Puffery: An Open-Source Benchmark Tool for PUFs

Hunter Nichols and Matthew R. Guthaus
Computer Science and Engineering
University of California Santa Cruz
Santa Cruz, CA 95064
{hznichol,mrg}@ucsc.edu

***Abstract*—As concern over security continues to grow within the hardware community, the ability to openly test and analyze these systems is a necessary requirement. An open-source solution encourages a wide range of perspectives and allows thorough inspection of possible faults within systems. Physically Unclonable Functions (PUFs) are hardware primitives which act as digital fingerprints. While existing in the literature for quite some time, their evaluation has no consensus and flaws are constantly found in state of the art designs. This work introduces Puffery, an open-source tool to evaluate PUFs of different designs and build a common, extensible standard for these security devices. Puffery maintains a collection of tests to evaluate PUFs such as Hamming Distance for reliability and learning attacks for vulnerabilities.**

I. Introduction

Meltdown [1] and Spectre [2] are recently discovered hardware vulnerabilities that affect modern processors. Before that, Row Hammer [3] was another hardware vulnerability that allowed for interference through DRAM cell coupling. Vulnerabilities are less common in hardware compared to software but their impact can be much more devastating as fixes in active hardware are near impossible. Issues will continue to appear as long as hardware continues to grow and improve, and we currently rely on careful observers and designers to find vulnerabilities before they can be utilized maliciously.

One hardware security primitive that has been actively explored in the literature is Physically Unclonable Functions (PUFs) [3] [4]. There is a general consensus on several factors that determine the quality of PUFs but no standard exists and each implementation applies a slightly different analysis.

PUFs represent a digital fingerprint for a circuit and cannot be reliably re-created even when knowing the exact hardware used to implement the PUF. This allows for each device to be authenticated by a centralized entity which has accepted the PUF as a known device. The secret key is inherent to the device, so the key is never transmitted and programmed within the device reducing any possibility it can be apprehended by an attacker.

At a high-level, a PUF represents a function which takes in an input challenge and outputs a response. Each PUF maintains a set of challenge-response pairs (CRP). There are two separate classes of PUFs: weak and strong. A weak PUF only maintains one or several CRPs and serves as a replacement to secret key storage e.g. the key is created by the devices rather than programmed externally. A strong PUF maintains a large amount of CRPs such that a subset would be used to authenticate the device.

PUFs are graded on a variety of aspects depending on implementation, but reliability and vulnerability to attacks are key traits. As PUFs are hardware based, reliability is important in every implementation. Ideally, a PUF should be consistent with its output and always return the same CRPs independent of the external conditions. The reality is all implemented PUFs will have varying CRPs. The foundation which determines the differences between devices are the transistor variations which eventually determine if a response bit outputs to 1 or 0. In many cases, noise or circuit conditions can overpower the effect of the variation leading to different responses with the same challenge. Hamming Distance (HD) is a common metric to measure the reliability and is also used as a metric to distinguish distinct PUFs.

Strong PUFs which have a large set of CRPs are mainly at risk of Machine Learning (ML) algorithms being able to accurately produce CRPs from a given device. A key idea behind strong PUFs is that an attacker cannot extract enough CRPs within a reasonable amount of time to be able pose as an authentic PUF. However, it has been demonstrated that many strong PUFs are vulnerable to learning attacks where a subset of CRPs can train an ML model to replicate the PUFs. This is due to most strong PUF implementations demonstrating correlation between challenges and responses. Some theoretical PUF papers doubt the ability to fully guard against learning attacks [5], so guarding access to the CRPs has been recommended in some cases. This is an easily exploitable vulnerability which can be replicated with a variety of open-source machine learning packages. This issue poses a major hurdle for the security of strong PUFs.

Literature of PUFs use these two categories as the most common analysis, however, every PUF implementation is different and uses its own set of tests. A common platform is needed to grade these devices to define and allow free comparison. An open-source development allows for two advantages. First, anyone can see the benchmarks and tests being performed on the PUFs, and second, anyone can contribute, discuss, and add to the project to improve it for future designs. Many PUFs which claim to be resistant to learning attacks are often found vulnerable at a later date, so updating the standard is required to reform to the ongoing security arms race between attackers and defenders.

We present the Puffery testing benchmarks as a beginning

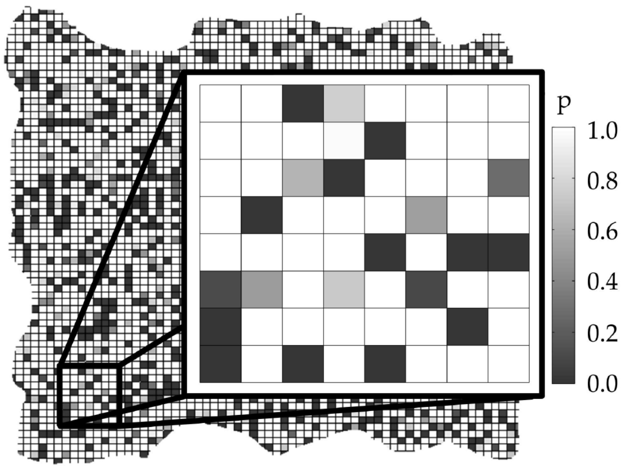

Fig. 1: Each cell of the PUF represents a probability, p, of the cell powering to a logical “1”. Some cells can appear random and are compensated using averaging techniques [7].

to evaluate PUFs under various metrics depending on type and design. Puffery debuts with evaluation of a strong and weak PUFs with learning attacks and Hamming Distances tests, respectively. All tests are in Python with widely accessible packages. Both PUFs were generated using OpenRAM [6].

II. PUF Overview

There are many implementations that focus on other design aspects such as low-cost or low-power, but any PUF can be generalized into the weak or strong categories. Weak and strong PUFs can tested in different ways, but in this section, a few representative tests and reasoning for these tests will be presented.

A. Weak PUF Analysis

Weak PUFs are categorized as only having one or several CRPs. An SRAM PUF is a common implementation of a weak PUF as the start-up values of the SRAM cells are determined by the variation differences in the cross-coupled transistors and are usually uncorrelated from device to device. There are other types of weak PUFs, but SRAM PUFs are the most common and considered the easiest to incorporate as it can be implemented with standard SRAM. As shown in Figure 1, a single SRAM may be separated into multiple PUFs. Each bit has a probability to startup as a “0” or “1” which is dependent on pre-existing variation, environmental factors, and noise sources. An SRAM PUF may use the address to separate different PUF fingerprints, or the challenge can be used as the input for the address as well.

The main metric for weak PUFs is to measure the intra-HD and inter-HD. The intra-HD represents the difference in response bits on separate measurements. Variations within the bitcell may cause the cell to be neutrally skewed and be susceptible to noise. Each bitcell will have random skew with cells skewed towards “0” or “1” being more reliable and consistent start-up values. Weak PUFs requires reliability

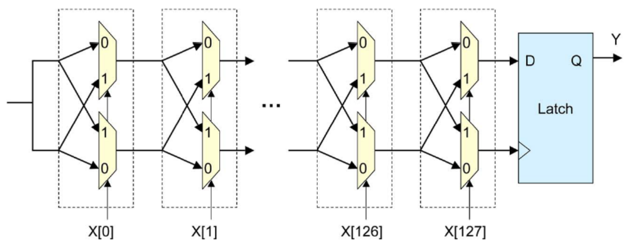

Fig. 2: An example arbiter PUF with 128 challenge bits and a single response. Structure can be replicated for as many response bits as desired [10].

analysis with Hamming distances in order to guarantee a correct authentication with the PUF. The intra-HD variance must not be large enough to be confused with a logically separate devices.

Inter-HD is the Hamming distance of separate PUFs. A perfect PUF would have an intra-HD equal to 0 and an inter-HD equal to half the number of response bits. Most implementations report an intra-HD equal to 5-20% of total response bits which can heavily vary by implementation and fabrication. To account for this small error, an average of several start-up values are averaged together to create a known fingerprint and compared with latent fingerprints from subsequent startups. Hamming distance is one of the most common metrics to measure the reliability of a PUF but error correction can also be employed to improve their reliability [8].

B. Strong PUF Analysis

Strong PUFs have many CRPs, and a key property is the area of the design should be, at most, linear with the total challenge bits such that the total challenges increase exponentially with the area. Otherwise, it would be difficult to efficiently create a large enough set of CRPs to ward off attackers. Using an SRAM to implement a strong PUF would be inefficient unless additional circuitry is added [9]. An arbiter PUF is a simple strong PUF built with cascading multiplexers connected to a flip-flop as shown in Figure 2. The input challenge determines the path through the MUXes, and a competing delay between two signals determines the final output on the flip-flop. This design has known vulnerabilities to learning attacks which has spawned different modifications and theory for resilient strong PUFs.

Learning attacks use ML models to find correlation in bits to predict unknown CRPs. The arbiter PUF output depends on a sum of delays, so a model can attempt to determine how each challenge bit affects the output. Given enough CRPs, it is easy for many ML models to achieve over 90% accuracy through training the models on known CRPs. Most learning attacks can be developed using commonly available open-source packages.

The arbiter PUF is known to be susceptible to learning attacks [10] as each stage produces a randomized delay, but the final output is dependent their linear summation. Linear

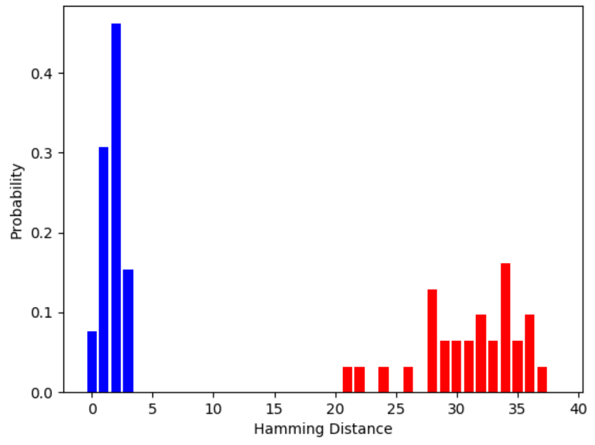

Fig. 3: The intra-Hamming Distance (blue) and inter-Hammering Distance of a 64-bit SRAM PUF (red). As long as the intra-HD and inter-HD are distinct then the device can be identified.

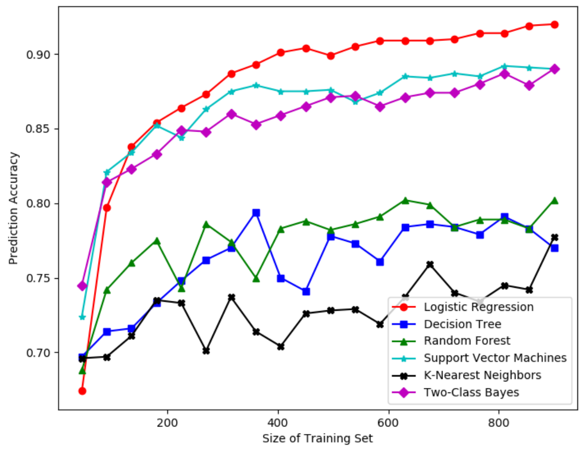

Fig. 4: Accuracies of different models over varying training data. The models average an accuracy over 80% while only knowing less than 1% of all possible CRPs.

system analysis can be used to gather information about the PUF and effectively model the PUF even if only given a small amount of CRPs. However, many standard machine learning models can also easily learn the PUF without understand the internals mechanics behind its operation.

Support Vector Machines and Logistic Regression are two popular ML models used to characterize the PUF against learning attacks. Each model attempts to find patterns in the challenges and responses using supervised learning techniques. ML is one of the fastest growing areas of research, and so, a copius amount of open-source software and packages have been recently released to support exploration. TensorFlow, PyTorch, and scikit-learn are a few examples that all have easily available Python packages with varying amounts of abstraction. With only a few lines of code, a model built with scikit-learn can learn any strong PUF with high accuracy depending on the implementation. The simplicity of this attack makes it an important metric that any strong PUF should be compared against.

There are nearly an unlimited amount of model configurations and hyper parameters that can be configured in order to tune the model to be more accurate, but in most cases, more well known and tested ML models are primarily used. PUF analysis is not at the point where a single model dominates all others. Therefore, the literature tends to use varying packages and models to explore which models are the most effective. Puffery uses scikit-learn as it represents bundle of common models which are easy to use. Utilizing Tensorflow could allow any model to be adapted, but that is left for a future update as the customization is not needed at this time.

III. Results

Puffery is a set of scripts implemented in Python. The language has support from many open-source projects, and specifically, Puffery makes use of the scikit-learn packages for strong PUF analysis and Matplotlib for displaying the results. Puffery is released publicly on Github[1] and is in active development. To use the tools, you only need to provide the CRPs and indicate in the scripts where the data is located.

OpenRAM was modified to generate data for weak and strong PUFs. An SRAM PUF was created to represent the weak PUF while an arbiter PUF module was added and characterized in OpenRAM to represent the strong PUF. Variation was applied using HSPICE variation options to adjust the voltage threshold of each transistor based on its area. Designs and simulations were created in FreePDK45 technology.

A. Hamming Distance

For weak PUFs, Puffery has a script to display the intra-HD and inter-HD of a PUF. Puffery provides data generated from a 64-bit SRAM as its weak PUF. Inter-device data was generated using the same design but with different HSPICE seeds to change the variation. Intra-device data was generated with a single seed but the supply voltage was varied between 80%-100%, and temperature was varied between 0-100 Celsius. Figure 3 displays the plot generated by Puffery with this data. The plot can be generated with any available data as long as it has a similar format to the existing example.

B. Hamming Distance

Puffery evaluates PUF CRPs on six different binary classification models provided by scikit-learn: logistic regression, support vector machines, k-nearest neighbors, decisions trees, random forest, and a two-class Bayes model. The current strong PUF evaluation uses these modules and performs training and testing on various amounts of the available data provided. This is one of the most useful metrics as is

[1]https://github.com/VLSIDA/puffery

demonstrates how accurate the models get even with limited data. The data provided by Puffery was generated by a 16-bit challenge, 1-bit output arbiter PUF at nominal conditions. Rather than use a flip-flop, both paths were measured and the response was determined post-simulation based on the delay. Figure 4 displays the plot generated by Puffery of prediction accuracy with different models. There is a clear accuracy divide between the models with logistic regression reaching as high as 90% without including any internal information about the PUF.

IV. Conclusion

Security in hardware is difficult task and there will be no one solution to prevent all possible exploits in the future. PUFs are just one technology that helps embed safety within the hardware that is performing authentication. Puffery presents an open analysis for PUFs and is almost entirely done with open-source tools. The tests represent a single link in the chain to help make hardware security an open-source effort and eliminate the need to meticulously reverse-engineer vulnerabilities.

References

[1] M. Lipp, M. Schwarz, D. Gruss, T. Prescher, W. Haas, A. Fogh, J. Horn, S. Mangard, P. Kocher, D. Genkin, Y. Yarom, and M. Hamburg, "Meltdown: Reading kernel memory from user space," in *27th USENIX Security Symposium (USENIX Security 18)*, 2018.

[2] P. Kocher, J. Horn, A. Fogh, , D. Genkin, D. Gruss, W. Haas, M. Hamburg, M. Lipp, S. Mangard, T. Prescher, M. Schwarz, and Y. Yarom, "Spectre attacks: Exploiting speculative execution," in *40th IEEE Symposium on Security and Privacy (S&P'19)*, 2019.

[3] M. Seaborn and T. Dullien, "Exploiting the dram rowhammer bug to gain kernel privileges," *Black Hat*, vol. 15, 2015.

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

[5] A. Vijayakumar, V. C. Patil, C. B. Prado, and S. Kundu, "Machine learning resistant strong puf: Possible or a pipe dream?" in *2016 IEEE international symposium on hardware oriented security and trust (HOST)*. IEEE, 2016, pp. 19–24.

[6] M. R. Guthaus *et al.*, "OpenRAM: An open-source memory compiler," in *ICCAD*. New York, NY, USA: ACM, 2016, pp. 93:1–93:6.

[7] D. E. Holcomb, W. P. Burleson, and K. Fu, "Power-up sram state as an identifying fingerprint and source of true random numbers," *IEEE Transactions on Computers*, vol. 58, no. 9, pp. 1198–1210, 2008.

[8] M.-D. Yu and S. Devadas, "Secure and robust error correction for physical unclonable functions," *IEEE Design & Test of Computers*, vol. 27, no. 1, pp. 48–65, 2010.

[9] D. E. Holcomb and K. Fu, "Bitline puf: Building native challenge-response puf capability into any sram," in *Cryptographic Hardware and Embedded Systems – CHES 2014*, L. Batina and M. Robshaw, Eds. Berlin, Heidelberg: Springer Berlin Heidelberg, 2014, pp. 510–526.

[10] C. Herder, M.-D. Yu, F. Koushanfar, and S. Devadas, "Physical unclonable functions and applications: A tutorial," *Proceedings of the IEEE*, vol. 102, no. 8, pp. 1126–1141, 2014.