A Novel Framework for Explainable Leakage Assessment

Si Gao and Elisabeth Oswald

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

Abstract. Non-specific leakage detection (initially introduced as "Test Vector Leakage Assessment", short TVLA) plays a vital role in practice because it detects (potential) leaks independently of assumptions about the leakage model and any specific attack vector. However, the non-specific nature means detected leaks might not be exploitable, and thus the current state of the art is to employ a battery of specific attacks to confirm the detection outcomes.
We propose a novel leakage assessment framework which enables to link non-specific leakage detection outcomes with size of the key guess that is necessary to exploit them. We therefore solve the problem of deciding if or not a leak is exploitable without the need for specific attacks. Our methodology furthermore enables (for a detected leak) to reveal the specific key bytes, and with that, it allows the construction of confirmatory attacks. This novel approach is enabled by proposing to cast the leakage detection problem as the statistical task of building *key-dependent* regression models: if such a model exists, then we know that the point leaks, and depending on the size and nature of the model, we can judge exploitability and provide a concrete attack vector.

Keywords: Leakage detection, Side channel analysis

1 Introduction

In the past two decades, the threat of side channel attacks has prompted academia and industry to come up with different ways to evaluate real-world products. Two evaluation philosophies have emerged: attack based evaluations (e.g. Common Criteria [1] and EMVCo [2]) versus compliance testing (e.g. FIPS 140-3 which is based on ISO 17825 [3]).

Attack based evaluation regimes require the evaluation lab to attempt "all" of the most effective known attacks to date. With new attacks developed each year, testing "all attacks"[1] quickly becomes increasingly infeasible.

[1] We use inverted commas here to indicate that in practice a sophisticated system of negotiation between certification authorities, evaluation labs and manufacturers who form the JHAS group takes place that results in a shared understanding of how to rate and assess attacks [4, 5]

A compliance testing based approach based on a "non-specific" leakage detection test — called Test Vector Leakage Assessment (TVLA)—was proposed back in 2011 by Cryptography Research, Inc. (now Rambus) [6]. Instead of testing whether an attack (targetting a specific state that occurs during a cryptographic operation) succeeds, TVLA searches for statistical evidence of *any* potential side channel leak (i.e. it is not specific to any intermediate state). This is achieved via the "fixed-versus-random" test: for a fixed key, side channel measurements are taken for a fixed plaintext input and for random plaintext inputs. A statistical procedure then decides if or not the two datasets come from the same distribution (if they do not, this indicates dependence on the plaintext and thereby side channel leakage). An implementation is then seen as insecure if the fixed-versus-random test shows any leakage.

At first glance, the non-specific detection based approach appears considerably more attractive than doing lots of specific attacks: using this non-specific detection, any type of leak is revealed with a single statistical procedure that requires few assumptions about the device and little control over the inputs. Upon closer inspection though, there is a serious shortcoming when relying (solely) on this non-specific detection: a detected leak may not necessarily be exploitable in an attack. For example, a leak that corresponds to a step in the second round of a block cipher may require a key guess that is too large to be practical for an attack.

Since the popularisation of the detection based evaluation paradigm, a number of papers have appeared that discuss and aim to improve the status quo, often focussing on another shortcoming (the probability of false positives) [7, 8]. In addition to these, papers suggested using a test statistic with more advantageous properties like χ^2 [9] or by looking at tweaks to the fixed-versus-random strategy leading to the fixed-versus-fixed strategy or moving towards more specific tests/attacks [10, 11]. Confirming TVLA results with specific attacks, implies though that the evaluator has to revert back to trying different attacks targetting different intermediate steps with different power models because none of the existing tests is able to give any indication *what* leaks. Evidently then, following non-specific detection with a barrage of attacks that require a large amount of guessing what leaks and how to exploit it is, after all, against the very intention of a non-specific leakage assessment.

1.1 Our contributions

In this paper, we propose a novel technique for leakage assessment that is the first to bridge the gap between non-specific leakage detection tests and specific attacks/tests without the need for guessing. It is based on a new application of *key-dependent* leakage modelling [12] (using the F-test on so-called collapsed inputs) as a non-specific leakage detection technique. This adaptation essentially gathers evidence in relation to statistical hypotheses that seek to explain the observed side channel data with key dependent models. If much of the observed data can be explained, then the observed data must show key dependence, and thus have side channel key leakage. Because of how we construct the test, we can infer the size of the necessary key guess to do a practical side channel attack (our assessment methodology explains if or not a detected leak is exploitable), and set up an exemplar attack for any leak in question without having to randomly guess target functions (our assessment methodology identifies exploitable key bytes which can be linked easily with intermediate steps).

Our technique can be used as a stand-alone methodology, or it can be used as a second step after a fixed-versus-random analysis. In the latter case, one would subject any leakage point that is flagged by the fixed-versus-random test to our novel method. Our method then clarifies if the leakage point depends on any key material (and the amount of effort required to exploit it).

Limitations. Our methodology enables to explain if and how observed leaks depend on secret key material, but it does not help to explain *why* the leakage is there. The reasoning for why there is leakage typically has to look at the interplay between the specific implementation option that we chosen and the (micro)architectural leakage of a device. Thus whilst our test helps to narrow down which intermediate computation step is the likely culprit of the leakage, it cannot explain why it is the culprit.

Our method is, like the TVLA framework, set up as a univariate method: if leakage depends on more than a single intermediate, then this cannot be dealt with as such. Like within the context of the TVLA framework, one must initially preprocess the traces, and only afterwards a fixed-versus-random test can succeed: the same constraint holds for our method. We leave it as future work to consider multivariate extensions.

Outline. The following section presents some basic background knowledge, including non-specific leakage detections, regression modelling and how the F-test can be utilised in side-channel applications. Section 4 describes our retrospective leakage analysis step by step using a simulated AES as an explanatory example. We further challenge our novel technique with measurements from a realistic masking implementation in Section 5, testing its efficacy in both theoretical and practical use cases. Section 6 demonstrates how our technique can be easily extended to higher order analyses, revealing valuable information as well as deriving trivial attacks.

2 Preliminaries

For ease of reading, we recall several basic statistical facts about two pertinent statistical tools: the Student t-test and the Fisher F-test. Alongside recalling their principles, we introduce the notation that we maintain throughout this papers.

2.1 Non-specific leakage detection

The adjective “non-specific” in the context of leakage detection means that the statistical test does target any specific intermediate state of the device. This in contrast to attacks or specific detection tests, which always work by targetting a specific intermediate device state. Non-specific procedures hence promise to detect all leaks, whereas any specific test can only ever confirm a leak that directly relates to the targetted (specific) intermediate state.

The TVLA document [6] suggests using a "fixed-versus-random" test can potentially ensure the test is not only non-specific about a target intermediate state, but also not restricted by any device leakage model assumptions (e.g. the Hamming Weight model). The document is in fact a compact set of engineering guidelines that, alongside explaining the statistical method for leakage detection, also touch on the setup and the data collection.

The statistical process works as follows. The evaluator collects two trace sets: $\mathcal{Q}_f$ and $\mathcal{Q}_r$. The set $\mathcal{Q}_f$ consists of traces where the plaintext and key to the cryptographic algorithm are fixed to some (arbitrary) value. The set $\mathcal{Q}_r$ consists of traces where the plaintexts to the cryptographic algorithm are chosen randomly, but the key remains at a fixed value.

We denote by n_f and n_r the corresponding sample sizes (i.e. number of traces in $\mathcal{Q}_f/\ \mathcal{Q}_r$), μ_f and μ_r the sample means and s_f^2 and s_r^2 the sample variances. TVLA now deploys Welch's t-test, which is a univariate hypthesis test, with the following hypotheses (H_0 indicates no leak, which is the hypothesis that we are trying to refute):

$$H_0 : \mu_f = \mu_r$$

against

$$H_1 : \mu_f \neq \mu_r$$

To refute H_0 one computes the t-test statistic and the corresponding degrees of freedom v:

$$t = \frac{\mu_f - \mu_r}{\sqrt{\frac{s_f^2}{n_f} + \frac{s_r^2}{n_r}}} \qquad v = \frac{\left(\frac{s_f^2}{n_f} + \frac{s_r^2}{n_r}\right)^2}{\frac{\left(\frac{s_f^2}{n_f}\right)^2}{n_f - 1} + \frac{\left(\frac{s_r^2}{n_r}\right)^2}{n_r - 1}}$$

The p-value can be derived from the quantile function of Student's t-distribution: smaller p-values give evidence to reject the null hypothesis which leads to conclude that a potential leakage is detected. The original proposal of [6] simplified the computation of the quantile function by using the threshold $t = \pm 4.5$ [6], which is close to $p = 10^{-6}$ for a large number of traces.

Because the test statistic is univariate, application of it to traces where key dependencies are only evident via multivariate statistics is doomed to fail. However, the well known mean-free product combining trick can circumvent this issue, and has been successfully applied to masked implementations (e.g. [13]).

2.2 Regression modelling and F-test

To date, the primary use of regression modelling in the side channel community has been to estimate the leakage distribution associated with some intermediate computation of a device (aka template building). We call the relevant n-bit intermediate state as X, and any concrete value that X takes by x. A template [14] for x is simply an estimate of the distribution of the leakage of x. For instance, assuming the leakage of x follows a Gaussian distribution, the template for x is given by $\mathcal{N}(\mu_x, \sigma_x^2)$. Instead of estimating the parameters of a Gaussian distribution, one can use linear regression to derive an equivalent model [15]. More specifically, any real valued function of X as can be written as a polynomial $L(X) = \sum_j \beta_j u_j(X)$. In this expression, the explanatory variables $u_j(x)$ are monomials of the form $\prod_{i=0}^{n-1} x_i^{j_i}$ where x_i denotes the i-th bit of X and j_i denotes the i-th bit of j. Regression modelling then estimates the (linear) coefficients β_j. As a result the leakage model of X is now expressed as $L(X) = \sum_j \beta_j u_j(X)$ (and we evaluate it by setting $X = x$). Evidently, without any restriction on u_j, there are exactly 2^n parameters to estimate in the regression equation, which is equivalent to estimating 2^n Gaussian distributions. However, when only certain parts of u_j are allowed (e.g. we only include linear terms where $HW(j) = 1$), then the regression approach requires the estimation of fewer parameters, which leads to a better estimation quality given some fixed size trace set.

Evaluating the "quality" of a regression-built model is a well understood challenge: a commonly used metric is the *coefficient of determination* (R^2), which portrays the ratio of explained/unexplained data variance. However, the principle of Least Squares Estimation (LSE) implies that when adding a new explanatory variable u_j to an existing model, R^2 will increase. In other words, R^2 is not a sensible comparison metric between models that have different numbers of parameters. A special case, that is of particular interest to us, is the case of models that are "nested" (i.e. one *restricted* model consists of only a subset of explanatory variables in the other *full* model).

Given two "nested" models, the F-test becomes the more suitable solution to compare two regression models. Specifically, with N measurements, if we are aiming to compare a *full* model $L_f(X) = \sum_j \beta_j u_j(X)$, $j \in J_f$ and a *restricted* model $L_r(X) = \sum_j \beta_j u_j(X)$, $j \in J_r \subset J_f$, we can construct the following hypothesis test:

- H_0: $\{u_j | j \in J_f \setminus J_r\}$ have coefficients 0 in $L_f(X)$
- H_1: $\{u_j | j \in J_f \setminus J_r\}$ have non-zero coefficients in $L_f(X)$

Informally, if the test above rejects H_0, we can conclude the missing explanatory variables have a significant contribution (i.e. $L_f(X)$ is statistically superior than $L_r(X)$). Denote $z_r = \#\{J_r\}$, $z_f = \#\{J_f\}$ and the number of available measurements as N, we compute the F-statistic as

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

where the *residual sum of squares* (RSS) is defined as

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

where $y^{(i)}/x^{(i)}$ represents the i-th measurement/the corresponding x.

The resulting value F follows the F distribution with $(z_f - z_r, N - z_f)$ degrees of freedom. A p-value below a statistically motivated threshold rejects the null hypothesis (i.e. the two models are equivalent) and hence suggests that at least one of the removed variables is potentially useful.

Using the ratio between variance estimates from different models was identified as a potential distinguisher called NICV (normalized inter-class variance)[16], which (as all distinguishers) can be used for specific leakage detection (or in other words, a known key attack).

3 A Novel Non-Specific Leakage Assessment Framework based on Key-Dependent Nested Models

Our framework consists of several steps, which we detail in the subsequent sections. These steps are refinements of the initial step, which identifies if there is *key-dependent* leakage or not. So far instantiations of the non-specific leakage detection method have all relied on the "fixed plaintext versus random *plaintext*" principle and suggested different statistical tests, e.g. [9, 6, 8]. However a core issue with the fixed-versus-random *plaintext* approach is that it targets *plaintext* dependencies: of course, cryptographic algorithms early on mix the plaintext and the key material, and thus side channel leaks reveal key dependencies in an encryption round implicitly. However leakage from the key schedule for instance cannot be revealed in any shape or form. Therefore in our framework we fix a plaintext, and then perform a"fixed-key versus random-key" detection process. We propose to rephrase the task of leakage detection as the task of deciding if the observed leakage can be explained by key-dependent regression models. If the data can be explained via a key-dependent regression model, then the data must be key dependent, and therefore a side channel key recovery attack could succeed.

Varying the key allows us to detect all key dependencies (the first step in our framework), whilst ignoring pure plaintext dependencies (i.e. no secret involved). However, this step alone does not give us any additional information about whether or not an indicated leak is exploitable by an actual side channel attack. Similar to conventional cryptanalysis, we consider a strategy as a valid attack only if it requires a key guess that is smaller than the full key space. Different to conventional cryptanalytic attacks, a side channel attack would not be considered as practical if it requires an unrealistically large key guess.

In the next step of our framework we assess if or not the indicated leaks from a non-specific detection test can be exploited in concrete attacks. We do this via assessing "restricted regression models". The last two steps of our framework allows identifying the exact key bytes (that are leaking) and with that derive some actual attack vectors. The first two steps of this framework deliver the non-specific detection of exploitable key leakage. Thus they provide a sound basis on which to reject (or accept) an implementation in an evaluation. The second two steps enable to explain what leaks and how to exploit it. Therefore they are helpful in identifying and fixing an implementation.

We now translate this informal description into mathematical formalism. Our setting here is compatible with the assumptions of the original TVLA process: we have control over the device (i.e. we control input and key, but not any internal randomness), and we treat side channel traces by testing each trace point independent of all other trace points (i.e. a univariate setting). Each of the following subsections details one of the steps in our leakage assessment framework.

3.1 Non-specific detection via key-dependent models (step 1)

We consider the two nested key-dependent models:

$$\begin{aligned} L_f(K) &= \sum_j \beta_j u_j(K),\ j \in J \\ L_0(K) &= \beta_0 \end{aligned}$$

The coefficients β_j are estimated from the side channel traces via least square estimation. The full model L_f fits a model as function of the key K to the observed data. The reduced model L_0 now contains only a constant term (i.e. the *null* model), which represents the case where there is no dependency on K. We test the null hypothesis H_0 (both models explain the observed data equally well) versus the alternative hypothesis H_1 (L_f explains that observed data better) with the F-test. If the F-test rejects H_0 (i.e. no key dependence), we conclude that the measurements are depending on K.

Whilst this is mathematically sound, the complexity of this test depends on the size of K. For a modern cryptographic scheme, this usually leads to a practically infeasible space to work with (e.g. $|K| = 2^{128}$). Recently, a workaround has been proposed to deal with this problem [12]. This work shows that it is sound to map the full key space into a smaller space, by collapsing each key byte to a smaller space[2]. For instance a single key byte can be represented by just a single bit of randomness, i.e. a byte is either 0 or 255 (all zeros or all ones), in which case the corresponding collapsed full model captures interactions between different key bytes. It is also possible to represent a key byte with more randomness in which case the corresponding full model then captures interactions between key bytes and within key bytes.

In the following analysis, we always use AES as our target cipher: as most operations in AES work on a byte level, we collapse each key byte to a single bit of randomness (the key independent model remains the same)

$$\begin{aligned} L_{cf}(K_c) &= \sum_j \beta_j u_j(K_c),\ j \in [0, 2^{16}) \\ L_0(K) &= \beta_0 \end{aligned}$$

[2] We adopt the notation and language from [12] although this trick can work for arbitrary "chunks" of bits of key material.

Thus our non-specific leakage detection test consists of estimating L_{cf} and L_0, from the observed side channel traces (point by point), and testing H_0 (the models explain the data equally well) versus H_1 (L_{cf} explains the data better than L_0) via the F-test. If our test finds enough evidence to reject H_0 for a trace point in this collapsed setting, we may conclude this point's leakage relies K_c. Considering K_c is a part of K, we can directly infer it also depends on K in the original un-collapsed setting (formal analysis see [12]).

Note. The collapsing setting relies on the target cipher, implementation as well as the acceptable complexity. The byte-to-bit strategy here suits AES's nature, while ensuring the complexity of the collapsed model is still acceptable (i.e. 2^{16}). Depending on the target ciphers and goals, users may choose a nibble-to-bit strategy (e.g. nibble-based ciphers), or even a bit-by-bit, but divide-and-conquer analysis.

Choice of statistical parameters We have yet to determine the number of side channel observations that are necessary for this test to succeed with high confidence. The confidence in test outcomes are reflected in the percentage of false positives α (i.e. trace points are identified as leaky although they are not) and the percentage of false negatives β (i.e. trace points that are leaky but that are not identified).

All parameters in any statistical test interact with each other, thus the α, the $1-\beta$, the difference between the two statistical hypothesis (aka the effect size f^2), and the number of observations N, need to be considered jointly. Luckily, the F-test is extremely widely used and there are explicit formulae for the α and β [17]. With these it is possible to set the decision threshold for rejecting the null hypothesis:

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

where Q_F is the quantile function of the central F distribution. Considering the tested full model contains z_f estimated parameters, while the null model L_0 always contains only one parameter , these two degrees of freedom are defined as:

$$df_1 = z_f - 1, df_2 = N - z_f.$$

Thus, the statistical power $1-\beta$ can be computed as

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

where F_{nc} is the cumulative distribution function of the non-central F distribution.

Given a selected set of (α, N, f^2), one can verify if the underlying statistic power $(1-\beta)$ reaches his/her expectation. As the case in compliance based testing based on TVLA, users may initially agrees on the number of traces, e.g.

$N = 10.000$ for FIPS Level 3 or $N = 100.000$ for FIPS Level 4, or $N = 1.000.000$ for a higher level of security as would be aimed for in CC [1], and also an acceptable level of α. Users can then determine if the statistical power $1 - \beta$ is acceptable via the supplied equations.

Take our leakage detection test above for instance, we compare the collapsed full model and the null model, where $df_1 = 2^{16} - 1$ and $df_2 = N - 2^{16}$. For a small effect size $f^2 = 0.02$ [17] with $N = 1.000.000$, $\alpha = 10^{-6}$, we have $1 - \beta \approx 1$, which suggests our test has highly effective for many realistic side channel leaks [18].

Note. As explained before, when examining an implementation with a countermeasure such as masking, any univariate test is likely to be ineffective when applied naively. In such a scenario, the traces must be processed prior to any statistical evaluation, by the so-called centred product combining technique [19] (multivariate case) or centred moment technique [13] (univariate case).

3.2 Degree Analysis (step 2)

In contrast to existing leakage detection processes, our statistical tooling enables us to determine *how large a key guess* is required to exploit an identified leak. We can do this by further restricting the degree of the key-dependent model that we are estimating from the side channel observations.

$$L_{cf}(K_c) = \sum_j \beta_j u_j(K_c);\ j \in [0, 2^{16})$$
$$L_{cr}(K_c) = \sum_j \beta_j u_j(K_c);\ j \in [0, 2^{16}), deg(u_j(K_c) \leq d)$$

We test again H_0 (both models explain the data equally well) versus H_1 (L_{cf} explains the data better than L_{cr}). If there is enough evidence to refute H_0, we can conclude that a model with only d or fewer key bytes suffices to explain the data. By successively reducing d we can therefore determine the maximum key guess that is required to explain the observed side channel data.

Note. Restricting d implies that all combinations of at most d key elements are included in the model. It is also possible to narrow down a specific combination of d key elements. In the latter case we need to test $\binom{16}{d}$ different models (corresponding to any d combinations of key bytes. In order to reduce the implied effort, it is advisable to consider what key guesses are practically possible. In the literature there exist attacks that exploit key guesses up to 32 bits [20]. Assuming that we are considering key bytes, then, we should definitely consider $d \leq 4$. When allowing for some extra margin, it seems sensible to consider $d \leq 6$, or if particularly conservative to consider $d \leq 8$.

Algorithm 1 Subkey Identification and Mapping to Attackable State

```
Require: the collapsed master key $K$
Require: Corresponding traces $Tr$
Require: Degree $d$
 1: for $i = 0$ to 15 do                                ▷ Check for not contributing subkeys
 2:     $K' = K - \{k_i\}$
 3:     $F$-test with $X' = K'$ and $X = K$
 4:     if $pv > \alpha$ then
 5:         $K = K'$
 6:     end if
 7: end for
 8: $l = |K|$
 9: $J = [0, 1, ..., 2^l - 1]$
10: Sort $J$ with the ascending order of the Hamming Weight
11: Delete any $j$ from $J$ if $HW(j) > d$                     ▷ Restrict order to $d$
12: while $J$ is not empty do
13:     $j = J[0]$, construct $u_j(K)$
14:     Specific test on $u_j(K)$, producing a statistic $p$-value $pv$
15:     if $pv < \alpha$ then
16:         Find one potential leakage, return $j$
17:     end if
18:     Delete $j$ from $J$
19: end while
```

3.3 Subkey Identification (step 3)

Using the technique of further restricting the reduced model (as hinted at in the note before), we can narrow down precisely which and how many key bytes are required to explain an identified leak. We do this by placing further constraints on the reduced model, e.g. by restricting its degree and including or excluding specific terms (corresponding to specific key bytes). We then test this further reduced model against the collapsed full model to check if these further restrictions are relevant or not. Algorithm 1 provides a high level description of this processes: we initially check if any subkey does not contribute to the overall joint leakage, and then based on the supplied maximum degree d we remove any term that exceeds the maximum degree to narrow in on the subkeys that actually leak. The while loop then tests every remaining term if it individually explains leakage better than the null model (if so, it become a candidate for a specific attack in the next step).

Note. In modern encryption schemes it is apparent from the description how key bytes interact, and at which point the full secret key has been mixed in with the plaintext, e.g. in AES it takes two rounds to achieve full diffusion of the key. Consequently, in the inner encryption rounds leaks are often non-exploitable in any efficient way. Clearly this knowledge is algorithm specific, but public, and it can be taken into account when identifying subkeys.

3.4 Converting to specific attacks(step 4)

If an evaluation regime requires an evaluator to demonstrate an actual attack against an identified leakage point (i.e. a point that was confirmed to depend on the key using our statistical tooling), we can now establish a relatively easy link towards a concrete profiled attack. More specifically, for any j found in Alg. 1, we can trivially build templates for all relevant subkey (i.e. at most $8d$-bit) and perform matching those key bytes in a realistic attack (as shown in Section 5.2).

However, considering there is little cipher-specific construction in this trivial attack, its efficiency is often far from ideal. We argue that in practice, a more sensible solution is taking advantage of the information revealed by j (alongside users' knowledge about the cipher and implementation) to design more efficient attacks. Specially, following the working principle of the encryption algorithm, we can link the relevant key bytes with involved plaintext bytes. Furthermore there will be only a very limited number of intermediate computation steps that depend on the identified combination of plaintext bytes and key bytes, which enables us to have a short list of potential target intermediate states.

Consequently, we can now construct a specific attack by either attacking the assumed intermediate states with known power models, or, if the size of the key guess allows this, directly performing template matching for all relevant key bytes. In order to verify the derived attack, we switch back to the typical"fixed secret key, chosen plaintext" scenario where an "attack" trace set can be generated in a more practical setup (i.e. without varying key or collapsed inputs). Since the leakage detection setting allows chosen plaintext access, we can even set all irrelevant key/plaintext bytes to zero, thereby enabling a much higher signal to noise ratio (SNR) to showcase subtle security concerns that could be easily overlooked otherwise.

4 Leakage Assessment: Example for Simulated Skeleton AES

For clarity, we illustrate our methodology using simulated leaks at first. The simulation corresponds to two rounds of AES (with a key size of 128 bits). We record six leakage points across the two rounds. Unlike in the experiments in Section 5 and 6, which are based on a real world device, we know exactly how each simulated trace point is generated. Consequently, this enables to test (and also demonstrate) our novel leakage assessment process.

4.1 Simulation setup

Our simulation consists of two rounds of AES. An AES round consists of the execution of four intermediate steps (AddRoundKey, SubBytes, ShiftRows, and MixColumns), and we create from these intermediate steps six leakage points, using the Hamming weight (HW) as a leakage function, as detailed in the list below. The selection of leakage points is meant to enable an interesting analysis: thus we choose intermediate states and leakage functions that are of some practical interest (any indices though where chosen arbitrarily and they have no impact on the actual analysis). Note that in a real world leakage evaluation, the evaluator does not know the exact correspondence between leaking intermediate variables and trace points (which is the very reason for needing an additional analysis after leakage detection).

- $l_0 = HW(Plain[0:3])$: this trace point is the leakage from the plaintext input. We consider a modern 32-bit processor, thus we return Hamming weight of the 32-bit word.
- $l_1 = HW(Rkey[0:3])$: similarly, this trace point is the leakage from the first word of the first round key (i.e. the master key for AES)
- $l_2 = HW(Sout[4])$: this sample represents a typical target for a side channel attack, i.e. the leakage from the output of one specific S-box (the index here is irrelevant, and was selected at random).
- $l_3 = HD(Sout[6], Sout[10])$: this trace point is an example of typical Hamming Distance (HD) leakage on the memory bus (between two randomly selected SubBytes results). This type of leakage often exists in modern processors, and it is often not trivial to determine in practice which two values are involved in generating it (i.e. compiler specified ordering, register allocation etc.).
- $l_4 = HW(MCout[8])$: this trace point represents the leakage of the MixColumns output. It represents exploitable leakage that requires a large key guess.
- $l_5 = HW(MCout[12])$: l_5 also relates to a MixColumns output, but from the second round.

Some independent Gaussian noise is added to each sample, drawn from $\mathbb{N}(0, \sigma^2)$ where $\sigma^2 = 16$. Our goal here is to demonstrate how our technique reveals the relevant key bytes for each sample. For the typical 8-bit HW leakage samples (i.e. l_2, l_3, l_4 and l_5), the Signal-to-Noise-Ratio (SNR) is $2/16 = 0.125$ (this corresponds to what is observed in modern microprocessors).

To emulate a typical leakage detection scenario within a "high stakes" evaluation (such as CC) we assume that we can sample 1.000.000 traces.

4.2 Applying TVLA

Since the AES implementation is unprotected, no trace processing is necessary prior to the application of TVLA. Using the conventional plaintext-based fixed-versus-random t-test, all the samples are reported as "leaky" (see the "TVLA" column in 1), except for l_1. This result is entirely expected: anything related to computations on the round keys cannot be detected by varying plaintexts. Since the correspondence between trace points and intermediate states is missing, no conclusion beyond "there is potentially some leakage" can be drawn.

4.3 Leakage assessment via key dependent models

We now demonstrate our assessment framework by following the statistical analyses laid out in Sections 3.1-3.4.

Table 1. Example: Simulated AES

Samples \Tests	$TVLA$	Key-Dependent Model-Based Leakage Assessment			
		Non-spec. Detect	Degree	Key Bytes	Specific
$l_0 = HW(Plain[0:3])$	✓		-	-	-
$l_1 = HW(Rk0[0:3])$		✓	1	k_0, k_1, k_2, k_3	k_0
$l_2 = HW(Sout[4])$	✓	✓	1	k_4	k_4
$l_3 = HD(Sout[6], Sout[10])$	✓	✓	2	k_6, k_{10}	(k_6, k_{10})
$l_4 = HW(MCout[8])$	✓	✓	4	k_2, k_7, k_8, k_{13}	(k_2, k_7, k_8)
$l_5 = HW(MCout[12])$	✓	✓	> 4	all	n.a.

Non-specific detection. Since our simulation is unprotected, all key-dependent trace points l_i can be written as functions of K.

However, a key challenge for performing such analysis is the size of K and thus we use the collapsing trick explained in Section 3.1: we use a single bit (e.g. the least significant bit) to represent one byte. Applying the same restriction on all 16 bytes, the entire 128-bit key space can be represented by a 16-bit collapsed key state K_c. In this collapsed model, $L_f(K_c)$ represents all possible leakage, yet only requires the number of traces N to be several times of 2^{16}. For our simulation experiments in this section, we stick with the same setting as TVLA (i.e. $N = 10^6$).

We want to emphasise at this point that there are multiple options of how to collapse the full key space, and one has to be familiar with the cryptographic scheme that is implemented. For instance, because the AES round function mainly works on the byte-level (except for the *xtime* function in MixColumn), using one bit to represent each byte is a natural choice. As consequence, some intra-byte leakage cannot be detected this way, e.g. the bit-interaction leakage within a software processor [21]. For such cases (or bit oriented block ciphers), a single byte should be represented by two bits, enabling the detection of interactions within a byte.

In addition, the actual values of the collapsed representation affect the efficiency of the test. For instance we can choose the values 0 and 1 to represent a byte, but the leakage may be harder to exploit than when choosing 0 and 255 (aka all bits are set to one). For AES-128, 00...0 and 11...1 usually produce signifiant leakage before the S-box; however, after the S-box, these values are mapped to the values $0x63$ and $0x16$, which only have only 5 bits difference. Considering MixColumn intensively mixes various bytes, in the following we use one-bit to represents each key byte k_i as $0x52$ or $0x7d$: by sacrificing some power before the S-box, we gain full power after the S-box as each byte becomes 0 or 255.

We can now set up the leakage detection test as:

$$L_f(K_c) = \sum_j \beta_j u_j(K),\ j \in [0, 2^{16}]$$
$$L_r(K) = \beta_0$$

The third column in Table 1 shows the analysis results of our simulated traces. All samples are determined as true "leaks", except for l_0, which indeed does not depend on the key.

Remark. There is a difference between our overall F-test here and individually testing all 16 key bytes k_i. For instance, l_3 in Table 1 jointly depends on k_6 and k_{10}, but it does not depend on any single key byte.

Degree analysis. We now limit the degree of the regression function to d and test the resulting reduced model:

$$L_f(K_c) = \sum_j \beta_j u_j(K_c),\ j \in [0, 2^{16})$$
$$L_r(K_c) = \sum_j \beta_j u_j(K_c),\ j \in [0, 2^{16}), deg(u_j(K_c)) \leq d$$

If the null hypothesis is rejected, we can conclude the tested trace point contains some leakage that requires $deg(u_j(X)) > d$. In our collapsed model, this implies the leakage jointly depends on more than d key bytes. Although it is possible to test any $d \in [1, 15]$, we argued before that based on knowledge for the encryption scheme, effort can be saved. Thus, within the context of this example, we test $d = 1, 2, 4$: $d = 1$ represents the unprotected value before the MixColumn, $d = 2$ often occurs when transition leakage appears between two individual bytes, while $d = 4$ represents the first MixColumn output as well as the attack enumeration complexity (2^{32}) which we are willing to accept.

The fourth column (labelled "Degree") gives the results on our simulated traces. Both l_1 and l_2 are marked with degree 1. This is because although l_1 does involve 4 key bytes, they do not interact with each other and therefore the attacker can directly recover k_0 and ignore the other 3 (i.e. treat as noise). The trace point l_3 has $d = 2$ (which is correct because it jointly depends on both k_6 and k_{10}). After the first round MixColumn, each state byte depends on 4 key bytes, therefore l_4 shows degree 4. After the second round, the degree of l_5 lies between 5 to 16, where we did not further explore as it is unlikely l_5 can be easily exploited.

Subkey Identification The degree of the key-dependent model already enables to judge if a leakage point can be exploited in practice. We can now use our statistical methodology again to determine which key bytes contribute to the leakage.

We do this by further restricting $L_r(K_c)$ in order to find the minimal set of $u_j(K)$ that is "enough" to explain the leakage. For instance, we can simply delete one key byte k_i from the 16-bit state K_c and construct

$$L_r(K_c') = \sum_j \beta_j u_j(K_c'),\ j \in [0, 2^{15}), K_c' = K_c - \{k_i\}$$

If $L_r(K_c')$ is not rejected, we conclude this key byte is irrelevant for the observed leakage. The second-to-last column (labelled "Key Bytes") shows the results of this analysis. For instance, we identify correctly that the key bytes k_2, k_7, k_8, k_{13} contribute to the degree four leakage of l_4: the other 12 key bytes are rejected by our test, therefore irrelevant for l_4.

Deriving specific attacks. At this point we know the dependency between each trace point and the key bytes. With this information, and a further trace set that has variable inputs, we can now build templates for the identified leaking points. Whilst template attacks are information theoretically optimal, templates that require four or more key bytes are perhaps not achievable for general adversaries. In this case attacks based on standard power models, or simplified templates, could be attempted.

The final column in Table 1 shows the lowest degree model that was identified by Alg. 1. These serve as the basis for confirmatory attacks (if desirable). For instance for l_0 we identify k_0 is a candidate for a specific attack: this means that an attack involving a plaintext byte that interacts with this key byte is highly likely to succeed; for l_3 we identify that the leakage can be explained only by involving two plaintext bytes that interact with an intermediate value that jointly depends on (k_6, k_{10}).

Confirmation must happen in the uncollapsed setting. A suspicious result in Table 1 is that our analysis claims l_4 can be exploited with the joint distribution of (k_2, k_7, k_8). This could be generally possible due to some implementation choice in MixColumns, but our implementation only leaks on the MixColumns output. Similar to the discussion in [12], for some collapsing parameters, the reported leak (in the collapsed setting) can be a degenerated version of the corresponding leak in uncollapsed setting (i.e. requiring only part of the relevant subkey, as some vanish in the collapsing procedure). Consequently we know that the identification of the least leaking term that we do in the collapsed setting has produced a too optimistic outcome, and an actual attack would require four key bytes. One can mitigate this issue with a different set of collapsing parameters (e.g. more bits for each byte). Nonetheless, we recommend to always run a confirmatory attack in an uncollapsed setting, for any specific attack derived from our framework.

5 Leakage Assessment: Example for a Boolean masked AES on a 32-bit processor

We now apply our leakage assessment framework to an implementation of AES that utilises a well known Boolean masking scheme [22]. The original code was written by Ermin Sakic [23]. It was then re-written by the Secure Embedded Systems Research group at Virginia Tech as a more general C-based implementation [24]. The entire AES encryption uses 6 bytes of randomness: a pair of (input/output) masks for masking the S-box input/output, while the other 4 protect the MixColumns as suggested by [22]. The masked S-box is pre-computed before each encryption according to the pair of input/output masks, then all S-box look-up(-s) use exactly the same mask.

We port this implementation on an ARM M3 core (NXP LPC 1313). The target core is running at 12 MHz, so each cycle takes approximately 83 ns. In order to capture the entire round, we adjusted the sampling rate to 12.5 MSa/s, i.e. close to one sample per cycle. The 10 rounds' masked encryption (excluding initialisation and key schedule) takes around 556 μs. As a consequence, our scope (Picoscope 5243D) captures 7000 samples per trace. Our analysis proceeds in the same sequence of steps as detailed in the two previous sections.

5.1 Applying TVLA

The results of the non-specific fixed-versus-random plaintext TVLA is shown in the upper half of Figure 1: nearly 20% of the samples on the traces are identified as "leakage"[3]: considering that our target implementation is a Boolean masking scheme that reuses masks across state bytes and is written completely in C, it is understandable that many leaks may appear [25]. The red asterisk in the following (Figure 1 and 2) marks a target point for our specific attack, which will be discussed in detail later.

5.2 Leakage assessment via key dependent models

Following the same routine as Section 4, we now embark on a leakage assessment using our novel methodology. Because our target implementation is again a byte-wise AES-128, we stick with similar collapsing parameters as Section 4, namely:

- Each key byte collapses to 1 bit.
- Plaintext set to 16 bytes of 0.
- Each key bit represents two values, hexidecimal $0x52$ and $0x7d$, which ensures after the S-box we get $0x00$ and $0xff$.

With this we set up the exact same detection test by building the key dependent (collapsed) model and checking via the F-test if or not explains the data variance (we do this for each trace point). The results (one value for each point)

[3] The red line is the threshold at which we refute the null hypothesis (no leakage).

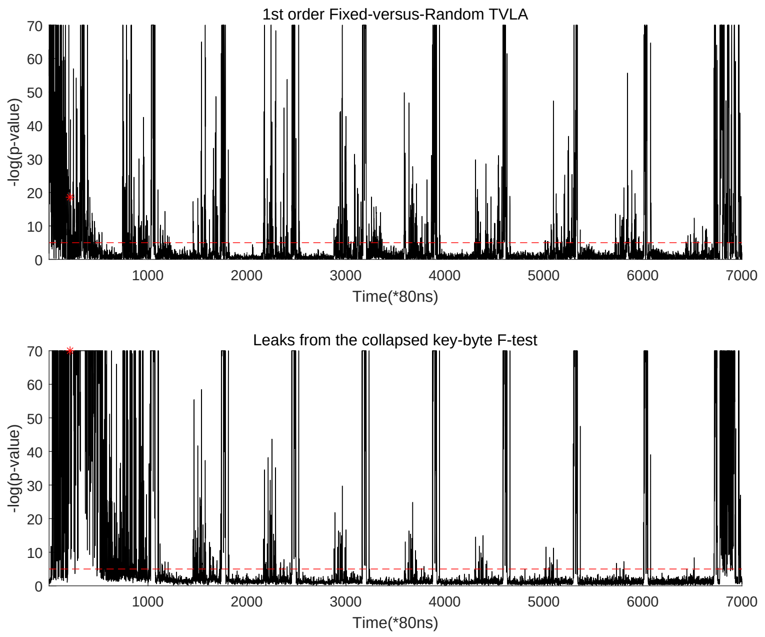

Fig. 1. Top figure: TVLA, Bottom figure: F-test

are plotted as the bottom figure in Figure 1. Like the TVLA test, we find many leaks, but we identify considerably more leaks at the beginning of the execution, i.e. the first round. Even more interestingly, the F-test finds considerably more leaks towards the end of the first round than TVLA does. Thus, although this implementation would be (rightfully) judged as leakage by both tests, TVLA misses a lot of leakage.

Furthermore we notice that the F-test (as well as TVLA) reports fewer leaks in the last rounds. This is likely caused by the strong diffusion of the algorithm in conjunction with having only a limited number of traces: TVLA only needs to sample from two groups whereas the F-test samples from more groups. With a fixed number of traces, it is then clear that TVLA has more traces per group than the F-test statistic[4].

Degree analysis. From the working principle of AES encryption, we know that it is likely to see leakage degrees of one or two prior to the execution of MixColumns, at which point we expect to see leakage models of degree four. We next run the degree analysis by dropping terms from the full (collapsed) model

[4] For better results on the last round, one has to therefore (in both cases) adapt the choice of key (F-test) or plaintext (TVLA) to ensure a good grouping in the final round

in order to determine the degree of the model (and thereby the size of the key guess that would be needed to exploit a leak). We do this analysis for each point of the trace, and produce Fig. 2.

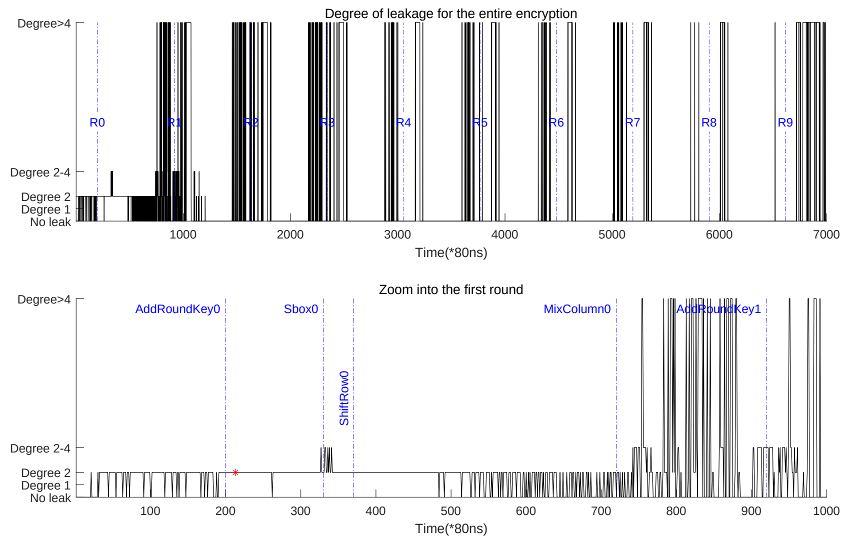

Fig. 2. Analysing the degree of the leakage function in a collapsed model

Our analysis results in Figure 2 are quite consistent with our theoretical justification. The leakage starts with $d = 1$, then expanded to $d \geq 4$ after the first round's *MixColumns*. After the second round, all leakage depends on at least 4 key bytes, which suggests the trivial attack becomes quite time consuming. However, when zooming into the first round (bottom plot of Fig. 2), we see some interesting behaviour:

– Even before the first round S-box, there are some leaks that depend on 2 key bytes: a plausible explanation is those are leaks caused by the register/bus bit-flips, where the random masks are accidentally cancelled out. Although exploiting those sample takes a bit of extra effort (possibly with 16-bit key guesses), their existence in realistic implementations highlights the limitation of following up a non-specific test with a specific test with one plaintext byte as the target state (e.g. [10]). Since no specific single-byte test is able to exploit this leakage, an evaluator would falsely conclude that the flagged trace point is a false positive.
– Some trace points show exploitability in one plaintext byte test (e.g. the ρ-test/NICV, see Appendix A)), yet have $d = 2$ in F-test: this implies that although these points can be exploited with only one key byte, they also contain information that can only be exploited with 2 key bytes.

Subkey identification. In an evaluation setting a device is rejected if a single realistic attack succeeds (the exact definition of realistic depends on the type of evaluation as we explained in our introduction). Thus we now select a single trace point and use our statistical tooling to determine which key bytes explain the leakage in this point, and then derive a concrete, realistic attack.

We choose as our point for deeper analysis the trace point at index 213, which is marked with a red asterisk in our figures. According to Figure 1 and 2: the leakage on this sample can be expressed with a model that includes the interaction of no more than two key bytes. Thus, we perform Algorithm 1 on the to find which byte(-s) is/are relevant for this trace point's leakage.

This analysis proceeds as explained in Section 3.3, which corresponds to the first step (line 1-7) in Algorithm 1. It shows that in total four key bytes contribute to the leakage in this point, which are the key bytes (k_0, k_1, k_2, k_3). This may seem surprising, because prior to MixColumns all AES round functions operate on single bytes. However, we are executing this algorithm on a 32-bit device: any load or store may operate on an entire 32-bit word [26, 12]. Thus, it is expected that an 8-bit S-box look-up returns not just the 8-bit result, but also the adjacent three 8-bit values in the corresponding table. We also know from our previous degree analysis, that only the combination of two out of these four key bytes suffices to explain the observed leakage.

Consequently the steps (line 8-19) in Algorithm 1 can be efficiently computed. Table 2 documents all the combinations of two bytes that statistically contribute to the leakage of trace point 213. As we are testing on single point on the trace (i.e. no multiple correction issue in [7]), we simply set $\alpha = 0.01$ and reject any $-log_{10}(p - value) > 2$. If the $p - value$ exceeds the precision of our computation script using numpy, we simply note these values as > 324 (i.e. the maximal precision for numpy's float64 on our PC).

Key bytes	$-log_{10}(p - value)$
(k_0, k_1)	14.93
(k_0, k_2)	> 324
(k_1, k_2)	> 324
(k_0, k_3)	140.26
(k_1, k_3)	93.12
(k_2, k_3)	> 324

Table 2. Potential target pairs of bytes from Algorithm 1

Converting to a specific attack Among the identified key byte pairs, the pair (k_0, k_2) has the highest significance level. For demonstrating a specific attack for trace point 213, we now take a further set of measurements, with a fixed key and varying inputs, and create Gaussian templates for this trace point.

In this templating step we can trade off trace efficiency for templating and trace efficiency during the actual attack. We know that trace point 213 depends on ($x_0 = k_0 \oplus p_0, x_2 = k_2 \oplus p_2$): thus we build 256^2 templates. Moreover, to reduce the noise, we take advantage our ability to choose plaintexts and set all other plaintext bytes to 0. In practice, if the chosen-plaintext access is not given, the attacker needs more traces in their exploit.

We build templates and then perform 100 repeat attacks in order to derive the average rank of (k_0, k_2), which is documented in Figure 3. Based on some pre-existing knowledge, we further make another educated guess and build (with the same trace set) templates for $S(x_0 = k_0 \oplus p_0) \oplus S(x_2 = k_2 \oplus p_2)$. Figure 3 confirms that trace point 213 reveals plenty of information about (k_0, k_2) (left plot); with some luck or insight an adversary can mount an even more efficient attack (right plot)[5]. We note that both attacks only use a single trace point, thus are univariate. They may appear simple, but finding them would require exhaustive guessing without our novel assessment framework. We re-emphasize our statistical analysis cannot reveal *why* the implementation leaks $S(x_0) \oplus S(x_2)$[6]: one plausible explantation could be the byte-wise interaction within a word (see [26]).

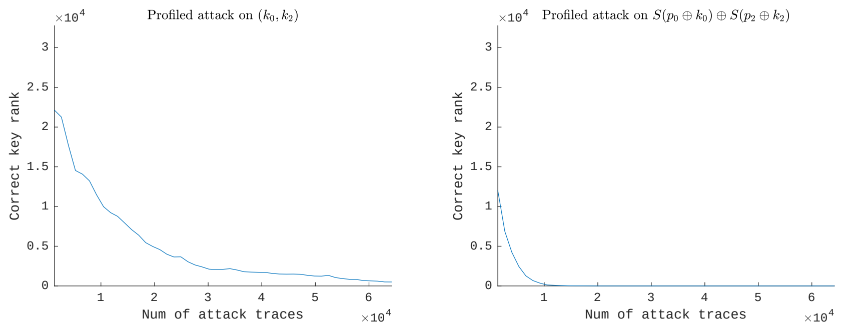

Fig. 3. Specific Attack Vectors

6 Leakage Assessment: Example for an affine masked AES on a 32-bit processor

A limit of TVLA and any univariate, moment-based test statistic is that it cannot capture multivariate leaks. Thus if an implementation is analysed that uses a more complex masking scheme, then traces have to be processed before the statistical test can be used. This was detailed in [13].

[5] The maximum rank here is 2^{16}, the more generic attack on the key pair reaches a rank of around 500 after 65k traces. The more specific attack on the SubBytes output reveals the pair after 10k traces

[6] The byte order in the code does not compute x_2 after x_0, therefore it is unlikely this is due to some bit-flip(-s).

To show that our statistical tooling has enough power to deal with such processed traces, we now analyse the affine masking implementation from ANSSI [27]. We stick with the same ARM Cortex M3 core (NXP LPC1313) as Section 5. The working frequency is still 12 MHz, while our trace set is recorded by a Picoscope 2205A running at 100 MSa/s. Considering the masked encryption takes longer than before, we directly focus on (a part of) the first round Sbox computations, which take about 1 500 samples on each trace. We turn off shuffling in their implementation: shuffling only scales the effort of an attack, but it does not change the nature of the attack strategy. Confirming the analysis of ANSSI, we did not find any leak with TVLA on the raw traces (i.e. no 1st order univariate leak).

6.1 Applying TVLA after trace processing

Our target device features a three-stage pipeline. This implies that three instructions are executed simultaneously (the instruction at time t is fetched, the instruction at time $t-1$ is decoded, and the instruction at time $t-2$ is executed). Consequently there exist leaks from instructions being executed in close succession (so they leak in parallel) and there exist leaks from instructions being executed multiple time points apart (so they leak in sequence).

We first pinpoint exploitable leakage that is due to the parallel processing of instructions. For this purpose we processes the traces by computing $y' = (y - \bar{y})^2$ (y denotes a trace point). As the affine masking scheme does not offer security beyond the first statistical moment, our result in the upper plot of Figure 4 is hardly surprising. Two samples (point 360 and 1428, i.e. the red/blue asterisk) clearly exceed the TVLA threshold and could be exploitable.

We then pinpoint exploitable leakage from the sequential processing of two instructions. Of course any combination of two trace points could be the source of such leakage, but in an evaluation it suffices to demonstrate one realistic attack. Consequently we check if the already identified leakage in point 1428 is also involved in sequential leakage. Thus we fix $j = 1428$ and then process our traces as: $y'_i = (y_i - \bar{y_i})(y_j - \bar{y_j})$. The lower plot in Figure 4 shows that a TVLA test on these processed traces also leads to points that exceed the threshold.

6.2 Parallel leakage assessment

Because the implementation is again byte oriented, we are using the same collapsing parameters as before. Our analysis now focuses on the leakage in time point 360.

We perform the degree analysis for this time point, which indicates that the leakage can be exploit by a combination of two key bytes. We then identify the subkey bytes, and show the outcomes in Table 3. As we are focusing on one single point, we simply set $\alpha = 0.01$ and reject any $-log_{10}(p - value) > 2$.

Converting to an attack. Table 3 suggests trace point 360 can be exploited if we use certain key byte combinations as the basis for a template (or other) attack; the pair (k_0, k_3) seems the most promising target as it exceeds the threshold

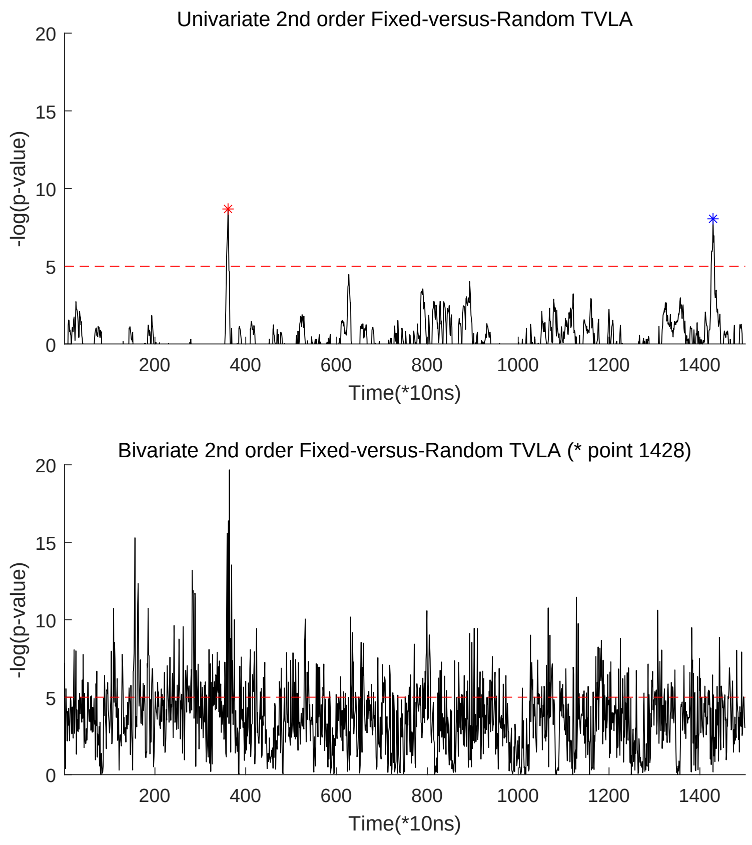

Fig. 4. 2nd order TVLA on affine masking implementation

Table 3. Potential target pairs of bytes from Algorithm 1, parallel leakage of affine masking

Key bytes	$-log_{10}(p - value)$
(k_0, k_1)	20.29
(k_0, k_2)	4.62
(k_1, k_2)	14.41
(k_0, k_3)	31.09
(k_1, k_3)	6.26
(k_2, k_3)	8.59

Table 4. Leaking key pairs from Algorithm 1, sequential leakage of affine masking.

Key bytes	$-log_{10}(p - value)$
(k_0, k_4)	46.23
(k_1, k_4)	6.61
(k_2, k_4)	4.15
(k_3, k_4)	5.18
(k_0, k_5)	10.62
(k_0, k_6)	10.60
(k_0, k_7)	10.52

most. We can now adopt the same strategy as in Section 5.2. However, we notice our result here fits well with the analysis in [12], where the authors stated there is a collision attack on this particular implementation (left plot, Fig. 2 in [12]). From Figure 4 and Table 3, it is likely we have re-produced the same issue from a leakage detection perspective.

6.3 Sequential leakage assessment

We used point 1428 (the blue asterisk in Figure 4) as y_j and identified that together with trace point 364 it leads to the leakage that exceeds the TVLA threshold in the lower plot of Figure 4. We now perform our leakage assessment on the joint leakage of these two trace points (as represented by the resulting point in the lower plot of Figure 4). The degree analysis shows a combination of two key bytes can explain the leakage, and we can identify a subset of key pairs that contribute to the leakage (Table 4). Similar to the parallel case, we set $\alpha = 0.01$ here.

Technically speaking, Table 4 represents the bivariate version of the previous collision attack. However, unlike the previous case, only some pairs can be exploited: namely, the leakage is only significant if the pair include k_0 or k_4. A concrete attack exploiting these key bytes was in fact given in [12] (i.e. Section 4, Figure 2 Bivariate case).

7 Conclusion

Leakage detection without device leakage assumptions or selection of intermediate targets is an extremely powerful concept and the foundation of the sound assessment of crypto implementations. So far, such a "non-specific" assessment has always implied that the detected leaks had to be confirmed by actual attacks, because the assessment process was based on identifying plaintext dependencies.

Our novel approach is set up to identify leakage that depends on the key and it does so by casting the detection problem as a key-dependent model building problem. This enables to detect key-dependent leaks and, through gradual refinement of the built models, to determine the number of key bytes that are required to explain an identified leak, and finally the exact key bytes that leak.

Once we know exactly which key bytes leak (we distinguish between independent and joint leakage) there is only a small step towards establishing precise attack vectors for confirmatory attacks.

Acknowledgements

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

References

1. Common Criteria: The Common Criteria for Information Technology Security Evaluation. https://www.commoncriteriaportal.org/cc/ (2017)
2. EMVCo, LLC: EMVCo Security Evaluation Process. https://www.emvco.com/wp-content/uploads/2017/04/EMVCo-SEWG-14-P02-V5-1_EMVCo_Security_Evaluation_Process_20160725082101992.pdf (2016)
3. ISO/IEC: Testing methods for the mitigation of non-invasive attack classes against cryptographic modules. https://www.iso.org/obp/ui/#iso:std:iso-iec:17825:ed-1:v1:en (2016)
4. SOG-IS: Application of attack potential to smartcards and similar devices (2019)
5. SOG-IS: Attack methods for smartcards and similar devices (2020)
6. 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
7. 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
8. Bronchain, O., Schneider, T., Standaert, F.X.: Multi-Tuple Leakage Detection and the Dependent Signal Issue. IACR Transactions on Cryptographic Hardware and Embedded Systems **2019**(2) (Feb. 2019) 318–345
9. Moradi, A., Richter, B., Schneider, T., Standaert, F.: Leakage Detection with the x2-Test. IACR Trans. Cryptogr. Hardw. Embed. Syst. **2018**(1) (2018) 209–237
10. Durvaux, F., Standaert, F.: From Improved Leakage Detection to the Detection of Points of Interests in Leakage Traces. In Fischlin, M., Coron, J., eds.: Advances in Cryptology - EUROCRYPT 2016 - 35th Annual International Conference on the Theory and Applications of Cryptographic Techniques, Vienna, Austria, May 8-12, 2016, Proceedings, Part I. Volume 9665 of Lecture Notes in Computer Science., Springer (2016) 240–262
11. Standaert, F.: How (Not) to Use Welch's T-Test in Side-Channel Security Evaluations. In: Smart Card Research and Advanced Applications, 17th International Conference, CARDIS 2018, Montpellier, France, November 12-14, 2018, Revised Selected Papers. (2018) 65–79
12. Gao, S., Oswald, E.: A Novel Completeness Test and its Application to Side Channel Attacks and Simulators. IACR Cryptol. ePrint Arch. (2021) https://eprint.iacr.org/2021/756.

13. Schneider, T., Moradi, A.: Leakage assessment methodology - A clear roadmap for side-channel evaluations. In Güneysu, T., Handschuh, H., eds.: Cryptographic Hardware and Embedded Systems - CHES 2015 - 17th International Workshop, Saint-Malo, France, September 13-16, 2015, Proceedings. Volume 9293 of Lecture Notes in Computer Science., Springer (2015) 495–513
14. 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
15. 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
16. Bhasin, S., Danger, J., Guilley, S., Najm, Z.: Side-channel leakage and trace compression using normalized inter-class variance. In Lee, R.B., Shi, W., eds.: HASP 2014, Hardware and Architectural Support for Security and Privacy, Minneapolis, MN, USA, June 15, 2014, ACM (2014) 7:1–7:9
17. 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
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. Chari, S., Jutla, C.S., Rao, J.R., Rohatgi, P.: Towards Sound Approaches to Counteract Power-Analysis Attacks. In Wiener, M., ed.: ”Advances in Cryptology — CRYPTO’ 99”, Berlin, Heidelberg, Springer Berlin Heidelberg (1999) 398–412
20. Mather, L., Oswald, E., Whitnall, C.: Multi-target DPA attacks: Pushing DPA beyond the limits of a desktop computer. In Sarkar, P., Iwata, T., eds.: Advances in Cryptology - ASIACRYPT 2014 - 20th International Conference on the Theory and Application of Cryptology and Information Security, Kaoshiung, Taiwan, R.O.C., December 7-11, 2014. Proceedings, Part I. Volume 8873 of Lecture Notes in Computer Science., Springer (2014) 243–261
21. 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
22. Mangard, S., Oswald, E., Popp, T.: Power Analysis Attacks: Revealing the Secrets of Smart Cards. Springer-Verlag, Berlin, Heidelberg (2007)
23. Sakic, E.: SmartCard Firmware Implementation with AES-128 decryption support. https://github.com/ermin-sakic/smartcard-aes-fw/ (2013)
24. Yao, Y., Yang, M., Patrick, C., Yuce, B., Schaumont, P.: Fault-assisted side-channel analysis of masked implementations. In: 2018 IEEE International Symposium on Hardware Oriented Security and Trust, HOST 2018, Washington, DC, USA, April 30 - May 4, 2018, IEEE Computer Society (2018) 57–64
25. 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
26. Marshall, B., Page, D., Webb, J.: MIRACLE: micro-architectural leakage evaluation A study of micro-architectural power leakage across many devices. IACR Trans. Cryptogr. Hardw. Embed. Syst. **2022**(1) (2022) 175–220

27. 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
28. Bellizia, D., Whitnall, C.: Leakage Detection Tutorial, Part 2: Detection on Real World Traces. http://reassure.eu/leakage_detection_tutorial_part2/ (2018)

A Why not specific tests?

ρ-test As stated in [28], reporting non-exploitable plaintext-dependent leakage is a well-known issue of TVLA. A workaround would be switching to some more specific techniques, for instance, the ρ-test [10]. The ρ-test is basically a profiling Correlation Power Analysis (CPA), where the leakage model is estimated from the trace set (rather than using the Hamming Weight directly in the non-profiled case) [10]. In a k-fold scenario, users should divide the trace set into k subsets, then repeating using one set as the test set and all others as the training sets. The averaged correlation ρ can be converted to a statistical p-value through the Fisher z-transformation. Unlike the TVLA, this test is closely linked to profiling CPA: for any detected leakage, there is at least one profiling CPA that can exploit such leakage. Considering the profiling setup does not restrict the leakage function (i.e. the determined function L), this approach detects all leakage that depends on the target input X.

We stress that "all leakage" and "all leakage that depends on the target input X" are not always the same. In order to compute the model for this test, we must compute the mean for each input x. As demonstrated in [10, 28], the common choice is using one byte plaintext as the input X. If the leakage depends on this byte alone, it can be detected in any form. However, if the leakage explicitly depends on two plaintext byte p_i and p_j, then testing p_i (or p_j) cannot illustrate such leakage. This drawback can be resolved if the model building considers both p_i and p_j as input; however, the required size of trace set grows quickly beyond feasible. Besides, the attacker needs some *a priori* knowledge to purposely build such models: in other words, a specific test degenerates to the situation where the attacker must know his/her target state in advance, while scarifying the ability to detect arbitrary unknown leaks like non-specific detections. Thus, we argue using specific tests to replace non-specific detections is never an option, as those two approaches serve for different purposes.

For clarity, we replot the 1st order TVLA result from Figure 1 in the upper graph of Figure 5. The middle graph shows the 10-fold ρ-test results for all 16 AES plaintext bytes with 50k traces. Only 2 bytes are identified as leakage, while each lasts for only 2 samples on the trace.

NICV Similarly, Bhasin, Danger, Guilley et al. proposed the Normalized Inter-Class Variance (NICV) as an alternative technique to detect point of interest or potential leakage [16]. Technically, NICV represents the R^2 of is the overall F-test, which can be converted to a statistical p-value from the F-distribution. Considering the F-test is strongly related to the coefficient of determination (R^2), which is also defined as the square of the correlation coefficient ρ, it is

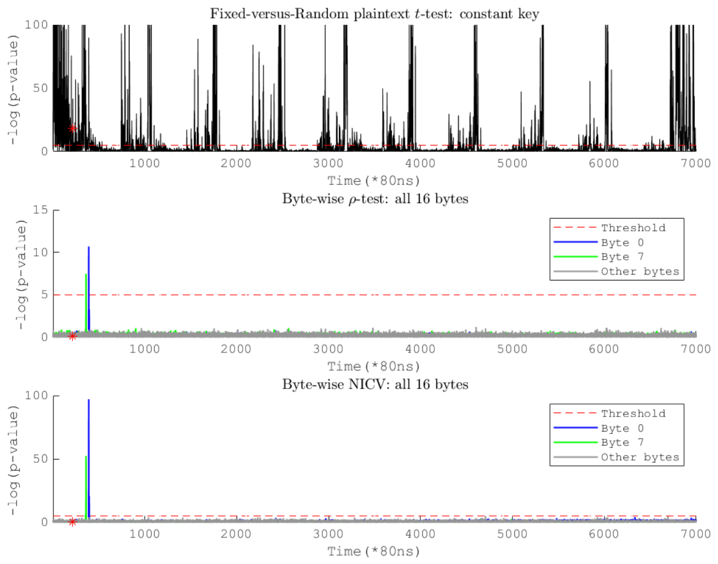

Fig. 5. Non-specific vs. specific: TVLA, ρ-test and NICV

hardly surprising to see that NICV gives exactly the same result as the ρ-test in Figure 5. Meanwhile, our computation of NICV in Figure 5 is also restricted to each byte of the plaintext, not detecting any leakage that is jointly determined by multiple plaintext bytes.