IoT-ID: A novel device-specific identifier based on unique hardware fingerprints

Girish Vaidya*, Akshay Nambi†, T. V. Prabhakar*, Vasanth Kumar T*, Suhas Sudhakara*
* Indian Institute of Science, Bangalore, India
† Microsoft Research India
Email: {vaidyab, tvprabs}@iisc.ac.in, Akshay.Nambi@microsoft.com, {vasanthkumar9242, sudhakarasuhas}@gmail.com

***Abstract*—A significant number of IoT devices are being deployed in the wild, mostly in remote locations and in untrusted conditions. This could include monitoring an electronic perimeter fence or a critical infrastructure such as telecom and power grids. Such applications rely on the fidelity of data reported from the IoT devices, and hence it is imperative to identify the trustworthiness of the remote device before taking decisions. Existing approaches use a secret key usually stored in volatile or non-volatile memory for creating an encrypted digital signature. However, these techniques are vulnerable to malicious attacks and have significant computation and energy overhead. This paper presents a novel device-specific identifier, `IoT-ID` that captures the device characteristics and can be used towards device identification. `IoT-ID` is based on physically unclonable functions (PUFs), that exploit variations in the manufacturing process to derive a unique fingerprint for integrated circuits. In this work, we design novel PUFs for Commercially Off the Shelf (COTS) components such as clock oscillators and ADC, to derive `IoT-ID` for a device. Hitherto, system component PUFs are invasive and rely on additional dedicated hardware circuitry to create a unique fingerprint. A highlight of our PUFs is doing away with special hardware. `IoT-ID` is non-invasive and can be invoked using simple software APIs running on COTS components. `IoT-ID` has the following key properties *viz.*, constructability, real-time, uniqueness, and reproducibility, making them robust device-specific identifiers.**

We present detailed experimental results from our live deployment of 50 IoT devices running over a month. Our edge machine learning algorithm has 100% accuracy in uniquely identifying the 50 devices in our deployment and can run locally on the resource-constrained IoT device. We show the scalability of `IoT-ID` with the help of numerical analysis on 1000s of IoT devices.

***Index Terms*—Device ID, Fingerprinting, PUFs, Edge ML**

I. Introduction

With the advent of Internet of Things (IoT), billions of tiny connected devices are being deployed in various domains spanning home applications, city-wide monitoring, healthcare and military applications [1]. These IoT devices are generally deployed in-the-wild, mostly in remote locations and untrusted conditions [2]. Due to these characteristics, they are potentially exposed to a number of malicious physical attacks. For example, one could replace a device with another cloned device or replace a sensor, say temperature sensor with another one or with a moisture sensor, all leading to a potentially catastrophic event [3].

One of the key challenges hindering successful large-scale IoT deployments is device dependability [4]. Device dependability aims to identify the trustworthiness and reliability of the remote IoT device, for example, how to determine if the remote IoT device transmitting the data is the same one as deployed? Device dependability is an essential property of IoT devices especially in safety-critical applications such as intrusion detection, and infrastructure monitoring such as telecom and power grids, where decisions are taken solely based on the trust of the IoT device and the data it is transmitting. In reality, they can easily be cloned or modified with counterfeit electronics or attacked all leading to false device identities.

Thus it is imperative to *identify the device* before trusting the sensed data for deriving insights and taking decisions. Device identification today relies on identifiers such as digital signatures or hash-based message authentication codes [5]. These techniques require a secret key, which is usually stored in volatile or non-volatile memory, and are vulnerable to attacks [6]. Furthermore, generating robust secret keys has significant energy and compute overhead, and suffer from key-recovery attacks, making them impractical [7].

Due to these challenges, physically unclonable functions (PUFs) have emerged as a promising alternative, which exploit manufacturing and fabrication process variations to generate a unique, device-specific key or an identifier [8] [9]. This identifier can then be used for device identification and authentication. Recent works have exploited PUF towards the identification of individual Integrated circuits (ICs) [10] [11], where a key is generated using the PUF for the corresponding system components such as radio module, SRAM, and ring oscillator (RO). This key/identifier is almost impossible to duplicate, even if the circuit is cloned, as the process variation in the manufacturing process ensures that the PUF cannot be exactly cloned.

Current PUF-based identification techniques have several inherent limitations and cannot be directly applied to IoT device identification. First, the majority of the works *require additional dedicated circuitry* to be embedded in the silicon during the fabrication of the ICs to create a PUF [12] [13]. For example, Microsoft recently developed a novel and secure IoT device called Azure Sphere [14], which has a hardware root of trust component and an additional silica to generate device-specific PUF. However, current IoT devices deployed rely mostly upon commercially available off-the-shelf components, where additional circuitry cannot be added due to intrusiveness and cost. Second, current *PUFs were developed for specific*

system components, which may not exist in a typical IoT device. Hence there is a need to generate PUFs for system components that are broadly available across numerous IoT devices and can be used to identify the complete device as opposed to individual components. Finally, *existing PUF-based solutions are invasive*, where the component must be rebooted (especially for SRAM PUFs [10]) before generating a PUF, thus altering the running state of the device, necessitating the design of non-invasive PUFs.

In this paper, we address the above challenges by developing a novel identifier, called the IoT-ID. Specifically, we focus on creating a device-specific identifier that is non-invasive and works on off-the-shelf components without any additional hardware. To this end, we leverage the inherent process variation in manufacturing of the component to derive a PUF. PUF acts as the basic building block and IoT-ID is constructed by combining features from PUFs of multiple IoT system components. The overall objective is to demonstrate the characteristics of IoT-ID and show that it can be used as an identifier for individual IoT devices and also for instances of the same device-type. Furthermore, IoT-ID can now be used as a primitive for generating secret key for device identification and authentication. In the remainder of the paper, we focus on how to design unique, non-invasive, robust PUFs that can be used as a device-specific identifier.

IoT-ID is extremely robust and secure, as it is practically impossible to replicate the same physical characteristics of a device. Furthermore, IoT-ID has the following key properties: *constructability, real-time, uniqueness and reproducibility*, that enables the generation of robust device-specific identifiers (see Section III). We have devised an edge machine learning (ML) algorithm to accurately identify a device using the IoT-ID. The ML model is trained on identifiers obtained from individual devices and this ML model can run on a resource-constrained IoT device such as Arduino [15] (Section. V).

We have evaluated the efficacy of IoT-ID in a live real-world deployment with 50 IoT devices for over a period of one month. The IoT devices in our deployment are from different manufacturers and are exposed to realistic environmental conditions with temperature variations between $15°C$ to $45°C$ and voltage variations between 2.7V-3.3V. While we show that operating temperature and voltage variations affect the PUF characteristics, our ML model trained on device-specific IoT-ID can still identify the IoT device instance with 100% accuracy (Section. VI-D). To the best of our knowledge, this is a first study to explore non-invasive device identification with 50 devices in real-world conditions.

Finally, as the number of devices increases in a deployment, the likelihood of two devices having similar IoT-ID also increases. This is due to the fact that IoT-ID is derived based on PUFs that rely on variations in the manufacturing process of system components, and inherently these variations are bounded. Today to derive a universal device-identifier one has to add additional hardware, as in the case of Microsoft Azure Sphere [14]. The objective of IoT-ID is to derive a robust device-specific identifier without additional hardware, which is distinguishable at least in a typical mid to large-scale IoT deployments with 100-1000s of IoT devices. To show the scalability of IoT-ID, we present numerical analysis for 100-1000s of IoT devices.

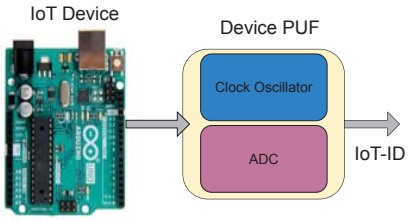


Fig. 1: Overview of device-specific IoT-ID creation.

Through this work, we make the following contributions:

- We introduce a novel device-specific identifier, called the IoT-ID that exploits physical variations in the manufacturing process to fingerprint IoT devices.
- We describe novel ways to generate physically unclonable functions (PUFs) for off-the-shelf components without any additional hardware, along with theoretical analysis and simulations on PUF generation.
- We present an edge ML algorithm that runs on a resource-constrained device to identify the device accurately using IoT-ID.
- We demonstrate the efficacy of the proposed techniques on 50 IoT devices in real-world conditions from our live deployment running over a month, along with numerical analysis to show the scalability of our technique in large-scale IoT deployments.

II. Background & System Overview

We present a brief overview of IoT devices, and then provide background on physically unclonable function (PUF).

A. IoT Device Description

The core components of a typical IoT device are (i) Microcontroller and its circuitry such as Clock oscillators, ADCs, GPIOs, Memory, (ii) Sensors, (iii) Communication module for collaboration between devices, and (iv) Power source.

A significant number of IoT devices are deployed in remote locations and untrusted conditions, consequently opening the doors to counterfeiting and malicious attacks. Therefore, it is important to uniquely identify the microcontroller, sensors attached to it and the communication module. Currently, there already exists fingerprinting techniques for sensor identification [16] and RF communication module identification [17]. The scope of this work is limited to overall IoT device identification without any additional hardware. We aim to uniquely identify commercially available off the shelf (COTS) device types and also instances of the same device type. For example, Arduino A1 and Nordic nRF52832 N1 should have unique device-specific IoT-ID and further, Arduino A1 and Arduino A2 should also have unique identifiers.

B. What is a PUF?

Physically Unclonable Functions (PUFs) give each integrated circuit (IC) a unique "fingerprint". They leverage inevitable variability in the physical device manufacturing process, for example, in transistors, the variations in length and width of transistors result in subtle differences in the behavior [8]. A device fabricated from the same process, same lot and same die will vary from its neighbors. This leads to variations in leakage current, threshold voltage, propagation delay, etc. As a result, one transistor will not be identical to another transistor. Recent works have exploited PUF towards the identification of ICs [11] [18], where a key is generated using the PUF for the corresponding system component such as radio module, SRAM, and ring oscillator (RO). This key is almost impossible to duplicate, even if the circuit is cloned, as the variation in the manufacturing process ensures that the PUF cannot be exactly cloned.

There are two major challenges in current PUF-based fingerprinting systems:

1. Need for additional hardware circuitry. Majority of the works introduce an additional dedicated circuitry in the silicon during the IC manufacturing process to create a PUF [12], [13], [18], [19]. For example, an RO-PUF [19] is designed with an additional ring oscillator circuitry to create a unique fingerprint. Here, each IC with this ring oscillator will have slightly different clock frequency due to manufacturing variations. These frequencies are compared to create PUF.

2. Invasive mechanisms to create a PUF. The startup values in an SRAM is an example of invasive PUF [10]. It has been observed that for a specific instance of SRAM, few memory cells in SRAM have the same values every time it is powered up. However, the location of these memory cells and the startup values differ from one instance to another. Thus, SRAM based PUF requires the device to be turned on/off to generate the PUF.

III. IoT-ID- Device-specific identifier

Existing PUF-based approaches fall short of creating a device-specific identifier for an IoT device. In this section, we discuss the IoT-ID generation based on novel non-invasive PUFs for two off-the-shelf device components viz. Clock Oscillator and Analog to Digital Converter (ADC).

A. IoT-ID *Overview*

A typical IoT node comprises of various system components such as clock oscillator, ADC, etc. The IoT-ID for a device is constructed by combining PUFs from clock oscillator and ADC as shown in Figure 1. Combining PUFs from multiple components enables IoT-ID to be robust and act as a device-specific identifier.

We now describe the key properties required to build a robust and scalable device-specific identifier:

(i) Constructability: An identifier exists that can be constructed by exploiting the variation in the physical manufacturing process of the system components.

(ii) Real-time: An identifier can be generated in real-time to identify the device in-the-wild.

(iii) Uniqueness: A unique identifier exists that can be constructed across all IoT devices in a deployment.

(iv) Reproducibility: Each time the identifier is generated for a given device, there is some variation due to operating conditions. We determine reproducibility by confirming if the variation of the identifier from the PUF of system components is within an intra-device threshold, and hence device is still uniquely identifiable.

We now present the details of the proposed novel PUFs for two device components *viz.,* clock oscillator and ADC. Typically these components are internal to the microcontroller. The device-ID generated using these components represents the microcontroller, which is the most important component of the system.

B. Clock oscillator PUF

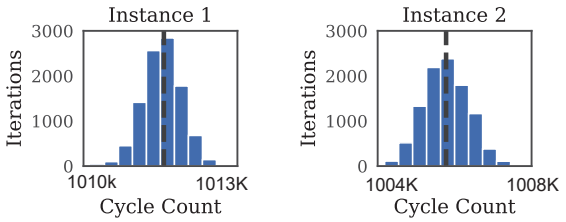
Every microcontroller needs a clock source as it determines the speed with which the microcontroller operates. Internal or external clock oscillators are used as the clock source for the microcontroller. Most of the present-day controllers such as Atmega2560 [20], CC3200 from Texas instruments [21], nRF52832 from Nordic [22] etc., used in IoT devices have at least two internal independent clock sources. Typically the high frequency clock is used for clocking the microcontroller and works in MHz range, whereas the low frequency clock is used for watchdog timer and works in kHz range.

Due to variations in the manufacturing process of clock oscillators, the number of clock cycles counted for a specific time period with one clock oscillator varies from another within a tolerance value. For example, consider two clocks $clock_1$ running at 32.768kHz and $clock_2$ running at 16MHz. $clock_1$ is used to generate an interrupt every 1 second and in this period, we count the number of clock cycles of $clock_2$. The expected number of clock cycles count of $clock_2$, in this case, should be 16000000, however, due to process variations in $clock_1$ and $clock_2$, this count varies for each clock oscillator instance. Unlike the external crystal clock which is accurate, typical internal clock oscillators might have tolerance values up to $\pm 10\%$ [20]. Thus leading to variations in clock count. For a certain time period 't', if M is the number of clock cycles counted by $clock_1$ and N is the number of clock cycles counted by $clock_2$, then,

$$M * clock_1 = N * clock_2 \Rightarrow \quad N = M * clock_1/clock_2 \tag{1}$$

The number of clock cycles counted, i.e., 'N' by the clock varies for each instance of clock oscillator, owing to the physical process variations. *We exploit this behavior to create a PUF for clock oscillator.* Specifically, we aggregate the clock cycle count values of $clock_2$ (with 16 MHz) for 2048 cycles of $clock_1$ (equivalent to 62.5ms duration with 32.768 kHz clock) to arrive at the total clock cycle count ($clock_{count}$), i.e.,

$$clock_{count} = \sum_{1}^{times} N \tag{2}$$



(a) Instance 1 -$clock_{count}$ with Mean=1011K, SD=404.37 (b) Instance 2 -$clock_{count}$ with Mean=1006K, SD=457.48

Fig. 2: Histogram of $clock_{count}$ for two identical instances.

where N is the number of clock cycles counted and *times* is the number of times the cycle count is aggregated. By aggregating the clock cycles we ensure the difference between $clock_{count}$ of one oscillator instance is different from another and also increases the robustness of the PUF.

Figure 2 shows the histogram of clock counts (from an experiment) for 10000 iterations across two identical IoT devices, I1 and I2 from the same manufacturer. The mean clock count for I1 is 1011.62K and for I2 is 1006.269K (as shown by the vertical dotted line). It can be seen that the mean cycle count for the two devices are quite distinct with a tight standard deviation. Thus $clock_{count}$ feature described in Equation 2 can be used as a clock oscillator PUF.

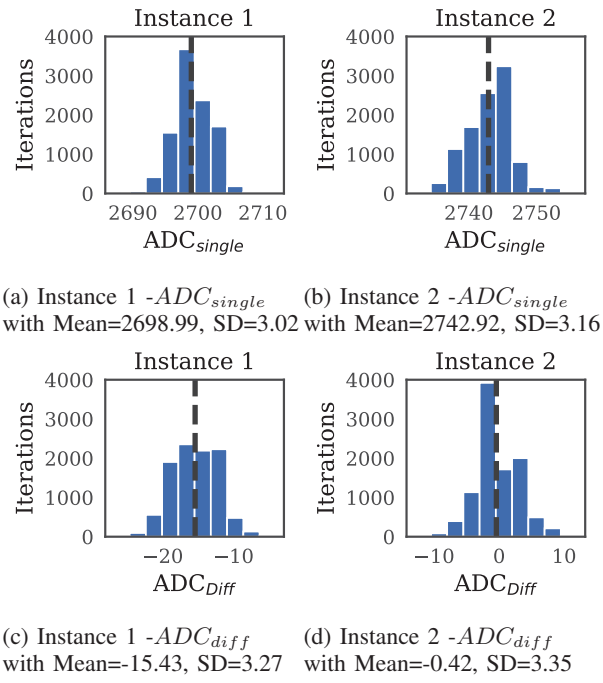
C. Analog-to-digital converter (ADC) PUF

Every microcontroller in an IoT device includes an internal analog-to-digital converter (ADC). ADCs generally support both single and differential modes, where the former measures the voltage difference between one pin and the ground, and the latter measures the voltage difference between two analog input pins [23]. The difference between the expected output and the actual output of an ADC for a given input voltage is defined as ADC error. This error occurs due to process and mismatch variations during the manufacturing of ADCs and includes gain errors, non-linearity errors, etc. *The key insight here is that, we can program the ADC pins in software so that we can provide a reference input and derive the ADC errors without any modification to the hardware. This makes it seamless to compute the errors even when the IoT device is connected to a sensor and is deployed.* We use this ADC error present in single and differential modes to create a PUF for an ADC.

1) ***Single mode output****:* In single ended mode, the ADC measures the voltage difference between the input voltage (V_{in}) and ground. The expected output (ADC_{single}) is defined as,

$$ADC_{single} = \sum_{1}^{times} \left\{ \frac{V_{in}}{V_{ref}} * 2^{res} \right\} \quad (3)$$

where, *times* is the number of times ADC_{single} value is accumulated per iteration, res indicates the supported ADC resolution, for example, 8, 10 or 12 bits. Typically, V_{ref} is connected to either internal or external reference, or to the device power supply in an IoT device. V_{in} is generally connected to an external sensor. In order to derive the ADC error, we set the register value so that V_{in} and V_{ref} are selected from internal voltage sources. For example, if V_{ref} is 3V and V_{in} is set to 2V via software, then the expected output (ADC_{single}) of a 12-bit ADC would be 2731. We accumulate the output value for 100 *times* and use it as a feature for an ADC PUF. This ensures the expected output value of a single mode ADC used for PUF is robust.



(a) Instance 1 -ADC_{single} with Mean=2698.99, SD=3.02 (b) Instance 2 -ADC_{single} with Mean=2742.92, SD=3.16

(c) Instance 1 -ADC_{diff} with Mean=-15.43, SD=3.27 (d) Instance 2 -ADC_{diff} with Mean=-0.42, SD=3.35

Fig. 3: Histograms of ADC values of single ended and differential modes for two identical instances.

Figure 3a and 3b show the histograms of ADC_{single} (from an experiment) over 10000 iterations for two device instances of ADC in single ended mode. The mean ADC_{single} value for instance 1 is 2699 and for instance 2 is 2743 (shown by the dotted vertical line). As shown in the figure, the standard deviations of the values is small relative to the difference between the mean values of the two instances. Thus enabling single mode output as a feature for ADC PUF.

2) ***Differential mode output****:* Here, ADC measures the voltage difference between two pins (V_{p1}&V_{p2}) and the expected output (ADC_{diff}) is defined as,

$$ADC_{diff} = \sum_{1}^{times} \left\{ \frac{V_{p1} - V_{p2}}{V_{ref}} * 2^{res-1} \right\} \quad (4)$$

To compute the offset error, V_{p1}&V_{p2} will be given the same voltage value and ideally the output value should be 0. However, due to process variations and mismatch in differential circuits [24], the ADC_{diff} value can be either positive or negative. We accumulate the value for 100 *times* and use it as a feature for ADC PUF. Again like in single mode, through software we can set the input voltage values of the two pins, i.e., V_{p1}&V_{p2} by writing to a register.

Figures 3c and 3d show the mean ADC_{diff} values of -15.42 for instance 1 and -0.42 for instance 2 (from an experiment)

in differential mode. Thus, combination of both ADC_{single} and ADC_{diff} accumulated over 100 *times* acts as features to create a PUF for an ADC.

D. IoT-ID *creation*

Since the objective is to derive a device-specific identifier, we combine features from clock and ADC PUFs, *viz.,* $clock_{count}$, ADC_{single}, and ADC_{diff}. Thus, IoT-ID is represented as,

$$\texttt{IoT} - \texttt{ID} =< clock_{count}, ADC_{single}, ADC_{diff} > \quad (5)$$

In Section VI we show the efficacy of IoT-ID as device-specific identifiers for instances from both, same manufacturer and different manufacturers. Further, we present a detailed evaluation of the proposed PUFs with both realistic temperature and voltage variations from our live deployment.

IV. Understanding the Physics of Process Variations

We now describe, why process variations exist during the manufacturing process? and present circuit simulations of a clock oscillator to show process variations across devices.

A. Why process variations exist?

During circuit manufacturing, process variation will be introduced in various fabrication steps such as ion implantation, thermal processes, leading to variations in physical parameters of semiconductor devices like length, width, oxide thickness, parasitic resistances and capacitances. In nanometer technology, it is practically impossible to have two identical circuits and hence, a circuit fabricated from the same process and same die will vary in its characteristics from its neighbors [8].

B. Circuit simulation overview

Figure 4 shows an overview of the circuit simulation flow. Semiconductor fabs characterise the process and develop model files to capture the process parameters (such as threshold voltage, oxide thickness, etc.) and their variations. These model files are specific to a particular technology and are part of the Process Design Kit [25]. A circuit designer designs a circuit in IC design flow and uses a simulator (such as spectre [26] and HSPICE [27]) to analyse the characteristics of the circuit. The circuit simulator accurately simulates the behavior of a circuit using the following three inputs: (a) Netlist of the circuit: This consists of a list of electronic components in a circuit and their interconnection. (b) Model files: This consists of the characterisation data of all the device components available in the process. (c) Simulation settings: Numerous parameters that can be set based on the simulation objective, such as total run time for transient analysis, temperature and voltage variations.

The circuit designer performs Monte Carlo simulations [28] to analyse the impact of process and mismatch variations. Each trial from the Monte Carlo simulation takes a value from the model parameter distribution and runs the simulation. The outputs from the simulations show the variations in the characteristics of a circuit. These can be used to determine if

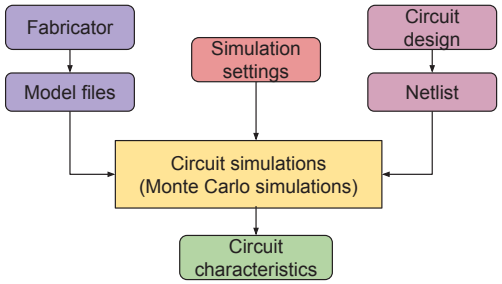
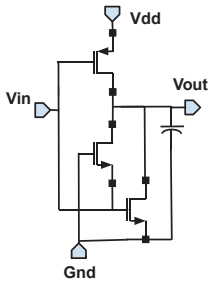
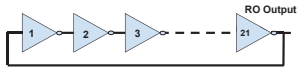


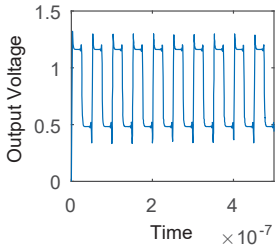
Fig. 4: Overview of Circuit simulation flow.



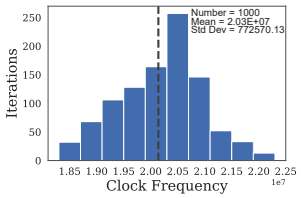
(a) Single-stage inverter.



(b) 21 stage cascading for ring oscillator.



(c) Clock generated using RO.



(d) Monte Carlo simulations.

Fig. 5: Circuit simulation results.

the circuit passes the necessary specifications and to derive the tolerance levels, typically used to populate the datasheet.

C. Clock oscillator circuit simulation

We now present the usage of circuit simulations to understand the variations in clock oscillator. We use a ring-oscillator architecture for the clock that consists of an odd number of cascaded inverters [29]. Figures 5a and 5b show the single-stage inverter design and 21 such inverters cascaded to constitute a clock oscillator. This clock oscillator generates a 20 MHz square wave clock output as shown in Figure 5c. The circuit design is implemented based on a popular fabricator UMC [30] with 180 nm technology and simulated in industry standard spectre simulator [26].

We performed Monte Carlo simulations for 1000 devices using the model files obtained from the fabricator i.e., UMC [30]. Figure 5d shows the clock frequency distribution across 1000 devices from circuit simulation of the clock oscillator. The variation in frequency is around $\pm 10\%$ of the nominal value for $\pm 3\sigma$ variation which is comparable to oscillators commercially available [20]. The simulation results demonstrate the variation in clock frequency due to process variations.

While we show process variations using circuit simulation for only clock oscillator, the approach is generic and could

be applied to any other circuit by using its design and corresponding model files.

D. Modeling process variations to derive an identifier

We now demonstrate how process variations can be leveraged to derive an identifier. As described in the previous section each semiconductor component has an inherent process variation that can be modeled as a Normal distribution (also known as Gaussian distribution) [31]. We have validated the feature is distributed normally based on the model files from the semiconductor fabs and also technical datasheets [32] [33]. For example, as shown in Section IV-C the clock frequency has near-normal distribution.

Consider two devices whose feature values are given by random variables X_1 and X_2. We assume that X_1 and X_2 are identical and independent random variables with distribution $N(\mu^*, \sigma^*)$, where μ^* and σ^* is the mean and standard deviation respectively for the normal distribution. The random variable $Z_{12} = |X_2 - X_1|$ gives the probability distribution of the feature difference between the 2 devices. The distribution for Z_{12} is given by the probability density function,

$$f_{Z_{12}}(x) = \frac{1}{\sigma}\sqrt{\frac{2}{\pi}} exp\left\{\frac{-x^2}{2\sigma^2}\right\}, \tag{6}$$

where $\sigma = \sqrt{2}\sigma^*$ [34]. Let d be the minimum acceptable difference between the two devices i.e. if the difference between the features of the two devices is less than d, then we consider that the two devices are conflicting and can not be distinctly identified. We can now define the cumulative distribution function (CDF) as,

$$F_{Z_{12}}(d) = \int_0^d f_{Z_{12}}(x)dx, \tag{7}$$

which describes the probability of conflict between two devices. $\overline{F_{Z_{12}}(d)} = 1 - F_{Z_{12}}(d)$ represents the probability that the two devices could be identified uniquely. For N devices, we consider a device to be unique if it does not conflict with any of the remaining $(N-1)$ devices. Hence, the probability that the device is unique amongst N devices is given by the equation,

$$P_{NC} = \overline{F_{Z_{12}}(d)}^{(N-1)}, \tag{8}$$

and the number of unique devices can be represented as,

$$U = P_{NC} * N, \tag{9}$$

Equation (8) assumes independence amongst the CDFs of difference in features (i.e. $F_{Z_{12}}(d)$ is independent of $F_{Z_{ij}}(d)$ for all $Z_{ij} \neq Z_{12}$ where $Z_{ij} = |X_j - X_i|$). Hence the equation (9) sets the lower bound for the number of unique devices. U is a function of N, d, and σ^*, assuming a single feature. If there are p features considered (in our case 3, i.e., $clock_{count}$, ADC_{single} and ADC_{diff}), each with $P_{NC} = P_{NCi}$, then U can be re-written as,

$$U = (1 - \prod_{i=1}^{p}(1 - P_{NCi})) * N. \tag{10}$$

Equation 10 can now be used to determine the unique non-conflicting identifiers derived based on p features. For example, if N=100, d=0.1, p=1 and σ^*=100, number of unique identifiers obtained are U=94.57%. U deteriorates to 56.91% when N increases to 1000. Furthermore, by considering p=3 features, U improves to 99.98% for N=100 and 91.99% for N=1000.

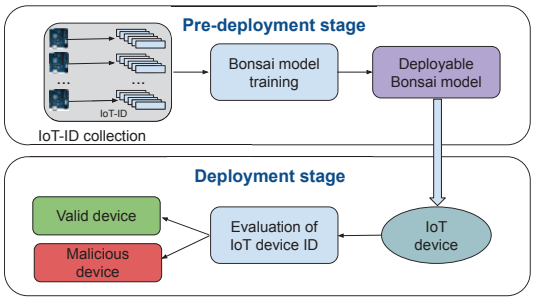


Fig. 6: Edge machine learning algorithm pipeline.

Using the proposed probabilistic models, we can also derive the lower bound on unique identifiers for mid to large scale deployments. For example, if the N devices are sampled repeatedly m times from a larger pool of devices, the probability of having unique identifiers through all the m trials can be written as,

$$P_{NCm} = (P_{NC})^m \tag{11}$$

Thus, for N=100 and p=3, if the 100 devices are sampled 1000 times from a larger pool of 10000 devices, with a P_{NC} of 0.9998, then the probability of having at least 100 unique devices is P_{NCm} is 0.8187. The parameters, d and σ^* could be controlled to further improve P_{NC}, P_{NCm}, and U by changing the accumulation count. We provide more details on the usage of this model in our scalability analysis described in Section VII.

V. Device identification with IoT-ID

We now present an edge machine learning (ML) model that is trained on IoT-ID for all devices, which is then used to identify the device instance accurately.

In Section IV-D we presented a probabilistic approach to derive unique identifiers. However, such an approach requires users to define the minimum separating distance "d". In order to generalize the usage of IoT-ID across various devices, we build an edge ML model to accurately distinguish between different identifiers.

Our proposed edge algorithm pipeline has two stages as shown in Figure 6: (i) pre-deployment stage, wherein the features of IoT-ID for each instance is collected and stored to build a model, and (ii) deployment stage, wherein the device generates a new IoT-ID which is then compared with the trained model to identify the device.

A. Pre-deployment stage

In this stage, we collect the IoT-ID for all IoT devices used in a deployment. We then build a classifier model to uniquely identify the device. We now describe the steps involved:

1) When installing the IoT device for the first time we record the IoT-ID's for the device for each iteration, this will

result in the following tuple:
$< Clock_{count}, ADC_{single}, ADC_{diff}, Device_{name} >$.

2) On each device instance we collect multiple IoT−ID's, say typically 100-1000.
3) We build a classifier model using the above data from all IoT device instances used in the deployment.
4) This classifier model is now embedded into the IoT device before deployment.

Note that the usage of IoT−ID is independent of the ML algorithm. In this paper, we use a state-of-the-art classifier model called Bonsai, which is a decision tree based classifier. However, other ML algorithms such as kNN algorithms can also be used for device identification.

B. Bonsai model for IoT device identification

Bonsai [35] is a state-of-the-art classification model that can run on resource-constrained devices. It is a novel tree based algorithm, which maintains high prediction accuracy while minimizing model size by performing numerous optimizations such as: (i) Developing a tree model which learns a single, shallow, sparse tree with powerful nodes, (ii) Sparsely projecting all data into a low-dimensional space in which the tree is learnt, and (iii) Jointly learning all tree and projection parameters.

We train the Bonsai model using the IoT−ID's collected from all IoT devices in the pre-deployment stage. The training requires the following input parameters, *viz.,* number of features, number of devices, number of train and test samples, projection dimension, and depth of the Bonsai tree. In our case, we have 3 features ($Clock_{count}$, ADC_{single}, ADC_{diff}), and number of devices corresponding to the number of IoT devices used in the deployment. The projection dimension specifies the dimension to which the input training data needs to be projected, the lower the projection dimension the sparser the data set and depth parameter indicates the depth of the tree, the larger the depth the higher the computation required. The trained model is then deployed on all IoT devices in the deployment. Further, the trained model is optimized to run on resource-constrained devices such as Arduino [15] with 2 KB RAM and 32 KB read-only flash.

C. Deployment stage

In the deployment stage, all the IoT devices are deployed (mostly remotely) according to the application requirements. IoT device identification can now follow the following two scenarios: (i) the IoT device automatically invokes the device identification before transmitting any data to a controller or before making any decision on the system, and/or (ii) the controller or aggregator before taking an action based on the device data can invoke the identification of remote IoT device. In either case, when the device receives an identification request, an IoT−ID for the device is then generated and compared to determine its identity.

Specifically, the device in question generates an IoT−ID using the clock and ADC PUF. This is then evaluated against the classifier model locally on the device. The classification model will output the device name (one of the devices deployed or labels it as an unknown/malicious device) along with a confidence metric. While we enable our model to run locally on resource-constrained devices, depending on the application requirement the device can send the IoT−ID directly to the cloud, where the device identification can be evaluated. Finally, necessary steps need to be taken to ensure upon invocation of IoT−ID generation, the software routine is actually executed on the device as we describe in Section VIII.

In Section VI-D we describe the model size, train and test data, time required to perform classification on Arduino and classification accuracy towards IoT device identification.

Fig. 7: Fuselage used for deployment of devices.

Fig. 8: Different device types used for evaluation.

VI. Experimental Evaluation and Results

We evaluate IoT−ID on a real-world live IoT deployment running over a month. In this section, we first present our real-world deployment setup along with the characteristics of the IoT devices used. We then discuss the efficacy of IoT−ID with respect to the four key properties as described earlier.

A. Deployment setup

We have 50 IoT devices deployed in a large model fuselage of an aircraft (see Figure 7). These devices are monitoring and sensing the ambient temperature every minute at various locations in the fuselage. The sensed data is then used to model the temperature variations inside a fuselage. Note that, there is no automated climate control setup in the fuselage, thus the devices are exposed to natural temperature and voltage variations. In our live deployment running over a month, we have observed the ambient temperature variations between $16°C$ to $45°C$ with an average of $27.5°C$. Further, the devices are powered using AA batteries and hence the supply voltage to the devices also varied over time.

We have considered three types of microcontroller (see Figure 8) in our deployment of 50 IoT devices *viz.,*

1) *Device type 1 (D1):* 10 off-the-shelf Arduino Mega IoT device with ATmega2560 microcontroller [20] [15].

TABLE I: Procedures for extraction of different features

Feature	Procedure for extraction
$Clock_{count}$	(i) Configure Real time clock (RTC) to generate interrupt after every interval T_{int}. For each device type, T_{int} could be configured through register settings. (ii) Initialize the high-frequency counter. (iii) Read the count of high-frequency counter at every interrupt. Reinitialize the counter to 0.
ADC_{single}	(i) Initialize the ADC in single ended mode. (ii) Apply a fixed voltage V_{in}. This fixed voltage V_{in} is one of the available internal voltages within each device type and routed to ADC core input through register settings. (iii) For each device type, the reference is selected from available input references through register settings. (iv) Perform 100 conversions and accumulate the output.
ADC_{diff}	(i) Initialize the ADC in differential mode. (ii) Connect the differential input to a common voltage V_{in}. This voltage V_{in} is one of the available internal voltages within each device type and routed to ADC core input through register settings. (iii) For each device type, the reference is selected from available input references through register settings. (iv) Perform 100 conversions and accumulate the output.

2) *Device type 2 (D2):* 38 custom IoT device developed by us with nRF52832 microcontroller [22].
3) *Device type 3 (D3):* 2 off-the-shelf programmable system-on-chip (PSoC) CY8CKIT-062-BLE devices with Cortex M4 microcontroller [36].

We selected the above devices in our deployment as these are some of the most common IoT devices used in mid to large scale IoT deployments across verticals. Furthermore, our objective is to not just identify the class of IoT device (e.g., Arduino or nRF) but also to identify devices at an instance level (e.g., Arduino A1 and Arduino A2). Note that, our approach is generalizable to any IoT device that supports the three features used by IoT-ID.

Data collection:
The 50 IoT devices in our deployment are sensing the ambient temperature every minute. Each device also generates an IoT-ID every time it senses the environment, by measuring the clock and ADC PUF. This IoT-ID is then used to determine the identity of the device before acting upon the sensed data. In our live deployment, we have collected over 500K IoT-ID's across the 50 devices. As mentioned earlier the IoT devices are exposed to variations in both operating temperature and voltage.

Software API for PUF feature extraction:
One key novelty of this work is the design of non-invasive PUF features without the need for additional hardware.

Table I shows the procedure for the extraction of $Clock_{count}$, ADC_{single} and ADC_{diff} signatures for various device families. It is clear from the table, the APIs are hardware agnostic, generic and require just minimal configurations in the device registers. The application developer can invoke these APIs to extract PUF features, thereby generating IoT-ID for the device.

For example, to derive ADC PUF features in device type D1, the registers ADMUX and ADCSRB are programmed in software for selecting the reference and setting the ADC in single ended or differential mode [22]. Through this programming the ADC error could be computed without any hardware modifications.

TABLE II: Time req. and current consumption for IoT-ID.

Device	Time required (sec)	Current (mA)
D1	1	65
D2	1	2.2
D3	1	5

B. Constructability and reproducibility of IoT-ID

We now present results to show the efficacy of IoT-ID with respect to constructability and reproducibility properties, across 50 IoT devices.

*1) **Clock oscillator PUF**:* Figure 9a and 9b shows the clock count feature from clock oscillator PUF for 10 instances of IoT device type D1 and D2, respectively. The x-axis represents the time in days and the y-axis represents the daily average of the $clock_{count}$. Each line in the plot represents clock PUF for an IoT device. We make two key observations for the 10 instances of device 1 and device 2: (i) Due to variations in the manufacturing process of clock oscillators, each device has a distinct clock count. Thus supporting the *constructability* property of IoT-ID. (ii) For each individual IoT device, the clock count feature does not vary significantly over time. Thus supporting the *reproducibility* property of IoT-ID.

*2) **ADC PUF**:* ADC PUF uses two features *viz.,* ADC_{single} and ADC_{diff}, as described in Section III-C.

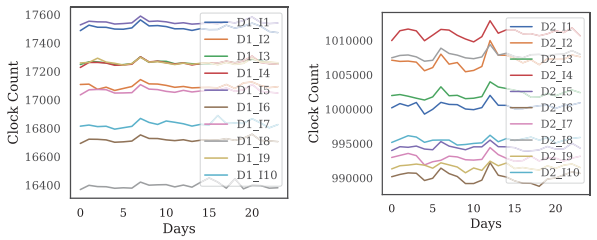
Figures 9c and 9d show the accumulated ADC output values for single mode across 10 instances of devices D1 and D2, respectively. The values are plotted over the duration of the deployment. Each line on the plot represents an IoT device with x-axis and y-axis corresponding to the time in days and daily average of ADC_{single} feature value, respectively. It can be seen that the ADC_{single} value is distinct across devices, supporting the constructability property. Further, the ADC_{single} value is stable with negligible variation over the duration of a month, supporting the reproducibility property of IoT-ID.

Figures 9e and 9f show similar characteristics for ADC_{diff} feature. Thus supporting both constructability and reproducibility properties of IoT-ID.

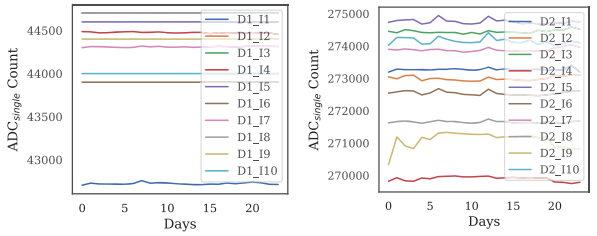
C. Time and current consumption for IoT-ID *generation*

Time required to generate IoT-ID: Table. II shows the time required to extract both clock and ADC features across devices D1-D3. In general, the clock oscillator PUF for all the devices is computed within 100 ms (for 100 accumulation count) and ADC features are computed within a second (for 100 accumulation). Overall, in one second IoT-ID can be generated across the three IoT device types. *Thus, the time required for IoT-ID generation supports the real-time property.*

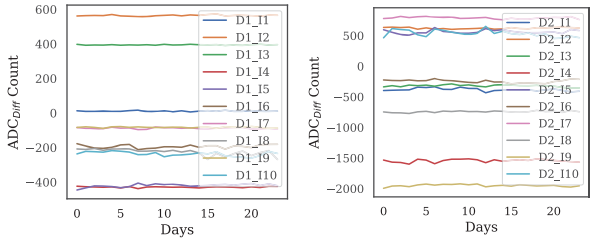
Current consumption: Table II shows the current consumption overhead for computing features of both clock and ADC PUFs in devices D1-D3. The current consumption is around 65mA, 2.2mA, 5mA, for the three device types D1, D2 and D3, respectively. Note that, the current consumption during feature extraction is in a similar range as that of the corresponding active state current of the device. For example,



(a) $clock_{count}$ - 10 D1 devices (b) $clock_{count}$ - 10 D2 devices



(c) ADC_{single} - 10 D1 devices (d) ADC_{single} - 10 D2 devices



(e) ADC_{diff} - 10 D1 devices (f) ADC_{diff} - 10 D2 devices

Fig. 9: PUF variations across 10 instances of D1 and D2.

D1 device with Arduino Mega consumes 65mA in active state and IoT-ID ID generation has a very minimal overhead on top of 65mA. *Thus we generate* IoT-ID *ID with negligible current and compute overhead.*

D. Uniqueness of IoT-ID *across IoT devices*

We now evaluate the robustness of IoT-ID towards accurately identifying the IoT device instance from our 50 device deployment. We use the Bonsai classification model described in Section V-B to determine the identity of the device.

1) ***Bonsai model accuracy*** *:* In our deployment of 50 nodes running over a month, we have collected around 500K IoT-ID's. Specifically, for each device, we collected around 100-1000 IoT-ID identifiers. The collected data is split into two sets; training dataset and test dataset. To show the effectiveness of IoT-ID we use various train and test splits ranging from 50% (15 days of data) to 1% (less than a day of data) for training the model.

We use the following parameters to train the Bonsai model: number of features = 3, i.e., $Clock_{count}$, ADC_{single}, ADC_{diff}, number of devices = 50, projection dimension = 2, i.e., projecting the dataset to two dimensional space and tree depth = 2, creating a Bonsai tree with just depth 2.

Table III shows the accuracy of device identification using IoT-ID with various train and test splits. We can see that

TABLE III: Device identification accuracy on 50 devices.

Train split	Test split	Clock count	ADC (single+diff)	Combined (clock+ADC)
50%	50%	54.0%	87.2%	**100%**
20%	80%	53.3%	87.0%	**100%**
5%	95%	53.0%	86.5%	**100%**
1%	99%	52.2%	86.1%	**100%**

when 50% of the data, i.e., 15 days of data is used to train the ML model, the device identification accuracy among 50 devices is 54% and 87.2% when only clock PUF and ADC PUF is used, respectively. However, when both clock and ADC PUF features are used, i.e., combined, the identification accuracy for the 50 nodes goes to 100%. Furthermore, as we reduce the training to 1%, i.e., less than one day of data, the ML model is still able to accurately identify the 50 devices with 100% accuracy when the combined features are used, *Thus showing the effectiveness of* IoT-ID *towards device-specific identifier.*

We also evaluated the importance of individual features on the classification task after training the model with all 3 features (combined). The higher the weightage the more important the feature is. We observe that all the three features, *viz.,* $Clock_{count}, ADC_{single}, ADC_{diff}$ have weights around 0.3, 0.34, 0.36, respectively, indicating the trained model relies on all the three features to uniquely identify the IoT device.

2) ***Bonsai Model size****:* Bonsai classification model is just 6KB in size, which can be easily embedded into resource-constrained IoT device.

3) ***Time required for classification****:* We benchmarked our Bonsai model on IoT device D1 based on Arduino. The time taken to perform a classification on Arduino is 20.2 milliseconds (ms). Furthermore, the overall accuracy of Bonsai running locally on all the three device types D1-D3 is 100% when all the three features are used.

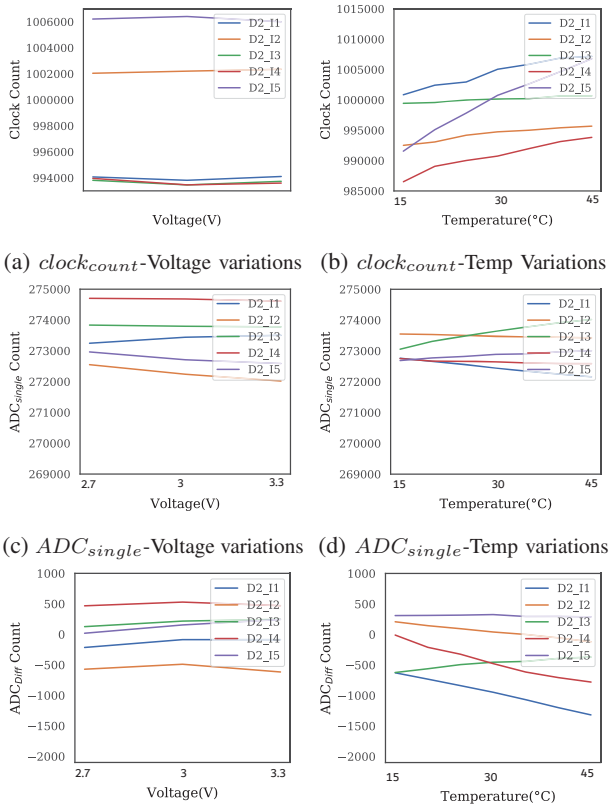
E. Impact of voltage/temperature variations on IoT-ID

Given IoT-ID is used as a device-specific identifier, reproducibility of IoT-ID under varying operating conditions such as variations in temperature and supply voltage is imperative.

To study the impact of supply voltage and temperature we performed both controlled experiments and also exposed IoT devices to real-world conditions in our deployment.

Controlled Experiments*:*

1) ***Impact of variation in supply voltage****:* We performed controlled experiments on 5 instances of IoT device type D2 by varying the supply voltage given to the device between 2.7V to 3.3V in a lab setting. At each supply voltage value, we extracted clock and ADC PUF features. Figure 10a, 10c and 10e shows the variations in $clock_{count}$, ADC_{single} and ADC_{diff} features for different supply voltage values, respectively. We can clearly see that as the supply voltage varies there are very minimal variations in all the three features. *Thus indicating the clock and ADC PUF features are agnostic to supply voltage variations.*



(a) $clock_{count}$-Voltage variations (b) $clock_{count}$-Temp Variations

(c) ADC_{single}-Voltage variations (d) ADC_{single}-Temp variations

(e) ADC_{diff}-Voltage variations (f) ADC_{diff}-Temp variations

Fig. 10: PUF variations across 5 device instances with respect to temperature and voltage.

*2) **Impact of variation in temperature**:* To study the impact of temperature variations on clock and ADC PUF, we used a temperature controllable thermal oven. We varied the temperature inside the oven from $15°C$ to $45°C$ in steps of $5°C$ and collected the three PUF features on the 5 devices. Figure 10b, 10d and 10f show the variations in $clock_{count}$, ADC_{single} and ADC_{diff} features at different operating temperature values. Unlike supply voltage, temperature variation induces changes in both clock and ADC features. As we increase the temperature from $15°C$ to $45°C$, we can see that the three features have a linear dependency on the temperature variation across 5 devices. Thus it is important to take into account temperature variations when devising device-specific identifier. Next, we present how to consider temperature variations during device identification using `IoT-ID`.

***Real-world Experiments**:*

From our controlled experiments it is clear that supply voltage variations do not affect PUF features. We will now describe how to take into account temperature variations and yet achieve 100% accuracy in device identification.

In Section VI-D we showed that Bonsai model was able to uniquely identify the IoT device from our 50 node deployment. While we had not explicitly considered operating temperature as a feature, the train and test data had similar/overlapping temperature variations and hence the model was able to distinguish the `IoT-ID`'s of different devices.

To validate this we performed an experiment, where we divided the `IoT-ID` data obtained from 50 devices into two sets with different temperature values. Thus, the training data includes `IoT-ID` with three features from all 50 devices when temperature values were $\leq 27°C$ and the test data includes `IoT-ID` with three features from all 50 devices when temperature values were $> 27°C$. We trained the Bonsai model with this train and test splits and the model accuracy towards device identification dropped to 90%. Thus validating our hypothesis that when the train and test data are collected over non-overlapping temperature ranges, the accuracy of our Bonsai ML model drops.

Thus to mitigate the effect of temperature variations on device identification, we propose two strategies, *viz.,*

1. Collect training and test data in similar operating conditions. In this case, there is no need for explicit temperature sensing as the train and test data are collected in similar operating conditions. Thus leading to unique identification as described above. However, when the operating conditions differ significantly from train environment the accuracy drops and one can re-train the model by including the new data.

2. Include temperature as an additional feature to train the model. All modern IoT device microcontrollers come with an in-built temperature sensor, which could be leveraged to sense the temperature and include that in the Bonsai model. The model can learn the mapping between the variations in temperature and PUF features. To verify this, we re-trained the Bonsai model with temperature as an additional feature, now the overall accuracy increased to 100% from 90%. Thus, using the above mentioned techniques, we ensure that the `IoT-ID` for the device remains unique across the operating temperature range of the device.

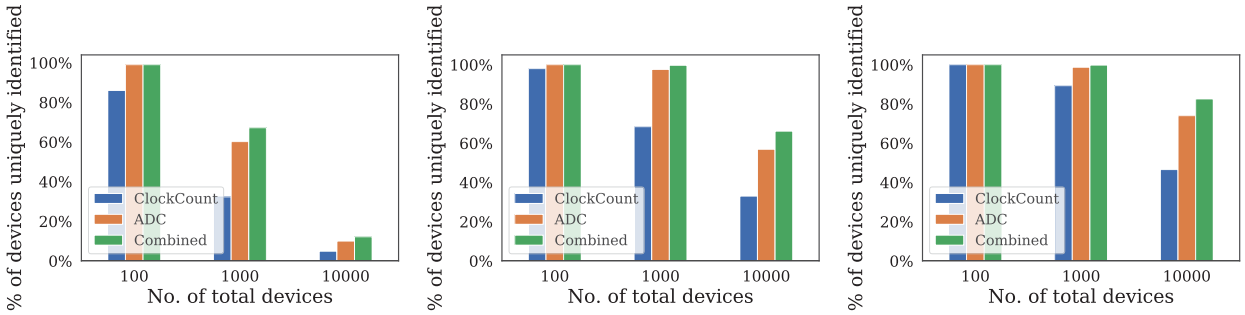
F. Impact of component aging on `IoT-ID`

The PUF features should not degrade significantly. If there is a significant deviation in PUF features this might indicate a fault in the system component. For example, if the clock count value varies significantly then the behavior of the clock oscillator has changed and may not be operating normally. Specifically, in our real-world deployment of 50 devices over a month, we did not observe any significant change in PUF behavior as shown in Figure 9.

Furthermore, due to the low-cost nature of these components, there will be some variation in PUF features over multiple years of operation. For example, a clock oscillator will have an aging of 5 ppm (i.e., $5X10^{-6}$) during the first year of operation and 3 ppm per year thereafter [37]. These variations are relatively small and are ignored for the first few years of operation. To ensure `IoT-ID` generated is robust, we propose that periodic re-training of the model should be carried out after a few years of deployment to take aging into account.

VII. Scalability Analysis of `IoT-ID`

Till now we showed that `IoT-ID` can be generated in real-time with minimal energy overhead, and is unique for



(a) Accumulation: clock & ADC for 1s. (b) Accumulation: clock & ADC for 100s. (c) Accumulation: clock & ADC for 300s.

Fig. 11: Scalability Analysis

a device even under varying operating conditions in our 50 node real-world deployment. However, it is also important to understand the bounds of IoT-ID when the number of devices increases in a deployment. As mentioned earlier, IoT-ID is not a universal IoT device identifier. Infact as the number of devices increases the likelihood of multiple devices having the same IoT-ID increases due to bounded process variations of system components. In this section, we systematically study the scalability of IoT-ID for 100-1000s of IoT devices.

The circuit designs for the semiconductor parts are intellectual property of the vendor and are not available for public usage. Given the non availability of circuit designs and corresponding models, performing monte carlo simulations for the circuit to analyse the scalability is not feasible. Further, given the limitation of deploying or working with such large-scale physical IoT devices, we look towards a numerical approach to analyze the scalability aspect. As described in this section, in a deployment with 10000 IoT devices, our approach can accurately identify 82.6% of the devices.

To conduct numerical analysis we first need to develop a realistic model of process variations for the three PUF features. To this end, we utilize the process distributions observed on our 50 node deployment over a period of one month. Specifically, we extract the mean, inter-device standard deviation and intra-device standard deviation values for the three features *viz.,* $Clock_{count}$, ADC_{single}, ADC_{diff}. The intra-device standard deviation is derived from the multiple measurements done on a single device over a period of time. Similarly, the inter-device standard deviation is derived from the variations in the mean values across the 50 devices over a period of time. We then run a MATLAB simulation to spawn multiple devices from the above distribution for each feature. This approach represents a spread of devices with parameters representing the 50 deployed devices. Further, like in our real-world deployment, we accumulate the raw values from clock for a period of 2048 cycles corresponding to 62.5 ms and ADC over 100 conversions corresponding to 1 second to derive the corresponding PUF features.

MATLAB model verification: We use the above simulation setup to spawn 50 devices and generate PUF features for each of them. In total for each device, we had 1000 IoT-ID's resulting in 50×1000 IoT-ID's. We trained the Bonsai ML model with 50% train and test data splits and the resulting device identification accuracy was 56%, 96% and 100% when only clock count, only ADC, and combined features were used, respectively. The device identification accuracy is very similar to the results obtained from real-world 50 node deployment as shown in Table. III.

Scenario-1: Accumulation of clock count for 62.5ms and ADC for 1s, total time per IoT-ID generation is 1s
This scenario corresponds to accumulation time periods used for generating IoT-ID in our real-world experiments. We now spawn 100, 1000 and 10000 devices using the above mentioned distributions and the accumulation count. Thus for each device, we generated 1000 IoT-ID's that can be used for training and testing purposes.

Figure 11a shows the accuracy of identifying a device uniquely for 100, 1000 and 10000 devices using only $Clock_{count}$, only ADC (single + diff) or through a combination of features. Similar to real-world experiments, the combined features, in general, give the highest accuracy towards device identification. When the device count is 100, the device identification accuracy using IoT-ID is still 100%. However, as the number of devices increases to 1000 and 10000, the device identification accuracy using IoT-ID drops to 67.2% and 12.2%, respectively. This is due to the negligible inter-device separation as we scale the number of devices resulting in overlapping IoT-ID's.

One obvious way to increase the overall identification accuracy is to increase the accumulation time period. The hypothesis here is that, as the accumulation is increased by N times, the mean of the accumulated feature is scaled by N and the standard deviation of the accumulated feature is scaled by $\sqrt{N}$ [38]. Hence the inter-device separation of the feature increases. This helps towards better distinction across the devices.

Scenario-2: 100x increase in accumulation of clock count and ADC, total time per IoT-ID generation is 100s
Here we increase the accumulation time for each feature by 100 $times$, e.g., 62.5ms of clock count accumulation takes now 6.25s and 1s of ADC accumulation takes 100s. Thus total time taken for generation of IoT-ID is 100s, consequently limiting the device identification once every 100s.

Figure 11b shows that as we increase the accumulation count by 100 $times$, the device identification accuracy for 1000 devices is increased to 99.7% and for 10000 devices is increased to 66.2%.

Scenario-3: 300x increase in accumulation of clock count and ADC, total time per IoT-ID generation is 300s
Here we increase the accumulation time for each feature by 300 $times$, thus resulting in IoT-ID generation once every 5 minutes. From Figure 11c it is clear that now 82.6% of the 10000 devices could be identified accurately within 5 minutes by increasing the accumulation across all features.

Thus as the number of devices in a deployment increases the identification accuracy using IoT-ID decreases. However, with an increase in accumulation time for deriving IoT-ID per device, we can have over 82% unique identification in a 10000 node deployment. This comes at the expense of increased time required for IoT-ID generation. An alternative approach is to include additional features towards the generation of IoT-ID. In our current scenario, we have used our three novel PUF features, but this can be extended by integrating features from other system components like sensors, power supply, communication modules, etc.

VIII. Security Analysis of IoT-ID

In the previous section, we have discussed the efficacy of our novel IoT-ID under different conditions. Our focus here is towards the generation of robust identifiers, which can then be used for various applications from device identification to authentication. Furthermore, based on the application requirement necessary precautions need to be taken for IoT-ID to be also secure. This is out of scope for this paper, however, we present some basic threat models and additional security measures that might be required for usage of IoT-ID.

*1) **Case 1- Incorrect or No IoT-ID generation in an installed device:*** In this case, we consider the following scenarios that might result in the generation of incorrect IoT-ID's.

- The execution of the classifier model is skipped and the classifier output is faked or tampered.
- The IoT-ID being communicated is not generated through the classifier model, but is faked or tampered.
- The classifier model is fed with incorrect inputs.
- The inputs to the classifier model are correct, but the classifier model is changed or tampered.

Each of the above scenarios would result in the generation of an incorrect IoT-ID. The above scenarios are possible only if the software running on the device is tampered or compromised. The software of the device could be secured through the following existing approaches:

- Many of the modern microcontrollers provide an option to disable the software read back. This ensures that logic for the IoT-ID generation is protected [22]. Further, one could prevent reprogramming of the microcontroller by connecting the pins used for programming to ground permanently in hardware.
- The software code for IoT-ID generation must be stored in the secure boot of the microcontroller or in the root of trust of the microcontroller. The software code stored in the secure boot memory of the microcontroller can not be modified and is executed during the boot up of the microcontroller.

*2) **Case 2- Subjecting the device to temperatures beyond the training range to generate wrong IoT-ID's:*** As discussed in Section VI-E, if the device is subjected to temperatures beyond the training range, the accuracy of IoT-ID degrades resulting in frequent incorrect IoT-ID generation. To prevent this, the training samples must be collected over the entire operating temperature range. The software must bypass IoT-ID generation and provide an alert if the measured temperature is beyond the operating temperature range.

*3) **Case 3- Device being replaced or tampered or damaged:*** If the installed device is replaced by another device of the same make and software, the IoT-ID generated would detect such a replacement. No additional measure is required. The device could be damaged physically, e.g. the power could be disconnected or the communication module could be damaged, making it impossible to generate or communicate IoT-ID. The physical tampering can be avoided by additional anti-tampering mechanisms.

*4) **Case 4- Stealing of IoT-ID for malicious usage:*** A malicious device could be passively listening to the transactions and might eventually steal the identity of a device. This is possible since IoT-ID is being transmitted over the air. Additional mechanisms like encrypting the IoT-ID with the public key must be utilised to prevent stealing. In such cases, IoT-ID acts like a Physically Obfuscated Key (POK) [39] or private key.

IX. Discussions and Future Work

Today to derive a unique device-specific identifier one has to add additional hardware as in the case of Microsoft Azure Sphere [14]. The proposed IoT-ID with novel PUFs shows a promising alternative approach towards device-specific identifier on off-the-shelf devices without additional hardware.

With extensive evaluations in both controlled environments and real-world deployment with 50 devices running over a month, we show that IoT-ID is a robust device-specific identifier even under varying operating conditions (temperature and supply voltage). We now discuss some limitations and possible future work to improve IoT-ID.

- *Design PUFs for other IoT system components*: In this work, we presented novel design of PUFs for clock and ADC. However to enable robust device identification in large-scale deployment of IoT devices, one can extend the PUF features to other system components such as the battery, radio module, etc.
- *Aging*: The experiments presented in this paper was conducted over a duration of a month in real-world conditions. As discussed in Section VI-F, we saw a negligible difference in PUF features over this duration. However, due to the

characteristics of IoT system components, the PUF features will vary after few years of operation. We believe a long-term data collection study is required to systematically understand the impact of aging.

- *IoT device fault detection using PUFs*: Current works aim to create new signatures for identifying faults in an IoT system, such as malfunctioning microcontroller and sensor, etc. In this work, while we focus on using PUFs to derive a device-specific identifier, we believe PUFs can also be employed to detect and isolate faults in an IoT device. PUFs of a working and malfunctioning component will vary significantly.

X. Related Work

The approach of device identification through signature analysis and behavioural profiling of protocols works well to identify class of devices. However, the approach fails to differentiate between multiple instances of the same device. Furthermore, these are application specific. Majority of the recent work for the identification of IoT devices have proposed the analysis and classification of signatures generated by network protocols. IoT Sentinel [40] proposes the extraction of 23 different features from the network packets. Similarly, ProfilIoT [41] and [42] use features across multiple protocol layers to identify different devices. DIoT [43] proposes the implementation of anomaly detection to identify suspicious nodes. BF-IoT [17] discusses the fingerprinting of Bluetooth devices. The temporal properties of the packets have also been studied as features for the identification of devices. The work in [44] extracts Inter-arrival time between consecutive packets and applies deep learning for the classification of devices.

For identifying the device at an instance level, a unique identity could be stored in the non-volatile memory. Whenever there is a need to know the identity, the system could be queried and the identity could be read from the memory. However multiple attacks have been reported, which could retrieve and change the information stored in the memory. The memory values could be read using scanning electron microscopy [6]. Further, fault injection attacks [7] have been reported, which could change the memory values. One proposed approach to create a unique and unclonable identity is to tag devices with plant DNA [3]. To confirm the identity of the chip, the swab of DNA is taken and DNA fingerprinting is performed. However, this approach is expensive and authenticating the device identity is time consuming. Dielets [45] have also been proposed to introduce unclonable identity during manufacturing. This approach becomes impractical to implement for the IoT devices, which are deployed in the wild.

Physically Unclonable Functions (PUFs) have been studied to generate unique device identities for semiconductor devices [8] [9]. Arbiter PUF [12] uses a common rising edge at the clock as well as data input at the D flip flop. In the RO PUF implementation [19], the ring oscillators are configured for the same frequency. Even custom analog circuits are being explored for generating PUFs. The analog PUFs use characteristics like mismatch in differential pairs [46] or comparator offsets [13] for generating signatures. These PUFs need custom circuit implementation. The memory PUF [10] uses the startup values of RAM as a signature. However, the generation of signatures demand power cycling and hence are invasive. The generation of PUF for off-the-shelf components by connecting DAC and ADC back to back and utilizing the offset of the system to generate the unique signature has been explored [47]. This approach necessitates usage of DAC which might not be commonly present in IoT devices. There has been minimal research for generating system identity based on combining individual component PUFs in the Commercially Off the Shelf Components, as proposed in this work. Further, this is one of the first studies to evaluate PUF based identifiers in a real deployment with 50 IoT devices.

XI. Conclusion

In this paper, we presented a novel device-specific identifier – `IoT-ID` derived based on physically unclonable functions (PUFs), that exploits variations in the manufacturing process to uniquely identify Integrated Circuits. `IoT-ID` is composed of multiple PUF features derived from basic system components such as clock oscillator and ADC, present in every microcontroller. Unlike previous efforts, PUFs designed in this work are non-invasive and do not require any additional hardware support. We have evaluated the efficacy of `IoT-ID` on our real-world deployment of 50 IoT devices from different manufacturers running over a month. Our edge ML model has an accuracy of 100% in uniquely identifying individual IoT device instance from our deployment. Furthermore, to analyze the robustness of `IoT-ID` we conducted extensive evaluations under varying operating temperature and supply voltage conditions. Finally, we show the scalability of `IoT-ID` to 1000s of IoT devices using realistic data and numerical analysis. `IoT-ID` is a first step towards robust device-specific identifier in the wild.

References

[1] P. Fraga-Lamas, T. Fernández-Caramés, M. Suárez-Albela, L. Castedo, and M. González-López, "A review on internet of things for defense and public safety," *Sensors*, vol. 16, no. 10, p. 1644, 2016.

[2] O. Arias, J. Wurm, K. Hoang, and Y. Jin, "Privacy and security in internet of things and wearable devices," *IEEE Transactions on Multi-Scale Computing Systems*, vol. 1, no. 2, pp. 99–109, 2015.

[3] M. M. Tehranipoor, U. Guin, and S. Bhunia, "Invasion of the hardware snatchers," *IEEE Spectrum*, vol. 54, no. 5, pp. 36–41, 2017.

[4] Gartner, " Leading the IoT," https://www.gartner.com/imagesrv/books/iot/iotEbook_digital.pdf, 2017, [Online; accessed 11-Jan-2019].

[5] B. V. Sundaram, M. Ramnath, M. Prasanth, and V. Sundaram, "Encryption and hash based security in internet of things," in *2015 3rd International Conference on Signal Processing, Communication and Networking (ICSCN)*. IEEE, 2015, pp. 1–6.

[6] F. Courbon, S. Skorobogatov, and C. Woods, "Reverse engineering flash EEPROM memories using scanning electron microscopy," in *International Conference on Smart Card Research and Advanced Applications*. Springer, 2016, pp. 57–72.

[7] J.-M. Schmidt, M. Hutter, and T. Plos, "Optical fault attacks on AES: A threat in violet," in *2009 Workshop on Fault Diagnosis and Tolerance in Cryptography (FDTC)*. IEEE, 2009, pp. 13–22.

[8] S. Joshi, S. P. Mohanty, and E. Kougianos, "Everything you wanted to know about PUFs," *IEEE Potentials*, vol. 36, no. 6, pp. 38–46, 2017.

[9] U. Rührmair and D. E. Holcomb, "PUFs at a glance," in *Proceedings of the conference on Design, Automation & Test in Europe*. European Design and Automation Association, 2014, p. 347.

[10] D. E. Holcomb, W. P. Burleson, K. Fu *et al.*, "Initial SRAM state as a fingerprint and source of true random numbers for RFID tags," in *Proceedings of the Conference on RFID Security*, vol. 7, no. 2, 2007, p. 01.

[11] B. Chatterjee, D. Das, S. Maity, and S. Sen, "RF-PUF: Enhancing IoT security through authentication of wireless nodes using in-situ machine learning," *IEEE Internet of Things Journal*, vol. 6, no. 1, pp. 388–398, 2019.

[12] J. W. Lee, D. Lim, B. Gassend, G. E. Suh, M. Van Dijk, and S. Devadas, "A technique to build a secret key in integrated circuits for identification and authentication applications," in *VLSI Circuits, 2004. Digest of Technical Papers. 2004 Symposium on*. IEEE, 2004, pp. 176–179.

[13] T. Bryant, S. Chowdhury, D. Forte, M. Tehranipoor, and N. Maghari, "A stochastic approach to analog physical unclonable function," in *Circuits and Systems (MWSCAS), 2016 IEEE 59th International Midwest Symposium on*. IEEE, 2016, pp. 1–4.

[14] "Microsoft Azure Sphere," https://azure.microsoft.com/en-in/services/azure-sphere/, Microsoft, [Online; accessed Oct-2019].

[15] "Arduino Mega 2560," https://www.arduino.cc/en/Guide/ArduinoMega2560, Atmel, [Online; accessed August-2019].

[16] T. Chakraborty, A. U. Nambi, R. Chandra, R. Sharma, M. Swaminathan, Z. Kapetanovic, and J. Appavoo, "Fall-curve: A novel primitive for IoT fault detection and isolation," in *Proceedings of the 16th ACM Conference on Embedded Networked Sensor Systems*. ACM SenSys, 2018, pp. 95–107, 2018.

[17] T. Gu and P. Mohapatra, "BF-IoT: Securing the IoT networks via fingerprinting-based device authentication," in *2018 IEEE 15th International Conference on Mobile Ad Hoc and Sensor Systems (MASS)*. IEEE, 2018, pp. 254–262.

[18] C. Marchand, L. Bossuet, U. Mureddu, N. Bochard, A. Cherkaoui, and V. Fischer, "Implementation and characterization of a physical unclonable function for IoT: a case study with the TERO-PUF," *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, vol. 37, no. 1, pp. 97–109, 2018.

[19] G. E. Suh and S. Devadas, "Physical unclonable functions for device authentication and secret key generation," in *Design Automation Conference, 2007. DAC'07. 44th ACM/IEEE*. IEEE, 2007, pp. 9–14.

[20] *ATmega640/V-1280/V-1281/V-2560/V-2561/V-DATASHEET*, Atmel, Feb. 2014.

[21] *CC3200 Datasheet*, Texas Instruments, Feb. 2015.

[22] *nRF52832 - Product Specification v1.0*, Nordic Semiconductors, Feb. 2017.

[23] N. Walt Kester, *The Data Conversion Handbook*. Analog Devices, 2005.

[24] "DC parameters: Input offset voltage," http://www.ti.com/lit/an/sloa059/sloa059.pdf, Texas Instruments, 2001.

[25] Wikipedia, " Process Design Kit," https://en.wikipedia.org/wiki/Process_design_kit, [Online; accessed Jan-2019].

[26] " Spectre Circuit Simulator," https://www.cadence.com/content/cadence-www/global/en_US/home/tools/custom-ic-analog-rf-design/circuit-simulation/spectre-circuit-simulator.html, Cadence, [Online; accessed Jan-2019].

[27] "Hspice," https://www.synopsys.com/verification/ams-verification/hspice.html, Synopsys, [Online; accessed Jan-2019].

[28] H. Hung and V. Adzic, "Monte carlo simulation of device variations and mismatch in analog integrated circuits," *Proc. NCUR 2006*, pp. 1–8, 2006.

[29] M. Frankiewicz and A. Kos, "Wide-frequency-range low-power variable-length ring oscillator in UMC CMOS 0.18 μm," in *Proceedings of the 20th International Conference Mixed Design of Integrated Circuits and Systems-MIXDES 2013*. IEEE, 2013, pp. 291–293.

[30] "UMC fabricator," http://www.umc.com/English/.

[31] "Behind the ADC Veil: Demystifying Common DC Specifications," https://www.electronicdesign.com/technologies/analog/article/21807150/behind-the-adc-veil-demystifying-common-dc-specifications, ElectronicDesign, [Online; accessed Jan-2020].

[32] "Understanding Operational Amplifier Specifications," http://www.ti.com/lit/an/sloa011/sloa011.pdf, Texas Instruments, [Online; accessed Jan-2020].

[33] "TI Precision Labs – ADCs: Statistics Behind Error Analysis," https://training.ti.com/ti-precision-labs-adcs-statistics-behind-error-analysis?HQS=asc-dc-padc-tipl2018-asset-tr-ElectronicDesign-wwe&DCM=yes&dclid=CPn_g_KTo-cCFTIYrQYdj3kNhg, Texas Instruments, [Online; accessed Jan-2020].

[34] "Half-normal distribution," https://en.wikipedia.org/wiki/Half-normal_distribution, stackexchange, [Online; accessed Jan-2020].

[35] A. Kumar, S. Goyal, and M. Varma, "Resource-efficient machine learning in 2 kb ram for the internet of things," in *Proceedings of the 34th International Conference on Machine Learning-Volume 70*. JMLR. org, 2017, pp. 1935–1944.

[36] *PSoC 63 with BLE Architecture Technical Reference Manual, Document No. 002-18176*, Apr. 2018.

[37] "Application Note-Clock Oscillators," https://www.vectron.com/products/literature_library/clock_oscillators.pdf, Vectron, [Online; accessed Aug-2019].

[38] "Central Limit Theorem and the Law of Large Numbers," https://ocw.mit.edu/courses/mathematics/18-05-introduction-to-probability-and-statistics-spring-2014/readings/MIT18_05S14_Reading6b.pdf, MIT, [Online; accessed Oct-2019].

[39] B. L. P. Gassend, "Physical random functions," Ph.D. dissertation, Massachusetts Institute of Technology, 2003.

[40] M. Miettinen, S. Marchal, I. Hafeez, N. Asokan, A.-R. Sadeghi, and S. Tarkoma, "Iot sentinel: Automated device-type identification for security enforcement in iot," in *2017 IEEE 37th International Conference on Distributed Computing Systems (ICDCS)*. IEEE, 2017, pp. 2177–2184.

[41] Y. Meidan, M. Bohadana, A. Shabtai, J. D. Guarnizo, M. Ochoa, N. O. Tippenhauer, and Y. Elovici, "ProfilIoT: a machine learning approach for iot device identification based on network traffic analysis," in *Proceedings of the Symposium on Applied Computing*. ACM, 2017, pp. 506–509.

[42] Y. Meidan, M. Bohadana, A. Shabtai, M. Ochoa, N. O. Tippenhauer, J. D. Guarnizo, and Y. Elovici, "Detection of unauthorized IoT devices using machine learning techniques," *arXiv preprint arXiv:1709.04647*, 2017.

[43] T. D. Nguyen, S. Marchal, M. Miettinen, M. H. Dang, N. Asokan, and A.-R. Sadeghi, "DIoT: A Crowdsourced Self-learning Approach for Detecting Compromised IoT Devices," *arXiv preprint arXiv:1804.07474*, 2018.

[44] S. Aneja, N. Aneja, and M. S. Islam, "IoT device fingerprint using deep learning," in *2018 IEEE International Conference on Internet of Things and Intelligence System (IOTAIS)*. IEEE, 2018, pp. 174–179.

[45] P. Ralston, D. Fry, S. Suko, B. Winters, M. King, and R. Kober, "Defeating counterfeiters with microscopic dielets embedded in electronic components," *Computer*, vol. 49, no. 8, pp. 18–26, 2016.

[46] S. Deyati, B. Muldrey, A. Singh, and A. Chatterjee, "Design of efficient analog physically unclonable functions using alternative test principles," in *Mixed Signals Testing Workshop (IMSTW), 2017 International*. IEEE, 2017, pp. 1–4.

[47] A. Duncan, L. Jiang, and M. Swany, "Repurposing SoC analog circuitry for additional COTS hardware security," in *2018 IEEE International Symposium on Hardware Oriented Security and Trust (HOST)*. IEEE, 2018, pp. 201–204.