# PUFGAN: Embracing a Self-Adversarial Agent for Building a Defensible Edge Security Architecture

JinYi Yoon and HyungJune Lee
Department of Computer Science and Engineering
Ewha Womans University, Seoul, South Korea
Email: hyungjune.lee@ewha.ac.kr

***Abstract*—In the era of edge computing and Artificial Intelligence (AI), securing billions of edge devices within a network against intelligent attacks is crucial. We propose *PUFGAN*, an innovative machine learning attack-proof security architecture, by embedding a self-adversarial agent within a device fingerprint-based security primitive, public PUF (PPUF) known for its strong fingerprint-driven cryptography. The self-adversarial agent is implemented using Generative Adversarial Networks (GANs). The agent attempts to self-attack the system based on two GAN variants, vanilla GAN and conditional GAN. By turning the attacking quality through generating realistic secret keys used in the PPUF primitive into system vulnerability, the security architecture is able to monitor its internal vulnerability. If the vulnerability level reaches at a specific value, PUFGAN allows the system to restructure its underlying security primitive via feedback to the PPUF hardware, maintaining security entropy at as high a level as possible.**

**We evaluated PUFGAN on three different machine environments: Google Colab, a desktop PC, and a Raspberry Pi 2, using a real-world PPUF dataset. Extensive experiments demonstrated that even a strong device fingerprint security primitive can become vulnerable, necessitating active restructuring of the current primitive, making the system resilient against extreme attacking environments.**

## I. Introduction

As the growth of edge computing technology accelerates, computer systems with distributed computation and networking capabilities have been developed. The widespread adoption and deployment of edge devices – devices such as routers and routing switches which provide entry points into enterprise or service provider core networks – across various categories of connected home devices, medical devices, and healthcare, has raised critical security threats. These threats include the potential for breaches of privacy data, or the ability for malicious system access and control. Insulin pumps or smart locks can easily be hacked, due to the relatively vulnerable authentication systems used in these devices [24]. Securing edge devices with low-end computing, storage, and networking is essential to their success.

To protect edge devices from potential threats in distributed network environments, the use of a lightweight yet strong cryptographic function in the system architecture level is essential. To mitigate the computational overhead incurred by software-based cryptography, hardware fingerprint-based approaches that can significantly reduce the burden of computation have been proposed as one of the most promising security primitives for low-end devices. In particular, the use of *physically unclonable functions (PUFs)* [18] is an innovative way of exploiting the unique hardware characteristics of devices, which are created during their integrated circuit (IC) fabrication process, as a strong device authentication signature.

Various sophisticated PUF predecessors have been proposed [1], [14], including public PUF (PPUF) that can effectively be integrated with end-to-end data encryption for IoT devices. However, these approaches can be vulnerable to adversarial attacks based on machine learning algorithms such as Support Vector Machine (SVM), Random Forest (RF), and Deep Learning (DL) techniques [4], [6], [12]. Attackers may succeed in recovering a single secret key from public information using brute force, and continue to obtain more data sample pairs of secret key-to-public information; that is, challenge and response pairs. At a certain point, even complex cryptographic relationships can be approximated with a reasonable precision, and the system can be hacked.

In this paper, we propose an innovative attack-proof security methodology and architecture, exploiting a deep generative model based on Generative Adversarial Networks (GANs) [7], [8] using a dedicated hardware based on PPUF. We embed one of the most advanced deep learning-based attack models into a self-adversarial agent for internal vulnerability monitoring. When the vulnerability level is found to exceed a specified limit, the system generates feedbacks, and restructures its hardware fingerprint security primitive, in order to maintain as high a security level as possible.

Our work addresses two main questions and provides concrete answers based on extensive empirical experiments: 1) how vulnerable can a system become as security-related information starts being exposed externally; and 2) when is it important to restructure a system's security primitive, given that it may potentially be exposed to the most advanced machine learning attack environments.

Without assuming any predefined security features, our proposed self-adversarial agent comprises two neural networks: a generator network, and a discriminator network. The generator network aims to learn and generate realistic challenge-response pairs (CRPs), while the discriminator network learns to distinguish real CRPs from fake CRPs. The two networks compete with each other, and converge at a

This work was supported by Samsung Research Funding & Incubation Center of Samsung Electronics under Project Number SRFC-IT1803-00.

specified equilibrium point, at which the generator produces real but hidden CRPs that can totally destroy the fingerprint-based cryptography.

To the best of our knowledge, this paper is the first to apply a self-adversarial agent using a state-of-the-art GAN framework with a fingerprint-based security architecture to make the system self-adaptive to harsh, dynamic environments. The main contributions can be summarized as follows:

- We develop one of the most intelligent AI-based hacking efforts using a PPUF-based fingerprint security primitive, and conduct extensive empirical experiments.
- Our work converts knowledge about the internally obtained vulnerability into a design asset for building a highly defensible edge security architecture. We exploit one of the most sophisticated AI algorithms, GAN, to design a self-adversarial agent that performs self-vulnerability diagnosis.
- We suggest simple yet efficient methods to restructure the underlying fingerprint security primitive for maintaining a vulnerability that is as low as possible, while adapting in such a way as to produce stable security resilience and attack prevention.

## II. Related Work

Our work is part of the discipline of security systems in which device fingerprint-based cryptography is tightly coupled with self-adaptive security enhancement.

### A. Device Fingerprint-Based Security

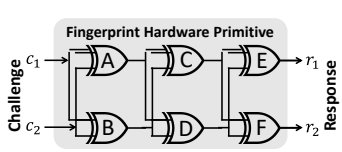
In comparison with complicated software cryptography-based security approaches, a device itself can be used as a highly distinct *fingerprint*, distinguishable from other similar devices, ensuring lightweight yet strong security identification. Some physical measures from accelerometer sensors [22], 3D magnetic sensors [10], network behavioral profiling [2], or RF signals [5], [15] can be turned into unique identification signatures. Specifically, the radio fingerprint-based security approach uses its radio chip as a unique security primitive without requiring additional dedicated hardware [3], [23].

A more generalized innovative security primitive, physically unclonable functions (PUFs) [18], has recently been introduced, and has attracted considerable attention in the security community. The variability that inevitably occurs during the IC hardware fabrication process can be used as an innate authentication method for a device. A more advanced PUF variation, public PUF (PPUF) [1], has been shown to be a practical way to implement end-to-end data encryption in edge networks [14].

However, this fingerprint-based security, that has been shown to have higher security than other classic approaches, may also be vulnerable to advanced intelligent attacks such as impersonation or machine learning techniques [4], [6].

### B. Self-Adaptive Security

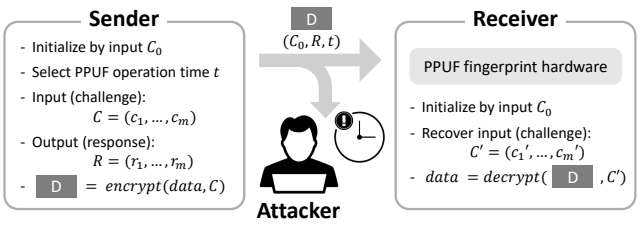
Current computer systems operate in dynamic, heterogeneous, and distributed network environments, and their security architecture tends to incorporate self-adaptive security components within the system [20]. To identify whether an event or action observed in a system is malicious, model-based and machine learning-based methodologies are used to detect malicious behaviors, on the assumption that they arise mostly from external sources. Such model-based detection [17], [19] exploits an explicit form of signature or rule that can differentiate malicious from normal behavior. Machine learning-based methods [16], [21], on the other hand, use supervised or unsupervised learning techniques for the detection of malicious intrusions. Once malicious behaviors are detected via either methodology, these algorithms update security policies or reconfigure application parameters. More closely related to our work, some recent studies [9], [25] have tried to identify some complex patterns using GAN-based approaches.




(a) PPUF logic

| Gate | Input 1 Delay ($\mu s$) | Input 2 Delay ($\mu s$) |
|---|---|---|
| A | 4.62 | 64.02 |
| B | 32.48 | 515.79 |
| C | 32.94 | 2.05 |
| D | 64.97 | 4.05 |
| E | 4.10 | 64.73 |
| F | 65.78 | 1030.47 |

(b) Gate delay measurement of PPUF logic




(c) End-to-end data encryption procedure

Fig. 1. Device fingerprint security architecture using public information and a PPUF-based communication procedure

Self-adaptive security has not been applied to edge security architecture. Further, self-adversarial agent-based security with vulnerability awareness has not been studied. Our work takes a new research direction, combining a deep neural network-driven self-adversarial security adaptation algorithm with fingerprint-based edge device security architecture.

## III. System Architecture

We present *PUFGAN*, a self-adversarial security architecture for edge devices. Although device fingerprint-based cryptography provides new opportunities for designing lightweight yet reliable security systems, intelligent machine learning attacks may reproduce even unknown secret keys from partial fingerprint-related information. Our work aims to integrate a machine learning-based attack model that can be located internally or externally within a network, to produce a device fingerprint security architecture. The self-adversarial agent is to continuously diagnose vulnerability for a system and then restructure the device fingerprint hardware at a certain point if necessary for maintaining the vulnerability as low as possible.

### A. Motivation

Public PUF is known to be a lightweight yet reliable device fingerprint-based security primitive for edge devices.

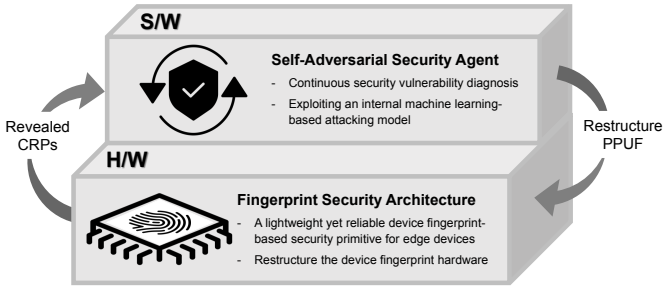



Fig. 2. Self-adversarial agent over PPUF-based security architecture

The challenge – the input of fingerprint hardware – is used as a secret key. The response – the corresponding output – is used as the public key. As illustrated in Fig. 1(a), the unique output response is determined by three factors: 1) the inherent PPUF hardware characteristics; 2) the input challenge; and 3) the measurement time of output.

Even if the elements of the response – the measurement time as well as the gate delay table (Fig. 1(b)) – that can be captured from reverse engineering are public information, only the PPUF device owner can quickly recover the original challenge for a specific item of information. Attackers who have obtained the gate delay table for the target device need to simulate all possible transition cases along each stacked stage in order to achieve secret key recovery, a process which takes exponentially increasing time.

A legitimate user can be distinguished using the time gap between the PPUF execution time and the PPUF simulation time. This innovative idea can be applied to end-to-end encrypted data transmission in networks, as in Fig. 1(c). A sender that wants to deliver data to a legitimate receiver uses the receiver's public information to encrypt and then sends the data. Only a legitimate user, who owns its dedicated PPUF logic can recover the original challenge, which is the secret key used and discarded at the sender side, and use it to decrypt the data.

Even though the challenge in a challenge-response pair (CRP) is discarded in order to guarantee security in end-to-end encryption, attackers with high-performance computing capability may collaborate with each other in order to recover the original challenge from public response information using a brute-force approach.

If more challenge-response pairs become disclosed after attackers' repeated hacking efforts, the underlying mapping distribution between challenge and response, the core of the security primitive, may be captured at some point using machine learning approaches. Once a probabilistic guess for a possible challenge can damage the entire end-to-end encryption, the overall security architecture may be jeopardized. Thus, to make any security architecture highly reliable, it is crucial for the system to constantly monitor its own security vulnerabilities against intelligent machine learning attacks.

### B. Threat Model

Our self-adversarial security system aims to prevent potential attacks. It is assumed that attackers are capable of recovering an original challenge from an exposed response by calculating and iterating the response for every possible challenge. The recovered challenge-response pairs can be aggregated at an attacking host. The attacking host may apply any of a number of machine learning techniques to infer the original challenges from even unknown responses, based on possible patterns extracted from the available CRPs, and can finally disable the entire PPUF fingerprint-based cryptography.

### C. Self-Adversarial Security Architecture

To defend an edge security architecture from highly intelligent attacking efforts, we take a bold approach by implementing a self-adversarial agent within the architecture itself. The self-agent tries to extract the underlying mapping distribution from current exposed challenge-response pairs, and starts generating realistic challenge-response pairs that are not even used yet, but turn out to be real.

Our security architecture involves three phases. We first produce a generic model, which learns from a set of training CRPs, and then identify the relationship between challenge and response during a training phase. The trained model is then used to generate all possible challenge-to-response pairs. The vulnerability of the current security architecture is constantly monitored. If the vulnerability level exceeds a certain limit, the currently used fingerprint primitive is restructured, in order to maintain the highest entropy in the fingerprint-based cryptography.

## IV. GAN-based Security System

When subjected to intelligent machine learning-based attacks, some crucial cryptographic patterns, even using one of the strongest fingerprint-based security primitives, can be disclosed. To tackle potential intelligent threats on edge devices, we exploit state-of-the-art Generative Adversarial Networks (GANs), one of the most effective deep generative models, to implement an attack-proof agent in the edge security architecture (Fig. 2).

We design a self-adversarial agent within a PPUF-based security architecture to perform three main tasks: 1) to generate realistic CRPs for a given PPUF security architecture, as close as possible to the data distribution of the actual CRPs; 2) to analyze system vulnerabilities based on synthetic accuracy; and 3) to restructure the underlying security primitive at times of high vulnerability.

### A. Background on GAN

Machine learning has been widely used to make predictions from data. Learning techniques can be categorized into two classes: supervised and unsupervised. Supervised learning models use labelled training data and predict the labels of unseen testing data (e.g., classification or regression). On the other hand, unsupervised learning identifies previously unknown patterns without using pre-existing labels (e.g., clustering).

Generative Adversarial Networks (GANs) are a promising class of generative models for unsupervised learning. Using

both real and fake training data, a generic model is trained to capture a general distribution and at the same time, generate realistic fake data. GANs can also discover unknown or unexplored features. This ability can be used to tackle the problem of limited training data via data augmentation, in a persistent way. Since GAN can be used to emulate nearly any kind of data, it has great potential for applications in a variety of fields.

GAN operates through competition between a generator network and a discriminator network. The key function of the generator is to fabricate fake data that should be as real as possible, by mimicking real data distributions, while the discriminator is used to distinguish fake from authentic data. Each neural network should be trained to have the lowest loss with respect to its respective purpose and converge to its best performance. The generator starts producing fake data almost indistinguishable from real ones, so that it can even deceive the discriminator. At the same time, the discriminator is optimized to classify those data correctly.

This system is a minimax problem, in which the discriminator tries to maximize the probability of not being deceived, whereas the generator tries to minimize the probability by making its best effort to deceive the discriminator. The minimax problem can effectively be solved with various deep learning architectures.

### B. Self-Adversarial Agent with GANs

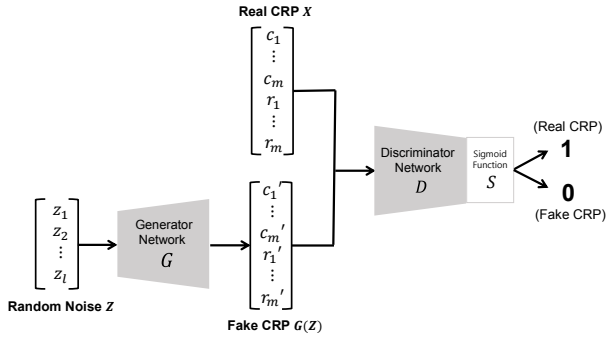
We instantiate a self-adversarial agent by adding a GAN architecture into the PPUF security framework. A PPUF-based GAN model is used to generate realistic CRPs, of which few challenge-to-response pairs are unused yet authentic, and therefore able to break the fingerprint cryptography. We pose a worst-case question: if past CRP samples for a given PPUF logic are exposed due to system malfunction or data breach, how vulnerable is the system?

We develop two neural networks: a Generator Network and a Discriminator Network in the self-adversarial agent. A generator network $G$ produces fake CRP $X_{fake}$ from input noise samples $Z$, such that $X_{fake} = G(Z)$. As the generator network, $G$, is trained, $G(Z)$ follows the same distribution as the training data of real CRP, $X_{real}$. The discriminator network $D$ examines CRP samples to discriminate whether they are real or fake. Both real and fake CRP are fed as training data into the discriminator, so it learns to classify them into two classes: 1 for real CRP, or 0 for fake CRP, using supervised learning.

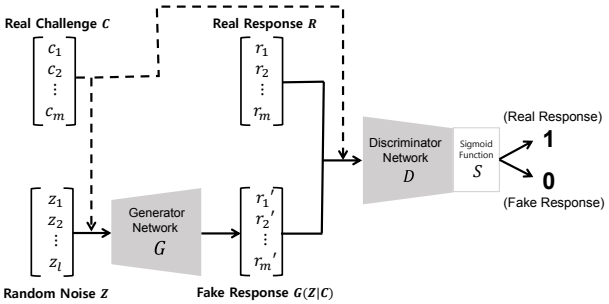
To investigate how differences in GAN architecture can affect hacking efficiency, we implement two GAN variants: *Vanilla GAN* [8] and *conditional GAN* [13].

*1) Vanilla GAN with PPUF (v-PUFGAN):* Vanilla GAN is the original GAN, the simplest form with which the two players, generator and discriminator, play a minimax game.

We construct a challenge-to-response pair as a single training datum, $X = [c_1, \cdots c_m, r_1, \cdots, r_m]^T$ where the challenge of $[c_1, \cdots, c_m]$ for its corresponding response of $[r_1, \cdots, r_m]$ is fed into the vanilla GAN input, as in Fig. 3(a).




(a) *v-PUFGAN* (Vanilla GAN coupled with PPUF)




(b) *c-PUFGAN* (Conditional GAN coupled with PPUF)

Fig. 3. Two GAN architectures for hacking the PPUF fingerprint primitive

The objective function of the vanilla GAN is as follows:

$$\min_G \max_D V(D,G) = \mathbb{E}_{X\sim p_{data}(X_{real})}[logD(X)] + \mathbb{E}_{Z\sim p_Z(Z)}[log(1-D(G(Z)))] \quad (1)$$

where $\mathbb{E}$ is an estimator, and $p_{data}(X_{real})$ is the true distribution of the CRP data, while $p_Z$ is the distribution of input noise variables, and $D(G(Z)$ is a fake response $X_{fake}$.

*2) Conditional GAN with PPUF (c-PUFGAN):* The conditional GAN is a conditional version of the original vanilla GAN, created by feeding additional data into both generator and discriminator.

To make a PPUF-based cryptographic security totally vulnerable, the self-adversarial agent should predict the correct responses for all possible challenges. We use the challenge information as the condition, and construct a conditional GAN model within the agent, as shown in Fig. 3(b).

By setting the condition of challenge into the original objective function of vanilla GAN as in Eq. (1), the objective function of the conditional GAN can be defined as follows:

$$\min_G \max_D V(D,G) = \mathbb{E}_{X\sim p_{data}(R_{real})}[logD(X|C)] + \mathbb{E}_{Z\sim p_Z(Z)}[log(1-D(G(Z|C)))] \quad (2)$$

where $p_{data}(R_{real})$ is a true distribution of real responses for a given challenge $C$, and $G(Z|C)$ is fake response $R_{fake}$, produced the generator network $G$.

Our PUFGAN architecture includes a training process with the following steps:

- We generate a fake CRP based on $G(Z)$ for the vanilla GAN and $G(Z|C)$ for the conditional GAN, which is a random sample with a random initial distribution in the generator network $G$.

- After both the real and fake CRPs are fed into the discriminator, $D$, a binary classification neural network, we calculate the cross-entropy loss based on the original correct labels of 0 for fake CRP and 1 for real CRP.
- The generator $G$ also calculates the cross-entropy loss where the loss is created when the previously generated fake CRP is correctly classified by the discriminator as fake.
- The loss information propagates back to its respective neural network and the network learns to adjust its weight parameters to minimize the loss through a gradient descent optimization, such as an Adam optimizer [11].
- We iterate the above steps until convergence, when the generator $G$ operates closely as $D(G(Z)) \sim 1$ where $G(Z) = X_{fake}$, while at the same time the discriminator $D$ does closely as $D(X_{fake}) \sim 0$ and $D(X_{real}) \sim 1$.

The algorithm is described in detail in Algorithm 1.

**Algorithm 1** PUFGAN: GAN-based Self-Adversarial Agent

**Input:** *revealed CRPs*
**Parameter:** *vulnerability threshold* $\theta_{risk}$

**while** TRUE **do**
  $vulnerability$ = *self-diagnosis* (revealed CRPs);
  **if** $vulnerability \geq \theta_{risk}$ **then**
    Restructure the fingerprint-based hardware device;
  **end if**
**end while**

**Function** *self-diagnosis* (revealed CRPs)
  **for** # of training epochs **do**
    // I. Generate fake data
    $Z \sim U(a, b)$: input noise samples;
    $X_{fake} \leftarrow G(Z)$: generator produces fake CRP from $Z$;
    // II. Train the discriminator model $D$
    $X_{real} \leftarrow$ samples from real CRP $X$;
    Calculate $loss(X_{real}, 1)$ and $loss(X_{fake}, 0)$;
    Update $D$ to minimize $loss(X_{real}, 1) + loss(X_{fake}, 0)$;
    // III. Train the generator model $G$
    Calculate $loss(X_{fake}, 1)$;
    Update $G$ to minimize $loss(X_{fake}, 1)$;
  **end for**
  // IV. Generate fake data and estimate the vulnerability
  $Z \sim U(a, b)$: input noise samples;
  $X_{fake} \leftarrow G(Z)$: generator produces fake CRP from $Z$;
  Calculate the $vulnerability$ reflecting the accuracy of $X_{fake}$;
  **return** $vulnerability$;
**end Function**

Due to the conflict of interests between the generator and discriminator sides, the convergence in the minimax game is often difficult to acheive. To tackle the imbalance between the two players, our PUFGAN makes one player optimize more frequently than the other by having $k$ inner iterations, as suggested in [8].

### C. Architecture Restructuring via Vulnerability Diagnosis

The security vulnerability can be considered to be the probability that a system can be hacked. It is related to how well our PUFGAN-based agent can discover new, correct challenge-to-response pairs, which have been neither exposed nor previously used. We quantify the vulnerability measure based on PUFGAN's prediction quality in terms of precision and recall, as described in Sec. V.

Time $t$ when the security level becomes vulnerable is determined as follows:

$$v(t) \geq \theta_{risk} \text{ for } t \in \{t_1, t_2, t_3, \ldots\} \tag{3}$$

where $v(t)$ is the vulnerability measure at time $t$, and $\theta_{risk}$ is the risk level threshold.

Once the vulnerability measure at time $t$, $v(t)$, exceeds a certain risk level, $\theta_{risk}$, we rearrange the underlying PPUF structure in three different ways:

- Shifting the PPUF operation time $t$ to a different operation time $t'$ under the same physical PPUF logic
- Rearranging each logic gate by shuffling only their physical connections, with the same width and height structure
- Composing totally different logic gates into a structure with different widths and heights

This architecture restructuring procedure makes the PPUF-based security architecture resilient against dynamic and potentially malicious environments.

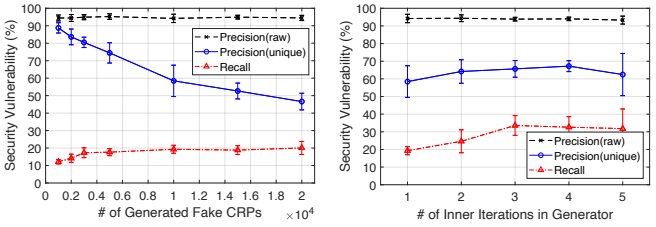
## V. Experiments

We validated our PUFGAN architecture by implementing two variants: vanilla GAN, called *v-PUFGAN*, and conditional GAN, called *c-PUFGAN*, in the TensorFlow framework along with a real-world PPUF architecture and dataset. We ran our algorithm in three different machine environments: 1) Google Colab with TensorFlow 1.14.0; 2) desktop PC with Intel Core i5-7500 CPU, 8GB RAM and Kaby Lake GT2 GPU with TensorFlow 1.13.1 on 64-bit Windows 10 OS; and 3) Raspberry Pi 2 Model B with TensorFlow 1.13.1 on Raspbian OS.

### A. Dataset

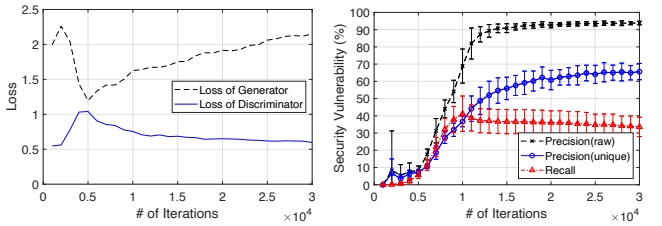
We obtained a real-world PPUF dataset by generating challenge-to-response pairs using PPUF simulations from real-world PPUF FPGA-based fingerprint hardware and its gate delay measurement [14]. Using gate delay measurement information for each of the 57 actual gates in the FGPA, various PPUF architectures were composed with width $w$ and height $h$. To differentiate between PPUF architectures, we identified specific PPUF types as $P^{c}_{w \times h}(t)$, where width is $w$, height $h$, gate inter-connection $c$, and operation time $t$. For example, $P^{c_1}_{11 \times 5}(700,000)$ describes a PPUF architecture consisting of 55 gates with a width of 11 and a height of 5, based on interconnection $c_1$ at an operation time of 700,000 time units, where one time unit is approximately 2.5 ns in the PPUF FPGA implementation.

The number of CRPs depends solely on the width $w$ of the PPUF, since the number of challenges is given by $2^w$. A width $w$ of 11 and a height $h$ of 5 of the PPUF architecture at an operation time of 700,000 time units were used, unless otherwise noted. Amongst a total of 2048 challenge-to-response pairs, we chose half at random as training data and the other half as testing data at the same operation time case. CRPs in the testing data were never present in the training data. Each learning run was affected by the randomness generated by the weight initialization, the training/testing data split, and



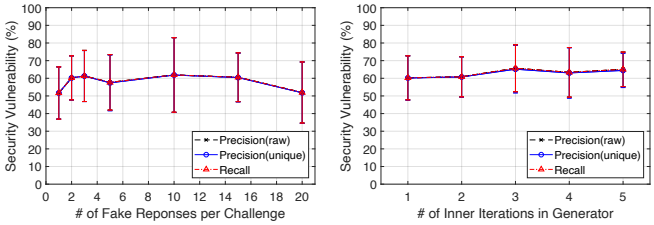

(a) # of generated fake CRPs (b) $k$ inner-iterations in the generator

Fig. 4. Parameter settings in v-PUFGAN with $t = 700,000$ time units




(a) Cross-entropy loss (b) Security vulnerability

Fig. 5. Loss of generator and discriminator with respect to the number of iterations in v-PUFGAN with $t = 700,000$ time units, 10,000 generated fake CRPs, and three inner iterations in the generator




(a) # of generated fake responses per challenge (b) $k$ inner-iterations in the generator

Fig. 6. Parameter settings in c-PUFGAN with $t = 700,000$ time units

optimization, so we ran 10 experiments and calculated the average performance with the standard deviation, wherever applicable.

### B. Performance Metric

We evaluated the security performance of our PUFGAN architecture primarily based on vulnerability. Vulnerability was measured with respect to prediction-wise hacking capability based on three criteria: *Precision(raw)*, *Precision(unique)*, and *Recall*.

*Precision(raw)* denotes the precision of true positives in response that is generated for a challenge in the testing data, which are not any challenge from the training data. This measure indicates how reliably and consistently a self-adversarial agent predicts a correct response for unknown challenge.

*Prediction(unique)* denotes a sanitized version of *Precision(raw)* produced after discarding redundantly generated CRPs. This measure indicates the precision of true positives in response per unique challenge from the testing data.

*Recall* denotes the recall of true positives in response for all the challenges in the testing data. The metric of $(1-$ *Recall*$)$ is the false negative ratio; the ratio of undiscovered challenge-to-response pairs in the testing data.

### C. Vulnerability Performance

The configurations of the generator networks and discriminator networks used in the experiments were as follows. Both networks included a single hidden layer with 128 neurons. The input random noise vector $Z$ comprised $100 \times 1$ (where $l = 1$) uniformly random variables over $U[-1, 1]$. A ReLU activation was used at the hidden layer, and an Adam optimizer with a learning rate of 0.001 was used.

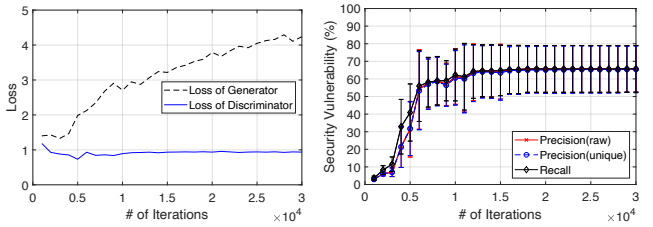
After numerous experiments, we chose a run time of 30,000 iterations, which was sufficient to ensure that the networks converged. This run time was used for all experiments, unless otherwise noted. The batch size was varied from 200 to 1000, and steady-state performance was observed at batch sizes of 600 and beyond, so a batch size of 600 was selected.

*1) v-PUFGAN:* First, we investigated how security vulnerability in terms of precision and recall was affected by the number of fake CRPs generated in v-PUFGAN, as shown in Fig. 4(a). As v-PUFGAN generates more fake CRPs, the number of correct predictions of CRPs increased up to 95% in terms of *Precision(raw)*. *Recall* increased slightly, and reached a steady-state point of 20%, whereas *Precision(unique)* decreased slightly. These results show that generating 10,000 fake CRPs may be sufficient to discover unknown CRPs using v-PUFGAN.

The discriminator tended to perform somewhat better than the generator, showing the imbalance between two players. To reduce this performance imbalance, we made the generator learn more frequently by allowing $k$ inner iterations. Fig. 4(b) shows that choosing a learning frequency ratio of 3:1 between the generator and the discriminator (that is, $k = 3$) resulted in effective performance, increasing the security vulnerability.

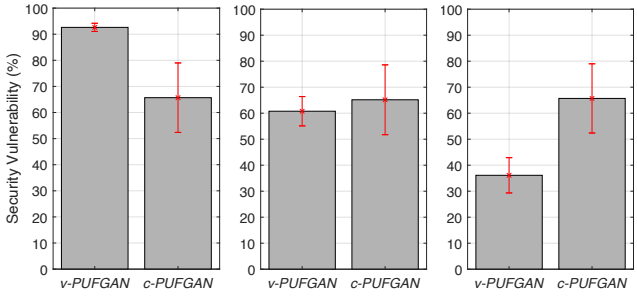
We investigated how cross-entropy loss and security vulnerability changed with the number of optimization iterations. As the number of iterations increased, the loss at both the generator and the discriminator converged, as shown in Fig. 5(a), and security vulnerability increased, reaching a high saturation point with respect to all three metrics near 20,000 iterations, a value that was chosen on the basis of preceding experiments as shown in Fig. 5(b). *Precision(raw)* reached 93.8%, indicating that v-PUFGAN successfully predicted correct challenge-to-response pairs. *Recall* rose above 33.6%, implying that almost 1 out of 3 unknown challenge-to-response pairs was discovered using v-PUFGAN.

*2) c-PUFGAN:* We examined changes in the vulnerability of c-PUFGAN depending on parameter settings and optimization iterations. As shown in Fig. 6(a), as the number of fake responses per challenge increased from 1 to 20, c-PUFGAN made the system quite stably vulnerable beyond 51%. This phenomenon arises because c-PUFGAN can generate the response for a specific challenge upon iterating over all unknown challenges (which are covered in the testing data). We select two fake response generations per challenge, an approach which is effective when the resulting vulnerability reaches around 60%. There was again an imbalance between the performance of the generator and that of the discriminator, as shown in Fig. 6(b), and it was found that selecting a learning




(a) Cross-entropy loss (b) Security vulnerability

Fig. 7. Loss of generator and discriminator with respect to the number of iterations in c-PUFGAN, with $t = 700,000$ time units, two fake responses generated per challenge, and three inner iterations in the generator




(a) *Precision(raw)* (b) *Precision(unique)* (c) *Recall*

Fig. 8. Security vulnerability with respect to GAN type and prediction metric

frequency ratio of 3:1 between generator and discriminator (where $k = 3$) achieved the highest vulnerability, as with v-PUFGAN.

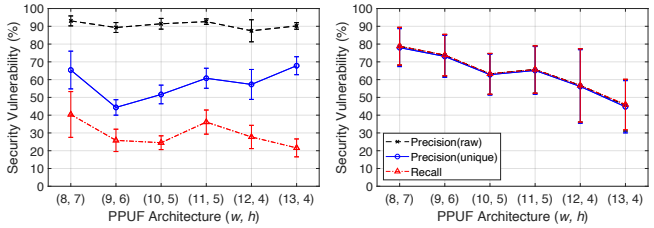
The number of optimization iterations also had an effect on the system. Fig. 7 shows that c-PUFGAN successfully made the system vulnerable with up to 66% in all three metrics, as two players at c-PUFGAN converged. We found 20,000 iterations to be effective for c-PUFGAN.

*3) v-PUFGAN vs. c-PUFGAN:* Using the previously described parameter settings for v-PUFGAN and c-PUFGAN, we compared vulnerability performance of the two architectures in terms of *Precision(raw)*, *Precision(unique)*, and *Recall* in Fig. 8. v-PUFGAN shows better performance for correctly predicting true response positives compared to false response positives for unknown challenges in the testing data. This finding indicates that v-PUFGAN is good at finding a set of frequently appearing CRPs correctly and consistently.

c-PUFGAN shows excellent performance in terms of *Recall*, achieving more than 65%. This observation implies that c-PUFGAN is good at finding unknown CRPs over all possible unknown challenges within the testing data.

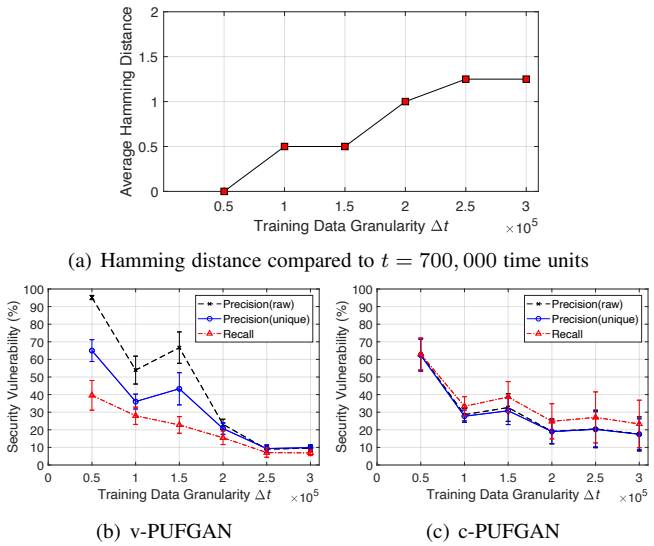
Since the two attack model variants can work together in a complementary way, a security system should perform more advanced prevention against machine learning-based attacks such as these.

*4) Effect of PPUF Structure Configuration:* We validated the generality of PUFGAN over various PPUF structures with different width and height configurations, as shown in Fig. 9. Both architectures showed a similar trend in drop of vulnerability as the complexity of the PPUF structure increased, with higher width for a similar number of gates in a PPUF architecture. However, it still achieved high vulnerability.




(a) v-PUFGAN (b) c-PUFGAN

Fig. 9. Security vulnerability for v-PUFGAN and c-PUFGAN with regard to PPUF structure configuration, with $t = 700,000$ time units




(a) Hamming distance compared to $t = 700,000$ time units

(b) v-PUFGAN (c) c-PUFGAN

Fig. 10. Hamming distance and vulnerability with respect to the neighboring training data at $\pm\Delta t$

This observation indicates that our PUFGAN-based attacking models are generally valid for various PPUF structure configurations.

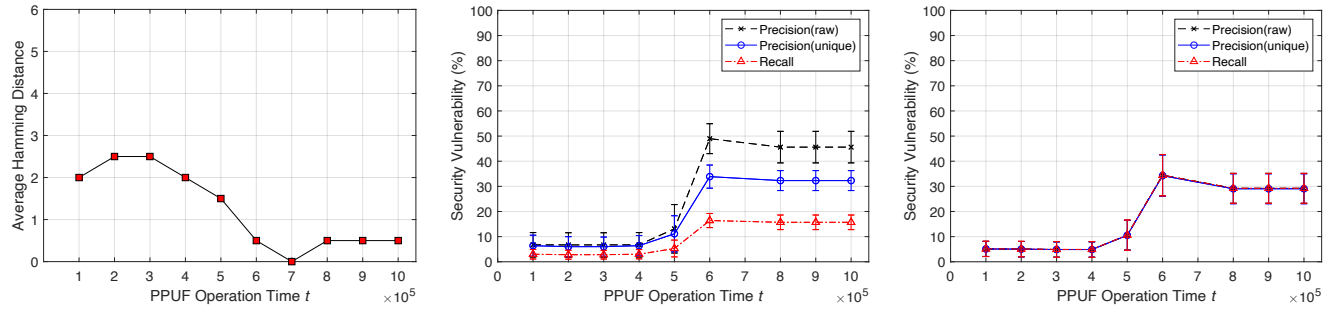
### D. Effect of Training Data Granularity

We investigated the way in which training data granularity affected testing performance. We used the training data which was at intervals $\pm\Delta t$ from the test operation time $t$, where $t = 700,000$ time units.

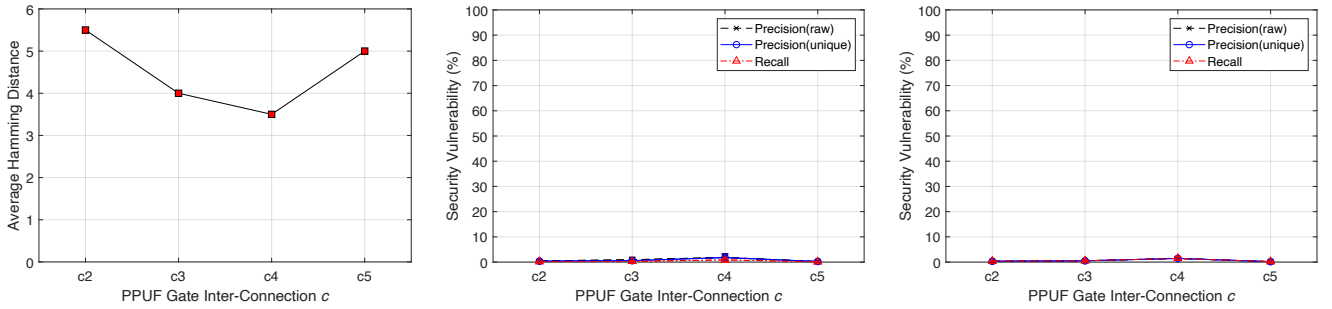
To investigate how the response space varied as the granularity, $\Delta t$, increased, we calculated the average Hamming distance on response per challenge between the training data and the testing data. As can be seen in Fig. 10(a), the average Hamming distance increased with the interval $\Delta t$. Therefore, we concluded that security vulnerability decreased with increasing distance between the training condition and the testing condition, as shown in Figs. 10(b) and 10(c).

### E. Effect of PPUF Restructuring

We examined how PPUF restructuring affected system vulnerability. We used three approaches: 1) different PPUF operation times using the same PPUF logic; 2) different gate inter-connections using the same structure configuration of the same gates; and 3) different structure configurations with different widths and heights.




(a) Hamming distance compared to $t = 700,000$ (b) The effect of operation time $t$ in v-PUFGAN (c) The effect of operation time $t$ in c-PUFGAN



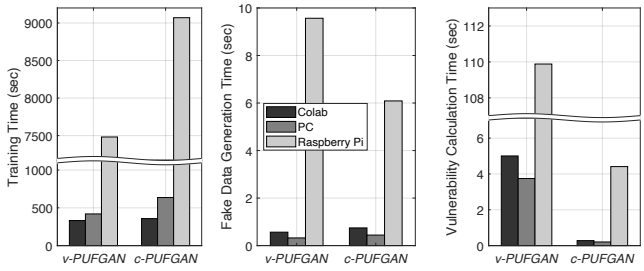

(d) Hamming distance compared to $c_1$ (e) The effect of connection type $c$ in v-PUFGAN (f) The effect of connection type $c$ in c-PUFGAN

Fig. 11. Effect of PPUF restructuring with respect to the operation time $t$ and the gate inter-connection type $c$

To examine the effects of different operation times in the execution (i.e., testing) other than $t$ in the training, we applied the PUFGAN model learned at $t = 700,000$ time units to PPUF operation at operation times from $t' = 100,000, \cdots, 1,000,000$ time units. To investigate how the response space varied over different PPUF operation times for the same challenge space with the same PPUF structure, we calculated the average Hamming distance on response per challenge, as shown in Fig. 11(a). As the PPUF operation time for testing was shifted farther from the original time for training, the response space became increasingly different. As shown in Figs. 11(b) and 11(c), the vulnerability of both v-PUFGAN and c-PUFGAN dropped significantly as the training data granularity increased. This observation indicates that when the system makes a PPUF security primitive while operating in a very different operation environment, its vulnerability is significantly reduced.

We also investigated how different inter-connections affected security vulnerability by testing with inter-connections $c'$ different from the $c_1$ used at the time of constructing the PUFGAN models. As shown in Fig. 11(d), the average Hamming distance on the response space over different inter-connection other than $c_1$, was significantly different, even more so than when using different PPUF operation times. Due to the relatively different CRP patterns, security vulnerability was very low, reaching almost 0% using both v-PUFGAN and c-PUFGAN, as shown in Figs. 11(e) and 11(f).

With respect to different structure configurations, since the PUFGAN model learned using a specific structure cannot be applied to a structure with different width and height configurations, due to the difference in model size, we can also expect that restructuring the PPUF configuration makes the system totally restored in terms of vulnerability.




(a) Training time (b) Fake CRP generation time (c) Vulnerability calculation time

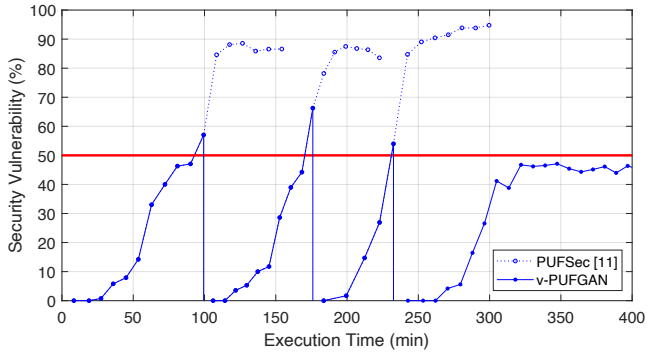
Fig. 12. Execution time in terms of training, fake CRP generation, and vulnerability calculation using Google Colab, desktop PC, and Raspberry Pi 2

Thus, if a system is allowed to perform a significant physical change in the security primitive, the system can become more resilient against various attacks.

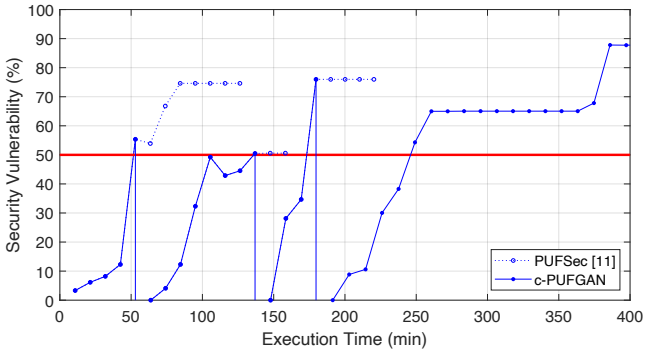
### F. Computational Feasibility

We investigated the computation feasibility of the system using different machine environments: Google Colab, a desktop PC, and a Raspberry Pi 2 Model B, as shown in Fig. 12. We measured execution time separately for training, fake CRP generation, and vulnerability diagnosis. Although it took over two hours to train both v-PUFGAN and c-PUFGAN on the Raspberry Pi 2, the performance was adequate. Even assuming a pessimistic scenario, a computation time of two to three hours may be acceptable for learning attacking behaviors for edge devices. For fake CRP generation and vulnerability diagnosis, all three machine environments running each PUFGAN model had computationally feasible performance.

Lastly, we investigated vulnerability dynamics over the execution time in PUFGAN compared to PUFSec [14], which does not consider any kind of adversarial attack based on PPUF security primitive. For both v-PUFGAN and c-




(a) Vulnerability of *Precision(raw)* over time with v-PUFGAN




(b) Vulnerability of *Recall* over time with c-PUFGAN

Fig. 13. Performance dynamics of security vulnerability for w/o vs. w/ restructuring via PUFGAN using a risk level of 50% as an example case

PUFGAN, we enforced PPUF architecture restructuring if the monitored vulnerability exceeded a risk level of 50%, sequentially applying three different restructuring methods with different operation times, different inter-connections, and different structure configurations. As can be seen in Fig. 13, our PUFGAN effectively maintained a healthy condition below the risk level over the execution time. However, the counterpart work, PUFSec, quickly degraded, showing significantly more vulnerable security performance.

## VI. Discussion

The findings of this research raise several interesting points.

### A. v-PUFGAN vs. c-PUFGAN

v-PUFGAN, which is based on vanilla GAN, shows high prediction quality in terms of raw precision. The large gap between raw precision and sanitized precision indicates that a set of correct challenge-response pairs are consistently and repeatedly generated. This observation implies that attackers with strong intelligence can obtain significant amounts of stable and consistent security primitive information.

c-PUFGAN, which is based on conditional GAN, is good at discovering unknown challenge-to-response pairs. This particular type of attacking behavior can be used as a way of quickly disabling the entire security primitive.

Thus, irrespective of the type of adversarial attack, a fingerprint-based device security system needs to be designed with these kinds of attack behaviors in mind. It should perform necessary active operations, such as security restructuring, to maintain as high a security entropy as possible.

### B. Restructuring Security Primitive

When security vulnerability reaches a certain risk level, the underlying security primitive needs to be restructured. The restructuring interval is highly dependent on the amount of revealed challenge-to-response information and the intensity of adversarial intelligence. Powerful attackers can obtain a large number of true challenge-to-response pairs. The more information on the challenge-to-response structure near the current operation environment is obtained, the earlier a system becomes vulnerable, even under the same adversarial attack model.

Attackers can take another approach, increasing the complexity of neural networks in the generator and the discriminator. A stronger adversarial attack model can achieve a specific vulnerability level earlier.

Therefore, the restructuring interval needs to be fine-tuned to prevent computationally more intensive attacks.

### C. Implications

Although it is usually very computationally intensive for potential attackers to obtain even one real challenge-to-response sample in a brute-force manner, we raise the question: if this kind of event continuously happens in a device fingerprint-based security primitive PPUF, which is known for strong security, what happens? A lucky guess or even more intelligent attacks on the original challenge used as the secret key from a publicly accessible response can quickly jeopardize a computer system or a group of systems in the network. Our work provides critical guidelines for designing attack-proof security systems robust against extreme attacking environments:

- Monitor real-time vulnerability using an internal agent mimicking various attacks via approaches such as machine learning, etc.
- Restructure or at least shuffle the underlying security primitives, either based on system vulnerability or in a simple periodic manner, changing the operation environment.

## VII. Conclusion

We have presented an innovative machine learning-based attack-proof security methodology and architecture based on a deep generative model using GANs. We implemented two GAN variants – vanilla GAN and conditional GAN – in the PPUF security framework within a self-adversarial agent. The self-adversarial agent performs internal vulnerability monitoring and feeds back about necessary security restructuring to the PPUF, maintaining a security entropy as high as possible over time.

This work provides a basis for the development of a variety of more advanced machine learning-based attack models, which can make a security system architecture stronger in hostile, evolving network environments. Our fundamental adversarial methodology can also be used as an effective way of attacking more general asymmetric cryptography using the secret key-to-public key relationship. In future work, it would be interesting to consider the energy factor, which critically affects the edge authentication performance.

## REFERENCES


[1] N. Beckmann and M. Potkonjak. Hardware-based public-key cryptography with public physically unclonable functions. In *International Workshop on Information Hiding*, pages 206–220. Springer, 2009.

[2] B. Bezawada, M. Bachani, J. Peterson, H. Shirazi, I. Ray, and I. Ray. Iotsense: Behavioral fingerprinting of IoT devices. *arXiv preprint arXiv:1804.03852*, 2018.

[3] V. Brik, S. Banerjee, M. Gruteser, and S. Oh. Wireless device identification with radiometric signatures. In *Proceedings of the 14th ACM international conference on Mobile computing and networking*, pages 116–127. ACM, 2008.

[4] B. Danev, H. Luecken, S. Capkun, and K. El Defrawy. Attacks on physical-layer identification. In *Proceedings of ACM WiSec*, 2010.

[5] B. Danev, D. Zanetti, and S. Capkun. On physical-layer identification of wireless devices. *ACM Computing Surveys (CSUR)*, 45(1):6, 2012.

[6] S. Dey, N. Roy, W. Xu, R. R. Choudhury, and S. Nelakuditi. Accelprint: Imperfections of accelerometers make smartphones trackable. In *NDSS*, 2014.

[7] I. Goodfellow. Nips 2016 tutorial: Generative adversarial networks. *arXiv preprint arXiv:1701.00160*, 2016.

[8] I. J. Goodfellow, J. Pouget-Abadie, M. Mirza, B. Xu, D. Warde-Farley, S. Ozair, A. Courville, and Y. Bengio. Generative Adversarial Networks. *arXiv e-prints*, page arXiv:1406.2661, Jun 2014.

[9] R. E. Hiromoto, M. Haney, and A. Vakanski. A secure architecture for iot with supply chain risk management. In *2017 9th IEEE International Conference on Intelligent Data Acquisition and Advanced Computing Systems: Technology and Applications (IDAACS)*, volume 1, pages 431–435. IEEE, 2017.

[10] H. Ketabdar, P. Moghadam, B. Naderi, and M. Roshandel. Magnetic signatures in air for mobile devices. In *Proceedings of ACM MobileHCI*, pages 185–188, 2012.

[11] D. P. Kingma and J. Ba. Adam: A method for stochastic optimization. *arXiv preprint arXiv:1412.6980*, 2014.

[12] H. Maghrebi, T. Portigliatti, and E. Prouff. Breaking cryptographic implementations using deep learning techniques. In *International Conference on Security, Privacy, and Applied Cryptography Engineering*, pages 3–26. Springer, 2016.

[13] M. Mirza and S. Osindero. Conditional generative adversarial nets. *arXiv preprint arXiv:1411.1784*, 2014.

[14] S.-Y. Park, S. Lim, D. Jeong, J. Lee, J.-S. Yang, and H. Lee. PUFSec: Device fingerprint-based security architecture for Internet of Things. In *IEEE Conference on Computer Communications (INFOCOM)*, 2017.

[15] L. Peng, A. Hu, J. Zhang, Y. Jiang, J. Yu, and Y. Yan. Design of a hybrid RF fingerprint extraction and device classification scheme. *IEEE Internet of Things Journal*, 6(1):349–360, 2018.

[16] M. Salehie, L. Pasquale, I. Omoronyia, R. Ali, and B. Nuseibeh. Requirements-driven adaptive security: Protecting variable assets at runtime. In *2012 20th IEEE International Requirements Engineering Conference (RE)*, pages 111–120. IEEE, 2012.

[17] A. C. Squicciarini, G. Petracca, and E. Bertino. Adaptive data protection in distributed systems. In *Proceedings of the third ACM conference on Data and application security and privacy*, pages 365–376. ACM, 2013.

[18] G. E. Suh and S. Devadas. Physical unclonable functions for device authentication and secret key generation. In *2007 44th ACM/IEEE Design Automation Conference*, pages 9–14. IEEE, 2007.

[19] C. Tsigkanos, L. Pasquale, C. Menghi, C. Ghezzi, and B. Nuseibeh. Engineering topology aware adaptive security: Preventing requirements violations at runtime. In *2014 IEEE 22nd International Requirements Engineering Conference (RE)*, pages 203–212. IEEE, 2014.

[20] G. Tziakouris, R. Bahsoon, and M. A. Babar. A survey on self-adaptive security for large-scale open environments. *ACM Computing Surveys (CSUR)*, 51(5):100, 2018.

[21] G. Tziakouris, C. J. M. Gomez, and R. Bahsoon. Securing cloud users at runtime via a market mechanism: A case for federated identity. In *IEEE International Conference on HPCC*, pages 221–228, 2014.

[22] T. Van Goethem, W. Scheepers, D. Preuveneers, and W. Joosen. Accelerometer-based device fingerprinting for multi-factor mobile authentication. In *International Symposium on Engineering Secure Software and Systems*, pages 106–121. Springer, 2016.

[23] T. D. Vo-Huu, T. D. Vo-Huu, and G. Noubir. Fingerprinting wi-fi devices using software defined radios. In *Proceedings of the 9th ACM Conference on Security & Privacy in Wireless and Mobile Networks*, pages 3–14. ACM, 2016.

[24] Y. Yang, L. Wu, G. Yin, L. Li, and H. Zhao. A survey on security and privacy issues in internet-of-things. *IEEE Internet of Things Journal*, 4(5):1250–1258, 2017.

[25] C. Zhao, M. Shi, Z. Cai, and C. Chen. Research on the open-categorical classification of the internet-of-things based on generative adversarial networks. *Applied Sciences*, 8(12):2351, 2018.