

École polytechnique de Louvain

Resilience of AES software implementations against side- channel attacks on micro-controllers

Author: **Brieuc BALON**

Supervisor: **François-Xavier STANDAERT**

Readers: **David BOL, Thi My DUYÊN PAY, Corentin VERHAMME**

Academic year 2022–2023

Master [120] in Electrical Engineering

Abstract

Embedded software security has become nowadays a prominent concern driven by the widespread use of micro-controllers across a diverse range of applications including IoT devices. Ensuring robust security against side-channel attacks has become a crucial objective for enterprises. This thesis specifically focuses on the evaluation of the resilience of AES software implementations on micro-controllers against side-channel attacks. The study delves into the process of conducting a template attack, encompassing both the training and attack phases, to assess the effectiveness of such attacks thanks to state-of-art methodologies. Furthermore, the resilience of two distinct AES implementations, namely lookup table based and fixsliced AES, is thoroughly examined. The findings underscore that, despite the incorporation of countermeasures, software implementations remain susceptible to compromise given a sufficient number of traces. This work sheds light on the significance of implementing robust security countermeasures in embedded software systems to effectively mitigate the risk of side-channel attacks.

Acknowledgements

First and foremost, I would like to thank my supervisor, Prof. François-Xavier Standaert, for introducing me to the captivating field of cryptography through side-channel attacks and for his valuable feedback and continuous support throughout year.

I would also like to thank Corentin and Balazs for their availability to address my queries at any moment of the day and for their assistance in the laboratory.

Furthermore, I am grateful to Prof. David Bol and Thi My Duyen Pay for accepting the role of readers for this thesis.

Finally, I would like to acknowledge the support and encouragement received from my friends and family throughout those 5 years.

Contents

Introduction	1
1 Theoretical background	3
1.1 AES	3
1.1.1 Buidling blocks	3
1.2 Divide and Conquer approach	5
1.3 Side-channel attack	5
1.4 Training phase	6
1.4.1 Point of Interest	7
1.4.2 Signal-to-Noise Ratio	7
1.4.3 Template modeling	8
1.4.4 Linear subspace template	9
1.4.5 LDA-based template	9
1.4.6 Information theory metrics	10
1.4.7 Mutual information	10
1.4.8 Perceived Information	11
1.4.9 Hypothetical Information	11
1.5 Attack phase	12
1.5.1 S-Box output probabilities derivation	12
1.5.2 Sub-key probabilities derivation	12
1.5.3 Key enumeration and rank estimation	13
2 AES 128-bits software implementations	15
2.1 Lookup table based AES	15
2.2 Fixsliced AES	16
3 Experimental results	21
3.1 Experimental setup	21
3.2 Tiny-AES	21
3.2.1 SNR	21
3.2.2 POI	24
3.2.3 LDA-based templates	25
3.2.4 Rank estimation	26
3.3 Fixsliced AES	28
3.3.1 SNR	29
3.3.2 POI	32
3.4 LDA-based templates	32
3.4.1 Rank estimation	34
3.5 Comparison between the implementations	35
Conclusion	37
References	39
Appendices	42

A	LuT based AES	42
B	Fixsliced AES	42

Introduction

In recent years, the proliferation of micro-controllers has led to their integration into a wide range of critical systems, including embedded devices, Internet of Things applications, communication systems and more. This widespread deployment of these devices, emphasizes the importance of ensuring the security and resilience of micro-controllers. While cryptographic algorithms are commonly used to protect sensitive information stored or transmitted by micro-controllers, those algorithms are not immune to attacks. One significant threat for the security of micro-controllers is the *side-channel attacks* (SCA). Instead of focusing on the possible breach of the algorithm, they target the physical security of the device.

Challenges associated with the physical security of micro-controllers against these threats emerged in the past years. First, because micro-controllers are often deployed in resource-constrained environments, limiting the implementation of complex security measures. Then, the small size and low power consumption characteristics of micro-controllers make them vulnerable to physical tampering and unauthorized access. Consequently, investigating the resilience of software encryption algorithm against side-channel attacks on micro-controllers becomes a critical aspect.

Numerous cryptographic algorithms exist to protect sensitive information. Not all of them are suitable for micro-controllers due to the resource constraints. The ideal algorithm should be fast and efficient in terms of space utilization and power consumption. Symmetric encryption schemes are more appropriate to meet those constraints. A widely recognized and robust cryptographic algorithm for symmetric encryption is the *Advanced Encryption Standard* (AES). Since its standardization by the *National Institute of Standards and Technology* (NIST) in 2001 [11], this algorithm has not encountered any critical threats and is still considered acceptable by the NIST for symmetrical encryption nowadays [2].

The resilience of the AES software encryption algorithm against side-channel attacks on micro-controllers is a critical aspect to investigate. Its algorithm is robust, but its implementation on micro-controllers may introduce vulnerabilities that can be exploited through side-channels. Understanding the challenges faced in securing micro-controllers against physical threats and side-channel attacks is essential for implementing effective countermeasures and enhancing the overall security of micro-controller-based systems.

This thesis investigates the resilience of different AES software implementations on micro-controllers against side-channel attacks. An introduction to the AES algorithm will be first provided. Then the focus will be shifted toward one of the most well-known side-channel attacks, namely template attack and various tools and techniques for assessing the effectiveness of this attack will be presented.

Next, two implementations of the AES will be introduced. One of them will be a naive implementation of the algorithm, while the other will be a more sophisticated

and optimized version.

Finally, the practical resilience of those implementations will be evaluated thanks to measurements conducted on a micro-controller. This will contribute to the understanding of the challenges and strategies involved in securing micro-controllers against physical security threats.

1 Theoretical background

1.1 AES

The *Advanced Encryption Standard*, also known as Rijndael, is a block cipher based on a substitution-permutation network developed by Joan Daemen and Vincent Rijmen whose specifications are described in [9]. It is a symmetric encryption algorithm (encryption and decryption use the same key) which operates with key sizes of 128, 192 or 256 bits. The block diagram of the AES execution can be seen in Figure 1. As

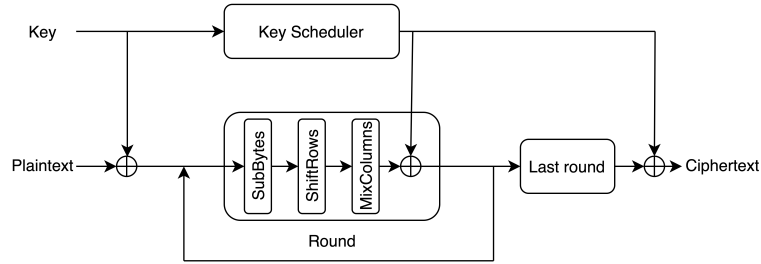


Figure 1: Block diagram of the AES encryption

illustrated, the AES performs the same operations many times, this operation, called round, is performed n times where n depends on the key size. For key size of 128, 192 or 256 bits, there are respectively 10, 12 or 14 rounds. The operations executed within a round are denoted by the building blocks and are described in the following. A different key is used for each round and those different 128 bits keys are derived from the initial one thanks to a key expansion algorithm. One should notice that the key used for the first round corresponds to the initial key and the template attacks against AES are therefore focusing the first round of the execution. This is the reason why the key expansion algorithm will not be described in the following. Moreover, only 128 bits key size will be investigated in this work and for now on, AES will refer to AES with 128 bits key size.

1.1.1 Building blocks

The AES operates on 4×4 matrix of bytes to represent the different internal states during the encryption. Let $\mathbf{s}^r$ be the state matrix at the beginning of the round r where the initial state $\mathbf{s}^0$ stores the 16 bytes of the 128 bits plaintext. The encryption is performed by different blocks working with arithmetic in $\mathbb{F}_{2^8}$.

$$\mathbf{s}^r = \begin{bmatrix} B_0^r & B_4^r & B_8^r & B_{12}^r \\ B_1^r & B_5^r & B_9^r & B_{13}^r \\ B_2^r & B_6^r & B_{10}^r & B_{14}^r \\ B_3^r & B_7^r & B_{11}^r & B_{15}^r \end{bmatrix}$$

AddRoundKey : The Add Round Key ensures key mixing thanks to a bitwise XOR between the state and the key. This is used to have a keyed permutation such as only

the receiver, knowing the key, can decrypt the ciphertext.

$$\text{AddRoundKey}(\mathbf{s}^r) = \mathbf{s}^r \oplus \mathbf{k}^r = \begin{bmatrix} B_0^r \oplus K_0^r & B_4^r \oplus K_4^r & B_8^r \oplus K_8^r & B_{12}^r \oplus K_{12}^r \\ B_1^r \oplus K_1^r & B_5^r \oplus K_5^r & B_9^r \oplus K_9^r & B_{13}^r \oplus K_{13}^r \\ B_2^r \oplus K_2^r & B_6^r \oplus K_6^r & B_{10}^r \oplus K_{10}^r & B_{14}^r \oplus K_{14}^r \\ B_3^r \oplus K_3^r & B_7^r \oplus K_7^r & B_{11}^r \oplus K_{11}^r & B_{15}^r \oplus K_{15}^r \end{bmatrix}$$

SubBytes SubBytes is the operation responsible of the confusion. It performs an inversion in $\mathbb{F}_{2^8}$ followed by an affine transformation on each byte.

$$\text{SubBytes}(B_i^r) = \text{affine}\left((B_i^r)^{-1}\right)$$

$$\text{affine}(B) = \begin{bmatrix} 1 & 0 & 0 & 0 & 1 & 1 & 1 & 1 \\ 1 & 1 & 0 & 0 & 0 & 1 & 1 & 1 \\ 1 & 1 & 1 & 0 & 0 & 0 & 1 & 1 \\ 1 & 1 & 1 & 1 & 0 & 0 & 0 & 1 \\ 1 & 1 & 1 & 1 & 1 & 0 & 0 & 0 \\ 0 & 1 & 1 & 1 & 1 & 1 & 0 & 0 \\ 0 & 0 & 1 & 1 & 1 & 1 & 1 & 0 \\ 0 & 0 & 0 & 1 & 1 & 1 & 1 & 1 \end{bmatrix} \begin{bmatrix} b_0 \\ b_1 \\ b_2 \\ b_3 \\ b_4 \\ b_5 \\ b_6 \\ b_7 \end{bmatrix} + \begin{bmatrix} 1 \\ 1 \\ 0 \\ 0 \\ 0 \\ 1 \\ 1 \\ 0 \end{bmatrix}$$

In the affine transformation, b_0 and b_7 corresponds respectively to the MSB and LSB of the byte B . SubByte is the only non-linear block of the AES and can be pre-computed to create a *substitution box* or S-Box, a table of 256 elements containing the corresponding output. It is why the SubBytes operation is sometimes called the S-Box.

ShiftRows : ShiftRows is one of the block performing the diffusion. It performs a circular left shift on each row with a shift value depending on the row index (left shift of i for the row i). In hardware, this can be implemented by a simple wire crossing.

$$\text{ShiftRows}(\mathbf{s}^r) = \begin{bmatrix} B_0^r & B_4^r & B_8^r & B_{12}^r \\ B_5^r & B_9^r & B_{13}^r & B_1^r \\ B_{10}^r & B_{14}^r & B_2^r & B_6^r \\ B_{15}^r & B_3^r & B_7^r & B_{11}^r \end{bmatrix}$$

MixColumns : MixColumns is the second block performing diffusion. Each column is associated to a polynomial such that $a(x) = a_3x^3 + a_2x^2 + a_1x + a_0$ where a_i is the i^{th} byte, of the column. This polynomial is then multiplied by another fixed one $c(x) = 03x^3 + 02x^2 + 01x + 02 \mod x^4 + 1$. This can be written as a matrix multiplication where $\mathbf{s}_{i,j}^r$ denotes the j^{th} byte of the i^{th} column of $\mathbf{s}^r$.

$$\text{MixColumns}(\mathbf{s}_i^r) = \begin{bmatrix} 02 & 03 & 01 & 01 \\ 01 & 02 & 03 & 01 \\ 01 & 01 & 02 & 03 \\ 03 & 01 & 01 & 02 \end{bmatrix} \begin{bmatrix} \mathbf{s}_{i,0}^r \\ \mathbf{s}_{i,1}^r \\ \mathbf{s}_{i,2}^r \\ \mathbf{s}_{i,3}^r \end{bmatrix}$$

1.2 Divide and Conquer approach

The AES uses a 128 bits key size. One could attack this 128 bits key at once by enumerating all the 2^{128} different possible values. Even if this attack is theoretically possible, in practice exhaustive key search is too long to be successful. To reduce the number of possible values in the enumeration process, one could try to attack more keys of smaller sizes called *sub-keys*. For instance, attacking 4 sub-keys of 32 bits or 32 sub-keys of 4 bits. This approach is called a *divide and conquer* attack.

The AES is originally designed not to encrypt the 128 bits plaintext at once but by encrypting it using arithmetic in $\mathbb{F}_{2^8}$. It implies that the 128 bits are sometimes manipulated by block of 8 bits and this operation is repeated 16 times. This behavior can be seen during the AES encryption with the SubBytes block which operates the inverse operation in $\mathbb{F}_{2^8}$ on the 16 bytes independently. Moreover, this block is also the only non-linear block of the AES execution. It means that two similar plaintexts (i.e. differing from one bit) encrypted with the same key can have more than one bit of difference at the output of SubBytes. This would not have been the case with a linear block such as xoring.

SCA commonly focuses on the output of the SubBytes operation of the first round. This place is chosen because of the non-linear behavior of the SubByte block but also because the secret key is xored with the plaintext just before this block. As those operations does not affect more than 8 bits together it is a suitable place to operate a divide and conquer attack where each attack targets the output of one S-Box. Finally, there are 16 independent attacks aiming the 8 bits S-Box output. Figure 2 represents the composition of the AddRoundKey and SubBytes blocks of the first round with notation for the different internal states. In this figure, B_i and K_i corresponds to the

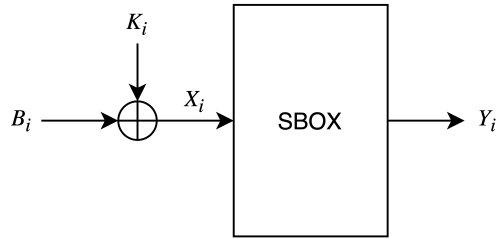


Figure 2: Schematic and notations for an attack on the S-Box output

i^{th} byte of the plaintext and key and X_i and Y_i to its corresponding S-Box input and S-Box output. Those notations will be used in the following and the i will generally be omitted for the sake of simplicity.

1.3 Side-channel attack

Side-Channel Attack refers to attacks that take advantage of any form of leakage of a device to find information about its sensitive variables (i.e. its cryptographic key). Introduced by *Kocher et al* in [15], they are nowadays used to evaluate the resilience of

processors against physical attacks. Those attacks can be invasive, by disrupting the behavior of the device, or they can just monitor the behavior and not interfere with the device operations.

The non invasive attacks are called as *Side-Channel Analysis Attacks* (SCAA) because they only analyze the device behavior. They exploit the information carried by the leakage and compare it with the knowledge of the encrypted plaintexts. There are many types of SCAA, depending on the type of leakage used. It gives different SCAA such as timing, electromagnetic or power analysis attacks. Timing attacks [16] exploit non constant time execution of an algorithm due to branches, cache misses, etc. Electromagnetic attacks [12] use the electromagnetic radiations emitted whereas *Power Analysis Attack* (PAA) takes advantage of the power consumption of the device during the encryption.

One of the most important PAA is the *Template attacks* (TA). It works on the principle that the power consumption of a device follows a given distribution which depends on the values being manipulated. It is based on divide-and-conquer principle where, instead of recovering the full key at once, each byte of the key is attacked independently. This helps to reduce the complexity of the attack as well as its time execution.

The TA is usually performed in two phases:

Firstly, a training phase can be conducted by the attacker where he collects multiple traces with different known plaintexts and keys. The goal of the training phase is to profile and model the leakage of a device when it manipulates a sensitive variable. To do so, the attacker chooses a model following the leakage distribution and he characterizes the model thanks to the training traces. For instance, one could assume that each sample of the leakage follows a gaussian distribution and the training traces are used to compute the corresponding mean and standard deviation. In the end, the training phase aims to obtain a model for every internal state value.

Secondly, the attacker tries to recover the value of the sensitive variables for new leakages. To achieve this, he uses the trained model on the new traces to derive different probabilities associated to the sensitive variable values. The highest probability should correspond to the correct internal state value if the model corresponds to the actual leakage distribution.

In this thesis, one will focus on recovering the 128 bits key used during an AES encryption by using a divide and conquer template attack with models on 8 bits.

1.4 Training phase

In the training phase, the attacker is able to record the leakage of many encryptions with different plaintexts and keys which are both known. This set of leakage and its corresponding data is referred to as the *training set*. The goal of the training phase is to use the training set in order to profile the leakage of the device. This leakage follows a certain distribution that is unknown by the attacker and which depends on the value of the internal state (i.e. the output of the S-Box). A statistical model must

be selected to represent the unknown leakage distribution for a given internal state value and the training set is then used to estimate the model's parameters. At the end of the training phase, 16 attacks are built where each of them is focusing on the S-Box output. One will only present how to construct the attack for one S-Box, the combination of the 16 S-Boxes will be shown in section 1.5.

One attack is composed of 256 models, one for each 8 bits S-Box output value. In this section, one will firstly present how to choose and build the model from the training set. Then, one will introduce some tools, relying on information theory, that measures the quality of the model built.

1.4.1 Point of Interest

The recorded leakage traces contain samples that can be relevant or irrelevant depending on which internal state is modeled. For instance, Figure 3 shows the execution of AES encryption. The attack focuses on the first round of the encryption thus the samples associated to the other rounds could be not useful. The relevant samples are commonly called *Point of Interest* (POI). They should be well chosen in order to fit accurately the model to the true leakage distribution of the device. Moreover, even if all samples of the first round can be useful, some of them could give more information than others. For instance, the samples acquired at the same time as the processor loaded the S-Box output are much more informative than others samples. There exists a statistical tool for finding those most informative POI which is called *Signal-to-Noise Ratio* (SNR).

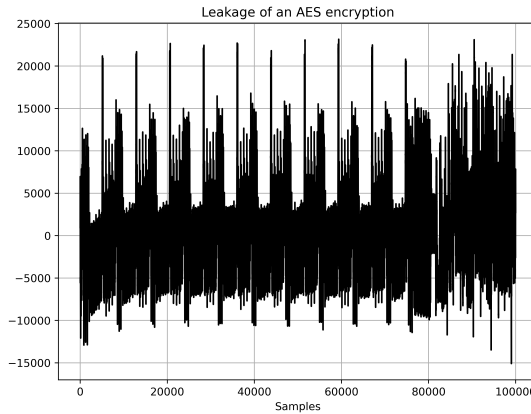


Figure 3: Ten rounds of an AES encryption

1.4.2 Signal-to-Noise Ratio

The SNR computes the signal to noise ratio between the leakage and a certain internal state Y . It quantifies the amount of information about the internal state Y within the

leakage and it is defined in [17] as :

$$\text{SNR} = \frac{\hat{\mathbb{V}}_y(\hat{\mathbb{E}}_T(L_y^T))}{\hat{\mathbb{E}}_y(\hat{\mathbb{V}}_T(L_y^T))}$$

where the hat symbol $\hat{\cdot}$ denotes the estimation of a statistic and L_y^T represents the random variable associated to the leakage of all the traces T that have an internal state Y with a value y .

After the SNR computation, each sample of the trace has a value corresponding to the amount of information about Y for that sample. This value is referred to SNR value and if a sample has a big SNR value, it indicates that the sample is relevant to get information about Y .

In the training phase, all internal states can be computed as the plaintexts and keys are known. That is why the SNR is useful to find the most informative samples associated to the corresponding internal state. The number of POI (n_{POI}) can then be chosen by selecting all the samples with a SNR value over a given threshold or by selecting the n_{POI} samples with the greatest SNR value.

1.4.3 Template modeling

The leakage distribution of a device is generally unknown by an attacker. *Template modeling* aims at approximating the unknown distribution with a simple model and then finding the parameters that characterize it thanks to the training set.

The leakage of a device encrypting 2 different plaintexts should be different. Actually it is also the case even if both plaintexts are the same. For the first case, it comes from the difference with the values manipulated and for the second one, the variations in the measurements are caused by electronic noise (coming from the device or the probing equipment). If one assumes that this electronic noise is gaussian, then one can represent each sample of the leakage by a normal law. This normal law is characterized by two parameters : a mean and a standard deviation. The mean depends on the plaintext encrypted and the standard deviation is coming from the noise. Each of them can be estimated from the training set and the attacker can use the model of one of the samples for the attack.

For a better modeling of the actual leakage distribution, multiple samples can be used together and multivariate analysis is useful to take them all into account. With the gaussian assumption, the best possible choice for the model is a multivariate normal distribution. The *probability density function* (pdf) of this distribution for a new trace $\mathbf{I}^{\text{new}}$ and for an internal state value y is given by :

$$\mathcal{N}(\mathbf{I}^{\text{new}}|\boldsymbol{\mu}_y, \boldsymbol{\Sigma}_y) = \frac{1}{\sqrt{(2\pi)^{N_t}|\boldsymbol{\Sigma}_y|}} \exp\left(-\frac{1}{2}(\mathbf{I}^{\text{new}} - \boldsymbol{\mu}_y)^\top \boldsymbol{\Sigma}_y^{-1}(\mathbf{I}^{\text{new}} - \boldsymbol{\mu}_y)\right)$$

Where $\mathbf{I}^{\text{new}}$ is the leakage vector of a new encryption with n_s samples, $\boldsymbol{\mu}_y$ and $\boldsymbol{\Sigma}_y$ are respectively the mean vector and the covariance matrix of the leakage distribution for

the internal state value y . Both of them are unknown but can be approximated by their sampled values. They are obtained thanks to N_t training traces and they can be written as :

$$\hat{\boldsymbol{\mu}}_y = \mathbb{E}(\mathbf{I}_y^{\text{tr}}) \quad \hat{\boldsymbol{\Sigma}}_y = \text{Cov}(\mathbf{I}_y^{\text{tr}}, \mathbf{I}_y^{\text{tr}})$$

Where $\mathbf{I}^{\text{tr}}$ corresponds to all of the training traces and $\mathbf{I}_y^{\text{tr}}$ is its linear subspace for which all leakages have the internal state value y . One attack will be modeled by 256 multivariate gaussians that can be represented thanks to the corresponding sampled mean vector and covariance matrix.

1.4.4 Linear subspace template

Template attacks is described as one of the most powerful divide and conquer SCA [7, 8] despite its high computing and storing cost. For instance, if all samples are used, there are $2 \times 256 \times 16$ vectors¹ of n_s elements to store. In order to reduce the size of the template, a linear projection can be performed from the leakage domain of size n_s to a linear subspace of size n_d by the use of a dimensionality reduction matrix $\mathbf{W}$ of size $n_d \times n_s$.

Therefore, the resulting pdf $\mathcal{N}(\mathbf{W}\mathbf{I}^{\text{new}\top} | \hat{\boldsymbol{\mu}}'_y, \hat{\boldsymbol{\Sigma}}'_y)$ is characterized by $\hat{\boldsymbol{\mu}}'_y = \mathbb{E}(\mathbf{W}\mathbf{I}_y^{\text{tr}\top})$ and $\hat{\boldsymbol{\Sigma}}'_y = \text{Cov}(\mathbf{W}\mathbf{I}_y^{\text{tr}\top}, \mathbf{W}\mathbf{I}_y^{\text{tr}\top})$

In this work, the leakage ($\mathbf{I}^{\text{tr}}$) used in the linear subspace transformation has already been reduced with the POI selection. This reduced leakage, denoted $\tilde{\mathbf{I}}^{\text{tr}}$, is obtained by only keeping the samples with a SNR value over a given threshold. This transformation, called SNR reduction, gives the matrix $\tilde{\mathbf{I}}^{\text{tr}}$ of size $N_t \times n_{\text{POI}}$ composed by the n_{POI} most informative columns of $\mathbf{I}^{\text{tr}}$.

1.4.5 LDA-based template

Linear Discriminant Analysis (LDA) based template is a linear subspace modeling where the matrix $\mathbf{W}$ is composed by the direction that maximize the inter-class and intra class ratio. Those directions $\{\mathbf{w}_m\}_{m=1}^{n_d}$ are chosen in order to maximize the ratio $\frac{\mathbf{w}^\top \mathbf{S}_B \mathbf{w}}{\mathbf{w}^\top \mathbf{S}_W \mathbf{w}}$ between $\mathbf{S}_B$, the inter-class scatter, and $\mathbf{S}_W$, the total intra-class scatter, as described in [22] :

$$\mathbf{S}_B = \sum_{k=0}^{|\mathcal{K}|-1} N_t (\hat{\boldsymbol{\mu}}_k - \bar{\boldsymbol{\mu}})(\hat{\boldsymbol{\mu}}_k - \bar{\boldsymbol{\mu}})^\top$$

$$\mathbf{S}_W = \sum_{k=0}^{|\mathcal{K}|-1} \sum_{n=0}^{N_t-1} (\mathbf{I}_n^{\text{tr}} - \hat{\boldsymbol{\mu}}_k)(\mathbf{I}_n^{\text{tr}} - \hat{\boldsymbol{\mu}}_k)^\top$$

¹The covariance matrix can be stored as a diagonal matrix because one can assume that the noise between 2 different samples is uncorrelated.

Where $\bar{\boldsymbol{\mu}} = \frac{1}{|\mathcal{K}|} \sum_{k=0}^{|\mathcal{K}|-1} \hat{\boldsymbol{\mu}}_k$ and $\mathbf{I}_n^{\text{tr}}$ is the n^{th} trace of $\mathbf{I}^{\text{tr}}$. As $\mathbf{S}_{\mathbf{B}}$ is semi-definite and symmetric and $\mathbf{w}$ is scale invariant, the maximization can be rewritten as :

$$\mathbf{S}_B^{1/2} \mathbf{S}_W^{-1} \mathbf{S}_B^{1/2} = \mathbf{U} \boldsymbol{\Delta} \mathbf{U}^\top$$

Where the matrix $\mathbf{U}$ corresponds to the eigenvectors of the empirical covariance matrix $\hat{\boldsymbol{\Sigma}}_k$. Finally, the directions $\{\mathbf{w}_m\}_{m=1}^{n_d}$ are associated to the n_d largest eigenvalues of $\boldsymbol{\Delta}$.

For the sake of understanding, the notation n_d will be referred as n_{lda} in the rest of this document. The final gaussian distribution, after SNR reduction and LDA-templates, is given by :

$$\mathcal{N}\left(\mathbf{W}\tilde{\mathbf{I}}^{\text{new}\top} \middle| \mathbb{E}(\mathbf{W}\tilde{\mathbf{I}}^{\text{tr}\top}), \text{Cov}(\mathbf{W}\tilde{\mathbf{I}}^{\text{tr}\top}, \mathbf{W}\tilde{\mathbf{I}}^{\text{tr}\top})\right)$$

Where $\mathbf{W}$ is the projection matrix of size $n_{\text{lda}} \times n_{\text{POI}}$ found by LDA. Thanks to SNR reduction and LDA-based templates, the vectors stored for representing the gaussian distributions are much smaller than before. Moreover, the cost of computing those parameters is reduced.

Limitation. As highlighted in [8, 22], the use of LDA templates is restricted to some conditions. Firstly, the condition of equal covariance or also known as homoscedascity that is assumed for simplifying the optimization problem. Secondly, the computation of $\mathbf{S}_W$ implies that the number of traces N_t must be greater than the number of POI n_{POI} otherwise the covariance matrix becomes singular.

1.4.6 Information theory metrics

During the training, the leakage for every internal state value can be modeled by a sampled statistical distribution. This sampled distribution could, sometimes, not represent the true distribution due to estimation or assumption errors in the leakage model. The estimation errors are caused by an insufficient number of traces used to profile the model and the assumption errors can be related to an incorrect choice of the density estimation (i.e. assumption of gaussian noise). Introduced in [13, 23], information theory helps to evaluate the accuracy of the model by giving bounds to the amount of information present in it for worst-case attacks. This amount of information corresponds to the number of bits that can be found thanks to the model when evaluating it on a new trace.

1.4.7 Mutual information

Denoting Y and L discrete random variables associated to the S-Box output and the leakage. The *Mutual Information* (MI) between the leakage and the S-Box output is given by :

$$\text{MI}(Y; L) = H(Y) + \sum_{y \in \mathcal{Y}} \Pr(Y = y) \sum_{l \in \mathcal{L}} \Pr(L = l | Y = y) \cdot \log_2 \Pr(Y = y | L = l) \quad (1)$$

Where $H(Y)$ is the entropy of Y and can be rewritten as $H(Y) = \log_2(|\mathcal{Y}|)$ assuming uniformly distributed values. $\mathcal{Y}$ denotes the set of all possible S-Box outputs and

$|\mathcal{Y}|$ its cardinality. As highlighted in [6], the real leakage distribution is unknown but it can be approximated by its sampled estimate. Using the simplified notation $\Pr(Y = y) := \Pr(y)$, the MI computed by sampling is given by:

$$\hat{\text{MI}}(Y; L) = H(Y) + \sum_{y \in \mathcal{Y}} \Pr(y) \sum_{i=1}^{N_t(y)} \frac{1}{N_t(y)} \log_2 \Pr(y|l_y(i)) \quad (2)$$

Where $N_t(y)$ corresponds to the number of traces in the training set with the same S-Box output y and $l_y(i)$ is the i^{th} leakage of an encryption with a S-Box output y . The true unknown distribution $\Pr(l|y)$ of equation (1) is replaced by its empirical value in equation (2). Since the empirical distribution converges toward its real value when $N_t \rightarrow \infty$, $\hat{\text{MI}}(Y; L)$ also tends to $\text{MI}(Y; L)$. Unfortunately the distribution $\Pr(y|l)$ is unknown and therefore $\hat{\text{MI}}(Y; L)$ cannot be computed. Even if the MI cannot be found, it may be bounded by two limits, a lower and upper one, and those bounds converge toward the same value as the number of traces for the training phase increases.

1.4.8 Perceived Information

The *Perceived Information* (PI) corresponds to the lower bound of the MI as shown in [6]. It represents the amount of information in the leakage that will be exploited by an attacker if he uses the model to get a value for $\Pr(y|l)$ [10]. Thus the unknown probability of Equation (2) is approximated by its estimate on the model. The PI is computed on a new set of traces consisting of N_{PI} leakages that are not used for the templates modeling and this gives:

$$\hat{\text{PI}}(Y; L) = H(Y) + \sum_{y \in \mathcal{Y}} \Pr(y) \sum_{i=1}^{N_{\text{PI}}(y)} \frac{1}{N_{\text{PI}}(y)} \log_2 \hat{\text{Pr}}_{\text{model}}[y|l_y(i)]$$

As the traces used for the PI evaluation are not the traces taken for the model building, the PI is always smaller than the MI. Moreover, it can have negative values indicating that the sampled distribution obtained on the new set of traces is too different from the estimated model.

1.4.9 Hypothetical Information

The *Training Information* (TI) corresponds to the higher bound of the MI as shown in [18]. It represents the amount of information obtained if the model built corresponded to true leakage distribution. It can be found by overfitting the MI which is achieved by using the model on the traces of the training set to get an estimation of $\Pr(y|l)$. This can be written as :

$$\hat{\text{TI}}(Y; L) = H(Y) + \sum_{y \in \mathcal{Y}} \Pr(y) \sum_{i=1}^{N_t(y)} \frac{1}{N_t(y)} \log_2 \hat{\text{Pr}}_{\text{model}}[y|l_y(i)]$$

In other words, TI relies on the model instead of its empirical distribution. This implies that it carries possible biases coming from the choice of the model.

1.5 Attack phase

When the attacker has obtained a model for every output of the S-Box, he can use it on new leakages. When he evaluates a new leakage on to the models, he gets 256 probabilities, one for each S-Box output. The probabilities can then be mapped to their corresponding S-Box input thanks to the bijective relation between the inputs and outputs of SubByte. As the attacker knows the plaintext used for the encryption, he can relate those probabilities to their corresponding sub-key value.

In this section, one will first present how to compute the different probabilities of the S-Box output given only one leakage. Afterwards, one will illustrate with an example how to find the probabilities associated to the sub-key from the ones associated to the S-Box output. Then, one will show how to combine those probabilities together when the attacker can have access to multiple leakages with the same encryption key. Finally, one will present how to combine the different sub-keys probabilities together to retrieve the full 128 bits key.

1.5.1 S-Box output probabilities derivation

In the attack phase, the goal is to derive the probabilities of the different S-Box outputs y given a new leakage $\mathbf{I}^{\text{new}}$ by computing $\Pr(y|\mathbf{I}^{\text{new}})$. Unfortunately, this probability cannot be found continuously and that is why it is approximated by its sampled value $\hat{\Pr}[y|\mathbf{I}^{\text{new}}]$. This value can be obtained by applying Bayes theorem such that :

$$\hat{\Pr}[y|\mathbf{I}^{\text{new}}] = \frac{\hat{\Pr}[\mathbf{I}^{\text{new}}|y]\hat{\Pr}[y]}{\underbrace{\sum_{y^*=0}^{|\mathcal{Y}|-1} \hat{\Pr}[\mathbf{I}^{\text{new}}|y^*]\hat{\Pr}[y^*]}_{\hat{\Pr}[\mathbf{I}^{\text{new}}]}} = \frac{\hat{\Pr}[\mathbf{I}^{\text{new}}|y]}{\sum_{y^*=0}^{|\mathcal{Y}|-1} \hat{\Pr}[\mathbf{I}^{\text{new}}|y^*]}$$

The last equality coming from the assumption that all y are supposed to be uniformly distributed.

In the training phase, the attacker has built a model giving the sampled probability $\hat{\Pr}_{\text{model}}[\mathbf{W}\tilde{\mathbf{I}}^{\text{new}}|y]$ for every internal state value y . Thus, after SNR reduction and LDA-projection of $\mathbf{I}^{\text{new}}$, the attacker can use the profiled gaussian models to get an approximation of $\hat{\Pr}[y|\mathbf{W}\tilde{\mathbf{I}}^{\text{new}}]$ for every y such that:

$$\hat{\Pr}[y|\mathbf{W}\tilde{\mathbf{I}}^{\text{new}}] \approx \frac{\hat{\Pr}_{\text{model}}[\mathbf{W}\tilde{\mathbf{I}}^{\text{new}}|y]}{\sum_{y^*=0}^{|\mathcal{Y}|-1} \hat{\Pr}_{\text{model}}[\mathbf{W}\tilde{\mathbf{I}}^{\text{new}}|y^*]} \quad \forall y \in [0, |\mathcal{Y}| - 1]$$

1.5.2 Sub-key probabilities derivation

The attacker generally wants to find the encryption key instead of the value of the S-Box output. To retrieve the probabilities associated to the key from $\hat{\Pr}[y|\mathbf{W}\tilde{\mathbf{I}}^{\text{new}}]$, one can use the bijective proprieties of AddRoundKey and SubBytes. Let use the notation from Figure 2 with SBOX^{-1} denoting the inverse of SubBytes, b , k and x the value of

the B , K and X . The sub-key probabilities derivation is illustrated in Table 1 for one leakage $\mathbf{I}^{\text{new}}$.

y	0	1	2	$\dots$	254	255
$\hat{\text{Pr}}[y \mathbf{W}\tilde{\mathbf{I}}^{\text{new}}]$	0.01	0.2	0.003	$\dots$	0.009	0.05
$x = \text{SBOX}^{-1}[y]$	82	9	106	$\dots$	12	125
$k = x \oplus b$	94	5	102	$\dots$	0	113

Table 1: Illustration of the sub-key probabilities derivation with $b = 12$ and random probabilities

The derivation corresponds to an index reordering based on the value of the plaintext. In this table the probability for the sub-key $k = 0$ when $b = 12$ is given by : $\hat{\text{Pr}}[k = 0|\mathbf{W}\tilde{\mathbf{I}}^{\text{new}}] = \hat{\text{Pr}}[y = 254|\mathbf{W}\tilde{\mathbf{I}}^{\text{new}}] = 0.009$

In most cases, the attacker has recorded multiple traces with the same encryption key but with different plaintexts. Let $\mathbf{I}_i^{\text{new}}$ denote each of those leakages with $i \in [1, N_a]$ and where N_a corresponds to the number of attacked traces. Assuming independent executions, he can combine every $\hat{\text{Pr}}[k|\mathbf{I}_i^{\text{new}}]$ together and the aggregate probability for the sub-key k is given by :

$$\hat{\text{Pr}}[k|\mathbf{I}_{\text{all}}^{\text{new}}] = \prod_{i=1}^{N_a} \hat{\text{Pr}}[k|\mathbf{W}\tilde{\mathbf{I}}_i^{\text{new}}]$$

The attacker can then sort the different k 's from the most probable to the least based on their probabilities. This most probable value $k^* = \underset{k \in [0, 255]}{\text{argmax}} \hat{\text{Pr}}[k|\mathbf{I}_{\text{all}}^{\text{new}}]$ should correspond to the true sub-key value. To avoid rounding errors, due to the product of small probabilities, one can compute $\hat{\text{Pr}}[k|\mathbf{I}_{\text{all}}^{\text{new}}]$ by summing all the log probabilities of $\hat{\text{Pr}}[k|\mathbf{W}\tilde{\mathbf{I}}_i^{\text{new}}]$.

1.5.3 Key enumeration and rank estimation

In the best scenario, the most probable value k^* for each sub-key corresponds to the correct one. But it usually happens that one or many correct sub-keys are not ranked as the likeliest. Therefore, key enumeration on the sub-keys probabilities vectors must be performed in order to find all correct sub-keys. An attacker can easily perform enumeration up to 2^{32} different 128 bits keys as shown in [28]. In our case of formal study of security analysis, it is useless to use this algorithm to evaluate the resilience of a device to a SCA. Why ? Because it gives a binary indication on whether the SCA can find the key or not. With this approach, it is impossible to see the level of security when the SCA is not able to retrieve the secret key. For instance, to see if the rank of the key is much higher (2^{100}) than 2^{32} or a bit above (2^{40}) that value. In the first

²This is for an attacker using a single core device, an attacker relying on the computation power of the Bitcoin infrastructure could enumerate 2^{58} keys per second.

case, the device resilience is much better than in the other one.

In order to separate those two different cases, a tool called *rank estimation* has been developed in [26, 20] to find the key rank without the need of enumeration. It computes the rank of the full key (between 1 and 2^{128}) based on the different sub-keys probabilities vectors and knowing the correct full key. This tool is commonly used by evaluators to have an indication on the resilience of devices against a certain SCA. The devices with a key rank below 2^{32} are considered as unsecured against the SCA and the other ones can be ranked based on their key rank value to indicate their resilience.

The rank estimation algorithm used in this work corresponds to the one described in [20]. It is based on histograms and can give the median rank of the attack as well as its lower and upper bound for the full 128 bits key.

2 AES 128-bits software implementations

One will investigate the resilience of different AES software implementations on the same device against the template attack described in Section 1. Those implementations follow the *Electronic Code Book* (ECB) mode of operation. ECB is the simplest encryption mode which operates independent AES encryption on consecutive plaintexts. One focuses on this mode because, even if it is the most vulnerable [19, 21], it is still proposed by many micro-controllers vendors such as STMicroelectronics or Texas Instrument in their new chips.

The execution of the AES, as shown in Section 1.1, is quite simple to implement in practice. The only difficulty resides in the software implementation of SubBytes, the only non-linear block. It can be implemented as a *Lookup Table* (LuT) or by using logic gates to execute the inversion and the affine transformation. The software implementations considered in this document are :

- *Tiny-AES* a LuT implementation optimized to reduce its code size.
- *Fixsliced AES* a constant-time implementation optimized for its speed execution.

2.1 Lookup table based AES

Tiny-AES is a LuT based AES written for 8 bits micro-controllers whose code can be found on github. LuT based means that it uses a precomputed table storing for each of the 256 inputs, the corresponding output of the SubBytes block. This choice of implementation is designed for small code size and intended for cases where memory footprint and portability are more important than performances.

To illustrate how the implementation works, one uses the following notations for both 128 bits plaintext and key where b_i is the i^{th} bit of the plaintext and B_i its i^{th} byte :

$$\begin{aligned} \text{pt} &= [\overbrace{b_0 b_1 b_2 b_3 b_4 b_5 b_6 b_7}^{B_0} \overbrace{b_8 b_9 b_{10} b_{11} b_{12} b_{13} b_{14} b_{15}}^{B_1} b_{16} \cdots b_{127}] \\ \text{key} &= [\overbrace{k_0 k_1 k_2 k_3 k_4 k_5 k_6 k_7}^{K_0} \overbrace{k_8 k_9 k_{10} k_{11} k_{12} k_{13} k_{14} k_{15}}^{K_1} k_{16} \cdots k_{127}] \end{aligned}$$

In this implementation, the key and the plaintext are manipulated one byte at a time. The two first operations performed on one byte of the plaintext are illustrated in Figure 4 for a 32 bits architecture.

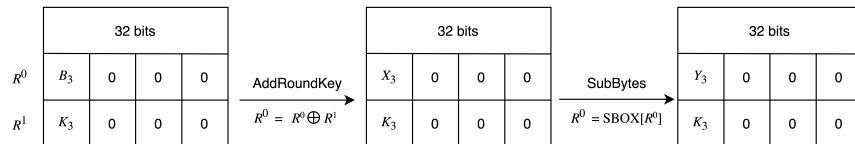


Figure 4: Tiny-AES implementation of AddRoundKey and SubBytes for 32 bits architecture

The figure represents the key xoring of AddRoundKey between the plaintext byte B_3

and the key K_3 followed by the LuT implementation of SubBytes. Each byte is stored in a 32 bits register R^0 and R^1 leaving 24 bits unused during the computation. The other 15 bytes of the plaintext are encrypted the same way as B_3 .

In the Tiny-AES implementation, AddRoundKey and SubBytes are not consecutively performed for each byte. In fact, the 16 key xorings are first performed and then the 16 LuT S-Boxes are executed and all of those 32 executions are performed independently. As a consequence, when the attacker records the leakage, each sample of the trace is going to carry information about the encryption of only one byte. It means that only the manipulation of those 8 bits is going to appear in the measurements with some unknown electrical noise. With enough training traces, the templates should model perfectly the leakage distribution associated to the 8 bits and the noise. Therefore, the probabilities $\hat{\Pr}[y|I^{\text{new}}]$ for all y should correspond to the true distribution if the leakage distribution of the device really follows a gaussian distribution. One will verify this hypothesis in Section 3.

2.2 Fixsliced AES

With the Tiny-AES implementation, only one byte is encrypted at a time even if the processor architecture is greater than 8 bits. One could try to take profit of the architecture size and encrypt as many bytes at the same time as possible. For instance, encrypting 2, 4 or 8 bytes for architecture with register sizes of 16, 32 or 64 bits. Figure 5 illustrates this approach for 32 bits registers. In this figure, one encrypts 4

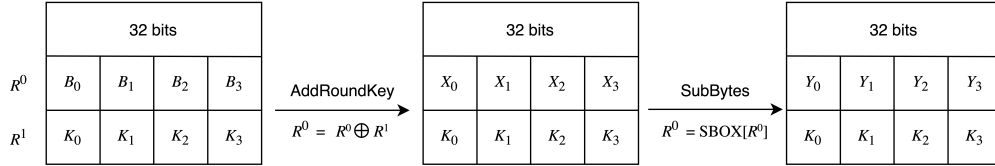


Figure 5: Illustration of the encryption of 4 bytes in parallel with LuT based AES on 32 bits architecture

plaintexts at the same time. The execution is faster compared to Tiny-AES, but the main advantage of encrypting more bytes in parallel, for side-channel resilience, is that the leakage becomes noisier. Let's assume that the attacker keeps the template attack with models on 8 bits. In this configuration when one register is used, 4 bytes are encrypted in parallel. The attacker is going to model the leakage with template on 8 bits. But in the leakage, the register manipulates those 8 bits in addition to 24 others that are not taken into account in the template. In fact, those 24 bits act as noise because they are not used within the model. This noise, due to other parallel encryptions, is called *algorithmic noise*. It should impact the attacker because he would need more training traces, compared with Tiny-AES, to have a good templates model and the models obtained should not give as much information as before. The reason of that is because the templates are built on the assumption that the leakage follows a gaussian distribution. This could be the case with only electrical noise, but the algorithmic noise does not follow this distribution. Its distribution is always deterministic and

cannot be modeled apart if the attacker considers the other 24 bits in the template modeling. This can be achieved by increasing the size of the template with models on 16, 24 or 32 bits. Due to algorithmic noise, the templates should not model the leakage distribution as accurately as with the Tiny-AES implementation.

Unfortunately, a 32 bits LuT, as in Figure 5, is not realistic for micro-controller. It would need to store a table of $2^{32} \times 4 \approx 17\text{GB}$ in memory which is too much for micro-controller application. To store less data, one could do 4 tables lookup, one for each byte, in parallel. But this approach would need 4 CPUs running the LuT operation in parallel and most of the micro-controllers only have one CPU at their disposal.

To counter this problem, one could change the way of calculating the SubBytes operation in order to still have this algorithmic noise in the leakage without relying on LuT. This can be achieved by performing the SubBytes operation with logic gates instead of precomputing it. With this approach, one stores each bit of a byte B_i in a different register, called *slice*. This approach is called *bitslicing* [29] and its bit packing is illustrated in Table 2 with one plaintext for 32 bits registers.

32 bits						
R^0	b_0	B_1	$\dots$	B_{15}	0	$\dots$
R^1	b_1					
R^2	b_2					
R^3	b_3					
R^4	b_4					
R^5	b_5					
R^6	b_6					
R^7	b_7					

Table 2: Conceptual bitsliced bit packing on 32 bits architecture for one plaintext

To perform the SubBytes operation, some logical functions (XOR, AND, OR,...) are applied between the registers in a correct way in order to do the inversion and the affine transform in $\mathbb{F}_{2^8}$. This logical implementation of the S-Box is commonly called *bitsliced S-Box*. It has the advantage to ensure a constant-time execution of SubBytes and it enables a parallel execution of the S-Box on multiple bytes. The attack, with the bit packing as shown in Table 2, still focuses on the output of the bitsliced S-Box Y_i . The only change with the LuT based S-Box is that the Y_i has now its different bits in different registers. By consequence, each register only manipulates one bit of Y_i with 15 other bits of algorithmic noise corresponding to the other Y_i . With this configuration the ratio between the bits of Y_i modeled in the template and the number of bit of algorithmic noise is reduced. It changes from $\frac{8}{24} = 0.25$ to $\frac{1}{15} = 0.0666$.

In Table 2, one can see that only half of the registers are used. One could use the

Many bitsliced S-Boxes have been developed in the literature [14, 29] in order to reduce the number of logical functions. The one used in the fixsliced implementation corresponds to smallest bitsliced S-Box known and has been developed by *Boyar and Peralta* in [3]

$$\begin{array}{l} \text{pt}^i = \overbrace{[b_0^i b_1^i b_2^i b_3^i b_4^i b_5^i b_6^i b_7^i b_8^i b_9^i b_{10}^i b_{11}^i b_{12}^i b_{13}^i b_{14}^i b_{15}^i]}^{B_0^i} b_{16}^i \cdots b_{127}^i \\ \text{key}^i = \overbrace{[k_0^i k_1^i k_2^i k_3^i k_4^i k_5^i k_6^i k_7^i k_8^i k_9^i k_{10}^i k_{11}^i k_{12}^i k_{13}^i k_{14}^i k_{15}^i]}^{K_0^i} \overbrace{[k_{16}^i \cdots k_{127}^i]}^{K_1^i} \end{array}$$

	32 bits						
R^0	b_{24}^0	b_{24}^1	B_7^0	B_7^1	$\dots$	B_{12}^0	B_{12}^1
R^1	b_{25}^0	b_{25}^1			$\dots$		
R^2	b_{26}^0	b_{26}^1			$\dots$		
R^3	b_{27}^0	b_{27}^1			$\dots$		
R^4	b_{28}^0	b_{28}^1			$\dots$		
R^5	b_{29}^0	b_{29}^1			$\dots$		
R^6	b_{29}^0	b_{29}^1			$\dots$		
R^7	b_{30}^0	b_{30}^1			$\dots$		

The execution of fix sliced AES for the two first rounds is represented with a block diagram in Figure 6 where the **blue** refers to the operations of AddRoundKey and the **red** to the operations of SubBytes. In this figure, the packing rectangle block

corresponds to the fixsliced packing of Table 3. A timeline execution of the code is also represented³ in Table 4. In the table, one can see that each operation manipulates 32 bits and the SubBytes operation is not as straightforward as the LuT of Tiny-AES. It corresponds to the operation done to for computing the bitsliced S-Box and it performs the 32 S-Boxes in parallel. In Table 4 for the bitsliced S-Box, the registers, previously storing the keys, are used as temporary registers for the computation.

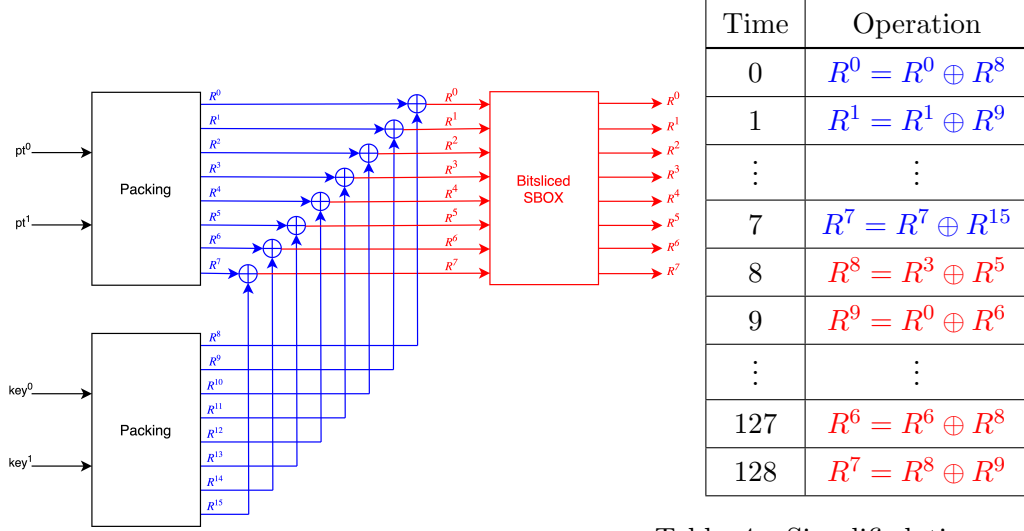


Figure 6: Block diagram of AddRoundKey and SubByte for the fixslice implementation

Table 4: Simplified time execution of the two operations of the first round

The attack mounted on this implementation focuses on the output of the bitsliced S-Box. The attacker still models the templates on the output of a classical S-Box Y_i with $i \in [0, 15]$ for one of the two plaintexts. The only difference is that when one register is manipulated, the attacker is going to find only one bit of information in the leakage for 31 bits of algorithmic noise. At each operation, the attacker is going to have a model build on one bit of data and 31 bits of algorithmic noise. For one register, the ration between the number of bits of Y_i and the number of bits of algorithmic noise is given by $\frac{1}{31} = 0.032$. Therefore, the attack performances should be less important than with Tiny-AES.

³In this table, Time does not correspond to a cycle count but it gives an indication on which operation is executed at which time.

3 Experimental results

In this section, the experimental setup and the application of the template attack on both implementations will be presented. The results obtained for each implementation will be analyzed and critiqued. Finally a comparison between both software will be conducted.

3.1 Experimental setup

The experiments are performed on a ARM Cortex-M4 STM32F303RD mounted on a CW308 UFO board. The Cortex-M4 has an architecture on 32 bits and is operating at a clock frequency of 8MHz, generated using an external quartz. The side-channel leakage is measured with a CT 1 current probe from Tektronix and the signal is sampled at 5 GSamples/sec by a PicoScope 5000 with 12 bits of resolution. With this configuration, one cycle on the targeted device corresponds to 63 samples in the measurements.

3.2 Tiny-AES

The leakage of the Tiny-AES implementation is shown in Figure 7. In this figure, one can see that the first round is between sample 0 and around sample 7500 thanks to the repetition of the rounds. The key expansion and the plaintext bit packing have already been performed before the recording and the trace in Figure 7 only shows the encryption of the plaintext for the first 3 rounds.

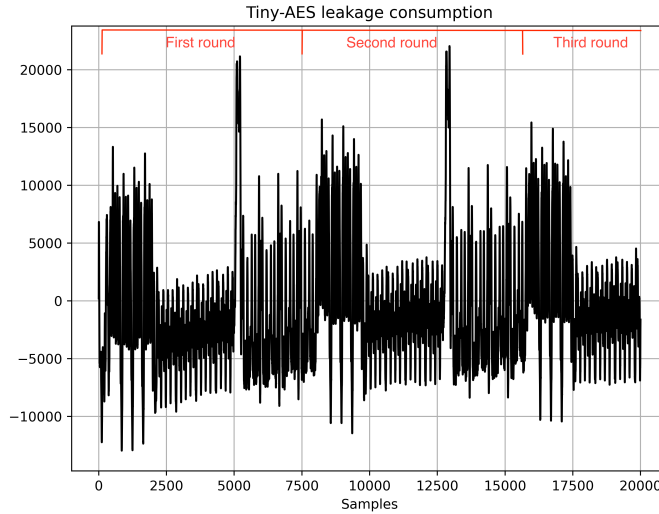


Figure 7: Power consumption of the first 3 rounds of the Tiny AES implementation

3.2.1 SNR

The SNR evaluated on the output of each S-Box (Y_i) is shown in Figure 8a for every sample in the trace. In this figure, one can distinguish 4 different sections :

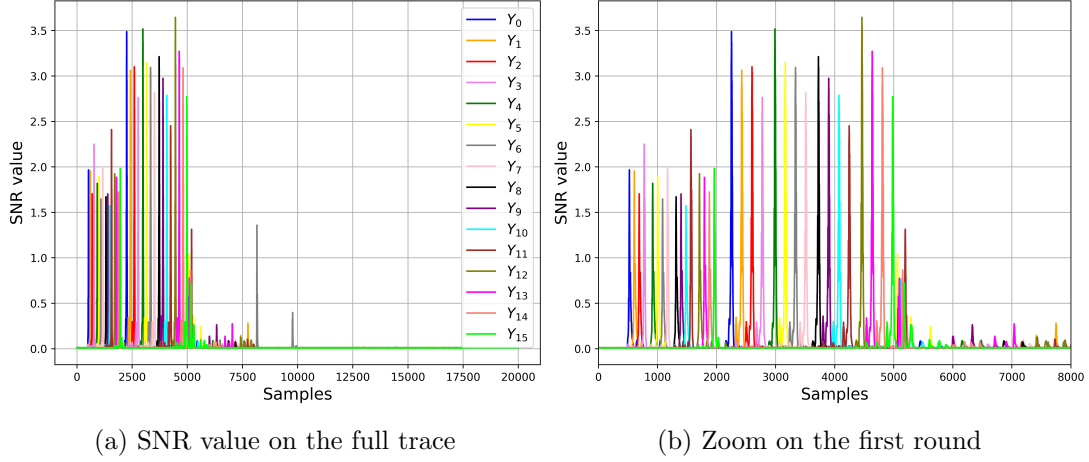


Figure 8: SNR values for each S-Box output computed on 25k leakages

- The first section, with samples between $[0, 2300]$, corresponds to moment where the AddRoundKey is performed. The S-Box is a bijective relation between its inputs (X_i) and outputs (Y_i). If one can find information on Y_i , then due to the bijective relation some information can be found on X_i . As one can see, the SNR value for the key xoring is less important compared to the output of SubBytes.
- The second section corresponds to samples between $[2300, 5050]$. It is where the LuT S-Box is performed. The S-Box is applied on each byte independently as one can see in Figure 8b. The SNR value decreases within each block of 4 consecutive bytes. It could be explained by the fact that the bytes are stored in a 4×4 matrix of `uint8_t` within a 32 bits architecture. When one byte of the matrix is manipulated, the 3 others bytes of the same row are loaded as well because of the 32 bits register size. The decrease could be linked the position of the byte in the register. Some bytes may leak more or less depending on where they are located in the register. This assumption has not been verified but we think that the decrease in SNR is more related to the position of the data in the 32 bits register than to the software execution of the AES.
- The third section of the traces, with samples between $[5050, 8000]$, corresponds to the execution of the diffusion layer which composed by ShiftRows and MixColumns. For ShiftRows, the bytes are shifted 1 by 1, it implies that 2 bytes are manipulated at the same time. For MixColumns, they are manipulated by blocks of 4. As multiple bytes are manipulated at the same time, it introduces algorithmic noise in the leakage. This is why the samples of this part of the trace have a smaller SNR value.
- The last part of the trace, with samples between $[8000, 20000]$, corresponds to the execution of the other rounds. As the SNR is computed on the output of the first S-Box, it is normal to have a SNR value of 0 for those samples. The diffusion layer and the key xoring of the second round has made the knowledge relative to Y_i disappeared in trace.

The SNR values of the AddRoundKey and SubBytes are quite important. For those operations, only one byte is manipulated by device and some variations between two same encryptions are due to electrical noise. SNR is defined as $\text{SNR} = \frac{\mathbb{V}[\mathbb{E}[P]]}{\mathbb{E}[\mathbb{V}[P+N]]} = \frac{\mathbb{V}[\mathbb{E}[P]]}{\mathbb{E}[\mathbb{V}[N]]}$ where P corresponds to the predictable signal and N to the noise. If one assumes a *Hamming Weight* (HW) leakage, then this can be rewritten as $\text{SNR} = \frac{\mathbb{V}[\text{bin}(p=0.5, n=n_p)]}{\mathbb{E}[\mathbb{V}[N]]} = \frac{0.5 \times n_p \times (1-0.5)}{\mathbb{E}[\mathbb{V}[N]]}$ where bin corresponds to a binomial. For AddRoundKey and SubBytes, the noise variations are only due to electrical noise $N = N_e$. As one computes the SNR on 1 byte ($n_p = 8$), the variations of the predictable signal $\mathbb{V}[P] = 2$. The values of the SubBytes operation have a SNR of 3.5. It means that the electrical noise variations $\mathbb{E}[\mathbb{V}[N_e]] = 0.57$. This value is quite small compare to the numerator. In addition, it is obtained for traces that are not averaged. One should obtain better SNR value for averaged traces at the condition that the electrical noise follows a gaussian distribution. In that case, the averaging should reduce the value of $\mathbb{E}[\mathbb{V}[N]]$.

For the ShiftRows operation, two bytes are manipulated at the same time, one is predictable and the other is not, therefore some bits act as algorithmic noise. Let N_a denote the algorithmic noise. Here, $n_a = 8$ and the SNR can be computed as :

$$\text{SNR} = \frac{\mathbb{V}[\mathbb{E}[P]]}{\mathbb{E}[\mathbb{V}[N_e + N_a]]} = \frac{\mathbb{V}[\text{bin}(p = 0.5, n = n_p)]}{\mathbb{E}[\mathbb{V}[N_e] + \mathbb{V}[\text{bin}(p = 0.5, n = n_a)]]} = \frac{2}{\mathbb{E}[\mathbb{V}[N_e]] + 2} \leq 1$$

with the equality when there are electrical noise variations. If one supposes that $\mathbb{E}[\mathbb{V}[N_e]]$ is the same as for the previous samples, then the SNR for ShiftRows should be $\frac{2}{2+0.57} \approx 0.8$. This approximation corresponds to the SNR values obtained for most of the peaks between [5050,5300]. For sure, one should keep in mind that this theoretical SNR value is obtained on the assumption of an HW leakage. This could explain why all samples do not have a SNR value around 0.8.

For the Mixcolumns operation, one predictable byte ($n_p = 8$) is manipulated at the same time as three others ($n_a = 24$). If one takes the electrical noise into account, then the SNR value observed can be approximated by $\text{SNR} = \frac{2}{\mathbb{E}[\mathbb{V}[N_e]]+6} \leq 0.33$ with the equality when there is no electrical noise variation. In our case, $\text{SNR} = \frac{2}{0.57+6} \approx 0.3$. This value corresponds to the SNR value of the most of the samples between [5300,8000] observed in Figure 8b. Nonetheless, some SNR peaks slightly above 0.3 can be seen in that figure. Those peaks occur only for the second byte of each row Y_i with $i \in [1, 5, 9, 13]$. They could be linked to the software code execution but we have not looked into it. The presence of the peaks for the byte Y_6 at sample 8000 and 9800 has also not been investigated and they are considered as noise in the following.

To see if the SNR analysis done previously is correct, one will look at the SNR values for averaged traces. With averaged traces the value of $\mathbb{E}[\mathbb{V}[N_e]]$ should be reduced. Therefore, the SNR values associated to the AddRoundKey and SubBytes should increase significatively. On the other hand, the SNR values of ShiftRows and MixColumns should tend to their corresponding bounds. It is illustrated in Figure 9 for traces averaged 10 times. As one can see in Figure 9a, the SNR values of the AddRoundKey and SubBytes increase as expected because the value of $\mathbb{E}[\mathbb{V}[N_e]]$ went from 0.57 to

0.14 tanks to the averaging. Figure 9b shows as well that the SNR values, for the ShiftRows and Mixcolumns, have changed and they are closer to their corresponding bound.

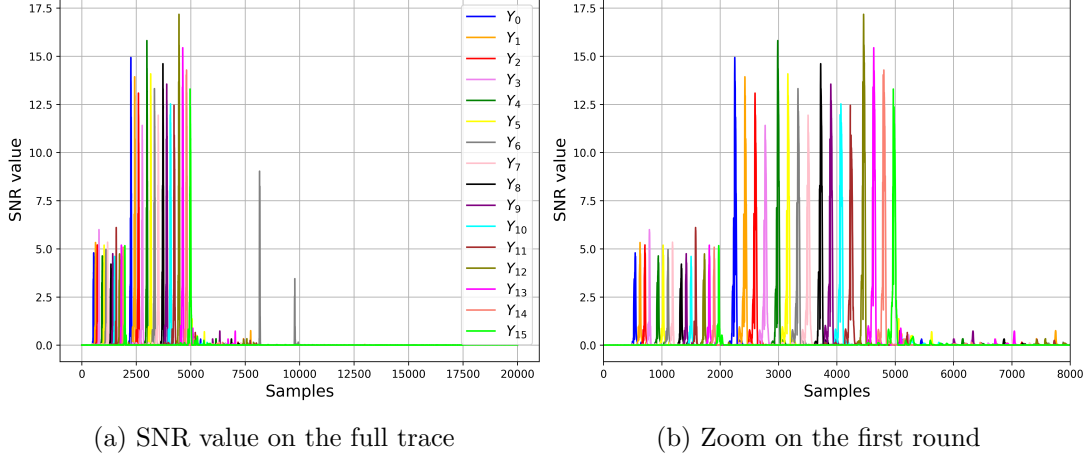
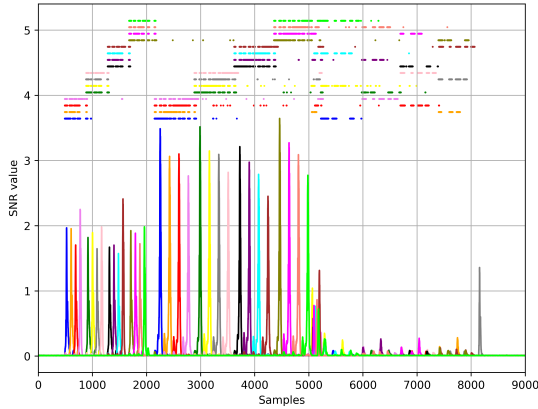


Figure 9: SNR values for each S-Box output computed on 25k leakages averaged 10 times

3.2.2 POI

In Figure 8a, there are a small number of samples with a significant SNR value for each Y_i . To take most of the relevant samples into account, one will choose the POI as the samples of the first round that have a SNR value over a given threshold SNR_{th} . The threshold used in this document corresponds to the maximal SNR value for the samples between 8000 and 20000. The SNR values obtained on this interval should correspond to the noise. Let SNR_{th} denotes the maximal SNR value on this interval. It could be assumed that SNR_{th} corresponds also to the maximum SNR value of the noise in the first round. Therefore, every sample within 0 and 8000 with a SNR value over SNR_{th} should carry information and could be used as POI. The samples selected as POI for each Y_i are represented in Figure 10 by the small points over the SNR curves. One should notice that the inexplicable peaks of Y_6 are excluded from the selection of the POI. The number of POI (n_{POI}) selected for each byte is written in Table 5.



Byte	n_{POI}	Byte	n_{POI}
Y_0	1063	Y_8	976
Y_1	868	Y_9	1029
Y_2	1121	Y_{10}	1073
Y_3	1137	Y_{11}	1105
Y_4	1083	Y_{12}	1033
Y_5	1119	Y_{13}	984
Y_6	1056	Y_{14}	1011
Y_7	1091	Y_{15}	1067

Figure 10: POI sample selection on the first Table 5: Number of POI selected for each round of the trace byte

The number of POI for each round is quite similar. It is around 1000 samples. By contrast, the first round consists of around 8000 samples. It means that in the trace only 1 sample out of 8 is useful to find information about a given byte. It is not a lot, but hopefully, the information carried in the sample is quite important. This is shown by the high value of SNR. In [24], it is shown that the correlation coefficient between the predictable transitions (P) and the global transitions ($G = P + N$) is proportional to the SNR value such as :

$$r_{P,G} = \frac{1}{\sqrt{1 + \frac{1}{\text{SNR}}}} \quad (3)$$

Based on this equation and the value of the SNR, one can see that the correlation coefficient has a value up to 0.9 for some samples. It indicates that for those samples, the information is highly correlated with the leakage. To sum up, although the number of POI is relatively small, they carry a lot of information.

3.2.3 LDA-based templates

The POI selected can be used to construct the LDA-based templates. To evaluate their quality, one can look into the MI contained in each template. This MI represents the information between the leakage and the output of the S-Box. As explained in Section 1.4.7, the MI cannot be computed, but it can be bound by the PI and TI. Moreover, with enough training traces, the PI and TI should converge toward the MI. The convergence of the PI and TI is illustrated for 2 bytes in Figure 11 for different values of n_{lda} representing the number of dimensions kept after the LDA projection. The 16 templates can be seen in Figure 22. In this figure, one can see that for any value of n_{lda} , the convergence of the PI and TI is achieved after 20k traces of profiling. The traces used during the profiling have random plaintexts and random keys. For the 16 templates, the PI is always increasing. It means that with more training traces, the model gives more information until it saturates. On the contrary, the TI stays almost constant for most of the templates. It shows that the model does not overfit the TI when a small number of traces are used during the training phase.

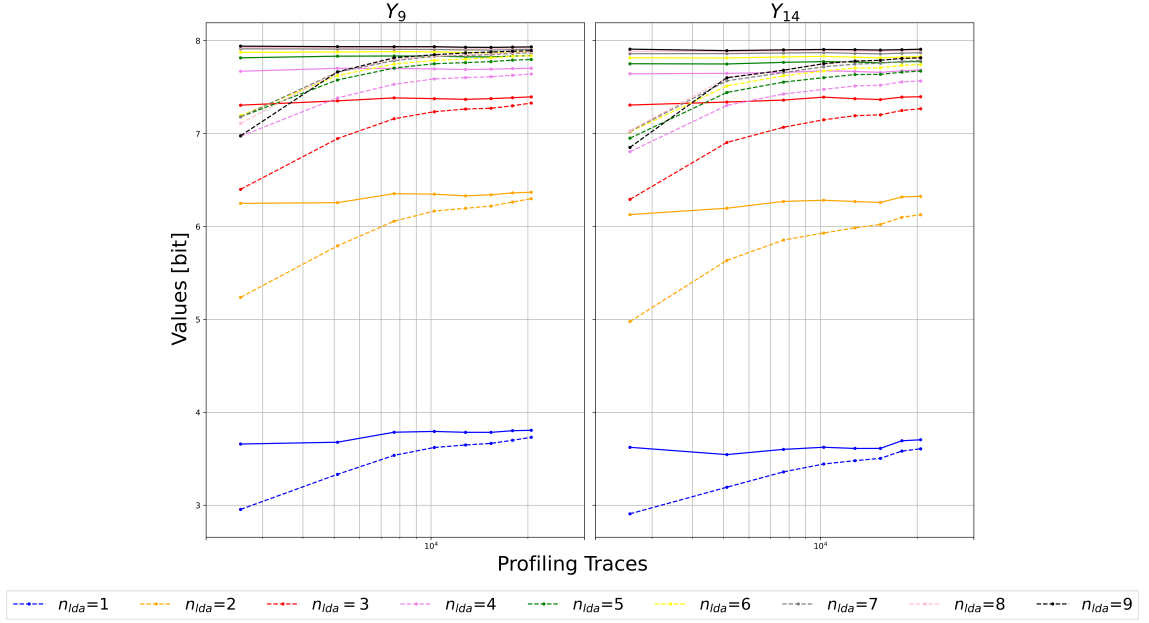


Figure 11: PI and TI convergence of the LDA-based templates of Y_9 and Y_{14} for different n_{lda} values. The PI, depicted by the dashed lines, is evaluated on 5k traces and the TI is drawn with plain lines.

It can be seen that the convergence value of the PI and the TI, in other word the MI, varies for different value of n_{lda} until saturation. At first sight, one could say that a small n_{lda} value should give good results for the MI because the first dimension of the LDA should carry more information than the following ones. But this is not the case. Figure 11 shows that higher dimensions also carry some information as n_{lda} increases, so does MI. The reason of this behavior is illustrated in Figure 12. The figure shows the different eigenvectors of the LDA projection matrix of Y_5 for the samples of the first round.

In this figure one can see that some eigenvectors, representing the dimensions, with a higher value still carry information for the POI. This is the case for the eigenvectors 7, 8 and 9 which can give information for the POI around sample 3800. Therefore, to find the good value of n_{lda} , one has to increase its value until the MI saturates. In Figure 11, the MI stops to increase after $n_{lda} = 8$. For this value, the MI is almost equals to 8 bits for all templates. It means that if an attacker uses one of the templates on a new leakage, he should retrieve the 8 bits corresponding to the S-Box output of that leakage. In other word, the attacker could retrieve the byte associated to the leakage with only one leakage.

3.2.4 Rank estimation

Once the PI and TI have converged and increasing n_{lda} no longer improves the MI, the attacker can build the templates for each byte. Then, he can use those templates on new leakage for which he only knows the plaintext. As explained in Section 1.5, he can retrieve the probabilities vectors associated to the sub-key thanks to one or multiple

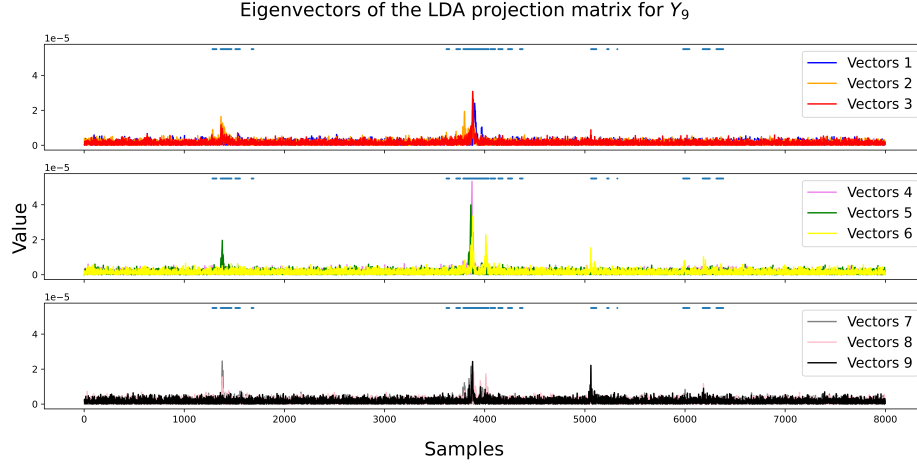


Figure 12: First 9 eigenvectors of the LDA matrix projection for the byte Y_9 . The blue points correspond to the selected POI.

traces. Those multiple traces use the same encryption key but they have different plaintexts. At the end, he gets 16 vectors, one for each sub-key. Each index of the vector corresponds to one of the 256 sub-keys. The value associated with the index represents the probability that the leakage was encrypted with that particular sub-key. To see if the attacker can find the full encryption key, one can evaluate the rank of the key based on the 16 vectors. The key rank of the attack on Tiny-AES is shown in Figure 13 for multiple attack traces. The graphs shows the median key rank after 200 executions of the attack on different traces with the same encryption key. If the median key rank is below 2^{32} , drawn in brown, then enumeration can be performed to retrieve the key and the implementation is not resilient to the SCA.

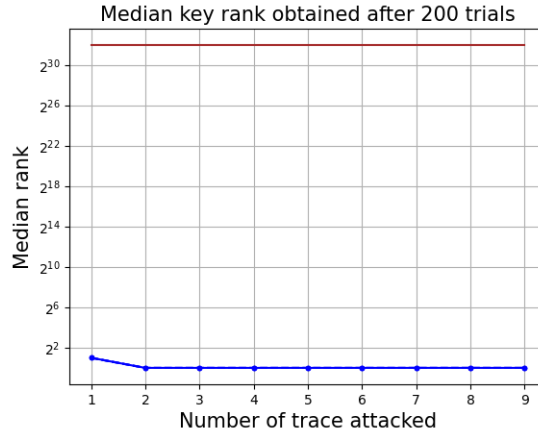


Figure 13: Median key rank after 200 executions for different number of attack traces. The templates are built with 20k training traces and $n_{lda} = 8$

As one can see in the figure, only 1 trace is needed to break the implementation. This is what was expected from the amount of MI present in the templates. For each of the

byte, the templates should give 8 bits of information with a new trace. Therefore, the entire key can be found with only one trace.

3.3 Fixsliced AES

The leakage of the Fixsliced AES implementation can be seen in Figure 14. The repetitions in the leakage are not as clear as in the Tiny-AES implementation. It is because there are 4 different implementations of the MixColumns operation. The first MixColumns is executed during the first round, the second MixColumns implementation during the second round and so on and this sequence is repeated each 4 rounds. The different MixColumns implementations can be seen in the leakage due to their difference in time execution. Like in the first implementation, the key expansion is performed before the recording, but this time, the bit packing of the plaintext is caught in the measurements. It takes place during the first 1000 samples of the trace. Those samples will be not used in the choice of the POI to provide a fair comparison with the other implementation. The code is designed for speed execution. This can be seen in the figure as it executes 2 times more rounds than Tiny-AES for the same number of samples.

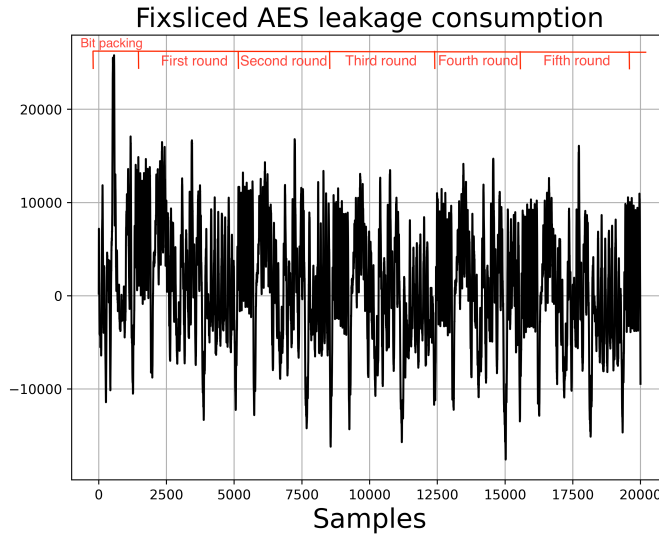


Figure 14: Power consumption of the first 5 rounds of the Fixslice implementation

The fixsliced implementation can encrypt 2 plaintexts with 2 different keys. For the sake of simplicity, one has chosen to encrypt 2 times the same plaintext with the same key. Compared to what was said in the global description of the code in Section 2.2, this choice implies that for each register it is possible to model 2 bits instead of 1 amongst the 32. Therefore, the ratio between predictable bits and bits acting as algorithmic noise is equals to $1/15$ instead of $1/31$.

The bit packing of the fixsliced implementation does not fill the registers with the bytes in a ascending order. The corresponding indexing is shown in Table 6 for the output of the S-Box. One should look at this table as an illustration to see where each

byte is located in the register and not for its strict representation. In this table, the Y_i represents 2 same bits used within a 32 bits register.

32 bits															
B^0				B^1				B^2				B^3			
Y_3	Y_7	Y_{11}	Y_{15}	Y_2	Y_6	Y_{10}	Y_{14}	Y_1	Y_5	Y_9	Y_{13}	Y_0	Y_4	Y_8	Y_{12}

Table 6: Illustration of the register indexing for fixsliced implementation

3.3.1 SNR

In this implementation, the 16 S-Boxes are executed in parallel. For the sake of readability, the SNR is represented in Figure 15 for each byte of the register. In this figure, only the SNR values of the first round are depicted because for the other rounds this value is close to 0. The figure clearly shows that more or less information is leaked

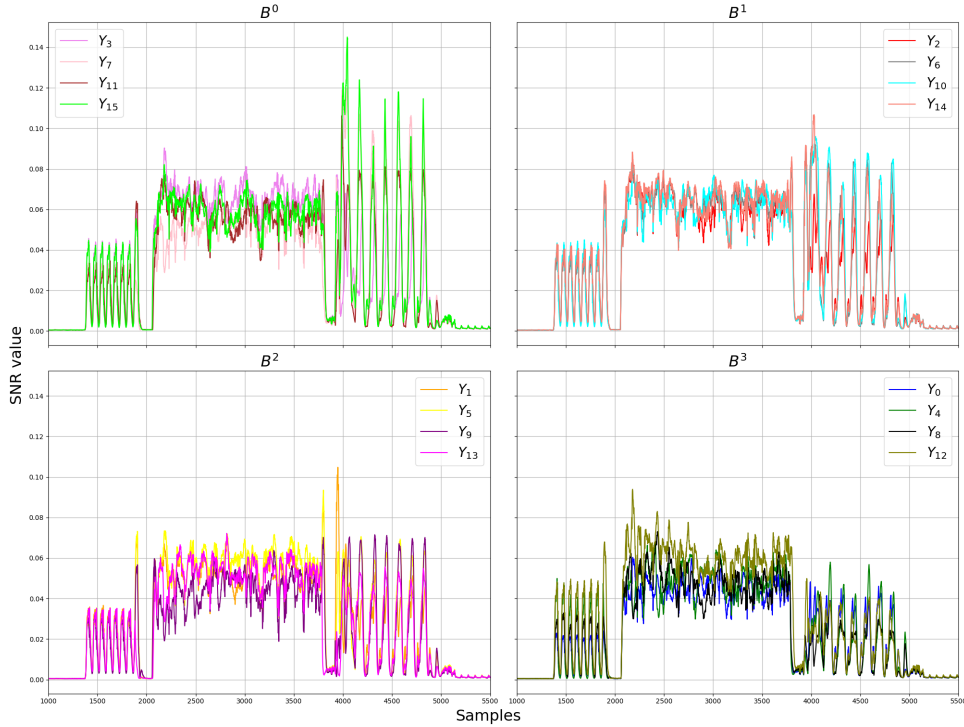


Figure 15: SNR values for each S-Box output computed on 500k leakages

depending on where the bits are stored in the register. The SNR curves of Y_i within each B^j are quite similar. On the contrary, the SNR curves between two different B^j do not look the same. It shows that our hypothesis, which was that some bits of the register leak more information than other due to their position, could be true. In the same figure, one can see that the SNR values are more than 10 times smaller than with the Tiny-AES implementation. The reason of this decrease is explained in the following.

In this figure, one can distinguish 4 different sections :

- The first section, with samples between [1400,2000], corresponds to the AddRoundKey operation. The 8 peaks correspond to the 8 key xoring, one for each register. The higher value for the last peak cannot be explained as the key xoring is executed as a loop. This may be due to the behavior of the CPU. After the last iteration, he could empty the register before the program continues its execution. Therefore, it could give more information than during the other executions of the loop but this assumption has not been verified.
- The second section, with samples between [2000, 3900], corresponds to the bit-sliced S-Box execution. The bitsliced S-Box manipulates the 8 registers altogether to perform the SubBytes operation. It executes several logical functions on different pairs of registers and the functions are executed in an order that may seem random at first glance. Those operations are the reason why the shape of the SNR values does not follow any particular pattern in the figure.
- The third section, with samples between [3900, 5000], corresponds to the execution of the first MixColumns implementation. In this section, the 8 peaks correspond to the execution of a XOR between 2 registers. The SNR values in between the peaks correspond to the execution of more complex instructions using multiple registers at the same time. The SNR decrease between the 4 graphs could be explained by the position of B^j in the register.
- The fourth section, with samples between [5000, 20000], corresponds to the execution of the other rounds. For this part of the signal, the samples have SNR values close to 0. After MixColumns, some information can still be found in the signal. It is illustrated by the small ripples after sample 5000, but their SNR values are too small compared to the rest of the signal. Thus, it is not useful to take those samples into account for the choice of the POI.

For the AddRoundKey operation, all SNR values fall below 0.06, apart for the last xoring operation. During this operation, each register manipulates 2 bits of predictable signal ($n_p = 2$) for 30 bits of algorithmic noise ($n_a = 30$). If the leakage follows a HW distribution, this gives a theoretical SNR value of :

$$\text{SNR} = \frac{\mathbb{V}[\text{bin}(p = 0.5, n = n_p)]}{\mathbb{E}[\mathbb{V}[N_e]] + \mathbb{V}[\text{bin}(p = 0.5, n = n_a)]} = \frac{0.5}{\mathbb{E}[\mathbb{V}[N_e]] + 7.5} \leq 0.066$$

This is what is observed for each Y_i in Figure 15. Nevertheless, this theoretical value is obtained under the assumption of a HW leakage which could not always be the case.

For the bitsliced S-Box and the MixColumns operations, every operation manipulates two or more registers at the same time. It means that the theoretical SNR value, assuming a HW leakage, should also be at most 0.066. On one hand, the SNR values for the bitsliced S-Box are around this value. For some Y_i it is a bit over 0.066 but this could be linked to the position of the Y_i within the register. On the other hand, the SNR values of MixColumns are sometimes much higher than 0.066. Therefore, the

assumption of HW leakage is not really valid for this part of the signal.

The leakage of MixColumns does not tend to follow a HW distribution. Nevertheless, one can still use Equation 3 to give an indication on the correlation between the leakage and the predictable transitions to see if the samples of this part are useful or not. The correlation coefficients fall between 0.2 and 0.35 depending on which byte and samples is considered in the trace. This means that, for the samples between [3900,5000], there is a correlation between the manipulated bits and the leakage. This correlation coefficient is not as high as for Tiny-AES but it still indicates that the information known for those samples are correlated to the leakage.

To see if the SNR curves become better with less electrical noise, one has performed the same SNR computation on traces averaged 10 times and this is shown in Figure 16.

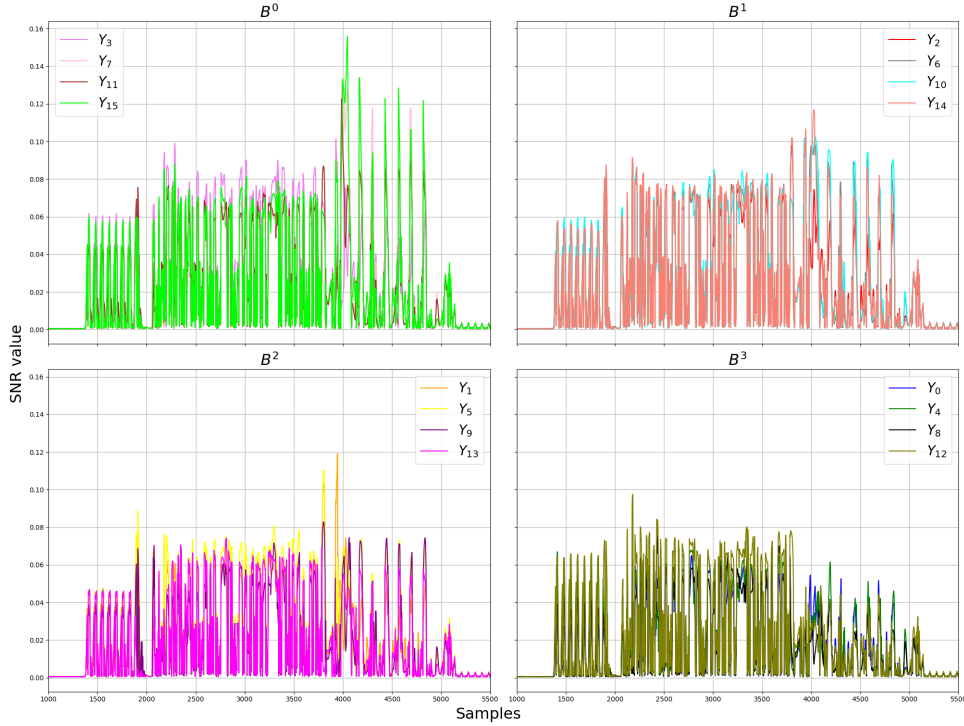


Figure 16: SNR values for each S-Box output computed on 500k leakages averaged 10 times

When the electrical noise is reduced, one can see that the SNR values increase for all samples of AddRoundKey and get closer to 0.066. For the bitsliced S-Box, one can see where the different registers are manipulated compared to the SNR curves of Figure 15. Averaging has reduced the electrical noise and allowed a better estimation of the SNR. Moreover, those value of SNR are now much closer to 0.066 for most of the bytes. with some exceptions for the Y_i of B^0 and B^1 . The MixColumns operation is still clearly visible with the 8 different peaks, but now the samples in between those peaks have a much important SNR values than before. It means that those samples contain as well

some information about the leakage that was not possible to see without averaging.

3.3.2 POI

The SNR values of the different samples in the trace are quite small compared to the previous implementation. However, in this case, because all S-Boxes are performed in parallel, there is a greater number of points with a relevant SNR value for each Y_i . In order not to miss information contained by some samples with smaller SNR values, one has chosen to take all samples of the first round, points between [1400,5000], as POI.

One could also have chosen the POI as previously. By computing SNR_{th} on the samples from the other rounds and taking all the samples within [1400,5000] with a SNR value greater than SNR_{th} . This has been done and the difference between both approaches is quite small. With the POI selection, one loses some information due to some informative samples that are not taken into account and it results in a smaller difference of MI after convergence of the templates.

3.4 LDA-based templates

The PI and TI convergence of the templates, using all samples of the first round as POI, is shown for byte Y_9 and Y_{14} in Figure 17 for different values of n_{lda} . The templates for the other bytes are shown in Figure 23. One can see in the figure that the

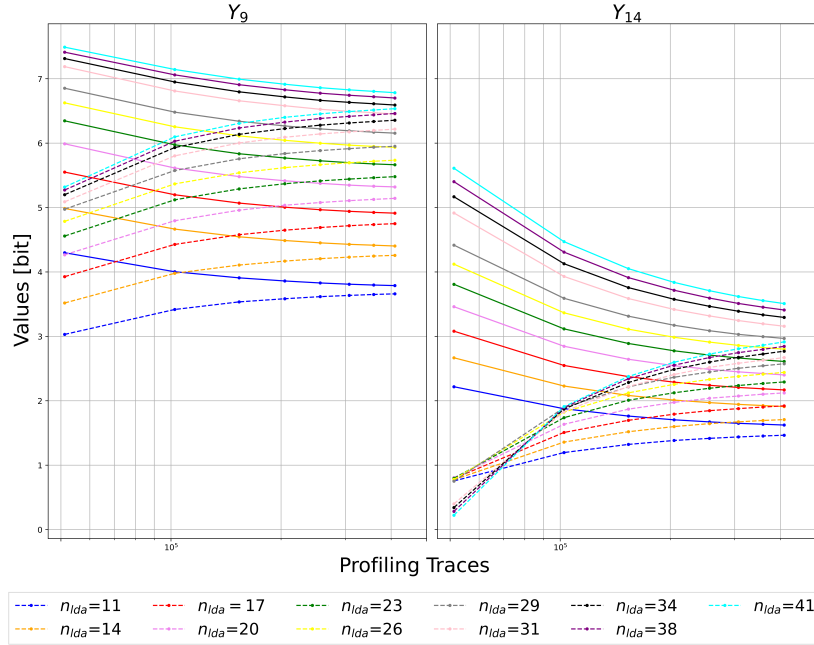


Figure 17: PI and TI convergence of the LDA-based templates of Y_9 and Y_{14} for different n_{lda} values. The PI, depicted by the dashed lines, is evaluated on 50k traces and the TI is drawn with plain lines.

convergence is achieved after 400k traces for most of the templates. Some templates, with a higher value of n_{lda} , do not converge precisely after 400k profiling traces. They would need a bit more training traces to converge toward the MI. In those cases, the gap between the PI and TI after 400k profiling traces is less than 0.5 bit. Thus, we assumed that those templates have also converged. In the figure, the PI (resp. TI) increases (resp. decreases) with respect to the number of profiling traces for all templates. It means that the MI presents in the templates is better approximated with more and more training traces.

In the figure, the values of the n_{lda} are bigger than those for Tiny-AEs. It can be explained by the number of POI which is bigger than before. More POI means that some samples could carry smaller amount of information. To take those samples into account one has to increase the number of dimensions. In addition, the maximum value of MI after convergence is also higher when n_{lda} is increased. It is because the higher dimension still have information. This can be seen in Figure 18 for the difference between $n_{\text{lda}} = 11, 14$ or 17 . In this figure, one can see that there is still information in the higher eigenvectors of the matrix projection than if one only kept the first 11 ones.

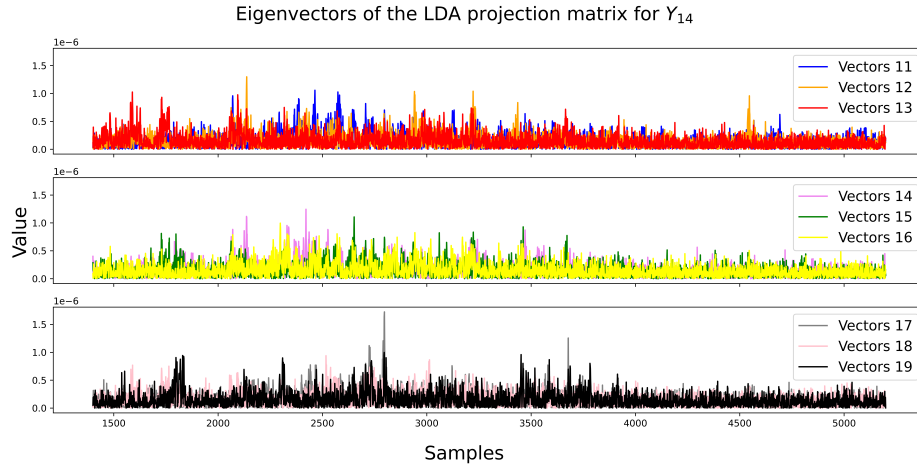


Figure 18: LDA projection matrix eigenvectors for the byte Y_{14} in the fixslided implementation

In Figure 23, one can see that after a given n_{lda} value, the MI starts to saturate and the saturation value is not the same for all bytes. For instance, the template for the byte Y_9 has a MI two times bigger than the one for the template of the byte Y_{14} as illustrated in Figure 23. The reason to this come from the value of the eigenvectors of the projection matrix. For byte Y_{14} , the maximal value of Figure 18 is 1.5×10^{-6} , the maximal value of the eigenvectors of Y_9 , shown in Figure 19, is 3×10^{-6} . As the values of the eigenvectors are 2 times more important for Y_9 than for Y_{14} , the MI is 2 times bigger in the first case than in the other one.

In Section 3.3.2, it has been said that the selection of POI can reduced to total MI in the templates. Figure 24 shows the PI and TI convergence with POI selection. If

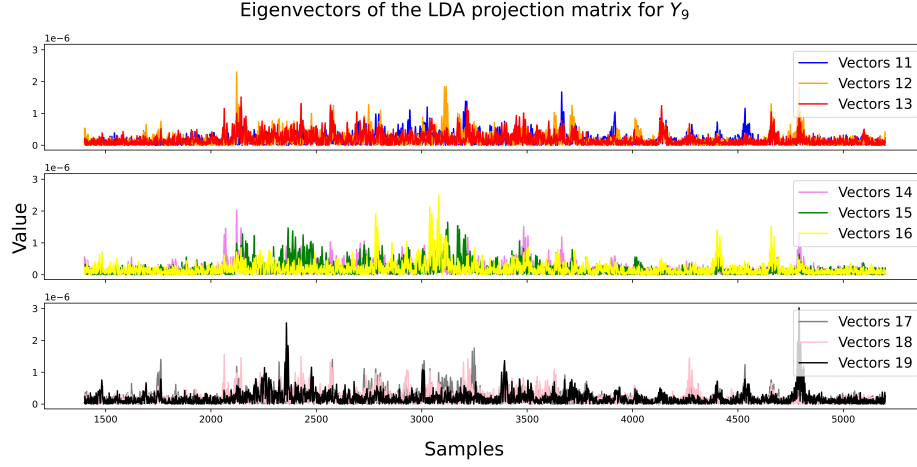


Figure 19: LDA projection matrix eigenvectors for the byte Y_9 in the fixslided implementation

one compares the different values with those of Figure 23 without POI selection, one can see that the POI selection reduces the MI in the templates. This reduction is not important but it shows that sometimes it is better to use all samples of the trace than to select some sample to speed up the calculations.

3.4.1 Rank estimation

One can use the templates obtained with 400k training traces and $n_{lda} = 41$ on new traces to see if an attacker can find the encryption key used for those leakages. The median key rank of the attack is illustrated in Figure 20. One can see that the attacker

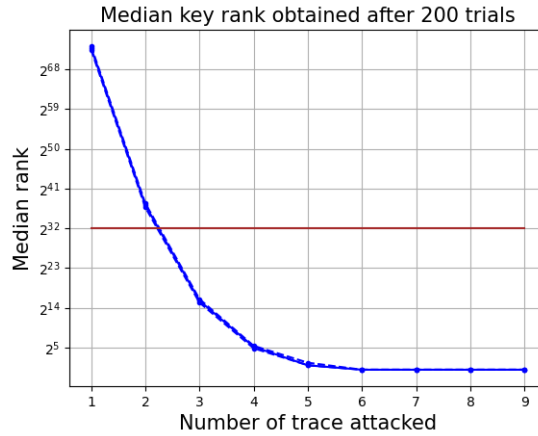


Figure 20: Median key rank after 200 executions for different number of attack traces. The upper and lower bounds of the median key rank correspond to the dashed lines. The templates are built with 400k training traces and $n_{lda} = 41$

needs at least 3 traces to be able to recover the key because after 3 traces the key rank is below 2^{32} and enumeration is possible. The reason of the 3 traces could come from

the MI of the byte 14. For this byte, the $MI = 3$. It means that with a new trace one, the template should be able to find 3 bits out of the eight for K_{14} . Therefore one would need at least 3 traces, in the best case, to find the full 8 bits of K_{14} . To verify this assumption, let's look at the byte median key ranking. It represents the same thing as the key ranking but the rank estimation is performed on each byte independently instead of the whole key. This byte key ranking is shown in Figure 21. One can see

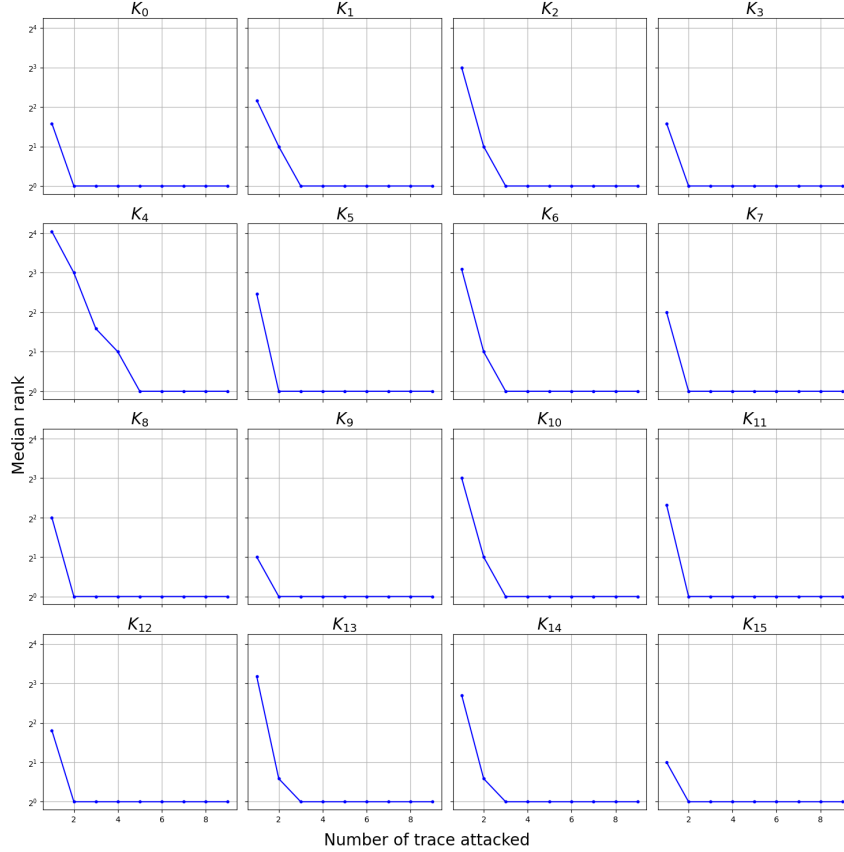


Figure 21: Median byte key rank after 200 executions for different number of attack traces. The templates are built with 400k training traces and $n_{lda} = 41$.

that 3 traces are needed for the median rank of K_{14} to be equals to 1 which is what was expected. It is interesting to note that the rank of K_4 requires more attack traces than K_{14} to reach the value of 1. This would not have been expected as the MI for this byte is 4.3.

3.5 Comparison between the implementations

Both implementations were attacked by the same SCA. The different parameters and results obtained for each implementation are summarized in Table 7.

For the Tiny-AES implementation, the attack focused a small amount of POI that carried a lot of information as indicated by the SNR value. The number of training

Implementation	SNR max	n_{POI}	Training traces	n_{lda}	MI	$n_{\text{trace attacked}}$
Tiny-AES	3.5	≈ 1000	20k	8	8	1
Fixsliced AES	0.14	3600	400k	41	≈ 3.5	3

Table 7: Summary of the different parameters and results between both attacks

traces required for the model to converge toward the maximal MI was relatively low and the full key could be retrieved using only one trace. This type of attack can be considered as "vertical" because it emphasizes the quality of the information carried by the selected POI rather than the quantity of samples. The attack was easily performed thanks to the high-quality of the leakage caused by the basic implementation of AES which do not use countermeasures.

On the other hand, the attack on the fixsliced AES implementation took into account a larger number of POI since obtaining high SNR values was not feasible. As a result, more training traces were required to converge toward a smaller MI. However, the number of attack trace needed to recover the full key was not significantly higher than for the Tiny-AES implementation. This attack can be considered as a "horizontal". Due to the low SNR value, the attacker focused on increasing the number of samples used to construct the templates rather than relying on the quality of individual samples. In this case, the attacker prioritized quantity over quality. A "vertical" attack would not have achieved the same results because the fixsliced implementation leverages the register size to minimize the amount of information leaked during measurements.

Overall, the results indicate that the Tiny-AES implementation is more vulnerable to the side-channel attack due to the higher quality of the leakage, whereas the fixsliced AES implementation provided a better resistance against the attack thanks to algorithmic noise. Nonetheless, both of them can be broken with a enough measurements.

Conclusion

This thesis aimed to evaluate the resilience of AES software implementations on micro-controllers against a powerful divide and conquer SCA. The results demonstrated that the basic implementation of Tiny-AES was found to be vulnerable as the key could be broken with one leakage. On the other hand, the fixsliced AES, which incorporates countermeasures such as algorithmic noise, provided a better resilience, as more training traces were needed to break the implementation.

It is important to note that the attack only modeled 8 bits of the signal. By expanding the number of bits, such as using 16, 24, or 32 bits, some parameters like the number of training traces or the dimension of the LDA could potentially be reduced. Increasing the number of bits modeled helps to reduce the impact of algorithmic noise and thus improves the accuracy of the templates, but at bigger computational cost.

Furthermore, an alternative attack strategy, that could have been explored, is the *Soft Analytical Side-Channel Attacks* (SASCA). This attack method, as described in [27], leverages the probabilities obtained by the divide and conquer attack and considers all bytes simultaneously. By incorporating the analytical strategies for key recovery with the traditional divide and conquer approach, SASCA has demonstrated good performances in side-channel attacks that could improve the results obtained.

Our results indicate that relying solely on algorithmic noise is insufficient to ensure resilience against SCA with software implementations. While algorithmic noise, such as in the fixsliced AES implementation, provides some level of protection, it is not enough on its own. Additional countermeasures could be explored in conjunction with algorithmic noise to enhance security. One possible improvement is the use of countermeasures like masking, shuffling, or noise addition. These techniques aim to further obfuscate the sensitive data and reduce the leakage of information. However, it is important to note that even advanced countermeasures, like software masking on 32-bit architectures, have been shown to be susceptible to attacks as demonstrated in [4]. Therefore, while these countermeasures can provide better protection, they may not be enough to ensure the resilience of software implementations against sophisticated SCA.

In order to significantly increase the resilience of implementations on micro-controllers, a potential solution lies in executing the encryption using dedicated hardware accelerators. Specialized hardware should reduce the leakage of sensitive information and thereby enhance the resilience. However, simply relying on hardware implementation alone may still leave the device vulnerable, as highlighted in [25].

To increase the resilience of AES implementations on micro-controllers, the aforementioned countermeasures, such as masking and noise addition, could be applied on the hardware accelerators as emphasized in [5]. This combined approach, leveraging hardware acceleration with countermeasures, offers a more robust solution to mitigate the leakage of information and should improve the overall security of AES implementations on micro-controller.

References

- [1] Alexandre Adomnicai and Thomas Peyrin. “Fixslicing AES-like Ciphers: New bitsliced AES speed records on ARM-Cortex M and RISC-V”. In: *IACR Transactions on Cryptographic Hardware and Embedded Systems* 2021.1 (Dec. 2020), pp. 402–425. DOI: 10.46586/tches.v2021.i1.402-425. URL: <https://tches.iacr.org/index.php/TCHES/article/view/8739>.
- [2] Elaine Barker and Allen Roginsky. *Transitioning the Use of Cryptographic Algorithms and Key Lengths*. en. Mar. 2019. DOI: <https://doi.org/10.6028/NIST.SP.800-131Ar2>.
- [3] Joan Boyar and Rene Peralta. *New logic minimization techniques with applications to cryptology*. Cryptology ePrint Archive, Paper 2009/191. <https://eprint.iacr.org/2009/191>. 2009. URL: <https://eprint.iacr.org/2009/191>.
- [4] Olivier Bronchain and François-Xavier Standaert. “Breaking Masked Implementations with Many Shares on 32-bit Software Platforms: or When the Security Order Does Not Matter”. In: *IACR Transactions on Cryptographic Hardware and Embedded Systems* 2021 (July 2021), pp. 202–234. DOI: 10.46586/tches.v2021.i3.202-234.
- [5] Olivier Bronchain et al. “Improved leakage-resistant authenticated encryption based on hardware AES coprocessors”. In: *IACR Transactions on Cryptographic Hardware and Embedded Systems* (2021), pp. 641–676.
- [6] Olivier Bronchain et al. “Leakage Certification Revisited: Bounding Model Errors in Side-Channel Security Evaluations”. In: *Advances in Cryptology – CRYPTO 2019*. Ed. by Alexandra Boldyreva and Daniele Micciancio. Cham: Springer International Publishing, 2019, pp. 713–737. ISBN: 978-3-030-26948-7.
- [7] Suresh Chari, Josyula R. Rao, and Pankaj Rohatgi. “Template Attacks”. In: *Cryptographic Hardware and Embedded Systems - CHES 2002, 4th International Workshop, Redwood Shores, CA, USA, August 13-15, 2002, Revised Papers*. Vol. 2523. Lecture Notes in Computer Science. Springer, 2002, pp. 13–28. DOI: 10.1007/3-540-36400-5_3.
- [8] Omar Choudary and Markus G. Kuhn. *Efficient Template Attacks*. Cryptology ePrint Archive, Paper 2013/770. <https://eprint.iacr.org/2013/770>. 2013. URL: <https://eprint.iacr.org/2013/770>.
- [9] Joan Daemen and Vincent Rijmen. *The Design of Rijndael: AES - The Advanced Encryption Standard (Information Security and Cryptography)*. 1st ed. Springer, 2002. ISBN: 3540425802.
- [10] François Durvaux, François-Xavier Standaert, and Nicolas Veyrat-Charvillon. “How to Certify the Leakage of a Chip?” In: *Advances in Cryptology – EURO-CRYPT 2014*. Ed. by Phong Q. Nguyen and Elisabeth Oswald. Berlin, Heidelberg: Springer Berlin Heidelberg, 2014, pp. 459–476. ISBN: 978-3-642-55220-5.
- [11] Morris Dworkin et al. *Advanced Encryption Standard (AES)*. en. Nov. 2001. DOI: <https://doi.org/10.6028/NIST.FIPS.197>.

- [12] Karine Gandolfi, Christophe Mourtel, and Francis Olivier. “Electromagnetic Analysis: Concrete Results”. In: *Cryptographic Hardware and Embedded Systems - CHES 2001, Third International Workshop, Paris, France, May 14-16, 2001, Proceedings*. Vol. 2162. Lecture Notes in Computer Science. Springer, 2001, pp. 251–261. DOI: 10.1007/3-540-44709-1_21.
- [13] Benedikt Gierlichs et al. “Mutual Information Analysis”. In: *Cryptographic Hardware and Embedded Systems – CHES 2008*. Ed. by Elisabeth Oswald and Pankaj Rohatgi. Berlin, Heidelberg: Springer Berlin Heidelberg, 2008, pp. 426–442. ISBN: 978-3-540-85053-3.
- [14] Jos Luis Imana, Romn Hermida, and Francisco Tirado. “Low Complexity Bit-Parallel Multipliers Based on a Class of Irreducible Pentanomials”. In: *IEEE Transactions on Very Large Scale Integration (VLSI) Systems* 14.12 (2006), pp. 1388–1393. DOI: 10.1109/TVLSI.2006.887835.
- [15] Paul Kocher, Joshua Jaffe, and Benjamin Jun. “Differential Power Analysis”. In: *Advances in Cryptology — CRYPTO’ 99*. Ed. by Michael Wiener. Berlin, Heidelberg: Springer Berlin Heidelberg, 1999, pp. 388–397. ISBN: 978-3-540-48405-9.
- [16] Paul C. Kocher. “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*. Vol. 1109. Lecture Notes in Computer Science. Springer, 1996, pp. 104–113. DOI: 10.1007/3-540-68697-5_9.
- [17] Stefan Mangard. “Hardware Countermeasures against DPA – A Statistical Analysis of Their Effectiveness”. In: *Topics in Cryptology – CT-RSA 2004*. Ed. by Tatsuaki Okamoto. Berlin, Heidelberg: Springer Berlin Heidelberg, 2004, pp. 222–235. ISBN: 978-3-540-24660-2.
- [18] Loïc Masure et al. *Information Bounds and Convergence Rates for Side-Channel Security Evaluators*. Cryptology ePrint Archive, Paper 2022/490. <https://eprint.iacr.org/2022/490>. 2022. URL: <https://eprint.iacr.org/2022/490>.
- [19] Alfred J. Menezes, Paul C. van Oorschot, and Scott A. Vanstone. *Handbook of Applied Cryptography*. CRC Press, 2001. URL: <http://www.cacr.math.uwaterloo.ca/hac/>.
- [20] Romain Poussier, François-Xavier Standaert, and Vincent Grosso. “Simple Key Enumeration (and Rank Estimation) Using Histograms: An Integrated Approach”. In: vol. 9813. Aug. 2016, pp. 61–81. ISBN: 978-3-662-53139-6. DOI: 10.1007/978-3-662-53140-2_4.
- [21] Bruce Schneier. *Applied Cryptography*. 2nd. Wiley, 1996. ISBN: 0471117099.
- [22] François-Xavier Standaert and Cedric Archambeau. “Using Subspace-Based Template Attacks to Compare and Combine Power and Electromagnetic Information Leakages”. In: *Cryptographic Hardware and Embedded Systems – CHES 2008*. Ed. by Elisabeth Oswald and Pankaj Rohatgi. Berlin, Heidelberg: Springer Berlin Heidelberg, 2008, pp. 411–425. ISBN: 978-3-540-85053-3.

- [23] François-Xavier Standaert, Tal G. Malkin, and Moti Yung. “A Unified Framework for the Analysis of Side-Channel Key Recovery Attacks”. In: *Advances in Cryptology - EUROCRYPT 2009*. Ed. by Antoine Joux. Berlin, Heidelberg: Springer Berlin Heidelberg, 2009, pp. 443–461. ISBN: 978-3-642-01001-9.
- [24] François-Xavier Standaert et al. “An Overview of Power Analysis Attacks Against Field Programmable Gate Arrays”. In: *Proceedings of the IEEE* 94 (Mar. 2006), pp. 383–394. DOI: 10.1109/JPROC.2005.862437.
- [25] Balazs Udvarhelyi et al. “On the Security of Off-the-Shelf Microcontrollers: Hardware Is Not Enough”. In: *Smart Card Research and Advanced Applications*. Ed. by Pierre-Yvan Liardet and Nele Mentens. Cham: Springer International Publishing, 2021, pp. 103–118. ISBN: 978-3-030-68487-7.
- [26] Nicolas Veyrat-Charvillon, Benoît Gérard, and François-Xavier Standaert. *Security Evaluations Beyond Computing Power: How to Analyze Side-Channel Attacks you Cannot Mount?* Cryptology ePrint Archive, Paper 2012/578. <https://eprint.iacr.org/2012/578>. 2012. URL: <https://eprint.iacr.org/2012/578>.
- [27] Nicolas Veyrat-Charvillon, Benoît Gérard, and François-Xavier Standaert. “Soft Analytical Side-Channel Attacks”. In: *Advances in Cryptology – ASIACRYPT 2014*. Ed. by Palash Sarkar and Tetsu Iwata. Berlin, Heidelberg: Springer Berlin Heidelberg, 2014, pp. 282–296. ISBN: 978-3-662-45611-8.
- [28] Nicolas Veyrat-Charvillon et al. *An optimal Key Enumeration Algorithm and its Application to Side-Channel Attacks*. Cryptology ePrint Archive, Paper 2011/610. <https://eprint.iacr.org/2011/610>. 2011. URL: <https://eprint.iacr.org/2011/610>.
- [29] Johannes Wolkerstorfer, Elisabeth Oswald, and Mario Lamberger. “An ASIC Implementation of the AES SBoxes”. In: *Topics in Cryptology — CT-RSA 2002*. Ed. by Bart Preneel. Berlin, Heidelberg: Springer Berlin Heidelberg, 2002, pp. 67–78. ISBN: 978-3-540-45760-2.

Appendices

A LuT based AES

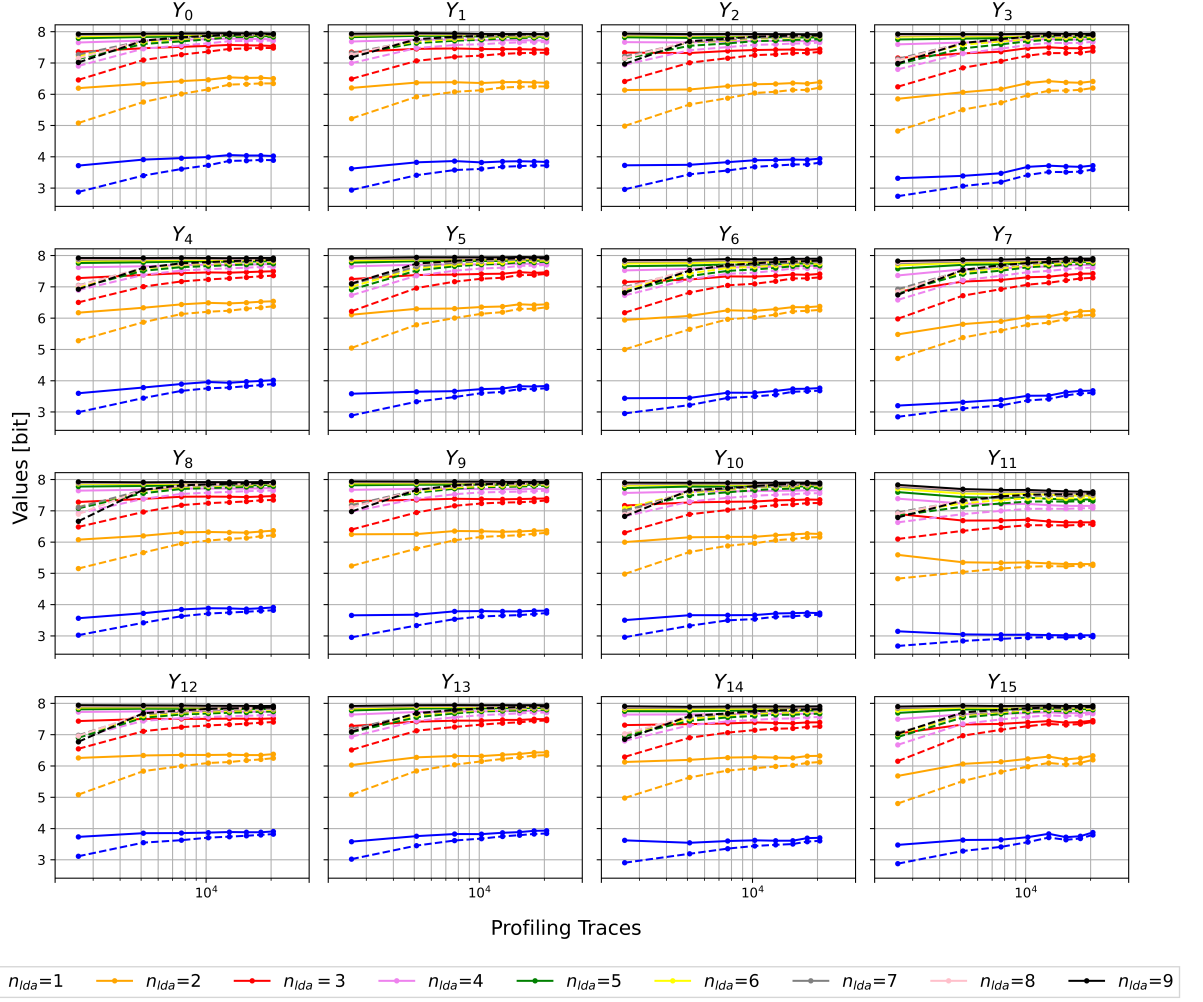


Figure 22: PI and TI convergence of the 16 LDA-based templates for different n_{lda} values. The PI, depicted by the dashed lines, is evaluated on 5k traces and the TI is drawn with plain lines.

B Fixsliced AES

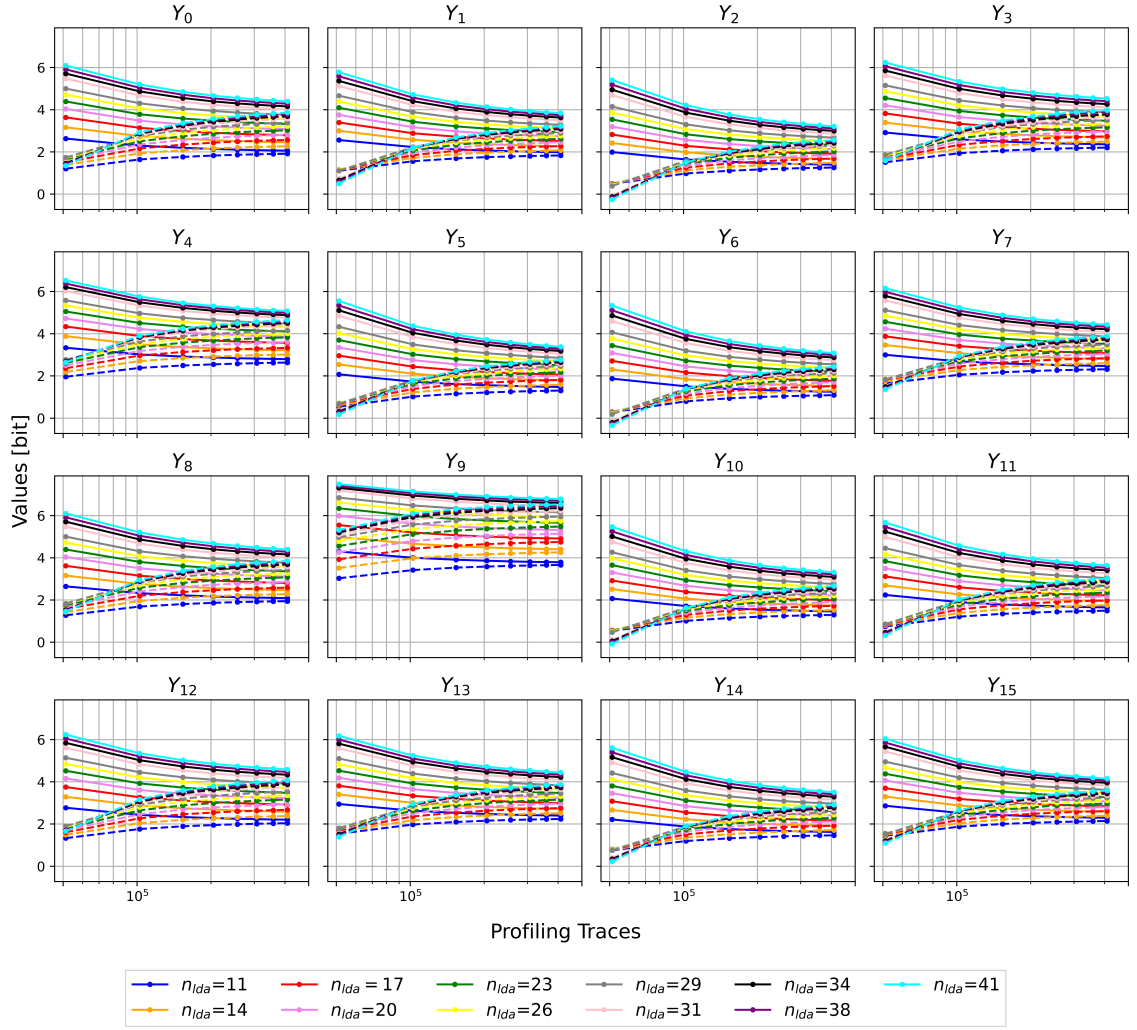


Figure 23: PI and TI convergence of the 16 LDA-based templates for different n_{lda} values. The PI, depicted by the dashed lines, is evaluated on 50k traces and the TI is drawn with plain lines.

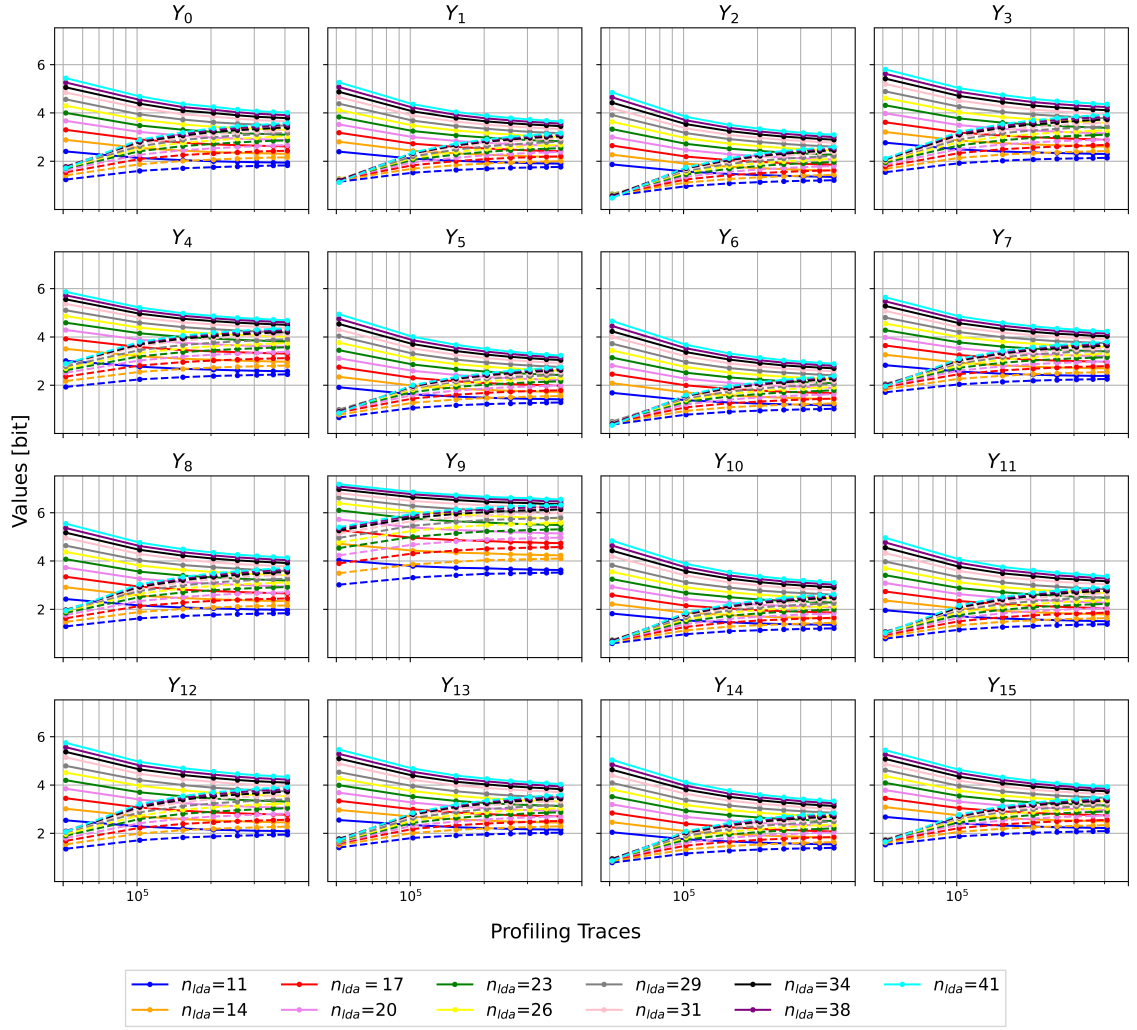


Figure 24: PI and TI convergence of the 16 LDA-based templates for different n_{lda} values with POI selection for Fixslice AES. The PI, depicted by the dashed lines, is evaluated on 50k traces and the TI is drawn with plain lines.

UNIVERSITÉ CATHOLIQUE DE LOUVAIN

École polytechnique de Louvain

Rue Archimède, 1 bte L6.11.01, 1348 Louvain-la-Neuve, Belgique | www.uclouvain.be/epl