# Machine Learning Based Attestation for the Internet of Things Using Memory Traces

Muhammad Naveed Aman, Haroon Basheer, Jun Wen Wong, Jia Xu, Hoon Wei Lim, Biplab Sikdar

***Abstract*—The advent of 4G and 5G mobile networks have made IoT devices an essential part of smart nation drives. Firmware integrity is crucial to the security of IoT systems. Most of the existing techniques for firmware attestation require a legitimate copy of an IoT device's firmware. However, firmware is considered an intellectual property (IP) of the manufacturer and may not be available. To solve this issue, this paper proposes a software based attestation technique where remote verifiers use machine learning classifiers on an IoT device's memory dump to verify the integrity of an IoT device's internal state. The experimental results from an actual prototype show that the proposed technique not only successfully detects attacks with high accuracy but also results in about 96% lower latency as compared to existing techniques. All this is achieved with high availability, low computational complexity, and without requiring a legitimate copy of the device's original firmware.**

***Index Terms*—Malware, Software Attestation, Internet of Things, Intrusion Detection, Memory Traces.**

## I. Introduction

INTERNET of Things (IoT) is gaining popularity in industrial control [1], [2], healthcare, smart cities, defense, and other fields. Therefore, protecting the data and code of IoT devices is critical. The large number of IoT devices producing data of critical nature, makes IoT devices an attractive target for cyber-criminals [3]. A recent study showed that 95% of vulnerabilities detected in smart things were firmware related [4], [5], [6]. The process of checking the authenticity of the firmware/software running on an embedded device against any malicious changes is termed attestation. A verifier is the trusted entity that decides to invoke attestation, while the prover is the embedded device proving its own authenticity. In traditional schemes, a challenge is sent to the prover by the verifier. A hash digest is calculated by the prover over its firmware source code and conveyed to the verifier. This checksum is then used by the verifier to detect compromised devices by calculating the same checksum locally using a saved copy of the prover's firmware. The main bottleneck in traditional schemes is the higher computational complexity and longer verification delay of iterating over the whole memory multiple times to calculate the hash based checksum. Moreover, the requirement of storing an actual copy of the prover's firmware at the verifier may lead to IP violations.

Existing attestation techniques can be categorized as follows: software-based, hardware-based, and hybrid. Software-based attestation techniques take advantage of the computing limitations of embedded devices, e.g., running an algorithm within a specific amount of time. These techniques are suitable for low-cost embedded devices, since they don't require any specialized hardware. Nevertheless, these techniques are computationally intensive. The attestation techniques that use advanced hardware, e.g., secure-coprocessing or trusted platform modules (TPMs) [7], are known as hardware-based. Although, the computational complexity of these techniques is low, in general, IoT devices do not have the required hardware modules. Conversely, hybrid attestation techniques attempt to find a middle ground by keeping the hardware requirement to a minimum and also reducing the computational complexity. However, most of the hybrid attestation techniques have strict architectural requirements which limits their applicability.

The degree to which the routine operation of an IoT device is unaffected by executing a security routine is referred to as availability. Real-time safety-critical applications deployed in IoT devices require security protocols that do not impair availability. However, The majority of current attestation techniques rely on the attestation routine running in an uninterrupted manner. Typically, attestation procedures may take long periods of time to complete, which yields sub par availability of IoT devices. Apart from computational complexity, hardware requirements, and availability, most of the existing techniques require a legitimate copy of a device's firmware. However, most manufactures consider the firmware as an IP and keep it protected. This invalidates the basic assumption of most existing attestation techniques and thus, making them essentially impractical.

To address the aforementioned problems, this paper proposes a software based attestation technique with significantly lower computational complexity as compared to the existing software based attestation techniques. Moreover, the proposed technique does not require a copy of the prover's legitimate

This work was supported in part by the National Research Foundation, Prime Minister's Office, Singapore under its Corporate Laboratory@University Scheme, National University of Singapore, and in part by the Singapore Telecommunications Ltd.

M. N. Aman was with the Department of Computer Science, National University of Singapore, 13 Computing Drive, Singapore 117417. He is now with the Department of Computer Science & Engineering, University of Nebraska-Lincoln, Lincoln, NE 68588, USA. e-mail: naveed.aman@unl.edu.

M. H. Basheer is with the Department of Computer Science, National University of Singapore, 13 Computing Drive, Singapore 117417, e-mail: haroon.basheer@nus.edu.sg.

J. W. Wong, J. Xu, and H. W. Lim are with the NUS-Singtel Cyber Security R&D Lab, e-mail: junwen.wong@trustwave.com, jia.xu@trustwave.com, hoonwei.lim@trustwave.com.

B. Sikdar is with the Department of Electrical and Computer Engineering, National University of Singapore, 4 Engineering Drive 3, Singapore 117576, e-mail: bsikdar@nus.edu.sg.

TABLE I: Summary of existing literature.

| Technique | [8], [9], [10], [11], [12],[13], [14] | [15] | [16] | [17] | [18] | [19], [20] | [21], [22], [23] | [24] | HAtt [25] | Proposed Technique |
|---|---|---|---|---|---|---|---|---|---|---|
| Checksum | ✓ | ✗ | ✗ | ✓ | ✓ | ✓ | ✓ | ✓ | ✗ | ✗ |
| Precise Time Stamps | ✓ | ✓ | ✗ | ✗ | ✓ | ✓ | ✓ | ✗ | ✗ | ✗ |
| Interrupts Disabled | ✓ | ✓ | ✗ | ✗ | ✓ | ✓ | ✓ | ✗ | ✗ | ✗ |
| Advanced Hardware | ✗ | ✗ | ✓ | ✓ | ✗ | ✓ | ✗ | ✓ | ✗ | ✗ |
| Low Availability | ✓ | ✓ | ✗ | ✗ | ✓ | ✓ | ✓ | ✗ | ✗ | ✗ |
| Firmware Required | ✓ | ✓ | ✓ | ✓ | ✓ | ✓ | ✓ | ✓ | ✓ | ✗ |

Checksums: Computationally complex checksums?
Precise Time Stamps: Needs precise time measurement for execution and network latency?
Interrupts Disabled: All interrupts need to be disabled?
Advanced Hardware: Needs expensive hardware primitives?
Low Availability: Affects the routine operation of the device?
Firmware Protection: An original copy of the prover's firmware required?

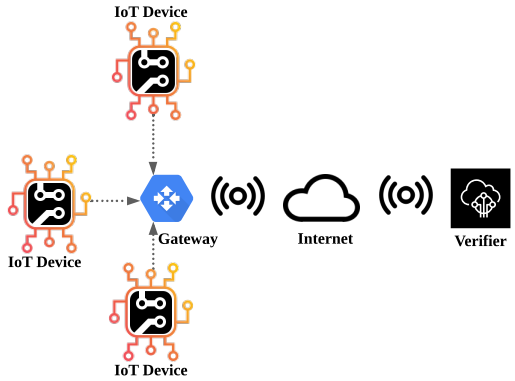



Fig. 1: Network model.

(i.e., original) firmware, ensuring IP protection. The proposed technique collects traces from an IoT device's main memory and then converts them into grayscale images. Using machine learning (ML) based classifiers on these images, the proposed technique can detect any tampering with an IoT device's firmware. The major contributions of this paper may be summarized as

(i) A software based attestation technique which does not require any computationally expensive operations.
(ii) Using memory traces rather than the actual firmware to verify the firmware integrity of an IoT device.
(iii) Validation of the proposed protocol using actual prototypes.

The rest of the paper is organized as follows. We discuss the related work in Section II. The system model is discussed in Section IV and the proposed technique is presented in Section V. The experiment design is described in Section VI while the performance analysis is presented in Section VII. The paper is concluded in Section VIII.

## II. Related Work

In the available literature, important software-based attestation techniques include SWATT [8], SCUBA [9], and SAKE [10]. SWATT performs numerous iterations on the memory in order to compute a collective hash. SWATT requires at least 50,000 iterations to detect an adversary. As a result, execution times and complexity increase. Similarly, [9] employs a technique similar to SWATT to identify contaminated sensor nodes. In a related study [10], the authors suggest a second way for establishing a secret key across sensor nodes in the event of an attack. These techniques, however, rely on precise time measurements and sophisticated procedures. To reduce computational cost, a recent approach [15] for attestation performs considers partial memory checksums. However, the attestation routine must continue to run uninterrupted, resulting in a poor level of availability. Additionally, this approach detects malware after a large number of rounds. Note that all these techniques use the actual firmware stored in the flash memory. On the other hand, the proposed technique is based on using the RAM traces instead of the actual firmware. This has two major advantages: firstly, the RAM size is magnitudes smaller than the flash memory, and secondly, the proposed technique does not iterate over the memory locations, instead we use features extracted from the memory traces to perform device attestation. This leads to significantly reduced computational complexity.

[16], [26], [17], [27], [28] are some of the prominent hardware-based attestation methodologies. The approach described in [26] presupposes TPM equipped sensor nodes. This, however, may not be a reasonable assumption for resource constrained IoT devices. The approaches described in [16] and [17] choose a cluster head based on TPM availability and form clusters. Verifying the nodes' integrity in a cluster is the responsibility of the cluster head. On the other hand, the cluster head's integrity is validated using the on-board TPM which acts as its root of trust. However, the single point of failure makes these systems susceptible to failure. Additionally, these solutions depend on the availability of TPMs, which are not appropriate for IoT devices. On the other hand, the proposed technique is based on the availability of RAM traces only and does not require any hardware modules.

SMART [18], TrustLite [19], TyTan [20], ATT-Auth [22], and PRoM [29] are some of the important existing works on hybrid attestation. These solutions, however, fall short of ensuring high availability. Furthermore, because these techniques do not incorporate wandering malware into their threat model, they are susceptible to roving malware. Another technique SMARM [24], proposes a hybrid attestation technique for

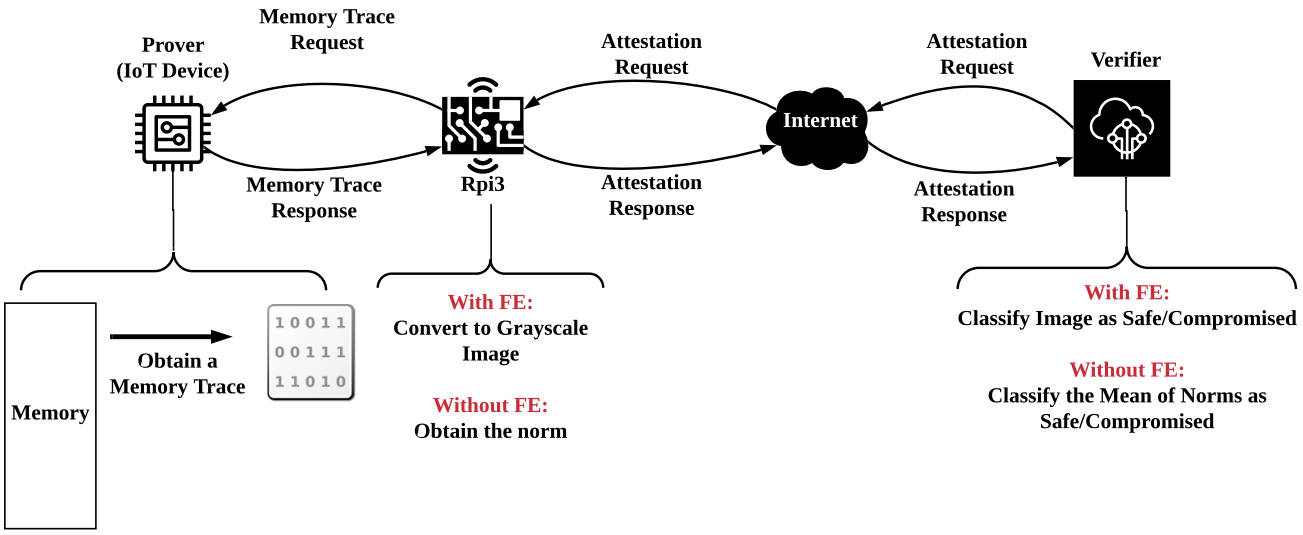



Fig. 2: Overall operation of proposed technique.

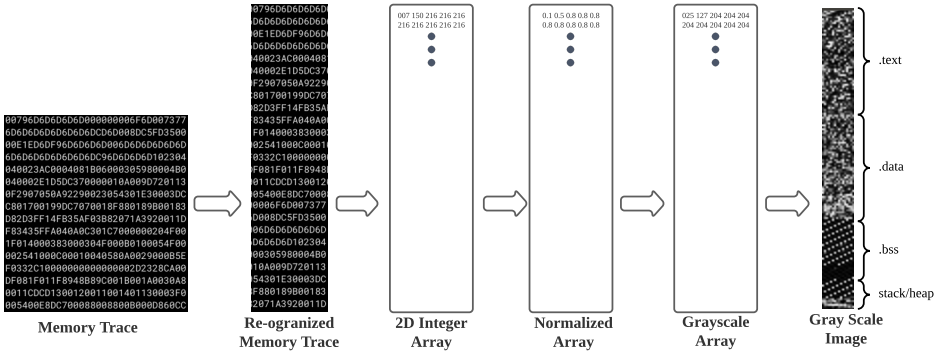



Fig. 3: From memory trace to grayscale image.

detecting roaming malware. SMARM is based on the SMART architecture with one notable exception: it relaxes the SMART atomicity constraint, allowing the attestation process to be interrupted. However, similar to software-based attestation solutions, SMARM requires a hash digest of an IoT device's full memory, leading to longer execution times and higher computational complexity. Additionally, because of the SMART architecture's unique needs, SMARM cannot be used with low-cost devices. Note that all these technique suffer from availability concerns and are susceptible to roving malware. In contrast, the proposed technique does not disable interrupts and can run without hindering the normal operation of the IoT device. Moreover, as the proposed technique is based on RAM traces rather than flash memory, it can detect roaming malware because the RAM contents do not depend on the location of the adversary in the flash memory [24].

A summary of existing techniques and comparison with the proposed technique is presented in Table I which shows that the existing attestation techniques suffer from one or more of the following problems:

i. Owing to the use of complex operations, devices have a **high overhead**.
ii. Rely on **expensive** hardware modules.
iii. Result in **low** availability due to the requirement of interrupts to be disabled.
iv. Require a copy of the legitimate firmware, leading to **IP loss**.

The proposed technique avoids these problems as follows:

1) No complex operation is carried out on the IoT device except for reading the memory dump.
2) No hardware modules are required.
3) As the memory traces are used to attest the IoT devices, the IoT device does not need to disable interrupts.
4) Firmware of IoT devices can be verified without a copy of the actual firmware using machine learning techniques.

## III. Background

This section describes the machine learning techniques used in the proposed technique.

### A. Classification and Regression Trees

A classification and regression tree (CART) minimizes the Gini diversity index (gdi) as follows:

$$Gini(D) = 1 - \sum_{l=1}^{L} (\frac{|C_l|}{|D|})^2, \tag{1}$$

where $L$ denotes the number of classes and $C_l$ denotes the number of samples in the $l-th$ class. The gdi can be thought of as the inverse of the goodness-of-fit for a model, i.e., a small value of gdi indicates a better match. In each iteration, the CART separates $D$ into two subsets and chooses the prediction that results in the smallest gdi of the subsets to make the decision at the current node/vertex of the tree by evaluation:

$$Gini(D, I_m) = \frac{|D_1|}{|D|} Gini(D_1) + \frac{|D_2|}{|D|} Gini(D_2), \tag{2}$$

where $I_m$ is one of the features with $m \in 1, 2$ and $D_1$ and $D_2$ are the two divided subsets of data. Thus, starting with the first node (referred to as the root node), the CART builds the binary tree for classification iteratively using the training data.

### B. Logistic Regression

In logistic regression, 'logit' is used as the link function:

$$\log(\frac{p}{1-p}) = \boldsymbol{\beta}^T \mathbf{x} \tag{3}$$

where the set of coefficients is represented by $\boldsymbol{\beta} = [\beta_0, \beta_1, \cdots \beta_n]^T$, and $\mathbf{x} = [x_1, x_2, \cdots, x_n]^T$ is an array formed by the features of an observation. Which leads to a sigmoid function:

$$p = \frac{1}{1 + e^{-\boldsymbol{\beta}^T \mathbf{x}}} \tag{4}$$

The weights $\boldsymbol{\beta}$ can be calculated by using the gradient descent method to minimize the cost function [30].

### C. Support Vector Machines

The hyper-plane for accurately dividing the training data set (to obtain the largest geometric margin) in a support vector machine (SVM) with a linear kernel is given by:

$$\mathbf{w}^T \mathbf{x} + b = 0 \tag{5}$$

where the predictor array is denoted by $\mathbf{x}$ and an observation is represented by $\mathbf{x}$, while the SVM hyper-plane coefficients are $\mathbf{w} = [w_1, w_2]^T$ and $b$ assuming two features. The margin ($d$) for an observation ($\mathbf{x}$) to a hyper-plane is given by:

$$d = \frac{1}{\|\mathbf{w}\|} (\mathbf{w}^T \mathbf{x} + b). \tag{6}$$

SVM learns by finding the hyper-plane $(\mathbf{w}, b)$ such that the minimum margin $d$ is maximized. This leads to a convex quadratic optimization problem [31]:

$$\begin{aligned} \underset{\mathbf{w}}{\text{minimize}} \quad & f(\mathbf{w}) = \frac{1}{2} \|\mathbf{w}\|^2 \\ \text{subject to} \quad & (\mathbf{w}^T \mathbf{X} + b) \geqslant 1 \end{aligned} \tag{7}$$

where $\mathbf{X}$ is the data set with $N$ observations, i.e., $\mathbf{X} = [\mathbf{x_1}, \mathbf{x_2}, \cdots, \mathbf{x_N}]$.

### D. Naive Bayes Classifier

The Naive Bayes classifier is based on the popular Bayes theorem:

$$p(A|B) = \frac{p(B|A)p(A)}{p(B)}, \tag{8}$$

i.e., the probability of the event $A$ given an observation B. The features are assumed to be independent. Thus, based on an observation the probability of each class is calculated and the observation is classified as the class with the highest probability.

## IV. Network Model

Figure 1 presents the network model for this paper with the following entities:

i.) **IoT Device**: A set of IoT devices is connected to a gateway through a wireless or wired connection. An IoT device acts as the prover for an attestation request. IoT devices are assumed to be resource constrained in terms of processing, memory, and energy.

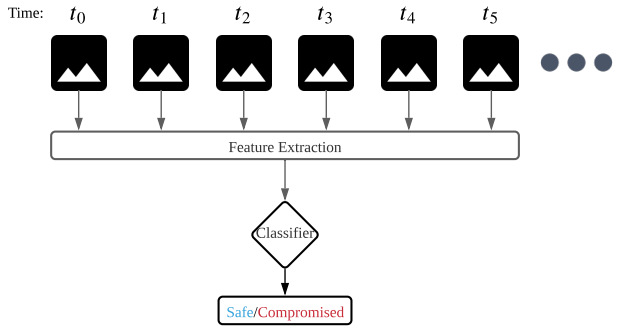
ii.) **Gateway**: Typically IoT devices do not have the complete transmission control protocol/Internet protocol (TCP/IP) stack. Therefore, they need to be connected to border router elements for Internet connectivity. In this paper, the gateway serves two purposes: (i) it functions as a device manager and a router for a set of IoT devices, enabling communication between an IoT device and a verifier; (ii) The gateway is used to process the raw hex dump of random access memory (RAM) collected from an IoT device. The gateway is assumed to be more powerful and less resource constrained than the IoT devices. The gateway is in-turn connected to a remote verifier through the Internet.

iii.) **Verifier**: The verifier is a trusted remote server that can initiate an attestation request. The verifier maintains the dictionary of gateways, IoT devices under each gateway, and application variants executed in each IoT device. Thus, It stores the legitimate traces for each IoT device for training classifiers to detect compromised devices. The frequency of attestation and training of classifiers is managed by the Verifier.

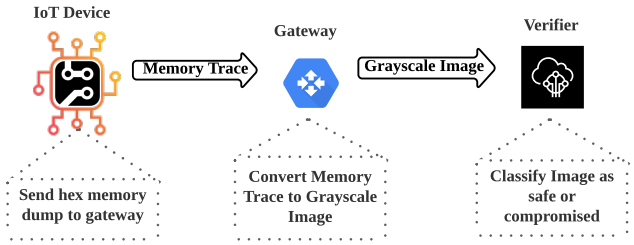
## V. Proposed Technique

The overall operation of the proposed technique is shown in Figure 2. The first step in the proposed technique is to extract the hex memory dump from the IoT device as discussed in Section VI and called a memory trace from hereon. A memory trace can be considered as a two-dimensional (2D) array of 8 bit unsigned integers. Therefore, the memory trace can be visualized as a grayscale image using the typical range of $[0, 255]$. In this paper, based on empirical results, we fixed the image width at 192-bits or 24 8-bit unsigned integers. The width of the image is fixed while the height depends on the size of memory. The process of converting a memory trace into a grayscale image is shown in Figure 3 with the following steps:

i. Re-organize the memory trace by changing the width and height according to the memory size [33].




(a) Training




(b) Attestation

Fig. 4: Proposed technique with a machine learning based classifier and feature engineering.

ii. Each word in the memory trace is converted from hexadecimal into its integer value.
iii. An $l_1$ normalization is done along the columns.
iv. Each element of the resulting array is multiplied by 255.
v. Encode each element of the resulting array to an unsigned 8-bit integer (uint8).

A typical memory is divided into the following sections:

i. **.text**: Use to store the executable code.
ii. **.data**: Stores initialized static and global variables.
iii. **.bss**: Memory allocated for uninitialized static and global variables.
iv. **stack/heap**: Memory allocated for dynamic memory allocation on a heap or used for function calls, interrupts, and local variables.

Thus, the content of memory is dependent on the code being executed and the data it is using. Based on this insight, the proposed technique aims at classifying a memory trace image as "safe" or "compromised". A safe or legitimate memory trace is one which does not indicate any tampering with the firmware, while, a compromised memory trace image is one that indicates abnormal behaviour of the memory and thus, points to possible tampering with the IoT device firmware.

To classify an image as safe or compromised, we can take two approaches, machine learning based classification with or without feature engineering. The classifiers considered in this paper include classification and regression trees (CART), logistic regression (LR), support vector machines (SVM), K-means nearest neighbour (K-NN), multi-layer perceptron (MLP), and Gaussian Naive Bayes (NB) [34].

### A. Classifier with Feature Engineering

The main focus of this section is to apply feature engineering to extract features to improve the classification accuracy of the classifiers. HOG is a feature descriptor that is used in computer vision and image processing to aid in the detection of objects. The approach identifies instances of gradient orientation within a specified region of an image. The HOG descriptor is concerned with an object's structure or form. It outperforms all other edge descriptors because it computes features using both the magnitude and angle of the gradient. It generates histograms for the image's regions based on the gradient's magnitude and orientation. Thus, given the dynamic nature of RAM, HoG was selected based on its robustness to changing objects. The steps to extract the HoG features are as follows:

i. The gradient is calculated by combining the image's magnitude and angle. The horizontal and vertical gradients $G_x$ and $G_y$ are initially calculated for each pixel in a block of $3 \times 3$ pixels. To begin, $G_x$ and $G_y$ are calculated for each pixel value using:

$$G_x(i,j) = I(i,j+1) - I(i,j-1) \quad (9)$$
$$G_y(i,j) = I(i-1,j) - I(i+1,j), \quad (10)$$

where $i$ and $j$ represent the the rows and columns, respectively. Furthermore, to calculate the magnitude $\mu$ and angle $\theta$ of the pixels, we have:

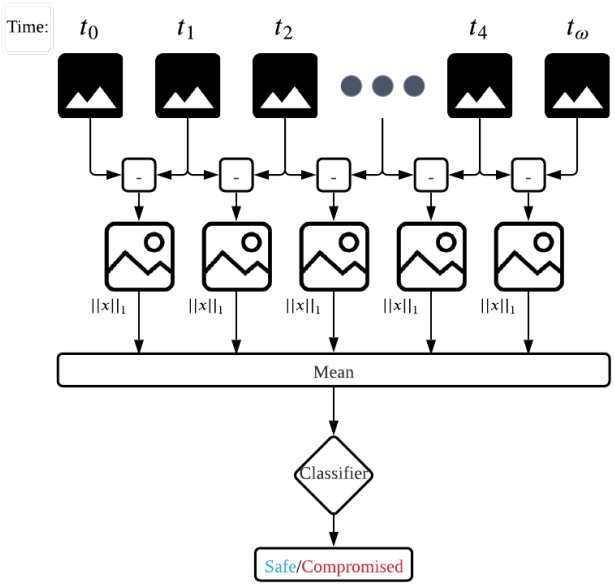
$$\mu = \sqrt{G_x^2 + G_y^2} \quad (11)$$
$$\theta = \left| \tan^{-1}\left(\frac{G_y}{G_x}\right) \right| \quad (12)$$

ii. The gradient matrices, i.e., magnitude and angle matrix, are divided into blocks ($8 \times 8$ cells) and for each cell in a block, we calculate:

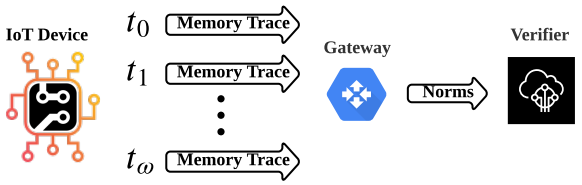
$$j = \left\lfloor \left(\frac{\theta}{\Delta\theta} - \frac{1}{2}\right) \right\rfloor \quad (13)$$
$$C_j = \theta(j + 0.5) \quad (14)$$
$$V_j = \mu \cdot \left[\frac{\theta}{\Delta\theta} - \frac{1}{2}\right] \quad (15)$$
$$V_{j+1} = \mu \cdot \left[\frac{\theta - C_j}{\Delta\theta}\right] \quad (16)$$

iii. The values of $V_j$ and $V_j + 1$ are appended to an array (used as a bin for a block) at the index of the $j$th and $(j+1)$th bin generated for each pixel. Thus, we obtain the histogram for all the blocks.
iv. The scikit-image processing library was used to extract the features from each block.

2000 features were extracted using the HoG technique out of which 6 features were used for classification. The overall operation of the proposed technique is shown in Figure 4. An IoT device sends a hex memory dump to the gateway through a wired or wireless connection. The gateway converts the memory trace into an image (as described previously) and forwards it to the verifier. The image is classified as safe or compromised by the verifier.




(a) Training




(b) Attestation

Fig. 5: Proposed technique with a machine learning based classifier and no feature engineering.

### B. Classifier without Feature Engineering

Although, the proposed technique with feature engineering may result in higher accuracy of detection, it may entail higher communication overheads. This is due to the fact that a complete grayscale image of the memory dump of the IoT device needs to be sent to the verifier (each time) for attestation. To solve this issue, we also propose a light-weight method for attestation without feature engineering. The process of training these classifiers is shown in Figure 5(a). Memory traces are converted into training images, a "difference" image is obtained by subtracting the previous image from the current image as follows:

$$\Delta X = X_{t_{i+1}} - X_{t_i}, \tag{17}$$

i.e., the difference of two consecutive images is obtained by subtracting the matrix $X_{t_{i+1}}$ from the matrix $X_{t_i}$, where $t_i$ represents the $i$th time sample. The difference image $\Delta X$ is then used to calculate the $l_1$ norm $||x||_1$ as follows:

$$||x||_1 = \sum_{i=1,j=1}^{i=m,j=n} |\Delta x_{ij}|, \tag{18}$$

where $m$ and $n$ represent the number of rows and columns in $\Delta X$ and $\Delta x_{ij}$ denotes the pixel at the $i$th row and $j$th column in the difference image. After calculating the $l_1$ norms of the difference images over the training data set, these $l_1$ norms were then used to train the classifiers with pre-assigned tags of safe or compromised.

The trained classifiers are then deployed at the verifier. The proposed attestation process after receiving an attestation request from the verifier is shown in Figure 5(b) and has the following steps:

i. The IoT device sends $\omega$ consecutive memory traces to the gateway.
ii. The gateway converts the consecutive memory traces into grayscale images and then calculates the $l_1$ norms over the difference images.
iii. The gateway calculates the mean of $(\omega - 1)$, $l_1$ norms, denoted by $\mu_{l_1}$. The gateway then sends this mean value to the verifier in a secure way (assuming a pre-shared symmetric key between the gateway and verifier). Note that instead of using just two memory traces to generate the $l_1$ norm, we used an average over a window of size $\omega$ to improve the accuracy of detection.
iv. After receiving $\mu_{l_1}$, the verifier uses this as a predictor with the trained classifier to detect attacks (if any).

Thus, it is clear from Figure 2 and the above description that the proposed technique only uses RAM traces to perform attestation and does not rely on keeping a firmware copy at the verifier. This section presented the proposed techniques for IoT device attestation with and without feature engineering. The main difference between the two techniques is the feature generation, i.e., with feature engineering the proposed technique has 2000 features to select from while without feature engineering the proposed technique uses the $l_1$ norm of the difference of consecutive images as the only predictor.

## VI. Experimental Design

The system model for the prototype developed to assess the proposed technique is shown in Figure 6. We implemented our gateway for IoT devices using a RaspberryPi-3 (Rpi3) with Broadcom BCM2837 64-bit central processing unit (CPU) operating at 1.2 Ghz. It has 4 universal serial bus (USB) ports through which up to 4 IoT devices can be connected. The Rpi3's BCM43438 wireless local area network (LAN) is used to establish a TCP/IP connection with the remote verifier. The Rpi3 has a 32 GB micro secure digital (SD) card running on the ARM port of the Debian Stretch desktop distro. The gateway has 3 important functions as follows:

i. Maintain TCP/IP connection with the verifier and a wireless or wired connection (through serial port) with the IoT devices.
ii. Collect RAM traces from the IoT device upon the verifier's request and synchronise the traces.
iii. Convert memory traces into images and send the norms or the images themselves to the verifier.

To create an experiment which is representative of a wide range of IoT devices, we selected the commonly used micro-controller, the Arduino Uno. Arduino supports analogue-to-digital input with a possibility of connecting light, temperature or sound sensor modules. Such sensors with the serial

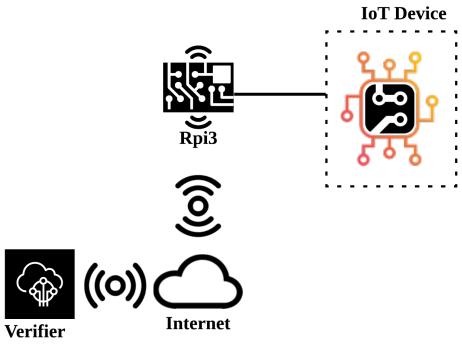



Fig. 6: Prototype system model.

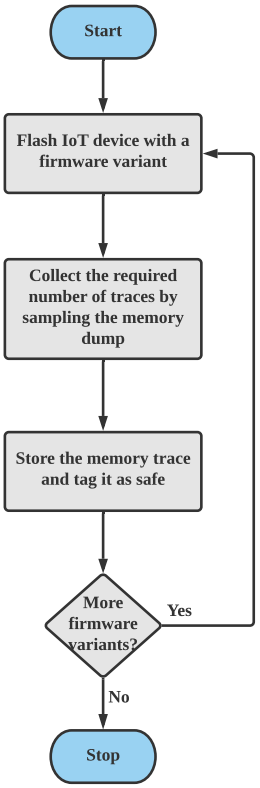



Fig. 7: Data set generation.

peripheral interface (SPI) or inter-integrated circuit interface (I2C) may also be used to cover up to 99% of these apps' market [35], [36]. Arduino Uno is an 8-bit ATmega328P AVR based microcontroller (mcu) development board with 32 KB of flash memory and 2 KB of RAM. The mcu's Universal Asynchronous Receiver Transmitter (UART) TTL is used for serial communication and data exchange with the gateway at a baud rate of 115200. The RAM is accessed through direct addressing mode ranging between 0x0100 to 0x08FF.

To verify the effectiveness of the proposed technique, we considered four different application types with different task objectives ranging from sensing to cryptography. For ease of notation the developed firmware are represented by $F_1$, $F_2$, $F_3$, and $F_4$ and are described as follows:

1) $F_1$**:** Encrypts a given input data block and then decrypts it to retrieve the original data. The encryption and decryption is done in separate functions and executed in an iterative manner.
2) $F_2$**:** An XTS-AES block cipher based on variable block cipher encryption. Plain Text, key and tweak modules are considered for encryption. For simplicity, key and tweak are sent from the gateway along with read input.
3) $F_3$**:** Samples the voltage at the temperature sensor pin and converts it into the corresponding Celsius value. A separate method is used to read the temperature and store the value, which is called in the Arduino loop method followed by the collection of the trace.
4) $F_4$**:** Changes the brightness of the built-in light emitting diode (LED), i.e., pin 13, based on the analog values read from the sensor (pin 3). The LED control method is defined separately and called in the Arduino loop method followed by trace collection

We observe that $F_1$ and $F_2$ are applications based on cryptography, while, $F_3$ and $F_4$ are sensor based applications. Note that the objective of considering these various types of applications is to evaluate the effectiveness of the proposed technique under various circumstances.

After developing different types of firmware, the next task is to introduce an attack on these applications. For this purpose, we consider three variants:

1) **Tampering with Control Dependency:** The control dependency graph of the code is tampered by introducing new pointers. This affects the .data section of memory.
2) **Tampering with Functional Dependency:** A new stack frame is generated in the stack section of memory.
3) **Tampering with Variable Initialization:** The code for variable initialization is tampered, leading to changes in the .bss section of memory.

The firmware generated after these three types of tampering is represented by $\mathcal{M}_C$, $\mathcal{M}_S$, and $\mathcal{M}_V$, respectively. Note that these three types of dependencies are the most critical when it comes to good software engineering practices [37]. Therefore, the results in this section are representative of the majority of tampering attacks.

The next step in our experiment design is to generate the training and testing data sets. For this purpose, Figure 7 shows the process of collecting memory traces with safe and compromised firmware. The training is done using the following steps:

1) A specific firmware is flashed onto the IoT device.
2) The IoT device is run for a specific time and the gateway is used to obtain 1500 traces. Each collected trace is tagged as *safe*.

Figure 7 shows that the IoT device is flashed using each of the firmware variants and then the gateway is used to collect the memory traces. These traces are stored and tagged safe at the verifier to train the proposed classifiers. For testing the classifiers, 1500 extra traces were collected composed of 300 safe traces and 400 traces for each type of attack, i.e., control

dependency, functional dependency, and variable initialization tampering.

## VII. Performance Analysis

In this section we discuss the accuracy of detecting compromised IoT devices, i.e., IoT devices whose firmware has been tampered, using the proposed technique. We present a discussion on the comparison and trade-offs at the end of the section.

### A. Performance of Classifiers with Feature Engineering

The top four classifiers in terms of classification accuracy on the test data set are shown in Table II. Note that every firmware has a different feature engineering model. We observe that MLP can detect attacks on the IoT device firmware almost perfectly in most of the variants except for $F_2$. To check the statistical significance of these results we repeated the process of training ML models 100 times to see the effect of randomly splitting the dataset on the performance of the models. For this purpose, we used the corrected paired Student's t-test proposed by Nadeau and Bengio [38]. The results are shown in Figure 8. We observe that at a significance level of $5\%$, the null hypothesis can not be rejected for LR and SVM. However, it is statistically evident that the performance of MLP is significantly different from the other classifiers, i.e., the null hypothesis can be rejected at a significance level of $5\%$. Thus, we only consider MLP in the following discussion.

| Firmware | LR | SVM | MLP | NB |
|---|---|---|---|---|
| $F_1$ | 0.98 | 0.98 | **1.00** | 0.93 |
| $F_2$ | 0.65 | 0.65 | **0.83** | 0.83 |
| $F_3$ | 0.99 | 0.99 | **1.00** | 0.98 |
| $F_4$ | 1.00 | 1.00 | **1.00** | 0.98 |

TABLE II: AUC of ML based classifiers with feature engineering.

| | **LR** ($p$) | **SVM** ($p$) | **MLP** ($p$) | **NB** ($p$) |
|---|---|---|---|---|
| **LR** | – | 0.1041 | 0.0018 | 0.0007 |
| **SVM** | – | – | 0.0069 | 0.0109 |
| **MLP** | – | – | – | 0.0006 |
| **NB** | – | – | – | – |

(a) $F_1$

| | **LR** ($p$) | **SVM** ($p$) | **MLP** ($p$) | **NB** ($p$) |
|---|---|---|---|---|
| **LR** | – | 0.1089 | 0.0021 | 0.0019 |
| **SVM** | – | – | 0.0219 | 0.0201 |
| **MLP** | – | – | – | 0.0004 |
| **NB** | – | – | – | – |

(b) $F_2$

| | **LR** ($p$) | **SVM** ($p$) | **MLP** ($p$) | **NB** ($p$) |
|---|---|---|---|---|
| **LR** | – | 0.2101 | 0.0019 | 0.0005 |
| **SVM** | – | – | 0.0019 | 0.0110 |
| **MLP** | – | – | – | 0.0010 |
| **NB** | – | – | – | – |

(c) $F_3$

| | **LR** ($p$) | **SVM** ($p$) | **MLP** ($p$) | **NB** ($p$) |
|---|---|---|---|---|
| **LR** | – | 0.1401 | 0.0209 | 0.0400 |
| **SVM** | – | – | 0.0210 | 0.0040 |
| **MLP** | – | – | – | 0.0056 |
| **NB** | – | – | – | – |

(d) $F_4$

Fig. 8: Corrected Student's t-test significance values for ML based classifiers with feature engineering.

A confusion matrix is shown in Figure 9 with the following definitions:

1) **True Negative (TN)**: A safe test image is observed to be part of the safe traces and is correctly classified as safe.
2) **False Negative (FN)**: A compromised test image is observed to be part of the safe traces and is erroneously classified as safe.
3) **True Positive (TP)**: A compromised test image is not observed to be part of the safe traces and is correctly classified as compromised.
4) **False Positive (FP)**: A safe test image is not observed to be part of the safe traces and is erroneously classified as compromised.

Moreover, P and P$'$ denote positive in terms of ground truth and classifier outcome, respectively, i.e., the device is compromised and the attestation was unsuccessful. Similarly, N and N$'$ are used to represent successful attestation in terms of ground truth and classifier outcome, respectively.

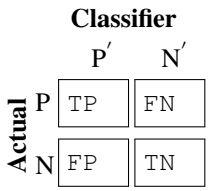
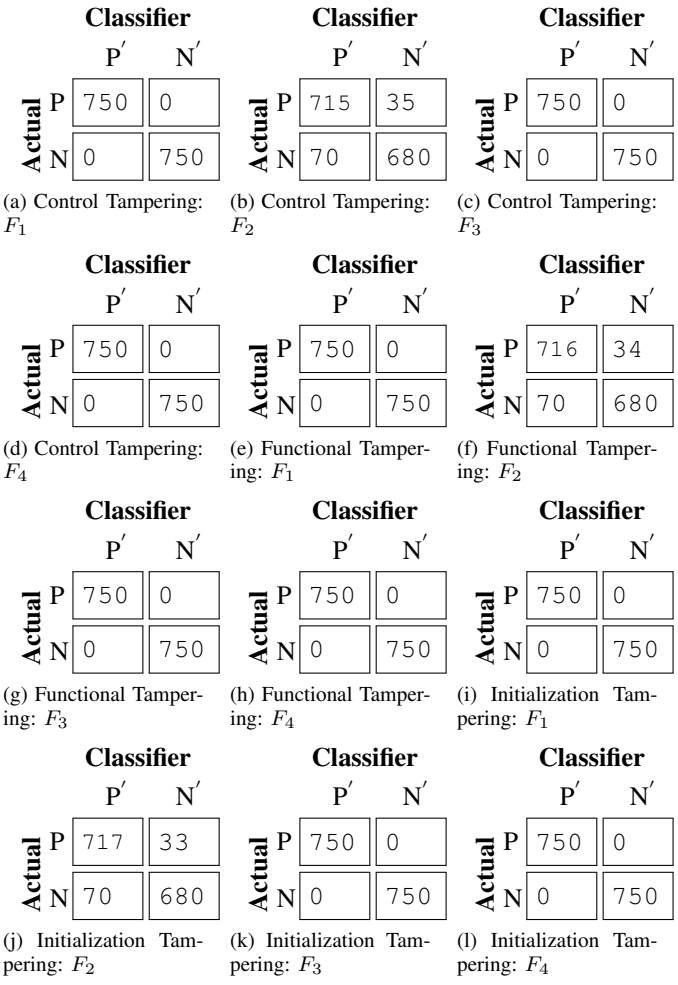



Fig. 9: Confusion matrix definition.

Note that missed detection of compromised devices should be kept low to make sure a compromised device is detected with high probability and does not infect other devices in the network. To observe the performance of the proposed classifier under a missed detection probability ($P_{MD}$) of less than $5\%$, the confusion matrices for the MLP classifier for the four firmware are shown in Figure 10. The matrices are shown for the three types of attacks defined in Section VI. It is observed that the performance of the proposed feature engineering based ML classifier does not depend on the type of attack.




(a) Control Tampering: $F_1$ (b) Control Tampering: $F_2$ (c) Control Tampering: $F_3$ (d) Control Tampering: $F_4$ (e) Functional Tampering: $F_1$ (f) Functional Tampering: $F_2$ (g) Functional Tampering: $F_3$ (h) Functional Tampering: $F_4$ (i) Initialization Tampering: $F_1$ (j) Initialization Tampering: $F_2$ (k) Initialization Tampering: $F_3$ (l) Initialization Tampering: $F_4$

Fig. 10: Confusion Matrices for MLP, $P_{MD} \leq 5\%$.

### B. Performance of Classifiers without Feature Engineering

Figure 11 shows the area under the curve (AUC) for top four classifiers. We observe that classification trees clearly outperform the other classifiers, while, the performance of SVM classifier is the worst and close to random classification. The AUC values for the classifiers are given in Table III for the test data set. We observe that at a window size of 30, CART has an AUC of approximately 95%. We observe that a window size of 30 images results in the best AUC performance for the CART classifier. This is also apparent from Figure 12 which shows the receiver operating characteristic curve (ROC) of the CART classifier using different window sizes. The corrected Student's t-test results for the classifiers without feature engineering with a window size of 30 images are given in Figure 13. We observe that LR and SVM performance is the same. However, the performance of CART and NB is statistically significantly different from the other classifiers. Thus, CART is considered the best among the other classifiers and is considered in the following discussion.

| Firmware Variant | Window Size | CART | LR | SVM | NB |
|---|---|---|---|---|---|
| $F_1$ | 10 | 0.9315 | 0.8741 | 0.8741 | 0.8390 |
| | 20 | 0.9524 | 0.8872 | 0.8872 | 0.8463 |
| | 30 | **0.9711** | 0.9037 | 0.9037 | 0.8622 |
| | 40 | 0.9614 | 0.9091 | 0.9091 | 0.8788 |
| $F_2$ | 10 | 0.7107 | 0.5574 | 0.5569 | 0.6510 |
| | 20 | 0.8633 | 0.5955 | 0.5000 | 0.6881 |
| | 30 | 0.9393 | 0.6044 | 0.5000 | 0.7630 |
| | 40 | **0.9477** | 0.6612 | 0.5702 | 0.7603 |
| $F_3$ | 10 | 0.7062 | 0.5534 | 0.5007 | 0.5337 |
| | 20 | 0.8245 | 0.6427 | 0.6405 | 0.6484 |
| | 30 | **0.9304** | 0.6015 | 0.5000 | 0.6489 |
| | 40 | 0.8871 | 0.6832 | 0.3388 | 0.6832 |
| $F_4$ | 10 | 0.9212 | 0.7501 | 0.4682 | 0.7888 |
| | 20 | 0.9486 | 0.7801 | 0.4959 | 0.8765 |
| | 30 | **0.9622** | 0.7615 | 0.3481 | 0.7970 |
| | 40 | 0.9091 | 0.7438 | 0.6832 | 0.8099 |

TABLE III: AUC of ML based classifiers without feature engineering.

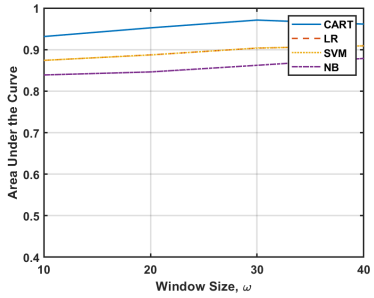
To further observe the performance of the proposed classifier under a missed detection probability ($P_{MD}$) of less than $5\%$, the confusion matrices for the CART classifier for the four firmware are shown in Figure 14. It is observed that the CART classifier performed slightly better in case of variable initialization attack. However, the difference is not significant and we observe stable performance from the proposed ML based classifier without feature engineering across the three attacks.

Thus, we observe that the proposed technique with feature engineering results in an overall better performance in terms of AUC.

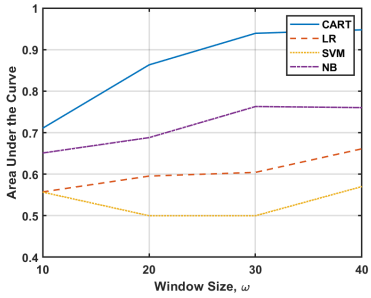
### C. Effect of Feature Engineering

Although, in the previous section we observed that feature engineering improves the AUC performance, AUC or classification accuracy is not the only performance metric that can decide the ultimate winner. There are two critical aspects related to IoT device attestation:

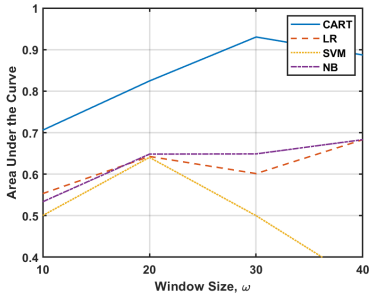
*1) Prover Side Complexity:* This relates to the computational complexity and latency required by the attestation routine on the prover side. Too much complexity or latency not



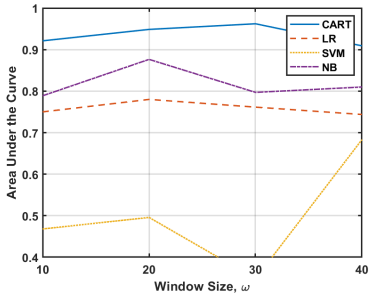

(a) $F_1$




(b) $F_2$

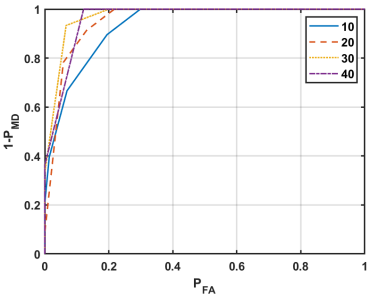



(c) $F_3$

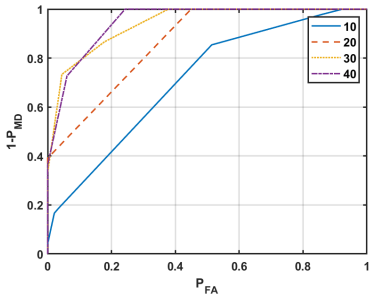



(d) $F_4$

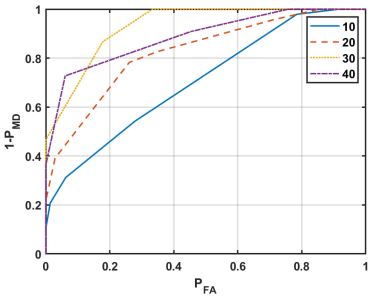
Fig. 11: Classification performance of ML based classifiers without feature engineering.



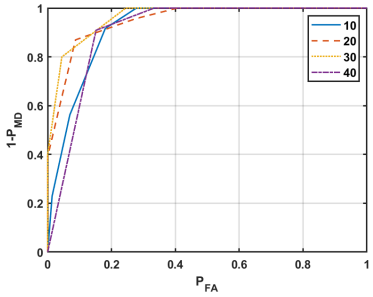

(a) $F_1$




(b) $F_2$




(c) $F_3$




(d) $F_4$

Fig. 12: Effect of window size on AUC performance of CART classifier.

| | LR ($p$) | SVM ($p$) | MLP ($p$) | NB ($p$) |
|---|---|---|---|---|
| **LR** | – | 0.2101 | 0.0010 | 0.0007 |
| **SVM** | – | – | 0.0011 | 0.0009 |
| **CART** | – | – | – | 0.0001 |
| **NB** | – | – | – | – |

(a) $F_1$

| | LR ($p$) | SVM ($p$) | MLP ($p$) | NB ($p$) |
|---|---|---|---|---|
| **LR** | – | 0.2209 | 0.0011 | 0.0009 |
| **SVM** | – | – | 0.0019 | 0.0009 |
| **MLP** | – | – | – | 0.0014 |
| **NB** | – | – | – | – |

(b) $F_2$

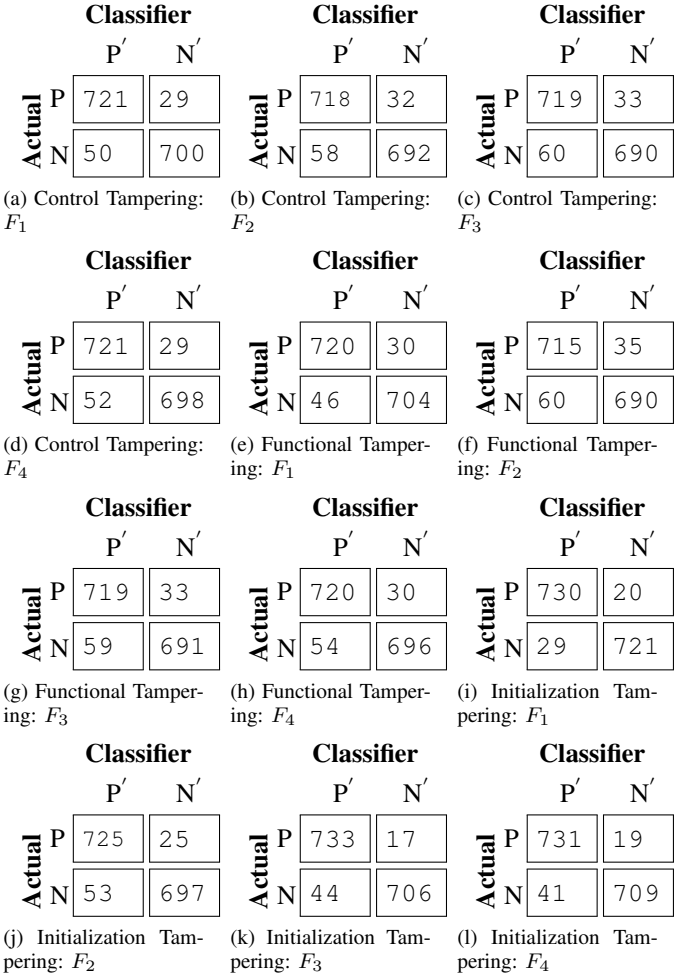
| | LR ($p$) | SVM ($p$) | MLP ($p$) | NB ($p$) |
|---|---|---|---|---|
| **LR** | – | 0.2001 | 0.0009 | 0.0006 |
| **SVM** | – | – | 0.0010 | 0.0007 |
| **MLP** | – | – | – | 0.0020 |
| **NB** | – | – | – | – |

(c) $F_3$

| | LR ($p$) | SVM ($p$) | MLP ($p$) | NB ($p$) |
|---|---|---|---|---|
| **LR** | – | 0.2400 | 0.0001 | 0.0030 |
| **SVM** | – | – | 0.0002 | 0.0031 |
| **MLP** | – | – | – | 0.0006 |
| **NB** | – | – | – | – |

(d) $F_4$

Fig. 13: Corrected Student's t-test significance values for ML based classifiers without feature engineering.




(a) Control Tampering: $F_1$ (b) Control Tampering: $F_2$ (c) Control Tampering: $F_3$ (d) Control Tampering: $F_4$ (e) Functional Tampering: $F_1$ (f) Functional Tampering: $F_2$ (g) Functional Tampering: $F_3$ (h) Functional Tampering: $F_4$ (i) Initialization Tampering: $F_1$ (j) Initialization Tampering: $F_2$ (k) Initialization Tampering: $F_3$ (l) Initialization Tampering: $F_4$

Fig. 14: Confusion matrices for CART, $P_{MD} \leq 5\%$, window size = 30.

only increases the time for carrying out IoT attestation but also limits the applicability of the technique to various IoT devices. Note that a higher latency results in low availability of the IoT device as well [25].

The computational complexity of using an ML based classifier with feature engineering in the proposed technique is essentially just the complexity of collecting a memory trace and converting it into a grayscale image. However, the computational complexity of using an ML based classifier without feature engineering in the proposed technique requires generating multiple grayscale images (depending on the window size) and calculating the mean of $l_1$ norms. Denoting the size of a grayscale image by $n_p = r_g \times c_g$, where $n_p$ is the total number of pixels given the number of horizontal and vertical pixels, i.e., $r_g$ and $c_g$, respectively. The computational complexity of taking the difference of two grayscale images is $O(n_p)$. Then, the computational complexity of calculating the $l_1$ norm of the difference images and taking the mean is also given by $O(n_p)$. Therefore, if we denote the computational complexity of collecting a memory trace and converting it to a grayscale image by $\zeta_{gray} = O(n_p)$, then the computation complexity of using an ML based classifier with feature engineering is $\zeta_{FE} = \zeta_{gray}$, while, the computation complexity of an ML based classifier without feature engineering is $\zeta_{l_1} = \omega\zeta_{gray} + (\omega - 1)O(n_p) \approx O(n_p)$.

The latency is defined as the execution time on the prover side to complete one round of an attestation request. Any security primitive which affects the normal operation of an IoT device is said to have low availability. As an IoT device may perform time critical operations, attestation techniques designed for IoT devices should have high availability, one aspect of which is low latency. The latency of the proposed technique is presented along with a comparison with state-of-the-art existing attestation techniques in Table IV. These results are for a probability of evasion less than 5%. It is observed that the proposed technique with feature engineering has 99%, 99%, 99%, 97%, and 96% lower latency compared to SWATT [8], SMARM [24], [15], TyTan [20], and HAtt [25], respectively. We observe similar results for the proposed technique without feature engineering. Moreover, the difference between the execution time of the proposed technique with and without feature engineering is negligible.

*2) Scalability:* Given the large number of IoT devices, the bottleneck in terms of scaling an attestation technique to thousands (or even more) IoT devices is the communication

TABLE IV: Comparison of Latency

| Technique | | Latency (sec) |
|---|---|---|
| SWATT [8] | | 80.97 |
| SMARM [24] | | 53.06 |
| [15] | | 0.855 |
| TyTan [20] | | 0.126 |
| HAtt [25] | | 0.104 |
| Proposed Technique | DL Classifier | 0.004 |
| | ML Classifier | 0.0043 |

overhead. The communication overhead is the number of bytes/bits that need to be sent from the prover to the verifier. Although, the proposed technique with feature engineering requires the gateway to collect a single memory trace from the IoT device, the whole grayscale image is sent to the verifier. Therefore, the communication overhead between an IoT device and gateway, and between the gateway and verifier scales as $O(n_p)$. On the other hand, the proposed technique without feature engineering requires the prover to send $\omega$ memory traces to the gateway leading to a communication overhead of $O(\omega n_p)$. However, the gateway only sends a single floating point value of 4 bytes to the verifier, which translates into $O(1)$ communication overhead. This shows that an ML based classifier without feature engineering has significantly better scalability in terms of the communication between the gateway and the verifier, while an ML based classifier with feature engineering scales well in terms of the communication between the IoT device and the gateway.

From the above discussion, we observe a trade-off between accuracy and scalability between the proposed technique with and without feature engineering, respectively. Although, the proposed technique without feature engineering leads to lower accuracy as compared to the proposed technique with feature engineering, the accuracy is still above 93% which in most applications may be an acceptable performance.

## VIII. Conclusion

This paper presented a software based attestation technique that uses an IoT device's memory to verify the integrity of its internal state. The proposed technique collects memory traces from an IoT device and converts them into grayscale images. These images are then used to detect any attack on the IoT device's firmware by classifying them into safe or compromised. The proposed technique can effectively detect attacks by training ML classifiers with or without feature engineering. Although, using a feature engineering based ML classifier results in higher AUC performance, ML based classifiers without feature engineering result in better scalability. The choice of the type of classifier used depends on the application requirements. The experimental results showed that the proposed technique is at least 96% faster than existing attestation techniques at the same level of accuracy. It is worth noting that these performance gains are achieved without a legitimate copy of the prover's firmware.

## Acknowledgment

The authors would like to thank Mr. Virat Kohli from the Birla Institute of Technology India, who worked in the capacity of an internee at the NUS-Singtel Cybersecurity R&D Lab.

## References

[1] M. M. Hassan, A. Gumaei, S. Huda and A. Almogren, "Increasing the Trustworthiness in the Industrial IoT Networks Through a Reliable Cyberattack Detection Model," in *IEEE Transactions on Industrial Informatics*, vol. 16, no. 9, pp. 6154-6162, Sept. 2020.

[2] C. Zhou, X. Li, S. Yang and Y. Tian, "Risk-Based Scheduling of Security Tasks in Industrial Control Systems With Consideration of Safety," in *IEEE Transactions on Industrial Informatics*, vol. 16, no. 5, pp. 3112-3123, May 2020.

[3] J. Lee, L. Kim and T. Kwon, "FlexiCast: Energy-Efficient Software Integrity Checks to Build Secure Industrial Wireless Active Sensor Networks," in *IEEE Transactions on Industrial Informatics*, vol. 12, no. 1, pp. 6-14, Feb. 2016.

[4] L. Ilascu, "When their firmware is vulnerable, its up to you to protect your smart devices," [Online] Available: https://www.bitdefender.com/box/blog/iot-news/bitdefender-box-data-firmware-vulnerable-protect-smart-devices/, Accessed: 5 May 2019.

[5] M. Cheminod, L. Durante, L. Seno and A. Valenzano, "Semiautomated Verification of Access Control Implementation in Industrial Networked Systems," in *IEEE Transactions on Industrial Informatics*, vol. 11, no. 6, pp. 1388-1399, Dec. 2015.

[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," in *IEEE Transactions on Industrial Informatics*, vol. 16, no. 3, pp. 1602-1616, March 2020.

[7] Trusted Computing Group. *TPM Main Specification Level 2 Version 1.2*.

[8] A. Seshadri et al., "SWATT: Software-based attestation for embedded devices." *In Proc. IEEE Symp. on Security and Privacy,* 2004.

[9] A. Seshadri et. al., "SCUBA: Secure Code Update By Attestation in sensor networks," *In Proc. WiSe'06*, pp. 85-94.

[10] A. Seshadri, M. Luk, and A. Perrig, "SAKE: Software Attestation for Key Establishment in Sensor Networks," *In Proc. International Conference on Distributed Computing in Sensor Systems,* pp. 372-385, 2008.

[11] C. Castelluccia, et. al, "On the difficulty of software-based attestation of embedded devices," in *Proc. ACM conference on Computer and Communications Security (CCS)*, 2009.

[12] M. Jakobsson and K. A. Johansson, "Retroactive detection of malware with applications to mobile platforms," in *ACM HotSec 10*, 2010.

[13] M. Jakobsson and A. Juels, "Server-side detection of malware infection," in *New Security Paradigms Workshop (NSPW)*, 2009.

[14] Y. Yang et. al, "Distributed softwarebased attestation for node compromise detection in sensor networks," in *Proc. IEEE International Symposium on Reliable Distributed Systems*, pp. 219–230, Washington, DC, USA, 2007.

[15] B. Chen et. al, "Secure and Efficient Software-based Attestation for Industrial Control Devices with ARM Processors," in *Proc. ACM Annual Computer Security Applications Conference*, pp. 425-436, orlando, FL, USA, 2017.

[16] C. Krauss et al., "Detecting node compromise in hybrid wireless sensor networks using attestation techniques," in *Proc. ESAS*, Berlin, Heidelberg, 2007, pp. 203-217.

[17] S. Agarwal et al., "Program integrity verification for detecting node capture attack in wireless sensor networks," in S. Jojodia and C. Manumdar, editors, *Information Systems Security*, vol. 9478 of LNCS, pp. 419-440. Springer International Publishing, 2015.

[18] K. Eldefrawy, "SMART: Secure and minimal architecture for (establishing dynamic) root of trust," in *Network and Distributed System Security Symposium (NDSS)*, 2012.

[19] P. Koeberl, "TrustLite: A security architecture for tiny embedded devices," in *ACM European Conference on Computer Systems (EuroSys)*, 2014.

[20] F. Brasser, "TyTAN: tiny trust anchor for tiny devices," in *ACM/IEEE Design Automation Conference (DAC)*, 2016.

[21] J. Kong et. al, "PUFatt: Embedded platform attestation based on novel processor-based PUFs," in *Proc. ACM/EDAC/IEEE Design Automation Conference (DAC)*, San Francisco, CA, 2014, pp. 1-6.

[22] M. N. Aman, B. Sikdar, "ATT-Auth: A Hybrid Protocol for Industrial IoT Attestation With Authentication," in *IEEE Internet of Things Journal*, vol. 5, no. 6, pp. 5119-5131, Dec. 2018.

[23] W. Feng et. al, "AAoT: Lightweight attestation and authentication of low-resource things in IoT and CPS," in *Computer Networks*, vol. 134, pp. 167-182, 2018.

[24] X. Carpent, "Remote attestation of IoT devices via SMARM: Shuffled measurement against roving malware," in *IEEE International Symposium on Hardware Oriented Security and Trust (HOST),* Washington, DC, 2018, pp. 9-16.

[25] M. N. Aman et al., "HAtt: Hybrid Remote Attestation for the Internet of Things With High Availability," in *IEEE Internet of Things Journal*, vol. 7, no. 8, pp. 7220-7233, Aug. 2020.

[26] H. Tan et al., "A TPM-enabled remote attestation protocol (TRAP) in wireless sensor networks," in *Proc. ACM PM2HW2N*, New York, NY, 2011, pp. 9-16.

[27] A. Visintin et al., "SAFE$^d$: Self-attestation for networks of heterogeneous embedded devices," preprint, available: https://arxiv.org/abs/1909.08168.

[28] W. Yan et al., "EAPA: Efficient Attestation Resilient to Physical Attacks for IoT Devices," in *Proc. ACM IoT S&P*, 2019, New York, NY, USA, 2–7.

[29] M. N. Aman et al., "PRoM: Passive Remote Attestation Against Roving Malware in Multicore IoT Devices," in *IEEE Systems Journal*, Early access, 2021.

[30] C. J. Peng, K. L. Lee and G. M. Ingersoll, " An Introduction to Logistic Regression Analysis and Reporting," in *The Journal of Educational Research,*, vol. 96, no. 1, pp. 3-14, 2002.

[31] V. Jakkula, "Tutorial on support vector machine (svm)," in *School of EECS, Washington State University*, vol. 37, 2006.

[32] K. M. Leung, "Naive bayesian classifier," Polytechnic University Department of Computer Science/Finance and Risk Engineering, 2007.

[33] L. Nataraj, S. Karthikeyan, G. Jacob, and B. S. Manjunath. "Malware images: visualization and automatic classification," In Proc. *ACM VizSec* New York, NY, USA, 2011, pp. 1–7.

[34] V. Lakshmanan, M. Gorner, and R. Gillard, "Practical Machine Learning for Computer Vision," 1st Ed., O'Reily Media Inc., Sebastopol, CA USA, 2021.

[35] A. Zola, "IoT Platforms Overview: Arduino, Raspberry Pi, Intel Galileo And Others," [Online] Available: https://intersog.com/blog/iot-platforms-overview-arduino-raspberry-pi-intel-galileo-and-others/, Accessed: 10 Feb. 2022.

[36] A. Velosa, "Critical Capabilities for Industrial IoT Platforms," Gartner, 2021.

[37] Viet Hung Nguyen and Le Minh Sang Tran, "Predicting vulnerable software components with dependency graphs," In Proc. *ACM MetriSec*, New York, NY, USA, 2010, pp. 1–8.

[38] C. Nadeau and Y. Bengio, "Inference for the Generalization Error," in *Machine Learning*, vol. 52, pp. 239–281, 2003.

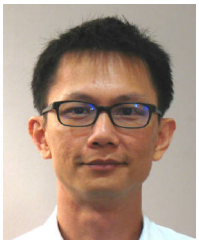

**Muhammad Naveed Aman** received the B.Sc. degree in Computer Systems Engineering from KPK UET, Peshawar, Pakistan, M.Sc. degree in Computer Engineering from the Center for Advanced Studies in Engineering, Islamabad, Pakistan, M.Engg. degree in Industrial and Management Engineering and Ph.D. in Electrical Engineering from the Rensselaer Polytechnic Institute, Troy, NY, USA in 2006, 2008, and 2012 respectively.

He is currently working as an Assistant Professor with the Department of Computer Science and Engineering at the University of Nebraska-Lincoln, USA. Dr. Aman previously served as a Senior Research Fellow with the Department of Computer Science at the National University of Singapore, Singapore and as an Assistant Professor with Department of Electrical Engineering at the National University of Computer and Emerging Sciences, Pakistan. His research interests include IoT and network security, wireless and mobile networks, and secure embedded systems.

**Haroon Basheer** (S'18) is a Research Assistant with NUS-Singtel Cybersecurity Research & Development Laboratory since Nov 2017. He received his Bachelor of Technology in Electronic Engineering from National University of Singapore, where he is also pursuing his Master of Computing degree in Computer Science.

**Jun Wen Wong** has more than 10 years of work experience in the cybersecurity field. He is currently a Senior R&D Engineer and one of the Co-Principal Investigators at Singtel Cybersecurity R&D in Trustwave, a Singtel Company. His areas of research include Trusted Computing, Internet of Things and Network Security. Prior to joining Singtel, Jun Wen was a Senior Research Engineer at the Institute for Infocomm Research, Agency for Science, Technology and Research (A*STAR), Singapore. He was involved in the European Union and Singapore Government funded projects relating to critical infrastructures such as sensor network, energy and rail transport.

**Jia Xu** is a Cyber Security R&D Manager specializing in Cloud and Data Security and Post Quantum Security in NUS-SingTel Cyber Security R&D Lab. He received PhD in computer science from the National University of Singapore in 2012. He worked as a research scientist in the Institute for Infocomm Research from 2012 to 2017. He's interested in applied cryptography and cloud computing security.



**Hoon Wei Lim** received the Ph.D. degree in Information Security from Royal Holloway, University of London. He held research positions at the Institute for Infocomm Research, Singapore, the National University of Singapore, Nanyang Technological University, and SAP, France. He is currently an Associate Director with the NUS-Singtel Cyber Security Research and Development Lab. His recent research interests have been centered around data security and privacy, and security intelligence and analytics within enterprise environments and cyber-physical systems.

**Biplab Sikdar** 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 an Associate 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. Dr. Sikdar is a member of Eta Kappa Nu and Tau Beta Pi. He served as an Associate Editor for the *IEEE Transactions on Communications* from 2007 to 2012. He currently serves as an Associate Editor for the *IEEE Transactions on Mobile Computing*.