From Cloud to IoT Device Authenticity under Kubernetes Management

George Kornaros
ECE Dept.
Hellenic Mediteranean Univ.
Iraklio, Greece
0000-0002-2371-0633

Dimitris Bakoyiannis
ECE Dept.
Hellenic Mediteranean Univ.
Iraklio, Greece
dbakoyiannis@hmu.gr

Othon Tomoutzoglou
ECE Dept.
Hellenic Mediteranean Univ.
Iraklio, Greece
otto@hmu.gr

Marcello Coppola
STM
Grenoble, France
marcello.coppola@stm.com

***Abstract*—The proliferation of headless IP devices has posed serious security issues. These gadgets, which include sensors, smart meters, and industrial controllers, frequently lack built-in security protections and are susceptible to cyber attacks. Understanding these risks and establishing effective management techniques is vital for safeguarding critical infrastructure and data. Many IoT devices lack suitable authentication measures, leaving them vulnerable to unwanted access, particularly when the Kubernetes management framework is used at the edge. To connect to the network, headless IoT devices need to re-key and validate their credentials. This requires easy scalability and a trusted platform module (TPM) to build a chain of trust and protect all credentials. To maintain data security in Internet of Things environments, we provide a totally trusted infrastructure utilizing the SPIFFE/SPIRE framework that can ensure that, additionally to authentic workloads, only verified, trusted IoT devices may participate in data transmission.**

***Index Terms*—zero-trust, IoT-Edge identity management, workload authentication, SPIFFE/SPIRE unique identity**

I. Introduction

Expanded edge computing infrastructure and geo-distributed installation improve data governance and performance, but they also bring up new security and privacy issues [1]. While the services and functionalities provided by edge and cloud computing are similar, the logical components of an edge computing stack – such as device hardware, firmware and system, network and communication, cloud, and edge stack (like Kubedge [2]) – have some distinct features. Edge devices are less capable of processing computation intensive tasks and have less resilient security measures. Furthermore, because many edge devices are connected to one another, a single intrusion may have a greater effect if the attack spreads to other devices. Extreme issues arise from the heterogeneity of devices, which is caused by completely distinct hardware and software stacks. The risk of widespread and successful exploitation becomes even greater by the heterogeneous and proprietary nature of hardware and firmware in edge computing. Although there has been significant research on zero-trust architectures [3] on managing cryptographic keys and digital certificates at the device level and on lightweight quantum-resilient security techniques [4], end-to-end authentication and authorization across cloud, leaf edge, and Internet of Things (IoT) devices is not fully covered [5] [6] [7].

Conventional cybersecurity depends on secure network perimeters (through employing firewalls and VPNs) and presumes that everything within is trustworthy. Upon breach, this "inner circle of entities" becomes susceptible to lateral expansion and insider attacks. It has become more challenging to identify a service based only on its location (such as its IP address or subnet) due to the quickly gaining popularity of distributed system architectures that break down functionality into microservices, use containers for portability, orchestrate containers with Kubernetes, and move workloads to the cloud. Moreover, end-to-end identification remains an open issue, particularly in Kubernetes setups with several clusters. One unresolved issue, for example, is how to guarantee a secure chain of identification between the service's launch and everything it connects to and communicates with, both on and off cluster. The solution to these challenges is the architecture offered by SPIFFE (safe Production Identity architecture For Everyone) [8], which obtains secure identities from a central server in order to achieve *Zero Trust* [9] security for workloads. The Zero Trust security model addresses the limitations of perimeter-based approaches by defining the need for explicit authentication controls enforced in each service.

A. Challenges in Kubernetes Security Management

In private and public cloud infrastructures, Kubernetes generates an auto-scaling, dynamic virtual architecture. We have an ephemeral environment that differs greatly from standard servers, with everything spinning up and down continuously dependent on demand. In this scope, special security problems arise. Traditional cloud security posture management (CSPM) technologies, i.e., monitoring cloud-based systems and infrastructures for risks and misconfigurations, just aren't made to manage the subtleties of the Kubernetes control plane because of how ephemeral and transitory Kubernetes is. They are unable to keep up with workloads that change every minute, network rules, or secret management.

Kubernetes today (as of v1.25) recommends that users use the Pod security standards and Pod security admission controllers [10]. These admission controllers provide flexibility and allow for the configuration of a Pod's security context. A Pod security context can now be used to set various security parameters

such as access control and network policies. Kubernetes uses the concept of ClusterRoles and Roles, which specify what each user can perform in a cluster or an entire Kubernetes namespace. You can use ClusterRoleBinding to apply a role to all resources in the cluster, or RoleBinding to apply the role to every resource in a namespace. Regarding the Kubernetes API authentication, the primary access point for a Kubernetes cluster is the Kubernetes API. Users and service roles access it via the kubectl utility, direct REST API requests, or client SDKs.

B. SPIFFE Goals

With the establishment of an open, unified workload identity standard based on the idea of zero-trust, SPIFFE's (Secure Production Identity Framework for Everyone) objective is to build a zero-trust, fully-identified data center network. It is possible for SPIRE (SPIFFE Runtime Environment) to regularly cycle secret keys and X.509 SVID (SPIFFE Verifiable Identity Document) certificates. Workload certificates can be dynamically provisioned by SPIRE based on policies provided by operators.

Workload identification is provided by SPIRE, independent of the workload's deployment location. DevSecOps[1] may provide more detailed identity based on elements beyond those used by Kubernetes, including host machine, hardware, and node characteristics, as well as environment metadata, including cloud provider, region, and network configurations. Moreover, federation across trust domains is enabled via SPIRE. DevSecOps may utilize SPIRE, for instance, to securely authenticate workloads from clusters with varying trustDomain values. SPIRE can federate communication between trust domains by establishing trust through the usage of trust bundles.

In this work, we argue that zero trust assesses all entities, workloads, and IoT device properties collectively, rather than protecting the network perimeter, and then permits or prohibits these entities from connecting with the network or with one another. This method works with any platform and in any setting. In section II we present related trust enabling frameworks. Next section III demonstrates our developed Spiffe-based infrastructure for IoT-Edge environments, while section IV presents our method for trusted IoT device provisioning and finally, we conclude with section V.

II. Related Work

In response to the growing attention that businesses are giving to admission policies, Kyverno – a dynamic admission controller that operates within a Kubernetes cluster – has surfaced [11]. The Kubernetes API server sends HTTP callbacks for verifying and altering admission webhooks to Kyverno, which then applies matching policies to produce results that either refuse requests or enforce admission policies.

[1]DevSecOps helps ensure that security is addressed as part of all DevOps practices by integrating security practices and automatically generating security and compliance artifacts throughout the process; [https://csrc.nist.gov/Projects/devsecops]

Core Linux functionality is crucial to Kubernetes and the technologies it interacts with. Among these is the Extended Berkeley Packet Filter (eBPF), which is becoming more and more used in tools for networking, security and auditing, and tracing and monitoring. Falco, a Kubernetes runtime security tool [12], and Cilium [13], which offers, protects, and monitors network connection across container workloads, are two of the several projects that use eBPF.

Using a layer seven service interconnect, Skupper leverages namespaces in one or more Kube clusters to build a virtual application network [14]. Without the need for VPNs or unique firewall rules, it permits safe communication across Kubernetes clusters. In a service network, every site has its own private certificate authority (CA) and Skupper router. Mutual TLS secures communication between sites, isolating the service network from outside access and avoiding security threats including malware infections, lateral assaults, and data exfiltration.

A growing body of research is being done on ways to ensure that critical applications can trust the functions they rely on to ensure the desired behavior. For trust verification for the Function as a Service (FaaS) model, proposals increase to enable function invokers to confirm the reliability of functions [15]. Alternative methods offer confidential computing for container workloads by fusing the robust security assurances of hardware-enforced Trusted Execution Environments (TEEs) with the manageability and agility of ordinary containers [16].

A. Identity and Access Management (IAM)

There are different ways that prominent public cloud systems support workload identification. To enable workloads to authenticate to other Google Cloud services, for example, Google Kubernetes Engine (GKE) leverages its *Workload Identity* feature. Both Azure and AWS provide Managed Identities for Azure resources and IAM roles for Amazon EC2, respectively, with comparable features.

Users are restricted to the native services and environments of each cloud provider, even if these cloud environments might offer comprehensive IAM features. For example, AWS IAM may control access to AWS services, but not to Azure or SaaS application services. IAM complexity is not uncommon even inside the same cloud environment. One example is the federation between Amazon EKS (Elastic Kubernetes Service) and AWS IAM. Using OpenID Connect (OIDC) for identity federation, this integration enables EKS workloads to take on an IAM role for gaining access to other AWS services using a feature known as IAM Roles for Service Accounts (IRSA).

B. SPIFFE, SPIRE Preliminaries

Within the context of workload IAM, it is critical to understand the differences between authorization, access rules, and authentication. The main objective of workload IAM is to create and enforce access policies, which act as a link between the identity (authentication) and permissions (authorization) of a client workload with regard to a server workload. Establishing trust across various resources and microservices in

the runtime environment depends on the issuing of workload identities. The organization's trust system may confirm that only authorized entities are accessing resources by assigning distinct identities to every task.

It's notably hard to securely bootstrap identity and trust in a distributed system, particularly at the lowest or starting level when no pre-existing trust exists, which is known as the "bottom turtle problem". SPIFFE resolves this challenge of initially establishing trusted identities by ensuring that services can build trust and securely interact with each other from the moment of deployment, even in settings like Kubernetes where services are generated and terminated on a continuous basis.

Moreover, based on SPIFFE, organizations can gain from smooth integration with platform-specific identity management systems by utilizing a workload identity issuer that is independent of the various runtime environments (e.g., Kubernetes, Azure, Google Cloud). The automation of credential maintenance and rotation for each task is provided by the workload identity issuer. By doing this, the danger of credential theft or misuse is reduced and credentials are consistently updated. By relieving administrators of the tedious chore of rotating credentials by hand, automated credential management also lessens their workload.

III. IoT Edge Architecture

Given the rapid adoption of KubeEdge in cloud-edge synergy in industries, transportation, energy, manufacturing, and smart cities, security has become increasingly important to the KubeEdge community. The integration of zero trust and compliance in workload identity management and resources identity management, such as IoT devices, plays a crucial role in the kubernetes architectural framework achieved in KubeEdge, consolidating the countermeasures proposed in [2].

SPIFFE and SPIRE can ensure that only verified edge nodes can participate in data transmission, which is essential for maintaining data security industrial IoT setups. Furthermore, For each service communication, rotation policies may be set and short-lived certificates are created. For the setup and maintenance of certificate rotation, there is no requirement for specialized agents or dependency on certain orchestrators. Automated deployment of non-root Certificate Authority (CA) certificate hierarchical architectures: Edge Spire servers have the option to be set up such that no root CA chain is shared with downstream nodes or workloads.

A. Edge Architecture Overview

Edge architecture integrating Leaf Edge Devices (LEDs), as figure 1 shows, consists of several components that work together to enable the deployment and management of containerized applications on edge devices, as well as the sharing and grouping of IoT resources among these applications.

By extending KubeEdge architecture, Figure 1 represents the trusted architecture for managing IoT edge devices. The main cloud-side components interact with EdgeCore component via Kubernetes key value store, usually etcd, using

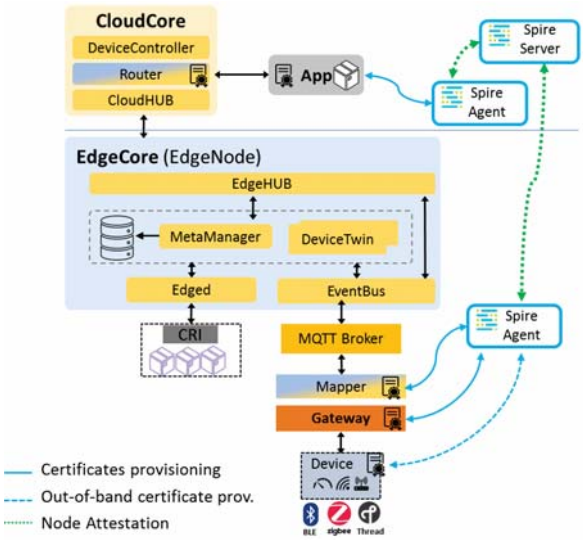

Fig. 1. Edge architecture integrating IAM framework

the Kubernetes API to manage devices and data in a cloud native way. Specifically, the DeviceController is the cloud component responsible for device management. By using the Kubernetes Custom Resource Definitions (CRDs), it describes device metadata/status and synchronizes these device updates between edge and cloud. The CloudHub component directly interacts with the edge counterpart and supports both web-socket based connection as well as a QUIC protocol access at the same time. Through two-way communication, CloudCore monitors Kubernetes resource changes and sends info to the edge. Through this channel, EdgeCore uploads the metadata to CloudCore, including edge node status and application status. The received metadata are then reported to the kube-apiserver by CloudCore.

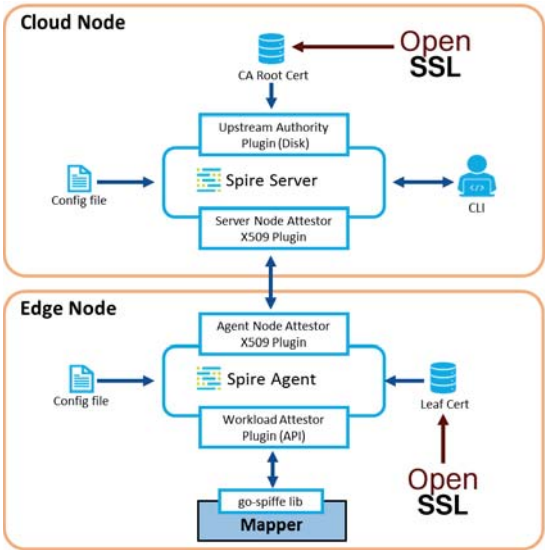

Fig. 2. Spire Server and Agent realization at the cloud and edge (including Mapper certificate installment)

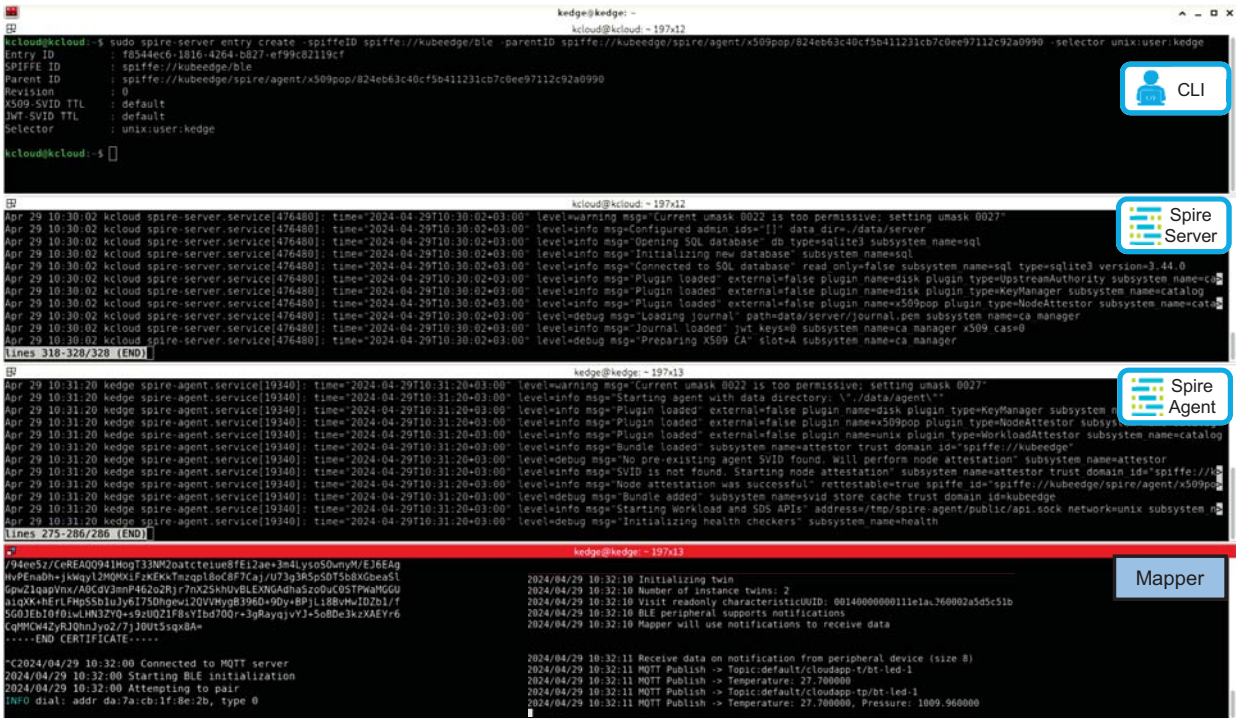

Fig. 3. Spire server and agent initialization, certificate provisioning and attestation chain at the cloud and edge. After Mapper component is successfully attested and receives X509 certificate, Mapper proceeds with a successful pairing with the BLE-enabled device to capture temperature and pressure sensors data to MQTT broker.

The previous two components, along with the Router which sets the appropriate traffic paths, constitute the CloudCore. The Router (which is already present in the KubeEdge framework) is enhanced to support the features described in the previous section, e.g., support multicasting routes of LED generated traffic. The combination of the three different logics provides the available information to the Resource Acquisition Manager to configure the resource allocation.

The EdgeCore counterpart comprises a set of components that enables respectively Pod lifecycle management (Edged), sending/ receiving messages on MQTT topics (EventBus), interacting with cloud-side components (EdgeHub), device management (DeviceTwin), message processing between Edged and EdgeHub and storing/ retrieving metadata related the Pods (MetaManager).

As shown in Figure 1, the EdgeCore communicates with the Mapper, which is an agent component (edge side) responsible to interface with the leaf devices. By implementing connection protocols (e.g., Bluetooth, ZigBee, Modbus, OPC UA) supported by leaf devices, it creates a physical connection between edge and an IoT device. The Mapper acts as an asynchronous message broker by using MQTT. The existing Mapper available in KubeEdge supports basic technologies such as Zigbee and Bluetooth. It faces several limitations, such as the missing support for multiple upstream data paths and downstream control paths down to a leaf device. Most important though, all these EdgeCore components lack authentication and authorization features.

B. Spire Implementation

This section describes the development of Spire components based on our Edge design shown in Figure 1.

The workflow to configure an IAM setup involves the following procedure, which is depicted in Figure 2. First, the certificate chain begins with the creation of a self-signed Certificate Authority (CA) root certificate, by using Open SSL, for the Spire Server, and then, we create a leaf certificate for the Spire Agent signed by the CA. Then, in the Spire Server, based on config file, we enable the *Upstream Authority plugin* to use the self-signed CA as the CA authority; then we enable the *Node Attestor plugin* to accept attestation of the Spire Agent.

In the Edge Node, we activate the Node Attestor plugin on the Spire Agent side so that it may use the leaf certificate to attest to the Spire Server. The Workload Attestor plugin is also enabled, which initiates the workload API and waits for workload attestation. Finally, at the Mapper, we integrate the go-spiffe library to access the Workload API and attest itself and additionally, it receives the certificate from the Spire Agent after successful attestation.

Figure 3 demonstrates the full infrastructure provisioning process that successfully realizes the certificate distribution for the BLE-aware Mapper component. Unless the Mapper entity is attested by the Spire agent, the Mapper is not allowed

to proceed with the pairing of the BLE-enabled device. The Mapper on the bottom left prints the X509 certificate received after successful attestation and on the right, the next step, the pairing process is depicted.

IV. IoT Device Identity Provisioning

The objective is to ensure trustworthy components and connections between all IoT entities that are engaged. Therefore, we guarantee both the uniqueness of the device itself and the legitimacy of the IoT device assigned to the Mapper.

Figure 4 shows the extensions to achieve IoT device certificate installment through the Spiffe Helper utility and fetching X.509 SVID certificates from the Spiffe workload API in order to get the certificate for the STM32 microcontroller device; in our implementation the STWIN SensorTile wireless industrial node (STEVAL-STWINKT1B) is used, which is powered by an ARM Cortex-M4 MCU.

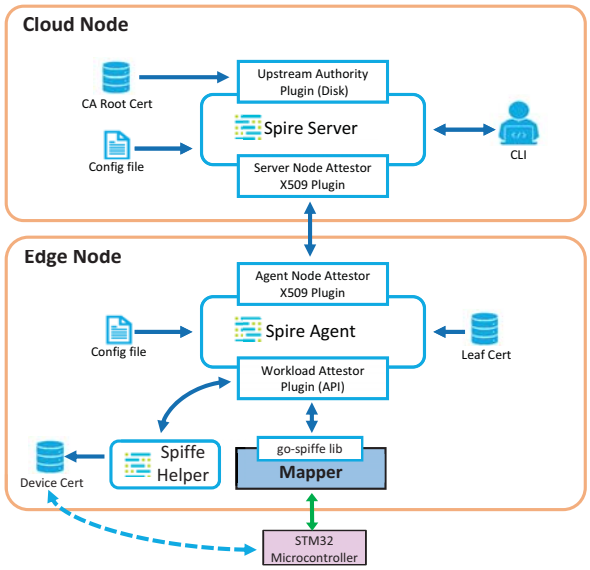

Fig. 4. Full system realization includeing Spire server and agent at the cloud and edge and IoT device certificate installment

Establishing a secure BLE connection with the IoT device is necessary when the Mapper is confirmed and granted its certificate by the Spire Agent. Utilizing "LE Secure Connections" which uses a Diffie-Hellman handshake to create a secure channel of communication by exchanging a Long-Term Key (LTK), is the most appropriate security solution among the numerous provided by the BLE standard [17]. Developments allowed the communication through SPI between the MCU and the BLE module through which the latter was configured to enable security, and thus to transfer the Spire certificate to the MCU. The Diffie-Hellman handshake requires a keypair, which the BLE module can only provide on its own. Future plans call for changing the BLE Module's source code to enable the use of custom credentials and session-specific SPIFFE certificates.

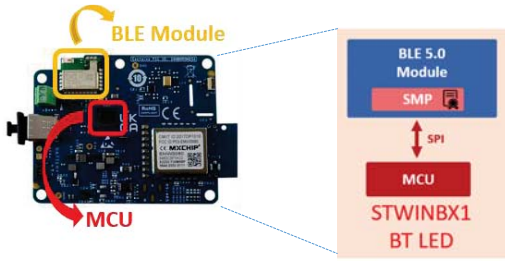

Fig. 5. Layout of the MCU and the Bluetooth module of the STEVAL-STWINBX1 device

A. IoT Device Authentication

The first step in determining an IoT device's trustworthiness is to verify that the secure storage–that is, the secure slots inside the STSAFE-A110 component that is integrated with the device–is legitimate.

The device secure bootloader uses the STM root certificate which enfolds the public key that can be used to verify the STSAFE-A110 leaf certificate. As shown in the process in Figure 6, upon successful verification the secure element is trustworthy. This certificate is available to the bootloader and is stored in the STSAFE-A110 secure data partitions.

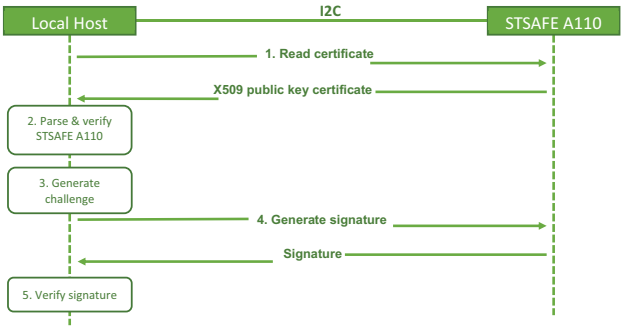

Fig. 6. MCU authenticating the certificate inside the STSAFEA110)

An additional procedure to establish secure connection with a remote host might be adopted [18], however we followed a homogeneous incremental strategy, by employing the SPIRE procedure. Therefore, we only use the STSAFE-A110 leaf certificate to verify the authenticity of the device and then the SPIRE certificate to establish a secure network connection. Figure 7 shows the validation of signature checking of the private STSAFEA110 leaf certificate, after successful provisioning.

V. Conclusions

Many edge devices and Internet of Things devices are underpowered to execute sophisticated authentication algorithms. Nevertheless in an IoT environment, enhanced trustworthiness with validated identities is essential for guaranteeing the integrity and reliability of linked devices. In this study, we incorporated Spiffe and Spire technologies into the Kubedge framework to support the establishment, management, and

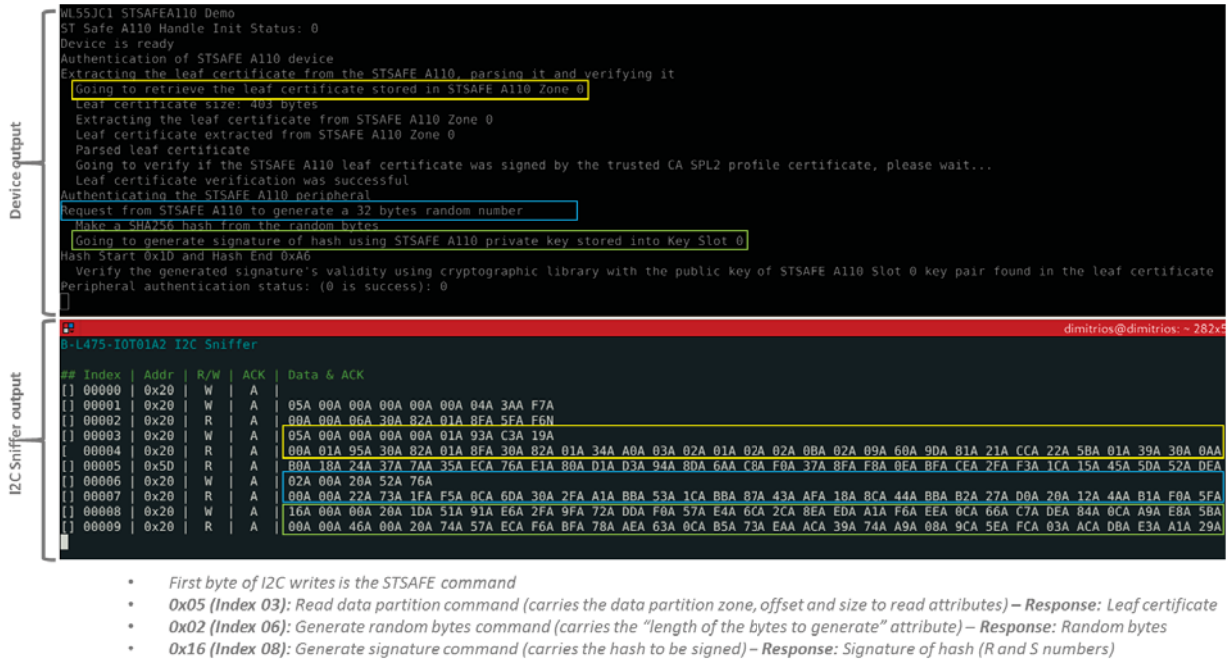

Fig. 7. Experimental validation of STSAFEA110 certificate verification

identity verification of entities in the distributed Kubedge environment, particularly in Edge IoT devices. With an emphasis on trustworthy identity issuance, attestation, and maintenance, Spire offers a standardized architecture for workload identification. By using the Spiffe Helper plugin, we are moving forward with extending certificate establishment to Internet of Things microcontroller devices. Furthermore, we integrated an advanced secure element technology that can be activated, personalized and managed remotely to make it easy to run differentiated and loyalty-building services with absolute confidence.

Acknowledgment

This work was supported in part by the EU Horizon 2020 Project FLUIDOS under GA No. 101070473.

References

[1] Y. Xiao, Y. Jia, C. Liu, X. Cheng, J. Yu, and W. Lv, "Edge computing security: State of the art and challenges," *Proceedings of the IEEE*, vol. 107, no. 8, pp. 1608–1631, 2019.

[2] KubeEdge, "Threat Model and Security Protection Analysis," https://github.com/kubeedge/community/blob/master/sig-security/sig-security-audit/KubeEdge-threat-model-and-security-protection-analysis.md, online; accessed 1 Dec 2023.

[3] N. F. Syed, S. W. Shah, A. Shaghaghi, A. Anwar, Z. Baig, and R. Doss, "Zero trust architecture (ZTA): A comprehensive survey," *IEEE Access*, vol. 10, pp. 57 143–57 179, 2022.

[4] G. Kornaros, G. Berki, and M. Grammatikakis, "Quantum-secure communication for trusted edge computing with IoT devices," in *ICT Systems Security and Privacy Protection*, N. Meyer and A. Grocholewska-Czuryło, Eds. Cham: Springer Nature Switzerland, 2024, pp. 163–176.

[5] G. Kornaros, "Hardware-assisted machine learning in resource-constrained iot environments for security: Review and future prospective," *IEEE Access*, vol. 10, pp. 58 603–58 622, 2022.

[6] B. Cremonezi, A. B. Vieira, J. Nacif, E. F. Silva, and M. Nogueira, "Identity management for internet of things: Concepts, challenges and opportunities," *Computer Communications*, vol. 224, pp. 72–94, 2024.

[7] L. Fotia, F. Delicato, and G. Fortino, "Trust in edge-based internet of things architectures: State of the art and research challenges," *ACM Comput. Surv.*, vol. 55, no. 9, jan 2023.

[8] SPIFFE, "The Secure Production Identity Framework For Everyone (SPIFFE) Project," https://spiffe.io, online; accessed 1 Sep 2023.

[9] Scott Rose, Oliver Borchert, Stu Mitchell and Sean Connelly, "Zero Trust Architecture, NIST Special Publication 800-207," https://nvlpubs.nist.gov/nistpubs/SpecialPublications/NIST.SP.800-207.pdf, online; accessed 1 Mar 2024.

[10] Kubernetes, "Security," https://kubernetes.io/docs/tutorials/security, online; accessed 1 Dec 2023.

[11] Kyverno, "Kubernetes Native Policy Management," https://kyverno.io, online; accessed 1 Mar 2024.

[12] The Falco Project, https://falco.org, online; accessed 1 Feb 2024.

[13] Cilium, "eBPF-based Networking, Observability, Security," https://cilium.io, online; accessed 1 Mar 2024.

[14] Skupper, "Multicloud communication for Kubernetes," https://skupper.io, online; accessed 1 Mar 2024.

[15] A. Shamendra, B. Peries, G. Seneviratne, and S. Rathnayake, "Trufaas - trust verification framework for faas," in *Ubiquitous Security*, G. Wang, H. Wang, G. Min, N. Georgalas, and W. Meng, Eds. Singapore: Springer Nature Singapore, 2024, pp. 304–318.

[16] F. Brasser, P. Jauernig, F. Pustelnik, A.-R. Sadeghi, and E. Stapf, "Trusted container extensions for container-based confidential computing," 2022. [Online]. Available: https://arxiv.org/abs/2205.05747

[17] ST, "STM32WB-WBA Bluetooth LE Security," https://wiki.st.com/stm32mcu/wiki/Connectivity:STM32WB-WBA_BLE_security, 2024.

[18] ST life.augmented, "STSAFE-A110, Authentication, state-of-the-art security for peripherals and IoT devices," https://www.st.com/en/secure-mcus/stsafe-a110.html, dS13039 Rev 1, Online; accessed 1 Feb 2024.