UDC 004.056

Power Analysis Template Attacks on AES-128 Hardware Implementations and Protection Against Them

Andrii Dehtyariov[1] and Mykola Graivoronskyi[1]

[1] *Igor Sikorsky Kyiv Polytechnic Institute, Beresteiskyi avenue 37, Kyiv, 03056, Ukraine*

Abstract

The purpose of this work is to research AES-128 power analysis template attack and propose a practical way to mitigate such kind of side-channel attacks. The research includes a review of power analysis side-channel attacks, an experiment with the collection of Atmega328PU chip power samples using Hantek 6022BE oscilloscope, processing collected data and modeling – building statistical template of the device and analyzing parameters of the side-channel attack.

The work is focused on preparation and carrying out the experiment. The experimental bench layout and procedures of collecting and processing the data are considered in details.

The result of this work is the confirmation of the effectiveness of power analysis template attacks on AES-128 for Arduino Uno hardware, and a mechanism for mitigating such kind of attacks on the particular hardware and software implementation. Research materials described in the current work could be used for developing another side-channel template attack mitigation mechanisms for other cryptographic implementations.

Keywords: Cryptanalysis, side-channel attacks, power analysis, SPA, DPA, template attacks, traces, samples, AES-128

Introduction

In today's conditions it is extremely critical to preserve such properties of information as confidentiality and integrity. Preservation of these properties becomes possible through extensive use of the achievements of modern cryptography. It is difficult to imagine any system for processing and storing information that does not use international or national cryptographic standards.

When accepting the proposed encryption algorithm as a standard, this algorithm is tested for compliance with a number of requirements, including resistance to various attacks: correlation, algebraic, linear and differential cryptanalysis, brute force attacks, etc. On the other hand, the encryption algorithm must meet purely practical criteria: relative simplicity of calculations (especially for streaming ciphers), unpretentiousness to such computing resources as RAM, and other criteria. Algorithms such as AES meet all these criteria and have been used in information and computing systems around the world for about two decades.

However, the practical implementation of such cryptographic algorithms, being a combination of a particular physical device and program code, is usually vulnerable to side-channel attacks. By sequentially measuring physical environmental parameters, such as the current in the processor power contacts or the energy of the electromagnetic field generated by the cryptographic computing device, the attacking party can infer from observations of changes in these parameters, and, in the worst case scenario, recover encryption keys, thereby compromising these devices without even interfering with their operation.

Template attacks belong to profiled side-channel attacks, and are considered to be the most powerful. Such attacks include the stage of building a statistical template of the cryptographic device under study using a power consumption information leakage side channel, provided the cryptanalyst has the exact copy of the device to be attacked. With this template, the cryptanalyst can recover a cryptographic encryption key or certain information about the internal state of the cryptographic implementation using the mini-

mal amount of data at the stage of collecting power consumption samples.

There are two ways to implement a cryptographic algorithm: 1) software implementation on general-purpose computing processors, such as x86, ARM, and AVR family processors, and 2) hardware implementation, i.e. the development of hardware design specifically for the efficient implementation of calculations related to the selected encryption algorithm. Each of these approaches has its advantages and disadvantages, but the most common in practice is the software implementation of the cryptographic algorithm on general-purpose processors.

In the case of hardware implementations, to protect against side-channel attacks the vulnerable device must be replaced by another device that includes a protection mechanism in its design, while in the case of software implementations for general-purpose processor, it is possible to simply replace the vulnerable software with a less vulnerable one.

The purpose of the study is to investigate a template attack on the implementation of the AES-128 encryption algorithm in the AVR Assembler programming language for the Atmega 128PU microcontroller from Atmel based on the Arduino Uno hardware platform [1], and to propose a mechanism for complicating this attack in practice.

Using the results obtained in this study, we can get a software implementation that complicates the task of building a statistical template of power consumption for Atmega 328PU. It is possible to develop similar mechanisms of protection against a similar template attacks for other hardware platforms.

1. Power consumption side-channel attacks

1.1. Known power consumption side-channel attacks review

Information leakage side channels have been and remain the current vector of attacks on software and hardware systems, as they have a number of advantages, including: no need to directly interfere in the work of systems and the inability to track attacks of this kind. The physical nature of such attacks makes all software and hardware systems vulnerable to one degree or another and various side channels can be combined to filter out noise and more accurately recover information processed by the device. There are a number of different information leakage side channels: optical, timing, power consumption, and even acoustic.

In this paper we will investigate power consumption information leakage side-channel attacks (power analysis attacks). One example of the practical application of power analysis attacks was a study of the “Trezor” crypto-wallet, carried out by researcher Jochen Hoenickie [2]. Using the relatively inexpensive Hantek6022BE oscilloscope, the researcher was able to recover the private key of his crypto-wallet via a side channel. Trezor later acknowledged the problem and accepted the patches developed by the researcher for their software.

Power analysis attacks can be divided into 2 major groups: attacks with building a cryptographic device template and attacks without building a template.

Attacks without building a template use power consumption data collected directly during the attack. Such attacks include Simple Power Analysis (SPA), and Differential Power Analysis (DPA). DPA attacks use known plaintext and, based on statistical analysis of the power consumption data measured during many known plaintext encryption iterations, choose the correct hypothesis about the value of the encryption key used during these iterations. SPA attacks do not require many iterations of encryption and are based on the idea that the encryption key can be inferred from the form of the graph of the dependence of the amount of power consumption on time. By selecting various plaintexts and analyzing the obtained dependency graphs, the cryptanalyst can hypothesize the value of part of the encryption key, or, in some cases, the entire encryption key as a whole. These types of attacks and their varieties are widely described in the book [3].

The complexity of these attacks is that in the case of SPA it is not always possible to suggest at least one hypothesis, because the difference between the graphs of dependencies is not obvious, while in the case of DPA building hypotheses requires too many iterations of encryption, which is not always possible to perform in practice.

Template attacks are based on the assumption that the attacking party has free access to an exact copy of the attacked device. With such a copy, it is possible to build a specific statistical template of the power consumption of this device based on many iterations of encryption using many different combinations of encryption keys and

plaintext. Having such a template, it is enough for the attacker to obtain data on the power consumption of the device for a relatively small number of encryption iterations based on various randomly selected plaintexts. The better and more accurately constructed the statistical template, the less data needed during the attack. In the best case (from the attacker point of view), one encryption iteration is enough to recover the encryption key by power analysis.

The difficulty of this type of attack is mainly the need to have an exact copy of the cryptographic device and the need to build a template of this device, which may require a relatively large amount of time and tangible computing resources, especially if the profiling device has protection against side-channel attacks.

1.2. Model of power consumption information leakage side channel

Based on observations of the power consumption of different computing devices, an obvious relationship was found between the instantaneous power consumption values and the data currently being processed. Due to the physical nature of modern processors that use registers and memory cells to preserve the binary representation of data before direct processing, as well as the most logical operations performed on data bits, the researchers can build a power consumption model based on the Hamming weight of bytes of this data. Accordingly, the correlation coefficient between the Hamming weight of byte and the values of power consumption during processing of this byte is given by the equation:

$$\rho(W,P) = \frac{Cov(W,P)}{\sqrt{Var(W)} * \sqrt{Var(P)}}, \quad (1)$$

where W is the value of power consumption measured by the oscilloscope, and P is the value of the Hamming weight for the calculated sequence. Based on this equation, we can build a model of a power consumption information leakage side channel.

In the case of a template attack, which we will consider in this paper, while building the template, the researchers put the known instantaneous value of W in accordance with the known Hamming weight P of the key.

During the attack phase, the Hamming weight of the byte of the key is unknown, but knowing the instantaneous value W of the consumed energy, and with the help of a well-constructed template, it is possible to hypothesize the Hamming weight P of the key.

Obviously, when there is a correlation of the values of instantaneous power consumption W with the Hamming weight P, there is also a correlation between the value of W and the value of byte A, which is obtained by the XOR operation of the corresponding bytes of the key and plaintext. Thus, there are 2 models of power consumption on the basis of which one can build a cryptographic device template: the Hamming weight, and the value of the byte of the key.

1.3. Cryptographic device template attacks against AES-128

First cryptographic device template attacks were proposed in work [4]. They were developed for the RC4 and DES encryption algorithms. But the principles underlying this family of attacks make it possible to organize side-channel attacks of this type against any cryptographic implementation. Template attacks are the strongest ones from an information-theoretical point of view [5].

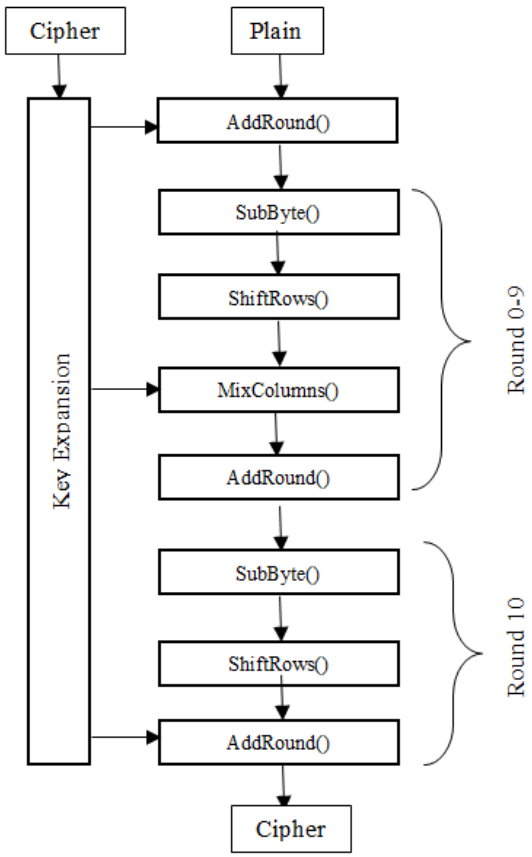

Figure 1: AES basic structure

Template attacks can also be applied to AES-128 implementations. Looking at the structure of AES-128 (Figure **1**), we can see that there are 2 potentially vulnerable operations that can be attacked: SubByte() at the first round and SubByte() at the last round.

In this work, we propose to consider the attack on SubByte() at the zero round, because it corresponds to the encryption key according to the AES-128 key schedule. Attacks on the SubByte() operation at the last round need to additionally reproduce the reciprocal transformations at rounds nine through zero.

$$\mathrm{Sbox}[a_i \oplus k_i] \quad (2)$$

The statistical template attack is possible due to the correlation between the Hamming weight of the bytes a_i and k_i.

The attack proposed in this work is carried out on each byte of the key separately using the same data. Accordingly, a separate statistical template is built for each of the 16 bytes of the AES-128 encryption key. Such a model of template attack is based on the assumption that in the encryption process, the values of the bytes of the key affect the final power consumption separately, independently of each other. In reality, this is not entirely true, but the dependence of power consumption on different combinations of bytes is so insignificant in practice that it can be neglected.

1.4. Types of AES-128 implementation template attacks

There are a large number of approaches to build and use a template against the AES-128 implementations. Here is an essential classification of the attacks.

According to the point of attack on the algorithm, attacks are divided into:

- AES-128 Zero Round Attack [6];
- AES-128 Ninth Round Attack [6];
- Attack on the AES-128 key schedule [7].

By template building and attack model:

- Hamming weight model (That is, for each byte there are 9 hypotheses: 0-8);
- Key byte value model (That is, for each byte there are 256 hypotheses: 0-255).

According to the method of building the template:

- Finding the average value of power consumption for each key byte hypothesis [8];
- Making assumptions about the distribution of power consumption depending on the byte of the encryption key and finding the statistical parameters of the distribution [9];
- Training of various machine learning models [10].

According to the preprocessing method:

- No preprocessing;
- Finding the average value of the power consumption measurement for each selected key and plain text [11];
- With decomposition into a spectrum using a fast Fourier transform and cutting off high-frequency noises [12].

According to the method of interpretation of the obtained results:

- Direct interpretation of the received most likely hypotheses as correct;
- Brute-forcing of the received possible hypotheses to find the key (to a greater extent applies to the Hamming weight model);
- Template-algebraic attack (TASCA) with the selection of hypotheses obtained as a result of the template attack using the selected Boolean function [13].

It is possible to combine some selected approaches and thus get even more varieties of the template attacks. Each of the above families of attacks has its own advantages and disadvantages and is used based on the amount and quality of data that can be collected at the statistical profiling phase and the attack phase.

In this paper, we will consider an attack with building a statistical template based on the parameters of the Gaussian distribution with a byte model and without preprocessing the input data for the attack. This type of template-building attack is simple in structure, but quite powerful when applied to raw data collected using a cryptographic device with no protection mechanisms against side-channel power analysis attacks.

1.5. Standard view of power consumption data for further analysis

Usually, the power consumption of cryptographic chips is measured by an oscilloscope and, depending on the operating frequency of the cryptographic device itself and the oscilloscope; a vector of the appropriate length is recorded for each encryption procedure, containing the value of instantaneous power consumption in watts.

$$T = \begin{bmatrix} x_1, \\ x_2, \\ x_3, \\ \dots \\ x_n \end{bmatrix} \quad (3)$$

where x_i — samples, values of instantaneous power consumption. The vector T is usually called the power consumption trace. An example of such a trace obtained by encrypting 16 bytes of plain text on the Atmega 328PU chip is shown in Figure 2.

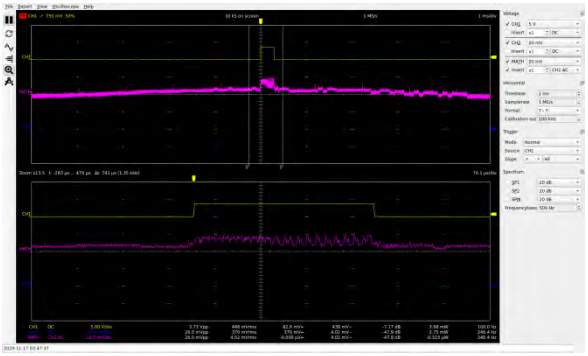

Figure 2: An example of a power consumption trace collected by a Hantek6022BE oscilloscope during encryption on an Atmega328PU

In the figure, the interface of the open implementation of the client program for Hantek oscilloscopes — OpenHantek [14] is shown. The upper plot demonstrates the power consumption data measured by an oscilloscope during the entire connection time. Below is the same graph, but on a larger scale. The main graph is the dependence of the current at the power supply pin of the Atmega328PU microcontroller over time. The thin line above it is the dependence of the current in the signal LED at the 13th digital pin of the Atmega328PU. A signal LED was used to indicate the start and end of encryption.

In this graph, we can see the typical power consumption peaks corresponding to the AES-128 key schedule (left) and the ten larger power consumption peaks corresponding to the ten rounds of AES-128 encryption.

1.6. Building the AES-128 implementation template based on multidimensional parameters of Gaussian distribution

In this work, we build the AES-128 implementation template on the basis of the obtained traces of power consumption through the determination of the parameters of the normal distribution: the mean μ and the variance Σ. Accordingly, we will consider such parameters for each of the 16 bytes of the round key with the assumption that the selection of each of the bytes of the round key are independent events [15].

When constructing a statistical distribution, we operate with sample vectors — traces. Thus, the obtained parameters μ and Σ are multidimensional and are found according to the equations 4, 5:

$$\mu = \begin{bmatrix} \overline{x}_1, \\ \overline{x}_2, \\ \dots \\ \overline{x}_n \end{bmatrix} \quad (4)$$

$$\Sigma(y,z) = \frac{1}{n}\sum_{i=1}^{n}(y_i - \overline{y})(z_i - \overline{z}) \quad (5)$$

where $\overline{y}$ and $\overline{z}$ are every 2 possible vectors of mean values.

Based on equations 4 and 5, the covariance matrix has the form:

$$\Sigma = \begin{pmatrix} \sigma_{\overline{x_0}} & \sigma_{\overline{x_0},\overline{x_1}} & \dots & \sigma_{\overline{x_0},\overline{x_n}} \\ \sigma_{\overline{x_1},\overline{x_0}} & \sigma_{\overline{x_1}} & \dots & \sigma_{\overline{x_1},\overline{x_n}} \\ \vdots & \vdots & \dots & \vdots \\ \sigma_{\overline{x_n},\overline{x_0}} & \sigma_{\overline{x_n},\overline{x_1}} & \dots & \sigma_{\overline{x_n}} \end{pmatrix} \quad (6)$$

This square covariance matrix has dimension $n \times n$, where n is the number of power consumption samples in one trace. In this matrix, σ is the value of the covariance between the values of the mean samples x_i and x_j.

However, handling matrices of such a large dimension can be a very resource-intensive task, since each additional dimension increases the number of calculations, which does not always lead to more accurate results in practice. To avoid this, there are many methods of data dimensionality reduction based on byte correlations (SOSD), the difference of the average values of the traces, *T*-tests, χ-square tests and others, which were described in work [5]. All these techniques aim to reduce the dimensionality of the profile, leaving only those samples that contain the most information about the signal and on the basis of which it is easier to build a qualitative statistical picture. Samples that carry the most signal can be seen in Figure **3**.

On this plot, they are represented by the peak values of the function of the correlation *vs.* the sample number.

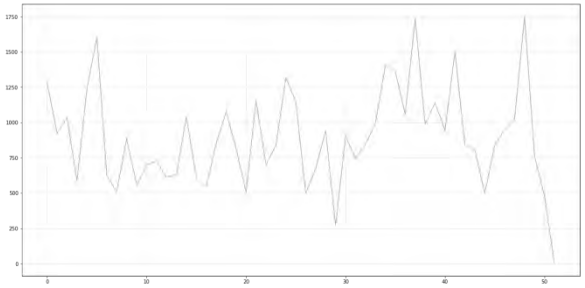

Figure 3: Distribution of the signal between the samples in the trace

To select the most informative samples, 2 algorithms were considered: based on the difference of mean values (DOM) and based on the square of the difference of mean values (SOSD).

Based on the difference of mean values, the samples of the resulting vector $\overline{X}$ can be found from the equation 7:

$$\overline{X} = \sum_{j=1}^{n} \sum_{i=1}^{n} (\overline{x}_i - \overline{x}_j) \quad (7)$$

Based on the square of the difference of mean values, the samples of the resulting vector $\overline{X}$ can be found using the equation 8:

$$\overline{X} = \sum_{j=1}^{n} \sum_{i=1}^{n} (\overline{x}_i - \overline{x}_j) \quad (8)$$

Thus, after obtaining the parameters μ and Σ for each of the 16 bytes of the AES-128 key, the cryptanalyst will be able to obtain the value of the density of the multivariate normal distribution function based on the obtained measurements on the attacked device:

$$p_i(C) = \frac{\exp\left[\frac{-1}{2}(x_i - \mu)^T \Sigma^{-1}(x_i - \mu)\right]}{\sqrt{(2\pi)^n |\Sigma|}} \quad (9)$$

where x_i is the power consumption trace taken during the attack phase, μ and Σ are parameters describing the currently selected profile for hypothesis C, n is the dimension of x, μ and Σ. C takes values from 0 to 255 in the case of the byte model and from 0 to 9 in the case of the Hamming weight model.

Thus, for each trace of the attack phase, possible hypotheses are sorted, for each hypothesis, the parameters of the normal distribution prepared for it are applied, and the resulting value p of the density of the normal distribution is accumulated. After processing all the attack traces, we get a cumulative value of p for each possible hypothesis C:

$$p(C) = \prod_i p_i(C) \quad (10)$$

The hypothesis that has the largest cumulative value of p is the most likely hypothesis.

With large amounts of data collected during the attack, it makes sense to reduce the order of magnitude of p so that the value of p can be processed by 64-bit data types. For these reasons, we used $\log(p(C))$ values in this paper instead of $p(C)$:

$$\log\big(p(C)\big) = \sum_i \log\big(p_i(C)\big) \quad (11)$$

By maximizing the cumulative value of the distribution density for each hypothesis C of each of the 16 bytes of the key, the attacker can sequentially recover the entire AES-128 encryption key byte by byte.

In the case of building a template based on the Hamming weight, the maximum value of the density of the normal distribution $p(x)$ would give an opportunity to hypothesize only the number of non-zero bits in the key byte, and not the value of the byte itself. In this case, it is necessary to restore the value by means of brute-forcing of all byte values corresponding to this value of the Hamming weight. If the cryptanalyst knows all 16 values of the Hamming weight, then in the worst case, when all 16 hypotheses have a Hamming weight of $P = 4$, the number of possible values of the encryption key for brute force will be $N = 70^{16} \approx 33 * 10^{28}$, since 70 of the possible values correspond to the Hamming weight of 4. On the other hand, without information about P for the key bytes, the cryptanalyst would have to brute force the sequences of 16 bytes. In this case, the number of the possible encryption key values would be $N = 256^{16} = 2^{128} \approx 34 * 10^{37}$, which is 9 orders of magnitude more than the number of such possible values with known Hamming weights.

In practice, in most cases, the Hamming weight will not always be 4 for all 16 bytes. In the case when the Hamming weight $P = 0$, there is only one key byte value corresponding to this weight, 0; similarly for $P = 8$ and other values. Thus, in real attacks using information about the Hamming weights of key bytes, the number of attempts is less than $33 * 10^{28}$.

2. Experimental details

2.1. Description of the experimental layout

To detect changes in the power consumption by the information processing chip, we need to register the current consumed by the chip. Having a stabilized power source (in our case – the USB power output of the laptop connector), we can include a serially connected resistor of an appropriate value into the power supply circuit of the Atmega328PU microcontroller. Thus, we will get the following wiring diagram of the controller and additions to the Arduino UNO board (Figure **4**):

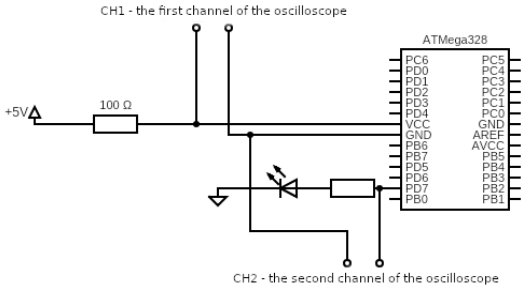

Figure 4: Schematic diagram of the experimental bench wiring

Figure **4** shows the connection diagram of two oscilloscope probes. The first channel CH1 is used to register changes in the voltage on the power contacts of the microcontroller, while the second channel CH2 is used to record the state of the signal LED.

The signal LED is necessary to separate the power consumption data corresponding to the encryption process itself. While the encryption of 16 bytes of plaintext is taking place, the LED is on, power is applied to it, and the oscilloscope registers a plot of the Heaviside function that makes it possible to separate the trace from other data of the microcontroller power consumption, such as downloading plaintexts from the control computer and uploading data back.

A Hantek 6022BE two-channel USB oscilloscope with a total sampling rate of 48 MS/s, 24 MS/s for each channel, was used for the experiments. But in this work, a much lower frequency of 1 MS/s was used. In practice, it turned out that a higher frequency creates less informative traces, which do not lead to greater detail of the signal leaking through the side channel, but, on the contrary, increase the noise component. In addition, a higher frequency produces a larger amount of data that very quickly fills the oscilloscope buffer, which does not have time to download data via the USB 2.0 interface to the control computer.

Figure **5** shows the assembled experimental bench. An Arduino Uno board and a 20x20 pin breadboard were used to assemble the elements.

Control and synchronization of the entire system was carried out using a Lenovo ThinkPad E570 computer with an Intel i5-7200U processor (4) @ 3.100GHz and 16 GB of RAM running the Linux operating system, kernel version 5.7.19.

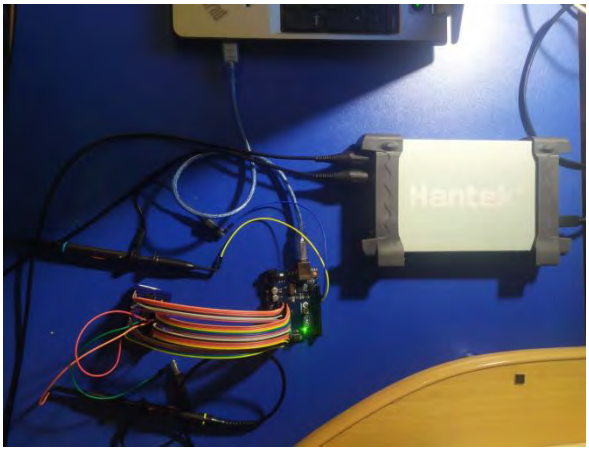

Figure 5: The experimental bench in the assembled state

The open API for Hantek oscilloscopes [16] in the Python programming language was used to control the oscilloscope.

To synchronize the Arduino board and the Hantek oscilloscope, a Python script was developed implementing the data flow according to the diagram in Figure **6**.

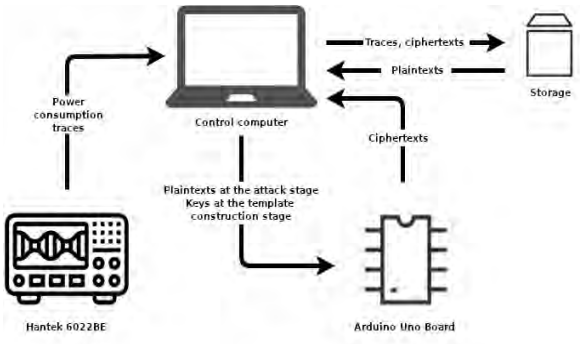

Figure 6: Experimental bench data flow diagram

The most difficult tasks were sending 16 bytes through the serial interface of the computer to the board, synchronizing the oscilloscope and the board with the microcontroller, and analysis of the output signal from the oscilloscope after each

encryption iteration. Let's consider these three operations in more detail.

2.2. Communication between the control computer and the Arduino Uno board

Both stages of the attack require communication between the Arduino Uno board and the computer. This communication is necessary for both data exchange and synchronization. On the control computer side, we used the interface */dev/ttyACM0* provided by the operating system. The interaction with the interface was carried out through the Python library *pyserial*. The data update frequency in the serial interface in our experiment was 9600 updates per second. This frequency is sufficient, while a higher frequency sometimes leads to overwriting the 64-byte buffer of the Arduino and thus corrupting the transmitted data. A delay of 0.01 seconds was introduced to ensure that the code on the Atmega328PU side has time to process the input byte.

Since the frequency of the controller is much higher than the frequency of the serial port, there is a need for an infinite loop that checks the buffer of the slow serial port for incoming data. Upon receiving 16 bytes of input, the Arduino board stops waiting for new data and continues the algorithm for processing the received data.

While the controller board has a serial USB port buffer limited to 64 bytes, there is no such limit in the case of the Linux OS, so the huge difference in the frequency of operation between the controller and the Python script is compensated by an almost unlimited buffer on the side of the control computer.

Thus, it is possible to transfer and receive bytes of data through the USB serial port successfully in most cases. However, in some rare cases various desynchronizations occur, that lead to data corruption at any of the four described stages. Such problems are very difficult to debug and they can be reliably solved by development of more complex synchronization algorithms, what goes far beyond the scope of this paper.

2.3. Synchronization of the oscilloscope and the microcontroller

Synchronization between these two nodes of the experimental bench turned out to be a difficult task for several reasons:

- Limitation of internal buffer of Hantek6022BE oscilloscope
- Limitation of the packet transfer rate from the oscilloscope to the USB interface of the computer and limitation of these packets size
- Different initialization and clearing times of Hantek6022BE oscilloscope buffer

These reasons determine the following feature implemented in the controller firmware – encryption with the same plain text and key occurs 100 times, both at the stage of building the template and at the attack stage. The number 100 was chosen through experimental studies, as this number of encryption stages is sufficient to expand the window of successive encryptions enough to overlap it with the measurement window of the oscilloscope, which by then manages to initialize in most cases.

The measurement window of an oscilloscope with a frequency of 1 MS/s was also quite small. In practice it was observed that in most cases it was even less than 100 iterations of encryptions on the controller. In all cases observed, the available measurement window was less than 1 ms required to send one byte to the controller and process it. Hence, it is impossible to first start the measurement and only then send data to the Arduino Uno board.

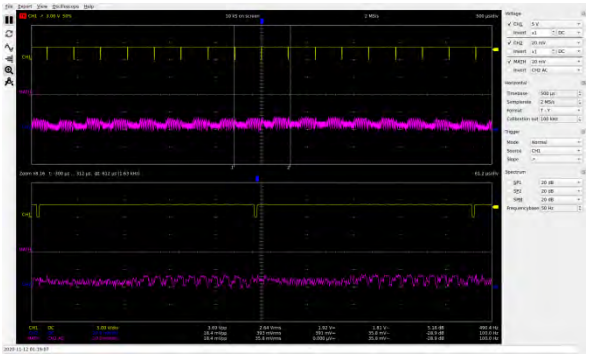

Figure 7: Trace view within a window of 100 encryptions

A window with several iterations of encryption, created for synchronization purposes, can be seen in Figure **7** in the form of consecutive identical traces, separated by a rectangular signal with

a low duty cycle, which corresponds to the time interval during which the encryption takes place.

2.4. Analysis of the output signal

After receiving the output signal, we have a graph similar to the one shown in Figure **8**. To separate and record one trace from it, we use the signal from the second channel containing rectangular pulses. Each of these pulses defines a part of the power consumption data that corresponds to the encryption process. Only this short part of the signal contains data about the encryption key.

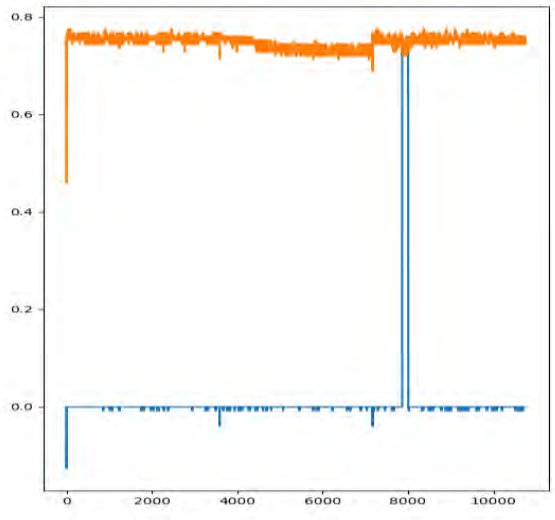

Figure 8: Traces from two oscilloscope channels combined on one time interval

To isolate this part of the signal, we need to set a threshold parameter for the pulse signal. Its value depends on the operating frequency of the board, the operating frequency of the oscilloscope, the level of its sensitivity, the quality of the digital signal generator used in the microcontroller. For our measurements, the threshold value of 0.7 V was chosen experimentally.

The data registration algorithm, implemented as a Python function, skips part of the square wave at the beginning, to account for the possibility that the measurement started in the middle of the square wave, and we no longer have the first part of the trace. Next, upon registering the leading edge of the rectangular signal, the function records all the values of the samples from the power supply pins of the Atmega328PU controller, corresponding in time to the moment of encryption.

As the result, we get a trace of 286 samples (Figure **9**):

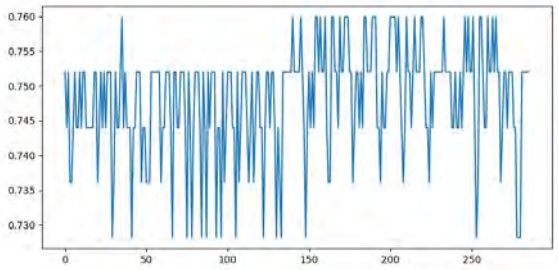

Figure 9: Sample values *vs* time for a sample trace collected on the experimental bench

2.5. Firmware for Arduino for the profiling stage and for the attack stage

Two variants of firmware for the Arduino Uno board were developed in the sketch language and using the library [1]. The first sketch was designed to gather data at the profiling phase to build the template, while the second sketch was designed to gather data at the attack phase to register traces of power consumption.

At the stage of data collection to build the statistical template of the Atmega328PU, according to the described algorithm for building a window to synchronize the Arduino and the oscilloscope, we form 100 iterations of encryption with the same plaintext and the same key, which we received through the serial USB port from the control computer. The Arduino board receives pre-generated encryption keys, and to obtain pseudo-random plaintexts, a chaining mechanism is used, which uses the previous ciphertext as the plaintext for the next encryption. In our experiment, the first plaintext was the string: "0123456789012345", the length of which corresponds to 16 bytes in ASCII encoding.

A fixed key, '1234567890123451' was used to collect power consumption data at the attack stage. The firmware for the attack is identical to the firmware for the profiling stage and differs only in that the plaintext received from the control computer is copied at 100 encryption repetitions.

3. Implementation of the template attack and analysis of the obtained results

3.1. Building an attack template

Based on the mathematical model described in the Section 1, we developed a software implementation of the algorithm for building a profile for cryptographic calculations on an unprotected AVR general-purpose Atmega328PU processor. Let's consider this implementation in detail.

After receiving profiling traces, we must first classify them according to the combination of plaintext and encryption key. Depending on which byte of the encryption key we attack, we must specify which byte numbers should be used for the XOR operation to sort all traces. Also, the number of equivalence classes in our data partition depends on the classification model: in the case of the byte model, it is 256 classes, while in the case of the Hamming weight model, it is 9 equivalence classes.

Having obtained the necessary equivalence classes, we need to build a profile based on them for each set of traces. From a practical point of view it is necessary to reduce the dimensionality of the data, because it is very difficult to operate with matrices of 286 by 286 samples even on the most modern server processors, taking into account the fact that for the attack stage we need to perform such resource-intensive operations as, for example, building inverse matrix.

To select points of interest for both described algorithms, it is necessary to find a signal trace. In the case of the DOM algorithm, it corresponds to the difference of the mean values, and in the case of the SOSD algorithm, it corresponds to the square of the difference of the mean values.

For each *i*-th and *j*-th element, we find the mean value of the sample in the set corresponding to the current *i*-th and *j*-th element in the corresponding equivalence classes. At the same time, *i* cannot be equal to *j*, because in this case the sought difference will always be equal to 0, which makes no practical sense. Then, *k* search samples are selected, in which the difference between the corresponding samples has the largest absolute value, where *k* is a parameter that allows us to adjust the dimensionality of the templates, and therefore the speed and accuracy of the algorithm.

After choosing a certain sample as a significant one, it must be zeroed, as well as the samples that are close to it on the time scale, because with a high probability they represent the same signal and the choice of the adjacent samples would result in us obtaining the same information multiple times, potentially missing other important signals.

Now, with trace partitioning classes and current points of interest selected, we are ready to construct the templates, namely the covariance matrix Σ and mean vectors μ.

As a result of the operation of this algorithm, we will eventually receive 9 pairs of Σ and μ parameters, ready for use at the attack stage for the Hamming weight model, and 256 such pairs for the byte model.

3.2. Attack phase using a pre-built template

Now, having collected power consumption data from the attacked device, in our case the same device — an Atmega328PU with a software implementation of AES-128 in AVR assembly language, we can use pre-built templates to recover the encryption key from the information leaked via the side channel.

To do this, we need to find the density value of the multivariate normal distribution for each trace collected and find the sum of logarithms of these values for the entire data set (see equation 11).

One of the problems that may arise at the attack stage when finding the value of the density of the normal distribution is the irreversibility of the covariance matrix. In this case, it is necessary to change the parameter k – the number of the most important samples, which was chosen during the construction of the profile, and to rebuild the profile anew. With a high probability, the inverse matrix can be found for all reconstructed covariance matrices.

3.3. The results of the template attack on a non-protected AES-128 software implementation

As we have mentioned above, the attack was carried out on the AES-128 encryption key K="1234567890123451". Several bytes of the cryptographic key were selected for analysis, in particular, the byte #3 0x34, which corresponds to the ASCII character of the code: '4'.

Using the example of this part of the encryption key, it is easiest and most obvious to consider the results of the attack. To restore this byte, relatively small amount of data is required, only 20,000 traces to build the template and 10,000 traces of power consumption at the stage of the attack. In addition, to reproduce the results described below, it is enough to select only 11 out of 286 samples to reduce the dimensionality of the data. Thus, it will be quite a simple task to reproduce the given results on an experimental bench.

Figure **10** shows a graph of the density values of the multivariate normal distribution for byte #3.

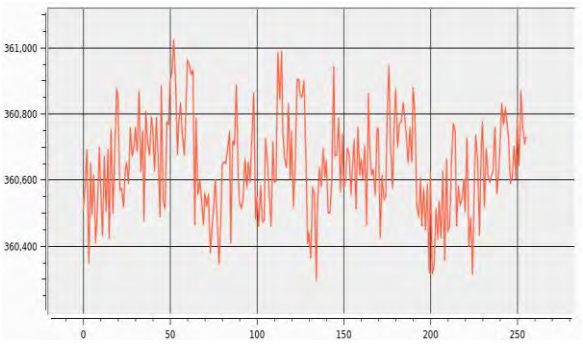

Figure 10: Density values of the normal distribution for the encryption key byte hypotheses

It can be seen that apart from the hypothesis with the highest value of the normal distribution density, which corresponds to the correct hypothesis 52_{10} (in the graph) or 0x34, there are several incorrect hypotheses that also show fairly high values of the normal distribution density. This suggests that more data is needed in the profiling phase for more accurate results.

Let's consider the graph of reduction of guessing hypotheses entropy regarding the value of byte #3 of the AES-128 encryption key (Figure **11**).

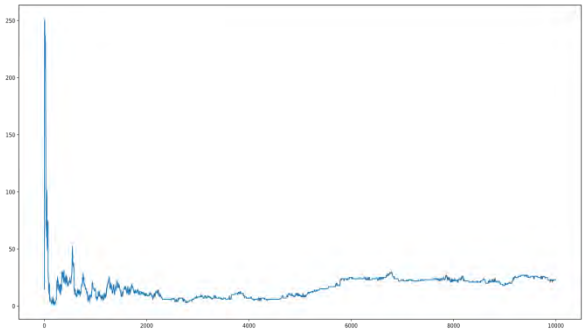

Figure 11: Value of encryption key guess entropy for byte #3

It can be seen from this figure that the correct hypothesis consistently wins only after about 3,000 traces required for the attack phase.

The value of the logarithm of the density of the Gaussian distribution for the correct hypothesis was: $p = 3.61022*10^5$.

By changing the parameter of the number of traces used for the attack and the parameter of the number of selected points in each trace, you can achieve more accurate results, while losing the speed of the algorithm. However, in some cases, the speed of the attack is not the main parameter.

4. Countering AVR328PU template attacks at the software level

4.1. An overview of some techniques for protection against power analysis attacks

According to the book [3], masking and noise contamination are the most commonly used methods of protection against power analysis attacks.

Masking can be thought of as a type of randomization of data processed by a cryptographic device. In this way, we separate the signal flowing through the side channel from the data being processed. When masking is used, leak-sensitive data is divided into several shares. These layers are created from raw data and processed separately until the very end of data processing. After that, by inverse transformation, they are restored as if the data processing took place without division into parts.

As the number of shares increases, the amount of power consumption data required for cryptanalysis increases exponentially. On the other hand, the use of such a mechanism of protection against signal leaks via side channels requires a source of entropy, which is known to provide an additional computational load for the cryptographic implementation.

There are several schemes for constructing secret partition schemes or shares: threshold scheme, Shamir scheme, Blackley scheme, schemes based on the Chinese remainder theorem, etc. Let's consider the threshold scheme in more detail.

The threshold scheme [17] proposes to reduce the required entropy for data masking through the use of permutations, similar to what is done in the underlying AES – Rijendael algorithm. Thus,

there is no need to generate a pseudo-random sequence at each round of AES-128 encryption. So, by using the threshold scheme, we can reduce the computational load on our cryptographic implementation. In figure **12** the scheme of masking according to the threshold scheme is presented, where x is the original secret, x_1, ... , x_s are shares.

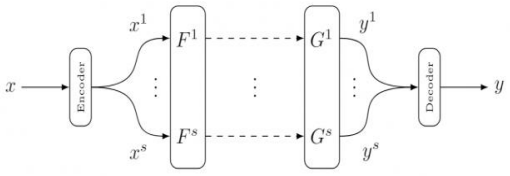

Figure 12: Scheme of masking using shares [18]

The effectiveness of masking increases exponentially with the increase in the number of shares.

The noise contamination of the signal consists in adding entropy to the power consumption side channel itself. Examples of the implementation of this idea can be the direct addition of white noise to the power circuit, and the construction of circuits with a double power source.

The mentioned methods of protection against attacks through an information leakage side channel have proven themselves well in practice, mostly in combination.

All mentioned mechanisms are effective in countering power consumption side-channel attacks. However, a well-implemented and configured template attack can easily cope with these countermeasures [19].

4.2. Proposed countermeasures against cryptographic device template attacks, available for implementation in the form of firmware

The above-mentioned side-channel leakage prevention mechanisms are definitely necessary for implementation in secure cryptographic devices, but they are not a complete protection against higher-level attacks.

If it is not possible to replace an existing vulnerable device with a more secure one, there is a way to at least increase the amount of data required for a successful template attack. For this purpose, it is possible to add random complex calculations intended to:

- Randomly increase the power consumption of the device
- Create additional noise in the outgoing signal
- Complicate building of a statistical profile and blur the statistical picture

Therefore, an algorithm was proposed that calculates the square root of a random number. The effect of running this algorithm on the entropy of guessing hypotheses after building a profile and attacking the software implementation of the AES-128 algorithm on the Atmega328PU microcontroller described in the previous sections is demonstrated in the Figure **13**:

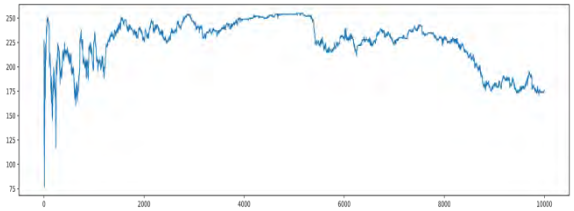

Figure 13: Key guessing entropy dependence *vs.* the number of traces for encryption key byte #3 after addition of the random calculations

As we can see, this algorithm made it almost impossible to build high-quality statistical templates for the byte value model when using 20,000 traces at the template construction stage and 10,000 traces at the attack stage with the parameter k=10 for the encryption key byte #3.

The graph of the density values of the multivariate normal distribution for our hypotheses is shown in the Figure **14**:

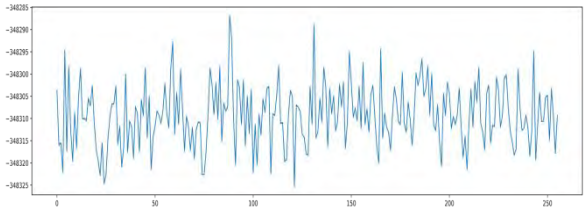

Figure 14: Density of normal distribution for encryption key byte hypotheses

Thus, we managed to complicate the process of building a profile for this cryptographic implementation.

We have to emphasize that this does not mean a complete protection against template attacks. Having sufficient data to analyze and, accordingly, well-chosen parameters, it is still possible to at least significantly reduce the number of possi-

ble variants of key byte values for brute force attack.

However, by combining this mechanism with masking and noise contamination, it is possible to achieve such a level of protection against template attacks in which the amount of data required for collection and analysis is so huge that it makes this type of attack purely theoretical.

Conclusions

Side channels of information leakage were and remain a promising direction for cryptanalysis, because they open up the possibility of attacking not the encryption algorithm, but its implementation. Moreover, information leakage side channels, and the power consumption information leakage channel in particular, cannot be completely eliminated due to their physical nature. Finally, it is impossible to track or in some way understand at what exact moment in time a side-channel attack is carried out, because there is no interference in the operation of a particular device.

To carry out such an attack, a cryptanalyst needs only a copy of the device and an oscilloscope. Building the device power consumption template with the correct choice of the profiling algorithm and dimensionality reduction can be fast enough, and the resulting templates can be accurate enough to quickly recover the encryption key even with a small amount of data obtained at the direct attack stage.

In our work, we made a review of various types of power analysis attacks, and chose a particular type to carry out a simulation of such attack, namely, an attack with building a statistical template based on the parameters of the Gaussian distribution with a byte model and without preprocessing the input data for the attack. The object of the attack was the implementation of the AES-128 encryption algorithm in the AVR Assembler for the Atmega 128PU microcontroller from Atmel based on the Arduino Uno hardware platform

During the preparation of the experimental bench, we found that while the wiring is quite simple, the serious problems exist in synchronization of the three main components – the microcontroller under investigation, the oscilloscope, and the control computer, and the most problematic were the limitations of the Hantek 6022BE oscilloscope buffer. We compensated these limitations by forging the firmware for the microcontroller, and while such solution was acceptable for our experiments, it would not be acceptable for a real attack on a cryptographic device. Thus, the requirements for the oscilloscope may be quite demanding.

Our experiment demonstrated the possibility to relatively easy recover bytes of the cryptographic key with a fairly small amount of input data, both for profiling the attacked device and for the attack. For greater accuracy in determining key byte hypotheses, an effective way (besides increasing the collected amount of data) is to combine attacks with an encryption key byte value model and a Hamming weight model. In our case, such a combination could be used for those bytes of the key for which the correct hypothesis is not found immediately, or does not reach the maximum value of the logarithm of the density of the normal distribution.

Various methods of protection against this type of attack have been developed, which include noise contamination, masking, and a number of other countermeasures. In this work, one of the protection mechanisms (i.e., adding some random calculations to the encryption process) against power analysis template attacks was proposed and tested. It demonstrated its effectiveness in this specific case. Among the shortcomings of the proposed mechanism, its resource-intensiveness should be mentioned.

However, it is not a universal protection neither against this type of attack nor against other side-channel attacks. To prove the absence of information leakage through a side channel, statistical tests should be used in each individual case.

References

[1] Avr-crypto-lib. URL: https://github.com/cantora/avr-crypto-lib

[2] TrezorWalletBroken. URL: http://johoe.mooo.com/trezor-power-analysis/

[3] S. Mangard, E. Oswald, T. Popp, Power analysis attacks – revealing the secrets of smart cards: Springer, 2007.

[4] S. Chari, J. R. Rao, P. Rohatgi, Template attacks, in Proceedings of the 4th International Workshop on Cryptographic Hardware and Embedded Systems (CHES 2002), Redwood Shores, CA, USA, August 13-15, 2002, pp. 13–28. doi: 10.1007/3-540-36400-5_3

[5] G. Fan, Y. Zhou, H. Zhang, *et al*, How to Choose Interesting Points for Template Attacks? Cryptology ePrint Archive, 2014, URL: https://eprint.iacr.org/2014/332.pdf.

[6] NewAE Template Attacks: URL: https://wiki.newae.com/Template_Attacks.

[7] Yoo-Seung Won, Bo-Yeon Sim, J.-Y. Park, Key schedule against template attack-based simple power analysis on a single target, Applied Sciences, 10 (2020), p. 3804.

[8] O. Choudary, M. G., Kuhn, Efficient template attacks, in: Revised Selected Papers of the 12th International Conference on Smart Card Research and Advanced Applications (CARDIS 2013), Berlin, Germany, November 27-29, 2013, Springer, 2013, pp. 253–270.

[9] S. Picek, A. Heuser, S. Guilley, Template attack vs Bayes classifier, Cryptology ePrint Archive, 2017, URL: https://eprint.iacr.org/2017/531

[10] L. Lerman, R. Poussier, G. Bontempi, *et al*, Template attacks *vs.* Machine learning revisited (and the curse of dimensionality in side-channel analysis), in: Revised Selected Papers of the 6th International Workshop on Constructive Side-Channel Analysis and Secure Design (COSADE 2015), Berlin, Germany, April 13-14, 2015, pp. 20–33. doi:10.1007/978-3-319-21476-4_2

[11] M. O. Choudary, M. G. Kuhn, Efficient stochastic methods: profiled attacks beyond 8 bits, Cryptology ePrint Archive, 2014, URL: https://eprint.iacr.org/2014/885

[12] C. Archambeau, E. Peeters, F.-X. Standaert, *et al*, Template attacks in principal subspaces, in Proceedings of the 8th International Workshop on Cryptographic Hardware and Embedded Systems (CHES 2006), Yokohama, Japan, October 10-13, 2006, Springer, 2006, pp. 1–14.

[13] Y. Oren, O. Weisse, A. Wool, Practical template-algebraic side channel attacks with extremely low data complexity, in: Proceedings of the 2nd International Workshop on Hardware and Architectural Support for Security and Privacy (HASP '13), June 2013, Article No.: 7, pp. 1–8. doi: 10.1145/2487726.2487733

[14] OpenHantek. URL: http://openhantek.org/

[15] J. Heyszl, K. Miller, F. Unterstein, *et al*, Investigating profiled side-channel attacks against the DES key schedule, Cryptology ePrint Archive, 2019, URL: https://eprint.iacr.org/2019/1448

[16] Hantek Python API. URL: https://github.com/Ho-Ro/Hantek6022API

[17] B. Bilgin, B. Gierlichs, S. Nikova, *et al*, A more efficient AES threshold implementation, Cryptology ePrint Archive, 2013, URL: https://eprint.iacr.org/2013/697.pdf

[18] CryptoBlog: URL: https://www.esat.kuleuven.be/cosic/blog/higher-order-threshold-implementations-wedding-cryptanalysis-to-masking/.

[19] E. Oswald, S. Mangard, Template attacks on masking – Resistance is futile, in: Topics in Cryptology – CT-RSA 2007, The Cryptographers' Track at the RSA Conference 2007, San Francisco, CA, USA, February 2007, pp. 243–256.