IoT Device Authentication via RAM Trace Analysis: A Representation Learning Framework

Asif Iqbal[†], Muhammad Naveed Aman[‡], and Biplab Sikdar[†]
[†]Department of Electrical and Computer Engineering, National University of Singapore, Singapore 117583.
[‡]School of Computing, University of Nebraska-Lincoln, Nebraska, USA.
Email: {aiqbal, bsikdar}@nus.edu.sg, naveed.aman@unl.edu

***Abstract*—Recent advances in IoT, machine learning, and edge computing have driven transformative paradigms like smart cities, grids, healthcare, and transportation systems, providing efficient solutions. This has led to a pervasive proliferation of connected devices, ranging from high-power computers to low-power sensors. Yet, the complex IoT architecture poses numerous vulnerabilities, demanding robust security measures. Existing firmware attestation techniques often encounter obstacles due to proprietary constraints, necessitating access to the device's authentic firmware. To address this challenge, this paper proposes a novel software-based attestation framework that utilizes RAM traces from IoT devices for remote verification. By employing deep learning models trained in a representation learning paradigm, our framework empowers the remote verifier to authenticate the internal state of IoT devices. Leveraging data collected from real-world prototype devices, our approach achieves an impressive 100% detection rate for critical attacks on IoT devices with a false positive rate of 10^{-3}. Remarkably, our framework preserves device availability and maintains low authentication latency, highlighting 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 witnessing a remarkable surge across diverse domains of our daily lives. According to the GSM report released in 2020, the annual growth rate of IoT devices is projected at 13%, with an anticipated total of around 2.46 billion various IoT devices by 2025 [1]. Industries experiencing rapid adoption of IoT devices include healthcare, smart cities, industrial manufacturing and control, forest monitoring, traffic monitoring, defense, and others [2]. Many of these devices deployed in such applications are characterized by their affordability, resulting in constraints in computational power, memory, and power consumption. As these devices become increasingly integrated into critical infrastructure, they are drawing the attention of cyber-criminals [3]. Moreover, their distributed nature and complex architecture make them vulnerable to cyber attacks targeting their network communications, posing threats to device and data integrity [4]. Recent research indicates that over 90% of vulnerabilities detected in IoT device-enabled smart applications are associated with firmware [5].

This research is supported by A*STAR, CISCO Systems (USA) Pte. Ltd and National University of Singapore under its Cisco-NUS Accelerated Digital Economy Corporate Laboratory (Award I21001E0002).

Typically, during the addition of a new IoT device or at each power-up cycle, the device must register with a central entity, known as the *verifier*, through *attestation*. The verifier, acting as a secure entity, can initiate attestation even during regular operation. It engages with the device under test, called the *prover*, to verify its authenticity. The verifier challenges the prover to generate a hash digest based on its firmware source code, which it shares for verification. Then, the verifier computes the same checksum using a locally stored version of the prover's firmware to authenticate it. However, the iterative checksum computations, requiring multiple passes over the device firmware, lead to increased computational overhead and verification time. Moreover, this approach demands the verifier to possess a local copy of the prover's firmware, which may not always be practical and raises potential concerns about intellectual property (IP) infringement.

IoT devices are typically designed for particular tasks, with their computational resources finely tuned to fulfill those tasks' requirements. The introduction of additional tasks or security protocols on IoT devices may hinder their routine operations to some degree, commonly known as impacting their *availability*. Ideally, implementing a security protocol on a device should not hinder its availability, especially for devices tasked with real-time or safety-critical applications. Nevertheless, many attestation techniques require uninterrupted execution of their security protocols, which often requires considerable time to execute, leading to reduced availability of the device.

Device attestation methods in the literature can be classified into two main categories: software-based, hardware-based, and a hybrid approach. Software-based techniques involve executing an algorithm on the device under test during runtime and comparing it with the expected value stored at the verifier. Due to their limited computational capabilities, the runtime of these techniques may vary based on the device's current state or configuration, making them suitable for attestation purposes. Although these techniques are computationally intensive, they are well-suited for low-cost embedded devices as they do not require additional hardware for execution. The classical approaches like SWATT [6], SCUBA [7], and SAKE [8] and more recent [9], [10] belong to this category. These techniques require a genuine copy of the firmware stored in the flash memory, which is not ideal. Moreover, during attestation, the device availability is severely reduced as well.

In contrast, hardware-based techniques require additional

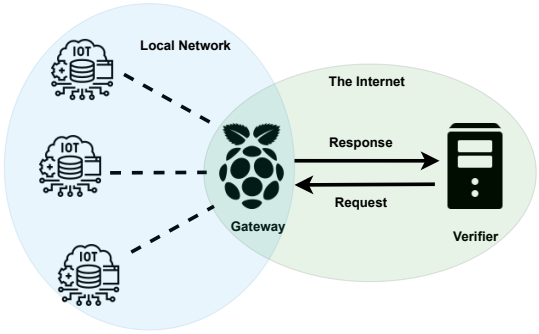

Fig. 1. The proposed system model.

components such as secure co-processors or Trusted Platform Modules (TPM) [11], resulting in lower computational demands, and do not impact the device availability. Techniques such as [12]–[14] belong to this category. However, due to the advanced hardware requirements, these methods are impractical for the majority of IoT devices. Hybrid techniques aim to strike a balance between the two approaches by minimizing computational complexity while requiring minimal additional hardware. Techniques proposed in [15]–[19] fall into this category. Nevertheless, due to the inherent complexity and stringent architectural requirements of these hybrid techniques, their applicability remains severely limited [20].

In recent years, machine learning (ML) techniques have emerged as viable options for IoT device attestation [20], [21]. The majority of ML methods adopted in this context operate under the supervised learning paradigm, where training a model requires samples from all anticipated classes. While supervised ML-based approaches have been effective in device attestation [20], [21], they are subject to a major limitation: the need for comprehensive acquisition and labeling of training datasets covering the diverse attack vectors encountered by devices during operation. This task proves exceptionally challenging due to the continual emergence of new attack vectors, rendering comprehensive coverage impractical. Consequently, when confronted with a test sample significantly dissimilar to all trained classes, the model's decision may lack reliability.

To address the challenges outlined above, this paper presents a software-based device attestation framework employing unsupervised Deep Learning (DL) techniques. In contrast to conventional methods, our approach imposes no additional computational burdens on the device under test, thus maintaining device availability. Moreover, instead of depending on the genuine firmware of the prover, our framework utilizes a Conditional Variational AutoEncoder (C-VAE) model [22] trained on features extracted from an IoT device's RAM. This strategy enables the detection of even subtle modifications to a device's firmware.

In summary, our primary contributions include:

1) Introducing a software-based attestation framework that utilizes preprocessed and condensed RAM trace features, eliminating the need for resource-intensive computations on the device under evaluation.
2) Developing a device tampering detector leveraging the C-VAE model trained within the representation learning paradigm.
3) Evaluating the proposed framework on real prototype devices across four distinct applications affected by three different types of attacks.

The remainder of the paper is structured as follows: Section II provides preliminary details about the underlying system model. Section III presents the proposed framework, detailing the feature generation procedure and the tampering detection model. Experimental evaluation is discussed in Section IV, utilizing data obtained from a real prototype system. Finally, Section V concludes the paper.

II. Preliminaries

A. System Model

The system model underlying our framework is depicted in Fig. 1, comprising three key entities:

1) **The IoT Device:** Multiple IoT devices are connected to a gateway node through wired or wireless interfaces. These devices may have varying capabilities in terms of battery life, memory, and processing power, and serve as the *provers* in attestation queries.
2) **The Gateway:** Typically, IoT devices lack a complete TCP/IP protocol stack due to resource constraints and rely on a relatively powerful *Gateway* node for routing and internet connectivity. In the proposed system model, the gateway node serves dual functions: (i) acting as a router and device manager for the connected IoT devices, enabling communication between IoT devices and the central entity (*verifier*), and (ii) processing the incoming binary RAM traces to extract training features (details in Section III). The gateway node uses the internet to communicate with the verifier system and can operate in either *training* or *testing* mode.
3) **The Verifier:** This entity represents a trusted remote server responsible for IoT device attestation during bootup or regular operation. It maintains records of all connected gateway nodes and the device IDs of subsequent IoT devices associated with them. Furthermore, it oversees the management of detection models for device attestation and operates in either *training* or *testing* mode.

B. Information Flow

The information flow, as seen in Fig. 1, is presented in this section. Commencing with the prover, upon receipt of an attestation request, it captures a snapshot of its current working memory and transmits it to the gateway node. This RAM trace consists of a 1D array of binary values saved in HEX format. By aggregating 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 to scale it into the range of $[0, 1.0]$. Depending on the operation mode, this trace undergoes one of two preprocessing pipelines at the gateway node to generate a feature vector (as detailed in Section III-A). Subsequently, the resulting feature vector is

transmitted to the verifier system. In training mode, the verifier utilizes the received feature set to commence model training. Conversely, in testing mode, the received feature is subjected to the detection algorithm and thresholding to classify the device as genuine or tampered. Further details on each of the aforementioned steps is provided in Section III.

C. The RAM Trace

The data stored in a device's RAM at any given moment during its operation is intricately tied to its ongoing functions. Typically, the RAM of an embedded device can be segmented into the following primary sections:

i. `.data`: It stores global and initialized static variables.
ii. `.bss`: Memory allocation for uninitialized static and global variables is managed within this section.
iii. `.text`: This section contains the executable code.
iv. `Heap/Stack`: Reserved for dynamic memory allocation, function return addresses, interrupts, and local variables storage.

The underlying assumption is that alterations to a device's firmware will impact its execution cycle, leading to changes in the device's RAM trace. Depending on the scale of firmware modifications, these changes can range from minor to significant and may appear in various segments of the RAM. For example, the introduction of new pointers into the control dependency graph of the executable code can affect all four memory areas of a device. Moreover, the detectability of alterations reflected in the main memory is influenced by the nature of tampering. While substantial modifications are generally easier to identify, subtle tampering may result in minimal or localized deviations in the RAM trace, posing greater challenges for detection.

III. The Proposed Framework

In this section, we introduce our proposed framework, comprising a pre-processing pipeline for generating highly informative training features and a detection framework based on the C-VAE model. Further details are provided in the subsequent subsections.

A. Feature Generation

To facilitate efficient detection modeling, it's crucial to generate RAM features that don't necessitate complex computations during the actual attestation process, thus minimizing their impact on device availability. Under typical operation, RAM changes are minimal, with only select locations being updated during application execution [20]. In the event of tampered firmware, the resulting RAM trace diverges from the norm. However, across both normal and tampered scenarios, the majority of RAM content remains unchanged. Effective model training requires preprocessing the RAM trace to eliminate such redundancy while preserving distinctive information. This ensures the model focuses on data variations rather than the baseline trends present in training samples.

Consider a dataset matrix containing m RAM traces with n elements each, denoted as $\mathbf{X} \in \mathbb{R}^{m \times n}$. The dataset matrix

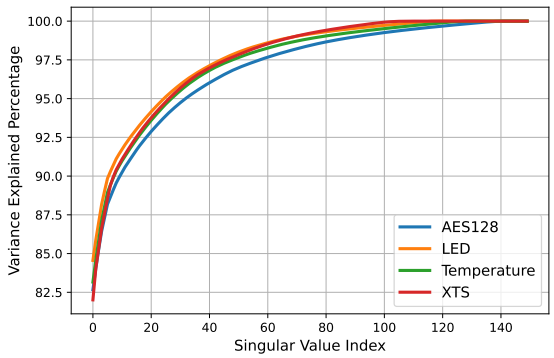

Fig. 2. Initial 150 singular values from different firmware datasets.

($m \neq n$ or $m = n$) with rank $r \leq \min(m, n)$ can be decomposed into three sub-matrices using the reduced singular value decomposition (r-SVD) [23] as

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

where $\mathbf{U} \in \mathbb{R}^{m \times r}$ and $\mathbf{V} \in \mathbb{R}^{n \times r}$ are left and right semi-unitary matrices, i.e., $\mathbf{U}^{\top}\mathbf{U} = \mathbf{V}^{\top}\mathbf{V} = \mathbf{I}_r$, containing singular vectors, and $\mathbf{S} \in \mathbb{R}^{r \times r}$ contains their unique respective singular values $\sigma_1 \geq \sigma_2 \geq \ldots \sigma_r > 0$ on its main diagonal, and r is the matrix rank. These unique σ values represent the amount of variance of $\mathbf{X}$ explained by the respective left and right singular vectors. This relationship can be seen through the outer product from of SVD, given by:

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

where $\mathbf{u}_i$ and $\mathbf{v}_i$ are the i^{th} left and right singular vectors, respectively. To provide insight, we present the first 150 singular values extracted from RAM traces collected from diverse applications (details in Section IV-A) in Fig. 2. It's observed that the leading singular value alone covers over 82.5% of the total variation within the RAM trace data. In ML, it's advisable to eliminate shared information from the dataset prior to training. Thus, we opt to discard the rank-1 outer product from the training dataset $\mathbf{X}$. Additionally, the figure indicates that approximately 99.5% of the total variation is captured by the initial $\beta = 150$ components, allowing for further dataset size reduction without significant loss of information. In essence, removing the first component eradicates redundant information, while discarding components beyond β mitigates noisy elements. Consequently, retaining components enables the model to focus on valuable distinctive information. As the individual RAM traces are kept as row vectors in $\mathbf{X}$, we use $\hat{\mathbf{V}} = \mathbf{V}_{2:\beta} \in \mathbb{R}^{n \times (\beta - 1)}$, containing $[2 : \beta]$ right singular vectors of $\mathbf{V}$, as the projector to reduce $\mathbf{X}$ as:

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

Following this transformation, the reduced matrix $\hat{\mathbf{X}}$ becomes suitable for effective training of ML models. Within the proposed framework, feature generation is handled by the

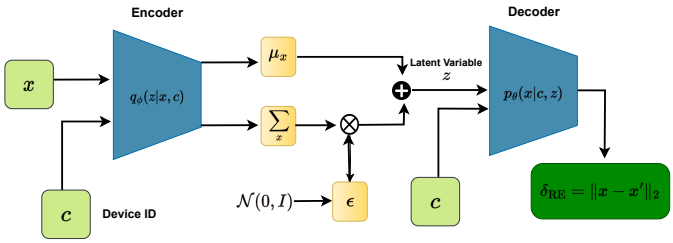

Fig. 3. The detector model architecture.

gateway node. However, the computations undertaken differ depending on whether the node is operating in training mode or testing mode (inference). The details are summarized below:

1) Training Mode: In the training mode, 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. With enough RAM traces at hand, the gateway computes the r-SVD of each dataset $\mathbf{X}_i$ separately. Subsequently, it utilizes the corresponding projection matrix $\hat{\mathbf{V}}_i$ to generate the feature matrix $\hat{\mathbf{X}}_i$, as described in (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 for training the model.

2) Testing Mode: In testing (verification) 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 stored projection matrix $\hat{\mathbf{V}}_i$ to project $\mathbf{x}_i$ onto the subspace spanned by the rows of $\hat{\mathbf{V}}_i$ to generate the feature vector $\hat{\mathbf{x}}_i = \hat{\mathbf{V}}_i^\top \mathbf{x}_i$. By doing so, we remove the redundant information and keep the variations along those components which were used to train the model.

B. Proposed Detection Model

As mentioned earlier, supervised learning mandates training data covering samples from all potential classes encountered during model operations. Models trained on such exhaustive datasets can identify test samples sharing similar features with their training counterparts. Nevertheless, these models frequently encounter challenges when presented with test samples dissimilar to any of the class samples they were trained on. For the IoT device attestation problem, our aim is to train a model capable of excelling not only on samples encountered during training but also adeptly identifying previously unseen samples. In the event of a novel attack, our model should have the ability to flag it as potentially malicious. In our attestation framework, the gateway node gathers RAM traces from IoT devices, preprocesses them, and sends the feature dataset to the verifier. The verifier trains a single model and, in testing mode, classifies feature samples as genuine or not.

This work harnesses the capabilities of a conditional-VAE (C-VAE), a DL model, for detecting firmware tampering in IoT devices. By training the model with data collected from authentic IoT devices, the model learns the inherent latent patterns and characteristics unique to these devices. Consequently, the C-VAE becomes adept at distinguishing between test samples resembling genuine patterns and those displaying anomalies, aiding in the identification of previously unseen test samples. Employing artificial neural networks (ANNs) as the base learners, our model architecture comprises an Encoder and a Decoder network, implemented through two ANNs interconnected and operating within the C-VAE setup [22], [24]. The model architecture is shown in Fig. 3.

The C-VAE model is trained by maximizing the variational lower bound [24], given by

$$\mathcal{L}_{VAE}(\boldsymbol{\theta},\boldsymbol{\phi};\mathbf{x},\mathbf{z},\mathbf{c}) = \mathbb{E}_{\mathbf{z}\sim q_{\boldsymbol{\phi}}(\mathbf{z}|\mathbf{x},\mathbf{c})}(log\ p_{\boldsymbol{\theta}}(\mathbf{x}|\mathbf{z},\mathbf{c})) - D_{KL}(q_{\boldsymbol{\phi}}(\mathbf{z}|\mathbf{x},\mathbf{c})||p_z(\mathbf{z})). \quad (4)$$

where $\mathbf{x}$ is an input sample, $\mathbf{c}$ is the one-hot-encoded device ID, $\mathbf{z}$ is the latent variable computed via the reparameterization trick [24] as $\mathbf{z} = \boldsymbol{\mu}_\mathbf{z} + \boldsymbol{\Sigma}_\mathbf{z} \odot \boldsymbol{\epsilon}$. Here $\boldsymbol{\epsilon} \sim \mathcal{N}(\mathbf{0},\mathbf{I})$ and $\odot$ denotes element-wise multiplication. The first component in (4) represents the log likelihood function and the second component constitutes the latent loss, which is computed using the Kullback-Leibler divergence (KLD) between the distribution $q_{\boldsymbol{\phi}}(\mathbf{z}|\mathbf{x},\mathbf{c})$ learned by the Encoder and some prior distribution $p_z(\mathbf{z})$, typically set to a standard Normal distribution, facilitating KLD computation without the need for estimation [24]. In essence, the first term encourages the decoder to reconstruct the input sample from the latent space as accurately as possible. Meanwhile, the latent loss compels the Encoder to learn a distribution that is symmetric around the origin, thereby ensuring a cohesive latent space. By incorporating auxiliary information such as the device ID $\mathbf{c}$, the encoder is encouraged to map samples from various devices closer to the origin of the latent space.

Utilizing a trained C-VAE model, our detection approach operates by computing a test statistic δ_{RE} for a given test sample $\bar{\mathbf{x}}$ and its corresponding reconstruction $\hat{\bar{\mathbf{x}}}$, defined as $\delta_{RE} = \|\bar{\mathbf{x}} - \hat{\bar{\mathbf{x}}}\|_2$. The underlying concept is straightforward: if the test sample is authentic, the model's reconstruction will be precise, resulting in a low δ_{RE}; conversely, for a test sample from a tampered device, the δ_{RE} will be high. Consequently, by employing a predefined threshold ζ, if $\delta_{RE} < \zeta$, the test sample is classified as authentic; otherwise, it is deemed tampered. The selection of a suitable threshold is pivotal to ensure robust detection of tampered samples. We determine the threshold ζ based on the predefined acceptable False Positive Rate (FPR) tolerance, typically established during the design phase of a detection model. Once the model is trained using the entire training dataset, this threshold can be computed by analyzing the δ_{RE} of all training samples and selecting the threshold leading to the specified FPR.

IV. Experimental Evaluation

A. Evaluation Datasets

The genuine and attack datasets utilized in this study were obtained from [20], where RAM traces for four distinct applications were generated using a real-world prototype setup. This setup employed the Arduino Uno, a commonly used IoT device, featuring an 8-bit microcontroller with 32 KB of flash

memory and 2 KB of RAM. For the gateway node, a Raspberry Pi 3 equipped with a 64-bit 1.2 GHz processor was utilized. As detailed in [20], two cryptography-based applications, denoted as F_1 (AES128) and F_4 (XTS), as well as two sensor-based applications, referred to as F_2 (LED) and F_3 (Temperature), were studied. These applications were chosen to assess the effectiveness of our proposed attestation method in real-world application scenarios. Subsequently, three distinct attacks are performed against each dataset F_i, which are:

- 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.

The attacks discussed above cover the most critical and fundamental attacks considering the three types of dependencies when it comes to good software engineering practices [20], [25]. In doing so, we cover majority of the tampering attacks carried on embedded devices.

The next step entails capturing RAM traces from the prototype IoT device while it runs one of the specified firmware variants. Initially, we upload firmware F_i onto the IoT device and let it operate for a set duration. During normal device operation, we collect numerous RAM trace samples using the gateway node. Each sample consists of 2048 bytes, with random intervals between consecutive samples. These samples are labeled as *genuine* and stored for further analysis. Subsequently, we upload each attack variant of a particular firmware to obtain their corresponding tampered RAM traces. This yields three distinct sets of tampered RAM traces (A_1, A_2, A_3) for each firmware variant F_i.

B. C-VAE Model Training

With genuine RAM trace datasets at hand, the subsequent model training and evaluations are conducted using Python v3.9 using the PyTorch deep learning library. The proposed C-VAE model, as shown in Fig. 3 contains two ANNs, each consisting of 3 layers. The input to the encoder model is a vector $\mathbf{x} \in \mathbb{R}^k$, where k is 2048 for raw RAM trace features, and $k = \beta - 1$ for the processed features. The device ID associated with $\mathbf{x}$ is encoded as a one-hot vector to create $\mathbf{c} \in \mathbb{R}^4$, reflecting the use of four different firmware classes to train the model. The encoder model is structured as an ANN with a node configuration of $100 - 50 - 4$ (a 2D latent space). Following generation of the latent variable $\mathbf{z}$ using the reparameterization trick, both $\mathbf{z}$ and device ID $\mathbf{c}$ are fed into the decoder model, which is also 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 no activation function at the output layer for both encoder and decoder networks. The model parameters are updated using the Adam optimizer with a decreasing learning rate of $[1e^{-4}, 5e^{-5}, 1e^{-5}]$, changed every 50 iterations. The batch size was set to 256, and the model was trained for 200 epochs.

TABLE I
THE DETECTION ACCURACY OF THE C-VAE MODEL TRAINED AND TESTED ON RAW AND PREPROCESSED RAM TRACES

	Test DS ↓	F_1	F_2	F_3	F_4
Raw	A_1	100	10.1	29	0.8
	A_2	3.5	100	0.5	3.2
	A_3	100	3.6	28.9	0.2
Processed	A_1	100	100	100	100
	A_2	100	100	100	100
	A_3	100	100	100	100

C. 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. To enable classification, we compute the test statistic δ_{RE} for all the training dataset and use the input FPR of $1e^{-3}$ to compute the threshold ζ_{RE}, both are shown in Fig. 4 (a). The detection accuracy of the models trained and tested on preprocessed data and raw RAM traces is shown in Table I. These results are averaged over 10 trials and the highest component kept was $\beta = 150$. Looking at the scores reported by the model trained on raw RAM samples, we see that it completely failed to detect tampering instances from most of the attack cases across all firmware. This failure can be attributed to the fact that without the removal of most dominant component (redundant information) from data leads the model to overlook smaller details during training, resulting in it failing to separate the genuine from attack samples effectively. The detection accuracy of the model trained on the proposed preprocessing pipeline, however, achieved 100% accuracy in detecting all attack types across all firmware applications, underscoring the efficacy of the preprocessing steps taken.

To visualize this result, we show the test statistics for all attack types across all firmware in Fig. 4. From Fig. 4 (a) we can see that the model was able to reconstruct the training samples with high precision. However, it struggled to accurately reconstruct the attack samples, as shown in Fig. 4 (b-d), leading to high test statistics. This underscores the effectiveness of projecting the test samples onto the subspace occupied by the training samples, assisting the model in distinguishing between genuine and attack samples. Moreover, owing to the distinction between the test statistics of authentic and attack samples, the detector maintains 100% accuracy even for input FPR as low as $1e^{-6}$.

D. Computational Workload Distribution

In the proposed framework, the primary computational burden is handled by the gateway and verifier nodes, both during

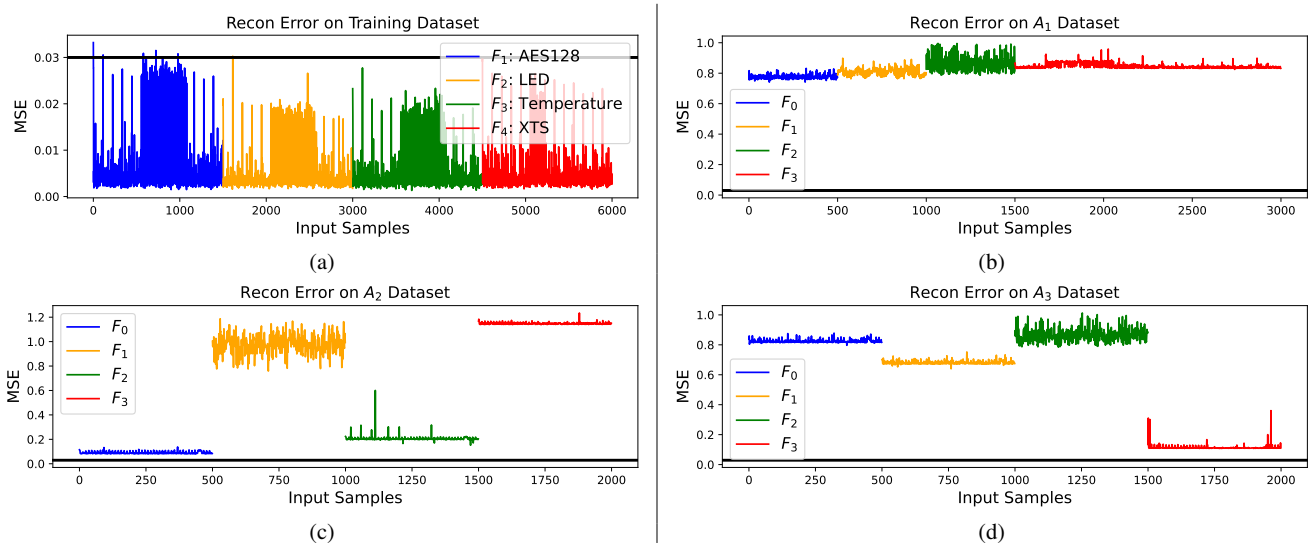

Fig. 4. Test statistic ζ_{RE} of all training and attack samples under 0.1% Input FPR, represented by the horizontal black line.

the training and testing (verification) modes. The IoT device's involvement is minimal, limited to providing a RAM snapshot. Thus, the IoT device remains unaffected during training and may only need to pause operations during verification until authentication is complete.

In training mode, the gateway node collects multiple RAM traces from each IoT device, computes and stores the projection matrix $\hat{\mathbf{V}}_i$, and forwards the processed features to the verifier. The verifier then trains the model once sufficient samples are collected, making this phase the most computationally intensive. Conversely, the testing/verification mode is less demanding. The IoT device sends a single RAM trace to the gateway node, which performs a matrix-vector multiplication and sends the result to the verifier. The verifier then performs a forward pass through the trained model for authentication, a relatively light computational task.

For instance, ignoring the limited communication overhead, our framework took approximately 1.3 ms to authenticate a single device when running on a desktop PC with an Intel Core i7-13700K and an Nvidia RTX 3080. If required, this time can be reduced further by utilizing more powerful hardware at the gateway and verifier nodes.

V. Conclusion

This paper presents a novel IoT device attestation framework, comprising preprocessing and detection based on representation learning. Utilizing RAM traces as the primary training feature, the gateway node collects and processes data for model training by the verifier node. In testing, the verifier utilizes the model to attest IoT devices, requesting preprocessed samples from the device under test via the gateway node. With a simple forward pass through the network, the test statistic verifies sample authenticity, with computational tasks managed by the gateway and verifier nodes to maintain device availability, which, alongside the 100% detection accuracy across all attack datasets, showcases the framework's suitability for real-time deployment in the IoT ecosystem.

References

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

[2] 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.

[3] 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.

[4] 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.

[5] 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.

[6] 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.

[7] 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.

[8] 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.

[9] 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.

[10] 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.

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

[12] 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.

[13] 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.

[14] 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.

[15] 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.

[16] 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.

[17] 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.

[18] 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.

[19] 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.

[20] 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.

[21] 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.

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

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

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

[25] 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.