A Novel Completeness Test and its Application to Side Channel Attacks and Simulators

Si Gao[1] and Elisabeth Oswald[1,2]

[1]University of Klagenfurt
[2]University of Bristol

Abstract

Today's side channel attack targets are often complex devices in which instructions are processed in parallel and work on 32-bit data words. Consequently, the *state* that is involved in producing leakage in these modern devices is large, and basing evaluations (i.e. worst case attacks), simulators, and assumptions for (masking) countermeasures on a potentially *incomplete state* can lead to drastically wrong conclusions.

We put forward a novel notion for the "completeness" of an assumed state, together with an efficient statistical test that is based on "collapsed models". Our novel test can be used to recover a *state* that contains multiple 32-bit variables in a grey box setting. We illustrate how our novel test can help to guide side channel attacks and we reveal new attack vectors for existing implementations. We also show how the application of our statistical test shows where even the most recent leakage simulators do not capture all available leakage of their respective target devices.

1 Introduction

Since Kocher's seminal work [21], research has explored the properties of all "key ingredients" that contribute to successful side channel (key recovery) attacks. These key ingredients include side channel distinguishers (i.e. the way in which side channel leakage is statistically exploited in attacks), and side channel leakage models (i.e. the way in which side channel leakage is predicted or explained by an adversary). The latter component, the leakage model, is crucial for attacks (a better model leads to attacks requiring fewer leakage observations), and therefore it is also of fundamental significance in the context of security evaluations, but it also greatly matters in the context of leakage simulators as well as for assumptions that designers make when implementing masking countermeasures.

But what does a leakage model constitute of? Informally, most of the existing literature understands a leakage model to be a *leakage function* that maps a

collection of device internal variables (*the state*) to a real value (if it is a univariate model). Considering this informal definition in the context of attacks, it is clearly desirable to try and find a function that offers good predictions for the true device leakage, because it enables successful attacks with fewer traces. Thus, a lot of research has gone into deriving good estimates for leakage functions from real device data [11, 31, 15, 35, 26]. However, the leakage function itself is only part of what consistitutes a leakage model: the state that is being leaked on is equally relevant.

In a realistic setting (such as low to mid-range processors, or dedicated hardware implementations of core crypto algorithms), finding the *state* is far from easy, not only because they tend to be closed source. If open source descriptions are available, e.g. ARM released some semi-obfuscated VHDL descriptions, then these are at best architecturally similar to the commercial products of the same type, but they are not micro-architecturally equivalent at all. Micro-architectural effects have been explored and exploited across many recent works [29, 19, 27, 18, 25]. These papers show how a wrong assumption about the *state* renders provably secure masking schemes completely insecure in practice. In the context of application specific crypto cores, the situation is not better as their descriptions are typically also not available to the public. Taking the perspective of a designer of an application specific crypto core (who has access to such a description), it is in principle possible to identify the components that are active during any cycle. However, inclusion of everything that contributes without understanding of the amount of its contribution or its relevance, may lead to a model that becomes entirely impractical to work with. Thus even in this context, a methodology to decide what is the "state that matters" would be desirable.

Our contribution. We stress that finding the exact intermediate state from a typical processor in a grey box setting is a long-standing problem: like many (statistical learning) problems, a universally optimal solution is unlikely to exist. Thus, whilst we do not claim optimality of our work, we claim the following contributions:

1. We clearly state the identification of the actual *state* as a fundamental problem and discuss its impact on attacks and leakage simulators.

2. We put forward a novel notion for models—denoted as "*completeness*"—which flags whether the tested model has captured all relevant state.

3. We put forward a novel statistical methodology based on what we call "collapsed" models: using the nested *F-tests*, we can test whether a leakage model is *complete* in a "collapsed" setup and infer whether it is *complete* in the original un-collapsed setup.

4. We show how our approach can find subtle leakage that can be easily neglected. Although such leakage does not necessarily contribute to more

effective attacks, it plays an important role in comprehensive security evaluations.

5. We discuss the importance of *completeness* in the context of simulators for leakage detections and demonstrate that our approach can lead to better models for simulations.

Organisation. We start our discussion with clarifying some definitions and introducing a few useful statistical tools in Section 2. Section 3 introduces the concept of *completeness* and proposes a necessary (but not sufficient) test to verify *completeness*. We further show how our novel test can be applied when analysing the leakage from both unprotected and masked implementations (Section 4), revealing subtle leakage that is otherwise difficult to find. Section 5 confirms *completeness* is also critical for leakage simulators, demonstrating how an incomplete leakage model can jeopardise the following detection accuracy. We summarise our discussion and emphasise a few important lessons learned in Section 7.

2 Preliminaries

2.1 Leakage modelling: state of the art

We use some simple notations throughout this paper: for the sake of readability we do not include time indices in our notations. Consequently, any set of variables, and any leakage function, is specific to a point in time during the computation of some algorithm[1].

We call the set $\mathbf{X}$ the entire device state. $\mathbf{X}$ is comprised of key and input dependent variables that the leakage function L acts on. The variable $Y = \{y\} \in \mathbf{Y}$ is a leakage observation that is available to an adversary. We also follow the usual convention that traces are noisy, whereby the leakage contribution $L(\mathbf{X})$ and the (Gaussian) noise $N(0, \sigma^2)$ are independent:

$$y = L(\mathbf{X}) + N(0, \sigma^2)$$

The leakage model is an approximation of the real device leakage. It consists of a function and a state: the function maps the state to a (set of) real values. In our work, which follows the trajectory of [26, 32], we consider univariate models, thus we write $L : \mathbb{F}_2^m \to \mathbf{R}$ and $x \in \mathbb{F}_2^m$.

Throughout this work we assume that we are working in a "grey box " setting, i.e. we have some basic knowledge about the device/implementation (e.g. instruction set architecture, executing code etc.), but not concrete gate-level hardware details. The relevant device state $\mathbf{X}$ (for a certain time index) is unknown in this setting. We can of course, build an overly conservative model using all possible *state* $\hat{\mathbf{X}}$ where $\mathbf{X} \subset \hat{\mathbf{X}}$ (all intermediate variables that

[1] We use the concept of "time points" for readability, but one could equally use the concept of clock cycles or instructions instead.

ever occured). However, such a model is clearly unusable for attacks/evaluations (because it requires guessing the entire key) and it is also unusable for simulators (because the estimation of its distribution is computationally infeasible).

The *de facto* practice in all attacks/evaluations, when building leakage models, is to divide the model building process into two steps. The first step is identifying a concise (i.e. easily enumerable) state $\mathbf{Z}$. For instance, a popular assumption is that the intermediate state depends completely on the output of an S-box (denoted by S_{out}) computation, which leads to the cardinality of the state being small (e.g. $\#\{\mathbf{Z}\} = 2^8$ for AES).

The second step is to estimate the leakage function assuming it only acts on $\mathbf{Z}$. Various techniques have been proposed, including naive templating [11], regression-based modelling [31, 15], step-wise regression [38] etc. Previous works [38, 31, 17] have also proposed various metrics to evaluate/certificate the device's leakage (as well as the quality of model that built from the measurements). As many will be utilised later, the next two subsections explain these techniques in details, then we move on to our point of interest: *what should we do about the first step?*

2.2 Leakage modelling: approaches

Already in the early days of side channel research, the concept of profiling (aka leakage modelling, aka templating) was introduced by Chari et al. [11]. In their original paper, the idea was to assume that the distribution of the measurements from the same state value should follow a (multivariate) normal distribution, and an adversary with *a priori* access to a device can simply estimate the parameters of the distribution.

An alternative to the direct parameter estimation is using regression techniques to derive an equivalent model. A paper by Schindler et al. [31] proposes the use of regression to derive a linear model of a considered state. The basis of building regression models is that we can express any real valued function of $\mathbf{Z}$ as a polynomial $\tilde{L} = \sum_j \beta_j u_j(\mathbf{Z})$ [14], or short $\tilde{L} = \vec{\beta}(\mathbf{Z})$. In this polynomial the explanatory variables u_j are monomials of the form $\prod_{i=0}^{n-1} z_i^{j_i}$ where z_i denotes the i-th bit of $\mathbf{Z}$ and j_i denote the i-th bit of j (with n the number of bits needed to represent $\mathbf{Z}$ in binary). Regression then estimates the coefficients β_j. The explanatory variables u_j simply represent the different values that $\mathbf{Z}$ can take. If we do not restrict the u_j then the resulting model is typically called the *full model*. If no subscript is given, we implicitly mean the *full* model. In many previous attacks, the leakage model is restricted to just contain the linear terms. We denote this particular *linear model* using $\tilde{L}_l$.

2.3 Model Quality

Any model is only an approximation: there is a gap between the model output and the observed reality. Statistical models are built for at least two purposes [33]. They are either used to predict events in the future, in which case the model quality relates to its predictive power; or they are used to help explain

reality, in which case the model quality relates to how many of the relevant factors it can identify. In the context of leakage attacks/evaluations, models are used to predict side channel leaks. Therefore we use metrics such as the coefficient of determination, and cross validation to judge the quality. In the context of leakage simulators, models are built that are supposed to include as many relevant leakage sources as possible. Therefore, the quality relates how two (or more) models compared with each other in terms of explaining the realistic leakage.

Coefficient of determination For any model $\tilde{L}(\mathbf{Z})$ that is estimated from the side channel measurements $\mathbf{Y}$, the "modelling error" can be defined as the *residual sum of squares* (RSS),

$$RSS = \sum_{i=1}^{q} (y^{(i)} - \tilde{L}(z^{(i)}))^2$$

where q represents the number of traces and $z^{(i)}$ represents the value of z for the i-th measurement. Meanwhile, the explained data-variance can be interpreted as the *explained sum of squares* (ESS),

$$ESS = \sum_{i=1}^{q} \left(\tilde{L}(z^{(i)}) - \bar{y} \right)^2$$

where $\bar{y}$ represents the mean of measured values $\mathbf{Y}$. If $\tilde{L}$ is derived from linear regression on $\mathbf{Y}$, RSS and ESS should sum up to the *total sum of squares* (TSS),

$$TSS = \sum_{i=1}^{q} \left(y^{(i)} - \bar{y} \right)^2$$

Then, the *coefficient of determination* (R^2) is defined as:

$$R^2 = \frac{ESS}{TSS} = 1 - \frac{RSS}{TSS}.$$

Given two estimated models $\tilde{L}_1$ and $\tilde{L}_2$, whereby both models are assumed to have the same number of terms (i.e. same restrictions on $u_j(\mathbf{Z})$ in Section 2.2), the model with the higher R^2 value would be considered as better. The crucial point here is that both models need the same number of terms, because the R^2 increases with the number of terms that are included in the model. Consequently, the R^2 does not lend itself to investigate models that represent approximations in different numbers of terms.

Cross-validation An important aspect in model validation is to check if a model overfits the data. If a model overfits the data, it will generalise badly, which means it is bad in terms of predicting new data. Therefore using cross-validation of any chosen metric (e.g. the RSS) is essential [17] when aiming for models with a high *predictive power*. Given two models, one can compute via cross validation both RSS values and then judge their relative predictive power.

F-tests Given two "nested" models (i.e. there is a so called *full* model and a *restricted* model which only consists of a subset of terms), the *F-test* is the most natural way to decide whether the *restricted* model is missing significant contribution compared with the *full* model. More specifically, assuming a *full* model $\tilde{L}_f(\mathbf{Z}_f)$ and a *restricted* model $\tilde{L}_r(\mathbf{Z}_r)$, where $\mathbf{Z}_r$ is constructed by removing $z_f - z_r$ explanatory variables (set regression coefficients to 0) from $\mathbf{Z}_f$, one can compute the F-statistic as

$$F = \frac{\frac{RSS_r - RSS_f}{z_f - z_r}}{\frac{RSS_f}{q - z_f}}.$$

The resulting F follows the F distribution with $(z_f - z_r, q - z_f)$ degree of freedom. A p-value above a statistically motivated threshold rejects the null hypothesis (the two models are equivalent) and hence suggests that at least one of the removed variables is potentially useful. This approach was used in [26] to derive relatively fine grained models on selected intermediate states.

2.4 Approaches to find the state

It is critical to remember that all approaches above (at least their current usage in the community) assume the concise $\mathbf{Z}$ has already been found. In other words, whether we assume users already know $\mathbf{X}$ beforehand and simply set $\mathbf{Z} = \mathbf{X}$ (eg. through analysing hardware details etc.) or users have already constructed an appropriate $\mathbf{Z}$ that ensures $\mathbf{X} \subseteq \mathbf{Z}$ through some trial-and-error process. To our knowledge, this step is often quite *ad hoc*: users try some set of $\mathbf{Z}$, evaluate the leakage model with R^2 or cross-validation (or alternatively, perform attacks with CPA). If the evaluation/attack result is not successful, it suggests the current $\mathbf{Z}$ is unlikely to be correct. Otherwise, $\mathbf{Z}$ might be a part of $\mathbf{X}$, but not necessarily complete. Based on their initial knowledge, users can repeat this process and determine what they believed to be the most suitable $\mathbf{Z}$, without much confidence if this $\mathbf{Z}$ is sufficient.

Leakage certification techniques, eg. "assumption error" v.s. "estimation error" [17] are also designed under the assumption that $\mathbf{Z}$ is given. One could be tempted to use such techniques to test the scenario where the selected $\mathbf{Z}$ is not sufficient: however, neither does not fit with the original intention of [17] nor is statistical power of those techniques sufficient. In the interest of flow we provide the reasoning for this statement in Appendix A.

3 Model quality: is my leakage model complete?

In this section, we introduce the novel concept of model completeness, and an associated test to measure it efficiently.

3.1 Completeness

So far the nested *F-test* has mainly been used for determining low degree terms of $\tilde{L}(\mathbf{Z})$ [26, 38]. However in the wider statistical literature, its primary usage is in selecting contributing factors from a variety of potential explanatory variables, which helps to build a model with a high explanatory power.

Definition 1. *For any two nested models $\tilde{L}(\mathbf{A})$, $\tilde{L}(\mathbf{B})$ we denote* $\mathbf{A}$ *as* **complete (with respect to B)** *if the F-test results support the assumption that $\tilde{L}(\mathbf{A})$ did not miss any significant contributing factor compared with $\tilde{L}(\mathbf{B})$. With infinite profiling traces (i.e. infinite statistical power), this usually suggests $\mathbf{B} \subseteq \mathbf{A}$ (if already assumed $\mathbf{A} \subseteq \mathbf{B}$, then $\mathbf{A} = \mathbf{B}$).*

In practice, considering the restriction of a grey box setting, we have limited information about the state (at any point in time during a computation). Thus our main challenge is to define a conservative set $\hat{\mathbf{X}}$ that satisfies $\mathbf{X} \subseteq \hat{\mathbf{X}}$, then drop terms from $\hat{\mathbf{X}}$ to see which (combinations of) terms actually matter (i.e. find an approximation $\mathbf{Z}$ for $\mathbf{X}$).

Toy example. Say we have an unknown relevant set $\mathbf{X}$. Although $\mathbf{X}$ is unknown, we can define an overly conservative set e.g. $\hat{\mathbf{X}} = \{x_0, x_1, x_2, x_3\}$ that satisfies $\mathbf{X} \subseteq \hat{\mathbf{X}}$. At this point, we wish to test if we can discard x_3 and see if $\mathbf{Z} = \{x_0, x_1, x_2\}$ is a good model. Following our discussion in Section 2.2, we can estimate from realistic measurements:

$$\tilde{L}_f = \vec{\beta}(\hat{\mathbf{X}})$$

$$\tilde{L}_r = \vec{\beta}(\mathbf{Z})$$

If the *F-test* reports a *p-value* higher than the significance level, we can conclude at least one of the regression terms that depends on x_3 is contributing significantly. In other words, $x_3 \in \mathbf{X}$ and $\mathbf{X} \not\subseteq \mathbf{Z}$, which suggests the model built with $\mathbf{Z}$ is missing some leakage, it is not complete with respect to $\hat{\mathbf{X}}$ and therefore it is also not complete with respect to $\mathbf{X}$.

Defining $\hat{\mathbf{X}}$ concretely For deterministic block cipher implementations with a constant key (a case frequently considered in the side channel literature), a conservative state assumption would be based on the entire plaintext. Because of the large block size, this is a starting point that is in fact unworkable from a statistical point of view (to estimate anything we need multiple observations for each value that the state can take). If an implementation is masked, then also the masks need to be included. The F-test then checks the contribution of all resulting explanatory variables. Unlike leakage detection tests (which consider only the unshared secret), the F-test is agnostic to specific statistical moments, as long as all explanatory variables are taken into consideration. We explain a statistical trick in the next section, that makes this seemingly infeasible problem, practical.

3.2 Collapsed F-test for completeness

The *full* model is often too large: it consists of many large variables in practice (in statistical terminology it is a factorial design). Consequently, we must find ways to reduce the inevitable complexity. Techniques such as aliasing (i.e. if several factors are known to be related) or identifying confounding variables (i.e. variables that impact on multiple factors) are well known in statistical model building. What is possible is typically situation dependent. We make two key observations that apply to side channel leakage modelling.

The first observation (we mention this before in the text) is that the *F-test*, although often used to deal with proportional models, actually tests for the inclusion/exclusion of explanatory variables in *nested* models. Such models are nominal models, i.e. the regression coefficient is either 0 or 1: either a variable matters or it does not.

The second observation is that although our explanatory variables contain n bits, they are rarely independent; and we consider a variable as relevant if any of its bits are relevant. Consequently, there is an elegant trick to bound our explanatory variables to small space.

Bounding the explanatory variables We suggest that it is possible to bound the explanatory variables to a much smaller space. For instance, assuming our target process has four inputs A, B, A' and B'. Each input is an n-bit state, where 2^n explanatory variables can be constructed from $(a_0, a_1, ..., a_{n-1}), a_i \in \mathbb{F}_2$. By setting $a_i = a_0$ (and a_0 drawn at random from $\{0, 1\}$), we can bound the input A to a much smaller space:

$$a = (a_0, a_0, ..., a_0), a_0 \in \mathbb{F}_2.$$

Applying this restriction to the other 3 inputs, the *full* model now contains only 2^4 parameters, which is easy to work with.

Of course, such a restriction is not for free: originally, there could be many interaction terms between the explanatory variables. In the model L_c these terms are "collapsed" and "added" to the remaining terms, e.g. $a_1 a_0$ becomes a_0 as $a_1 = a_0$. In fact, as there is only 1 bit randomness, a_0 now becomes an alias for the operand A: having this term in L_c suggests A appears in L, but certainly not in the same way as in L_c. We can expand this idea by allowing two bits of randomness: this enables us to differentiate between linear and non-linear models[2].

Formalising this idea, we define a mapping called "collapse" $Coll$ on the $u_j(\mathbf{Z})$, where $\mathbf{Z} = \{AA'BB'\}$. Recall that $u_j(\mathbf{Z})$ is defined (Section 2.2) as:

$$u_j(\mathbf{Z}) = \prod z_i^{j_i}$$

[2]We could take this further and include more "in-variable" interactions, but we left this for future considerations

where j_i represents the i-th bit of the binary representation of j. For any $j \in [0, 2^{4n})$, we define a $2^{4n} \rightarrow 2^n$ map $Coll$ as:

$$Coll(j) = j_{coll} = j_a||j_{a'}||j_b||j_{b'} \in [0, 2^4)$$

where $j_a = \bigvee_{i=0}^{n-1} j_i$, $j_{a'} = \bigvee_{i=n}^{2n-1} j_i$, $j_b = \bigvee_{i=2n}^{3n-1} j_i$, $j_{b'} = \bigvee_{i=3n}^{4n-1} j_i$. Readers can easily verify that when all operands are bounded to 1-bit, we have

$$u_j(\mathbf{Z}) = u_{j_{coll}}(\mathbf{Z}_c), \mathbf{Z}_c = \{z_c | z_c = a_0||a'_0||b_0||b'_0\}$$

The latter can be easily tested in an *F-test*. In the following, we show that the test model passes the *F-test* in our "collapsed" case is a necessary (but not sufficient) condition for passing the *F-test* in the original setup.

Theorem 1. *If a collapsed term $u_{j_{coll}}(\mathbf{Z}_c)$ cannot be ignored from $\tilde{L}_c$ (i.e. $\beta_{j_{coll}} \neq 0$), at least one of the corresponding $u_j(\mathbf{Z})$ cannot be ignored from $\tilde{L}$ (i.e. $\beta_j \neq 0$).*

Proof. In the original case, any leakage model can always be written as

$$\tilde{L}(\mathbf{Z}) = \sum_{j=0}^{2^{4n}-1} \beta_j u_j(\mathbf{Z})$$

However, considering the inputs have been bounded, such model collapses to:

$$\tilde{L}(\mathbf{Z}) = \sum_{j_{coll}=0}^{2^4-1} \left(\sum_{\forall j, Coll(j)=j_{coll}} \beta_j \right) u_{j_{coll}}(\mathbf{Z}_c)$$

Thus, if a certain collapsed term $u_{j_{coll}}(\mathbf{Z}_c)$ has a significant contribution to $\tilde{L}_c$ (i.e. $\beta_{j_{coll}} \neq 0$), one can conclude that:

$$\sum_{\forall j, Coll(j)=j_{coll}} \beta_j \neq 0 \Rightarrow \exists j, \beta_j \neq 0$$

Clearly nothing can be concluded if the above sum equals 0, which suggests this is only a necessary condition. □

Theorem 1 implies that whilst we still cannot directly test $\tilde{L}_f = \vec{\beta}(\hat{\mathbf{X}})$, we can estimate the restricted (and collapsed) models $\tilde{L}_{cr} = \vec{\beta}(\mathbf{Z}_c)$ from realistic measurements and test it against the estimated collapsed *full* model $\tilde{L}_{cf} = \vec{\beta}(\hat{\mathbf{X}}_c)$: if the *F-test* finds enough evidence to reject a model $\tilde{L}_{cr}$ in relation to the collapsed *full* model $\tilde{L}_{cf}$, then it is clear that the model $\tilde{L}_r = \vec{\beta}(\mathbf{Z})$ would also be rejected in comparison to the *full* model $\tilde{L}_f = \vec{\beta}(\hat{\mathbf{X}})$ (i.e. $\mathbf{Z}$ is not complete with respect to $\hat{\mathbf{X}}$).

Toy example. Suppose we want to test $\tilde{L}_r = \vec{\beta}\{AB\}, \vec{\beta} \in (0,1)^{2^{2n}}$ against $\tilde{L}_f = \vec{\beta}\{AA'BB'\}, \vec{\beta} \in (0,1)^{2^{4n}}$. As mentioned before, for $n = 32$, direct testing is not feasible. However, we can bound the inputs and test

$$\tilde{L}_{cr} = \beta_0 + \beta_1 a_0 + \beta_2 b_0 + \beta_3 a_0 b_0$$

$$\begin{aligned}\tilde{L}_{cf} = \beta_0 &+ \beta_1 a_0 + \beta_2 a'_0 + \beta_3 b_0 + \beta_4 b'_0 \\ &+ \beta_5 a_0 b_0 + \beta_6 a'_0 b'_0 + \beta_7 a'_0 b_0 + \beta_8 a_0 b'_0 + \beta_9 b_0 b_0 + \beta_{10} a_0 a'_0 \\ &+ \beta_{11} a_0 a'_0 b'_0 + \beta_{12} a_0 a'_0 b_0 + \beta_{13} a_0 b'_0 b_0 + \beta_{14} a'_0 b'_0 b_0 \\ &+ \beta_{15} a_0 a'_0 b_0 b'_0\end{aligned}$$

If the *F-test* rejects the null hypothesis, then we know that the missing terms make a difference not only in $\tilde{L}_c$ but also in $\tilde{L}$. Therefore, we can conclude the $\tilde{L}_r$ is also not complete, without explicitly testing it. The price to pay is that unlike the original *F-test*, our collapsed test becomes a necessary yet not sufficient condition: that being said, any $\mathbf{Z}$ that fails our test still presents a genuine concern, as it directly suggests the selected $\mathbf{Z}$ is unlikely to be complete and all following steps can be potentially jeopardised.

Considering from now on we will *always* work with collapsed models, we will not continue using the double subscript cr, but revert back to just using r.

3.3 Statistical power of the nested F-test

For any statistic test, an important question to ask is how much power does it preserve. To compute the power of collapsed F-tests, we first need to consider the *effect size* that we are dealing with. The *effect size* in our case relates to the difference between the *restricted* model and the *full* model, which can be computed (according to Cohen [12]) as:

$$f^2 = \frac{R_F^2 - R_R^2}{1 - R_F^2} = \frac{RSS_R - RSS_F}{RSS_F}$$

Under the alternative hypothesis, the computed F-statistic follows a non-central F distribution with non-centrality parameter λ and two degrees of freedom from the numerator df_1 and the denominator df_2. When $f^2 = 0$, this becomes the null distribution of the central F-distribution. Thus, when the false positive rate is set to α, the threshold of F-statistic is

$$Fstat_{th} = Q_F(df_1, df_2, 1 - \alpha)$$

where Q_F is the quantile function of the central F distribution. The *false-negative* rate β can be computed as

$$\begin{aligned}\beta &= F_{nc}(Fstat_{th}, df_1, df_2, \lambda), \\ \lambda &= f^2(df_1 + df_2 + 1),\end{aligned}$$

where F_{nc} is the CDF function of the non-central F distribution. The statistical power for effect size f^2 is then $1-\beta$. Our test in Section 5.1 has $df_1 = \{256-7, 256-19, 256-16\}$, $df_2 = q-256$, $q = 20000$, per-test $\alpha = 10^{-3.7}$, which comes to $1-\beta \approx 1$ for the *small effect size* $f^2 = 0.02$ in [12]. According to [37] this corresponds indeed to what they observed in similar experiments.

Summarising, the F-Test on collapsed models has high power for relevant side channel scenarios according to [37], and assuming sufficient traces (20k in our calculation).

4 Application to Leakage Attacks

4.1 Unprotected AES

To demonstrate how this novel testing tool could be applied in practice, let us start our discussion with a trivial target: the first round Sbox look-up in an AES-128 encryption. For succinctness, we only analyse the first 4 Sbox look-ups (the analysis of the remaining S-box is identical).

Experimental setup We select the TinyAES [22] as our target implementation, running on an ARM Cortex M3 core (NXP LPC1313). As the code is written in C, the compiling toolchain and commercial core create a grey-box scenario: we can locate the Sbox computation from C, yet do not fully understand what is happening in each cycle on the power trace. The working frequency is set to 12 MHz, while our oscilloscope (Picoscope 5243D) captures 10k traces at 250 MSa/s (for both the collapsed case and the un-collapsed case). Altogether the 4 Sbox computations cost 17 μs, which counts to around 204 cycles and 4250 samples on the power trace.

Only computation leaks. A natural way to analyse such implementation is following the "only computation leaks" principle [28], whereby "computation" is architecturally defined.

We denote the first 4 bytes of the Sbox input as $\{x_0, x_1, x_2, x_3\}$ and the output as $\{s_0, s_1, s_2, s_3\}$. If the processor is computing the first Sbox, we should see the leakage of both x_0 and s_0 but nothing else. As the outputs are completely determined by the inputs, following the same principle in Section 3.1, we define the uncollapsed conservative model as

$$\tilde{L}(\hat{\mathbf{X}}) = \vec{\beta}\{\forall j \ u_j(\hat{\mathbf{X}}) | \hat{x} = x_0||x_1||x_2||x_3, \hat{x} \in \hat{\mathbf{X}}\}$$

Respectively, for the first Sbox, if "only computation leaks",

$$\tilde{L}(\mathbf{Z}) = \vec{\beta}\{\forall j \ u_j(\mathbf{Z}) | z = x_0, z \in \mathbf{Z}\}$$

As we do not restricted the degree of $\tilde{L}$, s_0 is fully determined by x_0, therefore already included by $\tilde{L}(\mathbf{Z})$. We now collapse each byte to 2 bits[3], the former model contains only 2^8 terms whereas the latter has only 2^2.

[3]Note that although $S(x_0)$ is still an 8-bit value, in a collapsed case, it only has 2-bit

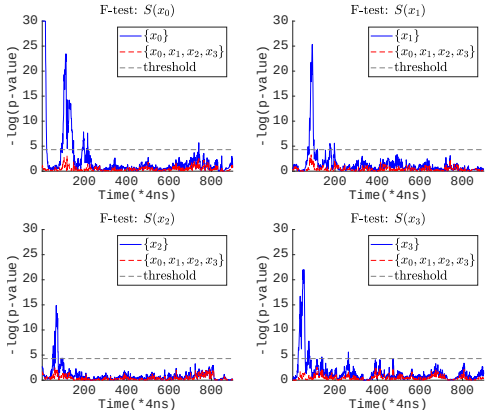


Figure 1: Collapsed F-test on each Sbox computation

Figure 1 illustrates the *F-test* results for all 4 Sbox computations. Clearly, architectural interpretation of "only computation leaks" is not enough: the F-test for the single byte models (blue line) all exceed the threshold, thus fail. This suggests any attack that is based on a single byte might be not optimal. Consequently, security evaluations/leakage certifications that are based on single bytes can be overly optimistic.

One step further. Now let us further investigate what variables may be missing. Through deliberately adding possible terms to the model and repeating the test, we found one acceptable leakage model that is always assuming each computation leaks all 4 bytes. The red line in Figure 1 shows that this model is not rejected.

With this new finding in mind, we revert back to the un-collapsed case: with the un-collapsed trace set, we plot the linear regression attack [15] results (targeting each Sbox output) with the correct key in Figure 2: as expected, each Sbox computation leaks mainly for the byte it is computing. However, from the second Sbox, we start to see a small peak (slightly above the threshold) for the first Sbox output. For the last byte, all previous three Sbox outputs have an influence, which is consistent with what we observed in Figure 1. The fact that a linear regression distinguisher can also find these leaks confirms the existence and exploitability of such leakage in realistic attacks[4]. This suggests that single byte attacks are far from optimal.

Architectural reasoning. It is possible to link our concrete statistical observations with some informed guesses about the processor architecture. For this

entropy and can be expressed with only 2 bits. Therefore $\tilde{L}(\mathbf{Z})$ can still portray any leakage caused by $S(x_0)$.

[4]Additional profiling is also possible where one could estimate coefficients separately to create proportional models for dedicated attacks which is however not the focus of this paper.

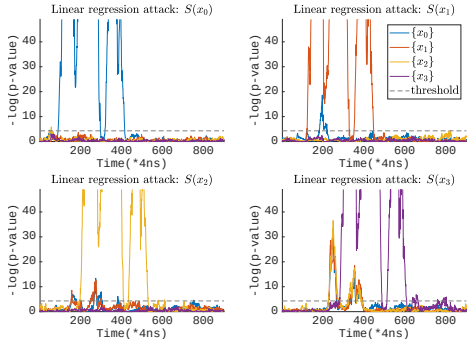


Figure 2: Linear regression attack for each Sbox computation

purpose, we examined the source code and noticed that the code in TinyAES [22] suggests the 4 S-box bytes are stored within one word, but not accessed adjacently in the Sbox look-ups (this explains why we did not see any Hamming distance style leakage). From the existing knowledge, we suspect the observed leakage is from the fact that the processor loads more than the target byte in a load instruction: as suggested in Section 5.2 of [25], it is indeed quite common to load the entire word from memory then discard the unnecessary bytes.

4.2 Masked AES: Sbox look-up

In the following, we further investigate the impact of our *F-test* on masked implementations. Specifically, we select the affine masking implementation from ANSSI [1] as our target. The security of this implementation has been questioned by Bronchain and Standaert [3]: through the so-called "dissection" procedure, they had successfully recovered the full key with less than 2000 traces in a profiling setup. In fact, the critical part of this attack is performing "sub-attacks" on each trace in order to recover the temporary masks [3]. Considering such implementation requires repetitively loading masks from the memory, this "dissection" procedure is often quite effective in a profiling setup, where the attacker can match the measured trace with a number of pre-built multivariate templates (eg. also applied by many attacks in the DPAContest [30]).

In the following, we investigate the leakage of this implementation in our experimental setup[5]. We stick with the same measurement setup as Section 4.1, while we are analysing the masked table look-ups of the first 4 Sboxes.

Note that the original implementation also includes hiding techniques, such as random shuffling. Unless stated otherwise, our following analysis always assume shuffling is not presented (i.e. "#define NO_SHUFFLE" in the code). Readers can take this as the non-shuffling analysis (which is an option in the

[5]The original *Compute_GTab* function contains a few instructions (eg. *uadd8*) that are not available on our platform. We had rewritten an equivalent version in pure Thumb-16 assembly. This makes no difference in our leakage analysis as we are not targetting at this part.

authors' analysis [1]), or take it as a follow-up analysis where the shuffling permutation has been already recovered using the technique in [3].

Affine Masking As this implementation is specific to AES, each data byte is protected as an element on the Galois Field $\mathbb{GF}(2^8)$. More specifically, each data byte x is presented as:

$$C(x; rm, ra) = (rm \otimes x) \oplus ra$$

where C is the encoding function, rm is called the *multiplicative mask* and ra the additive mask. Note that by definition, rm is uniform on $[1, 255]$ (i.e. cannot be 0). For the i-th state byte x_i, the implementation stores the additive mask ra_i accordingly in a mask array ra. The *multiplicative mask* rm, on the other hand, is shared among all 16 bytes within this encryption. Each linear operation (eg. ShiftRow, MixColumn) can be done separately on ra and x. Meanwhile, the masked Sbox is pre-computed according to the *multiplicative mask* rm and the Sbox input/output mask r_{in} and r_{out}:

$$S'(rm \otimes x \oplus r_{in}) = rm \otimes S(x) \oplus r_{out}$$

Code snippet for the Sbox In order to compute the Sbox's ouput using the pre-computed table, one must transfer the additive mask ra_i to r_{in}, then after the table look-up, transfer r_{out} back to ra_i. The *SubBytesWithMask* function performs this task as follow:

```
SubBytesWithMask:
...                     //r3=C(x) r10=ra
...                     //r0=i r8=S'
ldrb r4, [r3, r0]       //(1) r4=C(x)_i^rin
ldrb r6, [r10, r0]      //(2) r6=ra_i
eor  r4, r6             //(3) r4=C(x)_i^rin^ra_i
ldrb r4, [r8, r4]       //(4) r4=rmS(x)^rout
eor  r4, r6             //(5) r4=rmS(x)^rout^ra_i
strb r4, [r3, r0]       //(6) store r4 to state
...                     //removing rout later
```

Note that the r_{in} is added before this function, therefore line (1)-(3) purely focus on removing ra_i. Similarly, removing r_{out} is postponed to the end of the Sbox calculation, therefore not presented in this code.

Only computation leaks. Following the spirit of Section 4.1, we analyse the leakage of the first Sbox look-up and use 1 bit to represent each x_i. All random masks must also be considered in our leakage analysis: we use 6 bits to represent $ra_{0:3}$, r_{in} and r_{out} respectively. When collapsed to 1 bit, rm is restricted to 1

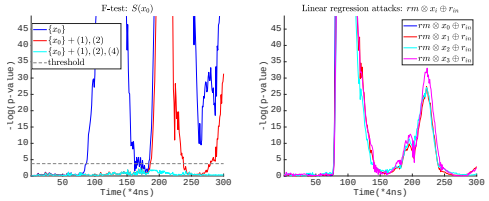


Figure 3: Leakage analysis for the first Sbox

(i.e. nullifies the protection of rm)[6]. Thus, we exclude this bit from our *F-test* and analyse the leakage where rm is set to 1. This means we will not cover any potential unexpected leakage introduced by rm in our experiment: of course, one can always switch to 2-bit version and use more traces to cover rm.

The complete model is therefore defined as

$$\tilde{L}(\hat{\mathbf{X}}) = \vec{\beta}\{\forall j \ u_j(\hat{\mathbf{X}}) | \hat{x} = x_{0:3} || ra_{0:3} || r_{in} || r_{out}, \hat{x} \in \hat{\mathbf{X}}\}$$

Following our common practice, it is expected that all the computed values are leaked plus some transitions. As a starter, let us first use some coarse-grained model that capturing all possible computations for the first Sbox:

$$\tilde{L}(\mathbf{Z}) = \vec{\beta}\{\forall j \ u_j(\mathbf{Z}) | z = x_0 || ra_0 || r_{in} || r_{out}, z \in \mathbf{Z}\}$$

Readers can easily verify that all the intermediate values appear in the code snippet can be expressed by this restricted model $\tilde{L}(\mathbf{Z})$. However, once again, we find this x_0-only model is hardly satisfying in the collapsed *F-test*: as we can see in Figure 3, the blue line clearly passes the threshold, which suggests the realistic leakage contains much more than what $\tilde{L}(\mathbf{Z})$ can express.

One step further. Following the same clue we found in Section 4.1, it is sensible to assume each *ldrb* loads not only the target byte, but also the other bytes within the same word. Thus, our line (1) loads:

$$\{rm \otimes x_0 \oplus ra_0, rm \otimes x_1 \oplus ra_1, rm \otimes x_2 \oplus ra_2, rm \otimes x_3 \oplus ra_3\}$$

Line (2) loads

$$\{ra_0, ra_1, ra_2, ra_3\}$$

If we add both these values (plus their transitions) into $\tilde{L}(\mathbf{Z})$, the red lines shows that the first peak around 100-150 is gone, suggesting the leakage has been safely captured in the collapsed model. However, the second half of the trace still presents some extra leakage.

[6]Note that this only applies to the collapsed case and the F-test: the following regression analyses and attacks are performed on the un-collapsed traces, where the protection of rm still applies.

Let us further consider line (4): if it also loads a word, then

$$\{S'(rm \otimes x_0 \oplus rin), S'(rm \otimes x_0 \oplus rin \oplus 1),$$
$$S'(rm \otimes x_0 \oplus rin \oplus 2), S'(rm \otimes x_0 \oplus rin \oplus 3)\}$$

The tricky bit of this word is its byte-order depends on rin, which varies from trace to trace. Therefore, if we calculate the memory bus transition leakage from line (2) to (4), the correct form can be complicated. Nonetheless, we can always create a conservative term $\mathbf{Z}_{a1}$ where $za_1 = x_0||rin||rout||ra_1$: adding $\vec{\beta}(\mathbf{Z}_{a1})$ covers all possible transitions between ra_1 and the Sbox output bytes from line(4), despite which byte it is transmitting to. Similarly, we add $\mathbf{Z}_{a2}$ and $\mathbf{Z}_{a3}$ to $\tilde{L}(\mathbf{Z})$ and construct a model that passes the *F-test* (i.e. the cyan line in the left half of Figure 3).

We further verify our inference from the *F-test—ldrb loads word and causes word-wise transitions*. In order to confirm such leakage does exist, we go back to the original un-collapsed implementation and perform a linear regression attack [15] on $rm \otimes x_i \oplus r_{in}$. In theory, *ldrb* should load x_0 only, which means only $rm \otimes x_0 \oplus r_{in}$ should be computed as for the masked table look-up. However, we did observe that the other 3 bytes also contribute to peaks on the regression results in the right half of Figure 3. To our knowledge, the most reasonable explanation is such leakage is from the transition from line (1) and (2), where the entire word is loaded in both cases.

Non-profiled attacks. The existence of leakage for $rm \otimes x_i \oplus r_{in}$ provides a clue for non-profiled attacks: as all 4 bytes leaks the same way around point 100, we can raise our measurements to their 2nd order moments, which cancels the influence of r_{in}. However, unlike the trivial Boolean masking schemes, now x_i (or $x_i \oplus x_{i+1}$) is still protected by rm. That being said, considering if we have a "collision" (aka $x_i = x_j$) within a word, we know for sure $rm \otimes x_i \oplus r_{in} = rm \otimes x_j \oplus r_{in}$ as both rm and r_{in} are shared among all bytes. Such restriction further affects the variance of the measured leakage, which could be exploited through 2nd order attacks.

Implementing this idea, we have tested 50 plaintexts that have collisions and 50 plaintexts without collision in the first word. Within each test, we perform a fixed-v.s.-random t-test and plot the minimal p-value in Figure 4. After 2500 fixed traces and 2500 random traces, nearly 90% of the collision cases can be identified, which confirms the validity of our analysis above.

It is not hard to see that such an oracle provides a link between each key bytes: in a chosen plaintext setup, attackers can manipulate the plaintext and learn information about the secret key. Ideally, if we found 3 collisions within the same word and figured out their collision indices through adaptively testing, the key space for each word can be reduced to 2^8. Repeat that with the other 3 words, we can enumerate the remaining 2^{32} key guess space and learn the full key. We leave the question of *what is the most efficient attack strategy* open, as it is out of the scope of this paper. Clearly, our attack strategy is no match for the attack in [3], however, our analyses could guide potentially even more

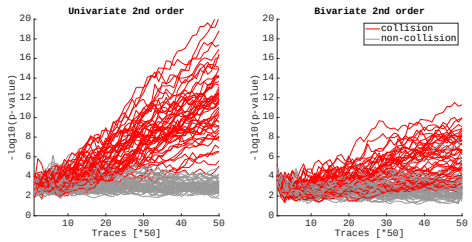


Figure 4: Collision Oracle

effective attacks that build proportional models guided by our nominal models. From a non-profiled perspective, it also considerably extends the attack scope from the authors' correlation analyses [1].

Notes. We list a few more intriguing facts about this implementation/attack which might be a worthwhile topic for future works:

- *Bivariate attacks.* A trivial alternative is to construct our oracle above with bivariate leakage (i.e. one sample for x_0 and one sample for x_1) and combine multiple points on the trace with the "mean-free" product. As we can see in the right half of Figure 4, this approach turns out to be less efficient. One possible explanation is combining 2 samples introduces two independent sources of noise.

- *Leakage for line (4).* At the first glance, the word-wise leakage for line (4) seems to be a better target. The entire word is masked with 1 byte rm, 1 byte r_{out} and contains 8-bit of secret key. In our experiment, we found the influence of rm to be onerous, at least in a non-profiled setup. However, as this leakage reveals a certain key-byte's value (v.s. reveals the key byte's relation to other key bytes), we leave the exploitability of such leakage as an open problem.

- *Avoiding leakage.* The exploited leakage above can be easily prevented, if the implementation loads something else between line (1) and (2). In other word, this is a specific implementation pitfall, not linked to the masking scheme itself. As a comparison, the bivariate version in the right half of Figure 4 is not affected by these subtle implementation details.

- *Link to t-test.* The exploited leakage can be found through 2nd order fixed-v.s.-random (or fixed-v.s.-fixed) t-test, suppose the selected fixed constant contains "collision". For a randomly selected constant, the probability that it has a "collision" in the first word is around 0.04, which poses again a question on the "coverage" of using 1 or 2 fixed constant(-s) in leakage detections [36].

4.3 A Case Study for Hardware

In this section we move on to another realistic scenario: the ASIC DES implementation that was the basis of the DPAContest [30]. As the goal of this trace set is validating attacks, for our purpose, it serves as a highly practical/realistic example.

The DPAContest website provides power traces for 2 unprotected DES cryptoprocessors in ASIC and one FPGA based implementation. We used the first data set, namely *secmatv1_2006_04_0809*. Our analysis for *"secmatv3_20070924"* lead to the same conclusions.

The DES core is running at 32 MHz, while the scope captures traces at 20 GSa/s. As a consequence, the captured traces cover a $1\,\mu s$ time period with 20000 measurement values, whereby each clock cycle contains 626.67 measurement values. To avoid any statistical influence from the input, we select the cycle when the third round encryption flips to the fourth round encryption (around index $[6893, 7520)$ in the traces). Considering the implementation is parallel, we further assume each S-box computation is independent from other concurrent S-boxes[7]. Our following analysis is limited to modelling the power consumption from the first DES S-box, while the power consumption from the other S-boxes simply becomes noise. We do not see this as a particular restriction because this is indeed a quite common choice for attacks/evaluations [17, 30], therefore our results can at least apply to those cases. The entire data set contains more than 80k traces: in our experiments, the first 60k were used for constructing models with our *F-test* methodology, while the last 20k traces serve as a cross-validation set.

4.3.1 Model comparison

We conduct a similar analysis as in Section 5: first check if various models have enough explanatory power via the *F-test*, and then analyse the models' predictive power via cross validation. Note that in the case of this hardware implementation, which corresponds to Figure **??**, because there is only one input, $\tilde{L}_{t-ext}$ is omitted ($\tilde{L}_{t-ext} = \tilde{L}_f$).

The upper half of Figure 5 plots the *F-test* results: clearly, transition leakage is important and should not be neglected[8]. In contrast to the case we studied in Section 5, the *F-test* suggests the combinatorial combinations do explain a significant part of the measured power consumption ($\tilde{L}_{TA}$). We believe this phenomenon is caused by two factors: firstly, the sampling rate here is 20 GSa/s, which might capture the temporal short-term power consumption better. Secondly, the DES S-box has a slightly more complicated critical path than the simple XOR in the ALU: using the input and output of the S-box alone can no longer portray all the internal combinatorial effects.

The lower half of Figure 5 plots the cross-validation results: due to the lower SNR, only a small number of samples provide a "useful" model. Most

[7]This assumption can be verified via an *F-test*.

[8]Note that part of this is from the round state register, not the S-box.

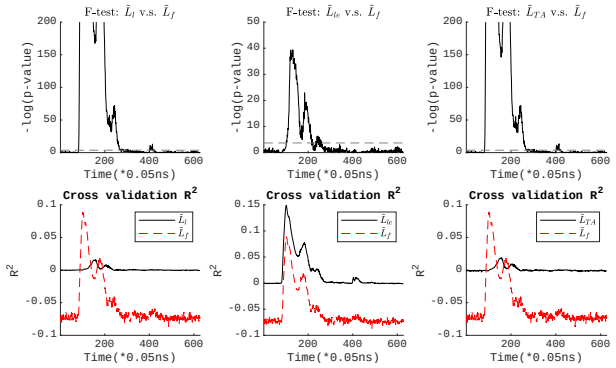


Figure 5: Comparing various models with traces from DPAContest 'SecMatV1 ASIC' implementation of DES

other samples lead to negative cross-validation R^2, which suggests significant over-fitting effects exist. Nonetheless, $\tilde{L}_{le}$ achieves the best predictive power, whilst it clearly has a low explanatory power according to the *F-test*.

4.3.2 Practical impact: attacks

One question remains from our discussion above: *does such unaccounted leakage contribute significantly to attack success*? From the attacker's perspective, we stress that *completeness* is not necessarily relevant: a leaking barrel will of course be "exploited" from the shortest stave, not from other "shorter-than-average" staves. To make things worse, not all "staves" are equal in a side channel attack: for example, exploiting Hamming distance leakage sometimes requires a larger key guessing space, which can quickly cancel out the advantage of capturing more leakage.

In the following experiments, we provide a profiling attack[9] based on the models in Figure 5: 60k traces were used for building the models. 1k/2k traces were then randomly picked from the remaining 20k traces for key recovery. The key guess can then be identified as the one corresponds to the highest prediction accuracy. To avoid enlarging the key space, we assume the previous S-box inputs are given: this is an unlikely case for DES, yet when the cipher has two adjacent independent round keys, attacking the second round key may lead to a similar scenario. Figure 6 portrays the average of the correct key rank after 100 attacks: as we are only targeting one key byte, the maximum key rank is 64.

Similar to Figure 5, we observe that linear models, especially $\tilde{L}_{le}$, perform reasonably well in our attacks. With less than 100 traces, the correct key guess is almost determined with certainty. This is consistent with the reports from [30]: many powerful attacks were built on profiled/non-profiled Hamming distance models, successfully recovering the full key with 100+ traces. The non-linear

[9] For the unprotected DES implementation in DPA Contest, there are however much easier attack options [30].

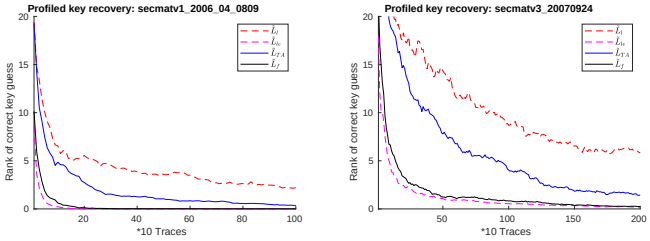


Figure 6: Profiled key recovery

combinatorial components can contribute (see $\tilde{L}_{TA}$ v.s. $\tilde{L}_l$), but the benefit will be counteracted by the over-fitting effect ($\tilde{L}_f$ v.s. $\tilde{L}_{le}$). Consistent with all previous results, transition leakage makes a significant contribution ($\tilde{L}_l$ v.s. $\tilde{L}_{le}$): this suggests if adding transitions does not enlarge the key guess space, one should prioritise models that contain transition leakage.

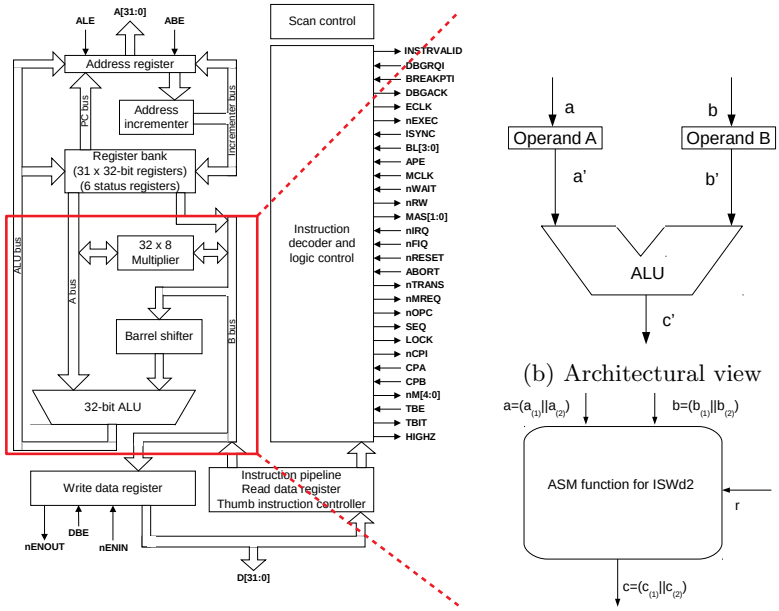
5 Application to Leakage Simulators

In recent years, various leakage simulators have been proposed in order to enable early-stage leakage awareness [4]. Using a leakage simulator, developers can identify and patch potential leakage at an early stage, even if they have no access to the target device. In this section, we utilise our new test to challenge existing leakage simulators that have either asserted models or estimated models.

Throughout this section, we use the same ARM Cortex M3 core as our target software platform. Each profiling trace set includes 20k traces to estimate and evaluate the model. The setup is the same as in Section 4, except for the working frequency which we reduced to 1 MHz: a lower frequency helps to have a clearer cycle-to-cycle view, which is essential for model building. Any model is ultimately data dependent: a proportional leakage model such as [26] represents the device leakage as it was captured by a specific setup. In particular, the explanatory variables in a leakage model relate to the (micro)architecture of the modelled processor, and the coefficients relate to the measurement setup. With our novel technique the emphasis is to build nominal models (models that initally at least only have 0/1 coefficients). By ensuring the processor runs at a frequency where we have a very clean view of the clock cycles, we are in a good position to build a nominal model. To build a proportional model on top of the recovered nominal model, it would be possible to estimate coefficients based on further measurement data (adjusted to the frequency that the target would run on in the wild).

5.1 Instruction-wise modelling

As pointed out in [4], one of the remaining challenge for grey-box simulators is "(they) target relatively simple architecture". In fact, many tools only target



(a) Substantial view

(b) Architectural view

(c) Abstract view on the code in Section 5.2

Figure 7: Different levels of abstraction for the M3 core.

algorithmic variables that may correspond to multiple intermediate states. Even if the simulator takes binary code as its input (eg. ELMO [26]), the undocumented micro-architectural effects can still cause all sort of issues [25]. Our novel statistical tool can help to identify missed leakage: if a leakage simulator fails our novel test, it suggests that some contributing factor is missing in its leakage model. Note that we can only fairly compare simulators that work on the same piece of code: considering our code is written in Thumb assembly, despite many leakage simulators exist in literature [4], the only options we have are the ELMO family [26, 32] and MAPS [13]. The ELMO*/ROSITA [32] model extended the ELMO model to cover various memory issues: as the following code does not utilise any memory instruction, it shares the same results as ELMO [26].

Architectural view Because the actual state is unknown, our analysis must be guided by (an abstraction of) the available information about the M3 architecture. Figure 7a shows a simplified architectural description for a realistic ARM M3 core [24]: whilst different realisations of an M3 adhere to this architecture, their micro-architectural features (such as buffers or registers) will differ. A common micro-architectural element for such a processor architecture would be some so-called pipeline registers: these are the input registers in Figure 7b. Thus we can map the entire red block in Figure 7a to Figure 7b.

Common instruction-wise model A common simplification in many previous grey-box simulators is focusing on the execute leakage within the ALU (i.e. Figure 7b instead of Figure 7a). This choice is quite reasonable: even if the processor has a multi-stage pipeline, we do not necessarily care about the leakage from fetching the instructions (as it is often not data-dependent[10]). Following our principles in Section 3.1, the reference full model for Figure 7b can be written as

$$\tilde{L}_f = \vec{\beta}\{AA'BB'\}$$

Note that the output value C is completely determined by A and B, therefore there is no need to add C into the model here. However, if further restrictions (eg. the leakage of A is linear) have been added, we might need to add C when necessary. In our experiments, we also consider the following leakage models that correspond to the most commonly used models in the existing literature:

$\tilde{L}_l = \vec{\beta}\{A, B, C\}_l$: this model is a linear function in the current inputs and output. Because of the linearity of the model, it is imperative to include the output here. E.g. if the circuit implements the bitwise-and function, the leakage on ab cannot be described by any linear function in a and b. In the existing literature this is often further simplified to the Hamming weight of just the output (aka the HW model).

$\tilde{L}_{le} = \vec{\beta}\{A, B, C, A', B', C', (dA), (dB), (dC)\}_l$, where $dA = A \oplus A', dB = B \oplus B', dC = C \oplus C'$: this model further includes Hamming distance information, which can be regarded as an extension for both the Hamming weight and the pure Hamming distance model (used in the MAPS simulator [13]); it therefore also generalises the ELMO model [26] which only fits a single dummy for the Hamming distance leakage.

$\tilde{L}_{TA} = \vec{\beta}\{AB\}$: this model represents template attacks [11], where all relevant current inputs are taken into consideration. In this model the output does not have to be included because we allow interactions between the input variables. This model can also be taken as a faithful interpretation of "only computation leaks" [28].

Target instruction. Before any further analysis, we craft a code snippet that can trigger the simplified leakage in Figure 7b, while not causing any other type of data-dependent leakage from other pipeline stages (i.e. *fetch* and *decode*):

```
eors  r2,r2            //r2=0
eors  r1,r3            //r1=a', r3=b'
nop
nop
eors  r5,r7            //r5=a,  r7=b **Target**
nop
nop
```

[10] Otherwise, the program has data-dependent branches, which should be checked through information flow analysis first.

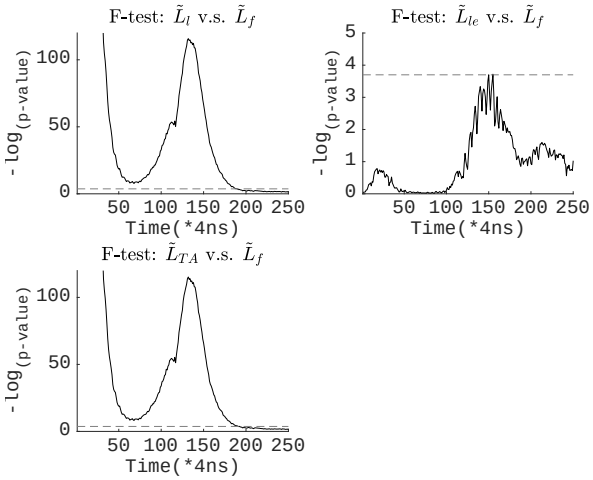


Figure 8: Comparing various models against $\tilde{L}_f$

eors r5,r7 represents the cycle we are targeting at in Figure 7b: the 2 pipeline registers are set to value a and b, where the previous values are a' and b'. a' and b' are set by *eors r1,r3*: since both lines use *eors*, a (b) and a' (b') should share the same pipeline register. The 2 *nop*-s before/after ensure all data-dependent leakage should be caused by *eors r5,r7*: in a 3-stage pipeline microprocessor, the previous XOR-s should already been committed and retired, while the fetcher/decoder should be executing *nop*-s (which in theory, does not cause any data-dependent leakage[11]).

Collapsed F-test Although we are working at an instruction level, because each operand has 32 bits, building the full model $\tilde{L}_f$ is still infeasible. Thus, we need to "collapse" $\tilde{L}_f$ to a smaller space. More specifically, we allow each operand to contain 2-bit randomness ($a = \{a_1a_2....a_1a_2\}$): comparing with the 1-bit strategy in Section 3.2, this option needs more traces to achieve reasonable statistical power. However, with 2-bit random operands we can distinguish whether the contribution of a specific term is linear or not, which is of interest when comparing existing simulators.

Figure 8 shows the *F-test* results: clearly, models that exclude transitions in the inputs massively exceed the rejection threshold. This means that in these cases we can conclude that the dropped transition terms have a statistically significant impact on the model. The linear model with transitions $\tilde{L}_{le}$ only marginally fails the test: thus it again demonstrates how significant the transitions are, but it also indicates that dropping higher-order terms does impact the quality of the leakage model.

Clearly, none of the three conventional models can be regarded as complete.

[11]In practice, this may depend on the content of $r8$ in ARM cores; our experiments had already set $r8$ to 0 beforehand.

Table 1: Leakage detection results on a 2-share ISW multiplication gadget

	Instruction	Device	ELMO	MAPS	$\tilde{L}_b$
0	$//r1 = a_{(1)}, r2 = a_{(2)}$ $//r3 = b_{(1)}, r4 = b_{(2)}, r5 = r$				
1	mov r6, r1(mov.w r6, r1 for MAPS)				
2	ands r6, r3$//r6 = a_{(1)}b_{(1)}$				
3	mov r7, r4(mov.w r7, r4 for MAPS)			✓	
4	ands r7, r2$//r7 = a_{(2)}b_{(2)}$				
5	ands r1, r4$//r1 = a_{(1)}b_{(2)}$	✓			✓
6	eors r1, r5$//r1 = a_{(1)}b_{(2)} \oplus r$	✓			✓
7	ands r2, r3$//r2 = a_{(2)}b_{(1)}$	✓	✓	✓	✓
8	eors r1, r2$//r1 = a_{(1)}b_{(2)} \oplus r \oplus a_{(2)}b_{(1)}$				
9	eors r6, r1$//c_{(1)} = a_{(1)}b_{(2)} \oplus r \oplus a_{(2)}b_{(1)} \oplus a_{(1)}b_{(1)}$	✓			✓
10	eors r7, r5$//c_2 = r \oplus a_{(2)}b_{(2)}$	✓	✓	✓	✓

As a consequence, simulators built on these models could miss leakage, due to the limited explanatory power of the respective leakage models. Various effects could be contributing here (including the bit-interaction [18]).

5.2 Gadget-wise modelling

Our novel testing methodology clearly shows that conventional models are not sufficient. But can we use it to develop better models?

In this section we extend our study to a more complex bit of code: masking gadgets. More specifically, we consider the Thumb-encoded 2-share ISW multiplication gadget that is given in the second column (under the header "Instruction") of Table 1. To avoid overloading notation, we denote the first share of input a as $a_{(1)}$. Considering this larger code sequence does not control the pipeline registers or the other components (eg. the decoding stage) in the processor, we need to be aware that our architectural view of the ALU (Figure 7b) may no longer be an adequate metal model for the actual reality in hardware (Figure 7a). In other words, it is expected that a *full* model that corresponds to Figure 7b is no longer *complete*.

Full model We now switch to a more abstract view: Figure 7c shows the functional view of our code in Table 1. Clearly, all functional inputs a, b and r no longer reflect any architectural port/bus/register. Having said that, assuming the processor starts from a constant state (in our experiments, ensured by clearing the data registers and memory buses before the function call), all leakage can still be bounded with all possible inputs. Thus, if both shares of a and b and r are collapsed to 2-bit, the *full* model can be defined as

$$\tilde{L}_f = \vec{\beta}\{A_{(1)}A_{(2)}B_{(1)}B_{(2)}R\}$$

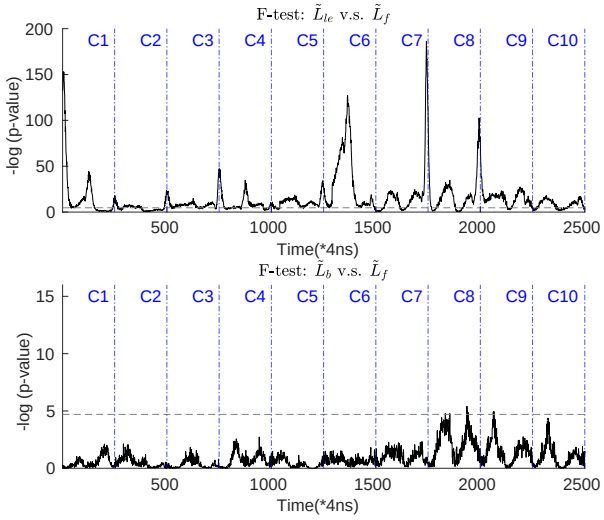


Figure 9: Model comparison based on a 2-share ISW multiplication in software

Collapsed F-test With a proper definition of $\tilde{L}_f$, we can again challenge $\tilde{L}_{le}$ in the context of this snippet. This model is of particular interest because the two most relevant (i.e. working with Thumb code on an ARM Cortex M processor) simulators ELMO [26] and MAPS [13] both use a subset of $\tilde{L}_{le}$. We recall that $\tilde{L}_l/\tilde{L}_{TA}$ was massively rejected in Figure 8), thus we only tested $\tilde{L}_{le}$ in the context of the more complex code snippet. The result is shown in the left picture in in Figure 9. We can see, rather unexpectedly, that the linear extended model is rejected by our test.

One step further. Similar to Section 4, we can try to build a better leakage model by adding terms and re-evaluating the model quality through the collapsed *F-test*. The final model out of this *ad-hoc* procedure is called $\tilde{L}_b$. Developing an architectural reasoning for this model is beyond the scope of this paper. However, Figure 9 shows that $\tilde{L}_b$ only marginally fails our test, and thus is considerably better than the linear extended model that many modern simulators use (with respect to our target processor). We build this model by observing that most operands influence the leakage for at least 2 cycles, which suggests that the decoding stage does significantly contribute to the data-dependent leakage. Consequently we include data from the decoding stage and this leads to $\tilde{L}_b$.

Comparing $\tilde{L}_b$. Whilst we now have a model that explains the device leakage of an M3 for a relatively complex gadget according to our novel test, it is still open if this better model helps to actually improve the simulator-based leakage detections. Thus we perform a leakage detection test (first order t-test) for the 2-share ISW implementation above, on realistic traces measured from our M3

core, traces from ELMO, traces from MAPS, and traces where we use $\tilde{L}_b$ to predict leakage. The last four columns in Table 1 show the resulting leakage detection test results.

MAPS captures all register transitions, including the pipeline registers in the micro-architecture (command line option "-p") [13]. MAPS reports 3 leaking instructions in our experiments: 2 are verified by the realistic 1st order t-test, while cycle 3 is not. Technically, this may not be a *false-positive* because MAPS is using the 32-bit instruction *mov.w* instead of the tested 16-bit instruction *mov*[12].

ELMO captures the operands and transitions on the ALU data-bus [26]: ELMO reports exactly the same leaking cycles as MAPS. Detailed analysis shows that both cycles leak information from their operands' transitions: ELMO captures these as data-bus transitions, while MAPS claims these as pipeline register transitions. Considering the pipeline registers are connected to the corresponding ALU data-bus, this is hardly unexpected[13].

Our manually constructed model leads to significantly better leakage predictions than both MAPS and ELMO as Table 1 shows. It reports the same leaking cycles as we found in the the real measurements. Specifically, cycle 5 reports a leakage from the ALU output bus transition (aka $C \oplus C'$ in Figure 7b), which is a part of $\tilde{L}_{le}$ but not covered by ELMO or MAPS. We suspect cycle 6 (1250-1500) and 9 (2000-2250) come from the decoding stage: they are merely a preview of the leakage of cycle 7 and 10.

Extrapolating from this example, it seems clear that building simulators based on insufficient models (in particular models that wrongly exclude parts of the state) leads to incorrect detection results.

6 Impact of Model Completeness on Security Proof Assumptions

It is critical to remember that the *completeness* property, relates to the question — *what is being leaked?*. This question is also fundamental to any security proof.

6.1 Security proofs

Security proofs too come with assumptions about leakage. There are range of different concepts that relate to leakage resilience in the literature, and a particularly well research concept is that of proofs in the proing model as it originated in [7]. It has become clear over the years that the type of probe that was put forward in this original work is not sufficient to cover leakage that can

[12] For some reason, MAPS seems to have a problem with the 16-bit *mov* instruction in our experiments.

[13] At this point we want to clarify that although ELMO is specifically designed for the M0 core architecture, and we are working with an M3 here, the parts of the architecture that relate to the instructions in our implementation are identical (to the best of our knowledge) to the M0.

Table 2: Leakage assumption from various models and tools (hardware gate-level)

	Name[Reference]	Leakage assumption
Proof: probing	standard probe [7]	$\tilde{L} = \boldsymbol{\beta}\{A, B, C\}$
	glitch-extended probe [9]	$\tilde{L} = \boldsymbol{\beta}\{AB\}$
	transition-extended probe [9]	$\tilde{L} = \boldsymbol{\beta}\{AA', BB', CC'\}$
	(1,1,0)-robust probe [9]	$\tilde{L} = \boldsymbol{\beta}\{AA'BB'\}$
Verification	REBECCA/COCO [6, 19]	$\tilde{L} = \boldsymbol{\beta}\{AA'BB'\}$
	maskVerif(HW) [10]	$\tilde{L} = \boldsymbol{\beta}\{AB, CC'\}$
	SILVER [5]	$\tilde{L} = \boldsymbol{\beta}\{AB\}$

be observed in practice. We show now how this can be rephrased as assumptions about the state in our notion.

Types of probes. We classify various concepts of probes by "what the attacker can learn by probing one unit/wire". For example, the standard ISW probing model [7] assumes the attacker can learn the value on a wire by placing a probe on it. In the following, we call this probe as the *standard probe*. Concepts like *non-interference* [8] and *strong non-interference* [8] also follow the same probing rule: probe one variable only reveals the value on this wire. *Glitch-extended probes* [9], however, allow the attacker learn all connected inputs from the last register layer. If the probing unit connects to k registered inputs, the attacker can potentially learn k values with only one probe. A probe can also be extended with *transition* leakage [9], where both the previous and current values will be given to the attacker. Finally, if a probe is extended by both glitch and transition, we use the notation from [9], denote it as *(1,1,0)-robust probe* (i.e. (transition,glitch,no coupling)). Of course, an attacker can place multiple probes and learn more values: using two standard probes may enable to learn the same information as one *transition* extended probe.

We now derive the leakage models that correspond to these different types of probes based on our exemplary circuits. It is important to bear in mind that the various types of probing models were defined with a "white box" gate level view. That is to say that there are no hidden elements in the circuit. If a circuit has only two input bits (A and B) and one output bit C the attacker can learn either A, B or C. Thus a standard probe can only capture a subset of

$$\tilde{L} = \boldsymbol{\beta}\{A, B, C\}.$$

When considering a glitch extended probe [9], probing C gives both A and B. If there are no hidden elements then C is fully determined by A and B, and thus we can omit the output C. Thus, the model of a glitch extended probe corresponds to

$$\tilde{L} = \boldsymbol{\beta}\{AB\}.$$

Similarly, when extended with *transition* leakage, A, B and C are both extended with their previous values. Namely, such probe captures a subset of

$$\tilde{L} = \boldsymbol{\beta}\{AA', BB', CC'\}.$$

Finally, when the probe is extended with both glitch and transitions, the adversary can potentially learn the *full* model:

$$\tilde{L} = \boldsymbol{\beta}\{AA'BB'\}.$$

Clearly the (1,1,0)-robust probing model contains all information that is available to an adversary for a simple (gate level) exemplary circuit. According to our discussion above, it is thus statistically *complete*. For clarity, we provide a full comparison in Table 2.

Theory vs. Practice The previous section strongly depends on the assumption that our exemplary circuit was a single gate with no hidden (micro-architectural) elements. If this assumption also holds in practice, then the (1,1,0)-probes capture all available information. However, this is likely to only ever hold in a white-box hardware context. This statement is particularly important when considering the use of probing based verification tools that operate on a "high level language": unless a verification tool is applied to a hardware netlist, the resulting security proof does not necessarily give any security for the resulting implementation. This is because the implementation may run on a device that contains unknown micro-architectural elements, or types of leakage that were not considered by the proof.

7 Discussion and conclusion

This paper puts the *state* that is captured by a leakage model at the centre stage. We put forward the novel notion of "*completeness*" for a model. A model is *complete* if it captures all relevant state information, thus suitable to be the basis for leakage simulators or security evaluations.

Deciding if a model is *complete* or not initially seems like an impossible task in the case of modern processors. Even for a 2-operand instruction, if we take previous values into account, there are 2^{4n} values to take into account. For $n = 8$ (i.e. in a small micro-controller), it is computationally expensive; but for $n = 32$ (i.e. in a modern microprocessor), it becomes clearly infeasible. We overcome this problem by introducing a novel statistical technique using *collapsed* models as part of a nested *F-test* methodology. This test is robust and effective as we illustrate based on a range of concrete experiments. As a bonus, our novel methodology helps to find a statistically *complete* model for a given device efficiently, with minimal device knowledge (all our examples in this paper are in a grey box setting).

Beyond this novel test, we affirm a range of important points when it comes to attacks and simulations:

- We have shown that in some reported best attacks, some leakage was missed.
- Predictive models are not guaranteed to capture all relevant leakage. Therefore accuracy metrics like cross-validation R^2 or SSE alone should not be the basis for a security certification in the style of [17, 2, 23].
- A complete model is essential for leakage simulators: the detection accuracy can be significantly affected if the leakage is overly-simplified compared with realistic measurements.
- Whilst we do not cover leakage detection explicitly in our paper, there are clear implications for detection from our findings. So-called "specific leakage detection" [20] relies on specifying a leakage model (and therefore *state*). Clearly a leakage detection that is based on an incomplete *state* can miss out leaks.

Acknowledgments

The authors were funded in part by the ERC via the grant SEAL (Project Reference 725042).

References

[1] Ryad Benadjila, Louiza Khati, Emmanuel Prouff, and Adrian Thillard. Hardened library for aes-128 encryption/decryption on arm cortex m4 achitecture.

[2] Olivier Bronchain, Julien M. Hendrickx, Clément Massart, Alex Olshevsky, and François-Xavier Standaert. Leakage certification revisited: Bounding model errors in side-channel security evaluations. In Alexandra Boldyreva and Daniele Micciancio, editors, *Advances in Cryptology - CRYPTO 2019 - 39th Annual International Cryptology Conference, Santa Barbara, CA, USA, August 18-22, 2019, Proceedings, Part I*, volume 11692 of *Lecture Notes in Computer Science*, pages 713–737. Springer, 2019.

[3] Olivier Bronchain and François-Xavier Standaert. Side-channel countermeasures' dissection and the limits of closed source security evaluations. *IACR Trans. Cryptogr. Hardw. Embed. Syst.*, 2020(2):1–25, 2020.

[4] Ileana Buhan, Lejla Batina, Yuval Yarom, and Patrick Schaumont. Sok: Design tools for side-channel-aware implementions, 2021.

[5] Knichel, D., Sasdrich, P., Moradi, A.: SILVER - statistical independence and leakage verification. In Moriai, S., Wang, H., eds.: Advances in Cryptology - ASIACRYPT 2020 - 26th International Conference on the Theory and Application of Cryptology and Information Security, Daejeon, South

Korea, December 7-11, 2020, Proceedings, Part I. Volume 12491 of Lecture Notes in Computer Science., Springer (2020) 787–816

[6] Bloem, R., Groß, H., Iusupov, R., Könighofer, B., Mangard, S., Winter, J.: Formal verification of masked hardware implementations in the presence of glitches. IACR Cryptol. ePrint Arch. **2017** (2017) 897

[7] Ishai, Y., Sahai, A., Wagner, D.A.: Private circuits: Securing hardware against probing attacks. In Boneh, D., ed.: Advances in Cryptology - CRYPTO 2003, 23rd Annual International Cryptology Conference, Santa Barbara, California, USA, August 17-21, 2003, Proceedings. Volume 2729 of Lecture Notes in Computer Science., Springer (2003) 463–481

[8] Barthe, G., Belaïd, S., Dupressoir, F., Fouque, P., Grégoire, B., Strub, P., Zucchini, R.: Strong non-interference and type-directed higher-order masking. In Weippl, E.R., Katzenbeisser, S., Kruegel, C., Myers, A.C., Halevi, S., eds.: Proceedings of the 2016 ACM SIGSAC Conference on Computer and Communications Security, Vienna, Austria, October 24-28, 2016, ACM (2016) 116–129

[9] Faust, S., Grosso, V., Pozo, S.M.D., Paglialonga, C., Standaert, F.: Composable masking schemes in the presence of physical defaults & the robust probing model. IACR Trans. Cryptogr. Hardw. Embed. Syst. **2018**(3) (2018) 89–120

[10] Barthe, G., Belaïd, S., Cassiers, G., Fouque, P., Grégoire, B., Standaert, F.: maskverif: Automated verification of higher-order masking in presence of physical defaults. In Sako, K., Schneider, S.A., Ryan, P.Y.A., eds.: Computer Security - ESORICS 2019 - 24th European Symposium on Research in Computer Security, Luxembourg, September 23-27, 2019, Proceedings, Part I. Volume 11735 of Lecture Notes in Computer Science., Springer (2019) 300–318

[11] Suresh Chari, Josyula R. Rao, and Pankaj Rohatgi. Template attacks. In *Cryptographic Hardware and Embedded Systems - CHES 2002, 4th International Workshop, Redwood Shores, CA, USA, August 13-15, 2002, Revised Papers*, pages 13–28, 2002.

[12] Jacob Cohen. Chapter 9 - f tests of variance proportions in multiple regression/correlation analysis. In Jacob Cohen, editor, *Statistical Power Analysis for the Behavioral Sciences*, pages 407 – 453. Academic Press, 1977.

[13] Yann Le Corre, Johann Großschädl, and Daniel Dinu. Micro-architectural power simulator for leakage assessment of cryptographic software on ARM cortex-m3 processors. In Junfeng Fan and Benedikt Gierlichs, editors, *Constructive Side-Channel Analysis and Secure Design - 9th International*

Workshop, COSADE 2018, Singapore, April 23-24, 2018, Proceedings, volume 10815 of *Lecture Notes in Computer Science*, pages 82–98. Springer, 2018.

[14] Yves Crama and Peter L. Hammer, editors. *Boolean Models and Methods in Mathematics, Computer Science, and Engineering*. Cambridge University Press, 2010.

[15] Julien Doget, Emmanuel Prouff, Matthieu Rivain, and François-Xavier Standaert. Univariate side channel attacks and leakage modeling. *J. Cryptogr. Eng.*, 1(2):123–144, 2011.

[16] François Durvaux, François-Xavier Standaert, and Santos Merino Del Pozo. Towards easy leakage certification: extended version. *J. Cryptogr. Eng.*, 7(2):129–147, 2017.

[17] François Durvaux, François-Xavier Standaert, and Nicolas Veyrat-Charvillon. How to certify the leakage of a chip? In Phong Q. Nguyen and Elisabeth Oswald, editors, *Advances in Cryptology - EUROCRYPT 2014 - 33rd Annual International Conference on the Theory and Applications of Cryptographic Techniques, Copenhagen, Denmark, May 11-15, 2014. Proceedings*, volume 8441 of *Lecture Notes in Computer Science*, pages 459–476. Springer, 2014.

[18] Si Gao, Ben Marshall, Dan Page, and Elisabeth Oswald. Share-slicing: Friend or foe? *IACR Transactions on Cryptographic Hardware and Embedded Systems*, 2020(1):152–174, Nov. 2019.

[19] Barbara Gigerl, Vedad Hadzic, Robert Primas, Stefan Mangard, and Roderick Bloem. Coco: Co-design and co-verification of masked software implementations on cpus. *IACR Cryptol. ePrint Arch.*, 2020:1294, 2020.

[20] 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.

[21] Paul C. Kocher, Joshua Jaffe, and Benjamin Jun. Differential power analysis. In *Advances in Cryptology - CRYPTO '99, 19th Annual International Cryptology Conference, Santa Barbara, California, USA, August 15-19, 1999, Proceedings*, pages 388–397, 1999.

[22] kokke. Tiny aes in c.

[23] Liran Lerman, Nikita Veshchikov, Olivier Markowitch, and François-Xavier Standaert. Start simple and then refine: Bias-variance decomposition as a diagnosis tool for leakage profiling. *IEEE Trans. Computers*, 67(2):268–283, 2018.

[24] ARM Limited. Arm7tdmi technical reference manual. https://developer.arm.com/documentation/ddi0210/c/, 2004.

[25] Ben Marshall, Dan Page, and James Webb. Miracle: Micro-architectural leakage evaluation. *IACR Cryptol. ePrint Arch.*, 2021. `https://eprint.iacr.org/2021/261`.

[26] David McCann, Elisabeth Oswald, and Carolyn Whitnall. Towards practical tools for side channel aware software engineering: 'grey box' modelling for instruction leakages. In *26th USENIX Security Symposium (USENIX Security 17)*, pages 199–216, Vancouver, BC, 2017. USENIX Association.

[27] Lauren De Meyer, Elke De Mulder, and Michael Tunstall. On the effect of the (micro)architecture on the development of side-channel resistant software. *IACR Cryptol. ePrint Arch.*, 2020:1297, 2020.

[28] Silvio Micali and Leonid Reyzin. Physically observable cryptography (extended abstract). In Moni Naor, editor, *Theory of Cryptography, First Theory of Cryptography Conference, TCC 2004, Cambridge, MA, USA, February 19-21, 2004, Proceedings*, volume 2951 of *Lecture Notes in Computer Science*, pages 278–296. Springer, 2004.

[29] Kostas Papagiannopoulos and Nikita Veshchikov. Mind the gap: Towards secure 1st-order masking in software. In Sylvain Guilley, editor, *Constructive Side-Channel Analysis and Secure Design - 8th International Workshop, COSADE 2017, Paris, France, April 13-14, 2017, Revised Selected Papers*, volume 10348 of *Lecture Notes in Computer Science*, pages 282–297. Springer.

[30] Télécom ParisTech. Dpa contest 2008/2009.

[31] Werner Schindler, Kerstin Lemke, and Christof Paar. A stochastic model for differential side channel cryptanalysis. In Josyula R. Rao and Berk Sunar, editors, *Cryptographic Hardware and Embedded Systems - CHES 2005, 7th International Workshop, Edinburgh, UK, August 29 - September 1, 2005, Proceedings*, volume 3659 of *Lecture Notes in Computer Science*, pages 30–46. Springer, 2005.

[32] Madura A. Shelton, Niels Samwel, Lejla Batina, Francesco Regazzoni, Markus Wagner, and Yuval Yarom. Rosita: Towards automatic elimination of power-analysis leakage in ciphers. *CoRR*, abs/1912.05183, 2019.

[33] Galit Shmueli. To explain or to predict? *Statist. Sci.*, 25(3):289–310, 08 2010.

[34] François-Xavier Standaert, Tal Malkin, and Moti Yung. A unified framework for the analysis of side-channel key recovery attacks. In Antoine Joux, editor, *Advances in Cryptology - EUROCRYPT 2009, 28th Annual International Conference on the Theory and Applications of Cryptographic Techniques, Cologne, Germany, April 26-30, 2009. Proceedings*, volume 5479 of *Lecture Notes in Computer Science*, pages 443–461. Springer, 2009.

[35] Carolyn Whitnall and Elisabeth Oswald. Profiling DPA: efficacy and efficiency trade-offs. In Guido Bertoni and Jean-Sébastien Coron, editors, *Cryptographic Hardware and Embedded Systems - CHES 2013 - 15th International Workshop, Santa Barbara, CA, USA, August 20-23, 2013. Proceedings*, volume 8086 of *Lecture Notes in Computer Science*, pages 37–54. Springer, 2013.

[36] Carolyn Whitnall and Elisabeth Oswald. A cautionary note regarding the usage of leakage detection tests in security evaluation. Cryptology ePrint Archive, Report 2019/703, 2019.

[37] Carolyn Whitnall and Elisabeth Oswald. A critical analysis of ISO 17825 ('testing methods for the mitigation of non-invasive attack classes against cryptographic modules'). In *Advances in Cryptology - ASIACRYPT 2019 - 25th International Conference on the Theory and Application of Cryptology and Information Security, Kobe, Japan, December 8-12, 2019, Proceedings, Part III*, pages 256–284, 2019.

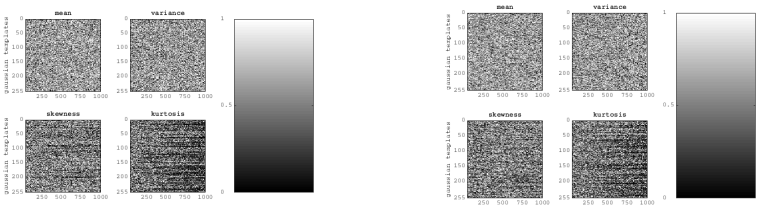
[38] Carolyn Whitnall, Elisabeth Oswald, and François-Xavier Standaert. The myth of generic dpa...and the magic of learning. In *Topics in Cryptology - CT-RSA 2014 - The Cryptographer's Track at the RSA Conference 2014, San Francisco, CA, USA, February 25-28, 2014. Proceedings*, pages 183–205, 2014.

A PI, HI & Assumption error

Leakage certification approaches such as described in [17, 16, 2] (based on the general framework introduced by Standaert, Malkin and Yung [34]) aim at providing guarantees about the quality of an evaluation, based on estimating the amount of information leaked by a target device.

In order to estimate the amount of leaked information (i.e. the mutual information), the intermediate state must be selected as a first step. In our notation, this means the user must correctly provide an enumerable state $\mathbf{Z}$ that ensures the corresponding model $\tilde{L}(\mathbf{Z})$ is close to the *full* model $\tilde{L}(\mathbf{X})$ w.r.t. its explanatory power. Then, one can estimate the mutual information of $MI(\mathbf{Z}; L)$ using concepts like perceived information (PI) or hypothetical information (HI) [2].

The common choice for $\mathbf{Z}$ is often a variable that relates to a single S-box [17, 16, 2]: because the MI calculation runs through all possible values of $\mathbf{Z}$, it corresponds to a template attack. This extremely popular choice is potentially inadequate because the device state is likely to be considerably more complex (as we have argued before), and it will likely include at least transition leaks, which cannot be captured in this way. Consequently, prior to any of these leakage certification approaches, it is imperative to test what state must be considered.



(a) HD leakage without any noise (b) HD leakage with noise variance 0.1

Figure 10: Moment based detection of "assumption error"

A.1 Estimating "assumption errors"

In [17] Durvaux et al. proposed a technique to test for (the so-called) assumption errors in the leakage model [17]. One could be tempted to regard this as an alternative solution for testing *completeness*. Unlike our *F-test*, their approach is based on checking if the distance between pairs of simulated samples (generated with a profiled model) and the distance between simulated and actual samples behave differently.

However, their technique of checking assumption errors is about ensuring that the estimation of MI is accurate. In order words, their technique is not an effective way to test whether $\mathbf{Z}$ is complete or not. To demonstrate this, we present a simple experiment that is based on the common example of leakage from an AES S-box output ($S(p_1 \oplus k_1)$, where p_1 is the plaintext byte and k_1 is the corresponding key byte). Let us further assume that the leakage function L depends on not only on $S(p_1 \oplus k_1)$, but also the previous S-box output $S(p_0 \oplus k_0)$:

$$L = HW(S(p_1 \oplus k_1)) + HD(S(p_1 \oplus k_1), S(p_0 \oplus k_0)).$$

Taking advantage of the code from [16], we can validate the power of detecting the above "assumption error": Figure 10a portrays the moment-based estimation on the leakage function above in a noise free setting. Each line corresponds to a model value, and if any value leads to a line that keeps getting "darker", it would suggest the p-value is small enough to confidently report an "assumption error". Even if there is no noise (left figure), only the *kurtosis* marginally reports errors. With some small noise added in (Figure 10b), the situation remains the same. Only the *kurtosis* gives some small p-values, but there is no statistical decision criterion that enables us to draw a firm conclusion here. This outcome should not be surprising. Because p_0 is an independent random variable, the Hamming distance part follows Binomial distribution $B\left(\frac{n}{2}, \frac{n}{4}\right)$ where n is the bit-length of p_0 (for AES, $n = 8$). With $\boldsymbol{Z} = P_1$, the estimated model would be:

$$M = HW(S(p_1 \oplus k_1)) + \mathcal{N}\left(\frac{n}{2}, \frac{n}{4}\right)$$

where $\mathcal{N}(\mu, \sigma^2)$ represents the Gaussian distribution. For any fixed value of

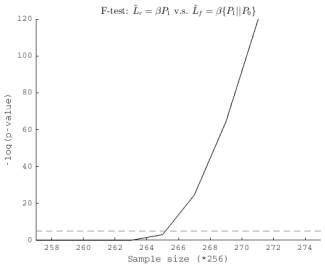


Figure 11: F-test with noise variance 0.1

$p_1 \oplus k_1$, the "distance between pairs of simulated samples" becomes

$$D_M = \left\{ l_1 - l_2 | l_1 \in \mathcal{N}\left(\frac{n}{2}, \frac{n}{4}\right), l_2 \in \mathcal{N}\left(\frac{n}{2}, \frac{n}{4}\right) \right\}$$

Meanwhile, "the distance between simulated and actual samples" becomes:

$$D_{LM} = \left\{ l_1 - l_2 | l_1 \in \mathcal{N}\left(\frac{n}{2}, \frac{n}{4}\right), l_2 \in B\left(0.5, n\right) \right\}$$

It is well-known that with reasonably large n, the binomial distribution will asymptotically approximate the Gaussian distribution. The idea behind this test in [17] is based on an expected inconsistency between the unexplained leakage distribution and estimated Gaussian distribution: the test becomes powerless if the former equals/stays close to Gaussian, which is not really a rare case in side channel applications.

In contrast, our *F-test* can detect such an "error" with ease, see Fig. 11. The advantage here requires though to explicitly assign $\mathbf{X} = \{P_1 P_0\}$. Without some guess work (or a priori knowledge) one may need to use a *collapsed full* model instead, say using 1 bit for each plaintext byte and testing on a trace set larger than 2^{16}.

We want to emphasize at this point that these previous works did not aim for testing the *completeness* of the state as such, so our findings do not invalidate their statements. We merely wish to point out that there is a difference between their ideas of "assumption errors" and our notion of "*completeness*".

A.2 HI&PI

Bronchain, Hendrickx and Massart et al. proposed that using the concepts of *Perceived Information* (PI) and *Hypothetical Information* (HI), one can "bound the information loss due to model errors quantitatively" by comparing these two metrics, estimate the true unknown MI and obtain the "close to worst-case" evaluations [2].

It is critical to remember the "worse-case" are restricted the computed MI: back to previous our example, estimating HI and PI still bound the correct mutual information $MI(K_1; P_1, L)$. The additional Hamming distance term

affects how we should interpret this metric: when combing multiple key-bytes to obtains the overall security-level, $MI(K_1; P_1, L)$ might not be as helpful as one may hope.

More concretely, we tested our example simulation leakage with the code provided in [2]: as we can see in Figure 12, PI and HI still bounds the correct MI. The only difference here is MI itself decreases as P_0 and K_0 are not taken into consideration.

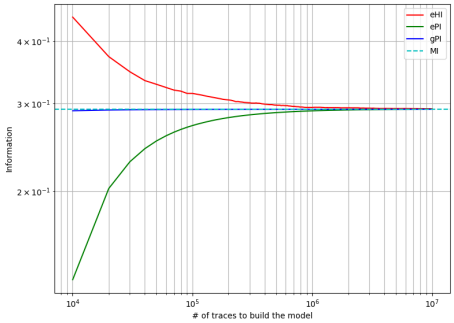


Figure 12: PI and HI estimation for the leakage function

A.3 Bias-Variance Decomposition

Lerman, Veshchikov and Markowitch et al. also proposed a diagnosis tool based on the bias-variance decomposition [23]. The goal of their tool is purely predictive—"guiding the design of the best profiled attack". In other words, the "syndrome to diagnose" is still restricted to the specific selected intermediate state. In our example, the additional Hamming distance will be taken as part of the random noise. Admittedly, unless the missing Hamming distance is taken into the model building procedure, any corresponding leakage will always end up in the noise. Therefore, any model can be perfectly estimated, yet that does not guarantee it is *complete*, as the estimated noise is not necessarily pure *measurement noise*.