Enhancing Solver-based Generic Side-Channel Analysis with Machine Learning

Kaveh Shamsi
kaveh.shamsi@utdallas.edu
ECE Department, University of Texas at Dallas
Richardson, Texas, USA

Guangwei Zhao
guangwei.zhao@utdallas.edu
ECE Department, University of Texas at Dallas
Richardson, Texas, USA

Abstract

Generic side-channel attacks, unlike traditional CPA/DPA which are specialized to individual cryptographic circuits, can take in an arbitrary circuit or its power model and try to learn user-designated secrets from its side-channel traces. In this paper, we explore the use of machine learning in the context of such generic attacks. We discuss and demonstrate the challenges of using end-to-end (trace-to-key) learning on generic circuits with larger key sizes. We instead propose a couple of ways to use machine learning to assist recent pseudo-Boolean solver-based generic attacks and report their effectiveness on FPGA power traces.

CCS Concepts

• **Security and privacy → Side-channel analysis and countermeasures**.

Keywords

Circuit Learning, Side-Channel Attacks, Generic Attacks

ACM Reference Format:
Kaveh Shamsi and Guangwei Zhao. 2023. Enhancing Solver-based Generic Side-Channel Analysis with Machine Learning. In *Proceedings of the Great Lakes Symposium on VLSI 2023 (GLSVLSI '23), June 5–7, 2023, Knoxville, TN, USA.* ACM, New York, NY, USA, 6 pages. https://doi.org/10.1145/3583781.3590237

1 Introduction

In recent years, the growing reliance on electronic systems has brought new security threats to the forefront. One such threat is the possibility of side-channel attacks, where an adversary can extract sensitive information from a system by analyzing its power consumption, electromagnetic emissions, or other unintended signals.

The field of side-channel analysis gained momentum with the seminal work of Kocher et al. [6] with the introduction of differential power analysis (DPA). This was followed by the introduction of other techniques such as correlation power analysis (CPA)[1], mutual information analysis (MIA)[5], and template attacks (TA)[2] over the years.

These frameworks were mostly developed in the context of attacking cryptographic hardware, which is a special class of high entropy circuits/functions. Generic side-channel attacks are those that unlike specialized attacks can be applied to arbitrary circuits as long as they are represented with some formalism. e.g. Boolean circuits of the form $c_e(k, x)$ where k is an unknown secret key vector which is the target of the attack, and x is an observable (sometimes controllable) primary input vector. An adaptive attack is one where the attacker can adjust their queries in response to previously observed side-channel information.

Recently in [7] a pair of generic adaptive attacks called PowerSAT attacks were proposed using pseudo-Boolean SAT solving. These attacks were inspired by SAT-based functional attacks against logic locking [4, 8]. These attacks use a Boolean satisfiability (SAT) solver to find informative input patterns, query them on an oracle of $c_e(k_*, x)$ to find the value of k_*. The PowerSAT attacks were developed by extending this approach to interactions with side-channel oracles instead of functional ones.

Orthogonally, there has been an array of recent advances in side-channel analysis using machine-learning techniques specifically deep learning [3]. These attacks similar to traditional attacks focus on cryptographic hardware. In this paper, we explore the use of machine learning (ML) in the context of generic attacks. We present the following contributions:

- We develop and deploy variants of end-to-end (trace-to-key) ML-based side-channel analysis against a generic set of circuits: ISCAS-85 benchmark circuits, comparators, and substitution permutation networks. We show that these attacks become less effective as the key size grows.
- We propose instead to use ML-based techniques to assist formal solver-based approaches such as PowerSAT. We show using FPGA traces how ML algorithms can be used for a) mapping hardware traces to hamming weights and b) trace comparison, both of which are important subroutines in PowerSAT attacks.

Organization: Section 2 discusses preliminaries, Section 3 end-to-end ML attacks and related experiments, Section 4 the same with hybrid attacks, and Section 5 the conclusion.

2 Preliminaries

Circuit Learning. Take a circuit $c_e : K \times X \to Y$ with K, X and Y being the l-bit key, n-bit input, and m-bit output spaces respectively. Functional oracle-guided circuit learning/deobfuscation is the problem of finding a key k_* given an oracle $c_o : X \to Y$ such that $\forall x \in X,\ c_o(x) = c_e(k_*, x)$.

Side-Channel Analysis. The hardware implementation of circuit $c_e(k, x)$ emits a dynamic power side-channel signal $\mathcal{L}(k_*, x, \delta x)[t] = f(k_*, x, \delta x)[t] + \mathcal{N}[t]$ when its input changes from x by δx, where f is the power model (function from input and key to side-channel signal) and $\mathcal{N}$ is the non-input-dependent noise. CPA is based on

GLSVLSI '23, June 5–7, 2023, Knoxville, TN, USA
© 2023 Copyright held by the owner/author(s).
ACM ISBN 979-8-4007-0125-2/23/06.
https://doi.org/10.1145/3583781.3590237

finding a key k that maximizes the correlation between some power model $h(k, x, \delta x)$ and a set of collected traces $T = \{\mathcal{L}(k_*, x_i, \delta x_i)[t]\}$.

SAT-based Circuit Learning. The SAT-attack proposed in [8] is a functional oracle-guided circuit learning procedure. It first builds a miter circuit $M = (c_e(k_1, x) \neq c_e(k_2, x))$. This miter is converted to a conjunctive normal form (CNF)-SAT formula and solved using a SAT-solver. The input $\hat{x}$ that satisfies M is called a discriminating input pattern (DIP) as it is an input on which there exist two keys k_1 and k_2 under which the circuit c_e produces different outputs. $\hat{x}$ is queried on the functional oracle: $y_i = c_o(x_i)$. This observation (called in input-output constraint) is reinforced back into the solver by appending $y_i = c_e(k, x)$ to M. Once the set of constraints+miter become unsatisfiable, it can be proven that the key that satisfies the constraints at this point must be a functionally correct key.

PowerSAT Attacks. The first variant of the PowerSAT attack, named PowerSATeq, extends the above (uncertainty-sampling) SAT-based circuit learning paradigm to power analysis attacks. The attacker will instead of the functional oracle have access to a side-channel oracle that returns traces on queries of $(x, \delta x)$. Also, the attacker must have a power model of the circuit $h_e(k, x, \delta x)$. A power miter is built by creating disagreements in the power model space $PM = (h_e(k_1, x, \delta x) \neq h_e(k_2, x, \delta x))$. The io-constraints in PowerSATeq are equality conditions of the form $[hw(t_i) = (h_e(k, x, \delta x)]$. One way to derive the power model h_e is to count the number of bit flips that occur in the circuit when the input changes from x to $x \oplus \delta x$. This can be built by taking the sum $\sum_{w \in W_e} w(k, x) \oplus w(k, x + \delta x)$ over the wires of c_e, W_e. Such conditions that include Boolean functions and summations are called pseudo-Boolean (PB) formulae. PB-solving has been an active area of research for a couple of decades. One common way to solve PB formula is to convert the summation part to adders/sorter networks that are then converted to CNF formula and solved using conventional SAT-solvers. This is what the PowerSAT attack in [7] used.

PowerSATdiff is a more robust variant of the PowerSAT attacks. It operates by finding pairs of input patterns for which the comparison of their power consumption has different directions for different keys. This involves making two input queries, comparing their traces, and appending the result as a constraint to the PB formula.

PowerSAT Attacks on Real Traces. PowerSAT attacks can be mounted easily against a simulated hamming weight oracle. i.e. an oracle that upon receiving $x, \delta x$ returns a precise bit-flip count that corresponds with the hamming weight power model used in the attack h_e. However, when it comes to practical attacks there is a question of converting the real power trace which is a time-series signal to a precise bit-flip count. The original PowerSATeq in [7] did not address this question. We propose an ML-based solution to this challenge in Section 4.

For PowerSATdiff, a direct bit-flip count derivation from the trace is not required. Instead, the attacker only needs to determine which one of two queried traces tr_1 and tr_2 correspond to a higher bit-flip count in the circuit c_e. In [7] authors proposed a couple of ways to do this by looking for peaks/valleys in the subtraction of the two traces from each other. In this paper, we improve the accuracy of this approach in Section 4.

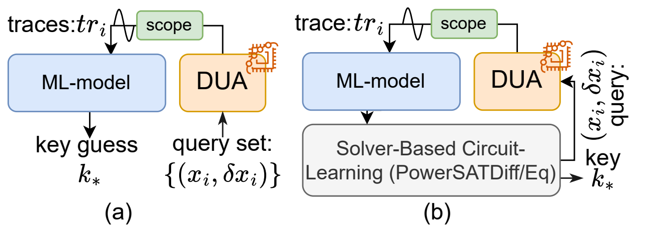
traces: tr_i
scope
ML-model
DUA
key guess k_*
query set: $\{(x_i, \delta x_i)\}$
(a)
trace: tr_i
scope
ML-model
DUA
query: $(x_i, \delta x_i)$
Solver-Based Circuit-Learning (PowerSATDiff/Eq)
key k_*
(b)

Figure 1: a) end-to-end ML-based side-channel analysis. b) integration of ML with solver-based (PowerSAT-style) attacks.

3 ML-based Generic Side-Channel Analysis

The generic circuit learning from the side-channels problem is as follows: Given a circuit $c_e(k, x)$ the goal is to interact with a trace oracle of this circuit and use the collected traces to recover the key k.

An end-to-end approach for using ML in this task is simple: Train an ML model F to take in observed traces, and produce correct keys (see Figure 1a). The training data here will be a corpus of existing trace/key pairs collected either from a real device or some simulation. This is precisely what many existing ML-based side-channel analysis approaches do. DeepSCA[3] for instance takes a few traces, passes them to a neural network with a few layers, and outputs subkeys.

3.1 Key Partitioning

A critical question here is how the model should produce the key. A neural network (NN) classifier can output a one-hot class among C different classes using C different output neurons. For a generic circuit with l key bits, such an approach requires 2^l different output neurons. This means that for circuits with non-trivial key bit counts (10>) the model itself can become prohibitively large.

The approach used in DeepSCA to resolve this issue is to partition the key into smaller chunks the same way that traditional CPA/DPA/TA all do. The partitioning for cryptographic hardware follows intuition. For instance, for AES, the 128-bit key is broken down into 16 8-bit chunks. This translates to 256 different output classes per chunk. A softmax output layer with 256 outputs can be used for classifying each chunk. One can train 16 different models to predict each chunk. Or alternatively one can use a few initial shared layers that collect trace features, followed by different output layers for the final classification.

For cryptographic circuits, the partitioning is derived intuitively from the design and structure of the hardware. For AES, the 8-bit chunks correspond to the 8-bit that enter each 8-bit s-box. For generic circuits however, it is not obvious how the key should be partitioned.

We can state the question more formally for the case of CPA. In CPA the chunks are optimized in some given order. On each lc-bit key chunk all 2^{lc} possible key values are explored one by one and the trace correlation with k_{lc} loaded into the hypothesis key vector is computed. Given an ordering O_c of the different chunks, and partitioning Kp, an optimal partition is one in which the number of iterations and traces needed for a correct key recovery is minimized. It is not difficult to show that the above problem is at least NP-hard. So finding such an optimal order is likely a very difficult computational task.

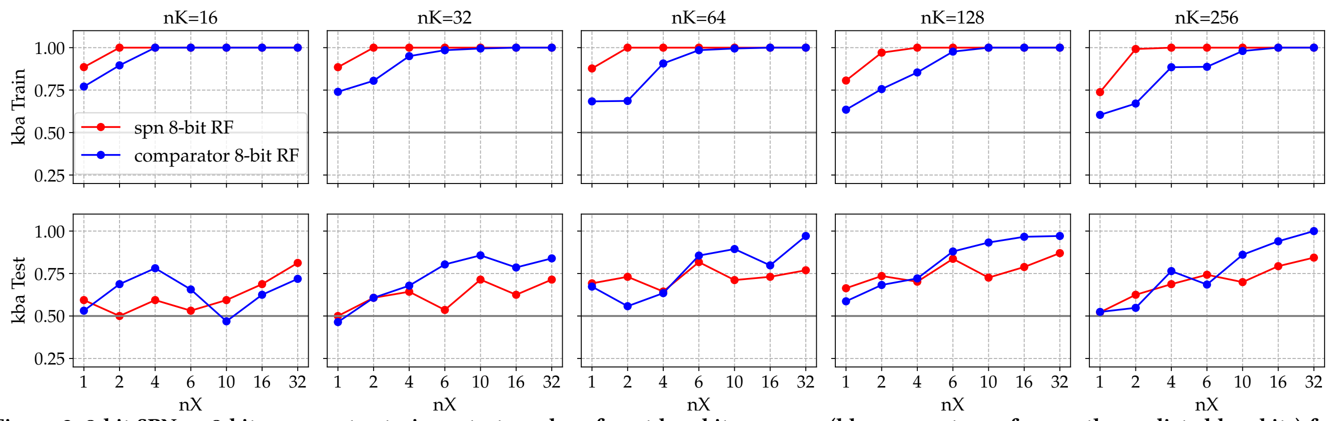

Figure 2: 8-bit SPN vs 8-bit comparator train-vs-test random-forest key-bit-accuracy (kba: percentage of correctly predicted key bits) for different nX and nK parameters on simulated noise-free traces (hamming weights).

We can however apply some heuristics. If two logic cones g_1 and g_2 have non-intersecting input and key vectors, then the change in the key bits of g_1 will have no impact on the bit-flip-rate of g_2 for a fixed $(x, \delta x)$. By iterating through the different key possibilities for g_1 the local optima of the trace correlation function will provably be the correct key for g_1. Then the attack can move to g_2 and repeat the process. Therefore, a somewhat sophisticated heuristic key partitioning will be to find cones with non-intersecting sets of keys.

A less computationally intensive approach that relates to the above approach is to search for keys that are near each other in a graph of the circuit. One can start a BFS search from a given key and find all the keys that are within a certain distance of it and append them to the same partition as long as the size of the partition stays below a certain value (e.g. 8). Note the order in which keys are picked here will impact the final result in non-trivial ways.

3.2 Experiments: Small Key Size

We implemented the above end-to-end-ML side-channel attack against generic circuits. Experiments were run in Python3.10 on a Threadripper 3990X CPU with 128 threads and 256GB of memory with a 2GB memory limit per task.

We begin with the case where key partitioning is not needed. Namely circuits with a small number of key bits (8-bits). The power consumption was simulated in software as the bit-flip-count (hamming sum/weight). In this idealized setting, we used ML models to map from a set of traces T to the correct 8-bit key k_* for generic circuit $c_e(k_*, x)$. The trace set is comprised of the noise-free hamming weight of the circuit on nX randomly selected input+flip patterns: set $X_q = \{(x_i, \delta x_i)\}$. This is repeated for nK different random key vectors: set K_q. The training data hence consists of $|K_q||X_q|$ traces: for each input in X_q, and for each key in K_q collect a trace and append to T.

For each possible $k_* \in K_q$ the goal of the ML attacker is to predict this k_* by looking at $|X_q|$ traces on the X_q inputs. The number of inputs in X_q captures the number of traces that the final attacker will look at to predict the key. So a single-trace attack corresponds to $|X_q| = nX = 1$. The training dataset therefore, consists of $|K_q|$ (input-output) samples.

We used the random-forest (RF) model from `sklearn` to predict one-hot encoded 8-bit/256-possibility keys in the above setting. We used RF instead of a deep NN as deep-NNs took more training time and produced equal or subpar results on these tasks compared to the RF. The results for select generic circuits: a) an 8-bit comparator and b) an 8-bit single-round substitution permutation network (SPN) can be seen in Figure 2. The SPN is generated following the cryptographic SPN construct by taking 8 input bits x, first XORing them with 8 key bits in a key-mixing layer $mx = k \oplus x$, and then sending the result through a substitution-box constructed by synthesizing a random 8-bit truth-table. Finally, if multiple parallel sboxes exist in multiple rounds, they go through a permutation layer that shuffles the bits between each two rounds.

Training Accuracy. As can be seen from the results, the training accuracy of the model becomes perfect as soon as the number of input queries is increased beyond a certain level. For an SPN 2-queries/2-traces are sufficient to with 100% accuracy determine the key vector k from the vector of two hamming weights (t_1, t_2), when the model has previously seen this exact sample. Note that what this means is that the function $F(t_1, t_2) \rightarrow k$ which maps from the two observed hamming weights to a key guess becomes unambiguous at $nX \geq 2$ for 8-bit SPNs. i.e. the key of an 8-bit SPN is fully resolvable from its noise-free flip rate on two input queries.

In other words, the training accuracy here is a measure of the learnability of the key from bit flip counts. Per the upper row in Figure 2 it can be seen that comparators are not as learnable as SPNs from a small number of queries/traces. 8-bit comparators need something on the order of 16 input queries to resolve the key. This is in line with theory from [7] that highly entropic circuits like SPNs not only require fewer functional queries to learn than low entropy circuits like comparators, but this loose relationship also holds for hamming weight queries.

For small key sizes, we can build the full mapping between traces and keys. We can place the training data traces $(t_1, t_2, ..., t_{nX})$ into a dictionary that maps to correct keys. If multiple trace vectors of this form, all map to the same key vector, or if one trace vector maps to multiple possible keys, then there is a theoretical ambiguity in the mapping between traces and keys. i.e. the true correct key may be unlearnable/ambiguous given the observation set.

For the 8-bit key case, we verified that there is true ambiguity for both comparators and SPNs for $nX = 1$, but the ambiguity quickly dissipates for SPNs as nX increases, whereas for comparators it

		#k=16		#k=32		#k=64	
bench	#i/#o/#g	kba-train	kba-test	kba-train	kba-test	kba-train	kba-test
c432	36/7/160	0.99	0.74	1.00	0.63	1.00	0.57
c499	41/32/202	0.85	0.66	1.00	0.57	1.00	0.52
c880	60/26/383	1.00	0.81	1.00	0.65	1.00	0.57
c1355	41/32/546	0.99	0.73	1.00	0.62	1.00	0.55
c1908	33/25/880	1.00	0.76	1.00	0.65	1.00	0.56
c2670	157/64/1193	0.90	0.72	1.00	0.66	1.00	0.57
c3540	50/22/1669	1.00	0.80	1.00	0.65	1.00	0.57
c5315	178/123/2307	0.97	0.77	1.00	0.66	1.00	0.58
c6288	32/32/2416	1.00	0.77	1.00	0.60	1.00	0.53
c7552	206/107/3512	1.00	0.75	1.00	0.67	1.00	0.58
spn	16/16/1766	1.00	0.74	1.00	0.60	1.00	0.54
comp	16/1/31	1.00	0.85	1.00	0.69	1.00	0.59

Table 1: End-to-end ML attacks on circuits with larger keys, using distance-based partitioning, with dataset params nK=512, nX=32, and a random-forest model.

takes longer. Both eventually turn into 1-to-1 mappings, i.e. fully learnable relations.

Testing Accuracy. The testing/validation accuracy of this approach however is not impressive. The testing accuracy refers to the accuracy of the model trained on the training data, then evaluated on a set of samples that were not shown to the model during training. We use a typical 80/20 train/test split, i.e. 20% of the data samples during training were kept aside for testing.

In the context of our end-to-end attack, this means that there are keys that are members of Kq, that the model was never trained on predicting during the training. This is akin to training a classification model where certain classes were simply not present in the dataset. As such, this type of low testing accuracy is not due to overfitting.

The above results indicate an important limitation of end-to-end ML attacks. If the key space is large, then all possible keys cannot be placed in the training set Kq. Therefore, the model's performance on unseen keys will be in question. If the key is partitioned into smaller chunks, then even though each full key vector is not seen, each chunk range may be fully explored. We show shortly how, with reasonably sized training datasets, this doesn't perform well either.

3.3 Experiments: Large Key Size

We implemented the above end-to-end ML approach on circuits with more than 8 bits of keys. Here we used the distance-based key partitioning strategy discussed earlier. The large key vector was split into 8-bit partitions and each partition's correct values were used as training samples in a multi-output ML model (random-forest).

Table 1 shows the results for this task. Here the circuits were locked with (or inherently contained) random XOR/XNOR insertion with 16, 32, 64 key bits. The training dataset parameters were $nX = 32$ and $nK = 512$ i.e. 16384 traces. As can be seen, training accuracy remains close to perfect. The testing accuracy falls to random guessing as the key size is increased. To put this in context, note that CPA with fewer traces than 16384 will perform very well with the same key sizes per [7]. PowerSATdiff attacks can provably recover keys in this case with tens of traces/queries. Note that these results are all in the ideal noise-free setting (traces = hamming weight).

Noisy Evaluations. Figure 3 shows results for end-to-end generic ML in the setting where the model is trained on noise-free ideal hamming weights, but then evaluated on noisy inputs (for seen/unseen training/testing patterns). The results show how the presence of noise can diminish the model's performance on both seen/unseen patterns albeit with some resiliency.

4 Hybrid Approach: ML + Solver Attacks

The above results indicate an important challenge for end-to-end (trace-to-key) ML-based attacks: For an effective end-to-end ML attack, the model will have to learn the relationship between circuit hamming weights and keys. This relationship is formally captured in $h_e(k, x, \delta x)$. The typical $h_e(k, x, \delta x)$ function consists of two copies of a generic Boolean circuit, an array of XOR gates, followed by some binary adder. The ML model will have to learn the inverse of this function $h^{-1}(t) \rightarrow (k, x, \delta x)$. This (as also mentioned in) [3] may be a difficult task.

More importantly, note that the ML model in these end-to-end attacks was not even given the structure of c_e/h_e. This means that we are asking an ML model to reverse h_e, without even seeing its description. Note that this is not a limitation imposed by the threat model, as the threat model assumes the attacker is given the netlist/description of c_e and hence some insight into the power model h_e.

The question of generalizability and learnability for an ML model here reduces to the following: given an unseen circuit c_e, if one makes N observation on the hamming weight function of c_e, can one predict the hamming weight of c_e on an unseen new $N + 1$th input? Formally, the hypothesis space for a c_e-blind attacker is large and uniform. For the c_e-blind attacker there are infinitely many circuits that produce the first N hamming weight observations that produce divergent hamming weights on a new unseen $N + 1$ query.

Note that the above issue is not present in the case of a PowerSAT-style attack. The PowerSAT attack is intimately aware of c_e/h_e. It can both simulate and invert these functions using a complex and sophisticated PB-SAT-solver decision procedure. Also via complexity reduction arguments, one can reason that an alternative better way to reverse a general h_e, would imply an alternative better way to solve general PB-formulae.

The above discussion serves as a motivation to perhaps combine the two approaches. The ML approach's main strength is in the modeling of behavior that is not readily captured in the PB formulation. Once the PB-based PowerSAT attacks are given accurate queries they can perform formal reasoning to recover correct keys. However, their basic implementations are highly sensitive to errors. The negation of a single variable in a CNF formula can have dramatic impacts on the status of the instance. Yet worse, in (Power)SAT attacks, an error in one step of the attack can cascade into more and more divergent and unpredictable behavior in future iterations.

We propose a couple of ways to use ML to assist PowerSAT attacks discussed herein:

Trace to Hamming Weight Bucket for PowerSATeq: The baseline hamming weight PowerSATeq attack requires traces collected from the oracle $tr_i[t]$ which are in the real-world setting time-series signals, to be mapped to integer values compatible

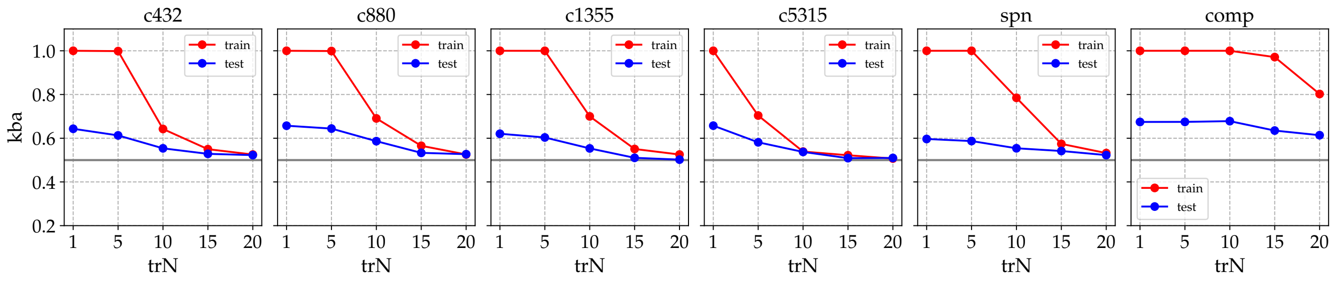

Figure 3: Training on noise-free data, evaluation on noisy data. A random Gaussian noise value is added to the raw hamming vectors with $\sigma = trN\% \times [max(hw) - min(hw)]$. The baseline model was random-forest trained on $nK = 512$, $nX = 32$ datasets.

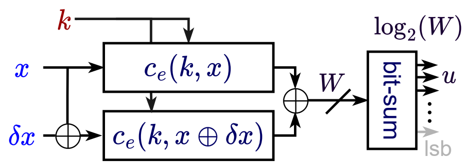

Figure 4: The hamming weight model in PowerSAT attacks consists of two copies of the circuit running on x and $x \oplus \delta x$. Pair-wise XOR of the internal wires is summed through an adder. By discarding the lower bits of the adder a natural hamming weight bucketing can be implemented.

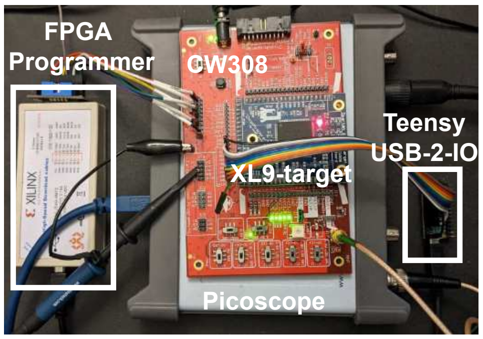

Figure 5: Trace collector hardware setup. CW308 with a Spartan6 XL9 FPGA target. The Teensy3.2 board is used to convert USB commands from the PC to FPGA x and δx fields. Traces are collected via the USB3 Picoscope5000 device.

with the hamming weight power model. i.e. a time-series trace $tr_i[t] \leftarrow toracle(k_*, x, \delta x)$, will have to be converted to a corresponding hamming weight $h_e(k, x, \delta x)$ before inclusion into the PowerSAT attack.

We can use an ML model to perform this conversion. By querying the trace oracle on a set of Xq and Kq, we can try to learn the mapping from traces to flip counts. This mapping should be much easier to learn than the end-to-end relationship between traces and keys. Note that for a non-ML approach to this problem, the attacker needs to study different traces, find points-of-interest (poi's) then decide manually what values on these points correspond to exactly how many bit flips in the device under attack (DUA).

Predicting the precise hamming weight with an ML model can be challenging. We can instead map from traces to hamming weight buckets/bins. Each bucket is a range of hamming weights $bckt_i : l_i < hw < h_i$. By dividing the full range of hamming weights into a power of 2 we can in fact nicely integrate these bucket observations with PowerSATeq. In the PowerSATeq model of the circuit's power consumption (seen in Figure 4) a bit-adder is typically used to sum bits. By discarding the lower bits of this adder and focusing on the u upper bits, we can effectively capture hamming weight buckets each including a $max(hw)/2^u$ range.

Figure 6 shows the results of the above approach. Random-forest regression and classification models were trained on 2000 traces collected from the hardware setup shown and described in Figure 5 for the different ISCAS and SPN circuits. For the regression models, the output is a real value whose mean square distance to the correct integer bit-flip count is minimized. For the classifier models, the output is an integer class corresponding to the predicted hamming weight. During model evaluation, the predicted integer value is rounded to the nearest bucket edge. Our experiments show a high accuracy for many circuits with 2-8 buckets and a diminishing accuracy for more buckets.

Note that absent trace to hamming-weight mappings, PowerSATeq attacks cannot be mounted against real traces. The original PowerSATeq work [7] did not address this challenge and only demonstrated PowerSATeq attacks against simulated traces. As such this work is the first to propose a potential automated solution to this challenge using ML.

Trace Comparison for PowerSATdiff. Our second integration approach is for trace comparisons in PowerSATdiff. Here, we train an ML model first to predict hamming weights from traces the same as our approach above. Now to compare the hamming weight of two traces (which is a critical part of PowerSATdiff), we compare the predicted hamming weight values from the ML model.

Note that the existing approach for trace comparison per [7] is to take the two traces, subtract them from each other and look for peaks or valleys in the difference. More precisely, first the maximum of the absolute value of the difference is found $p = \text{argmax}_i |tr_1[i] - tr_2[i]|$. Then the value of the difference trace at this peak/valley is tested. A negative/positive value on this point determines the direction of the trace comparison. An addition here in [7] was to take the moving average of the two traces before the subtraction. This methodology was compared to the ML-based technique in Figure 7. As can be seen, the ML-based approach has a higher overall accuracy in resolving comparison directions than non-ML trace differentiation especially when the hamming weight

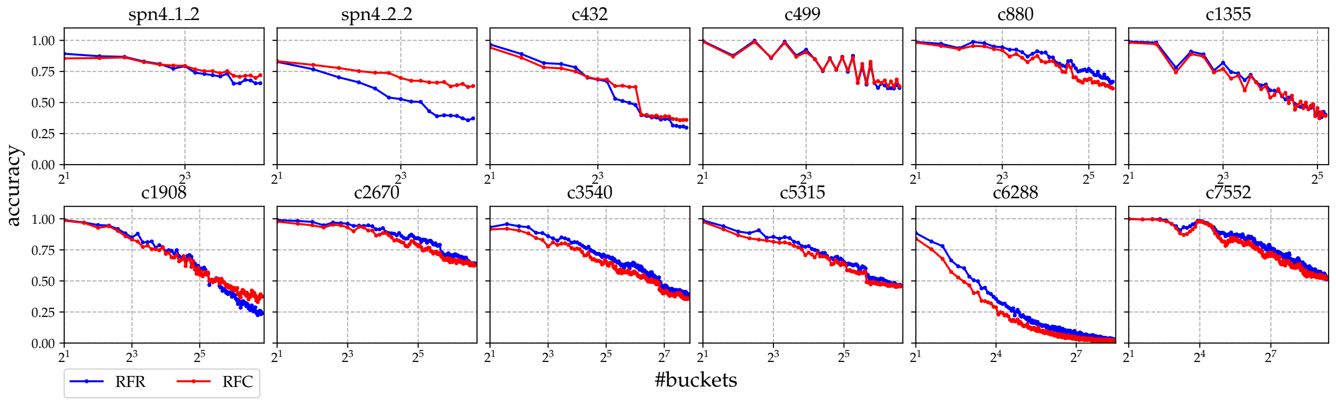

Figure 6: Trace to hamming weight buckets ML task. The random-forest regressor/classifier (RFR/RFC) model is trained on 2000 real traces. It is then evaluated on whether the model's predicted value rounded to the nearest bucket matches the real hamming weight bucket as the number of buckets increased from 2 to the full hamming weight range (number of flipping wires in the circuit).

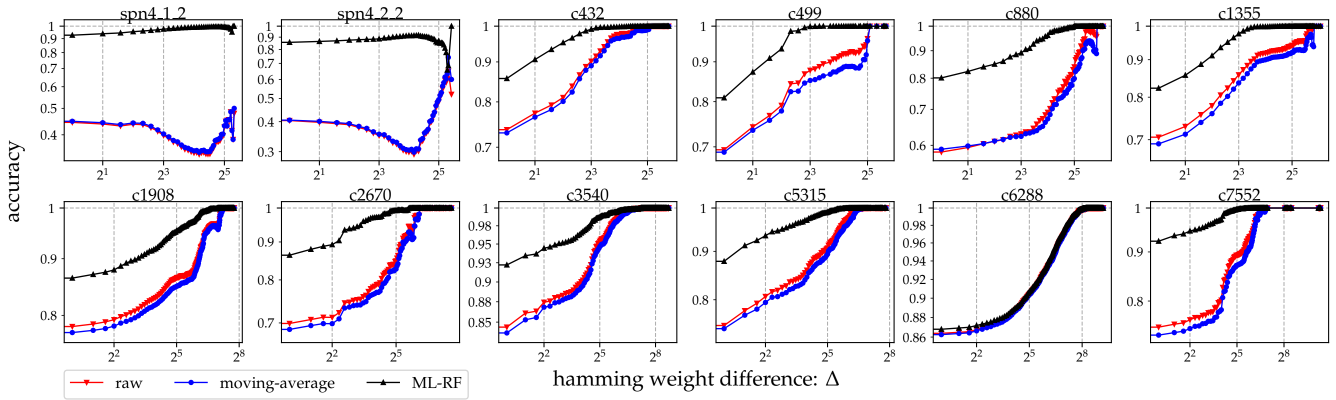

Figure 7: Trace hamming weight comparison. The accuracy of determining which of two given traces (from the FPGA hardware setup in 5) corresponds to a higher hamming weight (more bit-flips): using existing trace differentiation methods from [7], versus comparing the output of the trace-to-hamming-weight ML random forest regressor from Figure 6 tested on 5000 pairs of traces picked randomly from the unseen validation trace set. The Δ on the x-axis captures the real hamming weight difference between the two traces. As can be seen, the closer the hamming weight of the two traces the lower the trace comparison accuracy, but the ML approach (black lines) performs significantly better.

difference between the traces is smaller. This improved accuracy should translate to improved PowerSATdiff attack robustness.

5 Conclusion

In this paper, we showcased some of the limitations of end-to-end machine learning for generic side-channel analysis. We instead proposed a couple of ways to use ML techniques to assist formal solver-based generic PowerSAT attacks. For PowerSATeq ML-based trace-to-hamming-weight mapping overcomes a challenge not addressed before in the literature. For PowerSATdiff, ML-based trace comparison was shown to be more accurate than existing techniques. A full exploration of the multidimensional interplay between the ML techniques proposed here and the formal PowerSAT attacks is left to future work.

Acknowledgments

This work was supported by an award from the US National Science Foundation (NSF-SaTC-2155189).

References

[1] Eric Brier, Christophe Clavier, and Francis Olivier. 2004. Correlation power analysis with a leakage model. In *International workshop on cryptographic hardware and embedded systems*. Springer, 16–29.
[2] Suresh Chari, Josyula R Rao, and Pankaj Rohatgi. 2002. Template attacks. In *International Workshop on Cryptographic Hardware and Embedded Systems*. Springer, 13–28.
[3] Debayan Das, Anupam Golder, Josef Danial, Santosh Ghosh, Arijit Raychowdhury, and Shreyas Sen. 2019. X-DeepSCA: Cross-device deep learning side channel attack. In *Proceedings of the 56th Annual Design Automation Conference 2019*. 1–6.
[4] Mohamed El Massad, Siddharth Garg, and Mahesh V Tripunitara. 2015. Integrated Circuit (IC) Decamouflaging: Reverse Engineering Camouflaged ICs within Minutes.. In *Network and Distributed System Security Symposium (NDSS)*.
[5] Benedikt Gierlichs, Lejla Batina, Pim Tuyls, and Bart Preneel. 2008. Mutual information analysis. In *International Workshop on Cryptographic Hardware and Embedded Systems*. Springer, 426–442.
[6] Paul Kocher, Joshua Jaffe, and Benjamin Jun. 1999. Differential power analysis. In *Annual international cryptology conference*. Springer, 388–397.
[7] Kaveh Shamsi and Yier Jin. 2021. Circuit Deobfuscation from Power Side-Channels using Pseudo-Boolean SAT. In *2021 IEEE/ACM International Conference On Computer Aided Design (ICCAD)*. IEEE, 1–9.
[8] Pramod Subramanyan, Sayak Ray, and Sharad Malik. 2015. Evaluating the security of logic encryption algorithms. In *Proc. IEEE Int. Symp. on Hardware Oriented Security and Trust*. IEEE, 137–143.