Delay Model and Machine Learning Exploration of a Hardware-Embedded Delay PUF

Wenjie Che+, Manel Martinez-Ramon, Fareena Saqib*, Jim Plusquellic
Enthentica+, University of North Carolina*, University of New Mexico

Abstract

A special class of Physically Unclonable Functions (PUF) called strong PUFs are characterized as having an exponentially large challenge-response pair (CRP) space. However, model-building attacks with machine learning algorithms have shown that the CRP space of most strong PUFs can be predicted using a relatively small subset of training samples. In this paper, we investigate the delay model of the Hardware-Embedded deLay PUF (HELP) and apply machine learning algorithms to determine its resilience to model-building attacks. The delay model for HELP possesses significant differences when compared with other delay-based PUFs such as the Arbiter PUF, particularly with respect to the composition of the paths which are tested to generate response bits. We show that the complexity of the delay model in combination with a set of delay post processing operations carried out within the HELP algorithm significantly reduce the effectiveness of model-building attacks.

1 Introduction

Symmetric-key authentication and encryption are critically important to system security. Both are based on the notion of a 'shared secret' that only the communicating parties know. Physical unclonable functions (PUFs) have been proposed as an alternative to non-volatile memory (NVM), alone or in combination with cryptographic hash functions, as a mechanism of generating and/or regenerating these shared secrets. However, the requirements for authentication and encryption differ, with the former requiring the existence of an (ideally) unlimited number of shared secrets as a mechanism to protect the system against impersonation attacks. The term strong PUF was coined to represent PUFs with this special property, i.e., the ability to generate an exponential number of challenge-response pairs (CRPs).

Unfortunately, a very large CRP space is a necessary but not a sufficient condition. A PUF used for authentication must also be resistant to modern adversarial attack mechanisms, in particular, those that leverage machine learning (ML) algorithms to first learn the CRP behavior of a PUF using a small set of observations, and then later use the constructed model as a predictor of PUF responses to any challenge. Proving that a PUF is model-building resistant is non-trivial, requiring a complexity analysis of the underlying theoretical model of the PUF to be developed, and then a rigorous set of experiments which show that the complexity of the model requires large training set sizes to be used by ML algorithms to achieve sufficiently low prediction error rates to pass authentication protocol requirements.

In this paper, we investigate the model-building resistance of a strong PUF called the hardware-embedded delay PUF or HELP. Unlike other PUFs which leverage identically designed test structures, such as the Arbiter PUF (APUF), HELP measures path delays in arbitrarily synthesized on-chip macros or functional units such as a multiplier or cryptographic primitive. The random and complex nature of the paths implemented by synthesis tools for mathematical functions increases the complexity of the delay model over models proposed for PUFs with 'identically-designed' circuit structures. The HELP algorithm also batch processes groups of digitized path delay measurements to generate bitstring responses, in contrast to the single challenge-to-bit generation approach used in many other PUF algorithms. The post processing operations carried out by the HELP algorithm provide opportunities to further obfuscate the relationship between the challenges, the underlying source of entropy and the generated response bitstrings.

The main contributions of the paper include an investigation of:

1) Tactics that can be used by adversaries for carrying out model-building attacks on the HELP PUF, derived from the mathematical operations carried out within the HELP bitstring generation algorithm.

2) The effectiveness of these tactics evaluated using several machine learning algorithms.

The remainder of this paper is organized as follows. Related work is described in Section 2. An overview of the HELP algorithm is presented in Section 3. Section 4 presents results of applying ML algorithms using data from the delay post processing operations carried out within the HELP algorithm.

2 Background

The HELP PUF is proposed in [2] and used in a specialized authentication protocol in [3][4]. The authors in [5][6] investigate model-building attacks on typical strong PUFs (APUF and its variants [7]) using various machine learning algorithms. Reference [8] evaluates the effectiveness of modeling attacks on APUFs and indicate that APUFs are not secure for challenge-response authentication. Follow-on work in [9] shows that even PUF-based protocols in which responses are highly obfuscated can be attacked by machine learning. The authors in [10] prove that XOR APUFs can be provably PAC learned in polynomial time. Recent work described in [11] shows that the delay constraints of the APUF can be utilized to expand the known CRP set to facilitate model-building attacks. A statistical metric that evaluates the bias and quality of the APUF is proposed in [12].

3 Overview

HELP leverages within-die variations in path delays as a source of entropy to generate reproducible and unique bitstrings. The overall response generation process for HELP consists of three phases: (1) Path Sensitization, (2) Delay Digitization and (3) Delay Processing, as is depicted in Fig. 1. In the Path Sensitization Phase, a set of 2-vector sequences (challenges) are applied to the functional unit's inputs to sensitize a set of structural paths. The transitions

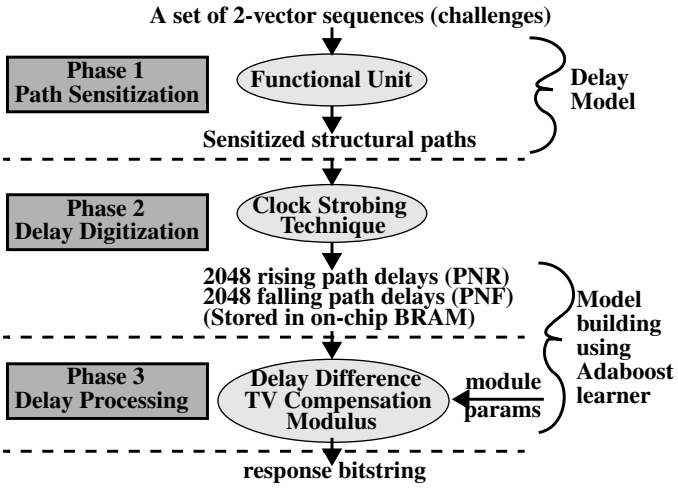

Fig. 1. Top-level overview of the response generation process for HELP PUF consists of three phases: (1) Path sensitization, (2) Delay digitization and (3) Delay processing.

introduced by the 2-vector sequences propagate along a set of sensitized paths and then are captured at the primary outputs (POs) by a set of capture flip-flops. The Delay Digitization Phase utilizes a clock-strobing technique to obtain the digitized delay values of the sensitized paths and stores them in an on-chip BRAM. The 4096 digitized delay values are post-processed in the Delay Processing Phase by a sequence of mathematical operations to generate the response bitstring.

3.1 Path Sensitization Phase

The delay model for most delay-based PUFs is derived within the Path Sensitization Phase since a simple comparison operation is typically used to generate the response bit after path sensitization. For example, a linear additive delay model for the Arbiter PUF is typically used to explain how the delay difference at each stage is constructed and how the delay difference is related to the challenge bit of that stage. The final response bit is derived from the sign of the accumulated delay difference ('1' if positive and '0' otherwise). Therefore, the complexity of the delay model is largely determined by the expressions used to specify how the sensitized path is constructed at each of the stages by the applied challenge.

The challenges in the Path Sensitization Phase for HELP are 2-vector sequences. The task of developing a delay model for HELP requires the relationship between the sensitized paths (each composed of individually sensitized **path segments**) and the applied 2-vector sequences to be specified.

3.2 Delay Digitization Phase

HELP applies a series of launch-capture clocking events (called clock strobing) using Clk_1 and Clk_2 to measure the path delays as shown in Fig. 2. A 2-vector sequence (V1, V2) is applied at the k-bit Primary Inputs, labeled PI, using the Launch Row flip-flops (FF) as a means of generating logic transitions in the functional unit. The first vector V1 represents the initialization vector. The application of V2 generates a set of transitions at the PI, some of which propagate through the functional unit and emerge on the Primary Outputs (PO). For each repeated application of the 2-vector

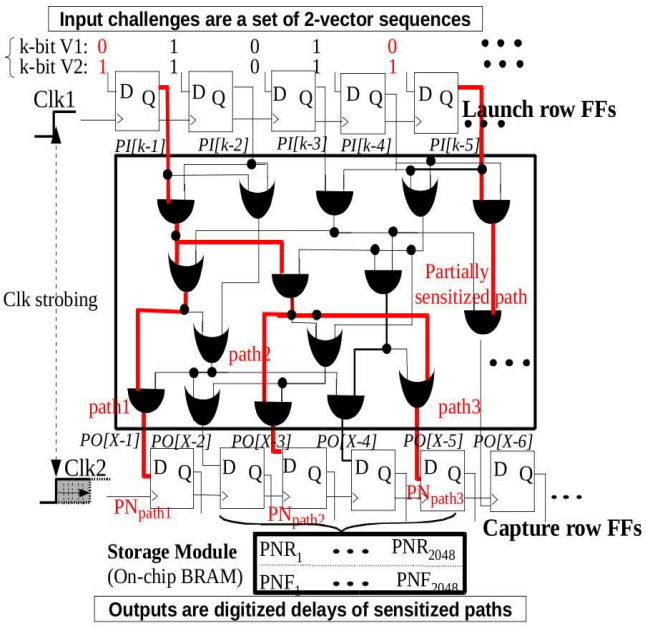

Fig. 2. Configuration of the functional unit for path sensitization and the clock strobing method for delay digitization.

sequence, the phase shift between Clk_1 and Clk_2 is increased by a small fixed Δt. The smallest phase shift value that allows the propagating transition along a path starting from a Launch FF to be successfully captured in a Capture FF is used as the digitized delay value for that path. Note that more than one structural path can be sensitized by a given 2-vector sequence (three are shown in Fig. 2). The Delay Digitization process within the HELP engine is configured to generate 4096 digitized path delays which are used as inputs to the Delay Processing Phase.

3.3 Delay Processing Phase

In contrast to the simple delay comparison operation used by, e.g., the Arbiter PUF, to generate response bits, HELP applies a sequence of operations to the stored delays including Delay Difference, TV Compensation and Modulus (see bottom of Fig. 1). Each of these processing operations is configurable using a set of module parameters. For example, the Delay Difference operation requires two 11-bit *seeds* for initializing two linear feedback shift registers (LFSRs), which are used to pseudo-randomly pair the 4096 delays to create 2048 delay differences. The TVComp operation utilizes reference mean (μ_{ref}) and range (Rng_{ref}) parameters to apply a linear transformation to the entire set of 2048 delay differences. The Modulus operation applies a division operation which makes use of a *Mod* parameter. The two 11-bit LFSR *seeds*, μ_{ref}, Rng_{ref} and *Mod* parameters are called **module parameters**. The values of the module parameters are determined using nonces generated by the token and verifier in the authentication protocol defined for HELP [4]. The parameter selection process within HELP is set up such that the adversary cannot specify (control) these parameters during authentication.

3.4 Model-Building Attacks on the Path Sensitization and Delay Processing Phases

In order to build an accurate model for an instance of the HELP PUF, a machine learning (ML) algorithm must learn the challenge-response (CR) behavior of the PUF. For HELP, the CR behavior is defined by activities which occur in two disjoint phases of the algorithm; the Path Sensitization Phase and the Delay Processing Phase. As discussed above, the inputs to the Path Sensitization Phase are a set of 2-vector sequences (challenges) and the outputs are a set of 4096 digitized path delays. The inputs to the Delay Processing Phase are the 4096 path delays and module parameters, and the output is the response bitstring. More specifically, ML must implicitly learn and predict (1) which structural paths will be sensitized under a given set of challenges during the Path Sensitization Phase, and (2) what the response bitstring will be for a chip-specific set of 4096 path delays and module parameters in the Delay Processing Phase.

The adversary can carry out an attack using any of the following scenarios. The first approach is to train a ML algorithm using CRPs, on the premise that it is capable of learning the behavior of both phases together. The benefit of this traditional strategy is that it does not require any knowledge of the structural characteristics of the functional unit. However, the delay model for HELP is complex, so this strategy is likely to yield poor results.

A second strategy is to simplify the attack by pre-processing the challenges using simulation experiments to determine which paths are actually tested (sensitized). The ML algorithm can then be directed to learn only the behavior of the Data Processing Phase. Obviously, this strategy requires knowledge of the implementation which might be difficult or impossible to attain. Moreover, unlike PUFs that use 'identically-designed' test structures, the details of the structural paths can vary widely depending on the synthesis tool (and even when using the same synthesis tool) so reverse engineering or IP theft will be required to obtain the structural (layout) details of the implementation.

The third tactic is to derive Boolean expressions that describe the functional behavior for each output and use an equation solver to determine which paths are sensitized. In order for this to be effective, the structural netlist must be available (as is true for the simulation approach). The Boolean expressions can become very complex, are ambiguous in at least some cases regarding which path is actually sensitized and are proportional in number to the number of paths in the functional unit (approximately 8 million in the current version of HELP).

As is true of the Arbiter PUF, the ML algorithm is ultimately tasked with learning the delay relationships of the individual path segments that define these paths **independent of which of these three attack scenarios is used**.

Under either of the latter two attack scenarios, the input to the attack is the 4096 paths and estimated delays derived from simulations or the delay model, and the set of module parameters. The ML algorithm must be capable of predicting the response bitstring under any combination of challenges and module parameters. The following sections of the paper investigate ML-based attacks and determine the prediction error rates under relaxed conditions which fix the challenges to a small subset of those available and allow the modules parameters to be systematically varied. Although the actual problem is much larger and harder in practice, our analysis does serve to provide evidence of the underlying strength of the HELP algorithm against ML attacks. Moreover, the success of a full-blown model-building attack is predicated on solving this simpler problem.

3.5 Delay Model Complexity of HELP PUF

The complexity of the delay model of HELP is related to the complexity of the Boolean expressions that represent path sensitization conditions. The number of terms and the size of each term in the Boolean equation are determined by the number of gates (stages) that define the path, the number of inputs to each of the gates along the path and the gates' logic functions.

A delay model constructed in this fashion presents several challenges. First, the number of expressions produced at the primary outputs (POs) is proportional to the number of paths in the circuit, which can be exponential to the number of inputs. Second, the size of the expressions grows exponentially as the logic depth (number of stages) of the path increases. Boolean expression optimization techniques can be used to reduce the size, but this will increase the level of effort expended to construct the model. Third, in cases where paths fanout and reconverge downstream, deciding which of the multiple reconverging path segments dominates the timing, i.e., determines the output delay characteristics of the path, requires accurate delay models. Even when such models are available, within-die variations within each chip may change which path segment actually dominates the timing, making it necessary to include 'possibly sensitized' path expressions in the delay model, which will greatly expand the number of expressions that need to be evaluated.

4 Machine Learning (ML)

In this section, we investigate the effectiveness of ML techniques as an attack strategy for predicting HELP response bitstrings. From the discussion in Section 3.4, the traditional attack which uses CRPs directly is more difficult than an attack focused only on the Delay Processing Phase of the HELP algorithm. The traditional attack, referred to as a CRP-based attack, requires the ML algorithm to learn which paths are sensitized by the challenges, which, by itself, is a difficult problem. Therefore, our tactic is to assume the adversary has obtained the netlist of the functional unit used within HELP, and can run simulations (or use the delay model described in the previous section) to determine which paths are tested by the challenges. If it is possible to use ML to predict responses by attacking only the Delay Processing Phase, then a full-blown CRP-based attack might be attempted. However, as we show below, even this simplified attack scenario is difficult to carry out so it follows that attacking the entire algorithm is likely to yield poor results.

Bear in mind that even though the adversary can determine which paths are tested for a given set of challenges, the behavior of the Delay Processing Phase is also effected by the module parameters. As indicated earlier, the HELP algorithm uses a nonce exchange scheme to prevent the adversary from having direct control over these parameters [4]. In particular, the authentication protocol requires the server to

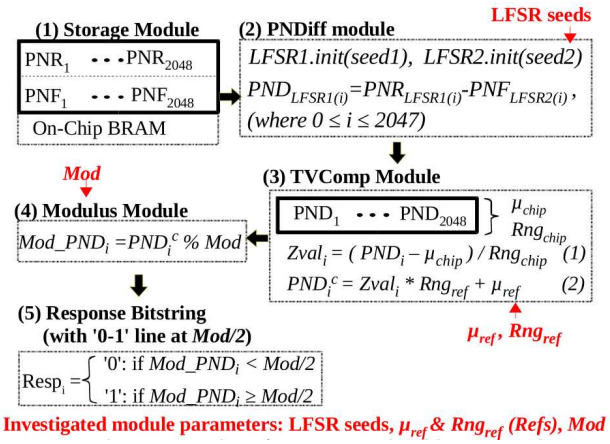

Fig. 3. Delay processing phase and module parameters that are investigated in machining learning experiments.

send a nonce to the token that is XOR'ed with a nonce generated by the token, and then this XOR'ed nonce is used to specify the module parameters. Therefore, even in the learning phase, the adversary cannot apply a systematic attack which varies the module parameters across all possible values, and instead, she must apply CRPs until the targeted set of module parameters is eventually generated and used by the HELP algorithm. This can add considerable time to the learning phase. In this paper, we attack the Delay Processing Phase assuming direct control of the module parameters is possible, again as a demonstration that solving this simpler problem is prerequisite to solving the problem constrained by the actual usage scenario.

The attack investigated in this section further simplifies the attack to using only one fixed set of 4096 delays. In other words, we do not vary the applied challenge set. In practice, the adversary would need to repeat the learning experiments described in this section to include a much larger set of challenges. In fact, deciding which of the exponentially large set of challenges are optimal for ML is by itself a difficult problem. Assuming the netlist is available, she can use compute-intensive automatic test pattern generation (ATPG) to determine challenges that test most of the paths within the functional unit (without the netlist, it would be impossible for the adversary to select challenges in a strategic fashion to obtain high path coverage). Using the functional unit currently instantiated within HELP as an example, we determined that more than 8 million rising and falling structural paths exist, and that approximately 80% of them are testable. In summary, the goal of our investigation is to determine for a fixed set of challenges (and tested paths) whether ML can be used to predict the response bitstring by allowing direct control over the module parameters.

4.1 Details of the Delay Processing Phase

The delay processing phase of the HELP PUF applies a series of mathematical operations to the set of 4096 path delays to produce the response bitstring. As illustrated in Fig. 3, the conversion process consists of a series of operations including (2) PNDiff, (3) TVComp, (4) Modulus and (5) bitstring generation.

The first step of the conversion process creates delay differences by subtracting the 2048 falling delays from the 2048 rising delays. The pairings created by the PNDiff module are selected pseudo-randomly using LFSRs. The digitized rising and falling delays are called PUF numbers (PN) and are designated as PNR for rising and PNF for falling, while the differences are referred to as PND. The two 11-bit seeds used to initialize the LFSRs are module parameters. The primitives used in the LFSRs ensure that all 2048 PNR and PNF are selected exactly once to produce a unique set of PND indexed from [0, 2047]. The PND are trivially created using the following expression: $PND_{LFSR1(i)} = PNR_{LFSR1(i)} - PNF_{LFSR2(i)}$, where 0<=i<=2047. The total number of unique PND that can be constructed from the PNR and PNF (assuming the tested paths are unique) is 2048*2048 = 4,194,304 (2^{22}).

The 2048 PND are used as input to the temperature-voltage-compensation (TVComp) module as shown in Fig. 3. TVComp first calculates the mean and range of the PND distribution by creating and parsing a histogram built from the PND. The computed mean (μ_{chip}) and range (Rng_{chip}) are used to standardize the set of PND as given by Eq. (1) in Fig. 3. The standardized values $Zval_i$ are then used as input to a second linear transformation given by Eq. (2). The *reference* mean and range, labeled u_{ref} and Rng_{ref}, used in the second transformation are also module parameters. The set of compensated PND are called PND^c. The primary goal of TVComp is to reduce undesirable variations in the measured path delays introduced by varying environmental conditions (TV noise).

The PND^c are then used as input to the Modulus module, which applies a modulus (*Mod*) to produce a set of 2048 Mod_PND. The modulus is a module parameter that transforms the PND to values in the range [0, *Mod*]. A response bit is then assigned to each of the Mod_PND by comparing the Mod_PND to a threshold defined as *Mod*/2. The response bit is '0' if the Mod_PND is less than the threshold and '1' otherwise. The values 0 and *Mod*, and the threshold, are referred to as '0-1' lines.

A margining scheme is also proposed within the HELP algorithm (not shown) that eliminates response bits that are deemed 'unstable', i.e., bits generated from Mod_PND which are within a 'margin' of the '0-1' lines. In our ML experiments, we ignore the margining process and instead use all of the response bits. We justify this simplifying assumption, as we did earlier for others, that incorporating the margining scheme will increase the difficulty of an ML attack.

A set of ML experiments are constructed to evaluate the model-building resistance of the PNDiff, TVComp and Modulus modules which use the PND, PND^c and Mod_PND as input in each experiment. As a further simplification, we reduce the number of fixed delay values from 4096 to 512 (256 PNR & 256 PNF) to deal with the huge data set sizes created by varying the module parameters over the range of values possible. The LFSRs and *seeds* are reduced accordingly to 8-bits. Under these conditions, the total number of unique PND that can be constructed by varying the LFSR *seeds* is 256*256 = 65,536 (2^{16}). The ML experiments report prediction errors using this subset of data.

4.2 Experimental Setup

A set of experiments are devised that construct data sets in which the module parameters are added one-at-a-time to determine the contribution of each to model-building resistance. Specifically, four experiments, identified as EXP1 through EXP4, are constructed in which the LFSR *seeds*, the u_{ref} and Rng_{ref} (referred to as *Refs* subsequently) and *Mod* parameters are added one-at-a-time into the input space of the ML experiments.

Table 1 identifies the goals in each experiment. The column labeled 'Varied Module Params' identifies the module parameters under investigation. In EXP1 through EXP3, two sub-cases are considered in the column labeled 'Status of Other Module Params' using module parameters that are NOT varied. For example, EXP2 identifies two sub-cases "No Mod" and "Fixed Mod" where the former indicates that no modulus operation is applied and the latter indicates that a modulus operation is applied but the *Mod* is fixed to a specific value. Note that the LFSR seeds are varied in all four experiments.

Table 1: Experimental Setup Summary

EXP	Varied Module Params	Status of Other Module Params	CRP space
EXP1	Seeds	a) No Refs, No Mod (Base case)	$65{,}536 = 2^{16}$
		b) Fixed Refs, No Mod	$65{,}536 = 2^{16}$
		c) Fixed Refs, Fixed Mod	$65{,}536 = 2^{16}$
EXP2	Seeds, Refs	No Mod vs. Fixed Mod	$65{,}536*16 = 2^{20}$
EXP3	Seeds, Mod	No Refs vs. Fixed Refs	$65{,}536*4 = 2^{18}$
EXP4	Seeds, Refs, Mods	None	$65{,}536*64 = 2^{22}$

A 'base case' is investigated in EXP1(a) where only the LFSR seeds are varied, identified in the 'Varied Module Params' column as *Seeds*. Here, the TVComp and Modulus operations are not applied to the PND. We use the results from EXP1(a) as a reference in which the prediction errors of the other experiments are compared. Here, all combinations of PNR and PNF are used to generate 256*256 = 65,536 or 2^{16} unique PND (this defines the CRP space for EXP1). Note that in order to generate bits from the PND without applying a Modulus operation, an alternative technique is needed to define the 0-1 line. Here, we use the median value of all 2^{16} PND as the 0-1 line. EXP1(b) and (c) add the TVComp and Modulus operations one-at-a-time using fixed values for the parameters.

EXP2 and EXP3 add *Refs* and *Mod* to the parameters that are varied, respectively. We use four fixed values for each of the u_{ref} and Rng_{ref} (*Refs*) and *Mod* parameters and therefore the CRP space of EXP2 and EXP3 is expanded by (4*4) = 16 and 4, respectively. The fixed values used for the *Refs* parameters are derived from characterization experiments using 45 FPGAs. The four *Mod* values used are 12, 14, 16 and 18. EXP4 adds both the *Refs* and *Mods* parameters and defines the most complex experiment with the CRP space expanded to 65,536*64 = 4,194,304.

4.3 Machine Learning Algorithms and Data Set Construction Process

The ML algorithms investigated include Logistic Regression (LR), Support Vector Machine (SVM) [14] and AdaBoost (AB) [15] using the *sklearn* platform [16]. We investigated a wide range of different solver, kernel and parameter combinations for LR and SVM but only the results from those combinations the produced the lowest prediction errors are reported. A similar approach is taken for AdaBoost using the 'depth' and 'N_classifiers' parameters. AdaBoost produced the lowest prediction errors for the base case EXP1(a) and therefore is the only ML investigated in the remaining experiments.

The challenge-response (CR) data sets are constructed in the following fashion. An exhaustive set of PND are created by pairing all PNR with all PNF. The binary representation of the index into the array of PND is used as input to the ML algorithm. The binary index concisely encodes the tested path pairing sequences and is able to represent any of the pairings created by the PND module. In our experiments, the index is 16 bits to represent the set of 256*256 = 2^{16} PND. The set of 2^{16} PND models the diversity introduced by varying the LFSR seeds (identified in Table 1 as 'Seeds'). The ML algorithms for EXP1 use only these 16 bits as input. A similar encoding scheme is used to enable inclusion of the *Refs* and *Mod* parameters for EXP2 through EXP4. The length of the binary index, and corresponding input to the ML algorithm, adds four bits to represent the 16 *Refs* combinations and two bits to represent the 4 *Mod* combinations. The output of the ML is configured to predict one bit at a time of the 256-bit response.

In each experiment, training sets of various sizes are selected randomly from the overall input-output CRP space. A total of five different **training data** sets are constructed for each training set size. The data evaluated for prediction errors, referred to as the **test data**, is randomly selected from the remaining CRP space and is always fixed in size to 1000 CRPs. The prediction error is calculated as the average value from evaluations carried out on the five data sets.

4.4 ML Experimental Results and Analysis

As indicated above, EXP1(a) only applies the PNDiffs operation and is used as the base case experiment. EXP1(a) is used to evaluate the effectiveness of the three ML algorithms LR, SVM and AdaBoost. The prediction errors are shown in Fig. 4(a) which plots curves for a set of training set sizes shown along the x-axis. The LR algorithm has the poorest performance with prediction errors remaining at approximately 40%. AdaBoost, on the other hand, produces the best result with prediction error falling below 5% for training set sizes > 2^{15}. Given these results, we use AdaBoost for the remaining experiments.

In contrast to EXP1(a) which uses the PND directly, EXP1(b) applies TVComp to the PND but fixes μ_{ref} & Rng_{ref} (*Refs*) to specific values. Similarly, EXP1(c) applies both the TVComp and Modulus operations but fixes the input parameters to both routines. Fig. 4(b) shows that the prediction errors are unchanged for EXP1(b) over the base case, which indicates that TVComp by itself does not improve ML resistance. However, ML resistance is significantly increased by

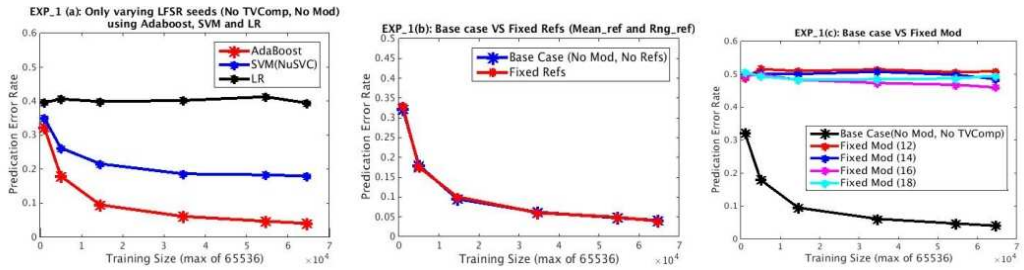

Fig. 4. Prediction error of EXP1 where only LFSR seeds vary: (a) base case using different ML algorithms including Adaboost, SVM and LR, (b) when fixed *Refs* are used and (c) when fixed *Refs* and fixed Mod are used.

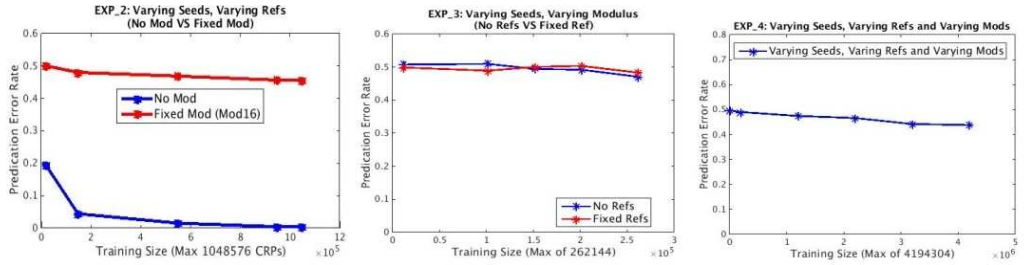

Fig. 5. Fig. 5. Prediction error when LFSR seeds and (a) *Refs* (EXP2), (b) *Mod* (EXP3) and (c) *Refs* and *Mod* (EXP4) are varied.

the Modulus operation as shown in Fig. 4(c) even under these simplifying conditions where the same *Mod* is used to construct the entire CRP space.

EXP2 varies both the *Seeds* and *Refs* parameters and investigates two sub-cases, namely 'No Mod' and 'Fixed Mod' as a means of providing further support that the Modulus operation is responsible for improving ML resistance. Fig. 5(a) plots the prediction errors, again with training set size along the x-axis but now for a larger 2^{20} CRP space. Once again, prediction errors remain large for the 'Fixed Mod' curve and drop to nearly 0 without the Modulus. EXP3 varies both the *Seeds* and *Mod* parameters and uses the *Refs* parameters in the two subcases 'No Refs' and 'Fixed Refs'. As expected, the prediction errors remain large for both subcases because the Modulus operation is included. EXP4 investigates the case which is closest to an actual usage scenario where the *Seeds*, *Refs* and *Mod* parameters all vary. The CRP space expands to 2^{22} as shown by the x-axis in Fig. 5(c). Here, prediction error remains above 40% even when up to 99% of the CRP space is used for training.

Note that the data set construction process which creates all combinations of PNR and PNF reuses each of the PNR and PNF in 256 different PND. Reuse of the PNR and PNF in this fashion reduces entropy because of correlations that exist in the PND. ML algorithms are able to learn these correlations and reduce prediction errors. However, the Modulus operation 'wraps' the PND into a much smaller range between 0 and *Mod* and effectively 'breaks' the classical correlations that are present in the PND. These experiments show that the reduction in these classical correlations make it difficult for ML algorithms to build a model that is capable of accurately predicting the responses.

5 Conclusion

The delay model for delay-based PUFs gives a mathematical representation of the sensitized path delays under a given challenge. In this paper, we investigated several machine learning (ML) algorithms and a wide range of strategies of applying model-building attacks to the HELP PUF. Our experimental results show that the Modulus operation carried out as a component of the HELP algorithm adds significantly to the ML resistance of HELP even when evaluated using data sets constructed under simplified usage scenarios.

6 References

[1] Gassend, B., Clarke, D., Van Dijk, M., Devadas, S., "Silicon physical random functions," *CCS* 2002, pp. 148-160.

[2] J. Aarestad, J. Plusquellic, D. Acharyya, "Error-Tolerant Bit Generation Techniques for Use with a Hardware-Embedded Path Delay PUF," *HOST*, 2013, pp. 151-158.

[3] W. Che, F. Saqib and J. Plusquellic, "PUF-Based Authentication", *ICCAD*, 2015.

[4] W. Che, M. Martin, G. Pocklassery, V. K. Kajuluri, F. Saqib and J. Plusquellic, "A Privacy-Preserving, Mutual PUF-Based Authentication Protocol", *Cryptography*, 2017.

[5] U. Rührmair et al., "Modeling attacks on physical unclonable functions," *CCS*, 2010, pp. 237-249.

[6] U. Rührmair, J. Solter, 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", *TIFS*, vol. 8, no. 11, 2013, pp. 1876-1891.

[7] M. Majzoobi, F. Koushanfar, and M. Potkonjak, "Lightweight Secure PUFs," *ICCAD* 2008, pp. 670-673.

[8] G. Hospodar, R. Maes, and I. Verbauwhede, "Machine Learning Attacks on 65nm Arbiter PUFs: Accurate Modeling poses Strict Bounds on Usability", *WIFS*, 2012, pp. 37-42.

[9] G. T. Becker, "On the Pitfalls of using Arbiter-PUFs as Building Blocks", *Trans. Comput.-Aided Design Integr. Circuits Syst.*, 2015, pp. 1295-1307.

[10] F. Ganji, S. Tajik, and J.P. Seifert, "Why Attackers Win: On the Learnability of XOR Arbiter PUFs", *Trust and Trustworthy Computing*, Springer, 2015.

[11] U. Chatterjee, R. S. Chakraborty, H. Kapoor, and D. Mukhopadhyay, "Theory and Application of Delay Constraints in Arbiter PUF", *Trans. on Embed. Comp. Syst*., 2016.

[12] D. P. Sahoo, P. H. Nguyen, R. S. Chakraborty, and D. Mukhopadhyay, "Architectural Bias: a Novel Statistical Metric to Evaluate Arbiter PUF Variants", *IACR Cryptology ePrint Archive Report*, 2016.

[13] K.Tiri and I. Verbauwhede, "A Logic Level Design Methodology for a Secure DPA Resistant ASIC or FPGA Implementation", *DATE*, 2004.

[14] http://scikit-learn.org/stable/modules/svm.html

[15] Y. Freund, R. E. Schapire, "A Decision-Theoretic Generalization of On-line Learning and an Application to Boosting", *J. Comput. Syst. Sci*., 1997. pp. 119-139.

[16] http://scikit-learn.org/stable/