First-Order Side-Channel Leakage Analysis of Masked but Asynchronous AES

Antoine Bouvet, Sylvain Guilley, Lukas Vlasak

First-Order Side-Channel Leakage Analysis of Masked but Asynchronous AES

Antoine Bouvet[1][0000−0002−4364−8371], Sylvain Guilley[1,2][0000−0002−5044−3534], and Lukas Vlasak[1][0000−0003−1141−2388]

[1] Secure-IC S.A.S., Think Ahead Business Line, 35510 Cesson-Sévigné, France
firstname.lastname@secure-ic.com
[2] Télécom ParisTech, 91120 Palaiseau, France
firstname.lastname@telecom-paristech.fr

Abstract. Masking schemes are classical countermeasures against Side-Channel Attacks on cryptographic implementations. This paper investigates the effectiveness of masking when the code does not run in constant time. We prove that in this case, a first-order Correlation Power Analysis can break an otherwise perfect masking scheme. Furthermore, with an in-depth leakage analysis on traces generated at a pre-silicon stage, we pinpoint the leaking instructions and recover a complex leakage model.

Keywords: Side-Channel Analysis · Masking scheme · Leakage model · Desynchronisation · AES · White-Box · Pre-silicon evaluation

1 Introduction

Cryptographic algorithms are known to be sensitive to physical attacks. While theoretically proven to resist algebraic cryptanalysis, the actual implementations of such an algorithm may leak information about a parameter through its observable physical behavior when running on an end-user device. The parameters which must be kept secret are called Critical Security Parameters (CSPs). The computation of an intermediate value, which depends on these CSPs, leads to a statistically biased activity on the device (*e.g.* its power consumption [23], its electro-magnetic (EM) emanations [17], the execution time [21] or the emitted acoustic waves [18]), which can be measured, recorded and analysed. This is called a Side-Channel Analysis (SCA). Such an analysis can be performed on subkeys, small parts of a CSP, in order to reduce the strength of a cryptosystem (a *Divide-and-Conquer* strategy). Even if not all subkeys have been broken, it may be weakened enough to accomplish a brute-force attack. Fortunately, many countermeasures exist, such as noise addition [12], [20, §2.1], constant time operations [1], shuffling [29] or masking schemes [5, 27]. Each one aims at protecting against specific attacks – for instance, constant time operations resist against Timing Attacks, while masking schemes are effective against vertical SCAs. That is why developers and designers do not use only one countermeasure when designing their software, but a combination of them. The goal is always to secure a device against as many attacks as possible.

Contributions. We present how a first-order SCA such as Differential Power Analysis (DPA) [23] or Correlation Power Analysis (CPA) [8] can break a (possibly higher-order) masking scheme. Normally the usage of an n^{th}-order masking scheme countermeasure protects a device against an n^{th}-order SCA (no subkey can be broken). Nevertheless, we show that instead of increasing the cryptosystem's security, the combination of masking and desynchronisations significantly improves the attack efficiency, allowing to break the whole secret key. More precisely, we prove that, if a Boolean masking scheme is used in combination with asynchronous operations that depend on a CSP, then the masking is completely ineffective. Moreover, our pre-silicon security White-Box evaluation approach, based on traces generated on CPU, allows an in-depth analysis of the leakage source, including the derivation of the optimal leakage model.
Outline. Boolean masking schemes against vertical SCAs and possible reasons for non-constant time cryptographic code are explained in Sec. 2. Our contributions start in Sec. 3; we outline theoretical examples of first-order SCAs against masking schemes. Side-Channel Analyses on pre-silicon traces are carried out in Sec. 4: the leakage is characterised and its source identified. Section 5 discusses general aspects about our results. Eventually, conclusions are given in Sec. 6.

2 Boolean masking schemes against vertical SCAs

Masking techniques aim to protect implementations against vertical SCAs by avoiding statistical dependencies between CSPs and the processed values, which influence the observable side-channel activity. **Boolean masking** [11,19,24] uses an *exclusive or* (XOR, $\oplus$). For a secret parameter $x \in \mathbb{F}_2^N$, and $d \in \mathbb{N}$ randomly chosen masks $x_1, ..., x_d \in \mathbb{F}_2^N$, every possible masked value $x_0 = x \oplus x_1 \oplus \ldots \oplus x_d$ has the same probability ($\frac{1}{2^N}$). This widely used masking scheme avoids first-order SCAs and increases the resistance against SCAs of arbitrary order, when $d > 1$.

One could argue that masked implementations ought be designed "constant time" in the first place. This is a good practice we do recommend. We want to underline that there are reasons for some developers – who are generally not security experts – to implement non-constant masked code:

- Blinding for RSA [22] works even if the implementation is non-constant time, but this countermeasure does not intend to protect against vertical SCA.
- One might think randomization would protect the data sufficiently also against Timing Attacks. This is untrue, as shown in [13]. In the next Sec. 3, we will show that even vertical SCAs can be applied on masked code which does not run in constant time.
- A programmer may not be aware that he is implementing non-constant time code. For instance, an AES with tabulated substitution boxes is not constant time, as the lookup time can depend on the address (see [4]). Examples of potentially vulnerable codes are [10, 15, 16, 31], which use tables addressed by sensitive variables. Furthermore, [15,16] resort to conditional control flow in $\mathbb{F}_{256}$ multiplication.

3 First-order vertical SCAs against masking schemes

Usually, vertical SCAs require that the activity traces are synchronised – the target nodes are temporally aligned according to a *clock period of interest*. In the case of the AES, common target nodes are the first round S-Box output $S(x_i \oplus k_i)$, where S is the S-Box, x_i and k_i are the i^{th} plaintext and key bytes respectively, or the last round S-Box input $S^{-1}(c_i \oplus k_i)$, where S^{-1} is the inverse S-Box, and c_i is the i^{th} ciphertext byte. In a protected algorithm with first-order Boolean masking, the first round's target node becomes $S(x_i \oplus k_i) \oplus M$, where a random byte value M is used as mask, hence even under the correct key hypothesis the value is unpredictable by an attacker. The current situation can be summarised in the following assertions:

- vertical SCAs are very effective against unprotected algorithms;
- desynchronisations (intentional or not) increase the difficulty of vertical SCA;
- when a cryptographic algorithm is implemented with a first-order (or higher-order) masking scheme, first-order CPAs do fail.

However, we highlight hereafter that *asynchronous* (*i.e.*, non-constant time) activity traces may lead to a side-channel leakage in a masked implementation.

Our study deals with an asynchronous, masked AES (defined in Alg. 1).

Input : $plaintext$, key, $(mi, mo) \in \mathbb{F}_{2^8}^2$, $Nr \in \{10, 12, 14\}$
Output : $ciphertext$

1 $KeySchedule = \texttt{KeyExpansion}(key, Nr)$ // Precomputation of the Nr round keys
2 **for** $byte = 0$ *to* 255 **do**
3 $\quad MS[byte] = S[byte \oplus mi] \oplus mo$ // Precomputation of the masked S-Box
4 $mr = mi \oplus mo$ // Precomputation of the mask refresh
5 **for** $i = 0$ *to* 15 **do**
6 $\quad state[i] = plaintext[i] \oplus mi$ // Masking the state
7 $\texttt{AddRoundKey}(state, KeySchedule, 0)$
8 **for** $round = 1$ *to* $Nr - 1$ **do**
9 $\quad \texttt{ShiftRows}(state)$
10 $\quad \texttt{SubBytes}(state, MS)$
11 $\quad \texttt{MixColumns}(state)$
12 $\quad \texttt{RefreshMask}(state, mr)$ // $\forall i \in \{0, \ldots, 15\}$, $state[i] \oplus= mr$
13 $\quad \texttt{AddRoundKey}(state, KeySchedule, round)$
14 $\texttt{ShiftRows}(state)$
15 $\texttt{SubBytes}(state)$
16 $\texttt{AddRoundKey}(state, KeySchedule, Nr)$
17 **for** $i = 0$ *to* 15 **do**
18 $\quad ciphertext[i] = state[i] \oplus mo$ // Unmasking the state
19 **return** $ciphertext$

Algorithm 1: Main ciphering function of the masked AES.

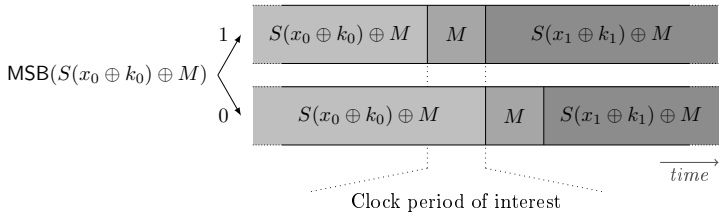


Fig. 1. Misaligned traces and the corresponding leakage model.

The desynchronisation occurs in the *xtime* sub-function which is used in *MixColumns*, to multiply a polynomial $b \in \mathbb{F}_2[x]/\langle x^8 + x^4 + x^3 + x + 1\rangle$ with the monomial x. A pseudo-code of its naive implementation is given in Alg. 2. As there is a conditional branch which depends on the Most Significant Bit (MSB) of the S-Box output, one gets a parameter-dependent misalignment between the traces. This can be exploited with a high-order Timing Analysis [13] and, as we show, it also opens the door to a first-order vertical SCA.

Input : $b \in \mathbb{F}_2[x]/\langle x^8 + x^4 + x^3 + x + 1\rangle$
Output : $res = b \times x$

1 $res \leftarrow b \ll 1$ // Multiplication by x
2 **if** $b \wedge$ *0x80* **then**
3 $res \leftarrow res \oplus$ 0x11b // Conditional reduction
4 **return** res

Algorithm 2: Naive *xtime* (insecure – with conditional branching).

The figure 1 illustrates the two paths of execution, depending on line 3 of Alg. 2 is executed or not. During the *clock period of interest*, if the MSB of the masked S-Box output is 1, the S-Box evaluation is over and the mask is manipulated, whereas if it is 0, at the same time the S-Box evaluation is still ongoing.

Let $i \in \{0, \ldots, 15\}$ and $j \in \{0, \ldots, 7\}$. For the rest of the paper, the notation S^i refers to the S-Box output $S(x_i \oplus k_i)$, and S^i_j to its j^{th} bit. Then the leakage model can be expressed as follows:

$$\begin{aligned} L(x_0, k_0, M) &= \begin{cases} M & \text{if } \mathsf{MSB}(S^0 \oplus M) = 1, \\ S^0 \oplus M & \text{otherwise.} \end{cases} \\ &= \mathsf{MSB}(S^0 \oplus M) \times M \\ &\quad + (1 - \mathsf{MSB}(S^0 \oplus M)) \times (S^0 \oplus M). \end{aligned} \tag{1}$$

The *optimal model* can be derived by averaging $L(x_0, k_0, M)$ over M [28]. If this model depends on the sensitive variable $x_0 \oplus k_0$, then a key extraction is possible.

The optimal model computes as follows:

$$\begin{aligned}
\ell(x_0, k_0) =& \mathbb{E}_M(L(x_0, k_0, M)) \\
=& \mathbb{E}_M(\mathsf{MSB}(S^0 \oplus M) \times M) + \mathbb{E}_M((1 - \mathsf{MSB}(S^0 \oplus M)) \times (S^0 \oplus M)) \\
=& \frac{1}{256} \sum_{m=0}^{255} (\mathsf{MSB}(S^0 \oplus m) \times m) \\
&+ \frac{1}{256} \sum_{m=0}^{255} ((1 - \mathsf{MSB}(S^0 \oplus m)) \times (S^0 \oplus m)) \\
=& \frac{1}{256} \sum_{m=0}^{255} \mathsf{MSB}(S^0 \oplus m) \times m \qquad (2) \\
&+ \frac{1}{256} \sum_{m'=0}^{255} (1 - \mathsf{MSB}(m')) \times (m'). \qquad (3)
\end{aligned}$$

using variable change $m' = m \oplus S(x_0 \oplus k_0)$. Notice that (2) does depend on the key, and the term (3) is a constant, thus it can be dropped.

Now, let m_i be the i^{th} bit of $m \in \mathbb{F}_2^8$:

$$\sum_{m=0}^{255} \mathsf{MSB}(S^0 \oplus m) \times m = \sum_{m \in \{0,1\}^8} (S_7^0 \oplus m_7) \times (m_7, \ldots, m_0).$$

For any $i \in \{6, \ldots, 0\}$:

$$\begin{aligned}
\frac{1}{256} \sum_{m \in \{0,1\}^8} (S_7^0 \oplus m_7) \times m_i =& \frac{64}{256} \sum_{(m_7, m_i) \in \{0,1\}^2} (S_7^0 \oplus m_7) \times m_i \\
=& \frac{64}{256} \sum_{m_7 \in \{0,1\}} (S_7^0 \oplus m_7) \\
=& \frac{64}{256} (S_7^0 + (1 - S_7^0)) = \frac{1}{4},
\end{aligned}$$

and for the particular case $i = 7$:

$$\frac{1}{256} \sum_{m \in \{0,1\}^8} (S_7^0 \oplus m_7) \times m_i = \frac{128}{256} \sum_{m_7 \in \{0,1\}} (S_7^0 \oplus m_7) \times m_7 = \frac{128}{256} (1 - S_7^0),$$

hence we can conclude:

$$\ell(x_0, k_0)_7 = \frac{1}{2} (1 - \mathsf{MSB}(S(x_0 \oplus k_0))). \qquad (4)$$

It turns out that the optimal model for a SCA on Fig. 1 is simply the selection signal of the unmasked target node. Correlation coefficients between measures and optimal model (4) are shown in Fig. 2, the correct key can easily be recovered.

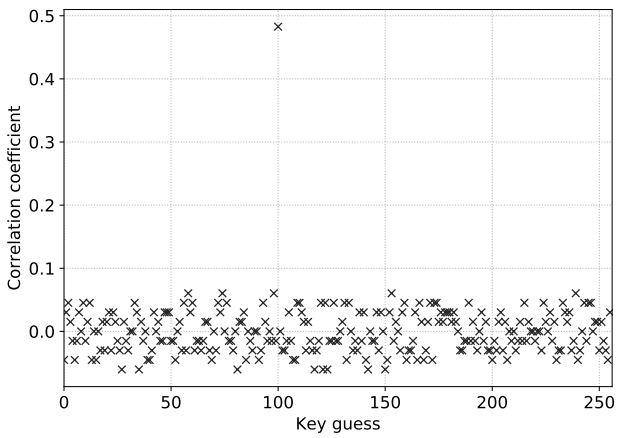


Fig. 2. CPA coefficients obtained for leakage (1) and leakage model (4) (correct key is $k_0 = 100$) – Ideal setup of Fig. 1.

However, if the traces are realigned, the leakage model does not depend on the MSB. Therefore we confirm that the code is first-order perfectly masked. Interestingly, this leakage would not have been detected by a static analysis method, such as the one of Barthe *et al.* [2], because this approach is unaware of timing issues between executions with different data.

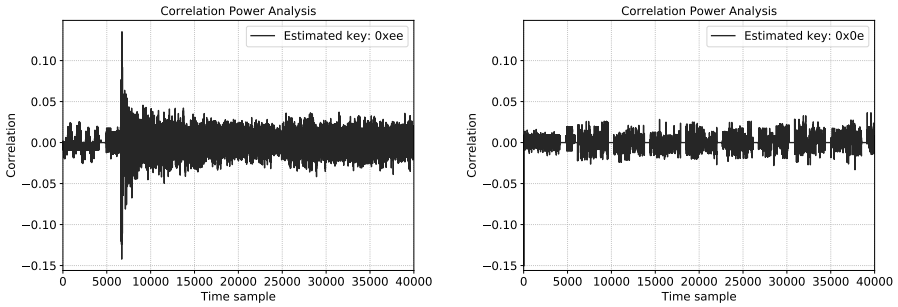
4 Experiments on a real-world AES code

In Section 3, a theoretical example of leakages due to the combination of masking and desynchronisation, has been studied. In the present section, we consider a real use-case.

4.1 Target agnostic analysis on CPU

Traces generation. In order to pinpoint such leakages, we analyse at a pre-silicon stage, a masked AES-128 which uses the *xtime* structure presented in Alg. 2, through a target agnostic approach as introduced in [6]. The implementation is protected with first-order Boolean masking, which theoretically resists first-order SCAs. Static leakage detection tools, such as [2], cannot detect the vulnerabilities we encounter (since they do not have a notion of time). Dynamic tools, such as SLEAK [30], can be used in this respect.

Our traces are generated at a binary level – on a CPU, using a debugger (`gdb`) as measurement tool which records the values of multiple registers –, while running on a 64-bit `x86` architecture. That is why in the following sections, bit indices of a byte can exceed 7 and extend up to 63. We need to record only



(a) Asynchronous AES — leakage (b) Constant time AES — no leakage

Fig. 3. CPA results with (Naive leakage model) on a masked AES-128, with desynchronisation (3a) and without (3b). Both target subkey $k_0 =$ `0xee`, with the same number of traces. The subkey is broken only in case of desynchronisation.

Input : $b \in \mathbb{F}_2[x]/\langle x^8 + x^4 + x^3 + x + 1\rangle$
Output : $res = b \times x$

1 **return** $(b \ll 1) \oplus (((b \gg 7) \wedge 1) \times$ `0x11b`$)$

Algorithm 3: Constant time *xtime*.

the RAX[3] register values. In such a White-Box context, we can map each RAX sample with its address, and are therefore able to find the corresponding line of the C source code [7] (leveraging DWARF format).

Naive analysis. We use the Pearson's correlation as distinguisher and target the AES first round using the Hamming Weight leakage model:

$$\ell_N(x_0, k_0) = \texttt{HW}(S(x_0 \oplus k_0)) \qquad \text{(Naive leakage model)}$$

where `HW` refers to the *Hamming Weight* function. Obviously, such analysis is not optimal, since the AES traces are not synchronised on the targeted operation sample. Nevertheless, one can easily break the secret key as if no mask was used, as shown in Fig. 3a. One may think that the implemented masking scheme is not really effective (*e.g.* in reality some sensitive intermediate value is not masked), however the countermeasure itself is well implemented. The linearity of the *xtime* function in $\mathbb{F}_2^8$ guarantees that the sensitive values stay perfectly masked. Besides, when repeating the same analysis on a constant time version described by Alg. 3, there is no first-order leakage (Fig. 3b). The AES is perfectly masked end-to-end, according to the requirement of Blömer *et al.* [5]. This observation is consistent with the situation in Sec. 3, that a protected AES leaks in first-order, when the

[3] RAX is the name of the accumulator register in `x86` assembly.

Table 1. Classes of each of the 2^5 possible combinations for $\mathcal{C}$.

$\mathcal{L}_1$	$\mathcal{L}_2$	$\mathcal{L}_3$	$\mathcal{L}_4$	$\mathcal{L}_5$	$\mathcal{L}_6$
$\{0;0;1;1;1\}$	$\{1;1;1;1;-\}$	$\{0;1;1;0;-\}$	$\{0;0;1;1;0\}$	$\{0;1;0;-;-\}$	$\{1;1;0;-;-\}$
		$\{1;0;0;1;-\}$	$\{0;0;1;0;-\}$	$\{0;1;1;1;-\}$	$\{1;1;1;0;-\}$
			$\{0;0;0;-;-\}$	$\{1;0;1;1;-\}$	
				$\{1;0;-;0;-\}$	

traces are misaligned. Furthermore, it should be noted that the constant time *xtime* (Alg. 3) increases the computation time by 46%, compared to Alg. 2. This again explains a possible motivation for non-constant solutions in practice.

Real leakage model. In White-Box evaluations on CPU, we can map each time sample with its source code. We can narrow this analysis down to the Points of Interest (PoIs) – the samples with the highest Pearson's correlation. We pinpoint all the used addresses, which can differ from one trace to another. If the situation matches with the theoretical one (Sec. 3), we should get two addresses (as Fig. 1 shows two execution paths at the clock period of interest). But actually, more than two addresses correspond to the analysed sample, therefore the leakage model is far more complicated than expected. After mapping the assembly instructions to the corresponding algorithmic values, we want to understand the conditions leading to these values. In fact, in our case, each call to the *xtime* function creates a misalignment of up to 2 samples, depending only on the MSB of *xtime* input. This leads to a global misalignment of up to 38 samples when targeting k_0, depending on the following MSB:

$$\mathcal{C} = \{\mathsf{MSB}(M); \mathsf{MSB}(\mathsf{xtime}(M)); \mathsf{MSB}(a); \mathsf{MSB}(\mathsf{xtime}(a)); \mathsf{MSB}(d)\}$$

where $a = S(x_0 \oplus k_0) \oplus M$, and $d = S(x_{15} \oplus k_{15}) \oplus M$. By classifying the traces according to this condition $\mathcal{C}$, we observe six different classes $\mathcal{L}_1, \ldots, \mathcal{L}_6$, at the leaking sample (defined in Table 1). As a consequence, the real leakage model $\hat{L} = L(x_0, k_0, x_{15}, k_{15}, M)$, outlined in Fig. 4, can be expressed as follows:

$$\hat{L} = \begin{cases} 1 & \text{if } \mathcal{C} \in \mathcal{L}_1, \\ 0 & \text{if } \mathcal{C} \in \mathcal{L}_2, \\ d \ll 1 & \text{if } \mathcal{C} \in \mathcal{L}_3, \\ \mathsf{xtime}(d) & \text{if } \mathcal{C} \in \mathcal{L}_4, \\ d & \text{if } \mathcal{C} \in (\mathcal{L}_5 \cup \mathcal{L}_6). \end{cases} \tag{5}$$

Fig. 4 also highlights that there are six execution paths. They happen not to be of the same duration. However, if they were, Timing Analyses [4] would not be possible, but the first-order SCA could still be carried out in the same way. Interestingly, we can check the consistency of this leakage model by comparing the theoretical and the experimental leaking values distributions.

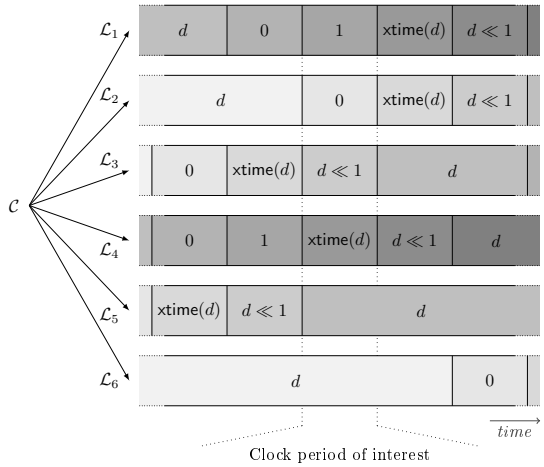


Fig. 4. Symbolic values in the registers, depending on the condition $\mathcal{C}$.

Indeed, over $2^5 = 32$ possible combinations (Table 1):

- $\mathcal{L}_1$ (symbolic value: 1) contains only 1 combination;
- $\mathcal{L}_2$ (symbolic value: 0) clusters 2 combinations;
- $\mathcal{L}_3$ (symbolic value: $d \ll 1$) clusters 4 combinations;
- $\mathcal{L}_4$ (symbolic value: $\mathsf{xtime}(d)$) clusters 7 combinations;
- $\mathcal{L}_5$ and $\mathcal{L}_6$ (symbolic value: d) cluster respectively $12+6 = 18$ combinations.

As shown in Fig. 5, the experimental distribution is close to the theoretical one, which validates this leakage model as realistic.

4.2 Optimal leakage model

Note that the former CPA breaks subkey k_0 (Fig. 3a) by algorithmic values which never depend on x_0 or k_0 (Fig. 4). Therefore, below we compute the optimal leakage model $\ell(x_0, k_0)$ by deriving the real leakage model $\hat{L}$, which is deduced from symbolic values stored in the registers.

Lemma 1 (Expression of the optimal leakage model). *Let $i \in \mathbb{N}_{<64}$. The optimal leakage model $\ell_{O,i} = \ell_O(x_0, k_0)_i$ for the i^{th} bit of $\hat{L}$ expresses as follows:*

$$\ell_{O,0} = \tfrac{1}{2^4}\begin{pmatrix} 2S_7^0S_6^0 & + \\ 2^2(\neg S_7^0) & + \\ 2^2S_7^0(\neg S_6^0) & + \\ 2S_7^0 & + \\ 2(\neg S_6^0) & + \\ 2^2(\neg S_7^0)S_6^0 & \end{pmatrix}, \ell_{O,i\in\{1,\ldots,7\}} = \tfrac{1}{2^4}\begin{pmatrix} (2^2+1)S_7^0S_6^0 & + \\ 2^2(\neg S_7^0) & + \\ 2^2S_7^0(\neg S_6^0) & + \\ 2S_7^0 & + \\ 2(\neg S_6^0) & + \\ 2^2(\neg S_7^0)S_6^0 & \end{pmatrix},$$

$$\ell_{O,8} = \tfrac{1}{2^4}\begin{pmatrix} 2^2S_7^0S_6^0 & + \\ 2(\neg S_7^0) & + \\ 2S_7^0(\neg S_6^0) & \end{pmatrix}, \quad \ell_{O,i\in\{9,\ldots,63\}} = 0.$$

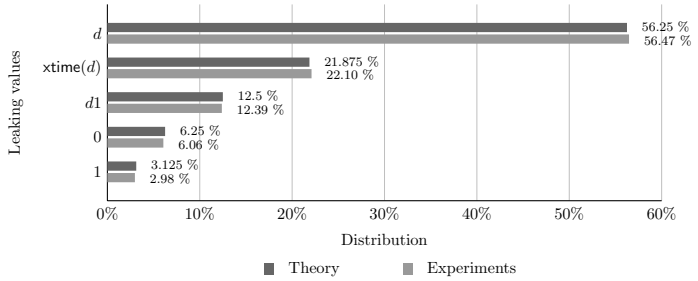


Fig. 5. Leaking values distribution for the leakage model: theory *v.s.* experiments (based on 9,356 execution traces).

Proof. First, we are concerned with x_0, in order to guess k_0 since the leaking values and the different combinations are functions of x_0, k_0, x_{15} and k_{15}. However, we disregard k_{15} which is constant, and x_{15}. Moreover, M is unknown, hence $\ell_O(x_0, k_0) = \mathbb{E}_{M,d}(\hat{L})$. While $\hat{L}$ depends on specific bits' value, which leads to exclusive conditions, we bit-wise construct the optimal leakage model. For instance, assume $\mathcal{C} \in \mathcal{L}_1$, then $\hat{L} = 1$:

$$\mathbb{E}_M(\hat{L}) = \frac{1}{2^8} \sum_{m \in \mathbb{F}_{256}} 1 \wedge \mathcal{C} = \frac{1}{2^8} \sum_{(m_7,\ldots,m_0) \in \mathbb{F}_2^8} \underbrace{(0 \wedge \mathcal{C}, \ldots, 0 \wedge \mathcal{C}, 1 \wedge \mathcal{C})}_{8 \text{ bits}}.$$

Now, $\mathcal{C}$ does not depend on $m_5, \ldots, m_0$. Thus the first bit of $\mathbb{E}_M(\hat{L})$ becomes:

$$\mathbb{E}_M(\hat{L})_0 = \frac{1}{2^2} \sum_{(m_7,m_6) \in \mathbb{F}_2^2} (\neg m_7)(\neg m_6)(S_7^0 \oplus m_7)(S_6^0 \oplus m_6)(S_7^{15} \oplus m_7).$$

$$\begin{aligned}\mathbb{E}_{M,d}(\hat{L})_0 &= \frac{1}{2^8} \sum_{x_{15} \in \mathbb{F}_{256}} \mathbb{E}_M(\hat{L})_0 = \frac{1}{2^8} \sum_{S^{15} \in \mathbb{F}_{256}} \mathbb{E}_M(\hat{L})_0 = \frac{1}{2} \sum_{S_7^{15} \in \mathbb{F}_2} \mathbb{E}_M(\hat{L})_0 \\ &= \frac{1}{2^3} S_7^0 S_6^0.\end{aligned}$$

Finally, for every register bit $i \in \{0, \ldots, 63\}$ and all possible values of $\hat{L}$ (5), the optimal leakage model is obtained by summing $\mathbb{E}_{M,d}(\hat{L})_i$ using the Python library SymPy dedicated to symbolic mathematics [25]. □

We perform a new SCA with this optimal leakage model and the maximum likelihood (ML) distinguisher [9], on the same traces as previously:

$$\mathcal{D}_{\mathsf{ML}}(X, \ell_O(x_0, k_0)) = \underset{k \in \mathbb{F}_8}{\operatorname{argmax}} \sum_{i \in \mathbb{F}_{64}} \|\rho(X, \ell_O(x_0, k_0)_i)\|^2, \tag{6}$$

where ρ is the Pearson's correlation, X the simulated side-channel activity, and k the key guess. Results are shown in Fig. 6.

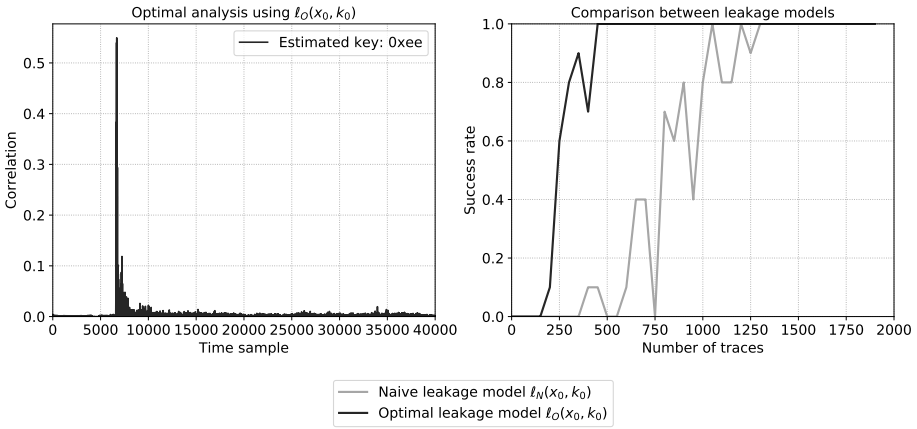


Fig. 6. Maximum likelihood analysis performed on the asynchronous masked AES-128 using the optimal leakage model (left figure). The subkey k_0 is broken with $9,356$ traces. Its success rate is compared to the naive leakage model in the right figure, averaging over 10 analyses, with randomly chosen traces.

Again, a peak is clearly visible at the first AES round (left figure), which corresponds to the correct key, confirming this model is effective. We estimate the number of traces needed for breaking the key, to compare it with the leakage model defined in (Naive leakage model). The success rates, using both the naive and the optimal leakage models, are given in the right figure: around 400 traces are needed with $\ell_O(x_0, k_0)$, whereas $1,300$ traces are necessary through the naive approach. This confirms, once again, that the side-channel leakage due to the combination of the masking scheme with some misalignment is not a coincidence, but an exploitable vulnerability.

5 Discussion

The first-order leakage on the non-constant time masked AES could as well be detected using metrics such as Quantitative Masking Strength (QMS [14]), Normalized Inter-Class Variance (NICV [3,26]), or Information Leakage Amount (ILA [32]). However, those statistical tools do not help in understanding the root-cause for the leakage – which is crucial for developers to fix their cryptographic implementations.

Our methodology, based on a study of the symbolic values stored in the registers (recall Fig. 4) allows to attribute the bias to the C code of Alg. 2. Such an approach makes non-regression security testing possible, *i.e.*, evaluating and fixing an implementation before any integration on end-user devices.

6 Conclusions

We studied an end-to-end masked AES algorithm. The masking implementation has been analysed and proven correct. Still, we present a successful first-order analysis. In order to track the leakage back to its responsible code lines, we used an emulated environment and identified that the leakage is produced by a peculiarity of the code: it has some non-constant time structures. As a consequence, the leaking code lines differ from one trace to the other, which complicates the leakage investigation. Evaluation tools which permit to perform Side-Channel Analyses at register level are good solutions for both, understanding leakages in general and improving analyses on end-user devices. Indeed, combining the analysis on symbolic values during the execution and a simulation at register level, allows to attribute the leakages to the corresponding non-constant time operations. The emerging problem is not trivial, as we are not aware of static tools for leakage detection which work turnkey, when masking is in place.

Again, we must point out the importance of a constant execution time: not only can it be exploited by timing- or cache-based attacks, but as we have shown, it can also ruin a masking scheme.

References

1. Almeida, J.B., Barbosa, M., Barthe, G., Dupressoir, F., Emmi, M.: Verifying Constant-Time Implementations. In: 25th USENIX Security Symposium, USENIX Security 16, Austin, TX, USA, August 10-12, 2016. pp. 53–70 (2016)
2. Barthe, G., Belaïd, S., Dupressoir, F., Fouque, P., Grégoire, B., Strub, P.: Verified Proofs of Higher-Order Masking. In: Advances in Cryptology - EUROCRYPT 2015 - 34th Annual International Conference on the Theory and Applications of Cryptographic Techniques, Sofia, Bulgaria, April 26-30, 2015, Proceedings, Part I. pp. 457–485 (2015)
3. Bhasin, S., Danger, J.L., Guilley, S., Najm, Z.: NICV: Normalized Inter-Class Variance for Detection of Side-Channel Leakage. In: International Symposium on Electromagnetic Compatibility (EMC '14 / Tokyo). IEEE (May 12-16 2014), Session OS09: EM Information Leakage. Hitotsubashi Hall (National Center of Sciences), Chiyoda, Tokyo, Japan
4. Bhattacharya, S., Maurice, C., Bhasin, S., Mukhopadhyay, D.: Branch Prediction Attack on Blinded Scalar Multiplication. IEEE Trans. Computers **69**(5), 633–648 (2020)
5. Blömer, J., Guajardo, J., Krummel, V.: Provably Secure Masking of AES. In: Selected Areas in Cryptography. pp. 69–83 (2004)
6. Bos, J.W., Hubain, C., Michiels, W., Teuwen, P.: Differential Computation Analysis: Hiding Your White-Box Designs is Not Enough. In: Cryptographic Hardware and Embedded Systems - CHES 2016 - 18th International Conference, Santa Barbara, CA, USA, August 17-19, 2016, Proceedings. pp. 215–236 (2016)
7. Bouvet, A., Bruneau, N., Facon, A., Guilley, S., Marion, D.: Give me your binary, I'll tell you if it leaks. In: 13th International Conference on Design & Technology of Integrated Systems In Nanoscale Era, DTIS 2018, Taormina, Italy, April 9-12, 2018. pp. 1–4 (2018)

8. Brier, É., Clavier, C., Olivier, F.: Correlation power analysis with a leakage model. In: Cryptographic Hardware and Embedded Systems - CHES 2004: 6th International Workshop Cambridge, MA, USA, August 11-13, 2004. Proceedings. pp. 16–29 (2004)
9. Bruneau, N., Guilley, S., Heuser, A., Marion, D., Rioul, O.: Optimal side-channel attacks for multivariate leakages and multiple models. J. Cryptographic Engineering **7**(4), 331–341 (2017)
10. Census labs: Masked AES (2012), http://census-labs.com/
11. Chari, S., Jutla, C.S., Rao, J.R., Rohatgi, P.: Towards Sound Approaches to Counteract Power-Analysis Attacks. In: CRYPTO. pp. 398–412 (1999)
12. Coron, J., Kizhvatov, I.: An Efficient Method for Random Delay Generation in Embedded Software. In: Clavier, C., Gaj, K. (eds.) Cryptographic Hardware and Embedded Systems - CHES 2009, 11th International Workshop, Lausanne, Switzerland, September 6-9, 2009, Proceedings. Lecture Notes in Computer Science, vol. 5747, pp. 156–170. Springer (2009)
13. Danger, J.L., Debande, N., Guilley, S., Souissi, Y.: High-order Timing Attacks. In: Proceedings of the First Workshop on Cryptography and Security in Computing Systems. pp. 7–12. CS2 '14, ACM, New York, NY, USA (2014)
14. Eldib, H., Wang, C., Taha, M., Schaumont, P.: QMS: Evaluating the Side-Channel Resistance of Masked Software from Source Code. In: Proceedings of the The 51st Annual Design Automation Conference on Design Automation Conference. pp. 209:1–209:6. DAC '14, ACM, New York, NY, USA (2014)
15. French ANSSI: Secure AES128 for ATMega8515 (2021), https://github.com/ANSSI-FR/secAES-ATmega8515/tree/master/src/Version2, accessed on May 14, 2021
16. French ANSSI: Secure AES128 for STM32 (2021), https://github.com/ANSSI-FR/SecAESSTM32/blob/master/src/aes/, accessed on May 14, 2021
17. Gandolfi, K., Mourtel, C., Olivier, F.: Electromagnetic analysis: Concrete results. In: Proceedings of the Third International Workshop on Cryptographic Hardware and Embedded Systems. pp. 251–261. CHES '01, Springer-Verlag, London, UK, UK (2001)
18. Genkin, D., Shamir, A., Tromer, E.: RSA Key Extraction via Low-Bandwidth Acoustic Cryptanalysis. In: Advances in Cryptology - CRYPTO 2014 - 34th Annual Cryptology Conference, Santa Barbara, CA, USA, August 17-21, 2014, Proceedings, Part I. pp. 444–461 (2014)
19. Goubin, L., Patarin, J.: DES and Differential Power Analysis. The "Duplication" Method. In: CHES. pp. 158–172. LNCS, Springer (Aug 1999), Worcester, MA, USA
20. Güneysu, T., Moradi, A.: Generic side-channel countermeasures for reconfigurable devices. In: CHES. pp. 33–48 (2011)
21. Kocher, P.C.: Timing Attacks on Implementations of Diffie-Hellman, RSA, DSS, and Other Systems. In: Proceedings of CRYPTO'96. LNCS, vol. 1109, pp. 104–113. Springer-Verlag (1996)
22. Kocher, P.C.: Timing Attacks on Implementations of Diffie-Hellman, RSA, DSS, and Other Systems. In: Advances in Cryptology - CRYPTO '96, 16th Annual International Cryptology Conference, Santa Barbara, California, USA, August 18-22, 1996, Proceedings. pp. 104–113 (1996)
23. Kocher, P.C., Jaffe, J., Jun, B.: Differential power analysis. In: Proceedings of the 19th Annual International Cryptology Conference on Advances in Cryptology. pp. 388–397. CRYPTO '99, Springer-Verlag, London, UK, UK (1999)

24. Maghrebi, H., Danger, J.L., Flament, F., Guilley, S., Sauvage, L.: Evaluation of countermeasure implementations based on boolean masking to thwart side-channel attacks. In: 2009 3rd International Conference on Signals, Circuits and Systems (SCS). pp. 1–6. IEEE (2009)
25. Meurer, A., Smith, C.P., Paprocki, M., Čertík, O., Kirpichev, S.B., Rocklin, M., Kumar, A., Ivanov, S., Moore, J.K., Singh, S., et al.: SymPy: symbolic computing in Python. PeerJ Computer Science **3**, e103 (2017)
26. Moradi, A., Guilley, S., Heuser, A.: Detecting Hidden Leakages. In: Applied Cryptography and Network Security - 12th International Conference, ACNS 2014, Lausanne, Switzerland, June 10-13, 2014. Proceedings. pp. 324–342 (2014)
27. Nassar, M., Souissi, Y., Guilley, S., Danger, J.L.: RSM: A small and fast countermeasure for AES, secure against 1st and 2nd-order zero-offset SCAs. In: 2012 Design, Automation & Test in Europe Conference & Exhibition, DATE 2012, Dresden, Germany, March 12-16, 2012. pp. 1173–1178 (2012)
28. Prouff, E., Rivain, M., Bevan, R.: Statistical Analysis of Second Order Differential Power Analysis. IEEE Trans. Computers **58**(6), 799–811 (2009)
29. Veyrat-Charvillon, N., Medwed, M., Kerckhof, S., Standaert, F.X.: Shuffling against Side-Channel Attacks: A Comprehensive Study with Cautionary Note. In: ASIACRYPT. pp. 740–757 (2012)
30. Walters, D.T., Hagen, A., Kedaigle, E.A.: SLEAK: A Side-channel Leakage Evaluator and Analysis Kit (November 2014), technical paper (Case Number 14-3463): https://www.mitre.org/publications/technical-papers/sleak-a-side-channel-leakage-evaluator-and-analysis-kit (accessed on 24 September 2020)
31. Yao, Y.: Masked AES Implementation (February 4 2020), https://github.com/Secure-Embedded-Systems/Masked-AES-Implementation/find/master, accessed on June 2nd, 2021
32. Zhang, L., Ding, A.A., Fei, Y., Luo, P.: A Unified Metric for Quantifying Information Leakage of Cryptographic Devices Under Power Analysis Attacks. In: Iwata, T., Cheon, J.H. (eds.) Advances in Cryptology - ASIACRYPT 2015 - 21st International Conference on the Theory and Application of Cryptology and Information Security, Auckland, New Zealand, November 29 - December 3, 2015, Proceedings, Part II. Lecture Notes in Computer Science, vol. 9453, pp. 338–360. Springer (2015)