Article

Evolution-Strategies-Driven Optimization on Secure and Reconfigurable Interconnection PUF Networks

Hongxiang Gu and Miodrag Potkonjak *

Computer Science Department, University of California, Los Angeles, CA 90095, USA; hxgu1992@gmail.com

* Correspondence: miodrag@cs.ucla.edu; Tel.: +1-310-825-0790

Abstract: Physical Unclonable Functions (PUFs) are known for their unclonability and light-weight design. However, several known issues with state-of-the-art PUF designs exist including vulnerability against machine learning attacks, low output randomness, and low reliability. To address these problems, we present a reconfigurable interconnected PUF network (IPN) design that significantly strengthens the security and unclonability of strong PUFs. While the IPN structure itself significantly increases the system complexity and nonlinearity, the reconfiguration mechanism remaps the input-output mapping before an attacker could collect sufficient challenge-response pairs (CRPs). We also propose using an evolution strategies (ES) algorithm to efficiently search for a network configuration that is capable of producing random and stable responses. The experimental results show that applying state-of-the-art machine learning attacks result in less than 53.19% accuracy for single-bit output prediction on a reconfigurable IPN with random configurations. We also show that, when applying configurations explored by our proposed ES method instead of random configurations, the output randomness is significantly improved by 220.8% and output stability by at least 22.62% in different variations of IPN.

Citation: Gu, H.; Potkonjak, M. Evolution-Strategies-Driven Optimization on Secure and Reconfigurable Interconnection PUF Networks. *Electronics* **2021**, *10*, 537. <https://doi.org/10.3390/electronics10050537>

Academic Editor: Luis Gomes

Received: 6 January 2021

Accepted: 19 February 2021

Published: 25 February 2021

Publisher's Note: MDPI stays neutral with regard to jurisdictional claims in published maps and institutional affiliations.

Copyright: © 2021 by the authors. Licensee MDPI, Basel, Switzerland. This article is an open access article distributed under the terms and conditions of the Creative Commons Attribution (CC BY) license (<https://creativecommons.org/licenses/by/4.0/>).

Keywords: physically unclonable functions; hardware security; internet of things; reconfigurable; evolution strategies

1. Introduction

As of today, the amount of private information stored and flows between electronic devices is unimaginable. Adversaries are highly motivated to attack these electronics because of the potential benefits that they can gain from the stolen personal information. Secure and robust protection of electronics, as a result, is essential for any individual seeking security and privacy.

Physical Unclonable Functions (PUFs) came to the stage when traditional cryptography failed to stand its ground against physical attacks, side-channel attacks, and API attacks. A PUF, different from traditional key-based cryptographic systems, does not require a secret binary key; instead, the physical entity itself serves as the key. One huge advantage of a PUF-based system is that the secret key that is hidden within the physical body is designed to be unclonable, since it utilizes uncontrollable, nanoscale process variations. The complex structure of a PUF makes the output much harder to be predicted or derived comparing to those digital systems that stores secret keys in non-volatile memories.

Strong PUFs are a major subtype of PUFs. Like all PUFs, a strong PUF implements a complex function that maps some challenges to some responses. A PUF is considered to be a strong PUF if it is capable of meeting all the following requirements:

- **Unclonability.** Unclonability is the most fundamental feature of a PUF. A specific strong PUF cannot be physically cloned or replicated by anyone. Even the manufacturer who produces the PUF should not be able to manufacture a copy of the PUF that implements the same mapping function between challenges and responses.

- Unpredictability. Predicting the response of a random challenge should be extremely difficult, even if the attacker is capable of obtaining a large number of challenge-response pairs (CRPs).
- Determination difficulty. A strong PUF cannot be fully measured or determined within a reasonable amount of time. A PUF with a small challenge-response mapping set does not meet this requirement, since all of the mappings can be recorded given enough time (hours, days, or weeks). In most cases, this requirement is equivalent to a large set of possible challenges and limited read-out frequency.

Several strong PUFs have been proposed and studied in the past, yet none have been proven to be secure enough to hold all three requirements. The rise of machine learning technology provides adversaries with a powerful weapon that is capable of creating a model of the function a PUF implements. A mathematical model is a software program that is capable of predicting the corresponding responses of a PUF when provided with random challenges with high probability. Such a statistical model can be easily established by learning from a small subset of CRPs.

We propose a reconfigurable interconnected PUF network structure that is capable of providing sufficient robustness and resilience against different types of machine learning attacks. The idea is essentially to create a network structure that interconnects multiple small PUFs, so that the system is so complex that current machine learning attack methods are unable to accurately predict the responses given arbitrary challenges in a reasonable amount of time. The proposed design is capable of reconfiguring itself, so that challenge-response mappings completely alter. The reconfiguration of an IPN forces an adversary to restart the attack to learn a new mapping function.

In addition, we observe that a subset of all connection configurations inside an IPN could generate outputs that meet specific randomness or stability requirements. Therefore, we propose exploring an optimal method to connect APUFs, so that we can significantly improve or even overcome the natural weakness of PUFs.

This paper is an extension of [1]. In the original manuscript, we claimed the following contributions:

- We have proposed a network structure that interconnects multiple PUFs. By doing so, we significantly increase the system complexity as well as break the linearity, so that the interconnected PUF network shows high resilience against current machine learning attacks.
- Our interconnected PUF network is compatible with any strong PUF. In this work, we simulated and implemented an interconnected PUF network with only delay-based PUFs and some well-known variations; however, the whole framework can be easily extended to other strong PUFs, such as Bitline PUF and current mirror PUF.
- We have tested our interconnected PUF network against different algorithms, with and without the reconfiguration functionality. We show that the sample complexity of an IPN is significantly larger than the state-of-the-art delay-based PUF and its variants. Modeling an IPN requires a much larger training set as well as a much longer time. We are the first to propose to reconfigure a PUF-based system before an adversary could collect the theoretical lower bound of the sample complexity.
- Our reconfigurable PUF network design can be reconfigured during runtime with much lower latency and overhead when comparing to other reconfigurable PUF design such as [2].

In this paper, we further explored the research direction and extend our contribution by:

- conducting a more thorough study on the security properties of IPN with much more complex architecture; and,
- proposing a machine learning-based method to further improve the security and reliability of the originally proposed architecture.

2. Related Work

2.1. Physical Unclonable Function (PUF)

PUF was first proposed by Pappu et al. using mesoscopic optical systems [3]. Gassend et al. developed the first silicon PUFs through the use of intrinsic process variation in deep sub-micron integrated circuits [4]. A variety of other types of PUFs have since been proposed, including arbiter PUFs [4], ring oscillator PUFs [5], SRAM PUFs [6], and butterfly PUFs [7]. Xu et al. have proposed digitalizing PUFs and creating digital PUFs [8]. Another popular robust PUF design is proposed by Maiti et al. with a selected PUF challenge-response set [9].

2.2. Modeling Attack on PUF

PUFs are vulnerable to modeling attacks. Early works on modeling attack targeting PUFs were focused on standard arbiter PUFs [10,11]. Later on, Rurmair et al. presented the modeling attack results on multiple commonly seen PUFs, including APUFs, XOR PUFs, and feed-forward PUFs. They proved that all of the investigated PUFs are vulnerable to machine learning attacks [12]. Vijayakumar et al. later presented more detailed insights on applying different machine learning attacks to popular PUFs and why simple PUF structures are weak against modeling attacks [13]. Our results show that the proposed IPN structure provides sufficient resilience against the modeling attacks proposed in the above papers.

The increase of deep neural networks has imposed new challenges to arbiter PUFs and variants. A deep neural network is capable of training a model that simulates the function a PUF implements with high accuracy within a short period. Yashiro et al. conducted a security evaluation of authentication systems using arbiter PUFs, and concluded that an arbiter PUF and its variants are vulnerable against deep learning attack [14]. We show that an IPN provides high resilience against deep learning attacks. The complexity of an IPN is significantly larger than other PUF-based systems, so that a deep neural network can easily fall into overfitting problems when attempting to model an IPN. The reconfiguration functionality provides additional protection by regularly changing the mapping function.

2.3. PUF Randomness

Many attempts have been made to improve PUF randomness in the literature. O'Donnell from MIT proposed using PUF as a hardware random number generator (RNG) [15]. The direct use of PUF generates outputs with mediocre randomness. Thus they proposed to use Von Neumann correction to enhance the randomness. Maiti et al. proposed to combine a delay-based PUF and jitter-based RNGs, where a delay-based PUF can be used to extract chip-unique signatures and volatile secret keys, and the RNGs are used for generating random padding bits and initialization vectors [16]. All past efforts use additional random boosters to help to improve the randomness of PUF, whereas our work improves the output randomness by configuring the connection of PUFs without any external resources.

2.4. PUF Stability

The stability problem of arbiter PUFs has been well recognized for decades. As Zhou et al. pointed out in the study of one trillion CRPs, instability is a huge weakness in multiple variations of APUFs [17]. Stable CRPs in a no XOR single-bit PUF (0.8 V~1.0 V, 0~60 °C) only count as low as 80% of the entire CRP space where a 10-XOR APUF has only 0.0028% of all CPRs being stable over a study of one million CRPs. A large number of new APUF designs that address the stability problems have been proposed in the past decades such as [7,18,19]. However, most of these PUFs are weak PUFs with a small number of possible CRPs and they require an external output stabilizer or error correction code, such as fuzzy extractors [20], to ensure the output stability. We take a different approach to eliminate as many unstable segments as possible by actively searching for an interconnection that provides the best stability in APUFs.

3. Preliminaries

Strong PUF Model

An IPN can use any strong PUF as fundamental building blocks, from regular arbiter PUF to LRR-DPUF [21]. In this paper, we use standard delay-based arbiter PUFs as an illustrative example for simplicity considerations. An n -bit APUF takes an n -bit vector as a challenge and produces a 1-bit response as output. The challenge is provided to configure two nominally identical paths. Each challenge bit controls whether a pair of paths should swap positions within a PUF segment. An impulse signal is fed into the system to excite both paths simultaneously in order to retrieve a response. The signal traveling along one of the two paths reaches the arbiter earlier, generating corresponding outputs, because of the uncontrollable, nanoscale process variation.

Assuming an arbiter PUF implements a function F that maps a set of n -bit challenges C to corresponding response set R . We assume no delays on the connection wires and all delays are contributed by the APUF segments. Given a specific challenge $c \in C$, the i th APUF segment generates a pair of delays with delay difference of Δd_i^c . The corresponding response $r \in R$ can be mathematically represented as Equation (1):

$$F(c) = r = \begin{cases} 0 & \text{if } \sum_{i=1}^n \Delta d_i^c > 0 \\ 1 & \text{if } \sum_{i=1}^n \Delta d_i^c < 0 \end{cases} \quad (1)$$

4. Interconnected PUF Network (IPN)

4.1. IPN Architecture

4.1.1. IPN Node

An IPN consists of nodes and edges. A node consists of multiple arbiter PUFs of the same length. We define an k -bit IPN node of size s consists of s k -bit APUFs. If $s = k$, a node is denoted as a homogeneous node, otherwise it is denoted as a heterogeneous node. The size of a node is the total number of arbiter PUFs running in parallel, and the length of a node is the number of segments of each arbiter PUF in the node. Figure 1 shows a demonstrative diagram of a node. An IPN node takes an k -bit vector as the challenge and generates s 1-bit responses. All of the APUFs within the same IPN node share the same challenges.

Figure 1. An interconnected physical unclonable functions network (IPN) node of size s and length k . If $s = k$, the node is homogenous, otherwise it is heterogenous.

4.1.2. IPN Edge

An IPN node connects to other nodes through edges. To achieve reconfigurability, an edge is essentially designed to be a shuffler that takes the output from the previous node, shuffles the order, and then feeds them to the next node. A configuration vector is used to configure how the connections between two nodes are shuffled. For example, if an edge is an k -bit shuffler that directs the i -th bit of the input to the $k - 1 - i$ -th output bit, the configuration vector would be $\{k - 1, k - 2, \dots, 2, 1, 0\}$. All output port numbers are represented in binary form. The connections between nodes can be reconfigured easily by changing the configuration vector. We define a configuration of an IPN as a collection of all configuration vectors for all shufflers in the IPN.

4.1.3. IPN Chain

A simple network can be constructed by connecting IPN nodes to form a chain. All of the IPN nodes are connected through IPN edges. An edge from $node_i$ to $node_{i+1}$ indicates that the output of $node_i$ is fed into a shuffler, and then connects to all APUFs in $node_{i+1}$. Thus, each APUF in $node_{i+1}$ depends on the outputs of all APUFs in $node_i$.

4.1.4. More Complex Connections

IPN nodes can be connected in more complex manners. IPN supports not only one-to-one, but also one-to-many, many-to-one, and many-to-many connections between nodes.

One-to-many connections can be used to increase the output length as multiple nodes take the output of a specific node as input. The k -bit output of $node_0$ is used as input for two n -bit homogenous nodes $node_1$ and $node_2$, eventually generating a $2k$ -bit response.

We borrow the idea of XOR PUFs to use logic, like AND, OR, or XOR, to create many-to-one connections in IPNs. Many-to-one connections can be used to break the linearity and increase system entropy. $node_0$ and $node_1$ takes the same input and generate two sets of corresponding outputs. A logic operation, such as XOR, is applied to the outputs, and the result is then taken by $node_2$ as input. Many-to-one connections are expensive since the input/output length ratio is significantly larger than that of a one-to-one connection, resulting in requiring more PUF segments to build.

A many-to-many connection is a mixture of both one-to-many and many-to-many connections. The XORed result of both $node_0$ and $node_1$ outputs is fed to both $node_2$ and $node_3$ as input. A many-to-many connection provides additional nonlinearity without sacrificing the output size.

The combination of one-to-one, one-to-many, many-to-one, and many-to-many connections enables the possibility of creating larger and more complicated IPN, providing additional resilience and robustness against various attacks.

4.2. IPN Parameters

IPN nodes, edges, and different connections provide tremendous freedom in constructing a network. In this section, we intend to define some parameters that are associated with IPN structures.

4.2.1. Network Depth

We define the depth of an IPN as the length of the shortest path from an input node to an output node. An IPN with greater depth theoretically creates more dependency within the network, which makes the entire structure not differentiable. Thus, machine learning techniques that are based on differentiable models (e.g., Support Vector Machine) are less efficient. Additionally, a deeper IPN has multiple layers of dependency and requires more PUF segments, which increases the system complexity and makes it more difficult to predict. The concept of levels in an IPN is strongly associated with depth. The level of an IPN node is defined by one plus the smallest number of connections between the node and root node.

4.2.2. Network Width

We define the width of an IPN as the maximum number of nodes that shares the same input. According to our design, a wider IPN would have more many-to-one or many-to-many connections when comparing to the slimmer topology. A wide IPN has more nonlinearity, since many-to-one, and many-to-many connections require nonlinear logics, such as AND, OR, and XOR.

5. Machine Learning Attacks

In this section, we first discuss the security assumptions. We then briefly explain some conventional modeling/characterization techniques on PUFs that have been proven to be

effective in state-of-the-art PUF-based systems. We later investigate some newly proposed modeling methods, including deep-learning-based attacks and autoML modeling.

5.1. Assumptions

We consider the same assumption for controlled PUFs [22] that physical attacks on the control logic (which, in our case, is the reconfiguration logic) are more likely to alter or even destroy the PUF itself. The adversary has physical access to the PUF and its public CRP interface, as it is common in the established PUF attack model. Thus, the adversary can repeat CRP measurements at will to gain output stability.

5.2. Logistic Regressions

Logistic regression is proven to be effective against conventional delay-based arbiter PUFs and variations, such as XOR PUFs and lightweight secure PUFs. Logistic regression-based PUF attacks use a weight vector $\vec{w}$ to encode the internal parameters within the PUF system. The conditional probability can be represented using a sigmoid function acting on the PUF function f , as shown below. c is a challenge and r is the corresponding response.

$$p(c, r|\vec{w}) = \text{sigmoid}(rf(\vec{w}))$$

For a training set τ , the goal of the regression is to find a weight vector $\vec{w}$, so that the likelihood of observing this set is maximized, which is equivalent to minimizing the negative log-likelihood that is shown in Equation (2).

$$\hat{\vec{w}} = \arg \min_{\vec{w}} -\log \mathcal{L}(\tau, \vec{w}) = \sum_{(c,r) \in \tau} -\log \text{sigmoid}(rf(\vec{w})) \quad (2)$$

Different from general logistic regression problems, the optimal parameter vector $\hat{\vec{w}}$ can not be analytically analyzed, the only option is to optimize iteratively. For the purpose of attacking a single-bit output PUF, the problem can be tackled by transforming it to a binary logistic regression problem and use iteratively reweighted least squares (IRLS) to minimize the Log-likelihood of a Bernoulli distributed process using Newton's method. Other options include using optimization methods, such as gradient decent and RProp, etc. The above optimization methods requires gradient information represented in Equation (3):

$$\nabla \mathcal{L}(\tau, \vec{w}) = \sum_{(c,r) \in \tau} r(\text{sigmoid}(rf(c, \vec{w})) - 1) \nabla f(c, \vec{w}) \quad (3)$$

5.3. Evolution Strategies

Evolution strategies are a commonly seen attack method on PUF-based systems. Evolution strategies are a different type of machine learning algorithms that performs random search intelligently. Being inspired by evolutionary adaption to environments, the evolution strategies method always chooses the best candidates from randomly generated models and further develops on them. In the case of modeling a PUF-based system, one instantiation of internal delay parameters is denoted as an individual, and all instantiations together are called the population. The population of each selection is called a generation and each selected individual is allowed to produce offsprings by randomly mutating the instantiation of delay parameters. The selection is performed based on how well an individual instantiation is capable of reproducing the correct CRPs (fitness).

Fitness evaluation over the entire training set is expensive and slow. We borrowed the mini-batch idea from stochastic gradient descent to only select a subset of training CRPs for fitness evaluation purposes.

5.4. Multilayer Perceptron

The development of deep neural networks has made tasks that were once believed to be undoable possible. From speech recognition to image captions, deep neural networks

have made miraculous progress and is still improving. It is not surprising that adversaries use artificial neural networks to model a PUF-based system. A multilayer perceptron (MLP) is a type of feedforward artificial neural network. An MLP consists of at least three layers of nodes, respectively input layer, the output layer, and hidden layers. Each node within an MLP is a neuron that uses a nonlinear activation function. MLP learns a model by iteratively changing the connection weights based on the error between the output and ground truth. This type of supervised learning using MLP is carried out through back-propagation using gradient descent.

For PUF attacking purpose, predicting PUF output is essentially a classification problem. MLP is more powerful when comparing to logistic regression methods in terms of the capability to learn non-linear models.

5.5. Other Machine Learning Algorithms

As conventional attacks on PUFs depend on attackers to manually choose a model and hyperparameters, the results of modeling attacks might not be optimal concerning both prediction accuracy and speed. AutoML is a new concept that focuses on the progressive automation of machine learning. AutoML aims to create an automated process that intelligently performs an architecture search over a wide range of machine learning algorithms and choose the one that best fits the data and the task, including, but not limited to, naive Bayes classifiers and decision trees. AutoML is also capable of performing hyperparameter optimization that aims to find the best-suited hyperparameters for a given model. We choose to use Auto-sklearn to search for an algorithm along with corresponding hyperparameters to predict the behavior of an IPN [23].

6. Reconfiguration

IPNs benefit from the complex structure so that it requires a much larger training set and longer training time to model. We propose reconfiguring the entire network from time-to-time by changing the connections between IPN nodes so that any obtained knowledge on the IPN would be invalidated. Essentially, we are running a race with adversaries. Before one can finish modeling an IPN or collect enough training set, we reconfigure it so that the input-output mapping alters and the attacker would need to remodel the new IPN.

6.1. Reconfigure Timing

We can initiate a reconfiguration either before an attacker could collect a sufficient number of CRP or before an attacker can finish modeling. However, we believe that limiting the total number of generated CRPs is more secure and feasible since the speed of the attack process is affected by many factors on the adversarial side that we have no control of. Thus, we intend to find a lower bound for the size of the training set of IPN.

The sufficient size of the training set is also known as the sample complexity. We consider models of all PUF-based system as a binary function that takes a challenge and generates a 1-bit output of either 0 or 1. The Vapnik–Chervonenkis theory suggests that a PUF-based system can be learned with a finite sample complexity and the minimum required training size (N) follow the Equation (4):

$$N = O\left(\frac{VC(\mathcal{H}) + \ln\left(\frac{1}{\delta}\right)}{\epsilon}\right) \quad (4)$$

where $VC(\mathcal{H})$ is the Vapnik–Chervonenkis dimension of the function $\mathcal{H}$ implemented by the attacked PUF-based system, δ is the failure probability and ϵ is the learning error. The learning error ϵ can be empirically calculated by selecting a training set of CRPs to model each PUF node. It can be approximated by 1 minus the training accuracies for the different attack methods shown in Section 5.

For arbiter PUF, the VC-dimension is the total number of stages, meaning for a k -bit arbiter PUF, $VC(\mathcal{H}) = k$. Ruhrmair et al. derived the VC-dimension for XOR PUFs as $VC(\mathcal{H}) = k \cdot l$ where k is the number of stages in each arbiter PUF and l is the total number

of XORs. For Feed-forward PUFs, $VC(\mathcal{H}) = k + l$ can be used to describe the model better, where k is the total number of stages and l is the total number of feed-forward loops.

The sample space of an IPN on the other hand largely depends on the topology of the network. We have to be conservative in terms of finding a uniform lower bound for all topologies. The depth of the network conceptually is very similar to Feed-forward loops in Feed-forward PUFs, whereas the width of the network can be analogized to the size of XOR PUFs. Equation (5) describes a sample size lower bound in terms of the IPN model parameters, where m is the depth of the IPN network and n is the width of the network. It is to be noted that we assume every single path within the network to be of width n and depth m .

$$N \sim \frac{(m \cdot k + m) \cdot n + \ln(\frac{1}{\delta})}{\epsilon} \quad (5)$$

For each IPN structure, we derive an empirical formula based on Equation (5) by assuming a linear $y = ax + b$ relationship. The derived formula failed to match with the evolution strategies result due to the random nature of evolution strategies. The data points that we collected from evolution strategies show a super-linear relationship between N and ϵ . Thus, we adopt the method proposed by Ruhrmair et al. and modify the relationship to Equation (6) when applying evolution strategies to match the superlinear relationship. c is a constant between 0 and 1. When calculating the sample-complexity lower bound, we take $c = 1$ for worst scenario considerations.

$$N \sim \frac{(m \cdot k + m) \cdot n + \ln(\frac{1}{\delta})}{\epsilon^c} \quad (6)$$

An IPN-based system requires a much larger training set when comparing to standard arbiter PUFs, XOR PUFs, and Feed-forward PUFs of the same size. This can be observed when comparing Equation (5) to the lower bounds proposed in [12]. The conclusion is also confirmed by our experimental results that are shown in Section 5.

6.2. Reconfiguration Logic

The reconfiguration is performed by reconfiguring interconnections between IPN nodes. Because a shuffler controls each edge, the interconnect can be reconfigured by changing the configuration vectors in the shufflers.

We use a counter to count how many CRPs have already been generated, and we compare it with a predefined threshold. To be more conservative, we set a reconfiguration threshold Θ to a number that is smaller than the theoretical lower bound of sufficient CRPs using Equation (7), where ϵ is the misclassification rate for the attacking algorithm and $c = 0$ for the most conservative estimation on Θ . Instead of assuming that the network has the maximum width n on every level (Equation (5)), we assume every single path within the network to be of minimum width n' and depth m . Once Θ has been reached, a random number generator generates a new set of configuration vectors, and feed them to the IPN shufflers.

$$\Theta \sim \frac{m \cdot k \cdot n' + \ln(\frac{1}{\delta})}{\epsilon^c} \quad (7)$$

6.3. Protecting Reconfiguration Logic

The configuration vectors are essential in such a way that an adversary could potentially use the information to collectively train a model over different sets of samples, even if each sample size is intentionally limited below our calculated lower bound. An intuitive idea is to store all of the configuration vectors in non-volatile memories that lay below PUF delay wires so that damaging any one of those wires would change the PUF, rendering the adversary's attack useless [22]. However, in our reconfiguration logic, a new set of configuration vectors are provided by a random number generator or the user, which is not secure if the adversary has physical access to the device, as we described in our assumption. We take a step forward by securing these configuration vectors using existing IPN nodes

in the system so that the real interconnection remains hidden. We propose encrypting the user-provided configuration vector to an IPN node in the previous level. The configuration vectors for all shufflers between level i and $i + 1$ depend on the encrypted result of the user-provided configuration bits using the IPN nodes from level 1 to level $i - 1$, for $i > 1$. It is important to note that we use IPN nodes in the previous levels to encrypt shuffler configurations to reduce the correlation between the output of an IPN node and its immediate shufflers. Figure 2 shows an illustrative example.

Figure 2. Encrypting random configuration logic using existing IPN nodes. K_j is the configuration vector encrypted by a chain of nodes from $node_1$ to $node_j$.

All of the shufflers within an IPN are initialized at the beginning of the reconfiguration process. The random or user-provided configuration vectors are passed to the nodes in the first level propagate along the network to configure the remaining shufflers in the later levels. The shufflers between the first and second layers have a non-reconfigurable static connection. The attacker, even with physical access to the IPN device, cannot obtain information on actual configurations in shufflers without characterizing each IPN node. On the other hand, since the attacker cannot obtain enough training data given a specific configuration without knowledge of real configuration vectors in each shuffler.

7. Modeling Attack Resilience Results

In this section, we apply all the attack techniques we introduced in Section 5.

Our evaluation is conducted on both simulated models as well as implementations on a Xilinx Virtex-5 XC5VLX50T FPGA (Xilinx, San Jose, CA, USA). Our simulation assumes a Gaussian distribution in all delays and no error in contrast to real distribution and real errors in the implementation. As a comparison, we compare different IPN setups along with standard arbiter PUFs, XOR PUFs, and feed-forward PUFs. For fairness considerations, we maintain the total number of PUF segments used in both simulation and implementation the same over different structures. Because we intend to prove that an IPN structure itself is more resilient against machine learning attacks, which means that it is much harder to predict using a machine learning model, we provided all PUF-based systems discussed in this section with the same number of challenge-response pairs as well as same run-time/iterations. Our primary focus is on single bit prediction rate, even though IPN generates multi-bit outputs. It is noted that the modeling of both simulation and actual FPGA implementation was performed offline, which means that the training set and the test set of CRPs were collected before modeling. Querying the IPN is not allowed during the modeling process.

7.1. Logistic Regressions

In our security evaluation of IPN using logistic regression, we use standard gradient descent, IRLS, and RProp as the optimization method. In an attempt to model a simple IPN with reconfiguration functionality disabled, the difference between all three optimization methods is negligible.

We maintain the total number of PUF segments used in all settings to be around 1024. For all architecture, except standard arbiter PUF, the training set contains 30,000 CRPs, and the running time is set to unlimited. For each setting, we run 100 times, and the simulated results shown in Figure 3 are chosen from the best of 100 runs.

We observe that, after around 20,000 iterations, the error for all five structures converges. Logistic regression attack is capable of successfully predicting 1024-bit standard arbiter PUF and 256-bit 4-XOR PUF with 99% accuracy. The simplest standard arbiter

PUF architecture compromises immediately after the attack begins, where the 4-XOR PUF eventually converges after approximately 17,000 iterations.

IPN of depth 4 and width 1 provides better resilience against logistic regression comparing to the IPN of width 4 and depth 2, and this result is observed in all 100 runs. IPN of width 4 and depth 2, on the other hand, shows a very similar result with a forward arbiter PUF with 1024 stages and 64 feed-forward loops. All three cases were allowed to run until time-out at 100,000 iterations, which takes roughly seven days for each run.

Figure 3. Best results in 100 logistic regression attacks using 30,000 challenge-response pair (CRP) training set on five Physical Unclonable Function (PUF)-based systems. Error vs. iterations.

Based on the result, we believe it is safe to conclude on two observations. (1) Feed-Forward loops provide excellent resilience against logistic regressions, because the internal dependency introduced by feed-forward loops makes the model of the whole architecture no longer differentiable. Any attack methods that take advantage of linear separable or differentiable models would be extremely inefficient or simply inapplicable. (2) A deeper IPN provides better protection against logistic regression attack. The multiple layers of dependencies make the system even more complicated, so that gradient information is of no help regarding modeling such a system.

7.2. Evolution Strategies

In our security evaluation of IPN using evolution strategies, we use both canonical versions, respectively $(\mu/\rho, \lambda) - ES$ and $(\mu/\rho + \lambda) - ES$, with and without the mini-batch style of fitness evaluation method implemented based on [24]. The difference between the two versions of evolution strategies is that $(\mu/\rho + \lambda) - ES$ considers the parent population during the selection process where $(\mu/\rho, \lambda) - ES$ only selects from the offspring population. Both of the canonical versions of evolution strategies were applied to all investigated PUF-based systems, each with 100 runs. Figure 4 shows the best results among the 100 runs.

IPN of width 4 and depth 2 is the most difficult for evolutionary strategies attack to tackle, while the 1024-bit arbiter PUF with and without feed-forward loops performs the worst. A general trend for all curves in Figure 4 is that the speed of progress is slowing down. The probability of observing a massive decline regarding errors dramatically decreases as the total number of generations increase. We can observe that, for IPN of width 4 depth 2, the curve is almost flat after 30,000 generations.

Based on the result, we believe that it is safe to conclude on two observations. (1) Nonlinear logic functions like XORs dramatically increase the difficulty for evolution strategies attack models, whereas the feed-forward loop provides limited additional complexity against evolution strategies. (2) A wider IPN provides better protection against evolution strategies attack. This conclusion is not surprising, as a wider IPN introduces more XORs, which provides much more nonlinearity. Despite the nonlinearity introduced by XORs, we

still observe that an IPN of width 1 and depth 4 still performs better than 256-bit 4-XOR PUF when provided with the same training set.

Figure 4. Evolution strategies attack result using 30,000 challenge-response pair (CRP) training set on five PUF-based systems. Error vs. iterations.

7.3. Multilayer Perceptron

In our security evaluation of IPN using MLP, we reform the task as a binary classification problem. We experiment with different network configuration parameters being implemented using Keras [25]. After some experiment, the following setup shown in Table 1 provides the best results and speed.

Table 1. Multilayer perceptron (MLP) parameters when modeling interconnected Physical Unclonable Function network (IPN). n is the depth of the network and m_i is the total number of Physical Unclonable Function (PUF) segments on the i -th level.

Layers	Layer Type	Units	Activation
1	Dense	m_1	ReLu
...	Dense	...	ReLu
n	Dense	m_n	ReLu
$n + 1$	Dense	2	Softmax

The loss function used is binary cross-entropy, and we use Adam as the optimizer. We set the number of epochs to a constant of 100 so that we have a total number of CRPs/100 as our batch size.

When comparing to logistic regression and evolution strategies, an MLP does not necessarily require details in PUF architecture; instead, it treats the entire PUF as a black box and learns the function based on only input and output. Table 2 shows the result of applying MLP modeling to all discussed PUF systems.

Table 2. Deep neural network attack results.

Architecture	Training Acc.	Test Acc.
IPN depth 4 width 1	99.93%	50.18%
IPN depth 2 width 4	99.77%	50.33%
256-bit 4-XOR PUF	99.73%	96.02%
1024-bit arbiter PUF	99.99%	98.28%
1024-bit 64-ff PUF	99.99%	95.68%

MLP with the structure that is described in Section 5 is capable of fitting 30,000 CRPs with above 99% training accuracy, and can predict 256-bit 4-XOR PUF, 1024-bit arbiter PUF, and 1024-bit 64-ff PUF with above 95% test accuracy. However, it ran into an overfitting problem when modeling IPNs. After attempting various overfitting prevention techniques, including regularization layers and dropouts, we conclude that insufficient training samples is the root of the overfitting problem. IPNs are more complicated when comparing to other PUF systems. Thus, given a non-sufficient training dataset, the overfitting problem is more severe. When provided with a much larger dataset (5,000,000 CRPs in simulation), the test accuracy can be boosted to 86.49% for IPN with depth 4 width 1 and 78.01% for IPN of depth 2 widths 4. When applying the same training set, the test accuracy converges at 54.77% and 62.54%, respectively, being much lower than MLP attack results.

7.4. Other Machine Learning Algorithms

AutoML is still under development, yet it provides promising results as compared to MLP attacks in terms of modeling PUF-based systems. We only provided raw CRPs to the auto-sklearn module, and Table 3 shows the results for all tested architectures.

Table 3. Auto-sklearn modeling results on raw CRPs.

Architecture	Best Algo.	Test Acc.
IPN depth 4 width 1	Decision tree	64.87%
IPN depth 2 width 4	Decision tree	67.27%
256-bit 4-XOR PUF	K-NN.	83.88%
1024-bit arbiter PUF	Multinomial NB	89.55%
1024-bit 64-ff PUF	Multinomial NB	72.33%

In general, the best classifiers for IPNs are decision-tree classifiers, which are capable of predicting over 65% of CRPs in the test set. XOR PUFs, arbiter PUFs, and Feed-forward PUFs are much easier to model, since auto-sklearn is capable of finding a classifier (such as K-nearest neighbor or multinomial naive Bayes classifiers) that successfully predicts the test set CRPs with an accuracy over 70%.

7.5. Implementations on FPGA

Two differences distinguish a simulated PUF-based system and real implementations. (1) The delays located within a PUF implementation do not necessarily follow a certain distribution, whereas in simulations we assume a Gaussian or uniform distribution for all delays in the PUFs. (2) Real-world implementations suffer from stability issues. Because PUFs are extremely sensitive to environmental factors, such as temperature and voltage, the response that is generated by the same PUF might not be consistent when providing the same challenge multiple times.

We repeated all of the experiments on data that were collected from FPGA implementations of all discussed architecture. Table 4 shows the best results of both simulation and implementation.

Table 4. Best prediction accuracy on different PUF architectures using machine learning algorithms out of 100 runs. Each cell contains simulated/implementation result.

Architecture	Logistic Regression	Evolution Strategies	MLP	Auto ML
IPN-1-1-1-1	53.62%/53.68%	67.57%/67.18%	61.98%/51.59%	64.87%/50.04%
IPN-4-1	60.56%/58.23%	57.20%/56.33%	62.08%/54.20%	67.27%/52.16%
256-bit 4-XOR PUF	98.86%/95.01%	79.12%/76.02%	86.30%/61.82%	83.88%/67.80%
1024-bit arbiter PUF	99.99%/96.57%	99.62%/98.28%	89.55%/80.85%	89.55%/73.22%
1024-bit 64-ff PUF	62.27%/58.29%	97.62%/95.68%	72.33%/67.98%	82.33%/68.58%

Based on the log provided by Xilinx System Monitor, the largest variations in the core temperature and the core voltage are 2 °C and 2.78%, respectively, during the entire CRP collection process. The environmental variations lead to 10.96% of CRPs being unstable in the training set. By applying ECC on IPN, the amount of instability drops significantly to 3.67%. The stability issue along with random delays in the hardware reduces the prediction accuracy in almost all cases. The MLP and autoML models suffer the most from the instability in the training set, while the reduction in evolution strategies attacks are minimal.

Regardless of network depth and width, IPN stays robust against all proposed attacks, while XOR PUF and Feed-forward PUF can be modeled using different machine learning algorithms. Standard arbiter PUF, on the other hand, can be accurately modeled by all attack methods.

7.6. Implementation Result

We implemented a 64-bit reconfigurable IPN, where depth = 4 and width = 4 (as shown in Figure 5) on a Xilinx Virtex-5 board (Xilinx, San Jose, CA, USA) as our test IPN. According to our derived formula on the sample complexity of IPNs, the sufficient number of CRPs required to predict a single bit response with 95% accuracy is 716,703 CRPs. We set the reconfiguration threshold to 358,350 CRPs, and we collected 1,000,000 CPRs (with duplications) as our training set. Table 5 shows the prediction accuracy of 10,000 test challenges with and without reconfiguration functionality.

Figure 5. The IPN under test. Width = 4, Depth = 4. Labeled as IPN-4-1-1-1.

Table 5. Prediction accuracy of three attack methods. Results collected from a 100,000 test set. Logistic regression: 14 days; evolution strategies: 250,000 generations, 14 days; MLP: 18 h.

Attack Method	w/out Reconfig.	with Reconfig.
Logistic Regression	64.62%	53.19%
Evolution Strategies	72.13%	49.99%
MLP	80.64%	51.89%

We excluded Auto-sklearn from our experiment, as it is extremely slow when handling very-large data. We observe that, without reconfiguration enabled, no attack can reach the theoretical 95% of accuracy. We believe that in addition to the complexity of the IPN structure, the instability in the implemented PUFs increases the difficulty of accurately modeling an IPN. When reconfigurability is enabled, MLP performs the best due to its ability to quickly adapt to the new labels. Evolution strategies perform the worst, because, at each IPN reconfiguration, all selected populations need to re-adapt the new fitness function and the algorithm has to learn from scratch again.

Table 6 shows the area overhead for implementing the IPN with and without reconfiguration functionality. The reconfiguration mechanism uses 5.04% additional hardware; in return, it efficiently reduces the prediction rate by 11.43% at worst and 30.65% at best. If all of our assumption holds, modeling a large IPN with reconfigurability is practically impossible.

Table 6. Area overhead for implementing a variety of IPNs. The number after each IPN label marks the width at each depth, e.g., IPN-4-1-1-1 is shown in Figure 5. * indicates a reconfigurable IPN.

Resource	IPN-1-1-1-1	IPN-4-1-1-1	IPN-4-1-1-1	IPN-4-1-1-1 *
LUTs	1280	2484	2001	2102
Occupied Slices	871	1522	1157	1227
Flip-flops	999	1522	1399	1525

8. IPN Configuration Optimization

During our process of evaluating the machine learning resilience of IPN, we observe that the configuration vector of IPN is highly correlated with output randomness and stability.

8.1. IPN Randomness

An IPN with a random configuration sometimes cannot generate responses that meet certain randomness requirements. Take 0/1 frequency requirement for example. If the j_{th} PUF in $node_i$ has much larger probability of producing a “1” as output, all j_{th} segment in $node_{i+1}$ would prefer one delay path over another. If, unfortunately, an affected segment generates dominating a delay difference, the 0/1 frequency balance of the APUF in $node_{i+1}$ that contains such a segment could be damaged.

To avoid such damage, we first propose using a brute force random search method to try out different connections until an acceptable one is found. A good configuration is defined as a set of configuration vectors that enables the IPN output to meet specific randomness requirements.

We ran simulations on 100 different IPN instances. We use the average passing rate for all tests in NIST test suite [26] as our randomness scoring function. Each test is modified to show the quality of IPN output regarding a specific randomness property. For a given configuration, a sample of 10,000 bits is collected before evaluating it while using the randomness scoring function. The maximum number of configurations to test on each IPN instance is set to 100. Table 7 shows the success ratio of passing a specific random test on average versus the best result obtained from the 100 instances.

The result shows that allowing for the exploration of multiple IPN configurations significantly increases the chance that an IPN is capable of passing a specific randomness test. This result is intuitive and straightforward, since the delay characteristics in APUFs are unpredictable. Certain combinations of such APUFs provide better randomness results than others. Exploring more configurations in an IPN naturally increases the chances of discovering an arrangement of APUFs that provides highly random results.

Table 7. Success ratio of passing a National Institute of Standards and Technology (NIST) test for 100 IPN instances with two 32-bit homogenous nodes. The maximum number of configurations allowed for each IPN is 100, each configuration collects 10,000-bit result.

Test	Direct Connection	Best in 100
Frequency Test	32%	89%
Run Test	14%	76%
Linear Complexity	98%	100%
Approx. Entropy	1%	15%

8.2. IPN Stability

Stability is an essential factor of IPN. Therefore, a single bit change in the first few nodes could cause a significant avalanche effect to the final output of IPN. We also ran simulations with the simple IPN chain structure. We model the relationship between transistor delays and environmental variations according to Equation (8) described in [27]. Here, k_{tp} is the delay-fitting parameter, C_L is the sum of the intrinsic capacitance and the load capacitance, V_{dd} is the supply voltage, n is the subthreshold slope, μ is the mobility, C_{ox} is the oxide capacitance, L is the effective channel length, W is the gate width, ϕ is the thermal voltage ($\phi = \frac{kT}{q}$), k_{fit} is a model-fitting parameter, and IC represents the inversion coefficient. We fixed all of the parameters as derived from a curve-fitting Spectre simulation results for a 65-nm CMOS technology, as described in [27] except V_{dd} and ϕ . We simulate voltage and temperature variations by sample the value of V_{dd} and ϕ from a normal distribution, where the mean and variance are collected from real chip measurement.

$$t_p = \frac{k_{tp} \cdot C_L \cdot V_{DD}}{2 \cdot n \cdot \mu \cdot C_{ox} \cdot \frac{W}{L} \cdot \phi_t^2} \cdot \frac{k_{fit}}{IC} \quad (8)$$

We run our simulation on a single PUF, a 10-XOR APUF, and a four-node IPN chain with one-million CRPs and we obtained 24.232%, 99.985%, and 64.312% of unstable CRPs. The first two observations are incredibly similar to the results that were reported by [17].

9. IPN Optimization Algorithms

Knowing that exploration for a good configuration could greatly improve the possibility for an IPN instance to pass a random test, we thus propose two algorithms for configuring a given IPN so that it can produce outputs that meet certain randomness requirements in this section.

9.1. Random Search

Based on our observation in Section 8.1, we first formally propose a random search algorithm as our baseline method in Algorithm 1.

The random search algorithm is a simple brute force algorithm that explores many configurations until a good configuration is found. If no good configuration can be discovered after a large number of iterations, the possibility of finding such a configuration is assumed to be low. It is also possible that dominating segments or hardware biases in some APUFs make it impossible to construct a desired IPN. Thus, an upper bound is set as the maximum number of iterations allowed. The algorithm returns nothing if no desired configurations are found after the maximum number of iterations.

Algorithm 1 Random Search in Interconnected Physically Unclonable Functions Network (IPN).

Require: (1) An IPN with n nodes and $n - 1$ shufflers. (2) A set of configuration vectors $V = \{V_1, V_2, \dots, V_{n-1}\}$ that configures each shuffler. (3) A randomness scoring function $f(x)$ where x is a sampled binary string of size s . (4) A passing threshold Θ . (5) Total number of iterations L .

Ensure: A set of configuration vectors $V = \{V_1, V_2, \dots, V_{n-1}\}$ that enables IPN to produce outputs that passes $f(x)$. If no such set of vectors can be found in L iterations, returns nothing.

while $L > 0$ **do**

    Sample a binary string x of size s using IPN.

if $f(x) < \Theta$ **then**

for each V_i in V **do**

            Randomly shuffles V_i

end for

else

        return V

end if

$L = L - 1$.

end while

If no configuration is found in L iterations, return nothing.

9.2. Evolution Strategies

Random search shows significant improvement when comparing to direct connection. However, the searching process is entirely random so that the speed of finding a good configuration is based on luck. Here we present an ES algorithm that does not necessarily test a completely different configuration when a randomness objective is not achieved, but instead only reconfigures a portion of the configuration based on how well the current configuration performs. The essential idea is not to recreate a new set of configuration vectors, but to improve on the current configuration. The ES algorithm is based on the observation that a minor modification on a fair configuration vector sometimes leads to better or even excellent results. Algorithm 2 describes the detailed algorithm.

The ES algorithm, similar to random search, uses a random configuration as a start. A swap ratio index is used to determine what portion of the configuration should be shuffled to achieve the desired randomness goal. If the result keeps improving, then we gradually decrease the swap ratio to make as few modifications as possible at each iteration, so that past progress could be preserved. If the results are worsening, we increase the swap ratio, and, in an extreme case, randomly shuffles all configuration vectors.

A key feature of our ES algorithm is the backtracking mechanism. If a shuffler configuration vector V already generates a fair result that is only slightly below expectation, we expect that minimum modification could provide sufficient results. If multiple modifications to the configuration vector V lead to a decrease in the randomness score, we conclude that it is unlikely that slight modification to V provides desired results, so we backtrack to V and increases the swap ratio to 1.

Algorithm 2 Evolution Strategies in IPN.**Require:** (1) An IPN with n nodes and $n - 1$ shufflers.(2) A set of configuration vectors $V = \{V_1, V_2, \dots, V_{n-1}\}$ that configures each shuffler.(3) A randomness scoring function $f(x)$ where x is a sampled binary string of size s .(4) A scoring function threshold Θ and a near-passing threshold $\theta < \Theta$.(5) A swap ratio $\alpha \in (0, 1]$, initially, $\alpha = 1$.(6) A backtracking counter $\beta = 0$ and a backtracking threshold λ .(7) Total number of iterations L .**Ensure:** A set of configuration vectors $V = \{V_1, V_2, \dots, V_{n-1}\}$ that enables IPN to produce outputs that passes $f(x)$. If no such set of vectors can be found in L iterations, returns nothing.**while** $L > 0$ **do**   Sample a binary string x of size s using IPN.   **if** $f(x) < \Theta$ **then**   **if** $f(x) > \theta$ **then**   Decrease the swap ratio α , $\alpha > 0$.   Save $f(x)$ and V as $f'(x)$ and V' .   **else**   **if** $f'(x) > \theta$ **then** $\beta = \beta + 1$ **end if**   Increase the swap ratio α , $\alpha \leq 1$.   **end if**   **if** $\beta > \lambda$ **then** V backtrack to V' $\beta = 0$, $\alpha = 1$ **else**   **for each** V_i in V **do**   Select $\frac{1}{2}\alpha|V_i|$ connections in V_i , swap connections with another $\frac{1}{2}\alpha|V_i|$ connections.   **end for**   **end if**   **else**   return V **end if** $L = L - 1$.**end while**If no configuration is found in L iterations, return nothing.**10. Experimental Results on IPN Optimization***10.1. Randomness Optimization*

Both random search and ES can handle multiple randomness tests at the same time. The scoring function $G(x)$ with N tests is a weighted composite function of results that were obtained from the NIST test suite [26], Diehard-1997 [28], Diehard-2009 [29], and Soto [30]. The scoring function is shown in Equation (9), where each test transforms to a scoring function $f_i(x)$. The result is normalized to $[0, 1]$. For the ES algorithm, the scoring function pass threshold Θ is set at 0.99 and the near-pass threshold θ is set at 0.9. The backtracking threshold λ is set at 10% of the maximum number of iteration. An instance passes the test if it achieves a score of at least Θ .

$$G(x) = \sum_{i=1}^N \frac{1}{N} f_i(x) \quad (9)$$

Table 8 shows our experimental results for each test in the NIST suite as well as the overall passing rate. The overall passing rate is calculated as the possibility of passing all

tests. ES outperforms unoptimized configuration by an average of 220.8% and random search by an average of 21.86% in all test cases.

Table 8. Percentage of 1000 IPN instances that pass the NIST test suite on the test IPN. Each configuration collects 10,000-bit result and the maximum number of iterations is set to 1000.

Statistical Test	Unoptimized	Rand.	Iter. Improv
Freq.	36.4%	88.4%	94.8%
Block Freq.	28.5%	82.5%	92.6%
Runs	15.7%	76.0%	82.8%
Longest Runs	13.6%	67.8%	81.1%
Binary Rank	54.5%	99.0%	99.5%
FFT	44.8%	89.7%	92.1%
Aperiodic	6.9%	14.6%	85.0%
Periodic	9.5%	15.4%	95.1%
Maurer	32.8%	79.2%	88.1%
Lin Complex.	85.0%	99.7%	100.0%
Serial	69.4%	80.2%	95.5%
Cusum	44.4%	86.4%	93.7%
Excursions	47.3%	88.4%	94.8%
Variant	37.9%	89.4%	92.7%

Both random search and ES algorithms have exploited the potential in generating highly random outputs in APUFs. By configuring the connections between IPN nodes, both of the algorithms seek to find the best combination of APUFs, so that randomness problems that are caused by hardware and structural factors can be fixed and compensated.

10.2. Stability Optimization

We reuse the experimental setup in Section 10.1 to improve the stability of IPN. Here, we define stability as the possibility of observing a stable response over repeated observation of 100 times when provided a single challenge. We conduct our experiments on 1000 IPN instances of the test IPN. Figure 6 presents the result.

Figure 6. Stability of 1000 test conducted on the test IPN instances. Environmental variance: 0.8 V~1.0 V, 0~60 °C.

We observe that both ES and the random search approach significantly boosts stability. Specifically, our ES approach outperforms random search, as shown in Figure 6. Despite the 22.62% improvement as compared to random non-optimized IPN in IPN chains and 25.38% in complex IPNs when comparing to baseline random search algorithm), additional error correction code might still be required to generate highly stable outputs for stability sensitive tasks for large IPN.

11. Potential Applications

When the total response that is generated for a specific configuration has reached the reconfiguration threshold (a lower bound of the approximated sample complexity), the old configuration is considered to be expired and it can be reconfigured. In real-world applications, searching for the next optimized configuration can be challenging. Table 9 describes how optimization can be done in different scenarios.

IPN can be viewed as a more secure extension of existing strong PUFs. At any given configuration IPN is equivalent to a strong PUF with higher complexity. An IPN can be easily replace all strong PUFs that are used in PUF-based security applications from anti-counterfeiting in supply chains [31–33] to Internet-of-Things (IoT) device authentication [34–36]. Depending on the nature of the application, IPN can be reconfigured using one of the methods described in Table 9. An IPN with a new optimized configuration is equivalent to a brand new PUF without requiring a hardware replacement.

Table 9. Reconfigurable IPN optimization choices in different scenarios.

Scenario	Method	Description	Pros	Cons
Low usage, not physically accessible	Offline optimization	Before the deployment of the device embedded with a reconfigurable IPN, a set of good configurations are explored. Upon a new reconfiguration request, a new configuration is randomly selected from a pool of candidates	Low cost	Additional attack surface (storage of configuration pool), limited number of configurations
High usage, physically accessible	Redeployment	Re-explore an optimized configuration and reconfigure	Minimum attack surface	High requirement on device accessibility, high deployment cost
High usage, not physically accessible	Online optimization	IPN optimization is done online via secure communication channels	High reconfiguration frequency	High cost, additional attack surface (communication channel), slow optimization

12. Conclusions

We have carefully studied an interconnected PUF network structure that connects PUFs to build a network in this paper. Our simulation and implemented results show that the IPN has a complex structure, so that it enables itself to stay robust against not only traditional PUF modeling methods, like logistic regression and evolution strategies, but also the state-of-the-art methods, like deep neural networks and AutoML.

We propose making an IPN reconfigurable by shuffling the interconnections between IPN nodes to eliminate the possibility of being modeled with a large training set. Before an adversary can collect sufficient CRP sets for training purposes, the IPN reconfigures itself, so that the attacker would not be able to obtain enough information on the IPN. To avoid storing the configuration vectors, we propose using another set of PUFs to protect the

configuration vectors. Our experimental results indicate that no investigated attack could accurately model an IPN. The single-bit prediction accuracy for all attacks, when provided with a training set larger than the theoretical lower bound and 14 days, is around 50%.

We have also proposed a novel evolution strategies algorithm to optimize the output randomness and stability of IPNs. Our experimental results indicate that our ES algorithm outperforms the unoptimized configuration by an average of 220.8% and standard search algorithm an average of 21.86% in all NIST randomness tests. We also observe that our method is also capable of improving output stability by at least 22.62% in IPN of different complexities.

Author Contributions: Conceptualization, H.G. and M.P.; methodology, H.G.; implementation, H.G.; writing, H.G.; supervision, M.P. All authors have read and agreed to the published version of the manuscript.

Funding: This work was supported in part by the NSF under award CNS-1513306.

Data Availability Statement: Data sharing is not applicable.

Conflicts of Interest: The authors declare no conflict of interest.

Abbreviations

The following abbreviations are used in this manuscript:

PUF	Physically Unclonable Functions
IPN	Interconnected PUF Network
CRP	Challenge Response Pairs
ES	Evolution Strategies
LR	Logistic Regression
MLP	Multilayer perceptron
NIST	National Institute of Standards and Technology
FPGA	Field-Programmable Gate Array

References

- Gu, H.; Potkonjak, M. Securing interconnected PUF network with reconfigurability. In Proceedings of the International Symposium on Hardware Oriented Security and Trust (HOST), Washington, DC, USA, 30 April–4 May 2018; pp. 231–234.
- Gehrer, S.; Sigl, G. Reconfigurable PUFs for FPGA-based SoCs. In Proceedings of the International Symposium on Integrated Circuits (ISIC), Singapore, 10–12 December 2014; pp. 140–143.
- Pappu, R.; Recht, B.; Taylor, J.; Gershenfeld, N. Physical one-way functions. *Science* **2002**, *297*, 2026–2030. [[CrossRef](#)] [[PubMed](#)]
- Gassend, B.; Clarke, D.; Van Dijk, M.; Devadas, S. Silicon physical random functions. In Proceedings of the 9th ACM Conference on Computer and Communications Security, Washington, DC, USA, 18 November 2002; pp. 148–160.
- Suh, G.E.; Devadas, S. Physical unclonable functions for device authentication and secret key generation. In Proceedings of the 44th Annual Design Automation Conference, San Diego, CA, USA, 4–8 June 2007; pp. 9–14.
- Guajardo, J.; Kumar, S.S.; Schrijen, G.J.; Tuyls, P. *FPGA Intrinsic PUFs and Their Use for IP Protection*; Springer: Berlin/Heidelberg, Germany, 2007.
- Kumar, S.S.; Guajardo, J.; Maes, R.; Schrijen, G.J.; Tuyls, P. The butterfly PUF protecting IP on every FPGA. In Proceedings of the IEEE International Workshop on Hardware-Oriented Security and Trust, Anaheim, CA, USA, 9 June 2008; pp. 67–70.
- Xu, T.; Potkonjak, M. Robust and flexible FPGA-based digital PUF. In Proceedings of the 24th International Conference on Field Programmable Logic and Applications (FPL), Munich, Germany, 2–4 September 2014; pp. 1–6.
- Maiti, A.; Kim, I.; Schaumont, P. A robust physical unclonable function with enhanced challenge-response set. *IEEE Trans. Inf. Forensics Secur.* **2012**, *7*, 333–345. [[CrossRef](#)]
- Öztürk, E.; Hammouri, G.; Sunar, B. Towards robust low cost authentication for pervasive devices. In Proceedings of the Annual IEEE International Conference on Pervasive Computing and Communications, Hong Kong, China, 17–21 March 2008; pp. 170–178.
- Majzoobi, M.; Koushanfar, F.; Potkonjak, M. Testing techniques for hardware security. In Proceedings of the IEEE International Test Conference, Santa Clara, CA, USA, 28–30 October 2008; pp. 1–10.
- Rührmair, U.; Sölter, J.; Sehnke, F.; Xu, X.; Mahmoud, A.; Stoyanova, V.; Dror, G.; Schmidhuber, J.; Burleson, W.; Devadas, S. PUF modeling attacks on simulated and silicon data. *IEEE Trans. Inf. Forensics Secur.* **2013**, *8*, 1876–1891. [[CrossRef](#)]

13. Vijayakumar, A.; Patil, V.C.; Prado, C.B.; Kundu, S. Machine learning resistant strong PUF: Possible or a pipe dream? In Proceedings of the IEEE International Symposium on Hardware Oriented Security and Trust (HOST), McLean, VA, USA, 3–5 May 2016; pp. 19–24.
14. Yashiro, R.; Machida, T.; Iwamoto, M.; Sakiyama, K. Deep-Learning-Based Security Evaluation on Authentication Systems Using Arbiter PUF and Its Variants. *International Workshop on Security*; Springer: Berlin/Heidelberg, Germany, 2016; pp. 267–285.
15. O'Donnell, C.W.; Suh, G.E.; Devadas, S. PUF-based random number generation. In *MIT CSAIL CSG Technical Memo*; MIT CSAIL: Cambridge, MA, USA, 2004; Volume 481.
16. Maiti, A.; Nagesh, R.; Reddy, A.; Schaumont, P. Physical unclonable function and true random number generator: A compact and scalable implementation. In Proceedings of the 19th ACM Great Lakes Symposium on VLSI, Boston Area, MA, USA, 10 May 2009; pp. 425–428.
17. Zhou, C.; Parhi, K.K.; Kim, C.H. Secure and reliable XOR arbiter PUF design: An experimental study based on 1 trillion challenge response pair measurements. In Proceedings of the 54th Annual Design Automation Conference 2017, Austin, TX, USA, 18 June 2017; p. 10.
18. Bhargava, M.; Mai, K. A high reliability PUF using hot carrier injection based response reinforcement. In *International Workshop on Cryptographic Hardware and Embedded Systems*; Springer: Berlin/Heidelberg, Germany, 2013; pp. 90–106.
19. Alvarez, A.; Zhao, W.; Alioto, M. 14.3 15fJ/b static physically unclonable functions for secure chip identification with <2% native bit instability and 140 × Inter/Intra PUF hamming distance separation in 65nm. In Proceedings of the IEEE International Solid-State Circuits Conference-(ISSCC), San Francisco, CA, USA, 22–26 February 2015; pp. 1–3.
20. Dodis, Y.; Reyzin, L.; Smith, A. Fuzzy extractors: How to generate strong keys from biometrics and other noisy data. In *International Conference on the Theory and Applications of Cryptographic Techniques*; Springer: Berlin/Heidelberg, Germany, 2004; pp. 523–540.
21. Miao, J.; Li, M.; Roy, S.; Yu, B. LRR-DPUF: Learning resilient and reliable digital physical unclonable function. In Proceedings of the IEEE/ACM International Conference on Computer-Aided Design (ICCAD), Austin, TX, USA, 7–10 November 2016; pp. 1–8.
22. Clarke, G.; Van Dijk, D.; Devadas, S. Controlled physical random functions. In Proceedings of the 18th Annual Computer Security Applications Conference, Las Vegas, NV, USA, 9–13 December 2002; pp. 149–160.
23. Feurer, M.; Klein, A.; Eggenberger, K.; Springenberg, J.; Blum, M.; Hutter, F. Efficient and Robust Automated Machine Learning. In *Advances in Neural Information Processing Systems 28*; Cortes, C., Lawrence, N.D., Lee, D.D., Sugiyama, M., Garnett, R., Eds.; Curran Associates, Inc.: Red Hook, NY, USA, 2015; pp. 2962–2970.
24. Evolutionary Computation: An Introduction—Ecsy 1.1 Documentation. Available online: https://pythonhosted.org/ecspy/ec_intro.html (accessed on 4 April 2017).
25. Chollet, F. Keras. 2015. Available online: <https://github.com/fchollet/keras> (accessed on 20 April 2017).
26. Rukhin, A.; Soto, J.; Nechvatal, J.; Smid, M.; Barker, E. *A Statistical Test Suite for Random and Pseudorandom Number Generators for Cryptographic Applications*; Special Publication (NIST SP); Information Technology Laboratory, National Institute of Standards and Technology: Gaithersburg, MD, USA, 2001.
27. Markovic, D.; Wang, C.C.; Alarcon, L.P.; Liu, T.T.; Rabaey, J.M. Ultralow-power design in near-threshold region. *Proc. IEEE* **2010**, *98*, 237–252. [\[CrossRef\]](#)
28. Marsaglia, G. The Diehard Random Number Test Suite. 1997. Available online: <http://stat.fsu.edu/pub/diehard> (accessed on 20 April 2017).
29. Brown, R.G.; Eddelbuettel, D.; Bauer, D. *Dieharder: A Random Number Test Suite*; Duke University Physics Department: Durham, NC, USA, 2009.
30. Soto, J. Statistical testing of random number generators. In Proceedings of the 22nd National Information Systems Security Conference, NIST Gaithersburg, MD, USA, 18–21 October 1999; Volume 10, p. 12.
31. Leem, J.W.; Kim, M.S.; Choi, S.H.; Kim, S.R.; Kim, S.W.; Song, Y.M.; Young, R.J.; Kim, Y.L. Edible unclonable functions. *Nat. Commun.* **2020**, *11*, 1–11. [\[CrossRef\]](#) [\[PubMed\]](#)
32. Islam, M.N.; Kundu, S. Enabling ic traceability via blockchain pegged to embedded puf. *ACM Trans. Des. Autom. Electron. Syst.* **2019**, *24*, 1–23. [\[CrossRef\]](#)
33. Devadas, S.; Suh, E.; Paral, S.; Sowell, R.; Ziola, T.; Khandelwal, V. Design and implementation of PUF-based ‘unclonable’ RFID ICs for anti-counterfeiting and security applications. In Proceedings of the 2008 IEEE International Conference on RFID, Las Vegas, NV, USA, 16–17 April 2008; pp. 58–64.
34. Gu, H.; Potkonjak, M. Efficient and secure group key management in IoT using multistage interconnected PUF. In Proceedings of the International Symposium on Low Power Electronics and Design, Seattle, WA, USA, 23 July 2018; pp. 1–6.
35. Idriss, T.; Idriss, H.; Bayoumi, M. A PUF-based paradigm for IoT security. In Proceedings of the 2016 IEEE 3rd World Forum on Internet of Things (WF-IoT), Reston, VA, USA, 12–14 December 2016; pp. 700–705.
36. Braeken, A. PUF based authentication protocol for IoT. *Symmetry* **2018**, *10*, 352. [\[CrossRef\]](#)