Implemention and evaluation of a 32 bit 8-XOR-PUF on a PYNQ-Z2 board

Xander Lenstra (s1935534)

December 12, 2023

1 Introduction

This report is written for the second assignment for the course System and Software Security. For this assignment, we programmed multiple 32 bit XOR Arbiter PUF onto an FPGA, specifically a PYNQ-Z2 board. We used this A-PUF to generate results for approximately 1 million challenges. We then evaluated the quality of this dataset, and used an existing technique for preforming a modelling attack on XOR Arbiter PUFs on this dataset. In this report, we will start with explaining the background of our project. We will then follow up with some information about our specific implementation. Afterwards, we will discuss the specific experiments we preformed for evaluating the quality of our dataset and for the modelling attack.

All code used for this assignment as well as all data generated can be found on GitHub: `https://github.com/xlenstra/SSSasg2`.

2 XOR Arbiter PUFs

A PUF or Physically Unclonable Function is a piece of hardware. It is created in such a way that its exact behaviour cannot be deterministically determined from its schematics, but instead is a result of slight variations in the hardware on which the circuit is implemented. As a result, a PUF should be unique between every hardware implementation, which is a very useful property for, for example, key generation and authentication.

A often-used version of the PUF is the Arbiter PUF, where an n-bit challenge can be inputted and a single response bit is outputted. A pair of a challenge the corresponding response is often called a Challenge-Response-Pair or CRP. These Arbiter PUFs (A-PUFs) have since been expanded to more complicated circuits, most notably for this report, k-XOR-PUFs. A k-XOR-PUF consists of k Arbiter PUFs which all take the same challenge. Their responses are merged through a k-XOR-gate to compute the final response. This is specifically useful in some implementations, such as the one we used for our project, where the output of a single A-PUF is a linear function in the challenge. The XOR-gate, as a non-linear function, destroys this linear relationship, making it more difficult to create a model for the PUF.

There are multiple ways to attack an Arbiter PUF. One of the most common is a modelling attack, where attackers try to create a model of the PUF, either mathematical or in software. This model can be obtained either by analyzing the hardware of the PUF directly or by analyzing the CRPs produced by this PUF. In our experiments, we only preformed an attack by analyzing CRPs.

3 Implementation

For our implementation, we have chosen to use A-PUFs that each take a 32 bit challenge. This choice was made due to the PYNQ-Z2 board we used for our implementation only allowing 64 bits of input from outside the internal system, and also 2 bits being required for the startup and reset sequence. While this is lower than the current researched standards of 64 bits or 128 bits[Wis+22], it still results in a complex circuit. Each of these APUFs consists of 32 bits switch blocks, each of which is implemented using two multiplexers, a common approach also proposed previously by Lee et al. [Lee+04].

Each multiplexer is implemented in the hardware of the PYNQ-Z2 board on a single 6LUT (6-bit Look-Up Table). Pairs of these multiplexer making up a single switch block are placed close together, while multiplexers of adjacent switch blocks are placed far apart. This is done to make sure that the time difference travelling between different switch blocks far outweighs the minor effects of which multiplexer the signal travels through. The arbiters for each of the A-PUFs are implemented using the Flop & Latches, each of which is effectively used as a D-flipflop.

Our implementation combined up to 8 A-PUFs into a k-XOR-PUF, outputting the response for the first A-PUF, the XOR of the first two A-PUFs, the XOR of the first three A-PUFs, etc., up to the XOR of all 8 A-PUFs. This way, we collected $2^{20} = 1048576$ million CRPs for all these k-XOR-PUFs at a speed of approximately 100 CRPs per second. This dataset is included with this report as '*training-CRPs-1to8.npy.* Additionally, we also generated an extra $2^{14} = 16384$ challenges to be used as a validation test set. These are saved in the file '*testing-CRPs-1to8.npy*'.

It turned out that during the running of our k-XOR-PUFs, the most time was spent on flipping input bits. We thus implemented a simple heuristic to reduce the amount of bit flips between adjacent challenges by swapping adjacent challenges if this would reduce the amount of bit flips required. Instead of the approximately 16 bits one would expect two adjacent challenges to differ on average, this simple heuristic reduced the amount of bit flips to only 10.3 on average.

4 Experiments

Using these CRPs, we tested an existing attack by Mursi et al. [Mur+20]. This attack was implemented in the python *pypuf* library, created by Wisiol et al. [Wis21]. This is a modelling attack using a dense neural network with three layers, of size $2^{k-1}, 2^k$ and 2^{k-1}, where k is the number of A-PUFs in the k-XOR-PUF. We have tested this attack on the CRPs generated by our hardware model. Additionally, we compared these results with the ones obtained by Wisiol et al. when they previously reimplemented the attack by Mursi et al. [Wis+22]. It should be noted, however, that our results use 32-bit k-XOR-PUFs, while both Mursi et al. and Wisiol et al. used 64-bit k-XOR-PUFs. For each value of k, we tried to find the number CRPs so that within 128 epochs of training the accuracy on a test set would be more than 0.9. The results can be found in Table 1.

It is notable that the amount of CRPs required for the attack is significantly lower. However, this is very likely due to our XOR-PUF consisting 32-bit A-PUFs, while the XOR-PUF Wisiol et al. attacked consisted of 64-bit A-PUFs. The difference in accuracy is likely explained by the fact that our XOR-PUFs are a bit noisy due to it being a hardware implementation, combined with our testing stopping when finding an accuracy over 90% making it less likely to find higher accuracies.

k	CRPs (our dataset)	accuracy (our)	CRPs (Wisiol et al. [Wis+22])	accuracy ([Wis+22])
1	2048	0.9603	-	-
2	32768	0.9775	-	-
3	16384	0.9567	-	-
4	19483	0.9390	150k	0.970
5	19483	0.9468	200k	0.973
6	27554	0.9350	2M	0.975
7	32768	0.9053	4M	0.975
8	65536	0.9047	6M	0.955

Table 1: Required number of CRPs to model PUF and accuracy on model

Additionally, we generated a few statistics for our generated training CRP set. These can be found in Table2 2, 3 and 4. From Table 2, one can see that all A-PUFs respond approximately in equal amounts with 0 and 1, but there is still a slight bias to respond with 1. From Table 3, it is, however, clear that our responses are quite biased. Almost all A-PUFs have agreements of over 60%, which is much higher than the expected agreement of 50%. Lastly, we can see from 4 that the the high correlation between A-PUF responses is not restricted to pairs of A-PUFs, but exists across all A-PUFs. The expected percentage of agreement assuming independence can be seen in Figure 1. It is clear these expected percentages differ wildly from those observed.

While some correlation is not unexpected, the percentages agreement seen in the tables mentioned above are much higher than we would expect. Thus, we can conclude that our generated dataset is heavily biased by correlations between different A-PUFs. It is likely this is a result of the specific way we placed the multiplexers of each A-PUFs on the PYNQ. This is done in a way where multiplexers of different A-PUFs likely share a large portion of the wires between switch blocks, the implications of which we didn't consider enough during the design phase.

A-PUF index	number of '1' responses	percentage of '1' responses
0	536122	51.13%
1	536103	51.22%
2	547976	52.25%
3	536746	51.88%
4	534651	50.99%
5	535148	51.04%
6	534747	50.99%
7	539343	51.44%

Table 2: Number of responses which equal '1' for each individual A-PUF.

index	0	1	2	3	4	5	6	7
0	100.0%	71.6%	62.8%	71.4%	72.9%	72.7%	69.3%	63.3%
1	71.6%	100.0%	62.9%	69.5%	70.5%	67.1%	64.5%	58.7%
2	62.8%	62.9%	100.0%	73.2%	70.3%	71.7%	70.2%	65.3%
3	71.4%	69.5%	73.2%	100.0%	67.2%	67.7%	68.9%	62.5%
4	72.9%	70.5%	70.3%	67.2%	100.0%	74.1%	82.3%	75.3%
5	72.7%	67.1%	71.7%	67.7%	84.1%	100.0%	82.3%	77.3%
6	69.3%	64.5%	70.2%	68.9%	82.3%	82.3%	100.0%	81.6%
7	63.3%	58.7%	65.3%	62.5%	75.3%	77.3%	81.6%	100.0%

Table 3: Percentage agreement between all implemented A-PUFs

n	Response count	Response percentage
0	143238	13.7%
1	119859	11.4%
2	103253	9.84%
3	96053	9.16%
4	92908	8.86%
5	97112	9.26%
6	106540	10.2%
7	126024	12.0%
8	163589	15.6%

Table 4: Amount of times n of the APUFs had response '1'

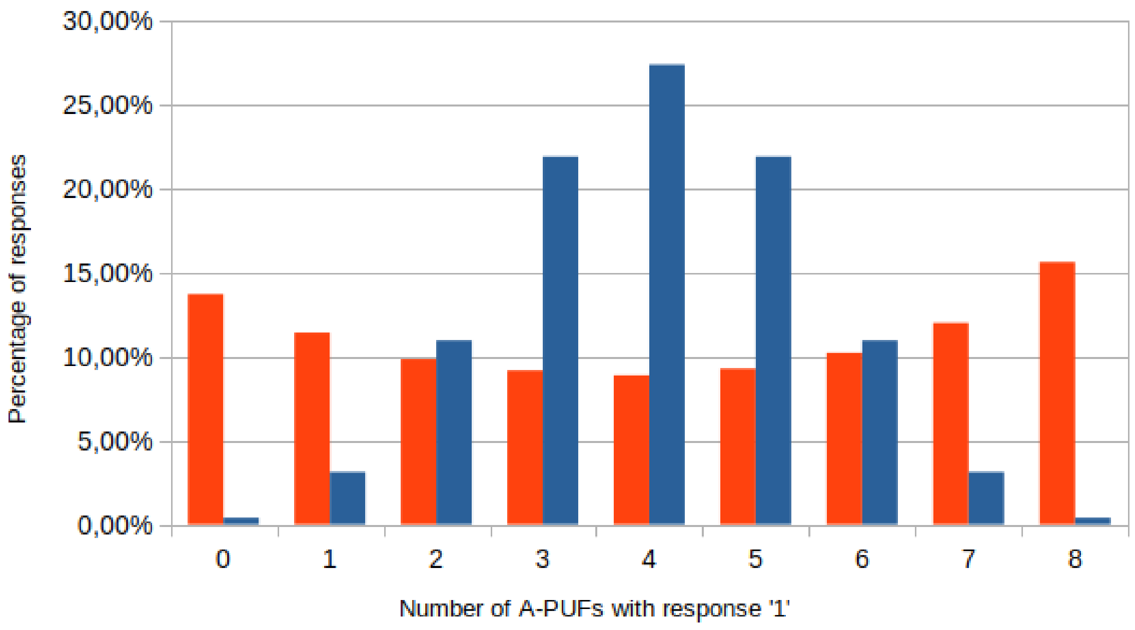

Figure 1: Observed (orange) and expected (blue) percentage of number of A-PUFs with response '1', assuming independence between A-PUFs.

5 Conclusion

In this report we discussed our hardware implementation of 8 32 bit A-PUFs and the corresponding 1- to 8-XOR-PUFs composed of those. We have showed that our implementation is indeed vulnerable to MLP attacks first developed for software. Additionally, we generated around one million CRPs and analyzed these. From here we found that our implementation is heavily biased towards different A-PUFs giving the same response. We believe this to be a result of the specific placement of multiplexers on the PYNQ board, but have not tested this further. This could be looked into in future work. Additionally, future work could be done on finding a faster way to generate CRPs, for example by sorting the input challenges better, or by using a different FPGA.

References

[Lee+04] J.W. Lee et al. "A technique to build a secret key in integrated circuits for identification and authentication applications". In: *2004 Symposium on VLSI Circuits. Digest of Technical Papers (IEEE Cat. No.04CH37525)*. 2004, pp. 176–179. DOI: 10.1109/VLSIC.2004.1346548.

[Mur+20] Khalid T. Mursi et al. "A Fast Deep Learning Method for Security Vulnerability Study of XOR PUFs". In: *Electronics* 9.10 (2020). ISSN: 2079-9292. DOI: 10.3390/electronics9101715. URL: https://www.mdpi.com/2079-9292/9/10/1715.

[Wis+22] Nils Wisiol et al. "Neural Network Modeling Attacks on Arbiter-PUF-Based Designs". In: *IEEE Transactions on Information Forensics and Security* Vol. 17 (2022). DOI: 10.1109/TIFS.2022.3189533.

[Wis21] Nils Wisiol. *nils-wisiol/pypuf: None.* Version v3.1.0. Aug. 2021. DOI: 10.5281/zenodo.5222515. URL: https://doi.org/10.5281/zenodo.5222515.