IACR Communications in Cryptology
ISSN 3006-5496, Vol. 1, No. 3, 19 pages.

https://doi.org/10.62056/ab89ksdja

Efficiently Detecting Masking Flaws in Software Implementations

Nima Mahdion[1] and Elisabeth Oswald[1,2]

[1] University of Klagenfurt, Digital Age Research Centre, Klagenfurt, Austria
[2] University of Birmingham, Computer Science, Birmingham, United Kingdom

Abstract. Software implementations of cryptographic algorithms often use masking schemes as a countermeasure against side channel attacks. A number of recent results show clearly the challenge of implementing masking schemes in such a way, that (unforeseen) micro-architectural effects do not cause masking flaws that undermine the intended security goal of an implementation. So far, utilising a higher-order version of the non-specific (fixed-vs-random) *input* test of the Test Vector Leakage Assessment (TVLA) framework has been the best option to identify such flaws. The drawbacks of this method are both its significant computation cost, as well as its inability to pinpoint which interaction of masking shares leads to the flaw. In this paper we propose a novel version, the fixed-vs-random *shares* test, to tackle both drawbacks. We explain our method and show its application to three case studies, where each time it outperforms its conventional TVLA counterpart. The drawback of our method is that it requires control over the shares, which, we argue, is practically feasible in the context of in-house evaluation and testing for software implementations.

Keywords: Side Channels · Leakage Assessment

1 Introduction

Secure implementations of cryptographic algorithms must consider defences against side channel adversaries. The canonical implementation strategy to provide (provable) security guarantees against side channel adversaries is masking: all sensitive intermediate values in an implementation are represented via multiple, statistically independent, shares. Practically evaluating the security of a concrete masked implementation can be done either via specific attacks (e.g. differential input attacks such as [Koc96, BCO04, CRR02]) or via a process called non-specific detection as part of a process called test vector leakage assessment (TVLA) [GGJR+11, SM15]. Both methods have advantages and disadvantages, and both are used in formal evaluation schemes [FIP19, Com17].

For an informal evaluation, which may be carried out by cryptographic software engineers during the implementation process, the speed and ease by which non-specific first-order leakage detection can be performed makes it a convenient tool for first-order leakage assessment. However, as soon as one wishes to determine the security level of an implementation against higher-order adversaries, or, if one wishes to determine if there are flaws in an implementation, even TVLA can become cumbersome. Recall that a masking scheme with d shares may (theoretically) offer provable security against an adversary who can exploit the o-th moment of a distribution ($o \leq d-1$) that they can generate from the measured traces. A flaw in a masking scheme refers to an unknown (and undesired) interaction between the shares of a sensitive intermediate, which could lower the envisioned security goal. An evaluation of such an adversary therefore must encompass processing

E-mail: m.e.oswald@bham.ac.uk (Elisabeth Oswald)

Received: 2024-07-08 Accepted: 2024-09-02

traces in such ways prior to launching non-specific detection. This processing is time consuming and the non-specific detection thereafter requires considerably more traces to return statistically significant results than the corresponding first-order testing [SM15, DS16a, DCE16, Sta18, MRSS18, ZQO19, BSS19, MWM21, YJ21, WTW+22].

1.1 The Pain of Evaluating Software Implementations

Even moderately complex processors, such as mid market processors based on the popular ARM Cortex M architecture [ARMa], create considerable challenge for cryptographic software implementations due to a range of different micro-architectural effects [MPW22]. These effects are known to be different for processors even if they are part of the same processor family from one vendor, as impressively demonstrated in [MPW22]. Consequently, an implementation that may be secure on one processor could be insecure on another processor (even when both are from the same family of processors by the same vendor). This implies that an implementation has to be tested on each processor, and if fails the desired security level, it may fail for different reasons.

The process of higher-order TVLA helps to identify if an implementation does not reach its desired security goal. However, its use to detect differences in higher order moments leads to an exponential (in the tested moment) increase in the number of traces, and one does not learn which shares are responsible for the problem: this makes finding and resolving the problem difficult.

Many papers have been written that consider (higher order) statistical aspects of the TVLA process, such as [MOBW13, SM15, DS16b, Sta18, DZD+17], from an efficiency point of view. Our work complements these by determining what leaks (i.e. we aim to identify the combination of shares that causes the problem).

1.2 This Work

Contributions. Given an implementation with d shares, we present a novel process that determines the combinations of shares that cause masking flaws. A masking flaw is due to an unintended combination of shares (often due to unknown micro-architectural properties of a processor), and it leads to not meeting the security goal of an implementation. Our process makes the assumption that in an "in-house" evaluation of cryptographic software, it is possible to control the values that shares take. Our process then re-interprets the "fixed vs. random" test paradigm of TVLA by fixing (some of) the shares and randomly choosing the rest. It turns out that this approach requires fewer traces to identify flaws than the application of a conventional higher-order TVLA. By combining knowledge about the implementation, and the time point of the identified flaw, we can, using the information about the fixed shares, relatively easily identify what causes the flaw.

We then show how to apply our process to three multiplication gadgets (ISW, DOM-indep, HPC-1) [ISW03, GMK16, GMK17, CGLS21] when implemented on a typical mid range processor. The gadgets satisfy two different security notions: t-probing and glitch-robust probing. We show that even for the gadgets that satisfy the (rather strong) glitch-robust probing definition, micro-architectural events cause them to fail at an order lower than the proven order. We relate the identified problem to the Assembly source code, which goes some way to fix the problematic piece of code.

All examples are available via `https://github.com/sca-research/SEAL_Gadgets`; a public repository that includes Assembly sources to gadgets, analysis scripts, and a link to the traces that we acquired and used in our analyses.

Limitations. The clear limitation of our approach is that it requires control over the choice of the shares: we argue that in a software implementation this is always possible (e.g. by manipulation via the software interface). In a dedicated hardware implementation

this should also be possible when simulating power traces directly from the hardware description, but it may not be possible when working post-silicon without direct access to the randomness.

Outline. We structure this paper as follows. In Sec. 2 we introduce notation and review the necessary facts around tests of distribution means as well as trace complexity estimates. In Sec. 3 we explain our new process. In Sec. 4 to Sec. 6 we give the three case studies.

2 Preliminaries

We briefly set out the notation that we use throughout this paper, and then review the necessary facts around testing distribution means, as well as the principle underlying the process of non-specific leakage assessment.

2.1 Notation

Leakage traces are vectors over real numbers, and all the tests that we consider in this paper treat the vector elements independently, i.e. we only consider univariate statistics. The random variable t refers to a trace point, which can also be understood to be a random variable, and any index to t (e.g. t_F, or t_R) further qualifies this random variable. The exact meaning of the qualifier will be clear from the text, but we reserve the letters F and R to refer to a "fixed input" and "random input" (i.e. t_F implies that the random variable relates to a measurement with fixed input). Because the distribution of t is typically unknown, we can only work with samples drawn from t, and unless stated otherwise, we assume that we get n such samples. We refer to the probability of incorrectly identifying data dependency with the letter α (the false positive rate), and we refer to the probability of incorrectly failing to identify data dependency via β (the rate of false negatives).

A natural number x can be shared into $d+1$ (Boolean) shares: $x = x_0 \oplus x_1 \oplus \cdots \oplus x_d$. Considering the $d+1$ shares of x as a set, the power set $\mathcal{P}(x)$ is the set of all subsets of of shares. For instance if $x = x_0 \oplus x_1$ then the power set for the set of shares is $\{x_0, x_1\}$ is $\mathcal{P}(x) = \{\{\}, \{x_0\}, \{x_1\}, \{x_0, x_1\}\}$.

2.2 Experimental Setup

In our experiments, we utilise software implementations of three different multiplication ($c = a \cdot b$) algorithms (gadgets), which are based on using 3 shares (we provide brief background for each gadget in the respective section later in this article).

The target device is a 32-bit ARM Cortex-M3 microprocessor located on a SCALE board [Pag] running at 2 MHz. For convenience, all input shares (including shares of a, b and randomnesses) are generated externally and then sent to the device. The information leakage from this data transfer is not utilised in our analysis, i.e. all our traces refer strictly to the multiplication gadgets processing only. The acquisition is performed using a Picoscope 5243D digital oscilloscope sampling at 250 MS/s where each cycle is sampled with 125 points.

Due to our target platform, all implementations are using Thumb-16 instructions [ARMb]. The implementations proceed sequentially, operating on byte-wise data, including inputs, intermediate values and outputs over the field $\mathbb{F}_{2^n}$. The key operation in these implementations is the Galois field multiplication for multiplication of shares (e.g. $a_1 \cdot b_2$).

Over $\mathbb{F}_2$, the multiplication of shares (e.g. $a_1 \cdot b_2$) is easy to implement. However, over $\mathbb{F}_{2^n}$, an efficient software implementation on a commodity process often requires some trickery because no dedicated modular multiplication is available. As comprehensively investigated in [GR17], a common implementation choice is to add the (base 2) logarithm

of the numbers: $a \cdot b = 2^{log(a)+log(b)}$, which in turn requires to compute their logarithm and anti-logarithm. This can be done easily by storing and using a logarithm and anti-logarithm table.

We used this implementation option for all gadgets. Thus each time a multiplication needs to be performed, the implementation converts both operands for the multiplication via the logarithm table, it then performs an addition over the respective finite field, and then converts the result back via the antilogarithm table.

In our assembly implementation, the logarithm and antilogarithm tables are stored in memory starting from an address that is kept in the internal register $r11$ (we move this into a lower register as and when needed). The shares are located in memory as well and are retrieved from their respective addresses as needed.

We compile the source codes using GNU Arm Toolchain (arm-none-eabi-gcc 10.3-2021.10) avoiding compiler optimisations by utilising the `-Os` flag for size efficiency in the C code section during compilation.

2.3 Testing Distribution Means

All tests in this paper are based on the following two-sided hypothesis test for the (difference of) means of two normal distributions $t_1 - t_2 \sim \mathcal{N}(\mu = \mu_1 - \mu_2, \sigma)$, where μ_1, μ_2 and σ are unknown parameters:

$$H_0 : \mu = \mu_0 \text{ vs. } H_{alt} : \mu \neq \mu_0. \tag{1}$$

In the leakage detection setting we are interested to test for a non-zero difference and thus we set $\mu_0 = 0$.

The canonical test for distribution means, is the Student t-test [Stu08], which assumes equal sample sizes and population variances (as implied by our explanation before). One can determine the necessary sample size to achieve some desired combination of α, β (and some fixed/implied difference between the means $\mu_1 - \mu_2$) via:

$$n \geq 2 \cdot \frac{(z_{\alpha/2} + z_\beta)^2 \cdot ({s_1}^2 + {s_2}^2)}{(\mu_1 - \mu_2)^2}, \tag{2}$$

(s_i are the unbiased estimators of the population variance), this formula can be found in any standard statistics textbook). In the case of non-specific leakage assessment, which is what we will consider next, the situation is slightly different: here we have to deal with sets of unequal variance. As a (non-trivial) consequence, there is no explicit formula for the sample size available. Instead, sample sizes must be derived via an iterative numerical process, which however starts with the application of Eq.(2), see [JS11]. Consequently, Eq.(2) is an approximation of the true sample size, and we will use it to argue why the method that we suggest in this paper is more trace efficient than previous work.

2.4 Non-specific Leakage Assessment (as defined by TVLA)

The TVLA framework was presented at the 2011 Non-Invasive Attack Testing workshop organized by NIST [GGJR+11]. It suggests using statistical hypothesis testing to reject (or not) the null hypothesis of "no data dependency" against the alternative hypothesis ("data dependency"). The core idea for the test, which is called non-specific leakage assessment, works as follows:

- An acquisition of size n is taken as the device operates with a fixed key on a fixed plaintext chosen to induce certain values in one of the middle rounds of a given encryption scheme. At least two different fixed inputs should be used.

- An acquisition of size n is taken as the device operates with the same fixed key on random inputs.
- Welch's t-tests [Wel47] are performed, setting $\alpha \approx 0.00001$, comparing the population means of each fixed input set with a random input set.

  Thus we test the hypotheses that the mean of the fixed trace set is equal to the mean of the random trace set (akin to "no data dependency"):

$$H_0 : \mu_F = \mu_R \text{ vs. } H_{alt} : \mu_F \neq \mu_R. \tag{3}$$

One concludes that there is data dependency if both (or all) tests reject the null hypothesis ("no data dependency") in the same data points.

2.4.1 Extension to Masked Implementations

Practical implementations of cryptographic algorithms typically utilise some secret sharing technique to provide provable security guarantees in the presence of side channel leakage. This implies that all sensitive intermediate values are represented by $d + 1$ shares, and any concrete masking scheme specifies an algorithm that ensures (at least at the level of the mathematical description) that shares are only processed in ways that joint leakage between shares of the same variable are avoided. The security proof of a masking scheme then makes clear up to which order (i.e. number of shares considered jointly) the scheme should be secure. Flaws in practical implementations arise when shares are manipulated concurrently in an implementation, which may lead to multiple shares leaking jointly in a trace point. In such cases, an implementation will permit an attack below the security order that is stated by the proof.

TLVA can be used to test concrete implementations of masking schemes, and to find such flaws. An in-depth explanation of how to use TVLA to assess masked implementations can be found in [SM15]. We informally summarise the method that is relevant for detecting typical flaws in software implementations. Assume an implementation with $d + 1$ shares, that theoretically supports security of up to order d. We wish to test for flaws, i.e. combinations of up to d trace points where the non-specific detection described previously indicates information leakage. We must thus consider all combinations of up to d trace points. We start from a sanity check at order $d = 1$ (using TVLA as described in the previous section with the Welch t-test) and then progress by increasing d and applying TVLA to all combinations of d points[1].

This process is very costly, not only because all d combinations of trace points have to be produced (leading to very long traces), but also because the pre-processed trace now consist of points with a much higher variance than the original traces (e.g. if pre-processing is based on multiplying trace points, then the newly create points have their variance multiplied too). The fact that the variance increases significantly increases the number of traces that is needed for the test to reach statistically significantly results (i.e. to have the same α and β as in the case without pre-processing) — this fact is clear from Eq. 2 (the variance of both trace sets is the dominant factor in determining the sample size).

2.4.2 Related previous work

The Welch t-test has several configuration options that are relevant for the application in detecting leakage.

Relating to the trace efficiency (i.e. n in Eq.(2), is the choice of the two groups for the test. According to the TVLA specification, one group corresponds to a fixed input value, and one group corresponds to randomly chosen inputs. This may look like an odd choice

[1] The canonical combination function is the mean-free product, see [SM15].

for the two groups: after all, why not choose two fixed (but different) inputs? The reason is that it is possible (or even likely) that two (out of many) randomly chosen inputs may not lead to two different distributions. Thus if we were only to pick two inputs at random, we might incorrectly conclude that there is no leakage. The workaround then is to either choose many pairs (and hope that at least one pair can indicate leakage) or, to allow the input in one group to vary (and hope that the mean across many fixed inputs is different to the mean of any individual input).

However, the fixed-vs-fixed test was in fact advocated in a later paper [DS16a]. Considering Eq.(2) we can motivate why a fixed-vs-fixed approach might be appealing: by either lowering the numerator (i.e. reducing the group variances) or increasing the denominator (i.e. increasing the distance between the group's means) we can lower n (the number of necessary traces). By fixing both inputs, we can ensure a low variance in both groups (as we fix the inputs, there can be no data dependent variance, in contrast to allowing one input to vary). By carefully choosing both inputs, we may be able to increase the difference in means. This typically requires some knowledge about the leakage behaviour of the device. For instance, if one expects to detect leakage from data transfer operations, then often these leaks are due to the Hamming weight of operands. Thus, if we suspect leakage in a specific intermediate value, we could choose the two groups for the detection test that maximise the Hamming distance between the groups.

The fixed-vs-fixed test was subsequently selected as the method of choice in [Rep16] to detect flaws in masking schemes in an efficient manner. The focus of this work was to optimise the existing TVLA setup, both in terms of trace efficiency as well as computational efficiency. Complementary to these previous works, our paper here focuses on identifying which shares contribute to identified leakage. In other words, we aim to increase the explainability of detection outcomes in a way that helps fixing implementation problems. Our method adds to the research contributions towards "explainability" in the sense that an implementer can narrow down the origin of leakage in a piece of code. This enables easier fixing of the problem on the code level, and it may help explain the problem. However, a full explanation of a leak in software will require detailed information about the micro-architecture of a device, which typically is not available to the software designer. Thus, there is a limit to the depth of the explanation in this situation. Coincidentally, our method also increases the trace efficiency of the method.

3 Fixed vs. Random Subset of Shares Test

As explained before, detecting an o-th order flaw in a masking scheme with d shares requires pre-processing of traces when using fixed-vs-random input TVLA. This significantly increases the computational effort, both in terms of the number of traces to have robust statistics as well as in the length of the traces that one must work with. Besides the computational drawback, we do not learn which shares contribute to the identified flaw with the identified time points.

To overcome both problems (computational overhead and lack of "hints" to fix the problem), we now suggest to take full advantage of the white-box situation in an in-house evaluation: when working with software it is typically easily possible to take control over how shares are being sampled. With this capability, a natural approach would then be to reconsider the fixed-vs-random *input* principle, and turning it into a fixed-vs-random *shares* approach.

3.1 Fixed-vs-Random Shares Test

Concretely, consider an implementation with d shares, for which we wish to identify all points with an o-th order flaw, *i.e.* points where o shares leak jointly. This implies, that we

do not make any assumption regarding the (non)-existence of first-order leakage, instead, thus we also test $o = 1$ in our approach. With control over the selection of shares, we can do this elegantly by repeating the following (for all desired share combinations):

Fixed: Select a subset of shares of cardinality o, *i.e.* $y \in \mathcal{P}(x)$ with $|y| = o$, and fix the values of the shares in this subset. Select the remaining shares at random.

Random: Select all shares at random.

$$H_0 : \mu_{F_y} = \mu_R \text{ vs. } H_{alt} : \mu_{F_y} \neq \mu_R. \tag{4}$$

This configuration now fixes o out of d shares and we can instantiate with it a first-order test for distribution means. The test will return trace points that depend on the joint distribution of o shares, without the need for any pre-processing. Consequently, we confirm what shares contribute to the data dependency in a trace point without the need for increasing the number of traces and without increasing the length of traces (at the expense of controlling the share generation).

3.2 Fixed-vs-Random Shares Test to Identify Transitional Leakage

We can refine the basic idea and tailor it specifically to identify transitional leakage, which has been reported as a source of many real-world implementation flaws in previous work [MPW22, BGG+14]. Fixing and mitigating transitional leakage is therefore a particularly relevant goal in software implementations. Transitional leakage can often be relatively easily mitigated, either by swapping Assembly instructions, or, by interleaving Assembly instructions with instructions operating on independent data.

In the context of real-world software implementation, a likely implementation choice for cryptographic software that needs to support the common AES standard [NIS01], is to utilise a Boolean masking scheme with three shares. This means that any sensitive intermediate value x is represented by the three shares x_0, x_1, x_2 (and $x = x_0 \oplus x_1 \oplus x_2$). The elements of the the power set of x with two shares are thus $\{\{x_0, x_1\}, \{x_0, x_2\}, \{x_1, x_2\}\}$. The refinement to identify transitional leakage in such a three share scenario works as follows:

Fixed: Select a subset of shares of cardinality 2, *i.e.* $y \in \{\{x_0, x_1\}, \{x_0, x_2\}, \{x_1, x_2\}\}$, and fix the exclusive-or sum of the values of the shares in this subset. Select the remaining shares at random.

Random: Select all shares at random.

$$H_0 : \mu_{F_y} = \mu_R \text{ vs. } H_{alt} : \mu_{F_y} \neq \mu_R. \tag{5}$$

We note that this adaptation has the advantage over the generic o-th order test that we now only fix the sum over o shares, rather than the o shares themselves, which enables us to sample more traces for the fixed class.

3.3 Computational Cost

Our test configuration works directly on the side channel traces. Consequently, the number of side channel traces to reach a specific rate of false positives and negatives (for the distance between the sets that is implied by the fixed vs random setup) can be approximated by Eq.(2). It is proportional to the variances of the traces.

In contrast, let us consider how we could detect such points with the classical fixed-vs-random input setup. Here, we cannot fix shares, but only the inputs, which implies

that we must pre-process traces to exhibit joint leakage within a trace point. Let us consider two trace points: one point jointly leaks (x_i, x_j) and one trace point leaks x_k (with $i, j, k \in \{0, 1, 2\}$). Two determine both points, we need to apply the "mean-free product" preprocessing method [PRB09]. Multiplying two trace points implies that (at least) we need to consider their product distribution for the number of necessary traces calculation. The distribution of the product of two independent normal distributions is well understood, and at least approximately, the number of traces is then again given by Eq.(2). The variance of the product distribution increases exponentially by a factor of 2, implying a significant increase in the number of traces via Eq.(2).

Clearly then, if it is possible to select shares, then our fixed-vs-random shares test is computationally significantly cheaper than the corresponding classical fixed-vs-random input test with preprocessing.

3.4 Extension to Arbitrary Masking Orders

We can apply our methodology to evaluate any arbitrary masking scheme with $d+1$ shares by considering all combinations of m shares ($1 \leq m \leq d$) from the $d+1$ shares of x. For an exhaustive evaluation the total number of tests is:

$$\sum_{m=1}^{m=d} \binom{d+1}{m} = 2^{d+1} - 2. \tag{6}$$

The previous discussion of the trace efficiency still applies to all of these individual tests: each test has a trace efficiency that directly depends on the variance of the unprocessed traces, thus each test remains trace efficient (but the number of tests grows exponentially).

In practical software implementations, we would expect that flaws mostly occur due the joint interaction of two or three shares because of the limited complexity of embedded processors (often featuring only a three stage pipeline, limited or no out-of-order execution). Furthermore in practice, a masking scheme may only be required to give order o security with o being relatively lower than $d+1$. Thus we conjecture that an exhaustive testing of all share combinations is not necessary in practice. When considering joint leakage of $o < d+1$ shares, then our method is particularly appealing, because the trace complexity is bounded by (order of) $\binom{d+1}{o}$ tests (whereby each test requiring n traces).

3.5 Application Across Different Devices

We mentioned before the pain of evaluating software implementations, because the same implementation may be secure one device, but insecure on another. Thus, implementations are typically not portable, but statistical analyses can be. Our method is applicable to all devices that feature certain statistical characteristics that are assumed for TVLA-like tests: these boil down to having additive independent Gaussian noise on trace points.

(1) In this paper we position our method as particularly suitable for software implementations on typically embedded processors. This point of view is driven by the requirement of our method to "control" the randomness of the shares. From our understanding, which is based on some interaction with actual evaluators, access to and control over randomness can be available in both software and hardware implementations.

In software implementations, if share generation takes place as part of the masking scheme (by sampling randomness), then, we argue, it should always be possible to craft this shares' generation code according to the requirements of our testing method. This piece of code is typically executed at the start of the encryption method, and thus its influence on leakage is typically isolated from the masked encryption. Refreshing and remasking are typically also separated into specific functions and thus also their impact on the leakage of the masked encryption should be minimal.

In hardware implementations, the initial generation of shares also takes place at the beginning of the execution, but it may be that share refreshing happens in parallel to the masked round functions. We did not consider the impact of such a situation in our work so far. If a hardware implementation is considered with limited parallelism, then control over the share generation and share refreshing should enable the application of our method as described in this paper.

4 Case Study: 3-share ISW

The first masked multiplication using an arbitrary number of shares was described by Ishai *et al.* [ISW03] over the finite field $\mathbb{F}_2$, providing security for up to order $d/2$ when using $d+1$ shares. We shall use the shortcut ISW to refer to their multiplication. Later on, Rivain and Prouff [RP10] showed that the same construction also applies when working over $\mathbb{F}_{2^n}$, and that order d security could be reached when using $d+1$ shares. For convenience, we provide a high-level description of the multiplication (with $d+1$ shares) in Alg. 1.

Algorithm 1 ISW multiplication gadget

Require: shares a_i, b_i $(0 \leq i \leq d)$ such as $a = \oplus_{i=0}^{d} a_i$, $b = \oplus_{i=0}^{d} b_i$
Ensure: $c_0, ..., c_d$ such that $c = \oplus_{i=0}^{d} c_i = a \cdot b$

```
for i = 0 to d do
    for j = i + 1 to d do
        r_ij  <-$  {0,1}^n
        r_ji <- [r_ij ⊕ (a_i · b_j)] ⊕ (a_j · b_i)
    end for
end for
for i = 0 to d do
    c_i <- (a_i · b_i)
    for j = 0 to d do
        if i ≠ j then
            c_i <- c_i ⊕ r_ij
        end if
    end for
end for
return c_0, ..., c_d
```

One can see that in Alg. 1, one proceeds by creating the r_{ij} and r_{ji} values, and thereafter they are exclusive-ored to the output shares: $c_i = a_i \cdot b_i \oplus r_{ij}$. This invites an implementation where the c_i are initialized as $a_i \cdot b_i$ and then, the necessary r_{ij} are produced and exclusive-ored to the c_i. The proof for the construction is agnostic to this kind of re-ordering.

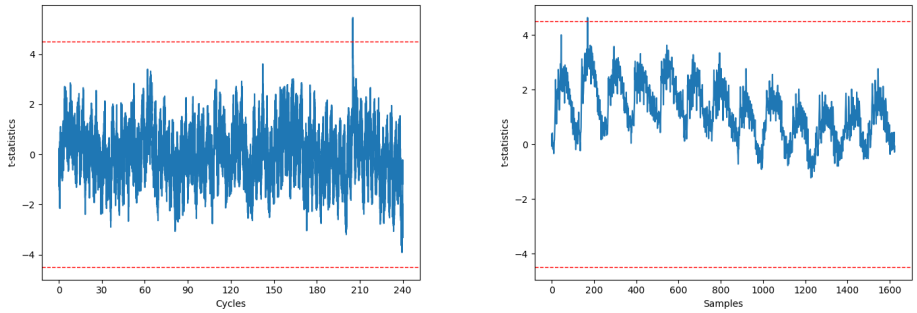
4.1 Detecting a Flaw

Recall that we specifically look for transitional leakage. The combination of two shares of a happens only during a specific part of the multiplication algorithm. Thereafter, this result is stored and re-used, but the leakage from having e.g. a_1 and a_2 simultaneously in the micro-architecture is no longer present. Consequently, the subsequent explanation and figures only show the clock cycles up to the relevant event.

We applied the fixed-vs-random shares test to this implementation. We found that when we fix the exclusive-or of second and third share of a $(a_1 \oplus a_2)$, we find that, using

3000 traces, the t-test value crosses our decision boundary in clock cycle 205 (see left plot in Fig. 1). This indicates that using just 3000 traces, there is enough evidence to conclude that the distributions are distinguishable. Relating the offending clock cycle to our Assembly code indicates that in clock cycle 205, the intermediate $a_2 \cdot b_2$ is loaded (i.e. the share c_2 is being calculated).

With this information we can set up an analogous, and confirmatory, analysis with the fixed-vs-random input test for masked implementations: we consider combinations of other traces points with the trace point at cycle 205 (i.e. we perform a second-order analysis). For ease of visibility we include a plot that relates only to a part of the analysis, see the right panel of Fig. 1. This right panel shows that at the point 170 (the combination of point 41 at cycle 151 and point 45 at cycle 205) the decision threshold is crossed, which leads to the conclusion that there is information leakage at order 2. In contrast to the fixed-vs-random shares test, we now need 1.6 million traces.



(a) Fixed-vs-random shares test on a_1 and a_2, 3K traces, leakage at cycle 205

(b) Second-order multi-variate t-test (1.6M traces), combining cycles 151 and 205

Figure 1: Analysis of ISW 3-shares

4.2 Diagnosing the problem

We already know that at clock cycle 205 the value $a_2 \cdot b_2$ is loaded. The Assembly implementation is faithful to the high level description, thus it is likely that some micro-architectural effect is causing the unintended information leakage. We thus consider what happens in the clock cycles around, before, and after clock cycle 205. We show this by including the relevant snippet from our Assembly implementation (for the sake of readability we remove all the irrelevant lines of code) below:

```
194 ldrb r5, [r4, r2]    @ r5 = a1*b2
      [...]          (no memory instructions between the loads)
205 ldrb r5, [r2, #8]    @ r5 = a2*b2
```

Listing 1: ISW 3-shares

The snippet shows that the nearest use of the register $r5$ in the code occurs in clock cycle 194, when the register $r5$ contains the value of $a_1 \cdot b_2$.

Let us consider why this happens. The implementation computes $a_1 \cdot b_2$ in cycle 194, then exclusive-ors the random value r_{12} (as per the algorithm), and then proceeds to compute $a_2 \cdot b_1$. Because the algorithm requires to add $a_2 \cdot b_1$ and then immediately add $a_2 \cdot b_2$, the implementation loads $a_2 \cdot b_2$ right after adding the mask r_{12} to $a_1 \cdot b_2$. Whilst this does not invalidate the proof, it does create the opportunity for micro-architectural leakage between $a_1 \cdot b_2$ and $a_2 \cdot b_2$ even though the random r_{12} was added in the mean time. This is because the implementation does not require any instruction between clock cycle

194 and 205 that uses the memory bus, which means that on this bus we have the values $a_1 \cdot b_2$ and $a_2 \cdot b_2$ consecutively, leading to well known transition leakage $b_2 \cdot (a_1 \oplus a_2)$ in clock cycle 205. This transition leakage can reveal information about a when combined with another trace point that leaks on a_0, which clearly must at some point happen in the implementation. The implementation thus is not secure at order 2 (which was its intended security goal).

5 Case Study: 3-share DOM-independent

Domain-Oriented Masking (DOM) takes the potential for glitches, i.e. uncontrolled transitions in (micro-)architectural processor elements, into account in its design. The idea of DOM is built around the concept of share domains, where each share of an input is exclusively associated with one domain. Additionally, in DOM, the multiplications of shares is divided into inner-domain terms, which multiply shares within the same domain $a_i \cdot b_i$, and cross-domain terms that use shares from different domains such as $a_i \cdot b_j$ and $a_j \cdot b_i$.

There are two types of DOM: DOM-dependent (DOM-dep) and DOM-independent (DOM-indep). Unlike DOM-dep which requires specific management of dependent input shares, DOM-indep offers d order security with only $d + 1$ shares. This substantially reducing the number of required input shares and the randomness needed compared to approaches such as Threshold Implementations (TI) [NRR06] and Consolidated Masking Scheme (CMS) [RBN+15]. We implemented DOM-indep as shown in Alg. 2.

As it can be seen in Alg. 2, the initial computation for each domain's output c_i begins with the calculation of the inner-domain terms, $u_{ii} = a_i \cdot b_i$, which are first assigned to c_i. Subsequently, as the algorithm progresses, fresh random values $r_{(i+j(j-1)/2)}$ are generated and exclusive-ored with the cross-domain product terms to ensure that they are statistically independent. These values are stored in variables u_{ij} and u_{ji} to avoid glitches, and then these randomized cross-domain terms are exclusive-ored into the c_i.

Algorithm 2 DOM-independent multiplication gadget

Require: shares a_i, b_i $(0 \le i \le d)$ such as $a = \oplus_{i=0}^{d} a_i$, $b = \oplus_{i=0}^{d} b_i$
Ensure: $c_0, ..., c_d$ such that $c = \oplus_{i=0}^{d} c_i = a \cdot b$

1: **for** $i = 0$ **to** d **do**
2: $u_{ii} \leftarrow (a_i \cdot b_i)$
3: **for** $j = i + 1$ **to** d **do**
4: $r_{(i+j(j-1)/2)} \xleftarrow{\$} \{0,1\}^n$
5: $u_{ij} \leftarrow (a_i \cdot b_j) \oplus r_{(i+j(j-1)/2)}$
6: $u_{ji} \leftarrow (a_j \cdot b_i) \oplus r_{(i+j(j-1)/2)}$
7: **end for**
8: **end for**
9: **for** $i = 0$ **to** d **do**
10: $c_i \leftarrow u_{i0}$
11: **for** $j = 1$ **to** d **do**
12: $c_i \leftarrow c_i \oplus u_{ij}$
13: **end for**
14: **end for**
15: **return** $c_0, ..., c_d$

5.1 Detecting a Flaw

We now apply the fixed-vs-random shares method on the DOM-indep implementation, which exhibits a flaw using 12,000 traces. The flaw appears when we fix the exclusive-or $a_0 \oplus a_2$. The flaw occurs in clock cycles 70 and 71, where t-test values exceeded our predefined threshold, as illustrated in the left plot in Fig. 2. Further investigation of the Assembly code for these specific cycles reveals that these relate to operations involving calculating the negative of a_2 and assigning the value 32 to register $r6$.

We use our fixed-vs-random shares test result to facilitate a confirmatory standard fixed-vs-random input second-order analysis (i.e. we only pre-process with cycle 70), and show the result as the right-hand side of Fig. 2. We identify leakage also in this experiment at point 1191, which represents the interaction between cycle 144 (point 46) and cycle 70 (point 66). However, we require 1.1 million traces this time.

5.2 Diagnosing the problem

We have identified clock cycles 70 and 71 as critical cycles in our Assembly implementation. The Assembly code accurately reflects the intended algorithm, yet, the operations in these cycles unintentionally create a pathway for information leakage due to unforeseen micro-architectural interactions. To understand this phenomenon, we will examine the relevant assembly code extract listed as follows:

```
# r4 = a0
# r6 = a2

70 negs r1, r6     @ r1 = -a2
71 movs r6, #32    @ r6 = 32
```

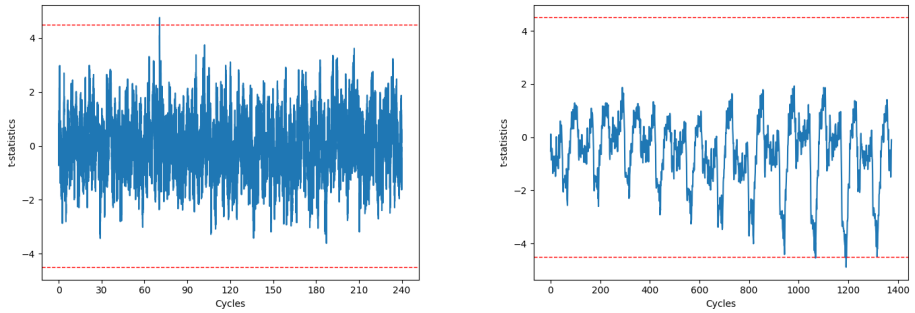
Listing 2: DOM-indep 3-shares

This code reveals a subtle yet significant issue, namely transitional leakage based on a glitchy-register-access, as described in [GOP22]. The root of the problem lies in the instruction decoding process, which produces unfavourable micro-architectural behaviour. To be precise, as the decoder receives the `movs` instruction (i.e. the bit-string that encodes the type of instruction, as well as the addresses or immediate value), the decoding processes the different fields at different speeds. Thus it is possible that part of decoder is still configured from the preceedings `negs` instruction, which has two registers as operands. If then the `movs` instruction is decoded, it so happens that for a brief moment its second operand, which is the immediate value `32`, is interpreted as a register, which according to the Thumb16 instruction format, gives `r4`. Thus for a brief moment, just after `negs` accessed $r6$ via port 2 of the decoder, it now accesses $r4$ also on port 2 of the decoder. This leads to the observable transition leakage $a_0 \oplus a_2$.

The fact that our fixed-vs-random shares test found leakage for $a_0 \oplus a_2$, together with the time point (=clock cycle) when the problem occurs, is very helpful to identify what causes the flaw in this example. Evidently, at clock cycle 70 the code accesses a_2. It is much more subtle to see that the immediate value 32, which is used in clock cycle 71 is interpreted as `r4` and therefore accesses a_0. We were able to make this connection relatively easily because we knew what to look for (i.e. the occurrence of $a_0 \oplus a_2$ due to our novel assessment strategy).

6 Case Study: 3-share HPC1

Hardware Private Circuits (HPC) multiplication, introduced by Cassiers et al. in [CGLS21], represents another glitch-robust, higher-order secure multiplication gadget. The authors initially proposed two multiplication gadgets, HPC1 and HPC2, whereby HPC1 operates



(a) Fixed-vs-random shares test on a_0 and a_2, 12K traces, leakage at cycle 70

(b) Second-order multi-variate t-test (1.1M traces), combining cycles 70 and 144

Figure 2: Analysis of DOM-indep 3-shares

over $\mathbb{F}_{2^n}$. The fact that HPC1 can be seen as an updated version of the popular DOM gadget is the reason for its inclusion in our study. We focus on HPC1 in our analysis. Subsequent improvements have further optimized the trade-off between efficiency and security of this family [KM22, FGM+23]. HPC1 is essentially a DOM-independent multiplication gadget with one main modification, wherein one of its inputs undergoes a refresh operation before the multiplication. This refreshing step is important to achieving the desired security properties. The structure of HPC1 for multiplying $a.b$, including the refreshing step which is based on exclusive-or of shares of b with $d+1$ randomness, is illustrated in Fig. 3.

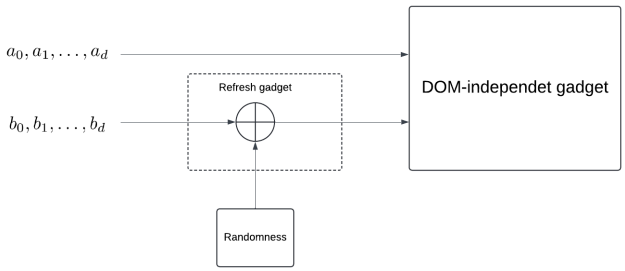
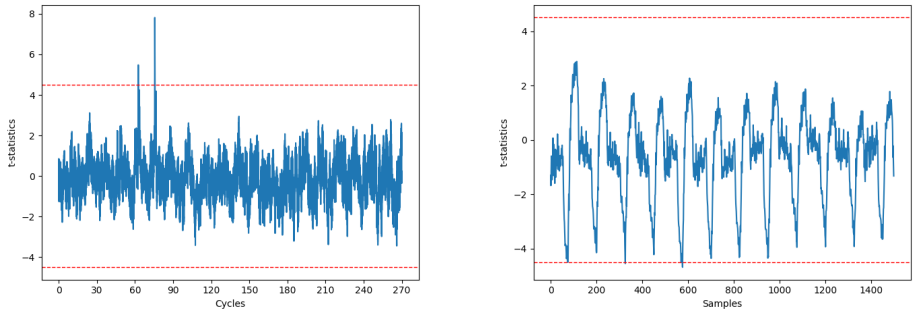


Figure 3: HPC1

6.1 Detecting a Flaw

Using the fixed-vs-random shares method we were again able to identify a flaw: this time when the exclusive-or of the first two shares of a ($a_0 \oplus a_1$) is kept constant, see the left plot in Fig. 4. In fact, there are two flaws clearly visible, one around cycle 62 and one around cycle 75.

Like in the previous case studies, we utilise this information to facilitate a second-order analysis via TVLA (fixed-vs-random input) by examining the interaction of trace points in cycle 62 with other points. The plot on the right side of Fig. 4 shows the result; the threshold is crossed at point 576, indicating leakage. This time we require 1.1 million traces for the conventional TVLA analysis, whereas our fixed-vs-random shares test only requires 15k traces.



(a) Fixed-vs-random shares test on a_0 and a_1, 15K traces, leakage at cycle 62

(b) Second-order multi-variate t-test (1.1M traces), combining cycles 62 and 224

Figure 4: Analysis of HPC1 3-shares

6.2 Diagnosing the problem

By examining the source code for these cycles, we found that the cause of the leakage at cycle 62 is the instructions related to clearing the write bus and loading a_1's logarithm into register $r5$. The leakage at cycle 75 stems from a value assignment operation (see Sec. 5.2).

The flaw in cycle 75 is linked to a glitchy-register-access during processing instruction `movs r6, #32`, which we encountered and analysed in the previous case study, see Sec. 5.2. Hence, our focus in this case study is to understand the flaw detected at cycle 62. For this purpose we include the following assembly snippet:

```
# r6 = a1

62 strb r0, [r0, #4]    @ clearing the write bus
63 ldrb r5, [r3, r6]    @ r5 = table[a1]
     [...]
75 movs r6, #32         @ r6 = a1, r4 = a0
```

Listing 3: HPC1 3-shares

This code section highlights a hidden subtle but critical problem——a transitional leakage based on glitchy-address-signals in the multiplexer tree, which is used to access the register file. For the sake of completeness, we briefly explain it's working principle, using Fig. 5 as a visual aid[2]. When an instruction requires access to a specific register, then the binary string representing the instruction is decoded and some of the bits select (see Fig. 5) which register is being accessed.

The leakage originates from a delay in switching the value of the address signal in port 2, where the second operands of instructions `strb r0, [r0, #4]` and `ldrb r5, [r3, r6]`, $r0$ and $r3$ are loaded sequentially into port 2. The transition from $r0$ to $r3$ necessitates changing the address signal of the multiplexer tree from 000 to 110.

When the address signal is 000 to select register $r0$, the values on wires a to g are as follow:

$a : r0$, $b : r2$, $c : r4$, $d : r6$, $e : r0$, $f : r4$ and $g : r0$.

However, during the transition from 000 to 110, there is no guarantee that all address signal bits change simultaneously. If the value of Bit 1 updates faster than Bit 0, it can cause a temporary state. Here the outputs of MUX 1 (wires a to d) remain unchanged and

[2]The multiplexer tree handles 16 registers in the register file and has a depth of four, with each level's selection controlled by a Bit of the signal address ADD. However, since the implementations are in Thumb-16, we consider only the lower registers $r0$ to $r7$, using three levels.

the transition of Bit 1: $0 \rightarrow 1$ causes $r6$ from wire d briefly appear on wire f (and same $r0$ on wire e). This results in, both $r4$ and $r6$ being present on wire f. Since $r4$ holds a_0 and $r6$ contains a_1, there is a leakage of $a_0 \oplus a_1$.

This flaw undermines the security of the implementation at a order two, which was its intended security goal.

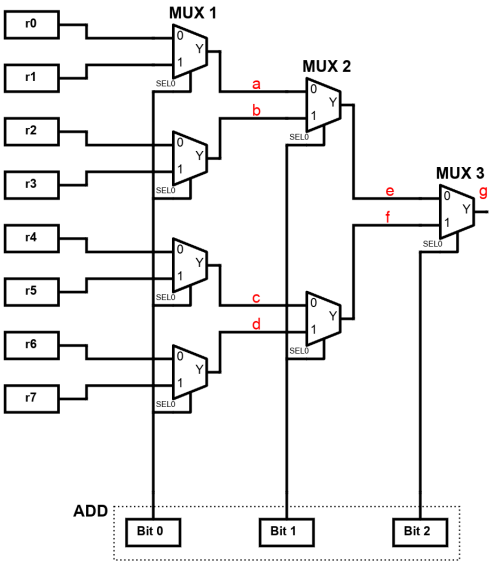


Figure 5: Multiplexer tree

7 Summary and Outlook

We suggest a new approach to identifying flaws in implementations of masking schemes. Our technique keeps some shares constant, whilst randomizing others. We tailor the idea to specifically find flaws that are based on unintended (micro-architectural) transitions, and showcase how to apply our method to three software implementations of multiplication gadgets. Our method is better than the conventional higher-order fixed-vs-random input TVLA in two significant ways. Firstly, it improves trace and computational efficiency by utilising raw traces. Secondly, it gives a clue about which shares are involved in the flaw, which together with the information about when the flaw occurs, facilitates finding/understanding/fixing the problem.

It is important to note that our method requires control over the shares. This should be possible in the context of an in-house evaluation or testing facility in the context of cryptographic software implementations. We leave it as an open question to analyse how applicable our technique is in the context of cryptographic hardware implementations.

Acknowledgments

Nima Mahdion and Elisabeth Oswald have been supported in part by the European Research Council (ERC) under the European Union's Horizon 2020 research and innovation programme (grant agreement number 725042). Both authors would like to thank the anonymous reviewers for their constructive comments.

References

[ARMa] ARM. ARM Achitecture. https://developer.arm.com/documentation/ddi0406/latest.

[ARMb] ARM. Thumb-16-bit instruction set quick reference card. https://developer.arm.com/documentation/qrc0006/e.

[BCO04] Eric Brier, Christophe Clavier, and Francis Olivier. Correlation power analysis with a leakage model. In Marc Joye and Jean-Jacques Quisquater, editors, *Cryptographic Hardware and Embedded Systems - CHES 2004: 6th International Workshop Cambridge, MA, USA, August 11-13, 2004. Proceedings*, volume 3156 of *Lecture Notes in Computer Science*, pages 16–29. Springer, 2004. doi:10.1007/978-3-540-28632-5_2.

[BGG+14] Josep Balasch, Benedikt Gierlichs, Vincent Grosso, Oscar Reparaz, and François-Xavier Standaert. On the cost of lazy engineering for masked software implementations. In Marc Joye and Amir Moradi, editors, *Smart Card Research and Advanced Applications - 13th International Conference, CARDIS 2014, Paris, France, November 5-7, 2014. Revised Selected Papers*, volume 8968 of *Lecture Notes in Computer Science*, pages 64–81. Springer, 2014. doi:10.1007/978-3-319-16763-3_5.

[BSS19] Olivier Bronchain, Tobias Schneider, and François-Xavier Standaert. Multi-tuple leakage detection and the dependent signal issue. *IACR TCHES*, 2019(2):318–345, 2019. URL: https://tches.iacr.org/index.php/TCHES/article/view/7394, doi:10.13154/tches.v2019.i2.318-345.

[CGLS21] Gaëtan Cassiers, Benjamin Grégoire, Itamar Levi, and François-Xavier Standaert. Hardware private circuits: From trivial composition to full verification. *IEEE Trans. Computers*, 70(10):1677–1690, 2021. doi:10.1109/TC.2020.3022979.

[Com17] Common Criteria. The Common Criteria for Information Technology Security Evaluation. https://www.commoncriteriaportal.org/cc/, 2017.

[CRR02] Suresh Chari, Josyula R. Rao, and Pankaj Rohatgi. Template attacks. In Burton S. Kaliski Jr., Çetin Kaya Koç, and Christof Paar, editors, *Cryptographic Hardware and Embedded Systems - CHES 2002, 4th International Workshop, Redwood Shores, CA, USA, August 13-15, 2002, Revised Papers*, volume 2523 of *Lecture Notes in Computer Science*, pages 13–28. Springer, 2002. doi:10.1007/3-540-36400-5_3.

[DCE16] A. Adam Ding, Cong Chen, and Thomas Eisenbarth. Simpler, faster, and more robust t-test based leakage detection. In François-Xavier Standaert and Elisabeth Oswald, editors, *Constructive Side-Channel Analysis and Secure Design - 7th International Workshop, COSADE 2016, Graz, Austria, April 14-15, 2016, Revised Selected Papers*, volume 9689 of *Lecture Notes in Computer Science*, pages 163–183. Springer, 2016. doi:10.1007/978-3-319-43283-0_10.

[DS16a] François Durvaux and François-Xavier Standaert. From improved leakage detection to the detection of points of interests in leakage traces. In Marc Fischlin and Jean-Sébastien Coron, editors, *Advances in Cryptology - EUROCRYPT 2016 - 35th Annual International Conference on the Theory and Applications of Cryptographic Techniques, Vienna, Austria, May 8-12, 2016,*

Proceedings, Part I, volume 9665 of *Lecture Notes in Computer Science*, pages 240–262. Springer, 2016. doi:10.1007/978-3-662-49890-3_10.

[DS16b] François Durvaux and François-Xavier Standaert. From improved leakage detection to the detection of points of interests in leakage traces. In Marc Fischlin and Jean-Sébastien Coron, editors, *EUROCRYPT 2016, Part I*, volume 9665 of *LNCS*, pages 240–262, May 2016. doi:10.1007/978-3-662-49890-3_10.

[DZD+17] A. Adam Ding, Liwei Zhang, François Durvaux, François-Xavier Standaert, and Yunsi Fei. Towards sound and optimal leakage detection procedure. In Thomas Eisenbarth and Yannick Teglia, editors, *Smart Card Research and Advanced Applications - 16th International Conference, CARDIS 2017, Lugano, Switzerland, November 13-15, 2017, Revised Selected Papers*, volume 10728 of *Lecture Notes in Computer Science*, pages 105–122. Springer, 2017. doi:10.1007/978-3-319-75208-2_7.

[FGM+23] Jakob Feldtkeller, Tim Güneysu, Thorben Moos, Jan Richter-Brockmann, Sayandeep Saha, Pascal Sasdrich, and François-Xavier Standaert. Combined private circuits - combined security refurbished. In Weizhi Meng, Christian Damsgaard Jensen, Cas Cremers, and Engin Kirda, editors, *ACM CCS 2023*, pages 990–1004. ACM Press, November 2023. doi:10.1145/3576915.3623129.

[FIP19] Security Requirements for Cryptographic Modules (FIPS PUB 140-3). https://nvlpubs.nist.gov/nistpubs/FIPS/NIST.FIPS.140-3.pdf, 2019.

[GGJR+11] Benjamin Jun Gilbert Goodwill, Josh Jaffe, Pankaj Rohatgi, et al. A testing methodology for side-channel resistance validation. In *NIST non-invasive attack testing workshop*, volume 7, pages 115–136, 2011.

[GMK16] Hannes Groß, Stefan Mangard, and Thomas Korak. Domain-oriented masking: Compact masked hardware implementations with arbitrary protection order. In Begül Bilgin, Svetla Nikova, and Vincent Rijmen, editors, *Proceedings of the ACM Workshop on Theory of Implementation Security, TIS@CCS 2016 Vienna, Austria, October, 2016*, page 3. ACM, 2016. doi:10.1145/2996366.2996426.

[GMK17] Hannes Groß, Stefan Mangard, and Thomas Korak. An efficient side-channel protected AES implementation with arbitrary protection order. In Helena Handschuh, editor, *CT-RSA 2017*, volume 10159 of *LNCS*, pages 95–112, February 2017. doi:10.1007/978-3-319-52153-4_6.

[GOP22] Si Gao, Elisabeth Oswald, and Dan Page. Towards micro-architectural leakage simulators: Reverse engineering micro-architectural leakage features is practical. In Orr Dunkelman and Stefan Dziembowski, editors, *Advances in Cryptology - EUROCRYPT 2022 - 41st Annual International Conference on the Theory and Applications of Cryptographic Techniques, Trondheim, Norway, May 30 - June 3, 2022, Proceedings, Part III*, volume 13277 of *Lecture Notes in Computer Science*, pages 284–311. Springer, 2022. doi:10.1007/978-3-031-07082-2_11.

[GR17] Dahmun Goudarzi and Matthieu Rivain. How fast can higher-order masking be in software? In Jean-Sébastien Coron and Jesper Buus Nielsen, editors, *EUROCRYPT 2017, Part I*, volume 10210 of *LNCS*, pages 567–597, April / May 2017. doi:10.1007/978-3-319-56620-7_20.

[ISW03] Yuval Ishai, Amit Sahai, and David Wagner. Private circuits: Securing hardware against probing attacks. In Dan Boneh, editor, *CRYPTO 2003*, volume 2729 of *LNCS*, pages 463–481, August 2003. doi:10.1007/978-3-540-45146-4_27.

[JS11] Show-Li Jan and Gwowen Shieh. Optimal sample sizes for welch's test under various allocation and cost considerations. *Behavior research methods*, 43:1014–1022, 2011. doi:10.3758/s13428-011-0095-7.

[KM22] David Knichel and Amir Moradi. Low-latency hardware private circuits. In Heng Yin, Angelos Stavrou, Cas Cremers, and Elaine Shi, editors, *ACM CCS 2022*, pages 1799–1812. ACM Press, November 2022. doi:10.1145/3548606.3559362.

[Koc96] Paul C. Kocher. Timing attacks on implementations of Diffie-Hellman, RSA, DSS, and other systems. In Neal Koblitz, editor, *Advances in Cryptology - CRYPTO '96, 16th Annual International Cryptology Conference, Santa Barbara, California, USA, August 18-22, 1996, Proceedings*, volume 1109 of *Lecture Notes in Computer Science*, pages 104–113. Springer, 1996. doi:10.1007/3-540-68697-5_9.

[MOBW13] Luke Mather, Elisabeth Oswald, Joe Bandenburg, and Marcin Wójcik. Does my device leak information? An a priori statistical power analysis of leakage detection tests. In Kazue Sako and Palash Sarkar, editors, *ASIACRYPT 2013, Part I*, volume 8269 of *LNCS*, pages 486–505, December 2013. doi:10.1007/978-3-642-42033-7_25.

[MPW22] Ben Marshall, Dan Page, and James Webb. MIRACLE: MIcRo-ArChitectural leakage evaluation A study of micro-architectural power leakage across many devices. *IACR TCHES*, 2022(1):175–220, 2022. doi:10.46586/tches.v2022.i1.175-220.

[MRSS18] Amir Moradi, Bastian Richter, Tobias Schneider, and François-Xavier Standaert. Leakage detection with the χ^2-test. *IACR TCHES*, 2018(1):209–237, 2018. URL: https://tches.iacr.org/index.php/TCHES/article/view/838, doi:10.13154/tches.v2018.i1.209-237.

[MWM21] Thorben Moos, Felix Wegener, and Amir Moradi. DL-LA: Deep learning leakage assessment. *IACR TCHES*, 2021(3):552–598, 2021. URL: https://tches.iacr.org/index.php/TCHES/article/view/8986, doi:10.46586/tches.v2021.i3.552-598.

[NIS01] Advanced Encryption Standard (AES), 2001. Federal Information Processing Standards Publication 197, http://csrc.nist.gov/publications/fips/fips197/fips-197.pdf.

[NRR06] Svetla Nikova, Christian Rechberger, and Vincent Rijmen. Threshold implementations against side-channel attacks and glitches. In Peng Ning, Sihan Qing, and Ninghui Li, editors, *ICICS 06*, volume 4307 of *LNCS*, pages 529–545, December 2006. doi:10.1007/11935308_38.

[Pag] Dan Page. SCALE: Side-Channel Attack Lab. Exercises. https://github.com/danpage/scale.

[PRB09] Emmanuel Prouff, Matthieu Rivain, and Régis Bevan. Statistical analysis of second order differential power analysis. *IEEE Trans. Computers*, 58(6):799–811, 2009. doi:10.1109/TC.2009.15.

[RBN+15] Oscar Reparaz, Begül Bilgin, Svetla Nikova, Benedikt Gierlichs, and Ingrid Verbauwhede. Consolidating masking schemes. In Rosario Gennaro and Matthew J. B. Robshaw, editors, *CRYPTO 2015, Part I*, volume 9215 of *LNCS*, pages 764–783, August 2015. doi:10.1007/978-3-662-47989-6_37.

[Rep16] Oscar Reparaz. Detecting flawed masking schemes with leakage detection tests. In Thomas Peyrin, editor, *FSE 2016*, volume 9783 of *LNCS*, pages 204–222. Springer, Heidelberg, March 2016. doi:10.1007/978-3-662-52993-5_11.

[RP10] Matthieu Rivain and Emmanuel Prouff. Provably secure higher-order masking of AES. In Stefan Mangard and François-Xavier Standaert, editors, *CHES 2010*, volume 6225 of *LNCS*, pages 413–427, August 2010. doi:10.1007/978-3-642-15031-9_28.

[SM15] Tobias Schneider and Amir Moradi. Leakage assessment methodology - A clear roadmap for side-channel evaluations. In Tim Güneysu and Helena Handschuh, editors, *Cryptographic Hardware and Embedded Systems - CHES 2015 - 17th International Workshop, Saint-Malo, France, September 13-16, 2015, Proceedings*, volume 9293 of *Lecture Notes in Computer Science*, pages 495–513. Springer, 2015. doi:10.1007/978-3-662-48324-4_25.

[Sta18] François-Xavier Standaert. How (not) to use welch's t-test in side-channel security evaluations. In Begül Bilgin and Jean-Bernard Fischer, editors, *Smart Card Research and Advanced Applications, 17th International Conference, CARDIS 2018, Montpellier, France, November 12-14, 2018, Revised Selected Papers*, volume 11389 of *Lecture Notes in Computer Science*, pages 65–79. Springer, 2018. doi:10.1007/978-3-030-15462-2_5.

[Stu08] Student. The probable error of a mean. *Biometrika*, pages 1–25, 1908. doi:10.2307/2331554.

[Wel47] B. L. Welch. The generalization of 'Student's' problem when several different population variances are involved. *Biometrika*, 34(1-2):28–35, 01 1947. arXiv:https://academic.oup.com/biomet/article-pdf/34/1-2/28/553093/34-1-2-28.pdf, doi:10.1093/biomet/34.1-2.28.

[WTW+22] Yaru Wang, Ming Tang, Pengbo Wang, Botao Liu, and Rui Tian. The levene test based-leakage assessment. *Integr.*, 87:182–193, 2022. URL: https://doi.org/10.1016/j.vlsi.2022.06.013, doi:10.1016/J.VLSI.2022.06.013.

[YJ21] Wei Yang and Anni Jia. Side-channel leakage detection with one-way analysis of variance. *Secur. Commun. Networks*, 2021:6614702:1–6614702:13, 2021. doi:10.1155/2021/6614702.

[ZQO19] Xinping Zhou, Kexin Qiao, and Changhai Ou. Leakage detection with Kolmogorov-Smirnov test. Cryptology ePrint Archive, Report 2019/1478, 2019. URL: https://eprint.iacr.org/2019/1478.