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. Schemes such as Common Criteria or FIPS 140-3 require the assessment of cryptographic implementations with respect to side channels at high security levels. Instead of a "penetration testing" style approach where specific tests are carried out, FIPS 140-3 relies on non-specific "leakage assessment" to identify potential side channel leaks in implementations of symmetric schemes. Leakage assessment, as it is understood today, is based on a simple leakage detection testing regime. Leakage assessment to date, provides no evidence whether or not the potential leakage is exploitable in a concrete attack: if a device fails the test, (and therefore certification under the FIPS 140-3 scheme) it remains unclear why it fails.
We propose a novel assessment regime that is based on a different statistical rational than the existing leakage detection tests. Our statistical approach enables non-specific detection (i.e. we do not require to specify intermediate values) whilst simultaneously generating evidence for designing an attack vector that exploits identified leakage. We do this via an iterative approach, based on building and comparing nested regression models. We also provide, for the first time, concrete definitions for concepts such as key leakage, exploitable leakage and explainable leakage. Finally, we illustrate our novel leakage assessment framework in the context of two open source masked software implementations on a processor that is known to exhibit micro-architectural leakage.

1 Introduction

Security certification schemes such as the Common Criteria (CC) framework [1], or FIPS 140 [2], mandate the evaluation of cryptographic software and hardware against side channel adversaries at high security levels. The philosophical approach in CC schemes follows on from penetration testing: an evaluator is tasked to identify specific side channel vulnerabilities and demonstrate exploits via attacks. Because this approach is based on specific attacks, it comes at a huge cost, especially if the leakage is caused by some hidden micro-architectural effects.

In contrast, the NIST based framework FIPS 140 (now at version 3 [2]) is based on a derivative of the Test Vector Leakage Assessment (TVLA) [3]— via referencing to an associated standard ISO 17825 [4]. TVLA includes a "non-specific" test, and therefore, it can potentially identify arbitrary leaks—a huge

advantage in the case of software implementations where the micro-architecture of the processor is unknown to the evaluator. TVLA utilises the simple statistical Welch's t-test[1] to ascertain if observations exhibit characteristics that depend on either data or key. For the purpose of finding general dependencies, TVLA suggests the use of a fixed-versus-random input test, which is called *non-specific detection*. In a nutshell, two sets of side channel traces are compared (one with a fixed input and one with randomly chosen inputs): if they are "different" then the device is said to have data dependent side channel leakage.

TVLA acknowledges the problem of false positives, in particular first and last round potential "leaks" may not always be exploitable (e.g. plaintext/ciphertext-based leakage). False positives are detection test outcomes that claim that there is leakage, although there is none. The common practice is then using the so-called *specific detection tests* (which are akin to "known key attacks") to confirm or refute the findings from the non-specific test. Importantly, the specific tests do not benefit from the previous non-specific test outcomes. The knowledge of time points is helpful to infer a specific test for leakage on an algorithmic level, but not if the data dependency arises from a micro-architectural component (as it interacts with data from different steps in an implementation). Thus, not much has been gained, and perhaps as a consequence, ISO 17825 does not include the specific testing regime.

1.1 The challenge of interpreting non-specific leakage detection outcomes

FIPS 140-3 uses a non-specific leakage detection *only* in the context of testing *symmetric* cryptographic algorithms. A previous study proposes a hybrid methodology to connect non-specific outcomes with specific tests [5], however, the approach requires *a priori* knowledge of specific intermediate states for setting up the tests. More recently non-specific leakage assessments were developed further by the introduction of a deep learning framework (DL-LA) [6]: the idea is that if a network can be trained to distinguish two sets of traces, then the network could be the "evidence" of leakage. Further sensitivity analysis on the trained network helps to determine the trace points contributing to the leakage and thus enables a similar result as TVLA—determining which (if any) trace points depend on inputs. Hence the same problem persists as before: one has to set up specific tests to confirm the leakage as exploitable in an actual attack.

There seems to be an unavoidable divide between non-specific and specific leakage detection tests: non-specific detection (per construction) is not based on specific intermediate steps in the cryptographic module. The advantage of the non-specific approach is thus also its biggest drawback: because we do not test specific intermediates, it is hard to link test results to specific intermediates and therefore concrete attack vectors.

[1] Notice that because the Welch's t-test only captures the first central moment of a distribution, complex leakage has to be "forced into" this central moment by means of pre-processing the observations.

The *status quo* is thus that there is a clear desire for non-specific leakage detection as part of a leakage assessment framework, but there have been no ideas so far about how a non-specific method could produce "usefully interpretable outcomes". This *status quo* leads to three interesting research questions:

- What are useful ways of "interpreting" the outcome of a non-specific leakage detection test?
- How can we capture the idea that something is "exploitable in an attack" in a leakage detection framework?
- Can we design a leakage assessment procedure that is not only non-specific, but also delivers results that can be "interpreted" as "exploitable leakage"?

1.2 Our contributions: an informal summary

Our research is situated in the context of the security evaluation of a cryptographic module (as part of a formal certification scheme) that considers side channel key-recovery attacks for symmetric cryptographic algorithms. The cryptographic module implements a symmetric cryptographic algorithm via a sequence of steps, which give rise to side channel observations. The evaluation seeks to establish if the observable side channel contains information about the secret key via statistical detection tests. In the context of a certification, the evaluator is provided with access to the device, and can supply inputs, such as plaintexts/ciphertexts and keys.

The aim of our work is to establish characteristics of the terms "explainable outcomes" and "exploitable leakage", and to map them to a concrete statistical decision making process for leakage detection that works in a non-specific manner yet delivers evidence for exploitable leakage.

As part of our contribution, we provide *novel definitions* for *key leakage*, *exploitable key leakage*, and a *explainable detection method*, together with *a concrete statistical test framework*. Our novel framework is based on iteratively building, and comparing, key-dependent nested regression models. The procedure enables us to test for key leakage, then to assess the effort of the inference step (exploitability), and finally to produce evidence for a concrete attack. We implement this framework via two algorithms, thereby representing a concrete instance of an explainable detection method. We demonstrate this framework on two open source masked software implementations (AES and ASCON) that run on a microprocessor that is known to have micro-architectural leakage.

We emphasise that our procedure is not yet another attempt to deal with the "(statistical) false positives problem of TVLA" [7]: instead, our novel framework goes beyond the original intent of TVLA (a quick and easy way to check for potential leakage) and proposes a complete method by which an evaluator can detect *arbitrary key dependencies*, for each detected key-dependent point to *decide if it is exploitable*, and if so, *deliver a concrete attack vector for specific key bytes*. Consequently, our explainable yet non-specific detection framework enables deeper understanding of observable key leakage, it enables distinguishing

non-exploitable data dependency from genuine security concerns, and it enables the better interpretation of the assessment results.

Yap, Benamira and Bhasin et al. proposed a sounds similar but entirely different concept called "interpretable neural network" [8]: their work starts from some *known specific state* (e.g. one Sbox output in AES) and investigates how the network combines parts on trace ("literals") to recover this state. Our work, on the other hand, starts from non-specific detection results and shows how/if they can be exploited in certain specific attacks (i.e. finds which Sbox/combinations of Sbox-es can be attacked).

Summarising, the primary goal of our assessment framework is to explain and confirm whether a detected leak is a genuine security concern (in a non-specific setup) through demonstrating confirmatory attacks. Such attacks may not be optimal, but they can facilitate further improvements via advanced attacks (e.g. deep learning networks).

Outline. After recalling the statistical foundations of the fixed-versus-random procedure and introducing some basic notation, we explain the essential background for our novel leakage assessment framework that is based on nested model building in Section 2. In Section 3 we discuss the novel notions "key leakage","exploitable" and "explainable" and provide concrete definitions. Following on we show the construction of our non-specific detection test in Section 4, and define an assessment framework that has the desired qualities (i.e. outcomes which are both explainable and exploitable) in Section 5. We show the assessment framework in action for an implementation featuring Boolean masking applied to ASCON, and for an implementation featuring affine masking applied to AES in Section 7. We discuss our research in Section 8, whereby we consider relations to existing work, the scope and limitations of our method, and efficiency. We also briefly point out open problems and give ideas for further research.

2 Preliminaries

For the ease of reading, we recall several basic facts about pertinent statistical tools and techniques, as well as the relevant assumptions about side channel observations. We also introduce the notation that we maintain throughout this paper, and refer to relevant related work. We keep explanations informal whenever we refer to relatively well known concepts, and introduce formalism only where it is required.

2.1 Notation

We typeset random variables by capital letters, and their realisations by lower case letters; when we write a random variable in a set notation then we refer to multiple samples of that variable. E.g. X may be a random variable representing an input, $x = 10$ is then a realisation of X, and $\{x_{(1)}, x_{(2)}, \dots\} = \{x\}$ refers

to a set of concrete realisations of X (note that we use subscripts to indicate different realisations $x_{(i)}$).

Side channel observations are often multivariate e.g. $\mathbf{T} = (T^1, T^2, \dots)$ refers to a random variable that represents side channel observation consisting of multiple elements. Note that we use superscripts to refer to the vector elements. Thus a set of concrete measurements is then $\{\mathbf{t}\}$, with $\mathbf{t_{(i)}} = ({t_{(i)}}^1, {t_{(i)}}^2, \dots)$ referring to the i-th multivariate observation.

Estimates of quantities are indicated by a hat over the respective quantity, e.g. E(X) is the expectation of the variable X and $\hat{E}(X)$ is the estimate of that expectation. We reserve the notation $\mathcal{N}(0, \sigma^2)$ to refer to the normal distribution with zero mean and σ^2 variance.

A secret key K is a binary string of a fixed length, which is sampled from a (uniform) distribution. We need to be able to refer to smaller portions or chunks of this key (sometimes also called subkeys in the literature) in our work, so K should also be understood as the concatenation $||$ of such chunks $K = K_1||K_2||\dots||K_s$. We assume that all chunks have the same size, and the size of a chunk will be clear from the context in any concrete example.

2.2 Statistical hypothesis testing

The purpose of a hypothesis test is to decide, based on some gathered data, if there is enough evidence to refute a particular hypothesis, which is typically called the "null hypothesis" H_0. The alternative hypothesis H_1 is considered to be (possibly) true if the null can be refuted.

To carry out a hypothesis test, a suitable test statistic must be chosen. Assuming the distribution of the chosen test statistic is known and well understood, a statistically motivated threshold can be derived that becomes the decision criterion for the null hypothesis. Typically the line of argument is such that if the test statistic surpasses the threshold, we say that we have enough evidence to refute the null hypothesis.

2.3 Side channel observations

We assume that an evaluator can sample side channel observations from a cryptographic module. The module operates on some inputs and secret key, as well as internally generated randomness. In a typical evaluation context, the evaluator may supply chosen inputs and keys, and they may even control internal randomness. Our work does not require any control over the internal randomness.

We denote by $L_D(P, K)^j$ the family of (unknown) leakage functions of the cryptographic module. The function family depends on the device state, which is some unknown function[2] of the input P and the key K. Recall that a cryptographic module implements an algorithm as a sequence of steps: each step j

[2] Note that this "state" is often **not** some known state defined by the cipher's specification. In practice, various micro-architectural effects exist that make exhaustively (specific) testing on all leaking states impossibly difficult.

may give rise to a different device leakage function $L_D(P,K)^j$. Thus side channel observations consist of a set of side channel traces $\{\mathbf{t}\}$, whereby a single trace of this set is a multivariate observation $\mathbf{T} = (L_D(P,K)^1 + \mathcal{N}(0,\sigma^2), L_D(P,K)^2 + \mathcal{N}(0,\sigma^2), \dots)$. Because in our work the statistical method is applied independently to each trace point (i.e. all discussed methods are univariate), we drop the superscript in the rest of this paper.

Remark 1. Side channel observations are noisy: $\mathcal{N}(0,\sigma^2)$ is independent of the device leakage function $L_D(P,K)^j$, and we assume that observations follow a multivariate normal distribution in expectation — this assumption also underpins TVLA, Gaussian templates, linear regression analysis, etc. Despite widely used in the side-channel community, there are a few techniques that do not require this assumption, e.g. χ^2 [9] or DL-LA [6].

Pre-processing The statistical procedures that we refer to are all univariate. By the virtue of countermeasures based on secret sharing, a single step during the computation may not leak directly on any K_s. In this case, trace points must be combined via $[3, 10]$ in an extra step to enable the application of univariate methods; also TVLA requires this extra step.

2.4 Side channel attacks (evaluation context)

Previous work [11] proposes a "Detect-Map-Exploit" framework for evaluations, in which the first step is to identify "leaking trace points" with minimal assumptions. Thereafter one attempts to map these trace points to key-dependent intermediate steps (of the cryptographic algorithm), and with this knowledge the evaluator extracts and processes information (aka performs a side channel attack) from the side channel observations (with or without characterising the observations). Our method neatly fits this proposal.

The last step of the proposed evaluation framework, which is the information extraction, is based on making a statistical inference. The inference is based on relating model-based predictions and side channel observations, and a large number of strategies for this step can be found in the academic literature. We mention two common types of models. If a "standard power model" is known to be suitable for a device (e.g. the Hamming weight of a value often is a good predictor for side channel observations stemming from bus transfers in a microcontroller), then an effective method is to estimate the Pearson correlation between a set of side channel observations, and a set of (key-dependent) Hamming weight predictions. Normalised correlation estimates are then interpreted as an *a posteriori* distribution for the key (chunk). If a "standard power model" is not suitable, then an evaluator must characterise the statistical distribution of the side channel observations in order to build *a model*.

Model based attacks (template attacks). Using a model L as a classifier for the side channel observations $\mathbf{t}$ enables to construct so called template attacks [12].

1. The **non-specific way of model building** requires the least information about the cryptographic module. It works by estimating models L for tuples of chunks of the input and key: e.g. if both the input (P) and key (K) can be divided into 2 chunks((P_1, P_2) and (K_1, K_2)), the overall model can always be written as $L(P_1, K_1, P_2, K_2)$, where the estimating complexity (i.e. the number of all possible models) would be $|P_1| \cdot |P_2| \cdot |K_1| \cdot |K_2|$.
2. The **specific way of model building** is to include some *a priori* knowledge about the cryptographic algorithm and estimate models for an intermediate X defined by a function F, e.g. if $X = F(P_1, P_2, K_1, K_2)$, then we build models $L(X)$, resulting in $|im(F)|$ models. Most typical attacks, e.g. attacking the AES Sbox output, follow this strategy. Note that leakage can only be captured if it truly relies on F: otherwise it will be simply discarded, even in a trivial template attack [13].
3. The **specific way of estimating model coefficients** is to even include some knowledge about the device leakage model, e.g. restricting the degree of the regression model (or the number of classes). This further reduces the number of distinct models, which leads to more efficient model building.

With a set of models the evaluator then computes the conditional probabilities $\Pr[\{\mathbf{K_i}\}|\mathbf{t}]$ (by ensuring that the cryptographic module operates on the same inputs as the learned models, and by exhaustively guessing K_i in the learned models)[3], thereby assigning each key value an *a posteriori* probability. There are many ways of estimating models, ranging from parametric Gaussian models [12], over machine learning models [15], to deep nets [16].

2.5 Regression modelling

Let X be an n-bit discrete variable. In the following X may represent an intermediate step in the computation, or the encryption input and output, or the key. Given a set of side channel observations, we can estimate the distribution of any trace point given X. This is because any real valued function of X can be written as $L(X) = \sum_j \beta_j u_j(X)$. In this expression, the variables $u_j(X)$ are called the *explanatory variables*. They are monomials of the form $\prod_{i=0}^{n-1} x_i^{j_i}$, whereby x_i denotes the i-th bit of x and j_i denotes the i-th bit of j. The monomial with the most number of terms defines the *degree* of the model. Evaluating the regression equation at $X = x$ gives the model prediction for x. The goal of leakage modelling is to determine a function L that is "close" to the (unknown) device leakage L_D.

Model estimation Estimating a regression model requires estimating the (linear) coefficients $\vec{\beta}$. The estimated model that describes the distribution of the side channel observations in dependence of X is now expressed as $\hat{L}(X) = \sum_j \hat{\beta}_j u_j(X)$ (and we evaluate it by setting $X = x$). Evidently, without any

[3] A recent and comprehensive analysis of classifiers of this nature is given in [14].

restriction on u_j, there are exactly 2^n parameters to estimate in the regression equation. A model that includes all 2^n terms is called a *full model*. However, we could also only allow certain u_j (e.g. linear terms where $HW(j) = 1$ [17, 18]). This leads to the question of how to compare models in terms of their quality?

Nested models. A special case for model comparison, that is of particular interest to us, is the case of models that are *nested*. A *restricted* model is said to be *nested* within a *full* model if it only contains a subset of the explanatory variables of the full model.

Toy example. Suppose a target state X contains 2 bits $X = (X_0||X_1)$. Then the full model is given by $L_f(X) = \beta_0 + \beta_1 X_0 + \beta_2 X_1 + \beta_3 X_0 X_1$ (we note that the degree of $L_f(X)$ is two). We can define a restricted model (of degree one) $L_r(X \setminus \{X_1\}) = \beta_0 + \beta_1 X_0$ (where β_2 and β_3 are restricted to be 0). Then L_f and L_r are called nested models (i.e. L_r is "nested inside" L_f).

Collapsing variables. In many real world regression problems, variables are drawn from large sample spaces, which makes the model building computationally infeasible. The concept of collapsibility, see [19], is about restricting the included explanatory variables to a smaller space, without negatively impacting on the model building. The concept of collapsing was recently used in the side channel setting as well, see [13], and we will use this trick also in our work.

The function $Coll(X) : 2^m \to 2^n$ restricts an m-bit variable X to an n-bit space. For instance, if $X \in \{0, \ldots, 255\}$ then a collapse function that maps it to a single bit can be defined as $X \to 0$: if $X_0 = 0$, and $X \to 255$: if $X_0 = 1$. Thus the corresponding collapsed model only needs to fit two coefficients β_0 and β_1. Obviously a model that is based on collapsed input variables has a poorer predictive quality than the same model over the full input range. But in our work, we do not use models as predictors but as decision making tool about input dependency. Thus, as long as a collapse mapping preserves some input dependence, our method works.

Comparing nested models Given two nested estimated models, the F-test is the suitable solution for comparison [20]. 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. 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

$$RSS = \sum_{i=1}^{N} (t_{(i)} - \hat{L}(x_{(i)}))^2$$

where $t_{(i)}/x_{(i)}$ represents the i-th observation/state.

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. Throughout this paper, we always convert the F-statistic to the corresponding p-value, then rejects the null hypothesis if the p-value is lower than the significance level α (i.e. the false positive rate). Equivalently, one can also compute a threshold for F from the null distribution and α (e.g. the 4.5 threshold in TVLA [3]), although such threshold varies alongside the degrees of freedom (i.e. the number of traces N).

Remark 2. In our work we will be often building models for collapsed input variables. An inappropriate collapse mapping can be recognised when a full model is compared with restricted models, because the individual test on the full model already fails to find any dependency. If input dependency has been reported by other techniques (e.g. TVLA), users should change the collapsing setup.

Statistical power of nested F-test The confidence in test outcomes is reflected in the percentage of false positives α (i.e. how often does the test indicate leakage although there is none) and the percentage of false negatives β_p[4] (i.e. how often does the test indicate no leakage although there is leakage).

The parameters in any statistical test interact with each other, thus the α, the $1 - \beta_p$, 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 statistic is well understood, and there are explicit formulae for the α and β_p [20]. With these it is possible to set the decision threshold F_t for rejecting the null hypothesis:

$$F_t = Q_F(df_1, df_2, 1 - \alpha)\,,$$

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 contains only one parameter, these two degrees of freedom are defined as:

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

The statistical power $1 - \beta_p$ can be computed as

$$\beta_p = F_{nc}(F_t, df_1, df_2, \lambda),$$

[4] Unfortunately the same β has been used in both statistics and linear regression models; for clarity, we denote β_p as the false positive rate.

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

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

3 Characterising Exploitability and Explainability in the Context of Leakage Detection

We wish to clarify the desirable goals of a leakage assessment framework, by formalising notions such as "leakage", "exploitable", and "explainable".

3.1 Defining Leakage

The term "leakage" is often overloaded in the side channel literature. Having "leakage" can refer to having access to side channel observations, or it can refer to the assertion that some observation contains information about a secret value. We now provide a definition of the term "leakage" that we will use in this paper.

In the context of side channel evaluations we wish to determine if a cryptographic module's observable side channel (power, timing, EM) contains information about the secret cryptographic key, potentially enabling a key recovery attack. We consequently understand the term "leakage" as the presence of some key dependent behaviour in side channel observations. Because we only wish to consider "efficient adversaries" in concrete security evaluations, we say there is "key leakage" if we can efficiently[5] detect key dependency in a (finite) set of side channel observations.

Definition 1 (Key Leakage).
We assume that the evaluator has access to a finite set of side channel observations $\{\mathbf{t}\}$*, which is obtained from a cryptographic module that performs some action with a secret value* K*. We set up the two hypotheses:*

H_0*: there is no dependency between* $\{\mathbf{t}\}$ *and* K*.*

H_1*: there is some dependency between* $\{\mathbf{t}\}$ *and* K*.*

We say that there is ***key leakage*** *if there exists an efficient statistical test that can refute* H_0 *given* $\{\mathbf{t}\}$*.*

The existence of key leakage does not yet imply that it is exploitable in a concrete attack: e.g. the key dependence could be only observable between very few keys (i.e. leakage-wise weak keys); or equivalently, the adversary has to make a too large key guess in the inference step.

Remark 3. "Key" represents the most typical side-channel key recoveries against block ciphers: for other cryptographic primitives or operation modes, depending on the context, one can expand this definition to other secret-related states.

[5] In practice, we expect "efficiency" to be determined by evaluation parameters, i.e. an given time/data budget.

3.2 Defining exploitable key leakage

To utilise key dependency in a side channel key recovery attack, an efficient attack vector must exist. We explained before in Sect. 2 that an attack is based on making a statistical inference about the key distribution based on a set of side channel observations.

There are three factors that determine the computational complexity of an attack:

Trace complexity: In current evaluations, millions of measurements are required for hardware implementations (short traces), and tens of thousands of measurements must be obtained from software implementations.

Inference complexity: Often, the size of a key guess for a single inference is typically 8 bits, but attacks simultaneously guessing 32 bits have been reported [21].

Enumeration complexity: a side channel attack may not necessarily reveal the secret key, but it will deliver a (non-uniform) distribution for the key space, which enables to reveal the key given a plaintext-ciphertext pair. The best such effort reported in the open literature has managed to search through a key space of just under 2^{50} keys [22].

The ability to carry out the inference is **a necessary condition** for all side channel attacks. The inference step's complexity is determined by the trace and the inference complexity.

Thus, we define the property of being "exploitable" (in reference to key leakage) as the required effort to carry out the inference step.

Definition 2 (Exploitable key leakage). *Given a set of side channel observations* $\{\mathbf{t}\}$ *from a cryptographic module that operates on* K*, we call key leakage to be* **b-exploitable** *if the complexity of the inference step in a concrete attack is bounded by* $O(2^b)$*.*

Remark 4. This definition for exploitable leakage implies that in the context of a specific evaluation regime the term "concrete attacks" has to be specified. For instance, in the practical examples that we will present later in this paper, we will use our method as a "replacement" for the current detection regime in ISO17825, which *only considers differential input attacks (e.g. typical DPA) at present.* Consequently, we will define **b-exploitable** in relation to DPA style attacks. Restricting attacks to DPA style attacks is perhaps a limitation of the current version of this standard, and our methodology works with it. Attacks that go beyond DPA style attacks include horizontal attacks [23], which always require knowledge of specific intermediate values. Depending on the application context, the most suitable list of specific intermediate values can be hard to find, due to restrictions on profiling (i.e. no access to random shares), hidden micro-architectural effects (i.e. software implementations) or complex hardware behaviour (e.g. all gate-level interaction within an Sbox). Thus discovering how to exploit a detected leak is still necessary, and once this has been achieved, it can potentially be incorporated into a more powerful attack vector.

3.3 Defining explainable key-leakage detection

The ultimate goal of an evaluator is to construct at least one practical attack vector. Thus we require a non-specific method that also delivers outcomes that can be "interpreted by the evaluator" so that they can derive a practical attack vector. This requirement links our research with the wider context of interpreting the outcomes of machine learning models, and a widely cited recent contribution by [24] discusses and analyses notions for "interpretable" methods such as

understandable: it is clear "what" the method achieves,
transparent: it is clear "how" the method works mathematically,
comprehensible: the output of the method can be understood by a human,
explainable: the method produces some form of evidence that enables a causal explanation for the subsequent decision making.

All notions but the last are clearly fulfilled when working with regression models assuming that evaluators have sound mathematical and computer science skills. The last notion of "explainability" requires more consideration. It challenges us to consider what is "evidence" and what is a "causal explanation". In our use case, the final goal is to produce outcomes/evidence that can be directly used to construct at least one key-recovery attack vector. If such an attack vector exists, then a device should fail an evaluation: the attack vector is the causal explanation that enables an evaluator to fail a device in an evaluation. We explained in Sect. 2.4 the concept of model-based attacks: the models are estimated based on (leaking) variables (depending on both key and plaintext bytes) in certain trace points.

Definition 3 (Explainable detection). *We call a leakage detection method* **explainable** *if it produces a list of leaking variables as evidence, alongside the associated trace points that exhibit exploitable key-leakage, both of which enable the construction of at least one concrete key-recovery attack vector.*

4 Detecting Key-Dependency via Non-Specific Models

We are now ready to detail the mathematical basis for the novel approach that underpins our proposal for explainable leakage assessment. The main question is how should a non-specific detection test be configured to ensure that the found dependencies are also explainable (and therefore exploitable) without relying on some *a priori* specific intermediate states? The perhaps initially surprising answer, to this question, is to build and compare key-dependent nested models.

4.1 Detecting key leakage

Starting from our definition in Section 2, any observed leakage in leakage assessment is a function of the inputs P and the key K. To create key dependent models, we now fix the plaintext to be a constant, and thus consider a regression model for $L(K)$(from now on we drop P as P is a constant).

Definition 4 (Key-dependent non-specific model). *Let K be an n-bit (full) secret key, we call the function*

$$L(K) = \sum_j \beta_j u_j(K),\ j \in J$$

a key-dependent non-specific leakage model for K.

If $|J| = |K| = 2^n$ (all terms of K are included, we have a degree 2^n model) then we call the corresponding model the full model, and denote this with the subscript f, i.e. $L_f(K)$ (we introduced these concepts already in Section. 2). As the leakage function L in Section 3 is deterministic, with a fixed plaintext, one can easily verify that $L_f(K)$ expresses any possible key-dependent leakage. The naive model $L_0(K)$ is the model that only depends on the term β_0. The naive model is independent of the key. Leakage detection can then be framed as a hypothesis test that compares a full (i.e. "key leakage") and a restricted model ("no key leakage") using the F-test statistic (introduced in Section. 2). If the naive model can explain the side channel observations as well as the full model, then clearly, because the naive model is completely independent of the key, there can be no information about the key in the side channel observations.

It is apparent that contemporary key sizes make building such a full model computationally infeasible, and thus we use the idea to "collapse" a full model to a smaller space whilst retaining some "salient" characteristics [13]. This is done by defining a suitable function *Coll* that maps every $j \in J$ to a $j_c \in J_c$, whereby $|J_c| << |J|$, $Coll(j) = j_c \in J_c$.

This leaves the question if working with a collapsed model is meaningful in the context of testing nested models. This question was already answered positively in [13]: if in a hypothesis test the null is rejected for a collapsed model, then it would also be rejected for the corresponding full model. Consequently we define a non-specific leakage detection test as follows.

Definition 5 (Model based Detection). *Let $L_{cf}(K) = \sum_j \beta_j u_j(K)$ denote the collapsed full model, and $L_0(K) = \beta_0$ denote the naive model (with $K \in [0, 2^n), j \in coll(J), \beta_j \in \mathbb{R}$ as before), and select a statistically motivated threshold th, together with a confidence level $0 < \alpha < 1 \in \mathbb{R}$, a statistical power $0 < 1 - \beta_p < 1 \in \mathbb{R}$, an assumed effect size $f^2 \in \mathbb{R}$ and a finite set of $N \in \mathbb{N}$ side channel observations.*

The null hypothesis is defined to correspond to "no leakage", in the sense that $L_0(K)$ explains the side channel observations. The alternative hypothesis corresponds to "leakage" in the sense that $L_{cf}(K)$ is required to explain the side channel observations.

$H_0 : L_0(K) = L_{cf}(K)$, the observations can be explained by the naive model
$H_1 : L_0(K) \neq L_{cf}(K)$, the observations cannot be explained by the naive model

We reject H_0 if the F-test statistic is higher than the threshold th with confidence α, power $1 - \beta_p$ given the set of N side channel observations.

Definition 5 is the basis for our novel leakage assessment framework. It enables the detection of key dependencies in a set of side channel observations. We will refine this technique in the next section so that it satisfies the notions of exhibiting exploitable leakage, and we will embed it in an algorithm that identifies the leaking key chunks (i.e. an explainable method). Before this, we discuss important details pertaining the choice of statistical parameters and the collapsing function.

4.2 Concrete parameter selection in an evaluation setting

In the setting that is described in FIPS 140-3 (by reference to ISO17825), the security level defines the number of side channel observations that are available for leakage detection testing, e.g. $N = 10,000$ for FIPS Level 3 or $N = 100,000$ for FIPS Level 4. More traces, e.g. $N = 1,000,000$ for a higher level of security, would be aimed for in CC [1]. The effect size is typically unknown (unless some prior characterisation has taken place, which is not how the test is used in FIPS 140-3), and therefore an assumption must be made. Furthermore, when examining an implementation with a countermeasure such as masking, the traces must be processed prior to any statistical evaluation, by the so-called centred product combining technique [25] (multivariate case) or centred moment technique [10] (univariate case). This additional step does not change our discussion of the selection of the statistical parameters, it however reduces the expected effect size f^2. It is therefore very important to assume f^2 to be potentially small to capture many real world leaks.

Example 1 (Parameters for a byte-oriented implementation with 128 bit key length.). Assume that we set up a detection test for a key $K = |2^{128}| = K_1||K_2|\ldots||K_{16}$. We set up the collapse function so that we represent each byte of the key with a single bit, thus the collapsed key space is of size 2^{16}. In our test 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 $f^2 = 0.2$ [20] with $N = 655360$, $\alpha = 10^{-6}$, we have $1 - \beta_p \approx 1$, which implies that test with this configuration is very powerful [7].

5 A Novel Leakage Assessment Framework

The novel model-based leakage detection test that we formalised in Def. 5 is able to detect key leakage (as defined in Def. 1). In this section, we explain how the repeated use of model-based detection enables to determine if identified key leakage is exploitable in differential attacks, and we provide an algorithm for explainable detection by a structured manipulation of the model configuration. For easier comprehension, we provide toy examples after each step: these examples come from an eye-to-eye adoption of our proposal in a Boolean masked AES implementation. Interested readers can find more details in Appendix B.

5.1 Detecting exploitable leakage

Model-based leakage detection reveals key dependent trace points by comparing a full model (i.e. a model that incorporates the entire key) and a naive model. It is likely though that an identified trace point does not depend on the entire key, but just a part of it. We can change the restricted model in Def. 5 from the naive model to some richer (but still restricted) model to determine how much key information is required to detect a trace point as leaky. More specifically, we can parametrise the restricted models to include terms of degree at most d. Recall that the degree d is the number of terms in the largest monomial of the model, whereby the terms represent the key chunks. Thus, the degree d gives an upper bound for relevant key chunks in the inference step of a differential attack.

We configure the test defined in Def. 5 using L_{cf} and a $L_{cr,d}$ by fixing a d (the test is repeated multiple times for different values of d). If a degree-d restricted model $L_{cr,d}$ is "as good as" the full model L_{cf}, then the side channel observations can potentially be exploited in a divide-and-conquer attack using no more than $|K_i|^d$ bits of the key. With this we immediately have the following corollary.

Corollary 1. *Let H_0 represents the hypothesis where the restricted degree-d model $L_{cr,d}(K)$ can explain the leakage, whereas H_1 stands for the leakage can only be fully explained by the full model $L_{cf}(K)$. With a suitable collapse function $coll(J)$ and*

$$L_{cr,d}(K) = \sum_j \beta_j u_j(K);\ j \in coll(J), deg(u_j(K) \leq d),$$
$$L_{cf}(K) = \sum_j \beta_j u_j(K);\ j \in coll(J)).$$

We can perform a model based detection per Def. 5. If H_0 is not rejected, and $2d \log |K_0| \leq b$, then the identified key leakage is b-exploitable.

Proof. If the restricted model of degree d is not rejected, then we know that a combination of at most d key chunks suffices to describe the leakage. Recall that we assume that all chunks of the input and keys are of equal size, thus $|K_0| = |K_1| = |P_0| = \ldots$. We can set up a non-specific model-based attack: we can build models for tuples of the form $((P_0, K_0), (P_1, K_1), \ldots, (P_{d-1}, K_{d-1}))$ for the inference step. The computational cost for building these tuples is $|K_0|^{2d}$. The computational cost for using the tuples in the inference step is $|K_0|^d$. Thus the overall computational cost is bounded by $O(|K_0|^{2d}) = O(2^{2d \log |K_0|})$. If $2d \log |K_0| \leq b$, then the identified leakage is b-exploitable.

A repeated application of Corollary 1 leads to Algorithm 1, which finds the model of the lowest degree that is not rejected. It does so by working down from a given maximum degree d_m (determined by the evaluator's time/data budget), and checking for each lower degree if or not the respective restricted model is rejected. As soon as it finds a restricted model that is rejected, it returns the degree of the previous model.

Algorithm 1 Exploitable leakage detection: identify leakage degree

Require: $K = K_0||K_1||\dots||K_{s-1}$, coll(J), $\{\mathbf{t}\}$, d_m

1: Test $L_{cf}(K)$ vs. $L_0(K)$
2: **if** Not Reject **then** ▷ Check if key dependent
3: **return** "not a leak"
4: **else** ▷ $L_{cf}(K)$ represents key leakage
5: **for** $i = d_m$ to 1 **do**
6: Test $L_{\{cf,i\}}(K)$ vs. $L_{cf}(K)$
7: **if** Reject **then return** $d = i + 1$ ▷ Return degree d
8: **end if**
9: **end for**
10: **return** $d = 1$
11: **end if**

Example 2. Assume that a cryptographic module implements a byte-wise implementation of AES-128. The evaluator "collapses" each key byte to 1 bit. As $d_m \leq s = 16$, $H_0 : L_{cf}(K)$ contains 2^{16} terms. If $H_1 : L_{cr,2}(K)$ is not rejected, then at most two bytes of the key suffice to exploit the leakage. Therefore the complexity of a divide-and-conquer attack is upper-bounded by $O(2^{4\times 8})$.

5.2 An explainable detection method

We now explain an algorithm that determines *which* key chunks are contributing to the detected leakage. We do this by narrowing down our model via placing further constraints on the restricted model. More precisely, we simultaneously restrict the degree and exclude specific terms. Algorithm 2 provides a generic description for identifying specific exploitable key chunks. It consists of two steps.

The first step is carried out in the first for loop, which removes a single chunk K_i from the key, and then checks if the corresponding restricted model is rejected. If not, then the key chunk is not important for the side channel observations, thus removed from the key. As a result of this first step, we now have a new key set K', which only consists of key chunks that are known to be relevant[6].

The second step is carried out in the while loop and it extracts each individual term from the remaining terms of K that has degree smaller than d (which is provided by Alg. 1) and it checks whether it individually explains the side channel observations better than the naive model. If so, then an attack can be constructed based on this individual term, and the term is returned.

Example 3. Following our previous example, with the side-channel observations $\mathbf{t}$ and $d = 2$, we run the first loop (lines 1-7) in Algorithm 2. Starting from the first key byte K_0, line 3 tests if the restricted model with $K' = K_1||...||K_{15}$ (2^{15}

[6] Recall that we apply this process independently to all trace points. Consequently, we expect that different trace points will lead to different models because different trace points correspond to different intermediate steps.

Algorithm 2 Subkey identification

Require: $K = K_0||K_1||\dots||K_{s-1}$, coll(J), $\{\mathbf{t}\}$, d
Ensure: The jointly leaking group represented by j (i.e. $\{K_i|j_i = 1\}$)
1: **for** $i = 0$ to $s-1$ **do** ▷ Check for not contributing subkeys
2: $K' = K - \{K_i\}$
3: Test $L(K')$ vs. $L(K)$
4: **if** Not Reject **then**
5: $K = K'$
6: **end if**
7: **end for**
8: $l = |K|$
9: $J = [0, 1, ..., 2^l - 1]$, $J' = \emptyset$
10: Sort J with the ascending order of the Hamming Weight of $j \in J$
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: Test $L(u_j(K))$ vs. L_0
15: **if** Not Reject **then**
16: Delete j from J
17: **else**
18: Add j to J' ▷ $u_j(K)$ leaks
19: **end if**
20: **end while**
21: **return** J'

terms) can pass the F-test comparing with the full model $L_f(K)$ (2^{16} terms). If so, K_0 can be discarded, as it does not contribute to the observed leakage. At the end of the loop, we have deleted all K_4 to K_{15}, which leaves $l = 4$ in line 8.

In theory, the observed leakage can be expressed by a function of the left 16 terms. However, since we already know the leakage only contains interaction from at most 2 key bytes, the relevant terms now are (sorted from 1 key bytes to 2 bytes, i.e. the Hamming weight of j from 1 to 2):

$$\{K_0, K_1, K_2, K_3, K_0K_1, K_0K_2, K_0K_3, K_1K_2, K_1K_3, K_2K_3\}.$$

Lines 12-18 search through this list and reports the terms that leak.

5.3 A framework for detection

Definition 5, Corollary 1, and Alg. 1 and 2 are all based on the same conceptual idea of testing/comparing key-dependent models, but with increasing levels of sophistication and effort.

Definition 5 enables to test for "any" key dependency. As such it can be used as a replacement of the detection method in TVLA/ISO 17825. The computational effort for the model-based detection is comparable with that of TVLA/ISO 17825. But the application of Corollary 1 within Alg. 1 and 2 leads to a framework that is *qualitatively* different to what any existing non-specific detection

method can achieve. We are in a position to argue which of the potential leakage points are b-exploitable, and how they can be exploited.

Thus when executed in sequence, we have a framework that enables to efficiently gather increasingly sophisticated evidence regarding the behaviour of a cryptographic module: all together our methods provide a framework for the full leakage assessment of a cryptographic module. In the next sections we apply this framework to two practical examples of (open source) masked implementations.

6 Application: A masked 32-bit ASCON implementation

Dietrich, Dobraunig and Mendel et al. provide a masked ASCON software implementation that "can be used as a starting point to generate device specific C/ASM implementations"[7]. Although the target platform is an ARM Cortex-M4 core (STM32F4) on ChipWhisperer CW308, the used assembly instructions are also supported by our platform [8]. We work on an ARM M3 core (NXP LPC 1313) which is running at 12 MHz. The trace set is recorded by a Picoscope 2205A running at 25 MSa/s, contains 8000 samples that covering the initial permutation (i.e. $p^a(IV||K||N)$) The authors provide a two share and a three share implementation, and they remark that for the two share implementation, extra "clearing" instructions (*MOV rd, #0*) are necessary to remove leaks that are found by TVLA.

6.1 Leakage detection, and why to dig deep

We ported their implementation to our device, and run TVLA (with a fixed key, and fixed as well as varying plaintexts and nonce values) on their implementation without using clearing instructions. Figure 1 (left panel) shows that without the clearing instruction, TVLA reports leaks at the beginning of the encryption process (which is consistent with the authors' experiment).

Recall that TVLA in its' fixed-versus-random input (= varying plaintext and nonce, fixed key) setting is prone to finding plaintext-only leaks at the start of encryption processes. In the case of ASCON, the encryption starts by applying the p^a permutation to the concatenation of a (fixed) IV value, the secret key K, and the nonce N: $IV||K||N$. Thus, purely based on the TVLA outcome, and knowing that TVLA picks up also leakage that does not depend on the key, it would be tempting to conclude that the identified leak is an artefact that relates

[7] https://github.com/ascon/simpleserial-ascon

[8] The authors provided various compile macros within their masked ASCON software implementation. One important option is whether the "bit-interleave" trick is applied (aka "ASCON_EXTERN_BI"). Our experiments in this section is captured when "ASCON_EXTERN_BI" is set. Note that this is not the default version: our experiments on the default version shows the same leakage can be found in a latter time point, yet related to a few different key bits (caused by the bit-interleave trick). More details can be found in Appendix E.

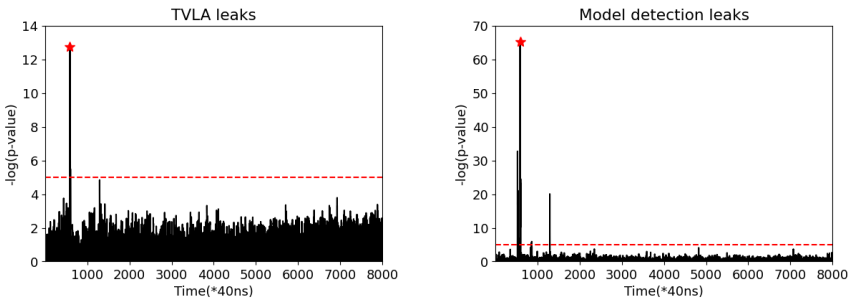

Fig. 1. TVLA vs. Model-based detection

just the leakage of the nonce N (indeed strictly following the TVLA document, the initial 1/3 points could be disregarded [4, 3]).

We now configure our model-based detection: ASCON is based on 64-bit words, and the implementation is based on 128-bit plaintext and key. Thus we choose to collapse the two 64-bit key words into single bits, and we fix all other inputs to some constant. As a consequence, our model-based key-leakage detection achieves high statistical power with only 20k traces. Figure 1 (right panel) shows our detection finds the same leak as TVLA (the red aster), while reporting a few extra leaks at beginning of the encryption.

The outcome of the model-based detection implies that the TVLA results are not due to N alone. We can now be certain that there is key leakage. This result creates the non-trivial challenge of finding an attack vector, which we can do elegantly leveraging our novel framework.

6.2 Assessing key leakage: degree analyses

We focus on the highest leaking point (indicated in red) in Figure 1. Following our framework, the next step is to analyse the degree of the leakage in this point. Because there are two 64-bit key words in our collapsed model, the degree of the largest monomial in our initial model is two. Figure 2 shows the degree for all trace points. The degree of the most leaky point is in fact one, which implies that this point should lead to an attack where only one of the key words is involved.

We next use Algorithm 2 to identify which 64-bit key word contributes to the leakage in the selected point. Within each experiment, we set $\alpha = 0.01$ and reject any $-log_{10}(p - value) > 2$ (i.e. observed key dependency). As we can see in Table 1, the leakage depends on only the first collapsed key word K_0, i.e. the first 64 bit within the secret key. This key word becomes the basis for our subsequent analysis.

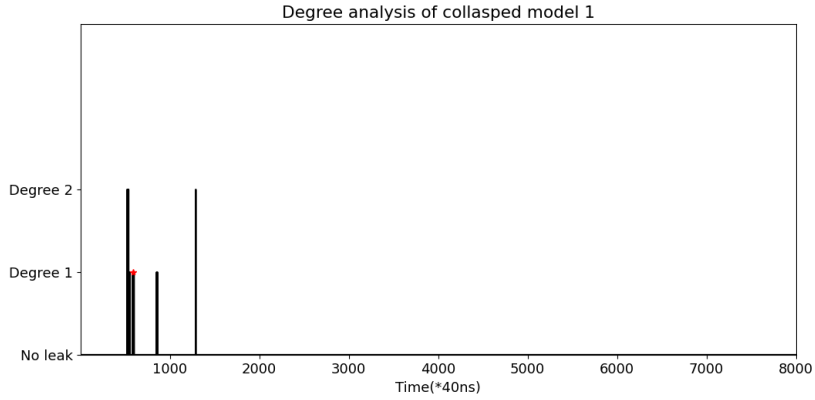

Fig. 2. Degree analysis with Collapsed model 1 (aka output from Alg. 1)

Table 1. Leaking key words from Algorithm 2 (masked Ascon, $d = 1$)

Key word	$-log_{10}(\text{p} - \text{value})$
K_0	66.62
K_1	1.67

6.3 Fine-grained analysis

Although our analysis above finds useful information, attacking a 64-bit word remains challenging in most realistic contexts. In the following, we iterate our analysis in Section 5 with various collapsing setups, in order to gain more fine-grained understanding of the detected leakage.

Identifying the leakage source within the first 64-bit word Within the first 64-bit word, we further refine our analysis with a more fine-grained model. Although we know ASCON is not a byte-wise cipher, we still use a byte-wise collapsing strategy to determine if interactions between small chunks of a key word are involved in the observed leakage. The choice of byte-wise collapsing is not the only possible next step: we could also collapse to 16-bit chunks, or to 32-bit chunks, or to 4-bit chunks. Our choice is simply a trade-off between the profiling effort and the type of model that we can build.

The collapsing strategy implies that the key word K_0 can take 2^8 different values, and the key word K_1 is set to zero. As a consequence, we can stick with the parameters for configuring the test that we derived in the previous examples, and proceed with a data set containing only 20k traces. We re-run the initial detection (and identify once more the same leaky point as before, see the left panel of Figure 3), and we also re-run the degree analysis. This time, because we represent our key word as eight bytes, the degree of the largest possible monomial is eight. Figure 3 shows the result of the degree analysis: our target point has

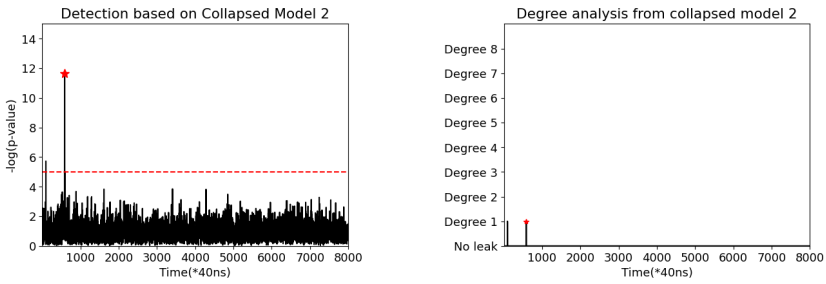

Fig. 3. Degree analysis (byte level)

Table 2. Leaking key bytes from Algorithm 2 (masked ASCON)

Key bytes	$-log_{10}(\text{p} - \text{value})$
$K_{0,0}$	3.82
$K_{0,1}$	4.67
$K_{0,2}$	4.97
$K_{0,3}$	4.62
$K_{0,4}$	1.12
$K_{0,5}$	0.76
$K_{0,6}$	0.61
$K_{0,7}$	0.48

degree $d = 1$. This means that the leakage in this point does not depend on the interaction between key bytes; key bytes contribute independently to the leakage. This information enables us to conclude that we have b-exploitable leakage for a very realistic values of b.

We run Algorithm 2 once more, which leads to Table 2. Clearly, in Table 2, the lower four bytes significantly contribute to the leakage (they lead to p-values lower than $\alpha = 0.01$). This fits neatly to the 32-bit architecture of the processor.

Identifying the leakage source within the first byte We now refine the model further in order to determine how a single byte of the key contributes exactly to the leakage. We pick the first key byte for the analysis: we do not collapse this byte, but allow all possible values to occur; we fix all other inputs to zero, and collect another trace set. Because we do not use any collapsing on the first byte, we now need a larger trace set (200k traces, adapting the analysis in Section 4.2). We confirm that the leakage is still detected (first step of the framework), and we perform the degree analysis for the bit level model. It shows that the degree is still one, which implies that the key bits are contributing independently to the leakage. We only show the output of Algorithm 2 in Table 3, which shows that the lowest key bits have the higher contributions. This gives us enough to construct a concrete attack vector.

Table 3. Leaking key bits from Algorithm 2 (masked ASCON)

Key bits	$-log_{10}(\text{p} - \text{value})$
k_0	3.16
k_1	5.71
k_2	2.23
k_3	1.23
k_4	0.81
k_5	1.77
k_6	1.97
k_7	1.70

6.4 Constructing a concrete attack vector

We wish to demonstrate a differential attack (DPA style), and thus we need to confirm that the key also interacts with the nonce (the nonce provides the differential input). Following our previous discussion, our goal is now to find out if the lowest four key bits interact with any part of the nonce. Mindful of space, we cut to the chase: we collapse the lowest 4-bit chunk of the key to a single bit, and set the remaining key to zero. We consider the nonce to consist of 4-bit chunks and collapse each chunk in the first nonce word to a single bit, and set the second nonce word to zero.

We perform key-leakage detection (which is successful in the same point as before), re-run the degree analysis, and then we perform Algorithm 2 to find out how key and nonce contribute to the leakage. Table 4 shows that the joint leakage between nonce and key leads to considerably higher test results than just the key itself. The fact that there is joint leakage between (a small part of) the nonce and (a small part of) the key enables the construction of a differential attack, with a very small inference complexity.

Table 4. Significant key and nonce interactions as determined by Algorithm 2 (masked ASCON)

(key, nonce)	$-log_{10}(\text{p} - \text{value})$
$K_{0,0..3}$	6.35
$(K_{0,0..3}, N_{0,0..3})$	15.76
$(K_{0,0..3}, N_{0,4..7})$	10.72
$(K_{0,0..3}, N_{0,8..11})$	10.08
$(K_{0,0..3}, N_{0,12..15})$	20.62
$(K_{0,0..3}, N_{0,20..23})$	26.64
$(K_{0,0..3}, N_{0,24..31})$	28.99

For the concrete attack we choose the pair $(K_{0,0..3}, N_{0,0..3})$ as our attack target (other choices are possible). Thus we assume the attacker attempts to recover the lowest four key bits, using the fact that the adversary has knowledge

of the nonce values. The attacker thus builds non-specific templates for all values of $(K_{0,0..3}, N_{0,0..3})$ (thus there are 256 templates that need to be built; the time point for profiling is known from the leakage detection step). In the attack, the adversary does a standard template-based attack to recover the lowest four key bits. Figure 4 shows how the rank of the correct key decreases as the number of attack traces increases. The correct key guess can be found within 600k attack traces (600k traces were used for training).

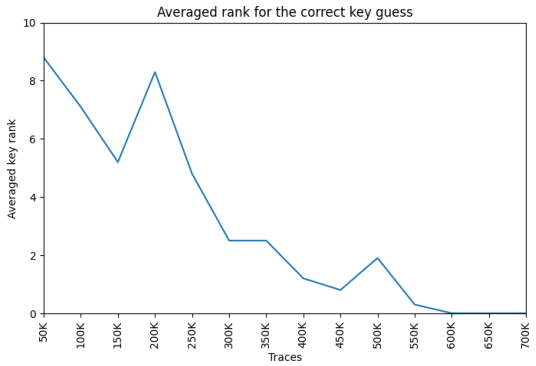

Fig. 4. Evolution of the key rank for attack on ASCON

Remark 5. We attempted a range of "natural attack vectors" arising directly from the algorithmic description and the Assembly code: targeting e.g. intermediate steps after the first round, but none of these targets lead to a successful attack. This is in fact consistent with the authors' patch: *MOV rd, #0* merely "clears" some micro-architecture, which suggests the leakage must arise from the micro-architecture.

7 Application: An Affine masked 32-bit AES implementation

We now analyse the affine masking implementation from ANSSI [26]. We run this implementation on the same ARM M3 core (NXP LPC 1313). The target core is running at 12 MHz. The trace set is recorded by a Picoscope 2205A running at 100 MSa/s, and we directly focus on (a part of) the first round Sbox computations, which leads to about 1500 points per 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 1st order TVLA on the raw traces (i.e. no 1st order univariate leak). This implies we must apply preprocessing on the trace set (see Sect. 2.3 for a brief reminder on preprocessing).

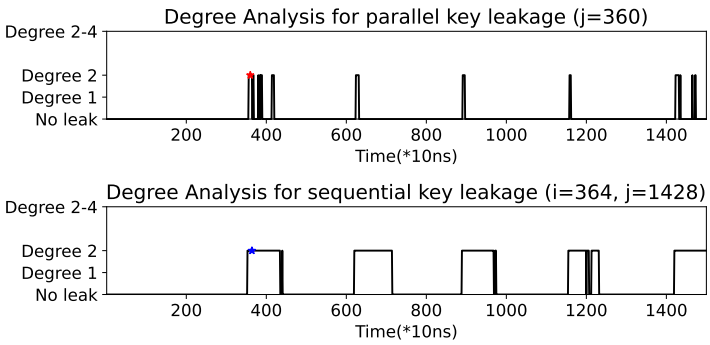

Fig. 5. Model degree analysis for affine masking

Remark 6. Our cryptographic module features a three stage pipeline. Consequently, each trace point in a side channel observation depends on three instructions which are simultaneously "in flight". The parallel nature of the processor implies that masked values and masks might be simultaneously computed on. Thus there could be key leakage due to parallel computations, and there could be key leakage due to sequential computations. We study both situations in turn using the canonical trace preprocessing functions, which are also used in TVLA.

7.1 Assessing key leakage due to parallelism

We preprocesses the traces by computing $t^{j\prime} = \left(t^j - \bar{t^j}\right)^2$ ($\bar{t^j}$ stands for the mean value of the observations at time j). Because the implementation is byte oriented, we use the same collapsing parameters and strategy as Sect. 4.2 and 5.

We initially perform a test for key leakage and identify multiple leaking points in the preprocessed trace set; the outcomes are akin to those obtained from TVLA. In contrast to TVLA, and its adoption in ISO17825, our framework now offers the ability to further analyse the identified points and ultimately construct (without undue overhead) an attack vector.

To do so we proceed as follows. For each point on the preprocessed trace, we adapt the full model from Def. 5 to find the maximum degree of the potential key leakage (we perform a leakage degree analysis). Figure 5 (upper half) shows that most leaking points have leakage degree two ($d = 2$). Because we represent the key as independent bytes, this analysis implies that the leakage points depend on $d = 2$ key bytes. This implies that there is $32-$exploitable key leakage ($b = 32$) (remember that this is an upper bound, and we will reveal an attack vector with a lower complexity).

We further pick one of these points (index 360, red asterisk in Fig. 5) and run Alg. 2 to identify the exact key byte combination, which leads to the outcomes in Table 5. As we are focusing on one single point, we simply set $\alpha = 0.01$ and reject any $-log_{10}(\text{p} - \text{value}) > 2$.

Table 5. Leaking key pairs of bytes from Algorithm 2 (affine masking, parallel leakage)

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 6. Leaking key pairs from Algorithm 2, 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

Table 5 shows that there are a number of good candidates. The pair that has the highest $-log(p - value)$ is the pair (K_0, K_3). This key pair leads to a concrete attack vector that has been reported in previous work [13], where the authors stated there is a collision attack on this particular implementation (left plot, Fig. 2 in [13]).

7.2 Assessing key leakage due to sequential processing

The sequential processing of shares can be "undone" by creating joint distributions via the multiplication of trace points. The canonical preprocessing method is then based on computing $t'^r = \left(t^i - \bar{t^i}\right)\left(t^j - \bar{t^j}\right)$. We select the point at index $(j = 1428)$ and run our leakage detection method to reveal key leakage for the entire trace (from $i = 0$ to $i = 1499$). The outcomes are akin to running TVLA, but we can proceed within our framework to determine a concrete attack vector. To do so, we determine the leakage degree for each of the (preprocessed) trace points.

Figure 5 (lower half) shows a combination of two key bytes suffices to describe most points on trace. This implies that the inference step is below $O(2^{32})$ bound, and thus we conclude that this is 32−exploitable key leakage $(b = 32)$.

We then select point 364 (blue asterisk in Fig. 5) as our target $(i = 364)$, use Alg. 2 to identify the key bytes that can describe the leakage, which leads to Table 6. Similar to the parallel case, we set $\alpha = 0.01$ here. The table shows that the key pair (K_0, K_4) is the best candidate for constructing a concrete attack vector. Like in the parallel case, this attack vector has been reported before in [13] (i.e. Section 4, Figure 2).

8 Discussion

8.1 Applications to other types of implementations

We provide two detailed application use cases for our technique, which show how to adapt the analysis to suit different ciphers and their resulting implementations. The two ciphers follow different construction principles and lead to different types of implementations even on the same platform. The AES cipher is byte-oriented and leads to a fully byte oriented implementation on our 32-bit platform. The ASCON cipher is word oriented, but the non-linear component is a 5-bit function. Thus some part of the ASCON cipher are mapped to the native 32-bit instructions of the processor, and some are implemented in a more bit wise fashion. These architectural differences have no impact on the principle of our analysis, they only guide the choices that we make to collapse the inputs in order to facilitate trace efficiency. The analysis of hardware implementations is akin to those of software implementations: the architecture of the implementation will guide analysis, and we expect that only the number of traces must increase. This will impact the effect size f^2 and thus the configuration of the test, we advice considering [7] to determine which (normalised) effect size to use. Our work thus provides evidence that implementations of similar byte oriented ciphers, or ciphers based on permutations are compatible with our detection framework.

8.2 Importance of explainability in leakage assessment.

Although rarely prioritised in the academic literature, explainability plays a critical role in adopting evaluation schemes like FIPS 140-3/ISO 17825. For instance, any fixed-versus-random plaintext test will report trace points that depend only on the plaintext. But the plaintext contains no information about the secret key. Thus, both TVLA [3] and ISO 17825 [4] have an exemption clause: only the middle $\frac{1}{3}$ of the side-channel trace is considered. The hope is, the plaintext dependent leakage will vanish in the middle part of the trace. As a result all leaks in the beginning rounds will be excluded, even if they are potentially exploitable, as we demonstrate in our ASCON example.

Our approach does not need such an "exemption clause" because all detected leaks are key dependent and our framework produces evidence that enables to reason about exploitability and explainability.

We also wish to emphasise that other existing specific tests/attacks (e.g. NICV [27] or ρ [28]) are unlikely to find the leaks from our examples either. This is because we identify leaks that are based on combinations of key and plaintext that are likely due to the micro-architecture. They cannot be explained based on (architectural states) that arise from the algorithmic description itself, not even from an Assembly representation. Even worse, micro-architectural effects do not neatly carry over from one version of a micro-processor to another (not even for similar models from the same manufacturer, see [29]), and therefore an evaluator is unlikely to know which effects are present in advance. Our new framework is currently the only method to systematically and effectively identify such leaks.

8.3 Complexity of our approach.

Data complexity. The largest model to build is L_{cf}, which suggests the data complexity (i.e. the number of available traces) should be at least $O(2^s)$ (where s is the number of collapsed key chunks). Our analyses in Section 7 uses $2^{16} * 10$ traces, which is around the same level of TVLA for hardware masking $N = 10^6$. Our analysis of ASCON is based on differently sized models, ranging from two to eight explanatory variables, all over a collapsed space, leading to trace requirements between 20k to 200k traces; which are clearly below what TVLA requires.

Time complexity. The largest computational effort comes from the application of Corollary 1, where our analysis attempts to determine the degree d through comparing different regression models. Assuming N traces were recorded for an implementation with s collapsed key chunks, building a model with degree d takes $O(C^2N)$ elementary arithmetic operations ($C = \binom{s}{d}$). In our AES experiments, we set $s = 16$ and $d = 4$, which gives $C = 1820$. This means the complexity for regression models is bounded by 2^{41}. If the underlying cipher has higher key size (e.g. AES-256) or smaller key chunks, one can always iteratively build more fine-grained models like our ASCON example in Section 6: as the regression models never exceed 256 terms ($s = 8$), the time complexity is around 2^{31}.

8.4 Extension to other model building methods and inherently multivariate methods.

Our work heavily uses existing statistical machinery for comparing nested regression models. The reason for this is that for such models an adequate comparison method exists: i.e. a formal statistical test exists, that can be correctly configured by an evaluator, so that the evaluator can derive the necessary number of side channel observation that they must gather. Such a comparison method currently only exists for nested regression models, and thus for now we must leave it as an open research question to extend our idea beyond nested regression models (e.g. machine learning or deep learning models). This implies that also the extension to inherently multivariate technique remains an open problem.

8.5 Optimal vs. confirmatory attack vectors.

Our method can reveal multiple attack vectors, with different data/time complexity. Since our primary goal is verifying a detected leak is exploitable in attacks, the "optimality" of the recovered attack vectors becomes out of our scope. However, once the attack vectors are known, it is possible to turn them into a more (trace) efficient attack. For instance, better profiling techniques can be incorporated into the very final step when an attack is carried out. It may also be possible to incorporate identified micro-architectural leakage into some horizontal attack vector. Both of these possibilities are interesting questions for further research.

Acknowledments

Si Gao was funded in part by National Key R&D Program of China (No. 2022YFB3103800) and the ERC via the grant SEAL (Project Reference 725042). Elisabeth Oswald was funded 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. Information Technology Laboratory,NIST: Security Requirements for Cryptographic Modules. https://nvlpubs.nist.gov/nistpubs/FIPS/NIST.FIPS.140-3.pdf
3. 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
4. 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)
5. Roy, D.B., Bhasin, S., Guilley, S., Heuser, A., Patranabis, S., Mukhopadhyay, D.: CC meets FIPS: A hybrid test methodology for first order side channel analysis. IEEE Trans. Computers **68**(3) (2019) 347–361
6. Moos, T., Wegener, F., Moradi, A.: DL-LA: deep learning leakage assessment. A modern roadmap for SCA evaluations. IACR Trans. Cryptogr. Hardw. Embed. Syst. **2021**(3) (2021) 552–598
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. Yap, T., Benamira, A., Bhasin, S., Peyrin, T.: Peek into the black-box: Interpretable neural network using SAT equations in side-channel analysis. IACR Trans. Cryptogr. Hardw. Embed. Syst. **2023**(2) (2023) 24–53
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. 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
11. Azouaoui, M., Bellizia, D., Buhan, I., Debande, N., Duval, S., Giraud, C., Jaulmes, É., Koeune, F., Oswald, E., Standaert, F., Whitnall, C.: A systematic appraisal of side channel evaluation strategies. In van der Merwe, T., Mitchell, C.J., Mehrnezhad, M., eds.: Security Standardisation Research - 6th International Conference, SSR 2020, London, UK, November 30 - December 1, 2020, Proceedings. Volume 12529 of Lecture Notes in Computer Science., Springer (2020) 46–66
12. 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

13. Gao, S., Oswald, E.: A novel completeness test for leakage models and its application to side channel attacks and responsibly engineered simulators. In Dunkelman, O., Dziembowski, S., eds.: Advances in Cryptology - EUROCRYPT 2022 - 41st Annual International Conference on the Theory and Applications of Cryptographic Techniques, Trondheim, Norway, May 30 - June 3, 2022, Proceedings, Part III. Volume 13277 of Lecture Notes in Computer Science., Springer (2022) 254–283
14. Picek, S., Heuser, A., Guilley, S.: Template attack versus bayes classifier. J. Cryptogr. Eng. **7**(4) (2017) 343–351
15. Picek, S., Heuser, A., Jovic, A., Ludwig, S.A., Guilley, S., Jakobovic, D., Mentens, N.: Side-channel analysis and machine learning: A practical perspective. In: 2017 International Joint Conference on Neural Networks, IJCNN 2017, Anchorage, AK, USA, May 14-19, 2017, IEEE (2017) 4095–4102
16. Prouff, E., Strullu, R., Benadjila, R., Cagli, E., Dumas, C.: Study of deep learning techniques for side-channel analysis and introduction to ASCAD database. IACR Cryptol. ePrint Arch. (2018) 53
17. Doget, J., Prouff, E., Rivain, M., Standaert, F.: Univariate side channel attacks and leakage modeling. J. Cryptogr. Eng. **1**(2) (2011) 123–144
18. 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
19. Huitfeldt, A., Stensrud, M.J., Suzuki, E.: On the collapsibility of measures of effect in the counterfactual causal framework. Emerging Themes in Epidemiology **16**(1) (2019) 1
20. 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
21. 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
22. Longo, J., Martin, D.P., Mather, L., Oswald, E., Sach, B., Stam, M.: How low can you go? using side-channel data to enhance brute-force key recovery. IACR Cryptol. ePrint Arch. (2016) 609
23. Veyrat-Charvillon, N., Gérard, B., Standaert, F.: Soft analytical side-channel attacks. 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) 282–296
24. Arrieta, A.B., Rodríguez, N.D., Ser, J.D., Bennetot, A., Tabik, S., Barbado, A., García, S., Gil-Lopez, S., Molina, D., Benjamins, R., Chatila, R., Herrera, F.: Explainable artificial intelligence (xai): Concepts, taxonomies, opportunities and challenges toward responsible ai. Inf. Fusion **58** (2019) 82–115
25. 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

26. Benadjila, R., Khati, L., Prouff, E., Thillard, A.: Hardened Library for AES-128 encryption/decryption on ARM Cortex M4 Achitecture. https://github.com/ANSSI-FR/SecAESSTM32
27. 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
28. 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
29. 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
30. 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
31. Mangard, S., Oswald, E., Popp, T.: Power Analysis Attacks: Revealing the Secrets of Smart Cards. Springer-Verlag, Berlin, Heidelberg (2007)
32. Sakic, E.: SmartCard Firmware Implementation with AES-128 decryption support. https://github.com/ermin-sakic/smartcard-aes-fw/ (2013)
33. 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
34. 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
35. Beckers, A., Wouters, L., Gierlichs, B., Preneel, B., Verbauwhede, I.: Provable secure software masking in the real-world. In Balasch, J., O'Flynn, C., eds.: Constructive Side-Channel Analysis and Secure Design - 13th International Workshop, COSADE 2022, Leuven, Belgium, April 11-12, 2022, Proceedings. Volume 13211 of Lecture Notes in Computer Science., Springer (2022) 215–235
36. Welch, B.L.: The generalization of student's problem when several different population variances are involved. Biometrika **34**(1-2) (1947) 28–35
37. 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
38. 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
39. Merino del Pozo, S., Standaert, F.X.: Getting the most out of leakage detection. In Guilley, S., ed.: Constructive Side-Channel Analysis and Secure Design, Cham, Springer International Publishing (2017) 264–281

A Application: Simulated 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 the experiments on a real device, we know exactly how each simulated trace point is generated. Consequently, this enables to test (and also demonstrate) our novel leakage assessment process.

A.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 $\mathcal{N}(0, \sigma^2)$ 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.

A.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 7), 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.

Table 7. 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.

A.3 Leakage assessment via key dependent models

We now demonstrate our assessment framework by following the statistical analyses laid out in Sections 5.

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

A key challenge for performing such analysis is the size of K and thus we use the collapsing trick explained in Section 5: 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 MixColumns),

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 [30]. 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 MixColumns 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 7 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 7 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 $deg(u_j(K_c)) > d$. 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 MixColumns, $d = 2$ often occurs when transition leakage appears between two individual bytes, while $d = 4$ represents the first MixColumns 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 MixColumns, 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 7 shows the lowest degree model that was identified by Alg. 2. 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 7 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 [13], 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.

B Application: A Boolean masked 32-bit AES implementation

We now assess a software implementation of AES that utilises a well known Boolean masking scheme [31]. The original code was written by Ermin Sakic [32] and published on GitHub. It was then re-written by the Secure Embedded Systems Research group at Virginia Tech as a more general C-based implementation [33]. 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 [31]. The masked S-box is precomputed before each encryption according to the pair of input/output masks, then all S-box look-up(-s) use exactly the same mask. In theory, no "first order attack" (i.e. attacks on raw traces, without pre-processing) should work for this implementation.

We port this implementation on an ARM M3 core (NXP LPC 1313), which is now our cryptographic module. 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, and all traces are of equal length.

B.1 Key leakage detection

In order to detect key leakage, we apply the test from Sect. 4 to our cryptographic module. This requires to collect traces for the model-based detection, which in turn requires to consider how to collapse the key space. The implementation is a very classical byte oriented AES implementation, and therefore the natural way to collapse the key is to define a collapse function for each key byte:

- Each key byte collapses to a single variable that can take two different values.
- The two different values that a key byte can take are chosen to be $0x52$ and $0x7d$, which ensures after the S-box we get the values $0x00$ and $0xff$ (we fix the plaintext to be the all zero vector).

These choices reflect minimal knowledge about the implementation: we do not make any assumption about the specifics of the masking scheme, and we make no specific assumptions about the device leakage function. However, we do assume that values are manipulated byte-wise. The collapsing function ensures that we can capture interactions between key bytes in the model.

Figure 6 shows the result, when applying the model-based detection test to a trace set of $2^{16} \times 10$ traces. Recall that we apply the test independently to all trace points (i.e. we extract the 1st, 2nd, 3rd, etc point across all side channel traces and apply the test to the j-th points); thus our result is again a trace of the same length as the original observations. The red dashed line indicates our rejection threshold, any value above indicates that there is enough evidence to reject H_0(i.e. "no leakage"in the sense of Def. 5). The red asterisk marks a trace point which we will discuss in the next sections.

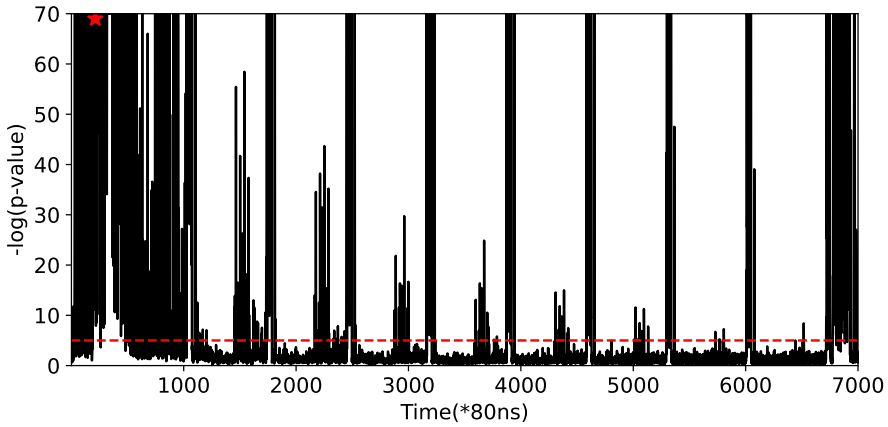

Fig. 6. Model-based detection

Why are there so many leaking points? Achieving just first-order security on a processor such as the NXP LPC 1313 is in fact not straightforward without the effort from side-channel experts. Proofs and security arguments can effectively protect the "masking scheme": however, in realistic implementations, various micro-architectural elements (e.g. buffers) contribute alongside architectural registers to the observed side channel. These contributions are not accounted in the proofs. Besides, many leaking points appear because of the transitions between the current and previous architectural states: these can completely undermine the security of any masking scheme. The recent paper by Marshall et al [29] shows many types of (unexpected) micro-architectural effects. The existence of such unexpected leakage is in fact well documented: already Balasch et al. reasoned back in 2014 that without handcrafted code from experts, one

should expect a second order scheme to be only first order secure in practice [34]. A recent study shows this remains relevant for many provably secured software masking schemes today [35].

B.2 Exploitable and explainable leakage

We now use the methodology from Sect. 5 to hunt for exploitable leakage using Alg. 1, and then demonstrate our explainable method using Alg. 2.

Detecting exploitable key leakage. In a byte oriented implementation, the interaction between key bytes will determine the complexity of the inference step (Section 3). Using the same collapse function as before, we can further analyse how many key bytes are necessary to explain the side channel observations, using Alg. 1, which leads to Fig. 7. In the first encryption round, we can see that either two bytes or four bytes of the key are required to describe the observations.

Afterwards, the full key is required to describe the observations. We can now conclude on b-exploitable leakage: because we have degree two leakage, i.e. two key bytes suffice to describe the side channel observations, we have $b = 32$-exploitable leakage, which is definitely practical. In fact, the example in Section 5.1 demonstrates this procedure for any degree-2 point in Fig. 7.

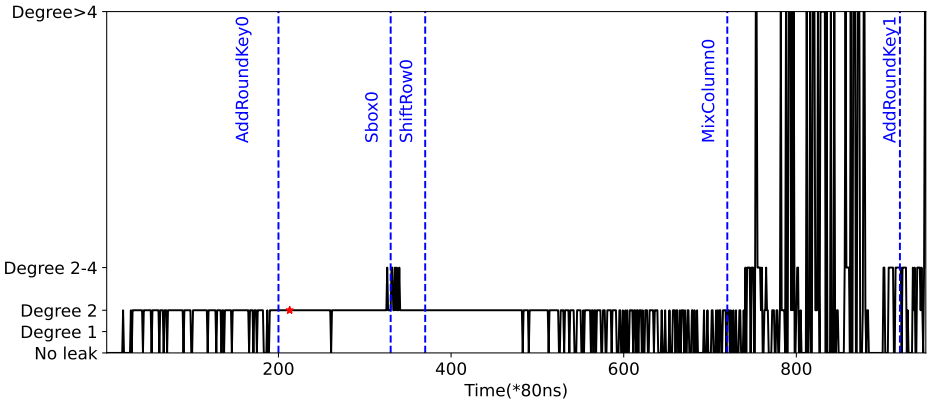

Fig. 7. Analysing the degree of the leakage function in a collapsed model

From Fig. 7, we can make two interesting observations:

– Even before the first round S-box, there are some trace points that depend on 2 key bytes: a plausible explanation is those are leaks caused by the micro-architecture register/bus bit-flips, where the random masks are accidentally cancelled out. No specific single-byte test, like [28, 27], can exploit this leakage: therefore, an evaluator would falsely conclude that the flagged trace

point is a false positive in the TVLA framework. However in our framework, an evaluator would recognise these points as exploitable.
– We found some trace points which depend on two key bytes according to our test, but also have a leakage component that only depends on a single key byte, e.g. $L(K) = \beta_0 + \beta_1 K_0 + \beta_2 K_0 K_1$. Such points can be analysed with specific detection methods that use the component $\beta_1 K_0$ but these methods would fail to use/notice the component $\beta_2 K_0 K_1$. As a result, attempts to "repair the code" based on the incomplete understanding of the true nature of the key leakage are prone to fail. This illustrates again, that single-byte specific tests would fail to deliver meaningful results, but our framework would guide an evaluator/implementer correctly.

Explainable key leakage detection. We now pick the point 213 (the red asterisk in Fig. 6 and 7) to illustrate how to produce concrete evidence that leads to an attack. According to Figure 6 and 7, the leakage on this sample can be expressed with a model that includes the interaction of no more than two key bytes. Thus, we follow the steps in Section 5.1 (i.e. Algorithm 2) to determine the exact key byte combinations. In fact, the example in Section 5.1 explains how Alg. 2 works on this exact point.

We set $\alpha = 0.01$ and reject any $-log_{10}(p - value) > 2$. If the $p - value$ exceeds the precision of our computation script (we use numpy), we simply denote them as > 324 (the maxima logarithm returned by numpy) in Table 8. Table 8 documents all the combinations of two bytes that statistically contribute to this leakage point. Perhaps surprisingly, all 6 2-byte combinations contribute to the trace point, which suggests it is unlikely this leak is caused by one register transition.

Table 8. Potential pairs of key bytes from Algorithm 2

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

The pair (K_0, K_2) has the highest significance level, and we proceed with this pair for demonstrating a specific attack for trace point 213. This requires us to take a further set of measurements, with a fixed key and varying inputs. Based on this new set, we create models/templates for this trace point.

Demonstrating an attack. To reduce the noise, we take advantage of our ability to choose plaintexts and set all plaintext bytes that are not relevant for

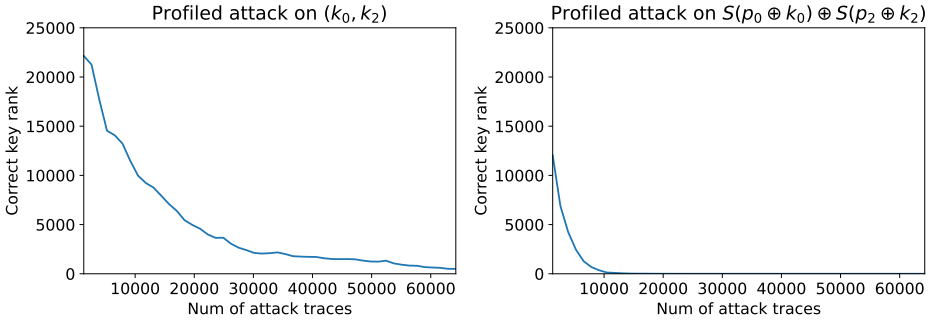

Fig. 8. Specific Attack Vectors

the attacks to 0. Varying then the values for P_0, P_2 we acquire a profiling set consisting of 650k traces, from which we randomly pick 65k traces as the attack set (i.e. 590k for profiling, 65k for attack).

We discussed the different options of model building in Sect. 2.4. We now show attack vectors for the option where we build specific models but we stick to incorporating only high level information about how AES works: from Fig. 7, we know that we must be dealing with some intermediate step prior to MixColumns. Thus we can build templates for the pair $(K_0 \oplus P_0, K_2 \oplus P_2)$, and so the inference step requires simultaneously guessing 2^{16} keys. We can build more specific models assuming that the intermediate occurs after the SubBytes step. Thus we estimate templates for $S(K_0 \oplus P_0) \oplus S(K_2 \oplus P_2)$ for point 213 as well. The cost for the templating step for this attack vector is lower than the cost for the previous attack vector: we only estimate templates for 256 values. The cost of the inference step is however the same as before (2^{16}).

The outcomes in terms of key rank are given in Figure 8, and both plots show clearly that both attack vectors are efficient. The right panel of Figure 8 confirms that this attack vector is even more efficient than the somewhat more "generic one" in reducing the average key rank: 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. We note that both attacks only use a single trace point, thus are univariate (and no pre-processing of traces was necessary in this particular case).

Both attack vectors may appear simple, but finding them without our framework would require to try out combinations of intermediate values. In the worst case all combinations of two bytes would have to be analysed — our framework is rather efficient in comparison.

C Non-specific leakage detection via the fixed-versus-random principle

The TVLA document [3] suggests using a "fixed-versus-random" methodology as a non-specific detection tool. We explain the methodology in a fixed-versus-

random plaintext mode, which works as follows. The evaluator supplies a fixed secret key, and then collects two trace sets. One trace set corresponds to a fixed plaintext input; the other corresponds to randomly chosen plaintext inputs[9].

The logic underpinning the non-specific leakage detection test is as follows. Suppose that there is no data dependency, then the two sets must have the same distribution. Otherwise, if the two sets can be distinguished, it implies that the observations are somehow dependent on the input data (i.e. plaintext). This idea translates neatly into a statistical hypothesis test as follows.

We define the null hypothesis H_0 to correspond to the situation that there is no data dependency. Then the alternative hypothesis H_1 implies that there is some data dependency. We select a (suitable) statistical procedure to check H_0. TVLA prescribes to use the Welch's t-test [36] statistic for this purpose, which is a well known (univariate) test statistic for the difference between two sample means. TVLA also prescribes a statistically motivated threshold: if the test statistic exceeds this threshold, we can refute H_0 (i.e. find data dependency).

Since the popularisation of the non-specific leakage detection approach in TVLA, several papers have appeared that discuss and aim to improve the statistical approach, often focussing on statistical shortcomings/properties of the non-specific detection test in TVLA, e.g. [7, 37, 9, 6], 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 [28, 38].

Although the TVLA framework includes the option of a "fixed-versus-random key" test alongside the "fixed-versus-random plaintext" test, all papers, and also the testing method included in [4] use the "fixed-versus-random plaintext" test. As a result, the test cannot detect any dependency that only comes from the secret key. Consequently, the key schedule cannot be analysed with the technique, and dependencies that are due to the plaintext only must be expected. Even if a "fixed-versus-random key" test is deployed, the reported leaks might still be too hard to exploit, e.g. the ciphertext-dependent leakage cannot be exploited since otherwise there will be an attack on the cipher itself.

D Comparison with TVLA

D.1 TVLA and Key leakage

Assume that a cryptographic module implements a function $f(p, k)$. The evaluator may sample traces while choosing a fixed input P and two key constants $K = c_1$ and $K = c_2$ where $c_1 \neq c_2$. They may repeat this experiment multiple times.

The evaluator defines H_0 (no leakage) as "the sets exhibit the same statistical distribution". This is the hypothesis that they seek to refute: if some statistical difference between sets can be asserted, then they conclude upon "key leakage".

[9] The sampling process needs to be controlled in specific ways to avoid unintended dependencies, and the process is fully specified in [3].

The evaluator collects all traces that correspond to $K = c_1$ into a set $\{\mathbf{t1}\}$ and similarly all traces with $K = c_2$ into $\{\mathbf{t2}\}$. Because of the assumption of Gaussian noise, the evaluator elects to use a test for the equality of distribution means. Because both sets are of equal size, they decide upon the student t-test, and set up $H_0 : \hat{\mu}(\{\mathbf{t1}\}) - \hat{\mu}(\{\mathbf{t2}\}) = 0$ vs. $H_1 : \hat{\mu}(\{\mathbf{t1}\}) - \hat{\mu}(\{\mathbf{t2}\}) \neq 0$. They also select a threshold based on their desired false rejection and false acceptance rates. They find that the test rejects the null hypothesis for their selected threshold.

Because they used a finite set of samples, and defined an efficient statistical test to refute H_0, they can conclude on the existence of "key leakage" (i.e. Def. 1).

The process above corresponds to a conceivable use case which is sometimes referred to as a "fixed-versus-fixed" technique [39], albeit as "fixed-versus-fixed key" rather than the most common "fixed-versus-fixed plaintext" setting [3]. If H_0 can be refuted, we know the observed side channel has a key dependent distribution. In other words, if the evaluator has access to both the plaintext and key, such key-based TVLA can correctly conclude of the existence of key leakage.

D.2 TVLA and Exploitable/Explainable

Although key-based TVLA can conclude on key leakage, further explaining its reported leaks is far from trivial. A typical counter-example is the ciphertext-dependent leakage: following our definition, such leak point is 128-exploitable, which is side-channel-wise useless. Nevertheless, within the "fixed-versus-fixed/random" principle, there is no further support for more fine-grained analysis. Thus, from TVLA results, we cannot simply conclude on whether the detected leak is exploitable/explainable.

D.3 Comparison of TVLA and model-based detection

For comparison, we demonstrate the results of the non-specific fixed-versus-random plaintext TVLA, from another experiment involving a Boolean masking scheme implemented for AES. The upper half of Figure 9 shows the corresponding TVLA results: the red dashed line is the threshold (appropriate for the TVLA test statistic) at which we refute the null hypothesis (no leakage).

Both tests reject a large number of points: in the TVLA result it is however unclear which of these points are plaintext-dependent only, and which points also depend on the key. In contrast, our test is certain to exhibit key leakage. Both tests flag many points at the beginning of the trace as leakage: but our test flags many more points as exhibiting key leakage in the first round than TVLA.

The fact that both tests find more in the first round (i.e. the beginning of the trace) than the later rounds (the rest of the trace contains the remaining nine rounds of AES) should not be surprising because we configured both tests with fixed inputs that lead to a nice "separation of intermediate values" after the first application of the SubBytes function. Thus we maximised the effect size for the first round, at the expense of later rounds. This is not a problem in

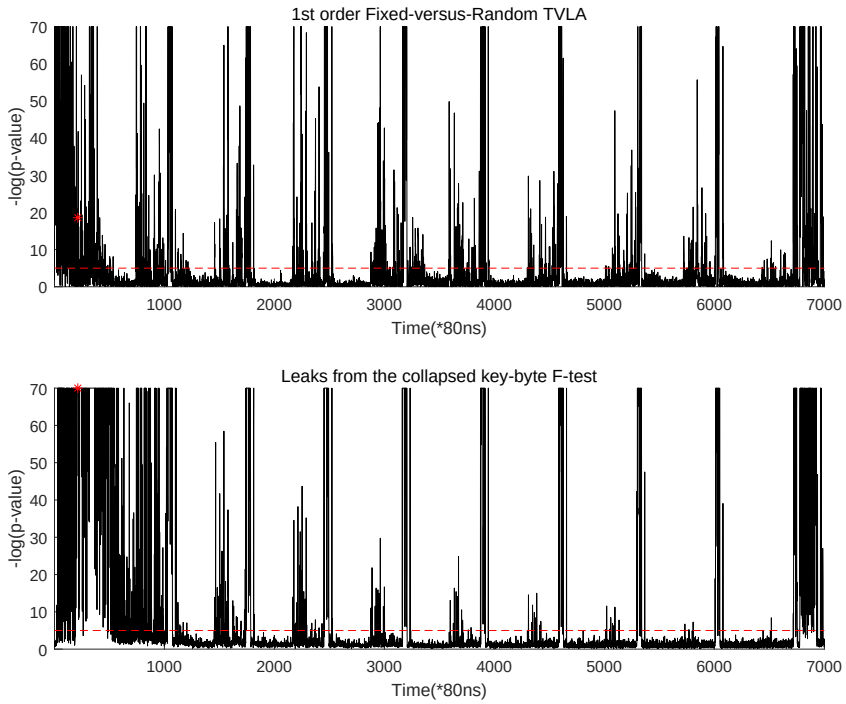

Fig. 9. TVLA vs. Model-based detection

software implementations, where all rounds typically use the same code (thus fixing problems in the first round will fix problems also in later rounds), but this can be changed so that the effect size in later rounds is maximised.

E Masked Ascon: bit-interleaved setup

It worth to mention though, Dietrich, Dobraunig and Mendel et al. provided various compile macros within their masked ASCON software implementation. Among these, one important option is whether the "bit-interleave" trick is applied (aka "ASCON_EXTERN_BI"). When "ASCON_EXTERN_BI" is set to zero, the device will reshuffles one 64-bit input word and stores as two 32-bit words for the odd and even bits respectively. Alternatively, one can also reshuffle the unshared input instead (i.e. before sending to the device), where the device does not have to further interleave the bit order.

Our experiments in Section 6 represent latter case: in other words, although we describe our attack on the lowest key/nonce nibble, they represents the lowest 4 even bits. In this section, we switch back to the default setup that consist with STM32F4 + ChipWhisperer CW308: the "bit-interleaving" is applied on device so that the unshared key/nonce data were recorded in the trivial bit order. Technically speaking, our result here should be exactly the same as Section 6, although the leakage might occur in a latter time sample (because bit-interleaving takes some execution time).

Leakage detection & 64-bit words Figure 10 (left panel) shows that without the clearing instruction, TVLA still reports leaks, but in a later position than Section 6 (1311 vs. 586). The same leak can be identified within 20k traces in our 64-bit collapsed model. Unlike our previous analysis, we can observe from Table 9 that, for time point 1311, both 64-bit words are relevant, yet the former dominates the contribution. Thus, we follow the same route as Section 6 and further refine our model on K_0.

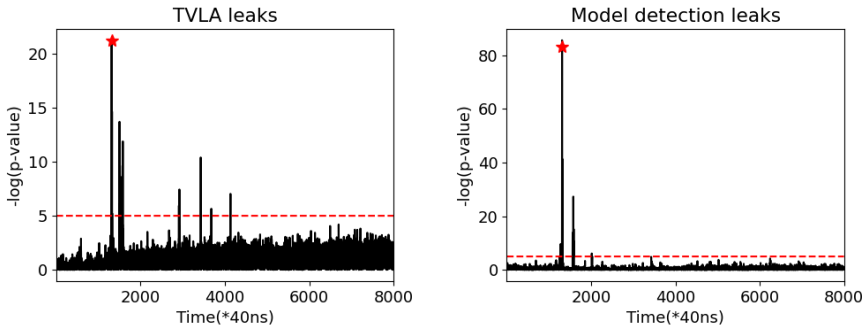

Fig. 10. TVLA vs. Model-based detection

Table 9. Leaking key words from Algorithm 2 (masked Ascon, $d = 1$)

Key word	$-log_{10}(\text{p} - \text{value})$
K_0	103.29
K_1	10.21

Relevant bytes within a 64-word Moving on to the byte-wise collapsed model within K_0, with 200k traces, Table 10 shows nearly all bytes contribute to such leakage, despite some bytes leaks more. Nevertheless, considering the bit-interleaving technique spread even and odd bits to different 32-bit words, if we were right in Table 2 saying the leakage lies in the lower 32-bit, it is expected such leakage should spread to all 8 bytes.

Within the first byte The impact of bit-interleaving becomes clear when focusing on one specific key byte. It is not hard to see from Table 11 (with 800k traces), that even bits contribute much more to the leakage.

Constructing a concrete attack vector Considering we have already described the procedure for finding the relevant nonce bits when targeting the 4 bits in Table 11, here we omit further details and directly presents an attack vector based the 4 lowest even key bits plus 4 lowest even nonce bits. The attacker still builds non-specific templates for all 256 values. Similar to Figure 4, the correct key guess can be found within 600k attack traces.

One can of course argue this is hardly surprising: following our analysis above, it is very likely we observe once again the same leakage as Section 6, just implemented in a different bit order. Nevertheless, being able to spot such leakage and deriving the same attack is reassuring: our technique is sound and stable with realistic power measurements on some state-of-the-art software platforms (e.g. ARM Cortex-M3).

Table 10. Leaking key bytes within the lowest 64 bits

Key bytes	$-log_{10}(\text{p} - \text{value})$
$K_{0,0}$	31.66
$K_{0,1}$	22.73
$K_{0,2}$	18.61
$K_{0,3}$	17.72
$K_{0,4}$	24.86
$K_{0,5}$	7.39
$K_{0,6}$	7.98
$K_{0,7}$	13.40

Table 11. Leaking key bits within the lowest byte

Key bits	$-log_{10}(\text{p} - \text{value})$
k_0	2.67
k_1	0.70
k_2	9.39
k_3	0.36
k_4	5.32
k_5	0.20
k_6	5.12
k_7	0.10

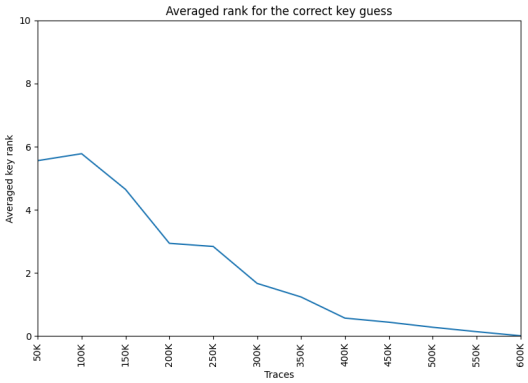

Fig. 11. Evolution of the key rank for attack on ASCON