Arbiter PUF Software Implementation and Evaluation

Samuel Barkley and Garrett Moncrief
Dept of Computer Science and Engineering
University of South Florida, Tampa, Florida, USA

Abstract - This document gives a general background on physically unclonable functions and attempts at reproducing their functionality through software applications. In this work we will attempt to produce an arbiter model of a physically unclonable function and discuss what elements we feel were adequately reproduced and where room existed for improvement on our design.

I. INTRODUCTION

The concept of using the randomness of physical events as a security measure is present anywhere from the biometrics of a fingerprint to using lava lamps for encryption [1]. Physically Unclonable/Random Functions, hereafter referred to as PUFs, extend this concept into the field of hardware security.

A PUF is based on a physical system that's easy to evaluate and produces an output that looks like a random function, making it unpredictable even for an attacker with physical access. Due to variations in the process, integrated circuits contain sufficient delays to allow this variance to be used for identification purposes. This means there are no integrated circuits that would produce exactly identical responses to a given challenge [2]. Experiments with identical circuits with identical layouts have been placed on different FPGAs to show that path delays are sufficient to use them for identification since path delays are statistically distributed due to random manufacturing variations [3].

In 2017 Intrinsic ID stated that it was working on BroadKey, an attempt at hardware intellectual property protection using software and SRAM behavior. The purpose of this was to secure the IoT devices without the need for security dedicated chips. This would allow BroadKey to be installed at any point in the supply chain, or even be retrofitted onto deployed devices [4].

The chief executive officer of Intrinsic ID elaborated that "BroadKey represents a revolution in key creation, wrapping and management... it is even possible to deliver authentication and encryption solutions to devices in the field through firmware updates."

Considering that software PUF implementation is an emerging field, the goal of this work is to examine the ways that PUFs can be implemented and modelled via code. It will examine metrics that can be simulated within a software only environment, as well as the limitations imposed by such an environment.

II. BACKGROUND

PUFs operate off challenge and response pairs, or CRPs, which they gather during in an enrollment phase and validate in a verification phase. It can be thought of as a lock which can only be opened by a person able to issue a challenge that gives a response on the lock's database [3]. These CRPs face a couple different reliability and security metrics such as inter-chip variation, intra-chip variation, and hamming distance.

Inter- and intra- variations are used for identification purposes. For a given challenge the inter-distance is giving the same challenge to two different PUFs and measuring the difference in the responses while intra-distance is the measure of the same challenge being repeatedly given to a single PUF and measuring the difference of the result [5]. This means that inter-distance is a measure of uniqueness, which the variation should be 50% on average, and intra-distance is a measure of output reproducibility with an ideal variation of 0% [3].

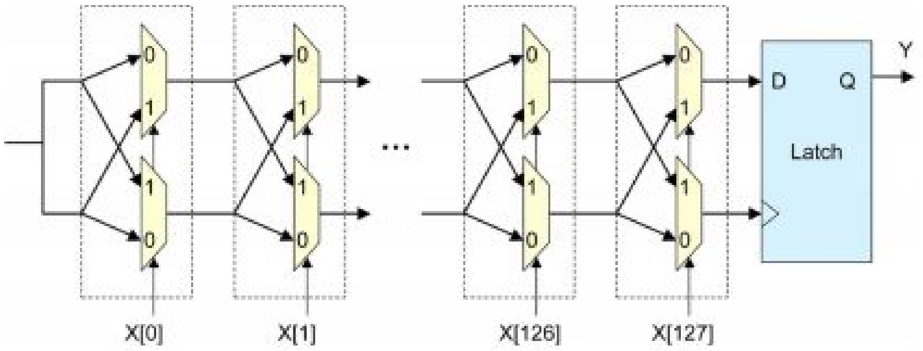

Fig. 1 displays an arbiter PUF circuit. The two paths each have the same layout so that when each are given an input X they will produce a Y output based on which path was faster [7].

Hamming distance is used when the response is a bit string. Hamming distance is the number of bits flipped in String A compared to String B, and represented as a percentage of the number of different bits vs the number of total bits [3]. In other words Hamming distance refers to the points at which two equal length lines of binary code differ from each other, meaning that strings with a Hamming distance of four indicates that four bits would need to be changed to make the lines identical [6].

As mentioned in the introduction, the digital fingerprint aspect of an integrated circuits relies on path delays due to process variation. These analog properties can change due to environmental factors such as temperature and supply voltage. As these parameters vary so does the digital fingerprint, and too much variance can change the digital key or cause the failure of cryptographic operations [7].

There are many instantiations of PUFs: non-electronic PUFs, analog PUFs, and delay-based intrinsic PUFs. During the hardware security course the type of PUF most frequently discussed was the "Delay-based Intrinsic PUF", of which the there are Arbiter and Ring Oscillator. An intrinsic PUF must meet two prerequisites: the PUF and measuring equipment are fully integrated in the embedding device, and PUF construction should not consist of specialized components or require extra manufacturing steps [8].

PUFs can also be categorized as weak or strong PUFs. The difference between the two is that strong PUFs can handle a greater number of CRPs, allowing for direct authentication rather than relying on additional cryptographic hardware [7].

III. METHODOLOGY

Our software based implementation and evaluation will focus on a strong arbiter PUF. The basic design of an arbiter PUF (Fig. 1) is two symmetric paths through the chip and a circuit to evaluate which path was completed first, the random and device specific delays between paths creating the response to given challenges.

The project will consider two different implementations of an arbiter PUF and test the inter- and intra- variations of each using the rand() pseudo-random number generator to generate 128 bit challenge of 0's and 1's.

Environmental variations of the type that could be caused through temperature or supply voltage needed to be abstracted due to the fact that these variations wouldn't be present in a solely software based implementation. This element is replaced with a Gaussian distribution range of +/- 10% of 0.1205ns. Choosing this abstraction is not arbitrary, as it has been noted that the delay difference of each MUX will follow a Gaussian distribution [9].

The first model implemented and tested was the 128 arbiter PUF. The first test examined the inter-distance hamming of 10 PUFs, each containing 128 sets of 128 MUX pairs and an arbiter. Each arbiter within a PUF was assigned the same key while each PUF was assigned a different key to check variation.

The test by default runs 10 times, representative of 10 PUFs. Each loop builds a 'new' PUF by first establishing a 128 bit challenge key randomly as described above. Next each PUF is tested by checking the challenge key against the MUX pairs and arbiter. This means that the PUF would only need to be run once to generate the 128 bit challenge response.

This involves first establishing the delays between each MUX pair by building two vectors from the Gaussian distribution range of +/- 10% of 0.1205ns. The PUF then iterates through the 128 pairs of MUXes and corresponding arbiters.

When the challenge bit is 0 it adds the upper delay to A and the lower delay to B, otherwise when the challenge bit is 1 the lower delay is added to A and upper delay added to B. At the end a final check acts as the arbiter choosing which path 'won' with the shortest delay and in return latching a 1 or a 0 to a response key.

The effect is having the circuit displayed in Fig. 1 running 128 times in each integrated circuit, each arbiter returning a portion of the response.

The second model was the intra-distance version of the 128 MUX pair arbiter, and was tested using a similar format. Each PUF is assigned a unique key with one bit difference. Rather than testing the challenge keys against other PUFs it tests the challenge key versus the same PUF.

The third model provided a variation of the inter-distance arbiter from the prior version that would utilize a single arbiter and 128 MUX pair setup along with a linear-feedback shift register and single challenge key. The key is processed through the MUX pairs as described in the prior model and is then loaded into the shift register. The 128th bit is popped off and the remaining register is shifted to the right. The removed bit is XOR'd against the bit in the 126th position, the result of which is then XOR'd against the bit in the 101st position, which is then XOR'd with the bit in the 99th position [12]. This result is then added to the beginning of the string of number in the register and the process repeats for all 128 bits.

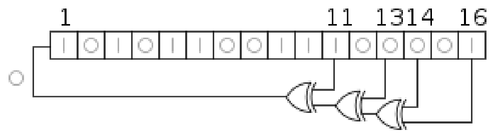

Fig.2 displays the concept of how a linear feedback shift register operates, with our implementation using 128 bits and the 128th, 126th, 101st, and 99th bit positions.

The final model was the intra-distance variation of the linear-feedback shift register PUF as described above. Similar to the second model, this model would also make comparisons against itself rather than the other 9 PUFs, thereby giving a measure of the reproducibility of the intra-distance implementation.

For comparing results we went with comparing every response to each other not just the original to the other responses. Instead of n-1 comparisons for each test we would test every result against each other giving us a total of $\frac{n(n+1)}{2}$ comparisons for each trial.

For inter-distance models the challenge key was provided to the PUF being tested and the response was compared against the response given by the other 9 PUFs when given the same key. This gives a measure of the uniqueness of the CRPs. A hamming distance of 0% would indicate the responses to the challenge were identical while 100% would mean that every response bit was different. The goal was 50%.

For intra-distance models each of the 10 PUFs are once again tested to provide n(n+1)/2 total comparisons, but this time the test runs are against itself. By seeing whether the same key ran through the same PUF will return a similar result, providing for a measure of the circuit's reproducibility. Here the hamming distance goal was ideally 0%.

A hamming distance is calculated by finding the difference between each CRP result. A total hamming distance is also provided for the total number of runs.

IV. RESULTS

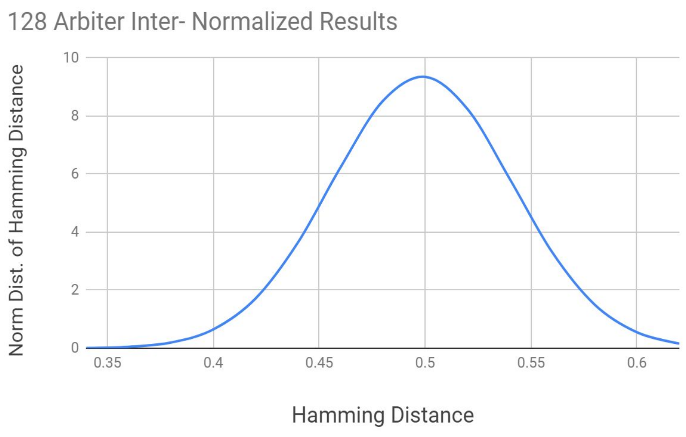

Fig. 3 shows the normalized 128 MUX pair Arbiter Inter-distance results.

Out of 1,000 run comparisons the hamming distance averaged to 49.85%

The minimum result was 38.28% and the maximum result was 64.06%.

The standard deviation of the dataset was 4.26%.

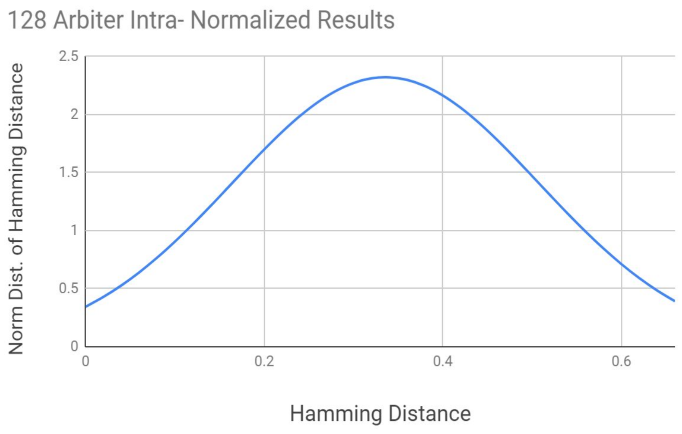

Fig. 4 shows the normalized 128 MUX pair Arbiter Intra-distance results.

Out of 1,000 run comparisons the hamming distance averaged to 33.60%

The minimum result was 0% and the maximum result was 64.06%.

The standard deviation of the dataset was 17.18%.

As mentioned in the Section II the idea inter-hamming distance should be 50%, meaning that for each CRP there should at least be a 50% variation between the bits. The 128 MUX pair Arbiter PUF was able to get almost this exact number over a course of 1,000 run comparisons.

The intra-hamming distance is far from the ideal, and this is due to the randomly generated nature of our path delays between MUXes. A physical implementation would typically not have such a wide range of possible delay variance as long as variables such as temperature, device age, and supply current remained constant.

The necessity to abstract these variables created a large standard deviation in our results, and while lower than the inter-distance variation it is far from the 0% ideal that would be found in a conventional hardware PUF or even the 12% variation in an SRAM PUF, as described below in the results section.

What this means is that based on this models a PUF could receive a CRP and provide enough variance to identify uniqueness, however the CRP would not be able to reliably reproduce the results within the same system.

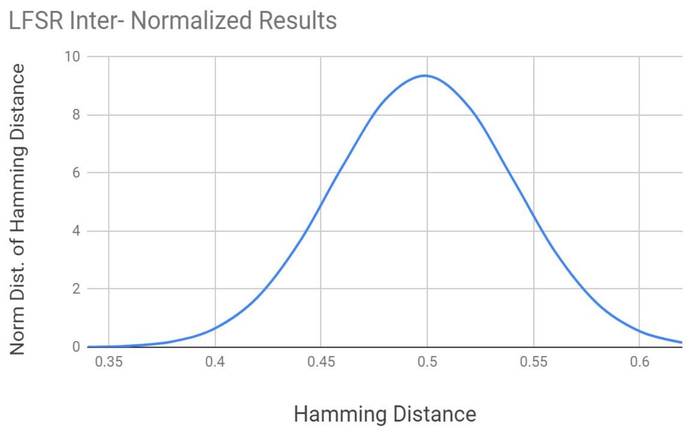

Fig. x shows the normalized Linear-Feedback Shift Register Arbiter Inter-distance results.

Out of 1,000 run comparisons the hamming distance averaged to 49.99%

The minimum result was 32.03% and the maximum result was 69.53%.

The standard deviation of the dataset was 4.41%.

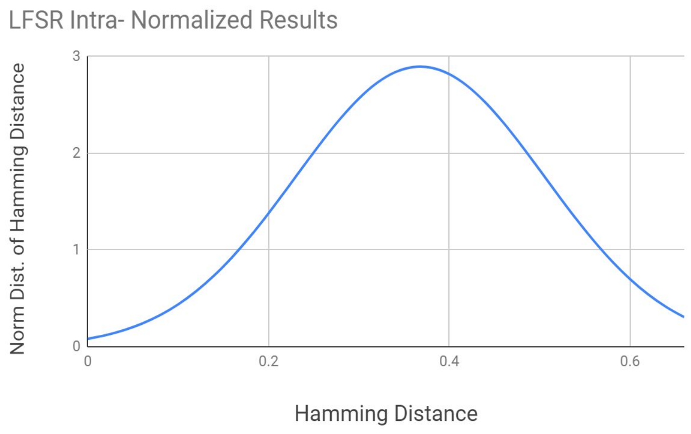

Fig. x shows the normalized Linear-Feedback Shift Register Arbiter Intra-distance results.

Out of 1,000 run comparisons the hamming distance averaged to 36.78%

The minimum result was 2.35% and the maximum result was 66.41%.

The standard deviation of the dataset was 13.78%.

The linear-feedback shift register model provided similar results to both inter- and intra-distance variation, and though the intra-distance model provided a tighter standard deviation the average and maximum results were still far out of range for an ideal PUF implementation. This shows that the intra-distance variation wasn't due to the presence of multiple arbiters or MUX pair paths but rather the random nature of the paths between them.

V. CONCLUSION

Through our attempts to abstract away the device delays it quickly became apparent the role that the SRAM component played in BroadKey's implementation. The startup values of uninitialized SRAM memory provide an average of 49.97% inter-distance variation and <12% intra-distance variation [10]. It is through this that BroadKey is able to implement its PUF utilization at any stage of a product's life cycle.

Without any physical basis to provide the delay times our model needed to randomly generate these path delays. That means that the same path could provide the shortest delay possible for one run and the greatest delay possible for the next, depending solely on the number generated. A physical system would provide more static delays between these paths.

Consideration was given to biasing path connections once they were established so that they would only fluctuate in range slightly after being established.

We were hesitant to provide any additional restrictions such as these since so many facets of our implementation were being abstracted. Already our models were 100% reliable, which is not realistic for a real world situation. Extreme temperatures or power fluctuations could cause for failure in a hardware PUF, and these were not reproducible without simply providing variables that were designed to fail.

While discussing the alternatives to FPGA implementation the idea of creating software modelling of an arbiter PUF was proposed. While trying to tie our models in with some of the emerging PUF technologies that can be implemented at any point of a product's life cycle was an interesting exercise, ultimately without at least accessing SRAM the 'physical' aspect of the PUF was largely lost.

By modelling the way an arbiter PUF worked it highlighted the necessity of the minor variations in a physical device that were necessary to establish a solid "fingerprint" of the device in use.

As discussed in our demonstration it was difficult to successfully model reproducibility with random delay generations. And if we were to bias paths or limit the randomness it would impact the variance of the CRPs, which we were informed was the most important of the two metrics to demonstrate in our model.

Ultimately we entered into the project aware that it wasn't likely that a PUF could be fully modelled in software, and the shortcomings discovered during the project helped us to appreciate the role that PUFs play in hardware security.

REFERENCES

[1] E. Airhart, "How a Bunch of Lava Lamps Protect Us From Hackers," Wired, 01-Aug-2018. [Online]. Available: https://www.wired.com/story/cloudflare-lava-lamps-protect-from-hackers/. [Accessed: 26-Apr-2019].

[2] S. Devadas, "Physical Unclonable Functions and Applications." [Online]. Available: http://people.csail.mit.edu/rudolph/Teaching/Lectures/Security/Lecture-Security-PUFs-2.pdf. [Accessed: 26-Apr-2019].

[3] R. Karam, "PUFs and TRNGs," 2019. [Online]. Available: https://usflearn.instructure.com/courses/1364787/files/folder/Lecture Slides?preview=74562452. [Accessed: 26-Apr-2019].

[4] G. Prophet, "Software-only PUF can 'secure the IoT'," Smart2.0, 09-Feb-2017. [Online]. Available: https://www.smart2zero.com/news/software-only-puf-can-secure-iot. [Accessed: 26-Apr-2019].

[5] R. Maes and I. Verbauwhede, "Physically Unclonable Functions: A Study on the State of the Art and Future Research Directions," Information Security and Cryptography Towards Hardware-Intrinsic Security, pp. 3–4, 2010.

[6] K. Burch, "How to Calculate Hamming Distance," Sciencing, 02-Mar-2019. [Online]. Available: https://sciencing.com/how-to-calculate-hamming-distance-12751770.html. [Accessed: 26-Apr-2019].

[7] 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.

[8] R. Maes and I. Verbauwhede, "Physically Unclonable Functions: A Study on the State of the Art and Future Research Directions," Information Security and Cryptography Towards Hardware-Intrinsic Security, pp. 5–14, 2010.

[9] R. Pegu and R. Mudoi, "Design and Analysis of Mux-based Physical Unclonable Functions," International Journal of Engineering Research and, vol. V4, no. 05, 2015.

[10] R. Maes and I. Verbauwhede, "Physically Unclonable Functions: A Study on the State of the Art and Future Research Directions," Information Security and Cryptography Towards Hardware-Intrinsic Security, pp. 16, 2010.

[11] Karam, R., Dr. PUFs and TRNGs. Lecture presented at CIS 4930 in USF, Tampa.

[12] Alfke, P. (n.d.). Efficient Shift Registers, LFSR Counters, and Long PseudoRandom Sequence Generators. XAPP 052 July 7,1996.

[13] Machida, T., Yamamoto, D., Iwamoto, M., & Sakiyama, K. (2015). A New Arbiter PUF for Enhancing Unpredictability on FPGA. The Scientific World Journal, 2015, 1-13. doi:10.1155/2015/864812

[14] Tajik, S. (2017). On the Physical Security of Physically Unclonable Functions. T-Labs Series in Telecommunication Services. doi:10.1007/978-3-319-75820-6