Analyzing Security Breaches of Countermeasures Throughout the Refinement Process in Hardware Design Flow

Jean-Luc Danger, Sylvain Guilley, Philippe Nguyen, Robert Nguyen and Youssef Souissi
Secure-IC S.A.S., 15 Rue Claude Chappe, Bât. B ZAC des Champs Blancs, 35510 Cesson-Sévigné, FRANCE

***Abstract*—Side-channel and fault injection attacks are two threats on devices carrying sensitive information. Protections are thus implemented at design time. However, CAD (Computer Aided Design) tools can compromise them, in ways we detail pedagogically in this paper. Then, we explain how a simulation-based methodology allows to check for non-regression, and find problems in case some are introduced while refining the design description from RTL (Register Transfer Level) source code to GDS (Graphic Display System) stream format.**

***Index Terms*—Secure hardware, countermeasures, CAD flow, optimizations, security properties setting, security verification tool.**

1. Introduction

Security is very important in information processing. Two important features are that the information does not leak, and that it is not corrupted. However, side-channel attacks on the one hand, and fault injection attacks on the other, allow that an attacker recovers or modifies the system. Protections are thus implemented, very often at source-code level. However, it is noteworthy that most countermeasures add extra logic on top of the functional design.

There is a risk that those countermeasures be altered during the design flow. Indeed, the countermeasures expand the logic for reasons not grasped by CAD tools. Thence, CAD tools might optimize them away, or change them in such a way they become less efficient.

Contribution.

It is the topic of this paper to analyze possible disruptions carried out by CAD tools, as well as remedies and the way to verify them.

Outline.

The rest of the paper is organized as follows. The steps involved in the refinement of a circuit are described in Sec. 2. Some classical countermeasures are presented in Sec. 3. In Sec. 4, we illustrate some possible optimizations which lead to security issues. The prevention of security issues, and most importantly, their test, is the topic of Sec. 5. Finally, Sec. 6 concludes the paper.

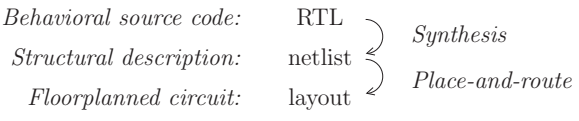

Figure 1. Design flow steps

2. Design Flow

In this paper, we consider that the design entry is the RTL code. At this level, the design can be simulated. Next, the user provides a target library, in which the design is logically mapped. The process is referred to as a logical synthesis, and yields a netlist.

After that, the netlist is laid out on the floorplan. This step is called place-and-route (PR). The result is a PR netlist, or layout, shipped to the foundry fabrication plant under the form of a GDS format file.

The process is described in Fig. 1. In this paper, we are concerned about the nature of the transformations occurring during *synthesis* and *PR* steps.

3. Countermeasures against side-channel and fault injection attacks

3.1. Countermeasures against side-channel attacks

In the perspective of side-channel analysis, any intermediate data used in a cryptographic algorithm is a potential source of leakage. It is thus important to protect any so-called *sensitive variable*. RTL solutions to withstand side-channel attacks follow two ideas:

1) randomize the data being manipulated, or
2) equalize the activity of the sensitive variables.

Both approaches are complementary, as they both contribute to reduce the signal-to-noise ratio (SNR) of the leakage. Indeed, randomization increases the noise, whereas activity balancing reduced the signal.

Amongst randomizing countermeasure, typical protections consist in:

- Replacing any sensitive variable x by the pair $(x \oplus m, m)$, where m is a random variable (which denotes a mask). Alternatively, x can be protected

978-3-9815370-8-6/17/$31.00 ©2017 IEEE

against high-orders attacks by adding more masks. For instance, where x is represented by the triple $(x \oplus m_1 \oplus m_2, m_1, m_2)$, that we talk about second-order perfect masking [1]. These high-order masking schemes are known to be transparent to linear operations, namely, if a is shared as $(a \oplus m_1, m_1, m_2)$ and b is shared as $(b \oplus m'_1, m'_1, m'_2)$, then $((a \oplus b) \oplus (m_1 \oplus m'_1), (m_1 \oplus m'_1), (m_2 \oplus m'_2))$ is a valid sharing of $(a \oplus b)$. Moreover, high-order masking schemes have been shown to be capable to pass through non-linear operations, such as multiplication [2]. This scheme, transparent to masking, is also known as ISW (authors initials of just mentioned paper). Shared output (c_1, c_2, c_3) of $c = a \wedge b$ can be calculated as follows:

$$c_1 = a_1 b_1 \oplus r_{1,2} \oplus r_{1,3} \quad (1)$$

$$c_2 = a_2 b_2 \oplus (r_{1,2} \oplus a_1 b_2) \oplus a_2 b_1 \oplus r_{2,3} \quad (2)$$

$$c_3 = a_3 b_3 \oplus (r_{1,3} \oplus a_1 b_3) \oplus a_3 b_1 \oplus (r_{2,3} \oplus a_2 b_3) \oplus a_3 b_2, \quad (3)$$

where $r_{i,j}$ are random numbers[1]. Since it is known that *glitches* (temporary signals occurring due to races in hardware circuits) can leak information [3], some masking schemes which intrinsically tolerate glitches have been invented. Typically, the TI (Threshold Implementation [4]) aims to reach this objective. A TI implementation ensures that a computation is split in a way not all shares are combined together into a single combinational function. For instance, for first-order resistance in the presence of glitches, it is shown that a spit in three shares is required[2].

- Balancing the leakage, e.g., using dual-rail with precharge logic (DPL). The idea is to trade any sensitive variable a by the dual-rail pair $(a, \neg a)$, which is also known as *Manchester encoding*. Such logic exists in many flavors, such as WDDL [6], or improved delay-insensitive balanced logics [7].

3.2. Countermeasures against fault injection attacks

Protection against fault injection attacks are based on redundancy checking. At the applicative layer, one test can be perceived as troublesome, since it is likely that the attacker aims at bypassing it. Therefore, the natural protection naturally consists in redundending it, i.e., replicate it as many times as possible, so as to make fault injection attacks unlikely to succeed. Such strategy can be thought of:

- *sequentially*, through multiple and diversified tests,
- *in parallel*, using DPL. Indeed, as discussed in [8], any inconsistency in the DPL protocole signals (e.g., the presence of $(1,1)$ when either $(0,1)$ or $(1,0)$ is expected) that a fault occurred.

In some contexts, it can be seen as a vulnerability to disclose that a fault is detected (it is typically exploited by *safe errors*). Indeed, it allows for the attacker to fine-tune its fault injection parameters. Therefore, in this case, *infective* countermeasures can be used. When a fault is successfully injected, the hardware detects it in real-time, and then randomizes the ciphertext[3]. As the output of block ciphers are indistinguishable from random numbers, the attacker never knows whether or not an effective fault has been injected, and cannot exploit the result since it is random.

3.3. Discussion about the countermeasures

As already mentioned, protections against side-channel leakage consist in decreasing the SNR an attacker gets. Typically, *masking countermeasures* consist in increasing the noise by mixing the signal with decorrelated random numbers, referred to as masks. Alternatively, *balancing countermeasures* try to lower the signal while keeping the noise constant. In both cases, the SNR in indeed lowered. However, the SNR is not a functionality of the circuit.

Protections against fault injection attacks consist in detecting some alteration in the circuit state. The detection can be high-order, meaning that several independent errors are detected. However, without error, it is clear that detection logic consists in dead code, which thus appears superfluous.

To summarize, the CAD tools are neither aware of side-channel leakage nor of faults injection, thus they are likely to transform the circuit description without preserving its security properties.

Some examples of possible breaks are detailed in the next section.

4. Break of countermeasures during refinement process

4.1. Breaches during synthesis

The protection can be removed altogether during logical synthesis, thereby causing a security regression. Examples are:

- Random register precharge; see example in Fig. 2. The randomization logic (a) allows to decorrelate the flip-flop contents from x_t (thanks to the mask m). However, the logic driven by m is non-functional: whatever the value m, the output x_{t+1} does only depend on the previous value x_t. There is thus a risk that the logic driven by m is altogether

1. The random numbers $r_{1,2}$, $r_{1,3}$ and $r_{2,3}$ do not alter the functionality condition: $\left(\bigoplus_{i=0}^{i=2} c_i\right) = \left(\bigoplus_{i=0}^{i=2} a_i\right) \wedge \left(\bigoplus_{i=0}^{i=2} b_i\right)$.

2. A larger splitting would even be necessary if one takes into account the intrinsic asymmetry of the XOR gates layout [5].

3. In practice, such protection can be obtained by duplicating the state, comparing both versions bitwise (with operator "$\oplus$"), thereafter randomizing the difference by a 128-bit multiplication (in Galois finite field $\mathbb{F}_{2^8}$, with operator "$\otimes$") by a random number, which is equal to zero in the absence of faults. This is further illustrated in Fig. 5.

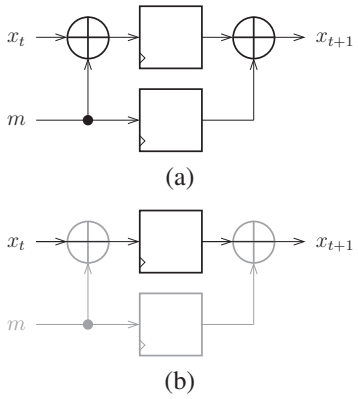

Figure 2. Random register recharge protection (a), removed (parts in gray) by the optimizing tool upon synthesis (b).

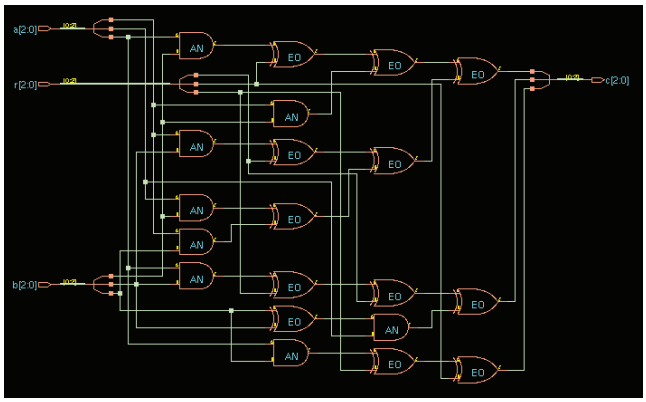

Figure 3. Optimized ISW gate AND with Optimization from Cadence RC compiler.

simplified (b), unless care is taken (e.g., with a `set_dont_touch` kind of directive).

- In ISW masking scheme, the order of the gates must be prevailed. One can see from Eqn. (1), (2) & (3) that in the 3 share private circuit implementation of $(a_i)_{0\leq i\leq 2}$ and $(b_i)_{0\leq i\leq 2}$, the bitwise products $a_i \wedge b_j$ must be computed first. However, a logic synthesis under Cadence yields the netlist shown in Fig. 3. Clearly, there are only 8 AND gates (instances called `AN` in Fig. 3), therefore an optimization has replaced one by reordering gates, thereby disrupting the evaluation order. Indeed, logic synthesizers are unaware of evaluation order in combinational circuits, hence assume gates reordering as a legitimate netlist transformation.
- Detective and infective [9] (*unmasked case* is depicted) countermeasures also add logic, which can be removed by the CAD tool without changing the functionality (in the absence of faults). It is illustrated respectively in Fig. 4 and Fig. 5: in (a), the countermeasure is inserted at RTL, and in (b), it is suppressed by the logic synthesizer.
- Another example is Common Subexpression Elimination (CSE). The multiply redundant code of Listing 1 (aiming at checking securely, that is *several* times, whether `M` is equal to `M2`, or not) is replaced, after synthesis, by the code of Listing 2 with a single redundancy, thereby introducing a SPOF (Single Point of Failure) in the code. In hardware, this would consist in FSM (Finite State Machine) simplification.

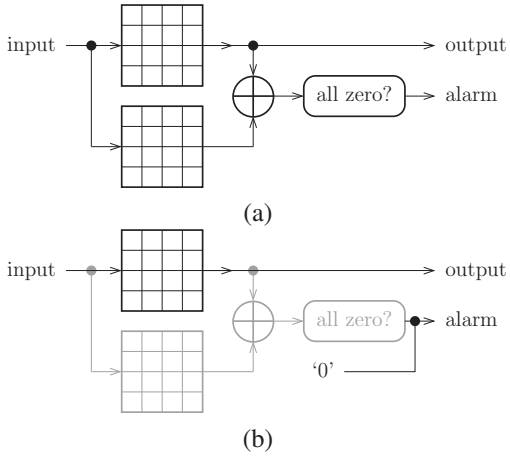

Figure 4. Detective countermeasure on AES (a), where the redundancy and the detective logic is removed (parts in gray) by the optimizing tool upon synthesis (b).

```
{
        alarm();
}
else
{
        counter += 0xfedcba98;
}
/* ... */
```

Listing 1. Original code for high-order fault detection

```
if( ( M == M2 ) && counter == 0xac70d145)
{
        counter ^= 0xffffffff;
}
else
{
        alarm();
}
if( ( M != M2 ) || counter != 0x538f2eba)
```

```
int test= ( M == M2 ); /* One Boolean */
/* The variable "test" becomes a SPOF */
if( test && counter == 0xac70d145)
{
        counter ^= 0xffffffff;
}
else
{
        alarm();
```

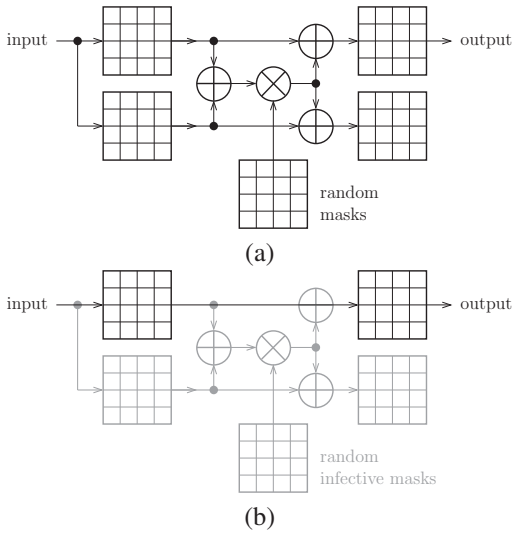

Figure 5. Infective countermeasure on AES (a), where the redundancy and the infective logic is removed (parts in gray) by the optimizing tool upon synthesis (b).

```
}
if( !test || counter != 0x538f2eba)
{
        alarm();
}
else
{
        counter += 0xfedcba98;
}
/* ... */
```

Listing 2. Optimized code of Listing 1 (using CSE)

4.2. Breaches during floorplanning

4.2.1. Late arrival at gates causes glitches. Let us consider the design of Fig. 6. It carefully protects the key k by mask m before manipulating the k. Therefore, the key is not leaking.

Late arrival at gates causes glitches. For example, Fig. 7 shows such a situation where it happens. This figure illustrates a masking scheme, where the key k is masked prior to being XOR-wise added to the plaintext p (like in case of Fig. 6). However, the mask m is computed as a function of the state of some PRNG (Pseudo Random Number Generators, such as a Linear Feedback Shift Register). Therefore, it is available a bit later (within the clock period) than p and k: indeed, p and k are readily available, as outputs of registers, at each clock rising edge. On the contrary, m is computed using combinational logics, hence takes on its value after some delay (for instance, 3 ns in Fig. 7). During this delay, the unmasked value $p \oplus k$ is held on a wire (namely equipotential d), hence is leaking. Notice that this kind of leakage is indeed of *glitch* type, as $k \oplus p$ is only transiently present unmasked on equipotential d.

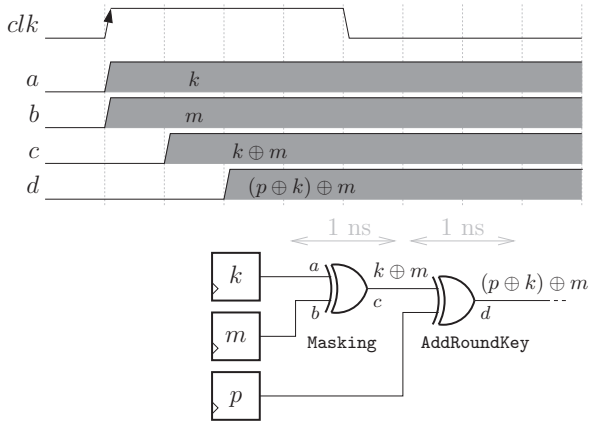

Figure 6. Design without glitch, where each sensitive variable (here, $p \oplus k$) is protected by a mask m.

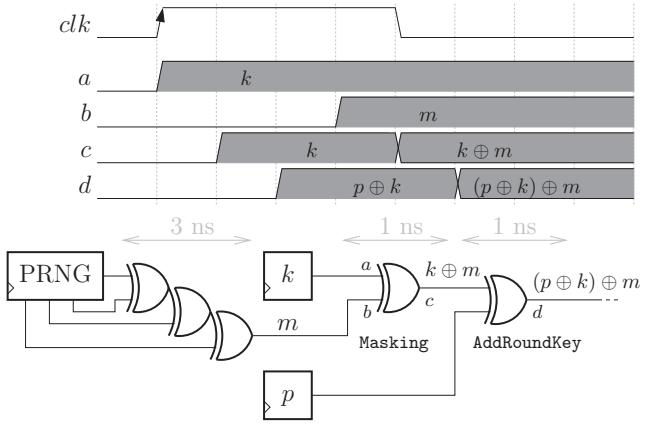

Figure 7. Glitch which reveals an unmasked sensitive operation, namely $p \oplus k$.

Another example is given in [10], where the Private Circuit I protection is broken in practice due to similar delays on the masks.

4.2.2. Correlated faults. In DPL circuits, it is generally assumed that it is unlikely two independent faults allow to simultaneously flip in opposite direction the pair. This is true for focused fault injections, e.g., thanks to LASER shots. However, so-called *global faults* [11] slightly affect all signals. Knowing that DPL logic has a balanced structure (*true* logic hides dual *false* logic), the critical path consists in a pair. Therefore, by a proper overclock or underfeed, it is possible to inject *correlated faults*, where $(0, 1)$ and $(1, 0)$ are swapped [12]. This is sketched in Fig. 8: nominal situation is represented in (a). In case of overclocking, the output (x_1, x_1') either (b) violates the invariant $x_1 = \neg x_1'$ or (c) samples an incorrect value which nonetheless respects the invariant. The situation (c) is dreadful, because there is no way to detect the fault, despite the redundant encoding of this DPL netlist.

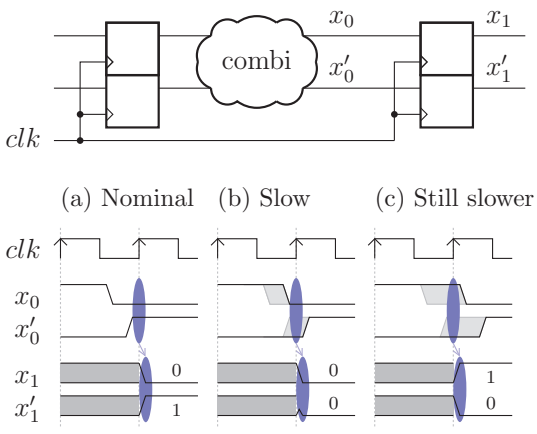

Figure 8. Illustration of setup time violation on a dual-rail circuit. (a): nominal case, the combinational logic evaluate before the rising edge of the clock; (b): slow case, the path x'_0 is violated; (c): still slower case, the two paths (x_0, x'_0) are violated, leading to a valid fault $(0,1) \rightarrow (1,0)$

4.2.3. IR Drop and capacitive coupling breaks isolation. Threshold Implementation enforces a logic separation between circuits parts, which ensures that any combinational logic part cannot leak information since not enough shares (e.g., only 2 out of 3). In the example of Fig. 9, it is shown in part (a) that indeed combinational parts are decoupled. However, this is only correct from a logic point of view. It happens that combinational parts might be layouted aligned, hence share the same power supply network. Therefore, when one part is computing, it is drawing electrical current, hence have the power rail (`gnd`) voltage drop (which is referred to as an *IR drop*), thereby influencing the way the other parts behave in terms of propagation delay and power consumption. Thus, an attacker observing the leakage of the TI implementation will have access to coupling between parts of the implementation which were assumed to be independent.

Besides, two lines are metal wires, separated by some dielectric. Metal-oxide-metal constitutes a capacitor. Therefore, there exists a capacitive coupling between signals, as shown in Fig. 9(c). This coupling is all the larger as lines y_i $(0 \leq i \leq 2)$ are longer and closer one to each other.

5. Solution to detect security breaches

Hopefully, CAD tools allow the user to place constraints on the synthesis and PR stages. Still, it is not obvious to set the constraints adequately. Besides, the constraints are usually not in the source code itself, and change from vendor to vendor, and sometimes even from one version to version for a given CAD vendor.

Therefore, an independent verification is mandatory. The goal of the verification can be to:

- perform some attacks the circuit is designed to resist to, and check that they are indeed thwarted;

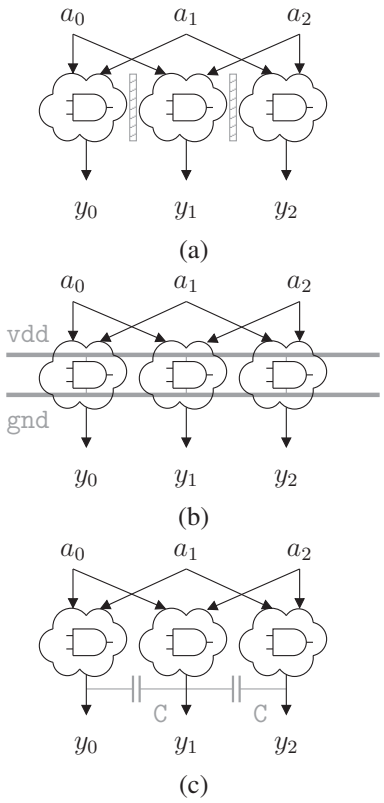

Figure 9. Threshold Implementation circuit. (a) Ideal view, where each combinational part is decoupled. (b) Coupling between parts due to IR-drop (coupling via the power supply network). (c) Coupling between parts due to parasitic capacitances (coupling via the oxide layer).

- in the case of side-channel attacks, some leakage metrics can be computed [13].

In this section, we describe a tool which allows to launch attacks and/or compute metrics, throughout the design flow.

The tool works in two steps:

1) fault injection and/or side-channel measurement campaign,
2) offline analysis.

The first step can be conducted in a straightforward way in a logic-level simulator. Faults are injected by forcing values. Side-channel traces are built from the VCD (Values Change Dump, defined in IEEE Standard 1364-1995) obtained after simulation. Propagation times in gates and in the interconnect yield races between signals, hence account for *glitches* (whose leakage can cause security issues). These can be simulated, and therefore assessed, if an SDF (Standard Delay Format, defined in IEEE Standard 1497-2001) file is provided along with the netlist.

The typical verification flow is depicted in Fig. 10. Specifically regarding side-channel matters, the analysis can be relaunched several times on the same traces. For instance, this allows to locate the leaky resources, e.g., by applying a *dichotomy* (traces are built from an always smaller selection of signals captured in the VCD file).

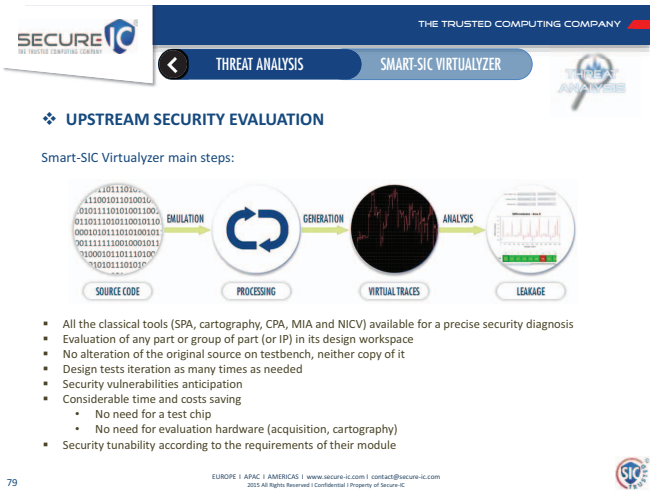

Figure 10. Methodology to evaluate the security of a design during the refinement flow.

There is only one limitation in using a logic-level simulator: the breaches such as IR-drop or capacitive coupling (Fig. 9(b) / (c)) cannot be simulated. To capture them, a SPICE simulator, or a mixed simulator (NanoSim by Synopsys / UltraSim by Cadence) is required.

6. Conclusions

In this paper, we have illustrated multiple real-world examples where *synthesis* and *PR* design stages enable transformations likely to break high-level security assumptions. We also show that such breaks can be remedied thanks to accurate and adequate scripts. However, the validation process needs to be independent, hence the need for a tool to check security while refining the circuit. Secure-IC Smart-SIC Virtualyzer can play such role, and thereby enable an end-to-end confidence in the overall security of the design.

References

[1] J. Blömer, J. Guajardo, and V. Krummel, "Provably Secure Masking of AES," in *Selected Areas in Cryptography*, ser. Lecture Notes in Computer Science, H. Handschuh and M. A. Hasan, Eds., vol. 3357. Springer, 2004, pp. 69–83.

[2] Y. Ishai, A. Sahai, and D. Wagner, "Private Circuits: Securing Hardware against Probing Attacks," in *CRYPTO*, ser. Lecture Notes in Computer Science, vol. 2729. Springer, August 17–21 2003, pp. 463–481, santa Barbara, California, USA.

[3] T. Popp, M. Kirschbaum, and S. Mangard, "Practical attacks on masked hardware," in *CT-RSA*, ser. Lecture Notes in Computer Science, vol. 5473. Springer, April 20-24 2009, pp. 211–225, San Francisco, CA, USA.

[4] S. Nikova, V. Rijmen, and M. Schläffer, "Secure Hardware Implementation of Non-linear Functions in the Presence of Glitches," in *ICISC*, ser. Lecture Notes in Computer Science, vol. 5461. Springer, 2008, pp. 218–234, Seoul, Korea.

[5] T. Sugawara, D. Suzuki, M. Saeki, M. Shiozaki, and T. Fujino, "On Measurable Side-Channel Leaks Inside ASIC Design Primitives," in *CHES*, ser. Lecture Notes in Computer Science, G. Bertoni and J.-S. Coron, Eds., vol. 8086. Springer, 2013, pp. 159–178.

[6] K. Tiri and I. Verbauwhede, "A VLSI Design Flow for Secure Side-Channel Attack Resistant ICs," in *DATE*. IEEE Computer Society, 2005, pp. 58–63, http://dx.doi.org/10.1109/DATE.2005.44.

[7] S. Bhasin, J.-L. Danger, F. Flament, T. Graba, S. Guilley, Y. Mathieu, M. Nassar, L. Sauvage, and N. Selmane, "Combined SCA and DFA Countermeasures Integrable in a FPGA Design Flow," in *ReConFig*. IEEE Computer Society, December 9–11 2009, pp. 213–218, Cancún, Quintana Roo, México, DOI: 10.1109/ReConFig.2009.50, http://hal.archives-ouvertes.fr/hal-00411843/en/.

[8] Y. Ishai, M. Prabhakaran, A. Sahai, and D. Wagner, "Private Circuits II: Keeping Secrets in Tamperable Circuits," in *EUROCRYPT*, ser. Lecture Notes in Computer Science, vol. 4004. Springer, May 28 – June 1 2006, pp. 308–327, St. Petersburg, Russia.

[9] V. Lomné, T. Roche, and A. Thillard, "On the need of randomness in fault attack countermeasures - application to AES," in *2012 Workshop on Fault Diagnosis and Tolerance in Cryptography, Leuven, Belgium, September 9, 2012*, G. Bertoni and B. Gierlichs, Eds. IEEE Computer Society, 2012, pp. 85–94. [Online]. Available: http://dx.doi.org/10.1109/FDTC.2012.19

[10] D. B. Roy, S. Bhasin, S. Guilley, J. Danger, and D. Mukhopadhyay, "From theory to practice of private circuit: A cautionary note," in *33rd IEEE International Conference on Computer Design, ICCD 2015, New York City, NY, USA, October 18-21, 2015*. IEEE Computer Society, 2015, pp. 296–303. [Online]. Available: http://dx.doi.org/10.1109/ICCD.2015.7357117

[11] S. Guilley and J.-L. Danger, "Global Faults on Cryptographic Circuits," pp. 297–314, Chapter 17 of [14].

[12] H. Rakotomalala, X. T. Ngo, Z. Najm, J. Danger, and S. Guilley, "Private circuits II versus fault injection attacks," in *International Conference on ReConFigurable Computing and FPGAs, ReConFig 2015, Riviera Maya, Mexico, December 7-9, 2015*, M. Hübner, M. Gokhale, and R. Cumplido, Eds. IEEE, 2015, pp. 1–9. [Online]. Available: http://dx.doi.org/10.1109/ReConFig.2015.7393338

[13] R. J. Easter, J.-P. Quemard, and J. Kondo, "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. (Online).

[14] M. Joye and M. Tunstall, *Fault Analysis in Cryptography*. Springer LNCS, March 2011, DOI: 10.1007/978-3-642-29656-7 ; ISBN 978-3-642-29655-0.