ABBY: Automating leakage modeling for side-channels analysis

Omid Bazangani, Alexandre Iooss, Ileana Buhan, Lejla Batina
Radboud University

Abstract

We introduce ABBY, an open-source side-channel leakage profiling framework that targets the microarchitectural layer. Existing solutions to characterize the microarchitectural layer are device-specific and require extensive manual effort. The main innovation of ABBY is the collection of data, which can automatically characterize the microarchitecture of a target device and has the additional benefit of being scalable.

Using ABBY, we create two sets of data which capture the interaction of instructions for the ARM CORTEX-M0/M3 architecture. These sets are the first to capture detailed information on the microarchitectural layer. They can be used to explore various leakage models suitable for creating side-channel leakage simulators. A preliminary evaluation of a leakage model produced with our dataset of real-world cryptographic implementations shows performance comparable to state-of-the-art leakage simulators.

1 Introduction

Kocher et al. [33] showed that showed that the power consumption of a device correlates with the data it processes, allowing the recovery of a cryptographic key. This interaction appears in physical side-channel(s) such as power [3, 10, 35], electromagnetic emanation [2, 47] or photonic emission [13]. An adversary can take advantage of these side channels and learn secret information during processing. Many studies show successful side-channel attacks leading to the recovery of secret keys or shares on various platforms [8, 21, 22, 26, 30, 41]. Technological advancements and twenty years of sustained effort by the cryptographic community significantly increased the workload required for successful key extraction. However, the problem of implementing a secure cryptographic algorithm on a given target device is not solved. The challenge for a developer is to balance the presence of countermeasures against information leaks. As the product changes during development, it is important to understand whether the changes are beneficial or, in contrast, whether they compromise the security of the implementation.

The appeal of side-channel *leakage simulators*, which model the instantaneous power consumption of a device, is evident from the effort towards creating such tools [12]. A leakage simulator generates side-channel measurements from a sequence of instructions with the help of a *leakage model*, a function that describes how the target devices consume power. In the absence of tools such as leakage simulators, a security researcher tasked with hardening a cryptographic implementation, will measure traces, detect leakage, change the implementation and reiterate until the implementation stops leaking. The process is slow, error-prone, and expensive. Moreover, the absence of leakage does not guarantee that there is no attack possible. A leakage simulator can automate the detection of side-channel leaks and, more importantly, can be used to explain the cause of a leak. A leakage simulator can automate the detection of side-channel leaks and, more importantly, can be used to explain the cause of a leak.

A leakage simulator transforms high-level code into traces similar to those collected from the target architecture. When adequate and informative, leakage simulators assist in the design of secure cryptographic implementation provided that the leakage model accurately reflects the reality of a key recovery attack. As in [43], we distinguish between *value* and *transition*-based leakage models. A leakage model is value-based if it takes the intermediate values of a cryptographic algorithm as arguments. Examples include the Hamming weight (HW) or the identity model (ID). A leakage model is *transition-based* if it takes as parameters any pairwise combination of intermediate values [43] such as the Hamming distance (HD) model which captures events such as the update of a register.

A popular defense against side-channel attacks is masking, where each secret value uses multiple shares [14, 32] to break the dependency between the power consumption and the processing of a given variable. Capturing interactions between intermediate variables is essential for verifying the correctness of a masked cryptographic implementation. However, theoretically secure masking implementation often fails in practice due to unexpected interactions between variables [9, 45].

Creating a transition-based leakage model is specific to the

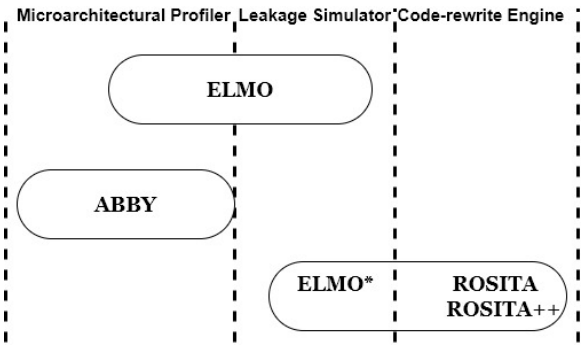


Figure 1: Overview of ABBY, ELMO, and ROSITA in application level design

target and requires intensive manual effort [36, 37, 50, 61]. Capturing the interaction of intermediate values requires profiling the microarchitectural layer, which may contain hidden storage elements where unexpected interactions between instructions can occur. Although the importance of transition-based leakage models has been established [7, 25, 36, 50], the details of microarchitectural implementation are often considered a trade secret and, therefore, not public information. By automating the creation of leakage models, more side-channel simulators, such as ELMO [37] which cover different architectures can be created with reasonable effort. *Automated microarchitectural profiling*, which captures the interaction between variables due to microarchitecture optimization is necessary for creating leakage simulators for different architectures.

Leakage simulators are the building block for rule-driven code rewriting engines such as ROSITA [50] and ROSITA++ [49] that patch the code automatically once the leakage is detected. Fig 1 shows the functional relationship between these tools, from an architectural profiler to a side-channel code rewriter engine. For example, ELMO as a side-channel simulator developed its microarchitectural profiler based on Thumbulator [58](an Instruction Set Simulator). ROSITA developed a code rewrite engine based on the ELMO* (upgraded ELMO) simulator combined with an assembly code modifier.

Although the methodology for building leakage simulators is known, the main limiting factor for their wide adoption is the limited number of supported target devices, a direct consequence of the effort required to reverse engineering the microarchitectural implementation. As the main barrier to overcome for the widespread deployment of leakage simulators is the characterization of the target device, we ask the following question: *Can we automate microarchitectural profiling of a chip to speed up the process for side-channel simulator design ?*

Summary of contribution. This paper makes the following contributions:

- We present ABBY, the first generic framework to automate the creation of training data used for creating a side-channel leakage simulator;
- Using ABBY we create two data sets ABBY-CM0 and ABBY-CM3 which capture the interaction of instructions for the ARM Cortex-M0 and Cortex-M3 boards.
- We applied a qualitative and quantitative analysis on ARM Cortex-M0 and Cortex-M3 leakage models conducted with statistical tests, dedicated leakage test vectors, and correlation power analysis.
- We demonstrate that deep learning can be used for realistic leakage models.

We propose ABBY, an open-source framework that *automatically* captures microarchitectural leakage. Furthermore, we offer the ABBY-CM0 dataset for ARM Cortex-M0 chips (STM32F0 families) produced with the ABBY framework. Based on our knowledge, this is the first open-source dataset to profile the microarchitectural layer, which can be used to study a target device's profiling further. In addition, we develop different transition-based leakage models, which allow us to create different simulators. We investigate the model's performance based on statistical parameters and learn how different microarchitectural features contribute to leakage. We compare the performance in detecting leakage of these simulators with ELMO and show that the performance is comparable to ELMO. We believe it is possible to improve further the leakage model.

Paper organization: Related works on microarchitectural leakage simulators are mentioned in Section 2. Section 3 briefly introduces the background on side-channel attacks and leakage detection and describes the hardware setup we used. Section 4, discusses building transition-based leakage models. We introduce our dataset in Section 5. In Section 6, we build several leakage simulators using the ABBY-CM0 data set and discuss their performance. Section 7 concludes the article.

2 Related works

Pinpas [20] is the first side-channel leakage simulator. Soon many more followed [19, 23, 44, 54]. However, the first open-source side-channel leakage simulator was SILK [56], which captures no specific hardware architecture and targets data-dependent power consumption. SAVRASCA [57] takes as input the compiled binary code and, using the tracing feature of the SimulAVR tool, will output simulated power traces for the AVR architecture. SAVRASCA was used to report a bug in the implementation code used for the DPAv4 trace set. ASCOLD [43] checks violations of the AVR architecture's independent leakage assumption (ILA). It takes as input the assembly file of the masked implementation and a configuration file of the system. The device shows the location of the

leak (line number) and the rule that was violated. The physical causes of the ILA breaching effects are device-specific (cannot be generalized) and counterintuitive when related to the assembly description of the target.

Going one level lower are the simulators, which capture some of the microarchitectural effects of the target. These simulators capture a more descriptive target model at the cost of a larger engineering effort. MAPS [34] is a power simulator designed for the ARM Cortex M3, which takes as input the source code of the masked implementation and outputs a simulated power trace. To capture the microarchitectural details, the authors used an HDL file of the target architecture and mainly focused on the leakage caused by the pipeline. In most cases, however, the target's HDL files are unavailable. We consider ELMO [37] to be the first transition-based leakage simulator for the ARM-Cortex M0/M4 family. ELMO models power consumption as a linear combination of values and transitions. Most remarkably, the simulator was created without detailed information on the hardware description or the target microprocessor. ELMO* [50] improves the leakage model of ELMO by capturing interactions that span multiple cycles. ROSITA [50] is a rule-driven code rewrite engine that automatically patches the code once a leakage is detected. ROSITA starts with a (masked) implementation of a cryptographic algorithm, cross-compiled to produce both the assembly and the binary executable. A very compelling feature of ROSITA is that it extends an existing leakage detection tool, ELMO [37], to report instructions that leak secret information. The new detection framework (ELMO*) uses the binary file to detect leakage and identify the offending machine instruction; ROSITA then applies a set of rules that replace the leaky instruction with an equivalent one (functionally) that does not leak. ROSITA repeats the process until no more leakage is detected. While the importance of microarchitectural details in a security analysis has been established [25], [36] access to its implementation is typically not available. The authors of ELMO had to reverse engineer the microarchitectural implementation of the target ARM Cortex M0 processor. The current state of the art allows reverse engineering a commercial ARM Cortex-M3 microprocessor [18, 25]. The authors note that the current methodology involves intensive manual effort. However, it is worth considering, as it shows the importance of capturing microarchitectural effects. A large body of works targets the creation of pre-silicon side-channel simulators [27, 29, 31, 42, 48, 51, 52, 60], which use design information of the target device to create the leakage model.

3 Background

3.1 Side channel.

Power consumption and EM signals emitted from a device correlate with the processed data and the executed instructions. The amount of power required to maintain a signal's value depends on the signal's logical state. In CMOS technology, the predominant choice when manufacturing integrated circuits, changing the value of a bit requires a different power level than keeping a bit constant. Therefore, the power consumption of a circuit directly correlates with the data processed by the circuit. Monitoring the physical properties of devices can reveal information about the operations and data processed. To perform a side-channel attack, an attacker attempts to correlate the observed physical side channels with the values processed by the device. Other effects, such as variations in signal propagation time or cross-capacitance effects, contribute to the device's power consumption and correlate with the data processes.

Leakage modeling. Let the set X be the data that we wish to monitor. When using side-channel analysis, X is typically the set of intermediate values created when transforming plaintext into ciphertext. We denote by $L(X)$ the leakage model of the variable X. An adversary collecting side-channel traces has access to variable y, defined using Equation 1. The measured power traces are conventionally considered noisy, and this Gaussian noise $N(0,\sigma^2)$ is independent of leakage $L(X)$ [24].

$$Y = L(X) + N(0,\sigma^2) \tag{1}$$

As $L(X)$ depends on the architectural design of the target and originates from the interaction between software and hardware. To improve this estimation and get it close to the real leakage, we consider the most relevant microarchitectural features of the target, such as instruction interaction, pipeline effects on instructions, operand values, and memory interactions. In this study, we consider the target as a "gray box" model. Although we do not have access to the design of the chip hardware description layer (white box), we have access to the instruction set architecture (ISA) and full control of firmware execution.

Leakage assessment. Test Vector Leakage Assessment (TVLA) [28] is one of the most popular methods for leakage detection due to its simplicity and relative effectiveness. It is based on statistical hypothesis tests and comes in two flavors: *specific* and *non-specific*. The 'fixed-vs-random' is the most common nonspecific test and compares a set of traces acquired with a fixed plaintext with another set of traces acquired with random plaintext. In the case of a specific test, the traces are divided according to a known intermediate value tested for leakage. In both cases, Welch's two-sample t-test for equality of means is applied for all trace samples. A difference between two sets larger than a given threshold is evidence of a leak's presence. Despite its simplicity, it is easy to misuse this test [59].

3.2 Regression model evaluation

The goal of most statistical models is to predict future events or to help explain reality [11]. In the former case, the quality of the model is defined by its predictive power. In contrast, the

quality of the model in the latter is related to the number of relevant factors it can identify. In leakage simulators, predictive models are used to estimate the power consumption of an intermediate variable. Although many options can fit the leakage model, we only consider linear regression in this paper. We use *coefficient of determination* (R^2) and *cross-validation* to judge the quality of the model we use, as these are popular choices to evaluate regression models [24]. For readability, we use the notation [24], [37].

R^2 measures how much of the variation in the dependent variable can be explained by the independent (explanatory) variable(s). An R^2 value close to one shows a good fit between the predicted and measured values. To compute R^2, we need to compute two types of sum of squares (SS). The first parameter is called residual sum of square(RSS) which measures how much of the explanatory variables' variation can not be explained by the model. It represents the sum of the squared differences between the actual measurement y_i and the predicted value $\widetilde{L}(Z_i)$(equation 2).

$$\mathrm{RSS} = \sum_{i=1}^{n} (y_i - \widetilde{L}(Z_i))^2 \tag{2}$$

where n is the number of samples. The second parameter explained sum of square (ESS) measures the variation of the explanatory variables(equation 3).

$$\mathrm{ESS} = \sum_{i=1}^{n} (\widetilde{L}(Z_i) - \bar{y})^2 \tag{3}$$

The total sum of square(TSS) is the sum of ESS and RSS(equation 4).

$$\mathrm{TSS} = \mathrm{RSS} + \mathrm{ESS} = \sum_{i=1}^{n} (y_i - \bar{y})^2 \tag{4}$$

The coefficient of determination R^2 can be calculated with equation 5 [24].

$$R^2 = \frac{\mathrm{ESS}}{\mathrm{TSS}} = 1 - \frac{\mathrm{RSS}}{\mathrm{TSS}} \tag{5}$$

The disadvantage of using the R^2 metric is that its value increases with increasing number of explanatory variables included in the model. To penalize additional explanatory variables added to the model and adjust this metric against the overfitting problem, we look at R^2 adjusted denoted by R^2_{adj}.

$$R^2_{adj} = 1 - \frac{(1-R^2)(n-1)}{(n-p-1)} \tag{6}$$

n is the number of samples, and p represents the number of explanatory variables fed to the model. According to Equation 6, if the number of explanatory variables is negligible compared to the number of samples($n \gg p$) then $R^2 \approx R^2_{adj}$.

F-test. To investigate the effect of adding different explanatory variables to the model, we used the F test introduced in [37]. We investigate the importance of explanatory variables based on their contribution to the model's performance. We check if a reduced model (fitted by a subset of explanatory variables) is missing a significant contribution compared to a full model, which consists of the full explanatory variables. Let us consider that B is a reduced model of model A. Therefore the number of explanatory variables of model A (p_A) is larger compared to the number of explanatory variables of model B (p_B), so we have $p_A > p_B$. In this example, the null hypothesis states that the extra parameters present in model A do not affect the model performance. The F-statistic is computed based on the residual sum of squares (RSS), while $p_A - p_B$ and $n - p_A$ are degrees of freedom as shown in equation 7.

$$\mathrm{F} = \frac{\left(\frac{RSS_B - RSS_A}{p_A - p_B}\right)}{\left(\frac{RSS_A}{n - p_A}\right)} \tag{7}$$

For a specific significance level (normally $\alpha = 5\%$), if the F value is more than the critical value under the $F_{p_A - p_B, n - p_A}$ distribution, the null hypothesis is rejected, which means that the parameters of model A, which are not present in model B have a significant effect.

3.3 Cortex-M0 Vs Cortex-M3 architecture

Cortex-M0 is based on ARMv6-M architecture with three stages pipeline. This architecture has the full support of the Thumb-1 instruction set and some of the Thumb-2 instructions, a total of 56 instructions [5]. Cortex-M3 with three stages pipeline is developed based on ARMv7-m architecture, having full instruction coverage for Thumb-1 and Thumb-2 with a total coverage of more than 90 instructions [6]. One of the key components used in this study for cycle counting is Data Watchpoint and Trace unit (DWT). All Cortex-M3 microcontrollers support the DWT register. Although ARMv6-M supports this feature, not all the cortex-M0 microcontrollers support the DWT register. Especially the chip that ELMO developed based on.

(Data Watchpoint and Trace) DWT is a debug unit that provides watchpoints, system profiling, and data tracing. This unit contains a cycle counter called Clock Cycles Counter (CYCCNT) to count the CPU clock cycle. We configure DWT in a hardware watchpoint mode to read DWT_CYCCNT register after each instruction processing to calculate the needed execution cycle for each instruction. There are two problems with using hardware breakpoints for each instruction. First, there are four available breakpoints. Second, breaking the normal process would affect the pipeline. We need to reuse available breakpoints (only one breakpoint is used) to solve the former problem simply. To investigate the second problem in detail, we can check it with a simple assembly code snippet like three LDR in a row. If we run this code without halting the process, it would take 4 cycles for these three LDR to get

executed, while halting the processor for each `LDR` takes 6 CPU cycles. The reason for this difference is related to the optimization that is happening in the pipeline which every `LDR` separately takes two cycles while for a series `LDR` in a row, only the first one takes two cycles and the rest would take one cycle to execute.

To solve the process halting effect on the pipeline and cycle counting, we are recording DTW_CYCCNT register while moving the breakpoint instruction by instruction until we reach the end of the target assembly snippet code Fig.13. In this way, we can calculate the cycle count for the first instruction(LDR) by a simple subtraction $X2 - X1$ To make sure that there are no other related instructions in the beginning that can affect the target assembly code snippet, we add 10 `NOP` instruction after and before the target assembly code.

3.4 Hardware Setup

For Cortex-M0 We use two different chips based on Armv6-M architecture manufactured by ST Microelectronics with STM Discovery Boards [38, 39]. The target boards have the `STM32F051R8T6` or `STM32F030R8T6` chip [40] and an external crystal oscillator (8MHz). Although we initially tested our framework on both boards, we continued on `STM32F030R8T6` target to compare the result with the ELMO. We modified the boards to filter the measurement noise by removing the power line capacitors and measuring the current through a current probe (Riscure CP271), which is used as a proxy for the target's power consumption. We used a PicoScope 3207B for data acquisition at a sampling rate of 500MS/s while the target runs at 8MHz. This oscilloscope can store up to 512Ms due to memory limitations. We used a physical 48MHz low-pass filter for the measurement. We are using two acquisition channels, a power signal, and a trigger signal. The trigger is fed by a GPIO of the target when the desirable segment of the code is running. The oscilloscope will be armed for signal recording as soon as the trigger signal is detected. Figure 2 shows the configuration block diagram. The setup for Cortex-M3 is as same as Cortex-M0 except for the clock source of the chip, which is fed by an external clock. The Cortex-M3 microcontroller has based on armv7-M architecture and is manufactured by ST Microelectronics `(STM32F107vct)` [40].

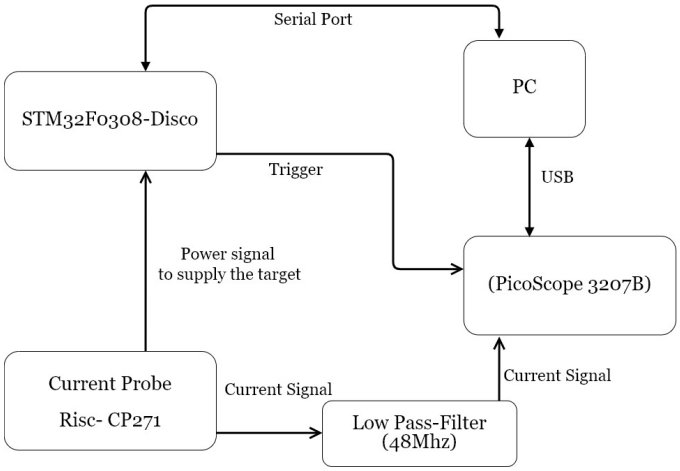


Figure 2: Setup Block Diagram

4 Automated Microarchitectural Profiler

In this section, we explain the challenges and innovations behind ABBY methodology to create an automated microarchitectural profiler. We address important questions such as 1) Why do we need automation? 2) what are the design requirements? 3) what are the design challenges? and 4) How does the ABBY framework tackle it?

Why do we need automation? Designing a side-challenge simulator like ELMO has different steps. These steps include selecting relevant instructions to decrease data space, profiling selected instructions, clustering instructions based on their profile, adding support for sequence dependency, and training a simulator model. despite the amount of effort and time-consuming, each step needs an expert decision to continue to the next step. Moreover, the ELMO model only supports symmetric crypto algorithms to reduce the effort in different design steps. We need automation to put all aforementioned efforts into a toolchain to design more sophisticated side-channel simulators.

Design requirements.Requirement 1: We need a model that covers as many instructions as possible from the target instruction set to not limited to some specific algorithms. Requirement 2: we need to remove human expert decisions between each step to decrease the needed effort. Requirement 3: we need a method to be scalable for other ARM Cortex-M families or even more different architectures.

Challenges. To satisfy the aforementioned requirements, we have some challenges to tackle. Challenge 1: To cover more assembly instructions, we need to consider how the instructions combination grows dramatically due to the pipeline effect. For instance, for a Cortex-M0 chip with 56 supported Thumb instructions, there are $56^3 \approx 175$K combinations. Challenge 2: to annotate the power trace with executing instruction in each clock cycle, we need to know how many cycles a specific instruction needs to be executed. Challenge 3: clock-drift might affect this annotation.

ABBY solution. ABBY Framework tackles these challenges by offering a methodology shown in Fig 3. ABBY framework handles the needed effort by adding more assembly instructions through an automated `Random Assembly Generator` block, which automatically generates, programs, and captures power traces for each firmware. To collect microarchitectural features, ABBY uses QEMU [53] and Genu Debugger(GDB) tools which are standard for not only dif-

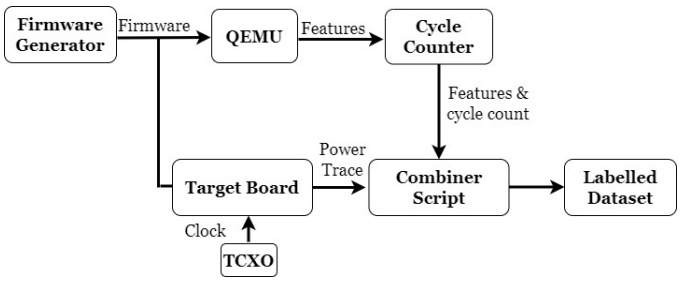


Figure 3: ABBY Cortex-M series block design

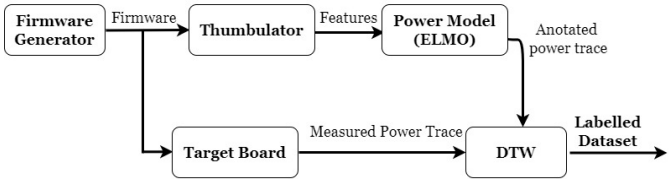


Figure 4: ABBY Cortex-M0 block design

ferent ARM cortex-M families but also other architectures. ABBY uses Dynamic Watchpoint (DWT) register for instruction cycle counting while feeding the target a precise external clock using a TCXO component. These two techniques help us label our power trace precisely with the related instruction processed in a specific clock cycle. The labeling process occurs in the Combiner block Fig 3.

Engineering challenges. Engineering challenges to fitting ABBY on a specific architecture are as follows.

– Supported by QEMU or support JTAG debugger

– Support GDB basic commands

– Support a cycle counter register like DWT

To extract microarchitectural features from a specific firmware, We can run it on a real target with a JTAG debugger or an emulation tool. ABBY uses a popular emulator called QEMU to speed up the process. QEMU supports many ARM Cortex-M families and many others under development. Even if QEMU does not support a specific target, using the JTAG debugger is an alternative option supported by all ARM Cortex-M families and even more architecture, as well as GDB.

All ARM Cortex-M families support the DWT register for cycle counting except Cortex-M0. To solve the cycle counting problem for Cortex-M0, we applied some modifications to the block design of ABBY. To annotate the power trace, we use the ELMO model that gives us a simulated trace with annotations. Next, we align the simulated trace by ELMO with the measured power trace using the DTW algorithm. As ELMO was developed based on Thumbulator, we replaced QEMU with Thumbulator for feature extraction.

5 Dataset creation

5.1 Feature Selection

The critical observation made by McCann et al. [37] when building the ELMO leakage model is that the power consumption of *the current instruction*, I_c depends on *the preceding instruction*, I_p and *the subsequent instruction* I_s [55]. The reason behind this observation can be found in the three-stage design of the target pipeline. Not only the instructions but also the operand values of these instructions contribute to the power consumption of the chip [37].

Instruction coverage of classical leakages model. Although executing instructions is one of the main contributors to the chip's power consumption, traditional leakage models only look at the HW or HD of each operand with its previous value. These models cover neither the pipeline effect nor instructions.

ELMO instruction coverage. ELMO is instruction-accurate, which has the advantage of allowing the quick identification of a leaky instruction. Following a cluster analysis to group "similar instructions" (that is, that leak information in the same way), the authors identify five groups, all of which include 21 instructions, see appendix Fig 12.

The groups correspond to the same processor component: ALU instructions in one group, shift instructions in another group, load and stores that interact with the memory are two or more groups and the `MULS` instruction with a distinct profile due to its fit in the implementation of the single cycle in a separate group. These groups also represent the internal structure we could expect from an ARM core since shift, multiplication, and arithmetic operations do not use the same CPU part.

ABBY-CM0 instruction coverage Similar to ELMO, ABBY captures the interaction between instructions in the pipeline registers and memory leakage. No assumptions about the operand interaction are made to simplify data collection and training. To keep the data collection process simple, ABBY leaves the modeling of the relation between operands for preprocessing. ABBY also adds memory read/write values for current and previous memory access to cover memory leaks, as these were shown to be important in [50].

As part of the effort to simplify training, we further remove the clustering of instructions. We keep the thumb instructions for crypto algorithms, representing 44 instructions for ARM Cortex M0, including arithmetic, shift, store, load, and multiplication operations. We do not profile branching, stack operations `PUSH`, `POP`, `LDR/STR sp`, or operations changing the `PC` register (Fig. 12) as these operations are not used for implementing cryptographic algorithms. For the storage and loading operations, we reserved a register `r0` to put the memory address of an empty data section. Furthermore, ignoring instructions such as branching `B`, `BEQ`, `BL` makes sense as block ciphers avoid using them to be time constant and to

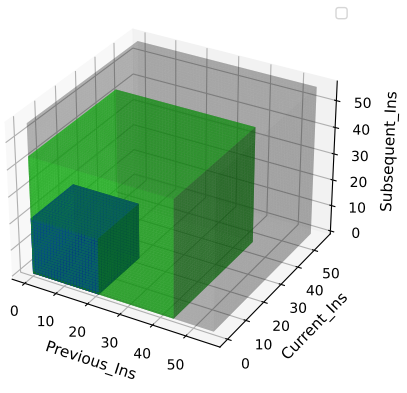


Figure 5: Possible instruction space in gray vs. ABBY and ELMO in green and blue respectively

prevent leaks from branch prediction.
For a 3-stage pipeline microprocessor, and considering 56 possible Thumb-instructions for Cortex-M0, the entire instructions space is 56^3. Figure 5 shows the coverage of the instruction space of ELMO vs. ABBY. We see that ABBY covers more instruction combinations compared to ELMO.

ABBY-CM3 instruction coverage.

The data collection phase is the same as CM0, with these differences that we target just one of the processing components in a CM3 chipset. Based on our knowledge, there is no cycle-accurate instruction simulator for CM3 like Thumbulator, and it's not our goal here to develop such a simulator. We reduced profiling space to the ALU component of CM3 as it guaranteed to consume one cycle per each different supported instruction of this unit[refernce of arm developer about this], and also this unit provides many popular instructions which are used in crypto algorithm [37]. We covered seven different instructions of ALU including `ADD`, `AND`,`CMP`, `EOR`,`MOV`,`ORR`, and `SUB`. Moreover, for these instructions, we considered different possible variants, like the version with suffix `S` or with and without immediate value and the extended version with suffix `.W`, which ended up with 17 different variants for these seven instructions. We implement 50 different firmware, each including 80 triplets of randomly chosen instructions from the aforementioned instruction space. For each instruction, we are collecting 1000 different traces while each of them is fed with random operands through the serial port. We ended up with $50 \times 80 \times 1\,000 = 4M$ data samples.

5.2 Automated Firmware creation

We automatically generate randomized assembly firmware, including triplet Thumb instructions, to cover the 3-stage pipeline of the target. We analyze the output to ensure that the instructions are uniformly chosen, and their operand value is uniformly distributed. As we do not have the clustering limitation (all instructions in one cluster), we keep all Thumb instructions typical for cryptographic implementations. The result is a group of 44 instructions for the ARM Cortex M0, including arithmetic, shift, store, load, and multiplication operations. For Cortex-M3, the result is 17 different instructions of the ALU component.

5.3 Dataset construction

Cortex-M0. Using the generated firmware, we collect 1,000 triplets [1] (or pipeline states) in a single acquisition. By generating and flashing[2] 50 000 different Random assembly firmware automatically, we collect for each triplet of instructions (with random operands) $\frac{50000 \times 1000}{44^3} \approx 587$ data points. Firmware with random instructions is loaded on the target device, and power consumption is measured while the target executes the firmware.
Cortex-M3. It is the same as the Cortex-M0 process except for the number of data points. We collect for each triplet of instructions (with random operands) $\frac{4M}{17^3} \approx 815$ data points.

5.4 Dataset labeling.

Cortex-M0. To label the measured power samples, we need to identify the corresponding triplet of instructions. A challenge when annotating the measured traces with the executed instructions is that different instructions might take different cycles, depending on the optimizations made by the manufacturer.

Our solution, dictated by the simplicity of the ARM Cortex M0 processor, is not perfect, but it is effective. We use ELMO as an initially estimated power consumption to compare with the feature annotation. Next, we replace the value generated by ELMO with the corresponding value extracted from the measured traces. ELMO is not cycle-accurate; the measured and estimated traces do not have the same number of samples. Our solution for aligning two-time series of varying sizes is *Dynamic Time Warping (DTW)*, a popular algorithm used for speech recognition. This technique solves the signal alignment problem and finds the corresponding power consumption value for each instruction triplet (Figure 4). DTW is not perfect, and alignment errors are possible. As a result of the first attempt to align the data, 22% of instructions

[1] A compromise between the oscilloscope memory and the duration of running Dynamic Time Warping for alignment ($t \sim O(n^2)$).

[2] We could run from RAM to preserve flash endurance, but it influences leaks and the number of cycles per instruction. Running the firmware from the flash is closer to real-world scenarios.

are dropped. We also expect errors in the remaining data, so we perform several passes. We observe that the alignment distance converges after four passes (there is no significant improvement with more iterations).

Cortex-M3. Dataset labeling for CM3 is a challenge due to the following limitations:

1. Lack of an open source ISS like Thumbulator
2. Lack of an available power model like ELMO

Lack of an open source ISS. Thumbulator is working only on ARMV6-m architecture. Based on our knowledge, there is no open-source cycle-accurate simulator for ARMV7-m that is the based architecture for CM3. To solve the feature extraction problem without an ISS tool, we take advantage of GNU Debugger (GDB). After loading firmware on the real target by one of the scripts, another script would run to extract features by using GDB-related commands through a J-link debugger. [Add a picture about how it can be done]. After we ran feature extraction using GDB on a real target, we observed that this process was time-consuming as we needed to stop on every assembly instruction to collect all operand values and related registers. We solved this problem using a well-known emulator called Quick Emulator (QEMU) [53]. By using QEMU, the process would be at least 10 times faster, and you would not need a physical setup for feature extraction. Fig. [add a figure] shows the architectural

Lack of available power model for CM3. Another limitation to extending the previous model for CM3 chips, is that there is no available power model to apply DTW based on it. To create our labeled dataset we need to annotate recorded power traces from a real measurement with the extracted features from the same execution. The main challenge for this annotation is that even with a slight drift in the signal annotation, like 100 nanoseconds in our case, as our target is running on 10MHz, we are not annotating the desired instruction but the next or previous instruction in the subsequent based on a positive or negative drift. So, if drift happens in power annotation, consequently it's leading to false labeling in the dataset and a corrupted dataset would not guarantee what the ML model would learn even if it learns anything of this dataset. One solution for this problem would be a precise execution timing to be able to calculate the execution timing of each instruction. For this precise calculation, first, we need to calculate the cycle count for each instruction and ensure that the target is fed with a precise clock. For the former, we know that our target execution cycle is deterministic, and the ARM developer manual guarantees it [4]. Based on the developer manual, all the ALU instructions need one clock cycle to execute. We confirm it in practice with the following methods:

1. Using Data Watchpoint and Trace unit (DWT)
2. Clock signal recording

Clock Signal Recording. As we record the clock signal fed to the Processor beside the power signal, we can count the number of clock cycles needed for our target assembly code to execute. This way, we can double-check the calculated clock with the ARM developer manual. For clock precision, We cannot rely on the chip's internal clock, as it's prone to drifts and changes by voltage or temperature changes. Among the options for external clock sources, we used a TCXO (Temperature compensated crystal oscillator) based clock source equipped with temperature changes compensation component. This solution provides a precise and steady enough clock cycle to can use for execution timing measurement. Fig. 3 shows this technique.

5.5 The ABBY dataset specification

ABBY-M0 dataset. The ABBY-CM0 dataset is a `CSV` file which includes $\approx$ 35 million samples with 12 columns representing all extracted features alongside the chip's power consumption while executing related features.

- **Assembly Instructions.** Concerning the three stages pipeline of the target, three different instruction columns exist in the ABBY-CM0 dataset. These columns represent the current executing instruction (I_c), previous instructions that have been executed (I_p), and subsequent instruction, which is in the decoding phase to get executed (I_s).
- **Operand Values.** These columns are related to the data processing by current and previous instructions. `op1_value_current` and `op2_value_current` Represent registers value used for executing I_c while `op1_value_previous` and `op2_value_previous` Represent values of corresponding registers used by previous instruction.
- **Memory transactions.** These features represent the data values to store to or load from the memory processing by `STR` or `LDR` assembly instructions. The `readbus_value_current` and the `readbus_value_previous` are presenting the loaded data on the memory bus while the current or previous load operation processing, respectively. The `writebus_value_current` and the `writebus_value_previous` present the same scenario for the store operation.
- **Power sample.** Shows a proxy value of target power consumption at the moment that all other features are processing.

We must pre-process features to fit any simulator model on the data set. Categorical data, Assembly instructions (I_P, I_C,

and I_S), are hot-encoded and numerical data are represented in a 32-bit binary system (ID_{32}) [3] instead of the decimal system (ID_{10}) to decompress information. Furthermore, we also add HW of operand and memory transaction values with their previous values. Moreover, the HD of each operand value is calculated and located in the dataset. Fig.**??** shows the shape and dimension of the data set before preprocessing.

ABBY-M3 dataset. We have two versions of the CM3 dataset, ABBY-M3V1 and ABBY-M3V2.

ABBY-M3V1. It's a `CSV` file with 4M samples and includes eight columns of data representing all extracted features alongside the target power consumption. All features are the same as ABBY-M0 except, in ABBY-M3V1, there are no memory transaction columns as we only cover the ALU component.

ABBY-M3V2. The only difference between this dataset with the previous version is the power consumption column. In this dataset, instead of considering the maximum value of the power sample in each clock as a proxy for the power consumption of the target, We simply add all ten samples we are acquiring per each clock cycle. In this way, we are not losing any information because of the downsampling, and our dataset represents the transient power consumption of the chip during the execution of an instruction with specific features.

6 Fitting Power Simulators using the ABBY dataset

In this section, we implement different power simulators based on the ABBY dataset and evaluate each model's performance. The leakage model is the heart of a simulator, and it's a bridge that connects the side-channel signal (power in this study) to the processing data (instructions and operands value) in the target. More precise leakage models can help detect more leakages. We analyze conventional linear regression and nonlinear deep learning models in this study.

6.1 Fitting Linear regression models on ABBY-M0

We constructed different leakage models using linear regression to evaluate the ABBY-M0 dataset and investigate microarchitectural leakage. In Equation 8, Y is the estimated power consumption of the target, β_0 is a constant, while β_i is the coefficient of the explanatory variable X_i, and ε is the error. A regression model aims to find the best coefficients for each explanatory variable.

$$Y = \beta_0 + \beta_1 X_1 + \beta_2 X_2 + ... + \beta_n X_n + \varepsilon \quad (8)$$

[3] ID represent the identity model in side-channel

In this study, `Y` would be our leakage model $Y = L(X)$. In this section, we fit different leakage models to investigate how the different microarchitectural features contribute to the leakage.

HW \HD classic leakage model As discussed in Section 4, classical leakage models do not consider instructions but operands. HW and HD are the most popular classical leakage models. We fit a leakage model based on the HW of the previous instruction and the current instruction operand values. Moreover, the HD between each operand's current and previous values was added to the model. Furthermore, we add HW for memory interactions. Equation 9 shows the fitted leakage model.

$$L_{HW}(X) = [HW(OPs)\,|\,HD(OPs)\,|\,HW(MRs)\,|\,HW(MWs)]\beta + \varepsilon \quad (9)$$

Where:
`HW(OPs)`: a matrix that includes the HW of each operand for the previous and current instruction.
`HD(OPs)`: a matrix that includes HD between current and previous instructions for each operand.
`HW(MRs)`: a matrix that includes the HW of the value of the read memory for the current and previous interaction.
`HW(MWs)`: a matrix that includes the HW of the memory write value for the current and previous interaction.

Identity leakage model. The identity model is a very simple classical model that uses the operand values without changes. Fitting a model based on the identity of the operand values requires normalization of the data. We chose a binary representation of the data instead of normalization because dividing the values to a 32-bit size makes the result less sensitive to small changes. Equation 11 shows the identity leakage model. Where ID_2 represents the binary representation of values, all other parameters are the same as the HW \HD model, except that the values are represented as binary instead of HW or HD.

$$L_{ID}(X) = [ID_2(OPs)\,|\,ID_2(MRs)\,|\,ID_2(MWs)]\beta + \varepsilon \quad (10)$$

Instruction-based leakage model. Based on section4, not only the processing data (operational values), but also the execution of operations (instructions) contribute to leakage. Concerning the three-stage pipeline of our target, equation 11 models the leakage related to the instructions L_{INS}`(X)`. To fit the mnemonic assembly instructions, we use a one-hot encoding technique. After one-hot encoding, we end up with a 44-bit value representing our 44 supported instructions in the framework for each instruction level in the pipeline.

$$L_{INS}(X) = [(I_P)\,|\,(I_C)\,|\,(I_S)]\beta + \varepsilon \quad (11)$$

Comprehensive (CH) leakage model. To consider the effect of operations and operand values, we build a leakage model

Table 1: Evaluation of linear regression leakage models

Model	L_{HW} (X)	L_{ID}(X)	L_{INS}(X)	L_{CH} (X)
OHE(I_P,I_C,I_S)			X	X
ID(Ops)		X		X
ID(Mem_Bus)		X		X
HW(Ops)	X			
HW(Mem_Bus)	X			
R^2 adj	0.30	0.32	0.57	0.58
F-Stat	6.92e + 5	9.39e + 5	1.67e + 5	1.04e + 5

based on operand values and instructions. For operand values, we use identity leakage model (L_{ID}(X)) as it shows slightly better performance compared to HW \HD model[4]. Combining the ID with the instruction-based leakage model, we make a more comprehensive leakage model and call it (L_{CH}(X)) that models not only the instruction and corresponding operand values but also the linear interaction between them.

$$L_{CH}(X) = L_{ID}(X) + L_{INS}(X) \tag{12}$$

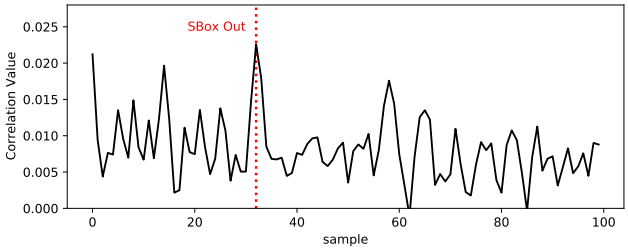
Considering L_{ID}(X) and L_{INS}(X) as a reduced model of the L_{CH}(X) model, we applied F-test. The result shows that the combination of characteristics in L_{CH} (X) shows a significant effect with $\alpha = 0.05$.

Model selection. We fit all the aforementioned leakage models in the ABBY data set, and Table 1 summarizes the result. Each column represents the constructed leakage model and rows are representing features. The X mark shows that the corresponding feature is used in that specific leakage model. For each leakage model the R^2adj and F-Stat are calculated sparately.

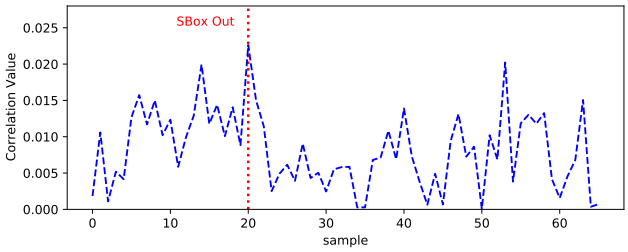
Although the identity leakage model shows slightly better performance, the classical HW\HD is a very strong operand leakage model (compared to the identity model), considering its simplicity. The instruction-based leakage model performs better than the operand leakage model based on the parameter R^2_{adj}. Albeit the CH leakage model performance increased slightly with combining instructions and operand values, this model covers the leakage related to both operands and instructions.

Model evaluation The goal of a side-channel simulator is leakage detection to help the developers during the design phase. To evaluate our model, we applied a correlation-based DPA attack on an AES Crypto algorithm to investigate the performance of leakage detection for our model versus real power trace and ELMO simulated trace while all are running the same firmware. Based on the previous section, we chose the CH model as the best model. For the AES implementation, we choose a first-order protected implementation by Yao et al. [62], which we refer to as *Byte Masked AES*. Although this implementation should be secure against first-order leaks, correlation analysis indicates leaks both on the measured power trace and on the simulated power traces by ELMO and CH models. Fig 6

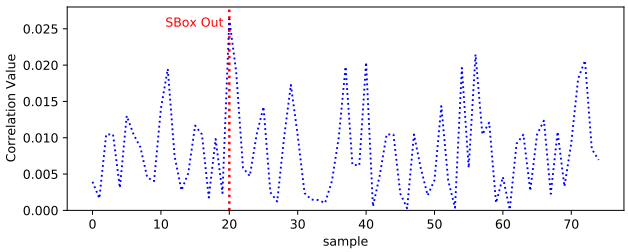
[4]We didn't consider the complexity of the model in this step and performance came first. From a complexity point of view, we have ten coefficients for the HW \HD model, while it is $(8 * 32)$ 256 for the ID. In addition, the identity model might interact better with the instructions



(a) Correlation based on the real measurement



(b) Correlation based on the CH model



(c) Correlation based on the ELMO model

Figure 6: DPA attack based on correlation on the first round Sbox of byte-masked AES shows successful leakage detection on the real power trace vs. the ELMO and the CH model.

6.2 Fitting Deep learning regression models on ABBY-M0

In this section, we apply the deep learning method to our dataset to evaluate nonlinear model performance on the ABBY-CM0 dataset. Based on the problem statement, predicting the chip's power consumption while specific features are processing requires a regression model to estimate the power consumption. We use the popular Multi-Layer Perceptron (MLP) model to train the ABBY-CM0 dataset. We use TensorFlow2 [1] with the Keras submodule to build a preprocessing pipeline and an MLP model with three hidden layers. The input layer consists of three different groups of data, mnemonic assembly instructions, operands value for these instructions, and memory bus values. Before delivering

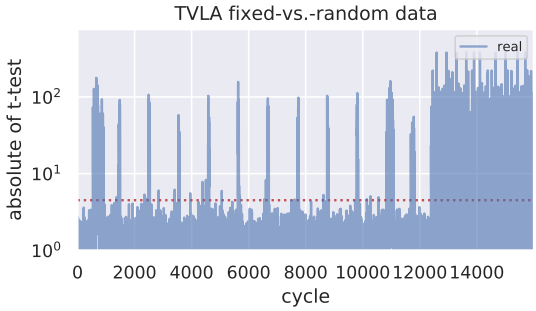
Table 2: MLP hyperparameter description

Layer Types	Details
Input Fully-connected	#neurons 388, Relu
Hidden Fully-connected	#neurons 388, Relu
Hidden Fully-connected	#neurons 16, Relu
Hidden Fully-connected	#neurons 16, Relu
Output	#neurons 1, Linear

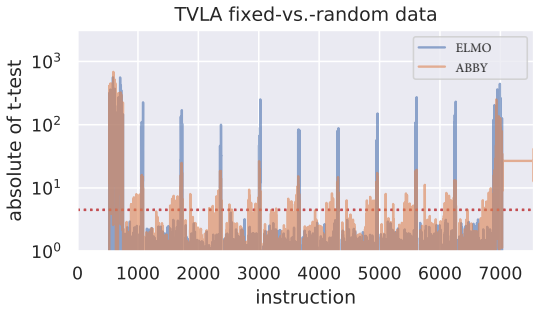
input data to an MLP model, input needs to be normalized to a number between zero and one. For mnemonic assembly instructions which are not ordinal categorical features, we use one-hot encoding [46]. For normalizing operands and memory read/write values, we extended each value to a 32-bits binary array. The input layer consists of $(8 \times 32) + (3 \times 44) = 388$ neurons. Table 2 summarizes the model architecture. The chip power consumption is predicted by the output layer, which consists of one neuron. After 341 training epochs, our MLP model reached $r^2 = 0.771$ on a training set of ≈ 35 Millions samples with a test set of ≈ 15 Millions while 20% of the training set used as validation. Although the r^2 metric evaluates the model's performance, we apply standard side-channel evaluation metrics to determine its usefulness for leakage detection.

model evaluation. TVLA is one of the side-channel domain's most popular leakage assessment methods. See Section 3 for a brief introduction. We applied the TVLA test on two cryptographic algorithms, AES [15] and Xoodoo [16]. We chose the AES implementation the same as we did for the correlation attack [62]. Although this implementation should be secure against first-order leaks, TVLA indicates leaks on the measured power trace as well as on the simulated power traces by ELMO and ABBY models(Figure 7). As proof that ABBY has learned the same leaks as ELMO, we notice how similar the t-trace scores produced by ELMO and ABBY are, with ABBY showing more leakage points compared to ELMO. Notice the difference in the x-axis between the measured traces (cycles) compared to the labels of the simulated traces (instructions).

To confirm that the positive results obtained for Masked AES are not just a lucky coincidence, we compare the t-test results of ELMO and ABBY on a different cryptographic algorithm, namely Xoodoo [16]. Xoodoo is the underlying permutation used in Xoodyak [17], one of the finalists in the NIST Lightweight Cryptography Standardization process. As shown in Figure 8, we confirm that also, in this case, the output of ABBY is comparable to the ELMO model and the measured power traces.

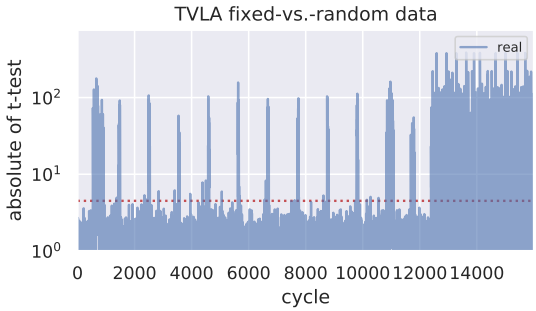


(a) Measured power trace

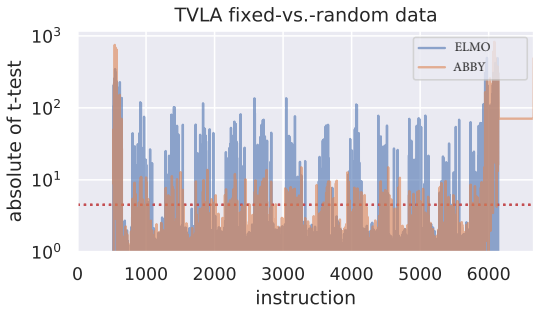


(b) ELMO model vs ABBY model

Figure 7: Byte-Masked-AES TVLA result real vs. simulation



(a) Measured power trace



(b) ELMO model vs ABBY model

Figure 8: Xoodoo TVLA result real vs. simulation

Table 3: Extended MLP model hyperparameter description

Layer Types	Details
Input Fully-connected	#neurons 180, Relu
Hidden Fully-connected	#neurons 32, Relu
Hidden Fully-connected	#neurons 32, Relu
Output	#neurons 1, Linear

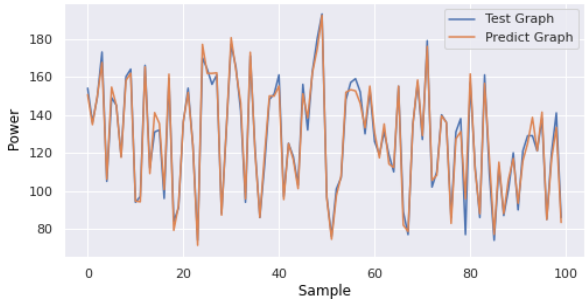


Figure 9: ABBY estimated power trace Vs. Measured trace

6.3 Fitting deep learning regression models on ABBY-M3

For model fitting, we use an MLP regression model like the previous model for CM0. The differences with the previous model are related to the input layer and model architecture. We have 17 mnemonic assembly instructions, ALU related, for the input layer.

After normalization of mnemonic assembly instructions by using one-hot encoding and using a 32-bit binary array extension for operand values, The input layer consists of $(4 \times 32) + (3 \times 17) + 1 = 180$ neurons. Table 3 summarizes the model architecture. Prediction of the ALU power consumption is done by the output layer, which consists of one neuron.

After 341 training epochs, our MLP model reach $r^2 = 0.977$ on a training set of ≈ 21 Millions samples with a test set of ≈ 14 Millions while 20% of the training set used as validation. Although the r^2 metric evaluates the model's performance, we apply standard side-channel evaluation metrics to determine its usefulness for leakage detection.

Model evaluation. Although the visual inspection of the estimated power shows a good similarity with the measured power Fig. 9, we need a metric to compare them.

One of the popular side-channel attacks is the Differential Power Analysis attack. This attack is one of the strong attacks that can reveal secret information due to the microarchitectural leakages of a chip. We apply correlation analysis for a random assembly firmware designed to target ALU instructions on the measured power trace as same as the estimated

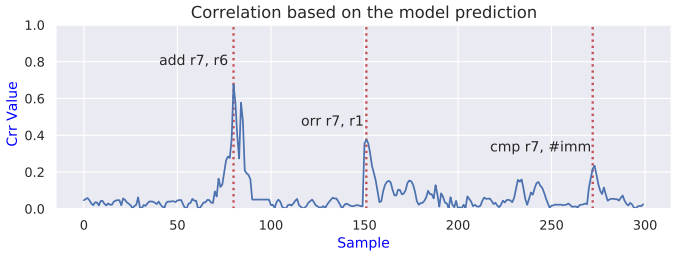


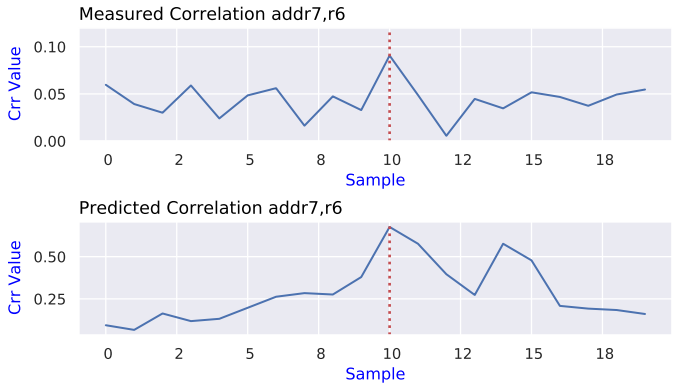
Figure 10: Correlation result based on the ABBY model prediction

power trace, which is produced by ABBY-CM3 deep learning model. We feed the designed firmware with 10K different values and use these values as an operand using the register `r7` for different instructions.Fig 10 shows the correlation for the ABBY model. Fig 11 compares the correlation peaks found by the ABBY model with the correlation peaks found in the measured traces. With this experiment, we could confirm that the model correlation result is similar to the measured power traces from the real target. Although the figure shows the result for some ALU instructions, for all covered ALU instructions, we observed that maximum correlation was founded, and it was happening in the same position that is happening for the real measurement.

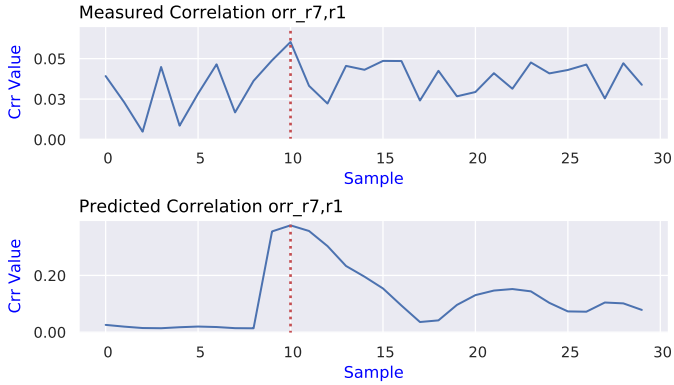
7 Conclusions and Future work

We propose ABBY, the first framework to automate the profiling of the architectural layer. As a result, ABBY significantly reduces the human effort necessary to create leakage models. ABBY is scalable and can be transferred to different architectures. The most challenging aspect of porting ABBY to different architectures is the creation of the labeled dataset. We used standard tools for developing the ABBY framework. Using the ABBY-CM0 dataset, we explored several leakage models ranging from primitive to transition-based, which include pipeline effects and instruction interaction via hidden registers. We evaluate and compare the performance of these leakage models using statistics metrics such as R^2 and $F-test$ and side-channel attacks. When considering side-channel attacks, we see those transition-based leakage models such as ELMO and our CH model are superior to primitive ones. When comparing the performance of ELMO with our deep learning leakage model, the results are very close, demonstrating the quality of the ABBY-CM0 dataset and ultimately the effectiveness of the ABBY framework. We constructed the ABBY-CM3 dataset to investigate the scalability of the ABBY framework while we developed a side-channel power simulator for the ALU component based on this dataset. Despite statistical metrics, correlation results for simulation are close to the measured trace.

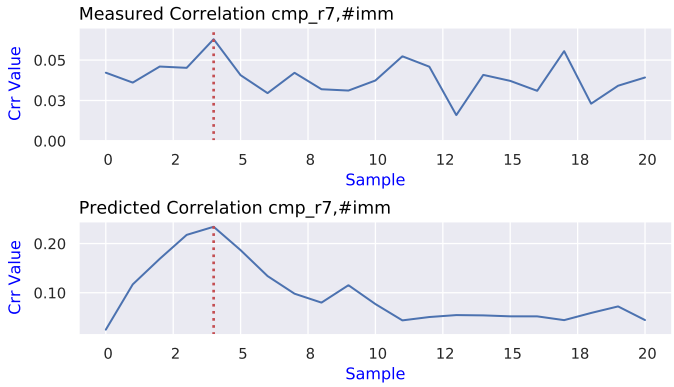
In future work, our objective is to develop a simulator for



(a) Correlation result for `add` instruction estimation Vs measured



(b) Correlation result for `orr` instruction estimation Vs measured



(c) Correlation result for `cmp` instruction estimation Vs measured

Figure 11: Correlation result of 1K estimated traces Vs. measured traces on ALU instructions

Cortex-M3 based on the ABBY framework that covers all the components besides the ALU. Moreover, we will investigate how targeted microarchitectural benchmarks such as [36] and optimizations of the model architecture can further enhance the performance of ABBY.

8 Availability

Our framework and dataset will be published upon the paper's acceptance and can be downloaded from GitHub: ABBY-Framework

References

[1] Martín Abadi, Ashish Agarwal, Paul Barham, Eugene Brevdo, Zhifeng Chen, Craig Citro, Greg S. Corrado, Andy Davis, Jeffrey Dean, Matthieu Devin, Sanjay Ghemawat, Ian Goodfellow, Andrew Harp, Geoffrey Irving, Michael Isard, Yangqing Jia, Rafal Jozefowicz, Lukasz Kaiser, Manjunath Kudlur, Josh Levenberg, Dandelion Mané, Rajat Monga, Sherry Moore, Derek Murray, Chris Olah, Mike Schuster, Jonathon Shlens, Benoit Steiner, Ilya Sutskever, Kunal Talwar, Paul Tucker, Vincent Vanhoucke, Vijay Vasudevan, Fernanda Viégas, Oriol Vinyals, Pete Warden, Martin Wattenberg, Martin Wicke, Yuan Yu, and Xiaoqiang Zheng. TensorFlow: Large-scale machine learning on heterogeneous systems, 2015. Software available from tensorflow.org.

[2] Dakshi Agrawal, Bruce Archambeault, Josyula R. Rao, and Pankaj Rohatgi. The EM side—channel(s). In Burton S. Kaliski, çetin K. Koç, and Christof Paar, editors, *Cryptographic Hardware and Embedded Systems - CHES 2002*, pages 29–45, Berlin, Heidelberg, 2003. Springer Berlin Heidelberg.

[3] Jude Angelo Ambrose, Naeill Aldon, Aleksandar Ignjatovic, and Sri Parameswaran. Anatomy of differential power analysis for AES. In *2008 10th International Symposium on Symbolic and Numeric Algorithms for Scientific Computing*, pages 459–466. IEEE, 2008.

[4] ARM. *Cortex™-M3 Revision r2p0 Technical Reference Manual*, 03 2010.

[5] ARM. Arm® v6-m architecture reference manual. https://developer.arm.com/documentation/ddi0419/e/?lang=en, 02 2018.

[6] ARM. ARM®v7-m architecture reference manual. https://developer.arm.com/documentation/ddi0403/ee/?lang=en, 02 2021.

[7] Vipul Arora, Ileana Buhan, Guilherme Perin, and Stjepan Picek. A tale of two boards: On the influence

of microarchitecture on side-channel leakage. In Vincent Grosso and Thomas Pöppelmann, editors, *Smart Card Research and Advanced Applications - 20th International Conference, CARDIS 2021, Lübeck, Germany, November 11-12, 2021, Revised Selected Papers*, volume 13173 of *Lecture Notes in Computer Science*, pages 80–96. Springer, 2021.

[8] Josep Balasch, Benedikt Gierlichs, Roel Verdult, Lejla Batina, and Ingrid Verbauwhede. Power analysis of Atmel CryptoMemory - recovering keys from secure EEPROMs, booktitle = Topics in Cryptology - CT-RSA 2012, The Cryptographers' Track at the RSA Conference, editor = O. Dunkelman, series = Lecture Notes in Computer Science, volume = 7178, publisher = Springer-Verlag, year = 2012, pages = 9–34,.

[9] Gilles Barthe, Marc Gourjon, Benjamin Grégoire, Maximilian Orlt, Clara Paglialonga, and Lars Porth. Masking in fine-grained leakage models: Construction, implementation and verification. *IACR Trans. Cryptogr. Hardw. Embed. Syst.*, 2021(2):189–228, 2021.

[10] Guido Bertoni, Vittorio Zaccaria, Luca Breveglieri, Matteo Monchiero, and Gianluca Palermo. AES power attack based on induced cache miss and countermeasure. In *International Conference on Information Technology: Coding and Computing (ITCC'05)-Volume II*, volume 1, pages 586–591. IEEE, 2005.

[11] Olivier Bronchain, Julien M. Hendrickx, Clément Massart, Alex Olshevsky, and François-Xavier Standaert. Leakage certification revisited: Bounding model errors in side-channel security evaluations. In Alexandra Boldyreva and Daniele Micciancio, editors, *Advances in Cryptology – CRYPTO 2019*, pages 713–737, Cham, 2019. Springer International Publishing.

[12] Ileana Buhan, Lejla Batina, Yuval Yarom, and Patrick Schaumont. SoK: Design tools for side-channel-aware implementations. In Yuji Suga, Kouichi Sakurai, Xuhua Ding, and Kazue Sako, editors, *ASIA CCS '22: ACM Asia Conference on Computer and Communications Security, Nagasaki, Japan, 30 May 2022 - 3 June 2022*, pages 756–770. ACM, 2022.

[13] Elad Carmon, Jean-Pierre Seifert, and Avishai Wool. Photonic side channel attacks against RSA. In *2017 IEEE International Symposium on Hardware Oriented Security and Trust (HOST)*, pages 74–78. IEEE, 2017.

[14] Suresh Chari, Josyula R. Rao, and Pankaj Rohatgi. Template attacks. In Burton S. Kaliski Jr., Çetin Kaya Koç, and Christof Paar, editors, *Cryptographic Hardware and Embedded Systems - CHES 2002, 4th International Workshop, Redwood Shores, CA, USA, August 13-15, 2002, Revised Papers*, volume 2523 of *Lecture Notes in Computer Science*, pages 13–28. Springer, 2002.

[15] J. Daemen and Vincent Rijmen. AES the advanced encryption standard. *The Design of Rijndael*, 26, 01 2002.

[16] Joan Daemen, Seth Hoffert, Gilles Van Assche, and Ronny Van Keer. The design of Xoodoo and Xoofff. *IACR Trans. Symmetric Cryptol.*, 2018:1–38, 2018.

[17] Joan Daemen, Seth Hoffert, Michaël Peeters, Gilles Van Assche, and Ronny Van Keer. Xoodyak, a lightweight cryptographic scheme. *IACR Trans. Symmetric Cryptol.*, 2020(S1):60–87, 2020.

[18] Arnaud de Grandmaison, Karine Heydemann, and Quentin L. Meunier. ARMISTICE: microarchitectural leakage modeling for masked software formal verification. *IEEE Trans. Comput. Aided Des. Integr. Circuits Syst.*, 41(11):3733–3744, 2022.

[19] Nicolas Debande, Maël Berthier, Yves Bocktaels, and Thanh-Ha Le. Profiled model based power simulator for side channel evaluation. Cryptology ePrint Archive, Report 2012/703, 2012. https://eprint.iacr.org/2012/703.

[20] Jerry den Hartog, Jan Verschuren, Erik P. de Vink, Jaap de Vos, and W. Wiersma. PINPAS: a tool for power analysis of smartcards. In *SEC*, pages 453–457, 2003.

[21] Margaux Dugardin, Louiza Papachristodoulou, Zakaria Najm, Lejla Batina, Jean-Luc Danger, and Sylvain Guilley. Dismantling real-world ECC with horizontal and vertical template attacks. In François-Xavier Standaert and Elisabeth Oswald, editors, *Constructive Side-Channel Analysis and Secure Design - 7th International Workshop, COSADE 2016, Graz, Austria, April 14-15, 2016, Revised Selected Papers*, volume 9689 of *Lecture Notes in Computer Science*, pages 88–108. Springer, 2016.

[22] Thomas Eisenbarth, Timo Kasper, Amir Moradi, Christof Paar, Mahmoud Salmasizadeh, and Mohammad T. Manzuri Shalmani. On the power of power analysis in the real world: A complete break of the KeeLoq code hopping scheme. In David Wagner, editor, *Advances in Cryptology – CRYPTO 2008*, volume 5157 of *Lecture Notes in Computer Science*, pages 203–220. Springer-Verlag Berlin Heidelberg, 2008.

[23] Georges Gagnerot. *Étude des attaques et des contremesures associées sur composants embarqués*. PhD thesis, Université de Limoges, 2013.

[24] Si Gao and Elisabeth Oswald. A novel completeness test for leakage models and its application to side channel attacks and responsibly engineered simulators. In Orr Dunkelman and Stefan Dziembowski, editors, *Advances in Cryptology – EUROCRYPT 2022*, pages 254–283, Cham, 2022. Springer International Publishing.

[25] Si Gao, Elisabeth Oswald, and Dan Page. Reverse engineering the micro-architectural leakage features of a commercial processor. *IACR Cryptol. ePrint Arch.*, page 794, 2021.

[26] Aymeric Genêt, Natacha Linard de Guertechin, and Novak Kaludjerovi'c. Full key recovery side-channel attack against ephemeral SIKE on the Cortex-M4. In Shivam Bhasin and Fabrizio De Santis, editors, *Constructive Side-Channel Analysis and Secure Design*, pages 228–254, Cham, 2021. Springer International Publishing.

[27] Barbara Gigerl, Vedad Hadzic, Robert Primas, Stefan Mangard, and Roderick Bloem. Coco: Co-design and co-verification of masked software implementations on CPUs. Cryptology ePrint Archive, Report 2020/1294, 2020. https://eprint.iacr.org/2020/1294.

[28] G. Goodwill and J.J.B. Jun andP.Rohatgi. A testing methodology for side channel resistance validation. *NIST non-invasive attack testing workshop*, 2018.

[29] Miao Tony He, Jungmin Park, Adib Nahiyan, Apostol Vassilev, Yier Jin, and Mark Mohammad Tehranipoor. RTL-PSC: automated power side-channel leakage assessment at register-transfer level. In *VTS*, pages 1–6, 2019.

[30] Benjamin Hettwer, Stefan Gehrer, and Tim Güneysu. Deep neural network attribution methods for leakage analysis and symmetric key recovery. In Kenneth G. Paterson and Douglas Stebila, editors, *Selected Areas in Cryptography – SAC 2019*, pages 645–666, Cham, 2020. Springer International Publishing.

[31] Sorin A. Huss, Marc Stöttinger, and Michael Zohner. *AMASIVE: An Adaptable and Modular Autonomous Side-Channel Vulnerability Evaluation Framework*, volume 8260 of *Lecture Notes in Computer Science*, pages 151–165. Springer, 2013.

[32] Yuval Ishai, Amit Sahai, and David A. Wagner. Private circuits: Securing hardware against probing attacks. In Dan Boneh, editor, *Advances in Cryptology - CRYPTO 2003, 23rd Annual International Cryptology Conference, Santa Barbara, California, USA, August 17-21, 2003, Proceedings*, volume 2729 of *Lecture Notes in Computer Science*, pages 463–481. Springer, 2003.

[33] Paul C. Kocher. Timing attacks on implementations of Diffie-Hellman, RSA, DSS, and other systems. In Neal Koblitz, editor, *Advances in Cryptology — CRYPTO '96*, pages 104–113, Berlin, Heidelberg, 1996. Springer Berlin Heidelberg.

[34] Yann Le Corre, Johann Großschädl, and Daniel Dinu. Micro-architectural power simulator for leakage assessment of cryptographic software on ARM Cortex-M3 processors. In *COSADE*, pages 82–98, 2018.

[35] Moritz Lipp, Andreas Kogler, David Oswald, Michael Schwarz, Catherine Easdon, Claudio Canella, and Daniel Gruss. PLATYPUS: Software-based power side-channel attacks on x86. In *2021 IEEE Symposium on Security and Privacy (SP)*, pages 355–371. IEEE, 2021.

[36] Ben Marshall, Dan Page, and James Webb. MIRACLE: Micro-architectural leakage evaluation. Cryptology ePrint Archive, Report 2021/261, 2021. https://ia.cr/2021/261.

[37] David McCann, Elisabeth Oswald, and Carolyn Whitnall. Towards practical tools for side channel aware software engineering: 'grey box' modelling for instruction leakages. In *USENIX Security Symposium*, pages 199–216, 2017.

[38] ST Microelectronic. *Discovery kit with STM32F030R8 MCU*.

[39] ST Microelectronic. *Discovery kit with STM32F051R8 MCU*.

[40] ST Microelectronic. *STM32 Mainstream MCUs*.

[41] Amir Moradi, Alessandro Barenghi, Timo Kasper, and Christof Paar. On the vulnerability of FPGA bitstream encryption against power analysis attacks: Extracting keys from xilinx virtex-ii fpgas. In *Proceedings of the 18th ACM conference on Computer and communications security*, pages 111–124, 2011.

[42] Adib Nahiyan, Jungmin Park, Miao Tony He, Yousef Iskander, Farimah Farahmandi, Domenic Forte, and Mark Mohammad Tehranipoor. SCRIPT: a CAD framework for power side-channel vulnerability assessment using information flow tracking and pattern generation. *ACM Trans. Design Autom. Electr. Syst.*, 25(3):26:1–26:27, 2020.

[43] Kostas Papagiannopoulos and Nikita Veshchikov. Mind the gap: Towards secure 1st-order masking in software. In *COSADE*, volume 10348 of *Lecture Notes in Computer Science*, pages 282–297. Springer, 2017.

[44] Riscure. Inspector-sca. https://www.riscure.com/security-tools/inspector-sca, 2002. [Online; accessed 7-Apr-2021].

[45] Matthieu Rivain and Emmanuel Prouff. Provably secure higher-order masking of AES. In Stefan Mangard and François-Xavier Standaert, editors, *Cryptographic Hardware and Embedded Systems, CHES 2010, 12th International Workshop, Santa Barbara, CA, USA, August 17-20, 2010. Proceedings*, volume 6225 of *Lecture Notes in Computer Science*, pages 413–427. Springer, 2010.

[46] Pau Rodríguez, Miguel A. Bautista, Jordi Gonzàlez, and Sergio Escalera. Beyond one-hot encoding: Lower dimensional target embedding. *Image and Vision Computing*, 75:21–31, 2018.

[47] Asanka Sayakkara, Nhien-An Le-Khac, and Mark Scanlon. A survey of electromagnetic side-channel attacks and discussion on their case-progressing potential for digital forensics. *Digital Investigation*, 29:43–54, 2019.

[48] Nader Sehatbakhsh, Baki Berkay Yilmaz, Alenka G. Zajic, and Milos Prvulovic. EMSim: A microarchitecture-level simulation tool for modeling electromagnetic side-channel signals. In *HPCA*, pages 71–85, 2020.

[49] Madura A Shelton, Łukasz Chmielewski, Niels Samwel, Markus Wagner, Lejla Batina, and Yuval Yarom. Rosita++: Automatic higher-order leakage elimination from cryptographic code. In *Proceedings of the 2021 ACM SIGSAC Conference on Computer and Communications Security*, pages 685–699, 2021.

[50] Madura A. Shelton, Niels Samwel, Lejla Batina, Francesco Regazzoni, Markus Wagner, and Yuval Yarom. Rosita: Towards automatic elimination of power-analysis leakage in ciphers. In *NDSS*, 2021.

[51] Danilo Sijacic, Josep Balasch, Bohan Yang, Santosh Ghosh, and Ingrid Verbauwhede. Towards efficient and automated side-channel evaluations at design time. *J. Cryptogr. Eng.*, 10(4):305–319, 2020.

[52] Patanjali SLPSK, Prasanna Karthik Vairam, Chester Rebeiro, and V. Kamakoti. Karna: A gate-sizing based security aware EDA flow for improved power side-channel attack protection. In *ICCAD*, pages 1–8, 2019.

[53] QEMU team. *QUICK EMULATOR tool*. Available at https://www.qemu.org/, version.

[54] C. Thuillet, P. Andouard, and O. Ly. A smart card power analysis simulator. In *2009 International Conference on Computational Science and Engineering*, volume 2, pages 847–852, 2009.

[55] Vivek Tiwari, Sharad Malik, Andrew Wolfe, and Mike Tien-Chien Lee. Instruction level power analysis and optimization of software. In *VLSI Design*, pages 326–328, 1996.

[56] Nikita Veshchikov. SILK: high level of abstraction leakage simulator for side channel analysis. In *PPREW@ACSAC*, pages 3:1–3:11, 2014.

[57] Nikita Veshchikov and Sylvain Guilley. Use of simulators for side-channel analysis. In *EuroS&P Workshops*, pages 104–112, 2017.

[58] David Welch. *Thumbulator-Tool*. Available at https://github.com/dwelch67/thumbulator.

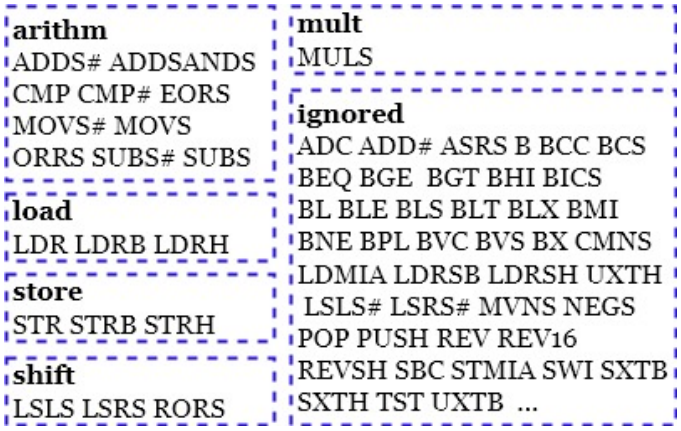
[59] Carolyn Whitnall and Elisabeth Oswald. A critical analysis of ISO 17825 ('testing methods for the mitigation of non-invasive attack classes against cryptographic modules'). In *ASIACRYPT (3)*, pages 256–284, 2019.

[60] Yuan Yao, Tarun Kathuria, Baris Ege, and Patrick Schaumont. Architecture correlation analysis (ACA): identifying the source of side-channel leakage at gate-level. In *HOST*, pages 188–196, 2020.

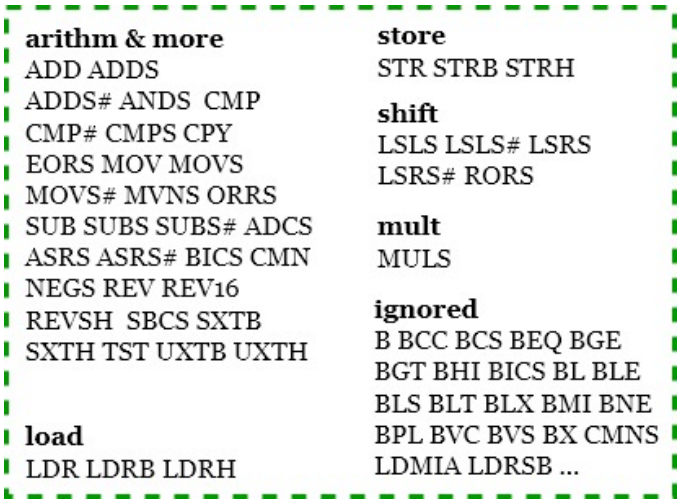
[61] Yuan Yao, Patrick Schaumont, Jasper Van Woudenberg, Cees-Bart Breunesse, Edgar Mateos Santillan, and Steve Stecyk. Verification of power-based side-channel leakage through simulation. In *MWSCAS*, pages 1112–1115, 2020.

[62] Yuan Yao, Mo Yang, Conor Patrick, Bilgiday Yuce, and Patrick Schaumont. Fault-assisted side-channel analysis of masked implementations. In *2018 IEEE International Symposium on Hardware Oriented Security and Trust (HOST)*, pages 57–64, 2018.

9 Appendix A



(a) ELMO instruction clusters



(b) ABBY instructions

Figure 12: ELMO instruction clusters vs ABBY instructions

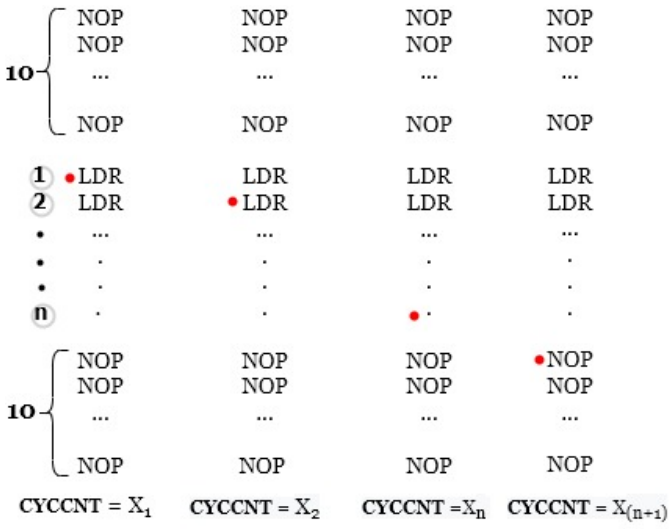


Figure 13: Breakpoint effect solution