On the Practicality of Relying on Simulations in Different Abstraction Levels for Pre-Silicon Side-Channel Analysis

Javad Bahrami[1], Mohammad Ebrahimabadi[1], Sofiane Takarabt[2], Jean-luc Danger[3], Sylvain Guilley[2,3] and Naghmeh Karimi[1]

[1]*University of Maryland Baltimore County, Baltimore, US 21250*

[2]*Secure-IC S.A.S., Think Ahead Business Line, Paris, France*

[2]*LTCI, Télécom Paris, Institut polytechnique de Paris, Paris, France*

Keywords: Side-Channel Attacks; Pre-Silicon Evaluation; Toggle Count; SPICE Simulation; Masked Implementations.

Abstract: Cryptographic chips are prone to side-channel analysis attacks aiming at extracting their secrets. Side-channel leakage is particularly hard to remove completely, unless using a bottom-up approach (compositional security). On the contrary, industrial secure-by-design methods are rather relying on a top-down approach: (would-be) protected circuits are synthesized by Electronic Design Automation (EDA) tools. Tracking that no leakage exists at any refinement stage is therefore a challenge. Experience has shown that multiple leakages can resurge out of the blue when a sound RTL design is turned into a technology-mapped netlist.

Checking for leaks and identifying them is a challenge. When the netlist is unstructured (e.g., it results from an EDA tool), dynamic checking appears as the most straightforward approach. It is feasible, given only a few thousand execution traces, to decide with a great certainty whether a leakage hides at some time samples within the trace or not. In practice, such easy detection is fostered by the fact that the activity of signals in cryptographic implementations (even more true for masked implementations) is almost maximal (=50%).

The remaining question is about the adequate abstraction level of the simulation. The higher as possible abstractions are preferred, as they potentially capture more situations. However, if the simulation is too abstract, it may model the reality inappropriately. In this paper, we explore whether or not an evenemential simulation (toggle count) is faithful with respect to a low-level simulation (at SPICE level). Our results show that both abstraction levels match qualitatively for unprotected implementations. However, abstract toggle count simulations are no longer connected to real SPICE simulations in masked implementations. The reason is that the effect of the random mask is to mix evenemential simulations (which only reflect "approximately" the SPICE reality) together, in such a way that the useful information is lost. Therefore, masked logic netlist implementations shall be analysed only at SPICE level.

1 INTRODUCTION

Electronic circuits which manipulate sensitive information must implement cryptographic algorithms. Typically, such algorithms are block ciphers, whose security relies on a secret key. Embedded devices, like IoTs, resort to lightweight cryptography, such as PRESENT. The most realistic threat against PRESENT is the side-channel analysis (such as power or electromagnetic analysis). Such attack exploits the activity which leaks information about (sensitive) internal signals. Many works attest of the practical feasibility of such attacks (Heuser et al., 2016).

In order to protect the implementation of cryptographic algorithms against Side-Channel Analysis (SCA), protections are deployed. Random masking (or simply "masking") is a protection against such attacks; it consists in mixing the sensitive information with unpredictable random numbers, also referred to as "masks", in a view to decorrelate the observable side-channel activity from the data handled internally of the device. There are several flavors of masking schemes, which differ by their security and cost. Some schemes are more costly (in terms of gate count and delay) because they address better the peculiarity of hardware implementations. Indeed, software masking schemes mainly focus on protecting variables, whereas hardware masking schemes also protect relevantly the combinational logic which lays between the sequential logic carrying the states.

Precisely, all masked logic can be implemented correctly for registers, which are well identified and controlled resources. This is positive as registers are also the resources which are the most prone to leak-

age. But so, when none of the registers are leaking, the second source of leakage is the combinational logic. This part of the circuit is less predictable, as:

- before tapeout, it is mapped, most probably restructured, and potentially even simplified, and
- upon execution, it features races which might differ on the PVTA (Process, Voltage, Temperature, and Transistor Aging) environmental conditions.

There is therefore the need for tools and methods to evaluate masked netlists. Obviously, preventing is better than curing. Therefore, in this paper, we study pre-silicon techniques. The central question is to determine which pre-silicon technique is reasonable.

2 State of the Art

Predicting whether an implementation is vulnerable to side-channel attacks when still at pre-silicon stage has been addressed by many works. Software simulators can be beneficial in this respect, since software leakage is complex. There are multiple points of interest that can leak, hence checking those mitigations in a program which embeds countermeasures is not trivial. Therefore, multiple simulators have been designed to extract the power leakage. They can be dedicated tools, or can consist in a post-processing of execution traces dumped by a COTS tool.

Program Inferred Power Analysis Simulator (PINPAS) is the first exclusive software for analyzing side-channel power written in JAVA which is primary designed for testing smart-cards (Hartog et al., 2003). This tool is able to simulate the execution of a program (in assembly code) in its virtual environment to subsequently analyze its estimated power. Kirschbaum et al. (Kirschbaum, 2007) designed a simulator based on the Cadence NCSim to generate an accurate leakage tracing. The designed tool extracts power traces based on the gate level netlist, cell information, and gates propagation time to count the transitions. However, it requires a lot of information of hardware implementation of devices which is not always accessible. Debande et al. (Debande et al., 2012) designed a profiled simulator based on the stochastic models and the successive values of device's registers. This tool is able to extract leakage traces close to the real measurement. This simulator does not need any specific information of the device, however, profiling step should be performed for every new device. SILK was the first fully open-source simulator for side-channel analysis which is unveil in (Veshchikov, 2014). The flexibility of SILK makes users capable to use different model such as Hamming Weight (HW), Hamming Distance (HD), and even user-defined leakage models.

Most of simulations leverage a **toggle count** leakage model (namely, the Hamming Distance between signals at each trace time sample). Indeed, it has been acknowledged for long in the field of embedded systems security, for instance, in the seminal paper about Correlation Power Analyses (Brier et al., 2004, §2). In this paper, the model focuses on flip-flops. But it has also been extended to combinational logic with fair correlation, e.g., to estimate glitching activity (Liu et al., 2011) (in unprotected circuits tough).

Compared to the aforementioned tools, **SPICE**[1] simulations are more realistic. They can be considered as the most accurate possible simulations, and are seen as references (Li et al., 2005). Notice that the Synopsys VCS and HSPICE netlists we consider are the same, except that Synopsys VCS reports only an "impulse" for each gate when toggles, whereas HSPICE produces a complete waveform (illustrations will be given in Fig. 3 and Fig. 4).

To fill the gap, in this paper, we revisit the use of toggle count vs SPICE simulations, applied to masked netlists. More precisely, we address the relevance of deploying toggle count in evaluating pre-silicon security level:

- If there is a significant correlation between sensitive variables and the toggle count, then there is obviously a vulnerability, hence the netlist must be patched. Notice that, in the past, the compromise in the netlists has been identified based on toggle count analyses. For example, the flaw in (non-glitch resistant) masking of AND gates has been detected by SPICE in (Mangard et al., 2005, §4) and subsequently qualified based on a toggle count in (Mangard et al., 2006, §2.2.). Recall that "glitches" are races between signals, which are likely to generate transient activity observable through side-channels.
- However, assuming that the toggle count analysis does not reveal signification biases (in terms of correlation with the underlying sensitive variable value), can it be claimed that the netlist is secure? Actually, the absence of correlation can also arise from the fact that the "toggle count may not be a valid abstraction" in some cases. To the best of our knowledge, this question has never been formulated as such and consequently has never been thoroughly studied. In this paper, we carry out a meticulous analysis and conclude that, though SPICE is always suitable a pre-silicon simulation tool, toggle count might not be indicated as

[1] For the sake of illustration, we use HSPICE from Synopsys Inc. as an electrical simulation tool for SPICE netlists.

a definitive argument to claim the trustworthiness of masked netlist.

3 PRESENT S-Box Implementations

In this paper, we target PRESENT cipher and mainly the hardware implementations of 2 unprotected and 5 masking-protected implementations of its S-box module. Table 1 compares the targeted implementations regarding the number of gates, equivalent gates (# gates normalized by the # of equivalent 2-input NAND gates), random bits, and propagation delay.

PRESENT is a lightweight block cipher with 64-bit blocks. It includes 31 rounds each consists of a bitwise XOR operation, a non-linear substitution (S-box) layer and a linear permutation layer. Being non-linear and possessing a contrasted confusion coefficient (Fei et al., 2015), the S-box module is a highly appealing target for adversaries to leak sensitive data.

3.1 Unprotected Implementations

Lookup Table (LUT): This baseline architecture can be implemented using a 4-bit LUT with 16 different addressable locations. It is a direct look-up in the plain table: *S-Box* : [`0xc`, `0x5`, `0x6`, `0xb`, `0x9`, `0x0`, `0xa`, `0xd`, `0x3`, `0xe`, `0xf`, `0x8`, `0x4`, `0x7`, `0x1`, `0x2`]. and is subsequently mapped to gates by the synthesizer.

Optimized Implementation (LUT-OPT): This implementation minimizes the number of non-linear (AND/OR) gates by trying different implementations using Boolean Satisfiability (SAT) solvers (Peralta et al., 2022). It has a longer critical path compared to LUT. Indeed LUT-OPT critical path includes more XOR gates, while LUT has more AND/OR gates; thus LUT-OPT has a longer propagation delay.

3.2 Protected Implementations

Global Lookup Table (GLUT): GLUT masking is a function $\mathcal{F}_2^4 \times \mathcal{F}_2^4 \times \mathcal{F}_2^4 = \mathcal{F}_2^{12} \rightarrow \mathcal{F}_2^4$ satisfying

$$Y = GLUT(A, MI, MO)$$

such that:

$$Y \oplus MO = S\text{-}Box(A \oplus MI)$$

here A and Y are the masked input and output, respectively and MI and MO are the input and output masks.

Rotating S-box Masking (RSM): In this first-order secure implementation, even 2nd and 3rd order secure based on (Carlet and Guilley, 2013), to minimize the required randomness, 1) the masks set is a subset of the full mask set; 2) the masks used at the S-box output are deduced deterministically from that at the input, by using the next one (in a circular manner) within the mask set.

In our implementation, since there are only 16 masks in the 4-bit PRESENT S-box, we refrain from using a strict subset of the masks. However, we generated output masks based on the available input randomness (Nassar et al., 2012), thus the output mask MO is generated as below:

$$MO = (MI + 1) \bmod 16$$

where MI and MO are integers $\in \{0, 1, \ldots, 15\}$, computed via $M \in \mathcal{F}_2^4 \mapsto \sum_{i=0}^{3} M_i 2^i \in \{0, 1, \ldots, 15\}$ mapping. So, the relationship between RSM and GLUT can be written as:

$$RSM(A, MI) = GLUT(A, MI, (MI + 1) \bmod 16).$$

From an area perspective, RSM has a more compact architecture compared to the GLUT. Note that the genuine version of RSM does not have S-box input on 8-bit, but solely on 4-bit (Guilley et al., 2017a) (wherein randomness comes from S-box shuffling).

ROM-based RSM (RSM-ROM): A stronger implementation of RSM can be realized using a Read-Only Memory (ROM). For this circuit we target logic designs built only from the instantiation of gates in a Boolean library (Giaconia et al., 2007). Here, initially, the datapath is synchronized for any input configuration which makes input-related deviations of leakage small. Then, the structure is designed with a *one-hot* strategy, i.e., only the required logic is activated, which further contributes to reducing the side-channel footprints.

Gate-Level Masking via Random Sharing (ISW): Proposed by Ishai, Sahai, and Wagner (Ishai et al., 2003), this implementation starts from the *LUT-OPT* netlist, and gradually replaces the gates with their gadgets, in order to deal with the non-linear gates. In this architecture, the gadget for the AND gate requires 1-bit of randomness, i.e., R. Given a random sharing (A_0, A_1) of bit A (where $A = A_0 \oplus A_1$), and a similar sharing for bit B, the AND of A and B denoted as Y is computed as below:

$$\begin{cases} Y_0 &= ((A_1 \wedge B_1) \oplus R) \oplus (A_0 \wedge B_0) \\ Y_1 &= ((A_0 \wedge B_1) \oplus R) \oplus (A_1 \wedge B_0) \end{cases}$$

In the above equations, the order of operations should be followed, thus the implementation must preserve the order of gates in the final netlist. In our implementation, we implemented OR via benefiting from De Morgan's law $\mathrm{OR}(a, b) = \neg\mathrm{AND}(\neg a, \neg b)$. Since combinational gates evaluate their outputs whenever their inputs change, preserving the order can be challenging in ISW due to the race condition. This can lead to a first-order leakage (Roy et al, 2018).

Threshold Implementation (TI): TI is an algorithmic countermeasure against power SCA, which benefits from multi-party computation and secret sharing. TI, alike ISW, divides input bits into $d+1$ shares, however, its underlying logic does not need to preserve gate ordering. Moreover, thanks to its non-completeness property, in TI any output share only depends on d shares of each input. Thus, glitches cannot lead to secret information disclosure in TI. In our netlist, terms of order 3 are required, hence 4 shares are needed; we synthesized a TI-compliant *fully combinational* netlist of PRESENT S-box.

Table 1: Gate-level specification of the targeted S-box implementations.

	LUT	LUT-OPT	GLUT	RSM	RSM-ROM	ISW	TI
# INV	7	1	12	20	750	7	0
# BUF	0	0	0	0	0	1	1
# AND	23	2	580	209	40	10	800
# NAND	0	0	0	0	0	0	0
# OR	11	2	254	102	468	0	0
# NOR	0	0	0	0	0	0	0
# XOR	0	9	0	0	492	26	647
# XNOR	0	0	0	0	0	0	2
# Gates	41	14	846	331	1750	43	1448
# Equ. Gates	36	18	846	317	1322	52	1871
Delay (ps)	160	210	400	330	1710	420	230
# Random Bits	0	0	8	4	4	4	12

4 Background on Walsh-Hadamard Transform

In practice, glitches may result in leaking sensitive data. However, getting information about when glitches happen, and what information they leak may not be easy during the circuit runtime or even in the SPICE level simulations. To investigate the exploitability of glitches Bahrami et al. use Fourier base transformation (the so-called Walsh-Hadamard) to be able to decompose the device's power traces into the unmasked input dependent basis, and in turn identify different contribution of leakage from first-order unmasked text t (Guilley et al., 2017b). The basis is denoted as ψ_u, for all vector u of n-bits ($n = 4$ for PRESENT). By definition, ψ_u function is $t \in \mathcal{F}_2^n \mapsto \psi_u(t) = \frac{1}{2^{n/2}}(-1)^{u \cdot t}$, where the operation $u \cdot t$ denotes the canonical scalar product, i.e., $u \cdot t = \bigoplus_{i=1}^{n} u_i \wedge t_i$. This basis is orthonormal, in that $\langle \psi_u \mid \psi_v \rangle = 0$ if $u \neq v$ and 1 otherwise. In sum, ψ_u shows the interaction between input bits corresponding to the 1 input in u. Here, we ignore the zero-th component, which is the waveform average shape, while extracting all the nonzero components.

Based on (Bahrami et al., 2022), the leakage $f : t \in \mathcal{F}_2^n \mapsto f(t) \in \mathcal{R}$ corresponding to (unmasked) S-Box input $t \in \mathcal{F}_2^n$, can be calculated as

$$f(t) = \sum_u \langle f \mid \psi_u \rangle \psi_u(t) = \frac{1}{2^{n/2}} \sum_u a_u (-1)^{t \cdot u}$$

where a_u (below) is the Walsh-Hadamard transform of f.

$$a_u = \frac{1}{2^{n/2}} \sum_t f(t)(-1)^{t \cdot u}$$

Using $(a_u)_{u \in (\mathcal{F}_2^n)}$ coefficients we can characterize the leakage to find out which combination of bits leaks more (if any). Typically, it can be due to a single bit, when $w_H(u) = 1$ (where w_H denotes to Hamming Weight) or otherwise to multiple bits leaking together. The former case is reasonably attributed to an unfortunate "demasking", while the latter case arises upon complex conditions on the unmasked inputs (connected to "glitch events") (Bahrami et al., 2022). For example in Fig. 3d, the combination of bit 1 and bit 2 leaks more (shown in solid red).

Interpreting the origin of leakage is out of the scope of this paper. We rather focus on comparing the amount of leakage of various implementations.

5 Results and Discussions

We implemented the add-round-key and S-box operations in the first round of the PRESENT cipher with 80-bit keys in the transistor level for the 7 types of S-box (2 unmasked and 5 masked versions) using a 14-nm Fin-FET – regular voltage threshold corner (rvt) – commercial library, and deployed Synopsys HSPICE for the transistor-level simulations at temperature of 125°C and $V_{dd} = 0.8\,V$.

The simulated traces contain two parts: the results of key addition and S-box outputs for each initial n-bit value as well as its following n-bit value. For the analysis, we considered only the power samples of the second part, i.e., when the cryptographic circuit transitions from initial to final value. We sample the power every 1ps after feeding the related final value till each circuit gets stable. To have a fair comparison among all implementations, the final values are generated randomly, yet such that we have equal number of traces from each class, i.e., equal number of unmasked inputs. The initial values generated randomly such that initial unmasked value is '0' (e.g., A_initial $\oplus$ MI_initial = 0 in GLUT). For ISW, TI, and GLUT, totally, we generated 1024 input traces among all possible input combinations, while for RSM and RSM-ROM the total input traces are 256 and for the LUT and LUT-OPT there are 16 traces considering their input sizes. We categorize the power traces into 16 classes based on the value of the unmasked inputs in their final stage, and use the mean trace of each class in our leakage assessments.

To assess the leakage from toggle counts occurring during gate-level simulations, we performed our simulations (using Synopsys VCS) and analysis after back annotating the timing information (SDF file) extracted during the gate-level synthesis.

5.1 Experimental Results

Toggle Count vs. SPICE Power: The first set of results deals with the total power for each class of unmasked plaintext. Recall that we randomly generated input patterns such that in all traces the initial unmasked value is "0000" (referring to class 0) while the final unmasked value can belong to any class (between 0 and 15 in PRESENT). Moreover, to avoid biases the final input patterns were generated such that we have similar number of power traces in each class. Fig. 1 shows the total extracted power values categorized based on the final value of the unmasked input.

Obviously, the unmasked circuitries do not reveal dynamic power consumption for class 0 input as the values represent the dynamic power consumption; thus class 0 for unmasked circuits relates to the case where both initial and final value are 0, so, no dynamic power is consumed. This is in contrast of masking circuits where even we have power consumption for class 0 since owing to the random masks not necessarily initial and final values match.

Based on Fig. 1, the unprotected circuits depict an intra-class match between the toggle count and HSPICE powers, i.e., tentatively class i ($0 \leq i \leq 15$) in both evaluations follow the same trend. However, such trend is not observed in the masked circuits owing to the dependency of power on both input and output masks. The takeaway from these observations is that for unprotected circuits, the toggle counts carry exploitable information and can be used as a source of leakage. However, toggle counts are not exploitable by the attacker in case of protected implementations.

The inter-class correlation between the HSPICE and toggle count results are shown in Table 2. As depicted, this correlation is high for unmasked circuits while low for the masked counterparts. This information shows the practicality of using toggle counts for leaking sensitive data in unmasked circuits, yet not in the masked implementations.

Table 2: Inter-class Pearson Correlation Between Toggle Count and HSPICE powers for each implemented circuit.

	LUT	LUT-OPT	GLUT	RSM	RSM-ROM	ISW	TI
ρ	0.78	0.78	-0.21	-0.23	-0.21	0.21	0.21

In order to validate that our simulations of masking schemes implementation is correct, we carry out

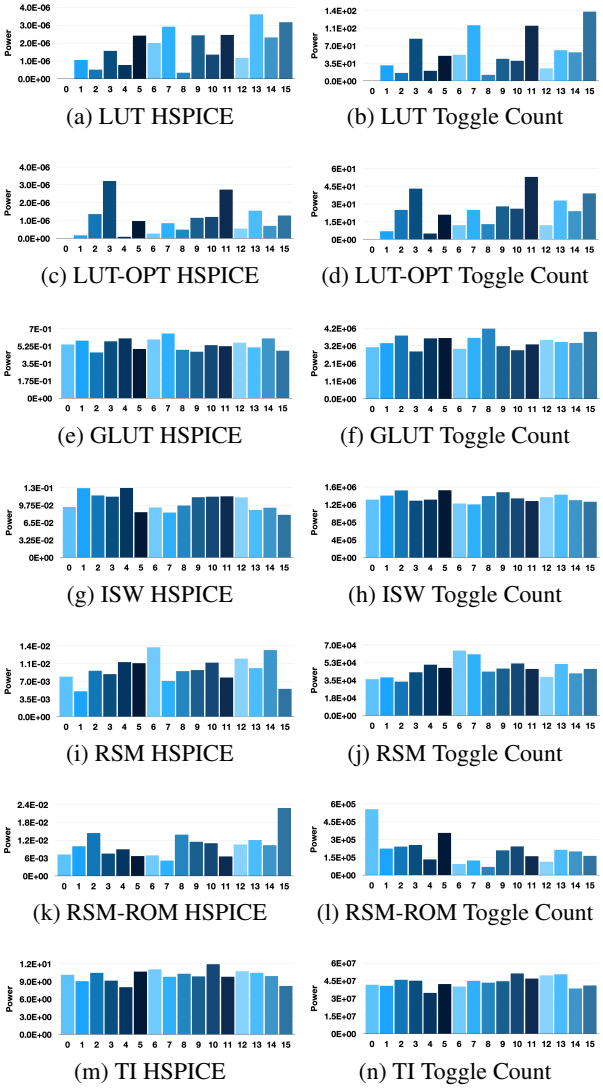


Figure 1: Intra-class powers of the targeted implementations.

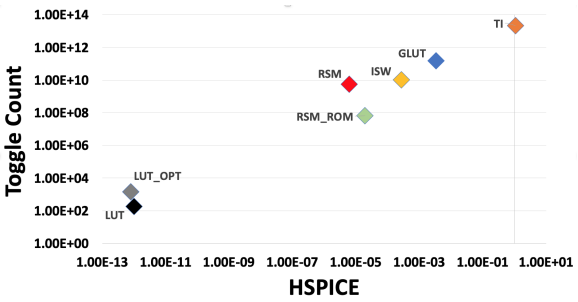


Figure 2: Toggle Count vs. HSpice variances between different input classes.

a sanity check on our simulated results. Namely, the next set of results depicted in Fig. 2 illustrates the variance of power values between 16 different classes

of all 7 implementations in a logarithmic scale for each of HSPICE and toggle count results. Indeed, the results shown in this figure relate to the variances of the bars shown in Fig. 1. The 7 points are almost aligned, which confirms that toggle count is valid estimation of the total HSPICE. This confirms that toggle count is meaningful to summarize global activity but it does not accurately capture data dependencies (as pointed in Table 2) in masked implementations.

Walsh-Hadamard Leakage Analysis using Toggle Count vs. SPICE Power: As described earlier, the Walsh-Hadamrd transform is used to extract all vulnerabilities as a result of bit interactions (cross-talk) inside implementations. Fig. 3 and Fig. 4 show the post Walsh-Hadamard leakage analysis when using HSPICE and toggle count results, respectively. Interestingly, it is known that TI implementation features glitches; nonetheless Fig. 4g shows that they **do not** leak information (i.e., all the spectrum have the same magnitude regardless of their polarity which itself does not convey any relevant information). This is expected since those glitches mix only at most $d = 3$ shares of each variable randomly split in $d + 1 = 4$ shares. As shown, the leakage sources are more distinguishable in HSPICE results depicted in Fig. 3 while the distinguishably is much less when using toggle counts. This confirms that it is not always practical to assess the implementations based on their toggle count data, and more precise simulations such as HSPICE are required.

Another interesting observation here is the leakage of bit 3 (shown in dotted cyan) in most of the implementations when using toggle counts as well as for unprotected implementations when using HSPICE. This is due to the fact that bit 3 in PRESENT S-box has a higher algebraic complexity than other bits.

6 Conclusion

Side-channel leakage detection can benefit from fast "toggle count" simulations when netlists are not implementing a masking countermeasure. Otherwise, we show on 5 different masked implementations that toggle count is consistently **not** related to the actual netlist leakage, as estimated by SPICE simulations. Therefore, leveraging "toggle count" to assess a netlist should be used with caution. Should any bias be found, then obviously the design features a side-channel vulnerability. However, when leakage analysis with "toggle count" features no bias, a refined analysis at SPICE level is required.

REFERENCES

Bahrami et al., J. (2022). Leakage Power Analysis in Different S-Box Masking Protection Schemes. In *DATE*.

Brier et al., É. (2004). Correlation power analysis with a leakage model. In *CHES*, pages 16–29.

Carlet, C. and Guilley, S. (2013). Side-channel indistinguishability. In *Hardware and Architectural Support for Security and Privacy*, pages 1–8.

Debande, N., Berthier, M., Bocktaels, Y., and Le, T.-H. (2012). Profiled model based power simulator for side channel evaluation. *Cryptology ePrint Archive*.

Fei et al., Y. (2015). A statistics-based success rate model for DPA and CPA. *J. Cryptographic Engineering*, 5(4):227–243.

Giaconia et al., M. (2007). Area and power efficient synthesis of DPA-resistant cryptographic S-boxes. In *Int'l Conf. on VLSI Design*, pages 731–737.

Guilley et al., S. (2017a). Codes for Side-Channel Attacks and Protections. In *Codes, Cryptology and Information Security (C2SI)*, pages 35–55.

Guilley et al., S. (2017b). Stochastic side-channel leakage analysis via orthonormal decomposition. In *Information Technology and Communications*, pages 12–27.

Hartog et al., J. d. (2003). Pinpas: a tool for power analysis of smartcards. In *IFIP International Information Security Conference*, pages 453–457. Springer.

Heuser et al., A. (2016). Side-Channel Analysis of Lightweight Ciphers: Does Lightweight Equal Easy? In *Radio Frequency Identification and IoT Security*, pages 91–104.

Ishai, Y., Sahai, A., and Wagner, D. (2003). Private circuits: Securing hardware against probing attacks. In *Annual Int'l Cryptology Conf.*, pages 463–481.

Kirschbaum, et al., M. (2007). *Evaluation of power estimation methods based on logic simulations*. Citeseer.

Li et al., H. (2005). Security Evaluation Against Electromagnetic Analysis at Design Time. In *CHES*, volume 3659, pages 280–292.

Liu et al., H. (2011). The Switching Glitch Power Leakage Model. *Journal of Software (JSW)*, 6(9):1787–1794. Academy Publisher.

Mangard et al., M. (2006). Pinpointing the Side-Channel Leakage of Masked AES Hardware Implementations. In *CHES*, volume 4249, pages 76–90.

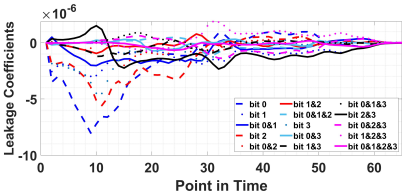
Mangard et al., S. (2005). Side-Channel Leakage of Masked CMOS Gates. In Menezes, A., editor, *CT-RSA*, volume 3376, pages 351–365.

Nassar et al., M. (2012). RSM: A small and fast countermeasure for AES, secure against 1st and 2nd-order zero-offset SCAs. In *DATE*, pages 1173–1178.

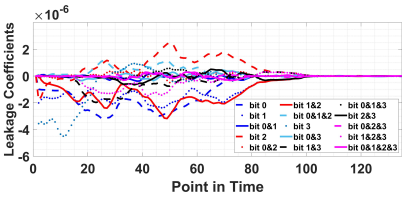
Peralta et al., R. (2022). Circuit Complexity Project. NIST, Computer Security Resource Center: https://csrc.nist.gov/Projects/circuit-complexity. Created December 29, 2016 – Updated February 08, 2022.

Roy et al, D. B. (2018). CC meets FIPS: A hybrid test methodology for first order side channel analysis. *IEEE Trans. on Computers*, 68(3):347–361.

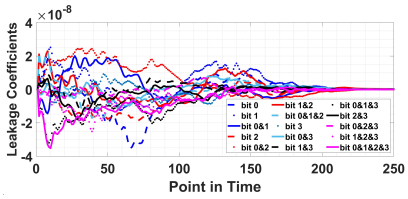
Veshchikov, N. (2014). Silk: high level of abstraction leakage simulator for side channel analysis. In *Program protection and reverse engineering workshop*, pages 1–11.



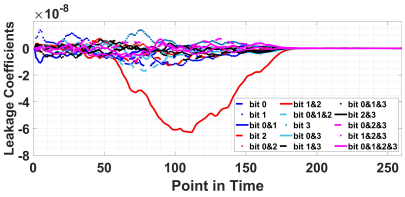
(a) LUT HSPICE



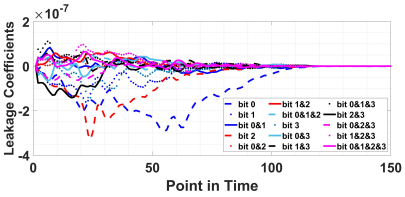
(b) LUT-OPT HSPICE



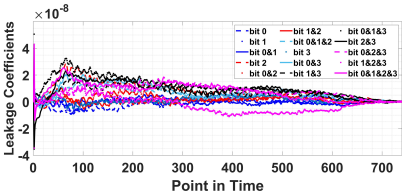
(c) GLUT HSPICE



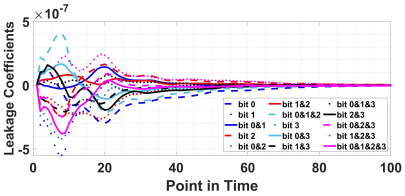
(d) ISW HSPICE



(e) RSM HSPICE

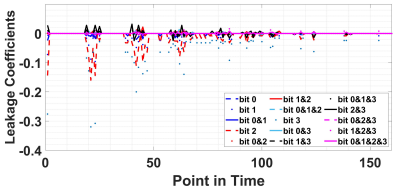


(f) RSM-ROM HSPICE

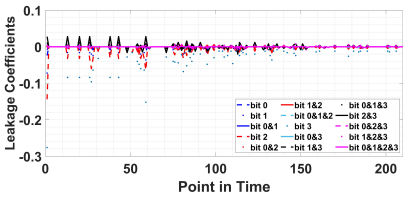


(g) TI HSPICE

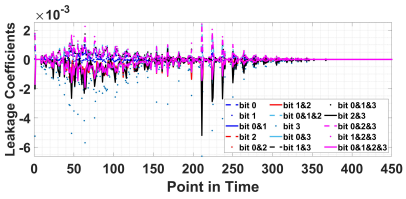
Figure 3: Walsh-Hadamard coefficients extracted using HSPICE.



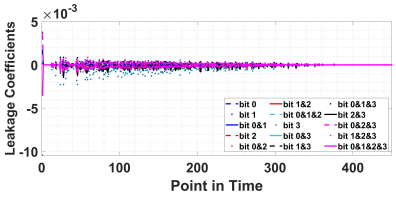
(a) LUT Toggle Count



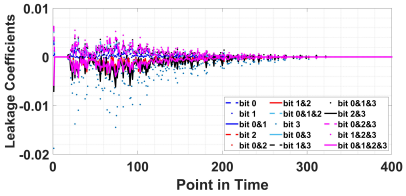
(b) LUT_OPT Toggle Count



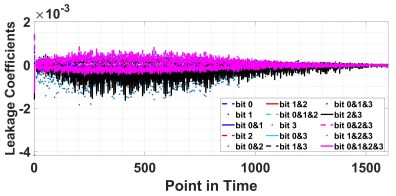
(c) GLUT Toggle Count



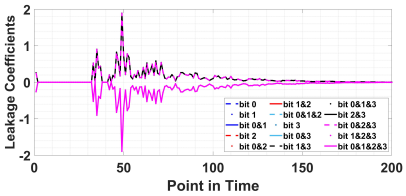
(d) ISW Toggle Count



(e) RSM Toggle Count



(f) RSM_ROM Toggle Count



(g) TI Toggle Count

Figure 4: Walsh-Hadamard coefficients extracted using Toggle Counts.