Detecting Flawed Masking Schemes with Leakage Detection Tests

Oscar Reparaz(✉)

KU Leuven Department of Electrical Engineering-ESAT/COSIC and iMinds, Kasteelpark Arenberg 10, 3001 Leuven-Heverlee, Belgium
oscar.reparaz@esat.kuleuven.be

Abstract. Masking is a popular countermeasure to thwart side-channel attacks on embedded systems. Many proposed masking schemes, even carrying "security proofs", are eventually broken because they are flawed by design. The security validation process is nowadays a lengthy, tedious and manual process. In this paper, we report on a method to verify the soundness of a masking scheme before implementing it on a device. We show that by instrumenting a high-level implementation of the masking scheme and by applying leakage detection techniques, a system designer can quickly assess at design time whether the masking scheme is flawed or not, and to what extent. Our method requires not more than working high-level source code and is based on simulation. Thus, our method can be used already in the very early stages of design. We validate our approach by spotting in an automated fashion first-, second- and third-order flaws in recently published state-of-the-art schemes in a matter of seconds with limited computational resources. We also present a new second-order flaw on a table recomputation scheme, and show that the approach is useful when designing a hardware masked implementation.

1 Introduction

Since Kocher published the seminal paper on side-channel attacks [Koc96], cryptographic embedded systems have been broken using some auxiliary timing information [Koc96], the instantaneous power consumption of the device [KJJ99] or the EM radiation [AARR02], among others. An attack technique of particular interest, due to its inherent simplicity, robustness and efficiency to recover secrets (such as cryptographic keys or passwords) on embedded devices is Differential Power Analysis (DPA), introduced in [KJJ99]. DPA relies on the fact that the instantaneous power consumption of a device running a cryptographic implementation is somehow dependent on the intermediate values occurring during the execution of the implementation. An especially popular countermeasure to thwart power analysis attacks, including DPA, is masking [CJRR99, GP99]. Masking works by splitting every sensitive variable appearing during the computation of a cryptographic primitive into several shares, so that any proper subset of shares is independent of any sensitive variable. This, in turn, implies that the instantaneous power consumption of the device is independent of any sensitive

T. Peyrin (Ed.): FSE 2016, LNCS 9783, pp. 204–222, 2016.
DOI: 10.1007/978-3-662-52993-5_11

variable, and thus vanilla DPA cannot be mounted. In theory, a $(d+1)$-order DPA attack can still be mounted against a d-th order masked implementation; however, in practice higher order DPA attacks are exponentially more difficult to carry out [CJRR99].

Crucially, in many cases the attacker is not required to perform a higher order attack because the masking is imperfect and thus does not provide the claimed security guarantees. The causes of the imperfections can be manifold: from implementation mistakes to more fundamental flaws stemming from the masking scheme itself. There are many examples in the literature of such flawed schemes: a "provably secure" scheme published in 2006 [PGA06] based on FFT and broken two years later [CGPR08], a scheme published in 2006 [SP06] and broken one year later [CPR07], another "provably secure" scheme published in 2010 [RP10] and (academically) broken three years later [CPRR13]; a scheme published in 2012 [BFGV12] and broken in 2014 [PRR14].

The verification process of a masking scheme is nowadays a very lengthy manual task, and the findings are published in solid papers involving convoluted probability arguments at leading venues, some years later after the scheme is published. Some even won a best paper award as [CPR07]. From the stand point of a system designer, it is often not acceptable to wait for a public scrutiny of the scheme or invest resources in a lengthy, expensive, evaluation.

Our Contribution. In this paper we provide an automated method to test whether the masking scheme is sound or not, and to what extent. The method is conceptually very simple, yet powerful and practically relevant. We give experimental evidence that the technique works by reproducing state-of-the-art first-, second- and third-order flaws of masking schemes with very limited computational resources. Our approach is fundamentally different from previously proposed methodologies and is based on sampling and leakage detection techniques.

2 Leakage Detection for Masked Schemes in Simulation

Core Idea. In a nutshell, our approach to detect flawed masking schemes is to simulate power consumption traces from a high-level implementation of the masking scheme and then perform leakage detection tests on the simulated traces to verify the first- and higher-order security claims of the masking scheme.

Input and Output of the Tool. The practitioner only ought to provide working source code of the masked implementation. The tool instruments the code, performs leakage detection tests and outputs whether the scheme meets its security goals or not. In addition, should a problem be detected, the tool pinpoints the variables causing the flaw and quantifies the magnitude of the statistical bias.

Security Claims of Masking Schemes. We use in this paper the conventional notions for expressing the security claim of a masking scheme. Namely, a masking scheme provides first-order security if the expected value of each single intermediate does not depend on the key. More generally, a masking scheme provides

k-order security if the k-th order statistical moment of any combination of k intermediates does not depend on the key. This formulation is convenient since leakage detection tests are designed specifically to test these claims.

Three Steps. Our tool has three main ingredients: trace generation, trace pre-processing and leakage detection. We describe each one in detail in the sequel.

2.1 Trace Generation

The first step of our approach is to generate simulated power traces in a noise-free environment.

Implementation. To accomplish this, the masking scheme is typically implemented in a high-level software language. The implementation is meant to generically reproduce the intermediate values present in the masking scheme, and can be typically written from the pseudo-code description of the masking scheme. (Alternatively, the description of the masking scheme can be tailored to a specific software or hardware implementation and incorporate details from those.)

Execution. This implementation is executed many times, and during each execution, the instrumentation environment observes each variable V that the implementation handles at time n. At the end of each execution, the environment emits a leakage trace $c[n]$. Each time sample n within this trace consists of leakage $L(V)$ of the variable V handled at time n. The leakage function L is predefined; typical instantiations are the Hamming weight, the least significant bit, the so-called zero-value model or the identity function.

Randomness. The high-level implementation may consume random numbers (for example, for remasking.) This randomness is provided by a PRNG.

2.2 Trace Pre-processing

This step is only executed if the masking scheme claims higher-order security. The approach is similar to higher-order DPA attacks [CJRR99] and higher-order leakage detection [SM15]. Suppose the scheme claims security at order k. We preprocesses each simulated trace $c[n]$ to yield $c'[n_1, \ldots, n_k]$ through a combination function as

$$c'[n_1, \ldots, n_k] = \prod_{i=1}^{i=k} (c[n_i] - \bar{c}[n_i]). \tag{1}$$

The result is a preprocessed trace c'. The length of the trace is expanded from N to $\binom{N}{k}$ unique time samples. (It is normally convenient to treat c' as a uni-dimensional trace.)

2.3 Leakage Detection

The next step of our approach is to perform a leakage detection test on the (potentially pre-processed) simulated traces. In its simplest form, a leakage detection test [CKN00, CNK04, GJJR11, CDG+13, SM15] tries to locate and potentially quantify information leakage within power traces, by detecting statistical dependencies between sensitive data and power consumption. In our context, if the test detects information leakage on the simulated traces, this means that the masking scheme fails to provide the promised security guarantees.

Procedure. The instrumentation environment performs a fixed-vs-fixed leakage detection test using the T-test distinguisher [CDG+13].

The process begins by simulating a set of power traces with fixed unmasked intermediate $z = z_0$ and another set of traces with different unmasked intermediate value $z = z_1$. Typical choices for the intermediate z are the full unmasked state or parts of it. Then, a statistical hypothesis test (in this case, T-test) is performed per time sample for the equality of means. The T-test [Stu08, Wel47] first computes the following statistic

$$t[n] = \frac{m_0[n] - m_1[n]}{\sqrt{\frac{s_0^2[n]}{N_0} + \frac{s_1^2[n]}{N_1}}} \tag{2}$$

where $m_i[n]$, $s_i^2[n]$, N_i are respectively the sample mean, variance and number of traces of population $i \in \{0, 1\}$ and n is the time index. This statistic $t[n]$ is compared against a predefined threshold C. A common choice is $C = \pm 4.5$, corresponding to a very high statistical significance level of $\alpha = 0.001$. If the statistic $t[n]$ surpasses the threshold C, the test determines that the means of the two distributions are significantly different, and thus the mean power consumption of (potentially pre-processed) simulated power traces carry information on the intermediate z. In this case, we say that the masking scheme exhibits leakage at time sample n and flunks the test. Otherwise, if no leakage is detected, another test run is executed with different specific values for z_0 and z_1. The test is passed only if no leakage is detected for any value of z_0 and z_1. (Typically, there are only a couple dozen of (z_0, z_1) pairs if the optimizations described in the next section are applied.) Note that a time sample n may correspond to a single variable (first-order leakage) or a combination of variables (higher-order leakage), if a pre-processing step is executed.

On Fixed-vs-Fixed. Using fixed-vs-fixed instead of fixed-vs-random has the advantage of faster convergence of the statistic (at the expense of leakage behavior assumptions that are benign in our context). (This has been previously observed by Durvaux and Standaert [DS15] in a slightly different context.) One could also use a fix-vs-random test. This usually results in a more generic evaluation.

2.4 Optimizations

We note that the following "engineering" optimizations allow to lower the computational complexity so that it is becomes very fast to test relevant masking schemes.

Online Algorithms. There is certainly no need to keep in memory the complete set of simulated power traces. For the computation of the T-test as Eq. 2, one can use online formulas to compute means and variances present in the formula. These algorithms traverse only once through each trace, so that a simulated power consumption trace can be generated, processed and thrown away. This makes the memory consumption of the approach independent of the number of traces used. More number of traces would require just more computational time, but not more memory. We note that the same is possible in higher-order scenarios. Lengthy but straightforward calculations show that a T-test on preprocessed traces can be computed online using well-known online formulae for (mixed) higher-order moments [P08]. (This was previously reported by Schneider and Moradi [SM15].)

Scale Down the Masking Scheme. It is usually possible to extrapolate the masking scheme to analogous, trimmed down, cryptographic operations that work with smaller bit-widths or smaller finite fields. For example, when masking the AES sbox, many masking schemes [RP10, CPRR13] rely on masked arithmetic (masked multiplication and addition blocks) in $\mathrm{GF}(2^8)$ to carry out the inversion in $\mathrm{GF}(2^8)$. It is often convenient to scale down the circuit to work on, say, $\mathrm{GF}(2^4)$ for testing purposes –since the masking approach normally does not rely on the specific choice of field size, any flaw exhibited in the smaller $\mathrm{GF}(2^4)$ version is likely to be exhibited in the $\mathrm{GF}(2^8)$ version of the algorithm (and vice versa). By experience we have observed that statistical biases tend to be more pronounced in smaller field sizes, and thus are more easily detectable. (See for instance [PRR14].) We suggest the use of this heuristic whenever possible for an early alert of potential problems.

Reduce the Number of Rounds. There is little sense to check for a first-order leak in more than a single round of an iterative cryptographic primitive, such as AES. If the implementation is iterative, any first-order flaw is likely to show up in all rounds. When testing for higher order security, however, one should take into account that the flaw may appear from the combination of variables belonging to different rounds.

Judiciously Select the Components to Check. For first-order security it is sufficient to check each component of the masking scheme one by one in isolation. The situation is slightly different in the multivariate scenario, where multiple components can interfere in a way that degrades security. Still, the practitioner can apply some heuristics to accelerate the search, such as testing for second-order leakage first only in contiguous components. For example, second-order

leakage is likely to appear earlier between two variables within the same round or belonging to two consecutive rounds.

Deactivate Portions of the Plaintext. To accelerate the leakage search, a substantial portion of the plaintext can be deactivated, that is, fixed to a constant value or even directly taken out from the algorithm. For example, in the case of AES-128 one could deactivate 3 columns of the state, test only 4 active plaintext bytes and still test for the security of all the components within one round.

Carefully Fix the Secret Intermediate Values. As we described, the framework fixes two values z_0, z_1 for the unmasked sensitive intermediate, and then compares the simulated traces distributions conditioned on z_0 and z_1. Depending on the algorithm, concrete choices for z_i (such as fixed points of the function being masked) can produce "special" leakage. For example, in AES if we choose z_1 such that the input to the inversion is 0×00, we can hit faster zero-value type flaws.

3 Results

In this section we provide experimental results. We first begin by testing the first-order security claim of two schemes, one that fails the claim (Sect. 3.1) and another that fulfills it (Sect. 3.2). Then we will focus on second- and third- order claims (Sects. 3.3 and 3.4 respectively). We point out a new second-order flaw in Sect. 3.5, we elaborate on how previously published flaws were discovered in Sect. 3.6. Finally in Sect. 3.7 we report on the use of the tool when designing masked circuits.

3.1 Smoke Test: Reproducing a First-Order Flaw

As a first test, we test the first-order security of the scheme published in [BFGV12]. We will refer to this scheme as IP in the sequel. We focus on reproducing the results from [PRR14],

Test Fixture. We study first the `IPRefresh` procedure. This procedure performs a refreshing operation on the input IP shares. We scale down the scheme to work in $\mathrm{GF}(2^2)$ following Sect. 2.4. The instrumentation framework finds 141 intermediate variables within a single execution of `IPRefresh`. The chosen leakage function is Hamming weight, and there is no pre-processing involved.

Leakage Detection. We ran the $\binom{4}{2} = 6$ possible fixed-vs-fixed tests covering all possible combinations of pairs of different unshared input values (z_1, z_0). (Here z_i is the input to `IPRefresh`.) For each test, the maximum absolute t-value, across all time samples, is plotted in the y-axis of Fig. 1 as a function of the number of simulated traces (x-axis). A threshold for the T-test at 4.5 is also plotted as a dotted line. This threshold divides the graph into two regions: a

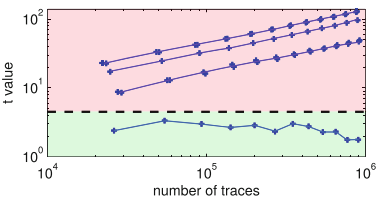

Fig. 1. T-statistic (absolute values) of the IP masking scheme, under a HW leakage model. Deemed insecure (clearly exceeds the threshold at $t = 4.5$.) (Color figure online)

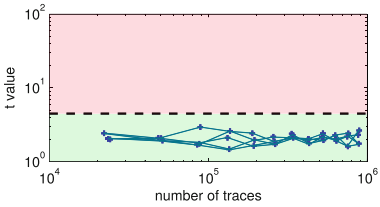

Fig. 2. T-statistic (absolute values) applied to the Coron table recomputation masking scheme, under an Identity leakage model. First order test. Deemed secure (no value beyond the threshold at $t = 4.5$.)

t-statistic greater than $|C| = 4.5$ (in red) means that the implementation fails the test, while a t-statistic below 4.5 (area in green) does not provide enough evidence to reject the hypothesis that the scheme is secure. We can see that 5 out of 6 tests clearly fail in Fig. 1, since they attain t-values around 100 greater than C. Thus, the `IPRefresh` block is deemed insecure. (Similar observations apply to the `IPAdd` procedure.)

It is also possible to appreciate the nature of the T-test statistic: the t-statistic grows with the number of traces N as of $\sqrt{N}$ in the cases that the implementation fails the test (note that the y-axis is in logarithmic scale.) This can be interpreted as follows: as we have more measurements, we build more confidence to reject the null hypothesis (in our context being that the masking is effective.) If the number of simulated traces is large enough and no significant t-value has been observed, the practitioner can gain confidence on the scheme not being flawed. We will find this situation in the next subsection and elaborate on this point.

3.2 A First-Order Secure Implementation

We tested the table recomputation scheme of Coron [Cor14]. This scheme passes all fixed-vs-fixed tests with the identity leakage model. The results are plotted in Fig. 2. We can observe that the t-statistic never crosses the threshold of 4.5 for any test, and thus we cannot reject the null hypothesis that the implementation is secure (i.e., the implementation is deemed secure, "*on the strength of the evidence presented*" [CKN00]). Although it is theoretically possible that the masking scheme exhibits a small bias that would only be detectable when using more than 10^6 traces, that flaw would be negligible from a practical point of view when using $\leq 10^6$ traces, and definitely impossible to exploit in a noisy environment if it is not even detectable at a given trace count, in a noiseless scenario.

```
 70 void MaskRefresh(u8 *s) {
 71  u8 r;
 72  for (int i = 1; i < number_shares; i++) {
 73   r = rnd ();
 74   s[0] ^= r;
 75   s[i] ^= r;
 76  }
 77 }
...
110 void SecMult (u8 *out, u8 *a, u8 *b) {
111  u8 aibj,ajbi;
...
114  for (int i = 0; i < number_shares; i++) {
115   for (int j = i + 1; j < number_shares; j++) {
...
119    aibj = mult(a[i], b[j]);
120    ajbi = mult(a[j], b[i]);
-------------------------------------------------
$ ./run
entering fixed_vs_fixed(00,01)
> leakage detected with 1.20k traces
  higher order leakage between
    line 74 and
    line 120
  with tvalue of -7.03
```

Fig. 3. Excerpts of the code and output of the leakage detection for the RP scheme.

3.3 Reproducing a Second-Order Flaw

To show that our proposed tool can also detect higher-order flaws, we implemented the scheme of Rivain and Prouff (RP) from [RP10]. For the allegedly second-order secure version of this scheme, there is a second-order flaw as spotted by Coron et al. in [CPRR13] between two building blocks: `MaskRefresh` and `SecMult`. We will see that we can easily spot this flaw with the methodology proposed in this paper.

Text Fixture. We implemented the second-order masked inversion $x \mapsto x^{-1}$ in $\mathrm{GF}(2^n)$ as per [RP10] with $n = 3$. This inversion uses the `MaskRefresh` and `SecMult` procedures. In this case, we enable second-order pre-processing (on the fly), expanding 135 time samples to $\binom{135}{2} = 9045$ time samples. Some excerpts of the implementation are shown in Fig. 3.

Results. The instrumentation frameworks takes less than 5 s to determine that there is a second order leakage between the variable handled at line `74` (inside `MaskRefresh`) and `120` (inside `SecMult`), as Fig. 3, bottom, shows. Note that it is trivial to backtrack to which variables corresponds a leaking time sample, and thus determine the exact lines that leak jointly.

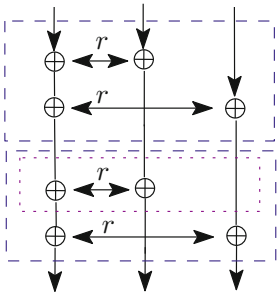

Fig. 4. Two `MaskRefresh` concatenated. As explained in the text, the second refresh can be optimized to reduce the randomness requirements yet still achieving second order security. (Color figure online)

Fixing the Second-Order Flaw. The folklore solution to fix the previous second-order flaw is to substitute each `MaskRefresh` module by two consecutive `MaskRefresh` invocations, as shown in Fig. 4. Applying the leakage detection tests to this new construction shows that the leak is effectively gone. However, it is quite reasonable to suspect that this solution is not optimal in terms of randomness requirements. We can progressively strip down this design by eliminating some of the randomness of the second refreshing and check if the design is still secure. We verified in this very simple test fixture that if we omit the last randomness call (that is, we only keep the dotted red box instead of the second dashed box in Fig. 4), the higher-order leaks are no longer present.

3.4 Reproducing a Third Order Flaw

Schramm and Paar published at CT-RSA 2006 [SP06] a masked table lookup method for Boolean masking claiming higher-order security. This countermeasure was found to be flawed by Coron et al. at CHES 2007. Coron et al. found a third-order flaw irrespective of the security parameter of the original scheme. We reproduced their results by setting $k = 3$ when preprocessing the traces as in Eq. 1. The flaw of [CPR07] was detected in less than one second, which demonstrates that the tool is also useful to test the higher-order security claims of masking schemes.

3.5 Schramm–Paar Second-Order Leak

Here we report on a new second-order flaw that we found with the presented tool in the masked table recomputation method of Schramm and Paar when used with unbalanced sboxes.

Schramm–Paar Method. The goal of the masked table recomputation is to determine the sbox output shares $N_0, N_1, \ldots, N_d$ from the sbox input shares $M_0, M_1, \ldots, M_d$. Schramm–Paar proceed as follows (we borrow the notation from [CPR07]):

1. Draw d output shares $N_1, \ldots, N_d$ at random
2. Compute from $N_1, \ldots, N_d$ a table S^* such that

$$S^*(x) = S\left(x \oplus \bigoplus_{i=1}^{d} M_i\right) \oplus \bigoplus_{i=1}^{d} N_i \tag{3}$$

3. Set $N_0 := S^*(M_0)$

We set here $d = 2$, and aim at second-order security. An important part of the procedure is to build the table S^* in a way that the higher-order security claims are fulfilled. [SP06] proposes several methods. However, for the purposes of this paper the details of the recomputation method are not important.

Test Fixture. Following the guidelines of Sect. 2.4, we implement a very stripped down version of the table recomputation method. We fix the simplest unbalanced sbox $S = (0, 0, 0, 1)$ (an AND gate), and work with 2-bit inputs and outputs leaking Hamming weights. In a couple of seconds the tool outputs 4 different bivariate second-order leakages, corresponding to the pairs $(S^*(i), N_0)$ for each i in the domain of S^*. Here $S^*(i)$ is the i-th entry on the S^* table, and N_0 is one output mask.

Once these leaks are located, proving them becomes an easy task. And also it becomes easy to generalize and see that the flaw appears whenever S is unbalanced. (We verified that second-order attacks using the leakage of $S^*(0)$ and N work as expected.)

3.6 Higher-Order Threshold Implementations

Here we report on how the observations from [RBN+15] regarding the security of higher-order threshold implementations [BGN+14] were found. The results of this section are obviously not new; the focus here is on the methodology carried out to find those.

Intuition. The first suspicion stems from the fact that higher-order threshold implementations originally claimed that the composition of sharings provides higher-order security, if the sharings satisfy some property, namely uniformity. This is a very surprising result, since it would imply that there is no need to inject fresh randomness during the computation, minimizing overheads. In contrast, all other previously published higher-order masking schemes need to inject randomness from time to time as the computation progresses. For example, the security proof of private circuits (one of the earliest masking schemes) [ISW03] critically relies on the fresh randomness to provide security.

Test Fixture. The hypothesis is that the previous security claim does not hold, that is, the concatenation of uniform sharings do not provide higher-order security. To test this, we design a minimal working test fixture consisting of a 32-round Feistel cipher with a blocksize of 4 bits. For more details see [RBN+15].

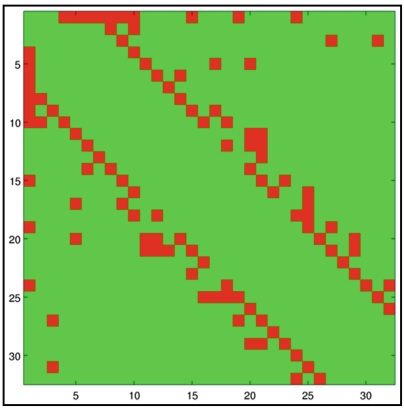

Fig. 5. Pairs of rounds with $|t| > 80$ (Color figure online)

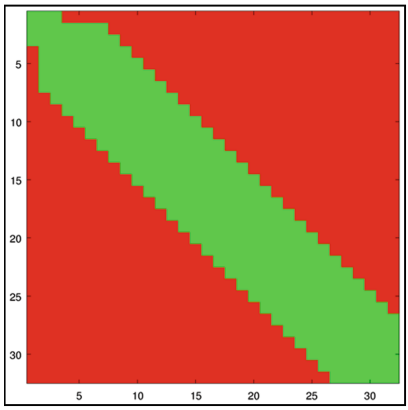

Fig. 6. Pairs of rounds with $|t| > 5$ (Color figure online)

The shared version aims at providing second-order security, and shares each native bit into 5 shares. The traces consist of 225 "timesamples" (each one comprising one leaked bit, including initialization.) This spans to 25650 timesamples after second-order pre-processing.

Cranking it up. We run the simulation for a night (about 8 h), having simulated 200 million traces. We performed a fixed-vs-fixed test with unshared initial state `0000` and `1111`. (There is no key in this cipher, the initial state is considered to be the secret.) (This is grossly unoptimized code.) The results of the leakage detection test is drawn in Fig. 5. We plot on the x- and y-axes the round index, and each pixel in red if the t statistic surpasses the value 80, green otherwise. We can see that many pairs of rounds leak jointly, in contradiction with the security claims of the scheme. In Fig. 6 the same information is plotted but changing the threshold to $|t| > 5$. We can see, surprisingly, that *almost all* pairs of rounds lead to second-order leakage. A bit of manual mechanical effort is required to prove this, but not more than taking a covariance.

3.7 Refreshing in Higher-Order Threshold AES Sbox

The designers from [CBR+15] had access to the tool presented in this paper. They performed several design iterations, and verified the design on each iteration. The final evaluation was performed on an FPGA.

Text Fixture. We implemented the whole sbox, with no downscaling of the components to work in smaller fields. We leak register bits and the input value (identity leakage function) to combinatorial logic blocks. (This is to account for glitches as will be explained below.)

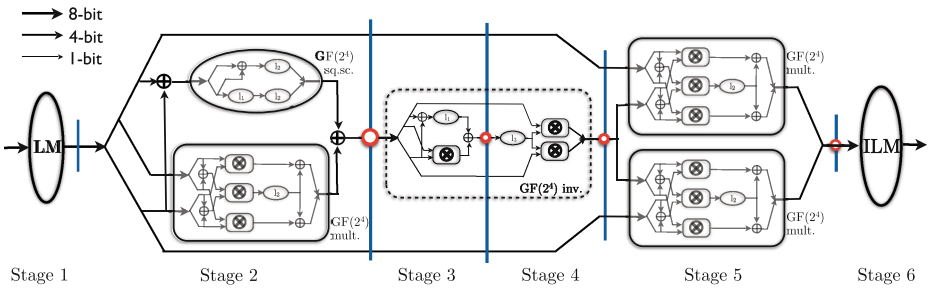

Fig. 7. Higher-order masked AES sbox from de Cnudde et al.

First-Order Leaks. Within one day, a first-order leak was identified due to a design mistake. This design error considered the concatenation of two values $a||b$ as input to the next stage; each value a and b considered independently is a uniform sharing but its concatenation $a||b$ is not, and hence the first order leak. This first-order leak disappears if a refresh is applied to the inputs of one GF(2^2) multiplier using 4 units of randomness (here 1 unit = 1 random field element = 2 bits). This refresh block is similar to the 2010 Rivain–Prouff refresh block [RP10], we remind it uses $n-1$ units of randomness to refresh n shares (in our particular case here $n = 5$). We will see later that this refresh is problematic in the higher-order setting.

Second-Order Leaks. Subsequently, two second-order bivariate leaks were identified between register values. This was solved by placing a refresh block between stage 2 and 3 from Fig. 7 (taken from [CBR+15]).

In addition, many second-order bivariate leaks were identified between input values to combinatorial logic blocks. In theory, hardware glitches could express these leakages. Those disappear if one uses a "full refresh" using 5 units of randomness. This effect was previously observed [BBD+15,RBN+15] and is a reminiscent of [CPRR13].

Other Uses. We also used a preliminary version of this tool in [RRVV15].

4 Discussion

4.1 Implementing the Framework

We implemented the instrumentation framework on top of `clang`-LLVM. The whole implementation (including leakage detection code) takes around 700 lines of C++ code, which shows the inherent simplicity of our approach. It is easy to audit and maintain.

4.2 Time to Discover Flaw, Computational Complexity and Scaling

The computational requirements of the proposed approach are very low. In Fig. 8 we write the elapsed execution times required to spot the flaws from Sects. 3.1, 3.3 and 3.4. We can see that the flaws were identified in a matter of seconds on a standard computer. All the experiments on this paper were carried out on a modest 1.6 GHz laptop with 2 GB of RAM.

Bottlenecks. The main bottleneck on the running time of the methodology is the first step: trace generation. The RP scheme is the one that took longer to detect the flaw (5 s), presumably because of two reasons: (a) the scheme is more inherently complex and thus it takes more time to simulate each trace and (b) the bias exhibited in the scheme is smaller than the bias of other schemes, and thus more traces are required to detect such a bias. We note that no special effort on optimizing the implementations was made, yet, an average throughput of 5 k trace per second (including instrumentation) was achieved. The overhead of instrumentation in the running time was estimated to make the implementation on average $\approx \times 1.6$ slower.

Time to Pass. The time it takes to discover a flaw is normally less than the time it takes to deem a masking scheme secure. For example, to assess that the patch of Sect. 3.3 is indeed correct, it took about 6 min to perform a fix-vs-fix test with up to 1 million traces (no leakage was detected). All possible 6 tests take around 37 min. (The threshold of 1 million traces was chosen arbitrarily in this example.)

Parallelization. We remark that this methodology is embarrassing parallel. Thus, it is much easier to parallelize to several cores or machines than other approaches based on SAT.

Memory. The memory requirements for this method are also negligible, taking less than 4.5 MB of RAM on average. More interestingly, memory requirements are constant and do not increase with the number of simulated traces, thanks to online algorithms.

Scheme	Flaw order	Field size	Time	Traces needed
IP	1	4	0.04s	1k
RP	2	4	5s	14k
SP	3	4	0.2s	2k

Fig. 8. Running time to discover flaw in the studied schemes, and number of traces needed to detect the bias.

Scaling. The execution time of our approach scales linearly with the number of intermediates when testing for first-order leakage, quadratically when testing for second-order leakage and so on. This scaling property is exactly the same as for DPA attacks. We could benefit from performance improvements that are typically used to mitigate scaling issues in DPA attacks such as trace compression, but did not implemented those yet.

4.3 Limitations

Risk of False Negatives. Our tool should not be taken as the only test when assessing a masked implementation, and is not meant to substitute practical evaluation with actual measurements. Our tool provides an early warning that the masking scheme may be structurally flawed, "by design". However, even when the masking scheme is theoretically secure, it is still possible to implement it in an insecure way. This will not be detected with the proposed tool. For example, in the case of a first-order masked software implementation, an unfortunate choice of register allocation may cause distance leakage between shares, leading to first-order leakage. Without register allocation information, our tool will not detect this issue. One could provide this kind of extra information to our tool. We left this as future work.

4.4 Related Works

There are already some publications that address the problem of automatic verification of power analysis countermeasure.

SAT-based. Sleuth [BRNI13] is a SAT-based methodology that outputs a hard yes/no answer to the question of whether the countermeasures are effective or not. A limitation of [BRNI13] is that it does not attempt to quantify the degree of (in)security. A first approximation to the problem was tackled in [EWTS14, ABMP13].

MiniCrypt-based. Barthe et al. [BBD+15] use program verification techniques to build a method prints a proof of security for a given masking scheme. It is very hard to compare our tool with theirs since they are fundamentally different. The goal is also different: while our results are probabilistic, the goal of Barth et al. is to categorically prove the security of the scheme. Depending on the context, one might be preferrable over the other. The two approaches are also very different. Barthe et al. base their approach on EasyCrypt, a sophisticated "toolset for reasoning about relational properties of probabilistic computations with adversarial code."

Considerations Related to Other Approaches. While our approach does certainly not carry the beauty of proofs and formal methods, it offers a very practice-oriented methodology to test the soundness of masking schemes. Our approach is in nature statistical, and is a necessary condition for a masked scheme to

be sound. It can be thought of a worst-case scenario, where the adversary has access to noiseless and synchronized traces. A more formal study can then be performed with the methods of Barthe et al. to gain higher confidence, since the output of the tool of Barthe et al. is a hard proof.

4.5 Which Leakage Function to Select?

In previous Sect. 2 we mentioned that the practitioner has to choose a leakage function to generate the simulated traces. It turns out that the specific choice of leakage function seems not to be crucial —any reasonable choice will work. Figure 9 compares different leakage functions: Hamming weight, identity, least-significant bit and zero-value. The test fixture is the same one as in Sect. 3.1. For each leakage function, we performed all possible fixed-vs-fixed tests. Figure 9 is composed of 4 plots, one per leakage function. We can see that for any leakage function, there is at least one fixed-vs-fixed test that fails. For the identity leakage function, *all* tests fail. Thus, it is often convenient to use it to detect flaws faster (more fixed-vs-fixed tests fail.) We speculate that this behavior may depend on the concrete masking method used, and leave a detailed study as future work.

Glitches and Identity Leakage. We note that we can include the effect of hardware glitches in our tool. Note that the information leaked by a combinatorial logic block F on input x due to glitches is contained already in the input x. Thus, we can simulate the information leaked by hardware glitches, even if we do not have a precise timing model of the logic function, by leaking the whole input value x (that is, using the identity leakage model.)

This would correspond to an extremely glitchy implementation of F where glitches would allow to observe the complete input. This is certainly a worst-case scenario. Crucially, glitches would not reveal *more* information than x. This trick

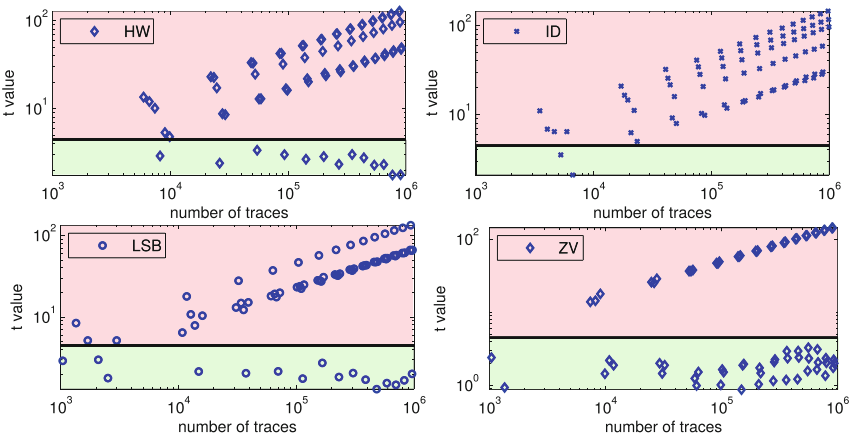

Fig. 9. Influence of leakage function.

of using the identity leakage model on inputs of combinatorial blocks is helpful when evaluating, for example, masked threshold implementations.

Another alternate approach is to add a detailed gate-level timing model to simulate glitches. If such timing model is available, the detection quality can be substantially enhanced.

5 Conclusion

We described a methodology to test in an automated way the soundness of a masking scheme. Our methodology enjoys several attractive properties: simplicity, speed and scalability. Our methodology is based on established and well-understood tools (leakage detection). We demonstrated the usefulness of the tool by detecting state-of-the-art flaws of modern masking designs in a matter of seconds with modest computational resources. In addition, we showed how the tool can assist the design process of masked implementations.

Acknowledgements. We thank an anonymous reviewer that found a mistake in Sect. 3.5, François-Xavier Standaert for extensive comments and Ingrid Verbauwhede. The author is funded by a PhD fellowship of the Fund for Scientific Research - Flanders (FWO). This work was funded also by Flemish Government, FWO G.0550.12N, G.00130.13N, Hercules Foundation AKUL/11/19, and through the Horizon 2020 research and innovation programme under grant agreement 644052 HECTOR.

Auxiliary Supporting Material

A MATLAB Code

This code prints the distribution of $Z = S(M \oplus M_0) \oplus S(M)$ for a fixed M and varying M_0.

```
% the sbox
S=[0 0 0 1];
% number of samples
N=10000;

% the sbox input
for M=0:3
    M0=floor(4.*rand(1,N));
    Z =bitxor(S(bitxor(M,M0)+1),S(M+1));

    for i=0:1
        fprintf(' p(Z=%d|M=%d) = %1.2f\n', i, M, sum(Z==i)./length(Z))
    end
    fprintf('\n')
end
```

B Examplary Output

This is the distribution of Z when the secret M takes different values. We can see that the expected value of Z is different when conditioned on $M = 0$ than when $M = 3$. This means that there is a second-order information leak between $(S^*(0), N_0)$ and the secret M.

```
p(Z=0|M=0) = 0.75
p(Z=1|M=0) = 0.25

p(Z=0|M=1) = 0.75
p(Z=1|M=1) = 0.25

p(Z=0|M=2) = 0.75
p(Z=1|M=2) = 0.25

p(Z=0|M=3) = 0.25
p(Z=1|M=3) = 0.75
```

References

[AARR02] Agrawal, D., Archambeault, B., Rao, J.R., Rohatgi, P.: The EM side-channel(s). In: Kaliski Jr., B.S., Koç, Ç.K., Paar, C. (eds.) CHES 2002. LNCS, vol. 2523. Springer, Heidelberg (2003)

[ABMP13] Agosta, G., Barenghi, A., Maggi, M., Pelosi, G.: Compiler-based side channel vulnerability analysis and optimized countermeasures application. In: 2013 50th ACM/EDAC/IEEE Design Automation Conference (DAC), pp. 1–6, May 2013

[BBD+15] Barthe, G., Belaïd, S., Dupressoir, F., Fouque, P.-A., Grégoire, B., Strub, P.-Y.: Verified proofs of higher-order masking. In: Oswald, E., Fischlin, M. (eds.) EUROCRYPT 2015. LNCS, vol. 9056, pp. 457–485. Springer, Heidelberg (2015)

[BFGV12] Balasch, J., Faust, S., Gierlichs, B., Verbauwhede, I.: Theory and practice of a leakage resilient masking scheme. In: Wang, X., Sako, K. (eds.) ASIACRYPT 2012. LNCS, vol. 7658, pp. 758–775. Springer, Heidelberg (2012)

[BGN+14] Bilgin, B., Gierlichs, B., Nikova, S., Nikov, V., Rijmen, V.: Higher-order threshold implementations. In: Sarkar, P., Iwata, T. (eds.) ASIACRYPT 2014, Part II. LNCS, vol. 8874, pp. 326–343. Springer, Heidelberg (2014)

[BRNI13] Bayrak, A.G., Regazzoni, F., Novo, D., Ienne, P.: Sleuth: automated verification of software power analysis countermeasures. In: Bertoni, G., Coron, J.-S. (eds.) CHES 2013. LNCS, vol. 8086, pp. 293–310. Springer, Heidelberg (2013)

[CBR+15] De Cnudde, T., Bilgin, B., Reparaz, O., Nikov, V., Nikova, S.: Higher-order threshold implementation of the AES S-box. In: Homma, N., Medwed, M. (eds.) CARDIS 2015. LNCS, vol. 9514, pp. 259–272. Springer, Bochum (2015)

[CDG+13] Cooper, J., DeMulder, E., Goodwill, G., Jaffe, J., Kenworthy, G., Rohatgi, P.: Test Vector Leakage Assessment (TVLA) methodology in practice. In: International Cryptographic Module Conference (2013)

[CGPR08] Coron, J.-S., Giraud, C., Prouff, E., Rivain, M.: Attack and improvement of a secure S-Box calculation based on the Fourier Transform. In: Oswald, E., Rohatgi, P. (eds.) CHES 2008. LNCS, vol. 5154, pp. 1–14. Springer, Heidelberg (2008)

[CJRR99] Chari, S., Jutla, C.S., Rao, J.R., Rohatgi, P.: Towards sound approaches to counteract power-analysis attacks. In: Wiener, M. (ed.) CRYPTO 1999. LNCS, vol. 1666, pp. 398–412. Springer, Heidelberg (1999)

[CKN00] Coron, J.-S., Kocher, P.C., Naccache, D.: Statistics and secret leakage. In: Frankel, Y. (ed.) FC 2000. LNCS, vol. 1962, pp. 157–173. Springer, Heidelberg (2001)

[CNK04] Coron, J.-S., Naccache, D., Kocher, P.C.: Statistics and secret leakage. ACM Trans. Embed. Comput. Syst. **3**(3), 492–508 (2004)

[Cor14] Coron, J.-S.: Higher order masking of look-up tables. In: Nguyen, P.Q., Oswald, E. (eds.) EUROCRYPT 2014. LNCS, vol. 8441, pp. 441–458. Springer, Heidelberg (2014)

[CPR07] Coron, J.-S., Prouff, E., Rivain, M.: Side channel cryptanalysis of a higher order masking scheme. In: Paillier, P., Verbauwhede, I. (eds.) CHES 2007. LNCS, vol. 4727, pp. 28–44. Springer, Heidelberg (2007)

[CPRR13] Coron, J.-S., Prouff, E., Rivain, M., Roche, T.: Higher-order side channel security and mask refreshing. In: Moriai, S. (ed.) FSE 2013. LNCS, vol. 8424, pp. 410–424. Springer, Heidelberg (2014)

[DS15] Durvaux, F., Standaert, F.-X.: From improved leakage detection to the detection of points of interests in leakage traces. IACR Cryptology ePrint Archive, 2015:536 (2015)

[EWTS14] Eldib, H., Wang, C., Taha, M.M.I., Schaumont, P.: QMS: evaluating the side-channel resistance of masked software from source code. In: The 51st Annual Design Automation Conference 2014, DAC 2014, San Francisco, 1–5 June 2014, pp. 1–6. ACM (2014)

[GH15] Güneysu, T., Handschuh, H. (eds.): CHES 2015. LNCS, vol. 9293, pp. 517–534. Springer, Heidelberg (2015)

[GJJR11] Goodwill, G., Jun, B., Jaffe, J., Rohatgi, P.: A testing methodology for side channel resistance validation. In: NIST Non-invasive Attack Testing Workshop (2011)

[GP99] Goubin, L., Patarin, J.: DES and differential power analysis. In: Koç, Ç.K., Paar, C. (eds.) CHES 1999. LNCS, vol. 1717, pp. 158–172. Springer, Heidelberg (1999)

[ISW03] Ishai, Y., Sahai, A., Wagner, D.: Private circuits: securing hardware against probing attacks. In: Boneh, D. (ed.) CRYPTO 2003. LNCS, vol. 2729, pp. 463–481. Springer, Heidelberg (2003)

[KJJ99] Kocher, P.C., Jaffe, J., Jun, B.: Differential power analysis. In: Wiener, M. (ed.) CRYPTO 1999. LNCS, vol. 1666, pp. 388–397. Springer, Heidelberg (1999)

[Koc96] 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)

[P08] Pébay, P.: Formulas for robust, one-pass parallel computation of covariances and arbitrary-order statistical moments. Technical report SAND2008-6212, Sandia National Laboratory (2008)

[PGA06] Prouff, E., Giraud, C., Aumônier, S.: Provably secure S-Box implementation based on Fourier Transform. In: Goubin, L., Matsui, M. (eds.) CHES 2006. LNCS, vol. 4249, pp. 216–230. Springer, Heidelberg (2006)

[PRR14] Prouff, E., Rivain, M., Roche, T.: On the practical security of a leakage resilient masking scheme. In: Benaloh, J. (ed.) CT-RSA 2014. LNCS, vol. 8366, pp. 169–182. Springer, Heidelberg (2014)

[RBN+15] Reparaz, O., Bilgin, B., Nikova, S., Gierlichs, B., Verbauwhede, I.: Consolidating masking schemes. In: Gennaro, R., Robshaw, M. (eds.) CRYPTO 2015. LNCS, vol. 9216. Springer, Heidelberg (2015)

[RP10] Rivain, M., Prouff, E.: Provably secure higher-order masking of AES. In: Mangard, S., Standaert, F.-X. (eds.) CHES 2010. LNCS, vol. 6225, pp. 413–427. Springer, Heidelberg (2010)

[RRVV15] Reparaz, O., Roy, S.S., Vercauteren, F., Verbauwhede, I.: A masked ring-LWE implementation. In: Güneysu, T., Handschuh, H. [GH15], pp. 683–702

[SM15] Schneider, T., Moradi, A.: Leakage assessment methodology - a clear roadmap for side-channel evaluations. In: Güneysu, T., Handschuh, H. [GH15], pp. 495–513

[SP06] Schramm, K., Paar, C.: Higher order masking of the AES. In: Pointcheval, D. (ed.) CT-RSA 2006. LNCS, vol. 3860, pp. 208–225. Springer, Heidelberg (2006)

[Stu08] Student. The probable error of a mean. Biometrika, pp. 1–25 (1908)

[Wel47] Welch, B.L.: The generalization ofstudent's' problem when several different population variances are involved. Biometrika **34**, 28–35 (1947)