Securing Interconnected PUF Network with Reconfigurability

Hongxiang Gu, Miodrag Potkonjak
Department of Computer Science, University of California, Los Angeles
Los Angeles, CA, USA
Email: hxgu@cs.ucla.edu,miodrag@cs.ucla.edu

ABSTRACT

Physical Unclonable Functions (PUFs) are known for their unclonability and light-weight design. Recent advancement in technology has significantly compromised the security of PUFs. Machine learning-based attacks have been proven to be able to construct numerical models that predict various types of PUFs with high accuracy with a small set of challenge-response pairs (CRPs). To address the problem, we present a reconfigurable interconnected PUF network (IPN) design that significantly strengthens the security and unclonability of strong PUFs. While the IPN structure itself provides high resilience against modeling attacks, the reconfiguration mechanism remaps the input-output mapping before an attacker could collect sufficient CRPs. Experimental results show that all tested state-of-the-art machine learning attack methods have prediction accuracy of around 50% on a single bit output of a reconfigurable IPN.

1. INTRODUCTION

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

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 mathematical model can be easily established by learning from a small subset of CRPs.

In this paper, we intend to address this problem. We propose a reconfigurable interconnected PUF network structure that is capable of providing sufficient robustness and resilient against different types of machine learning attacks. Essentially the idea is to create a network structure that interconnects multiple 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.

2. RELATED WORK

PUF was first proposed by Pappu et al. using mesoscopic optical systems [1]. A variety of PUFs and related applications have been proposed ever since including APUFs [2], ring oscillator PUFs [3], and SRAM PUFs [4]. PUFs have been well studied in many hardware security applications. Gu et al. propose several low power applications based on PUF including computing-while-racing PUF [5] and PUF-based system anomaly detector [6]. Zhang et al. proposed a PUF-FSM binding scheme for IP-protection [7]. Xu et al. proposed an ultra-low energy PUF matching scheme using programmable delay lines [8]. PUFs can also be used to construct Recursive Inverse Functions(RIF) that provide fast and ultra-low energy encryption and decryption for data protection [9].

Though PUFs have been improved over the years, they are vulnerable to a variety of modeling attacks. Early works on modeling attack targeting PUFs were focused on standard arbiter PUFs [10]. Later on Rührmair et al. presented modeling attack results on multiple commonly seen PUFs, including APUFs, XOR PUFs, feed-forward PUFs. The proved that all investigated PUFs are vulnerable to machine learning attacks [11].

To the best of our knowledge, we are the first to propose an easy reconfigurable interconnected PUF network structure to strengthen the security and resilience against machine learning attacks.

3. PRELIMINARIES

3.1 IPN

3.1.1 *IPN Node*

An IPN consists of nodes and edges. A node consists of multiple arbiter PUFs of the same length. We define an n-bit IPN node of size m consists of m n-bit APUFs. If $m = n$, 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. An IPN node takes an n-bit vector as the challenge and generates m 1-bit responses. All PUFs within the same IPN node share the same challenges.

3.1.2 IPN Edge

An IPN node connects to other nodes through edges. In order to achieve reconfigurability, an edge is essentially designed to be a shuffler that takes the output from the previous node, shuffles the order and feed 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 n-bit shuffler that directs the i-th bit of the input to the $n-1-i$-th output bit, the configuration vector would be $\{n-1, n-2, ..., 2, 1, 0\}$. All output port numbers are represented in the 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.

3.1.3 IPN Chain

A simple network can be constructed by connecting IPN nodes using to form a chain. Figure 1 shows a simple IPN chain with three nodes. All 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, then connects to all APUFs in $node_{i+1}$. Thus, each APUF in $node_{i+1}$ depends on the outputs of all APUFs in $node_i$. Figure 1

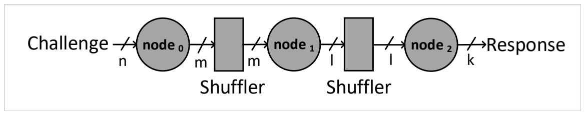


Figure 1: A simple IPN chain with three nodes and two edges. $node_0$ has size m and length n, $node_1$ has size l and length m, $node_0$ has size k and length l. $n \geq m \geq l \geq k$.

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.

We define the depth of an IPN as the length of the shortest path from an input node to an output node. The width of an IPN is defined as the maximum number of nodes that shares the same input.

4. RECONFIGURATION

IPNs benefit from the complex structure so that it requires much larger training set and longer training time to model. We propose to reconfigure 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.

4.1 Reconfigure Timing

Sufficient size of the training set is also known as the sample complexity. We consider models of all PUF-based system mentioned in this paper as a binary function that takes a challenge and generates a 1-bit output of either 0 or 1. 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 below relation:

$$N = O(\frac{VC(\mathcal{H}) + ln(\frac{1}{\delta})}{\epsilon}) \tag{1}$$

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.

For arbiter PUF, the VC-dimension is the total number of stages, meaning for a k-bit arbiter PUF, $VC(\mathcal{H}) = k$. Rührmair 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 2 describe 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. To be noted that we assume each single path within the network to be of width n and depth m.

$$N \sim \frac{(m \cdot k + m) \cdot n + ln(\frac{1}{\delta})}{\epsilon} \tag{2}$$

For each IPN structure, we derive a empirical formula based on equation 2 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 we collected from evolution strategies shows a super-linear relationship between N and ϵ. Thus, we adopt the method proposed by Rührmair et al. and modify the relationship to equation 3 when applying evolution strategies to match the superlinear relationship. c is a constant between 0 and 1.

$$N \sim \frac{(m \cdot k + m) \cdot n + ln(\frac{1}{\delta})}{\epsilon^c} \tag{3}$$

An IPN-based system requires much larger training set comparing to standard arbiter PUFs, XOR PUFs and Feed-forward PUFs of the same size. This can be observed when comparing equation 2 to the lower bounds proposed in [11].

4.2 Reconfiguration Logic

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 4. Instead of assuming the network has the maximum width n on every level (Equation 2), we assume each 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.

$$\Theta = \frac{m \cdot k \cdot n' + ln(\frac{1}{\delta})}{\epsilon} \quad (4)$$

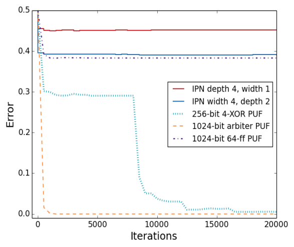
5. EVALUATION RESULTS

Our evaluation is conducted on both simulated models as well as implementations on a Xilinx Virtex-5 XC5VLX50T FPGA. 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.

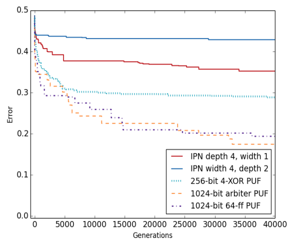
5.1 Logistic Regressions

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 method is negligible.

We maintain the total number of PUF segments used in all settings to be around 1,024. 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 showed in figure 2a is chosen from the best of 100 runs.



(a) Logistic regression attacks result. Error vs. iterations.



(b) Evolution strategies attack result. Error vs. iterations.

Figure 2: Logistic regression and evolution strategies attack result using 30,000 CRP training set on five PUF-based systems.

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 not work at all. (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.

5.2 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. Both canonical versions of evolution strategies were applied to all investigated PUF-based systems, each with 100 runs. The best results among the 100 runs are shown in figure 2b.

Based on the result, we believe it is safe to conclude on two observations. (1) Nonlinear logic functions like XORs dramatically increase the difficulty for evolution strategies attack models, whereas 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.

5.3 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. After some experiment, we find that a MLP with n layers and m_i neurons in each layer provides the best results and speed where n is the depth of the network and m_i is the total number of PUF segments on the i-th level. The activation function we used are ReLU activations and the loss function used is binary cross entropy. 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.

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 1 shows the result of applying MLP modeling to all discussed PUF systems.

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%

Table 1: Deep neural network attack results.

MLP is capable of fitting 30,000 CPRs with above 99% training accuracy, and can predict 256-bit 4-XOR PUF, 1024-bit arbiter PUF and 1024-bit 66-ff PUF with above 95% test accuracy. However, it ran into overfitting problem when modeling IPNs. After attempting various overfitting prevention techniques including regularization layers and dropouts, we conclude that the root of the overfitting problem is insufficient training samples. IPNs is more complex 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 width 4. When applying the same training set, the test accuracy converges at 54.77% and 62.54% respectively, much lower than MLP attack results.

5.4 Other Machine Learning Algorithms

AutoML is still under development, yet it provides promising results compared to MLP attacks in terms of modeling PUF-based systems. We provided only raw CRPs to the auto-sklearn module, and the results for all tested architectures are shown in Table 2.

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	Multinominal NB	89.55%
1024-bit 64-ff PUF	Multinominal NB	72.33%

Table 2: Auto-sklearn modeling results on raw CRPs.

In general, the best classifiers for IPNs are decision-tree classifiers, which is 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 multinominal naive Bayes classifiers) that successfully predicts the test set CRPs with accuracy over 70%.

5.5 Implementation Result

We implemented a 64-bit reconfigurable IPN of depth 4 and width 4 on a Xilinx Virtex-5 board. According to our derived formula on 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 3 shows the prediction accuracy of 10,000 test challenges with and without reconfiguration functionality.

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

Table 3: 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 hours.

6. CONCLUSION

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 to the state-of-the-art methods like deep neural networks and autoML.

To eliminate the possibility of being modeled with a large training set, we propose to make an IPN reconfigurable by shuffling the interconnections between IPN nodes. 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 to use another set of PUFs to protect the configuration vectors from being stolen. Our experimental results indicate that no investigated attack is capable of accurately modeling 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 of time, is around 50%.

7. ACKNOWLEDGEMENT

This work was supported in part by the NSF under Award CNS-0958369, and Award CNS-1059435.

8. REFERENCES

[1] R. Pappu, B. Recht, J. Taylor, and N. Gershenfeld, "Physical one-way functions," *Science*, vol. 297, no. 5589, pp. 2026–2030, 2002.

[2] B. Gassend, D. Clarke, M. Van Dijk, and S. Devadas, "Silicon physical random functions," in *Proceedings of the 9th ACM conference on Computer and communications security*, pp. 148–160, ACM, 2002.

[3] G. E. Suh and S. Devadas, "Physical unclonable functions for device authentication and secret key generation," in *Proceedings of the 44th annual Design Automation Conference*, pp. 9–14, ACM, 2007.

[4] J. Guajardo, S. S. Kumar, G.-J. Schrijen, and P. Tuyls, *FPGA intrinsic PUFs and their use for IP protection*. Springer, 2007.

[5] H. Gu, T. Xu, and M. Potkonjak, "An energy-efficient puf design: Computing while racing," in *Proceedings of the 2016 International Symposium on Low Power Electronics and Design*, pp. 142–147, ACM, 2016.

[6] H. Gu, T. Xu, and M. Potkonjak, "A low-power apuf-based environmental abnormality detection framework," in *Low Power Electronics and Design (ISLPED, 2017 IEEE/ACM International Symposium on*, pp. 1–6, IEEE, 2017.

[7] J. Zhang, Y. Lin, Y. Lyu, and G. Qu, "A puf-fsm binding scheme for fpga ip protection and pay-per-device licensing," *IEEE Transactions on Information Forensics and Security*, vol. 10, no. 6, pp. 1137–1150, 2015.

[8] T. Xu, H. Gu, and M. Potkonjak, "An ultra-low energy puf matching security platform using programmable delay lines," in *Reconfigurable Communication-centric Systems-on-Chip (ReCoSoC), 2016 11th International Symposium on*, pp. 1–8, IEEE, 2016.

[9] T. Xu, H. Gu, and M. Potkonjak, "Data protection using recursive inverse function," in *Field Programmable Logic and Applications (FPL), 2015 25th International Conference on*, pp. 1–4, IEEE, 2015.

[10] M. Majzoobi, F. Koushanfar, and M. Potkonjak, "Testing techniques for hardware security," in *Test Conference, 2008. ITC 2008. IEEE International*, pp. 1–10, IEEE, 2008.

[11] U. Rührmair, J. Sölter, F. Sehnke, X. Xu, A. Mahmoud, V. Stoyanova, G. Dror, J. Schmidhuber, W. Burleson, and S. Devadas, "PUF modeling attacks on simulated and silicon data," *IEEE Transactions on Information Forensics and Security*, vol. 8, no. 11, pp. 1876–1891, 2013.