A Novel Completeness Test for Leakage Models and its Application to Side Channel Attacks and Responsibly Engineered Simulators

Si Gao and Elisabeth Oswald

Digital Age Research Center (D!ARC), University of Klagenfurt, Austria
`firstname.lastname@aau.at`

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 not only large, but also hard to predict due to various micro-architectural factors that users might not be aware of. On the other hand, security evaluations— basing on worst case attacks or simulators — explicitly rely on the underlying *state*: a potentially *incomplete state* can easily lead to 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 then demonstrate the application of this test in the context of leakage modelling for leakage simulators and confirm that even the most recent leakage simulators do not capture all available leakage of their respective target devices. Our new test enables finding nominal models that capture all available leakage but do not give a helping hand to adversaries. Thereby we make a first step towards leakage simulators that are responsibly engineered.

1 Introduction

Leakage models are crucial not only for attacks and leakage simulators, but also for implementing masking schemes. Having a complete, accurate leakage model, experienced cryptographic engineers can diligently examine if the security assumptions are respected and avoid potential implementation pitfalls.

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 few traces. Thus, a lot of research has gone into deriving good proportional or direct estimates for leakage functions from real device data [1,2,3,4,5]. However, the leakage function itself is only part of what constitutes a leakage model: the *state* that is being leaked on is equally relevant.

In a realistic setting (such as typical 32-bit processors based on the ARM Cortex M architecture, or dedicated hardware implementations of cryptographic algorithms), finding the *state* is far from easy — not only because they tend to be closed source. Even if open source descriptions are available, e.g. ARM released some semi-obfuscated VHDL descriptions of the M0 and a non-obfuscated VHDL description of an implementation of their M3 architecture, these are not necessarily micro-architecturally equivalent to the corresponding commercial products on the market. Micro-architectural effects have been explored and exploited across many recent works [6,7,8,9,10]. These papers show how a wrong assumption about the *state* renders provably secure masking schemes completely insecure in practice. Leakage models matter thus for implementing provably secure masking schemes because they help engineers to ensure the security assumptions are respected. Leakage models can also guide an evaluator to demonstrate attacks for specific implementations. Last but not least, they are fundamental to a prevalent type of early-stage evaluation tool— leakage simulators.

1.1 State of the art

Leakage modelling, or profiling, has been an active area of research within the side channel community since its very beginning. Typical models are based on some assumption about what leaks (the *state*) and how it leaks (the function form in regression models, or the number of templates in direct parameter estimation), and then the corresponding coefficients are determined from the data. Typical examples of such profiling approaches are [1,2,3,4,5].

A leakage simulator is a tool that takes a software implementation of a cryptographic algorithm (in e.g. C or Assembly) as input, then outputs a leakage trace that is meant to capture (ideally) all the leakage characteristics of a target device. Various leakage simulators have been built within the last decade, e.g. industrial examples include PINPAS [11] (no device specific leakage model, limited to simple 8 bit devices), esDynamic [12] (multiple platforms and leakage models, but no device specificity), Virtualyzr [13] (needs the HDL description, no specific leakage model); academic examples include [14], [15] and [16]. Both sides develop the concept and recognise the importance of both accurate device emulation as well as leakage models.

Only relatively recently, a profiling approach was developed in the context of the ELMO simulator [5], which enabled some progress towards capturing complex leaks[1]. According to the authors' estimation, the ELMO model captures over 80% of the data dependent power consumption of the modelled processor [5,17]. However the ELMO approach cannot be pushed further, as it only captures limited micro-architecture states (e.g. operand buses and bitflips [5])

[1] Estimating leakage profiles in the context of multiple interacting 32-bit variables requires a non-trivial approach because the implied data complexity for naively estimating templates is infeasible.

and their "lower order" interactions. There is another potential drawback: the ELMO model demonstrates (in [5]) various non-trivial leaks (i.e. from covert micro-architecture operations) that can be efficiently exploited by attackers in practice. The natural question is then: is it ethical to make highly accurate predictive leakage models publicly available (in other words, release them to potential adversaries)?

1.2 Our contributions

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-test*, 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 overlooked. 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 leakage simulators and demonstrate that our approach can lead to better models for simulations.
6. We discuss the importance of responsibly engineering leakage simulators and put forward a first promising step in this direction.

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 6.

2 Preliminaries

2.1 Notation and background to leakage modelling

We aim for a simple notation throughout this paper: observed leakage is treated in a univariate fashion, therefore we drop indices for time points[2].

We call the set $\mathbf{X}$ the entire device state (at some time point). $\mathbf{X}$ is comprised of all input/key 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 value(-s). Following the trajectory of [5,18], we consider univariate models, thus we write $L : \mathbb{F}_2^n \to \mathbf{R}$ and $x \in \mathbb{F}_2^n$.

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. the Instruction Set Architecture, ISA) and we can execute arbitrary code during profiling, but we do not have the concrete gate-level hardware details (i.e. a typical software implementation with a commercial core). The relevant device state $\mathbf{X}$ (for a certain time index) is unknown in this setting, as $\mathbf{X}$ may contain various micro-architecture elements that are transparent to developers. We can of course, build an overly conservative model using all possible *state*s $\hat{\mathbf{X}}$ where $\mathbf{X} \subset \hat{\mathbf{X}}$ [3]. However, such a model is clearly undesirable for both attacks/evaluations (because it requires guessing the entire key) and simulators (because its estimation requires an impractical amount of real device data).

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 a state with small cardinality (e.g. $\#\{\mathbf{Z}\} = 2^8$ for AES).

The second step is to estimate the coefficients of the leakage function assuming it only acts on $\mathbf{Z}$. We use the standard notation $\tilde{L}$ to indicate the estimation of L. Various techniques have been proposed, including naive templating [1], regression-based modelling [2,3], step-wise regression [19], etc. Previous works [19,2,20] 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

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

[3] $\hat{\mathbf{X}}$ contains all intermediate variables that occur during the *entire* execution of an instruction sequence or algorithm

in details, then we move on to our main topic: *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. [1]. 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 this distribution.

An alternative to the direct parameter estimation is using regression techniques to derive an equivalent model. A paper by Schindler et al. [2] 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 $\tilde{L} = \sum_j \beta_j u_j(\mathbf{Z})$ [21]. 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). For clarity, we further define a mapping function $\mathbf{U}$ that maps the n-bit state $\mathbf{Z}$ to a 2^n-length vector:

$$\mathbf{U}(\mathbf{Z}) = (u_0(\mathbf{Z}), u_1(\mathbf{Z}), ..., u_{2^n-1}(\mathbf{Z}))$$

In the following, we can simply use $\tilde{L}(\mathbf{Z}) = \vec{\beta}\mathbf{U}(\mathbf{Z})$ instead of $\tilde{L}(\mathbf{Z}) = \sum_j \beta_j u_j(\mathbf{Z})$.

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* as

$$\tilde{L}_l(\mathbf{Z}) = \vec{\beta}\mathbf{U}_l(\mathbf{Z}) = \vec{\beta}(u_{2^0}(\mathbf{Z}), u_{2^1}(\mathbf{Z}), , ..., u_{2^{n-1}}(\mathbf{Z}))$$

2.3 Judging 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 [22]. 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, the goal of building these models is 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* (aka the *sum of squared estimate of errors, SSE*),

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

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* (aka the *sum of squares due to regression*, SSR),

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

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* (aka SST),

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

Then, the *coefficient of determination* (R^2) is defined as[4]:

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

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), intuitively, 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 [20] 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 contains a subset of the terms of the full model), the *F-test* is the most natural way to decide whether the *restricted* model is

[4] Alternatively, it can also be written as $R^2 = 1 - \frac{SSE}{SST}$. Note the distinction between *RSS* (i.e. "R" for *residue*) and *SSR* (i.e. "R" for *regression*)

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 below 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 [5] to derive relatively fine grained models on selected intermediate states.

2.4 Leakage certification techniques

Leakage certification techniques, e.g. "assumption error" and "estimation error" [20] are also designed under the assumption that $\mathbf{Z}$ is given. One could be tempted to use such techniques to test if the selected state $\mathbf{Z}$ is "good enough" : however, neither does this fit with the original intention of [20] nor is the statistical power of the "leakage certification" 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?

It is critical to remember that all approaches for judging model quality assume that the state $\mathbf{Z}$ has already been found. As we argued in the introduction, in practice, this state is not only unknown but also non-trivial to determine.

In existing publications, the suitable state $\mathbf{Z}$ is often found through an *ad hoc* procedure: one tries some set $\mathbf{Z}$, and then evaluates the leakage model (using $\mathbf{Z}$) via R^2 or cross-validation (or alternatively, performs 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 it remains unclear if $\mathbf{Z}$ already contains all variables that leak (in relation to the code sequence that the attack/evaluation relates to). In this section, we introduce the novel concept of model completeness, and an associated test to measure it efficiently.

3.1 Completeness

We make the important observation that the task of finding the *state* is the same as the task of deciding whether some variables contribute significantly to a model. This implies that we can find the state by testing nested models via an F-test to determine if the "bigger" model explains the real data better than the "smaller" model. This approach has been used before in the side channel literatures for deciding if low degree terms should be included in $\tilde{L}(\mathbf{Z})$ in [5,19]. We thus suggest to use this idea to define the notion of model completeness.

Definition 1. *We consider two nested models $\tilde{L}(\mathbf{Z}_f)$ and $\tilde{L}(\mathbf{Z}_r)$, and a corresponding F-test with*

$$H_0 : \tilde{L_f}(\mathbf{Z}_f) \text{ and } \tilde{L_r}(\mathbf{Z}_r) \text{ explain the observations equally well, and} \quad (1)$$

$$H_1 : \tilde{L_f}(\mathbf{Z}_f) \text{ explains the observations better than } \tilde{L_r}(\mathbf{Z}_r). \quad (2)$$

We call $\mathbf{Z}_r$ as **complete (with respect to $\mathbf{Z}_f$)** *if the null hypothesis of the F-test cannot be rejected.*

The notion of model completeness (of a reduced model with respect to some full model) means that the reduced model does not miss any (statistically) significantly contributing factor when compared with the full model.

Toy example. Suppose that $\tilde{L}(\hat{\mathbf{X}})$ contains the variables $\{x_0, x_1, x_2, x_3\}$ and the model $\tilde{L}(\mathbf{Z})$ contains the variables $\{x_0, x_1, x_2\}$. Following our discussion in Section 2.2, we estimate the model coefficients from realistic measurements:

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

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

We subject these models (which are nested as $\mathbf{Z}$ is included in $\hat{\mathbf{X}}$) to the F-test. If the F-test reports a *p-value* lower than the significance level α, we can conclude at least one of the regression terms that depends on x_3 is contributing significantly, and therefore we reject the null hypothesis. Thus, according to our definition, $\mathbf{Z}$ is not complete with respect to $\hat{\mathbf{X}}$.

Towards finding the state in practice In practice 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 $\mathbf{X}$).

As $\hat{\mathbf{X}}$ is a superset of $\mathbf{X}$, it is not hard to see that if $\mathbf{Z}$ is *complete* w.r.t. $\hat{\mathbf{X}}$, it should also be *complete* w.r.t. $\mathbf{X}$. Informally, this means if we can first explicitly define $\hat{\mathbf{X}}$, then test our selected $\mathbf{Z}$ against it, the F-test result will illustrate whether $\mathbf{Z}$ is *complete* or not (up to some statistical power), even if the true $\mathbf{X}$ remains unknown. The remaining challenges are firstly how to define a complete $\hat{\mathbf{X}}$ at the first place, and secondly how can we test $\mathbf{Z}$ against $\hat{\mathbf{X}}$ in an F-test.

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. 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[5]. Because of the large block size, this is a

[5] Note that this is not the same application scenario as leakage detection tests, which consider only the unshared secret. We did not propose the F-test as a leakage detection test here, therefore "which statistical moment the corresponding leakage detection is" is out of the scope.

starting point that is in fact unworkable from a statistical point of view (i.e. requires too many observations for model estimation). We explain a statistical trick in the next section, that makes this seemingly infeasible problem, practical.

3.2 Collapsed *F*-test for completeness

As we explained before, the *full* model is often too large in relevant practical scenarios: it consists of many large variables. In statistical terminology such models are called factorial designs, and the challenge is how to reduce the number of factors. Techniques such as aliasing or identifying confounding variables (i.e. variables that impact on multiple factors, or identifying factors that are known to be related) are well known in statistical model building. These techniques rely on *a priori* knowledge about the model (potentially because of prior work, or other sources of information), which we do not have in our setting.

However, we identified two ideas/observations that enable us to define a novel strategy to deal with factorial designs in our setting. 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. This implies that the test is agnostic to the actual value of the regression coefficient, thus we can set them to 0/1 and work with nominal models.

The second observation is that although our explanatory variable contains n independent bits, their leakages often share a similar form (e.g. the standard HW model can be written as the sum of every single bit), because the underlying circuit treats them as “a word”. Furthermore, for leakage modelling, we want to include any variable in our model if a single bit of that variable contributes to leakage. Based on these observations we show an elegant trick to bound our explanatory variables to a small space.

Bounding the explanatory variables We demonstrate our trick based on an example at first: assume we know the leakage can be conservatively determined by four n-bit values A, B, A' and B' (i.e. $\tilde{L}(\hat{\mathbf{X}}) = \vec{\beta}\mathbf{U}(\hat{\mathbf{X}})$, where $\hat{\mathbf{X}} = \{\hat{x}|\hat{x} = a||b||a'||b'\}$). In order to test if a chosen partial state $\mathbf{Z} \subset \hat{\mathbf{X}}$ is *complete* w.r.t $\hat{\mathbf{X}}$, following our previous discussion, we should test the following two regression models:

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

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

The nested F-test then applies to verify the following hypotheses:

H_0*:* $\tilde{L}_r(\mathbf{Z})$ *explains the observations as well as* $\tilde{L}_f(\hat{\mathbf{X}})$*.*

H_1*:* $\tilde{L}_f(\hat{\mathbf{X}})$ *explains the observations significantly better than* $\tilde{L}_r(\mathbf{Z})$*.*

Unfortunately, testing $\tilde{L}_f(\hat{\mathbf{X}})$ would require 2^{4n} explanatory variables in $\mathbf{U}(\hat{\mathbf{X}})$ as $\hat{x}$ is a $4n$-bit state. For $n \geq 32$, it requires an infeasible number of observations to properly estimate $\tilde{L}_f(\hat{\mathbf{X}})$. However, via setting $a_i = a_0$ (a_i stands for the i-th bit of a, a_0 drawn at random from $\{0,1\}$), we can bound the variable A to a much smaller space:

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

Since a_0 is a binary variable that satisfies $a_0^i = a_0, \quad i > 0$, $\hat{\mathbf{X}} = \{\hat{x}|\hat{x} = a||b||a'||b'\}$ now "collapses" to a $(3n{+}1)$-bit state set $\hat{\mathbf{X}}_c = \{\hat{x_c}|\hat{x_c} = a_0||b||a'||b'\}$. Applying this restriction to the other 3 variables, the collapsed leakage function $\tilde{L_c}(\hat{\mathbf{X}}_c) = \vec{\beta}\mathbf{U}(\hat{\mathbf{X}}_c)$, $\hat{\mathbf{X}}_c = \{\hat{x_c}|\hat{x_c} = a_0||b_0||a'_0||b'_0\}$ contains only 2^4 explanatory variables, which makes it 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 $\tilde{L_c}$ these terms are "collapsed" and "added" to the remaining terms, e.g. a_1a_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 $\tilde{L_c}$ suggests A appears in $\tilde{L}$, but certainly not in the same way as in $\tilde{L_c}$. We can expand this idea by allowing two (or more) bits of randomness: this enables us to differentiate between linear and non-linear models[6].

Slightly formalising this idea, we define a mapping called "collapse" *Coll* that converts a term $u_j(\hat{\mathbf{X}})$ in the original "un-collapsed" setup to $u_{Coll(j)}(\hat{\mathbf{X}}_c)$. Recall that $u_j(\hat{\mathbf{X}})$ is defined (Section 2.2) as:

$$u_j(\hat{\mathbf{X}}) = \prod \hat{x}_i^{j_i} \,,$$

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

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

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

$$u_j(\hat{\mathbf{X}}) = u_{j_{coll}}(\hat{\mathbf{X}}_c) \,, \hat{\mathbf{X}}_c = \{\hat{x}_c|\hat{x}_c = a_0||a'_0||b_0||b'_0\} \,.$$

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 $u_{j_{coll}}(\hat{\mathbf{X}}_c)$ cannot be ignored from $\tilde{L}_c$ (i.e. $\beta_{j_{coll}} \neq 0$), at least one of the corresponding $u_j(\hat{\mathbf{X}})$ 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

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

[6] We could take this further and include more "intra-variable" interactions, but we left this for future considerations. The "inter-variable" interactions remain in $\tilde{L_c}$ anyway.

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

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

Thus, if a certain collapsed term $u_{j_{coll}}(\hat{\mathbf{X}}_c)$ 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 $\tilde{L}_f = \vec{\beta}\mathbf{U}(\hat{\mathbf{X}})$, given a partial state $\mathbf{Z}$, we can test:

$$\tilde{L}_{cf}(\hat{\mathbf{X}}_c) = \vec{\beta}\mathbf{U}(\hat{\mathbf{X}}_c)\,,$$

$$\tilde{L}_{cr}(\mathbf{Z}_c) = \vec{\beta}\mathbf{U}(\mathbf{Z}_c)\,,$$

as the cardinalities of both $\mathbf{U}(\hat{\mathbf{X}}_c)$ and $\mathbf{U}(\mathbf{Z}_c)$ are at most 2^4. Then the nested F-test applies to:

H_0: *$\tilde{L}_{cr}(\mathbf{Z}_c)$ explains the observations as well as $\tilde{L}_{cf}(\hat{\mathbf{X}}_c)$,*

H_1: *$\tilde{L}_{cf}(\hat{\mathbf{X}}_c)$ explains the observations significantly better than $\tilde{L}_{cr}(\mathbf{Z}_c)$.*

If this F-test rejects H_0, we can also conclude that in the original un-collapsed setup, $\tilde{L}_f(\hat{\mathbf{X}})$ fits the observations significantly better than $\tilde{L}_r(\mathbf{Z})$. Thus, we learn that $\mathbf{Z}$ is not complete with respect to $\hat{\mathbf{X}}$ without testing it explicitly.

Toy example. Suppose we want to test $\tilde{L}_r = \vec{\beta}\mathbf{U}(\mathbf{Z}), \vec{\beta} \in \mathbb{R}^{2^{2n}}$ where $\mathbf{Z} = \{z | z = a||b\}$ against $\tilde{L}_f = \vec{\beta}\mathbf{U}(\hat{\mathbf{X}}), \vec{\beta} \in \mathbb{R}^{2^{4n}}$ where $\hat{\mathbf{X}} = \{\hat{x} | \hat{x} = a||b||a'||b'\}$. As mentioned before, for $n = 32$, direct testing is not an option. However, we can bound the inputs and test the collapsed models instead:

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

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

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 $\mathbf{Z}$ is also not complete (w.r.t. to $\hat{\mathbf{X}}$) in the original setup, without explicitly testing it. The price to pay is that unlike the original F-test, our collapsed test becomes a necessary condition instead of a sufficient condition. Nevertheless, any $\mathbf{Z}$ that fails our test presents a genuine concern, as it directly suggests the selected $\mathbf{Z}$ is unlikely to be complete.

In the remainder of this paper we will *always* work with collapsed models, thus the subscript c will be dropped from all models.

3.3 Statistical power of the nested F-test

It is imperative to understand the power of any statistical test. The power of a test is the probability that it detects an effect if the effect is present. To compute the power of a collapsed F-test, 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 [23]) 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 the 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

$$\beta = F_{nc}(Fstat_{th}, df_1, df_2, \lambda),$$

$$\lambda = f^2(df_1 + df_2 + 1),$$

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 3 tests in Section 5.1 have $df_1 = \{256 - 7, 256 - 19, 256 - 16\}$, $df_2 = q - 256$, $q = 20000$, per-test $\alpha = 10^{-3.7}$, which all come to $1 - \beta \approx 1$ for the *small effect size* $f^2 = 0.02$ in [23]. According to [24] this corresponds indeed to what they observed in similar experiments.

Summarising, assuming sufficient traces (20k in our calculation), the F-test on collapsed models has high power for relevant side channel scenarios according to [24].

4 Dissecting Attacks: Towards Worst-Case Adversaries

A recent paper by Bronchain and Standaert [25] demonstrated impressively how knowledge of an implementation (affine masking implementation from ANSSI [26] on an off-the-shelf, thus "grey box" processor) can help to derive extremely powerful profiled attack strategies. We wondered if our novel technique of the collapsed F-test could reveal further leakage that they might have missed (they did not explicitly consider if or not they had included all leaking variables).

To this end, we investigate the leakage of this implementation[7]. Our measurement setup consists of 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 S-box computation in the power traces, but we have no full state information. 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). We are analysing the masked table look-ups of the first 4 S-boxes in the first round. Altogether the 4 S-box computations take 17 μs, which amounts to around 204 clock cycles and 4250 samples on the power trace.

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, which corresponds to the non-shuffling analysis in [26]). Alternatively one can take our analysis as a "follow-up" after the shuffling permutation has been already recovered using the technique in [25].

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 (e.g. ShiftRow, MixColumn) can be done separately on each share. Meanwhile, the masked S-box is pre-computed according to the *multiplicative mask* rm and the S-box 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 S-box. In order to compute the S-box's output 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'
```

[7] The original *Compute_GTab* function contains a few instructions (e.g. *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 targeting this part.

```
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 S-box calculation, therefore not presented in this code.

Initial analysis. We first analyse the leakage of the first S-box look-up and use 1 bit to represent each x_i. All random masks (used within the captured 4 Sbox computation) 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 (i.e. nullifies the protection of rm)[8]. 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 the 2-bit version and use more traces to cover rm.

The complete model is therefore defined as

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

Targeting the leakage from the first S-box computation, we assume that all the computed values are leaking plus their transitions (i.e. following the spirit of [27]) . Thus we first define a coarse-grained model capturing all possible computations for the first S-box:

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

One can check that all the intermediate values that appear in the code snippet can be expressed by this restricted model $\tilde{L}_r(\mathbf{Z})$. Applying the F-test between $\tilde{L}_f(\hat{\mathbf{X}})$ and $\tilde{L}_r(\mathbf{Z})$ to the leakage measurements from computing the first S-box, our analysis finds that this x_0-only model fails in the collapsed F-test. As Figure 1 shows, the blue line clearly passes the threshold (i.e. $-log_{10}(pv) > th \Rightarrow pv < 10^{-pv}$), which implies that the realistic leakage contains much more than what $\tilde{L}_r(\mathbf{Z})$ can express.

One step further. We notice that the *ldrb* instruction probably loads not only the target byte, but also the other bytes within the same word (see also in Section 5.2 of [10]). Thus, our line (1) loads:

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

[8] 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.

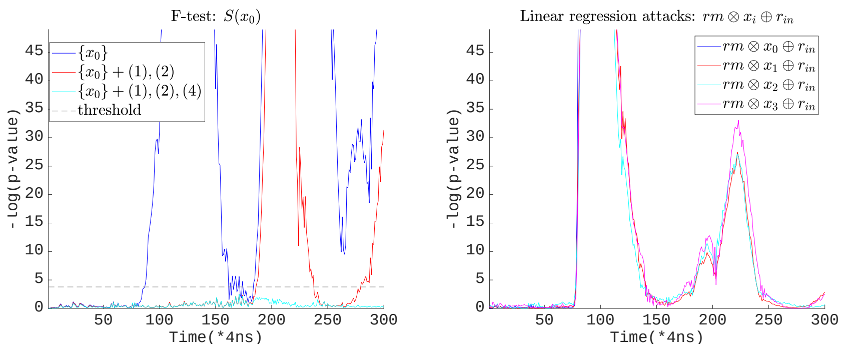


Fig. 1: Leakage analysis for the first S-box

Line (2) loads

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

As a consequence, the transition from Line (1) to Line (2) presents

$$\{rm \otimes x_0 \oplus r_{in}, rm \otimes x_1 \oplus r_{in}, rm \otimes x_2 \oplus r_{in}, rm \otimes x_3 \oplus r_{in}\}\,.$$

If we add both these values (plus their transitions) into $\tilde{L}(\mathbf{Z})$, the red lines show 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.

Let us further consider line (4): if it also loads a word, then the observed leakage depends not only on the target byte, but also on 3 adjacent bytes,

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

Unfortunately, the expression above does not follows high-to-low byte-order: as the memory address is masked by rin, the byte-order in this word 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 $za_1 = x_0||r_{in}||r_{out}||ra_1$: adding $\vec{\beta}\mathbf{U}(\mathbf{Z}_{a1})$ covers all possible transitions between ra_1 and the S-box 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 1).

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 [3] 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 loaded 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 1. To our knowledge, the most reasonable explanation

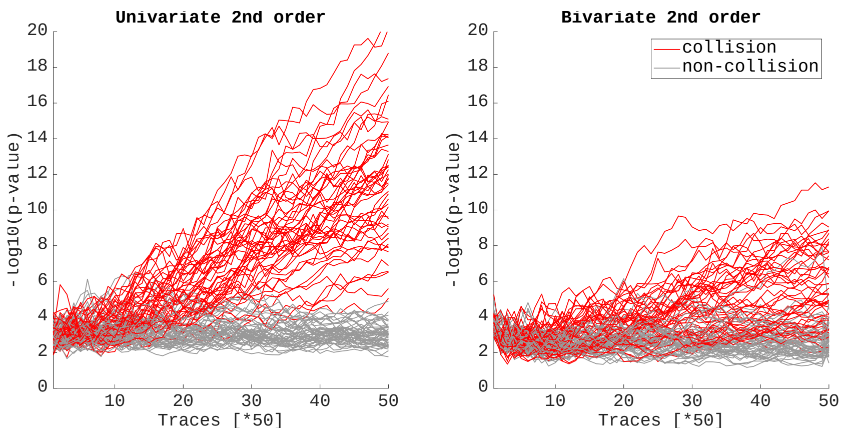


Fig. 2: Collision Oracle

is that such leakage is from the transition from line (1) and (2), where the entire 32-bit word is loaded in both cases.

A novel non-profiled attack option. The existence of leakage for multiple $rm \otimes x_i \oplus r_{in}$ provides a clue for a novel non-profiled attack strategy for this implementation: as all 4 bytes leak simultaneously around point 100, we can raise our measurements to their 2nd power (i.e. a univariate 2nd order attack), 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-versus-random t-test and plot the minimal p-value in Figure 2. 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 find 3 collisions[9] within the same word and determine their collision indices through adaptive testing, the key space for each word can be reduced to 2^8. Similar procedure can be applied to the other 3 32-bit words. Consequently, to recover the master key, we only need to enumerate the remaining 2^{32} key guess space. We leave the question of *what is the most efficient attack strategy* open, as it is out of the scope of this paper.

[9] 4 bytes in one word, therefore there are at most $\binom{4}{2} = 6$ collisions.

Summary. Our analysis demonstrates that there are considerably better non-profiled attacks available than anticipated in [26], and it raises the question of developing further even more advanced profiled attacks based on the knowledge of the state.

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 2, 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 (versus 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 words, 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 2 is not affected by these subtle implementation details.
– *Link to t-test.* The exploited leakage can be found through 2nd order fixed-versus-random (or fixed-versus-fixed) t-test, suppose the selected fixed constant contains a "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 [28].

5 Application to Leakage Simulators

In recent years, various leakage simulators have been proposed in order to enable early-stage leakage awareness in the context of software implementations on off-the-shelf processors [29]. Using a leakage simulator, developers can identify and patch potential leaks at an early stage, even if they have no physical 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 models, in line with the statistical analysis that we provided for our novel test. The measurement setup is the same as in Section 4, except for the working frequency which we reduced to 1 MHz: a lower frequency helps to provide a clearer cycle-to-cycle view, which is essential for model building. Any model is ultimately data dependent: a proportional leakage model such as [5] represents the device leakage as 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, i.e. models that only have 0/non-0 coefficients, at least initially[10].

5.1 Modelling leakage of individual instructions

As pointed out in [29], one of the remaining challenge for grey-box simulators is "(they) target relatively simple architectures". In fact, many tools only target algorithmic variables that may correspond to multiple intermediate states. Even if the simulator takes binary code as its input (e.g. ELMO [5]), the undocumented micro-architectural effects can still cause all sorts of issues [10]. Our novel statistical tool can help to identify missed leakage: if the leakage model utilised by a leakage simulator fails our novel test, it suggests that some contributing factor is missing.

The most sophisticated grey-box simulators that exist at the moment have all been built for the ARM Cortex M family, including ELMO [5] and its extension ELMO* [18], as well as MAPS [30]. ELMO derives its models from profiling traces from an actual Cortex M0 processor. The ELMO*/ROSITA [18] tools extend the ELMO model to cover various memory related leaks. The MAPS simulator does not have a profiled power model, since it uses value/transition-based models that are derived directly from the hardware description of an Cortex M3 core (provided by ARM under an educational licence). Technically speaking, there is no guarantee that this example core is identical to the IP cores that ARM sold to the manufacturers.

The code snippet that we will use in the following does not utilise any memory instruction. Thus, any observed difference is purely down to how the simulators model the leakage within the data processing unit. Considering that ELMO* extends ELMO mainly in the memory sub-system, in most cases below, they should produce the same result as ELMO (therefore omitted in the comparison). The ELMO repository offeres a range of leakage models all sitting on top of a Thumb instruction set emulator: there is a power model for an M0 by STM, an EM model derived from an M4, and an additional power model derived from an M3 [17]. This last model/version corresponds to the processor in our setup (an NXP LPC1313). In the following, our comparison is always on this M3 version of ELMO. Our comparision also include the simulator MAPS, which is also designed for a Cortex M3 [30] .

Simplified instruction-wise model. A common simplification in the three grey-box simulators ELMO, ELMO* and MAPS is that they all focus on the leakage

[10] To build a proportional model on top of the recovered nominal model is possible. One would need to estimate coefficients based on further measurement data (adjusted to the frequency that the target would run on in the wild).

within the ALU. This is a sensible choice: even if the processor has a multi-stage pipeline, we do not necessarily care about the leakage from fetching or decoding the instructions (as it is often not data-dependent[11]).

Sticking with the same notation as before, we describe the full model for instructions with two inputs (current A and B, previous A' and B') as $\tilde{L}_f = \vec{\beta}\mathbf{U}(\hat{\mathbf{X}})$, where $\hat{\mathbf{X}} = \{AA'BB'\} = \{\forall \hat{x} | \hat{x} = a||a'||b||b'\}$. The output value of an instruction, denoted as C (previous C'), is completely determined by A and B, therefore there is no need to add C (or C') into $\tilde{L}_f$. However, if restrictions on the inputs are added (e.g. the leakage of A is linear, denoted as $\mathbf{U}_l(A) = \{u_j(A)|HW(j) < 2\} = \{1, a_0, ..., a_{n-1}\}$), we might need to add the instruction output C to our model. In our experiments, we also consider the following leakage models that correspond to the most commonly used models in the existing literature:

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

$\tilde{L}_{le}$:

$$\tilde{L}_{le} = \vec{\beta}(\mathbf{U}_l(A), \mathbf{U}_l(B), \mathbf{U}_l(C), \mathbf{U}_l(A'), \mathbf{U}_l(B'), \mathbf{U}_l(C'), \\ \mathbf{U}_l(A \oplus A'), \mathbf{U}_l(B \oplus B), \mathbf{U}_l(C \oplus C'))$$

this model further includes Hamming distance information, which can be regarded as an extension for both the Hamming weight and the Hamming distance model (used in the MAPS simulator [30]); it therefore also generalises the ELMO model [5] which only fits a single dummy for the Hamming distance leakage.

$\tilde{L}_{TA} = \vec{\beta}\mathbf{U}(\{AB\})$: this model represents template attacks [1], 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" [31].

Challenge code snippet. Before any further analysis, we craft a code snippet that can trigger core leakage in the execute cycle, 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
```

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

```
nop
eors  r5,r7            //r5=a,  r7=b **Target**
nop
nop
```

eors r5,r7 represents the cycle we are targeting: 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 micro-processor, 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[12]).

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 3 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. As a consequence, existing simulators that built on $\tilde{L}_{le}$ (e.g. ELMO/ELMO* and MAPS) are expected to miss leakage, due to the limited explanatory power of the respective leakage models. Various effects could be contributing here (including the bit-interaction [9]).

5.2 Modelling leakage of complex instruction sequences

Our novel methodology clearly shows that conventional models are not sufficient even when just considering a single instruction, dispatched in a way that respects the simulator assumptions (no fetch or decode leakage, no memory leakage). Two questions follow immediately: can we develop better models and what do existing simulators miss when it comes to complex code sequences?

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

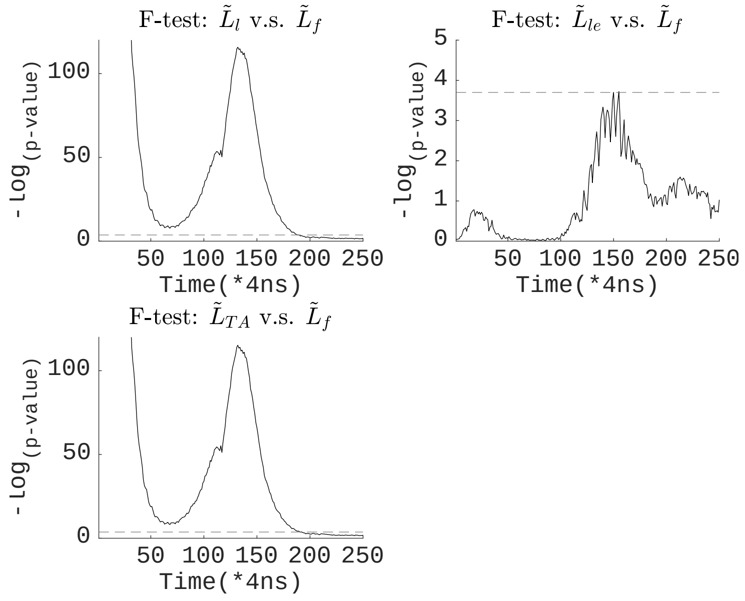


Fig. 3: Comparing various models against $\tilde{L}_f$

We answer these questions based on an Assembly sequence that implements a 2-share ISW (bitwise) multiplication gadget. Such a multiplication gadget is an integral part of a typical masking scheme.

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 $a_{(1)}$. To define the full model we assume that the inputs to the gadget, a and b, and the randomness r, are collapsed to 2-bit variables. The *full* model is then given as

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

Working with this larger code sequence implies that we must expect leakage from the pipeline registers and the fetch and decode stages (i.e. leakage that we prohibited in our much simpler analysis before).

Collapsed F-test. We recall that both $\tilde{L}_l$ and $\tilde{L}_{TA}$ were already clearly rejected by the F-test for just a single instruction (as shown in Figure 3), thus we only challenge the completeness of $\tilde{L}_{le}$ (with respect to $\tilde{L}_f$) in the context of the more complex code snippet. $\tilde{L}_{le}$ is of particular interest because ELMO/ELMO* [5,18] and MAPS [30] use a subset of $\tilde{L}_{le}$. The result is shown in the left picture in Figure 4. We can see, unlike in Figure 3, the linear extended model is clearly rejected by our test for multiple instructions.

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$//c_2 = r \oplus a_{(2)}b_{(2)}$	✓	✓	✓	✓

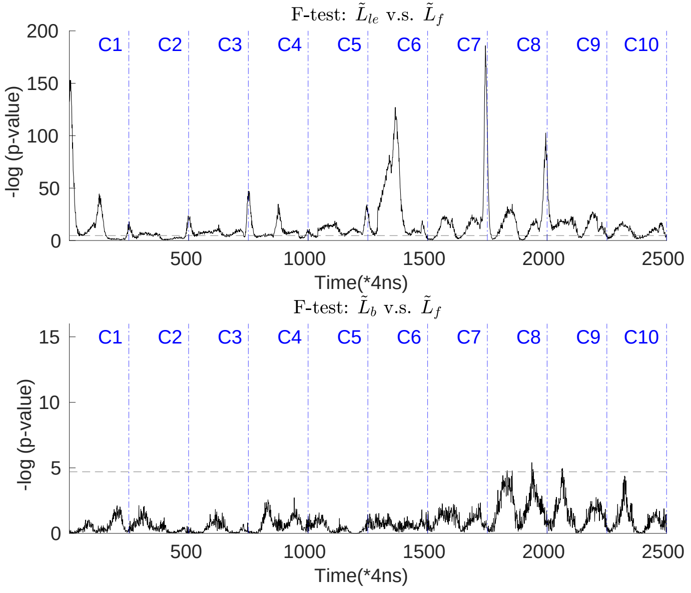


Fig. 4: Model comparison based on a 2-share ISW multiplication in software

Building a better model $\tilde{L}_b$. 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. We call the final model out of this *ad-hoc* procedure $\tilde{L}_b$. We derived 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 in $\tilde{L}_b$. Developing an architectural reasoning for this model is beyond the scope of this paper. However, Figure 4 shows that $\tilde{L}_b$ only marginally fails our test, and thus is considerably better than the linear extended model that simulators like ELMO/ELMO* and MAPS use.

Challenging $\tilde{L}_b$. Whilst we now have a model that explains the device leakage of our Cortex M3 for a relatively complex gadget, 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") [30]. 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*[13].

ELMO captures the operands and transitions on the ALU data-bus [5]: 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.

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, 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 is clear that building simulators based on insufficient models (in particular models that wrongly exclude parts of the state) lead to incorrect detection results.

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

6 Ethical considerations and Conclusions

This paper puts the *state* that is captured by a leakage model at the centre stage. Knowledge of this state matters: in the case where we want to argue that we have the "best possible attack" (perhaps in the context of an evaluation where we try to argue that the evaluation result corresponds to the worst-case adversary), and in the case where we want to build an accurate leakage simulator (perhaps to evaluate software countermeasures such as masking).

In our paper 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* initially seems like a computationally infeasible task in the case of modern processors: even for a 2-operand instruction, if we take previous values into account, there are 2^{4n} possible values. For $n = 8$ (i.e. in a small microcontroller), 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. Our novel technique is robust and effective, and works in a grey box setting, as we illustrate based on a range of concrete experiments.

The leakage models that result from our test are qualitatively different to leakage models that are currently in use: they are nominal models, which means that the coefficients in these models only describe if a variable (or an interaction term of multiple variables) contributes to the device leakage, or not. The models thus are no longer proportional to the real device leakage, consequently they are of less use for proportional attack techniques (e.g. correlation or template attacks). Our approach trades off less complete proportional models for nominal models that are statistically closer to a complete model—one could consider combining both modelling techniques to reintroduce some proportionality.

However, we argue that nominal models represent an important option also from a research ethics perspective. Research into modelling techniques leads to a dual-use question: techniques that are explicitly developed to work on off-the-shelf processors, and potentially released as part of open source projects such as ELMO and ROSITA may play in the hands of adversaries. Indeed the ELMO paper already shows an example of using ELMO's leakage model in a correlation based attack on a physical M0: it considerably improves the attack performance and does clearly have potential for dual use. Nominal models, on the other hand, can help to develop more sophisticated attacks (in the sense that more leakage can be included in an attack), or confirm that the best attack has already been found (again in the sense that all leakage was included). However, they do not lead to an immediate improvement for simple model based attacks (such as correlation attacks). Finally research ethics also ties in with practice: when discussing the possibility of developing models for processors with commercial vendors, an immediate concern is that of "helping adversaries" by supplying high quality leakage models that can be readily used in attacks. Therefore, the option to supply nominal models that can be used for leakage detection (or potentially for automated leakage proofs) is seen as a potential way forward for the practical deployment of simulators.

Acknowledgments

We would like to thank all anonymous reviewers for their constructive comments. The authors were funded in part by the ERC via the grant SEAL (Project Reference 725042).

References

1. Chari, S., Rao, J.R., Rohatgi, P.: Template Attacks. In: Cryptographic Hardware and Embedded Systems - CHES 2002, 4th International Workshop, Redwood Shores, CA, USA, August 13-15, 2002, Revised Papers. (2002) 13–28
2. Schindler, W., Lemke, K., Paar, C.: A Stochastic Model for Differential Side Channel Cryptanalysis. In Rao, J.R., Sunar, B., eds.: 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., Springer (2005) 30–46
3. Doget, J., Prouff, E., Rivain, M., Standaert, F.: Univariate side channel attacks and leakage modeling. J. Cryptogr. Eng. **1**(2) (2011) 123–144
4. Whitnall, C., Oswald, E.: Profiling DPA: efficacy and efficiency trade-offs. In Bertoni, G., Coron, J., eds.: 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., Springer (2013) 37–54
5. McCann, D., Oswald, E., Whitnall, C.: Towards Practical Tools for Side Channel Aware Software Engineering: 'Grey Box' Modelling for Instruction Leakages. In: 26th USENIX Security Symposium (USENIX Security 17), Vancouver, BC, USENIX Association (2017) 199–216
6. Papagiannopoulos, K., Veshchikov, N.: Mind the Gap: Towards Secure 1st-Order Masking in Software. In Guilley, S., ed.: 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., Springer 282–297
7. Gigerl, B., Hadzic, V., Primas, R., Mangard, S., Bloem, R.: Coco: Co-Design and Co-Verification of Masked Software Implementations on CPUs. IACR Cryptol. ePrint Arch. **2020** (2020) 1294
8. De Meyer, L., De Mulder, E., Tunstall, M.: On the Effect of the (Micro)Architecture on the Development of Side-Channel Resistant Software. IACR Cryptol. ePrint Arch. **2020** (2020) 1297
9. Gao, S., Marshall, B., Page, D., Oswald, E.: Share-slicing: Friend or Foe? IACR Transactions on Cryptographic Hardware and Embedded Systems **2020**(1) (Nov. 2019) 152–174
10. Marshall, B., Page, D., Webb, J.: MIRACLE: MIcRo-ArChitectural Leakage Evaluation. IACR Cryptol. ePrint Arch. (2021) `https://eprint.iacr.org/2021/261`.

11. den Hartog, J., Verschuren, J., de Vink, E.P., de Vos, J., Wiersma, W.: PINPAS: A tool for power analysis of smartcards. In: International Conference on Information Security (SEC2003). Volume 250 of IFIP Conference Proceedings., Kluwer (2003) 453–457
12. eshard: esDynamic. `https://www.eshard.com/product/esdynamic/`, accessed June 2018.
13. Secure-IC: Virtualyzr. `http://www.secure-ic.com/solutions/virtualyzr/`, accessed June 2018.
14. Thuillet, C., Andouard, P., Ly, O.: A smart card power analysis simulator. In: Proceedings of the 12th IEEE International Conference on Computational Science and Engineering, CSE 2009, IEEE Computer Society (2009) 847–852
15. Debande, N., Berthier, M., Bocktaels, Y., Le, T.H.: Profiled Model Based Power Simulator for Side Channel Evaluation. Cryptology ePrint Archive, Report 2012/703 (2012)
16. Gagnerot, G.: Étude des attaques et des contre-mesures assoccées sur composants embarqués. PhD thesis, Université de Limoges (2013)
17. McCann, D.: ELMO. `https://github.com/bristol-sca/ELMO` (2017)
18. Shelton, M.A., Samwel, N., Batina, L., Regazzoni, F., Wagner, M., Yarom, Y.: Rosita: Towards Automatic Elimination of Power-Analysis Leakage in Ciphers. NDSS 2022 (2022)
19. Whitnall, C., Oswald, E., Standaert, F.: 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. (2014) 183–205
20. Durvaux, F., Standaert, F., Veyrat-Charvillon, N.: How to Certify the Leakage of a Chip? In Nguyen, P.Q., Oswald, E., eds.: 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., Springer (2014) 459–476
21. Crama, Y., Hammer, P.L., eds.: Boolean Models and Methods in Mathematics, Computer Science, and Engineering. Cambridge University Press (2010)
22. Shmueli, G.: To Explain or to Predict? Statist. Sci. **25**(3) (08 2010) 289–310
23. Cohen, J.: CHAPTER 9 - F Tests of Variance Proportions in Multiple Regression/Correlation Analysis. In Cohen, J., ed.: Statistical Power Analysis for the Behavioral Sciences. Academic Press (1977) 407 – 453
24. Whitnall, C., Oswald, E.: 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. (2019) 256–284
25. Bronchain, O., Standaert, F.: Side-Channel Countermeasures' Dissection and the Limits of Closed Source Security Evaluations. IACR Trans. Cryptogr. Hardw. Embed. Syst. **2020**(2) (2020) 1–25
26. Benadjila, R., Khati, L., Prouff, E., Thillard, A.: Hardened Library for AES-128 encryption/decryption on ARM Cortex M4 Achitecture. `https://github.com/ANSSI-FR/SecAESSTM32`
27. Balasch, J., Gierlichs, B., Grosso, V., Reparaz, O., Standaert, F.: On the Cost of Lazy Engineering for Masked Software Implementations. In: Smart Card Research and Advanced Applications - 13th International Conference, CARDIS 2014, Paris, France, November 5-7, 2014. Revised Selected Papers. (2014) 64–81

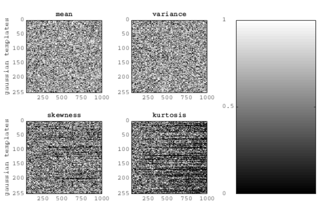
28. Whitnall, C., Oswald, E.: A Cautionary Note Regarding the Usage of Leakage Detection Tests in Security Evaluation. Cryptology ePrint Archive, Report 2019/703 (2019)
29. Buhan, I., Batina, L., Yarom, Y., Schaumont, P.: SoK: Design Tools for Side-Channel-Aware Implementions (2021)
30. Le Corre, Y., Großschädl, J., Dinu, D.: Micro-architectural Power Simulator for Leakage Assessment of Cryptographic Software on ARM Cortex-M3 Processors. In Fan, J., Gierlichs, B., eds.: 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., Springer (2018) 82–98
31. Micali, S., Reyzin, L.: Physically Observable Cryptography (Extended Abstract). In Naor, M., ed.: 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., Springer (2004) 278–296
32. Durvaux, F., Standaert, F., Pozo, S.M.D.: Towards easy leakage certification: extended version. J. Cryptogr. Eng. **7**(2) (2017) 129–147
33. Bronchain, O., Hendrickx, J.M., Massart, C., Olshevsky, A., Standaert, F.: Leakage Certification Revisited: Bounding Model Errors in Side-Channel Security Evaluations. In Boldyreva, A., Micciancio, D., eds.: 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., Springer (2019) 713–737
34. Standaert, F., Malkin, T., Yung, M.: A Unified Framework for the Analysis of Side-Channel Key Recovery Attacks. In Joux, A., ed.: 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., Springer (2009) 443–461
35. Lerman, L., Veshchikov, N., Markowitch, O., Standaert, F.: Start Simple and then Refine: Bias-Variance Decomposition as a Diagnosis Tool for Leakage Profiling. IEEE Trans. Computers **67**(2) (2018) 268–283

A PI, HI & Assumption error

Leakage certification approaches such as described in [20,32,33] (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) [33].

The common choice for $\mathbf{Z}$ is often a variable that relates to a single S-box [20,32,33]: because the MI calculation runs through all possible values of $\mathbf{Z}$, it corresponds to a template attack. This extremely popular choice is potentially


(a) HD leakage without any noise

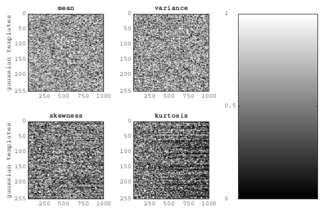

(b) HD leakage with noise variance 0.1

Fig. 5: Moment based detection of "assumption error"

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.1 Estimating "assumption errors"

In [20] Durvaux et al. proposed a technique to test for (the so-called) assumption errors in the leakage model [20]. 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 [32], we can validate the power of detecting the above "assumption error": Figure 5a 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 5b), 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

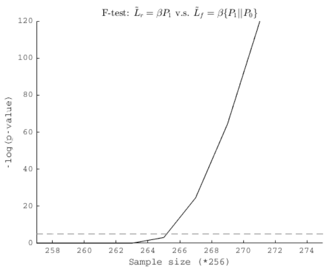


Fig. 6: F-test with noise variance 0.1

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 $p_1 \oplus k_1$, 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 [20] 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. 6. 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 [33].

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 [33]: as we can see in Figure 7, 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.

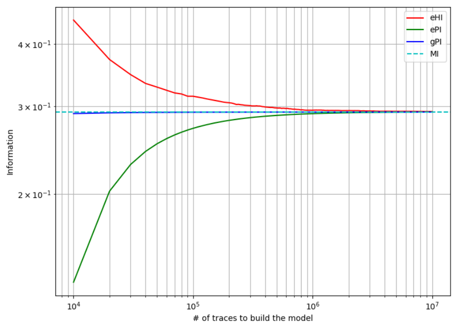


Fig. 7: PI and HI estimation for the leakage function

A.3 Bias-Variance Decomposition

Lerman, Veshchikov and Markowitch et al. also proposed a diagnosis tool based on the bias-variance decomposition [35]. 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*.