RAM-Based Firmware Attestation for IoT Security: A Representation Learning Framework

Asif Iqbal, *Member, IEEE*, Usman Zia, Muhammad Naveed Aman, *Senior Member, IEEE*, and Biplab Sikdar, *Senior Member, IEEE*

***Abstract*—With the proliferation of 4G and 5G mobile networks in smart cities, the adoption of IoT devices has surged, emphasizing the critical need for robust security measures. Existing firmware attestation techniques often require high computational budget or access to the device's authentic firmware, posing challenges due to resource and proprietary constraints. To counter these two fundamental challenges, this paper introduces a novel software-based attestation framework utilizing RAM traces from IoT devices for remote verification. In the proposed framework, the need for an authentic firmware copy is eliminated, and the most computationally intensive task is assigned to the gateway node of the IoT ecosystem. This approach yields a robust and highly accurate device attestation strategy, while imposing minimal computational demands on the verification device itself. Employing deep learning models trained in a representation learning paradigm, our framework enables the remote verifier to authenticate the internal state of IoT devices. Leveraging data collected from real-world prototype devices, under eight different applications, our approach achieves a remarkable 100% accuracy in detecting critical attacks on IoT devices with a false positive rate of 10^{-3}. Notably, our framework preserves device availability and maintains low authentication latency, underscoring its efficacy and practicality for securing IoT ecosystems.**

***Index Terms*—Device Attestation, Firmware, Internet of Things, RAM Trace, Variational Autoencoder**

I. Introduction

THE utilization of Internet of Things (IoT) devices is experiencing exponential growth across various domains of our daily lives. According to the GSM report published in 2020, the annual growth rate of IoT devices is estimated at 13%, with the total number of diverse IoT devices projected to reach approximately 2.46 billion by 2025 [1], [2]. Among the sectors witnessing a significant induction of IoT devices are healthcare, smart cities, industrial manufacturing and control, forest monitoring, traffic monitoring, defense, and more [3]. Many of the deployed devices in these applications are low-cost, leading to constraints in computational power, memory, and power consumption.

This work is supported in part by the National Research Foundation, Singapore and Infocomm Media Development Authority under its Future Communications Research Development Programme, under grant FCP-NUS-RG-2022- 019.

A. Iqbal, and B. Sikdar are with the Department of Electrical and Computer Engineering, National University of Singapore, Singapore, e-mail: {aiqbal, bsikdar}@nus.edu.sg

U. Zia is with the Department of Computer Systems Engineering, University of Engineering and Technology, Peshawar, Pakistan, email:usmanzia600@gmail.com

M. N. Aman is with the School of Computing, University of Nebraska-Lincoln, Nebraska, USA, e-mail:naveed.aman@unl.edu

The rising popularity of IoT devices, coupled with their integration into critical infrastructure, has drawn attention from cyber-criminals [4]. Moreover, their distributed nature and multi-layered architecture make them susceptible to cyber attacks targeting network communications, thereby jeopardizing device and data integrity [5]. Recent studies indicate that over 90% of the vulnerabilities identified in IoT device-enabled smart applications are firmware-related [6]. Typically, upon the addition of a new IoT device or during each power-up cycle, the device must register itself with a trusted entity, known as the *verifier*, through a process called *attestation*. The *verifier* is responsible for validating the authenticity of the device, examining the firmware or software running on the embedded device to guard against malicious tampering by third parties.

The verifier serves as a secured and trusted entity capable of invoking attestation even during normal operation, where the device in question, referred to as the *prover*, substantiates its authenticity. Typically, the verifier initiates a challenge to the prover, which then computes a hash digest using its firmware source code and shares it with the verifier. Subsequently, the verifier computes the same checksum using a locally stored copy of the prover's firmware to validate its authenticity. However, the checksum computations, which necessitate multiple iterations over the device firmware, result in heightened computational and time requirements, constituting the primary bottleneck in such strategies and leading to prolonged verification delays. Additionally, this approach mandates the verifier to possess a local copy of the prover's firmware, which may not be feasible and could potentially result in intellectual property (IP) violations.

The attestation techniques documented in literature can be categorized into two distinct types and an additional hybrid category. The distinct categories include software-based and hardware-based techniques, while the hybrid category encompasses a combination of both. Software-based techniques rely on the runtime execution of an algorithm on the device under test, comparing it with the expected value stored at the verifier. Given the limited computational resources of IoT devices, this runtime may vary depending on the device's current state or configuration, and can be useful for attestation purposes. These techniques are computationally intensive yet are well-suited for low-cost embedded devices, necessitating no additional hardware for execution. Conversely, hardware-based techniques, as implied by their name, necessitate additional hardware such as secure co-processors or Trusted Platform Modules (TPM) [7], and entail low computational requirements. However, owing to the advanced hardware pre-

requisites, these methods are unsuitable for the majority of IoT devices. Hybrid techniques aim to strike a balance between the two extremes by minimizing computational complexity while requiring minimal additional hardware. Nonetheless, owing to the inherent nature of these hybrid techniques and their strict architectural requirements, their usability remains severely limited [8].

IoT devices are typically designed for specific tasks, with their computational resources tailored to meet the demands of those tasks. Introducing additional tasks or security routines on IoT devices may impede their routine operations to some extent, commonly referred to as affecting the *availability*. Ideally, executing a security protocol on a device should not compromise its availability, particularly for devices responsible for real-time or safety-critical applications. However, the majority of attestation techniques necessitate uninterrupted execution of their security routines, which often consume significant time to complete. Consequently, this results in diminished availability of an IoT device. Furthermore, beyond concerns regarding availability, computational resources, and hardware requirements, many of these techniques mandate access to a genuine copy of a device's firmware. However, this may not always be feasible as most manufacturers and service providers consider their device firmware as IP. Consequently, if the firmware is inaccessible, most existing attestation techniques become impractical.

In recent years, machine learning (ML) based techniques have found successful applications across various fields, including medical, industrial, military, and social sciences. These techniques have also been leveraged for IoT device authentication [8], [9]. The majority of ML methods employed thus far operate within the supervised learning framework, wherein training a model requires samples from all expected classes. During testing, the model determines the closest class distribution for a new sample and classifies it accordingly. While supervised ML-based methods have been effective for device attestation [8], [9], they exhibit a significant limitation: acquiring and labeling training datasets comprehensively covering the various attack vectors encountered by devices during their operation. This task is exceptionally challenging due to the emergence of new attack vectors regularly, making comprehensive coverage unfeasible. Consequently, if a model encounters a test sample significantly different from all trained classes, its decision may be unreliable.

To tackle the aforementioned challenges, this paper introduces a software-based device attestation framework employing unsupervised deep learning (DL) methods. Unlike traditional approaches, our method imposes no additional computational requirements on the device under scrutiny, thereby preserving device availability. Additionally, instead of relying on the genuine firmware of the prover, our framework trains a Conditional Variational Autoencoder (C-VAE) model [10] using features extracted from an IoT device's main memory. This approach enables the detection of even subtle alterations to a device's firmware.

To summarize, our major contributions are:

1) A software based attestation framework which uses the preprocessed and reduced RAM trace features without requiring expensive computations on the device under test.
2) Two device tampering detectors using the C-VAE model trained under the representation learning paradigm.
3) Evaluation of the proposed framework on actual prototype devices, with eight different applications, affected by three different attack types.

The rest of the paper is organized as follows: In Section II we provide a discussion on related works. The preliminary information about the system model and information flow through different entities discussed in the model are provided in Section III. The proposed detection framework is presented in Section IV. Data acquisition and prototype setup for testing the proposed framework is discussed in Section V, followed by experimental evaluation in Section VI. Section VII concludes the paper.

II. Related Work

The classical software-based attestation approaches include SWATT [11], SCUBA [12], and SAKE [13]. SWATT computes a hash through multiple iterations on the prover's memory, imposing substantial computational demands on the device. Typically, around 50,000 iterations are necessary for effective adversary detection. Similarly, SCUBA utilizes a comparable technique for device authenticity verification. SAKE, proposed in [13], presents an alternative protocol for attestation, offering a means to establish a shared key across neighboring sensor nodes, even in the event of device compromise. These approaches share the assumption of computationally-limited devices and require precise time measurement and sophisticated procedures. In contrast, [14] suggests using partial memory checksum for attestation to reduce computational costs. However, this method necessitates full control of the device to execute the attestation routine uninterrupted, requiring a large number of cycles and resulting in poor availability. In [15], the authors propose a software-based remote attestation. They introduce a delayed observation mechanism to mitigate inherent limitations and propose a "filling memory at attestation-time" approach to counteract potential exploitation by attackers. Additionally, the implementation of a reputation mechanism and load balancing principle significantly enhances attestation efficiency and system security. Similarly, authors in [16] utilize side-channel observables such as light and sound to attest the IoT device.

Most of the techniques discussed above require a genuine copy of the firmware stored in the flash memory of the devices, which is not ideal in many cases, as discussed in the previous section. The features utilized in the proposed framework are computed using device RAM traces, which are not only smaller than the respective flash memory but also computed outside the device, requiring only a single trace sample instead of multiple iterations over the entire trace. This results in minimal computational and communication overhead on the device itself.

Hardware-based attestation techniques, such as those described in [17]–[21], do not impact device availability. However, these methods necessitate the presence of a Trusted Platform Module (TPM) in sensor nodes, which may not always

be feasible in resource-constrained IoT devices. Protocols outlined in [17], [18] designate the node with the TPM as a cluster head to form multiple node clusters, delegating node integrity verification within a specific cluster to its cluster head. The on-board TPM within the cluster head is tasked with verifying the integrity of the cluster head node and serves as the trusted root. Nevertheless, in this scenario, the failure of the cluster head can result in attestation failure for all nodes encompassed in the managed cluster. In [22], the authors introduce a novel attestation technique leveraging Quantum Physical Unclonable Functions (QPUFs) to enhance security. By exploiting the unique properties of quantum mechanics, it offers heightened security measures. It utilizes quantum superposition to simultaneously attest multiple memory locations, effectively guarding against mobile malware. Meanwhile, our proposed attestation framework does not necessitate any specialized hardware and only uses a single memory trace to authenticate the IoT device.

The methodologies proposed in [23]–[29] fall into the hybrid category for IoT device attestation. However, not only do they fail to ensure high availability, but they are also highly vulnerable to the threat posed by roaming malware. Authors in [26] introduce a hybrid attestation protocol by incorporating roaming malware into their threat model. This protocol is closely related to the architecture proposed in [23], albeit it relaxes the atomicity constraint of [23] to enable uninterrupted attestation routines, computing the hash digest of the device's full memory. Consequently, this leads to heightened computational requirements and execution times. Furthermore, it diminishes device availability, and its unique architectural requirements make it is unsuitable for low-cost devices. In contrast, our proposed framework does not necessitate disabling interrupts and requires only a single copy of the RAM trace instead of flash memory to conduct attestation. Furthermore, as our framework relies solely on RAM traces, it can detect roaming malware, which may alter its presence within the infected device's flash memory [26].

The closest work to our proposed framework is [8], where the introduction and feasibility of utilizing IoT device RAM traces for device attestation were presented. They utilized RAM traces to generate two distinct features, subsequently training several binary classifiers under a supervised learning setting and demonstrating excellent detection performance. However, in our experimental section, we illustrate that even without feature engineering, several supervised learning-based ML methods trained on raw RAM traces are able to achieve 99 - 100% detection accuracy. However, we elucidate why this can be misleading and argue that supervised learning may not be the optimal detection criterion for any detection problem, unless all potential classes a test sample can belong to are included in the training set.

To summarize, currently available methods for device attestation face several limitations: a) the need for a genuine firmware copy, b) the requirement of additional hardware, c) significant computational overhead, and d) the necessity to cover all possible attack vectors for training ML classifiers. The proposed device attestation framework addresses the issues a) and b) by utilizing on-device RAM to generate features, c) by offloading the computational work associated with the novel feature preprocessing pipeline to the gateway node of the IoT ecosystem, and d) by employing a representation learning-based detector trained solely on genuine device RAM features. This approach results in a detector capable of classifying tampered devices with 100% accuracy. A summary of the techniques discussed above and their requirements are presented in Table I.

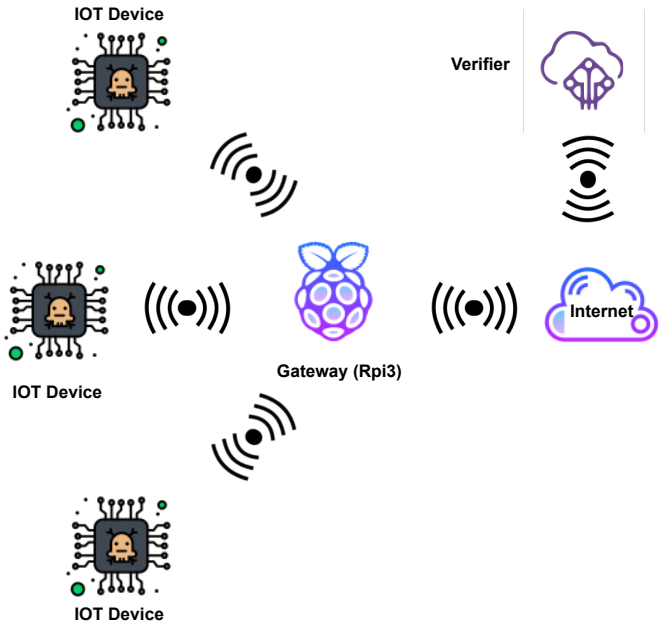

Fig. 1. The proposed system model.

III. Preliminaries

In this section we first introduce the underlying system model requirements of the proposed attestation mechanism, followed by discussion about the overall flow of information between the system's entities.

A. System Model

The system model at the heart of our framework is shown in Fig. 1, consisting of the following three entities:

1) **The IoT Device:** A number of IoT devices are interfaced with a gateway node using wired or wireless connections. These devices are characterized by limited battery life, memory, and computational capabilities, serving as the *provers* in an attestation query.
2) **The Gateway:** Typically, IoT devices, constrained by resources, lack a complete TCP/IP protocol stack and rely on an intermediary node for routing and internet connectivity, known as a *Gateway*. In our system model, this comparatively more powerful node fulfills two primary functions: (i) serving as a router and device manager for connected IoT devices, facilitating communication between IoT devices and a central entity (*verifier*), (ii) extracting features by applying the proposed preprocessing pipeline to incoming binary RAM trace dumps (details in Section IV). The gateway node connects to the verifier

TABLE I
COMPARATIVE ANALYSIS OF EXISTING LITERATURE

Requirements	[11]–[13]	[14]	[17]	[18], [26]	[23]	[24], [25]	[27]	[15]	[16], [21], [22], [30]	[29]	[31]	[8]	Proposed
Checksums?	✓	✗	✗	✓	✓	✓	✓	✗	✗	✓	✗	✗	✗
Precise Time Stamps?	✓	✓	✗	✗	✓	✓	✓	✓	✗	✗	✗	✗	✗
Disable Interrupts?	✓	✓	✗	✗	✓	✓	✓	✗	✗	✗	✗	✗	✗
Additional Hardware?	✗	✗	✓	✓	✗	✓	✗	✗	✓	✓	✓	✗	✗
Low Availability?	✓	✓	✗	✗	✓	✓	✓	✓	✗	✓	✗	✗	✗
Genuine Firmware?	✓	✓	✓	✓	✓	✓	✓	✗	✗	✗	✓	✗	✗
Attack Data?	✗	✗	✗	✗	✗	✗	✗	✗	✗	✗	✗	✓	✗

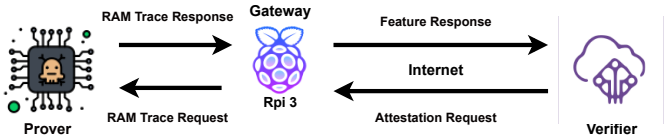

Fig. 2. Overall data flow under the proposed attestation framework.

system via the internet and operates in either *train* or *test* mode.

3) **The Verifier:** Represents a trusted remote server tasked with IoT device attestation during bootup or normal operation. It maintains records of all connected gateway nodes along with the device IDs of all subsequent IoT devices linked under each gateway node. Additionally, it is responsible for maintaining the detection models for device attestation and operates in either *train* or *test* mode.

B. Information Flow

The flow of information, whether the mode of operation is *train* or *test*, is depicted in Fig. 2. Beginning with the prover, upon receiving an attestation request, it captures a snapshot of its current working memory and transmits it to the gateway node. This RAM trace dump is composed of a 1D array of binary values saved in HEX format. By combining consecutive HEX digits, each representing 1 byte, the data is encoded into decimal values within the range of $[0, 255]$ (8-bit unsigned integers), followed by normalization through division by 255.0. This process transforms the 1D array of binary values into a 1D array where each entry falls within the range of $[0, 1.0]$. Depending on the operation mode, this trace undergoes one of the two preprocessing pipelines at the gateway node to generate a feature vector (detailed in Section IV-A). Subsequently, the resulting feature vector is transmitted to the verifier system. In training mode, the verifier utilizes the received feature set to initiate model training. Conversely, in testing mode, the received feature is subjected to detection algorithms and thresholding to classify the device as genuine or tampered with. Further details on each step discussed above are elaborated in Section IV.

C. The RAM Trace

The contents stored in a device's RAM at any given point during its execution are directly correlated with its ongoing operations. Typically, the RAM of an embedded device can be categorized into the following four main sections:

i. `.text`: This section contains the executable code.
ii. `.data`: Global and initialized static variables are stored here.
iii. `.bss`: Memory allocation for uninitialized static and global variables is managed in this section.
iv. `Heap/Stack`: Reserved for dynamic memory allocation, return from function calls, interrupts, and storage of local variables.

The premise is that modifications to a device's firmware will be reflected in that device's RAM trace. Depending on the magnitude of firmware changes, these alterations can range from minor to significant and may manifest in various sections of the RAM. For instance, the introduction of new pointers into the control dependency graph of the executable code can impact all four memory areas of a device. Furthermore, the detectability of changes reflected in the main memory is influenced by the nature of tampering. While substantial alterations are generally easier to detect, subtle tampering may result in minimal or localized variations in the RAM trace, posing greater difficulty in detection.

D. The Attack Model

In this work, we consider a sophisticated and resourceful adversary (*Adv*) whose goal is to remain undetected within an IoT device's memory. The following assumptions are made regarding the capabilities and limitations of the *Adv*:

- **Physical Access and Firmware Modification:** The *Adv* possesses physical access to the IoT device and can modify its firmware. This allows for a wide range of malicious activities, such as injecting unauthorized code or altering operational parameters.
- **Packet Manipulation:** The *Adv* has the ability to replay, tamper with, and eavesdrop on packets transmitted by the IoT device. This capability enables the attacker to manipulate communication and possibly disguise malicious activity as legitimate traffic.
- **Memory Access and Manipulation:** The *Adv* can interrupt and alter the device's memory at any time, providing opportunities to modify data and control the device's behavior dynamically.
- **Persistence in Memory:** Despite its capabilities, the *Adv* cannot remove itself from the device's memory. It remains persistently embedded in the system, which makes detecting its presence crucial for effective attestation.

- **Immutable Attestation Routine:** The attestation process on the IoT device is secure and immutable. When the gateway node issues an attestation query to request the device's runtime RAM contents, the *Adv* is unable to modify the state of the RAM to evade detection. This ensures the integrity of the attestation process.
- **Secure Communication:** The gateway node and IoT device communicate through a secure channel, safeguarded by standard security protocols [32]. This secure communication prevents the Adv from compromising the attestation exchange and ensures the authenticity and confidentiality of the data being transmitted.

IV. The Proposed Framework

In this section, we present our proposed framework, beginning with the introduction of a preprocessing pipeline designed to mitigate redundancy across training samples, thereby facilitating effective model training. Subsequently, we discuss the core component of our framework: a Conditional Variational Autoencoder (C-VAE) model. Leveraging a trained C-VAE model, we setup two detectors and derive two decision metrics capable of flagging input samples as genuine or tampered. Further details regarding these aspects are provided in the subsequent subsections.

A. Feature Generation

Our primary aim is to devise an IoT device attestation framework that utilizes the RAM trace acquired from the device under test as its primary fingerprint for detection. During normal device operation, the majority of RAM content remains unchanged, with only a few locations potentially being updated depending on the application. Consequently, in the event of device firmware tampering, the resulting variations in RAM content are expected to differ from those observed during normal operation. However, in both scenarios—whether with genuine firmware or tampered—most entries in the RAM traces are anticipated to remain unaltered. To facilitate effective downstream model training, these RAM traces must undergo preprocessing to reduce redundancy while preserving distinctive information as much as possible. This approach enables the models to focus on data variations rather than baseline trends that may be present across training samples.

Consider a dataset $\mathbf{X} \in \mathbb{R}^{m \times n}$ comprised of m RAM traces containing n samples each. For a rectangular $\mathbf{X}$ with dimensions $m \neq n$ (or square with $m = n$) and rank $r \leq \min(m, n)$, we can decompose this matrix into three component matrices using the reduced singular value decomposition (r-SVD) [33] as follows:

$$\mathbf{X} = \mathbf{U}\mathbf{S}\mathbf{V}^\top, \tag{1}$$

where $\mathbf{U} \in \mathbb{R}^{m \times r}$ and $\mathbf{V} \in \mathbb{R}^{n \times r}$ are semi-unitary matrices, i.e., $\mathbf{U}^\top\mathbf{U} = \mathbf{V}^\top\mathbf{V} = \mathbf{I}_r$, $\mathbf{S} \in \mathbb{R}^{r \times r}$ is the diagonal matrix containing the unique singular values $\sigma_1 \geq \sigma_2 \geq \ldots \sigma_r > 0$ on its main diagonal, and r is the matrix rank. The left singular vectors of $\mathbf{U}$ form an orthonormal basis of column-space of $\mathbf{X}$ ($\mathcal{C}(\mathbf{X})$) and the right singular vectors of $\mathbf{V}$ form the orthonormal basis of the row-space of $\mathbf{X}$ ($\mathcal{R}(\mathbf{X})$). The computed singular values are unique for $\mathbf{X}$ and represent the amount of variance explained by each corresponding left and right singular vectors across the entire dataset $\mathbf{X}$. This relationship can be clearly observed through the outer product form of the SVD, given as:

$$\mathbf{X} = \sum_{i=1}^{r} \sigma_i \mathbf{u}_i \mathbf{v}_i^\top, \tag{2}$$

where $\mathbf{u}_i$ and $\mathbf{v}_i$ are the ith left and right singular vectors, respectively. For visualization, we present the first 200 singular values computed from the RAM traces obtained from various applications (details in Section V) in Fig. 3. Examination of the very first singular value reveals that it alone is capable of representing over 82% of the total variation present in their respective RAM trace datasets.

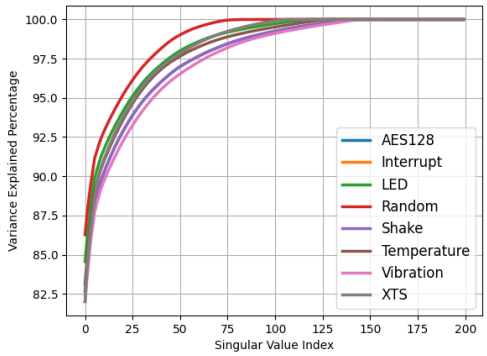

Fig. 3. Initial 200 singular values from different firmware datasets.

Aligning with the intuition prevalent in machine learning, which suggests removing common information across all training samples before training, we choose to eliminate the first rank-1 outer product from our dataset $\mathbf{X}$. Additionally, as depicted in Fig. 3, we observe that more than 99.5% of the data variation is already explained by the initial $\gamma = 175$ components. Hence, we can reduce the feature size of the RAM trace without significant loss of information. Consequently, the removal of the first component will primarily reduce redundancy within $\mathbf{X}$, while eliminating components beyond γ will enable us to focus on directions capturing the remaining data variation and removing noisy components. As the individual RAM traces are kept as row vectors in $\mathbf{X}$, we use $\hat{\mathbf{V}} = \mathbf{V}_{2:\gamma} \in \mathbb{R}^{n \times (\gamma-1)}$, containing $[2 : \gamma]$ right singular vectors of $\mathbf{V}$, as the projector to reduce $\mathbf{X}$ as:

$$\hat{\mathbf{X}} = \mathbf{X}\hat{\mathbf{V}} = \mathbf{U}\mathbf{S}\mathbf{V}^\top\hat{\mathbf{V}} = \mathbf{U}_{2:\gamma}\mathbf{S}_{2:\gamma}. \tag{3}$$

We call the subspace spanned by the rows of $\hat{\mathbf{V}}$ as the training subspace, where all our training data is mapped. With the above transformation, the reduced matrix $\hat{\mathbf{X}}$ can be used to train the ML models effectively. In the proposed framework, feature generation is the responsibility of the gateway node shown in Fig. 2. However, the computations carried out are different when operating in the training mode or testing mode (inference). The difference is discussed below:

1) Training Mode: The gateway node assumes responsibility for acquiring and processing the data required for training the model, while the actual model training takes place at the verifier node. During the training mode of operation, the gateway node gathers a set of RAM traces from all connected IoT devices, storing them as $\mathbf{X}_i \in \mathbb{R}^{m \times n}$ matrices, with i representing the index of the respective IoT device. Once the necessary number of traces have been acquired and the matrices are configured, the gateway computes the r-SVD of each dataset $\mathbf{X}_i$ individually. Subsequently, it utilizes the corresponding projection matrix $\hat{\mathbf{V}}_i$ to generate the feature matrix $\hat{\mathbf{X}}_i$, as described in Equation (3). The gateway stores the projection matrices corresponding to each IoT device for future use and forwards the processed feature matrices from all devices to the verifier system.

2) Testing Mode: In testing mode, the gateway node receives a RAM trace, call it $\mathbf{x}_i \in \mathbb{R}^n$, from the i^{th} device. It then uses the projection matrix $\hat{\mathbf{V}}_i$ to project $\mathbf{x}_i$ onto the subspace spanned by the rows of $\hat{\mathbf{V}}_i$ (the training subspace) to generate the feature vector $\hat{\mathbf{x}}_i = \hat{\mathbf{V}}_i^\top \mathbf{x}_i$, existing in the training subspace. By doing so, we remove the redundant information and keep the variations along those directions which were used to train the model. This transformed feature vector is then forwarded to the verifier for attestation.

Algorithm 1: Preprocessing at the Gateway node.

Training Mode:

Input: RAM traces: $\mathbf{X}_i$, highest component to keep: γ, total number of devices: I.

Create two empty datasets $\mathcal{X}$ and $\mathcal{V}$.
for *i in I* **do**
  Compute r-SVD of $\mathbf{X}_i$ as
  $[\mathbf{U}_i, \mathbf{S}_i, \mathbf{V}_i] = \text{r-SVD}(\mathbf{X}_i)$
  Create $\hat{\mathbf{V}}_i = \mathbf{V}_{i,2:\gamma}$
  Generate $\hat{\mathbf{X}}_i$ using (3)
  Append $\hat{\mathbf{X}}_i$ to the training dataset $\mathcal{X}$
  Append $\hat{\mathbf{V}}_i$ to the projection dataset $\mathcal{V}$

Output: Training dataset: $\mathcal{X}$ forwarded to the Verifier. Projection dataset: $\mathcal{V}$ saved at the gateway.

Testing Mode:

Input: A RAM trace: $\mathbf{x}_i$ from i^{th} device, projection dataset: $\mathcal{V}_i$

Compute $\hat{\mathbf{x}}_i = \hat{\mathbf{V}}_i^\top \mathbf{x}_i$

Output: Forward $\hat{\mathbf{x}}_i$ to verifier for inference.

The operations performed at the gateway node are summarized in Algo. 1 covering both training and testing modes.

B. Proposed Detection Model

As previously discussed, supervised learning necessitates training data comprising samples from all the different classes that the model might encounter during its operations. Models trained on such comprehensive datasets can effectively recognize test samples that share similar features with their training counterparts. However, these models often struggle when confronted with test samples dissimilar to any of the class samples they were trained on. For the IoT device attestation problem, our objective is to train a model capable of not only performing well on samples (RAM traces) seen during training but also robustly recognizing previously unseen samples. In the event of a novel attack, our model should be capable of flagging it as potentially malicious.

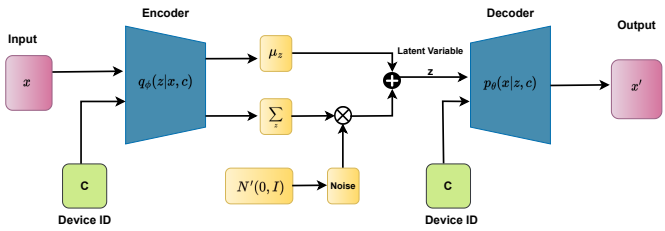

Fig. 4. The Conditional Variational Autoencoder model architecture.

Representation learning-based training frameworks, such as Variational Autoencoders (VAE) [34], Generative Adversarial Networks (GAN) [35], and Deep Belief Networks (DBN) [36], offer a promising approach in this context. These models rely on profiling techniques, where they are trained solely on normal data samples. Once trained, they evaluate whether a test sample conforms to the learned distribution of normal data. If the test sample deviates significantly from this distribution, it is flagged as potentially anomalous. This characteristic makes these models valuable tools for identifying previously unseen attack instances.

This work harnesses the capabilities of a VAE, a deep learning model, for detecting firmware tampering in IoT devices. By training the VAE with data gathered from genuine IoT devices, the model acquires insights into the inherent latent patterns and characteristics defining these devices. Consequently, the VAE becomes effective at distinguishing between test samples similar to genuine patterns and those exhibiting anomalies, facilitating the identification of previously unseen test samples. Using the Artificial Neural Networks (ANNs) as the base learners, our model architecture comprises an Encoder ($\mathcal{E}_{\boldsymbol{\phi}}$) and a Decoder ($\mathcal{D}_{\boldsymbol{\theta}}$) network, implemented through two ANNs interconnected and operating within the VAE configuration [34].

1) The Encoder: The encoder model receives the feature vector $\mathbf{x}$, containing the preprocessed RAM trace sample outlined in Section IV-A, alongside a one-hot-encoded device ID $\mathbf{c}$ as the auxiliary information. The device ID $\mathbf{c}$ serves a dual function: it mitigates mode collapse and guides the encoder to map samples from diverse classes symmetrically closer to the origin of the latent space. This mechanism ensures that the encoder's output can serve as a robust decision metric (discussed further in Section IV-C). The encoder generates two output vectors of equal dimensions: a mean vector $\boldsymbol{\mu}_\mathbf{z}$ and a variance vector $\boldsymbol{\Sigma}_\mathbf{z}$, tailored to the latent space's dimensions. These vectors collectively define a latent probability distribution $q_{\boldsymbol{\phi}}(\mathbf{z}|\mathbf{x}, \mathbf{c})$, parameterized by the encoder. During training, the objective is to enable the encoder to map its inputs into a continuous latent space, where each input corresponds to a region in the latent space rather than a single point [34].

This facilitates the subsequent generation of samples by the decoder, ensuring they are not mere replicas of their input counterparts.

2) The Decoder: The primary objective of the decoder model is to regenerate the input to the encoder model using solely the latent variable as its input. The decoder accepts the latent variable $\mathbf{z}$, sampled from the latent probability distribution $q_{\boldsymbol{\phi}}(\mathbf{z}|\mathbf{x},\mathbf{c})$, utilizing the reparameterization trick as proposed by [34]:

$$\mathbf{z} = \boldsymbol{\mu}_{\mathbf{z}} + \boldsymbol{\Sigma}_{\mathbf{z}} \odot \boldsymbol{\epsilon}, \tag{4}$$

where $\boldsymbol{\epsilon} \sim \mathcal{N}(\mathbf{0}, \mathbf{I})$ and $\odot$ denotes element-wise multiplication. This technique facilitates gradient backpropagation through the sampling process into the encoder model, ensuring the preservation of the probabilistic nature of the sampling. Leveraging $\mathbf{z}$ and the device ID $\mathbf{c}$, the decoder is trained to reconstruct the original input $\mathbf{x}$ by sampling from the distribution $p_{\boldsymbol{\theta}}(\mathbf{x}|\mathbf{z},\mathbf{c})$ (parameterized by the decoder). Training exclusively on genuine data enhances the proficiency of both the encoder and decoder in reconstructing samples that conform to the training data distribution.

3) The Loss Function: Consider a data vector denoted as $\mathbf{x}$ and its corresponding label $\mathbf{c}$. Initially, a probabilistic encoder transforms these inputs into a latent vector $\mathbf{z}$ according to the distribution $q_{\boldsymbol{\phi}}(\mathbf{z}|\mathbf{x},\mathbf{c})$. Subsequently, a probabilistic decoder reconstructs $\mathbf{z}$ to yield $\hat{\mathbf{x}}$ using $p_{\boldsymbol{\theta}}(\mathbf{x}|\mathbf{z},\mathbf{c})$. The encoder and decoder networks are trained concurrently by maximizing the ensuing loss function:

$$\begin{aligned}\mathcal{L}_{VAE}(\boldsymbol{\theta},\boldsymbol{\phi};\mathbf{x},\mathbf{z},\mathbf{c}) = \mathbb{E}_{\mathbf{z}\sim q_{\boldsymbol{\phi}}(\mathbf{z}|\mathbf{x},\mathbf{c})}(log\ p_{\boldsymbol{\theta}}(\mathbf{x}|\mathbf{z},\mathbf{c})) \\ - D_{KL}(q_{\boldsymbol{\phi}}(\mathbf{z}|\mathbf{x},\mathbf{c})||p_z(\mathbf{z})).\end{aligned} \tag{5}$$

Equation (5) denotes the variational lower bound [34]. The initial term in (5) represents the log likelihood function, while the subsequent term signifies the latent loss employing Kullback-Leibler Divergence (KLD) between the learned latent distribution $q_{\boldsymbol{\phi}}(\mathbf{z}|\mathbf{x},\mathbf{c})$ and a prior distribution $p_z(\mathbf{z})$. The KLD serves as a regularizer to impart structure onto the latent distribution. In its current form, $q_{\boldsymbol{\phi}}(\mathbf{z}|\mathbf{x},\mathbf{c})$ is intractable, necessitating the estimation of the KLD between the latent and the selected prior [34]. However, assuming $q_{\boldsymbol{\phi}}(\mathbf{z}|\mathbf{x},\mathbf{c})$ as Gaussian with an approximately diagonal covariance, and $p_z(\mathbf{z})$ as a unit Gaussian, facilitates the computation of the KLD without estimation [34]. Furthermore, assuming a Gaussian $p_{\boldsymbol{\theta}}(\mathbf{x}|\mathbf{z},\mathbf{c})$, (5) can be written as

$$\begin{aligned}\mathcal{L}_{VAE}(\boldsymbol{\theta},\boldsymbol{\phi};\mathbf{x},\mathbf{c}) \simeq \frac{1}{2}\sum_{i=1}^{N}\sum_{j=1}^{J}(1 + log(\sigma_{ij}^2) - \mu_{ij}^2 - \sigma_{ij}^2) \\ - \frac{1}{N}\sum_{i=1}^{N}\|\mathbf{x}_i - \hat{\mathbf{x}}_i\|_2,\end{aligned} \tag{6}$$

here N is the total number of batch samples, J is the dimension of $\mathbf{z}$, μ_{ij} is the j^{th} element of $\boldsymbol{\mu}_i$, σ_{ij} is the j^{th} element of $\boldsymbol{\Sigma}_i$, and $\|\cdot\|_2$ is the vector ℓ_2-norm. The model is trained by maximizing the loss function given in (6).

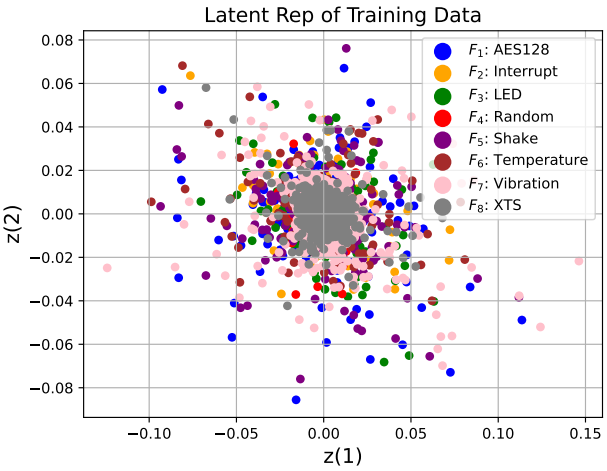

Fig. 5. Latent representation for all training samples with $\alpha = 2$.

C. Detection Methods

In this section, we present two detection methods leveraging the trained C-VAE model. The concept is straightforward: these methods calculate test statistics by measuring the similarity between the test sample and the training data. By carefully selecting thresholds, these test statistics determine whether samples are classified as genuine or tampered. Subsequent sections detail the computation of these test statistics and thresholds.

1) The Latent Variable based Detector D_{LV}: The encoder network, denoted as $\mathcal{E}_{\phi}$, plays a crucial role in our initial detection approach. Its primary aim is to map genuine input features into a latent space centered at the origin and symmetrically distributed around it. This constraint is enforced through the KLD in (5). Consequently, RAM trace samples obtained from tampered devices would display significant deviations from the origin of latent space. To establish a baseline for our detection, denoted as $\mathbb{E}[\boldsymbol{\mu}_{z_g}]$, we compute the expected value of the mean vector $\boldsymbol{\mu}z_g$, where $\boldsymbol{\mu}_{z_g}, \boldsymbol{\Sigma}_{z_g} = \mathcal{E}_{\boldsymbol{\phi}}(\mathbf{X}_g, \mathbf{C}_g)$, illustrated in Fig. 4. Here, $\mathbf{X}_g$ represents the genuine training samples, and $\mathbf{C}_g$ denotes their corresponding device ID labels.

Our test statistic for a given test sample $\bar{\mathbf{x}}$ is computed as $\zeta_{LV}(\bar{\mathbf{x}}) = \|\boldsymbol{\mu}_{z_{\bar{\mathbf{x}}}} - \mathbb{E}[\boldsymbol{\mu}_{z_g}]\|_2$. For visualization, we show an example of the 2D latent representation generated by the trained encoder for all training samples in Fig. 5 which shows that $\mathbb{E}[\boldsymbol{\mu}_{z_g}]$ closely approximates to the origin. Subsequently, for a predefined threshold ρ, if $\zeta_{LV} < \rho$, the sample is classified as genuine; otherwise, it is classified as tampered. This detection method, is termed as D_{LV}.

2) The Reconstruction Error based Detector D_{RE}: The decoder network, denoted as $\mathcal{D}_{\theta}$, serves a crucial role in our alternative detection model. Its objective is to generate samples that resemble those from the genuine sample distribution, accomplished by maximizing the objective functions in (6). When a test sample $\mathbf{x}$, along with its corresponding test device ID $\mathbf{c}$, passes through the $\mathcal{E}_{\phi}$ model, it produces a latent variable $\mathbf{z}$ (computed using (4)). Then, $\mathbf{z}$ and $\mathbf{c}$ are fed into $\mathcal{D}_{\theta}$ to generate the reconstructed sample $\hat{\mathbf{x}}$. For genuine test samples,

the model should accurately reconstruct them. However, when dealing with samples from tampered devices, the model may struggle to replicate them due to its lack of exposure to such samples during training.

Under this premise, we calculate a test statistic for a given test sample $\bar{\mathbf{x}}$ and its reconstruction $\hat{\bar{\mathbf{x}}}$ as $\zeta_{RE} = \|\bar{\mathbf{x}} - \hat{\bar{\mathbf{x}}}\|_2$. Subsequently, employing a predefined threshold ρ, if $\zeta_{RE} < \rho$, the test sample is classified as genuine; otherwise, it is designated as tampered. This detection method is termed D_{RE}.

3) Threshold Computation: Determining an appropriate threshold is pivotal to ensure the effectiveness of the detectors outlined above. The threshold ρ is selected based on a predefined acceptable False Positive Rate (FPR) tolerance. Post model training, we compute the two test statistics (ζ_{LV} and ζ_{RE}) for the entire training dataset and establish their respective thresholds. These thresholds are calibrated to align with the specified FPR for the training dataset. Upon receiving a test sample $\bar{\mathbf{x}}$, each detection criterion independently marks it as tampered if it surpasses its assigned threshold.

V. Experimental Setup

A. Hardware Description

Our experimental setup for evaluating the proposed detection framework consists of three main components: an IoT device, a gateway node, and a remote verifier system. We setup a Raspberry Pi 3 (Rpi3) as the gateway, equipped with a Broadcom BCM2837 64-bit CPU operating at 1.2 GHz. This gateway features 4 onboard USB ports, supporting connections to up to 4 individual IoT devices simultaneously. For wireless communication, the gateway employs the BCM43438 WLAN board to establish TCP/IP connectivity with the remote verifier. The Rpi3 operates on a 32 GB micro SD card and runs the ARM port of the Debian Stretch operating system. In our prototype, the gateway node is responsible for three primary functions:

i. Collection of RAM traces from the IoT device for training or attestation upon the verifier's request.
ii. Execution of the preprocessing routine on the received traces, as outlined in Algorithm 1 and Section IV-A, followed by forwarding the processed features to the verifier.
iii. Maintenance of a wired connection (via serial port) or wireless connection with the IoT devices and the TCP/IP connection with the remote verifier.

To ensure the representativeness of a diverse array of IoT devices, our prototype setup incorporates a widely utilized microcontroller-based device package, namely the Arduino Uno. This selection is motivated by the versatility of the Arduino platform, which supports analog-to-digital converter (ADC) inputs, facilitating the integration of various sensing modules such as sound, temperature, or light sensors. These modules can be interfaced using serial peripheral interface (SPI) or inter-integrated circuit interface (I2C), thereby encompassing a significant portion (upto 99%) of the application market [37]. The Arduino Uno used in our setup features an 8-bit ATmega328P AVR-based microcontroller with 32 KB of flash memory and 2 KB of RAM. Serial communication with the gateway node is established via the Universal Asynchronous Receiver Transmitter (UART) TTL, with data exchange occurring at a baud rate of $115,200$. Moreover, the direct addressing mode is employed to capture the RAM content from addresses `0x0101` to `0x08FF`.

B. Firmware Description

To cover a large number of IoT use cases, we test our proposed framework against eight different application types with diverse task objectives, ranging from sensing to cryptography. For notational ease, the developed firmware are represented as F_i, with $i = 1, 2, \ldots, 8$. The specific operations coded in these firmware are discussed below:

F_1: A given input data block is first encrypted and then decrypted to retrieve the original data. The AES128 algorithm is used in this process and both encryption and decryption is performed in different functions, executed iteratively.
F_2: Push-button interrupts are linked to the built-in button's IO PIN-2. An Interrupt Service Routine (ISR) function is designated to toggle the state of the built-in LED on PIN-13 according to the button pin's state. The button's state is stored in a variable and then transmitted to the LED pin. Interrupts are sampled every 3 seconds before capturing the trace data.
F_3: Based on the analog value read from the sensor at PIN-3, it changes the brightness of the built-in LED connected to PIN-13. The LED control method is defined separately and invoked in the Arduino loop method followed by capturing the trace data.
F_4: Based on the serial seed received from the gateway, a pseudo random number is generated.
F_5: The distance between an ultrasound sensor and an obstacle is measured while the device is moved laterally.
F_6: The voltage is read at the temperature sensor's input PIN- and converted to the respective Celsius value. The loop method of the Arduino is used to call a method to read the temperature and store the read value.
F_7: The vibration sensor application activates the built-in LED (PIN-13) upon detecting vibrations. A function is defined to read digital values from the vibration sensor (PIN-7) and feed them as input to the LED. This function is invoked within the Arduino loop method before initiating trace collection.
F_8: The XTS-AES block cipher with a variable block cipher encryption is used here. The plain text, key, and tweak modules are used for encryption. For simplicity, the read input, key, and tweaks are sent from the gateway node.

We observe that F_1 and F_8 are cryptography-based applications and the rest are variations of sensor based applications. By including such wide spectrum of applications, we aim to evaluate our proposed framework against various real-world application scenarios.

With the baseline firmware applications established, we proceed to introduce specific attacks on these applications.

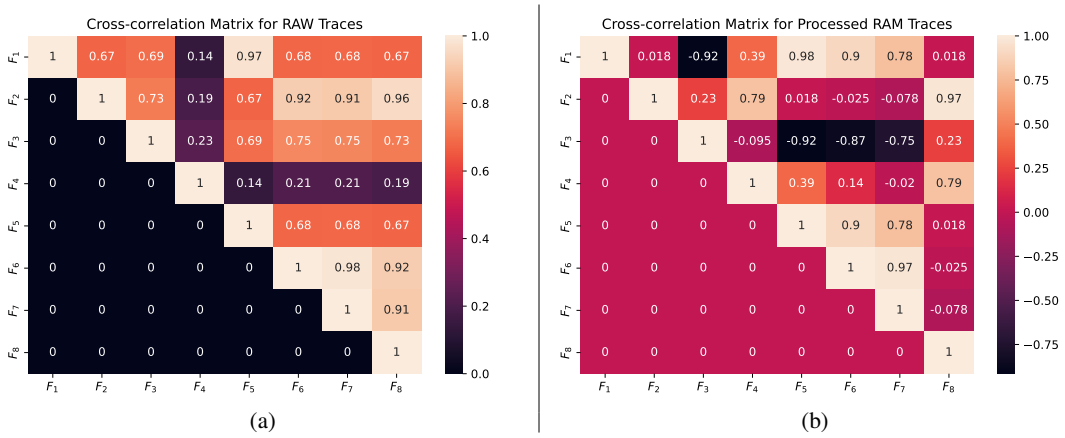

Fig. 6. Correlation matrix for both (a) raw and (b) processed genuine firmware datasets.

The following outlines three distinct attack variations we implemented to test the system's effectiveness:

- A_1: **Attack via Control Dependency Graph:** Under this attack scenario, we introduce new pointers to alter the control dependency graph of the device. This attack mainly affects the `.data` section of the RAM.
- A_2: **Attack via Functional Dependency:** Here we setup a new stack frame which affects the `stack` section of the RAM.
- A_3: **Attack via Variable Initialization:** Here we alter the code for variable initialization which results in changes in `.bss` section of the RAM.

With the attacks discussed above, we have tried to cover the most critical and fundamental attacks considering the three types of dependencies when it comes to good software engineering practices [38]. In doing so, we cover majority of the tampering attacks carried on embedded devices.

TABLE II
ACRONYMS AND THE NUMBER OF ATTACK AND GENUINE SAMPLES ACQUIRED FOR EACH FIRMWARE TYPE.

$\overrightarrow{Acronym}$	F_1	F_2	F_3	F_4	F_5	F_6	F_7	F_8
Type ↓	AES128	Interrupt	LED	Random	Shake	Temperature	Vibration	XTS
Genuine	1500	1500	1500	500	1500	1500	1500	1500
A_1	500	1500	500	100	500	500	500	1500
A_2	500	500	500	100	500	500	500	500
A_3	500	500	500	100	500	500	500	500
Total	3000	4000	3000	800	3000	3000	3000	4000

C. Dataset Generation

The next step involves capturing RAM traces from the prototype IoT device while it executes one of the specified firmware variants (F_i). Initially, we upload the firmware F_i onto the IoT device and allow it to operate for a predefined time period of 300 seconds. Throughout the device's normal operation, we utilize the gateway node to collect numerous RAM trace samples, each comprising 2048 bytes, with randomized intervals between consecutive samples. These samples are labeled as *genuine* and preserved for subsequent analysis. Once genuine RAM traces are obtained for all eight device firmware F_i, we proceed by uploading each attack variant of a particular firmware to obtain their corresponding tampered RAM traces. Consequently, we obtain three distinct sets of tampered RAM traces (A_1, A_2, A_3) for each firmware variant F_i. The specific count of samples acquired in this process is detailed in Table II.

To highlight the diversity of the genuine RAM trace data utilized for training the detection model, we computed the cross-correlation between all 8 application firmware using mean traces from both raw and processed datasets. The resulting scores are presented in Fig. 6. For raw RAM traces, only 7 out of 28 cross-correlation pairs exhibited scores above 0.9. Similarly, for processed RAM traces, 5 out of 28 cross-correlation pairs had scores above 0.9, while only 2 out of 28 had correlation scores below -0.9. It's important to note that these reported scores are based on mean RAM traces. Therefore, the prevalence of cross-correlation pairs with low scores highlights the diversity of the data employed in our analysis.

VI. Experimental Evaluation

In this section, we conduct a detailed analysis of our proposed firmware attestation framework. Following the acquisition of RAM traces, as outlined in Section V, all subsequent training and evaluations are conducted using Python v3.9 with the Scikit-Learn [39] and PyTorch [40] libraries. Before proceeding with the preprocessing pipeline detailed in Section IV-A, the RAM traces, initially ranging from $[0, 255]$, are normalized by dividing each value by 255.0, thereby scaling them to a range of $[0, 1.0]$.

The performance metrics presented in our analysis are classification-based, including Accuracy (ACC), True Positive Rate (TPR), also known as detection rate, and False Positive Rate (FPR), commonly referred to as the false alarm rate. Moreover, in the context of tampering detection, we designate tampered data as the positive class and genuine data as the negative class.

A. Binary Classification

To establish a baseline for our evaluation, we initially address the classic binary classification problem. Five commonly employed ML classifiers—Random Forest Classifier

TABLE III
TRUE POSITIVE RATES FOR EACH ML CLASSIFIER TRAINED ON RAW RAM TRACES FROM EACH FIRMWARE.

		F_1	F_2	F_3	F_4	F_5	F_6	F_7	F_8	
	Test DS ↓	AES128	Interrupt	LED	Random	Shake	Temperature	Vibration	XTS	MEAN
RFC	A_1	100.0	0.2	100.0	100.0	100.0	100.0	100.0	0.1	75.0
	A_2	0.0	100.0	100.0	100.0	0.0	0.0	100.0	100.0	62.5
	A_3	0.2	0.0	0.4	100.0	0.0	100.0	0.0	0.0	25.1
SVM	A_1	100.0	77.1	100.0	49.0	100.0	100.0	100.0	77.1	87.9
	A_2	100.0	100.0	100.0	0.0	100.0	1.0	100.0	100.0	75.1
	A_3	1.0	0.4	100.0	100.0	1.0	100.0	0.4	0.4	37.9
KNN	A_1	100.0	0.1	99.8	100.0	100.0	100.0	87.0	0.1	73.4
	A_2	0.0	100.0	75.0	100.0	0.0	0.0	100.0	100.0	59.4
	A_3	0.0	0.0	0.0	100.0	0.0	100.0	0.0	0.0	25.0
DTC	A_1	100.0	43.1	100.0	100.0	100.0	100.0	100.0	99.7	92.9
	A_2	0.0	100.0	100.0	100.0	0.0	0.0	100.0	100.0	62.5
	A_3	100.0	0.0	0.0	100.0	100.0	100.0	100.0	0.0	62.5
GBC	A_1	100.0	42.9	100.0	100.0	100.0	100.0	98.6	42.9	85.6
	A_2	0.0	100.0	100.0	100.0	0.0	0.0	100.0	100.0	62.5
	A_3	100.0	0.0	0.0	100.0	100.0	100.0	0.0	0.0	50.0

(RFC), Support Vector Machine (SVM), K-Nearest Neighbours (KNN), Decision Tree Classifier (DTC), and Gradient Boost Classifier (GBC) utilized. Each of the eight RAM trace datasets (F_i), along with their corresponding attack samples, is included in this analysis. The models are trained using their default parameters on each F_i individually, employing a train-test split of 75-25 without executing the preprocessing steps proposed in Section IV-A. Notably, the accuracy scores from this evaluation are remarkable, with RFC, DTC, and GBC achieving as high as 99.5% accuracy (on average) over the entire test set. Conversely, SVM and K-NN achieve an average accuracy of 95%. However, upon closer examination, it was evident that the test set contained samples closely resembling those in the training set. This outcome was expected as the samples were acquired while the device executed the same task, resulting in most features showing slight variations. Hence, we underscore that simple train-test split-based classification tests are unsuitable for such data.

To establish a robust baseline for our analysis, we evaluated the generalization capabilities of the ML classifiers using the Leave-One-Out (LOO) training strategy. This approach involved using the genuine and two attack datasets from a single firmware type to train the models, while reserving the remaining attack dataset as the test set. Consequently, we could assess the generalization capacity of each classifier by testing it on a fully unseen dataset, thus surpassing mere memorization of the training data and evaluating the models' ability to generalize effectively across diverse attack scenarios. Initially, we utilized the raw RAM traces from all datasets separately to train the classifiers 10 times, presenting their average TPR scores in Table III.

Analyzing the TPR reported in Table III reveals the inconsistent performance of the classifiers across different firmware types and attack datasets. Taking the DTC as an example, it achieved the highest average TPR across all firmware datasets. Notably, it successfully detected all three attack datasets for F_4 and F_7 at 100% accuracy. Moreover, its detection rates for A_1 and A_3 attack datasets were the highest on average. Conversely, the SVM classifier demonstrated the most accurate detection of the A_2 attack samples, averaging a 75% accuracy rate. Across different firmware types, all classifiers except the SVM classifier successfully identified all attacks against F_4. Conversely, attacks on F_2 and F_8 posed the greatest challenge, with these firmware types proving the most difficult to flag for all included classifiers. Notably, the A_3 attack on F_2 and F_8 presented the most significant challenge, as all classifiers failed to detect it. In summary, no single classifier emerged as the clear winner, underscoring the challenge classifiers face when tested against samples they have not been trained on. This highlights the limitations of classifiers trained under supervised learning, particularly when faced with previously unseen samples.

To assess the impact of training the classifiers on preprocessed and reduced features, as discussed in Section IV-A, we conducted a similar test using the reduced datasets with $\gamma = 200$ (selected empirically). The TPR results under this setting are presented in Table IV. We observed notable improvements in the scores reported by the SVM, which achieved 100% accuracy in detecting the A_1 attack across all firmware datasets. The DTC followed closely with a 95% accuracy. Moreover, all classifiers successfully detected all three attacks performed against the F_4 firmware dataset. However, similar to the results on raw data, all classifiers failed to detect the A_3 attack against F_2, F_7, and F_8 using the preprocessed data.

TABLE IV
TRUE POSITIVE RATES FOR EACH ML CLASSIFIER TRAINED ON PREPROCESSED RAM TRACES FROM EACH FIRMWARE.

		F_1	F_2	F_3	F_4	F_5	F_6	F_7	F_8	
	Test DS ↓	AES128	Interrupt	LED	Random	Shake	Temperature	Vibration	XTS	MEAN
RFC	A_1	100.0	0.1	100.0	100.0	100.0	100.0	50.2	0.0	68.8
	A_2	0.0	100.0	30.4	100.0	0.0	0.0	3.0	100.0	41.7
	A_3	100.0	0.0	0.0	100.0	100.0	99.8	0.0	0.0	50.0
SVM	A_1	100.0	100.0	100.0	100.0	100.0	100.0	100.0	100.0	100.0
	A_2	0.0	100.0	100.0	100.0	0.0	0.0	100.0	100.0	62.5
	A_3	100.0	0.0	99.4	100.0	100.0	100.0	0.0	0.0	62.4
KNN	A_1	100.0	76.8	100.0	100.0	100.0	100.0	100.0	76.8	94.2
	A_2	0.0	100.0	0.0	100.0	0.0	0.0	100.0	100.0	50.0
	A_3	100.0	0.0	99.4	100.0	100.0	100.0	0.0	0.0	62.4
DTC	A_1	99.8	99.9	100.0	100.0	100.0	100.0	60.6	100.0	95.0
	A_2	0.0	0.0	98.8	98.0	0.0	0.0	58.8	0.0	32.0
	A_3	100.0	0.0	99.4	100.0	100.0	98.8	0.0	0.2	62.3
GBC	A_1	100.0	100.0	100.0	100.0	100.0	100.0	98.8	23.3	90.3
	A_2	0.0	100.0	99.8	100.0	0.0	0.0	13.6	100.0	51.7
	A_3	100.0	0.0	99.4	100.0	100.0	98.6	0.0	0.0	62.3

Similarly, for the A_2 attack against F_1, F_5, and F_6 datasets, all classifiers failed to detect it. These results from Table IV do not imply that the preprocessing of the RAM traces failed to enhance the results. Instead, they underscore the inherent limitations of supervised learning when the models are not trained on all possible sample classes they might encounter during operation. Furthermore, in the subsequent section, we demonstrate that by utilizing preprocessed genuine samples, our C-VAE-based detectors achieved exceptional (100%) detection accuracy across all firmware types and their respective attack datasets.

B. Analysis using the Proposed C-VAE based detectors

In the preceding section, we highlighted the primary challenge encountered by supervised learning-based ML classifiers, which exhibit markedly reduced performance when confronted with test samples possessing distributions significantly different from the training data. This limitation underscores the advantage of representation learning-based methods. Hence, in this section, we employ the Conditional Variational Autoencoder (C-VAE) framework to train a model exclusively using genuine training features. Our objective is to train a model that extracts all distinctive information from the training set and subsequently utilizes this knowledge to classify test samples, regardless of their class distribution. This approach holds promise for enhancing the model's detection performance when confronted with novel and previously unseen data samples, a critical requirement in our firmware attestation framework where attack scenarios may vary widely.

In our attestation framework, model training is conducted at the verifier system in the training mode of operation. As discussed in Section IV-A1, the gateway node aggregates a number of RAM traces from the IoT devices, preprocesses them, and shares the processed feature dataset with the verifier. Subsequently, the verifier assumes responsibility for training the model and, in testing mode, classifies a given feature sample as genuine or not. In the following sections, we provide detailed model implementation details and analyze the performance of the proposed detectors under various hyper-parameter configurations. It is worth noting that under a fixed model architecture, the resulting trained model demonstrates robustness across a range of hyper-parameter selections, exhibiting consistent and stable performance. For results reproducibility, we have made our dataset[1] and code available at GitHub[2].

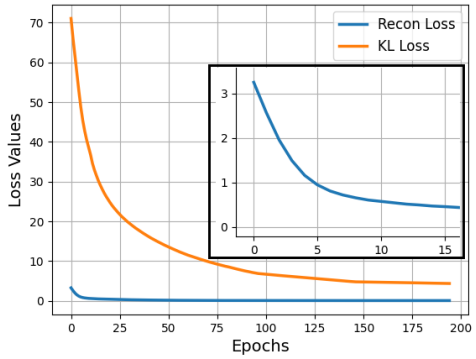

Fig. 7. Convergence graph for the trained model.

[1] https://dx.doi.org/10.21227/0nze-r023
[2] https://github.com/AsifIqbal8739/Firmware-Attestation-for-IoT.

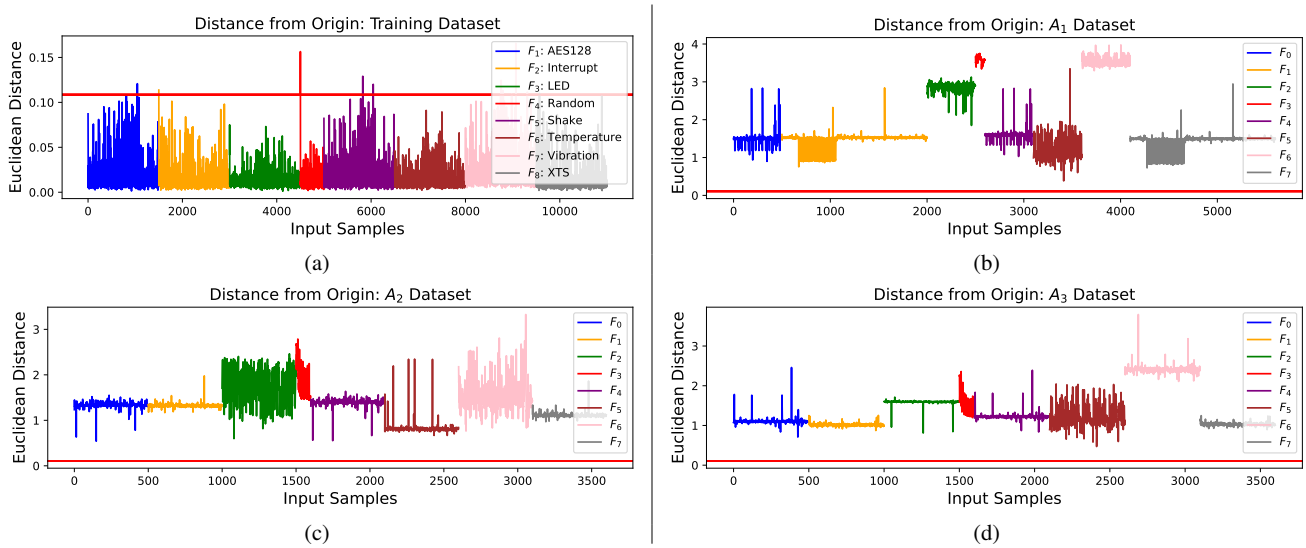

Fig. 8. Test statistic ζ_{LV} based on latent representation of all training and attack samples under 0.1% Input FPR and $\alpha = 5$.

1) Model Implementation Details: The proposed C-VAE model, illustrated in Fig. 4, consists of two ANNs, each composed of three layers. First input to the encoder model $\mathcal{E}_\phi$ is a vector $\mathbf{x} \in \mathbb{R}^k$, where k is 2048 for raw RAM trace features, and $k = \gamma - 1$ for the processed features. The device ID associated with $\mathbf{x}$ is used as the second input for $\mathcal{E}_\phi$. This input is encoded as a one-hot vector to generate $\mathbf{c} \in \mathbb{R}^8$, aligning with the utilization of eight distinct firmware classes employed for training the model. The encoder model is structured as an ANN with a node configuration of $100 - 50 - 2\alpha$, where α denotes the dimension of the learned latent distribution. As discussed in Section IV-B, the encoder models a latent distribution characterized by μ_z and Σ_z vectors, which are provided at its output. Utilizing (4), these outputs are used to generate a latent variable $\mathbf{z} \in \mathbb{R}^\alpha$. Subsequently, the variable $\mathbf{z}$ and device ID $\mathbf{c}$ are fed into the decoder model $\mathcal{D}_\theta$, which is likewise an ANN with a node configuration of $50 - 100 - k$, where k represents the size of the input feature $\mathbf{x}$. The Rectified Linear Unit (`ReLU`) activation function is applied at all inner nodes, with linear (no) activation utilized at the output layer for both encoder and decoder networks.

Before the training is initialized, the full matrix containing the concatenation of training features from all firmware classes is scaled such that each feature is mapped to a range of $[0, 1]$ using the MinMaxScaler from the Scikit-Learn library. This ensures that all features are kept within a similar scale, thereby facilitating stable learning. The parameters of this scaler will subsequently be employed during test mode to scale incoming test samples as well, thereby avoiding any information leakage between training and testing samples.

The model underwent exclusive training on genuine RAM trace features sourced from all eight firmware classes provided by the gateway node. A batch size of 512 was employed to train the model across 200 epochs. Following gradient computation, the Adam optimizer was utilized to update the parameters, with a learning rate that gradually decreased after the initial 100 epochs, at intervals of every 50 epochs, adopting the sequence $[1e^{-4}, 5e^{-5}, 1e^{-5}]$. Uniform-Glorot initialization was employed to initialize the model weights, ensuring stable training [41]. To visualize the convergence behavior of the proposed model, we present the reconstruction error and KLD (refer to (6)) in Fig. 7 after the initial 5 iterations, demonstrating stable convergence. Additionally, as an example, the learned latent distribution for all training samples, with $\alpha = 2$, is shown in Fig. 5, highlighting the encoder model's capability to symmetrically map all training samples in close proximity to the origin of the latent space.

While our model was trained using data from eight distinct firmware variants, it allows integration of data from new devices without necessitating a complete retraining process. In order to mitigate the catastrophic forgetting phenomenon commonly observed in ANNs, we can use continual learning strategies to facilitate this integration by enabling the model to progressively accumulate knowledge from newly acquired data classes, thus avoiding the need for training from scratch [42].

2) Detector Performance: With the trained model in hand, the verifier can attest any device by requesting the gateway node to acquire and process a RAM trace sample from the target device. Let's denote this received test sample as $\bar{\mathbf{x}}$ and its encoded device ID vector as $\bar{\mathbf{c}}$. Initially, the verifier employs the scaler fitted to the training data to scale this test sample. To prevent individual features from assuming arbitrarily large values due to scaling, their values are clamped to the range of ± 2.0. Subsequently, utilizing the test sample and its encoded label, we calculate the two test statistics, ζ_{LV} and ζ_{RE}, using the detectors based on latent distribution (D_{LV}) and reconstruction error (D_{RE}), as discussed in Section IV-C.

To facilitate classification into genuine and tampered classes, each detector requires a threshold ρ. We determine these thresholds for both detectors by computing their respective ζ_{LV} and ζ_{RE} over all the training data. Specifically, for an input False Positive Rate (FPR) of $1e^{-3}$, we compute ρ_{LV} and ρ_{RE}. These thresholds are then used by each detector to perform classification. For visual inspection, the test statistics on training data and the corresponding thresholds for a latent

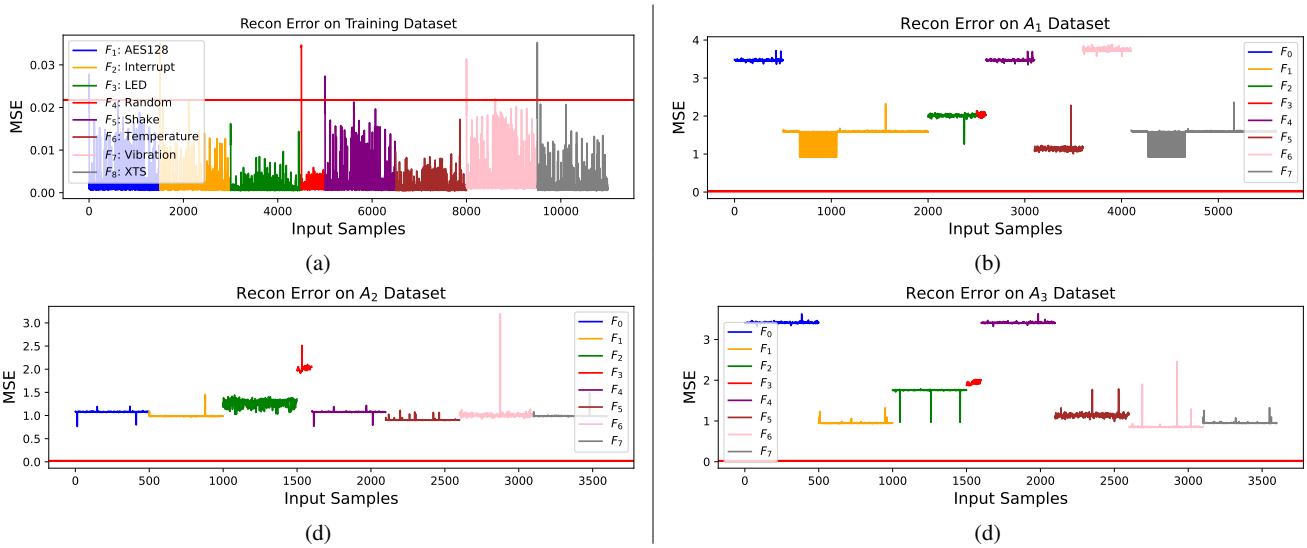

Fig. 9. Test statistic ζ_{RE} based on reconstruction error of all training and attack samples under 0.1% Input FPR and $\alpha = 5$.

TABLE V
THE ACCURACY OF THE C-VAE MODEL TRAINED AND TESTED ON THE PROCESSED RAM TRACE DATASETS UNDER 0.1% INPUT FPR AND $\alpha = 5$.

		F_1	F_2	F_3	F_4	F_5	F_6	F_7	F_8	
	Test DS ↓	AES128	Interrupt	LED	Random	Shake	Temperature	Vibration	XTS	MEAN
D_{RE}	A_1	100.0	100.0	100.0	100.0	100.0	100.0	100.0	100.0	100.0
	A_2	100.0	100.0	100.0	100.0	100.0	100.0	100.0	100.0	100.0
	A_3	100.0	100.0	100.0	100.0	100.0	100.0	100.0	100.0	100.0
D_{LV}	A_1	100.0	100.0	100.0	100.0	100.0	100.0	100.0	100.0	100.0
	A_2	100.0	100.0	100.0	100.0	100.0	100.0	100.0	100.0	100.0
	A_3	100.0	100.0	100.0	100.0	100.0	100.0	100.0	100.0	100.0

distribution size $\alpha = 5$ are shown in Fig. 8 (a) and Fig. 9 (a) for D_{LV} and D_{RE}, respectively.

The detection accuracy across all firmware attack classes for an input FPR of $1e^{-3}$, evaluated using both detectors, is summarized in Table V. These results are averaged over 10 trials with the model's latent size set to $\alpha = 5$ with $\gamma = 200$ components kept. From the table, we observe that both detectors achieved 100% accuracy in detecting all attack types across all firmware samples, underscoring the effectiveness of the overall detection framework. To provide visual insight, we present the ζ_{LV} test statistic generated by D_{LV} for all attack datasets in Fig. 8 which represents the distance from the origin of the latent space. As evident from the values on the training dataset in Fig. 8 (a), the model successfully mapped all eight firmware class samples close to the origin, with minimal deviations. However, when the attack samples pass through the encoder network, they are mapped at a significant distance away from the origin, facilitating their classification as tampered samples. This trend persists across all attacks on all firmware datasets, as depicted in Figs. 8 (a), (b), and (c).

To further illustrate this phenomenon, we present the γ_{LV} for all attack samples with $\alpha = 2$ in Fig. 10, showcasing the divergence of attack samples from the origin. Although some attack samples may be mapped closer to the origin, particularly with $\alpha = 2$ (used here for visualization purposes), however, in a high-dimensional latent space (e.g., $\alpha = 5$), the encoder network possesses adequate capacity to ensure that all attack samples are mapped away from the origin, as highlighted in Fig. 8.

Similarly, the discussion extends to the ζ_{RE}, test statistic computed by the D_{RE} detector. Looking at Fig. 9 (a), we observe the C-VAE model's ability to reconstruct input samples with high precision. While the decoder network effectively maps samples closer to the latent space origin back to their original shape, it struggles to accurately reconstruct attack samples, resulting in large reconstruction errors. Consequently, all attack samples are confidently classified as tampered. Moreover, due to the substantial disparity between the genuine and attack test statistics, both detectors maintain 100% detection accuracy even for input FPRs as low as $1e^{-6}$.

Although the proposed detectors achieved 100% accuracy in detecting tampered samples when trained using genuine preprocessed RAM traces, we conducted another experiment using raw genuine RAM traces for training the C-VAE model. Keeping the parameters consistent, we tested two network node architectures: one identical to the previous section and another with an encoder node architecture of $200 - 100 - 50 - 10$ ($\alpha = 5$) and a decoder architecture of $50 - 100 - 200 - k$, where $k = 2048$ for raw RAM traces. Out

TABLE VI
THE ACCURACY OF THE C-VAE MODEL TRAINED AND TESTED ON THE RAW RAM TRACE DATASETS.

		F_1	F_2	F_3	F_4	F_5	F_6	F_7	F_8	
	Test DS ↓	AES128	Interrupt	LED	Random	Shake	Temperature	Vibration	XTS	MEAN
D_{RE}	A_1	100.0	0.4	7.8	100.0	100.0	9.3	99.9	0.4	52.2
	A_2	2.2	2.8	99.0	100.0	2.0	0.3	22.5	2.9	29.0
	A_3	100.0	0.0	1.4	100.0	100.0	9.7	1.0	0.0	39.0
D_{LV}	A_1	100.0	40.0	77.1	100.0	100.0	77.5	88.3	40.5	77.9
	A_2	46.7	70.1	86.8	100.0	52.5	0.3	80.5	75.4	64.0
	A_3	100.0	0.5	49.1	100.0	100.0	76.0	1.8	0.5	53.5

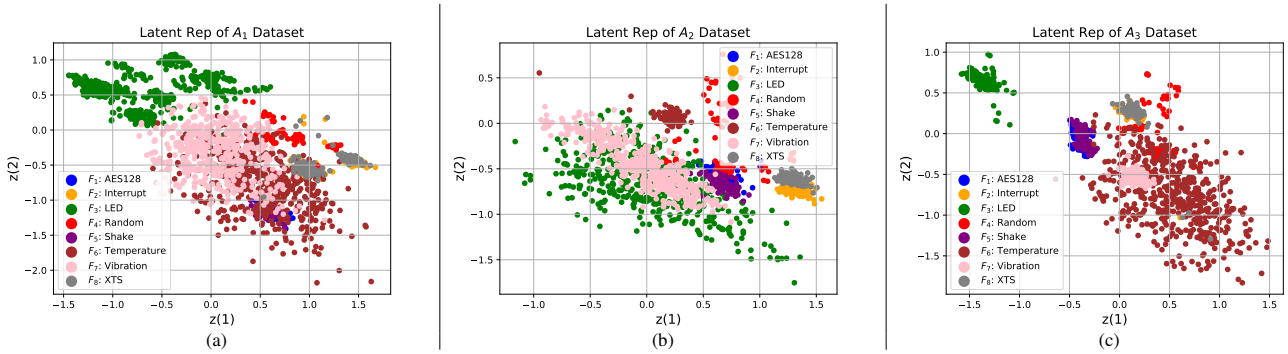

Fig. 10. Latent representation for all attack firmware samples with $\alpha = 2$.

of the two architectures, the one used in the previous section demonstrated superior performance, which is shown in Table VI.

From the results, we observe significant performance degradation for both detectors. They only achieved 100% accuracy in detecting attacks on firmware F_4, along with A_1 and A_3 attacks on F_1 and F_5 firmware. However, D_{RE} detector completely failed to detect any attacks against F_2, F_6, and F_8, while D_{LV} detected only 37%, 51%, and 57% of the respective attacks. Overall, the performance of the D_{LV} detector was relatively better. This outcome can be attributed to the bias of the C-VAE model towards reconstructing the training data accurately. Failure to remove the most dominant component (analogous to mean) from both the training and testing data leads the model to overlook minute details during training, resulting in reduced class separation in the test statistics when tested on the attack samples.

3) Hyper parameter Analysis: To analyze the impact of the latent distribution size α on the detection performance of our attestation framework, we conducted the experiment outlined in the previous section, varying $\alpha \in [1, 2, 5, 10]$. The average detection scores across all firmware and over 10 trials are summarized in Table VII which shows that the reconstruction error-based detector D_{RE} consistently maintained a 100% detection score across all values of α. However, the detection accuracy of the latent variable-based detector D_{LV} reached 100% at $\alpha \geq 5$. For $\alpha = [1, 2]$, the model encountered difficulty in mapping the A_1 attack samples away from the latent space origin. This difficulty is evident in Fig. 10 (a), where several samples from A_1 were mapped very close to the origin.

TABLE VII
TPR SCORES FOR BOTH DETECTORS OVER A RANGE OF LATENT DISTRIBUTION SIZES α.

	α	1	2	5	10
D_{RE}	A_1	100	100	100	100
	A_2	100	100	100	100
	A_3	100	100	100	100
D_{LV}	A_1	77.18	87.35	100	100
	A_2	83.7	95.52	100	100
	A_3	98.93	99.93	100	100

To examine the impact of γ, the number of singular components retained during the preprocessing pipeline conducted by the gateway node, we replicated the experiment detailed in Section VI-B2 across various γ values. The average detection scores over all firmware and 10 trials are presented in Table VIII. From the table, it's evident that the accuracy reported by both detectors improves with increasing γ, reaching a detection rate of over 50% across all three attack types when $\gamma \geq 125$. Furthermore, for $\gamma \geq 175$, both detectors achieve 100% accuracy, indicating that retaining all relevant information necessary for effective classification does not require keeping all components, thus achieving feature reduction. Additionally, as shown in Fig. 3, approximately 99.5% of the data variation is already captured by the first 175 components for all firmware. We would like to emphasise that both detectors consistently maintain a detection rate of 100% even when γ is set to its maximum value. Thus, while feature reduction is

TABLE VIII
TPR SCORES FOR BOTH DETECTORS OVER A RANGE OF γ COMPONENTS KEPT.

	γ	10	20	30	50	75	100	125	150	175	200
D_{RE}	A_1	0.1	0.1	0.4	13.6	29.7	57.1	90.9	100	100	100
	A_2	0.3	0.3	0.5	10.8	25.7	37.5	62.3	62.9	100	100
	A_3	0.3	0.3	0.5	0.7	10.2	49.7	57.5	65.3	100	100
D_{LV}	A_1	2.3	14.1	11.4	13	10.3	52	79.6	100	100	100
	A_2	1.8	7.4	5.6	0.7	6.1	28.9	51	53.7	100	100
	A_3	1.8	7.7	11	0.9	2.4	43	50.8	74.6	100	100

not mandatory for the model, its computational benefits make it a worthwhile step to undertake.

C. Time Complexity of the Framework

The proposed framework assigns specific responsibilities for each of the three system entities. The prover's role is to transmit a single RAM trace dump to the gateway node upon request. The gateway node, in turn, is tasked with sample preprocessing based on the operational mode and subsequently relaying the processed samples to the verifier. Throughout this process, the prover is solely responsible for transmitting its RAM trace and is not burdened with additional tasks. Consequently, the availability of the device under test remains relatively unaffected. This holds true even during the framework's training mode, wherein the IoT device can continue its operations while the gateway node acquires data. However, in the test mode, the device may need to pause its operations until authentication is complete. Therefore, it's imperative that the overall latency associated with the attestation process remains sufficiently low to prevent any disruption to device availability.

The attestation latency of our proposed detector framework is primarily governed by the performance of the gateway and verifier nodes, since the device under test only needs to transmit the RAM snapshot upon request. To provide a comprehensive comparison, we report the attestation latency of our framework alongside existing techniques, ensuring a probability of miss less than 5%, as shown in Table IX. Our detectors, especially D_{LV}, demonstrate the lowest latency among all the compared methods, highlighting the efficiency of our approach.

Let n represent the size of the IoT device's RAM. Then, the computational complexity of forwarding the RAM trace by the prover is $\mathcal{O}(1)$. During the training mode, the gateway node acquires m RAM traces from a single device, creating a matrix $\mathbf{X} \in \mathbb{R}^{m \times n}$ and computes its SVD. The computational complexity of this operation is $\mathcal{O}(mn \times \min(m, n))$, which simplifies to $\mathcal{O}(m^2 n)$ as, according to Table II, the maximum value of $m = 1500$. The projection computation in (3) has a complexity of $\mathcal{O}(mn(\gamma - 1))$, making the overall complexity at the gateway node during the training mode $\mathcal{O}(m^2 n)$. In the test mode, the gateway only performs a single projection (for a single RAM trace) using the appropriate projection matrix for the firmware in question, with a complexity of $\mathcal{O}(n\gamma)$.

At the verifier's end, in testing mode, we have access to the two detectors: D_{LV} requires only passing the input through the encoder network, making it less computationally taxing than D_{RE}, which requires passing through both the encoder and decoder networks. Additionally, the inference time through the detector to generate the test statistic can be reduced by utilizing a GPU. Thus, if the reduction of overall attestation latency is critical, infrastructure changes can be implemented to achieve this. Furthermore, our framework is easily scalable to a large number of IoT devices with a powerful enough gateway node, as the computational complexity of processing a RAM trace at test time is a low $\mathcal{O}(n\gamma)$ and the communication overhead over a single link is only $\mathcal{O}(n)$ bytes.

TABLE IX
ATTESTATION LATENCY FOR DIFFERENT FRAMEWORKS

Method	Latency (sec)
[11]	81.02
[26]	52.97
[14]	0.846
[25]	0.122
[31]	0.098
[8]	0.0042
D_{RE}	0.0032
D_{LV}	0.0013

VII. CONCLUSION

In conclusion, this paper introduced a novel software-based IoT device attestation framework leveraging RAM traces to authenticate the internal state of IoT devices. The framework operates in two modes: training and testing. During training, the gateway node aggregates RAM traces, preprocesses them to eliminate redundancy, and transmits them to the verifier node for training a C-VAE model. In testing mode, the verifier employs two detectors derived from the trained model to verify the authenticity of incoming RAM traces. Through extensive experimentation with eight distinct IoT applications, each subjected to three distinct attack types, our proposed detectors demonstrated robustness by successfully identifying all attacks on the IoT firmware. Notably, our framework's design ensures minimal impact on the IoT device availability, as computational tasks are primarily handled by the gateway and verifier nodes. During testing, the gateway node performs efficient computations on the input RAM trace, followed by inference through the detector models at the verifier. This streamlined process results in low authentication latency, making the framework suitable for real-time deployment in IoT ecosystems.

A potential avenue for future research stemming from our current work involves exploring the scenario of a sophisticated adversary capable of gaining full control over the IoT device under test. In such a scenario, the adversary could continuously transmit a pre-saved copy of genuine RAM trace in response to authentication requests. To mitigate such active attacks, incorporating encryption or hashing algorithms into the proposed authentication protocol presents an intriguing research direction. This approach could enhance the security of the authentication process, safeguarding against adversarial

manipulation and ensuring the integrity of the authentication mechanism.

REFERENCES

[1] G. Association *et al.*, "The mobile economy 2020," *GSM Association*, 2020.

[2] B. Mbarek, M. Ge, and T. Pitner, "An efficient mutual authentication scheme for internet of things," *Internet of things*, vol. 9, p. 100160, 2020.

[3] T. N. Alrumaih, M. J. Alenazi, N. A. AlSowaygh, A. A. Humayed, and I. A. Alablani, "Cyber resilience in industrial networks: A state of the art, challenges, and future directions," *Journal of King Saud University-Computer and Information Sciences*, p. 101781, 2023.

[4] J. Lee, L. Kim, and T. Kwon, "Flexicast: Energy-efficient software integrity checks to build secure industrial wireless active sensor networks," *IEEE Transactions on Industrial Informatics*, vol. 12, no. 1, pp. 6–14, 2015.

[5] V. Hassija, V. Chamola, V. Saxena, D. Jain, P. Goyal, and B. Sikdar, "A survey on iot security: application areas, security threats, and solution architectures," *IEEE Access*, vol. 7, pp. 82721–82743, 2019.

[6] Y. Shi, W. Wei, F. Zhang, X. Luo, Z. He, and H. Fan, "Sdsrs: A novel white-box cryptography scheme for securing embedded devices in iiot," *IEEE Transactions on Industrial Informatics*, vol. 16, no. 3, pp. 1602–1616, 2019.

[7] T. C. Group, "TPM main specification level 2 version 1.2."

[8] M. N. Aman, H. Basheer, J. W. Wong, J. Xu, H. W. Lim, and B. Sikdar, "Machine-learning-based attestation for the internet of things using memory traces," *IEEE Internet of Things Journal*, vol. 9, no. 20, pp. 20431–20443, 2022.

[9] K. Istiaque Ahmed, M. Tahir, M. Hadi Habaebi, S. Lun Lau, and A. Ahad, "Machine learning for authentication and authorization in iot: Taxonomy, challenges and future research direction," *Sensors*, vol. 21, no. 15, p. 5122, 2021.

[10] B. Zhang, D. Xiong, J. Su, H. Duan, and M. Zhang, "Variational neural machine translation," 2016.

[11] A. Seshadri, A. Perrig, L. Van Doorn, and P. Khosla, "Swatt: Software-based attestation for embedded devices," in *IEEE Symposium on Security and Privacy, 2004. Proceedings. 2004*, pp. 272–282, IEEE, 2004.

[12] A. Seshadri, M. Luk, A. Perrig, L. Van Doorn, and P. Khosla, "Scuba: Secure code update by attestation in sensor networks," in *Proceedings of the 5th ACM workshop on Wireless security*, pp. 85–94, 2006.

[13] A. Seshadri, M. Luk, and A. Perrig, "Sake: Software attestation for key establishment in sensor networks," in *Distributed Computing in Sensor Systems: 4th IEEE International Conference, DCOSS 2008 Santorini Island, Greece, June 11-14, 2008 Proceedings 4*, pp. 372–385, Springer, 2008.

[14] B. Chen, X. Dong, G. Bai, S. Jauhar, and Y. Cheng, "Secure and efficient software-based attestation for industrial control devices with arm processors," in *Proceedings of the 33rd Annual Computer Security Applications Conference*, pp. 425–436, 2017.

[15] J. Cao, T. Zhu, R. Ma, Z. Guo, Y. Zhang, and H. Li, "A software-based remote attestation scheme for internet of things devices," *IEEE Transactions on Dependable and Secure Computing*, vol. 20, no. 2, pp. 1422–1434, 2023.

[16] S. Surminski, C. Niesler, S. Linsner, L. Davi, and C. Reuter, "Scatt-man: Side-channel-based remote attestation for embedded devices that users understand," in *Proceedings of the Thirteenth ACM Conference on Data and Application Security and Privacy*, pp. 225–236, 2023.

[17] C. Krauß, F. Stumpf, and C. Eckert, "Detecting node compromise in hybrid wireless sensor networks using attestation techniques," in *Security and Privacy in Ad-hoc and Sensor Networks: 4th European Workshop, ESAS 2007, Cambridge, UK, July 2-3, 2007. Proceedings 4*, pp. 203–217, Springer, 2007.

[18] S. Agrawal, M. L. Das, A. Mathuria, and S. Srivastava, "Program integrity verification for detecting node capture attack in wireless sensor network," in *Information Systems Security: 11th International Conference, ICISS 2015, Kolkata, India, December 16-20, 2015. Proceedings 11*, pp. 419–440, Springer, 2015.

[19] H. Tan, W. Hu, and S. Jha, "A tpm-enabled remote attestation protocol (trap) in wireless sensor networks," in *Proceedings of the 6th ACM workshop on Performance monitoring and measurement of heterogeneous wireless and wired networks*, pp. 9–16, 2011.

[20] W. Yan, A. Fu, Y. Mu, X. Zhe, S. Yu, and B. Kuang, "Eapa: Efficient attestation resilient to physical attacks for iot devices," in *Proceedings of the 2nd International ACM Workshop on Security and Privacy for the Internet-of-Things*, pp. 2–7, 2019.

[21] T. Van Strydonck, J. Noorman, J. Jackson, L. A. Dias, R. Vanderstraeten, D. Oswald, F. Piessens, and D. Devriese, "Cheri-tree: Flexible enclaves on capability machines," in *2023 IEEE 8th European Symposium on Security and Privacy (EuroS&P)*, pp. 1143–1159, IEEE, 2023.

[22] M. A. Khan, M. N. Aman, and B. Sikdar, "Soteria: A quantum-based device attestation technique for the internet of things," *IEEE Internet of Things Journal*, pp. 1–1, 2023.

[23] K. Eldefrawy, G. Tsudik, A. Francillon, and D. Perito, "Smart: secure and minimal architecture for (establishing dynamic) root of trust.," in *NDSS*, vol. 12, pp. 1–15, 2012.

[24] P. Koeberl, S. Schulz, A.-R. Sadeghi, and V. Varadharajan, "Trustlite: A security architecture for tiny embedded devices," in *Proceedings of the Ninth European Conference on Computer Systems*, pp. 1–14, 2014.

[25] F. Brasser, B. El Mahjoub, A.-R. Sadeghi, C. Wachsmann, and P. Koeberl, "Tytan: Tiny trust anchor for tiny devices," in *Proceedings of the 52nd annual design automation conference*, pp. 1–6, 2015.

[26] X. Carpent, N. Rattanavipanon, and G. Tsudik, "Remote attestation of iot devices via smarm: Shuffled measurements against roving malware," in *2018 IEEE international symposium on hardware oriented security and trust (HOST)*, pp. 9–16, IEEE, 2018.

[27] M. N. Aman and B. Sikdar, "Att-auth: A hybrid protocol for industrial iot attestation with authentication," *IEEE Internet of Things Journal*, vol. 5, no. 6, pp. 5119–5131, 2018.

[28] M. N. Aman, M. H. Basheer, S. Dash, A. Sancheti, J. W. Wong, J. Xu, H. W. Lim, and B. Sikdar, "Prom: Passive remote attestation against roving malware in multicore iot devices," *IEEE Systems Journal*, vol. 16, no. 1, pp. 789–800, 2021.

[29] M. Zhang, Y. Zhang, S. Li, and Q. Wan, "Software trusted startup and update protection scheme of iot devices," in *2023 IEEE 9th Intl Conference on Big Data Security on Cloud (BigDataSecurity), IEEE Intl Conference on High Performance and Smart Computing, (HPSC) and IEEE Intl Conference on Intelligent Data and Security (IDS)*, pp. 147–152, 2023.

[30] I. Makhdoom, M. Abolhasan, J. Lipman, D. Franklin, and M. Piccardi, "I2map: Iot device attestation using integrity map," in *2023 IEEE 22nd International Conference on Trust, Security and Privacy in Computing and Communications (TrustCom)*, pp. 1900–1907, 2023.

[31] M. N. Aman, M. H. Basheer, S. Dash, J. W. Wong, J. Xu, H. W. Lim, and B. Sikdar, "Hatt: Hybrid remote attestation for the internet of things with high availability," *IEEE Internet of Things Journal*, vol. 7, no. 8, pp. 7220–7233, 2020.

[32] Y. Harbi, Z. Aliouat, S. Harous, A. Bentaleb, and A. Refoufi, "A review of security in internet of things," *Wireless Personal Communications*, vol. 108, pp. 325–344, 2019.

[33] A. Roy and S. Banerjee, *Linear algebra and matrix analysis for statistics*. Chapman and Hall/CRC, 2014.

[34] D. P. Kingma and M. Welling, "Auto-encoding variational bayes," 2013.

[35] I. Goodfellow, J. Pouget-Abadie, M. Mirza, B. Xu, D. Warde-Farley, S. Ozair, A. Courville, and Y. Bengio, "Generative adversarial nets," *Advances in neural information processing systems*, vol. 27, 2014.

[36] Y. Hua, J. Guo, and H. Zhao, "Deep belief networks and deep learning," in *Proceedings of 2015 International Conference on Intelligent Computing and Internet of Things*, pp. 1–4, IEEE, 2015.

[37] A. Zola, "Iot platforms overview: Arduino, raspberry pi, intel galileo and others," June 2017.

[38] V. H. Nguyen and L. M. S. Tran, "Predicting vulnerable software components with dependency graphs," in *Proceedings of the 6th International Workshop on Security Measurements and Metrics*, pp. 1–8, 2010.

[39] F. Pedregosa, G. Varoquaux, A. Gramfort, V. Michel, B. Thirion, O. Grisel, M. Blondel, P. Prettenhofer, R. Weiss, V. Dubourg, J. Vanderplas, A. Passos, D. Cournapeau, M. Brucher, M. Perrot, and E. Duchesnay, "Scikit-learn: Machine learning in Python," *Journal of Machine Learning Research*, vol. 12, pp. 2825–2830, 2011.

[40] A. Paszke, S. Gross, F. Massa, A. Lerer, J. Bradbury, G. Chanan, T. Killeen, Z. Lin, N. Gimelshein, L. Antiga, A. Desmaison, A. Kopf, E. Yang, Z. DeVito, M. Raison, A. Tejani, S. Chilamkurthy, B. Steiner, L. Fang, J. Bai, and S. Chintala, "Pytorch: An imperative style, high-performance deep learning library," in *Advances in Neural Information Processing Systems 32*, pp. 8024–8035, Curran Associates, Inc., 2019.

[41] X. Glorot and Y. Bengio, "Understanding the difficulty of training deep feedforward neural networks," in *Proceedings of the thirteenth international conference on artificial intelligence and statistics*, pp. 249–256, JMLR Workshop and Conference Proceedings, 2010.

[42] M. De Lange, R. Aljundi, M. Masana, S. Parisot, X. Jia, A. Leonardis, G. Slabaugh, and T. Tuytelaars, "A continual learning survey: Defying forgetting in classification tasks," *IEEE transactions on pattern analysis and machine intelligence*, vol. 44, no. 7, pp. 3366–3385, 2021.

Asif Iqbal received the BS degree in Telecommunication Engineering from NUCES-FAST, Peshawar, Pakistan, MS degree in Wireless Communications from LTH, Lunds University, Sweden, and Ph.D in Electrical & Electronics Engineering from The University of Melbourne, Melbourne, Australia in 2008, 2011, and 2019 respectively.
He is currently working as a Research Fellow with the Department of Electrical & Computer Engineering at the National University of Singapore, Singapore. Dr. Iqbal previously served on the faculty of National University of Computer and Emerging Sciences Pakistan as an Assistant Professor. His research interests include signal processing, deep learning, sparse signal representations, and privacy preserving machine learning.

Usman Zia is an undergraduate student of Computer Systems Engineering at the University of Engineering & Technology Peshawar, Pakistan. His research interests include security of embedded devices.

Muhammad Naveed Aman (S'12-M'17-SM'23) is an Assistant Professor in the University of Nebraska-Lincoln. He received the B.Sc. degree in Computer Systems Engineering from KPK UET, Peshawar, Pakistan, M.Sc. degree in Computer Engineering from the Center for Advanced Studies in Engineering, Islamabad, Pakistan, M.Engg. degree in Industrial and Management Engineering and Ph.D. in Electrical Engineering from the Rensselaer Polytechnic Institute, Troy, NY, USA in 2006, 2008, and 2012, respectively. His research interests include IoT and network security, hardware systems security and privacy, wireless and mobile networks and stochastic modelling.

Biplab Sikdar (S'98-M'02-SM'09) received the B.Tech. degree in electronics and communication engineering from North Eastern Hill University,Shillong, India, in 1996, the M.Tech. degree in electrical engineering from the Indian Institute of Technology, Kanpur, India, in 1998, and the Ph.D. degree in electrical engineering from the Rensselaer Polytechnic Institute, Troy, NY, USA, in 2001. He was on the faculty of Rensselaer Polytechnic Institute from 2001 to 2013, first as an Assistant and then as an Associate Professor. He is currently a Professor with the Department of Electrical and Computer Engineering, National University of Singapore, Singapore. His research interests include wireless network, and security for IoT and cyber physical systems.