Virtual Security Evaluation

An Operational Methodology for Side-Channel Leakage Detection at Source-Code Level

Youssef Souissi[1], Adrien Facon[1,2], and Sylvain Guilley[1,2,3](✉)

[1] Secure-IC S.A.S., 15 Rue Claude Chappe, Bât. B, 35 510 Cesson-Sévigné, France
sylvain.guilley@secure-ic.com

[2] École Normale Supérieure, Département d'informatique, 75 005 Paris, France

[3] LTCI, Télécom ParisTech, Institut Polytechnique de Paris, 75 013 Paris, France

Abstract. "An ounce of prevention is worth a pound of cure". This paper presents a methodology to detect side-channel leakage at source-code level. It leverages simple tests performed on noise-less traces of execution, and returns to the developer accurate information about the security issues. The feedback is in terms of location (where in code, when in time), in terms of security severity (amount and duration of leakage), and most importantly, in terms of possible reason for the leakage. After the source code (and subsequently the compiled code) has been sanitized, attack attempts complement the methodology to test the implementation against realistic exploitations. This last steps allows to validate whether the tolerated leakages during the sanitizing stage are indeed benign.

Keywords: Virtual evaluation methodology · Pre-silicon analysis · Source code vulnerability · Exploitability checking

1 Introduction

It is known since more than twenty years (recall the seminal paper of Kocher about timing attacks [12] in 1996) that some non-functional aspects of computation can be exploited to extract information from sensitive computations. Such key recovery attacks are referred to as side-channel analyses. They are very popular since they have allowed to break many real-world products in the past. Their threat was so scaring that side-channel analyses have been formalized in evaluation and test methodology, such as Common Criteria (ISO/IEC 15408), FIPS 140 and its international extension (ISO/IEC 19790), etc.

Today, side-channel analyses are well understood. It is clear how their success probability is related to the cipher architecture (e.g., through confusion coefficients [8]) and to the measurement conditions (e.g., the signal-to-noise ratio [16, Sect. 4.3.2, p. 73]). Countermeasures, such as hiding [16, Chap. 7], masking [16, Chap. 9], shuffling [23], resilience [13], etc. have been proposed and widely studied.

Alongside with side-channel analysis becoming more mature, the validation of protections on real (i.e., complex) systems has emerged as a new topic of interest [11]. The first step has been to set up some evaluation frameworks [19,20, 24]. They were complemented by tests to assess for leakage presence: this practice is called leakage detection. The leakage is characterized by differences [10], T-tests [9], or variance tests [2]. Such tests allow to detect vulnerabilities, but still, they apply only on the final product.

Checking whether attacks would be successful on source code is already a known technique [21]. In this paper, we highlight a novel methodology to detect leakages directly on the source code. The motivation is to allow for a fast feedback to the developer about vulnerabilities present in the source code. It is known that curing security issues in advance allows very quickly to converge to a side-channel-clean design, whereas chasing leakages late in the product development phase is costly and slow.

The rest of this paper consists in a explanation of the methodology (Sect. 2). The key messages are then gathered in a conclusion (Sect. 3), which also opens some perspectives for still better high-level detection techniques.

2 The Presented Methodology

We illustrate a situation where the design to be evaluated is a cryptographic application, which has the requirement to keep the secret key confidential. Moreover, we focus on a hardware implementation, since the security of hardware (in comparison with that of software) is the less easy to control. In particular, hardware consists in sequential ressources (which are clocked and are responsible to keep the state of the design) and in combinational ressources (which evaluate as soon as inputs change). The difference between the two ressources is illustrated in Fig. 1. The workflow to have the design be validated as correctly protecting the key is depicted in Fig. 2. It presents the methodology implemented in Secure-IC Virtualyzr$^{\mathrm{TM}}$ tool.

This figure shows that the path to obtain a secure design is iterative. First of all, the developer writes source code of its security application. Second, he provides a testbench. It can be written specifically for the application, or generated by the automated maintenance system. For instance, the testbench can be reused from a functional verification tool for security validation. The source code together with the testbench enable a simulator tool (e.g., Secure-IC Virtualyzr$^{\mathrm{TM}}$ supports Cadence `ncsim`$^{\mathrm{TM}}$, Synopsys `vcs`$^{\mathrm{TM}}$, Mentor Graphics `modelsim`$^{\mathrm{TM}}$ or `veloce`$^{\mathrm{TM}}$, etc.) to generate traces. As a next step, traces are analyzed with respect to leakage. For this purpose, the Virtualyzr shall be aware of the name of the assets to protect. Hence the indication of secrets as a user input in Fig. 2. Security analysis will reveal multiple information:

- for every bit in the system, at each clock cycle,
- how much and when is there a non-negligible dependency?

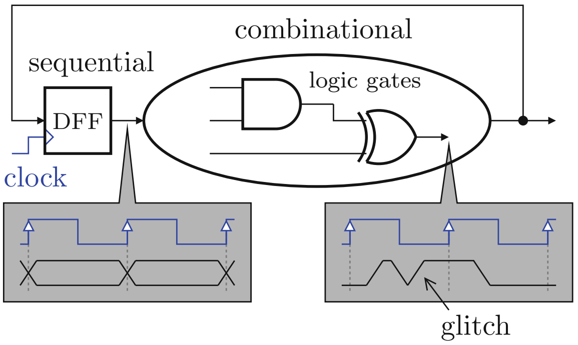

Fig. 1. Illustration of hardware as a Moore state machine, where sequential ressources sample at clock rising edge whereas combinational ressources evaluate each time an input changes, hence glitches at their output. Exemplar chronograms are displayed in the grey boxes

At bit level and without noise, all statistical tools to analyse dependency collapse to the same distinguisher, namely the Pearson correlation. Notice that this analysis is exhaustive. However, it is feasible since estimation of dependency can be achieved with little amount of traces. For example, for an accuracy of 1%, only $(1/0.01)^2 = 10,000$ traces are needed. The outcome of the analysis is a table containing those pieces of information:

- list of events (i.e., signal-time pairs),
- amount of correlation,
- duration of leakage. For sequential signals (registers), the duration is irrelevant, since those signals vary only exactly once per clock period. However, for combinational signals (output of Boolean or arithmetic gates), the duration is important when the code is not simulated at Register Transfer Level (RTL). Indeed, at RTL, combinational gates are evaluated without delay. But at later stages of refinement (RTL synthesized to logic gates), the combinational signals feature delays. The duration of correlation therefore indicates how transient the vulnerability is showing up. In particular, it is relevant to make a difference between *steady* versus *glitching* leakage [15]. A steady leakage basically means that some gates take on a value which can be exploited. This situation is more critical than glitch leakage where the leakage is transient: the dependency does not last until the end of the clock cycle, but shows up surreptitiously only at the favor of input signals uncoordinated arrival times.

In theory, all vulnerabilities should be fixed, which is the purpose of the feedback loop to the original design (cf. Fig. 2). The tool must be fast and automated to make this iterative sanitation process as smooth as possible. Now, in practice, some vulnerabilities might be identified as tolerable. This is the intent of the security policy. Examples are as follows:

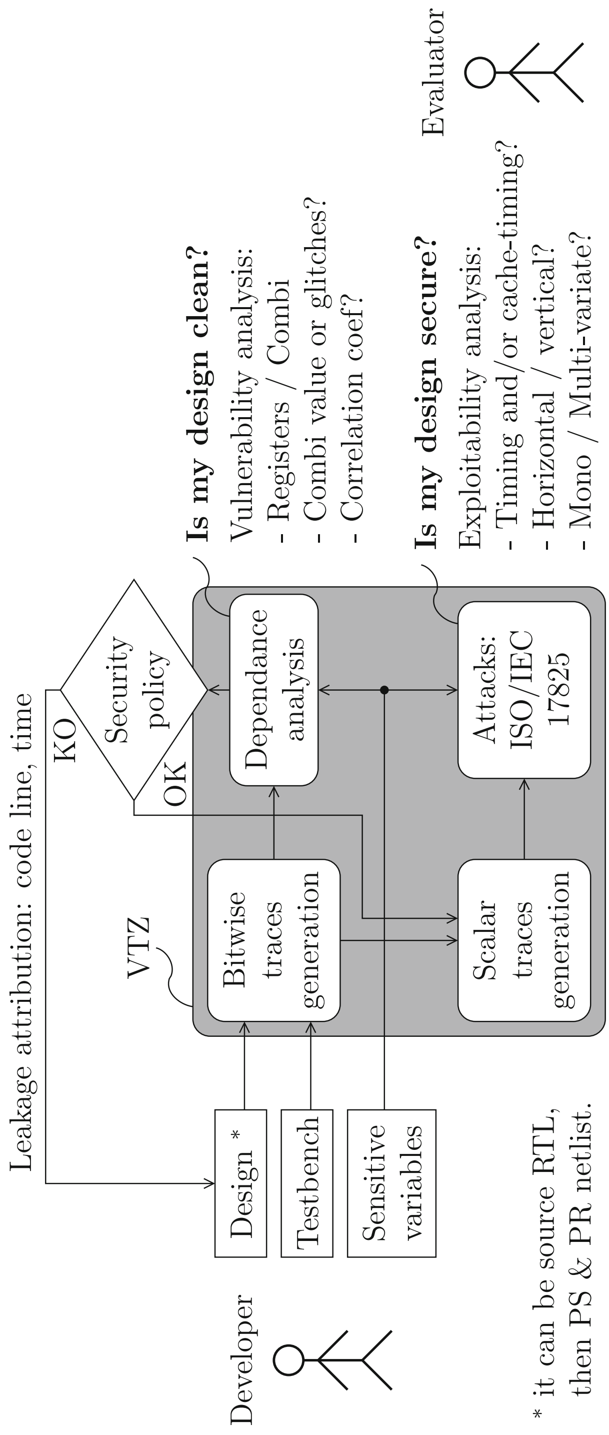

Fig. 2. Usage of Virtualyzr$^{\text{TM}}$ (VTZ$^{\text{TM}}$) tool to validate for errors at source-code level

– Key scheduling logic is certainly dependent on the key, however, exploitability requires a *template attack* [5], since the key cannot (in general) be manipulated by the user (i.e., the attacker);
– Leakage within cryptographic primitives might be computationally hard to exploit. Refer for example to the exploitation of HMAC: all executions of the hash primitive depend on the key, however only the head and tail ones are really affected in a realistic manner by the leakage of the HMAC key; exploiting others would require for the attacker the ability to inverse hash functions. If this was feasible, side-channel attacks would not be the easiest attack paths.

Apart from these exceptions, all vulnerabilities on RTL source are expected to be fixed, since they make up structural weaknesses. After this step, the design is refined into a netlist, either post-synthesis (delays are added in the gates) or post place-and-route (delays are added in the interconnection between the gates as well). Therefore, more avenues for leakage occur: indeed, the combinational gates evaluate as soon as one input changes (contrary to sequential gates). In addition, after synthesis, some restructurations or even optimizations are carried out. They include:

– gate factorization,
– gates reordering,
– simplification.

Due to these modifications, the architectural structure and the signal names are altered. Thus one aspect is to keep track of RTL resource names with netlist resource names. The vulnerability analysis now might evolve. The differences can be classified in two categories:

1. regressions (such as countermeasure alteration),
2. refinement (such as providing more details for a primitive, which induces more complex timing behavior).

Regressions must be addressed, because they highlight problems arising from the use of EDA tools. For instance, the first-order masked AND gate (computing $c = a \wedge b$, where each Boolean variable x is randomly split in two, as per perfect masking [3], that is $x = x' \oplus x''$) represented in Fig. 3(a)) will leak with:

– correlation 1 (because the sensitive signal c appears in the clear) with simplification of Fig. 3(b), and
– correlation $1/2$ (because $a' \wedge b' \oplus a' \wedge b'' = a' \wedge b$, which is equal to b if $a' = 0$ and to 0 otherwise, hence a match with clear value b half of the time—a' is a random mask, i.e., $\mathbb{P}(a' = 0) = \mathbb{P}(a' = 1) = 1/2$) with simplification of Fig. 3(c).

Both netlists (b) and (c) are functionally correct. They are even improved in terms of critical path (for equal gate-count), as compared with (a). Nonetheless, their rearrangement induced a side-channel leakage. Interestingly, the amount of leakage (e.g., correlation 1 versus 1/2) reveals a precious hint to the designer. It helps him understand if the problem is structural (case of correlation = 1) or due to some unbalance in the masking scheme (case of correlation = 1/2, refer for instance to [18]).

The correction can be some actions either on the compilation side (typically by placing *constraints* in scripts), or on the design-side (typically by adopting a *robust coding style*, which resists optimizations).

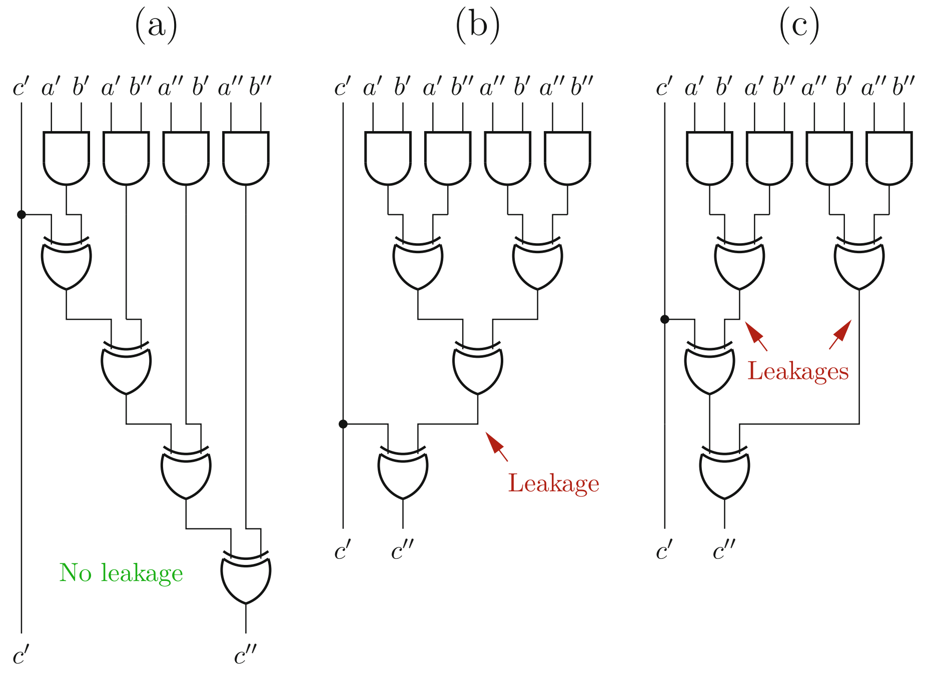

Fig. 3. Example of simplifications on masked $c = a \wedge b$ which induce side-channel leakage of sensitive information. In this figure $x \in \{a, b, c\}$ is randomly split as (x', x'') such that $x = x' \oplus x''$.

Refinements might highlight transient leakages due to glitches [17]. Here, the security policy decides whether those glitches are tolerable or not. Indeed, it might be time-consuming to remove all glitches which carry (thence leak) sensitive information. Typically, if the glitch lasts 10 ps and that the attacker makes use of a sampling device which allows only to measure at a bandwidth of 1 GHz, then the glitch will be smoothed by the measurement system and its energy will be spread over 100 time samples, thereby making its exploitability 100 times more difficult. In these conditions, a decision to live with such glitch can be made. An exemplar security policy is exposed in Table 1. It makes a difference between the type of incriminated resource, either seq(uential) or combi(national).

As mentioned, leakage of sequential ressources is to be considered more seriously than combinational ressources, because the leakage is synchronized, hence the signal-to-noise ratio measured by an attacker is high. Also, sequential elements (typically flip-flops or memories) consume/radiate much more than combinational gates, and their leakage model is simple (Hamming weight and/or Hamming distance [4, Sect. 2]).

Table 1. Exemplar security policy

Type	Duration	Correlation	Origin	Policy
Seq.	One clock period	=1	Structural	Fatal design bug or simplification upon synthesis optimization
Seq.	One clock period	<1	Unbalanced mask, causing first-order leakage	Critical countermeasure issue
Combi.	Steady	=1	Structural	Critical design bug
Combi.	Steady	<1	Unbalanced mask, causing first-order leakage	Serious countermeasure issue
Combi.	Transient	<1	Timing race	Can be tolerated—typically requires 100,000+ traces

Finally, attacks are tested on aggregated traces. The methodology applied to virtual traces is that of ISO/IEC 17825 [6]. It is a classical approach, where:

- *timing* and *cache-timing* analyses are experienced first,
- second *single trace* leakage is assessed, and
- finally, *vertical attacks* are tested.

We refer the reader to papers such as [22] for in-depth discussions about the use of simulation for exploitability analysis.

3 Conclusion and Perspectives

We have presented a fast methodology based on simulation to identify precisely and with little number of test vectors the presence of leakage. We show that some basic characterizations allow to narrow down the root cause of the leakage, which allows to accompany the developer in researching for leakage reduction or cancelling techniques (based on coding style or compilation options). This approach is innovative and can be implemented in a tool (we illustrate the case of Secure-IC Virtualyzr).

As a perspective, we notice that formal methods could complement the presented methodology. Formal methods ensure 100% coverage and are typically faster to yield their conclusions than dynamic methods. It is also to be noticed that formal methods (as of today, such as [7]) cannot discriminate between serious and tiny leakages. Moreover, formal methods are more comfortable at high-level since they need to approximate the design; therefore, they do not allow a traceability after compilation stages. Still, the synergy between static (i.e., formal) and dynamic leakage evaluation is foreseen as a winning combination for side-channel eradication from secure designs.

Eventually, we underline that the methodology presented in this paper could apply as well to software evaluation, where the most threaten leakage arises from cache-timing problems. A parallel between hardware and software is sketched in Table 2. Recall that:

– Hardware of secret key applications is computing in constant time, but features exploitable *vertical* leakage [14];
– Software is mostly vulnerable regarding *horizontal* leakage. Indeed, software development practices today is at a less advanced stage, security-wise, than that of dedicated cryptographic hardware.

Table 2. Comparison between security issues in hardware vs software

Programme nature	Cause of information leakage		
	Structural leakage	Optimization upon code generation	Dynamic leakage
Hardware	Unmasked or unbalanced signals	Simplification (recall Fig. 3)	Glitches
Software	Conditional control flow and/or table lookups	Seldom, since unpredictability hurts performances	Cache hit/miss, speculation, out-of-order execution

References

1. 3rd IEEE International Verification and Security Workshop, IVSW 2018, Costa Brava, Spain, July 2–4, 2018. IEEE (2018)
2. Bhasin, S., Danger, J.L., Guilley, S., Najm, Z.: NICV: normalized inter-class variance for detection of side-channel leakage. In: IEEE International Symposium on Electromagnetic Compatibility (EMC 2014/Tokyo), May 12–16 2014. Session OS09: EM Information Leakage. Hitotsubashi Hall (National Center of Sciences), Chiyoda, Tokyo, Japan (2014)
3. Blömer, J., Guajardo, J., Krummel, V.: Provably secure masking of AES. In: Handschuh, H., Hasan, M.A. (eds.) SAC 2004. LNCS, vol. 3357, pp. 69–83. Springer, Heidelberg (2004). https://doi.org/10.1007/978-3-540-30564-4_5
4. Brier, E., Clavier, C., Olivier, F.: Correlation power analysis with a leakage model. In: Joye, M., Quisquater, J.-J. (eds.) CHES 2004. LNCS, vol. 3156, pp. 16–29. Springer, Heidelberg (2004). https://doi.org/10.1007/978-3-540-28632-5_2
5. Chari, S., Rao, J.R., Rohatgi, P.: Template attacks. In: Kaliski, B.S., Koç, K., Paar, C. (eds.) CHES 2002. LNCS, vol. 2523, pp. 13–28. Springer, Heidelberg (2003). https://doi.org/10.1007/3-540-36400-5_3

6. Easter, R.J., Quemard, J.-P., Kondo, J.: Text for ISO/IEC 1st CD 17825 - Information technology - Security techniques - Non-invasive attack mitigation test metrics for cryptographic modules, March 22 2014. Prepared within ISO/IEC JTC 1/SC 27/WG 3 (2014)
7. Facon, A., Guilley, S., Lec'hvien, M., Schaub, A., Souissi, Y.: Detecting cache-timing vulnerabilities in post-quantum cryptography algorithms. In: 3rd IEEE International Verification and Security Workshop, IVSW 2018, Costa Brava, Spain, July 2–4, 2018 [1], pp. 7–12 (2018)
8. Fei, Y., Luo, Q., Ding, A.A.: A statistical model for DPA with novel algorithmic confusion analysis. In: Prouff, E., Schaumont, P. (eds.) CHES 2012. LNCS, vol. 7428, pp. 233–250. Springer, Heidelberg (2012). https://doi.org/10.1007/978-3-642-33027-8_14
9. Gilbert Goodwill, B.J., Jaffe, J., Rohatgi, P.: A testing methodology for side-channel resistance validation, September 2011. In: NIST Non-Invasive Attack Testing Workshop (2011). http://csrc.nist.gov/news_events/non-invasive-attack-testing-workshop/papers/08_Goodwill.pdf
10. Jaffe, J., Rohatgi, P., Witteman, M.F.: Efficient side-channel testing for public key algorithms: RSA case study, September 2011. In: NIST Non-Invasive Attack Testing Workshop (2011). http://csrc.nist.gov/news_events/non-invasive-attack-testing-workshop/papers/09_Jaffe.pdf
11. Kocher, P.: Complexity and the challenges of securing SoCs. In: Stok, L., Dutt, N.D., Hassoun, S. (eds) Proceedings of the 48th Design Automation Conference, DAC 2011, San Diego, California, USA, June 5–10, 2011, pp. 328–331. ACM (2011)
12. Kocher, P.C.: Timing attacks on implementations of Diffie-Hellman, RSA, DSS, and other systems. In: Koblitz, N. (ed.) CRYPTO 1996. LNCS, vol. 1109, pp. 104–113. Springer, Heidelberg (1996). https://doi.org/10.1007/3-540-68697-5_9
13. Kocher, P.C.: Leak-resistant cryptographic indexed key update, March 25 2003. United States Patent 6,539,092 filed on July 2nd, 1999 at San Francisco, CA, USA (2003)
14. Kocher, P., Jaffe, J., Jun, B.: Differential power analysis. In: Wiener, M. (ed.) CRYPTO 1999. LNCS, vol. 1666, pp. 388–397. Springer, Heidelberg (1999). https://doi.org/10.1007/3-540-48405-1_25
15. Liu, H., Qian, G., Tsunoo, Y., Goto, S.: The switching glitch power leakage model. JSW **6**(9), 1787–1794 (2011)
16. Mangard, S., Oswald, E., Popp, T.: Power Analysis Attacks: Revealing the Secrets of Smart Cards. Springer, New York (2006). ISBN 0-387-30857-1. http://www.dpabook.org/
17. Mangard, S., Schramm, K.: Pinpointing the side-channel leakage of masked AES Hardware implementations. In: Goubin, L., Matsui, M. (eds.) CHES 2006. LNCS, vol. 4249, pp. 76–90. Springer, Heidelberg (2006). https://doi.org/10.1007/11894063_7
18. Moradi, A., Guilley, S., Heuser, A.: Detecting hidden leakages. In: Boureanu, I., Owesarski, P., Vaudenay, S. (eds.) ACNS 2014. LNCS, vol. 8479, pp. 324–342. Springer, Cham (2014). https://doi.org/10.1007/978-3-319-07536-5_20
19. Souissi, Y., Danger, J.-L., Guilley, S., Bhasin, S., Nassar, M.: Common framework to evaluate modern embedded systems against side-channel attacks. In: IEEE International Conference on Technologies for Homeland Security (HST), pp. 86–91, November 15–17 2011. Westin Hotel, Waltham, MA, USA (2011). https://doi.org/10.1109/THS.2011.6107852

20. Standaert, F.-X., Malkin, T.G., Yung, M.: A unified framework for the analysis of side-channel key recovery attacks. In: Joux, A. (ed.) EUROCRYPT 2009. LNCS, vol. 5479, pp. 443–461. Springer, Heidelberg (2009). https://doi.org/10.1007/978-3-642-01001-9_26
21. Takarabt, S., et al.: Pre-silicon embedded system evaluation as new EDA tool for security verification. In: 3rd IEEE International Verification and Security Workshop, IVSW 2018, Costa Brava, Spain, July 2–4, 2018 [1], pp. 74–79 (2018)
22. Veshchikov, N., Guilley, S.: Use of simulators for side-channel analysis. In: 2017 IEEE European Symposium on Security and Privacy, EuroS&P 2017, Paris, France, April 26–28, 2017, pp. 51–59. IEEE (2017)
23. Veyrat-Charvillon, N., Medwed, M., Kerckhof, S., Standaert, F.-X.: Shuffling against side-channel attacks: a comprehensive study with cautionary note. In: Wang, X., Sako, K. (eds.) ASIACRYPT 2012. LNCS, vol. 7658, pp. 740–757. Springer, Heidelberg (2012). https://doi.org/10.1007/978-3-642-34961-4_44
24. Whitnall, C., Oswald, E.: A fair evaluation framework for comparing side-channel distinguishers. J. Crypt. Eng. **1**(2), 145–160 (2011)