Article

Not peer-reviewed version

Secure TinyML on Edge Devices: A Lightweight Dual Attestation Mechanism for Machine Learning

Vlad-Eusebiu Baciu [*] , An Braeken , Laurent Segers , Bruno da Silva [*]

Posted Date: 26 December 2024

doi: 10.20944/preprints202412.2251.v1

Keywords: Edge Devices; Remote Attestation; TinyML; Edge AI; IoT Security; Federated Learning

Article

Secure TinyML on Edge Devices: A Lightweight Dual Attestation Mechanism for Machine Learning

Vlad-Eusebiu Baciu[1], An Braeken[1,2], Laurent Segers[1] and Bruno da Silva[1]

[1] Department of Electronics and Informatics (ETRO), Vrije Universiteit Brussel (VUB), 1050 Brussels, Belgium
[2] Department of Engineering Technology (INDI), Vrije Universiteit Brussel (VUB), 1050 Brussels, Belgium
* Correspondence: vlad-eusebiu.baciu@vub.be (V.E.B.); an.braeken@vub.be (A.B.); bruno.da.silva@vub.be (B.d.S.)

Abstract: Emerging edge devices are transforming the Internet of Things (IoT) by enabling more responsive and efficient interactions between physical objects and digital networks. These devices support diverse applications, from health-monitoring wearables to environmental sensors, by moving data processing closer to the source. Traditional IoT systems relied heavily on centralized servers but advances in edge computing and Tiny Machine Learning (TinyML) now allow for on-device processing, enhancing battery efficiency and reducing latency. While this shift improves privacy, the distributed nature of edge devices introduces new security challenges, particularly regarding TinyML models, which are designed for low-power environments and may be vulnerable to tampering or unauthorized access. Since other IoT entities depend on the data generated by these models, ensuring trust in the devices is essential. To address this, we propose a lightweight dual attestation mechanism utilizing Entity Attestation Tokens (EATs) to validate the device and ML model integrity. This approach enhances security by enabling verified device-to-device communication, supports seamless integration with secure cloud services, and allows for flexible, authorized ML model updates, meeting modern IoT systems' scalability and compliance needs.

Keywords: Edge devices; remote attestation; IoT security; TinyML; Edge AI; federated learning

1. Introduction

Embedded edge devices play a vital role in daily life, supporting tasks like environmental monitoring, remote appliance control, security, health tracking, and smart home automation with minimal user involvement. Advances in CPU architectures, hardware design, and artificial intelligence (AI) have enhanced TinyML (Tiny Machine Learning), enabling resource-limited devices such as microcontrollers to perform machine learning (ML) inference at the network edge. By monitoring parameters like temperature, humidity, and sensors, these devices facilitate applications such as fault detection in wind turbines [1] and gas leak detection [2] using ML algorithms.

Edge devices running unprotected ML models pose risks to systems that rely on their prediction outputs. TinyML systems are particularly vulnerable to attacks like model flashing and malicious updates, while advanced threats such as model inversion are less common due to resource limitations [3]. To address these challenges, stakeholders are improving security standards. The Confidential Computing Consortium defines confidential computing as using hardware-based Trusted Execution Environments (TEEs) to protect data during use, ensuring confidentiality, data integrity, and code integrity [4].

Resource-constrained devices increasingly incorporate features like tamper protection, true random number generators, secure storage, secure boot, and roots of trust, which are already utilized in low-power applications such as environmental monitoring [5]. Hardware-based TEEs leverage these technologies to provide secure enclaves for sensitive computations, ensuring protection even in the presence of physical attacks or compromised software. System-on-chip (SoC) vendors implement various isolation mechanisms, such as CPU access isolation or encrypted memory isolation, with solutions like Intel SGX [6] and AMD SEV [7] targeting cloud-based applications, while ARM TrustZone [8] and

RISC-V PMP [9] are optimized for edge devices, providing a balance between security and energy efficiency.

Additionally, device attestation, when implemented within a TEE, offers a strong security solution [10]. Attestation verifies the authenticity and integrity of the TEE, ensuring that sensitive computations occur only on trusted devices. The Remote Attestation Procedures (RATS) defined by the Internet Engineering Task Force (IETF) [11] provide a standardized approach to validate the integrity and authenticity of remote devices. RATS specifies components such as the Attester, Verifier, and Relying Party, with attestation tokens serving as authenticated claims about the device's state.

Several open-source TEE solutions, such as the ARM Confidential Compute Architecture (CCA) [12] and TrustedFirmware-M [13], align with the Platform Security Architecture (PSA) guidelines, offering well-defined structures for attestation tokens. Vendor-specific solutions such as Silicon Labs' Secure Engine Manager [14] and STM Microelectronics' Secure Manager [15] generate PSA attestation tokens signed with hardware-specific keys.

In addition to the TEE attestation, a verifier may also need to prove the authenticity of the ML model running on the device. For example, to ensure the system operates correctly and securely [16], or to confirm that a device is capable of participating in collaborative training, such as in federated learning, and can safely exchange trained model weights [17]. However, to support standardized attestation for TinyML models, ML attestation tokens should be independent of the TEE attestation process. This separation ensures that TEE tokens focus solely on verifying the hardware platform, while the trustworthiness of the ML model is attested individually.

Figure 1 illustrates a conceptual data flow of device attestation and the actors involved, based on the RATS specification. The SoC and device manufacturers securely provision cryptographic keys and other artifacts (1), establish attestation policies, and define claims, which are then stored in a shared database accessible to an authorized verifier (2). The ability to update ML models may vary by device type, with manufacturers potentially restricting updates or allowing user-initiated updates (3).

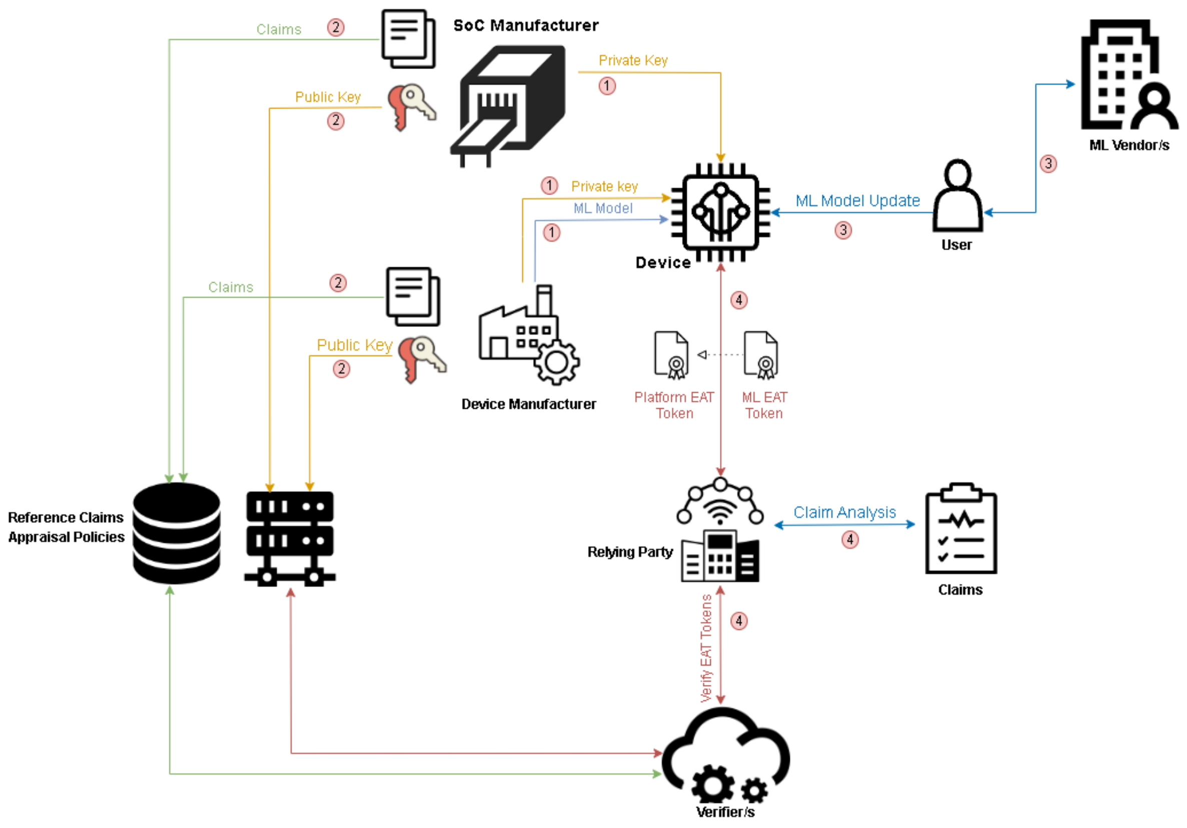

Figure 1. Device attestation and involved entities: Soc Manufacturer, Device Manufacturer, Relying Party, Verifier, User.

A relying party initiates the device attestation process (4) and relies on a trusted verifier to assess the authenticity and integrity of the tokens. Depending on the architecture, the relying party may also analyze the token claims and decide if the device is trustworthy, such as verifying whether the ML model meets specific criteria or if the device's state aligns with certain security requirements.

This paper presents a dual attestation mechanism leveraging the concept of Entity Attestation Tokens (EAT) [18] and CBOR Object Signing and Encryption (COSE) [19] encoding. The approach utilizes two complementary tokens: one attesting to platform integrity via PSA initial attestation and another verifying the ML model's integrity and parameters.

By decoupling the ML model attestation from the TEE, the system enhances resource efficiency and flexibility. The separation allows for more efficient use of computational resources and greater adaptability in managing various components, thereby fostering trust among different entities and networks.

Existing attestation tokens primarily focus on platform integrity, leaving a gap in verifying the trustworthiness of ML models. This limitation is particularly evident in applications like federated learning systems with edge devices, where ensuring the integrity of the ML model is crucial. Additionally, allowing an edge device to join a network based on the model it uses requires a robust mechanism to attest to the model's authenticity and behavior.

To address these challenges, we propose a set of claims and a token structure specifically designed for ML models. This approach enhances trustworthiness across various applications by providing a standardized method to attest to the integrity and behavior of ML models.

We implement a prototype that provides an end-to-end security solution, leveraging a combination of RoT, secure boot, trusted execution environment, PSA initial attestation, and the proposed ML-EAT attestation. Deployed on an STM32H573I development board, the prototype is evaluated in terms of memory footprint, scalability, and attestation time to assess its effectiveness in ensuring both platform and ML model integrity, establishing trust across the system.

The main contributions of this work are:

- A novel, lightweight dual attestation mechanism that validates the integrity of both the device and the ML model, facilitating trust verification by a remote entity.
- Propose a new ML attestation token format based on EAT, containing claims that attest to both general ML use cases and sensitive ML parameters, such as architecture and weights.

The paper is organized as follows: The related work is presented in Section 2. In Section 3 the background is provided while in 4 our proposed attestation method for ML and its workflow is described. The overview of our methodology for the assessment is given in Section 5. The results and discussion are performed in Section 6. Finally, the conclusion is drawn in Section 7.

2. Related Work

Embedded devices that handle privacy-sensitive data or perform safety-critical functions are increasingly becoming prime targets for cybersecurity threats, such as denial-of-service (DoS) attacks, unauthorized network access, firmware tampering, and more [20]. These devices often run embedded firmware from multiple vendors, meaning that a single vulnerability in one vendor's software can potentially affect a large number of devices that depend on the same component [21].

Nowadays, there is an increasing focus on detecting compromised devices before they can affect the functionality of connected systems. Remote attestation is one method used to ensure device integrity. Depending on the system architecture and security goals, various attestation approaches can be employed. These include single attestation, which verifies the integrity of individual devices, and collective attestation protocols, where multiple devices work together to validate the security of the entire network [22].

Large-scale IoT networks require collective attestation methods to effectively ensure the security of the entire network. One such example is PASTA [23], a fully decentralized attestation mechanism based on Schnorr-based multisignatures. It supports multiple provers and verifiers that can attest their integrity using lightweight tokens, making it ideal for resource-constrained embedded devices with limited computational capabilities.

Other collective attestation protocols, such as SEDA [16], use a spanning tree topology where each parent node aggregates the attestation results from its child nodes. The underlying protection mechanisms of the edge devices rely on SMART [24], or TrustLite [25] architectures. Both TEE architectures provide remote attestation for low-end embedded systems by leveraging custom hardware memory protection units and carefully designed software. However, their disadvantage is that they require hardware modifications to be integrated into current CPU architectures, which poses a challenge for widespread adoption.

Currently, collective attestation protocols lack the granularity to provide detailed information about individual devices, making them more suitable for homogeneous sensor networks where devices share similar configurations. In TinyML systems, devices may run different ML model versions, software stacks, and have varying requirements that need verification by a remote verifier. It is essential not only to confirm that a device is operational but also to assess its internal state and the state of its ML model. Methods such as single-device attestation are better suited to meet these specific needs.

In [26], the authors present a single device attestation flow based on an off-the-shelf MCU that leverages ARM TrustZone. The mechanism employs a paging-based process integrity measurement in a Linux-based IoT environment. It also relies on a ROM-based RoT to securely store and manage cryptographic keys, providing a tamper-resistant foundation for the attestation process. The attestation mechanism of the device relies on a simple hash value that is encrypted and sent to a remote server for verification.

Relying solely on a hash value for attestation is inadequate, as it does not enable structured, interoperable communication of evidence. Different attestation models have been proposed for resource-constrained devices, where efficiency is crucial. Property-based attestation, utilizing structured claims such as those in PSA or EAT, enables more effective and scalable communication of trust, allowing relying parties to easily verify the security and integrity of devices or systems [27].

The Veraison project [28] seeks to establish standardization and interoperability across various remote attestation frameworks. It contains a set of software components that can be employed to construct attestation verification services that are common across different classes of devices, from IoT devices to large computing servers. It is based on the RATS architecture proposed by the IETF, focusing on configuring the verifier with provisioning endorsements and trust anchors such as keys and certificates. Further, the Verifier is capable of validating evidence or attestation results using appraisal policies. It supports attestation for both the ARM CCA on the cca-ssd platform and the ARM PSA on the psa-iot platform.

There is a need for TinyML devices to provide claims and follow the RATS format to enhance trust and security in embedded environments. Operating in resource-constrained, dynamic, and heterogeneous settings with models from various vendors, TinyML systems can use claims to communicate detailed, structured evidence of their state. This allows verifiers to assess their trustworthiness effectively. Verifiers can query the device state to extract evidence such as model parameters, firmware versions, or runtime behavior, and link this data with reference values or appraisal policies.

In this work, we introduce an ML attestation mechanism designed for edge devices and TinyML applications, integrating the PSA initial attestation token with a specialized ML attestation token (ML-EAT). This approach provides an end-to-end solution for systems where machine learning models need to be remotely verified, updated, and maintained, ensuring both platform integrity and model authenticity. The PSA token confirms platform integrity, while the ML-EAT token attests to the correctness of the machine learning model running on the device. To the best of our knowledge, this is the first solution specifically tailored for TinyML, offering an effective and scalable method to secure and attest both the platform and ML models in resource-constrained environments.

3. Background

3.1. Platform Security Architecture

Platform Security Architecture (PSA) is a security framework developed by ARM that became an industry standard for embedded devices. PSA provides a set of guidelines and best practices for designing secure hardware and software environments. It includes features like secure boot, hardware isolation, TEE, and secure communication protocols to protect devices, data, and communications.

Secure boot ensures that only authenticated software can run on the device, protecting it against unauthorized or tampered code during the boot process. The entire chain of trust is built on a RoT component, which is inherently trusted and executed after a system reset. It relies on hardware elements like eFuses, ROM, or protected flash memory, which are immutable once programmed. For example, the hash of the verification public key can be stored in one-time programmable eFuses. The boot sequence may involve multiple stages, where each firmware image is responsible for verifying the integrity of the next image in the chain before it is allowed to execute. The hash of the public key used to verify a specific image is stored in the preceding boot stage. Therefore, any attempt to compromise the chain of trust would require altering the RoT public key hash, which is central to the verification process.

Hardware isolation utilizes CPU addressability isolation to protect specific registers and memory regions. ARM architectures incorporate the TrustZone security extension, which introduces the concept of secure and non-secure environments through a new CPU state bit. This allows or denies transactions initiated by the core based on the execution context and the security attributes of the target address. Besides, all slaves connected to the bus are either TrustZone-aware or are protected by a security firewall. This new privilege level is independent of the original exception levels (EL 0-2) and operates orthogonally to them. Additionally, a Security Monitor at EL3 may be present in some architectures to manage the software transition between the two trust domains.

Trusted Execution Environment is generally implemented using both hardware and software techniques. It creates an isolated execution context in which untrusted applications cannot access sensitive data or operations. This environment leverages hardware features such as secure memory regions, cryptographic keys, and secure boot mechanisms to ensure that code and data remain protected. The TEE ensures that even if the underlying system is compromised, the execution of sensitive code remains secure, providing integrity, confidentiality, and isolation for critical operations.

Attestation serves as a critical mechanism for verifying a device's trustworthiness, offering essential insights to cloud service providers or peer nodes. It enables informed decisions on whether to trust a device and allow its interaction within a secure system. Built on EAT, attestation relies on signed claims to provide key information, such as a device's security configuration, software version, and identity. The flexible format of EAT tokens also allows for the inclusion of custom claims tailored to specific use cases, such as runtime integrity verification or compliance with regional regulations. By refreshing attestation tokens periodically, the process supports continuous monitoring, ensuring that devices remain secure and compliant over time.

3.2. Attestation tokens

An attestation token is a message containing a claim set about an entity, which can be either a hardware component (e.g., IoT device, network router) or a software component (e.g., TEE, process). It is defined by the RATS Working Group [29], which specifies common claims that an EAT token can include, with potential broad applicability. Additionally, they describe EAT as a framework for defining attestation tokens tailored to specific use cases, facilitating further standardization of proprietary claims.

The EAT specification builds upon CBOR Web Token (CWT) [30] or JSON Web Token (JWT) [31], with an emphasis on claims specifically designed for attestation purposes. For devices with limited resources, CBOR (Concise Binary Object Representation) [32] is the preferred format because it provides a more compact and efficient encoding, reducing both bandwidth usage and processing overhead compared to JSON. This makes CBOR particularly suitable for IoT devices and other constrained environments where minimizing resource consumption is critical.

A claim is represented as a key-value pair, where the key identifies the claim and the value holds its content. Claims can be application-specific or standardized by recognized bodies to ensure interoperability. IANA maintains a global registry of defined claims [33], which can be used in application-specific EATs. If a required claim is not listed, organizations or developers can create their own, deciding whether to keep them private, document them publicly, or pursue standardization. For instance, the IETF defines standardized claims like `iss` (issuer), `sub` (subject), and `exp` (expiration time) in CWT, which are widely used in secure communication protocols.

In the context of PSA Attestation, claims such as `psa_instance_id` (device identifier), `psa_security_lifecycle` (security state), or `psa_implementation_id` (RoT version) are documented and standardized as outlined in the PSA Attestation Token specification [34].

The EAT is created on the device and securely transmitted to the relying party using a COSE-based container [19]. COSE is an open standard designed for signing, encryption, and ensuring data integrity, leveraging CBOR for efficient data encoding. Compared to JSON-based schemes such as JOSE [35], COSE-signed tokens are significantly smaller in size, making them particularly suitable for IoT devices that operate in low-bandwidth environments or where data transmission rates are constrained.

3.3. EAT Token Structure and Encoding

A significant portion of the standardization efforts surrounding attestation tokens has focused on defining the basic structure of the token container, along with establishing interoperable claims and verification mechanisms. A token is generally composed of three main components: the header, the payload, and the signature. Listing 1 depicts a sample attestation token formatted according to COSE specification.

The payload is wrapped inside a container that can be an `COSE_Sign1` or `COSE_Encrypt` object. The protected header contains metadata such as the algorithm used for signing the token or other contextual information such as the reference to the public key. The unprotected header is a component of the message structure that contains metadata not subject to integrity protection such as negotiation parameters or context identifiers. The payload contains a list of claims which contains device attributes such as the device ID or security state.

The claims in the payload can be categorized as follows:

- Standardized claims, where there is widespread agreement on how the information should be structured
- Claims that require standardization to ensure interoperability
- Claims that can be defined by individual parties according to their specific needs

Standardized claims, such as those found in the COSE header, are included in an algorithms registry and are publicly available. Each claim set is associated with a fixed integer key ID (label) that has a clearly defined function. For example, in the COSE Header registry [36], the integer label 1 corresponds to the algorithm used. Additionally, the values for these algorithms are specified by the standard.

The claims that are not standardized are assigned private-usage IANA registry keys, such as values less than -65536. If the token includes more complex structures, such as JSON, the object can be serialized into the token format, where the claim labels can have a byte string format.

```
<< {
  / protected / << {
    / alg / 1: -7  / ECDSA 256 /
  } >>,
  / unprotected / {
    / kid / -75000: h'4173796d6d65747269634543445341323536'
  },
  / payload / << {
    / boot_seed / -75004: h'3d4ffc935d4df450cb...',
    / challenge / -75008: h'2c50207544b0...',
    / client_id / -75001: -1,
    / hardware_id / -75005: "04080401000007",
    / implementation_id / -75003: h'7b37279c6db435063...',
    / instance_id / -75009: h'00506a3491d39ca57a4d9b9...',
    / security_lifecycle / -75002: "SL_NON_PSA_ROT_DEBUG"
  }} >>,
  / signature / -75010: h'5e7a1f2a89b1f39a...'
}} >>
```

Listing 1. A sample Entity Attestation Token and its format.

The entire COSE container, including the protected header, unprotected header, payload, and signature, is serialized using CBOR. CBOR uses different major types for encoding, such as integers, byte strings, text strings, arrays, and maps. In COSE, the headers and payload are typically encoded as maps, with keys and values using various CBOR types like integers or strings.

4. Proposed ML Attestation Method and Workflow

4.1. Dual-Token attestation method

In the context of the attestation of ML models, we propose a dual EAT attestation method, as shown in Figure 2. We decouple the PSA attestation token from the ML-specific claims for the following reasons: the PSA token is typically generated by a secure partition manager component, which is usually provided by the vendor and is not intended for modification. Additionally, the ML-specific claims we propose at this time are not standardized. The PSA token attests to the device's identity and ownership, while the ML-EAT token provides evidence of the ML model and user application.

The tokens are signed with different private keys (ECDSA-256) belonging to distinct entities. The first EAT token is signed with the SoC manufacturer's private keys, attesting to the chip's identity and integrity. The second EAT token is signed with the device manufacturer's keys or other third-party keys, provisioned during manufacturing. The second token is linked to the platform token via its hash (SHA256), which functions as a nonce. Additionally, the second token includes an authorization challenge from the verifier as one of its claims, ensuring freshness and preventing replay attacks.

Furthermore, specific claims in the ML-EAT can be encrypted (AES-CBC-128), ensuring that sensitive information, such as the model architecture or proprietary details, remains confidential. This is particularly beneficial if the device participates in a collaborative federated learning environment, where multiple devices share updates without revealing sensitive data. In this context, the device needs to prove that its model architecture matches the one agreed upon by the federation, ensuring consistency and compatibility during the collaborative model training.

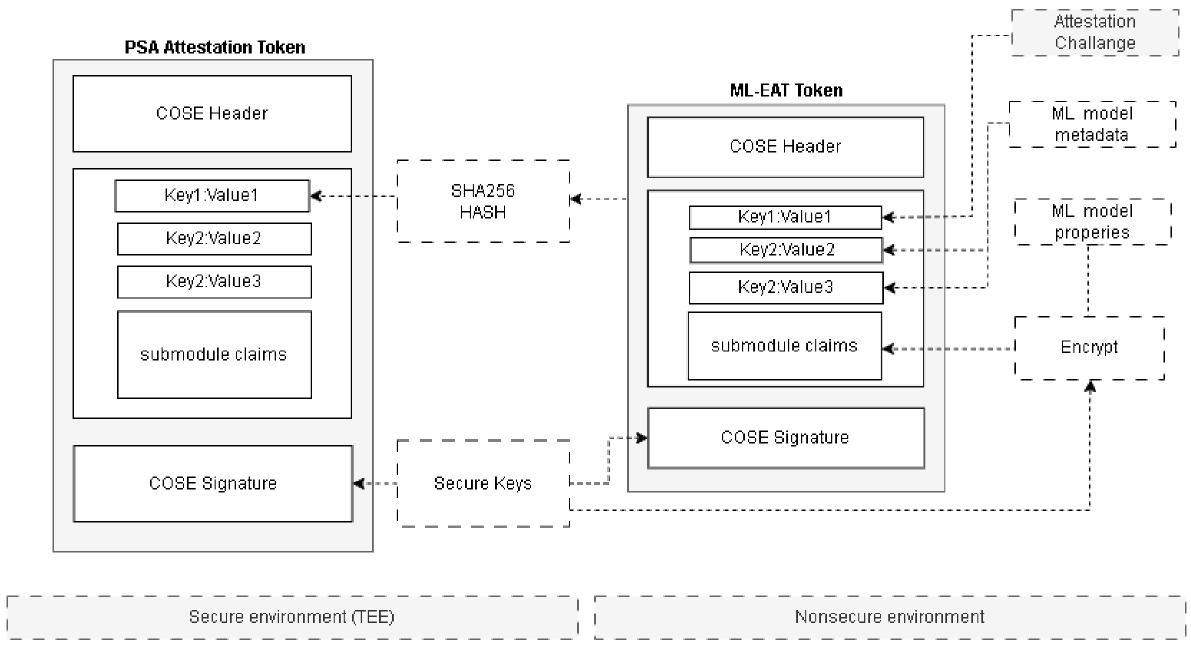

Figure 2. Dual attestation method: ensure device and model trust through PSA and ML-EAT tokens.

4.2. ML Attestation Token

At present, there are no standardized attestation claims for ML models that define a consistent set of key IDs, along with their types and encoding formats, to ensure seamless interoperability between devices and services.

As shown in Figure 2, the proposed ML attestation token includes a collection of general claims that serve as metadata for the model, providing details such as the model version, source, integrity attributes, and other model-specific information.

The submodule claims are represented as a Claim-Set, as defined in the EAT specification. This Claim-Set contains the model architecture, formatted according to the framework used for its creation. The claims are part of the authenticated container and encoded in CBOR format, consistent with the other general claims.

Various ML frameworks offer methods to retrieve a summary of a model's structure and configuration. For example, TensorFlow or Keras provides the `model.summary()` function, which generates a tabular overview of the model's layers, their shapes, and parameters. Similarly, scikit-learn enables access to model properties such as coefficients, intercepts, and hyperparameters. The `get_params()` method in scikit-learn further allows users to obtain the hyperparameters of a model, including regularization strength, the solver used in logistic regression, or the depth of a decision tree. These features provide valuable insights into the model's configuration, facilitating optimization and fine-tuning.

For the general claims, we propose a set of claims outlined in Table 1. For the submodule claims, the process involves creating a human-readable and serializable summary of the model in JSON format, converting it into a Python dictionary for further manipulation or storage, and then using the Python `cbor` package to serialize the dictionary into a compact CBOR representation.

Table 1. ML-EAT token definition. Proposed generic claims and model architecture subclaims. Model architecture claims not shown as are specific the ML model and framework.

CLAIM FIELD	TYPE (CBOR)	DESCRIPTION
Challenge	Byte String (3)	An unique 32-byte value from the verifier to mitigate reply attacks
Model information (Array of maps)		
Model ID	Text String (3)	An unique model identifier (e.g DOI, Name||Creation date)
Model version	Text String (3)	Version of the model (e.g. "1.0.0")
Model publisher	Text String (3)	Publisher or creator of the model
Hash algorithm	Text String (3)	Cryptographic hash (e.g., SHA256) used for the following claims
Model hash	Byte String (2)	Hash of the model weights and nounce H(w||nounce)
Hash of public key	Byte String (2)	Hash of the public key used for secure updates
Training Summary (Array of maps)		
Dataset name	Text String (3)	Dataset name as per the creator or DOI
Dataset identifier	Byte String (2)	Identifier of the dataset used for training (e.g UUID, hash)
Last update	Tag (6)	Standard Date/Time string
Performance (Array of maps)		
Accuracy	Floating-point (7)	Accuracy of the model
F1-Score	Floating-point (7)	F1-Score of the model
SRAM footprint	Unsigned Integer (0)	Memory footprint of the model at runtime
FLASH footprint	Unsigned Integer (0)	Flash memory used by the model
Inference latency	Floating-point (7)	Average inference time on device
Model Parameters (Array of maps)		
Input format	Array (4)	The format of the model input. Array of unsigned integers.
Output format	Array (4)	The format of the model output. Array of unsigned integers.
Quantization	List(4)	Quantization details
Method	Text String (3)	The quantization method used. E.g., "8-bit", "Binary", "Float16"
Bits	Unsigned Integer (0)	Number of bits used for quantization. E.g., 8
Weight Quantization	Text String (3)	Describes the quantization method for the model weights. E.g., "uniform", "asymmetric"
Activation Quantization	Text String (3)	Describes the quantization method for activations. E.g., "symmetric", "asymmetric"
Post training	Unsigned Integer (0)	Indicates if post-training quantization was used. E.g., 1 (true), 0 (false)
ML Framework (Array of maps)		
Name	Text String (3)	The name of the machine learning framework used. E.g., "TensorFlow", "PyTorch", "ONNX"
Version	Text String (3)	The version of the machine learning framework. E.g., "2.8.0"
Runtime	Text String (3)	he runtime environment of the framework. E.g., "TensorFlow Lite", "uTensor", "uTVM"
Hardware acceleration	Unsigned Integer (0)	Indicates if the model uses hardware accelerators (NPU, DSP)
Supported operators	List (4)	List of operators supported by the runtime framework
Operation N	Text String (3)	Name of each supported operation. E.g., "Conv2D", "BatchNorm", "Dense"
Model architecture (Array of maps)		
...	...	...

Due to the lack of standardization, token claim labels can be assigned to IANA private claim keys or encoded as text strings. While mapping general claims to predefined identifiers may be straightforward, the situation becomes more complex for the model architecture subclaims. This is because the model architecture can vary significantly across different machine learning frameworks, each supporting multiple layer types and configurations.

In Section 6, we assess both encoding methods. Analogously to the EAT or PSA standards, converting claim labels into integer keys results in a more compact form and ensures cross-device compatibility. Nonetheless, for ML models, claims encoded as strings might be more effective since they naturally match the JSON or other formats frequently employed by ML frameworks. This approach enables a verifier or relying entity to use the string-encoded keys of model subclaims directly, bypassing extra processing. Conversely, adopting integer-encoded claim keys requires a prior consensus for each ML framework along with a mapped key set for all corresponding ML layers and parameters.

4.3. Remote attestation workflow

The remote attestation procedure involves different actors, as shown in Figure 3. We utilize the RATS terminology to define a possible attestation scenario workflow, ensuring a standardized approach to describing the roles and interactions of key entities.

The device, referred to as the prover, generates a nonce derived from the random boot seed created during system boot. The relying party, which relies on attestation results to make trust decisions, sends an attestation request to the device along with the specified type of token to be generated. If the ML-EAT token is selected, the label encoding can be further chosen as integer-encoded or string-encoded.

The device sends the nonce to the relying party, which processes it to compute a hash value, $H(\text{nonce}\,||\,H(w)G)$, and returns it to the device. When the relying party requests ML attestation, it employs a zero-knowledge proof over the hash of the model weights. Here, $H(w)$ represents the hash of the model weights or the hash over the ML model object. $H(w)G$ represents a point encoded on a `secp256k1` elliptic curve. The device verifies that the hash of the current nonce combined with the encoded point retrieved from secure storage matches the received value. Depending on the verification result, the device either aborts the process or generates the attestation proof as requested.

The nonce is incremented with each attestation request to prevent replay attacks. The device generates authentication tokens only if the remote relying party can compute the same hash as the expected one. Additionally, a Denial-of-Service (DoS) attack from a malicious verifier attempting to overwhelm the device with multiple attestation queries can be mitigated. Since attestation is not a frequent process, rate limiting can be implemented to restrict the number of requests allowed within a given time frame. A Man-in-the-Middle attack is mitigated by design, since the tokens are signed with private keys that are only used in the secure environment and by the verifier party. The integrity of the token and its claims cannot be altered in transit, only if the private key is known.

The verifiers might be an independent entity or integrated into the relying party's system. With the help of a verifier entity, the relying party can evaluate the evidence of the produced tokens and determine its validity.

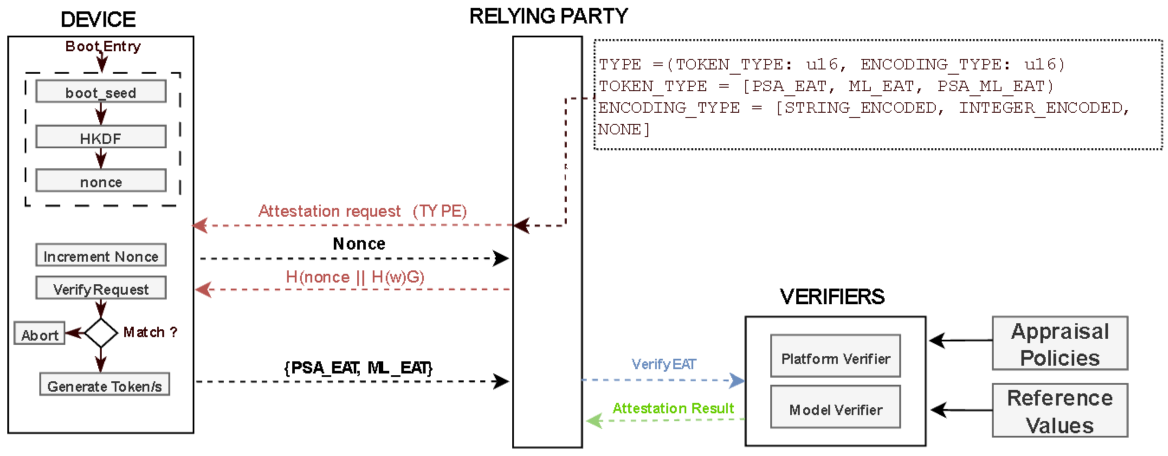

Figure 3. Remote attestation mechanism. Attestation request from a remote relying party.

5. Methodology

5.1. System Overview

Here we provide a description of the selected platform, software components and details about how the attestation and secure boot are performed.

5.1.1. Platform

To implement an end-to-end ML attestation solution, the STM32H573I-DK [37] development platform (Figure 4) is utilized. This platform features an STM32H573 ARM Cortex-M33 core with TrustZone enabled, 2MB of internal flash, 640kB of SRAM, a hardware security module with dual AES coprocessors, a hash accelerator, a True Random Number Generator (TRNG), secure data storage, and tamper detection capabilities. Additionally, it incorporates an immutable root of trust (ST-iRoT) that functions as the first boot stage. An advantage of this platform is its integration into the STM32Trust Ecosystem, which enhances security and simplifies implementation. The STM32Trust Ecosystem provides a TEE Secure Manager (SM), streamlining the setup and management of secure services.

The SM is designed to reduce the development complexity of secure applications. However, one drawback is that the SM image is provided in an encrypted format, meaning developers do not have full control over how security services are implemented. A more customizable solution would be TF-M [38], offered by ARM Community, but this comes at the cost of increased complexity, particularly when integrating TF-M with platform-specific hardware features.

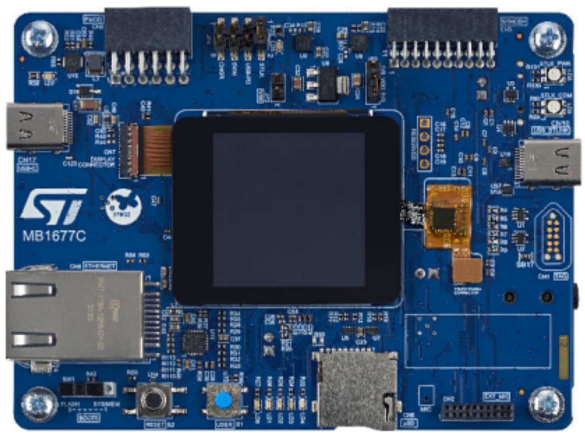

Figure 4. STM32H573I-DK Discovery development platform. [37]

5.1.2. Software components

The software architecture is based on the principle of separation between the non-secure and secure environments. The separation is done by the STuRoT which further loads the SM firmware and the nonsecure application.

The SM operates within a secure processing environment, consisting of the SM-Core and a range of secure services, including cryptography, initial attestation, trusted internal storage, and firmware updates. The SM-Core facilitates communication between non-secure applications and secure services through PSA-API, manages the scheduling of secure service execution, and, with the help of a memory protection unit, ensures the isolation of secure services from one another.

The non-secure application consists of an ML user application that performs ML model inference while simultaneously handling attestation queries through the serial interface. The ML inference framework used in this system is TensorFlow Lite (TFLite), which is specifically designed for mobile and embedded devices.

There are multiple tools and packages available for converting ML models to formats suitable for embedded systems [39]. Some of these tools directly generate code, representing the model as an instruction list that can be executed as part of the application. Others generate the model as an array of bytes, which is interpreted at runtime by the application or an inference engine. As shown in Figure 5, the nonsecure application consists of multiple software components that are compiled together into a final binary. Similarly, a ML model can be integrated directly into the application binary during the build process or stored separately in external flash memory.

TFLite [40] is a lightweight, portable framework optimized for running ML models efficiently on devices with limited computational resources, such as microcontrollers, smartphones, and IoT devices. It supports a wide range of platforms, making it an ideal choice for embedded applications. TFLite's flexibility ensures that it can be integrated seamlessly into various embedded devices, while maintaining high efficiency in terms of both processing speed and resource usage.

Preprints.org (www.preprints.org) | NOT PEER-REVIEWED | Posted: 26 December 2024 doi:10.20944/preprints202412.2251.v1

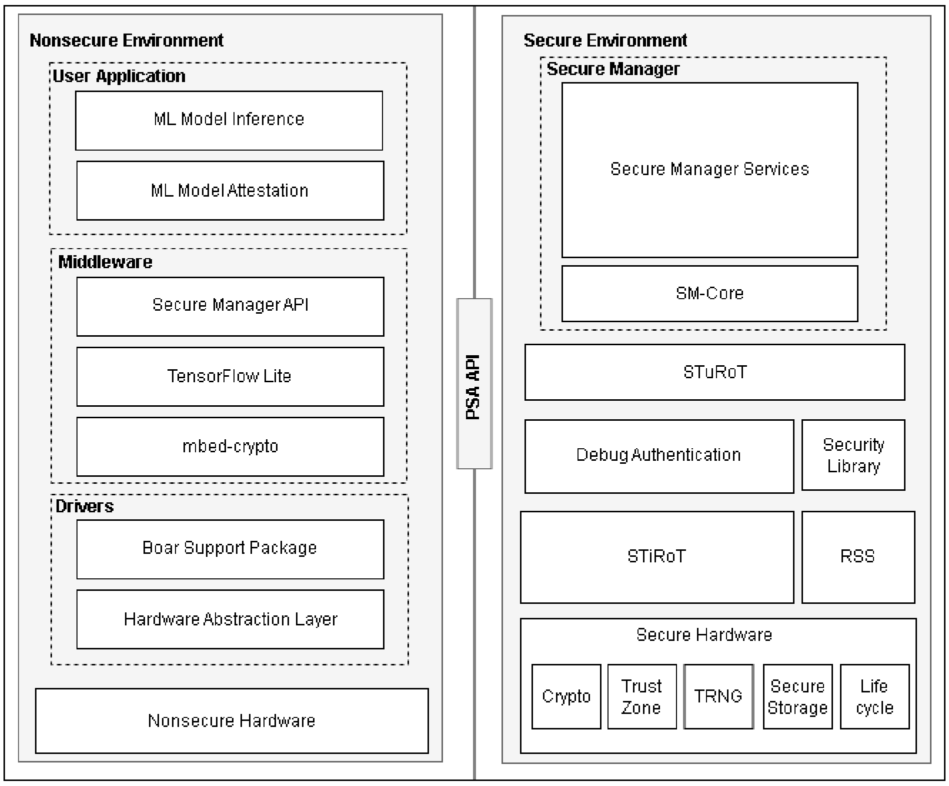

Figure 5. Software architecture of the proposed dual attestation mechanism.

5.1.3. Device provisioning

The entire attestation process depends on the device being pre-provisioned with the necessary key material, secure application components, and firmware images, as illustrated in Figure 6. The secure provisioning phase involves setting up the secure boot chain by defining the flash layout, specifying the locations of the SM and application images, and determining the keys used to verify and decrypt these images. It also includes configuring boot options to enable TrustZone and activate STiRoT boot.

The key material and application data are stored in the internal trusted storage (ITS), which is created using STM's ITSBuilder tool. The ITS file system is encrypted and can only be decrypted by the SM-Core.

Application provisioning involves the ML training pipeline and model conversion to TFLite, where the compressed ECC `secp256k1` point (H(w)G) of the model is stored in the internal ITS. The ML-EAT token is generated off the device, with only the fields that need to be computed at runtime being populated during the attestation request.

The non-secure application is compiled and encrypted using the pre-provisioned keys on the device. Flashing is performed at a specific address, as defined by the flash layout. This ensures that only authorized, pre-configured software can execute on the device, thereby supporting a secure boot process and maintaining overall system integrity.

The STuROT supports a dual-slot boot interface. During a firmware update, the new image is placed in the download slot. Upon the next power cycle, if the image is authenticated and its integrity is verified, it is copied to the active slot and executed. This mechanism enables over-the-air updates for the ML model, as illustrated in Figure 1.

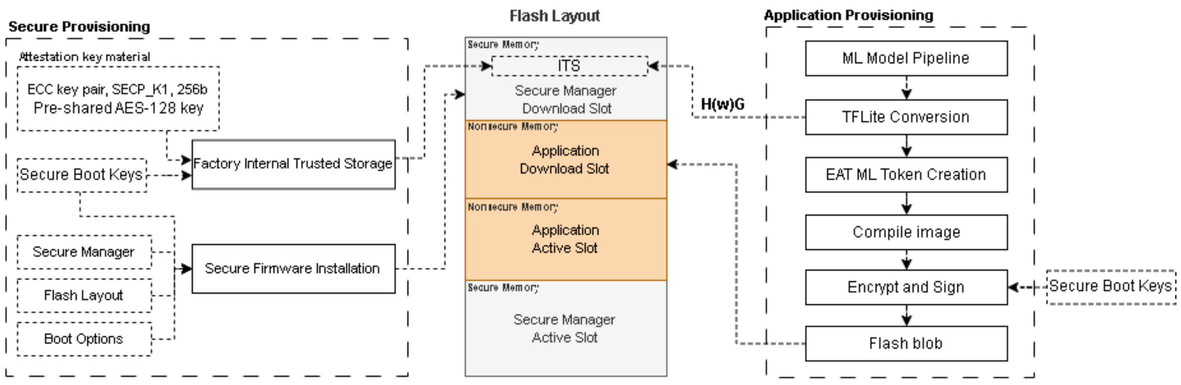

Figure 6. Provisioning of secure artifacts and ML application provisioning flow.

5.1.4. Secure Boot

The secure boot process is carried out in two distinct stages. The first stage is managed by an immutable Root of Trust provided by ST, which is natively embedded in the system flash. This RoT is fixed and never erased, providing a strong foundation for secure boot operations. TrustZone is enabled during the secure provisioning phase through a specific flash option byte (TZEN), and the system's clock speed is configured to 250 MHz to ensure stable operation. As part of the immutable artifacts, two key components are included: the RSS and the STiRoT. The STiRoT is responsible for authenticating, loading, and decrypting the STuRoT, which serves as the second-stage bootloader. This multi-layered approach ensures a robust and secure initialization process for the system. All cryptographic operations during this phase are hardware-accelerated, significantly enhancing performance and security.

The RoT implements several cryptographic schemes to safeguard the boot process. These include ECDSA-P256 for image authentication, ECIES-P256 for confidentiality, and SHA-256 for integrity checks. These cryptographic operations provide a strong defense against tampering and unauthorized modifications, ensuring that only verified and trusted code is executed on the device. The STuRoT verifies both the integrity and authenticity of the user application firmware images, including the non-secure application and the SM binary.

Both firmware binaries, the SM and the non-secure application, are encrypted using the ECIES-P256 algorithm for confidentiality and are authenticated with separate ECC secp256r1 keys. The Secure Monitor is encrypted and authenticated using proprietary keys, while the non-secure application is encrypted and authenticated using keys installed during the secure provisioning process. These keys are securely stored in the ITS flash area, which is protected by a temporal isolation mechanism. This mechanism ensures that the keys remain secure and isolated, preventing unauthorized access. Additionally, the firmware executing in a specific layer `N` is prevented from accessing the secure storage of any lower layers `N-x`. This separation enforces strict security boundaries, ensuring that the secure storage of lower layers remains inaccessible and maintaining proper isolation between the secure and non-secure components of the system.

5.1.5. PSA initial attestation token

The format of a PSA attestation token is shown in Listing 1. The payload, created by the STM SM within the secure environment, includes a map of custom claims that are specific to the PSA specification.

Apart from the listed claims, the token also includes a map of claims for all the software components in the system, specifically the non-secure application, STuRoT, and the SM itself. These claims provide additional information about the software running in the secure environment and its integrity, ensuring that each component can be attested and trusted (Table 2).

The signature is generated using a preprovisioned key, the DUA initial attestation key, provided by the SoC manufacturer. This key is unique to each device and is utilized by the attestation service to sign the token. The corresponding public key can be retrieved from the device in the form of a certificate, which serves as proof of the device's identity and confirms that the token was signed by a trusted source within the device's secure environment.

Table 2. Software component details for PSA token.

Claim Name	Details	Possible Values
`measurement_description`	Describes the type of hash algorithm used for the measurement.	"SHA256", "SHA512", etc.
`measurement_value`	Computed hash value of the software component	Any valid byte array
`signer_id`	Hash of the public key used to verify the software component.	Any valid byte array
`sw_component_type`	Specifies the type of the software component being described.	"NSPE" (ML application) "SPE" (STM SM) "BL" (STuRoT)
`sw_component_version`	Version of the software component.	Any valid version string (e.g., "1.0.0", "1.1.0").

The initial PSA attestation token profile used claim key IDs reserved for private use, as registered by IANA, with values less than -65536. However, a later PSA attestation profile adopted standardized claims, which are listed in [33].

5.2. Description and implementation of the model under test

We assess the on-device token generation for models applied in anomalous sound detection based on the autoencoder (AE) architecture proposed in [41], which can be used in federated learning approaches. The AE comprises eight dense layers, each with 128 units, an 8-unit bottleneck, and ReLU activations after each layer except the final one. Additionally, we evaluate simplified architectures with 32 and 64 units, available in both quantized and non-quantized versions. The quantized models implement full integer quantization for weights and activations, utilizing int8 inputs and outputs to enhance on-device efficiency.

The models are built with Keras 2.4.0, trained on parts of ToyADMOS and the MIMII Dataset [42] (normal/anomalous sounds of six toys and real machines), and converted to TensorFlow Lite. Before conversion, batch normalization layers are folded into dense layers to improve efficiency and reduce model size during inference.

The ML-EAT token is generated off-device by combining the COSE header, the generic claims from Table 1, and the model architecture, extracted from the summary of each network layer using Keras APIs. This data, structured in JSON format, is then serialized to CBOR using the `cbor` Python package. Since the token structure is created during the training phase, the device does not have a CBOR encoder firmware. At runtime, the device fills in the hash of the TFLite buffer data, the hash of the public key for model updates, and the signature at predefined offsets.

5.3. Evaluation metrics

The evaluation of the proposed solution focuses on key metrics that reflect its efficiency and applicability. Initially, two key aspects are evaluated: the memory footprint, representing the resource demands of the implementation, and the secure boot time, critical for ensuring secure system initialization. These foundational metrics provide a basis for evaluating the feasibility and suitability of the solution on the experimental platform.

Further, other metrics are be derived from the following use cases:

1. A cloud-based service, acting as a relying party, wants to verify the device's trustworthiness and obtain high-level details about the ML model, such as vendor, version, accuracy, and the latest update. It is not interested in the detailed specifics of the model itself.
2. A federated learning system queries a device that has just joined the collaborative network for the weights it has trained. In this scenario, the system needs to verify both the trustworthiness of the device and the model architecture. Depending on the security requirements of the model, the token claims can either be encrypted or unencrypted. Additionally, the system may either recognize standardized ML claims (integer-encoded) related to the model architecture or rely on string-encoded claims that align with the structure of the ML framework.

The inference time of an ML model depends on its complexity, as demonstrated by the MLPerf Benchmark [43]. For instance, a basic image classification task might require around 700 ms for inference. In contrast, anomaly detection tasks using the same AE architecture employed in this study, running on an STM NUCLEO-L4R5ZI, take only a few milliseconds, typically ranging from 15 to 20 ms.

Attestation time is crucial in systems that perform inference at the edge, as it should be minimal to ensure quick decision-making and maintain real-time performance. Any significant delay in attestation could negatively impact the system's responsiveness and overall efficiency.

We evaluate how the model architecture impacts the ML-EAT token size and the attestation time. Further, we compare how important is to have standardised ML claim labels for the model architecture and how this impacts the token generation time.

6. Results

6.1. Memory footprint

The system memory comprised of SRAM and flash is segmented into various regions, with allocations to both secure and non-secure environments, as illustrated in Table 3. The flash memory mapping for the secure world is determined by the SM, which enforces a predefined memory configuration that cannot be modified by the user. This mapping includes the SM and STuROT firmware images, the ITS file system, and the ITS provisioning area. The SRAM memory required for the SM is 140 kB and the flash memory footprint is 360 kB. Additionally, a shared buffer (NS/S) area is allocated for data exchange between the non-secure application and the SM. This buffer is sized to a maximum of 16 kB to optimize the hashing time of the TFLite model buffer, which is processed in chunks of 12 kB.

The flash memory for the non-secure environment primarily contains the application components, with the TFLite interpreter occupying the largest portion of 242 kB. The size of the ML models varies depending on their hyperparameters and quantization settings. The difference in size between quantized and non-quantized models is approximately 3 to 4 times smaller for the quantized versions, offering a significant reduction in memory usage. Nevertheless, apart from the benefit of reduced memory usage, quantization comes with a penalty in model accuracy [44].

The SRAM footprint of the model refers to the memory allocated for tensors, which includes the input, output, and intermediate data used during model inference. The size of this memory area depends on the target architecture and must be explicitly defined. To determine the optimal memory allocation, we use an open-source divide-and-conquer algorithm [45]. This helps in efficiently sizing the tensor memory required for the model's execution, considering the constraints of the target system.

Table 3. Component memory usage for secure and non-secure environments. ML model size for quantized and non-quantized models.

	SW Component	Flash [kB]	SRAM [kB]
Secure Environment	Secure Manager	360	140
	NS/S Area	-	16
Non-secure Environment	Application Components	358.2	4.7
	Model AE-32	198.27	7.48
	Model AE-32-quantized	57.6	6.46
	Model AE-64	438.9	7.61
	Model AE-64-quantized	120	6.49
	Model AE-128	1067.64	7.87
	Model AE-128-quantized	257.72	6.63

The STuROT supports dual-slot boot for both the SM and non-secure firmware images, enabling over-the-air firmware updates. This dual-slot configuration effectively doubles the available flash memory. With a total of 2 MB of flash memory, the SM and application components occupy 718.2 kB, leaving 1281.8 kB for the ML model in a single-slot setup. However, in the dual-slot configuration, this available memory is divided, leaving only 640.9 kB for an ML model. As shown in Table 3, the AE-32, AE-64 (both quantized and non-quantized), and AE-128 quantized models can fit within this available flash memory in the dual-slot configuration. However, larger models, such as the non-quantized AE-128, exceed the available memory limits.

6.2. Secure boot

The boot times vary depending on the configuration of the non-secure application and the size of the model. As both the application and model sizes increase, the boot time also increases, reflecting the additional data that must be loaded during the boot process. The ML models are integrated into the final firmware image, with boot times provided for both quantized and non-quantized models. Since quantized models are typically smaller, they lead to faster boot times compared to non-quantized models of the same architecture. This demonstrates how model size and quantization impact boot time in a system with a 250 MHz clock speed. The details of these variations can be seen in Table 4.

Table 4. Boot times for different image sizes based on the application and model configurations.

Component	Image Size (kB)	Boot Time (ms)
Application + AE-32	556.47	50.82
Application + AE-32-quantized	415.8	51.25
Application + AE-64	797.1	52.63
Application + AE-64-quantized	478.2	51.25
Application + AE-128	1425.84	53.06
Application + AE-128-quantized	615.92	51.69

6.3. Attestation tokens size

The size of the attestation tokens is a key factor, as larger tokens require more computational resources and take longer to process. In non-hardware-accelerated cryptographic implementations, increasing token size leads to higher power consumption and longer execution times. Additionally, for IoT devices, the time required to transmit the data over the air grows with the size of the token, as larger payloads take more time to send. This extended transmission time directly affects the device's energy consumption, particularly in battery-powered IoT devices [46].

The PSA initial attestation token, generated by the SM, has a total size of 547 bytes. It consists of a 10-byte COSE header and a 68-byte COSE signature. The token contains integer-encoded claim keys that are part of the standardized format.

The size of the ML-EAT token is sum of the general claims, which are constant regardless of the model architecture, and the submodule claims that define the model architecture.

The general claim keys can be integer-encoded using private-usage IANA registry keys. For this implementation, the base address for the first claim is set to -70000, with each subsequent claim having a decremented value from this base. The total size of the general claims is 537 bytes.

Table 5. Size comparison (bytes) of Integer-Encoded and String-Encoded Claim Keys for different Keras generated layers.

Layer	String-Encoded Claim Keys	Integer-Encoded Claim Keys
Activation	158	95
Add	125	70
BatchNormalization	660	327
Conv2D	544	243
Dense	427	207
Dropout	166	98
Flatten	159	95
LSTM	801	358
MaxPooling2D	210	117
Multiply	135	80
Input	89	49
ReLU	182	101
Softmax	139	82

The subclaims regarding model architecture depend on the ML framework used to create the model and should follow a standardized format to ensure compatibility with the framework and model-saving APIs. For example, Keras provides the `get_config()` function, which returns a JSON-serializable dictionary representing the model architecture. Similarly, TensorFlow offers `model.to_json()`, and PyTorch uses `torch.save()` to export the model architecture in a consistent format.

Table 5 shows the CBOR encoding sizes of commonly used Keras layers compatible with TensorFlow Lite, for both string-encoded and integer-encoded claim keys. For integer-encoded keys, the base value used is -75000, with subsequent keys decremented accordingly. These sizes are determined by saving each layer's configuration in a JSON format, serializing it to CBOR, and then measuring the memory footprint. The serialized sizes are compared to evaluate the impact of different encoding schemes on model memory usage.

For the evaluated AE architecture, the ML-EAT token size is 9468 bytes when using string-encoded claim keys. Since each encoded layer includes a Dense layer, followed by a BatchNormalization and Activation layers, the size is nearly halved to 4218 bytes when integer-encoded claim keys are used, excluding the input and output layers. This reduction in token size significantly enhances transmission efficiency in IoT applications, leading to reduced transmissions times and lower energy consumption.

In Figure 7, we show how the token size varies with the number of encoder layers for the evaluated AE architecture. Depending on the model architecture, this trend can accelerate further. For instance, an LSTM layer becomes 2.2x smaller when integer-encoded claim keys are used instead of string-encoded claim keys. This demonstrates that the impact of the encoding method on token size can be more pronounced for certain layers, highlighting the importance of encoding choices in optimizing token sizes.

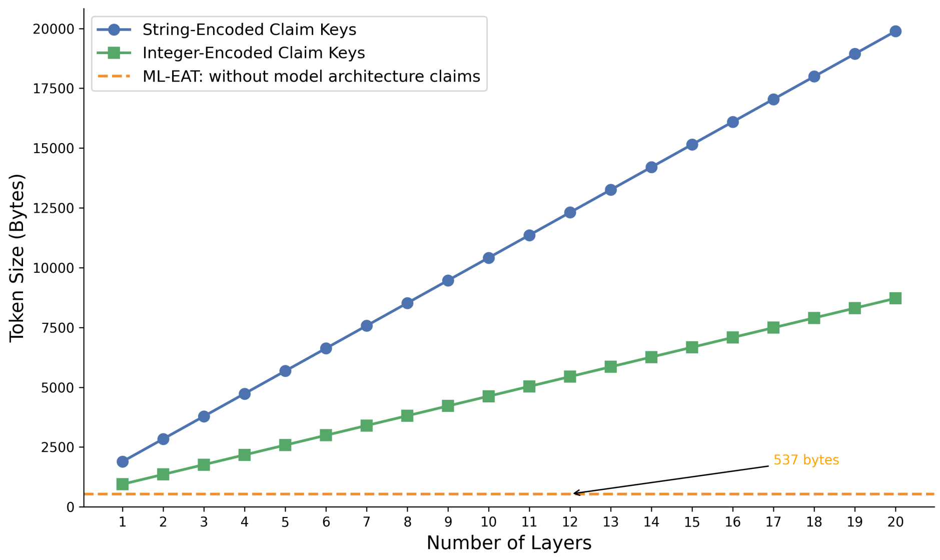

Figure 7. ME-EAT token size based on the number of encoder layers for the evaluated AE architecture.

6.4. Attestation time

The inference time of a ML model refers to the duration it takes to process input data and generate predictions, and it must be maintained within application-specific limits to ensure timely decision-making. However, if the ML model attestation process is resource-intensive, it may compete with the inference time, potentially leading to delays in decision-making.

We evaluate the attestation time of the dual attestation method, specifically the time elapsed from when the prover receives the challenge until the device generates the two linked attestation tokens for the verifier. This evaluation focuses solely on the time it takes for the device to generate the tokens and does not include the reply protection checks, as illustrated in Figure 3.

One of the claims in the ML-EAT token is the model hash (see Table 1), which represents the hash of the TFLite flatbuffer array, flavored with a nonce. This hash is computed on the device to ensure that the model has not been modified or tampered with during runtime. This is critical in an attestation system, where the verifier needs to confirm that a specific model is being executed on the device. In a federated learning scenario, the device-computed hash ensures that the updated weights sent to the server are identical to those at the time of attestation. Nevertheless, if model weights are not updated at runtime, the hash value of the TFLite flatbuffer array can be stored in the ITS area during the provisioning stage.

The hash of the model is computed at runtime over the entire TFLite flatbuffer array, which includes weights, layer configurations, and metadata. Extracting just the weights would require parsing the entire flatbuffer, which would not only cause a performance bottleneck on resource-constrained devices but also require integrating a parser, increasing the memory footprint.

Figure 8 illustrates the attestation time for each model, with the hash computation time included as part of the overall token generation time. This is shown to highlight the additional overhead introduced by the hash process in relation to the token generation.

The attestation time includes the creation of the PSA attestation token, the hashing of the ML model, and the generation of the ML-EAT token. The creation of the PSA token, which is independent of the specific model being executed, takes approximately 15.2 ms. The hash chain, which links the two tokens, requires around 1.8 ms.

The variation in token generation time based on the encoding type of the claim keys is primarily attributed to the AES-CBC-128 encryption time. When the ML-EAT token uses integer-encoded claim keys for the model architecture, the average difference in time generation is approximately 8 ms, resulting in an average improvement of 9.4%.

Quantized models consistently demonstrate faster hash times across all sizes and also reduce token generation times. On average, quantized models show a 31.17% improvement in hash times compared to non-quantized models. Additionally, the average improvement in token generation times for quantized models is approximately 14.66%.

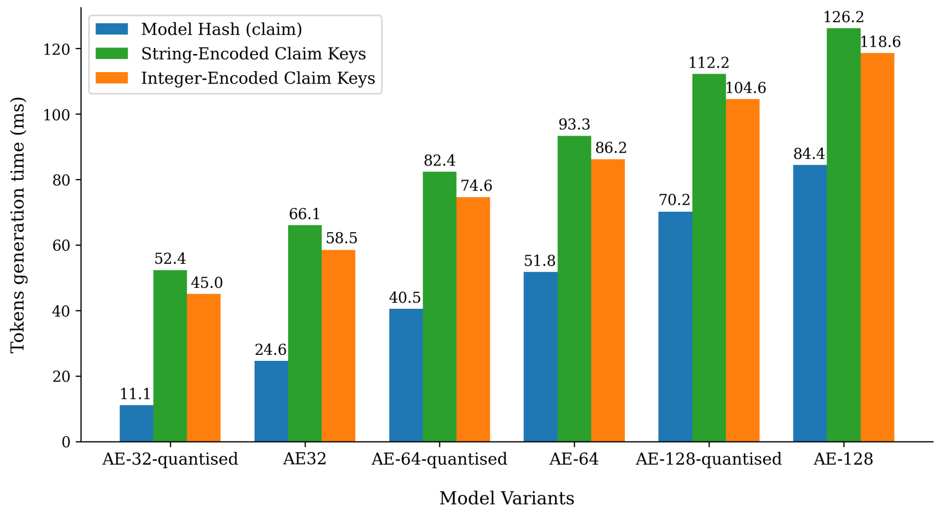

Figure 8. Performance of the dual attestation mechanism for the evaluated models. Tokens generation time and model hash overhead.

In addition to the hash time of the model, the token generation time is influenced by the model architecture. The ML-EAT token includes the encoded model architecture, stored as subclaims, which is produced by the model framework and encoded in CBOR. An increase in the number of layers will expand the model subclaims section, resulting in longer ML-EAT hash times and additional time for encrypting the model subclaims.

The attestation time for different numbers of encoder layers in the AE model is shown in Figure 9. The plot excludes the model hash overhead and focuses solely on the impact of encryption on the model size, depending on the encoding type of the claim keys.

We evaluated how the attestation time differs between the ML-EAT token containing model subclaims as plain data versus encrypted data. If the ML-EAT token uses string-encoded claim keys, the encryption overhead for model subclaims averages around 6.39 ms for architectures with fewer than 10 layers, while for models with more than 10 layers, the average overhead increases to 19.71 ms. In comparison, for integer-encoded claim keys, the encryption overhead is lower, with an average difference of 7.42 ms for models with fewer than 10 layers and 10.45 ms for models with more than 10 layers.

Integer-encoded tokens are generally more efficient, especially for larger models, in terms of both overall time and the added encryption overhead. On average, the attestation time for an integer-encoded token with encrypted model subclaims is 2.6 ms higher than that of a string-encoded token with plain data. By standardizing the ML-EAT token to use integer-encoded claim keys, token generation time could be significantly reduced, while maintaining the encryption of model subclaims. This approach would lead to more efficient processing and lower overhead, particularly for larger models.

The baseline of 27.8 ms corresponds to the attestation generation time when the ML-EAT does not include the model architecture subclaims, and the model's hash is precomputed during the device's provisioning phase.

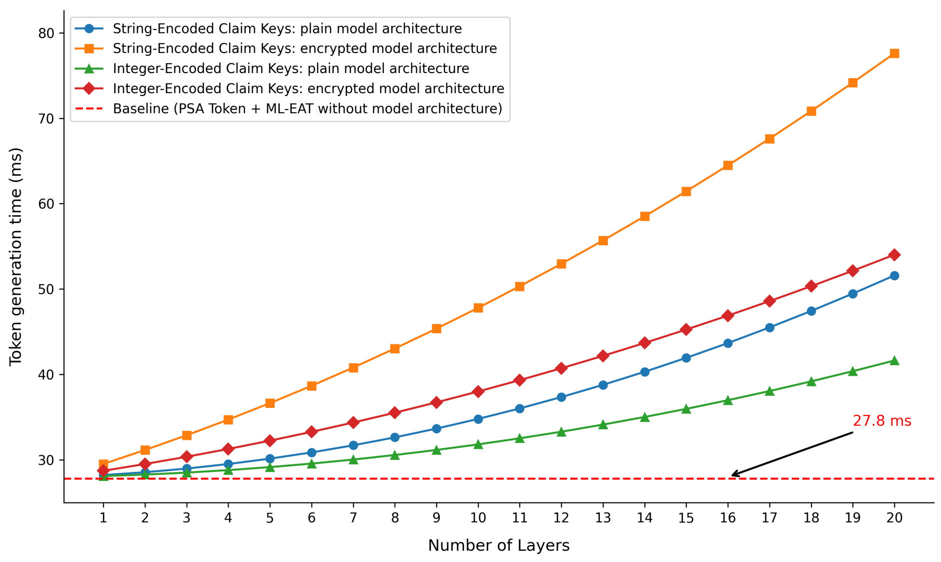

Figure 9. Token time generation based on the number of encoder layers for the evaluated AE architecture. Model hash time not included.

6.5. Discussion

In Section 6.4, we demonstrate the applicability of ML attestation for a model architecture proposed in the literature and used on TinyML devices. We assess how various model hyperparameters influence the attestation time and the size of the ML-EAT token. Additionally, we evaluate the benefits of standardization or consensus between devices when comparing integer-encoded claims to string-encoded ones.

Based on the use cases described in Section 5.3, the following insights can be drawn:

1. In the first scenario, the token generation time of 27.8 ms on the device would not affect the real-time requirements of the anomaly detection system. Given that the inference time is between 15-20 ms and the attestation is not performed frequently, the impact remains negligible. Furthermore, the token size is small, consisting of 1084 bytes, which includes both the PSA and ML-EAT tokens.
2. In the second scenario, it is evident that both the ML-EAT token size and the encoding type of the claim keys play a crucial role. For the evaluated model architecture, the ML-EAT token size can be reduced from 9468 bytes to 4218 bytes, resulting in shorter attestation times and a smaller token size that needs to be transmitted.

   Moreover, for the models evaluated, the attestation time may impact the real-time requirements of the anomaly detection system on the device, as the attestation time can be up to six times longer than the assumed inference time. However, in a TinyML federated learning system, the attestation process may occur less frequently, since the model may not require frequent updates due to the stability of the patterns being learned across updates.

7. Conclusions

The proposed dual attestation mechanism provides an effective and robust solution to ensure the trustworthiness of machine learning models deployed on edge devices, while simultaneously maintaining the integrity of the device itself. This approach not only addresses security concerns but also enhances the reliability of the system as a whole. Furthermore, we have demonstrated that this solution can be implemented using existing, readily available hardware and software platforms, making it highly adaptable and suitable for seamless integration into other TinyML embedded designs.

In addition, the concept of token interoperability is explored through detailed discussions on standardized claims across popular machine learning frameworks. This highlights the critical role of standardization, especially in scenarios where attestation time is of utmost importance for real-time applications. By emphasizing the significance of uniformity in claim formats, we present a path towards more efficient and scalable solutions for edge-based ML applications. These findings contribute to a more comprehensive understanding of secure, efficient, and practical methods for remote attestation of ML models in the context of edge devices, paving the way for more trustworthy and performant machine learning solutions in the future.

Author Contributions: Conceptualization, V.E.B, B.d.S and A.N; methodology, V.E.B, B.d.S, A.N and L.S; software, V.E.B.; writing—original draft preparation, V.-E.B, B.d.S; writing—review and editing, B.d.S and A.N; visualization, V.E.B and B.d.S; supervision, B.d.S; project administration, B.d.S and A.N; funding acquisition, B.d.S and A.N

Funding: This work is part of the COllective Research NETworking (CORNET) project “TrustedIOT: Trusted Computing Architectures for IoT Devices” funded by VLAIO under grant number HBC.2021.0895 This publication is also based upon work from COST Action CA22104 Beingwise (Behavioral Next Generation in Wireless Networks for Cyber Security), supported by COST (European Cooperation in Science and Technology; www.cost.eu) and Cybersecurity Research Program Flanders - second cycle (VOEWICS02).

Conflicts of Interest: The authors declare no conflict of interest.

References

1. Mikhaylov, D.; Polonelli, T.; Magno, M. On-Sensor TinyML Event-Based Fault Detection Strategies on Wind Turbine Blades. In Proceedings of the 2024 IEEE Sensors Applications Symposium (SAS). IEEE, 2024, pp. 1–6.
2. Gkogkidis, A.; Tsoukas, V.; Papafotikas, S.; Boumpa, E.; Kakarountas, A. A TinyML-based system for gas leakage detection. In Proceedings of the 2022 11th international conference on modern circuits and systems technologies (MOCAST). IEEE, 2022, pp. 1–5.
3. Huckelberry, J.; Zhang, Y.; Sansone, A.; Mickens, J.; Beerel, P.A.; Reddi, V.J. TinyML Security: Exploring Vulnerabilities in Resource-Constrained Machine Learning Systems. *arXiv preprint arXiv:2411.07114* **2024**.
4. Confidential Computing Consortium. Confidential Computing: Outreach Whitepaper, 2022. Accessed: 2024-11-21.
5. Segers, L.; Talebi, B.; da Silva, B.; Touhafi, A.; Braeken, A. Trustworthy Environmental Monitoring Using Hardware-Assisted Security Mechanisms. *Sensors (Basel, Switzerland)* **2024**, *24*.
6. Costan, V. Intel SGX explained. *IACR Cryptol, EPrint Arch* **2016**.
7. Sev-Snp, A. Strengthening VM isolation with integrity protection and more. *White Paper, January* **2020**, *53*, 1450–1465.
8. Pinto, S.; Santos, N. Demystifying arm trustzone: A comprehensive survey. *ACM computing surveys (CSUR)* **2019**, *51*, 1–36.
9. Cheang, K.; Rasmussen, C.; Lee, D.; Kohlbrenner, D.W.; Asanović, K.; Seshia, S.A. Verifying RISC-V physical memory protection. *arXiv preprint arXiv:2211.02179* **2022**.
10. Confidential Computing Consortium. Common Terminology for Confidential Computing, 2022. Accessed: 2024-11-18.
11. Birkholz, H.; Thaler, D.; Richardson, M.; Smith, N.; Pan, W. Remote attestation procedures (RATS) architecture. *RFC 9334* **2023**.
12. Weidner, J. ARM Confidential Compute Architecture.
13. Trusted Firmware-M Documentation, 2024. Accessed: 2024-11-23.
14. Silicon Labs Gecko Platform: Secure Element Manager API Documentation, 2024. Accessed: 2024-11-23.
15. STM32 MCU Secure Manager Documentation, 2024. Accessed: 2024-11-23.
16. Asokan, N.; Brasser, F.; Ibrahim, A.; Sadeghi, A.R.; Schunter, M.; Tsudik, G.; Wachsmann, C. Seda: Scalable embedded device attestation. In Proceedings of the Proceedings of the 22nd ACM SIGSAC conference on computer and communications security, 2015, pp. 964–975.
17. Ficco, M.; Guerriero, A.; Milite, E.; Palmieri, F.; Pietrantuono, R.; Russo, S. Federated learning for IoT devices: Enhancing TinyML with on-board training. *Information Fusion* **2024**, *104*, 102189.
18. Lundblade, L.; Mandyam, G.; O'Donoghue, J.; Wallace, C. The Entity Attestation Token (EAT).
19. Schaad, J. Cbor object signing and encryption (cose). Technical report, 2017.
20. Ul Haq, S.; Singh, Y.; Sharma, A.; Gupta, R.; Gupta, D. A survey on IoT & embedded device firmware security: architecture, extraction techniques, and vulnerability analysis frameworks. *Discover Internet of Things* **2023**, *3*, 17.

21. Costin, A.; Zaddach, J.; Francillon, A.; Balzarotti, D. A Large-scale analysis of the security of embedded firmwares. In Proceedings of the 23rd USENIX security symposium (USENIX Security 14), 2014, pp. 95–110.
22. Ankergård, S.F.J.J.; Dushku, E.; Dragoni, N. State-of-the-art software-based remote attestation: Opportunities and open issues for Internet of Things. *Sensors* **2021**, *21*, 1598.
23. Kohnhäuser, F.; Büscher, N.; Katzenbeisser, S. A practical attestation protocol for autonomous embedded systems. In Proceedings of the 2019 IEEE European Symposium on Security and Privacy (EuroS&P). IEEE, 2019, pp. 263–278.
24. Eldefrawy, K.; Tsudik, G.; Francillon, A.; Perito, D. Smart: secure and minimal architecture for (establishing dynamic) root of trust. In Proceedings of the Ndss, 2012, Vol. 12, pp. 1–15.
25. Koeberl, P.; Schulz, S.; Sadeghi, A.R.; Varadharajan, V. TrustLite: A security architecture for tiny embedded devices. In Proceedings of the Proceedings of the Ninth European Conference on Computer Systems, 2014, pp. 1–14.
26. Ling, Z.; Yan, H.; Shao, X.; Luo, J.; Xu, Y.; Pearson, B.; Fu, X. Secure boot, trusted boot and remote attestation for ARM TrustZone-based IoT Nodes. *Journal of Systems Architecture* **2021**, *119*, 102240.
27. Ott, S.; Kamhuber, M.; Pecholt, J.; Wessel, S. Universal Remote Attestation for Cloud and Edge Platforms. In Proceedings of the Proceedings of the 18th International Conference on Availability, Reliability and Security, 2023, pp. 1–11.
28. Ferro, L.; Lioy, A.; et al. Standard-Based Remote Attestation: The Veraison Project. In Proceedings of the Proceedings of The Italian Conference on Cybersecurity (ITASEC 2024). CEUR-WS, 2024, pp. 1–13.
29. RATS, L.L. The Entity Attestation Token (EAT), 2024. Accessed: 2024-11-21.
30. Jones, M.; Camarillo, G.; Bormann, C. CBOR Web Token (CWT). https://www.rfc-editor.org/rfc/rfc8392.html, 2018. Accessed: 2024-12-24.
31. Internet Assigned Numbers Authority (IANA). JSON Web Token (JWT) Claims Registry. https://www.iana.org/assignments/jwt/jwt.xhtml. Accessed: 2024-12-24.
32. Bormann, C.; Hoffman, P. Concise Binary Object Representation (CBOR). https://www.rfc-editor.org/rfc/rfc8949.html, 2020. Accessed: 2024-12-24.
33. Internet Assigned Numbers Authority (IANA). CWT (CBOR Web Token) Parameters, 2024. Accessed: 2024-12-22.
34. Tschofenig, H.; Smith, N.; Frost, S.; Fuchs, A. Attestation Token for PSA. Internet-Draft draft-tschofenig-rats-psa-token-05, 2020. Expires: May 6, 2021.
35. O'Donoghue, J. Towards Lightweight and Interoperable Trust Models: The Entity Attestation Token **2019**.
36. Internet Assigned Numbers Authority (IANA). COSE (Concise Binary Object Representation) Parameters, 2024. Accessed: 2024-12-24.
37. STMicroelectronics. Discovery Kit with STM32H573II MCU. https://www.st.com/resource/en/user_manual/um3143-discovery-kit-with-stm32h573ii-mcu-stmicroelectronics.pdf, 2020. Accessed: 2024-12-24.
38. Trusted Firmware M. Trusted Firmware M Documentation. https://trustedfirmware-m.readthedocs.io/en/latest/index.html, 2024. Accessed: 2024-12-24.
39. Baciu, V.E.; Stiens, J.; da Silva, B. MLino bench: A comprehensive benchmarking tool for evaluating ML models on edge devices. *Journal of Systems Architecture* **2024**, *155*, 103262.
40. David, R.; Duke, J.; Jain, A.; Reddi, V.; Jeffries, N.; Li, J.; Kreeger, N.; Nappier, I.; Natraj, M.; Regev, S.; et al. Tensorflow lite micro: Embedded machine learning on tinyml systems. arXiv 2020. *arXiv preprint arXiv:2010.08678* **2010**.
41. Tanabe, R.; Purohit, H.; Dohi, K.; Endo, T.; Nikaido, Y.; Nakamura, T.; Kawaguchi, Y. MIMII DUE: Sound dataset for malfunctioning industrial machine investigation and inspection with domain shifts due to changes in operational and environmental conditions. In Proceedings of the 2021 IEEE Workshop on Applications of Signal Processing to Audio and Acoustics (WASPAA). IEEE, 2021, pp. 21–25.
42. Purohit, H.; Tanabe, R.; Ichige, K.; Endo, T.; Nikaido, Y.; Suefusa, K.; Kawaguchi, Y. MIMII Dataset: Sound dataset for malfunctioning industrial machine investigation and inspection. *arXiv preprint arXiv:1909.09347* **2019**.
43. Banbury, C.; Reddi, V.J.; Torelli, P.; Holleman, J.; Jeffries, N.; Kiraly, C.; Montino, P.; Kanter, D.; Ahmed, S.; Pau, D.; et al. Mlperf tiny benchmark. *arXiv preprint arXiv:2106.07597* **2021**.
44. Novac, P.E.; Boukli Hacene, G.; Pegatoquet, A.; Miramond, B.; Gripon, V. Quantization and deployment of deep neural networks on microcontrollers. *Sensors* **2021**, *21*, 2984.

45. tflite-find-arena-size. https://github.com/edgeimpulse/tflite-find-arena-size, 2024. Accessed: 2024-12-22.
46. Jiang, W.; Guo, Z.; Ma, Y.; Sang, N. Measurement-based research on cryptographic algorithms for embedded real-time systems. *Journal of Systems Architecture* **2013**, *59*, 1394–1404.