Diagnosis of Malicious Bitstreams in Cloud Computing FPGAs

Jayeeta Chaudhuri and Krishnendu Chakrabarty
Department of Electrical and Computer Engineering, Duke University, Durham, NC, USA
School of Electrical, Computer, and Energy Engineering, Arizona State University, Tempe, AZ, USA

***Abstract*—Multi-tenant FPGAs are increasingly being used in cloud computing technologies. Users are able to access the FPGA fabric remotely to implement custom accelerators in the cloud. However, the sharing of FPGA resources by untrusted third-parties can lead to serious security threats. Attackers can configure a portion of the FPGA with a malicious bitstream. Such malicious use of the FPGA fabric may lead to severe voltage fluctuations and denial-of-service. In this work, we consider FPGAs that support time-based multi-tenancy i.e., a single user has access to the FPGA at a time. We propose a convolutional neural network (CNN)-based approach to detect malicious RO-like circuits that are configured on an FPGA by learning features from the data-series representation of the bitstreams of malicious circuits. We use the classification accuracy, true-positive rate, and false-positive rate metrics to quantify the effectiveness of CNN-based classification of malicious bitstreams. Our threat model includes a variety of power-wasting circuits that are used to configure FPGAs in the cloud. We propose a two-stage malicious bitstream detection framework for classification and diagnosis of the type of malicious circuit implemented by a particular bitstream. We further propose a novel window-merging technique to improve model performance in the second stage of the detection framework. Experimental results on Xilinx FPGAs demonstrate the effectiveness of the proposed method.**

I. Introduction

Field-programmable gate-arrays (FPGAs) are now ubiquitous in cloud computing infrastructures and reconfigurable system-on-chips (SoCs). The availability of FPGAs in cloud data centers has opened up new opportunities for users to improve application performance by enabling them to implement customizable hardware accelerators directly on the FPGA fabric. In addition to increasing computational efficiency at reduced cost, partial reconfiguration allows new types of FPGA designs that would be otherwise impossible to implement. FPGAs are therefore being incorporated today for specialized compute-intensive services in cloud data centers, e.g., by Amazon and Microsoft [1] [2]. FPGAs in the cloud support multi-tenancy, which allows users to perform customized operations.

As FPGAs are increasingly shared and remotely accessed by multiple users and third parties, they are a major reason for rising security concerns. Modules running on an FPGA may include circuits that induce voltage-based fault attacks and denial-of-service (DoS) [3] [4]. An attacker might configure some regions of the FPGA with bitstreams that implement malicious circuits. The FPGA is split into logically isolated, separate regions that can be occupied by multiple users at the same time. As a majority of integrated circuits are supplied by a common power distribution network (PDN) for the entire FPGA board, there exists an electrical connection between the victim and attacker modules [5]. Therefore, voltage fluctuations caused at the victim end can be measured by voltage sensors such as a ring oscillator (RO) and a time-to-digital converter (TDC) at the attacker end. This may lead to voltage-based attack and DoS of the FPGA device. Moreover, a grid of ROs can be activated simultaneously at a particular frequency of activation (f_{RO}) to generate voltage-drop pulses in rapid succession; this shuts down the on-board voltage regulator and causes the FPGA to crash [6].

Various countermeasures have been adopted to protect FPGAs from unauthorized third-party access. In order to prevent the attacker from directly configuring the FPGA with an invalid or malicious bitstream, the defender (e.g., the FPGA vendor) can incorporate an on-chip bitstream checking mechanism that detects and blocks such bitstreams before they are loaded to the FPGA fabric [7]. Alternatively, the FPGA bitstream structures can be examined to identify signatures that satisfy the requirements of FPGA-based fault attacks [8]. However, both [7] and [8] require reverse-engineering (RE) of the bitstreams to their corresponding netlists; this technique is time-consuming as well as complex. Additionally, the RE tools are specific to each FPGA family and vendor. Therefore, applying RE techniques to extract signatures of malicious structures is not always feasible.

In this paper, we provide a more general methodology to examine FPGA bitstreams for malicious RO-like patterns. Analyzing features from the bitstream itself is time-consuming as well as computationally intensive. Therefore, we focus on visualizing these bitstreams as a data-series plot and use a convolutional neural network (CNN)-based approach to learn and detect malicious patterns from the data series. We also propose a robust two-tier bitstream partitioning-based framework that detects a malicious bitstream and also diagnoses the type of power-wasting circuit implemented by that bitstream. Our proposed frameworks can be extended to perform detection and classification of malicious configuration bitstreams of multiple FPGA families and vendors. The key contributions of this paper are as follows:

- Generation of RO variants, non-combinational ROs, glitch amplification circuits, and other power-wasting circuits which are major security threats to cloud FPGAs.
- Visualization of FPGA bitstreams as data series and the

*This work was supported in part by the National Science Foundation under grant no. CNS-2011561.

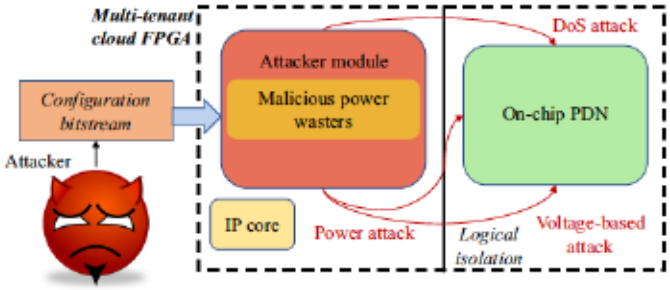


Fig. 1: Schematic of the threat model.

extraction of signature patterns that differentiate benign bitstreams from malicious RO-based bitstreams.

- A CNN-based framework that detects malicious bitstreams based on features extracted from RO patterns.
- An end-to-end robust machine learning (ML)-based framework to detect and diagnose malicious power-wasting circuits that are used to configure FPGAs in the cloud.
- Evaluation of the proposed frameworks for multiple FPGA families and real-life FPGA bitstreams.
- Quantitative comparison with the state of the art, in terms of performance metrics and run time.

The remainder of the paper is organized as follows. Section II discusses how FPGAs are prone to voltage-based attacks, describes related prior work on FPGA bitstream checking, and presents the threat model. Section III describes the flow of our CNN-based feature extraction framework for detecting malicious RO-like patterns. Section IV further broadens the threat model and presents a novel two-stage ML-based framework for diagnosis of a wide range of malicious power-wasting circuits. Experimental results and observations are presented in Section V. Section VI discusses future work and new research directions. Section VII concludes the paper.

II. Background and Motivation

A. Threat Model

We assume that the FPGAs support time-based multi-tenancy i.e., a single user has access to the FPGA at a time. In this scenario, an attacker can configure the FPGA with bitstreams that implement malicious power-wasting circuits. An attacker with no physical access to the target FPGA device can still launch voltage- and power-based attacks on the FPGA. Our proposed CNN-based detection framework is based off-chip and is assumed to be trusted. Therefore, an adversary or any third-party user is unaware of the functionality of the proposed detection framework and will be unable to physically tamper it. As shown in [9], an FPGA device must first decrypt the incoming FPGA bitstream before it is being configured. The FPGA bitstreams can be decrypted using bitstream encryption tools available from FPGA vendors such as Xilinx. Therefore, in this work, the user-input bitstream is assumed to be decrypted before it is used for evaluation by our detection framework. It is also assumed that an attacker cannot hide power-wasting circuits in the obfuscated design, as it would be difficult to detect them without taking the design apart using RE methods. Fig. 1 illustrates the threat model considered for remote attacks on cloud-based FPGAs.

B. Attacks on Cloud-Computing FPGAs

Multi-tenant FPGAs are now increasingly being used in cloud computing environments. These FPGAs also support time-multiplexing, which enables multiple users to partially configure the FPGAs with custom, logically isolated modules during different time windows. Multi-tenancy significantly improves FPGA utilization and flexibility. Unfortunately, the modeling of multi-tenant FPGAs in the cloud can pose a serious threat due to malicious users. Attackers can configure a portion of the FPGA with a potential power-wasting circuit that affects the power distribution network (PDN) of the FPGA. The PDN is carefully designed to supply power to all the modules of the FPGA. The voltage drop V_{drop} across the PDN is the summation of voltage drops across all the logic blocks of the FPGA and may be expressed as follows:

$$V_{drop} = IR + L \times di/dt \tag{1}$$

where R is the resistance of the PDN, L is the inductance of the PDN, and di/dt is the rate of change of the electric current inside the PDN, which depends on the workload that the PDN is subjected to [6]. Although the modules of every tenant are physically isolated, an adversary can still inject malicious power-wasting circuits in the FPGA that critically affect the PDN. A power-wasting circuit is a malicious circuit that consumes a large amount of power and potentially affects the PDN of the FPGA. Such circuits contribute to voltage and power-based attacks on FPGAs in the cloud. On-chip voltage sensors have been used in [10] to measure voltage changes in the PDN of an FPGA. Since all the modules of the multi-tenant FPGA share the same PDN, voltage fluctuations in any particular co-tenant module will cause voltage variations in the entire PDN. The sensor readings generated in [10] are utilized to identify the type of computations performed by a co-tenant of that FPGA. Thus, an attacker may receive confidential information about the tasks performed on the multi-tenant FPGAs, and launch attacks on the sensitive modules.

The work in [11] demonstrates a denial-of-service (DoS) attack as well as the injection of timing faults by deploying a grid of ROs on the FPGA fabric. In [12], customized power sensors using delay lines have been configured on a multi-tenant FPGA, which are then used to perform side-channel analysis attacks on an AES-128 core present in the same FPGA. Ring oscillator (RO)-based voltage sensors have been demonstrated to induce fault attacks and crash the FPGA [6]. It has been shown in [6] how a DoS attack on an FPGA requires only a small number of ROs, occupying about 12% of the available LUTs. By suddenly enabling all the ROs, the attacker can cause a significant voltage drop.

Non-combinational oscillators have been proposed in [13]; these escape design rule check (DRC) by FPGAs in the cloud e.g., in the case of Amazon Web Services (AWS). As AWS rejects a design containing combinational loops, an attacker can resort to generating loop-free ROs, which can be a major threat to cloud FPGAs.

Enabling a grid of ROs simultaneously results in a high

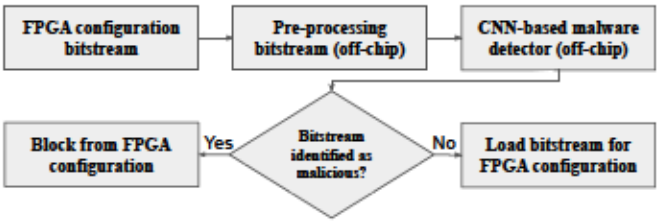


Fig. 2: CNN pipeline for detecting malicious bitstreams.

current and hence, a large voltage drop, according to Equation (1). To cause a significant voltage drop enough to crash the FPGA, the ROs are toggled at a frequency . Experiments have shown that 1920 ROs switching at KHz successfully crashes a Lattice FPGA board [6]. Currently, Amazon cloud service providers (CSPs) prohibit bitstream generation of netlists representing RO-like circuits. However, in [13], the authors show how non-combinational oscillators evade the AWS design rule check (DRC) and successfully generate bitstreams on the cloud FPGAs.

C. Defenses

Several methods have been proposed to evaluate bitstreams before FPGA configuration. The *icebox_vlog* tool is used to reverse-engineer a bitstream to the technology-mapped netlist [8]. However, different FPGA families and FPGA vendors may use different bitstream formats, which require RE tool modification.

An approach to analyze FPGA bitstreams using neural networks is presented in [14]. The dataset used in this work consists of partial bitstreams with different IPs, including adders, multipliers and subtractors. It focuses on partial bitstreams because they can be trained faster, as compared to full bitstreams. However, [14] does not consider the more complicated cases of detecting malicious ROs and RO variants, especially when they are embedded within larger designs.

ML-based approaches for malicious circuit detection (especially hardware Trojans) have been proposed in [15] and [16]. These methods detect malicious circuits either from gate-level netlists or by using frequency domain signals and layout images. The work in [17] demonstrates a localized feature-based approach using a recurrent neural network model. Studies on the detection of internet-of-things (IoT) malware using feature extraction techniques have also been conducted [18]. ML-based techniques have also been adopted in [19] and [20] to distinguish between Trojan-free and Trojan-inserted circuits based on features extracted from their gate-level netlists. However, [19] and [20] are limited with their ability to accurately identify hardware Trojans.

III. Identification of RO-like patterns

In this section, we describe the proposed methodology to detect malicious RO-like signatures from data-series representation of bitstreams. We check for specific structural attributes of these series that are not found in benign bitstreams, thus indicating the presence of malicious logic. We propose a CNN-based image classification framework that learns these key features and detects a malicious bitstream before it is used for FPGA configuration. Note that our CNN pipeline for malicious bitstream detection is based off-chip. The end-user inputs a bitstream to the pre-trained CNN model for authentication before loading it to the FPGA. If the CNN classifies the bitstream as malicious, it is blocked from FPGA configuration. Fig. 2 illustrates the proposed CNN pipeline.

A. Justification for the use of ML Models

A FPGA configuration bitstream has a vendor-specific format. Although the file format of the bitstream is publicly available, the mapping of the bits to FPGA LUTs and the format of the configuration bits are not documented by the FPGA vendors [9]. The FPGA configuration bitstream consists of a sequence of frames, each of which contains the configuration information about the LUTs and their interconnects. All the frames have a fixed and identical length. We target the Xilinx Virtex Ultrascale (VU440) FPGA in our work because it is widely used in high-performance applications and provides integration capabilities on a 20nm FinFET node [21]. The number of configuration frames for the VU440 bitstream is 262110, with each frame consisting of 123 32-bit words. The LUT and the interconnect configuration data are encoded in a compressed binary format, which allows for efficient storage of the FPGA configuration bitstream. Since both the LUTs and interconnects are arranged as a grid of rows and columns in the FPGA, they form a well-organized structure. A malicious bitstream may contain structures such as XOR gates or ROs that are not typical in benign bitstreams. Therefore, by analyzing the statistical properties of the configuration bitstream (including the type and the number of LUTs and interconnects), well-trained ML models will be able to identify data patterns that might be indicative of a malicious bitstream.

B. Data Collection

We generate a large dataset consisting of benign and malicious bitstreams that are used to configure an FPGA. We generate 95 benign bitstreams and 80 malicious RO-based bitstreams. FPGA bitstreams can be of the following two types - partial and full. A partial bitstream is used to configure only a portion of the FPGA, which in turn increases the flexibility of the system. Partial bitstreams contain all the configuration logic necessary for partial reconfiguration of the FPGA module. Note that the size of a partial bitstream is directly proportional to the size of the FPGA region it is configuring. For example, if the reconfigurable region comprises 10% of the entire device resources, the size of the partial bitstream is almost 10% of the full bitstream. The bitstreams generated in this work are full bitstreams in .bin format (i.e., binary data files without the ASCII header at the beginning of the file). We use full bitstreams because they are of fixed size and can configure the entire FPGA at a particular time. A full bitstream is used in time-sharing applications where multiple users can access the FPGA at different times. For example, the size of a full bitstream for VU440 FPGA is 128,966,372 bytes. The bitstreams used for our experiments are as follows:

Benign Bitstreams: We generated bitstreams that configure designs such as USBs, keyboard controllers, AES cores,

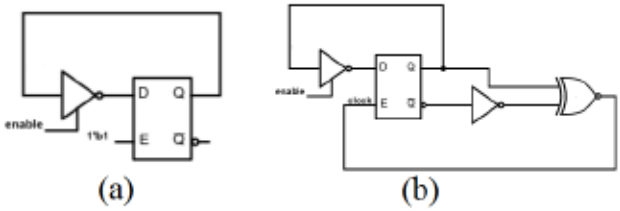


Fig. 3: Non-combinational loops in oscillators: (a) Latched RO; (b) Self-clocked RO.

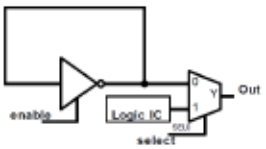


Fig. 4: Conditionally active RO.

MIPS cores, IPs that support trigonometric, quadratic, and other arithmetic operations, and VGA OpenCores. We also generated bitstreams that implement ISCAS '85, ITC '99, and EPFL benchmarks. We ensured that these bitstreams are representative of data used in real-life benign applications. The designs are implemented in Verilog and VHDL.

Malicious Bitstreams: We define a malicious bitstream as a bitstream implementing a circuit that is capable of overheating the FPGA or launching power- and voltage-based attacks on the FPGA. We focus on simple ROs as well as variants of ROs. Most CAD tools can detect a combinational loop and raise an error during DRC. However a non-combinational RO escapes DRC and also supports successful bitstream generation. Therefore, we take into consideration such loop-free ROs and treat them as malicious circuits in this paper. We design two such variants of loop-free ROs:

Latch-based RO: Fig. 3(a) presents an example of a latch-based RO. As illustrated, the output of the last stage of RO is fed as the input to a latch in the middle of the loop. Since this latch divides the circuit into two separate combinational loops, the overall design is no longer combinational.

Self-clocked RO: An example of a self-clocked RO is illustrated in Fig. 3(b). Here, the Q' output of the D flip-flop faces one inverter delay. The inverter output is XNOR-ed with the Q output of the flip-flop and the result is fed back to the clock input of the register. The output of the XNOR gate glitches every time the register output changes; this occurs due to the signal delay from Q' through the inverter.

We implement the above non-combinational ROs using the Vivado design tools and request bitstream generation using the *write_bitstream* command. We observe that bitstreams are generated for both latch-based RO and self-clocked RO, supporting the claim that they are capable of evading DRC.

Conditionally active RO: Note that an attacker might not use a bare RO to configure the FPGA. The malicious activity might be concealed using circuits that conditionally activate the ROs. These circuits lead to a failure of the entire PDN. We generate MUX-based conditionally active ROs that are capable of serving the intent of the attacker. An implementation of a conditionally active RO is shown in Fig. 4.

```
Input: Xilinx FPGA bitstream
Output: CSV file corresponding to
Read file as hex bytes and store in Content
Insert delimiters in Content
Split Content into array
for i in range (length(Content) do
  | Convert Content to decimal
end
for i in range (length(Content) do
  | Insert newline n in Content, where n = 10
end
Convert Content to string
return Content
```

Fig. 5: Procedure for converting a full bitstream to its corresponding CSV format.

C. Mapping Bitstreams to Data-Series Representation

The sparsity score of an FPGA bitstream is calculated based on the fraction of bitstream bytes that have a value of zero. The average sparsity score for all the generated benign and malicious bitstreams is 0.92, indicating that a large portion of the FPGA LUTs is not being utilized. Therefore, we convert these bitstreams to their corresponding data series for efficient data preprocessing. This data is then represented as image files for each bitstream. Plotting bitstreams as data series and storing them as image files enables us to:

1) Identify specific patterns in the image files that represent malicious behaviour;
2) Utilize a CNN-based image classification framework to detect these malicious patterns in images.

CNNs are widely used in a variety of applications, e.g., computer vision, medical image analysis, and image classification and segmentation. These networks rely on a large amount of data to avoid overfitting. However, it is sometimes difficult to have access to the required amount of data. Note that the procedure of generating a bitstream takes upto minutes. Therefore, generating a large dataset of benign and malicious bitstreams and then converting them to image files is time-consuming. Image augmentation, a well-known type of data augmentation technique, is used in such scenarios. Image augmentation increases the size of the training dataset by creating transformed versions of training set images. In the particular problem being studied in this paper, i.e., extracting features from bitstream image files and classifying them as being either malicious or benign, image augmentation serves two important purposes:

It helps in building a well-trained CNN model that learns RO-based features from the image, rather than from the bitstream values.

It artificially expands the training dataset with new and realistic examples from existing training data.

Note that image augmentation is performed on the training dataset only and not on the validation or test dataset. Deep-learning techniques such as the CNN can learn meaningful features from the transformed training data as well as the original training data. Many techniques for augmentation of

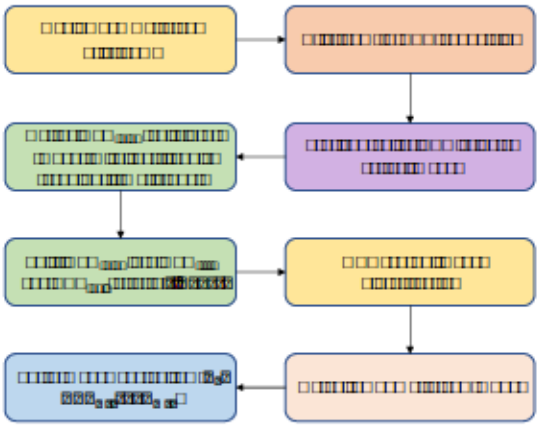


Fig. 6: Illustration of the steps involved in the training and evaluation of CNN-based malicious bitstream detection model.

image data exist, specifically image rotation, random cropping, flipping images, adding Gaussian noise, and adjusting brightness, contrast, and saturation of images. We use image rotation as the data augmentation technique for increasing the size of our training dataset. We choose this technique because it has been shown to be effective for the detection problem [22] with 1) a significant improvement in training accuracy, and 2) reduction in training loss as the number of training examples is increased. By applying rotation-based image augmentation, we generate a greater number of artificial training data images since the original dataset is limited. These newly generated images retain the visual properties of RO-based circuits, allowing the recognition of malicious patterns from different angles of the bitstream-generated images.

We use the series representation of bitstreams to train our CNN-based image classification framework. To obtain the bitstream as data series, we perform the following operations:

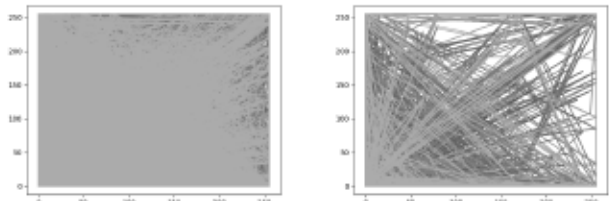
1) We convert the benign and malicious bitstreams into comma-separated values (CSV) files. The procedure for bitstream-to-CSV conversion is illustrated in Fig. 5.
2) We plot the content of each CSV file as two-dimensional series data and store them as image files (.png format).
3) We define another .csv file, namely CSV_{total}. This file stores all of the generated image files along with their marked labels ('0' or '1'). The value '0' corresponds to the image file of a benign bitstream and the value '1' corresponds to the image file of a malicious bitstream.
4) We split CSV_{total} into the training dataset CSV_{train} and the test dataset CSV_{test} in the ratio 80:20. We choose the split ratio in such a way that it represents all the data with minimum training loss and avoids underfitting as well as overfitting, both of which are undesirable.

D. Malicious Bitstream Detection using CNNs

Next, we proceed to build our CNN-based image classification framework and use it to learn and classify the data. The overall steps involved in the CNN-based detection model are shown in Fig. 6. A CNN is a feed-forward type of neural network that takes an image as input and convolves it with filters or kernels to extract features. Each filter is designed to detect a certain feature in the data, such as edges or boundaries. Next, the output of the filter is passed through a pooling layer that combines the individual filter outputs. The pooling layer allows the CNN to understand the spatial proximity in the bitstream [23]. Therefore, we apply CNN-based learning to gain insight into how certain bytes of the bitstream can affect the performance of the FPGA. By applying CNN-based feature extraction, specific malicious patterns can be accurately detected in the bitstreams that otherwise might not be visible if RE methods are applied. The neural network used in our classification framework has four convolutional layers, four max pooling layers, and four linear layers. We choose this architecture for the following reasons:

1) The data received at the CNN input layer is an image i.e., an array of pixels. This data may have noise included in it. Therefore, we increase the number of filters by adding more convolutional layers, and extract useful features as the network gets deeper.
2) As we are looking at the problem of identifying specific patterns indicating malicious behaviour, we utilize gray-scale images for the training and evaluation of the CNN model. Note that the colored images are associated with a range of colors that do not signify a particular bitstream feature and hence, is not relevant in the current problem. However, extracting meaningful features from a gray-scale image is much more complex than extracting features from a colored image. In such a scenario, it is desirable to have more linear layers using the non-linear activation function. We choose the Rectified Linear Unit (ReLU) as our activation function because it trains the CNN model faster and more effectively, without causing a significant drop in classification accuracy [24].

Large neural networks trained on relatively small datasets can cause overfitting. To mitigate this issue, we add a dropout layer after every maxpooling layer. We perform hyperparameter tuning to select the dropout value . We have considered values of in the range . However, choosing = 0.25 yields the least training loss and the highest training accuracy. The image files corresponding to a benign bitstream and a malicious bitstream are shown in Fig. 7(a) and Fig. 7(b), respectively. For the malicious bitstream, the intensity of patterns across the image follows a non-uniform distribution. For the benign bitstream, we observe a higher intensity of a particular pattern in the image. However, the same pattern



(a) (b)

Fig. 7: Image files corresponding to: (a) benign bitstream; (b) malicious bitstream.

appears with less intensity in the malicious bitstream. These observations guide us to use these specific attributes as features to train our proposed neural network-based malicious bitstream detector. Note that the CNN-based classification framework is the foremost and the most important step as it helps in distinguishing between benign and malicious FPGA bitstreams through static analysis of the bitstreams.

IV. Proposed Two-Stage Malicious Bitstream Detection Framework

A. Motivation

Our primary goal is to enhance the detection of ROs in bitstreams as well as detect a wider range and substantial number of power-wasting circuits, with reduced processing time. In this scenario, reducing the size of the data being processed can help in reducing the time overhead. By reducing the number of features to be processed by the detection model, the amount of time required to evaluate an FPGA bitstream can be significantly reduced. Most of the recent work on power and voltage-based attacks on cloud-computing FPGAs focus on the implementation of RO-like circuits as voltage sensors. Even though ROs are typical examples of power wasting circuits, the authors of [25] have proposed an evolved version of the standard 128-bit AES circuit that is capable of consuming aggressive amount of power. The AES-based power waster is modified by introducing XOR gates between each round to induce glitching, and therefore high power wastage. Recently, circuits based on glitch amplification have been explored that do not incorporate oscillators [26]. Such circuits evade DRC and are capable of causing power-hammering attacks, thus crashing the FPGA board.

It is important to detect the presence of such power-wasting circuits before they are configured on the FPGA. Note that there exist digital circuits that are used for genuine, real-life applications but consume high power due to heavy FPGA LUT utilization. In other words, the percentage of FPGA utilization is expected to be more for a densely packed, large circuit compared to a much smaller circuit. However, a grid of ROs that occupies as little as 12% of FPGA logic is still capable of consuming similar or higher amount of power compared to larger, benign circuits occupying 65% of FPGA [6]. Hence, if we utilize a mechanism that blocks all circuits with high power consumption, it can incorrectly block compute-intensive benign circuits. Therefore, we first identify circuits that occupy 12% of the FPGA but consume a significantly high amount of power. We synthesize and implement the malicious power-wasting circuits on the VU440 board. We list these circuits and their power consumption in Table I. We monitor the power consumption using the power analysis feature. Note that the power-wasting circuits itemized in Table I are used to launch power and voltage-based attacks in [3], [6], [25], and [26] have LUT utilization less than %.

B. Malicious Power-Wasting Circuits

The following power-wasting circuits are evaluated:

1) RO: We implement single-stage and three-stage ROs that are capable of launching voltage-based attacks [6]. The oscillation frequency of an -staged RO with propagation delay is defined by the following equation:

—— (2)

From Equation (2), we observe that the oscillation frequency will be higher for ROs with single and three stages. Therefore, we choose these ROs as potential power-wasting circuits. We also implement conditionally active ROs.
2) Non-combinational ROs: We implement single-stage and three-staged latched ROs.
3) Power-wasting AES: We implement the AES-based power waster used in [25]. Note that this circuit does not perform encryption but only contributes to high power consumption. The original 128-bit AES core is modified by adding a XOR gate between the AES rounds. Placing the XOR gate in the AES core induces glitching due to gate delays.
4) Glitch amplification-based circuits: We implement single-stage and three-stage self-oscillating circuits based on glitch amplification [7]. We also evaluate glitch amplification circuits having -input XOR gates, [26]. An example circuit is illustrated in Fig. 8.
5) -bit shift register: Although a shift register is used for benign computational purposes, replicating multiple instances of this circuit can dangerously affect the PDN and cause voltage and power-based attacks on the FPGA.

TABLE I: List of power-wasting circuits.

Type of circuit	Power consumption (W)
Single-stage RO (1920 instances)	32.4
Three-stage RO (3600 instances)	27.9
Latched RO (1500 instances)	26.7
Self-clocked RO (1800 instances)	26.8
Unrolled AES core	384.12
AES core (10 instances)	696.34
AES core (20 instances)	925.9
Chained 16-bit shift registers	26.5
16-bit shift registers (10 instances)	42.8
16-bit shift registers (20 instances)	76.7
3-input XOR-based glitch amplification	26.5

C. Characteristics of Power-Wasting Circuits

From Section IV-A, we recognize the features of a power wasting malicious circuit as follows:

1) Combinational cycles, indicating the presence of ROs;
2) Data-to-clock routings, which may hide non-combinational loops e.g. self-clocked ROs;
3) Presence of XOR gates at unusual locations of the circuit that may induce glitching;
4) Significantly high FPGA power consumption even when less than 10% of FPGA LUTs are occupied.

Table II shows the bitstream generation times for Xilinx FPGAs on a 2.4 GHz Intel Xeon Gold 5115 CPU with 768 GB of RAM. The bitstreams have been implemented for a single-stage RO. The data in Table II explains that the time

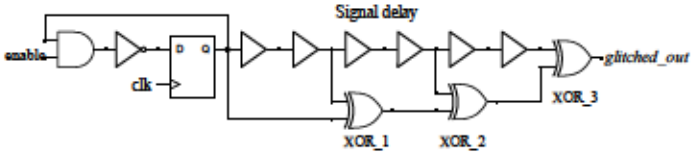


Fig. 8: A glitch amplification-based circuit.

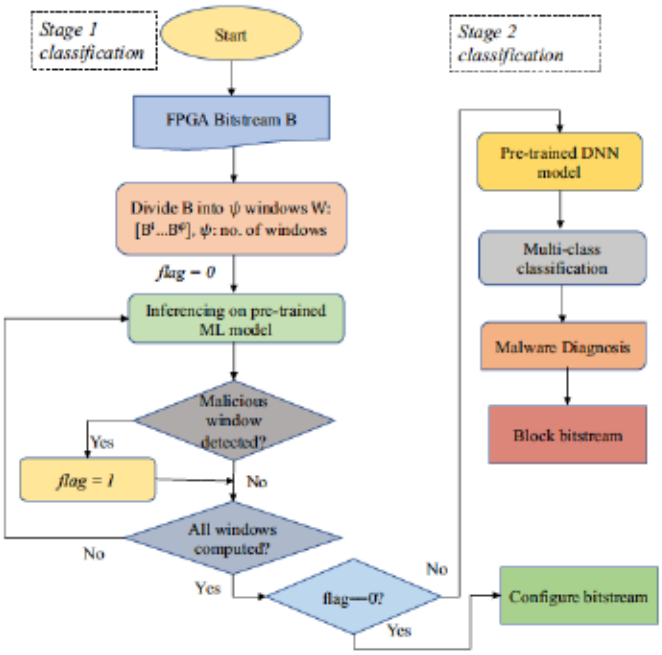


Fig. 9: Overall architecture of the two-stage malicious bitstream diagnosis pipeline.

TABLE II: Bitstream generation times on FPGA devices.

FPGA device	Bitstream generation time (s)
Zynq-7000	15
Virtex 7	49
Kintex Ultrascale	297
Virtex Ultrascale	625

required for bitstream generation increases with the size of the FPGA. Note that the time taken to reverse engineer a bitstream is directly proportional to the size of a bitstream. Therefore, if a bitstream has a higher generation time, the time taken to reverse engineer the bitstream also increases. As explained in [27], a full bitstream performing high-computing applications, such as data analytics, may require several weeks to be reverse-engineered. Let us consider a scenario where we have to evaluate an FPGA bitstream for analyzing malicious structures, before the bitstream is deployed on the AWS cloud instance. If we were to use RE-based approach, we would need to perform several iterations of RE until all the malicious structures are detected [8]. This process can take several weeks depending on the size and complexity of the bitstream. We propose a two-stage ML-based malicious bitstream detection framework that identifies malicious bitstreams across different FPGA devices. The overall framework is illustrated in Fig. 9. In Stage 1, a sliding window-based byte extraction is performed and malicious signatures in the bitstream are identified using pre-trained ML models. The Stage 2 model proceeds to diagnose the type of malicious circuit from the malicious windows that are detected in Stage 1 of the pipeline.

D. Stage 1: Sliding Window-based Detection

1) Training of Stage 1 ML Model

Note that our bitstream detection pipeline is executed off-chip and can be extended to any FPGA device or family, as required. As explained in [9], the information about the mapping of bits to LUTs and circuit functionality are not stored arbitrarily in different parts of the configuration bitstream; instead it is stored in the form of continuous sequence of bytes in the bitstream. Therefore, we first try to identify features from the bitstreams that indicate the presence of malicious signatures. Here, the signature is determined by a set of concurrent bytes in the bitstream. In order to get the most accurate locations of such malicious signatures, we divide the bitstream into a group of non-overlapping windows of size n bytes. The procedure for bitstream generation and extraction of sub-bytes is illustrated in Fig. 10.

Overlapping vs. non-overlapping windows: In this work, we have not considered the scenario when an attacker splits power-hungry circuits across multiple windows, such that each window is classified as benign. However, if we had to address this situation, we would propose partitioning the bitstreams using overlapping windows (instead of non-overlapping windows) for more accurate data analysis as well as examining the interaction between multiple windows for the same circuit.

We initiate training of the ML model with a given number of windows, ψ. The training dataset includes a large selection of benign and malicious bitstreams. The nature of the bitstreams is elaborated in Section V-G. Since the size of a VU440 bitstream is 128966372 bytes, the size of each sliding window $n = \lceil \frac{128966372}{\psi} \rceil$. We further increase the number of sliding windows (reduce the size of each window) to narrow down to those windows with the most likelihood of containing malicious signatures. We perform hyperparameter tuning to arrive at the most suitable choice of the number of windows ψ. Once we arrive at an optimum value of ψ by hyperparameter tuning, we divide an user-input bitstream into ψ windows during execution of the two-stage pipeline. For a given ψ, let us denote the i^{th} window by $\phi^i_\psi, 1 \leq i \leq \psi$.

Fig. 11 illustrates the methodology for identifying malicious windows during the training phase in Stage 1 of our malicious bitstream detection pipeline. The procedure for the selection of an optimum value of ψ is as follows:

1) For each window $\phi^i_\psi \in \{\phi^1_\psi, \phi^2_\psi, ..., \phi^\psi_\psi\}$, extract the array of bytes present in that window, for every benign and malicious bitstream;
2) Train ML classifiers separately for each window $\phi^i_\psi \in \{\phi^1_\psi, \phi^2_\psi, ..., \phi^\psi_\psi\}$; therefore, for ψ windows, number of ML classifiers that needs to be trained is ψ;
3) Evaluate the confidence score of each window based on the training data. The score lies in the range [0, 1] and it indicates the region of bitstream where malicious signatures are most likely to be present. If a window has a high confidence score, it is considered to be a malicious window and will be assessed in the next stage of the detection pipeline;
4) Choose the window with the highest confidence score, $\phi^{max}_\psi, \phi^{max}_\psi \in \{\phi^1_\psi, \phi^2_\psi, ..., \phi^\psi_\psi\}$. Next, arrange the remaining windows in increasing order of their confidence scores.

Generally, the confidence scores are evaluated on the test data [28]. However, we calculate the confidence scores based on the training data. This is because if the model fails to

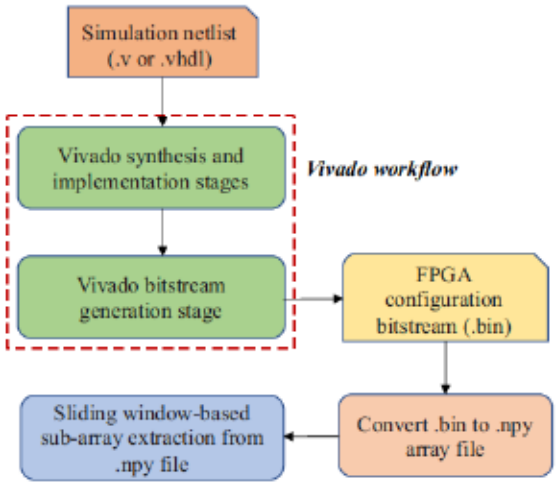


Fig. 10: Procedure for sub-array extraction from bitstreams.

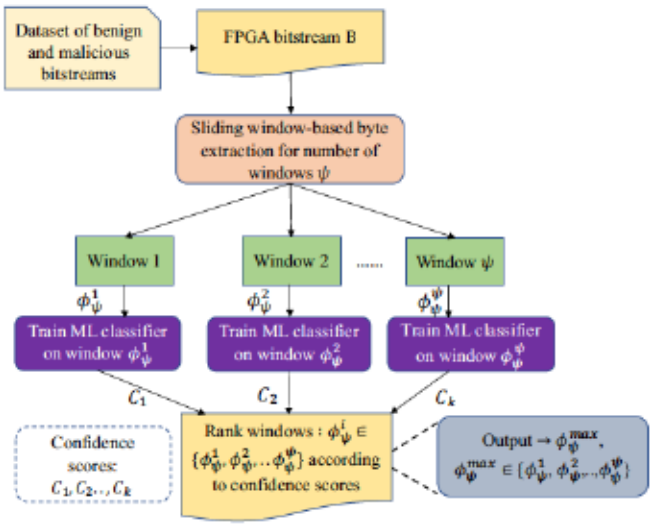


Fig. 11: ML-based malicious window extraction in Stage 1.

differentiate between benign and malicious windows in the training stage itself, it is expected to perform poorly during the inferencing stage. From the model training stage, we evaluate the confidence scores of each window and identify those windows where the model is able to efficiently learn the non-linear relationship between the window features and the malicious nature of the full FPGA bitstream. In other words, if the confidence score of window ϕ_ψ^i exceeds that of window ϕ_ψ^j, we can infer that ϕ_ψ^i contains more useful information in its bitstream than ϕ_ψ^j that correlates better to the criticality of the entire bitstream. The first stage of our malicious bitstream detection pipeline is, therefore, essential for the following reasons -

1) It enables pruning of insignificant bytes from the input bitstream and only passes critical windows to the next stage for diagnosis. In other words, a sample sent for training in Stage 2 should be precise with only useful information. This reduces model confusion in Stage 2 and hence, the model can converge using fewer ground-truth data samples;
2) Windows containing malicious signatures are prioritized;

Input: User-input bitstream B
Output: C_{mal}, W_{mal}
Read bitstream B
$flag \leftarrow 0$
$W_{mal} \leftarrow \emptyset$ /*Set of malicious windows*/
$C_{mal} \leftarrow \emptyset$ /*Confidence scores of malicious windows*/
Pre-trained models M_i, $1 \leq i \leq \psi$
Divide B into $\psi = 14$ windows /*Selected using hyperparameter tuning*/
Denote each window as ϕ_ψ^i
for i *in range* (ψ) **do**
  Run inferencing of ϕ_ψ^i on pre-trained model M_i
  if $M_i.predict(\phi_\psi^i)$ *is malicious* **then**
    Calculate C_i /*Confidence score of i^{th} window*/
    W_{mal}.append(ϕ_ψ^i)
    C_{mal}.append(C_i)
    $flag \leftarrow 1$
  end
end
if $flag == 0$ **then**
  Classify B as benign **else**
  Classify B as malicious
end
return C_{mal}, W_{mal}

Fig. 12: Pseudo-code for Stage 1 inferencing of bitstream.

this strategy can help in performing bitstream manipulations such as modifying LUT content and clock signal rerouting in order to remove malicious content prior to FPGA programming [29].

2) Inferencing

Fig. 12 describes Stage 1 inferencing for a given FPGA configuration bitstream. As shown in the algorithm, the input bitstream is divided into a set of ψ windows. We obtain $\psi = 14$ using hyperparameter tuning. We run inferencing on each window based on the pre-trained models. If a window is found to be malicious, it is appended to the list W_{mal} and the corresponding confidence score is also noted. After all the windows are analyzed, we check the status of the *flag*. If the value of *flag* is 0, we classify the bitstream as benign. Otherwise, we classify the bitstream to be malicious and proceed to the second stage for diagnosis.

From the algorithm, we can observe that Stage 1 prunes insignificant windows from the input bitstream. As a result, the sample sent for inferencing in Stage 2 contains only useful information regarding the presence of malicious circuits.

E. Stage 2: Multi-Class Classification

1) Training of Stage 2 DNN Model

From Stage 1 of the two-tier framework, we extract the set of malicious windows for the power-wasting circuits in our training dataset using the algorithm in Fig. 12. Next, we merge the malicious windows and plot the data-series representation of the concatenated window for each type of malicious circuit, namely the RO, latched RO, self-clocked, AES power waster, and shift register. We generate several images corresponding to the malicious power-wasting circuits by varying the number of instances in each of them. Next, we use the features from the images, namely the pixel values corresponding to each byte position to train our Stage 2 DNN-based multi-class classification model. Since our target is to detect the

five types of malicious power-wasting circuits, we train our model on five classes of images, each class corresponding to a malicious circuit. We perform random search to determine the best combination of hyperparameters that maximizes the performance of the model. These hyperparameters include the number of hidden layers, the number of neurons in each layer, the type of activation function, and the loss function. Results obtained from hyperparameter tuning are shown in Section V.

2) Inferencing with DNN Model

Next, we proceed to diagnose i.e., identify the type of malicious circuit after our pre-trained SVM model in the first stage classifies a user-input bitstream as malicious. We extract the windows with the maximum confidence scores from the input bitstream using the algorithm in Fig. 12 and convert each window to its data-series representation, in the form of image files. Then, we evaluate the image files using our pre-trained DNN model. The DNN model generates a label and a confidence score for the image files corresponding to the malicious windows. We also perform merging operation on all the malicious windows of the evaluated bitstream and perform inferencing on the image file corresponding to the resulting merged window using our pre-trained DNN model.

During inferencing, it is possible that the labels generated by the DNN model vary among the malicious windows. In other words, the model can predict incorrect labels for some of the image files. Therefore, it is necessary to address these cases and improve the generalization ability of the neural network architecture to correctly diagnose the type of malicious circuit implemented by a specific bitstream that has been classified as malicious in Stage 1. We develop an algorithm to diagnose a malicious bitstream based on the weighted confidence scores of its malicious windows. Let us say, the first stage generates malicious windows for a given bitstream . Each of the malicious windows has a confidence score ^{1}, . The value ' ' in the superscript of ^{1} signifies that the confidence scores are generated by the Stage 1 ML models. Now, Stage 2 generates a label ('RO', 'self-clocked RO', 'latched RO', 'AES', 'shift register') for each of the malicious windows with a confidence score of ^{2}, . Similarly, the value ' ' in the superscript of ^{2} denotes the confidence score generated from the second stage DNN model. Next, we merge the windows and generate an image file corresponding to the concatenated window. We get a label and confidence score for this window, namely ^{2} . In other words, the confidence score generated by the Stage 2 DNN model for the merged window is assigned to ^{2} . We can set the corresponding Stage 1 confidence score, namely ^{1} since we do not perform evaluations on the merged window in Stage 1. Now, we perform a weighted summation of the confidence scores for the generated labels as follows -

$$\sum_{=1} {}^{1} \quad {}^{2} \quad \text{if} \quad 1 \qquad (3)$$

In equation (3), represents the weighted confidence score corresponding to a particular label . The value lies between and , and represents those malicious windows that generate the same label . The score for the merged window, ^{2} , is added to the weighted confidence score for the particular label that matches the label for the merged window. After we get the weighted confidence scores for all the generated labels, we identify the label that gives the maximum confidence score. This label is, therefore, the power-wasting circuit implemented by the given bitstream. Hence, from the weighted confidence scores, we can efficiently diagnose the type of circuit implemented by a given bitstream.

F. Comparison with Existing ML-Based Classification Models

In [10], the authors study the voltage changes in the PDN of a multi-tenant FPGA and use this information to determine the type of computations performed by a co-tenant of the FPGA. In this work, the attacker configures a portion of the multi-tenant FPGA with a voltage fluctuation sensor, particularly the TDC. The TDC sensors are implemented using buffers and latch elements. Note that the delay through the buffer elements (propagation speed) is inversely (directly) proportional to the supply voltage of the PDN. Therefore, it is possible to evaluate the voltage fluctuations in the PDN due to computations on the multi-tenant FPGA. The sensor readings (or traces) captured by the TDCs are analyzed by the attacker using the following steps. The traces are transformed into images using a short-term Fourier Transform function. Finally, the image is passed through a ResNet50 inference model that classifies it into one of the several types of computations.

While [10] focuses on identifying cryptographic core and RO-based computations, it has not explored a number of other power-wasting circuits that can be used to attack cloud FPGAs. In our approach, we aim to cover a broader threat model, including conditionally active ROs and non-combinational ROs. Moreover, we explore power-wasting AES cores that are significantly different from normal cryptographic AES cores.

While our work and the work in [10] both make use of ML models to identify the type of computations, a key difference of our work from [10] is as follows. While [10] focuses on a side-channel analysis attack on the multi-tenant FPGA to extract information about the type of computation in a co-tenant module, our goal is to defend a multi-tenant FPGA from malicious circuit configuration and also identify the type of malicious circuit. We explore fundamental properties of malicious power-wasting circuits and apply our two-stage detection framework directly to the user input bitstream to immediately identify and block malicious circuits from being configured on the FPGA.

V. Results

A. Experimental Setup

We use Verilog to implement the malicious power-wasting circuits. We obtain the benign circuits from several benchmarks and OpenCore repository. Next, we synthesize, implement, and generate bitstreams corresponding to the benign and malicious circuits using Xilinx Vivado 2018.2. We implement the overall CNN-based classification framework using

. For image augmentation, we use the and tools from the Scikit-learn library [30]. We have trained the CNN using the *Adam* optimizer and with a learning rate of 0.00075. The number of training epochs is chosen to be 300. Dropout layers with a probability of and batch normalization layers have been added after every convolutional layer for better training accuracy and reduced overfitting.

We use and to build our two-stage malicious bitstream detection framework. The DNN model utilized in Stage 2 has five convolutional layers and five maxpooling layers, followed by two fully connected layers. We trained the DNN over epochs using the optimizer with a learning rate of . All the experiments are run on a 2.4 GHz Intel Xeon Gold 5115 CPU with 768 GB of RAM.

B. Evaluation Metrics

The following metrics are used to evaluate the effectiveness of our CNN model and our two-stage bitstream diagnosis framework, described in Section III and IV, respectively.

: Percentage of malicious bitstreams correctly classified as malicious.

: Percentage of benign bitstreams incorrectly classified as malicious.

Classification accuracy ( ): The ratio of the number of correct predictions to the total number of predictions. It is computed as: $\frac{-p}{p}$, where is the number of correct predictions and is the total number of predictions.

C. Detection of RO-like Patterns Using CNN-based Classification Framework

Recall that our dataset comprises of 95 image files generated from benign bitstreams and 80 image files generated from malicious bitstreams. The bitstreams are generated using the *write_bitstream* command available in Xilinx Vivado 2018.2. These bitstreams target the VU440 board. After the bitstreams are generated, they are converted to their corresponding CSV files (as illustrated in Fig. 5). Next, they are plotted as data-series representation and stored as image files. Since the dataset is of relatively small size, we use image augmentation to increase the size of our training dataset. We choose the and as our image augmentation method; this method shows the highest classification accuracy compared to other commonly used image augmentation techniques, such as the and the operations [22]. The and image augmentation techniques enable our model to extract meaningful features from the image representation of bitstreams. We performed several experiments to determine the best operation among - , , and . These operations are described below:

1) : Flip the image horizontally, either left or right;
2) : Flip the image vertically, either up or down;
3) : Flip the image along any axis or multiple axes.

We present the evaluation results in Table III. We note that gives us significantly better results when used as the image augmentation technique. After image augmentation, the size of our dataset increases to 314 image files. The number of benign image files and malicious image files are 146 and 168, respectively. We use -fold cross validation to evaluate multiple versions of train-test split. In our experiments, we select in accordance with common practice [31]. Therefore, parts are used to train our model and the remaining one part is used for model evaluation. We obtain an average training accuracy of 99.2% and an average test accuracy = 96.17% after 300 epochs, over the five folds. Also, we obtain = 97.02% and = 4.79%.

TABLE III: Exploring methods for image augmentation.

Technique	Training acc. (%)	Test acc. (%)
$flip_{lr}$	93.9%	95.7%
$flip_{ud}$	99.2%	96.4%
$flip$ (along axis $(1,2)$)	90.8%	87.4%

We next compare the proposed CNN-based detection approach with prior work on classification of FPGA bitstreams. The work in [14] uses a CNN to detect a particular hardware module (e.g. adder, subtractor, or multiplier) in a "one versus all" classification problem. To highlight that our proposed CNN architecture is tailor-made for detecting malicious bitstreams, we apply the CNN architecture described in [14] to our dataset. The results in Table IV show that our CNN architecture is carefully designed to be applicable to the security problem of malicious bitstream detection and authentication.

D. Malicious Bitstream Detection using AWS Tools

Users now have the freedom to upload their own design files on multi-tenant cloud FPGAs and request bitstream generation. However, during design rule check (DRC), the AWS cloud FPGAs reject circuits that contain combinational loops, e.g., ROs. Therefore, if an attacker attempts to configure the FPGA with a malicious grid of ROs, it will be instantly blocked by the AWS detection tools. However, in [13], the authors present examples of non-combinational ROs that cause oscillations but are not detected by the AWS. The description of these non-combinational ROs, namely the latched RO and the self-clocked RO are provided in Section III-A.

We implement the simple staged RO, latched RO, and self-clocked RO and submit the design checkpoints (.dcp) to AWS using Cloudshell. We observe the oscillating behavior of these circuits in the Vivado simulation stage. For bitstream generation on the AWS, we follow the following steps:

We first create an AWS Vivado 2020.2 Developer Amazon machine image. This will create an AWS instance (we use) with Vivado for building aws-fpga into .dcp;
We upload the HDL files corresponding to the non-combinational ring oscillators (ROs) and several RO variants using the Cloudshell;
Next, we invoke the Vivado Design Suite TCL shell;
We run a customized TCL script to synthesize and run implementation on .dcp, and request bitstream generation.

Table V shows the results that we obtained. The latched RO, self-clocked RO, and conditional RO remain undetected by the AWS tools. However, we observe a 'Combinational Loop' alert during AWS bitstream generation for all the staged ROs, i.e., these oscillators are detected during the AWS design rule check and prevented from requesting bitstream generation.

TABLE IV: Comparison with prior machine learning architecture used for FPGA bitstream classification [14].

Characteristics	CNN architecture in [14]	Proposed method
No. of conv. layers	One	Four
No. of FC layers	Two	Four
Training accuracy	91.6%	99.2%
Training loss	0.36	0.031
Test accuracy	85.7%	96.4%

TABLE V: Evaluation results on AWS platform.

Type of circuit	Detected during DRC check?	Bitstream generated?
Staged RO	✓	✗
Latched RO	✗	✓
Self-clocked RO	✗	✓
Conditional RO	✗	✓

E. Comparison of Proposed Framework with RE Method

Finally, we compare the proposed approach with [8]; see Table VI. Note that [8] requires reverse-engineering techniques to generate the technology-mapped netlist from the FPGA bitstream, and the tools are specific to a FPGA family. In addition, [8] is focused on determining how many malicious ROs can crash the FPGA. Therefore, a direct quantitative comparison with [8] is not feasible. Nevertheless, we note that the proposed approach is complementary to [8] for securing FPGAs from malicious bitstreams, and its goal of malicious bitstream detection can be synergistically combined with the vulnerability assessment provided by [8].

F. Sliding Window-based Malicious Bitstream Detection

1) Stage 1 Training

We proceed to detect malicious windows in the FPGA bitstreams using the sliding window-based approach. We begin the search for the optimized number of windows with and continue the procedure until we narrow down to the location(s) of malicious signatures in a given configuration bitstream. Table VII presents the evaluation results for the sliding window-based malicious bitstream detection framework. For every value of , we calculate the size of each window (in bytes). For example, if , we divide the bitstream into nine non-overlapping windows of size $\frac{128966372}{9}$. We choose the function to avoid losing significant information about malicious structures from the bitstream. Once we generate the bounding bytes of each such window for a given , we perform window-based byte extraction for all the benign and malicious bitstreams present in our experimental dataset. For our experiments, we generate 120 benign bitstreams and 116 malicious bitstreams. The benign bitstreams implement designs from ITC'99, ISCAS'85, and EPFL benchmarks. We also generate bitstreams that implement AES cores, microprocessor cores, and microcontrollers. The malicious bitstreams include a wide range of power-wasting circuits that are capable of causing voltage-based attacks and DoS attack on the FPGA. We randomly split our dataset into training and test datasets in the ratio of : . Random splitting of dataset ensures that our test dataset includes various power-wasting circuits such as non-combinational ROs and power-wasting AES cores. After random split, we have 89 benign bitstreams and 76 malicious bitstreams in the training dataset and 31 benign bitstreams and 40 malicious bitstreams in the test dataset. We define the following terms and as follows:

1) : The lower bound on the number of windows to divide the bitstream into;
2) : The upper bound on the number of windows to divide the bitstream into; this also indicates the number beyond which the ML classifier fails to identify malicious signatures in the sub-windows with acceptable classification accuracy.

Choosing the value of : Our target is to locate the window in the bitstream that contains most of the malicious features and distinguishes itself from a benign bitstream. Therefore, we choose .

Choice of ML Classifier: We perform -fold cross-validation to select a suitable ML classifier to train our dataset of benign and malicious bitstreams. We evaluate the following four supervised learning models: (1) Random Forest (RF), (2) Support Vector Machine (SVM), (3) XGBoost (XGB), and (4) Multilayer Perceptron (MLP).

We select in accordance with common practice [31]. Therefore, parts are used to train our model and the remaining one part is used for model evaluation. Fig. 13 illustrates the F1-scores of the four ML classifiers after -fold cross-validation. We observe that SVM generates the highest F1-score of among all the tested ML classifiers. The best hyperparameters chosen for the SVM are as follows - regularization parameter (, kernel: RBF. A high F1-score for the SVM model can be attributed to the ability of the model to handle high-dimensional datasets and accurately classify FPGA bitstreams based on their relevant features [32]. FPGA bitstreams are often represented by a large number of features (in this case, features), which is a challenge for traditional classification algorithms. Therefore, we utilize the SVM classifier to evaluate the benign and malicious bitstreams in Stage 1 of our proposed framework.

For a given number of windows , we train number of SVM classifiers on the dataset of

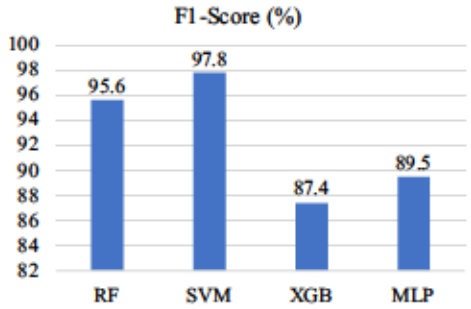


Fig. 13: F1-score of ML models after -fold cross-validation.

TABLE VI: Comparison with RE method [8].

Characteristics	Proposed method	[8]
Dataset	Bitstreams plotted as data-series	Netlist representation of bitstreams
RE used?	No	Yes
Loop-free ROs analyzed?	Yes	No
Framework	CNN-based feature extraction	*icebox_vlog*, *yosys* tools
Evaluation results	TPR_{mal} = 97.02% FPR_{mal} = 4.79%	No. of ROs required to crash the FPGA = 1920

TABLE VII: Evaluation of confidence scores to determine the value of .

ψ	n	C_1	C_2	C_3	C_4	C_5	C_6	C_7	C_8	C_9	C_{10}	C_{11}	C_{12}	C_{13}	C_{14}	C_{15}	C_{16}	C_{17}	C_{18}	C_{19}	C_{20}
6	21494396	76.92	73.56	73.56	75.4	93.2	75.4	-	-	-	-	-	-	-	-	-	-	-	-	-	-
9	14329596	42.36	75.42	77.38	84.62	75.38	88.36	84.62	72.43	42.36	-	-	-	-	-	-	-	-	-	-	-
12	10747198	42.36	75.38	73.85	95.38	73.85	73.85	93.85	93.85	73.85	72.31	67.32	43.78	-	-	-	-	-	-	-	-
14	9211884	43.85	75.83	84.62	72.31	95.38	73.85	73.85	73.85	97.69	43.78	86.15	42.86	73.85	73.85	-	-	-	-	-	-
15	8597759	45.38	73.85	75.38	73.85	75.38	93.85	73.85	75.38	67.43	97.69	93.85	73.85	73.85	75.38	68.46	-	-	-	-	-
20	6448319	42.36	43.85	75.38	76.92	75.38	93.85	73.85	73.85	73.85	72.43	73.85	83.08	73.85	86.15	75.83	73.85	73.85	68.46	73.85	73.85

TABLE VIII: Malicious windows detected in the first stage of the two-tier classification pipeline.

Window	*RO_7*	*RO_53*	*RO_105*	*DF_9*	*DF_15*	*DF_101*	*SC_81*	*SC_117*	*AES_10*	*AES_20*	*SR_4*
ϕ_{14}^{3}	0.11	0.984	0.984	0.11	0.11	0.11	0.984	0.981	0.895	0.895	0.165
ϕ_{14}^{5}	0.815	0.815	0.815	0.815	0.815	0.815	0.815	0.815	0.25	0.815	0.193
ϕ_{14}^{9}	0.943	0.979	0.979	0.095	0.095	0.979	0.979	0.979	0.095	0.095	0.987
ϕ_{14}^{11}	0.987	0.99	0.99	0.933	0.723	0.99	0.99	0.99	0.037	0.037	0.25
ϕ_{14}^{14}	0.302	0.838	0.838	0.92	0.849	0.603	0.348	0.499	0.116	0.116	0.703

bitstreams of size $\frac{128966372}{}$. Now, we calculate the confidence scores of each classifier on the sub-windows. Therefore, we get a set of confidence scores, one score for each sub-window. Our target is to identify those specific windows in the entire bitstream that have the highest confidence score, and are therefore most likely to contain malicious signatures. We stop our search for the suitable choice of once the confidence scores for all the sub-windows start saturating and stop showing significant increment. From Table VII, we observe that choosing generates an appropriate number of malicious windows, with higher confidence scores compared to other values of . Therefore, for the first stage of our bitstream diagnosis pipeline, we choose for sub-array extraction of an user-input configuration bitstream. Thus, the number of input features to the SVM classifier is equal to the window size $\frac{128966372}{14}$ = bytes. The confidence score is maximum for the window $\frac{9}{14}$ and is equal to 97.69%. The window $\frac{5}{14}$ has the next highest confidence score of 95.38%. These results help us to identify the specific windows in a given bitstream where malicious signatures are most likely to be present. We use these observations to further analyze the type of malicious circuit implemented by this bitstream in Stage 2 of our bitstream diagnosis framework.

2) Inferencing

The test dataset includes the following 40 types of malicious bitstreams (‘ ’ denotes the number of instances):

1) 8 bitstreams corresponding to RO and conditional RO circuits (denoted by _);
2) 6 bitstreams implementing variants of the latched RO circuit (denoted by _);
3) 7 bitstreams corresponding to the self-clocked RO circuit (denoted by _);
4) 4 bitstreams that implement rounds of AES-based power waster, where (denoted by _);
5) 7 bitstreams corresponding to a chained shift register circuit (denoted by _);
6) 8 bitstreams corresponding to -input XOR-based glitch amplification circuits, .

We achieve for all the malicious bitstreams in our test dataset using the pre-trained sliding window-based ML classifier. Furthermore, we evaluate the confidence scores of every malicious window detected for a particular malicious bitstream. These scores are used in the second stage of our classification pipeline to diagnose the type of circuit implemented by the bitstream. The experimental results are shown in Table VIII. We show inferencing results for eleven malicious bitstreams from our test dataset and present evaluation results for only those windows that detect malicious windows with a confidence score . The shaded cells in Table VIII correspond to the malicious windows that are detected with a high confidence score during the first stage. All the evaluated glitch amplification circuits were identified as malicious by the specific SVM classifier that was trained on the window $\frac{3}{14}$; the average confidence score generated being %. Furthermore, the malicious windows returned by the Stage 1 ML model for an AES power waster circuit and a glitch amplification circuit are similar, thereby implying that our model can efficiently detect XOR gates placed at unusual locations of a circuit, which contribute to glitching. We observe that all the malicious bitstreams in our test dataset are detected by our Stage 1 model, thereby justifying the effectiveness of the proposed method.

G. Evaluation Results for Multi-Class Classification of Malicious Bitstreams

Once we identify all the malicious windows in the bitstream using the 14 SVM classifiers, we plot the Numpy arrays for each malicious window and store them as images. The images corresponding to the evaluated power-wasting circuits are shown in Fig. 14 (a)-(d). The -axis label indicates the byte position and the -label represents the pixel value of that byte position. From Fig. 14, we can visually differentiate the malicious circuits from one another. Each pixel value is in the range [ ]. Next, we normalize the array of the image pixel values to be between [0, 1] by specifying the parameter “ ”. Finally, we reshape the images to dimension before feeding them to the pre-trained DNN multi-class classifier. The image files fed to the DNN model are illustrated in Fig. 14 (e)-(h). Each - dimensional image having normalized input values act as input to the first convolutional layer of the DNN model. Note that the labels generated by the model are one-hot encoded values and hence, do not require normalization.

Furthermore, we concatenate all the malicious windows and use the resulting image for inferencing. For example, if a

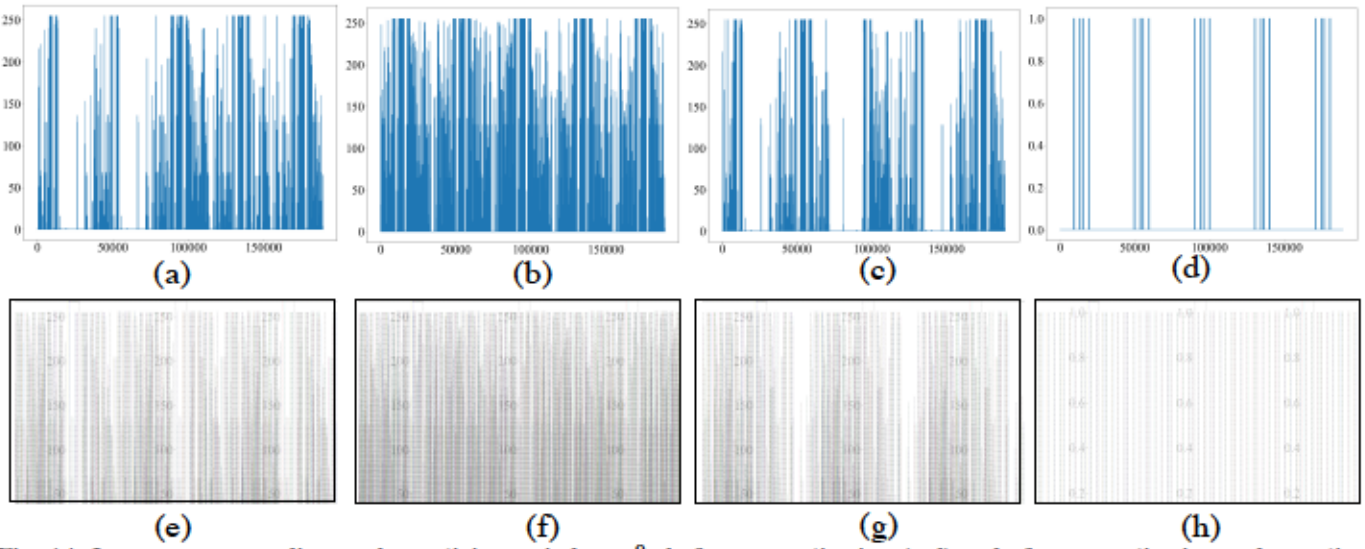


Fig. 14: Images corresponding to the malicious window $\frac{9}{14}$ before normalization (a-d) and after normalization and rescaling (e-h) for the RO, Latched RO, Self-clocked RO, and AES power waster circuits, respectively.

malicious bitstream has malicious windows, the number of images passed through the DNN classifier for inferencing is . The DNN classifier predicts a label for each of the images, along with a confidence score. Due to a well-trained DNN model (we achieve a training accuracy of , over 200 epochs), the DNN classifier predicts the label with a confidence score of '1'. We use Equation (3) to calculate the confidence score of the generated labels for the images. Finally, we identify the label which produces the highest weighted confidence score.

Using the weighted confidence score-based approach, we correctly diagnose out of the malicious bitstreams in our test dataset. In other words, our DNN classifier correctly identifies the type of malicious circuit implemented by a user-input bitstream with a classification accuracy of .

H. Comparison Between Proposed Detection Frameworks

The efficiency of partitioning-based bitstream detection framework in identifying a broad variety of power-wasting circuits compared to prior detection methods is demonstrated in Table IX. The model used in [33] is chosen as the baseline framework. The CNN-based malicious bitstream detection framework proposed in Section III is denoted by [34]. The current partitioning-based ML-DNN framework is denoted as . For fair comparison, the evaluation dataset used for testing the above three frameworks consists of 120 benign bitstreams and 116 malicious bitstreams. The percentage of glitch amplification circuits correctly identified as malicious is denoted by . We observe that achieves a = % i.e., all the malicious bitstreams in our evaluation dataset are correctly identified as malicious. This means that actual malicious activities are detected by , while fewer false alarms are generated. Moreover, we are able to detect the glitch amplification circuits in the test dataset with an accuracy of %. Overall, the proposed bitstream partitioning-based framework achieves a speedup compared to the baseline framework [33] to detect a malicious user-input bitstream on a 2.4 GHz Intel Xeon Gold 5115 CPU with 768 GB of RAM.

I. Extending the framework for multiple FPGA families

Note that the proposed technique of detecting malicious bitstreams can be extended to other FPGA versions and families. Only the training dataset changes, while the bitstream detection framework remains unchanged. For the Xilinx Kintex Ultrascale (KU085) FPGA, we generate similar number of benign and malicious bitstreams as were generated for VU440. We generate = 8 by hyperparameter tuning for a KU085 bitstream (size: 48251520 bytes). Evaluation on a test dataset of 31 benign and 40 malicious bitstreams yields: = 95% and = 9.6%. The time overhead associated with the evaluation on a user-input bitstream is 1.3 minutes.

VI. Discussion

The primary motivation behind our proposed approach is to detect malicious RO-based and non-RO-based circuits that have the potential of causing power- and voltage-based attacks on FPGAs [6], [25], [13]. Currently, our ML-based framework is able to detect any bitstream that contains RO-

TABLE IX: Performance comparison of partitioning-based malicious circuit detection with prior detection frameworks.

Framework	No. of input features	Selected Hyperparameters	$TPR_{mal}(\%)$	$FPR_{mal}(\%)$	$A_c(\%)$	f (%)	Time (s)	Speedup (w.r.t. Baseline [33])
Baseline [33]	128966372	max_depth = 60, no. of estimators = 100	72.5	19.31	76.05	37.5	247	–
$ML1_{CNN}$ [34]	50176	Optimizer: Adam, lr = 7.5e(-3)	90	6.4	91.54	62.5	235.3	1.05
$ML2_{DNN}$	9211884	C = 10, kernel: RBF	**100**	**3.2**	**97.18**	**100**	**88.3**	**2.79**

like signatures, including ROs used for benign purposes. In order to distinguish between malicious ROs and benign ROs (implemented in PUFs and TRNGs), we first need to collect a sufficient amount of training data for both RO types. Next, we can extend our two-tier bitstream diagnosis framework to learn features from the RO variants and assess the malware criticality of the RO i.e., identify whether the RO belongs to a malicious category or whether it is implemented in benign circuits. We will explore this scenario in our future work.

Note that the proposed methodology offers several advantages over signature-based approach for malicious bitstream detection. First, signature-based approaches are limited to detecting only known malicious bitstreams. Attackers can easily evade this defense by devising previously unseen malicious bitstreams. However, supervised ML algorithms can detect both known and unknown malicious bitstreams, making them more robust against such attacks. Second, ML-based methods are more cost-effective for implementation, in terms of requiring less computing power than signature-based methods.

The attacker may use additional circuits (along with the malicious components) to obfuscate the high power consumption of the malicious components. To address this adversarial assumption, we can apply the proposed two-tier malware detection framework in conjunction with run-time anomaly detectors based on long short term memory (LSTM) to monitor the power consumption of the FPGA and successfully detect an obfuscated attack [35]. We do not go into the details of this adversarial assumption as it is beyond the scope of this paper.

VII. Conclusion

First, we have presented a CNN-based malicious bitstream detection framework for FPGAs. By utilizing specific patterns from the data-series representation of bitstreams, the proposed CNN model uses negligible knowledge about the actual bitstream structure to achieve a highly accurate classification of benign and malicious bitstreams. Next, we have further broadened our threat model by evaluating a wide variety of power-wasting circuits and glitch amplification circuits that are a threat to cloud computing FPGAs. We have presented a two-tier malware detection framework that identifies a malicious bitstream and also diagnoses the type of malicious circuit implemented by that bitstream. We have demonstrated the effectiveness of our methodology for multiple FPGA families.

References

[1] Amazon, "Amazon EC2 F1 Instance," https://go.aws/3ENtUj9, 2021.
[2] K. Eguro *et al.*, "FPGAs for trusted cloud computing," in *FPL*, 2012.
[3] M. Zhao *et al.*, "FPGA-based remote power side-channel attacks," in *2018 IEEE Symposium on Security and Privacy (SP)*, 2018.
[4] F. Schellenberg *et al.*, "An inside job: Remote power analysis attacks on FPGAs," in *DATE*, 2018.
[5] K. M. Zick *et al.*, "Sensing nanosecond-scale voltage attacks and natural transients in FPGAs," in *FPGA*, 2013.
[6] D. Gnad *et al.*, "Voltage drop-based fault attacks on FPGAs using valid bitstreams," *Proc. FPL*, pp. 1–7, 2017.
[7] T. M. La *et al.*, "FPGADefender: Malicious self-oscillator scanning for Xilinx UltraScale + FPGAs," *ACM TRETS*, 2020.
[8] D. Gnad *et al.*, "Checking for electrical level security threats in bitstreams for multi-tenant FPGAs," in *FPT*, 2018.
[9] Xilinx, "Ultrascale architecture configuration," https://bit.ly/3yyxvQ9 .
[10] M. Gobulukoglu *et al.*, "Classifying computations on multi-tenant FPGAs," in *DAC*, 2021, pp. 1261–1266.
[11] J. Krautter *et al.*, "FPGAhammer: Remote voltage fault attacks on shared FPGAs, suitable for DFA on AES," *IACR TCHES*, 2018.
[12] F. Schellenberg *et al.*, "An inside job: Remote power analysis attacks on FPGAs," *IEEE Design & Test*, vol. 38, no. 3, pp. 58–66, 2021.
[13] T. Sugawara *et al.*, "Oscillator without a combinatorial loop and its threat to FPGA in data centre," *Electronics Letters*, 2019.
[14] S. Mahmood *et al.*, "IP core identification in FPGA configuration files using machine learning techniques," in *Proc. IEEE ICCE-Berlin*, 2019.
[15] N. Vashistha *et al.*, "Trojan scanner: Detecting hardware trojans with rapid sem imaging combined with image processing and machine learning," in *ISTFA*, 2018.
[16] K. Hasegawa *et al.*, "Hardware Trojans classification for gate-level netlists based on machine learning," in *IOLTS*, 2016.
[17] S. Shukla *et al.*, "RNN-based classifier to detect stealthy malware using localized features and complex symbolic sequence," in *ICMLA*, 2019.
[18] J. Jeon *et al.*, "Dynamic analysis for IoT malware detection with convolution neural network model," *IEEE Access*, 2020.
[19] T. Inoue *et al.*, "Designing hardware Trojans and their detection based on a SVM-based approach," in *ASICON*, 2017, pp. 811–814.
[20] K. Hasegawa *et al.*, "Trojan-feature extraction at gate-level netlists and its application to hardware-Trojan detection using random forest classifier," in *ISCAS*, 2017, pp. 1–4.
[21] Xilinx, "AWS cloud," https://bit.ly/3Vgx1aT.
[22] B. Zoph *et al.*, "Learning data augmentation strategies for object detection," *CoRR*, 2019.
[23] F. Rodrigues *et al.*, "Beyond expectation: Deep joint mean and quantile regression for spatiotemporal problems," *IEEE TNNLS*, vol. 31, 2020.
[24] X. Glorot *et al.*, "A semantic matching energy function for learning with multi-relational data," *Machine Learning*, 2013.
[25] G. Provelengios *et al.*, "Power wasting circuits for cloud FPGA attacks," in *FPL*, 2020.
[26] K. Matas *et al.*, "Power-hammering through glitch amplification – attacks and mitigation," in *FCCM*, 2020, pp. 65–69.
[27] T. Wollinger *et al.*, "Security on FPGAs: State-of-the-art implementations and attacks," *ACM TECS*, p. 534–574, 2004.
[28] S. Enomoro *et al.*, "Learning to cascade: Confidence calibration for improving the accuracy and computational cost of cascade inference systems," in *AAAI*, vol. 35, no. 8, 2021.
[29] K. D. Pham *et al.*, "Bitman: A tool and API for FPGA bitstream manipulations," in *DATE*, 2017.
[30] Scikit-learn, "Machine learning in *python*," https://bit.ly/3OzdLBZ .
[31] SKlearn, "Grid search with cross validation," https://bit.ly/3hEHNnQ .
[32] T. Joachims, "Text categorization with support vector machines: Learning with many relevant features," in *ECML*. Springer, 1998.
[33] R. Elnaggar *et al.*, "Learning malicious circuits in FPGA bitstreams," *IEEE Trans. CAD*, 2022.
[34] J. Chaudhuri and K. Chakrabarty, "Detection of malicious FPGA bitstreams using CNN-based learning," in *ETS*, 2022.
[35] M. Villarreal-Vasquez *et al.*, "Hunting for insider threats using LSTM-based anomaly detection," *IEEE Trans. DSC*, vol. 20, no. 1, 2023.

Jayeeta Chaudhuri (Member, IEEE) received the B.E. degree in ECE from Jadavpur University, India, in 2020. She is currently pursuing the Ph.D. degree in ECE at Duke University, Durham, USA. Her current research interests include hardware security, security of analog ICs using machine learning-based models, and analog Trojans.

Krishnendu Chakrabarty (Fellow, IEEE) received the B.Tech. degree from the Indian Institute of Technology Kharagpur, Kharagpur, India, in 1990, and the M.S.E. and Ph.D. degrees from the University of Michigan at Ann Arbor, Ann Arbor, MI, USA, in 1992 and 1995, respectively. He is the Fulton Professor of Microelectronics in the School of Electrical, Computer and Energy Engineering, part of the Ira A. Fulton Schools of Engineering at Arizona State University, Arizona, USA.