres to World Wide Web Consortium (W3C) standards for DIDs and VCs, promoting interoperability, modular integration, and long-term adaptability across heterogeneous smart home IoT environments.

1.1. Contributions

While prior research has explored the use of DIDs and VCs in IoT device registration and firmware integrity verification [30–32], these solutions often lack a fully integrated and enforceable life cycle for firmware authentication. In contrast, DIDAuth-IoTFW contributes the following:

- **End-to-end Firmware Credential life cycle:** We implemented an end-to-end firmware authentication pipeline, encompassing DID registration, VC issuance, IPFS-based credential publishing, gateway verification, device level enforcement, and smart contract-based revocation. This unified approach ensures secure management across the entire firmware life cycle
- **Blockchain-integrated DID and VC Management:** DIDAuth-IoTFW explicitly leverages Ethereum Layer-2 (Arbitrum) smart contracts and IPFS decentralized storage to securely anchor device identities and firmware credentials. This integration provides a cost efficient, scalable, and tamper resistant identity life cycle management.
- **On-chain Verifiable Revocation:** Unlike Gnomon [30] or Pescetelli et al. [31], our framework employs a dedicated smart contract to record and revoke VCs. This enables any verifier to check revocation status via an auditable and fully decentralized source in real-time.

- **Device-side Verification and Enforcement:** We demonstrated the implementation of VC and firmware validation directly on resource-constrained devices. This ensures robust endpoint level trust enforcement, even in the presence of compromised gateways, an aspect not fully supported by prior systems.
- **Hash Binding for Firmware Verification:** Each issued VC cryptographically embeds a SHA-256 hash of the firmware binary. This mechanism enables devices to validate integrity before execution, providing a strong defense against unauthorized modifications and tampering.
- **Security-driven Evaluation:** We conducted empirical validation of threat mitigation strategies, simulating adversarial scenarios such as tampering, replay attacks, and credential revocation. Our evaluation rigorously measured authentication latency and system resilience, providing comprehensive insights into the framework's practical security performance.
- **Formalization and proofs:** We state explicit security goals (G1–G4) and give provable guarantees (Claims 1–3 and Lemma 1 in Section 3.4) under standard assumptions.

This comprehensive integration at the architectural level yields a single, enforceable trust chain that enables trustworthy, scalable, and privacy-preserving firmware authentication for smart-home IoT. The remainder of the paper is organized as follows: Section 2 reviews related work. Section 3 presents the architecture. Section 3.3 formalizes the model and goals; Section 3.4 states our guarantees. Section 5 reports evaluation. Section 6 concludes.

2. Related works

As the requirement for secure digital identity management in IoT is being increasingly emphasized, related research explored blockchain as a viable solution to reduce security vulnerabilities and enhance authentication protocols. In an extensive effort regarding this, Mukhandi et al. (2022) [33] tried to recognize important technical and design-related difficulties that could be faced in establishing ideal identity management solutions, by studying blockchain and IoT technologies. However, their work did not address firmware authentication for resource-constrained IoT devices. Ullah et al. (2024) [34] introduced a method to securely store data in a decentralized manner, resulting in a lower risk for data breaches. Their model incorporates the Ethereum blockchain with attribute-based encryption. Their model demonstrates secure data storage and access control but does not focus on firmware authentication. In another related work [35], deep learning based on blockchain was applied to improve the security of data transmission in an IoT healthcare system. While this approach is adaptable to smart home IoT, it does not comprehensively address decentralized firmware authentication.

In the context of smart homes, Sharma et al. (2025) [36] was able to further demonstrate blockchain's strong abilities of secure data management. This work highlights the data management ability of blockchain for smart homes but relies on centralized gateways, limiting full decentralization. As an alternative, studies have presented SSI as a promising method for users to have control over their data, secure decentralized interactions among devices, and reduce vulnerability to unauthorized access [37,38]. However, existing SSI implementations often lack integration with firmware authentication in smart home IoT [39]. In our team's previous work [21,40], traditional and decentralized methods were used together to ensure secure firmware authentication in smart homes, with an advantage of the latter with respect to their applicability for IoT. Qashlan et al. (2020) [41] also supported the relevance of VCs within decentralized authentication frameworks. Their authentication method reduces the dependency on centralized architecture, but does not address integrity of updating firmware.

In terms of security, Ali et al. (2025) [42] emphasized the impact of blockchain on identity and access management (IAM). They introduced an application of Vision Transformer (ViT) architectures to identify Distributed Denial of Service (DDoS) and Denial of Service (DoS) attacks during over-the-air (OTA) firmware updates. They were able to achieve a detection accuracy of 99.50%, along with optimum use of resources which is suitable for IoT applications. However, their framework does not provide a decentralized firmware authentication mechanism. Another firmware update system [43] provided users with proof-of-delivery and a method of installation verified via blockchain. Despite being effective, the model experienced resource intensiveness and reliance on trusted gateways for resource-constrained IoT devices.

Complementary to identity and firmware authenticity, the security of IoT data flows in cloud fog settings has also been previously strengthened using machine learning. Aknan et al. (2025) [44] proposed a secure cloud-assisted fog computing framework in which ensemble learning executed on fog nodes to detect anomalies and malicious activity while maintaining low latency for edge workloads. This line of work is complementary to our current work: it enhances runtime threat detection and data-path security within fog/edge orchestration. However, their study does not address decentralized firmware provenance, DID/VC life cycle management, or on-chain revocation. In contrast, DIDAuth-IoTFW focuses on cryptographic authorization of firmware issuer authenticity, hash binding, and real-time revocation. These two lines of work can be viewed as complementary, but serve distinct roles in a secure IoT stack.

In summary, while previous studies have managed to significantly improve security of IoT applications through blockchain-based technologies and decentralized identity management, noteworthy gaps still exist. The comprehensive integration of DIDs, VCs, immutable blockchain storage and decentralized storage systems customized to smart home IoT firmware authentication remains incomplete [45]. While prior works address individual aspects—such as integrity validation, decentralized distribution, or DID-based identity, the comprehensive integration of VC life cycle, on-chain revocation, IPFS-based identity anchoring, and lightweight device-side enforcement uniquely distinguishes DIDAuth-IoTFW.

2.1. Decentralized firmware authentication and DID/VC approaches

Recent studies have explored decentralized strategies for secure firmware update delivery in IoT environments. Choi and Lee [15] proposed a blockchain-based architecture that anchors firmware manifests using IETF SUIT (Internet Engineering Task Force Software Updates for Internet of Things specification) metadata, enhancing integrity verification. Kaptan et al. [46] introduced a model combining IPFS for firmware hosting with on-chain hash anchoring, offering a decentralized alternative to vendor-controlled update servers. Similarly, Oktian et al. [10] presented "Patchman", a framework that supports OTA firmware distribution via smart contracts, incorporating vendor bidding and reward incentives with blockchain-based delivery proofs.

While these approaches ensure firmware authenticity and transparency, they do not incorporate decentralized identity mechanisms such as DIDs or VCs. As a result, identity binding, delegation of trust, and fine-grained revocation remain insufficiently addressed.

Some studies have begun integrating DID and VC concepts into firmware workflows. Gnomon [30] employed DIDs and VCs for 5G IoT device registration and update authorization. However, its reliance on external identity hubs and off-chain verification makes it unsuitable for fully decentralized and resource-constrained environments. Evoke [32] proposed a scalable VC revocation method for IoT but did not integrate firmware content binding, nor did it offer end-to-end enforcement across gateways and devices.

In contrast, DIDAuth-IoTFW introduces a complete decentralized firmware authentication life cycle, tailored for smart home IoT. Its key contributions include: (1) issuance of VCs bound to firmware via embedded SHA-256 hashes; (2) anchoring of DID Documents on IPFS with content-addressable integrity; (3) smart contract-based on-chain VC registration and revocation; and (4) layered VC verification at both gateway and device levels, including on low-resource platforms like the ESP32. These integrated capabilities provide tamper resistance, identity assurance, and real-time revocation, addressing critical limitations in prior models and enabling trustworthy, decentralized firmware authentication at scale.

3. Methodology

3.1. Fundamentals

This section introduces core technologies underpinning the proposed decentralized firmware authentication framework: W3C standard DIDs and VCs 1.0 with Verifiable Presentations (VPs). It also covers IPFS, Ethereum smart contracts, and cryptographic primitives. These components collectively facilitate secure, independent identity management and trusted firmware validation in smart home IoT environments.

3.1.1. Decentralized identifiers (DIDs)

DIDs are a new type of global identifier that enables entities to create and control their own digital identities without depending on centralized registries or trusted intermediaries. Each DID is bound to a cryptographic key pair and resolves to a DID document that contains relevant metadata.

- **Format:** A DID typically follows the structure did:method:unique_identifier.
- **DID Document:** A JSON-based metadata file that includes:
  - @context: Context for semantic interpretation
  - id: The DID string itself
  - verification method: Public keys and signature mechanisms
  - authentication: Methods used to authenticate the DID controller
  - service: Optional endpoints associated with the identity
- **Ownership and Control:** The controller of a DID, defined as the entity possessing the associated private key, has full control over identity operations such as signing credentials or updating DID documents.

In the proposed framework, each firmware vendor generates a DID and publishes the corresponding DID document to IPFS. The IPFS content identifier (CID) is registered on-chain using an Ethereum smart contract deployed to the Arbitrum Sepolia testnet, allowing verifiers to resolve and authenticate the vendor's identity.

3.1.2. Verifiable Credentials (VCs)

VCs are cryptographically signed statements issued by a DID controller. They enable trusted and tamper proof sharing of claims between an issuer, a holder, and a verifier. The structure includes:

- @context: Defines the semantic structure of the VC
- id: Unique identifier for the credential
- type: Includes "VerifiableCredential" and an application-specific type such as "FirmwareCredential"
- issuer: The DID of the issuing entity
- issuanceDate: Timestamp of issuance
- credentialSubject: Contains claims about the subject (e.g., firmware version, device model, firmware hash)
- proof: Digital signature using Ed25519 to ensure authenticity and integrity

Firmware vendors issue VCs that contain the firmware version, device model, and the SHA-256 hash of the firmware binary. These credentials are signed using the vendor's private key, stored on IPFS, and later retrieved by gateways or devices to validate firmware authenticity before installation.

Unified definitions for VC and VP mathematical formulations

Please refer to the definitions listed below, which apply to the mathematical formulations presented here onward.

- $\mathcal{VC}$: VC issued by the vendor DID, containing firmware-related claims and a cryptographic proof.
- $\mathcal{VP}$: VP created and signed by the IoT device DID, encapsulating one or more VCs.
- $\texttt{did}_{\text{issuer}}$: DID of the credential issuer (vendor).
- $\texttt{did}_{\text{holder}}$: DID of the credential holder (IoT device).
- F: Firmware binary received by the IoT device.
- $\texttt{sig}_{\mathcal{VC}}$: Ed25519 digital signature on the VC body.
- $\texttt{sig}_{\mathcal{VP}}$: Ed25519 digital signature on the VP body.
- $\mathcal{VC}_{\text{body}}$: VC body excluding the proof field.
- $\mathcal{VP}_{\text{body}}$: VP body excluding the proof field.
- $\texttt{pk}_{\text{issuer}}$: Public key of the issuer, resolved via DID.
- $\texttt{pk}_{\text{holder}}$: Public key of the holder, resolved via DID.
- h_{local}: SHA-256 hash of the firmware binary received F, computed locally.
- h_{vc}: Firmware hash claimed in $\mathcal{VC}_{\text{body}}$.
- SHA256$(\cdot)$: Standard SHA-256 cryptographic hash function.
- VerifySig$(\cdot)$: Digital signature verification function. Returns `True` if valid.
- $H(\cdot)$: SHA-256 hash of the canonical JSON representation of the VC.
- isVCRevoked$(\cdot)$: Returns `True` if the VC hash is revoked on-chain.
- $\mathcal{VC}_{\text{contract}}$: Smart contract address embedded in the VC metadata.
- C_{expected}: Expected smart contract address known to the verifier.
- `Firmware_Verified`: Boolean outcome indicating successful verification of firmware integrity and authorization.

Mathematical formulation of VC verification and firmware integrity check

The verification process consists of the following steps:

1. **Signature Verification of VC:**

$$\text{VerifySig}\left(\texttt{sig}_{\mathcal{VC}}, \mathcal{VC}_{\text{body}}, \texttt{pk}_{\text{issuer}}\right) = \text{True}$$

2. **Firmware Hash Verification:**

$$h_{\text{local}} = \text{SHA256}(F), \quad h_{\text{vc}} = \mathcal{VC}_{\text{body}}[\texttt{firmwareHash}]$$

$$\text{Pass} \iff h_{\text{local}} = h_{\text{vc}}$$

3. **Revocation Status Check:**

$$\text{isVCRevoked}\left(H(\mathcal{VC})\right) = \text{False}$$

4. **Smart Contract Binding Verification:**

$$\mathcal{VC}_{\text{contract}} = C_{\text{expected}}$$

The firmware is accepted only if all checks pass:

$$\textbf{Firmware\_Verified} \Leftrightarrow \begin{cases} \text{VerifySig}\left(\texttt{sig}_{\mathcal{VC}}, \mathcal{VC}_{\text{body}}, \texttt{pk}_{\text{issuer}}\right) = \text{True} \\ h_{\text{local}} = \text{SHA256}(F), \quad h_{\text{vc}} = \mathcal{VC}_{\text{body}}[\texttt{firmwareHash}] \\ \neg\, \text{isVCRevoked}(H(\mathcal{VC})) \\ \mathcal{VC}[\text{contract}] = C_{\text{expected}} \end{cases}$$

3.1.3. Verifiable Presentations (VPs)

VPs enable a holder to securely present one or more VCs to a verifier while preserving privacy. A VP functions as a cryptographically signed container that proves that the holder has valid credentials without necessarily revealing all underlying details. The structure includes:

- @context: Inherits from the embedded VCs

- type: Includes "VerifiablePresentation"
- verifiableCredential: One or more VCs to be presented
- holder: The DID of the entity presenting the VCs
- proof: A digital signature over the presentation contents

Challenge-Based Verification:

VPs are used by smart devices or gateways to present firmware credentials securely. For example, when a device receives a firmware update, it can present the firmware VC in a VP to the IoT gateway, proving both the integrity of the firmware and the legitimacy of the vendor who issued the VC.

Mathematical formulation of VP and VC verification

The validation process at the verifier consists of:

1. **VP Signature Verification:**

$$\text{VerifySig}\left(\texttt{sig}_{\mathcal{VP}}, \mathcal{VP}_{\text{body}}, \texttt{pk}_{\text{holder}}\right) = \text{True}$$

2. **VC Extraction from VP:**

$$\mathcal{VC} \in \mathcal{VP}_{\text{body}}[\texttt{verifiableCredential}]$$

3. **VC Signature Verification:**

$$\text{VerifySig}\left(\texttt{sig}_{\mathcal{VC}}, \mathcal{VC}_{\text{body}}, \texttt{pk}_{\text{issuer}}\right) = \text{True}$$

4. **Firmware Hash Check:**

$$h_{\text{local}} = \text{SHA256}(F), \quad h_{\text{vc}} = \mathcal{VC}_{\text{body}}[\texttt{firmwareHash}]$$

$$\text{Pass} \iff h_{\text{local}} = h_{\text{vc}}$$

5. **Credential Revocation Check:**

$$\text{isVCRevoked}(H(\mathcal{VC})) = \text{False}$$

6. **Smart Contract Binding Check:**

$$\mathcal{VC}_{\text{contract}} = C_{\text{expected}}$$

The presentation is accepted and the firmware is validated only if all checks pass:

$$\textbf{Firmware\_Verified} \Leftrightarrow \begin{cases} \text{VerifySig}_{\mathcal{VP}} = \text{True} \\ \text{VerifySig}_{\mathcal{VC}} = \text{True} \\ h_{\text{local}} = \text{SHA256}(F), \quad h_{\text{vc}} = \mathcal{VC}_{\text{body}}[\texttt{firmwareHash}] \\ \neg\, \text{isVCRevoked}(H(\mathcal{VC})) \\ \mathcal{VC}[\text{contract}] = C_{\text{expected}} \end{cases}$$

3.1.4. InterPlanetary File System (IPFS)

IPFS is a decentralized peer-to-peer file system that uses content-based addressing to store and retrieve files reliably and securely. Its key features are as follows:

- Content Addressing: Files are addressed by a hash of their content (CID)
- Immutability: Any modification results in a new CID, preventing tampering
- Decentralization: Files can be hosted and retrieved from any IPFS node globally

In the proposed framework, IPFS is used to store vendor DID documents (public keys, authentication methods), firmware binaries, and VCs for firmware updates. The corresponding CIDs are registered in Ethereum smart contracts, which enables the integrity and verifiability of the content without relying on centralized servers.

3.1.5. Ethereum smart contracts (Arbitrum)

An Ethereum smart contract deployed on Arbitrum Sepolia operates as a decentralized execution environment that is resistant to tampering. This framework employs a custom contract implemented in Solidity and deployed to Arbitrum Sepolia, using Ethereum Virtual Machine (EVM)-compatible logic for DID mapping and revocation tracking. The contract maintains an auditable registry of DIDs and associated credential metadata.

- registerDID(string did, string cid): Binds a DID to a DID Document stored in IPFS.

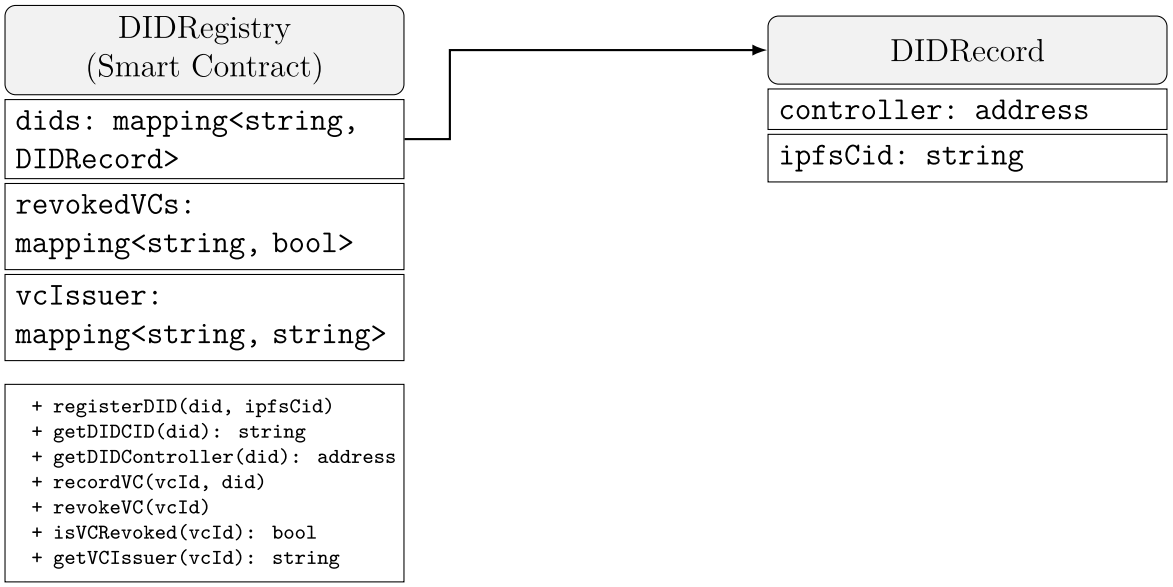

Fig. 1. UML-style diagram of `DIDRegistry.sol` contract storage layout, showing mappings and `DIDRecord` structure.

- getDIDCID(string did): Resolves the latest CID for a given DID.
- recordVC(bytes32 vcId, string did): Record issuance of a VC by a DID.
- revokeVC(string vcHash): Marks a specific VC hash as revoked.
- isVCRevoked(bytes32 vcId) → bool: Query revocation status.
- getDIDController(string did): Returns the Ethereum address that originally registered the DID. This helps verify DID ownership and enforce permission checks.
- getVCIssuer(string vcId): Returns the DID that issued a specific VC. Verifiers use this to validate that a VC is recorded and linked to an authorized issuer.

These helper functions are particularly useful during verification. `getDIDController` ensures that only the original controller (registrant) of a DID can revoke credentials, while `getVCIssuer` enables on-chain validation that a VC was indeed issued by a known and registered DID, reducing the risk of forged or off-chain credentials being accepted as valid. Vendors use this contract to register their DID documents by submitting the corresponding IPFS CID. When a VC is issued, its identifier and associated issuer DID are recorded on-chain via `recordVC`, providing immutable proof of issuance. Similarly, credentials can be revoked by their issuer through `revokeVC`. This on-chain tracking allows smart home gateways and verifiers to check both authenticity and revocation status without relying on centralized revocation lists.

Fig. 1 illustrates the internal storage layout of the `DIDRegistry` smart contract, showing the relationships between DID mappings, credential revocation tracking, and the `DIDRecord` structure.

3.1.6. Cryptographic primitives

The proposed system relies on modern cryptographic algorithms to ensure secure identity binding, data integrity, and authentication. Ed25519 is used for signing DIDs, VCs, and VPs, while SHA-256 is used to generate hashes of firmware binaries. These cryptographic tools enable a trustless mechanism for verifying firmware authenticity without the involvement of centralized authorities.

3.2. Threat model and assumptions

We adopt a Dolev-Yao style adversary model [47], assuming the adversary possesses full control over the communication network. This allows the adversary to intercept, modify, inject, and replay messages between IoT devices, gateways, and the backend infrastructure. However, we assume that the underlying cryptographic primitives (Ed25519 signatures, SHA-256 hashing) are cryptographically secure, and that both the blockchain (Ethereum Layer-2 Arbitrum) and IPFS layers function as intended without compromise.

System Assumptions:

- **Trusted Firmware Vendor:** The Firmware Vendor is a trusted authority responsible for issuing valid credentials and uploading the correct firmware.

- **Secure Device Provisioning:** IoT devices are securely provisioned with their public keys and trusted root information during manufacturing or initial setup.
- **Immutable Smart Contract:** The smart contract on Arbitrum is immutable and deployed by the trusted vendor.

Adversarial Capabilities:

- **A1. Credential forgery:** The adversary attempts to create or modify VCs by forging digital signatures or manipulating metadata.
- **A2. Replay attacks:** The adversary reuses previously valid credentials (e.g., revoked or expired VCs) to authorize outdated or malicious firmware.
- **A3. Gateway compromise:** The adversary compromises the IoT gateway and attempts to bypass verification mechanisms to install unauthorized firmware.
- **A4. Tampering During Transit:** Firmware binaries or VCs are altered during OTA delivery, either between the gateway and the device or during VC upload to IPFS.

Defense Strategies in DIDAuth-IoTFW:

- **Ed25519 Cryptographic Signatures:** All DIDs, VCs, and VPs are signed using Ed25519, ensuring authenticity and integrity to prevent forgery (A1).
- **Firmware Hash Binding:** The SHA-256 hash of each firmware binary is embedded within its corresponding in the VC, enabling cryptographic validation on the device to detect tampering (A4).
- **On-chain Smart Contract Revocation:** Verifiers and devices query the Arbitrum smart contract to check VC the real-time revocation status, effectively preventing replay (A2).
- **Device-side Enforcement:** Even if the gateways are compromised, the devices independently verify VC integrity and firmware hash before installation, ensuring robust endpoint-level trust (A3).

3.3. Security goals and model

We target the following goals under the Dolev–Yao network model given in Section 3.2, assuming EUF-CMA (Existential Unforgeability under Chosen-Message Attack) security of Ed25519 and collision/preimage resistance of SHA256.

- **G1 Authenticity:** Only issuer-signed VCs are accepted.
- **G2 Integrity:** Accepted firmware bytes equal the bytes bound in the VC.
- **G3 Replay safety:** Revoked VCs are never accepted after chain finality.
- **G4 Gateway-independence:** Acceptance depends only on device-side checks.

3.4. Formal guarantees

We now formalize the guarantees corresponding to goals G1–G4 under the adversarial model defined in Section 3.3. All security notions and symbols are summarized in Table 1 for reference. Throughout, adversaries are assumed to be PPT (Probabilistic Polynomial-Time) algorithms, i.e., computationally bounded but probabilistically adaptive.

Claim 1 (*Authenticity and Integrity (G1–G2)*). *Let h_{vc} be the firmware hash embedded in $\mathcal{VC}$ and let $F^{\star}$ be any string such that* $\mathsf{SHA256}(F^{\star}) = h_{\mathrm{vc}}$. *For any PPT adversary $\mathcal{A}$,*

$$\Pr\left[\mathsf{Accept}(F, \mathcal{VC}) \ \wedge \ (F \neq F^{\star})\right] \ \leq \ \mathrm{Adv}^{\text{euf-cma}}_{\text{Ed25519}} \ + \ \mathrm{Adv}^{\text{cr}}_{\mathsf{SHA256}}.$$

Proof sketch. If Accept holds with $F \neq F^{\star}$, then either (i) the adversary altered $\mathcal{VC}$ without the issuer's key (signature forgery), or (ii) $\mathsf{SHA256}(F) = \mathsf{SHA256}(F^{\star})$ with $F \neq F^{\star}$ (collision).

Interpretation. A device can only be tricked into accepting tampered firmware if the adversary forges the vendor's Ed25519 signature or finds a SHA collision. Both are widely considered infeasible.

Claim 2 (*Replay Safety (G3)*). *If verification queries* $\mathsf{isVCRevoked}(H(\mathcal{VC}))$ *against a finalized block b, then any VC revoked at or before b is rejected.*

Proof sketch. The contract's `revokedVCs` mapping is monotone. Finality prevents reorgs that would remove revocations at or before b.

Interpretation. Once a credential is revoked and included in a finalized block, it cannot be replayed. Blockchain finality guarantees that revocations are irreversible.

Claim 3 (*CrosS-Registry Replay Resistance (G1–G2)*). *Let $C_{expected}$ be the verifier's contract and let $\mathcal{VC}$ carry field* `contract`. *For any PPT adversary, the probability that a VC accepted by a verifier bound to $C' \neq C_{expected}$ is also accepted by the verifier bound to $C_{expected}$ is at most the probability of forging either the issuer signature or a valid on-chain issuance/revocation state under $C_{expected}$.*

Table 1
Glossary of security notation and variables used in formal guarantees.

Symbol/Term	Meaning
PPT adversary $\mathcal{A}$	Probabilistic Polynomial-Time adversary: an efficient attacker model with randomness.
$\mathsf{Accept}(F, \mathcal{VC})$	Predicate: device accepts firmware F under Verifiable Credential $\mathcal{VC}$.
$F^\star$	The canonical firmware whose hash matches the value embedded in $\mathcal{VC}$.
h_{vc}	Firmware hash value carried in $\mathcal{VC}$.
`firmwareHash`	VC field carrying the SHA-256 hash of the firmware binary.
$H(\cdot)$	SHA-256 of the canonical JSON representation of its argument (used for VC/VP hashing).
$\mathrm{Adv}^{\text{euf-cma}}_{\text{Ed25519}}$	Adversary's advantage in breaking Ed25519 signatures under Existential Unforgeability with Chosen-Message Attack (EUF-CMA).
$\mathrm{Adv}^{\text{cr}}_{\text{SHA256}}$	Adversary's advantage in breaking SHA's collision resistance (CR).
c_i	Individual claim value (e.g., firmware version, device model).
R	Signed Merkle root committing to all claims in a VC.
$\mathsf{isVCRevoked}(H(\mathcal{VC}))$	Smart contract query checking whether a VC has been revoked.
C_{expected}	Verifier's expected smart contract (registry) binding for VCs.

Proof sketch. Acceptance requires $\mathcal{VC}[\text{contract}] = C_{\text{expected}}$ and consistency with the `recordVC`/`revokeVC` state in C_{expected}. Rebinding a VC issued under C' to C_{expected} without re-signing is a signature forgery; copying state across contracts or chains does not affect the EVM state of C_{expected}.

Interpretation. A credential tied to one contract cannot be reused under another contract or chain. Any attempt requires either signature forgery or tampering with blockchain state.

Lemma 1 (*Gateway-independence (G4)*)**.** *Let $V(F, \mathcal{VC}, \mathcal{VP}, b)$ denote the device verifier that rechecks signatures, hash equality, contract binding, and revocation at block b. For any gateway strategy, the device's acceptance/rejection decision is equal to V on the inputs it receives; the gateway can affect liveness (delays, drops) but cannot change a rejection into an acceptance without causing a violation covered by the limits in the claims above.*

Interpretation. A gateway can delay or drop messages but cannot force a device to accept invalid inputs. Safety is enforced locally on the device.

3.5. Architecture and components

The system utilizes the following hardware components: Two Raspberry Pi 5, 16 GB RAM that serves as the vendor's node and smart home IoT gateway, respectively, with ESP32 WROOM-32 boards that act as IoT end-devices. The software stack used in the system consists of Arbitrum testnet for DID registration and VC revocation smart contracts (Solidity Implementation), IPFS (Go implementation + Pinata/web3.storage) for decentralized firmware and credential storage, Python FastAPI Backend for DID document publishing, VC issuance/verification, and smart contract interaction, ESP-IDF (C++) firmware for ESP32 supporting OTA, hash checking and Ed25519 signature verification, and HTTPS for secure firmware distribution. Six main entities are assumed, namely the user (owns the IoT gateway and IoT end device), firmware publisher (IoT vendor), the Arbitrum testnet, the IPFS storage layer, the smart home gateway and IoT end-devices. These entities interact using cryptographically authenticated protocols to facilitate secure firmware life cycle management orchestrated through REST APIs, smart contract calls, and IPFS retrievals.

The firmware distribution process is as follows. The firmware publisher, who is the root of trust, generates a DID using an Ed25519 key pair and publishes a corresponding DID document to IPFS. The CID of this document is registered on the Ethereum smart contract, enabling decentralized resolution. Firmware binaries are similarly uploaded to IPFS while its metadata, which include version number, compatible hardware model, SHA-256 hash, and CID, are included in a VC. This VC is signed with the publisher's private key and published to IPFS. There is an option to register the VC's hash on-chain to support credential revocation.

The smart contract that provides an efficient and tamper-proof mechanism for the decentralized management of identity and credentials eliminates the need for off-chain blacklists or centralized revocation services. This smart contract is deployed on the Arbitrum network. It has three main functions: registering DIDs with their associated IPFS CIDs (registerDID), retrieving the latest CID for a DID (getDIDCID) and publishing revocations of specific VCs (revokeVC). The IPFS network serves as the distributed file system for storing firmware binaries, DID documents, and VCs. IPFS uses content-based addressing to ensure stability and tamper-evidence, which in turn allows devices to verify the integrity of retrieved content using the embedded hash in the CID.

The smart home gateway validates firmware, resolves the VP, extracts the VC, and manages OTA updates. It maintains a list of trusted publisher DIDs and continuously monitors for new firmware credentials. When a new VC is detected, the gateway resolves the publisher's DID via the smart contract, then retrieves the DID document from IPFS and verifies the VC signature using the corresponding Ed25519 public key. Thereafter, the gateway checks the revocation status of the VC hash on-chain. Then, if the VC is valid and current, the gateway downloads the firmware binary from IPFS and verifies its SHA-256 hash against the value embedded in the VC. Before transmitting the firmware to a device, the smart home gateway verifies the authenticity of the target ESP32 by validating its VC, including digital signature, DID resolution, and revocation status. If the authentication is successful, the firmware is transmitted over a secure channel (Transport Layer Security: TLS). Thus, the smart home gateway enforces mutual trust and validates both identity and firmware integrity.

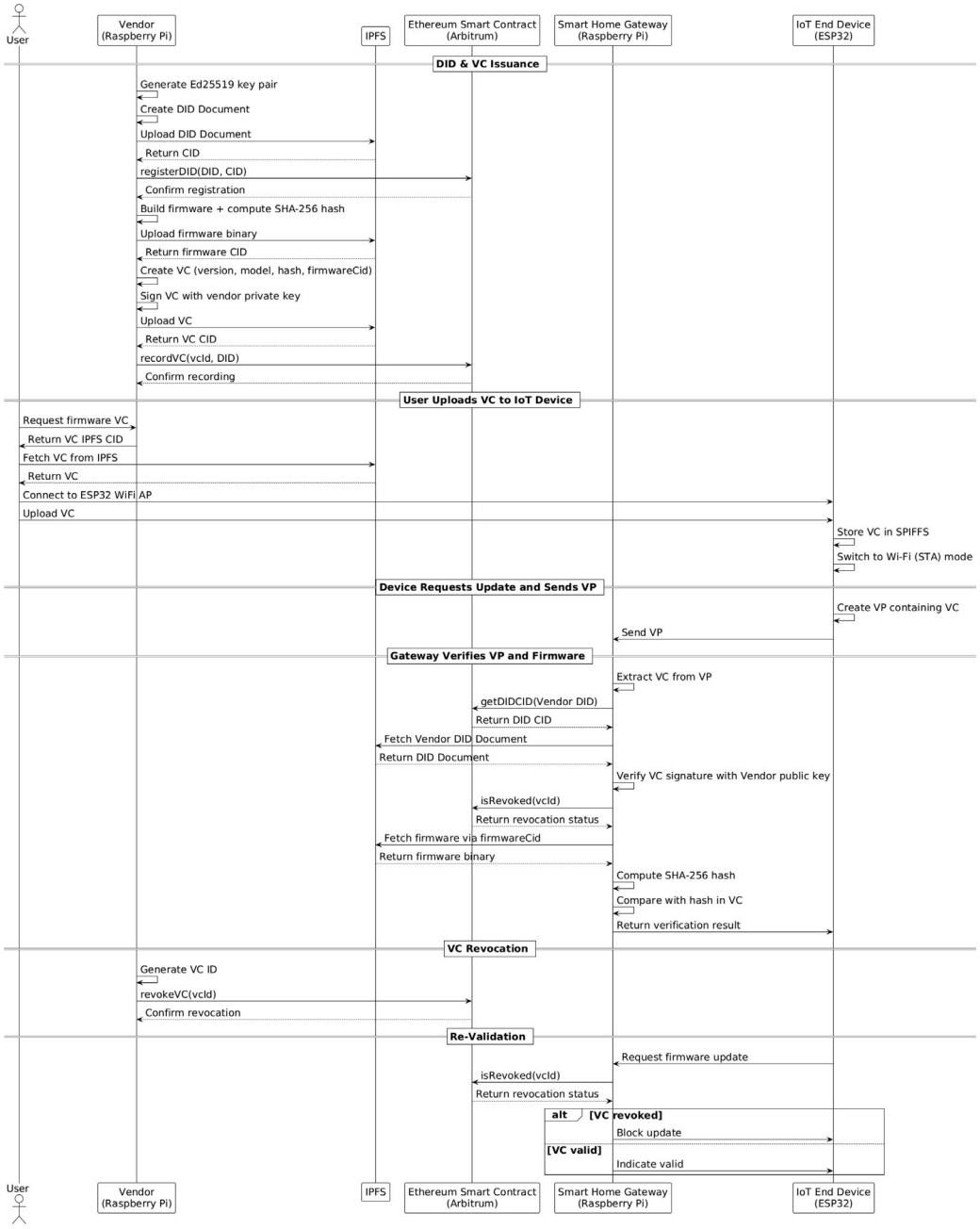

Fig. 2. Sequence diagram of the proposed firmware authentication architecture.

IoT end-devices were implemented using ESP32 development boards. Each of them was configured with a unique DID and a corresponding private key. These devices connect to the gateway via Wi-Fi and communicate using HTTPS. During the update process, each device verifies the authenticity of the VC received from the gateway and checks the firmware binary's hash. The update is installed only if all validations succeed. The devices can issue a signed VP to acknowledge that the firmware has been successfully installed, which can be logged by the gateway or forwarded to external monitoring systems.

Revocation management is integrated directly into the smart contract. Firmware publishers can revoke compromised or outdated credentials by publishing the hash of the VC via revokeVC. Before initiating any firmware update, the smart home gateway consults the on-chain revocation registry. If a credential is found to be revoked, the gateway stops the update and logs the event, ensuring that no invalid or unauthorized firmware is installed on the device. The process described above is summarized in Fig. 2.

The components of the proposed system interact through well-defined interfaces. The Python FastAPI backend exposes RESTful endpoints for DID creation (/did/create), VC issuance (/vc/create), VP creation (/vp/create), VP verification (/vp/verify), and VC revocation (/vc/revoke). The Ethereum smart contract maintains a registry of DIDs and revoked credentials, while IPFS stores the actual DID documents, VCs and firmware metadata. ESP32 devices communicate with the gateway via HTTPS and first authenticate through a challenge-response mechanism using their device DID and then receive and verify firmware updates. This modular design of the proposed system ensures that each component can be independently upgraded while maintaining interoperability through standardized interfaces.

Private key management is handled differently across the tiers of the system. The vendor's and the gateway's private keys use its local storage for its keys. When it comes to IoT devices, ESP32 boards store private keys in flash memory with enabled hardware encryption. Although this approach does not provide the same security level as dedicated secure elements, it offers a reasonable compromise for resource-constrained devices. For production environments, the optional ATECC608A secure element can be integrated to provide hardware-based key protection, which can prevent extraction even if the firmware is compromised.

3.6. Firmware verification life cycle

The overall firmware verification life cycle in DIDAuth-IoTFW is illustrated in Fig. 3. This diagram outlines the key stages, from initial VC issuance by the vendor, through user-side upload and validation, to revocation procedures. It emphasizes explicit life cycle management of firmware credentials and highlights the decentralized roles of each entity in securing firmware updates. Fig. 3 explicitly captures the key stages of the firmware verification life cycle highlighting the explicit life cycle management of firmware credentials, aligning with DIDAuth-IoTFW's decentralized approach to validation and revocation.

- **S1. Issue VC:** The firmware vendor issues a VC for a specific firmware version, including metadata and the firmware hash.
- **S2. Upload VC:** The user uploads the issued VC to the IoT gateway or device during onboarding or firmware update preparation.
- **S3. Validate VC:** The gateway or device verifies the authenticity, integrity, and revocation status of the uploaded VC against the on-chain registry.
- **S4. Firmware Accepted:** If the VC is valid and not revoked, the firmware is accepted and authorized for installation or execution.
- **S5. Revoke VC:** The vendor revokes a VC (e.g., due to detected vulnerabilities in the firmware), updating the on-chain revocation registry.
- **S6. VC Revoked:** Revocation status is synchronized across verifiers to prevent acceptance of the revoked VC.
- **S7. Issue New VC:** A new VC is issued by the vendor for the updated or remediated firmware, enabling continued trusted updates.

This figure highlights the explicit life cycle management of firmware credentials, aligning with DIDAuth-IoTFW's decentralized approach to validation and revocation.

3.7. Implementation algorithms

This section describes the algorithms 1–5 used in each step of the process explored in this study. Table 2 contains descriptive details about the notations used in the algorithms that follow.

Algorithm 1 Vendor DID Registration

Description: This algorithm defines how a vendor generates a key pair, writes the DID document to IPFS, and records it on-chain.
Input: *name*
Output: *Vendor_DID* registered on blockchain

1: Generate an Ed25519 key pair and save it
2: Construct *Vendor_DID* as `did:local:{name}`
3: Build DID document with verification method referencing the public key
4: Upload the DID document to IPFS → *DID_CID*
5: Invoke `registerDID(Vendor_DID, DID_CID)` on the registry contract
6: Store an API key for later authenticated calls
7: **return** *Vendor_DID*

3.8. Smart contract design and gateway synchronization

The smart contract `DIDRegistry` deployed on Arbitrum is central to the proposed trust model. It ensures registration of DIDs in a tamper-proof manner and records revoked VCs. The contract has four primary functions: `registerDID`, `recordVC`, `revokeVC`, and `isVCRevoked`. Internally, each DID is mapped to a DIDRecord structure containing the controller's Ethereum address and the IPFS CID of the DID document. A `revokeVC` event is issued upon credential revocation.

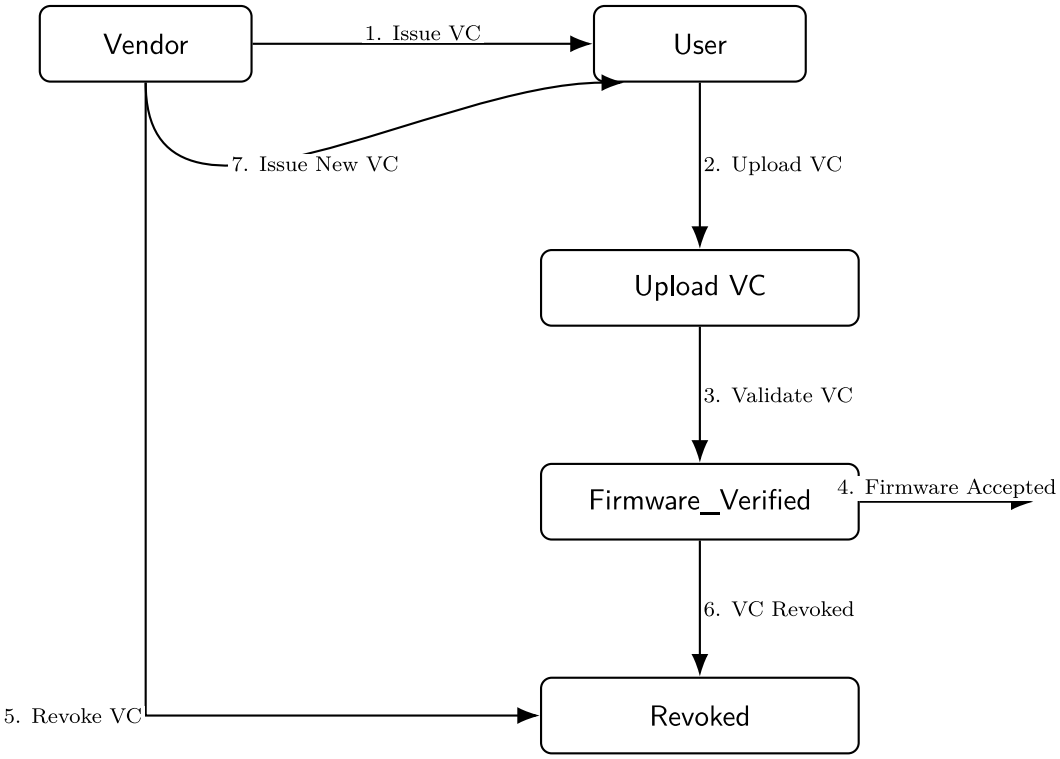

Fig. 3. Firmware verification life cycle with explicit VC issuance, validation, and revocation.

Table 2
Notations used in the algorithms.

Notation	Description
`Vendor_DID`	Decentralized Identifier created by the firmware vendor
`Vendor_PublicKey`	Public key of the vendor used to verify digital signatures
`Vendor_PrivateKey`	Private key of the vendor used to sign VCs
`Device_PublicKey`	Public key of the IoT end-device used to verify digital signatures
`Device_PrivateKey`	Private key of the IoT end-device used to sign a VP
`DID_CID`	IPFS Content Identifier (CID) of the DID document
`VC_CID`	IPFS CID of the verifiable credential
`VC`	Verifiable Credential containing firmware metadata and the vendor's digital signature
`VC_Hash`	SHA256 hash of the VC document (used for revocation tracking)
`Firmware_Binary`	Binary data of the firmware intended for OTA deployment
`Firmware_Hash`	SHA256 hash of the firmware binary computed during VC issuance
`Local_Hash`	SHA256 hash of the received firmware computed locally during verification
`Device_Model`	Identifier representing the specific IoT device model
`Firmware_Version`	Semantic version string of the firmware release
`VP`	Verifiable Presentation that encapsulates a VC and is signed by the IoT device
`isVCRevoked`	Smart-contract query returning whether a VC (by ID/hash) is revoked
`getDIDCID(did)`	Smart-contract function that returns the IPFS CID bound to a DID
`registerDID(. . . )`	Smart-contract function to register a new DID and its CID
`revokeVC(VC_Hash)`	Smart-contract function to revoke a VC by marking its hash as invalid
`getVCIssuer(vcId)`	Smart-contract function returning the DID that issued a given VC
`getDIDController(did)`	Smart-contract function returning the controller address of a DID
`Firmware_Verified`	Boolean result indicating VC signature, integrity, and policy checks passed
$\mathsf{JCS}(c_i)$	JSON Canonicalization Scheme representation of claim c_i prior to hashing
s_i	Per-claim random salt used to compute commitments
h_i	Per-claim commitment $h_i = \mathsf{SHA256}(s_i \parallel \mathsf{JCS}(c_i))$
R	Merkle tree root over the set $\{h_i\}$
$\texttt{merkleProof}_i$	Authentication path proving inclusion of h_i under root R

In order to preserve integrity, only the controller of a registered DID is authorized to revoke associated VCs. The controller refers to the externally owned Ethereum address that originally calls `registerDID` on the smart contract. It is distinct from the smart contract address itself, which acts only as a passive verification layer and not an identity holder. This constraint is enforced using the `onlyController` modifier. Internally, the contract uses a getDIDController() function to retrieve the address

Algorithm 2 Verifiable Credential Issuance

This algorithm describes how a vendor issues a credential, uploads firmware and VC to IPFS, and records the VC on-chain.
Input: *Vendor_PrivateKey*, *Firmware_Binary*, *Vendor_DID*, *Firmware_Version*, *Device_Model*, *Contract_Address*
Output: *VC_CID*

Compute $Firmware\_Hash \leftarrow \text{SHA256}(Firmware\_Binary)$
Upload the firmware binary to IPFS → *Firmware_CID*
Construct the VC including:
    firmwareHash, *Firmware_CID*, *Firmware_Version*, *Device_Model*, *Contract_Address*
Sign the VC using *Vendor_PrivateKey*
Upload the signed VC to IPFS → *VC_CID*
Call `recordVC(vc_id, Vendor_DID)` on the smart contract
return *VC_CID*

Algorithm 3 Verifiable Presentation Creation on IoT Device

Input: stored VC and *Device_PrivateKey*
Output: *VP*

Load the VC from SPIFFS (`/vc.json`)
Build a VP object with fields:
    `holder`: device DID
    `verifiableCredential`: the loaded VC
    `proof`: JSON-LD proof signed using *Device_PrivateKey* (Ed25519)
        includes fields: `type`, `created`, `proofPurpose`, `verificationMethod`, and `jws`
Send the VP JSON to the gateway's `/vp/verify` endpoint

Algorithm 4 Verifiable Credential Verification

This algorithm shows how the IoT device validates a presentation, checks credential authenticity, and ensures firmware integrity.
Input: *VP*, *Firmware_Binary*, *Known_Contract_Address*
Output: *Firmware_Verified* $\in \{\text{True}, \text{False}\}$

Verify the VP signature if a proof is present
Extract the embedded VC
Resolve the issuer's DID document (from IPFS or blockchain)
Verify the VC signature using the issuer's public key
Check that *vc.contractAddress* matches *Known_Contract_Address*
Query the smart contract using `isVCRevoked(VC_ID)` to check revocation status
If the VC contains *firmwareCid*, download firmware from IPFS
Compute SHA-256 hash and compare with *firmwareHash* in VC
if all checks succeed **then**
    return True
else
    return False
end if

Algorithm 5 Verifiable Credential Revocation

This algorithm describes how the vendor revokes a credential using the smart contract.
Input: *Vendor_PrivateKey*, *VC_ID*
Output: VC marked as revoked on-chain

Build a transaction calling `revokeVC(VC_ID)`
Sign the transaction with *Vendor_PrivateKey* to prove authority
Broadcast the transaction and wait for confirmation
VC is now publicly marked as revoked

that originally registered the DID. This ensures that only this controller can revoke the credentials. Events such as `DIDRegistered` and `revokeVC` enable transparent auditing of identity actions. Verifiers may optionally use `getVCIssuer(vcId)` to resolve the issuer DID from a credential ID. To avoid inconsistency and ensure verifiability, the contract address used during VC issuance is embedded within the VC. This binds the credential to a specific registry instance. All verifiers, including smart home gateways and

IoT devices must reference the same `DIDRegistry` instance. If the gateway resolves a DID using a different contract than the issuer, resolution failures or incorrect revocation status may occur, resulting in false negatives or false positives during validation. To further strengthen contract reliability, we subjected the 'DIDRegistry.sol' smart contract to static analysis using Slither [48], a leading open-source static analysis framework for Solidity. Slither analyzed 100 vulnerability patterns including reentrancy, uninitialized storage, dangerous delegate calls, and visibility issues. No vulnerabilities were reported, confirming the absence of known anti-patterns or insecure logic in the contract.

The gateway component is designed to operate in two modes:

- **Full Verification Mode**: When both `ARB_SEPOLIA_RPC` and `CONTRACT_ADDRESS` environment variables are defined, the gateway resolves DID documents and revocation status via on-chain queries. It retrieves the correct IPFS CID using `getDIDCID(did)` and checks VC revocation using `isVCRevoked(vcId)`. This mode is recommended when reliable internet connectivity is available, ensuring continuous DID resolution and real-time revocation check.
- **Degraded Mode (Offline)**: If `ARB_SEPOLIA_RPC` and `CONTRACT_ADDRESS` values are undefined or unreachable, the gateway performs only cryptographic signature validation using the embedded public key. Although it enables basic credential verification in environments with limited or intermittent connectivity, it lacks the capability to detect credential revocation or resolve updated DID documents, reducing overall security. Periodic synchronization in the Full Verification mode is therefore recommended.

This dual-mode design ensures resilience in constrained applications. Meanwhile, it preserves the option for robust on-chain validation when infrastructure allows. It also emphasizes the need for vendors and verifiers to remain synchronized on a shared registry contract to ensure trust.

3.9. Backend access control using API keys

The proposed framework uses access control based on the API key in order to secure the interactions with backend services responsible for DID registration, VC issuance, and revocation. When a DID is registered, each vendor receives a unique API key associated with their identity. This key is required when calling protected endpoints such as `/vc/issue` or `/vc/revoke`.

API keys are generated as high-entropy tokens and encrypted using the Fernet scheme, which utilized AES-128 in CBC mode along with HMAC-based integrity verification. These encrypted keys are stored alongside vendor metadata in the backend and decrypted solely during authentication checks. API keys never stored or transmitted in plaintext.

All protected API endpoints enforce strict validation of the API key. Any unauthenticated request is immediately rejected, thereby ensuring that only authorized vendors can issue or revoke credentials. This off-chain security mechanism effectively complements the on-chain contract enforcement by integrating an application-layer trust model.

In contrast to the on-chain model, where authorization is derived from the Ethereum address invoking a smart contract, the API key model introduces a lightweight but effective access control layer for off-chain services. This abstraction is especially advantageous when simulating real-world behavior of vendors, integrating with Continuous Integration and Continuous Delivery/Deployment (CI/CD) pipelines or isolating credential management logic from end-user-facing interfaces.

In addition, the API key mechanism reduces the complexity of operation by allowing vendors to use a single token to manage their DIDs securely. Meanwhile, it also eliminates the need to directly sign transactions or expose private keys at the application layer. Vendors may also request API key rotation or revocation through an authenticated endpoint, enabling secure life cycle management in case of key compromise or operational changes.

4. Performance and security evaluation

To assess the efficiency of the proposed decentralized firmware authentication system, we conducted a series of real-world performance tests focusing on three critical operations: DID creation, VC issuance and VC revocation. Tests were run using structured payload sizes to simulate varying metadata volumes. Performance metrics were collected in a systematic manner. For each operation, we measured 100 latencies per payload size with five different payload sizes of 100, 200, 300, 400 and 500 characters. All tests were conducted on the same hardware configuration to ensure consistency, with network conditions simulating real-world deployment scenarios including variable latency (5–50 ms). The following durations were measured:

- Latency for DID creation: the duration required to generate a new DID, upload its corresponding DID document to IPFS and to register the associated CID on the Ethereum smart contract deployed to the Arbitrum network.
- Latency for VC creation: the time needed to construct a VC, sign it using Ed25519, upload the credential to IPFS and to return its CID. This reflects the efficiency of credential issuance and metadata handling in the decentralized stack.
- Latency for VC revocation: the time taken to publish the hash of a previously issued credential to the smart contract via the revokeVC function. This step ensures the on-chain invalidation of revoked or compromised firmware credentials.

Table 3
Results of threat model evaluation against credential forgery (A1), replay attack (A2), and gateway compromise (A3). All credential forgery and gateway compromise scenarios were successfully detected through DID resolution and signature verification failures. Replay attacks were consistently blocked through revocation checks.

Iteration	Scenario	Result	HTTP	Status message
1	Credential forgery (A1)	Success	200	Failed to fetch DID: did:local:attacker
1	Replay attack (A2)	Failure	200	VC is revoked
1	Gateway compromise (A3)	Success	200	Failed to fetch DID: 'proof'
2	Credential forgery (A1)	Success	200	Failed to fetch DID: did:local:attacker
2	Replay attack (A2)	Failure	200	VC is revoked
2	Gateway compromise (A3)	Success	200	Failed to fetch DID: 'proof'
3	Credential forgery (A1)	Success	200	Failed to fetch DID: did:local:attacker
3	Replay attack (A2)	Failure	200	VC is revoked
3	Gateway compromise (A3)	Success	200	Failed to fetch DID: 'proof'
4	Credential forgery (A1)	Success	200	Failed to fetch DID: did:local:attacker
4	Replay attack (A2)	Failure	200	VC is revoked
4	Gateway compromise (A3)	Success	200	Failed to fetch DID: 'proof'
5	Credential forgery (A1)	Success	200	Failed to fetch DID: did:local:attacker
5	Replay attack (A2)	Failure	200	VC is revoked
5	Gateway compromise (A3)	Success	200	Failed to fetch DID: 'proof'

4.1. Device-level evaluation method

To evaluate the feasibility of DIDAuth-IoTFW on resource-constrained hardware, we implemented a custom firmware on an ESP32 microcontroller. The evaluation focused on validating the device's ability to manage Verifiable Credentials (VCs) within its limited memory and processing resources.

The evaluation workflow involved the ESP32 initially starting in Access Point (AP) mode to receive a signed VC through an HTTP POST request. Upon successful upload, the VC was stored in the device's SPI Flash File System (SPIFFS). The device then rebooted into Wi-Fi Station mode, retrieved the stored VC, and submitted it as a Verifiable Presentation (VP) over HTTPS to the backend gateway server for verification.

The following metrics were collected:

- Heap memory usage before and after VC read and transmission.
- Total verification latency from Wi-Fi connection initiation to VC verification response.
- Credential payload size (2–4 KB typical).
- Revocation detection accuracy for valid, tampered and revoked credentials.
- HTTP response status codes and distribution.

Each test was conducted over 100 automated cycles under consistent environmental conditions, ensuring reproducibility and eliminating variance due to network fluctuations.

4.2. Security evaluation under adversarial conditions

To explicitly validate our threat model (see Section 3.2), we conducted controlled experiments by simulating three adversarial scenarios as follows.

Credential Forgery (A1): VCs were intentionally tampered with modifying metadata after signature issuance. The system successfully detected all such forgery attempts through cryptographic signature verification and DID resolution failure, consistently rejecting them with 100% accuracy. The responses explicitly indicated the failure to resolve unregistered or invalid DIDs.

Replay Attacks (A2): Revoked credentials were replayed to gateways and devices. DIDAuth-IoTFW's on-chain revocation mechanism reliably detected all such attempts, preventing unauthorized firmware installations. Revocation status checks consistently identified revoked credentials and rejected their use.

Gateway Compromise (A3): A compromised gateway scenario was simulated by attempting to verify tampered VCs directly against the backend, bypassing normal gateway-side verification. The backend independently validated credentials through DID resolution and signature verification. Invalid credentials referencing non-existent DIDs were consistently rejected with explicit error responses. These results demonstrated the system's resilience, even in cases where gateway defenses are disabled or compromised.

The detailed results of these adversarial tests are summarized in Table 3. Across all iterations, credential forgery and gateway compromise were successfully detected through verification failures, while replay attacks were effectively blocked through revocation enforcement.

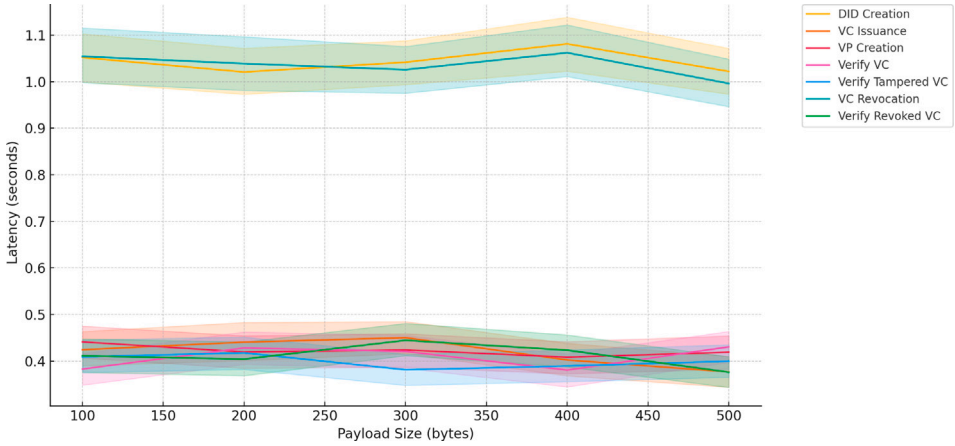

Fig. 4. Latency distribution demonstrating stable performance across varying payload sizes.

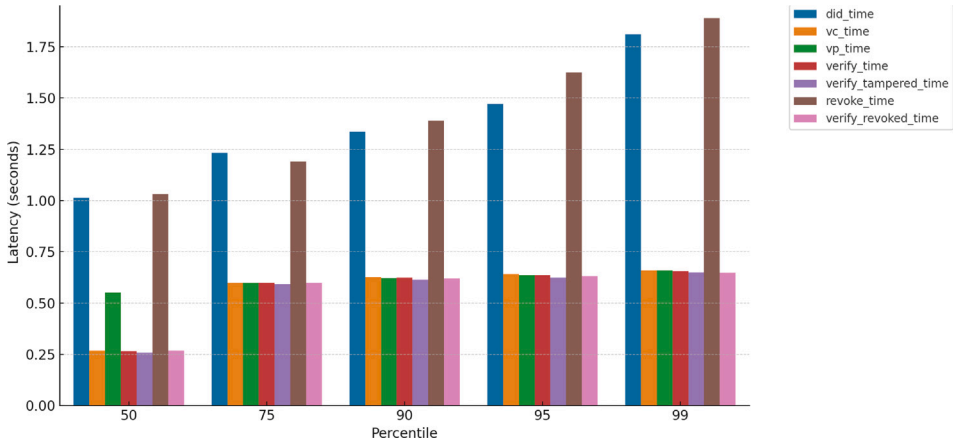

Fig. 5. Percentile latency distribution for all operations.

5. Results and discussion

The collected results are comprehensively summarized in Figs. 4 and 5. The latency measurements confirm that VC issuance `vc_time`, VP creation `vp_time`, and all verification-related tasks operated consistently below 0.75 s across all payload sizes tested. In contrast, operations involving DID registration `did_time` and VC revocation `revoke_time` exhibited higher latencies, with some instances approaching or exceeding 1.75 s due to the overhead of interacting with blockchain smart contracts and IPFS updates. As highlighted in Fig. 5, these operations present the upper limits of latency within the system. These performance data were further validated by Fig. 4, which illustrates stable latency trends with respect to increasing payload sizes. These findings demonstrate that, while most identity-related operations scale efficiently, DID creation and revocation require additional consideration in time-sensitive applications. To further examine the relationship between payload size and operational latency, we computed the Pearson correlation matrix across all measured operations, as shown in Fig. 6. Most operations exhibited low correlation with payload size (near-zero correlation coefficients), indicating performance stability and independence from metadata size. High correlations between related operations such as `verify_time` and `verify_tampered_time` suggest a consistent cryptographic overhead profile. This matrix confirms the deterministic nature of the system and validates the reliability of our decentralized execution pipeline.

Our framework consistently demonstrated consistently low-latency performance across all core operations: DID registration, VC issuance, revocation, and verification, confirming its feasibility for real-time use even on resource-constrained devices. DID registration latency remained below 1.2 s for all payload sizes, with minimal deviation. Credential issuance, which involves structured metadata generation, Ed25519 signing, and IPFS publishing, completed within 0.5 s on average.

Verifiable Credential revocation, requiring an on-chain Ethereum Layer-2 transaction, showed slightly higher variability but consistently completed within 1.1 s in all test cases. Most notably, VC validation operations executed directly on ESP32 and Raspberry Pi devices, remained under 1.2 s, even during adversarial replay attempts and gateway compromise simulations. This

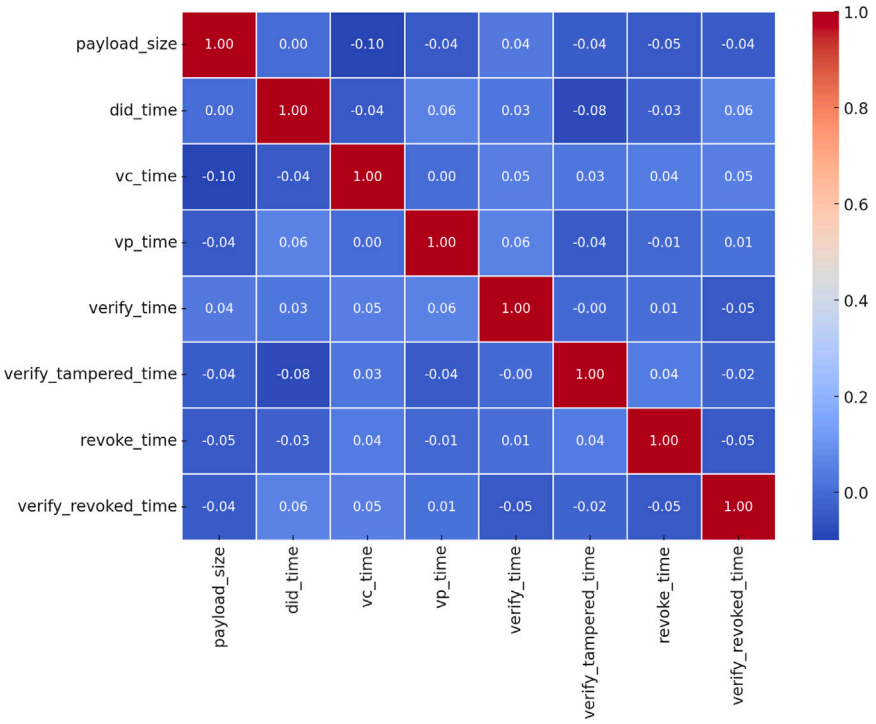

Fig. 6. Correlation matrix showing relationships between payload size and operation latencies.

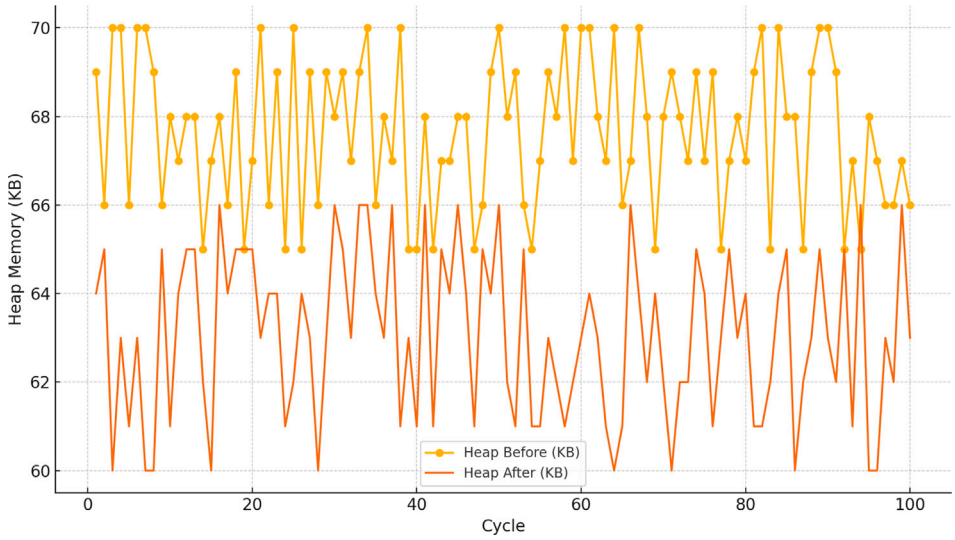

Fig. 7. Heap memory usage on ESP32 over 100 cycles, confirming stable memory management.

includes time for SHA-256 hash verification, digital signature checks, and on-chain revocation status resolution. These results confirm the suitability of DIDAuth-IoTFW for time-sensitive firmware authentication in smart home environments.

Additionally, Fig. 7 illustrates the heap memory usage of the ESP32 across 100 test cycles. Both pre and post verification measurements remained stable, with available heap memory consistently between 60–70 KB, confirming the absence of memory leaks or performance degradation over repeated VC processing operations. This stability further validates the suitability of DIDAuth-IoTFW for deployment on constrained embedded devices.

Our approach demonstrated several notable advantages over existing solutions. A comparative analysis of DIDAuth-IoTFW with existing decentralized firmware authentication approaches is summarized in Table 4, highlighting the unique combination of features provided by our proposed framework. Compared to [49], who proposed a blockchain-based firmware authentication system with

Table 4
Comparative analysis of DIDAuth-IoTFW against recent decentralized firmware authentication approaches.

Feature	DIDAuth-IoTFW (Proposed)	Gnomon (2019) [30]	EVOKE (2024) [32]	Oktian et al. (2024) [10]	Ali et al. (2025) [42]
Identity model	DID, VC (SSI)	DID, VC	VC-based (partial SSI)	Not DID/VC-based	Not identity-focused
Decentralization	Full (Arbitrum L2 + IPFS)	Partial (external identity hubs, off-chain)	Partial (on-chain revocation only)	Partial (smart contract only)	No (trusted gateway model)
Firmware authentication	Yes (VC-bound hash, IPFS, on-chain revoke)	Partial (registration; no firmware hash)	No (not firmware-specific)	Yes (manifests on-chain; no VC binding)	No (OTA anomaly detection only)
Privacy/disclosure	Yes (off-chain VC/VP, DID)	Partial (limited off-chain)	Yes (minimal disclosure; revocation-focused)	No	No
Revocation mechanism	Real-time on-chain	Off-chain	On-chain (privacy-preserving; limited scope)	On-chain (limited)	Not addressed
Hardware evaluation	Yes (ESP32, Raspberry Pi)	No explicit hardware testing	No explicit hardware testing	No explicit hardware testing	No explicit hardware testing
Interoperability	W3C VC/DID, IPFS, Ethereum L2	Partial (W3C DID)	W3C VC standards	Ethereum-based (non-SSI)	Not applicable
OTA security	Hash verify; decentralized delivery	Not explicitly addressed	Not explicitly addressed	Blockchain proof of delivery	Attack detection only (DoS/DDoS)
Zero-knowledge privacy	Future work	No	Partial	No	No
Latency	<1.2 s average	Not reported	Millisecond-range (revocation only)	1.5–2.0 s on constrained devices	Not measured (anomaly detection only)

centralized storage, our solution effectively eliminates single points of failure through comprehensive IPFS integration. Real-time attack detection during OTA firmware updates was achieved typically within 1 to 2 s, as reported by [42], with the requirement of a trusted gateway or edge device as the enforcement point. In contrast, our system operated in a fully decentralized manner using DIDs, VCs and on-chain revocation, eliminating the need for any trusted intermediary. Furthermore, unlike [41], whose credential verification consumed 1.5–2.0 s on resource-constrained devices, our implementation completes verification in under 1.2 s (as shown in Fig. 8 which illustrates combined SHA-256 hashing and signature verification), making it highly suitable for real-time IoT environments.

The results further highlight the resilience and accuracy of the system under adversarial testing. Verifications occurred in real time and revoked credentials were reliably blocked by both the gateway and the ESP32 device. In 200 trials (100 with valid credentials, 100 revoked), the system achieved 100% detection accuracy, with no false positives or negatives. Simulated replay attacks using outdated timestamps and credential forgery attempts were successfully rejected, affirming tamper resistance at multiple enforcement (see Fig. 8).

In terms of system robustness, our revocation mechanisms proved reliable, enabling firmware validation without delay. Immutability and integrity were ensured through the combined use of IPFS and SHA-256. Even under simulated credential compromise, the system maintained consistent security behavior. Overall, the experimental outcomes conclusively confirm that decentralized identity frameworks can offer secure, scalable, and tamper-evident firmware validation suitable for smart home environments, when carefully implemented with lightweight cryptography and efficient smart contracts. This proposed approach demonstrates a practical alternative to cloud-based or certificate-dependent solutions and provides a pathway toward resilient, self-sovereign IoT device management.

In summary, the evaluation results confirm that DIDAuth-IoTFW achieves reliable, low-latency, and resource-efficient operation on constrained IoT hardware, while maintaining security guarantees through decentralized revocation and validation mechanisms. These findings demonstrate its practicality for real-world smart home firmware authentication scenarios and its advantages over existing solutions in terms of decentralization, performance, and robustness.

6. Conclusions & future work

This paper presented DIDAuth-IoTFW, a novel decentralized firmware authentication architecture specifically designed to address the critical trust, privacy, and scalability challenges prevalent in modern IoT environments. By integrating Ethereum Layer-2 smart contracts (Arbitrum), IPFS-based decentralized storage, and W3C-compliant decentralized identity components (DIDs and VCs), the system effectively eliminates the reliance on centralized verification authorities. This comprehensive approach enables secure, tamper-evident firmware life cycle management from initial issuance to device-side enforcement.

Through a rigorous real-world implementation on ESP32 WROOM-32 and Raspberry Pi 5 devices, our evaluation conclusively confirmed that all key operations, including DID registration, VC issuance, on-chain revocation, and credential verification, were

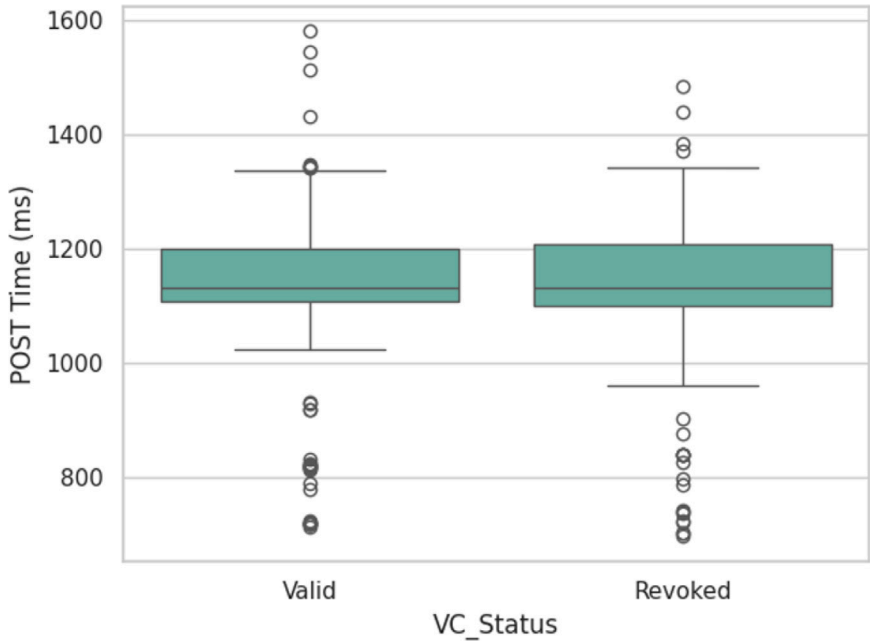

Fig. 8. Validation latency on ESP32 devices illustrating reliable real-time verification under resource constraints.

completed consistently within 1.2 s, even under adversarial conditions. Specifically, credential revocation was robustly enforced on-chain by marking credential hashes as invalid, while firmware verification was successfully validated directly on resource-constrained endpoints. The system achieved 100% detection accuracy across all credential states without false positives or negatives. Furthermore, the implementation of encrypted API key-based access control for backend services established a secure and scalable off-chain identity life cycle management.

Unlike general-purpose Layer-2 solutions, such as Arbitrum, which are not inherently designed to offer firmware-level trust primitives, DIDAuth-IoTFW bridges this gap. It achieves this by providing device-bound credential validation and decentralized enforcement mechanisms that are aligned with W3C VC and DID standards. This design enables vendor-neutral participation without relying on centralized certificate authorities or proprietary verification infrastructure.

Future Work: Building upon the work presented in this article, future research could explore the application of DID and VC technologies in environments such as healthcare and industrial IoT to enable secure and privacy-preserving access control over sensitive data. Particular focus could be given to selective disclosure consistent with relevant standards. In the near term, per-claim SHA-256 commitments could be used with a Merkle root embedded in the VC. In the longer term, zero-knowledge proofs could be investigated to minimize attribute disclosure. In addition, compliance with sector-specific data-protection regulations could be assessed and credential life-cycle management and real-time revocation across dynamic, cross-organizational ecosystems could be evaluated. In parallel, device key custody could be strengthened by anchoring long-term private keys in hardware secure elements with on-chip Ed25519 signing. Furthermore, performance and adversarial testing could be extended to mission-critical settings and multi-chain deployments across EVM-compatible networks could be analyzed to reduce dependence on a single Layer-2 and to improve resilience to governance risks and outages.

CRediT authorship contribution statement

W.M.A.B. Wijesundara: Writing – original draft, Visualization, Validation, Software, Methodology, Investigation, Formal analysis, Data curation, Conceptualization. **Joong-Sun Lee:** Writing – review & editing, Validation, Supervision. **Eleni Aloupogianni:** Writing – review & editing, Visualization, Validation, Software, Investigation. **Dara Tith:** Writing – review & editing, Visualization, Validation, Software, Investigation. **Hiroyuki Suzuki:** Writing – review & editing, Validation. **Takashi Obi:** Writing – review & editing, Validation, Supervision, Resources, Project administration, Investigation, Funding acquisition, Formal analysis.

Code and data

All implementation details have been validated against the described system design, and the complete source code along with the experimental data sets supporting this study are publicly available at: https://github.com/awijesundara/DIDAuth-IoTFW.

Declaration of competing interest

The authors declare that they have no known competing financial interests or personal relationships that could have appeared to influence the work reported in this paper.

Acknowledgment

The authors thank the open-source contributors of FastAPI, IPFS, Web3.py, Solidity, and Arbitrum Sepolia for enabling decentralized identity testing.

Code and data

All implementation details have been validated against the described system design, and the complete source code along with the experimental data sets supporting this study are publicly available at: https://github.com/awijesundara/DIDAuth-IoTFW.

References

[1] H. Rastegari, F. Nadi, S.S. Lam, M. Ikhwanuddin, N.A. Kasan, R.F. Rahmat, W.A.W. Mahari, Internet of Things in aquaculture: A review of the challenges and potential solutions based on current and future trends, Smart Agric. Technol. 4 (2023) 100187, http://dx.doi.org/10.1016/j.atech.2023.100187, URL https://www.sciencedirect.com/science/article/pii/S2772375523000175.

[2] A. Aldahmani, B. Ouni, T. Lestable, M. Debbah, Cyber-security of embedded IoTs in smart homes: Challenges, requirements, countermeasures, and trends, IEEE Open J. Veh. Technol. 4 (2023) 281–292, http://dx.doi.org/10.1109/OJVT.2023.3234069.

[3] S. Kotel, F. Sbiaa, R.M. Kamoun, L. Hamel, A Blockchain-based approach for secure IoT, Procedia Comput. Sci. 225 (2023) 3876–3886, http://dx.doi.org/10.1016/j.procs.2023.10.383, 27th International Conference on Knowledge Based and Intelligent Information and Engineering Sytems (KES 2023). URL https://www.sciencedirect.com/science/article/pii/S1877050923015417.

[4] A. Javed, A. Ehtsham, M. Jawad, M.N. Awais, A.-u.-H. Qureshi, H. Larijani, Implementation of lightweight machine learning-based intrusion detection system on IoT devices of smart homes, Futur. Internet 16 (6) (2024) http://dx.doi.org/10.3390/fi16060200, URL https://www.mdpi.com/1999-5903/16/6/200.

[5] Y.I. Alzoubi, A. Al-Ahmad, H. Kahtan, A. Jaradat, Internet of things and blockchain integration: Security, privacy, technical, and design challenges, Futur. Internet 14 (7) (2022) http://dx.doi.org/10.3390/fi14070216, URL https://www.mdpi.com/1999-5903/14/7/216.

[6] N. Denis, S. Chabridon, M. Laurent, Bringing privacy, security and performance to the Internet of Things using IOTA and usage control, Ann. Telecommun. 79 (7) (2024) 585–601, http://dx.doi.org/10.1007/s12243-023-01005-1.

[7] M. Bhargavi, Y. Pachipala, Enhancing IoT security and privacy with claims-based identity management, Int. J. Adv. Comput. Sci. Appl. 14 (11) (2023) http://dx.doi.org/10.14569/IJACSA.2023.0141183.

[8] A. Attkan, V. Ranga, Cyber-physical security for IoT networks: a comprehensive review on traditional, blockchain and artificial intelligence based key-security, Complex Intell. Syst. 8 (4) (2022) 3559–3591, http://dx.doi.org/10.1007/s40747-022-00667-z.

[9] P. Sun, Y. Wan, Z. Wu, Z. Fang, Q. Li, A survey on privacy and security issues in IoT-based environments: Technologies, protection measures and future directions, Comput. Secur. 148 (2025) 104097, http://dx.doi.org/10.1016/j.cose.2024.104097.

[10] Y.E. Oktian, T.-T.-H. Le, U. Jo, A.M.A. Laksmono, H. Kim, Secure decentralized firmware update delivery service for Internet of Things, Internet Things 26 (2024) 101136, http://dx.doi.org/10.1016/j.iot.2024.101136, URL https://www.sciencedirect.com/science/article/pii/S2542660524000787.

[11] G. Solomon, P. Zhang, R. Brooks, Y. Liu, A secure and cost-efficient blockchain facilitated IoT software update framework, IEEE Access 11 (2023) 44879–44894, http://dx.doi.org/10.1109/ACCESS.2023.3272899.

[12] J. Benet, IPFS – content addressed, versioned, P2P file system, 2014, https://ipfs.io/ipfs/QmR7GSQM93Cx5eAg6a6dDG1t9A8V6u8RvG1j1h4i6RXSvG.

[13] G. Wood, Ethereum: A secure decentralised generalised transaction ledger, 2025, https://ethereum.github.io/yellowpaper/paper.pdf.

[14] B. Cohen, The BitTorrent protocol specification, 2008, https://www.bittorrent.org/beps/bep_0003.html.

[15] S. Choi, J.-H. Lee, Blockchain-based distributed firmware update architecture for IoT devices, IEEE Access 8 (2020) 37518–37525, http://dx.doi.org/10.1109/ACCESS.2020.2975920.

[16] N. Mtetwa, P. Tarwireyi, M. Adigun, Secure the internet of things software updates with ethereum blockchain, in: 2019 International Multidisciplinary Information Technology and Engineering Conference, IMITEC, 2019, pp. 1–6, http://dx.doi.org/10.1109/IMITEC45504.2019.9015865.

[17] M.R. Ahmed, A.K.M.M. Islam, S. Shatabda, S. Islam, Blockchain-based identity management system and self-sovereign identity ecosystem: A comprehensive survey, IEEE Access 10 (2022) 113436–113481, http://dx.doi.org/10.1109/ACCESS.2022.3216643.

[18] R. Bochnia, D. Richter, J. Anke, Self-sovereign identity for organizations: Requirements for enterprise software, IEEE Access 12 (2024) 7637–7660, http://dx.doi.org/10.1109/ACCESS.2023.3349095.

[19] B. Houtan, A.S. Hafid, D. Makrakis, A survey on blockchain-based self-sovereign patient identity in healthcare, IEEE Access 8 (2020) 90478–90494, http://dx.doi.org/10.1109/ACCESS.2020.2994090.

[20] M. Popa, S.M. Stoklossa, S. Mazumdar, ChainDiscipline - towards a blockchain-IoT-based self-sovereign identity management framework, IEEE Trans. Serv. Comput. 16 (5) (2023) 3238–3251, http://dx.doi.org/10.1109/TSC.2023.3279871.

[21] W.M.A.B. Wijesundara, J.-S. Lee, D. Tith, E. Aloupogianni, H. Suzuki, T. Obi, Security-enhanced firmware management scheme for smart home IoT devices using distributed ledger technologies, Int. J. Inf. Secur. 23 (3) (2024) 1927–1937, http://dx.doi.org/10.1007/s10207-024-00827-x.

[22] L. Wang, Y. Yuan, Y. Ding, Analysis and design of identity authentication for IoT devices in the blockchain using hashing and digital signature algorithms, Int. J. Distrib. Sens. Netw. 2023 (1) (2023) 2524051, http://dx.doi.org/10.1155/2023/2524051, arXiv:https://onlinelibrary.wiley.com/doi/pdf/10.1155/2023/2524051. URL https://onlinelibrary.wiley.com/doi/abs/10.1155/2023/2524051.

[23] C.D. Nassar Kyriakidou, A.M. Papathanasiou, G.C. Polyzos, Decentralized identity with applications to security and privacy for the internet of things, Comput. Netw. Commun. (2023).

[24] K.P. Satamraju, B. Malarkodi, A decentralized framework for device authentication and data security in the next generation internet of medical things, Comput. Commun. 180 (2021) 146–160, http://dx.doi.org/10.1016/j.comcom.2021.09.012, URL https://www.sciencedirect.com/science/article/pii/S0140366421003492.

[25] B. Alamri, K. Crowley, I. Richardson, Blockchain-based identity management systems in health IoT: A systematic review, IEEE Access 10 (2022) 59612–59629, http://dx.doi.org/10.1109/ACCESS.2022.3180367.

[26] D.-H. Shin, S.-J. Han, Y.-B. Kim, I.-C. Euom, Research on digital forensics analyzing heterogeneous internet of things incident investigations, Appl. Sci. 14 (3) (2024) http://dx.doi.org/10.3390/app14031128, URL https://www.mdpi.com/2076-3417/14/3/1128.

[27] Statista, Number of connected devices worldwide as of February 2025, by device, 2025, URL https://www.statista.com/statistics/1559435/connected-devices-worldwide/.
[28] S. Venkatraman, S. Parvin, Developing an IoT identity management system using blockchain, Systems 10 (2) (2022) http://dx.doi.org/10.3390/systems10020039, URL https://www.mdpi.com/2079-8954/10/2/39.
[29] R. Neiheiser, G. Inácio, L. Rech, C. Montez, M. Matos, L. Rodrigues, Practical limitations of ethereum's layer-2, IEEE Access 11 (2023) 8651–8662, http://dx.doi.org/10.1109/ACCESS.2023.3237897.
[30] R. Ansey, J. Kempf, O. Berzin, C. Xi, I. Sheikh, Gnomon: Decentralized identifiers for securing 5G iot device registration and software update, in: 2019 IEEE Globecom Workshops, GC Wkshps, 2019, pp. 1–6, http://dx.doi.org/10.1109/GCWkshps45667.2019.9024702.
[31] G. Pescetelli, L. Petrosino, S.D. Valle, G. Ronga, M. Merone, L. Vollero, Framework for IoT ecosystems based on distributed ledger technologies and decentralized identifiers, in: 2022 IEEE International Workshop on Metrology for Industry 4.0 & IoT, MetroInd4.0&IoT, 2022, pp. 87–91, http://dx.doi.org/10.1109/MetroInd4.0IoT54413.2022.9831460.
[32] C. Mazzocca, A. Acar, S. Uluagac, R. Montanari, EVOKE: Efficient revocation of verifiable credentials in IoT networks, in: 33rd USENIX Security Symposium, USENIX Security 24, USENIX Association, Philadelphia, PA, 2024, pp. 1279–1295, URL https://www.usenix.org/conference/usenixsecurity24/presentation/mazzocca.
[33] M. Munkenyi, D. Francisco, G. Jorge, V.J. P., Blockchain-based device identity management with consensus authentication for IoT devices, in: 2022 IEEE 19th Annual Consumer Communications & Networking Conference, CCNC, 2022, pp. 433–436, http://dx.doi.org/10.1109/CCNC49033.2022.9700534.
[34] Z. Ullah, G. Husnain, M.I. Mohmand, M. Qadir, K.J. Alzahrani, Y.Y. Ghadi, H.K. Alkahtani, Blockchain-IoT: A revolutionary model for secure data storage and fine-grained access control in internet of things, IET Commun. 18 (19) (2024) 1524–1540, http://dx.doi.org/10.1049/cmu2.12845, arXiv:https://ietresearch.onlinelibrary.wiley.com/doi/pdf/10.1049/cmu2.12845. URL https://ietresearch.onlinelibrary.wiley.com/doi/abs/10.1049/cmu2.12845.
[35] P. Kumar, R. Kumar, G.P. Gupta, R. Tripathi, A. Jolfaei, A. Najmul Islam, A blockchain-orchestrated deep learning approach for secure data transmission in IoT-enabled healthcare system, J. Parallel Distrib. Comput. 172 (2023) 69–83, http://dx.doi.org/10.1016/j.jpdc.2022.10.002, URL https://www.sciencedirect.com/science/article/pii/S0743731522002106.
[36] P. Sharma, S. Namasudra, N. Chilamkurti, B.-G. Kim, R. Gonzalez Crespo, Blockchain-based privacy preservation for IoT-enabled healthcare system, ACM Trans. Sen. Netw. 19 (3) (2023) http://dx.doi.org/10.1145/3577926.
[37] A. Khurshid, D.T. Harrell, D. Li, C. Hallmark, L. Hanson, N. Viswanathan, M. Carr, A. Brown, M. McNeese, K. Fujimoto, Designing a blockchain technology platform for enhancing the pre-exposure prophylaxis care continuum, JAMIA Open 7 (4) (2024) ooae140, http://dx.doi.org/10.1093/jamiaopen/ooae140, arXiv:https://academic.oup.com/jamiaopen/article-pdf/7/4/ooae140/61240100/ooae140.pdf.
[38] A. Dayyani, M. Abbaspour, SIoT identity management with the ability to preserve owner privacy and built-in resistance to Sybil attacks, Concurr. Comput.: Pr. Exp. 36 (23) (2024) e8201, http://dx.doi.org/10.1002/cpe.8201, arXiv:https://onlinelibrary.wiley.com/doi/pdf/10.1002/cpe.8201. URL https://onlinelibrary.wiley.com/doi/abs/10.1002/cpe.8201.
[39] J. García-Rodríguez, A. Skarmeta, A privacy-preserving attribute-based framework for IoT identity lifecycle management, Comput. Netw. 236 (2023) 110039, http://dx.doi.org/10.1016/j.comnet.2023.110039, URL https://www.sciencedirect.com/science/article/pii/S138912862300484X.
[40] A. Wijesundara, J.-S. Lee, D. Tith, H. Suzuki, T. Obi, Development of a firmware authenticating and updating scheme for smart home IoT devices using distributed ledger technologies, in: Proceedings of the 2019 Computer Security Symposium, 2019, pp. 817–823.
[41] A. Qashlan, P. Nanda, X. He, Security and privacy implementation in smart home: Attributes based access control and smart contracts, in: 2020 IEEE 19th International Conference on Trust, Security and Privacy in Computing and Communications (TrustCom), IEEE Computer Society, Los Alamitos, CA, USA, 2020, pp. 951–958, http://dx.doi.org/10.1109/TrustCom50675.2020.00127, URL https://doi.ieeecomputersociety.org/10.1109/TrustCom50675.2020.00127.
[42] M. Ali, Y. Saleem, S. Hina, G.A. Shah, DDoSViT: IoT DDoS attack detection for fortifying firmware Over-The-Air (OTA) updates using vision transformer, Internet Things 30 (2025) 101527, http://dx.doi.org/10.1016/j.iot.2025.101527, URL https://www.sciencedirect.com/science/article/pii/S254266052500040X.
[43] X. He, S. Alqahtani, R. Gamble, M. Papa, Securing over-the-air IoT firmware updates using blockchain, in: Proceedings of the International Conference on Omni-Layer Intelligent Systems, COINS '19, Association for Computing Machinery, New York, NY, USA, 2019, pp. 164–171, http://dx.doi.org/10.1145/3312614.3312649.
[44] M. Aknan, M.P. Singh, R. Arya, Secure cloud assisted fog computing framework for IoT applications using ensemble learning, SN Comput. Sci. 6 (6) (2025) 717, http://dx.doi.org/10.1007/s42979-025-04257-x.
[45] R. Xiong, W. Ren, X. Hao, J. He, K.-K.R. Choo, BDIM: A blockchain-based decentralized identity management scheme for large scale internet of things, IEEE Internet Things J. 10 (24) (2023) 22581–22590, http://dx.doi.org/10.1109/JIOT.2023.3303922.
[46] M. Kaptan, E. Tomur, T. Ayav, Y.M. Erten, Secure IoT update using blockchain, in: 2021 2nd International Informatics and Software Engineering Conference, IISEC, 2021, pp. 1–6, http://dx.doi.org/10.1109/IISEC54230.2021.9672424.
[47] D. Dolev, A. Yao, On the security of public key protocols, IEEE Trans. Inform. Theory 29 (2) (1983) 198–208, http://dx.doi.org/10.1109/TIT.1983.1056650.
[48] Crytic, Slither: Static analysis framework for Solidity, 2023, https://github.com/crytic/slither.
[49] J. Parra Moyano, O. Ross, KYC optimization using distributed ledger technology, Bus. Inf. Syst. Eng. 59 (6) (2017) 411–423, http://dx.doi.org/10.1007/s12599-017-0504-2.