Preprint: International Workshop on Combinatorial Testing, 2022

Combination Frequency Differencing for Identifying Design Weaknesses in Physical Unclonable Functions

D. Richard Kuhn
National Institute of Standards & Technology
Gaithersburg, MD, USA
kuhn@nist.gov

M S Raunak
National Institute of Standards & Technology
Gaithersburg, MD, USA
raunak@nist.gov

Charles Prado
National Institute of Metrology, Quality and Technology
Rio de Janeiro, Brazil
cbprado@inmetro.gov.br

Vinay C. Patil
University of Massachusetts
Amherst, MA, USA
vcpatil@umass.edu

Raghu N. Kacker
National Institute of Standards & Technology
Gaithersburg, MD, USA
raghu.kacker@nist.gov

***Abstract*—Combinatorial coverage measures have been defined and applied to a wide range of problems. These methods have been developed using measures that depend on the inclusion or absence of t-tuples of values in inputs and test cases. We extend these coverage measures to include the frequency of occurrence of combinations, in an approach that we refer to as combination frequency differencing (CFD). This method is particularly suited to artificial intelligence and machine learning (AI/ML) applications, where training data sets used in learning systems are dependent on the prevalence of various attributes of elements of class and non-class sets. We illustrate the use of this method by applying it to analyzing the susceptibility of physical unclonable functions (PUFs) to machine learning attacks. Preliminary results suggest that the method may be useful for identifying bit combinations that have a disproportionately strong influence on PUF response bit values.**

***Index Terms*—combinatorial frequency, combinatorial testing, Physical(ly) Unclonable Functions, PUF**

I. Introduction

Methods and tools for measuring combinatorial coverage were initially developed for analyzing the degree to which test sets included t-way combinations of test values (for some specified level of t) [1]–[4]. Combinatorial coverage measures have been defined and applied to a wide range of problems, specifically for fault localization, and for evaluating the adequacy of test inputs and input space models. These measures can also be applied in artificial intelligence and machine learning (AI/ML) systems, including recent applications to explainability in machine learning (ML) [5], [6], improving ML results [7], analyzing aspects of transfer learning [8], and improving robustness of ML models [9].

For many aspects of AI/ML assurance, combinatorial coverage measures are valuable, and possibly essential, as the behavior of AI systems is primarily determined through training inputs, instead of the model code. Conventional structural code coverage measures are not applicable to such black-box behavior [10]. Consequently, it is essential to evaluate the degree to which possible combinations of input attribute values have been included in training and test sets for AI and autonomy. We note that attributes in a machine learning setting correspond to parameters in a test design for regular software; both are the inputs to the system. If the system has not been shown to function correctly for an input combination that may be encountered in use, then assurance is inadequate. However, for machine learning, we are often interested in the frequency (or rate) of occurrence of t-tuples of values in input, and how two different sets may compare or differ in combinatorial coverage [11]. In general, machine learning for classification problems attempts to identify differences among two or more classes of objects based on their attributes. The frequency of occurrence of these attributes, or combinations of attributes, however, may be significantly different among classes, and current assurance practices do not consider these differences.

In this paper, we apply combinatorial coverage measures [11], [12] that include the frequency of occurrence of combinations, in an approach referred to as combination frequency differencing (CFD). CFD may be contrasted with previous combination coverage metrics, referenced above, that are based on the presence or absence of value combinations. CFD is particularly suited to AI/ML applications, where accuracy may be dependent on the prevalence of various attributes of elements of a specified class and non-class sets. We illustrate this method by applying it to analyzing physical unclonable functions (PUFs) for potential weaknesses in design that may make the PUFs susceptible to machine learning attacks.

II. Combinatorial Coverage for Classification

Many applications related to software engineering require separating elements into two classes. For example, distinguishing or differentiating the set of input values that cause a software to fail vs. the set of values that result in proper execution without failure. We will use the terms *Class* and *Non-class* to identify elements or objects that can be distinguished based on some attributes or properties such as the set of failed test cases (class) vs. the set of passed test cases (non-class) in a test suite. Furthering this idea, one can distinguish two groups by analyzing the presence of certain t-way (*t = 2, 3, 4, etc.*) input or attribute combinations in the *Class* set and absence of those combinations in *Non-class* set. Analyzing the set differences for t-way coverage has been successfully used to identify causes of failures in a software [13]. The method utilizes set differencing to identify input combinations that occur in the class set, and do not occur, or are rare, in the non-class set. If this difference is computed on t-tuples of values in failed tests vs. passed tests, then the difference contains t-tuples of values that have triggered the failure (in a deterministic system).

This idea of distinguishing two sets based on the coverage of some t-way input or attribute combinations is applicable in the context of machine learning applications as well. The goal of many machine learning models is to learn to distinguish members of one class from another using attributes that identify them, such as dogs distinguished from cats using attributes such as size, ear shape, hair texture, etc. In machine learning, the difference represents properties or attributes that occur in the class (e.g., a particular animal species) that do not occur, or are rare, in the non-class examples (other species). This is a generalized version of the fault localization problem in software testing, where the class whose distinguishing features are to be identified is the set of failing tests, and the features to be found are the combinations that lead to a test resulting in a failure. Similarly, the combinatorial set differencing idea has been used for identifying the potential of a machine learning model trained in one environment to be successful in a different environment [8]. This method has also been used in explaining the classification decision of machine learning models [6].

A. Frequency Differences of t-way Tuples

The combinatorial coverage measures as applied in fault localization, transfer learning, and model explainability are based on the presence or absence of t-tuples of values in input files for testing or training machine learning algorithms. The concept of 'combination coverage' comes into play here. A combination of values or attributes is counted as covered if it occurs once or multiple times in the input file. For applications such as software testing related coverage criteria it is generally important to determine if a t-tuple of values has been included in the test suite, but the number of times it occurs is less important. Here multiple occurrences of a combination mean some duplication of effort, but do not affect the requirement for ensuring that all t-way combinations have been covered [13]. In transfer learning evaluation, the same type of requirement holds – we want to ensure that states and environments, as represented by t-tuples of values of the input model, are handled correctly [8]. If it can be shown that the ML model produces the right prediction or classification for a t-tuple of values, multiple occurrences of the combination are not needed. We should note, we are not considering the effect of input sequences here.

In this paper, instead of focusing only on the coverage of t-way values in class and non-class set, we introduce a new metric: the number or frequency of occurrence of t-way t-tuples of values. The idea here is to determine the degree to which an attribute is associated with a particular class. If a combination of attribute values is seen in a high proportion of class members, but not in non-class members, then it may be a reasonable indicator for differentiating instances or at least for narrowing the range of possibilities for class identification. For example, many dog breeds may have a long tail, and many may have a curled tail, but a much smaller number of breeds have both attributes. Considering the frequency of both long and curled tail could be a useful differentiating factor. Thus it is important to have a measure that considers the frequency of instances of t-tuples of values in class and non-class instances.

Let us identify sets of class and non-class t-tuples as C_t and N_t respectively. We will abbreviate C_t and N_t as C and N, where interaction level t is clear or is not needed for discussion. In the following discussion, we refer to a t-way combination c_t (notice the small c) as a differentiating combination for the class C if it is present in a class instance of class set C, and absent in non-class instances N, or if it is more common in C than N as determined by a threshold value. The key point here is to identify combinations of attribute values that are strongly associated with one class but not with others, based on the *frequency* or *rate of occurrence* in one class as compared with others.

B. Measures of Combination Frequency Difference

Below we give a definition of combinatorial frequency difference metric, CFD, followed by a discussion of how it may be used to distinguish class instances [11].

The frequency (or rate) of occurrence refers to the number of times a t-tuple of values is present per number of rows in the file or array [11]. Here each row represents an instance of an object. Thus the combination frequency difference, for a t-tuple of values x in two arrays of instances of two different classes can be defined as the difference between the fraction of occurrences in one array vs. the second:

$$CFD = F_{Cx} - F_{Nx}$$

Here,

- $F_{Cx} = M_{Cx}/R_C$ = fraction of occurrences of a t-tuple of values in C
- $F_{Nx} = M_{Nx}/R_N$ = fraction of occurrences of a t-tuple of values in N
- M_{Cx} = number of occurrence of a particular t-tuple of values x in C
- M_{Nx} = number of occurrence of a particular t-tuple of values x in N
- R_C = number of rows in the class instance file
- R_N = number of rows in the non-class instance file

For the following discussion, let us also define:

- r = number of rows of input file
- k = number of columns or attributes, excluding class or response variable
- v = number of values for attributes
- t = the interaction level such as 2, 3, 4 ... etc.
- v^t = number of t-way combinations of v values

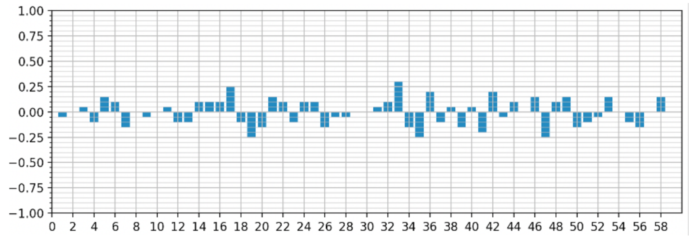
Fig. 1. Example of frequency difference for two classes of 6 binary variables

The frequency difference values can be graphed, where the height on the Y axis shows the difference $F_{Cx} - F_{Nx}$ for every t-tuple of values x. The X axis is indexed by $v^t\binom{n}{t}$ points for t-way combinations, as there are v^t possible values or settings of the t attributes or variables in the combination, for each of the $\binom{n}{t}$ t-way combinations. X axis indices reflect the lexicographic ordering of these points. So for example, 2-way t-tuples of values are displayed in Fig. 1 in the order given by: i, j for i in $0 \leq i < k-1$ for j in $i+1 \leq j < k$.

Let us consider the scenario depicted in Fig. 1 as an example. Suppose we have a data set with six binary attributes (variables), i.e., N=6 numbered as 0, 1, 2, 3, 4, 5. For t=2, the 2-way combinations will be indexed as $(0, 1, 00), (0, 1, 01), \ldots, (4, 5, 11)$, where the first two elements of each tuple represents the attribute number and the third element represents their binary value combination. There will be a total of $2^2\binom{6}{2}$, = 60 X-axis points, numbered $0, ..., 59$. For each of these, the Y-axis shows the difference in frequency of occurrence between C and N, normalized for the size of sets C and N. For example, if the value 01 for attribute combination $i = 1, j = 4$ occurs 60 times in a C file of 100 rows, and occurs 40 times in an N file of 120 rows, then the Y axis value for $i, j = 1, 4$ for value 01 is (60/100)–(40/120) = 0.3 (bit 33 in Fig. 1).

In other words, the graph of Fig. 1 displays for each 2-way combination the fraction of occurrences of each set of v^2 t-tuples of values on the X axis in the order described above. For each of these 2-way combinations x, $F_{Cx} - F_{Nx}$ for four t-tuples of values are displayed for the four possible value settings 00, 01, 10, 11. Thus the difference in coverage for C and N for $i = 1, j = 4$ will be found on the horizontal axis at $x = 32, ..., 35$.

A threshold value T can be used to determine if a t-tuple of values c_t is common in set C_t and rare in set N_t, based on the CFD measures. We say a combination x_t is $differentiating$ for a class $C \iff F_{Cx} > T \times F_{Nx}$. Specifically, the threshold value T identifies t-tuples x for which we can say "x is T times more common in C than it is in N", an intuitive way to identify t-tuples of values that are associated closely with the class C. Note that the phrase "T times more common" suggests that T will normally be 1 or greater.

Note that we have used $t = 2$ in this paper, but other values of t may be appropriate in other cases. A value of $t = 1$ could detect some differences but would miss interactions. A larger value of $t \geq 3$ can also be used in the tools we have developed, but results are not shown in this paper due to space limitations. Please see [11] for examples.

III. Physical Unclonable Functions

We have applied our idea of Combinatorial Frequency Differencing or CFD to identify potential weakness in physical unclonable functions or PUFs [14]–[17]. A PUF may be regarded as a physical implementation of a black box function that produces a response r for a given challenge string of bits c, that is, r=$f(c)$. The unit (single bit) response is binary, i.e., can be represented as 0 or 1. A series of PUFs can be put together to produce a larger response sequence. As the name suggests, PUFs are designed using physical hardware devices. These functions utilize unique properties of the physical elements within the hardware such as the small variation in propagation delays between identical circuit gates or small threshold mismatches in transistor feedback loop due to manufacturing process variation. These physical characteristics are very difficult to reproduce in the hardware, which is what makes them physically unclonable. Using such physical characteristics PUFs can be utilized to authenticate hardware devices, and thus combat insecure storage, hardware counterfeiting, and other security problems [14], [15], [18]. (Following most of the literature, we do not distinguish between the terms "physical unclonable" and "physically unclonable", although some authors have proposed different definitions for these terms.)

An ideal PUF should be stable over time, unique in its existence, easy to evaluate, and very difficult or impossible to predict. Thus, it should not be possible to generate a function that has the same behavior or produces the same output as the PUF for challenge inputs. In this sense, the PUF function is "unclonable". It should also be infeasible to determine components of the physical hardware or input values that influence the PUF output, such that a 0 or 1 value in some positions of the input string makes a 0 or 1 output more likely for the output r.

A. Machine Learning Attacks on PUFs

The primary use of PUFs is related to authentication. In a simple use case, during manufacturing, the physical system is subjected to one or more challenges, and the responses to these challenges are recorded and stored in a secure location. When the PUF is deployed in the field, a set of the recorded challenges are transmitted and if the expected response is received, then the device is authenticated.

Depending on the strength of their implementation and consequent scalability, PUFs are categorized into two levels – weak and strong. Weak PUFs have a limited number of challenge-response pairs (CRPs) that can be generated from a

TABLE I
ML PREDICTION RESULTS FOR FOUR PUF DESIGNS

	Ada Boost	Bayes Net	Decision Table	J48/ C45	JRip	Logistic	Naive Bayes	Random Forest	Stoch. Grad. Descent	ZeroR (baseline)	Average Accuracy	combined diff 2-way
DB1	77.1	96.2	75.6	72.1	77.2	99.7	96.2	87.2	99.3	55.0	86.7	0.489
DB2	54.8	54.9	76.7	68.1	75.2	54.9	54.9	71.9	52.4	55.6	62.6	0.309
DB3	50.7	50.1	71.0	63.9	67.2	50.3	50.1	62.6	50.2	50.1	57.3	0.248
DB4	57.5	56.5	58.8	54.6	60.7	56.4	56.5	55.3	54.6	50.6	56.8	0.216

single device, while strong PUFs can generate a much larger set of CRPs. One of the key requirements for a strong PUF design is that it should not be possible to infer information about the internal structure by observing inputs and outputs [17]. Many authors have shown that machine learning (ML) models can be constructed to predict the output of PUFs for a given input string, i.e., "breaking" the PUF by defeating its authentication function [18]–[21]. This vulnerability of failing in the face of a machine learning attack can vary significantly based on the PUF design. One of the challenges in developing PUFs is thus to identify potential weaknesses before constructing the PUF.

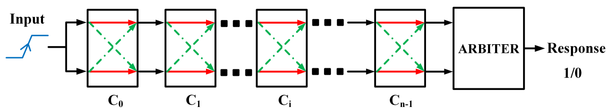

Fig. 2. A simple arbiter PUF

Table I shows the accuracy of ML prediction results for the four PUF designs discussed in this paper, for 10 ML algorithms available through the Weka machine learning tool package [22]. Note that "Zero R" is a baseline, where predictions are simply the proportion of 0 or 1 results for the challenge/response pairs in the training set. The other algorithms were selected to provide a representative sample of popular ML algorithms of different types. *AdaBoost* is an adaptive ensemble algorithm that uses a phased sequence of basic decision tree algorithms, improving on prediction results with each phase. *Bayes Net* and *Naïve Bayes* are based on Bayesian statistical concepts. *Decision Table* is a majority classifier based on a nearest-neighbor algorithm. *J48* and *Random Forest* are based on decision trees (note, J48 is a variant of the widely used C4.5 algorithm). *Stochastic gradient descent* minimizes a loss function that is a weighted linear combination of the attributes, and *logistic regression* uses weighted attributes in a regression function. *JRip* is a propositional logic-based rule learning algorithm. Although there is a wide range of results for different algorithms, it is clear that DB1, the arbiter design, is much more vulnerable to ML attacks, where two algorithms are able to predict the response to challenges with near perfect accuracy. Even the best two PUF implementations (DB3 and DB4) are not fully resistant to revealing some bias in their response. Note that their averages are all considerably above the baseline ZeroR, which simply guesses in proportion to 0 or 1 responses in challenge-response pairs.

B. Analysis with Combination Frequency Differences

In this section, we show how combination frequency differences can be used to gather empirical information about the design and internal structure of a PUF. This is achieved by measuring the difference between the occurrence-frequency of t-way combinations associated with a 0 response as compared with a 1 response. Ideally, there should be little difference, except for random variances. As shown below, however, these differences vary considerably and align with the difference in predictability using machine learning, i.e., the effectiveness of ML attacks increases with the level of frequency differences. Although this is a small set of PUFs, and this work is only preliminary, we believe this information may be useful in identifying design deficiencies and make PUFs more resistant to ML attacks.

We measure the combinatorial frequency differences (CFD) for four sets of PUF data, namely DB1, DB2, DB3, and DB4. The results of the CFD measures are graphed in figures Fig.3 to Fig.7. Values labeled min, max, and *diff* are shown, along with variance, and a density value based on the sum of absolute differences over the number of value combinations (density is not used in this analysis but is included for completeness as it is being considered in future work). Note that differences are given as difference $F_{Cx} - F_{Nx}$, so negative values are cases where non-class t-tuples of values exceed class t-tuples of values. For example, in Fig. 1, $min = -0.25$, at index 47, and $max = 0.3$, at index 33, so *diff* $= max - min = 0.55$, or visually the sum of lengths of longest line above and longest line below the center line.

Comparing the accuracy of ML predictions in Table I with these graphs, it becomes immediately apparent that there is a relationship between the "noisiness" of the graphs and the success of ML algorithms in predicting the response of the PUF. In case of the arbiter PUF, DB1 (Fig. 3), the CFD produces a very noisy graph with differences for nearly every 2-way combination of bits ranging from about 0.10 to 0.25. For this PUF, ML algorithms predict the response with up to 99.7% accuracy. For the PUF most resistant to ML predictions, DB4 (Fig. 7), the graph shows small frequency differences, with nearly all under 0.05, up to a few around 0.10.

The other two data sets fall within the range between DB1 and DB4, for both frequency differences and prediction accuracy, which is a metric for the potential of breaking the PUF. See the last column of Table I, which shows that the difference (above and below the center line) for 2-way combination frequency shows the same ordering as the average

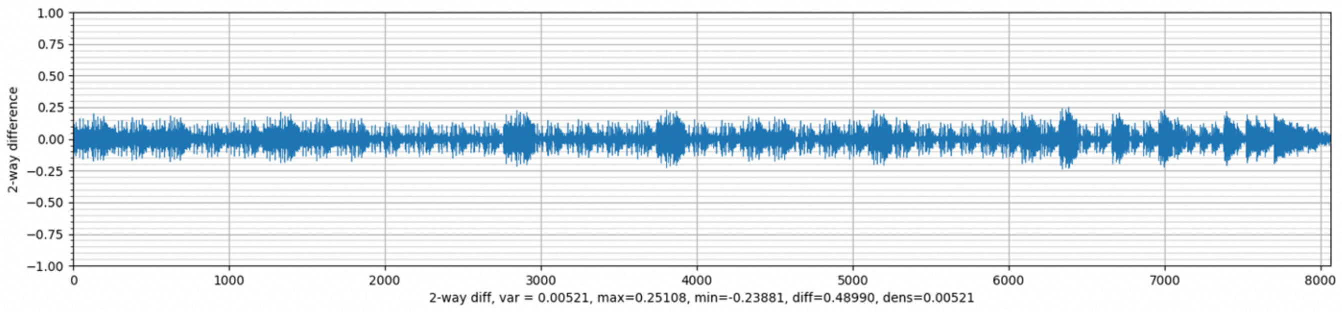

Fig. 3. 2-way frequency differences for a 64 bit arbiter PUF (DB1)

ML prediction accuracy and the highest prediction accuracy for DB1 through DB4.

C. *Analyzing different PUF types with CFD*

There are two major types of hardware implementation of PUFs: memory based and delay based. A typical memory based PUF is the SRAM PUF. Delay based PUFs include arbiter PUFs, pseudo-linear feedback shift register PUF, and ring oscillator (RO) PUF. In the following sections we analyze four different PUFs. The dataset related to these PUFs are available at [23].

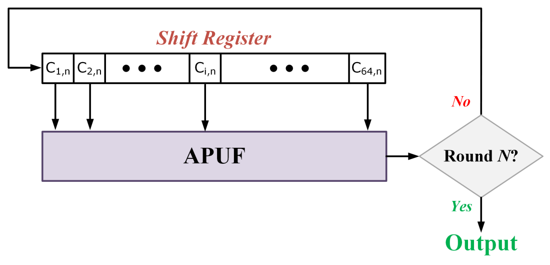

Fig. 4. N-rounds Shift Register Arbiter PUF (DB2)

1) Arbiter PUF (DB1): The main idea of an arbiter PUF is to create a digital race for signals through two paths within a chip, and to have an arbiter circuit that decides which signal has won the race [17] [16]. The two paths are designed identically, but controlled by the challenge inputs. However, the manufacturing process usually introduces a very slight longer delay in one of the paths as compared with the other. Given a particular challenge, the arbiter PUF therefore will produce an output dictated by the physical characteristics of that unique hardware implementation. During an arbiter PUF design, one has to make sure that the delays between the two paths are not too close to each other. Otherwise, the output will be dictated by noise in the signal rather than the delay uniquely introduced through the manufacturing variation.

As Fig. 2 shows, each gate or switch-block introduces a delay for one of the outputs, which accumulates over the blocks. This gives rise to the opportunity of building what is known typically as cloning attacks, also known as model building attacks or model learning attacks. The idea is that one can build a mathematical model of the PUF which, after observing several CRP queries, will be able to predict the response for a given challenge with a high level of accuracy. With the proliferation of machine learning algorithms, this type of model building, or model learning has become easier to implement. To make model building attacks more difficult on basic arbiter PUFs, non-linearity can be introduced into the delay lines of the designed circuit. For example, in case of feed-forward arbiter PUFs, some challenge bits are not set by the user. Rather, they are connected to the outputs of the intermediate arbiters evaluating the race at some intermediate point of the circuit. This technique, however, increases the noise in the output of the arbiter PUF. Although initial results with feed-forward arbiter PUFs were shown to be resistant to model-building or model-learning attacks, more sophisticated learning models have been able to break them [24].

In the following we discuss in more detail the alignment between CFD measures and predictability using machine learning, i.e., vulnerability to model-building attacks.

Fig. 3 shows 2-way frequency differences for a 64-bit PUF, DB1, an early arbiter design with delays placed randomly in the hardware. With 64 bits, there are $2^2\binom{64}{2} = 8064$ 2-way differences indexed. Differences range from a low of -0.23881 to a high of 0.25108, for a range of 0.48990. These values are noted as *min, max,* and *diff* respectively at the bottom of Fig. 3.

2) 8-rounds Shift Register Arbiter PUF (DB2): The next PUF we consider is a modified version the Arbiter PUF (DB1), as illustrated in Fig.4. A shift register is integrated to modify the challenge in each *round* by incorporating the response generated by PUF in the previous round. Since the attacker only observes the final response and cannot access the intermediate responses, this process increases the complexity for the machine learning attacks compared to DB1 approach. We detail the steps for Shift Register PUF implementation below:

1) Challenge is input to both the shift register and the PUF
2) For each challenge applied to the PUF:
   - Generate a response using the input challenge
   - Challenge is shifted 1-bit to the right.
   - First challenge position receives PUF response.
3) Repeat the above procedure N-times.
4) Choose the last response as the PUF output.

Let us now consider the utility of the metrics included in this study, with respect to PUF vulnerability to ML attacks.

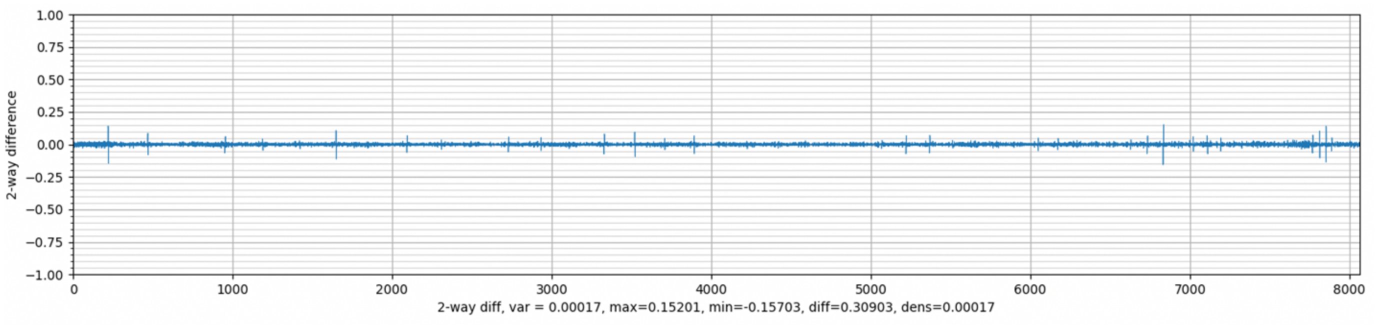

Fig. 5. 2-way frequency differences for DB2

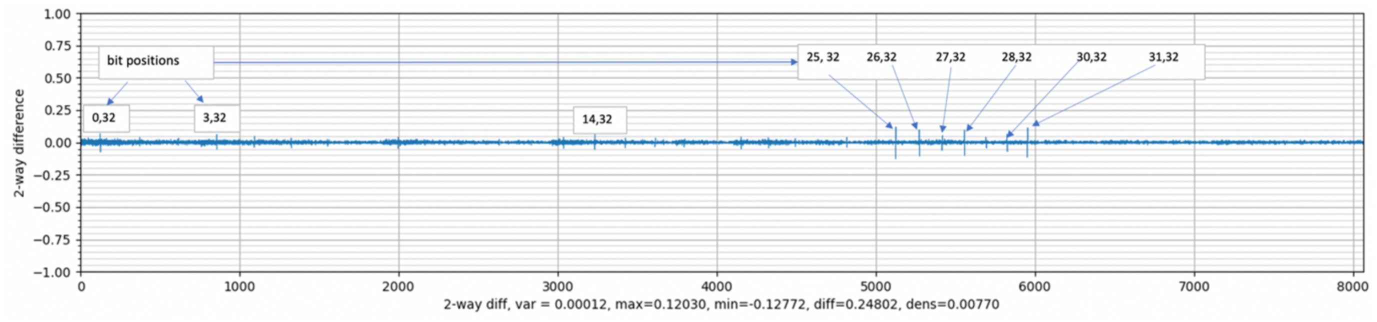

Fig. 6. 2-way frequency differences for DB3

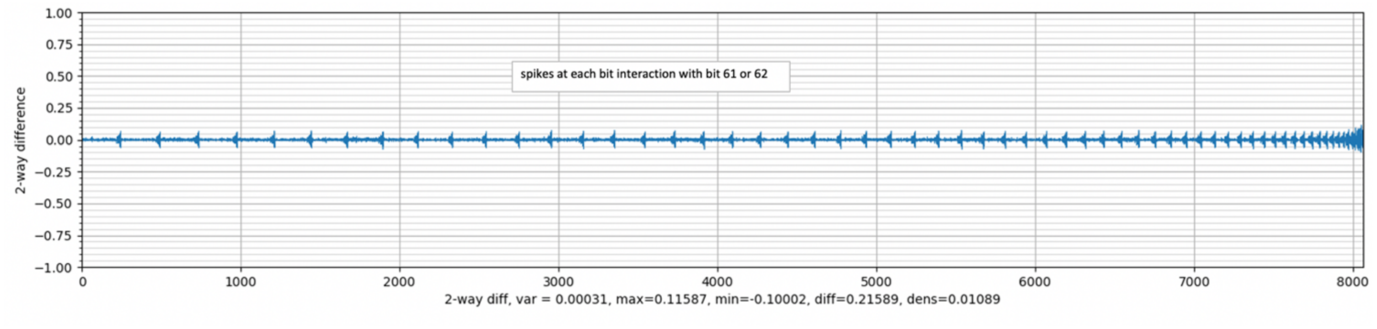

Fig. 7. 2-way frequency differences for DB4

TABLE II
VARIANCE AND STD. DEV. FOR 2-WAY AND 3-WAY COMB., DB1 TO DB4

PUF	Variance	Std. Dev.	Average Accuracy	Combined diff 2-way
DB1	.00521	.07218	.867	.489
DB2	.00017	.01304	.626	.309
DB3	.00012	.01095	.573	.248
DB4	.00031	.01761	.568	.216

Fig. 5 (2-way) shows the frequency differences for 8-rounds shift register PUF. Note that the variance is much smaller, 0.00017 as compared with 0.0521 for 2-way combinations of DB1 inputs, as shown in Table II.

There is much more uniformity in the response of DB2 to 2-way combinations of input bits, and as expected, this makes it much more difficult for ML to derive a model for the PUF that can successfully reproduce its response to inputs. As can be seen in Table II , the accuracy of prediction, which corresponds to vulnerability to ML attacks, matches the level of difference shown in the last column, although the number of spikes is clearly higher for DB4. (See also Sect. III-B.) Based on the analysis we present in this paper, it appears that *difference* is the most useful metric. We plan to pursue additional future work to further evaluate this conclusion.

Note however that Fig. 5 also shows a small number of spikes in the combination frequency chart. Combinations producing these spikes are shown in Table III, which shows 2-way bit combinations where the frequency difference exceeds a threshold value of T=3 std. dev. (σ). Note that combinations of almost all bits with bit 56 result in a spike that exceeds the 3σ threshold (others have spikes that are slightly below this value, but still clearly different from the other combinations). It should be noted that the appearance of spikes compresses towards the right end of the graph because combinations are indexed in a loop computation: i, j, b: for $i \in 0 \leq i < 63$ for $j = i + 1 \leq j < 64$ for $b \in \{00, 01, 10, 11\}$, for 2-way

TABLE III
2-WAY COMBINATIONS WITH GREATEST FREQUENCY DIFFERENCES, DB2

bits = values	bits = values	bits = values	bits = values
(0,56) = (1,0)	(11,56) = (0,0)	(26,56) = (1,1)	(41,56) = (0,1)
(0,56) = (0,1)	(12,56) = (1,0)	(26,56) = (0,0)	(42,56) = (1,1)
(1,56) = (1,0)	(12,56) = (0,1)	(31,56) = (1,1)	(42,56) = (0,0)
(1,56) = (0,1)	(14,56) = (1,1)	(31,56) = (0,0)	(51,56) = (1,1)
(3,56) = (1,1)	(14,56) = (0,0)	(32,56) = (1,1)	(51,56) = (0,0)
(3,56) = (0,0)	(15,56) = (1,0)	(37,56) = (1,1)	(52,56) = (1,0)
(4,56) = (1,0)	(15,56) = (0,1)	(37,56) = (0,0)	(52,56) = (0,1)
(6,56) = (1,0)	(16,56) = (0,1)	(38,56) = (1,1)	(53,56) = (1,1)
(6,56) = (0,1)	(17,56) = (1,0)	(38,56) = (0,0)	(53,56) = (0,0)
(8,56) = (1,1)	(17,56) = (0,1)	(40,56) = (1,0)	(54,56) = (1,1)
(8,56) = (0,0)	(25,56) = (1,1)	(40,56) = (0,1)	(54,56) = (0,0)
(11,56) = (1,1)	(25,56) = (0,0)	(41,56) = (1,0)	

combination indexes.

A potential explanation can be developed for the pattern of spikes in combinations that include bit 56 by noting that 8 is an even divisor of 56. PUFs accumulate differences as steps progress, so bit 56 occurs at the final stage before the last 8-rounds shift register. In a design situation, the next step would be to analyze the hardware components to determine why this irregularity was occurring.

3) 32-rounds Shift Register VTC-PUF (DB3): In the previous sections, CFD analysis was performed on Arbiter PUFs. As they are based on linear delay models, ML attacks have been successful in these implementations. To make PUF more resistant to ML attacks, some PUF´s implementations have been developed using non-linear functions like VTC-PUF (voltage transfer characteristics) [25].

This section shows the results of our analysis performed on a shift register VTC-PUF. As the name suggests, in a 32-rounds shift register VTC-PUF (DB3), Arbiter PUF is replaced by a VTC-PUF and the number of rounds is 4 times longer than the DB2 design. The increased number of rounds of shift register and the non-linearity of the VTC-PUF increases the complexity of the PUF and therefore makes the design less susceptible to model building attacks. The results of applying our analyses are shown in Fig. 6.

As can be seen in Fig. 6, the 32-bit position is present in CFD combinations with the greatest frequency difference. That is, a spike in the graph occurs for each bit paired with bit 32, of approximately equal size for bits 0 to 24. At bit 25, there is a significant increase in the size of spikes for bits paired with bit 32, indicating less variability in response associated with these bits. As the N-rounds of Shift Register is equal to 32, it is reasonable to infer that the variable N-rounds is directly related to spikes position. Therefore, without knowing anything about the internal construction of the PUF, the significance of bit 32 could be inferred. The same conclusion was obtained for DB2 implementation. We believe this type of pattern is related to the number of rounds, but additional study is needed to determine the nature of this relationship.

4) Uniform distribution PUF (DB4): Results are shown in Fig. 7 for a PUF with the most uniform distribution of entropy. This PUF has the greatest resistance to machine learning attacks, which are able to predict responses only somewhat better than chance (see Table I). In this case, the variations used in producing PUF response accumulate uniformly, with slight frequency differences for t-tuples of bits that include either bit 61 or 62. (Compression of the spikes towards the right side of the graph occurs because of the loop computation, as explained in section III-C2)

IV. Related Work

Security weaknesses in PUFs can be identified directly using ML attacks, as described in section III-A, and many approaches have shown success [19]. Arbiter PUFs, among the most popular designs, have been shown susceptible to support vector machine (SVM) methods [19], [26], [27], and deep learning models can also be used [28]. Genetic programming techniques have also shown success [29].

PUF strength has also been studied using entropy analysis [30], [31]. Entropy and related properties are measured on the bit strings generated by PUFs in responses. Poor entropy scores have been associated with weak PUF properties [32], [33] and increasing entropy of response strings improves PUF resistance to machine learning attacks [34]. Note that these entropy analysis approaches apply to response strings, and recall that the structure for PUF usage is $challenge \rightarrow response$. In contrast with entropy analysis methods, our approach measures properties of the challenge string associated with individual response bits. Measuring entropy of response strings has an intuitive association with strong PUF properties, since PUFs are intended to produce random, difficult to duplicate, strings - increasing randomness increases the difficulty of duplicating. Our approach, in contrast, reveals certain properties of the response generation function, rather than its response produced.

Combinatorial coverage has been used to identify areas of the test input space that have not been well covered, and thus not tested adequately [1]–[3]. Combinatorial coverage has also been applied in machine learning contexts, to identify areas of the machine learning model that are weak and determine improvements [7]. Another application of combinatorial coverage in machine learning is for analysis of transfer learning [8], evaluating the potential for a model trained in one environment, or set of inputs, to perform in a new or changed environment. A related task applies coverage analysis to improve the robustness of models [9].

V. Discussion and Conclusions

We have presented a method for measuring and visualizing differences in the frequency or rate of occurrence of t-way combinations in different data sets [23]. This measure, combination frequency differencing (CFD), has potential use in a variety of applications. Initially applied to challenge-response pairs for physical unclonable functions (PUFs), CFD was shown to provide the ability to identify combinations of bits in the challenge that are more or less strongly associated with particular output values of 0 or 1. In general, the combinations that became apparent in analyzing differences represented a single bit of interest, paired with other bits. See Table III, for example. Differences in the frequency of occurrence of particular value combinations were strongly associated with the success of machine learning attacks on PUFs. Thus the level of difference appears to be a useful metric for vulnerability of PUFs to machine learning attacks.

In future research we hope to develop ways to trace these strongly non-uniform bit combination associations to the hardware components that produce them. This ability might be useful in the design and development of PUFs, to identify design weaknesses, and correct them before deployment. It may be useful to find bits with outsize influence in determining the output of the PUF. If a bit-position is found to be 'compromised', the design could be changed with respect to that bit position to rectify the weakness. Additional metrics are also being considered for this analysis, such as density of differences between class and non-class instances.

We are also investigating using frequency differences directly for identifying combinations that predict class membership [12]. A longer term prospect could be to apply the CFD analysis to adversarial imaging, using CFD to determine which pixels or bits have the most significant influence on neural net output for such image recognition problems.

References

[1] D. R. Kuhn, I. Dominguez Mendoza, R. N. Kacker, and Y. Lei. Combinatorial coverage measurement concepts and applications. In *Software Testing, Verification and Validation Workshops (ICSTW), 2013 IEEE Sixth Intl Conf on*, pages 352–361, 2013.

[2] M. Fifo, E. Enoiu, and W. Afzal. On measuring combinatorial coverage of manually created test cases for industrial software. In *2019 IEEE Intl Conf on Software Testing, Verification and Validation Workshops (ICSTW)*, pages 264–267, 2019.

[3] R. Smith, D. Jarman, J. Bellows, R. Kuhn, R. Kacker, and D. Simos. Measuring combinatorial coverage at adobe. In *2019 IEEE Intl Conf on Software Testing, Verification and Validation Workshops (ICSTW)*, pages 194–197, 2019.

[4] M. Ozcan. Applications of practical combinatorial testing methods at siemens industry inc., building technologies division. In *2017 IEEE Intl Conf on Software Testing, Verification and Validation Workshops (ICSTW)*, pages 208–215, 2017.

[5] D. R. Kuhn, R. N. Kacker, Y. Lei, and D. E. Simos. Combinatorial methods for explainableAI. In *2020 IEEE Intl Conf on Software Testing, Verification and Validation Workshops (ICSTW)*, pages 167–170, 2020.

[6] L. Kampel, D. E. Simos, D. R. Kuhn, and R. N. Kacker. An exploration of combinatorial testing-based approaches to fault localization for explainableAI. *Annals of Mathematics and Artificial Intelligence*, pages 1–14, 2021.

[7] G. Barash, E. Farchi, I. Jayaraman, O. Raz, R. Tzoref-Brill, and M. Zalmanovici. Bridging the gap between ml solutions and their business requirements using feature interactions. In *Proc. 2019 27th ACM Joint Meeting on European Software Engineering Conf. and Sym. the Foundations of Software Engineering*, pages 1048–1058, 2019.

[8] E. Lanus, L. Freeman, D. R. Kuhn, R. N. Kacker, and Y. Lei. Combinatorial testing metrics for machine learning. In *2021 IEEE Intl Conf on Software Testing, Verification and Validation*, 2021.

[9] T. Cody, E. Lanus, D. D. Doyle, and L. Freeman. Systematic training and testing for machine learning using combinatorial interaction testing. *arXiv preprint arXiv:2201.12428*, 2022.

[10] P. Arcaini, A. Calò, F. Ishikawa, T. Laurent, X.-Y. Zhang, S. Ali, F. Hauer, and A. Ventresque. Parameter-based testing and debugging of autonomous driving systems. In *2021 IEEE Intelligent Vehicles Symposium Workshops (IV Workshops)*, pages 197–202. IEEE, 2021.

[11] D. R. Kuhn, M S Raunak, and R. N. Kacker. Combination frequency differencing. Technical report, National Institute of Standards and Technology, November 2021.

[12] D R Kuhn, MS Raunak, and R N Kacker. Combinatorial coverage difference measurement. Technical report, National Institute of Standards and Technology, June 2021.

[13] D. R. Kuhn, R. N. Kacker, and Y Lei. Combinatorial coverage as an aspect of test quality. *CrossTalk*, 28(2):19–23, 2015.

[14] G. E. Suh and S. Devadas. Physical unclonable functions for device authentication and secret key generation. In *2007 44th ACM/IEEE Design Automation Conf*, pages 9–14, 2007.

[15] C. Böhm and M. Hofer. *Physical unclonable functions in theory and practice*. Springer Science & Business Media, 2012.

[16] U. Rührmair, J. Sölter, and F. Sehnke. On the foundations of physical unclonable functions. *IACR Cryptol. ePrint Arch.*, 2009:277, 2009.

[17] C. Herder, M.-D. Yu, F. Koushanfar, and S. Devadas. Physical unclonable functions and applications: A tutorial. *Proceedings of the IEEE*, 102(8):1126–1141, 2014.

[18] R. Maes and I. Verbauwhede. Physically Unclonable Functions: A Study on the State of the Art and Future Research Directions. In A. Sadeghi and D. Naccache, editors, *Towards Hardware-Intrinsic Security*, Information Security and Cryptography, pages 3–37. Springer Berlin Heidelberg, 2010.

[19] U. Rührmair, F. Sehnke, J. Sölter, G. Dror, S. Devadas, and booktitle=Proc of the 17th ACM Conf on Computer and communications security pages=237–249 year=2010 Schmidhuber, J. Modeling attacks on physical unclonable functions.

[20] H. Su, M. Zwolinski, and B. Halak. A machine learning attacks resistant two stage physical unclonable functions design. In *2018 IEEE 3rd Intl Verification and Security Workshop (IVSW)*, pages 52–55, 2018.

[21] E. Strieder, C. Frisch, and M. Pehl. Machine learning of physical unclonable functions using helper data. *IACR Trans. Cryptographic Hardware and Embedded Systems*, pages 1–36, 2021.

[22] I. H. Witten and E. Frank. *Data Mining: Practical Machine Learning Tools and Techniques, 2nd Edition*. Morgan Kaufmann, 2005.

[23] D. R. Kuhn, M S Raunak, and R. N. Kacker. CFD analysis of PUF data. https://github.com/raunaknist/cfd-puf/, 2022.

[24] M. Majzoobi, F. Koushanfar, and M. Potkonjak. Testing techniques for hardware security. In *2008 IEEE Intl Test Conf*, pages 1–10, 2008.

[25] Arunkumar Vijayakumar and Sandip Kundu. A novel modeling attack resistant PUF design based on non-linear voltage transfer characteristics. In *2015 Design, Automation & Test in Europe Conference & Exhibition (DATE)*, pages 653–658.

[26] C. Helfmeier, C. Boit, D. Nedospasov, and J.-P. Seifert. Cloning physically unclonable functions. In *2013 IEEE Intl Sym. Hardware-Oriented Security and Trust (HOST)*, pages 1–6, 2013.

[27] C. Helfmeier, C. Boit, D. Nedospasov, S. Tajik, and J.-P. Seifert. Physical vulnerabilities of physically unclonable functions. In *2014 Design, Automation & Test in Europe Conf & Exhibition (DATE)*, pages 1–4.

[28] Y. Ikezaki, Y. Nozaki, and M. Yoshikawa. Deep learning attack for physical unclonable function. In *2016 IEEE 5th Global Conf on Consumer Electronics*, pages 1–2, 2016.

[29] I. Saha, R. Rahul Jeldi, and R. S. Chakraborty. Model building attacks on physically unclonable functions using genetic programming. In *2013 IEEE Intl Sym. Hardware-Oriented Security and Trust (HOST)*, pages 41–44, 2013.

[30] W. Che, V. K Kajuluri, M. Martin, F. Saqib, and J. Plusquellic. Analysis of entropy in a hardware-embedded delay puf. *Cryptography*, 1(1):8, 2017.

[31] P. Koeberl, J. Li, A. Rajan, and W. Wu. Entropy loss in puf-based key generation schemes: The repetition code pitfall. In *2014 IEEE Intl Sym. Hardware-Oriented Security and Trust (HOST)*, pages 44–49, 2014.

[32] C. Gu, W. Liu, N. Hanley, R. Hesselbarth, and M. O'Neill. A theoretical model to link uniqueness and min-entropy forPUF evaluations. *IEEE Trans. Computers*, 68(2):287–293, 2018.

[33] A. Schaub, J. Danger, S. Guilley, and O. Rioul. An improved analysis of reliability and entropy for delay PUFS. In *2018 21st Euromicro Conf on Digital System Design (DSD)*, pages 553–560, 2018.

[34] A. Vijayakumar, V. C. Patil, C. B. Prado, and S. Kundu. Machine learning resistant strong puf: Possible or a pipe dream? In *2016 IEEE Intl Sym. hardware oriented security and trust (HOST)*, pages 19–24.

Disclaimer