TOFU – Toggle Count Analysis made simple

Michael Gruber[1] and Georg Sigl[1,2]

[1]Technical University of Munich,
Chair for Security in Information Technology,
Munich, Germany,
{m.gruber, sigl}@tum.de
[2]Fraunhofer Institute for Applied and Integrated Security,
Garching, Germany,
georg.sigl@aisec.fraunhofer.de

Abstract

Protection against physical attacks is a major requirement for cryptographic implementations running on devices which are accessible to an attacker. Side-channel attacks are the most common types of physical attacks, the most frequent side-channel is the device's power consumption. In this work we propose a novel open-source tool called TOFU which synthesizes VCD simulation traces into power traces, with adjustable leakage models. Additionally, we propose a workflow which is only based on open-source tools. The functionality of TOFU and the proposed workflow was verified by a CPA of a AES hardware implementation. We also provide numbers for the required running time of TOFU for a trace synthesis with respect to the according VCD file size. Furthermore, we provide TOFU's source code.

1 Introduction

According to Kerckhoffs's principle a cryptographic algorithm's security should only rely on the key, everything else is expected to be public knowledge. While this holds in general for a cryptographic algorithm, physical attacks deliberately exploit the violation of Kerckhoffs's principle. In the context of physical attacks a possible extension of Kerckhoffs's principle may be: A cryptographic algorithm's power consumption should not depend on the processed data, and key. Which is essentially the precondition for Side-Channel Analysis (SCA), where data-dependent power consumption is deliberately exploited. The most prominent side channel used is the device's power consumption. One approach to exploit this data dependent power consumption is Differential Power Analysis (DPA) as introduced by Kocher et al. [9]. Leakage occurs due to the physical properties of hardware, i.e., power consumption of Complementary Metal-Oxide-Semiconductor (CMOS) circuits. To test if sensitive leakage occurs, one

can measure the device's power consumption during the execution of a cryptographic algorithm. However, this approach has the disadvantage that some hardware to run the algorithm is needed, as well as a device to record the power consumption, such as an oscilloscope. Furthermore, the measurements acquired by the oscilloscope are noisy by nature. An alternative to measure the power consumption on an actual device is a simulation, which result in absolutely noiseless measurements.

Related Work: Veshchikov et al. proposed a simulator called SAVRASCA for the AVR architecture [14] in 2017. Mc Cann et al. proposed another simulator called ELMO in 2016 [11]. ELMO was developed as a profiled simulator and is specifically targeted at the ARM Cortex-M0 architecture. Another approach was taken by Le Corre et al. [3] with their simulator MAPS which was tailored for the ARM Cortex-M3 architecture. MAPS was build based on the VHDL model of an ARM Cortex-M3 microcontroller where the pipeline was analyzed for instructions which exhibit data-dependent power consumption. In contrast, Sadhukhan et al. [12] focused on hardware implementations and proposed an SCA resistant design flow, they also proposed a toggle count based leakage model. Additionally, Wamser et al. were able to show a toggle count based Correlation Power Analysis (CPA) on the Advanced Encryption Standard (AES) [16] using the `VCD2R` package available for the R programming language [15]. Commercial solutions are available as well [4].

Contributions: In this work we propose TOggle Foul-Up (TOFU), a versatile open-source tool to synthesize Value Change Dump (VCD) [1] simulation traces into power traces. Furthermore, we propose a security evaluation workflow based only on open-source tools. We verify the capabilities of TOFU at the example of a CPA on AES.

Although TOFU shares some similarities with `VCD2R` the focus of TOFU is on performance and the evaluation of protected implementations.

Organization: The rest of the work is structured as follows: Section 2 introduces the necessary preliminaries. Section 3 introduces TOFU. Section 4 evaluates TOFU's performance. Section 5 introduces the proposed workflow, while Section 6 showcases the workflow. Finally, Section 7 concludes our work.

2 Preliminaries

This section introduces the necessary preliminaries, i.e., leakage models, CMOS power consumption, and CPA.

2.1 CMOS Power Consumption

According to [13] the power consumption of CMOS circuits can be divided into static power consumption P_{stat} and dynamic power consumption P_{dyn} as shown in Equation (1). Static leakage current occurs since N-Type Metal-Oxide-Semiconductor (NMOS) and P-Type Metal-Oxide-Semiconductor (PMOS) tran-

sistors are not perfectly switching. The static power consumption can be calculated as the product of the leakage current I_{CC} and the supply voltage V_{CC} and is therefore assumed to be independent of switching activities. Static power consumption can also cause side-channel leakage as shown by Khairallah et al. [8]. The dynamic power consumption P_{dyn} can be divided into transient power consumption P_{tran} and capacitive-load power consumption P_{cap_load} which is proportional to the output's capacitive load P_{cap_load}. Transient power consumption occurs when the input to the gate changes which is indicated by f_I. Switching the value of the gate output leads to the charging or discharging of the output capacitance which is indicated by f_O.

$$\begin{aligned} P_{stat} &= V_{CC} \times I_{CC} \\ P_{dyn} &= P_{tran} + P_{cap\_load} \\ P_{tran} &= C_{pd} \times {V_{CC}}^2 \times f_I \times N_{switches} \\ P_{cap\_load} &= C_L \times {V_{CC}}^2 \times f_O \times N_{switches} \end{aligned} \tag{1}$$

2.2 Leakage Models

A simplification of the power consumption calculation outlined in Section 2.1 are leakage models [2]. These leakage models directly translate processed data into a hypothetical power consumption, the most common ones are the Hamming Weight (HW), and the Hamming Distance (HD). The calculation of the HW, and the HD for processing the data sequence $\{D_1, D_2, ..., D_m\}$ with n-bit data words $D_i = d_1^{(i)} d_2^{(i)} ... d_n^{(i)}$ is shown in Eq. (2), where every $d_n^{(i)}$ represents a single bit.

$$\begin{aligned} HW(D_i) &= \sum_{j=1}^{n} d_j^{(i)} \\ HD(D_i, D_{i+1}) &= HW(D_i \oplus D_{i+1}) \end{aligned} \tag{2}$$

The most common leakage model used is the HW as it requires no knowledge about previously processed data in contrast to the HD cf. Eq. (2).

2.3 Correlation Power Analysis

DPA is a type of side-channel attack proposed in 1999 by Kocher et al. [9] and is based on partitioning. A similar attack called CPA which uses correlation as a distinguisher was proposed by Brier et al. in 2004 [2]. For a CPA attack, the power consumption of a cryptographic operation is measured multiple times, i.e., different plaintexts with the same key (for AES). Next, an intermediate value depending on known values, as well as an unknown key byte is chosen. A possible intermediate value t during the encryption of AES is the output of the first round's S-box, i.e., $t = S(k_0^{(0)} \oplus p_0)$. For all key hypotheses of the key

Key	Description
vcdGlob	Glob to find the VCD files.
signalsFileNameLiterals	Signals used for the leakage synthesis.
leakageModel	Leakage model used for the synthesis.
window	Use only samples from a window.
windowFrom	Specify window start.
windowTo	Specify window stop.
valueExtractFunction	Name of function which extracts values.
valueExtractIndex	Store the trace index as value.
writeTraces	Store traces in file or memory.
writeTracesBatchSize	How many traces to write at once.
traceFileName	Filename of the generated traces.
format	Format of the generated traces.

Tab. 1: TOFU Settings Summary

byte $k_0^{(0)}$, the intermediate value is calculated. By the application of a leakage model to the intermediate value, the correlation of all key hypotheses with the measured power traces can be calculated. The correct key byte is then identified by the hypothesis which results in the highest absolute correlation.

3 TOFU

As TOFU is intended to be a helpful utility for research, and teaching the source code can be found here[1].

We will now introduce the most important settings of TOFU used during trace synthesis. TOFU is implemented in Python, but parsing is done in C++ alternatively parsing can be done in Python as well, which is helpful for testing, e.g., new leakage models. There are various settings which can be passed to TOFU, the most important ones are shown in Table 1.

A regular expression describing the VCD files to parse is given by *vcdGlob*.

Supported leakage models are either HW, or HD and can be chosen by *leakageModel*.

It is possible to filter the signals used for the leakage generation by specifying them in *signalsFileNameLiterals*.

The value of a signal can be extracted for any timestamp by specifying a *valueExtractFunction*. This can for instance be used to extract the value of, e.g., plaintexts, keys, or ciphertexts. Furthermore, values for multiple signals can be extracted as a single value, which is for instance useful if the plaintext consists of multiple signals like in masking. Alternatively, *valueExtractIndex* can be used as value extraction function, which is useful in Test Vector Leakage Assesment (TVLA) [5] for differentiating between traces with fixed plaintext and random

[1] `https://gitlab.lrz.de/tueisec/tofu`

plaintext, based on the index of the current trace.

If *window* is set, the generated leakage is restricted to all timestamps between *windowFrom* and *windowTo* which also speeds up the synthesis.

Setting *writeTraces* to true indicates that the generated traces should be written to the file given by *traceFileName* with the batch size of *writeTracesBatchSize*.

As TOFU is tightly integrated with Ledger's Advanced Side Channel Analysis Repository (LASCAR) [10] the currently only supported *format* is LASCAR's default format.

4 Performance

In order to evaluate the performance of TOFU in terms of synthesis speed we evaluated the required time to parse VCD files of different sizes. Additionally, we used either the Python based parser, or the C++ version.

Using C++ instead of Python is motivated by higher performance, which mainly arises due to the C++ code being compiled and optimized. Moreover, C++ offers more control than Python, for instance through pointers and manual memory management.

Besides using a different programming language, the C++ implementation employs additional optimizations. Firstly, VCD files are memory-mapped, thus allowing for fast read operations of the VCD file. This is because no system calls are necessary, among others.

The time required by the C++ implementation to parse a VCD file only increases proportionally to the size of the VCD file, i.e., $\mathcal{O}(n)$. For instance, if parsing a 5 MB file takes 1 s, then parsing a 10 MB file should take about 2 s. This can only be achieved by constant lookup times of the values of symbols in the value change section of the VCD file. These lookups are necessary for calculating the leakage. For this reason, the C++ implementation uses a hash table (std::unordered_map), which gives average constant-time insertion and search.

Figure 1 compares the running time of the C++ implementation with the Python implementation for different VCD file sizes. For a better comparison, the time required for LASCAR to write the resulting traces is omitted. As one can see the required running time for the C++ implementation grows linearly with respect to the file size.

TOFU was successfully used during the development of DOMREP a combined countermeasure against Fault Injection Analysis (FIA), and SCA proposed by Gruber et al. [7] cf. the biggest file of Table 1.

5 Workflow

One of the main objectives of this work was to specify a workflow which is only based on open-source tools. The necessary steps of the workflow can be formulated as: *Simulation*, *Synthesis*, and *Analysis*.

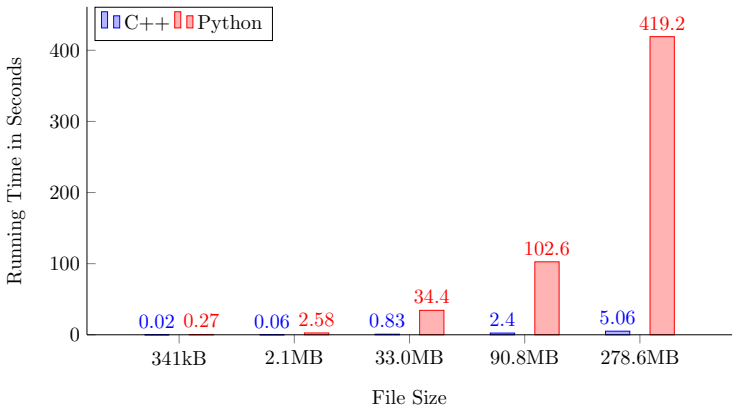


Fig. 1: Running Time Comparison

Simulation: The *Simulation* step (behavioral) is based on GHDL an open-source VHDL simulator developed by Gingold et al. [6]. GHDL compiles VHDL files directly to machine code and allows native execution to allow high speed simulations. We have also verified the *Simulation* step of the workflow with *Vivado 2020.2*, which enables also simulations which take hardware specific delays into account, e.g., post-implementation.

Synthesis: The *Synthesis* step is based on TOFU as introduced in this work. Prior to the *Synthesis* suitable settings must be chosen as specified in Section 3.

Analysis: The *Analysis* step is based on LASCAR an open-source framework developed by Ledger [10]. LASCAR is a versatile SCA framework which supports several SCA attacks, e.g., DPA, and CPA. Also, LASCAR provides a convenient container format to store, and access acquired traces.

6 Example

In the following section we will now outline the workflow introduced in Section 5 at the example of a CPA of AES[2] using 10 000 traces. The implementation applies all S-boxes simultaneously.

6.1 Simulation

The TOFU repository contains a testbench for the implementation of AES, which can serve as the basis for other VHDL projects. The integration of the AES's testbench and GHDL was done with parallelism in mind, to utilize several processor cores. Unfortunately, GHDL does not allow vectors as parameters [6]. Instead, it is possible to pass multiple integers as parameters and combine them

[2] The AES implementation can be found inside TOFU's repository.

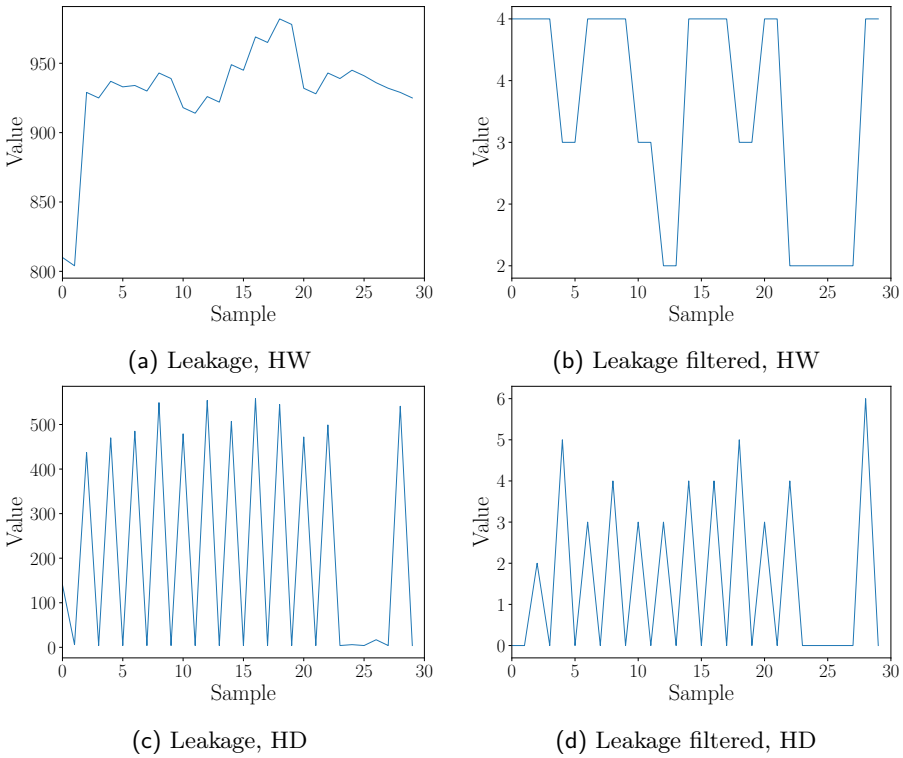
into a vector. Passing parameters to the GHDL simulation avoids repetitive recompiling of the VHDL source to native code. The generation of the random plaintexts required by the DPA is done in a pseudo-random manner where a random number generator is seeded deterministically.

6.2 Synthesis

In order to demonstrate a subset of TOFU's features we have done several syntheses. For the leakage model we either used the HW, or HD. Also, for the filtering, i.e., *signalsFileNameLiterals* we either used all signals available in the VCD file, or only the signals of the first S-box's output only. The corresponding VCD has a size of 26 kB. The synthesized trace for an exemplary AES encryption (unfiltered) is shown in Fig. 2a under the assumption of an HW leakage model, while Fig. 2c shows the same encryption under the assumption of an HD leakage model.

In contrast, if only the signals from the first S-box's output are used for the synthesis this is shown in Fig. 2b under the assumption of an HW leakage model, while Fig. 2d shows the same encryption under the assumption of an HD leakage model where the values take values between zero and eight as one may expect due to the S-box used in AES.

Noteworthy, in Fig. 2c, and Fig. 2d every second sample has a value of zero, as the circuit is only sensitive to the clock's rising edge. The circuits sensitivity to the clock's rising edges can also be used to fasten up the *Analysis* step if every second sample is discarded which results in a smaller power trace.



(a) Leakage, HW

(b) Leakage filtered, HW

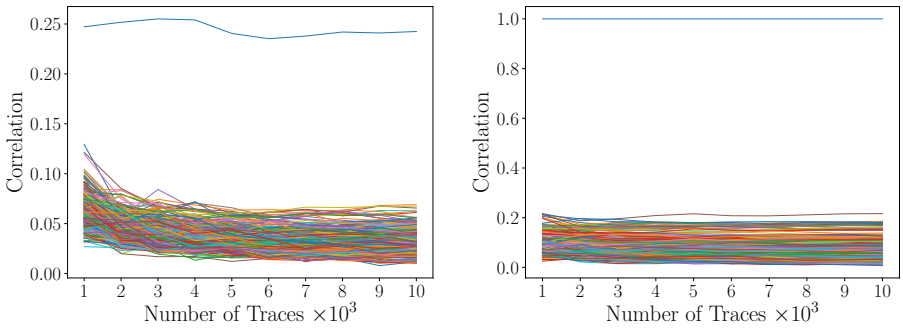
(c) Leakage, HD

(d) Leakage filtered, HD

Fig. 2: AES Workflow – Synthesis

6.3 Analysis

In the last step of the proposed workflow the *Analysis* takes place, i.e., in the context of a CPA on AES the first round key is attacked (encryption). As usual, the DPA assumes the HW as underlying leakage model, the whole CPA is conducted by LASCAR. For the analysis we consider two different cases, i.e., traces generated from either all signals (*unfiltered*) or only the first S-box's output (*filtered*). The *unfiltered* correlation progression plot for all key hypotheses is shown in Fig. 3a, as one can see the correlation of the correct key hypothesis converged to approximately 0.25 after 1000 traces. Respectively for the *filtered* case Fig. 3b shows that the correct key hypothesis converges to 1.0 instantly. None of the other 15 bytes can be recovered from such filtered traces correctly as the correlation of the remaining bytes does not converge towards a value in their respective correlation progression plots.



(a) Correlation, HW (b) Correlation filtered, HW

Fig. 3: AES Workflow – Analysis using 10 000 Traces

7 Conclusion

In this work we presented TOFU, a new robust tool for the synthesis of power traces from VCD files with different leakage models. In addition, we evaluated the running time of TOFU with respect to different VCD file sizes, and implementations of TOFU. Furthermore, we proposed a workflow which only relies on open-source tools to verify the security of an implementation. The open-source based workflow was verified at the example of AES using a CPA. We encourage the usage of TOFU as a helpful tool for research, and teaching.

Acknowledgements

This work was partly funded by the Bavarian Ministry of Economic Affairs, Regional Development and Energy in the project MITHRIL through grant number IUK-1903-0003 // IUK623/002. We would like to thank Florian Kasten for his help with the C++ implementation of TOFU's VCD parser.

References

[1] IEEE Standard for Verilog Hardware Description Language. *IEEE Std 1364-2005 (Revision of IEEE Std 1364-2001)*, pages 1–590, 2006.

[2] E. Brier, C. Clavier, and F. Olivier. Correlation Power Analysis with a Leakage Model. In M. Joye and J.-J. Quisquater, editors, *Cryptographic Hardware and Embedded Systems - CHES 2004*, pages 16–29, Berlin, Heidelberg, 2004. Springer Berlin Heidelberg.

[3] Y. L. Corre, J. Großschädl, and D. Dinu. Micro-architectural Power Simulator for Leakage Assessment of Cryptographic Software on ARM Cortex-M3 Processors. In *Constructive Side-Channel Analysis and Secure Design*, pages 82–98. Springer International Publishing, 2018.

[4] FortifyIQ. SideChannel STUDIO. https://www.fortifyiq.com/sidechannel-studio.html. Accessed: 2022-02-02.

[5] B. J. Gilbert Goodwill, J. Jaffe, P. Rohatgi, et al. A testing methodology for side-channel resistance validation. In *NIST non-invasive attack testing workshop*, volume 7, pages 115–136, 2011. https://csrc.nist.gov/csrc/media/events/non-invasive-attack-testing-workshop/documents/08_goodwill.pdf.

[6] T. Gingold. GHDL. https://github.com/ghdl/ghdl. Accessed: 2022-02-02.

[7] M. Gruber, M. Probst, P. Karl, T. Schamberger, L. Tebelmann, M. Tempelmeier, and G. Sigl. DOMREP – An Orthogonal Countermeasure for Arbitrary Order Side-Channel and Fault Attack Protection. *IEEE Transactions on Information Forensics and Security*, pages 1–1, 2021.

[8] M. Khairallah, A. Chattopadhyay, B. Mandal, and S. Maitra. On Hardware Implementation of Tang-Maitra Boolean Functions. Cryptology ePrint Archive, Report 2018/667, 2018. https://ia.cr/2018/667.

[9] P. Kocher, J. Jaffe, and B. Jun. Differential Power Analysis. In M. Wiener, editor, *Advances in Cryptology — CRYPTO' 99*, pages 388–397, Berlin, Heidelberg, 1999. Springer Berlin Heidelberg.

[10] Ledger. LASCAR. https://github.com/Ledger-Donjon/lascar. Accessed: 2022-02-02.

[11] D. McCann, E. Oswald, and C. Whitnall. Towards Practical Tools for Side Channel Aware Software Engineering: 'Grey Box' Modelling for Instruction Leakages. In *26th USENIX Security Symposium (USENIX Security 17)*, pages 199–216, Vancouver, BC, Aug. 2017. USENIX Association.

[12] R. Sadhukhan, P. Mathew, D. B. Roy, and D. Mukhopadhyay. Count Your Toggles: a New Leakage Model for Pre-Silicon Power Analysis of Crypto Designs. *Journal of Electronic Testing*, 35(5):605–619, oct 2019.

[13] Texas Instruments. CMOS Power Consumption and Cpd Calculation. *SCAA035B June*, 1997. https://www.ti.com/lit/an/scaa035b/scaa035b.pdf.

[14] N. Veshchikov and S. Guilley. Use of Simulators for Side-Channel Analysis. In *2017 IEEE European Symposium on Security and Privacy Workshops (EuroS&PW)*. IEEE, apr 2017.

[15] M. S. Wamser. VCD2R. https://github.com/wamserma/VCD2R. Accessed: 2022-02-13.

[16] M. S. Wamser. *Serialisation of Inversion-Based S-Boxes*. Dissertation, Technische Universität München, München, 2019.