Hacking the Systems from Within

André Nunes da Silva

***Abstract*—Information security is a fundamental aspect of our current technological society. *Side-Channel Attacks* allow compromising a device using information leaked through its physical properties, such as the power consumption or the duration of certain operation. On the other hand, modern processors include power consumption sensors which can be accessed by its users. The goal of this research is to analyse the possibility and threat of a side-channel attack using the processors' energy counters. This method aims for an attack without physical access to the device, in contrast with the usual usage of an oscilloscope. Challenges include the much lower sampling frequency and worst resolution, imposing new adaptations of the current attacking methods. For this article, an Intel processor was used to analyse and carry out a modulation attack against a simplified version of AES. Its performance was measured and results show that a detectable leakage exists which allows compromising the simplified version of the algorithm, but is not enough for the real one using the proposed method. Future evaluation of this leakage is proposed towards other methods and algorithms.**

***Index Terms*—cryptography, side-channel attacks, power analysis, energy-management software, running-average power limit.**

I. Introduction

The amount of available information and the connectivity among people distinguish our era from the past. More and more information is being stored in digital format, in an on-going process called digitalization. At the same time, the network connects us and increasingly our devices as well, with the so-called Internet of Things. This makes the cyberspace more and more relevant, valuable and real, and like in anything that is valuable and real, threats and risks surge. This way, Information Security rises as an extremely important area of today's society, being crucial for different sectors such as communications, economics, health systems, and even our own entertainment.

Information Security is constantly challenged by the discovery of new vulnerabilities and the creation of new attacks. In order to protect the information from unauthorised parties the data is many times encoded or, in other words, encrypted. Due to its importance and past study, there are widely-used algorithms that are mathematically very secure. Given the difficulty to break such algorithms, the current strongest attacks do not target the algorithm itself, but its physical implementation, that is device-dependent and consequently more difficult to protect. These are the so-called Side-Channel Attacks (SCAs). Examples of this kind of attacks use channels like electromagnetic emissions, power consumption or sound to discover the secret information.

Parallel to the increasing technology presence comes the concern of energy usage, both from an environmental point of view and from an economic perspective. One example of this is the strategic placement of cryptocurrency mining farms in colder parts of the globe in order to lower the energy expenses related with cooling the computers. Another, being both an example and a consequence, is the inclusion of power meters in modern processors, in order to monitor and limit their energy consumption. In Intel's processors, this feature is named Running Average Power Limit (RAPL) and is included in all chips since the Sandy Bridge architecture. Bringing this together with the side-channel attacks that exploit the power consumption, the possibility of new threats arise.

II. Background

A. Cryptography

Cryptography is the area of Information Security that deals with protecting sensitive information from unauthorised users. This information can be anything from resting data files, to messages, passwords, among others. It is used everyday in many forms of technology, like the internet, Bluetooth, mobile telephones, wireless systems, bank ATM's, among others; many times without the average user even noticing it.

The means to implement cryptography are the cryptographic algorithms. These algorithms encrypt data, referred to as plaintexts, by transforming them into incomprehensible ciphertexts, based on a secret value, referred to as the key. The inverse operation, called decryption, transforms the ciphertext back into the plaintext, that is the original data. Again, a key is needed to perform this operation. Here, cryptographic algorithms branch into two families: *Symmetric Cryptography*, where the same key is used for encryption and decryption, and *Asymmetric Cryptography*, where a key pair with two different keys is used instead.

Contrary to cryptography there is *cryptanalysis*, referring to attacks that try to recover the secret information manipulated by a cryptographic algorithm. The most trivial attack is the *Brute Force Attack*, where every possible key is tried one by one. This attack is unfeasible for most cases, since current algorithms usually use keys with a size of 128 or more bits that make such search an impossible task with even the most recent technology.

Different adversary models can be considered in cryptanalysis, based on the amount of information available to the attacker. For instance, knowing some plaintext/ciphertext pairs, the algorithm or the key size are all variables that are considered. It is generally assumed that the attacker has a perfect knowledge of the algorithm, and has access to the inputs and outputs of the operations. This consists of the *Black-Box Model*, and aids revealing if an algorithm is mathematically secure, independently of the physical implementation. Adding to that, the *Grey-Box Model* considers that the attacker also has access to the physical device where the cryptographic algorithm is implemented and to the information leaking through the respective side-channels, like power consumption, time, or radiation. This leaked information usually depends on the data being processed, and opens a door for exploitation.

Side-Channel Attacks (SCAs) are the family of attacks that exploit those side-channels instead of attacking the algorithm itself. These attacks can be much more powerful than traditional Black-Box attacks. In fact, most actual cryptographic algorithms have been broken with Side-Channel Attacks. The technique to exploit an algorithm based on the power consumption is called *Power Analysis*. It has been used across different implementations, such as Smart-Cards [1] , FPGAs [2], to even more recent smartphones [3]. These attacks are usually invasive, meaning that the attacker requires physical access to the chip in order to connect an oscilloscope and make power measurements.

Advanced Encryption Standard (AES)

The AES is the most commonly used symmetric algorithm for cryptography. It is a fixed-size block cipher, with a key of 128, 192 or 256 bits. It consists of a loop of rounds of a *substitution-permutation network*, where the operations are sequentially applied to the original plaintext, called state, viewed as a 4×4 matrix of bytes. The number of rounds depends on the key size, being 10, 12 and 14 accordingly for the 128, 192 and 256-bit versions, with a slightly different last round. For each round there is a *subkey* (or *roundkey*) derived from the main key though a *key scheduler*.

```
input : 128-bit Plaintext; 128/192/256-bit Key
output: 128-bit Ciphertext
KeyExpansion;
AddRoundKey;
for 1 to 9/11/13 do
    SubBytes;
    ShiftRows;
    MixColumns;
    AddRoundKey;
end
SubBytes;
ShiftRows;
AddRoundKey;
```

Fig. 1. AES Overview - Encryption

KeyExpansion One roundkey per round are derived from the main key through the AES Key Schedule, which is an algorithm that expands one key into several.

AddRoundKey Bitwise Exclusive-OR (XOR) between State and RoundKey: $State \oplus RoundKey$

SubBytes Substitution of each State byte $a_{i,j}$ based on a fixed *Substitution-Box* (S-Box), in the form $a_{i,j} = S(a_{i,j})$.

ShiftRows Cyclically shifts to the left the bytes of each row n times, being $n = 0, ..., 3$ the number of the row.

MixColumns Combines bytes of each column independently based on an invertible linear transformation in the finite field $GF\{2^8\}$ given by a fixed matrix.

B. Architecture

The Central Processing Unit (CPU) is the component of a computer responsible for performing instructions provided by a program in order to do any task. This way, usual programming languages are ultimately translated to a low-level programming language called *assembly*, which has a strong correspondence to what the computational architecture actually does at the architectural level. This assembly code is then *assembled* into executable machine code instructions by an *assembler*, specific for the processor in question. Those instructions are then executed, performing the desired tasks, such as arithmetic operations or data manipulation. To this logic and physical design, together with its organisation and implementation, it is called a microarchitecture.

One important characteristic of a CPU is its clock rate, which refers to the frequency at which the components of the processor operate in order to synchronise them. This directly reflects on the processing speed, as it determines the number of instructions performed per time unit. To further increase the performance, many computers' have a microprocessor chip with several processors inside them, called *cores*. Other components of a typical processor include the *integrated graphics unit*,the *system agent*, the shared last-level-cache, and the interconnect ring which connects all the components.

Processors are designed towards an instruction set, which represents the instructions they can perform. These instructions are represented by a mnemonic that is possibly combined with one or two operands, being then translated to a series of bytes called an *opcode*, that generally represents a single executable machine instruction. The operands correspond to the places where the values used in the operation are stored, called *registers*.

In order to perform the different types of instructions in the most efficient way, a series of components exist inside the core of a processor, separated into two main areas: the *Front End* and the *Execution Engine*.

The goal of the Front End is to feed the Execution Engine with a stream of operations it gets by decoding instructions coming from memory. This pipeline acts *in order*, meaning that the chronological order of the instructions is respected. After being decoded in a series of components, instructions are stored in the *Allocation Queue*, which acts as an interface to the *Execution Unit*. From there on, the flow of instructions operates in an *out of order* way. Here, they proceed through more components until they reach the *Scheduler*, which leads to the different *Execution Units*. Here, the instructions are directed according to its operation, and executed. Finally, they are retired in the *Reorder Buffer*, releasing the corresponding resources are restoring their chronological order.

Digital circuits operate based on two levels of discrete voltage levels, *0* and *1*, usually implemented using CMOS transistors to build *logic gates*. One advantage of CMOS is the low power dissipation, due to the complementary pull-up and pull-down network organisation, which is arranged such that when one is activated the other is disabled, resulting in a static consumption power near zero. The main source of power consumption is the dynamic part, that corresponds to the energy that is consumed during the transitions of state, as a result of brief moments when both networks conduct and the connection between the power source and the ground is established. This way, dynamic power consumption is dependent on the data being manipulated, including both the flow of instructions being processed, and the operands used in the execution units. Consequently, if the same algorithm is executed on different pieces of data, the power consumption is different due to the different values of the registers.

Due to the rising importance of saving energy and controlling its expenses, power consumption has become a critical metric in the design and usage of electronic systems. This led to the introduction of new components into CPUs that allow energy and performance measurement, in order to measure or limit the consumption as necessary. Despite their importance, their consequences on information security are not completely understood[4]. In Intel's CPUs, this feature is covered by Intel's Running Average Power Limit (RAPL), which consists of a set of measuring sensors and counters, as described in the Intel Software Developer's Manual [5]. It is included in their CPUs since the *SandyBridge* microarchitecture, with slight differences across the many subsequent ones.

The units for the power, time and energy readings vary according with the microarchitecture, and are found in the register *MSR_RAPL_POWER_UNIT*. For example, in order to convert an energy measure to Joules, it has to be multiplied by $1/2^{ESU}$, with ESU being the value represented in the bits 8 to 12 of that register. For the Skylake microarchitecture studied in this work, the ESU has the value 14. This way, the measures are transformed to Joules after being multiplied with $1/2^{14} = 61$ μJ. This corresponds to the minimum difference that two power/energy samples can have, and is refereed as the resolution. Notice that a smaller resolution allows for more accurate measures, as smaller differences in the input signal can be detected. This way, the raw integers stored in the register have to be multiplied by the resolution to be transformed in Joules.

Together with the resolution and the sampling rate, the Signal-to-Noise Ratio (SNR) is other import characteristic that indicates the behaviour of an Analog-to-Digital Converter (ADC), by comparing the level of a signal to the level of the background noise. It is given by

$$SNR_{dB} = 10\log_{10}\left(\frac{P_{S+N} - P_N}{P_N}\right) \tag{1}$$

where P_{S+N} is the power of the meaningful signal plus noise, and P_N is the power of the noise alone. The power of a digital signal can be calculated with

$$P_s = \frac{1}{N}\sum_{k=0}^{N-1}|s(k)|^2 \tag{2}$$

with N being the number of samples of the signal s.

III. State of the Art

Power Analysis is the branch of Side-Channel Attacks in which the channel used for exploitation is the power consumption of a device. Most Power Analysis attacks are based on acquiring *power traces*, which are a set of consumption measurements taken during the execution of one cryptographic operation. Different techniques exist to extract the sensitive information from the power traces, branching them into Differential Power Analysis or Template Attacks.

A. Differential Power Analysis

The first power analysis technique is Simple Power Analysis, which exploits the information available at one or few power traces by visually interpreting it. Figure 2 illustrates a power trace where a clear pattern is repeated 16 times, corresponding to the 16 rounds of the DES algorithm. That alone provides a clue on the used cipher.

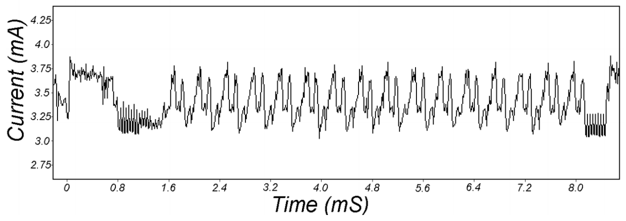

Fig. 2. Power trace during one encryption with the DES algorithm on a Smart-Card, retrieved from [6]

Kocher et al. [6] demonstrated how zooming in the plot highlights other details that leak sensitive data. However, a much more powerful attack is possible by statistically comparing differences across several traces, with the so-called DPA. This technique targets algorithms where the key K is split in small parts, allowing a divide and conquer approach, such as many block ciphers.

The success of this technique comes from modelling an intermediate bit that depends both on a small number b of key bits, K^*, and on the plaintext P, forming a *selection function* with the form $D(P, K^*)$. For AES, the output of the first S-Box is a good place to target, resulting in $D(P, K^*) = S(P \oplus K^*)$, with S being the S-Box substitution.

Then, to perform the attack, N power traces $T = T_1, ..., T_N$ are acquired with W samples each, corresponding to the execution of the algorithm with N different plaintexts $P = P_1, ..., P_N$, and the same unknown key K. Let $T[j]$ be the j^{th} sample of a given trace T. The power traces are separated into two sets, S_0 and S_1 according to the value of the most significant bit of the *selection function* output, with the respective plaintext P and a given key part K^* as input.

Finally, a *distinguisher* Δ_D is applied to them. The one used originally by Kocher et al. [6] consisted on calculating the trace corresponding to the difference of the averages of each set S_0 and S_1

$$\Delta_D = \bar{S_0} - \bar{S_1} \tag{3}$$

If the guessed key part K^* is wrong, then the output Δ_D results in a small noise near zero, because the result of the selection function D was different from the target for about half of the plaintexts. However, if K^* is correct, the output of the selection function is the same as the computed by the device for all the plaintexts, creating a correlation. As a result, Δ_D will approach the effect of the target bit on the power consumption as N increases. The plot will be flat with spikes in the zones where D is correlated to the processed values. The attacker can then identify the correct key-part K^* by trying the method with all the 2^b possibilities, and evaluating the computed Δ_D. Knowing one part of the key, other intermediate bits can be guessed to allow discovering the respective missing parts.

After this research work, many works were proposed to complete and diversify this technique. Several other distinguishers have been compared [7, 8]. The most widely used uses the *Pearson Correlation Coefficient*, p, as the *distinguisher*:

$$\rho_{x,y} = \frac{N\sum x_i y_i - \sum x_i \sum y_i}{\sqrt{N\sum x_i^2 - (\sum x_i^2}\sqrt{N\sum y_i^2 - (\sum y_i)^2}}. \tag{4}$$

For it, a *Power Model* is defined for estimating the power. Because the dynamic power consumption depends on the number of bit changes, the *Hamming Distance* can be used for this model: the Hamming Distance (HD) between two binary numbers is the number of different bits, position-wise. This way, the *Hypothetical Power Consumption* can be computed with $H_{n,k} = HD(D(P_n, K_k^*))$ for the n^{th} trace/plaintext pair and k^{th} possible key part, with $0 \leq k < 2^b$. Then, for each sample j and possible key part K^* the Pearson Correlation Coefficient $\rho_{j,k}$ is calculated to measure the correlation between the *Hypothetical Power Consumption* and the real measures. The output is a value between -1 and 1. Higher absolute value means that the two data sets compared have the best correlation, while a value of zero means that there is no correlation at all. The higher correlation between the *Hypothetical Power Consumption* and the real power traces will represent the correct key part. Similar to the DPA, after matching a part of the key, the same method can be used to discover the remaining parts.

B. Template Attacks

Template Attacks were introduced by Chari, Rao, and Rohatgi [9] in 2003 and are very strong in an information theoretic sense because all the information leaking is possibly used, requiring samples than DPA. In contrast, the attacker requires an exact copy of the targeted device in order to create a model of its leakage. A typical template attack is then composed of two phases:

1) **Profiling Phase (or Training Phase)**. The attacker uses his replica of the targeted device to model the leakage regarding different operations.
2) **Attack Phase**. The power traces acquired for the targeted device are processed and compared with the model constructed in the first phase to obtain the secret key.

Profiling Phase

The goal of this phase is to develop a Multivariate Gaussian Model for the power consumption associated with different operations as accurately as possible. A large number of power traces are gathered using the clone system corresponding to the power consumption of the device while performing the operations being modelled. Each operation is considered as performing an encryption with different bits depending on a key part K^* and plaintext P, based on a selection function that models part of the algorithm, similar to the one used in DPA: $D(P, K) = S(P \oplus K^*)$, with S being the S-Box substitution. This way, there are $2^b = 256$ different operations, where operation O_j is defined as encrypting a plaintext byte P and respective possible key part k such that $S(P \oplus k) = j$. The set of power traces acquired while performing O_j is represented by S_j.

To reduce the dimension of the distribution, and because most points of the power traces are not relevant since they are not directly affected by the key, a group of N Points of Interest (PoIs) are selected that represent the critical and key-dependent moments of the trace. An usual method to choose the PoIs consists of computing the square of the sum of the pairwise differences between the average signal of each template, Δ, and selecting only the N points at which large differences show up. In [10] a more advanced method is proposed, using the T-Test to successfully detect the PoIs with noisy signals. The T-Test is a statistical hypothesis test that distinguishes two data sets (i, j), by comparing the distance of the corresponding means (m_i, m_j) and their variability (σ_i^2, σ_j^2). Among various implementations, the Welch's T-Test is given by

$$t_{value} = \frac{m_i - m_j}{\sqrt{\frac{\sigma_i^2}{n_i} + \frac{\sigma_j^2}{n_j}}} \tag{5}$$

and is used when the different sample sizes and variances are a possibility. This way, the PoIs can be detected with

$$\Delta = \sum_{u=1,v=1}^{K} \left(\frac{m_u - m_v}{\sqrt{\frac{\sigma_u^2}{n_u} + \frac{\sigma_v^2}{n_v}}} \right)^2, \quad u \geq v \tag{6}$$

Having chosen N PoIs, $I_1, ..., I_N$, the power traces can be transformed by keeping only the positions related to those points. Finally, the average, $\mu_k \in \mathbb{R}^N$ the covariance matrix $\Sigma_k \in \mathbb{R}^{N \times N}$ are computed, defining the template of operation O_k.

In an attempt to reduce complexity and resources, Elaabid et al. [11] exploited the fact that the power dissipation in a CPU can be proportionally approximated by the Hamming weight of the computed data to reduce the number of templates required to 9, one for each possible Hamming Weight of the state after the S-Box. Despite this only allowing the discovery of the Hamming weight of the key part, its exact value can be recovered by repeating the attack with other plaintexts.

Attack Phase

Having computed the templates (μ_k, Σ_k), the probability distribution of the noise occurring from operation O_k is given by the N-dimensional normal distribution $p_k(.)$ where the probability of observing a noise vector z is

$$p_k(z) = \frac{1}{(2\pi)^N |\Sigma_k|} exp\left(-\frac{1}{2} z' \Sigma_k^{-1} z\right), \quad z \in \mathbb{R}^N \tag{7}$$

with $|\Sigma_k|$ representing the determinant of Σ_k, and Σ_k^{-1} its inverse.

This way, to classify a power trace $\hat{t}$ from the set $\hat{S}$ a maximum likelihood hypothesis test is performed. For each $k \in [0, K[$, the noise in $\hat{t}$ is extracted at the N PoIs, yielding a noise vector $n_k(\hat{t})$ with

$$n_k(\hat{t}) = t[I_1] - \mu_k[I_1], ..., t[I_N] - \mu_k[I_N] \in \mathbb{R}^N. \tag{8}$$

Then, the probability to observe such a noise vector can be computed with Equation (7). Consequently, the hypothesis k that maximises that probability is the best candidate for the observed trace $\hat{t}$. When more than one trace is available, the probability to be maxed is found with

$$P_k = \prod_{\hat{t} \in \hat{S}} p_k(n_k(\hat{t})). \tag{9}$$

After acquiring one byte of the key, the power traces can be mapped to new sets according to the next key and plaintext bytes, and the process is repeated.

C. Countermeasures

Countermeasures to side-channels can be separated into hardware-based and software-based approaches. The first relates to the root of the problem: the physical leakage of the device. They focus on securing that no information leaks, and therefore no more measures are necessary at a software-level. Examples include a *dual-rail* technique that spreads the complement of every signal, or the inclusion of a noise source component to shadow any possible correlation. On the other hand, the software-based approach consists of algorithmic countermeasures less dependent on the device. For example, *masking* consists of splitting the intermediate values of cryptographic computation into randomised shares to avoid dependencies between these values and the power consumption. Despite existing countermeasures, SCAs still present a significant threat. This is a result of improvements to attacks that partially or totally bypass current defences, and because countermeasures usually bring decreased performance in terms of computational time and memory, and/or a higher hardware space and cost.

D. Attacks based on energy measurement by software

Despite the short amount of literature regarding this topic, studies show that arising threats exist [4, 12]. Mantel, Shickel, Weber, and Weber [4] distinguished two secret keys of the Rivest, Shamir and Adelman using only RAPL measures through a modified template attack. Paiva, Navaridas, and Terada [12] used the DRAM power consumption to create covert channels, allowing two processes to communicate within a supposedly isolated environment. This demonstrates another level of relevant threats created by these features. Finally, a relevant work was done by O'Flynn and Dewar [13] by breaking AES using an on-board ADC on a SAML11 microcontroller without any external measuring equipment. While not using power measuring counters as usual of desktop computers, it consists of a type of Power Analysis attack without physical access to the device.

IV. Methodology

This work studies the possibility of exploiting the correlation between the power consumption and the processed data in a CPU joined with its energy measuring capabilities provided by the energy counters in order to discover secret information related to cryptographic operations. Compared to usual power analysis attacks where an oscilloscope is used to gather the so-called power traces, this method has the advantage of not requiring physical access to the targeted computer and the possibility of isolating the core consumption by choosing the appropriate power plane. On the other hand, the resolution is worse and the sampling rate is significantly lower than the clock frequency, bringing new challenges and difficulties.

Figure 3 illustrates an overview of the conceptual architecture of a proposed attack, inspired by current template attacks. This study can be separated into three steps: measuring the capabilities of the interface, post processing the measured data and methods to create templates, and finding methods to match the templates and attacks to exploit those methods in order to acquire sensitive information. The experiments are done using Intel's RAPL in a Skylake client microarquitecture with a clock of 2.5 GHz.

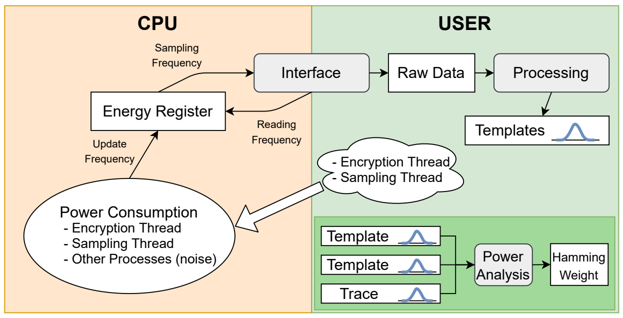

Fig. 3. Conceptual architecture of a possible attack

A. Using RAPL for sampling Power Consumption

Power consumption related to data computed at the cores is measured at the PP0 domain, and the respective values are stored in the 32-bit register *MSR_PP0_ENERGY_STATUS*. Since the relative differences of the values alone contain the leakage, the raw readings from the register are used without any conversion to a unit.

To read the MSR register, some interface must be used to provide those values to the user. For this article three interfaces were compared: Intel's PAPI, *Power Capping Framework (powercap)*, and a *Custom Framework* made by IST students. For each, an acquisition loop was performed by reading the register at the maximum possible rate in order to catch all the updates. The frequency at which the interface is able to read the register is called *Reading Frequency*, and it represents the computational resources and time used by the interface to read the register, resulting in undesired overheads. The rate at which updated measures are acquired is called *Sampling Frequency*, representing the samples of energy values per time unit.

When doing this type of acquisitions, special attention should be given to minimise resources used and noise introduced. For example, the measured values should be stored in an static array in order to avoid allocating memory in run time, which is a computationally expensive function. Also, the assembly code should be inspected to assure that the compiler is not removing important instructions being measured that it might consider useless. Looking at Table I it is seen how the Custom Framework achieves higher sampling and reading rates as a result of being lighter than the others. It is therefore used for the remaining readings.

Interface	Reading	Sampling
PAPI	1.00 kHz	575.35 kHz
Powercap	17.09 kHz	103.79 kHz
Custom	17.22 kHz	1033.12 kHz

TABLE I
SAMPLING AND READING FREQUENCIES ACROSS INTERFACES.

To measure how the power measurements relate to the activity happening in the cores the SNR is calculated, with the noise measurements taken while the CPU is as idle as possible and the signal values measured while the computational load is at its highest value. Being a computer that hosts an operative system, it is impossible to eliminate all side tasks, and some system calls and processes will remain, which will also take place during encryption operations. On the other hand, a high computational load is achieved with the *dd* command to shift a number of zero bytes in virtual memory by calling it with `/dev/zero` as source and `/dev/null` as destination. Figure 4 shows the energy measurements during idle time.

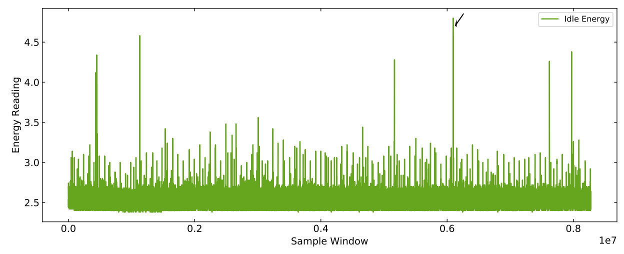

Fig. 4. Moving average of idle energy readings during 5 minutes, with an average window of 50 samples.

An aspect of the graph that stands out is the existence of spikes. In order to find out why those happen, a zoom is applied to the plot of the raw samples in the position of those spikes. This is done in Figure 5 to the spike of the upper plot pointed by the arrow.

It is possible to see that the spikes are a result of consecutive higher samples. It is difficult to track the exact reason for this happening. This experience was repeated, and the positions of

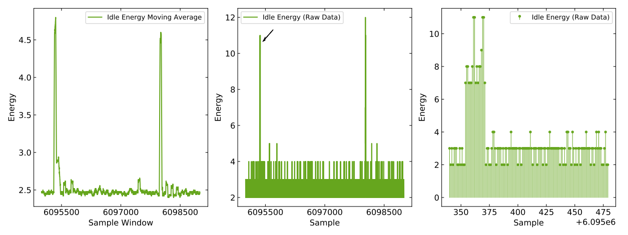

Fig. 5. Different zooms of the pointed energy spike. Direct zoom of the average window of 50 samples (left), raw samples of the same zone (middle), detail of the pointed spike of the middle plot (right).

the spikes shifted, which led to believe this is due to system calls and unavoidable for now. Three possible ways to solve this occurrence are simply filtering them out, saturating them by imposing a limit on the value they can take, or, a more complex option is to detect them and take it into account in following calculations. For now the first option is chosen, leaving the others to future research.

Another important source of invalid data is what is refereed as *machine warm up*, referring to the fact that the beginning of an acquisition tends to have its values decreased and gradually increasing to the established trend afterwards. After noticing that this happens due to an idle time between acquisitions, during which no measures were being acquired, more attention was given to this issue.

This effect is illustrated in Figure 6, where it is seen that the longer the thread is idle via a sleep function, the lowest the energy values become when it resumes computations. This leads to a longer time reaching the trend value, that is maintained from that point forward. It is important to underline that during the sleep time there are no energy recordings. This means that the expected plot would not show the low-energy moments at all. Also, the first sample after the idle time is removed in order to avoid any contamination of a very low sample to the moving average window. The explanation of this effect is not found in literature, and can be further studied if future research. This way, the used method to tackle this effect was to avoid idle times between acquisitions and eliminate the first samples of every acquisition during post-processing.

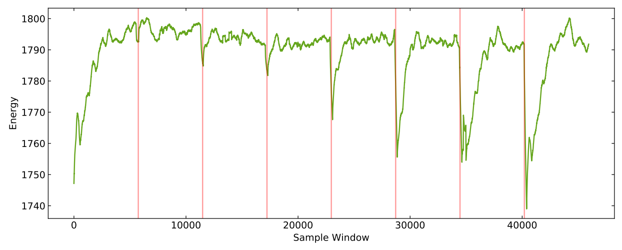

Fig. 6. Machine Warm Up - the encryption loops seems to spend less power during the beginning, proportionally to the idle time before. Red bars represent increasingly longer idle moments.

Computing the SNR value with Equation (II-B) resulted in a value of 21.98 dB, which is considerably low for power measuring, but still allows for a possible leakage.

B. Leakage Identification

An operation is leaking information through the energy registers if such information can be used to distinguish two similar algorithms applied on different pieces of data. In order to identify leakage and define further settings and parameters, a simplified AES is defined, allowing a consecutive increase in difficulty towards a real scenario and consequently a way of measuring the threat an attack provides. The conditions are as follows:

1) 1 round (instead of 10) to allow the exposure of the leakage of the first S-box without the further rounds hiding it.
2) 16 similar bytes in the keys and plaintexts - Amplifies the leakage of the S-box by 16.
3) 2 possible Hamming weights of the values loaded from the S-box (`0x00` and `0xFF`)

The values that load `0x00` and `0xFF` from the S-box are `0x52` and `0x7D`. Fixing the key k at a value, for example `0x50`, the plaintext p is chosen such that $p \oplus \texttt{0x50} = \texttt{0x00}$ and $p \oplus \texttt{0x50} = \texttt{0xFF}$. This results in $p_1 = \texttt{0x02}$ and $p_2 = \texttt{0x2D}$.

With the goal of obtaining the testing data to create templates, the power consumption profile of a loop of encryptions is acquired under the simplified scenario conditions. In order to produce the best results, the two plaintexts should be alternating after some amount of samples are obtained, so that outside effects such as the room temperature, usage of the cores, or any other noise affects in a more similar way the various acquisitions. The most promising results were found using values of around 5 minutes, as seen in figure 7.

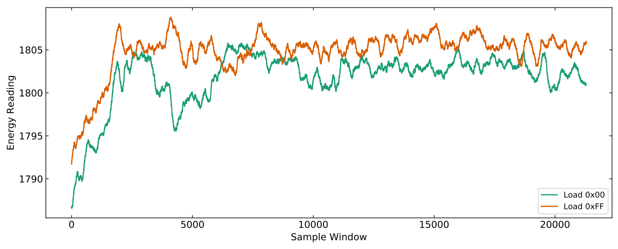

Fig. 7. Energy Moving Averages of loading `0x00`'s vs loading `0xFF`'s. Window of 55k (non-grouped) samples. It is visible that one operation consumes more energy than the other.

One aspect that arises is that the energy resolution is very high for the signals being acquired, resulting in the measured values being low and belonging to a small group of discrete values. Grouping by summing (or averaging) δ consecutive samples increases this resolution, at the cost of reducing the number of available samples. While this does not necessarily increase the chances of exploiting the measurements, it facilitates the filtering of outliers and allows for a better visualisation and demonstration of the leakage.

The histograms of Figure 8 show the corresponding distributions of the previous plots. The samples are grouped in groups of 256 and filtered by three standard deviations from the mean. The two distributions are easily distinguished, exposing a leakage of information.

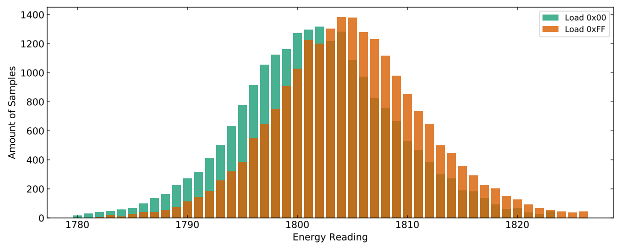

Fig. 8. Energy histograms of loading `0x00`s vs loading `0xFF`s.

After detecting this leakage, the experiments were repeated in another architecture in order to study possible RAPL differences across processors. This time, it was used an Intel i7-8700k with 6 cores and a base clock frequency of 3.7 GHz, both resulting in an overall increase of power consumption. This higher clock also led to an increased Reading Frequency of 1400.41 kHz, while the Sampling Rate was approximately the same, with a value of 17.336 kHz. These results match the expected ones. To study the leakage in this machine, the simplified scenario is used again. From the results illustrated in Figure 9, different observations are taken. Besides the widest span in the x-axis of the histograms, there is a significant deviation from a normal curve happening in the both cases. While being difficult to track the exact explanation of this, it is most likely that it happens due to other processes of the system and not from the data manipulation itself. Secondly is the fact that unlike the previous results for the other machine, the histograms mode value overlap each other. These results were repeated and this was observed every time. The proof of leakage does not seem to happen so firmly in this case. One possible reason is the increase of the number of cores, making the parallel computations of other processes overshadow the leakage resulting from the data processed in one of the cores.

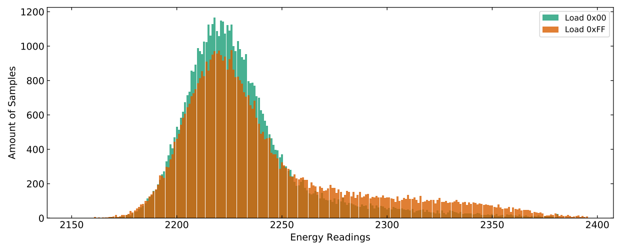

Fig. 9. Energy histograms of loading values with Hamming weights 0, 4 and 8, in a different machine. The distributions are not visually distinguished in this case..

This reflects how the targeted processor is an important feature of a possible attack, as they provide different levels of security against it. The initial machine is used for the remainder of this work.

C. Templates Matching

In order to match a trace to one of the possible templates, it is necessary a function that takes as input a set of distributions (templates and trace), and returns values that reflect their similarity, with the template that represents better similarity with the trace being the matched one. While this function's goal is to guess to which template the trace belongs, it can also be used to characterise and evaluate different parameters of the acquisition.

One possible approach to output the similarity across distributions is to use the T-Test formulated in Equation (5), which has the advantage of permitting different sizes of the distributions. This operation returns a value referred to as the t_{value}, that represents the similarity between those data-sets, being zero when they are completely similar, and increasing in absolute value as the similarity drops. This value is usually compared against a critical value table to determine if the data-sets differ, or if that difference is due to the effect of chance. For the sake of simplicity, it is common practise to use the value of 4.5 to draw the line that distinguishes different sets from similar sets [14]. Since it is desired to always select one template to match the trace, the proposed matching function selects the template that together with the sample trace results in the lesser t_{value}.

One last requirement to evaluate the different parameters and conditions is a way to numerically measure the performance of that acquisition, based on the acquired samples. This performance reflects how well can the different templates be distinguished and each trace matched with the respective template, using the already mentioned T-Test method. The method used generally in cryptanalysis is to do the whole attack with those acquisitions and check the position of the correct key in a ranking of keys ordered by the probability of being correct, returned by the algorithm. This allows an intelligent and feasible brute force search even if the correct key is not the first in the ranking.

However, in this case in particular, the probability of finding the correct key strongly depends on the number of plaintexts used for encrypting, and therefore it using that metric would require making acquisitions with a large number of plaintexts for each key guess. What can be used instead, being a common practise in classification problems, is the *confusion matrix*. The confusion matrix is a square matrix that describes the performance of a classification model by displaying the probability of choosing each predicted class (template) according to each actual class (trace). This way, the position i, j has the probability of matching the trace j to the template i. Using this for performance control has the advantage of having parameters that reflect aspects of the classification model in a single value, such as the F1 score and the Matthews Correlation Coefficient.

To calculate the confusion matrix, suppose an acquisition test with two resulting distributions, one with the Hamming weight of the S-box output as 0, and the other as 8. For each distribution, the samples related to 1 minute are considered the attacking trace, and the remaining are considered the template. Performing the matching function with one of the two attacking traces results in two t-values, that represent the possibility of that trace belonging to either template 0 or to template 8. Doing it for the other trace creates another row of two more values, creating the matrix at Table II (left). Using different groups of samples for the attacking trace and for the template allows this experience to be repeated for statistical purposes. Assuming that the template with the smaller t-value of each trace is the matched one, the confusion matrix at Table II (right) is created, providing the mentioned statistical values for the probabilities of matching each template to each trace.

	Template 0	Template 8
Trace 0	**2.135**	7.955
Trace 8	17.113	**1.468**

	Template 0	Template 8
Trace 0	**0.88**	0.12
Trace 8	0.00	**1.00**

TABLE II
EXAMPLES OF T-TEST RESULTS (TOP) AND CONFUSION MATRIX (DOWN) FOR TEMPLATES 0 AND 8.

D. Attacks and Performance

The algorithm depicted in Figure 10 illustrates an attack highlighting how knowing the Hamming weight of the S-box output through the previous matching templates function can be used to recover the key during the attacking phase.

```
Input: HW - Possible Hamming Weights P -
       Number of Plaintexts            K -
       Possible Keys
Profiling Phase :
1  for $h \in HW$ do
2      Tp[h] = createTemplate();
3  end
Capture Traces :
4  for $i \in 0, ..., P$ do
5      Tr[i], Pt[i] = Encrypt(pt = rand());
6  end
Attacking Phase:
7  for $i \in 0, ..., P$ do
8      h = MatchTemplates(Tp, Tr[i]);
9      for $k \in K$ do
10         if $HW(SBox(k \oplus Pt[i])) == h$ then
11             score[k]++;
12         end
13     end
14 end
```

Fig. 10. Attack against AES

The first part of the attack consists in creating templates in the cloned device owned by the attacker. On the second part, the attacking traces are captured on the machine under attack. Finally there is the attacking phase, when the attacker tries to recover the secret key part from the h generated templates and the P pairs of captured traces and corresponding plaintexts. For that, the function *MatchTemplates* takes the templates and each attacking trace to guess h, that is the Hamming weight of the output of the AES S-box under attack. Knowing that value, one can iterate all the possible keys and calculate which of them would, *XORed* with the plaintext associated to that trace, load a value from the S-box with that same Hamming weight h. The score of such keys is incremented, and the loop continues for the next trace/plaintext pair.

At the end of that cycle, each key will have a score. Since the used key was the same for all the iterations, assuming that the matching function classifies correctly, there will be one possible key with the maximum score similar to the number of iterations, which is the correct key. The usage of a score allows for an intelligent brute force search in case the accuracy of the templates matching function is not 100%.

The number of available pairs of plaintexts/traces can compensate for a poorer classification accuracy. In order to effectively study this effect, a simulation is performed with a function that mimics the template matching according to an error parameter, P_e, indicating the percentage of choosing a wrong template given a trace and the group of possible templates. Due to the constraints of the simplified scenario, the generated plaintexts have to result, after being *XORed* with the key, in a value that loads from the S-box a byte with a valid Hamming weight. This way, assuming the Hamming weights 0, 4, and 8, there are $1 + 70 + 1 = 72$ available plaintexts. Figure 11 shows the evolution of the key position according to the error rate and number of plaintexts, for this scenario.

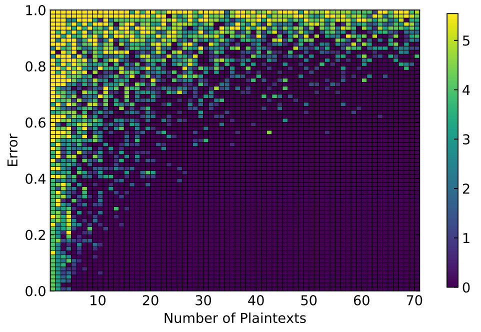

Fig. 11. Logarithm of key position per error rate and number of plaintexts for templates with Hamming weight 0, 4 and 8.

One can see that increasing the number of plaintexts leads to higher chances of finding the correct key, as the corresponding position is lower. Here, the position means the number of keys with a score higher or equal to the correct key. Despite the fact that lowering the error rate contributes to finding the correct key, it is relevant that such a key can be found even with error rates higher than the template matching function, given a feasible number of plaintexts.

Upon successfully attacking the simplified scenario, the difficulty is increased in three orthogonal ways in order to evaluate how much of a real threat this attack constitutes.

Adding Hamming Weights

As more Hamming weights and the corresponding templates are included, it gets more difficult to match them. This happens firstly because there are more classifying options, and secondly because they are more similar to each other. With 5 different Hamming Weights the error could still be contoured by a realistic amount of plaintexts. However, with all the nine Hamming Weights the error rate is particularly high and most of the traces are wrongly classified. While these difficulties compromise the attack against a system using the real AES, ideas exist to contour them. An example includes exploiting the fact that the miss-classifications tend to hit a template with an adjacent Hamming weight.

Adding Different Bytes

In the simplified scenario, the keys and the plaintexts have all the 16 bytes alike. This means that the leakage in a S-box is being multiplied 16 times. Such does not happen in a real case. Besides that, it is necessary to isolate small parts of the key to make an attack feasible as the solution proposed is to randomise the outputs of some S-boxes, ideally all but one, so that the non-randomised one can be isolated. This is usual practise in cryptography as the random values tend to null each other with the increment of samples, and randomising certain S-boxes outputs is easily achieved by randomising the correspondent part of the plaintexts. Actually, it is fundamental to distinguish every singular byte, as attacking the bytes in pairs would result in a search space of $8 \times 2^{16} = 524238$ which requires an unfeasible amount of plaintexts. While using 8 random bytes the error rate of the templates matching was still sufficient, with the necessary 15 random bytes the classification was similar to a random one.

Increasing Number of Rounds

By increasing the number of rounds, the leakage at the S-boxes of the first round becomes overshadowed by the other rounds. Unlike usual attacks, the time resolution of the acquisitions does not permit isolating certain parts of the trace. However, since the rows and columns are mixed in the rounds of AES, the previously-mentioned randomised bytes will create whole random states after the first round, which will null each other, being the only non-random component the S-box operation of the first round, which leaks the key information. However, the increase of rounds resulted in too many miss-classifications.

V. Conclusions

Information security is an extremely important aspect of today's modern society. The evolution of technology results in safer solutions, but also brings new vulnerabilities that can be exploited. Extreme precaution must be taken, specially for widely-used and trusted devices such as computer's processors, namely Intel's. Side-channel attacks in particular require an extra attention given their difficulty to detect and prevent, in conjunction with their dangerous potential.

This research builds a foundation to the study of software-based energy measuring side-channel attacks against AES, in an attempt to completely recover the secret key from the power consumption profiles. The characterisation of RAPL as an ADC across different interfaces showed that PAPI is not suitable for fast acquisition, since it limits the information provided by the RAPL registers. Instead, a custom-made framework is recommended.

A simplified scenario was designed in order to perform experimental work aiming the Advanced Encryption Standard. Analysing the power profiles of encryptions under such conditions concluded that there is a power leakage that can be detected with RAPL and possibly exploited. This is related to the processed data and can compromise its security. A method was created to compare the power profiles acquired during the encryption operations with previously acquired ones, recalling the template attacks. This allowed discovering the Hamming weight of the value loaded from the first S-box during the encryptions. Then, an algorithm was designed to use it together with the original plaintexts to successfully obtain AES secret key parts under the simplified scenario conditions.

A full key recovery was possible in simplified scenarios but seemed very difficult for now to be done with the proposed approach in a realistic scenario. These difficulties come from the trouble of detecting a smaller leakage, as a result of randomising a big portion of the plaintext. This is a crucial step to recover the key, as it is how the bytes are isolated in order to attack only a certain key part, and how the energy consumption from the consequent rounds is prevented from hiding the first round's leakage. While the theory suggests that it can be done, better acquisitions would be required to do so. Secondly, there is the classification performance, that although it proves capable of distinguishing some templates, it does not allow a correct matching of all the 9 templates with a limited or feasible number of power traces and plaintext pairs. Future upgrades to software-based energy measurement tools must have information security taken into account. For example, refinements in the resolution or sampling rate to this ADCs can be enough to improve the quality of the acquisitions in order to make a full key recovery possible.

VI. Future Work

Multiple times it happened that a branch of ideas happened and it was required to choose a path to stick to in order to proceed with the research. Different algorithms can be analysed for this end, as there many till used today simpler than AES. The same applies for the used architecture. The usage of multiple threads to allow encrypting and measuring at the same time was left unexplored, as well as using a periodic operation to measure the background noise and using it to adjust the captured values and its resultant distribution. In terms of template matching, a way to contour the difficulties related to the increase of the number of Hamming weights is proposed, as the miss-classifications have a tendency to hit templates adjacent to the correct one. Finally, being a classification problem, it would be a great contribution to see the performance of artificial intelligence and machine learning algorithms for such a task.

References

[1] S. Mangard, E. Oswald, and T. Popp, *Power Analysis Attacks: Revealing the Secrets of Smart Cards*. Springer, 01 2007.

[2] S. B. Örs, E. Oswald, and B. Preneel, "Power-analysis attacks on an fpga - first experimental results," in *Cryptographic Hardware and Embedded Systems - CHES 2003*, C. D. Walter, Ç. K. Koç, and C. Paar, Eds. Springer Berlin Heidelberg, 2003, pp. 35–50.

[3] L. Yan, Y. Guo, X. Chen, and H. Mei, "A study on power side channels on mobile devices," 12 2015, pp. 30–38.

[4] H. Mantel, J. Shickel, A. Weber, and F. Weber, "Vulnerabilities introduced by features for software-based energy measurement," Department of Computer Science, Technische Universitatät Darmstadt, Germany, Tech. Rep., 2017.

[5] *Intel Architecture Software Developer's Manual, Volume 3: System Programming Guide*, Intel, 2009.

[6] P. Kocher, J. Jaffe, and B. Jun, "Differential power analysis," *Advances in Cryptology - CRYPTO99*, p. 388–397, 1999.

[7] E. Oswald, L. Mather, and C. Whitnall, "Choosing distinguishers for differential power analysis attacks."

[8] H. Maghrebi, O. Rioul, S. Guilley, and J.-L. Danger, "Comparison between side-channel analysis distinguishers," in *Information and Communications Security*, T. W. Chim and T. H. Yuen, Eds. Berlin, Heidelberg: Springer Berlin Heidelberg, 2012, pp. 331–340.

[9] S. Chari, J. R. Rao, and P. Rohatgi, "Template attacks," in *Cryptographic Hardware and Embedded Systems - CHES 2002*, B. S. Kaliski, ç. K. Koç, and C. Paar, Eds. Berlin, Heidelberg: Springer Berlin Heidelberg, 2003, pp. 13–28.

[10] B. Gierlichs, K. Lemke-Rust, and C. Paar, "Templates vs. stochastic methods," in *Cryptographic Hardware and Embedded Systems - CHES 2006*, L. Goubin and M. Matsui, Eds. Berlin, Heidelberg: Springer Berlin Heidelberg, 2006, pp. 15–29.

[11] M. Elaabid, S. Guilley, and P. Hoogvorst, "Template attacks with a power model." *IACR Cryptology ePrint Archive*, vol. 2007, p. 443, 01 2007.

[12] T. B. Paiva, J. Navaridas, and R. Terada, "Robust covert channels based on dram power cnsumption," in *Information Security*, Z. Lin, C. Papamanthou, and M. Polychronakis, Eds. Springer International Publishing, 2019, pp. 319–338.

[13] C. O'Flynn and A. Dewar, "On-device power analysis across hardware security domains: Stop hitting yourself," *IACR Transactions on Cryptographic Hardware and Embedded Systems*, vol. 2019, pp. 126–153, Aug. 2019.

[14] T. Schneider and A. Moradi, "Leakage assessment methodology," in *Cryptographic Hardware and Embedded Systems – CHES 2015*, T. Güneysu and H. Handschuh, Eds. Berlin, Heidelberg: Springer Berlin Heidelberg, 2015, pp. 495–513.