Leakage, leakage, leakage . . . models: How to determine what contributes to leakage

Si Gao and Elisabeth Oswald

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

Abstract. Leakage attacks and simulators strongly rely on crucial knowledge about the *state* that is being leaked on. Despite 20 years of effort, in terms of how to find the relevant *state*, we did not actually go very far: to date, we still constantly assume users already know the *state*, or users can reliably find it based on a few attack trials and their own experience. This is far from the truth that is encountered in practice: whilst software platforms give an illusion of a sequential update to variables, the reality in the underlying hardware is that previous values remain part of the state and many things happen in parallel. 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". This test can even cope in a grey box setting where the state contains multiple 32-bit variables. We illustrate how our novel test can help to guide attacks and leakage simulators, reveal new form of leakage that is previously unknown and deepen our understanding of the realistic leakage as well as the underlying architecture.

Keywords: Leakage model, Intermediate state, Security evaluation

1 Introduction

Since Kocher's seminal work [1], 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 modelled by an adversary). The latter component, the leakage model, is crucial for attacks (a better model leads to attacks requiring fewer leakage observations), but it is also of fundamental significance in the context of security evaluations.

Looking at the leakage model in the context of attacks, it is perhaps obvious that attacks using models that are closer to the true device leakage should be better (i.e. require fewer traces for successful attacks). Thus, a lot of research has gone into deriving good estimates for the functional representation of models from real device data [2–6]. Such leakage model takes the relevant intermediate state as input, then map the *state* into leakage value(-s).

Perhaps surprisingly, not much work has been done that establishes what the relevant *state* actually is *on modern devices*. Modern cryptographic devices range from simple 8-bit processors with little parallelism, over mid-range 32-bit processors with multiple components operating in parallel, to complex application specific crypto processors again featuring significant amounts of parallelism. If we consider some crypto implementation on an off-the-shelf processor, the most basic question would be: "what components are actually contributing to the computation at any moment in time"?

In a realistic setting (such as the ones mentioned above), finding the *state* is far from easy. The vast majority of low to mid-range processors (e.g. ARM-Cortex M family, AVR, etc) are 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 [7–11]. 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 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 statistically tools in Section 2. Section 3 introduces the concept of *completeness* and proposed 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, as an incomplete leakage model could jeopardise the following detection accuracy. We summarise our discussion and emphasise a few important lessons learned in Section 6.

2 Preliminaries

2.1 Leakage modelling: state of the art

We use some simple notation throughout this paper. We call the set $\boldsymbol{X}$ the entire device state (i.e. it is a set of key and input dependent variables) that contributing to the leakage function L at some point in time during the computation. The variable $Y = \{y_i\} \in \boldsymbol{Y}$ is a leakage observation that is available to an adversary. We also follow the usual convention that traces are noisy, whereby the leakage contribution $L(\boldsymbol{X})$ and the (Gaussian) noise $N(0, \sigma^2)$ are independent:

$$y_i = L(\boldsymbol{X}) + N(0, \sigma^2).$$

For the sake of readability we will drop the time index in our notation: but it should be understood throughout this paper that the set $\boldsymbol{X}$ is different for different points (i.e. y_i) in time during the computation of an algorithm[1].

Considering the detailed hardware specification is not always available (eg. ARM cores, commercial IP cores etc.), the relevant device state $\boldsymbol{X}$ (for a certain time index) is typically unknown. We can of course, build a overly conservative model using all possible *state* $\hat{\boldsymbol{X}}$ where $\boldsymbol{X} \subset \hat{\boldsymbol{X}}$ (eg. all appeared intermediate states within the encryption). However, such model is neither valid for attacks (depends on too many key bits), nor ideal for profiling (requires infeasible amount of traces).

The *de facto* practice is, building leakage models (i.e. the purple part in Figure 1) is usually divided into two steps. The first step is identifying a concise state $\boldsymbol{Z}$ (the purple trapezoid in Figure 1): unlike $\hat{\boldsymbol{X}}$, $\boldsymbol{Z}$ is relatively small so profiling/model building can be finished within reasonable time and effort. For instance, the typical assumption is that the intermediate state depends completely on the output of the S-box computation (denoted by the 8-bit variable S_{out}), which leads to the cardinality of the state being rather small, i.e. $\boldsymbol{Z} = \{S_{out}\}$ and hence $\#\{\boldsymbol{Z}\} = 2^8$ (there are 2^8 different values that S_{out} can take, each of which could in principle leads to a unique leakage value).

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

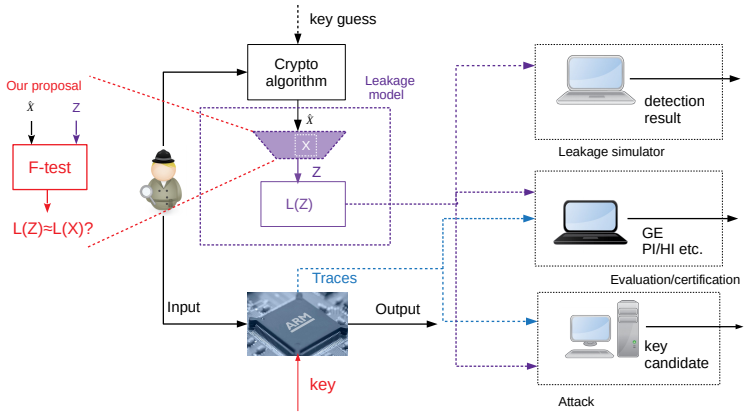


Fig. 1: A typical SCA workflow

The next step builds a specific model using only $\boldsymbol{Z}$ (i.e. the purple rectangle in Figure 1): various techniques have been proposed, including trivial templating [2], regression-based modelling [3, 4], step-wise regression [12] etc. Previous works [12, 3, 13] have also proposed various metric 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 move on to our point of interest: *what should we do about the first step?*

2.2 Approaches to build $\tilde{L}(\boldsymbol{Z})$

Already in the early days of side channel research, the concept of profiling (aka leakage modelling, aka templating) was introduced by Chari et al. [2]. 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 could simply estimate the parameters of the distribution.

An alternative to the direct parameter estimation is the use of regression techniques to derive an equivalent model. A paper by Schindler et al. [3] proposes the use of regression to derive a model for a considered intermediate value. They select $\boldsymbol{Z}$ to be a single variable such as S_{out} and then fit coefficients to a bit-linear model (i.e. to each bit of S_{out}). This linear model is a slightly more flexible alternative for the standard Hamming weight model, as each bit now can have an individual (unequal) contribution to the leakage. But a regression model for S_{out} can also be based on 256 coefficients and this would be equivalent to building 256 templates (in the spirit of Chari et al.) for the state S_{out}.

More generally, the basis of regression is that we can express any real valued function of Z as the polynomial $\tilde{L} = \sum_j \beta_j u_j(\boldsymbol{Z})$ [14], or short $\tilde{L} = \boldsymbol{\beta}(\boldsymbol{Z})$. 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 Z and j_i denote the i-th bit of j (with n the number of bits needed to represent Z in binary). Regression then estimates the coefficients β_j. The explanatory variables u_j simply represent the different values that 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* modelIn many previous attacks, the leakage model is restricted to just contain the linear terms. We denote this particular *linear model* using a subscript l.

2.3 Quality of $\tilde{L}(Z)$

Coefficient of determination For any model $\tilde{L}(\boldsymbol{Z})$ that is estimated from the side channel measurements $\boldsymbol{Y}$, 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 $\boldsymbol{Y}$. If $\tilde{L}$ is derived from linear regression on $\boldsymbol{Y}$, 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 $u_j(\boldsymbol{Z})$ 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 [13] 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.

Perceived information Leakage certification aims at providing guarantees about the quality of an evaluation, based on estimating the amount of information leaked by a target device [13, 15, 16]. Consequently, ensuring that the used model is good becomes a critical pre-condition when using such an approach [16]. Various techniques have been proposed, including comparing the "assumption error" against the "estimation error" [13], moment-based comparison of the measurements and the model [15], and bounding the mutual information with *Hypothetical information (HI)* [16].

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 to the *full* model. More specifically, assuming a *full* model $\tilde{L}_f(\boldsymbol{Z}_f)$ and a *restricted* model $\tilde{L}_r(\boldsymbol{Z}_r)$, where $\boldsymbol{Z}_r$ is constructed by removing $z_f - z_r$ explanatory variables (set regression coefficients to 0) from $\boldsymbol{Z}_f$, 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 suggests that at least one of the removed variables is potentially useful. This approach was used in [6] to derive relatively fine grained models on selected intermediate states.

2.4 Approaches to find $\boldsymbol{Z}$

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

Leakage certification techniques, eg. "assumption error" v.s. "estimation error" [13] are also designed under the assumption that $\boldsymbol{Z}$ is given. Such technique can sometimes be utilised to test the scenario where the selected $\boldsymbol{Z}$ is not sufficient: however, we remind readers this does not fit with authors' original intension and its statistical power is far from ideal (see Appendix A).

Unfortunately, as we can see in Figure 1, having one insufficient $\boldsymbol{Z}$ affects all following steps. Take an AES encryption as an instance, if our target measurement depends on both the first Sbox output ($S_{out,1}$) and the second Sbox output ($S_{out,2}$), it is fairly possible that users did not notice $S_{out,1}$ also appears and simply define $\boldsymbol{Z} = \{S_{out,2}\}$. As a consequence, all following results will be jeopardised—attacks being less efficient, evaluations/certifications being overly optimistic or leakage simulators miss a certain flaw.

3 Model quality: is my leakage model complete?

In this section, we focus on the other side of model quality: more specifically, we propose a necessary (but not sufficient) test to check whether the selected $\boldsymbol{Z}$ is complete or not.

3.1 Relatively-completeness

Although in our community the nested *F-test* is mainly used for determining the degree of $\tilde{L}(\boldsymbol{Z})$ [6, 12], as an ANalysis Of VAriance (ANOVA) technique, its primary usage is to determine whether a certain regression term has statistically significant contribution to the regression model. In a side channel context, this means we can not only verify if there are second order interactions in $\tilde{L}(\boldsymbol{Z})$ (aka the usage in ELMO [6]), but also in a coarser grain, verify whether a certain element/bit is contributing or not.

Definition 1. *For any selected $\boldsymbol{Z}$, we denote it as complete (with respect to $\boldsymbol{X}$) if the F-test claims $\tilde{L}(\boldsymbol{Z})$ did not miss any significant contributing factor compared with $\tilde{L}(\boldsymbol{X})$. With infinite profiling traces (i.e. infinite statistical power), this usually suggests $\boldsymbol{X} \subseteq \boldsymbol{Z}$.*

The problem with this definition is usually we do not know the exact $\boldsymbol{X}$, therefore cannot construct the model $\tilde{L}(\boldsymbol{X})$. However, as we will discuss in details later, it is possible to explicitly define an overly-conservative set $\hat{\boldsymbol{X}}$ that ensures $\boldsymbol{X} \subseteq \hat{\boldsymbol{X}}$.

Corollary 1. *For any selected $\boldsymbol{Z}$, we denote it as complete (with respect to $\boldsymbol{X}$) if the F-test claims $\tilde{L}(\boldsymbol{Z})$ did not miss any significant contributing factor compared with $\tilde{L}(\hat{\boldsymbol{X}})$.*

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

Toy example. Say we have an unknown relevant set $\boldsymbol{X}$. Although $\boldsymbol{X}$ is unknown, we can define an overly conservative set $\hat{\boldsymbol{X}} = \{x_0, x_1, x_2, x_3\}$ that satisfies $\boldsymbol{X} \subseteq \hat{\boldsymbol{X}}$. At this point, we wish to test if we can discard x_3 and see if $\boldsymbol{Z} = \{x_0, x_1, x_2\}$ is a good model. Following our discussion in Section 2.3, we can estimate from realistic measurements:

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

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

If the *F-test* reports a *p-value* lower than the significance level, we can conclude at least one of the regression terms involves x_3 is contributing. In other words, $x_3 \in \boldsymbol{X}$ and $\boldsymbol{X} \not\subseteq \boldsymbol{Z}$, which suggests the model built with $\boldsymbol{Z}$ is missing some leakage.

Ensuring $\hat{\boldsymbol{X}}$ is complete Assuming a block cipher encryption (ECB mode) always starts from a "clean context"[2], the encryption process itself becomes an oracle that embedded with a secret constant key. Any intermediate state, no matter what it represents, can always be fully determined by the full plaintext (unprotected case, no masking). Considering our primary assumption defines L as a deterministic function, defining $\hat{\boldsymbol{X}}$ as all possible inputs (i.e. plaintext) enables a conservative model that can express any leakage during the encryption. Although such $\hat{\boldsymbol{X}}$ can trivially define a complete model, the problem is for most block ciphers (eg. AES-128), this full model requires an infeasible number of traces to built (eg. more than 2^{128} for AES-128). This is the core challenge we are aiming at in the next section.

Note. We remind readers that the key point here is including "all possible inputs": if the target is a masked AES-128, building leakage model should use $\hat{\boldsymbol{X}}$ that include all random masks that added into the encryption. We propose here a test that verifies the quality of the leakage model in a profiling setup: it cannot be used as a leakage detection test where the fixed/random bit is about the secret, rather than any existing *state*. The *F-test*, as an ANOVA test, is not restricted to any specific statistical moment: however, if applied in a masked implementation, the corresponding leakage detection is specific to 1st order moment (note that this version is no longer an ANOVA test) [18].

3.2 Collapsed F-test for completeness

Following our previous discussion, the *full* model is often too large that there is little chance we can use the *F-test* "as is". There are two key observations that help us to derive a novel work-around, that deals with arbitrary size variables.

The first observation is that the *F-test*, although typically used to deal with proportional models, actually tests for the inclusion/exclusion of explanatory variables. Thus in principle, the *F-test* does not care about the specific regression

[2] This can be archived by clearing the contexts using techniques like [17], as long as the procedure is not called too frequently.

coefficients (for as long as they are non-zero), and thus it lends itself to also work with nominal models. The latter do not require estimation of coefficients, but just a decision if or not a coefficient is included in the model (if so, it is set to 1; otherwise, it is set to 0).

The second and crucial observation is that for the use case of finding models with a high explanatory power, we can bound the explanatory variables to a smaller 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 a 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 to 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 clearly feasible.

Of course, such a restriction is not for free: originally, there could be many interaction terms between the explanatory variables. In the "collapsed" model L_c where we have bounded the inputs, 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 a "leakage vane" 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.

Formalising this idea, we define a mapping called "collapse" $Coll$ on the $u_j(\boldsymbol{Z})$, where $\boldsymbol{Z} = \{AA'BB'\}$. Recalled that $u_j(\boldsymbol{Z})$ can be written as

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

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

$$u_j(\boldsymbol{Z}) = u_{j_{coll}}(\boldsymbol{Z}_c), \boldsymbol{Z}_c = \{z_c | z_c = a_0 || a_0' || b_0 || b_0'\}$$

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

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

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

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

Thus, if a certain collapsed term $u_{j_{coll}}(\boldsymbol{Z}_c)$ 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 $\tilde{L} = \boldsymbol{\beta}(\hat{\boldsymbol{X}})$, we can now test the restricted (and collapsed) models $\tilde{L}_{cr} = \boldsymbol{\beta}(\boldsymbol{Z}_c)$ against the collapsed *full* model $\tilde{L}_c = \boldsymbol{\beta}(\hat{\boldsymbol{X}}_c)$: if the *F-test* finds enough evidence to reject a model $\tilde{L}_{cr}$ in relation to the collapsed *full* model $\tilde{L}_c$, then it is clear that the model $\tilde{L}_r$ would also be rejected in comparison to the *full* model $\tilde{L}$.

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

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

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

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 them. 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 $\boldsymbol{Z}$ that fails our test still presents a genuine concern, as it directly suggests the selected $\boldsymbol{Z}$ 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, 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 [19]) 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

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

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 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 [19]. According to [20] this corresponds indeed to what they observed in similar experiments.

3.4 Finding the exact $\boldsymbol{X}$

It is not hard to see our collapsed *F-test* is only a necessary step for the *decision problem*, whereas in practice users might be interested to learn the answer of the *optimisation problem —what does the "exact"* $\boldsymbol{X}$ *contain?* Unfortunately, our test only provides some informal "clues": one can of course repetitively apply our test to find a model that *F-test* favoured; however, the order of adding/deducting terms play a critical role and the overall methodology becomes less reliable.

On the other hand, what we are genuinely interested in is not *what does* $\boldsymbol{X}$ *consist of*, but *why does* $\boldsymbol{X}$ *contain those elements*. In other words, apart from the fact that the "statistical diagnose" is far from ideal, we usually prefer the "(micro-)architectural diagnose", which cannot be simply asserted from statistics. For example, let us consider

$$L(X) = x_1 + x_2 - 2x_1x_2 = x_1 \oplus x_2$$

Two leakage models can end in this form, namely

$$L(Z) = \langle \boldsymbol{\beta}, Z \rangle, \boldsymbol{\beta} = \{0, 1, 1, -2\}, Z = \{1, x_1, x_2, x_1 x_2\}$$

$$L(Z') = \langle \boldsymbol{\beta}', Z' \rangle, \boldsymbol{\beta}' = \{0, 1\}, Z' = \{1, x_1 \oplus x_2\}$$

Unless further restrictions have been added for the coefficients ($\boldsymbol{\beta}$), from a nominal view, there is no chance to distinguish one from the other. However, these two models can have entirely different architectural reasoning, where the first one is about two concurrent bit with coupling/cross-line interaction, while the second could be about transition leakage. In this specific case, no statistical method can resolve the issue: although they are architecturally different, their leakage models are exactly the same.

Therefore, in this paper, we do not propose a method to find the exact $\boldsymbol{X}$: in practice, we believe it is more sensible to take these statistical clues as supplements for users' understanding about the device. Useful facts can be drawn from our test, yet the conclusion cannot be made purely from statistical reasoning. In our following experiments, we sometimes explore such clues to some possible architectural reasoning: note that our test here cannot guarantee the reasoning is correct.

4 Application 1: leakage analysis

4.1 Unprotected AES

To demonstrate how our test should be applied, let us start our discussion with a trivial target: the first round Sbox look-up in an AES-128 encryption. For convenience, we only analyse the first 4 Sbox look-ups: for all 16 Sboxes, one can take more traces and perform the same analysis, or further extend the model step by step to cover all Sboxes.

Experimental setup We select the TinyAES [21] 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 trivial way to analyse such implementation is following the "only computation leaks" principle [22]: 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 core is computing the first Sbox, we should see the leakage of both x_0 and s_0 but

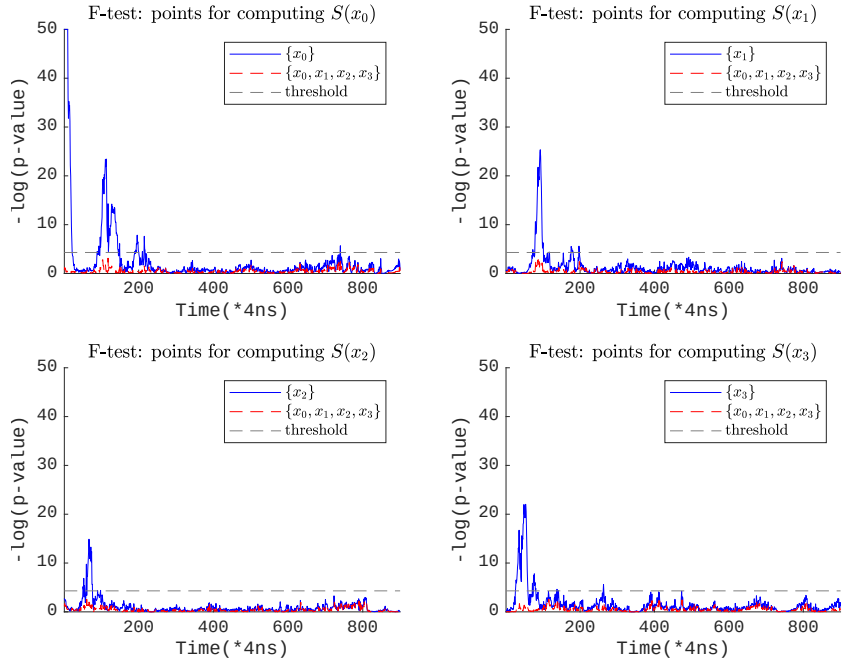


Fig. 2: Collapsed F-test on each Sbox computation

nothing else. As the outputs are completely determined by the inputs, following the same principle in Section 3.1, we define the uncollapsed complete model as

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

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

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

As we do not restricted the degree of $\tilde{L}$, s_0 is fully determined by x_0, therefore already included by $\tilde{L}(\boldsymbol{Z})$. Ideally, we should compare these two models in an *F-test*. Although the latter is fine, the former, as we stated in Section 3.2, cannot be built easily as it takes more than 2^{32} traces. However, if we "collapse" each byte to 2 bits[3], the former model contains only 2^8 terms whereas the latter has only 2^2.

Figure 2 illustrates the *F-test* results for all 4 Sbox computations. Clearly, "only computation leaks" is hardly enough: the measured leakage does contain more than just the computed byte. This suggests any following attack might be not optimal, while security evaluations/leakage certifications that built on

[3] Note that although $S(x_0)$ is still an 8-bit value, in a collapsed case, it only has 2-bit entropy and can be expressed with only 2 bits. Therefore $\tilde{L}(\boldsymbol{Z})$ can still portray any leakage caused by $S(x_0)$.

this assumption can be overly optimistic. We stress this is only a "can/may" rather than a "must": take attacks for instance, even if the missing leakage does contain useful information, considering the relevant key bits and the Signal to Noise Ratio (SNR), there is no guarantee it can definitely contribute to more efficient attacks. On the other hand, it is also quite risky to simply ignore the missing leakage and assume it will not help attackers.

One step further. Now let us further investigate what is the missing part . Through deliberately adding possible terms to the model and repeating the test, we found one acceptable leakage model is always assuming each computation leaks all 4 bytes. However, within 10k traces, it seems each byte's leakage is still on its own (i.e. no significant cross-byte interaction). The red line in Figure 2 suggests 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 regress attack [4] results (targeting each Sbox output) with the correct key in Figure 3: 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 2. The fact that the linear regression distinguisher can also find this peak confirms the existence and exploitability of such leakage in realistic attacks.

Realistic attacks. In a non-profiled attack setup, the existence of such extra leakage does not improve the overall attack. The full 32-bit key in our experiments can be reliably extracted within 200 traces, whereas the subtle leakage we found above only started to contribute after 1000 traces. The SNR gap makes adding such leakage pointless. Again, we argue that it is equally risky to proceed security evaluation with the trivial model and neglect such leakage, as the trade-off might change between each implementation, especially if the implementation is masked.

Architectural reasoning. As said before, the statistical reasoning cannot answer why such leakage appears. Therefore, we have no choice other than reading the source code and making guesses about the hidden processor details. The code in TinyAES [21] suggests these 4 bytes are stored within one word, but not accessed adjacently in Sbox look-up: 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 [11], 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

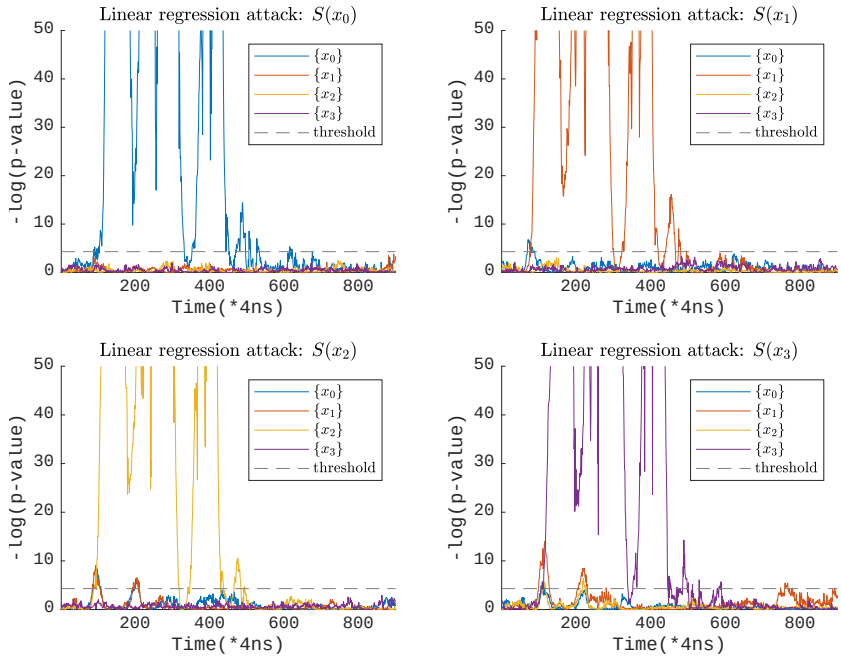


Fig. 3: Linear regression attack for each Sbox computation

ANSSI [23] as our target. The security of this implementation has been questioned by Bronchain and Standaert [24]: through the so-called "dissection" procedure, they had successfully recovered the full key with less than 2000 traced in a profiling setup. In fact, the critical part of this attack is performing "sub-attacks" on each trace in order to recover the temperate masks [24]. Considering such implementation requires repetitively loading masks from the memory, this "dissection" procedure is often quite effective in a profiling setup, where attacker can match the measured trace with a number of pre-built multivariate templates (eg. also applied by many attacks in the DPAContest [25]).

In the following, we investigate the leakage of this implementation in our experimental setup[4]. We stick with the same measurement setup as Section 4.1, where we analyse the masked Sbox calculation 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 authors'

[4] 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 this part.

analysis [23]), or take it as a follow-up analysis where the shuffling permutation has been already recovered using the technique in [24].

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 same spirit of Section 4.1, we analyse the leakage of the first Sbox look-up and uses 1 bit to represent the first 4 x_i. Similarly, 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} representatively. When collapsed to 1 bit, rm is restricted to 1 (i.e. nullifies the protection of rm). 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

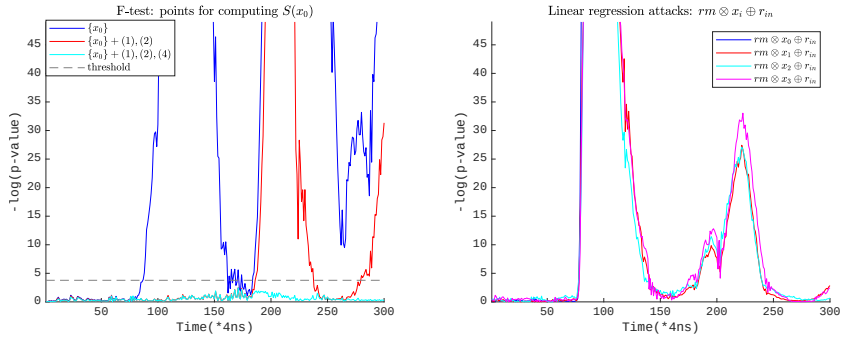


Fig. 4: Leakage analysis for the first Sbox

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

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

Following our common practice, it is expected that all the computed values are leaked plus some transitions in the code snippet. As a starter, let us first use some coarse-grained model that is ensured to captured all possible computation for the first Sbox:

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

Readers can easily verify that all the intermediate values appear in the code snippet can be expressed by this restricted model $\tilde{L}(\boldsymbol{Z})$. However, once again, we find this x_0-only model is hardly satisfying in the collapsed *F-test*: as we can see in Figure 4, the blue line clearly passes the threshold, which suggests the realistic leakage contains much more than what $\tilde{L}(\boldsymbol{Z})$ 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) loaded:

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

Line (2) loaded

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

If we added both these values (plus their transitions) into $\tilde{L}(\boldsymbol{Z})$, 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.

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

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

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 $za_1 = x_0||rin||rout||r_1$: adding $\boldsymbol{\beta}(\mathbf{Z}_{a1})$ 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 $\tilde{L}(\boldsymbol{Z})$ and construct a model that passes the *F-test* (i.e. the cyan line in the left half of Figure 4).

We further verifies the inference we made based on 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 [4] 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 4. 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 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 can plot the minimal p-value in Figure 5. 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 such 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 index 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* to readers, as it is out of the scope of this paper. However, based on our experimental results, the trace efficiency of this attack is no match for its profiling counterpart [24]. Nevertheless, our collapsed *F-test* clearly reveals new leakage presented in this implementation that enriches our previous understanding: whether such leakage is exploitable or can be exploited efficiently is not ensured by the *F-test* itself.

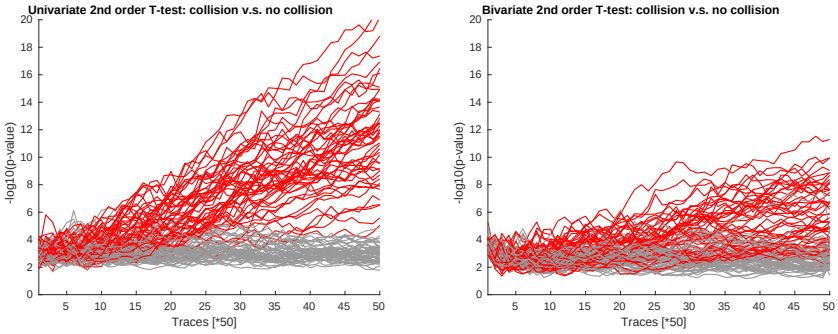


Fig. 5: Collision Oracle

Notes. Although slightly off the topic, we list a few 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 5, 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-profiling 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 5 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 [18].

5 Application 2: Leakage Simulators

In recent years, various leakage simulators have been proposed in order to enable early-stage leakage awareness [26]. Using a leakage simulator, developers can identify and patch potential leakage in a very 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 include 20k profiling traces to estimate and evaluate the model. All the setup remains the same as Section 4, except for the working frequency is reduced to 1 MHz: lower frequency helps to build a clearer cycle-to-cycle view, which is essential for analysing the quality of the estimated models. Note that every following statements will be specific to this setup: our test judges the leakage model based on the realistic measurements. Changing the setup affects the measurements, which should have an impact on the leakage model as well.

5.1 Instruction-wise modelling

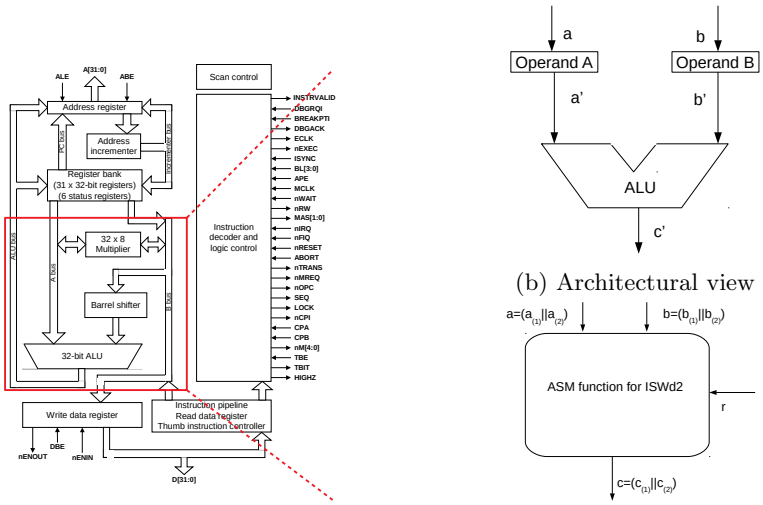
As pointed out by Buhan, Batina, Yarom and Schaumont, one of the remaining challenge for gray-box simulators is "(they) target relatively simple architecture" [26]. In fact, many tools only target at algorithmic variables that may correspond to multiple intermediate states in the core. Even if the simulator takes binary code (eg. ELMO [6]), the undocumented micro-architectural effects can still cause all sort of issues [11]. In theory, our collapsed *F-test* can be a confirmation step here: if a leakage simulator failed our test, it suggests some contributing factor can be missing in its leakage model. Thus, in the following, we first check whether the common instruction-wise leakage models are complete (or not), then move on to check whether such tools provide a complete model for an entire gadget (code snippet).

Architectural view Figure 6a shows a simplified architectural description for a realistic ARM M3 core [27]: 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 6b. Thus we can map the entire red block in Figure 6a to Figure 6b.

Common instruction-wise model A common simplification in many previous gray-box simulators is focusing on the execute leakage within the ALU (i.e. Figure 6b instead of Figure 6a). This choice is quite reasonable: even if the processor has a multi-staged pipeline, we do not necessarily care about the leakage from fetching the instructions (as it is often not data-dependent[5]). Following our principles in Section 3.1, the reference full model for Figure 6b can be written as

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

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



(a) Substantial view

(b) Architectural view

(c) Abstract view on the code in Section 5.2

Fig. 6: Materialise the prototype on software: architectural v.s. abstract

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:

- $\tilde{L}_l = \boldsymbol{\beta}\{A, B, C\}_l$: 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).
- $\tilde{L}_{le} = \boldsymbol{\beta}\{A, B, C, A', B', C', (dA), (dB), (dC)\}_l$, 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 [28]); it therefore also generalises the ELMO model [6] which only fits a single dummy for the Hamming distance leakage.
- $\tilde{L}_{TA} = \boldsymbol{\beta}\{AB\}$: this model represents template attacks [2], 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" [22].

Target instruction. Before any further analysis, we need to craft a code snippet that can trigger the simplified leakage in Figure 6b, while not causing any other

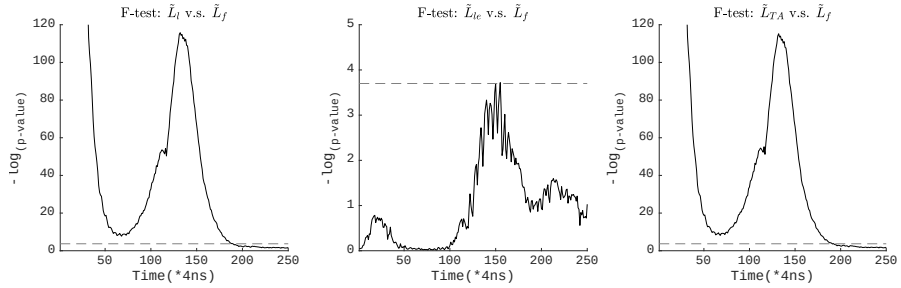


Fig. 7: Comparing various models against $\tilde{L}_f$

type of data-dependent leakage from other pipeline stages (i.e. *fetch* and *decode*). Unfortunately, this is not always possible in a commercial closed source core, as many resources lie in the micro-architecture. Nonetheless, in our experiments, we evaluate the following code snippet on our device:

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

eors r5,r7 represents the cycle we are targeting at in Figure 6b: the 2 pipeline registers are setted to value a and b, where the previous values are a' and b'. a' and b' are setted 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 fetching/decoding should be executing *nop*-s (which in theory, does not cause any data-dependent leakage[6]).

Collapsed F-test Despite we are working at an instruction level, considering each operand has 32-bit, building the full model $\tilde{L}_f$ is still infeasible. Thus, we still need to "collapse" $\tilde{L}_f$ to a smaller space. More specifically, we allow each operand to contain 2-bit randomness ($a = \{a_1 a_2 .... a_1 a_2\}$): 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 non-linear, which is of interest when comparing existing simulators.

Figure 7 shows the *F-test* results: clearly, models that exclude transitions in the inputs massively exceed the rejection threshold. This means that in these

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

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.

Conclusion. Clearly, none of the three models can be regarded as complete: even if we had crafted a code snippet that avoids various disturbance, these models do not looks promising. As a consequence, leakage detections built on these models could miss potential leakage, due to the limited explanatory power of their leakage model. Various effects could be contributing here (including the bit-interaction [10]): we re-emphasize that we cannot confirm what is the exact cause based on statistical reasoning alone.

5.2 Gadget-wise modelling

In this section we extend our study to a more interesting and 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 notations, 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 core, we need to be aware that our architectural view of the ALU (Figure 6b) may no longer be an adequate mental model for the actual reality in hardware (Figure 6a). Considering mapping to the architecture in Figure 6b already loses some information, it is expected that using a *full* model in Figure 6b is no longer *complete*.

Gadget-wise full model Alternatively, we can switch to a more abstract view: Figure 6c shows the functional view of our code in Table 1. Clearly, all functional

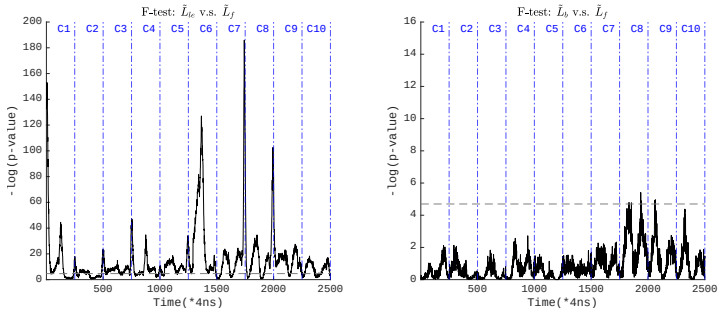


Fig. 8: Model comparison based on a 2-share ISW multiplication in software

inputs a, b and r no longer reflect any architectural port/bus/register. Having said that, assuming the core 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

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

Collapsed F-test With a proper definition of $\tilde{L}_f$, we can again challenge $\tilde{L}_{le}$ in the context of this snippet. Considering both ELMO and MAPS use a subset of $\tilde{L}_{le}$ (and $\tilde{L}_l/\tilde{L}_{TA}$ is not even close in Figure 7), we only tested $\tilde{L}_{le}$ in Figure 8. Compared with Figure 7, the left half clearly shows that this execute-only simplification is far from ideal. As a consequence, at least in theory, some leakage might be hard to find in both ELMO and MAPS.

One step further. Similar to Section 4, we can also try to build a better leakage model by adding terms and re-evaluating the model quality through *F-test*. Of course, the final output of this *ad-hoc* procedure ($\tilde{L}_b$ in Figure 8) does not have any architectural reasoning. Nonetheless, as we can see in the right half of Figure 8, such model is still a valid one and can potentially do better than $\tilde{L}_l$ in our *F-test*.

Considering here we have much more inputs with complicated computation than Section 4, building $\tilde{L}_b$ takes much more effort and sophisticated decisions. Each cycle usually involves a fairly complicated leakage model: in the following, we simply briefly summarise our findings. In our experience, it seems most operands will influence the leakage for at least 2 cycles, which seems to suggest the decoding stage does significantly contribute to the data-dependent leakage.

Leakage detection. We further investigate how these leakage models affect the efficacy of the leakage detections on the corresponding simulators. The last four columns in Table 1 show leakage detection results (obtained via a first order t-test) on realistic devices, simulations based on ELMO, simulations based on

MAPS and simulations based on the model we manually constructed respectively.

MAPS captures all register transitions, including the pipeline registers in the micro-architecture (command line option "-p") [28]. MAPS reported 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*[7].

ELMO captures the operands and transitions on the ALU data-bus [6]: perhaps surprisingly, ELMO reported exactly the same leaking cycles as MAPS. Detailed analysis shows that both cycles leak information from their operands' transitions: ELMO captured this as data-bus transitions, while MAP claimed this as pipeline register transitions. Considering the pipeline registers are connected to the corresponding ALU data-bus, this is hardly unexpected[8].

Our manually constructed model is indeed significantly better than both MAPS and ELMO as Table 1 shows. It reports the same leaking cycles as are reported for the real measurements. Specifically, cycle 5 reported leakage from the ALU output bus transition (aka dC in Figure 6b), 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.

At least in this specific example, it is evident that strongly failed *F-test* results do indicate that a model is likely to lack the explanatory power to be useful for simulators, which further affects the detection accuracy. If designers prefer to have some confidence about the accuracy of those tools, we recommend to at least capture some realistic measurements and verify their models' quality with our collapsed *F-test*: if the *F-test* already reports a warning, designers should keep in mind that some implementation flaw can potentially escape the early-stage detection.

6 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 $n = 8$

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

[8] 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.

(this corresponds to a small micro-controller), it is computationally expensive; but for $n = 32$ (this correspond to a modern microprocessor), it becomes clearly infeasible. We overcome this problem by introducing a novel statistical technique by 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 determine the *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, and our demonstration of its efficacy, we affirm a range of important points when it comes to attacks and simulations:

- Although our novel test does not guarantee the missing leakage will contribute to attacks, it is always recommended to apply this test and ensure the security evaluations are based on solid ground, rather than thin air.
- An accurate simulation model is not guaranteed to capture all relevant leakage. Therefore simulation accuracy (i.e. cross-validation R^2 or SSE) alone should not be the basis for a security certification in the style of [13, 16, 29].
- 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" [30] relies on specifying a leakage model (and therefore *state*). Clearly a leakage detection that is based on an incomplete *state* can miss out leaks.

References

1. Kocher, P.C., Jaffe, J., Jun, B.: Differential power analysis. In: Advances in Cryptology - CRYPTO '99, 19th Annual International Cryptology Conference, Santa Barbara, California, USA, August 15-19, 1999, Proceedings. (1999) 388–397
2. 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
3. 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
4. Doget, J., Prouff, E., Rivain, M., Standaert, F.: Univariate side channel attacks and leakage modeling. J. Cryptogr. Eng. **1**(2) (2011) 123–144
5. 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

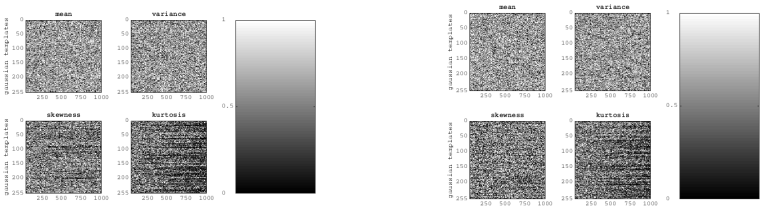
6. McCann, D., Oswald, E., Whitnall, C.: Towards practical tools for side channel aware software engineering: 'grey box' modelling for instruction leakages. In: 26th USENIX Security Symposium (USENIX Security 17), Vancouver, BC, USENIX Association (2017) 199–216
7. Papagiannopoulos, K., Veshchikov, N.: 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
8. 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
9. Meyer, L.D., Mulder, E.D., Tunstall, M.: On the effect of the (micro)architecture on the development of side-channel resistant software. IACR Cryptol. ePrint Arch. **2020** (2020) 1297
10. 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
11. Marshall, B., Page, D., Webb, J.: Miracle: Micro-architectural leakage evaluation. IACR Cryptol. ePrint Arch. (2021) https://eprint.iacr.org/2021/261.
12. 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
13. 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
14. Crama, Y., Hammer, P.L., eds.: Boolean Models and Methods in Mathematics, Computer Science, and Engineering. Cambridge University Press (2010)
15. Durvaux, F., Standaert, F., Pozo, S.M.D.: Towards easy leakage certification: extended version. J. Cryptogr. Eng. **7**(2) (2017) 129–147
16. 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
17. Gao, S., Marshall, B., Page, D., Pham, T.H.: FENL: an ISE to mitigate analogue micro-architectural leakage. IACR Trans. Cryptogr. Hardw. Embed. Syst. **2020**(2) (2020) 73–98
18. Whitnall, C., Oswald, E.: A cautionary note regarding the usage of leakage detection tests in security evaluation. Cryptology ePrint Archive, Report 2019/703 (2019)
19. 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
20. 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
21. kokke: Tiny aes in c
22. 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
23. Benadjila, R., Khati, L., Prouff, E., Thillard, A.: Hardened library for aes-128 encryption/decryption on arm cortex m4 achitecture
24. 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
25. ParisTech, T.: Dpa contest 2008/2009
26. Buhan, I., Batina, L., Yarom, Y., Schaumont, P.: Sok: Design tools for side-channel-aware implementions (2021)
27. Limited, A.: Arm7tdmi technical reference manual. https://developer.arm.com/documentation/ddi0210/c/ (2004)
28. Corre, Y.L., 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
29. 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
30. Gilbert Goodwill, B.J., Jaffe, J., Rohatgi, P., et al.: A testing methodology for side-channel resistance validation. In: NIST non-invasive attack testing workshop. Volume 7. (2011) 115–136
31. 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

A PI, HI & Assumption error

Leakage certification approaches such as described in [13, 15, 16] (based on the general framework introduced by Standaert, Malkin and Yung [31]) 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 $\boldsymbol{Z}$ that ensures the corresponding model $\tilde{L}(\boldsymbol{Z})$ is close to the *full* model $\tilde{L}(\boldsymbol{X})$ w.r.t. its explanatory power. Then, one can estimate the mutual information of $MI(\boldsymbol{Z}; L)$ using concepts like perceived information (PI) or hypothetical information (HI) [16].



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

Fig. 9: Moment based detection of "assumption error"

The common choice for $\boldsymbol{Z}$ is often a variable that relates to a single S-box [13, 15, 16]: because the MI calculation runs through all possible values of $\boldsymbol{Z}$, 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.1 Estimating "assumption errors"

In [13] Durvaux et al. proposed a technique to test for (the so-called) assumption errors in the leakage model [13]. 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 $\boldsymbol{Z}$ 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 [15], we can validate the power of detecting the above "assumption error": Figure 9a 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 9b), the situation remains the same. Only the *kurtosis* gives some small p-values, but there is no statistical decision

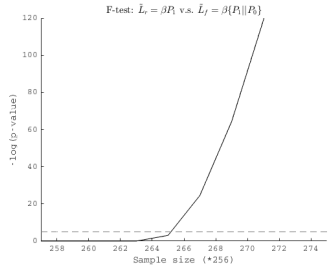


Fig. 10: F-test with noise variance 0.1

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 $p_1 \oplus k_1$, 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 [13] 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. 10. The advantage here requires though to explicitly assign $\boldsymbol{X} = \{P_1 P_0\}$. 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 [16].

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 [16]: as we can see in Figure 11, 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.

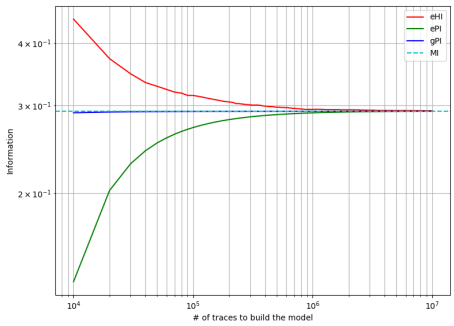


Fig. 11: PI and HI estimation for the leakage function

A.3 Bias-Variance Decomposition

Lerman, Veshchikov and Markowitch et al. also proposed a diagnosis tool based on the bias-variance decomposition [29]. 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*.