# Information theoretic distinguishers for timing attacks with partial profiles: Solving the empty bin issue

Eloi de Chérisey, Sylvain Guilley, Olivier Rioul, Darshana Jayasinghe

# Information Theoretic Distinguishers for Timing Attacks with Partial Profiles: Solving the Empty Bin Issue

This paper is an extended version of a paper accepted at HASP workshop and presented at Seoul, Korea, on June 18, 2016, under the title "Template Attacks with Partial Profiles and Dirichlet Priors: Application to Timing Attacks".

Eloi de Chérisey[1], Sylvain Guilley[2,1], Olivier Rioul[1], Darshana Jayasinghe[1]

[1] Télécom Paris, Institut Polytechnique de Paris, France.
[2] Secure-IC S.A.S, Tour Montparnasse (27th floor), Paris, France.

***Abstract*—In any side-channel attack, it is desirable to exploit all the available leakage data to compute the distinguisher's values. The profiling phase is essential to obtain an accurate leakage model, yet it may not be exhaustive. As a result, information theoretic distinguishers may come up on previously unseen data, a phenomenon yielding empty bins. A strict application of the maximum likelihood method yields a distinguisher that is not even sound. Ignoring empty bins reestablishes soundness, but seriously limits its performance in terms of success rate. The purpose of this paper is to remedy this situation. In this research, we propose six different techniques to improve the performance of information theoretic distinguishers. We study them thoroughly by applying them to timing attacks, both with synthetic and real leakages. Namely, we compare them in terms of success rate, and show that their performance depends on the amount of profiling, and can be explained by a bias-variance analysis. The result of our work is that there exist use-cases, especially when measurements are noisy, where our novel information theoretic distinguishers (typically the soft-drop distinguisher) perform the best compared to known side-channel distinguishers, despite the empty bin situation.**

**Key words:** Timing Attacks, profiling Attacks, Dirichlet Priors, Success Rates.

## I. Introduction

The field of cryptography is currently very sensitive as it deals with data protection and safety. Thus, in order to assess the security of cryptographic devices, it is crucial to know and test their weaknesses. For example, the Advanced Encryption Standard (AES) [1] is renowned as trustworthy from a mathematical point of view—there is currently no realistic way to cryptanalyze the AES-128. However, it is possible to break the 128-bit secret key byte by byte using side-channel analysis (SCA). SCA exploits the physical fact that the secret key leaks some information out of the device boundary through various "side-channels" such as power consumption or *timing*—number of clock cycles to perform a given operation. These leakages, correctly analyzed by SCA, yield the secret key of a device.

A good side-channel attack needs a good leakage model. Timing, for example, can be modeled easily when the implementation is unbalanced: Several successful attacks [2], [3], [4], [5] exploit a timing leakage in the conditional extra-reductions of Montgomery modular multiplications. Some conditional operations can also happen in AES, e.g. in field operations, as for instance discussed in [6, Alg. 1].
Even when the code is balanced—a recommended secure coding practice—some residual unbalances in timing can result from the hardware which executes the code. Indeed, processors implement speed optimization mechanisms such as memory caching and out-of-order execution. As a consequence, it is not possible to predict with certainty how timing leaks information. The attacker is then led to make predictions about the way the device leaks.

In this paper, we consider side-channel attacks that are performed in two phases:

1) a *profiling phase* where the attacker accumulates leakage from a device with a known secret key;
2) an *attacking phase* where the attacker accumulates leakage from the device with an unknown secret key.

This type of attack is known as a *template attack* [7]. It has been shown [7] to be very efficient under three conditions: (a) leakage samples are independent and identically distributed (i.i.d.); (b) the noise is additive white Gaussian; and (c) the secret key leaks byte by byte, which enables a divide-and-conquer approach. For some side-channels, such as power or electromagnetic radiations, condition (b) is met in practice. However, for timing attacks, the noise cannot be Gaussian as timing is discrete. Moreover, the noise source is non-additive in this case, since it arises from complex replacement policies in caches and processor-specific on-the-fly instructions reordering.

The first proposed profiled timing attack is the seminal timing attack of Kocher [8]. The same methodology can be used on AES, as noted by Bernstein in 2005 [9]. Further works used the same method [10], [11], [12]. To our best knowledge, all these works consist in profiling moments, such as the average timing under a given plaintext and key. However, it is known [7] that the best attacks should use maximum

likelihood[1].

In this paper, as illustrated in Tab. I, we focus on a profiling where the distribution is characterized and used as such, and is not reduced to its moments. The attacker computes distributions using histogram methods. These distributions are then used to recover the correct secret key.

TABLE I: State-of-the-art on profiled timing attacks

| **Profiling method** | **Reference articles** |
|---|---|
| Moments | [9], [10], [11], [12] |
| Distributions | Our paper (Caution about *empty bins*) |

The discrete nature of timing leakage leads to an *empty bin* issue which appears when a data value in the attacking phase has never been seen during the profiling phase. Based on profiling only, this data should have a zero probability, which can be devastating for the attack. One known workaround is to use kernel distribution methods [13] to estimate probabilities since the smoothing can be such that no empty bins remain. This method can however be seen as a postprocessing in existing information. This alters therefore the data. In addition, this method has very large computational complexity and its performance highly depends on *ad-hoc* choices of several parameters such as kernel type and bandwidth. Moreover the estimation via the kernel method depends on other parameters such as the choice of the kernel and the size of the kernel. In our paper, we have decided to keep information as it comes as we focus on information theoretic distinguishers.

*a) Contributions:* In this paper, we show that even when all abovementioned requirements (a), (b), and (c) are not present, timing attacks with incomplete profiling can be achieved successfully by adapting the maximum likelihood distinguisher and keeping the histogram method for probabilities estimation. We build six different distinguishers, which are all good answers to the empty bin issue. For some of them, new histograms are built, such that the empty bin issue totally disappears. Furthermore, we compare these distinguishers and show which one of them is the best in every specific context. We underline that, in practice, for a moderate profiling with $256\,000$ offline measurements, the *soft drop* and the combined *offline-online profiling* approaches are clearly the two best strategies: the AES key is typically extracted with only about $2\,000$ online measurements, i.e., a complete break in about $0.2$ ms. Finally, we provide some theoretical results proving how optimal some of the distinguishers can be.

*b) Organization:* The paper is organized according to the following structure. Section II provides mathematical tools to understand distinguishers and notations. Section III introduces new distinguishers that are suitable in the context of empty bins. Section IV provides simulations for these distinguishers and Section V investigates real attacks on an ARM processor. Interestingly, all proposed distinguishers work, albeit with very noticeably different performances. In section VI, some interpolations of the obtained results in the presence of external measurement noise are derived. Section VII concludes.

[1]We will explain in Subsec. VI-B that in practice, maximum likelihood might not always perform better than moment-based distinguishers in ideal situations (no noise), because the learning stage for probability mass functions demands too many traces; besides an imperfect profiling is very detrimental to maximum likelihood distinguishers, and affects less the moment-based distinguishers. However, in non-ideal situations, e.g., in the presence of *random delay* kind of noise, maximum likelihood remains robust, where the model-based distinguishers collapse (since they are value-based).

## II. Mathematical Derivations

### A. Notations and Assumptions

We consider a side-channel attack with a profiling stage and use the following notations:

- during the profiling phase, a vector $\widehat{\mathbf{t}}$ of $\widehat{q}$ text bytes is sent and the profiler garners a vector of $\widehat{\mathbf{x}}$ measurements;
- during the attacking phase, a vector $\widetilde{\mathbf{t}}$ of $\widetilde{q}$ text bytes is sent and the attacker gathers a vector $\widetilde{\mathbf{x}}$ of leakage measurements—also customarily known as *traces*;
- we use simplified notations $\mathbf{t}$, $q$ and $\mathbf{x}$ when discussing either profiling data or attacking data;
- the probability of a vector $\mathbf{x}$ with i.i.d. components $x_i$ is denoted by $\mathbb{P}(\mathbf{x}) = \prod_i \mathbb{P}(x_i)$;
- we define the following sets:
  1) $\widehat{\mathcal{X}}$, $\widehat{\mathcal{T}}$, $\widetilde{\mathcal{X}}$ and $\widetilde{\mathcal{T}}$ are the sets of possible values of components $\widehat{x}$, $\widehat{t}$, $\widetilde{x}$ and $\widetilde{t}$, respectively;
  2) $\mathcal{X} = \widehat{\mathcal{X}} \cup \widetilde{\mathcal{X}}$ and $\mathcal{T} = \widehat{\mathcal{T}} \cup \widetilde{\mathcal{T}}$;
  3) $\mathcal{K}$ is the set of all possible values for the key $k$.
- $k$ and $t$ are made of $n$ bits (in particular, they are "bytes" when $n = 8$).

Here all sample components of one vector are i.i.d. and belong to some discrete set. Typically, $\mathcal{X}$ is a finite subset of $\mathbb{N}$ and $\mathcal{T}$ is equal to $\{0,1\}^n$.

In the profiling stage, the secret key $\widehat{k}^*$ is known and variable. In the attacking phase, the secret key $\widetilde{k}^*$ is unknown but fixed. Further, we assume that $x_i$ depends only on $t_i$ and $k^*$ for all $i = 1, 2, \ldots, q$, in the form:

$$x_i = \psi(t_i \oplus k^*) \qquad (i = 1, 2, \ldots, q) \tag{1}$$

where $\oplus$ is the XOR (exclusive or) operator and $\psi$ is an unknown function which may contain noise, masking and other hidden parameters[2].

Furthermore, in this paper, we use of the notation $n_{x,t}$ to denote the number of occurrences of $(x, t)$. Thus we can write

$$\widehat{n}_{x,t} = \textstyle\sum_{i=1}^{\widehat{q}} \mathbb{1}_{\widehat{x}_i = x, \widehat{t}_i = t} \qquad \widehat{n}_x = \textstyle\sum_{i=1}^{\widehat{q}} \mathbb{1}_{\widehat{x}_i = x}, \tag{2}$$

$$\widetilde{n}_{x,t} = \textstyle\sum_{i=1}^{\widetilde{q}} \mathbb{1}_{\widetilde{x}_i = x, \widetilde{t}_i = t} \qquad \widetilde{n}_x = \textstyle\sum_{i=1}^{\widetilde{q}} \mathbb{1}_{\widetilde{x}_i = x}. \tag{3}$$

where $\mathbb{1}_A = 1$ if $A$ is true, $= 0$ otherwise.

**Definition 1** (Probabilities)**.** *We define three*[3] *different types of probabilities* $\mathbb{P}$, $\widehat{\mathbb{P}}$ *and* $\widetilde{\mathbb{P}}$. $\mathbb{P}$ *is the actual (real) underlying probability distribution, but it is generally not available and has to be estimated by either* $\widehat{\mathbb{P}}$ *or* $\widetilde{\mathbb{P}}$.

[2]The AES meets the secret and the text byte through a xor (`SubBytes`) executed in a fixed number of clock cycles. However, the rest of the AES consists in table look-ups and other miscellaneous operations which are difficult to model and need different amounts of time to execute, hence the use of unknown function $\psi$.

[3]For the sake of evading the empty bin issue, we will also introduce yet another notation "$\overline{\mathbb{P}}_\alpha$" in section III-A (Equation (19)).

- $\widehat{\mathbb{P}}$ *is computed using the profiling data:*

$$\widehat{\mathbb{P}}(x,t) = \frac{1}{\widehat{q}} \sum_{i=1}^{\widehat{q}} \mathbb{1}_{\widehat{x}_i=x, \widehat{t}_i=t} = \frac{\widehat{n}_{x,t}}{\widehat{q}}, \tag{4}$$

$$\widehat{\mathbb{P}}(x) = \frac{1}{\widehat{q}} \sum_{i=1}^{\widehat{q}} \mathbb{1}_{\widehat{x}_i=x} = \frac{\widehat{n}_{x}}{\widehat{q}}. \tag{5}$$

- $\widetilde{\mathbb{P}}$ *is computed using the attacking data:*

$$\widetilde{\mathbb{P}}(x,t) = \frac{1}{\widetilde{q}} \sum_{i=1}^{\widetilde{q}} \mathbb{1}_{\widetilde{x}_i=x, \widetilde{t}_i=t} = \frac{\widetilde{n}_{x,t}}{\widetilde{q}}, \tag{6}$$

$$\widetilde{\mathbb{P}}(x) = \frac{1}{\widetilde{q}} \sum_{i=1}^{\widetilde{q}} \mathbb{1}_{\widetilde{x}_i=x} = \frac{\widetilde{n}_{x}}{\widetilde{q}}. \tag{7}$$

In practice, as the secret key leaks through the function via a XOR (Equation (1)), we shall often consider $\mathbb{P}(x, t \oplus k)$.

For a fair comparison between distinguishers, Standaert et al. [14] have put forward the *success rate* as a measure of efficiency of a given distinguisher.

**Definition 2** (Success Rate)**.** *The success rate* SR *is probability, averaged over all possible keys, of obtaining the correct key.*

$$\mathsf{SR} = \frac{1}{2^n} \sum_{k^*=0}^{2^n-1} \mathbb{P}_{k^*}(\widetilde{k} = k^*), \tag{8}$$

*where $\widetilde{k}$ is the key guess obtained by the distinguisher during the attack.*

It has been proven [15, Theorem 1, equation (3)] that for equiprobable keys the optimal distinguisher maximizes likelihood:

$$\mathcal{D}_{\mathsf{Optimal}}(\widetilde{\mathbf{x}}, \widetilde{\mathbf{t}}) = \arg \max_{k \in \mathcal{K}} \mathbb{P}(\widetilde{\mathbf{x}} | \widetilde{\mathbf{t}} \oplus k). \tag{9}$$

In equation (9), we use the "arg max" operator, which is defined as follows: let a function $f : \mathcal{K} \to \mathbb{R}$, then

$$\arg \max_{k \in \mathcal{K}} f(k) = \{k \in \mathcal{K} \text{ such that } \forall k' \in \mathcal{K}, f(k) \geq f(k')\}.$$

In real life, however, the attacker does not know the leakage model perfectly and thus $\mathbb{P}(\widetilde{\mathbf{x}}|\widetilde{\mathbf{t}} \oplus k)$ is not available. In order to get an estimation of $\mathbb{P}$, we use the profiling data to build $\widehat{\mathbb{P}}$ defined in Equation (4). This is the classical *template attack*. The distinguisher becomes

$$\mathcal{D}_{\mathsf{Template}}(\widetilde{\mathbf{x}}, \widetilde{\mathbf{t}}) = \arg \max_{k \in \mathcal{K}} \widehat{\mathbb{P}}(\widetilde{\mathbf{x}} | \widetilde{\mathbf{t}} \oplus k). \tag{10}$$

This distinguisher *is no longer optimal* as it does not use the real distribution $\mathbb{P}$. However, if profiling tends to exhaustivity, $\widehat{\mathbb{P}}$ and $\mathbb{P}$ will be very close since by the law of large numbers,

$$\forall x, t \quad \widehat{\mathbb{P}}(x,t) \underset{\widehat{q} \to \infty}{\longrightarrow} \mathbb{P}(x,t). \tag{11}$$

Moreover, we notice that non-optimality is not the only issue with template attacks in the context of discrete leakage. The attacker also faces the problem that the attack is ill-formed. In practice, it is convenient to use the logarithm $\arg \max_{k \in \mathcal{K}} \log \widehat{\mathbb{P}}(\widetilde{\mathbf{x}}|\widetilde{\mathbf{t}} \oplus k)$. Notice that the basis of the logarithm is arbitrary, as all key hypotheses scale alike when switching bases. In fact, since the samples are i.i.d., we have

$$\mathbb{P}(\widetilde{\mathbf{x}}|\widetilde{\mathbf{t}} \oplus k) = \prod_{i=1}^{\widetilde{q}} \mathbb{P}(\widetilde{x}_i|\widetilde{t}_i \oplus k) \tag{12}$$

and

$$\widehat{\mathbb{P}}(\widetilde{\mathbf{x}}|\widetilde{\mathbf{t}} \oplus k) = \prod_{i=1}^{\widetilde{q}} \widehat{\mathbb{P}}(\widetilde{x}_i|\widetilde{t}_i \oplus k). \tag{13}$$

Therefore, the attacker computes

$$\mathcal{D}_{\mathsf{Template}}(\widetilde{\mathbf{x}}, \widetilde{\mathbf{t}}) = \arg \max_{k \in \mathcal{K}} \sum_{i=1}^{\widetilde{q}} \log \widehat{\mathbb{P}}(\widetilde{x}_i|\widetilde{t}_i \oplus k) \tag{14}$$

where the logarithm is used to transform products into sums for a more reliable computation. However, we would like to avoid empty bins for which $\widehat{\mathbb{P}}(\widetilde{x}_i|\widetilde{t}_i \oplus k) = 0$; otherwise, Equation (14) would not be well defined.

### B. About Empty Bins

The empty bin issue appears when there exists $i \in \{1, \ldots, \widetilde{q}\}$ and $k \in \mathcal{K}$ such that $\widetilde{\mathbb{P}}(\widetilde{x}_i|\widetilde{t}_i \oplus k) > 0$ and $\widehat{\mathbb{P}}(\widetilde{x}_i|\widetilde{t}_i \oplus k) = 0$. This may even happen for the *correct* key hypothesis, leading to a wrong key guess during the attack.

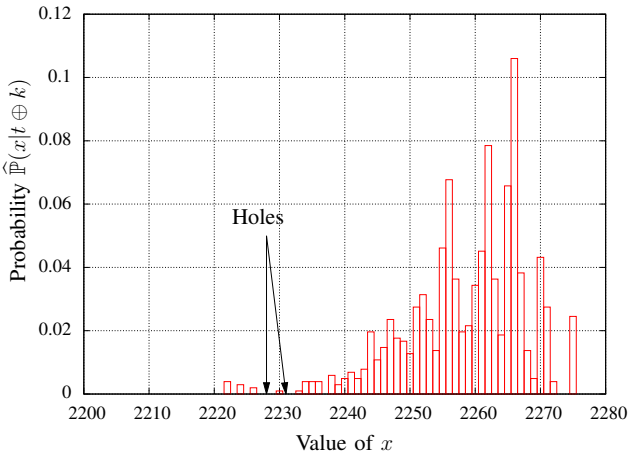



Fig. 1: Empirical probability $\widehat{\mathbb{P}}(x|t \oplus k)$ for $t = 0$ and $k = 67$ and $\widehat{q} = 2\,560\,000$

Figures 1 and 2 show how empty bins can look like after a profiling phase[4]. We notice that some parts of the histograms are left blank, some of them indicated by arrows noticed as "holes" in the figures. These timing values $x$ are possible "empty bins". When such a hole is called during the attack, meaning that the attacker gets a trace with corresponding with a hole, we call this an *empty bin*. Notice that no additional "binning" is needed as in the case of continuous distributions. The figures also show that the noise is not Gaussian as can be observed from the shape of the distribution.

[4]Figures obtained with the STM Discovery Board presented in Section V. The unit of $x$ is the "clock cycle".

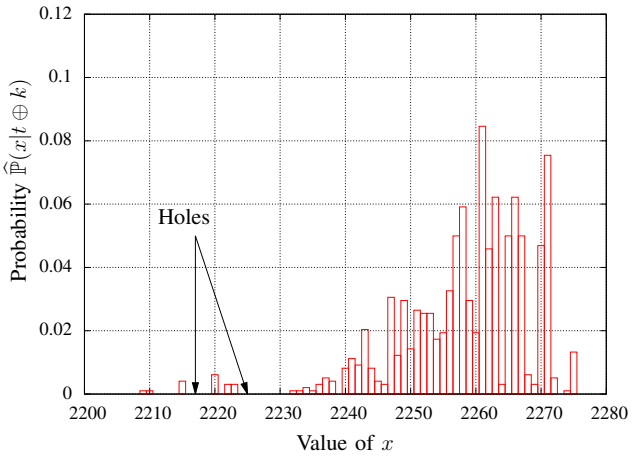



Fig. 2: Empirical probability $\widehat{\mathbb{P}}(x|t\oplus k)$ for $t=0$ and $k=149$ and $\widehat{q}=2\,560\,000$

The shortcoming of empty bins can be seen when evaluating the likelihood. The attacker encounters a zero probability, which makes the product vanish for the probability of a given key guess, even if many traces are used. As we wrote earlier, the empty bin may appear even for the correct key guess in template attacks, leading to a null success rate if not taken into account and not well treated. As an example, the number of empty bins for the practical example presented Section V for the *correct key guess* is around 500 for a poor learning phase ("poor" in that the amount of training data is limited) and around 50 for a good learning phase. This multiplication by zero is not inherent to the attack; it is rather a profiling artifact. In fact, with more profiling traces, the empty bin would likely be populated. Thus, the empty bin issue is a mere side-effect of insufficient profiling, which results in an attack failure if it is encountered in the computation of the likelihood of the correct key.

## III. Distinguishers which Tolerate Empty Bins

### A. Building Distributions or Models

Before presenting the novel distinguishers in Subsection III-B, we need to define yet another other type of distribution known as a Dirichlet *a posteriori* in a Bayesian approach.

*The Dirichlet A Posteriori:* In order to avoid zero probabilities, we use a method based on Dirichlet Prior calculations [16, Section 1]. This method leads to a new distribution denoted by $\overline{\mathbb{P}}_\alpha$, where $\alpha>0$ is a user-defined parameter whose value (typically $=1$) will be discussed next.

Let $\mathcal{X}$ be the set of possible values for $x$ and $\mathcal{T}$ be the set of possible values for $t$. For any $x$, we set $p_{x,t}=\mathbb{P}(x,t)$ their joint probability and $\mathbf{p}=(p_{x,t})_{x,t}$. Prior to obtaining any trace, $p_{x,t}$ is completely unknown and we consider a Bayesian approach to estimate $p_{x,t}$.

1) We consider the following *a priori*: without further information, we suppose that for all $x,t$,

$$\overline{\mathbb{P}}_\alpha(x,t)=\frac{\alpha_{x,t}}{\sum_{x',t'}\alpha_{x',t'}},$$

where $\alpha_{x,t}>0$ is an *a priori* parameter. To simplify, we may choose $\alpha_{x,t}=\alpha$ constant for all $x,t$. Let us suppose that $\mathbf{p}$ follows a Dirichlet (prior) distribution, whose probability density function is

$$f(\mathbf{p})=\frac{\Gamma(\sum_{x,t}\alpha_{x,t})}{\prod_{x,t}\Gamma(\alpha_{x,t})}\prod_{x,t}p_{x,t}^{\alpha_{x,t}-1}, \tag{15}$$

where $\Gamma$ is the Gamma function defined for $x>0$ as

$$\Gamma(x)=\int_0^{+\infty}t^{x-1}e^{-t}\,\mathrm{d}t. \tag{16}$$

The Dirichlet distribution can also be written as

$$f(\mathbf{p})=\mathcal{N}_\alpha\prod_{x,t}p_{x,t}^{\alpha_{x,t}-1}, \tag{17}$$

where $\mathcal{N}_\alpha=\frac{\Gamma(\sum_{x,t}\alpha_{x,t})}{\prod_{x,t}\Gamma(\alpha_{x,t})}$ is a normalization factor. Notice that the prior distribution is *uniform* when $\alpha_{x,t}=\alpha=1$ for all $x,t$.

2) Then suppose we know $\widehat{\mathbf{x}}$, $\widetilde{\mathbf{x}}$, $\widehat{\mathbf{t}}$ and $\widetilde{\mathbf{t}}$. We can now compute the *a posteriori* probability

$$\mathbb{P}(x,t|\widehat{\mathbf{x}},\widetilde{\mathbf{x}},\widehat{\mathbf{t}},\widetilde{\mathbf{t}})=\int f(\mathbf{p},x,t|\widehat{\mathbf{x}},\widetilde{\mathbf{x}},\widehat{\mathbf{t}},\widetilde{\mathbf{t}})\,\mathrm{d}p.$$

By Bayes' rule,

$$f(\mathbf{p},x,t|\widehat{\mathbf{x}},\widetilde{\mathbf{x}},\widehat{\mathbf{t}},\widetilde{\mathbf{t}})=\mathbb{P}(x,t|\mathbf{p},\widehat{\mathbf{x}},\widetilde{\mathbf{x}},\widehat{\mathbf{t}},\widetilde{\mathbf{t}})f(\mathbf{p}|\widehat{\mathbf{x}},\widetilde{\mathbf{x}},\widehat{\mathbf{t}},\widetilde{\mathbf{t}}).$$

As components $x_i$ and $t_i$ are i.i.d., we can write

$$\begin{aligned}f(\mathbf{p},x,t|\widehat{\mathbf{x}},\widetilde{\mathbf{x}},\widehat{\mathbf{t}},\widetilde{\mathbf{t}})&=\mathbb{P}(x,t|\mathbf{p})\cdot f(\mathbf{p}|\widehat{\mathbf{x}},\widetilde{\mathbf{x}},\widehat{\mathbf{t}},\widetilde{\mathbf{t}},t)\\&=p_{x,t}\cdot f(\mathbf{p}|\widehat{\mathbf{x}},\widetilde{\mathbf{x}},\widehat{\mathbf{t}},\widetilde{\mathbf{t}})\end{aligned}$$

Again by Bayes' rule,

$$\begin{aligned}f(\mathbf{p}|\widehat{\mathbf{x}},\widetilde{\mathbf{x}},\widehat{\mathbf{t}},\widetilde{\mathbf{t}})&=\frac{\mathbb{P}(\widehat{\mathbf{x}},\widetilde{\mathbf{x}},\widetilde{\mathbf{t}},\widehat{\mathbf{t}}|\mathbf{p})f(\mathbf{p})}{\mathbb{P}(\widehat{\mathbf{x}},\widetilde{\mathbf{x}},\widetilde{\mathbf{t}},\widehat{\mathbf{t}})}\\&=\frac{\prod_{x',t'\in\mathcal{X}\times\mathcal{T}}p_{x',t'}^{\widehat{n}_{x',t'}+\widetilde{n}_{x',t'}(k)}}{\mathbb{P}(\widehat{\mathbf{x}},\widetilde{\mathbf{x}},\widetilde{\mathbf{t}},\widehat{\mathbf{t}})}f(\mathbf{p})\\&=\frac{\mathcal{N}_\alpha}{\mathbb{P}(\widehat{\mathbf{x}},\widetilde{\mathbf{x}},\widetilde{\mathbf{t}},\widehat{\mathbf{t}})}\prod_{x',t'\in\mathcal{X}\times\mathcal{T}}p_{x',t'}^{\widehat{n}_{x',t'}+\widetilde{n}_{x',t'}+\alpha_{x',t'}-1}.\end{aligned}$$

We recognize another Dirichlet distribution with parameters $\widehat{n}_{x',t'}+\widetilde{n}_{x',t'}+\alpha_{x',t'}$. Let $\mathcal{N}_{\alpha'}=\frac{\Gamma(\sum_{x',t'}\alpha_{x',t'}+\widetilde{n}_{x',t'}+\alpha_{x',t'})}{\prod_{x,t}\Gamma(\alpha_{x,t}+\widetilde{n}_{x',t'}+\alpha_{x',t'})}$ be the new normalization constant for this distribution. We, finally, obtain

$$\begin{aligned}&f(\mathbf{p},x,t|\widehat{\mathbf{x}},\widetilde{\mathbf{x}},\widehat{\mathbf{t}},\widetilde{\mathbf{t}})=\\&\qquad p_{x,t}\cdot\mathcal{N}_{\alpha'}\prod_{x',t'\in\mathcal{X}\times\mathcal{T}}p_{x',t'}^{\widehat{n}_{x',t'}+\widetilde{n}_{x',t'}+\alpha_{x',t'}-1}.\end{aligned}$$

Therefore,

$$\begin{aligned}&\mathbb{P}(x,t|\widehat{\mathbf{x}},\widetilde{\mathbf{x}},\widehat{\mathbf{t}},\widetilde{\mathbf{t}})=\\&\int p_{x,t}\cdot\mathcal{N}_{\alpha'}\prod_{x',t'\in\mathcal{X}\times\mathcal{T}}p_{x',t'}^{\widehat{n}_{x',t'}+\widetilde{n}_{x',t'}+\alpha_{x',t'}-1}\,\mathrm{d}p.\end{aligned}$$

which is known as the Dirichlet *a posteriori*.

3) The integral can be easily expressed in terms of the Gamma function:

$$\mathbb{P}(x,t|\widehat{\mathbf{x}},\widetilde{\mathbf{x}},\widehat{\mathbf{t}},\widetilde{\mathbf{t}}) = \frac{\Gamma(\sum_{x',t'}\alpha_{x,t}+\widehat{n}_{x',t'}+\widetilde{n}_{x',t'})}{\prod_{x',t'}\Gamma(\alpha_{x,t}+\widehat{n}_{x',t'}+\widetilde{n}_{x',t'})} \times \frac{\prod_{x',t'}\Gamma(\alpha_{x,t}+\widehat{n}_{x',t'}+\widetilde{n}_{x',t'}+\delta_{x,t})}{\Gamma(\sum_{x',t'}\alpha_{x,t}+\widehat{n}_{x',t'}+\widetilde{n}_{x',t'}+\delta_{x,t})}$$

which simplifies to

$$\mathbb{P}(x,t|\widehat{\mathbf{x}},\widetilde{\mathbf{x}},\widehat{\mathbf{t}},\widetilde{\mathbf{t}}) = \frac{\widehat{n}_{x,t}+\widetilde{n}_{x,t}+\alpha_{x,t}}{\widehat{q}+\widetilde{q}+\sum_{x',t'}\alpha_{x',t'}}.$$

This new distribution will now be noted:

$$\bar{\mathbb{P}}_\alpha(x,t) = \mathbb{P}(x,t|\widehat{\mathbf{x}},\widetilde{\mathbf{x}},\widehat{\mathbf{t}},\widetilde{\mathbf{t}}) = \frac{\widehat{n}_{x,t}+\widetilde{n}_{x,t}+\alpha_{x,t}}{\widehat{q}+\widetilde{q}+\sum_{x',t'}\alpha_{x',t'}}. \tag{18}$$

It is important to notice that for all $(x,t)\in\mathcal{X}\times\mathcal{T}$, one has $\bar{\mathbb{P}}_\alpha(x,t)>0$. In other words, $\bar{\mathbb{P}}_\alpha$ has **no empty bin issue**.

4) With $\bar{\mathbb{P}}_\alpha(x,t)$ we can calculate

$$\bar{\mathbb{P}}_\alpha(t) = \sum_x \bar{\mathbb{P}}_\alpha(x,t) = \sum_x \frac{\widehat{n}_{x,t}+\widetilde{n}_{x,t}+\alpha_{x,t}}{\widehat{q}+\widetilde{q}+\sum_{x',t'}\alpha_{x',t'}} = \frac{\widehat{n}_t+\widetilde{n}_t+\sum_t\alpha_{x,t}}{\widehat{q}+\widetilde{q}+\sum_{x',t'}\alpha_{x',t'}} = \frac{\widehat{n}_t+\widetilde{n}_t+\alpha_t}{\widehat{q}+\widetilde{q}+\sum_{x'}\alpha_{x'}},$$

where $\alpha_t = \sum_x \alpha_{x,t}$. The resulting conditional probability[5] is

$$\bar{\mathbb{P}}_\alpha(x|t) = \frac{\bar{\mathbb{P}}_\alpha(x,t)}{\bar{\mathbb{P}}_\alpha(t)} = \frac{\widehat{n}_{x,t}+\widetilde{n}_{x,t}+\alpha_{x,t}}{\widehat{n}_t+\widetilde{n}_t+\alpha_t}. \tag{19}$$

*The Learned MIA Model:* When $\widehat{q}$ is small, the model cannot be profiled accurately, and $\widehat{\mathbb{P}}$ is a bad approximation of $\mathbb{P}$. However, these profiled values $\widetilde{\mathbf{x}}$ and $\widetilde{\mathbf{t}}$ can still be useful, yet they require a more robust distinguisher.

Distinguishers that compute models using profiling have already been proposed. For example, [17], [**?**] computes a correlation on moments. However, correlations analysis may be sensitive to model errors [18]. Mutual Information Analysis (MIA) yields a distinguisher that can be robust when models are not perfectly known [18, Section 4], but it requires at least a vague estimation of the leakage model.

Since our function $\psi$ is unknown, we can create a first-order model $\widehat{\psi}$ with the profiled data as

$$\widehat{\psi}(t\oplus\widehat{k}^*) = \mathsf{Step}\Big(\frac{1}{n_t}\sum_{i\ \mathsf{s.t.}\ \widehat{t}_i=t}\widehat{x}_i\Big) \qquad (\forall t\in\mathcal{T}). \tag{20}$$

The Step function is a function that ensures the non-injectivity of the model. The simplest way to define Step is the following:

$$\mathsf{Step}(x) = \frac{\lfloor d\cdot x\rfloor}{d} \qquad (x\in\mathbb{R})$$

where $d>0$—the greater $d$, the smaller the step size. This parameter $d$ has to be small enough in order to make the model non-injective [19, Sec. 4.1]. In our case, we choose, for all our experiments, $d=1$. With such a model, it is possible to compute a MIA, which successfully distinguishes the correct key.

[5]We should normally have used the notation $\bar{\widetilde{\mathbb{P}}}_\alpha$ instead of $\bar{\mathbb{P}}_\alpha$, but we found this too heavy and confusing; hence the use of $\bar{\mathbb{P}}_\alpha$.

### B. Robust distinguishers

In this subsection, we present six distinguishers that tackle null probabilities. Some of these solutions seem quite obvious while others are deduced from the notions presented in the preceding Subsection III-A.

❶ *Hard Drop Distinguisher:* The first naive method consists in removing all the traces which, for any key guess, have a zero probability.

**Definition 3** (Hard Drop Distinguisher)**.** *The hard drop distinguisher is defined as followed:*

$$\mathcal{D}_{\mathsf{Hard}}(\widetilde{\mathbf{x}},\widetilde{\mathbf{t}}) = \arg\max_{k\in\mathcal{K}}\sum_{i\in\mathcal{I}}\log\widehat{\mathbb{P}}(\widetilde{x}_i|\widetilde{t}_i\oplus k), \tag{21}$$

*where set $\mathcal{I}$ is defined as*

$$\mathcal{I} = \Big\{i\in\{1,\dots,\widetilde{q}\}\mid\forall k\in\mathcal{K},\ \widehat{\mathbb{P}}(\widetilde{x}_i|\widetilde{t}_i\oplus k)>0\Big\}. \tag{22}$$

Recall that $\widehat{\mathbb{P}}$, defined in Equation (4), is an empirical histogram estimated on profiled data $\widehat{\mathbf{x}}$ (along with corresponding texts $\widehat{\mathbf{t}}$).

The Hard Drop Distinguisher, as the name indicates, drops some data. In very noisy cases, it may even drop most of the data.

❷ *Soft Drop Distinguisher:* The second possibility is to drop values only for some keys. However, it has to be done carefully because dropping a value in a product implicitly implies a probability value of one. For this reason, instead of removing the trace, we replace the zero probability by a constant which is smaller than the smallest probability.

**Definition 4** (Soft Drop Distinguisher)**.** *We define the Soft Drop Distinguisher as*

$$\mathcal{D}_{\mathsf{Soft}}(\widetilde{\mathbf{x}},\widetilde{\mathbf{t}}) = \arg\max_{k\in\mathcal{K}}\sum_{i\ \mathsf{s.t.}\ \widehat{\mathbb{P}}(\widetilde{x}_i|\widetilde{t}_i\oplus k)>0}\log\widehat{\mathbb{P}}(\widetilde{x}_i|\widetilde{t}_i\oplus k) + \sum_{i\ \mathsf{s.t.}\widehat{\mathbb{P}}(\widetilde{x}_i|\widetilde{t}_i,k)=0}\log\gamma, \tag{23}$$

*where $\gamma\in\mathbb{R}_+^*$ is a constant such that $\forall i,k\in\{1,\dots,\widetilde{q}\}\times\mathcal{K}$, $\gamma\le\widehat{\mathbb{P}}(\widetilde{x}_i|\widetilde{t}_i\oplus k)$. This means that we penalize data with zero probability. The smaller $\gamma$, the harder the penalty.*

The choice of parameter $\gamma$ is thus important in order to get a fair result for the distinguisher. If we choose $\gamma\ge\frac{1}{\widehat{q}}$, the penalty may be greater than the smallest strictly positive probability. This case would mean that the penalty is less important than some licit probabilities. On the other hand, choosing $\gamma$ smaller than $\frac{1}{\widehat{q}}$ means a very strong penalty. In this case, the limit when $\gamma\to0$ is a distinguisher for which only the number of empty bins is really matters. This leads to the *Empty Bin Distinguisher* presented next in Definition 8.

❸ *The Dirichlet Prior Distinguisher:* The Dirichlet Prior Distinguisher uses the Dirichlet *a posteriori* distributions presented in Subsection III-A.

**Definition 5** (The Dirichlet Distinguisher)**.** *We define the Dirichlet Distinguisher as:*

$$\mathcal{D}_{\mathsf{Dirichlet}}(\widetilde{\mathbf{x}},\widetilde{\mathbf{t}}) = \arg\max_{k\in\mathcal{K}} \bar{\mathbb{P}}_\alpha(\widetilde{\mathbf{x}}|\widetilde{\mathbf{t}}\oplus k). \tag{24}$$

**Remark 1.** *As can be seen in the construction of the Dirichlet a posteriori, the Dirichlet distinguisher is $\alpha$-dependent. It is important to evaluate the influence of $\alpha$ over the success rate. In practice, $\alpha = 1$ seems a natural choice since the corresponding prior is uniform, which minimizes the impact of the* a priori*. In contrast, another value of $\alpha$ like $1/2$ can be interpreted as an* a priori *bin count. We may also consider scenarios where $\alpha \approx 0$ to have the least possible impact to the modified values of the histogram.*

❹ *Offline-Online Profiling:* The Dirichlet Prior Distinguisher is set by $\alpha$. As we discussed in Remark 1, we can choose any $\alpha$ so long as it is strictly positive (the Dirichlet distribution would not be defined if $\alpha = 0$). However, it is interesting to study its asymptotical behavior as $\alpha$ vanishes:

$$\lim_{\alpha\to 0} \bar{\mathbb{P}}_\alpha(x|t) = \frac{\widehat{n}_{x,t} + \widetilde{n}_{x,t}}{\widehat{n}_t + \widetilde{n}_t}.$$

This distribution can be denoted as $\bar{\mathbb{P}}_0(x|t)$ and resembles a profiling stage that would start offline and continue online.

**Definition 6** (Offline-Online Profiling)**.** *The Offline-Online Profiled (OOP) distinguisher is defined as:*

$$\mathcal{D}_{\mathsf{OOP}}(\widetilde{\mathbf{x}},\widetilde{\mathbf{t}}) = \arg\max_{k\in\mathcal{K}} \bar{\mathbb{P}}_0(\widetilde{\mathbf{x}}|\widetilde{\mathbf{t}}\oplus k) \tag{25}$$

The OOP distinguisher seems easier than the Dirichlet prior distinguisher since $\alpha$ is no longer in use. Of course, it also solves the empty bin issue since for all $(x,t) \in \mathcal{X}\times\mathcal{T}$, one has $\bar{\mathbb{P}}_0(x,t) > 0$.

❺ *Learned MIA Distinguisher:* The Learned MIA Distinguisher is constructed with the profiled model function $\widehat{\psi}$ presented in Eqn. (20) of Subsection III-A.

**Definition 7** (The Learned MIA Distinguisher)**.**
*The Learned MIA Distinguisher is defined as:*

$$\mathcal{D}_{\mathsf{MIA\_Learned}} = \arg\max_{k\in\mathcal{K}} \widetilde{\mathrm{I}}\left(\widetilde{\mathbf{x}}; \widehat{\psi}(\widetilde{\mathbf{t}}\oplus k)\right), \tag{26}$$

*where $\widetilde{\mathrm{I}}$ is the empirical mutual information [20].*

❻ *Empty Bin Distinguisher:* The empty bin Distinguisher is yet another intuitive solution based on the idea that instead of avoiding null probabilities, we may take only these into account. It is the key guess with the least number of null probabilities that "should" be the correct key.

**Definition 8.** *The Empty Bin Distinguisher is defined as:*

$$\mathcal{D}_{\mathsf{Empty\_Bin}}(\widetilde{\mathbf{x}},\widetilde{\mathbf{t}}) = \arg\min_{k\in\mathcal{K}} \sum_{i=1}^{\widetilde{q}} \mathbb{1}_{\widehat{\mathbb{P}}(\widetilde{x}_i|\widetilde{t}_i\oplus k)=0}. \tag{27}$$

The Empty Bin Distinguisher assumed that missing data contain more information than actual (measured) data. More precisely, a drop should normally not happen unless the guessed key is wrong; hence, the key guess with the least drops should be the correct key. Obviously, this distinguisher is not effective anymore if no drop occurs for at least two key guesses.

*a) Further Remarks:* All these distinguishers use a profiling phase. Before comparing them, we would like to make *a priori* discussion about their respective efficiency. As the Hard Drop Distinguisher does not take into account some data, we may suppose that it will be the one with the least success rate for a given number of traces. The OOP Distinguisher takes into account two types of data: profiling and attacking data. Therefore, it should be more efficient than other distinguishers. Lastly, we build the Learned MIA Distinguisher in order to prevent model errors, such as inaccurate profiling. In that case, we suppose that Learned MIA should work better with few data during the profiling stage.

## IV. Simulated Results

In this section, we present the results obtained on a simulated model. With these results, we can give a comparison of the proposed distinguishers.

### A. Presentation of the Simulated Model

The simulated model is built as follows:

$$\begin{aligned} x_i &= \mathrm{H_w}(\mathrm{SubBytes}(t_i \oplus k^*)) + u_i \\ &= \phi(t_i \oplus k^*) + u_i = y_i(k^*) + u_i, \end{aligned} \tag{28}$$

where $u_i$ is a discrete uniformly distributed noise $u_i \sim \mathcal{U}(-\sigma,\sigma)$, SubBytes is the AES substitution box function, and $\mathrm{H_w}$ is the Hamming weight of a byte.

This very simple leakage is used to compare distinguishers in the case the attacker has no information about the model.

**Remark 2** (Optimal Distinguisher)**.** *The optimal distinguisher* (9) *can be easily calculated if the model is perfectly known, as*

$$\mathcal{D}_{\mathsf{Optimal}}(\widetilde{\mathbf{x}},\widetilde{\mathbf{t}}) = \arg\max_{k\in\mathcal{K}} \prod_{i=1}^{\widetilde{q}} \delta_\sigma(\widetilde{x}_i - \mathrm{H_w}(\mathrm{SubBytes}(\widetilde{t}_i \oplus k))), \tag{29}$$

*where $\delta_\sigma$ is defined such that $\delta_\sigma(x) = 1$ if $|x| \le \sigma$ and $0$ otherwise. In Figures 3, 4 and 5, we include the optimal distinguisher for reference, to show how far the other curves are from the fundamental limit of performance.*

By construction, the leakage simulation (28) generates some traces with zero probability, but notice that there is no $i$ such that $\mathbb{P}(x_i|t_i,k) = 0$ for the correct key guess. This academic example is useful to compare the distinguishers defined in Section III.

### B. Attack Results

We computed the success rates (8) of the various attacks (namely attacks ❶, ❷, ❹, ❺ and ❻ — attack ❸ being less

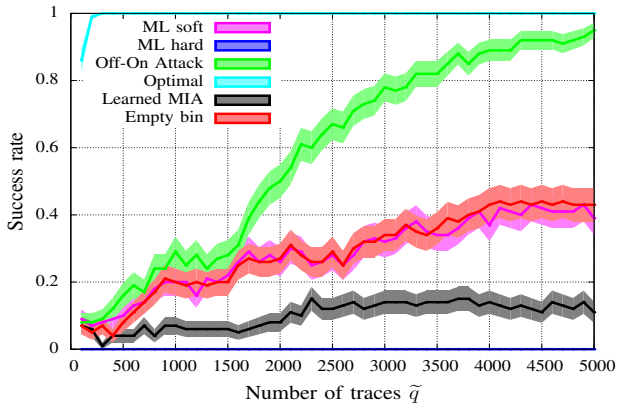



Fig. 3: SR for $\widehat{q} = 320$ and $\sigma = 24$ on synthetic measurements

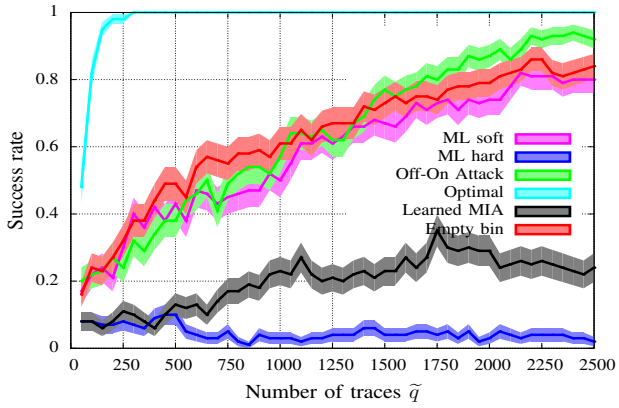



Fig. 4: SR for $\widehat{q} = 1\ 600$ and $\sigma = 24$ on synthetic measurements

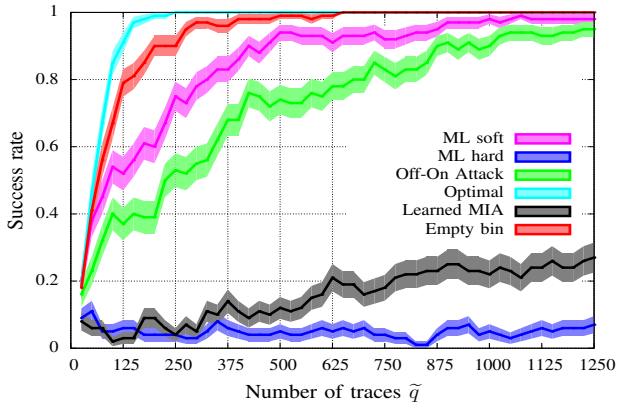



Fig. 5: SR for $\widehat{q} = 4\ 000$ and $\sigma = 24$ on synthetic measurements

efficient than its limit ❹) for $\sigma = 24$, $n = 4$ bits, and $\widehat{q}$ ranging from small to high values.

The only difference between Figures 3, 4, and 5, is that we have increased the number of data during the profiling stage. When profiling is bad (Figure 3), the best distinguisher is the Offline-Online profiling distinguisher, while the Learned MIA Distinguisher is not as good as was expected. When $\widehat{q} = 1\ 600$ (Figure 4), all distinguishers improve. Finally, when profiling is good ($\widehat{q} = 4\ 000$, Figure 5), the best distinguisher is now the Empty Bin distinguisher, followed by the Soft Drop distinguisher and the Offline-Online profiling.

**Remark 3.** *In this very special case, we can show that the Empty Bin Distinguisher can accurately approximate the Optimal Distinguisher. Indeed, the actual probability is such that for all* $(x, t) \in \mathcal{X} \times \mathcal{T}$*,*

$$\mathbb{P}(x|y(k)) = \begin{cases} \frac{1}{2\sigma+1} & \text{if } -\sigma \leq x - \phi(t \oplus k) \leq \sigma, \\ 0 & \text{otherwise,} \end{cases} \tag{30}$$

*which is constant if* $x$ *is in the appropriate interval. For the Empty Bin Distinguisher,*

$$\widehat{\mathbb{P}}(x|y(k)) > 0 \implies \mathbb{P}(x|y(k)) = \frac{1}{2\sigma+1}$$

*due to the leakage model. Therefore, we can predict that at least* $\widehat{q} = (2\sigma+1)|\mathcal{Y}|\frac{1}{\min \mathbb{P}(y)} = 3\ 920$ *profiling traces are needed to make sure that the Empty Bin Distinguisher becomes as efficient as the Optimal Distinguisher. As profiling consists in random draws* with replacement*, the* $\mathcal{D}_{\mathsf{Empty\_Bin}}$ *distinguisher is found very close to the* $\mathcal{D}_{\mathsf{Optimal}}$ *distinguisher with* $\widehat{q} = 4\ 000$ *profiling traces.*

## V. Results on Real Devices

We have chosen to carry out a timing attack on an STM32F4 discovery board [21]. One interesting aspect is that we do not make any assumption on the model. In real life, the leakage model happens to be much more complex than the one employed in simulations (e.g., Equation (28)). As will be seen, in practice empty bins appear even for the correct key guess and for a "good" profiling phase. This observation differs from the ideal case of our simulations carried out in the preceding Section IV.

### A. The ARM processor

We used a STM32F4 discovery board by STMicroelectronics [6]. It contains an STM32F407VGT6 microcontroller, which has an ARM cortex-M4 MCU with 1 MB flash memory for instructions and data, and a 192 KB Random Access Memory (RAM). The RAM is divided into three sections: one of 16 KB, another one of 112 KB, and the last one consisting of 64 KB Core Coupled Memory (CCM). The CCM has a zero flash wait state and is often used to store critical data such as data from the operating system. Since the RAM is

[6]We emphasize that the attacks we present are not due to a flaw in ARM or STMicroelectronics processors. Instead, as we will discuss next, the CCM feature of STM32F4 processors allows to protect the implementation against timing attacks by granting a constant execution time.

divided into three regions, the users are unable to make use of the 192 KB RAM as a continuous memory block.

STM32F4 microcontrollers contain a proprietary prefetch module (Adaptive Real-Time memory accelerator - ART accelerator). ART accelerator contains an instruction cache which has 64 lines and a data cache which contains 8 lines. The line size of both instruction cache and data cache is 128-bits. The precise details about ART accelerator (cache replacement policy and cache associativity) are not mentioned as the module is an intellectual property of STMicroelectronics

The STM32F407VGT6 microcontroller does not have either a CPU cycle counter or a performance register to measure a cycle accurate time. However, the Data Watchpoint and Trace (DWT) unit has a cycle accurate 32 bit counter (DWT_CYCCNT register), which can be used for measuring the duration of critical operations. When processor runs at 168 MHz, the DWT_CYCCNT register will overflow at every 25.5 seconds providing enough time window to measure the encryption / decryption time for an adversary to measure the elapsed time without timer overflowing. In practice, we collected timing data repeatedly within the ARM, and then dump it as large data buffers sporadically. This modus operandi allowed us to reach about $10\,000$ measurements per second.

### *B. Weaknesses - Non Constant AES Time*

We use OpenSSL (version 1.0.2) AES as the cryptographic library, where the $\mathrm{SubBytes}$ function is implemented with large 1 KB T-boxes (see [22, Sec. 5.2.1, page 18]). Interestingly, the OpenSSL code (copied in Appendix A) does not contain any conditional statement, hence can be considered constant-time by a code review. However, once programmed on the STM32F4 processor, one notices that the execution duration depends on the inputs. The AES timing acquisition is illustrated in Figure 6. Before each encryption, we reset DWT_CYCCNT register. This yields the exact timing of the AES execution (which is about $2\,600$ clock cycles in average — recall Figure 1 and 2). In a real attack, an attacker would measure a noisy timing using an external "chronometer". However, our attack models the best case for an attacker; hence, bounds the security of the analyzed implementation. In particular, we underline that our measurement methodology is fully *non invasive*: the timing measurement is performed in parallel to the AES computation, thereby keeping the victim circuit run at full speed, without interference.

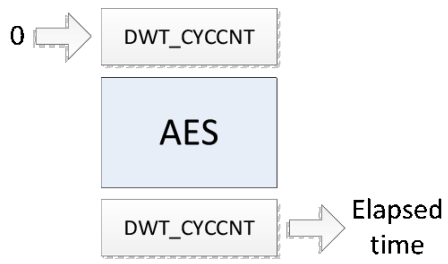



Fig. 6: Measuring elapsed time for AES encryption

Time deviations for different configurations of Instruction Cache (IC) and Data Cache (DC) are shown in Figure 7. We observe a huge time difference when data cache is turned Off / On. When DC is turned off, there is no timing leakage as AES is constant time. Yet, when DC is turned on, AES is not time constant. This non-constant time on AES leads to the following conclusions:

- This is a weakness for the security of the processor as two different plaintext lead to two different time clock to compute AES.
- Following Figure 7, it seems the enabling or not Instruction Cache, does not modify the behaviour of the leakages.
- Data presented Figure 7 are obtained using a fixed key and varying one byte of the plaintext.

Figure 7 instructs us that caches shall be disabled to reduce the leakage in timing. However, we emphasize that such decision has a strongly negative impact on the AES performance: with DC off, the overall AES execution time is about $27\%$ longer.

Therefore, in a realistic context, we shall assume that both DC and IC are enabled, which we will do in the sequel (see next Sec. VI for some indications how well attacks perform when caches are disabled).

### *C. Characterizing the leakages for Data Cache On*

As seen earlier, when the Data Cache in enabled, the AES computation is not time constant. This can be due to the T-boxes called during the computation. Indeed, calling a value in a table also stores this in the Data Cache. If this value is called within the eight next calls, the load will be faster. In Appendix A, we have copied the OpenSSL source code for the AES encryption with a 128 bits key. In this code, we notice that there are 160 calls to the T-boxes.

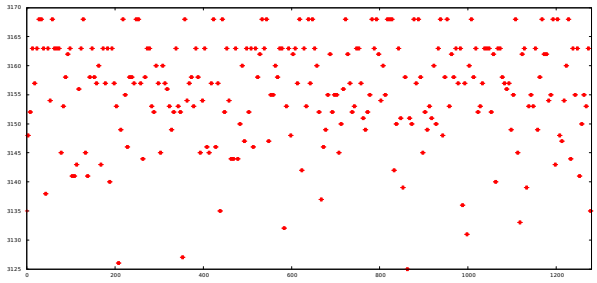
In order find a model of the leakage, we inferred the cache policy of STM32F4 ARM micro-controllers based on a thorough study of their timing response to some adaptively constructed requests. We discovered that it is actually a FIFO (First-In, First Out) cache. If one requests a particular table lookup within last eight cache accesses, then the access is a hit (if not, it is a miss).

In case of a hit, the time to access such register is 5 or 6 clock cycles faster than a miss. To show this behaviour, we have done a very simple experiment:

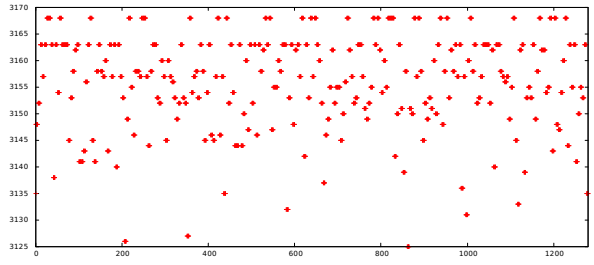
- We generate a table of length 256;
- We generate 16 random values between 0x00 and 0xff;
- We call 16 elements of the table corresponding to the 16 values generated previously;
- We measure the time to call these 16 elements of the table.

We have plotted in Figure 8 the histogram of the clock cycles. the negative number in the x axis is due to the fact that we have set the 0 at the maximum value of the clock cycles, which is the obtained value for not hit at all[7]. We notice that when a hit occurs, the time is faster by 5 or 6

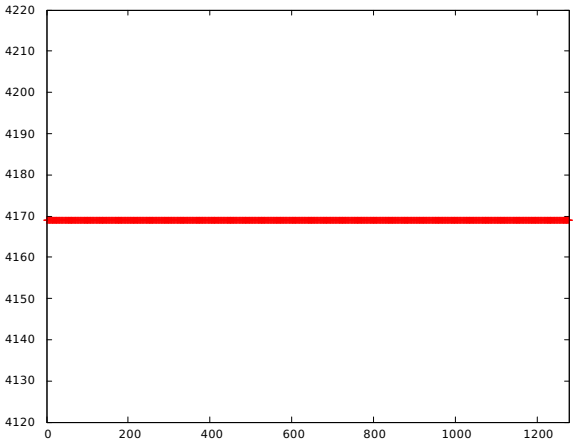
[7]This is a voluntary choice as we only focus on the gap between two picks of distribution. The absolute value has no real sense since we are comparing two computations that are not the same.



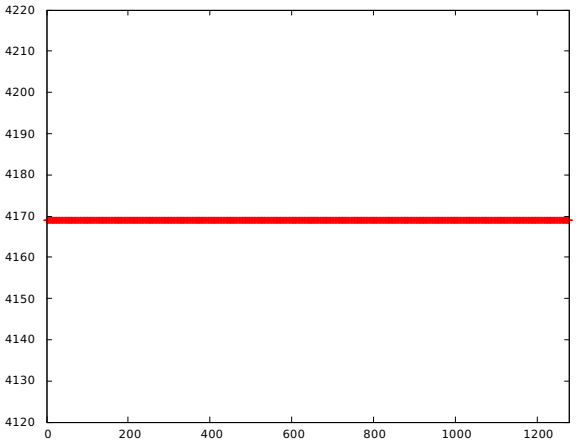
(a) IC ON and DC ON



(b) IC OFF and DC ON



(c) IC ON and DC OFF



(d) IC OFF and DC OFF

Fig. 7: Time deviations for different configurations of Instruction Cache (IC) and Data Cache (DC).

clock cycles. For two hits, there are three possible values: 10, 11 or 12 clock cycles.

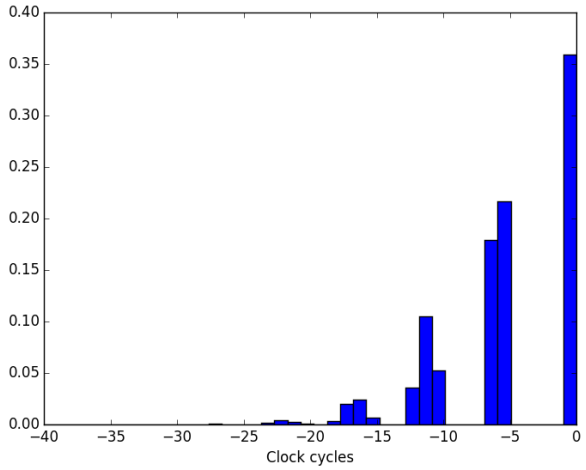



Fig. 8: Distribution of the clock cycles for a simple example

Figure 8 has to be compared with a full AES encryption timing in order to see if this model is relevant. Therefore, we have plotted in Figure 9 the histogram for a full AES encryption. Once more, the 0 in the x axis is set to the maximum.

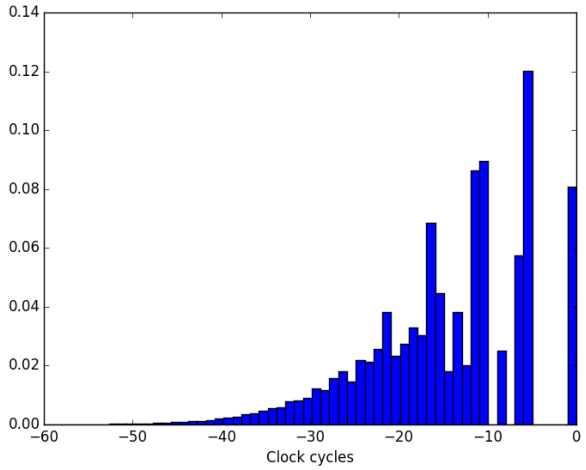



Fig. 9: Distribution of the clock cycles for a full AES encryption

Very interestingly, we can observe in this figure high density levels corresponding to the hits:

1) One hit at -5 and -6;
2) Two hits at -10 and -11;
3) Three hits at -15 and -16.

Below -16 clock cycles, the hits are lost into the noise.

The comparison of these two figures show that the FIFO model for table hits is correct, but does not explain all the time leakage due to the cache policy of the processor.

### D. Attack Results

As already noticed above, the leakage model is mostly unknown. We only suppose that the text byte is mixed with the key through a XOR operation. As a consequence, the optimal distinguisher (giving the limit of performance) is not known. The SNR of the leakage is $\mathsf{Var}(\mathbb{E}(x|t))/\mathbb{E}(\mathsf{Var}(x|t)) = 0.4$.

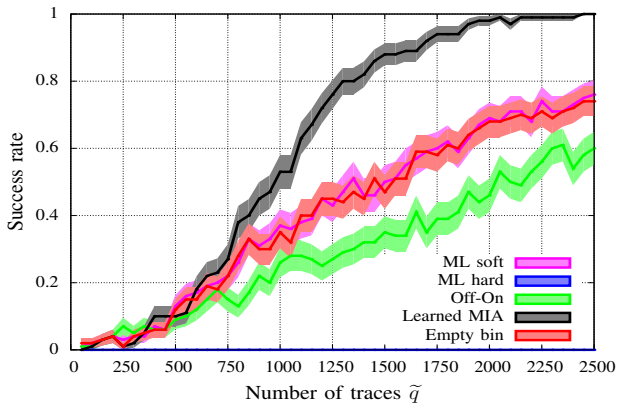



Fig. 10: SR for $\widehat{q} = 25\,600$ on real-world measurements

In Figure 10, we notice that Learned MIA is the best distinguisher in the case of poor profiling. The Hard Drop Distinguisher is not succeeding at all since it drops about 90% of the data.

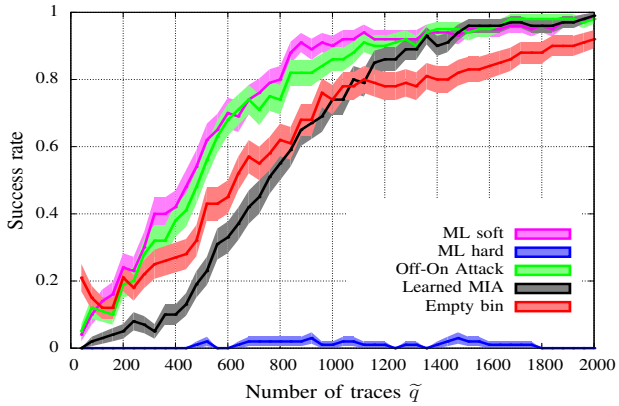



Fig. 11: SR for $\widehat{q} = 256\,000$ on real-world measurements

Figure 11 presents the success rate for a better profiling stage. We notice the following interesting improvements:

- The Learned MIA distinguisher is only slightly better than in Figure 10. To reach 80% success rate, 1 100 traces are needed as compared to 1 250 traces previously.
- The Soft Drop and Offline-Online distinguishers are the best distinguishers in this scenario, with a small advantage for the Soft Drop distinguisher.
- The Hard Drop distinguisher remains unsuccessful.

We notice that the Soft Drop Distinguisher has been established using the $\gamma$ parameter defined in Equation 23 such that $\gamma = 1/\widetilde{q}$.

Figure 12 is the continuation of Figure 11 with much more traces in the profiling stage. The resulting profiling is very

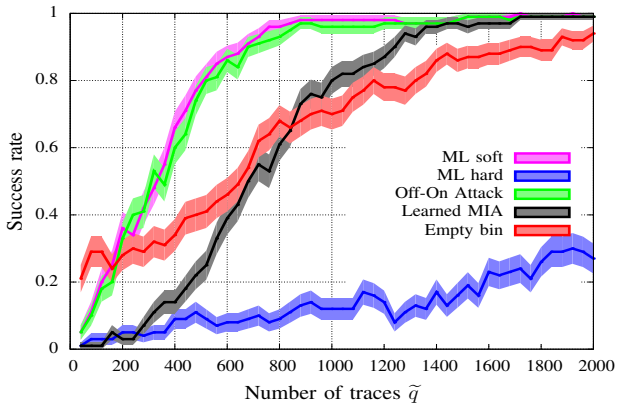



Fig. 12: SR for $\widehat{q} = 2\,560\,000$ on real-world measurements

good and one may consider that the approximation of $\mathbb{P}$ is tight. In this case, Soft Drop and OOP Distinguishers are both very successful, which seems natural regarding the fact that $\widehat{\mathbb{P}}$ has converged to the actual probability $\mathbb{P}$. For this attack, we recall that the timing of $10\,000$ traces can be acquired in one second. Therefore, the attack is successfully in about $0.2$ second using Soft Drop or OOP distinguishers.

As a conclusion to this study on the STM32F4 discovery board, we have learned the following comparisons between the proposed distinguishers:

- when the profiling stage is poor, the best distinguisher is the Learn MIA Distinguisher;
- when there is enough data in the profiling stage, the best distinguisher is the Soft Drop Distinguisher, closely followed by the OOP Distinguisher;
- the Empty Bin Distinguisher converges to the optimal success rate, but is not as efficient as previously in Section IV. This can be explained by the fact that we skip a lot of data in the computation;
- the Hard Drop Distinguisher is the slowest to converge to 100% success rate.

**Remark 4.** *When comparing Figures 11 and 12, we notice that the Empty Bin distinguisher does not improve as the number of profiling traces increases. An explanation that there is no more empty bins to be filled between these two situations; then only a more precise estimation of the probability would make the difference.*

**Remark 5.** *As discussed in Definition 4, the value of $\gamma$ is important. We have run the same experience as in Figure 11 with $\gamma = \frac{1}{\widetilde{q}\times 10^{10}}$. The results, we obtained, are presented in Figure 13. When comparing this figure with Figure 11, we notice that the performance of the Soft Drop Distinguisher has dropped and is now much closer to that of the Empty Bin Distinguisher, as we had forecast.*

### E. Nature of Empty Bins

Defined in II-B, Empty Bins can appear under two circumstances. The first possibility is insufficient profiling: some rare occurrences are not encountered by lack of

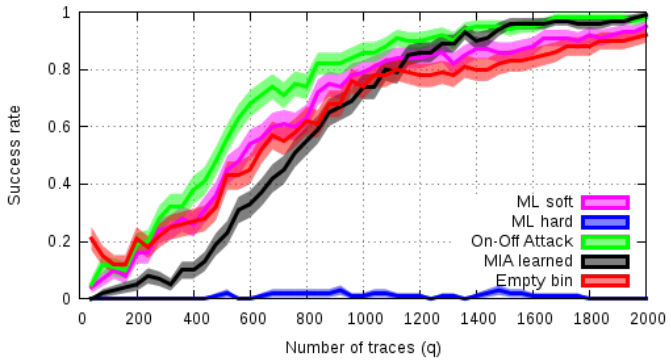



Fig. 13: SR for $\widehat{q} = 256\,000$ with $\gamma = \frac{1}{\widehat{q} \times 10^{10}}$.

training measurements. The second possibility is what we call *Structural Empty Bins*. They are present whatever the profiling under fixed key and do not depend on the number of traces $\widehat{q}$ in the profiling stage. In order to decide for the reason of Empty Bins, we have drawn the number of empty bins for a given key according to the number of traces in the profiling stage.

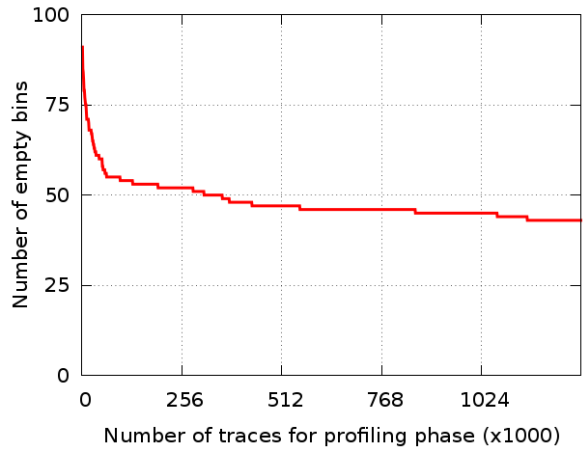



Fig. 14: Empirical number of empty bins

Figure 14 presents this study obtained with the STMicroelectronics Discovery Board. We considered $\widehat{q} = 1\,280\,000$, and define the number of empty bins as:

$$\left| \left\{ x \in \{ \min_{q=1}^{\widehat{q}} \widehat{x}_q, \ldots, \max_{q=1}^{\widehat{q}} \widehat{x}_q \}, \text{ such that } \nexists q,\ \widehat{x}_q = x \right\} \right|.$$

We can see that the number of empty bins decreases, but never reaches 0. At the beginning, the high number of empty bins is due to both poor profiling and structural empty bins. With a good profiling, we only keep the structural empty bins.

### F. Study on the Mean-Square Error

An interesting point noticed in Figures 10, 11, and 12 is that the Learned MIA distinguisher is working better than the Soft Drop Distinguisher for a poor learning phase (i.e., $\widehat{q} = 25\ 600$). However, with a better learning phase (i.e., $\widehat{q} = 256\,000$ and $\widehat{q} = 2\,560\,000$), the Soft Drop Distinguisher has a much better success rate. In order to understand why the Learned MIA Distinguisher does not improve that much with a better learning phase, we have computed the Mean-Square Error of these two distinguishers for the three learning phases (i.e., $\widehat{q} \in \{25\,600, 256\,000, 2\,560\,000\}$).

**Definition 9** (MSE, Bias and Variance)**.** *Let us consider a random variable $X$ and its expectation $\theta = \mathbb{E}[X]$. An estimator of the random variable is noted $\bar{X}$. The MSE is defined as follows:*

$$\mathsf{MSE} = \mathbb{E}\left[(\bar{X} - \theta)^2\right].$$

*The bias of the estimator is the expectation of the difference between the estimator and the mean of the random variable:*

$$\mathsf{Bias} = \mathbb{E}\left[\bar{X} - \theta\right].$$

*At last, the variance of the estimator is:*

$$\mathsf{Variance} = \mathbb{E}\left[\bar{X}^2\right] - \mathbb{E}\left[\bar{X}\right]^2$$

From these definitions, we have the following relation between MSE, bias and variance:

$$\mathsf{MSE} = \mathsf{Bias}^2 + \mathsf{Variance} \tag{31}$$

The Mean-Square Error (MSE) is computed using the following method:

1) For the secret key $k^*$, we calculate the value of the distinguisher i.e. the value of $\widehat{\mathbb{P}}(\widetilde{\mathbf{x}}|\widetilde{\mathbf{t}} \oplus k^*)$ for the Soft Drop and $I(\widetilde{\mathbf{x}}; \widehat{\phi}(\widetilde{\mathbf{t}} \oplus k^*))$ for the Learned MIA. We compute this value for different number of traces $\widetilde{q}$. This gives an estimation of the normalized distinguisher for the correct key.
2) The most accurate estimation is obtained for the highest value of $\widetilde{q}$. Therefore, taking the average over a large number of experiences for this highest value of $\widetilde{q}$ gives a good estimation of the Expectation of the estimator.
3) Then we calculate, for every value of $\widetilde{q}$ the bias and the variance of the estimator, and the Average MSE is obtained using the formula: $\mathsf{MSE} = \mathsf{Bias}^2 + \mathsf{Variance}$.

We have plotted in Figures 15 and 16 the Average MSE for the two distinguishers. In order to be more relevant, we have plotted the logarithm of the MSE. Furthermore, we have chosen to plot the MSE separately as the distinguishers are not comparable.

The MSE for the Learned MIA Distinguisher stays almost constant with the improvement of the learning phase whereas the MSE of the Soft Drop Distinguisher is much smaller. This means that a better learning phase gives a much better estimator of the distinguisher.

To understand more deeply this MSE, we separate bias and variance for these two distinguishers. The results are computed Figure 17 for the Learned MIA Distinguisher and Figure 18 for the Soft Drop Distinguisher.

We notice the following aspects:

- For the Soft Drop Distinguisher, the bias is almost equal to zero. In fact, the MSE is the variance.
- For the Learned MIA Distinguisher, it is mainly the opposite: the biggest part of the MSE is the bias.

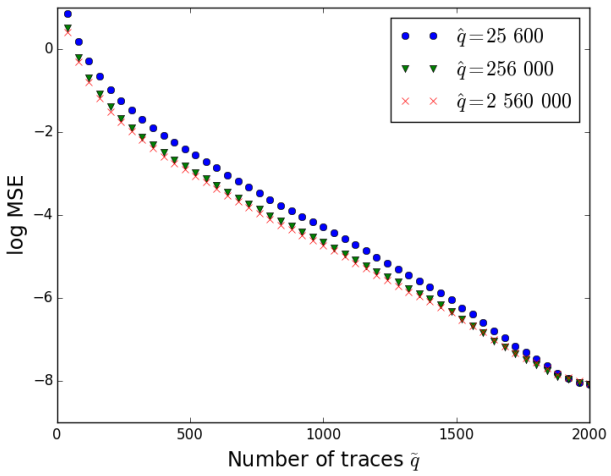



Fig. 15: Average MSE for the Learned MIA Distinguisher

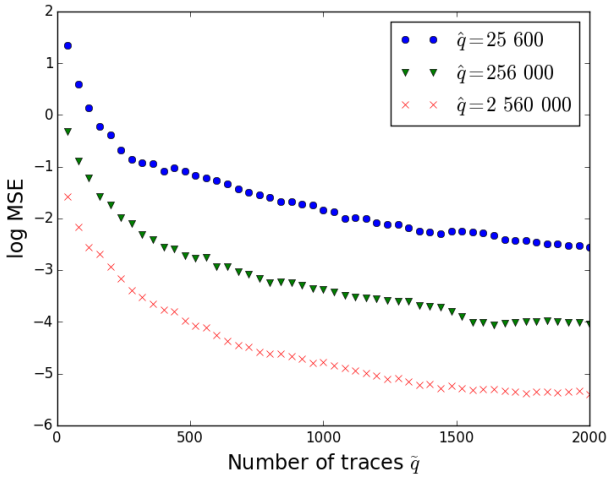



Fig. 16: Average MSE for the Soft Drop Distinguisher

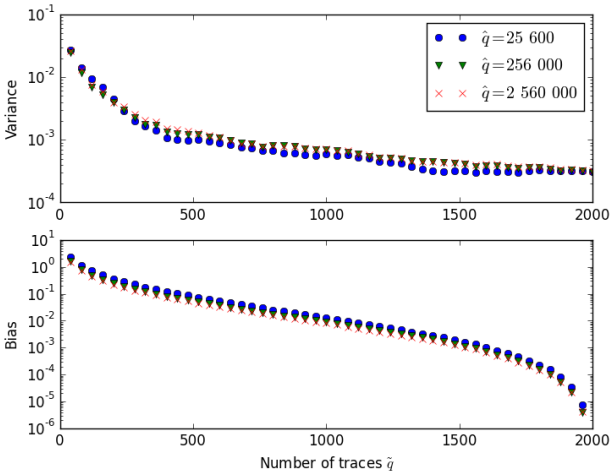



Fig. 17: Variance and bias of the Learned MIA Distinguisher

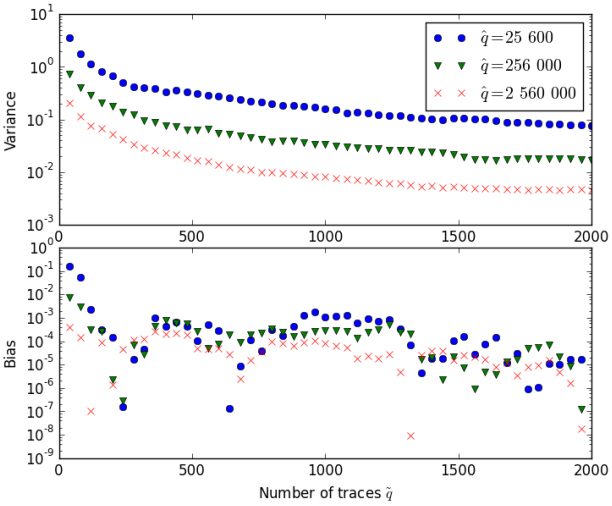



Fig. 18: Variance and bias of the Soft Drop Distinguisher

To conclude with the MSE, the Soft Drop Distinguisher improves because the estimator has a much smaller variance with a better learning phase. Meanwhile, the Learned MIA Distinguisher does not improve because it is a biased estimator and a better learning phase does not reduce this bias.

## VI. Success Rate in Presence of External Noise

The measurement setup used in simulation (Sec. IV) and on real-world traces (Sec. V) is ideal. Indeed, the only considered noise is said *algorithmic*, i.e., it consists in the varying timing which arise from the parts of the algorithm not under study. In this section, we analyse the effect of noise external to the monitored cryptographic algorithm. Subsection VI-A discusses in general terms the effect of noise addition, and subsection VI-B details quantitatively how distribution-based distinguishers cope efficiently with noise (while moment-based distinguishers fail to resist noise).

### A. Effect of Measurement Noise

However, in practice, timing measurements contain a noisy part. Let us give three examples:

1) Measure of a difference of timing between request and response from the AES (over a network of unknown latency);
2) Use of a side-channel signal (such as the power or the electromagnetic field) to observe the AES computation; the beginning and the end of an AES are easy to identify, as they consist in sixteen consecutive operations (namely sixteen XOR making up the AddRoundKey operations). As these patterns have a remarkable signature, they can be extracted with great accuracy thanks to a mere cross-correlation. Still, the AES itself might not be executed in constant time, hence some alignments issues;
3) Use of a cache attack, which would disclose that the program flows entered and exited the AES function. However, the timing for access to cache is non deterministic.

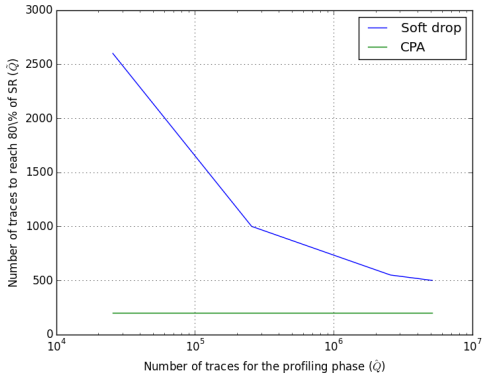



Fig. 19: Comparison between CPA and soft drop distinguisher at 80% of success rate

Let us denote the variance of the added noise as $\sigma^2$.

Now, it is known that any additive distinguishers (which is the case of our distinguishers), the number of traces to recover the secret for a given success rate is inversely proportional to the inverse of the signal-to-noise ratio (see e.g., [23, Corollary 2]).

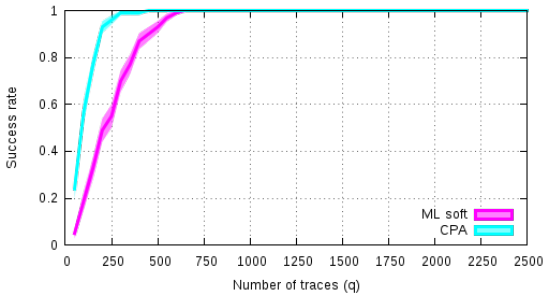
As a direct consequence, we can predict the complexity of the attacks when IC and DC are disabled. It can be seen in Figure 7 that the timing variation is about divided by three (from $\approx$ 20 to $\approx$ 8) when the DC is disabled. Therefore, the number of required traces to recover the key is about multiplied by three.

In addition, we can approximate the required number of traces to extract the key in presence of external noise of standard deviation $\sigma$. In our case-study of OpenSSL AES on ARM, the algorithmic noise has standard deviation about 20 clock cycles (see Figure 1 and 2).

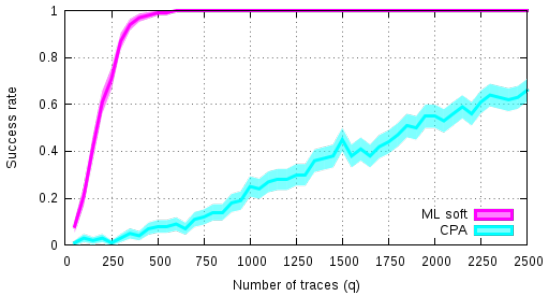
So, if the external noise has standard deviation $\sigma < 20$, the impact is small. But when $\sigma/20 > 1$, the influence of the external noise becomes preponderant. As the algorithmic noise and the external noise are independent, the number of traces required to extract the key will actually grow linearly with $\sigma$ as soon as $\sigma/20 \gg 1$.

### *B. Comparison with Existing Methods in the Presence of Noise*

In this subsection, we aim at comparing our distribution-based method with the existing methods (moment-based method mentioned in Tab. I). In particular, we focus on the representative Bernstein correlation [9] with a learned model [the timing expectation for each value of the target AES byte], that we refer to as "CPA". This "CPA" between timing measurements and the learned average of timing per byte of the key does not suffer from the empty bin issue. We start by a comparison with little external noise. In this case, we have plotted in Figure 19 the success rate for both the soft drop distinguisher and the CPA. The $x$ axis represents the number of traces for the profiling phase while the $y$ axis is the number of traces needed during the attack to reach 80% of success rate. We notice that the CPA performs better than the soft drop method, for any profiling (even when learning with several million of traces). This can be due to bias between the profiled distribution and the attack distribution.




(a) Standard deviation $= 5$




(b) Standard deviation $= 50$

Fig. 20: Success rate for soft drop versus CPA for small noise and noise of standard deviation $T = 50$ (recall (32))

However, in a practical case, we encounter noisy timing leakages. In order to compare our methods with the existing methods (such as CPA) in the presence of external noise, we plotted Figure 20. In this figure, we took a good profiling phase ($\widehat{q} = 3 \times 10^6$), i.e., profiling is performed on sufficiently enough traces. This figure is obtained for a noisy timing, that is the nominal time to compute AES (as in Subsec. II-B), where the noise follows the following law:

$$\begin{cases} 0 & \text{added time with probability 50\%,} \\ T & \text{added } (T \in \mathbb{N}, \text{ a number of clock periods),} \\ & \text{with probability 50\%.} \end{cases} \quad (32)$$

This models the interruption of the CPU from a peripheral when AES is baremetal, or a descheduling of the AES process during one *time slot* on systems with an operating system (OS). Indeed, such events have the consequence, when they occur, to add a long period of time (often as long or even longer than the duration of the AES) to the encryption time, so that the interruption can be served, or so that the OS re-schedules the AES process. We notice that, in such case, it is more interesting to compute one of our methods, rather than previous existing methods such as CPA. Indeed, distribution-based profiling is more accurate than CPA estimation with noisy signals. For instance, the results from Hassan Aly and Mohammed ElGayyar [24] show that $2^{22}$ encryptions are required for a key extraction on a more recent processors (Pentium Dual-Core and Core 2 Duo), which is significantly more than that used by Bernstein CPA in

his original attack [25]. The authors of this paper remark incidentally that the best method is not to use correlation with the *means* of each class, but with the *minimum* value in each class. This confirms that the complexity of the distributions are better suited for distinguishing that simply the average per class. This justifies that our study focuses on distribution-based distinguishers (more robust to binary noise situations encountered while measuring durations) rather than moment-based distinguishers (recall Tab. I).

## VII. Conclusion and Perspectives

We have derived several "information-theoretic" distinguishers as possible solutions to the empty bin issue. Some of them, like the Dirichlet Prior and the Offline-Online distinguishers, required the computation of novel distributions. We have shown in particular that the empty bins, previously believed to be an annoyance and dropped accordingly, can turn out to be valuable assets for the attacker as long as they are treated carefully. In all the paper, real timing data are used, making the results very practical.

We have also compared the various distinguishers under two frameworks: a simulated test with synthetic leakage and real-world timing attacks. In both cases, we noticed that the outcome of the attacks depends on the quality of the profiling stage. A good profiling improves the results, where the best distinguisher seems to be the Soft Drop Distinguisher. A poor profiling makes the traditional distinguishers break down. More sophisticated solutions like Offline-Online Profiling and Learned MIA distinguishers are very useful in this case. A possible way to investigate more on this aspect is to use more powerful statistical tools in order to extract the most precise model for the Learned MIA Distinguisher.

The interesting aspect on the studied timing attack is that one does not have to make any assumption on the leakage model. In addition to this, the main advantage of the new distinguishers is that the empty bin issue is completely solved. We also introduced distinguishers which can jointly exploit offline and online side-channel measurements. As an interesting perspective, our approach could advantageously be analyzed using the "perceived information" metric recently introduced by Standaert et al. in [26, Eqn. (1)].

Another perspective would be to compare our information-theoretic attacks with attacks based on machine learning techniques. Surprisingly and contrary to results reported in other papers, our preliminary results show that SCA based on support vector machines [27] has poor performance, even when profiling with very few traces ($\widehat{q}$ is small), which may be due to the univariate nature of the leakage.

An interesting observation is that writing cryptographic code robust to timing attacks is challenging. While the OpenSSL code for AES has no obvious flaw (such as unbalanced branches which depend on sensitive data), the timing of AES is data-dependent, due to microarchitectural features of the studied ARM core. There seem to exist two classes of solutions against timing attacks: The first aims at randomizing the execution timing, as studied for instance in [6]. Such an implementation can still be attacked with high-order distinguishers, albeit with more traces than without any protection. The second would attempt to balance the timing, yet this requires some hardware support such as the CCM feature of the STM32F4 processors.

*Acknowledgements*

Part of this work has been funded by "Archi-Sec" (Micro-Architectural Security) 2019-2023 Project, within ANR AAP Générique 2019.

## References

[1] J. Daemen and V. Rijmen, *The Design of Rijndael: AES – The Advanced Encryption Standard*. Springer, 2002.

[2] W. Schindler, "A Timing Attack against RSA with the Chinese Remainder Theorem," in *CHES*, ser. Lecture Notes in Computer Science, Ç. K. Koç and C. Paar, Eds., vol. 1965. Springer, 2000, pp. 109–124.

[3] ——, "Optimized timing attacks against public key cryptosystems," *Statistics & Risk Modeling*, vol. 20, no. 1-4, pp. 191–210, 2002, DOI: 10.1524/strm.2002.20.14.191.

[4] D. Brumley and D. Boneh, "Remote Timing Attacks Are Practical," in *Proceedings of the 12th USENIX Security Symposium, Washington, D.C., USA, August 4-8, 2003*. USENIX Association, 2003. [Online]. Available: https://www.usenix.org/conference/12th-usenix-security-symposium/remote-timing-attacks-are-practical

[5] B. B. Brumley and N. Tuveri, "Remote Timing Attacks Are Still Practical," in *ESORICS*, ser. Lecture Notes in Computer Science, V. Atluri and C. Díaz, Eds., vol. 6879. Springer, 2011, pp. 355–371.

[6] J.-L. Danger, N. Debande, S. Guilley, and Y. Souissi, "High-order Timing Attacks," in *Proceedings of the First Workshop on Cryptography and Security in Computing Systems*, ser. CS2 '14. New York, NY, USA: ACM, 2014, pp. 7–12. [Online]. Available: http://doi.acm.org/10.1145/2556315.2556316

[7] S. Chari, J. R. Rao, and P. Rohatgi, "Template Attacks," in *CHES*, ser. LNCS, vol. 2523. Springer, August 2002, pp. 13–28, San Francisco Bay (Redwood City), USA.

[8] P. C. Kocher, "Timing Attacks on Implementations of Diffie-Hellman, RSA, DSS, and Other Systems," in *Advances in Cryptology - CRYPTO '96, Santa Barbara, California, USA, August 18-22, 1996, Proceedings*, ser. LNCS, N. Koblitz, Ed., vol. 1109. Springer, 1996, pp. 104–113. [Online]. Available: http://dx.doi.org/10.1007/3-540-68697-5_9

[9] D. J. Bernstein, "Cache-timing attacks on AES," pp. 1–37, April 14 2005, http://cr.yp.to/antiforgery/cachetiming-20050414.pdf.

[10] C. Rebeiro and D. Mukhopadhyay, "Boosting Profiled Cache Timing Attacks With A Priori Analysis," *Information Forensics and Security, IEEE Transactions on*, vol. 7, no. 6, pp. 1900–1905, 2012.

[11] M. Weiß, B. Heinz, and F. Stumpf, "A cache timing attack on AES in virtualization environments," in *Financial Cryptography and Data Security - 16th International Conference, FC 2012, Kralendijk, Bonaire, Februray 27-March 2, 2012, Revised Selected Papers*, ser. LNCS, A. D. Keromytis, Ed., vol. 7397. Springer, 2012, pp. 314–328. [Online]. Available: http://dx.doi.org/10.1007/978-3-642-32946-3_23

[12] S. Bhattacharya, C. Rebeiro, and D. Mukhopadhyay, "Hardware prefetchers leak: A revisit of SVF for cache-timing attacks," in *45th Annual IEEE/ACM International Symposium on Microarchitecture, MICRO 2012, Workshops Proceedings, Vancouver, BC, Canada, December 1-5, 2012*. IEEE Computer Society, 2012, pp. 17–23. [Online]. Available: http://dx.doi.org/10.1109/MICROW.2012.13

[13] E. Parzen, "On estimation of a probability density function and mode," *Ann. Math. Statist.*, vol. 33, no. 3, pp. 1065–1076, 09 1962. [Online]. Available: http://dx.doi.org/10.1214/aoms/1177704472

[14] F.-X. Standaert, T. Malkin, and M. Yung, "A Unified Framework for the Analysis of Side-Channel Key Recovery Attacks," in *EUROCRYPT*, ser. LNCS, vol. 5479. Springer, April 26-30 2009, pp. 443–461, Cologne, Germany.

[15] A. Heuser, O. Rioul, and S. Guilley, "Good Is Not Good Enough - Deriving Optimal Distinguishers from Communication Theory," in *CHES 2014, Busan, South Korea, September 23-26, 2014. Proceedings*, ser. LNCS, L. Batina and M. Robshaw, Eds., vol. 8731. Springer, 2014, pp. 55–74. [Online]. Available: http://dx.doi.org/10.1007/978-3-662-44709-3_4

[16] B. A. Frigyik, A. Kapila, and M. R. Gupta, "Introduction to the Dirichlet Distribution and Related Processes," Tech. Rep. 206, 2010.

[17] A. Moradi and F. Standaert, "Moments-correlating DPA," *IACR Cryptology ePrint Archive*, vol. 2014, p. 409, June 2 2014. [Online]. Available: http://eprint.iacr.org/2014/409

[18] N. Veyrat-Charvillon and F.-X. Standaert, "Mutual Information Analysis: How, When and Why?" in *CHES*, ser. LNCS, vol. 5747. Springer, September 6-9 2009, pp. 429–443, Lausanne, Switzerland.

[19] B. Gierlichs, L. Batina, P. Tuyls, and B. Preneel, "Mutual information analysis," in *CHES, 10th International Workshop*, ser. Lecture Notes in Computer Science, vol. 5154. Springer, August 10-13 2008, pp. 426–442, Washington, D.C., USA.

[20] ——, "Mutual information analysis," in *CHES, 10th International Workshop*, ser. LNCS, vol. 5154. Springer, August 10-13 2008, pp. 426–442, Washington, D.C., USA.

[21] S. Microelectronics, "STM32F4DISCOVERY Discovery kit with STM32F407VG MCU," http://www.st.com/web/catalog/tools/FM116/SC959/SS1532/PF252419?sc=internet/evalboard/product/252419.jsp [Accessed March 19, 2016].

[22] NIST, "AES Proposal: Rijndael (now FIPS PUB 197)," 9 April 2003, http://csrc.nist.gov/archive/aes/rijndael/Rijndael-ammended.pdf.

[23] S. Guilley, A. Heuser, and O. Rioul, "A Key to Success - Success Exponents for Side-Channel Distinguishers," in *Progress in Cryptology - INDOCRYPT 2015 - 16th International Conference on Cryptology in India, Bangalore, India, December 6-9, 2015, Proceedings*, ser. Lecture Notes in Computer Science, A. Biryukov and V. Goyal, Eds., vol. 9462. Springer, 2015, pp. 270–290. [Online]. Available: http://dx.doi.org/10.1007/978-3-319-26617-6_15

[24] H. Aly and M. ElGayyar, "Attacking AES Using Bernstein's Attack on Modern Processors," in *Progress in Cryptology - AFRICACRYPT 2013, 6th International Conference on Cryptology in Africa, Cairo, Egypt, June 22-24, 2013. Proceedings*, ser. Lecture Notes in Computer Science, A. Youssef, A. Nitaj, and A. E. Hassanien, Eds., vol. 7918. Springer, 2013, pp. 127–139. [Online]. Available: https://doi.org/10.1007/978-3-642-38553-7_7

[25] D. J. Bernstein, "Cache-timing attacks on AES," April 2005, http://cr.yp.to/antiforgery/cachetiming-20050414.pdf.

[26] M. Renauld, D. Kamel, F.-X. Standaert, and D. Flandre, "Information Theoretic and Security Analysis of a 65-Nanometer DDSLL AES S-Box," in *CHES*, ser. LNCS, B. Preneel and T. Takagi, Eds., vol. 6917. Springer, 2011, pp. 223–239.

[27] A. Heuser and M. Zohner, "Intelligent Machine Homicide - Breaking Cryptographic Devices Using Support Vector Machines," in *COSADE*, ser. LNCS, W. Schindler and S. A. Huss, Eds., vol. 7275. Springer, 2012, pp. 249–264.


## APPENDIX

We have copied here the OpenSSL C code for the encryption function. We notice that this is a straightline code, and that there is a use of Look Up Tables (the T boxes) that may cause the non constant time.

```
void AES_encrypt(const unsigned char *in, unsigned char *out,
    const AES_KEY *key) {

 const u32 *rk;
 u32 s0, s1, s2, s3, t0, t1, t2, t3;

 int r;



# 796 "aes_core.c" 3
 ((void)0)
# 796 "aes_core.c"
                    ;
 rk = key->rd_key;




 s0 = (((u32)(in)[0] << 24) ^ ((u32)(in)[1] << 16) ^ ((u32)(in)[2] << 8) ^ ((u32
       )(in)[3])) ^ rk[0];
 s1 = (((u32)(in + 4)[0] << 24) ^ ((u32)(in + 4)[1] << 16) ^ ((u32)(in + 4)[2]
       << 8) ^ ((u32)(in + 4)[3])) ^ rk[1];
 s2 = (((u32)(in + 8)[0] << 24) ^ ((u32)(in + 8)[1] << 16) ^ ((u32)(in + 8)[2]
       << 8) ^ ((u32)(in + 8)[3])) ^ rk[2];
 s3 = (((u32)(in + 12)[0] << 24) ^ ((u32)(in + 12)[1] << 16) ^ ((u32)(in + 12)
       [2] << 8) ^ ((u32)(in + 12)[3])) ^ rk[3];


    t0 = Te0[(s0 >> 24)] ^ Te1[(s1 >> 16) & 0xff] ^ Te2[(s2 >> 8) & 0xff] ^ Te3[
          s3 & 0xff] ^ rk[ 4];
    t1 = Te0[s1 >> 24] ^ Te1[(s2 >> 16) & 0xff] ^ Te2[(s3 >> 8) & 0xff] ^ Te3[s0
          & 0xff] ^ rk[ 5];
    t2 = Te0[s2 >> 24] ^ Te1[(s3 >> 16) & 0xff] ^ Te2[(s0 >> 8) & 0xff] ^ Te3[s1
          & 0xff] ^ rk[ 6];
    t3 = Te0[s3 >> 24] ^ Te1[(s0 >> 16) & 0xff] ^ Te2[(s1 >> 8) & 0xff] ^ Te3[s2
          & 0xff] ^ rk[ 7];

    s0 = Te0[t0 >> 24] ^ Te1[(t1 >> 16) & 0xff] ^ Te2[(t2 >> 8) & 0xff] ^ Te3[t3
          & 0xff] ^ rk[ 8];
    s1 = Te0[t1 >> 24] ^ Te1[(t2 >> 16) & 0xff] ^ Te2[(t3 >> 8) & 0xff] ^ Te3[t0
          & 0xff] ^ rk[ 9];
    s2 = Te0[t2 >> 24] ^ Te1[(t3 >> 16) & 0xff] ^ Te2[(t0 >> 8) & 0xff] ^ Te3[t1
          & 0xff] ^ rk[10];
    s3 = Te0[t3 >> 24] ^ Te1[(t0 >> 16) & 0xff] ^ Te2[(t1 >> 8) & 0xff] ^ Te3[t2
          & 0xff] ^ rk[11];

    t0 = Te0[s0 >> 24] ^ Te1[(s1 >> 16) & 0xff] ^ Te2[(s2 >> 8) & 0xff] ^ Te3[s3
          & 0xff] ^ rk[12];
    t1 = Te0[s1 >> 24] ^ Te1[(s2 >> 16) & 0xff] ^ Te2[(s3 >> 8) & 0xff] ^ Te3[s0
          & 0xff] ^ rk[13];
    t2 = Te0[s2 >> 24] ^ Te1[(s3 >> 16) & 0xff] ^ Te2[(s0 >> 8) & 0xff] ^ Te3[s1
          & 0xff] ^ rk[14];
    t3 = Te0[s3 >> 24] ^ Te1[(s0 >> 16) & 0xff] ^ Te2[(s1 >> 8) & 0xff] ^ Te3[s2
          & 0xff] ^ rk[15];

    s0 = Te0[t0 >> 24] ^ Te1[(t1 >> 16) & 0xff] ^ Te2[(t2 >> 8) & 0xff] ^ Te3[t3
          & 0xff] ^ rk[16];
    s1 = Te0[t1 >> 24] ^ Te1[(t2 >> 16) & 0xff] ^ Te2[(t3 >> 8) & 0xff] ^ Te3[t0
          & 0xff] ^ rk[17];
    s2 = Te0[t2 >> 24] ^ Te1[(t3 >> 16) & 0xff] ^ Te2[(t0 >> 8) & 0xff] ^ Te3[t1
          & 0xff] ^ rk[18];
    s3 = Te0[t3 >> 24] ^ Te1[(t0 >> 16) & 0xff] ^ Te2[(t1 >> 8) & 0xff] ^ Te3[t2
          & 0xff] ^ rk[19];

    t0 = Te0[s0 >> 24] ^ Te1[(s1 >> 16) & 0xff] ^ Te2[(s2 >> 8) & 0xff] ^ Te3[s3
          & 0xff] ^ rk[20];
    t1 = Te0[s1 >> 24] ^ Te1[(s2 >> 16) & 0xff] ^ Te2[(s3 >> 8) & 0xff] ^ Te3[s0
          & 0xff] ^ rk[21];
    t2 = Te0[s2 >> 24] ^ Te1[(s3 >> 16) & 0xff] ^ Te2[(s0 >> 8) & 0xff] ^ Te3[s1
          & 0xff] ^ rk[22];
    t3 = Te0[s3 >> 24] ^ Te1[(s0 >> 16) & 0xff] ^ Te2[(s1 >> 8) & 0xff] ^ Te3[s2
          & 0xff] ^ rk[23];

    s0 = Te0[t0 >> 24] ^ Te1[(t1 >> 16) & 0xff] ^ Te2[(t2 >> 8) & 0xff] ^ Te3[t3
          & 0xff] ^ rk[24];
    s1 = Te0[t1 >> 24] ^ Te1[(t2 >> 16) & 0xff] ^ Te2[(t3 >> 8) & 0xff] ^ Te3[t0
          & 0xff] ^ rk[25];
    s2 = Te0[t2 >> 24] ^ Te1[(t3 >> 16) & 0xff] ^ Te2[(t0 >> 8) & 0xff] ^ Te3[t1
          & 0xff] ^ rk[26];
    s3 = Te0[t3 >> 24] ^ Te1[(t0 >> 16) & 0xff] ^ Te2[(t1 >> 8) & 0xff] ^ Te3[t2
          & 0xff] ^ rk[27];

    t0 = Te0[s0 >> 24] ^ Te1[(s1 >> 16) & 0xff] ^ Te2[(s2 >> 8) & 0xff] ^ Te3[s3
          & 0xff] ^ rk[28];
    t1 = Te0[s1 >> 24] ^ Te1[(s2 >> 16) & 0xff] ^ Te2[(s3 >> 8) & 0xff] ^ Te3[s0
          & 0xff] ^ rk[29];
    t2 = Te0[s2 >> 24] ^ Te1[(s3 >> 16) & 0xff] ^ Te2[(s0 >> 8) & 0xff] ^ Te3[s1
          & 0xff] ^ rk[30];
    t3 = Te0[s3 >> 24] ^ Te1[(s0 >> 16) & 0xff] ^ Te2[(s1 >> 8) & 0xff] ^ Te3[s2
          & 0xff] ^ rk[31];

    s0 = Te0[t0 >> 24] ^ Te1[(t1 >> 16) & 0xff] ^ Te2[(t2 >> 8) & 0xff] ^ Te3[t3
          & 0xff] ^ rk[32];
    s1 = Te0[t1 >> 24] ^ Te1[(t2 >> 16) & 0xff] ^ Te2[(t3 >> 8) & 0xff] ^ Te3[t0
          & 0xff] ^ rk[33];
    s2 = Te0[t2 >> 24] ^ Te1[(t3 >> 16) & 0xff] ^ Te2[(t0 >> 8) & 0xff] ^ Te3[t1
          & 0xff] ^ rk[34];
    s3 = Te0[t3 >> 24] ^ Te1[(t0 >> 16) & 0xff] ^ Te2[(t1 >> 8) & 0xff] ^ Te3[t2
          & 0xff] ^ rk[35];

    t0 = Te0[s0 >> 24] ^ Te1[(s1 >> 16) & 0xff] ^ Te2[(s2 >> 8) & 0xff] ^ Te3[s3
          & 0xff] ^ rk[36];
    t1 = Te0[s1 >> 24] ^ Te1[(s2 >> 16) & 0xff] ^ Te2[(s3 >> 8) & 0xff] ^ Te3[s0
          & 0xff] ^ rk[37];
    t2 = Te0[s2 >> 24] ^ Te1[(s3 >> 16) & 0xff] ^ Te2[(s0 >> 8) & 0xff] ^ Te3[s1
          & 0xff] ^ rk[38];
    t3 = Te0[s3 >> 24] ^ Te1[(s0 >> 16) & 0xff] ^ Te2[(s1 >> 8) & 0xff] ^ Te3[s2
          & 0xff] ^ rk[39];
    rk += key->rounds << 2;
# 944 "aes_core.c"
 s0 =
  (Te2[(t0 >> 24) ] & 0xff000000) ^
  (Te3[(t1 >> 16) & 0xff] & 0x00ff0000) ^
  (Te0[(t2 >> 8) & 0xff] & 0x0000ff00) ^
  (Te1[(t3 ) & 0xff] & 0x000000ff) ^
  rk[0];
 { (out)[0] = (u8)((s0) >> 24); (out)[1] = (u8)((s0) >> 16); (out)[2] = (u8)((s0
       ) >> 8); (out)[3] = (u8)(s0); };
 s1 =
  (Te2[(t1 >> 24) ] & 0xff000000) ^
  (Te3[(t2 >> 16) & 0xff] & 0x00ff0000) ^
  (Te0[(t3 >> 8) & 0xff] & 0x0000ff00) ^
  (Te1[(t0 ) & 0xff] & 0x000000ff) ^
  rk[1];
 { (out + 4)[0] = (u8)((s1) >> 24); (out + 4)[1] = (u8)((s1) >> 16); (out + 4)
       [2] = (u8)((s1) >> 8); (out + 4)[3] = (u8)(s1); };
 s2 =
  (Te2[(t2 >> 24) ] & 0xff000000) ^
  (Te3[(t3 >> 16) & 0xff] & 0x00ff0000) ^
  (Te0[(t0 >> 8) & 0xff] & 0x0000ff00) ^
  (Te1[(t1 ) & 0xff] & 0x000000ff) ^
  rk[2];
 { (out + 8)[0] = (u8)((s2) >> 24); (out + 8)[1] = (u8)((s2) >> 16); (out + 8)
       [2] = (u8)((s2) >> 8); (out + 8)[3] = (u8)(s2); };
 s3 =
  (Te2[(t3 >> 24) ] & 0xff000000) ^
  (Te3[(t0 >> 16) & 0xff] & 0x00ff0000) ^
  (Te0[(t1 >> 8) & 0xff] & 0x0000ff00) ^
  (Te1[(t2 ) & 0xff] & 0x000000ff) ^
  rk[3];
 { (out + 12)[0] = (u8)((s3) >> 24); (out + 12)[1] = (u8)((s3) >> 16); (out +
       12)[2] = (u8)((s3) >> 8); (out + 12)[3] = (u8)(s3); };
}
```