# ABBY: Automating the creation of transition-based leakage models

No Author Given

No Institute Given

**Abstract.** We introduce ABBY, an open source side-channel leakage profiling framework that targets the microarchitecture layer. Existing solutions to characterize the microarchitecture layer of a given target require extensive manual effort. The main innovation of ABBY is the training framework, which can automatically characterize the microarchitecture of a target device. To benchmark ABBY, we target an ARM CORTEX-MO board, for which tools that profile the microarchitecture already exist. Using ABBY, we create the ABBY-CM0 dataset, which covers 80% of the entire ARMv6 instruction set. ABBY-CM0 will be the first open source dataset with detailed information about the microarchitecture layer and can be used to create any number of transition-based leakage models. The main application for such leakage models is the creation of leakage simulators. A preliminary evaluation of a leakage model produced with the ABBY-CM0 dataset of real-world crypto implementations shows performance comparable to state-of-the-art leakage simulators.

**Keywords:** side-channels, leakage simulator, linear regression, microarchitecture

## 1 Introduction

Kocher et al. [14] showed that when code runs on a chip, it interacts with the hardware environment. This interaction appears in physical side-channel(s) such as power [3, 2, 15], electromagnetic emanation(EM) [1, 20], photonic emission [6], etc. An adversary can take advantage of these side channels and learn about secret information during processing. Many studies show successful side channel-based attacks leading to key or secret share recovery on various platforms [12, 13].

Technological advancements and twenty years of sustained effort by the cryptographic community significantly raised the workload required for successful key extraction. However, the problem of implementing a secure cryptographic algorithm on a given target device is not solved. Piling up countermeasures is not a solution, as countermeasures come at the cost of resources. 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 if the changes are beneficial or, in contrast, if 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 [5]. 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. 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 correct and informative, leakage simulators assist in the design of secure cryptographic implementation. An additional benefit is to make side-channel analysis accessible to non-experts.

Side-channel leakage simulators use *leakage models* to capture the physical implementation of the target device To be useful, a leakage model must be "correct" to accurately reflect reality and be informative to be useful for key recovery. As in [18], we distinguish between *value* and *transition*-based leakage models. The leakage model is value-based if it takes the intermediate values of a cryptographic algorithm as arguments. Typical examples include the popular 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 [18] such as the Hamming-distance (HD) model.

Creating a transition-based leakage model requires intensive manual effort and is specific to the target. The reason is that capturing the interaction of intermediate values requires profiling the microarchitecture layer, which may contain hidden storage elements where unexpected interactions between instructions can occur. Although the importance of fine-grained leakage models has been established [16, 22, 11], the microarchitecture implementation is considered a trade secret and, therefore, is not public information. By automating the creation of various transition-based leakage models, more side channel simulators, such as ELMO [17] can become available and cover different architectures. Automated microarchitecture profiling is necessary to have different simulators. This development can open the way for rule-driven code rewriting engines [1] such as ROSITA [22] and ROSITA++ [21] 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[2].

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

[1] Side-channel code-rewrite engines use leakage simulators to mitigate leakage

[2] For Example, ELMO as a side-channel simulator developed its microarchitectural profiler based on Thumbulator [26](an Instruction Set Simulator). ROSITA developed a code rewrite engine based on the ELMO* (upgraded ELMO) simulator combined with an assembly code modifier

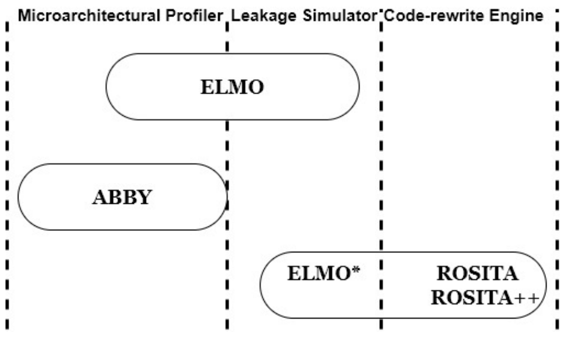


Fig. 1: Overview of ABBY, ELMO, and ROSITA in application level design

the microarchitecture 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 the creation of transition-based leakage models without reverse engineering the micro-architecture implementation?*
**Contribution**. In this work, we answer the question in an affirmative sense. 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:** Section 2 briefly introduces the background on side channel attacks and leakage detection and describes the hardware setup we used. Related works on microarchitectural leakage simulators are mentioned in Section 3. Section 4, discusses building transition-based leakage models. In Section 5, we build several leakage simulators using the ABBY-CM0 data set and discuss their performance. Section 7 concludes the article.
**Implementation code:** The implementation of the ABBY and ABBY-CM0 dataset will be available upon acceptance of this article.

## 2 Background

### 2.1 Side channel attacks

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.

## 2.2 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 $L(X)$ [10].

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

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.

## 2.3 Model evaluation

The goal of most statistical models is to predict future events or to help explain reality[4]. In the former case, the quality of the model is defined by its predictive power, while the quality of the model of 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 are possible to fit the leakage model, in this paper we only consider linear regression. 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 [10]. For readability, we use the notation [10], [17].
$R^2$ 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 value and the measured value. 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 and represents the sum of the squared differences between the actual measurement $y_i$ and the predicted value $\widetilde{L}(Z_i)$(equation 2).

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

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

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

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

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

The coefficient of determination $R^2$ can be calculated with equation 5 [10].

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

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 [17]. 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.

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

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.

The F-test leads to a model that contains only significant parameters. To simplify a model further, we consider information-theoretic measures such as the Akaike Information Criterion (AIC) and the Bayesian Information Criterion (BIC). AIC scores a model based on the log-likelihood[3] (`LL`), and a low score means higher performance [7]. To penalize the complexity of the model with the AIC metric, we use BIC. This metric can be calculated using log-likelihood, the number of parameters, and the sample size (Equation 8).

$$\begin{aligned} \texttt{AIC} &= (-2\,/\,n * LL) + (2 * p\,/\,n) \\ \texttt{BIC} &= (-2 * LL) + (p * log(n)) \end{aligned} \tag{8}$$

## 2.4 Instruction Emulators

To build a leakage simulator, we need an emulator that outputs an instruction trace from the compiled machine code. Most emulators are only instruction-accurate and not cycle-accurate, i.e., ignoring that one instruction may take more than one clock cycle. It is possible to achieve cycle accuracy in emulation when a detailed hardware description of all the peripherals is available. Verilator can convert Verilog hardware descriptions to cycle-accurate behavioral models. For ARM, these cycle-accurate models (Arm Fixed Virtual Platforms) are closed source and do not typically describe other peripherals that might affect instruction execution speed. ELMO is instruction accurate and uses Thumbulator to emulate ARM Thumb-1. Due to the discrepancy between instruction vs. cycle accuracy, the measured side-channel traces might not align with the instruction trace.

## 2.5 Hardware Setup

We use two different ARM Cortex-M0 processors based on Armv6-M architecture manufactured by ST Microelectronics with STM Discovery Boards. The target boards have the `STM32F051R8T6` or `STM32F030R8T6` chip 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

[3] Log-likelihood measures the goodness of the fit for a model. The higher the value of the log-likelihood, the better a model fits a dataset.

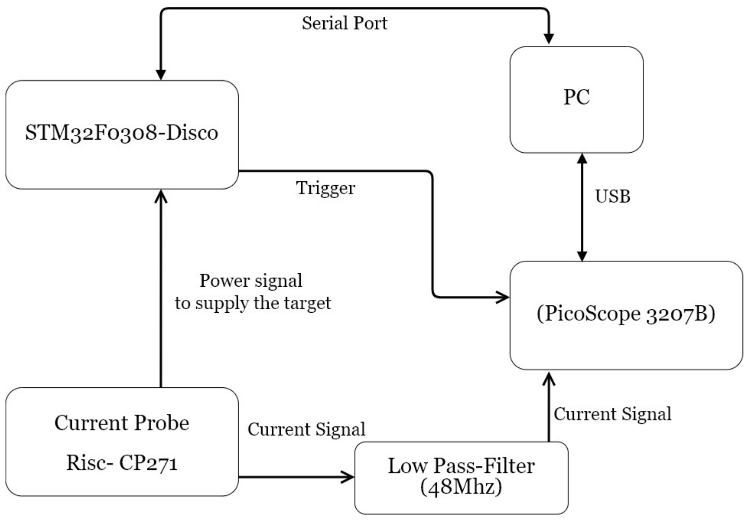


Fig. 2: Setup Block Diagram

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.

## 3 Related works

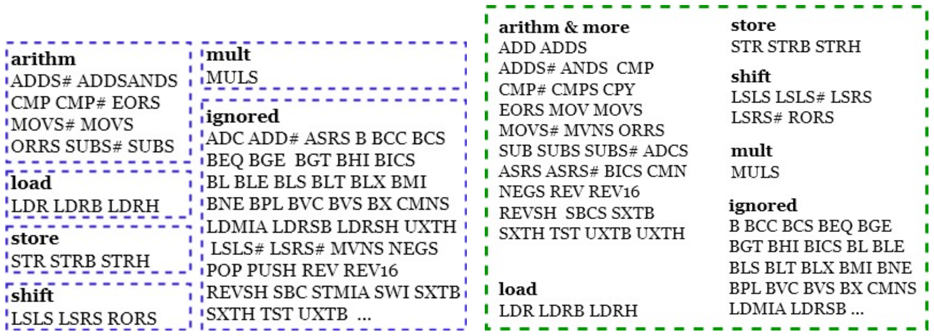
SILK [25], is the first open-source side-channel leakage simulator, which inputs the source code of the cryptographic algorithm and some user-defined parameters (leakage function, number of leakage points, etc.) and generates power traces. It captures no specific hardware architecture and targets data-dependent power consumption. It is suitable for the early stages of cryptographic algorithm design. Instruction-level simulators (ISA) look at the machine code compiled for a certain architecture to predict side-channel leakage. ISA simulators use machine code that will be executed by a specific CPU, but does not require information about the processor or the process technology. SAVRASCA [24] 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 [18] checks violations of the independent leakage assumption (ILA) for the AVR architecture. 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 microarchitecture effects of the target. These simulators capture a more descriptive target model at the cost of a larger engineering effort. MAPS [8] 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 details of the microarchitecture, 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 [17] to be the first genuinely fine-grained 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* [22] improves the leakage model of ELMO by capturing interactions that span multiple cycles. ROSITA [22] 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 [17], 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 microarchitecture details in a security analysis has been established [11], [16] access to its implementation is typically not available. The authors of ELMO had to reverse engineer the microarchitecture implementation of the target ARM Cortex M0 processor. The current state of the art allows reverse engineering a commercial ARM Cortex-M3 microprocessor [11]. 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.

## 4 Automated Microarchitectural Profiling

### 4.1 Capturing instructions interaction

The critical observation made by McCann et al. [17] 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* $I_s$ [23]. 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 [17].
**Instruction coverage of classical leakages model.** Although executing instructions is one of the main contributors to the power consumption of the chip; 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.



**arithm**
ADDS# ADDSANDS
CMP CMP# EORS
MOVS# MOVS
ORRS SUBS# SUBS

**load**
LDR LDRB LDRH

**store**
STR STRB STRH

**shift**
LSLS LSRS RORS

**mult**
MULS

**ignored**
ADC ADD# ASRS B BCC BCS
BEQ BGE BGT BHI BICS
BL BLE BLS BLT BLX BMI
BNE BPL BVC BVS BX CMNS
LDMIA LDRSB LDRSH UXTH
LSLS# LSRS# MVNS NEGS
POP PUSH REV REV16
REVSH SBC STMIA SWI SXTB
SXTH TST UXTB ...

(a) ELMO instruction clusters

**arithm & more**
ADD ADDS
ADDS# ANDS CMP
CMP# CMPS CPY
EORS MOV MOVS
MOVS# MVNS ORRS
SUB SUBS SUBS# ADCS
ASRS ASRS# BICS CMN
NEGS REV REV16
REVSH SBCS SXTB
SXTH TST UXTB UXTH

**load**
LDR LDRB LDRH

**store**
STR STRB STRH

**shift**
LSLS LSLS# LSRS
LSRS# RORS

**mult**
MULS

**ignored**
B BCC BCS BEQ BGE
BGT BHI BICS BL BLE
BLS BLT BLX BMI BNE
BPL BVC BVS BX CMNS
LDMIA LDRSB ...

(b) ABBY instructions

Fig. 3: ELMO instruction clusters vs ABBY instructions

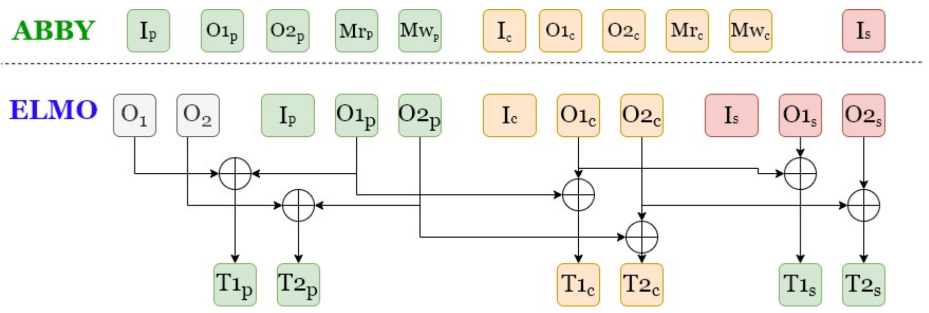


Fig. 4: ABBY vs ELMO features to profile power consumption of the chip

**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 “similarinstructions ” (that is, that leak information in the same way), the authors identify five groups altogether including 21 instructions, see Figure 3.

The groups correspond to the same processor component: ALU instructions in one group, shift instructions are 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 single cycle implementation fit in a separate group. These groups also represent the internal structure that we could expect from an ARM core, as shift, multiplication, and arithmetic operations do not use the same part of the CPU.

Figure 4 (bottom) is a visual representation of the interaction between the different instructions in the ELMO model. Transitions for instruction operands across the data bus are captured in the form of a 32-bit matrix (represented as $T_p^1$, $T_p^2$ for the previous instruction, $T_c^1$,$T_c^2$ for the current instruction, and $T_s^1$, $T_s^2$) for the subsequent instruction).

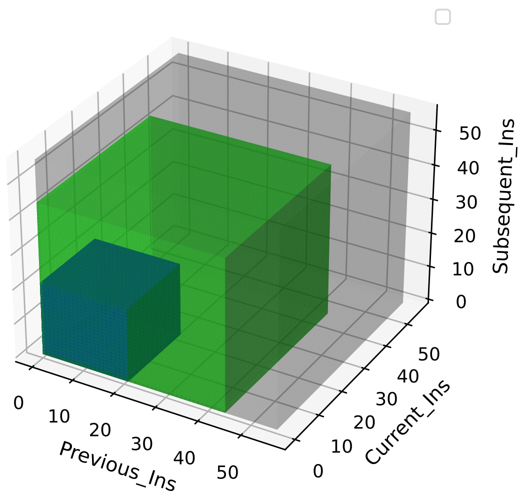


Fig. 5: Possible instruction space in gray vs ABBY and ELMO in green and blue respectively

**ABBY instruction coverage** Similar to ELMO, ABBY captures the interaction between instructions in the pipeline registers, see Figure 4(top) and memory leakage. No assumptions about the operand interaction are made to simplify the collection of training data. 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 the current and previous memory access to cover memory leaks, as these were shown to be important in [22].

As part of the effort to simplify training, we further remove the clustering of instructions. We keep the necessary 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. 3) as these operations are not used for implementing cryptographic algorithms. For store and load operations, we reserved 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 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.

### 4.2 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.

### 4.3 Trace recording.

Using the generated firmware, we collect 1,000 triplets [4] (or pipeline states) in a single acquisition. By generating and flashing[5] 50 000 different Random assembly firmware automatically, we collect for each triplet of instructions (with random operands) $\frac{50\,000 \times 1\,000}{44^3} \approx 587$ data points.
Firmware with random instructions is loaded on the target device and power consumption is measured while the target executes the firmware.

### 4.4 Trace annotation.

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 to align 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 6). DTW is not perfect, and alignment errors are possible. As a result of the first attempt to align the data, 22% of instructions are dropped. We also expect errors on 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).

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

[5] 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.

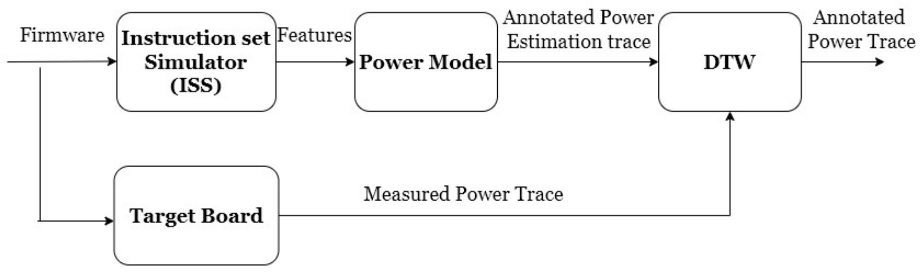


Fig. 6: Dataset generation process of ABBY

### 4.5 The ABBY-M0 dataset.

In its original form, the ABBY-CM0 dataset is a file in the format `CSV`, which includes ≈ 35 million samples with 11 columns representing all extracted features (Fig. 4). We need to 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 ($ID_{32}$) [6] 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.7 shows the shape and dimension of the data set before preprocessing.

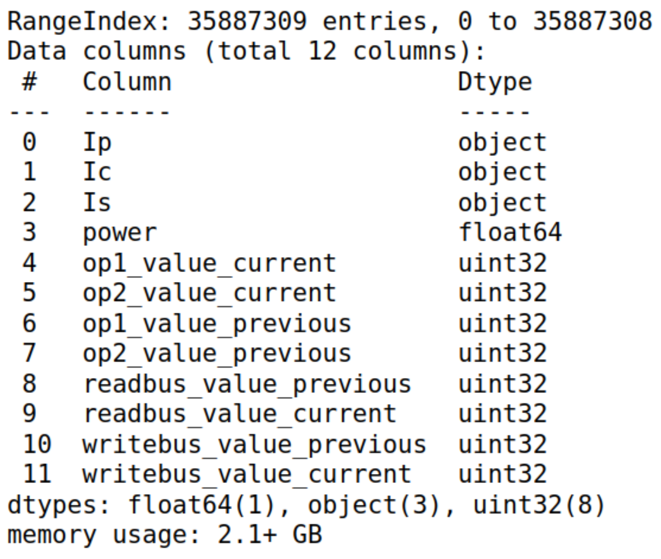

```
RangeIndex: 35887309 entries, 0 to 35887308
Data columns (total 12 columns):
 #   Column                  Dtype
---  ------                  -----
 0   Ip                      object
 1   Ic                      object
 2   Is                      object
 3   power                   float64
 4   op1_value_current       uint32
 5   op2_value_current       uint32
 6   op1_value_previous      uint32
 7   op2_value_previous      uint32
 8   readbus_value_previous  uint32
 9   readbus_value_current   uint32
 10  writebus_value_previous uint32
 11  writebus_value_current  uint32
dtypes: float64(1), object(3), uint32(8)
memory usage: 2.1+ GB
```

Fig. 7: ABBY-M0 Dataset info

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

## 5 Fitting Power Simulators using the ABBY-M0 dataset

We can fit different simulator models using the ABBY-M0 dataset. One can use the ABBY-M0 to create different leakage models and even go further to a code rewriter engine like ROSITA [22] or ROSITA++ [21].

### 5.1 Fitting Leakage models

We constructed different leakage models using linear regression to evaluate the ABBY-M0 dataset and investigate microarchitectural leakage. In Equation 9, Y is the estimated power consumption of the target, $\beta_0$ is a constant, while $\beta_i$ is the coefficient of the explanatory variable $X_i$, and $\epsilon$ is the error. A regression model aims to find the best coefficients for each explanatory variable.

$$Y = \beta_0 + \beta_1 X_1 + \beta_2 X_2 + ... + \beta_n X_n + \epsilon \tag{9}$$

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, HD between each operand's current and previous values was added to the model. Furthermore, we add HW for memory interactions. Equation 10 shows the fitted leakage model.

$$L_{HW}(X) = [\texttt{HW(OPs)} \ \mid \ \texttt{HD(OPs)} \ \mid \ \texttt{HW(MRs)} \ \mid \ \texttt{HW(MWs)}]\beta + \epsilon \tag{10}$$

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

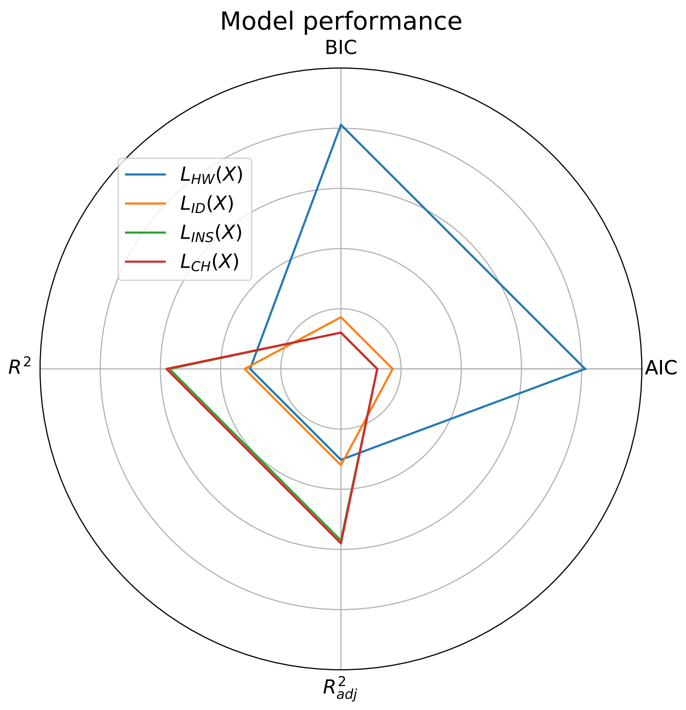


Fig. 8: Model performance parameters

12 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.

$$L_{ID}(X) = [ID_2(\texttt{OPs}) \ \texttt{|} \ ID_2(\texttt{MRs}) \ \texttt{|} \ ID_2(\texttt{MWs})]\beta + \epsilon \tag{11}$$

**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 12 models the leakage related to the instructions $L_{INS}(\texttt{X})$. 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.

$$L_{INS}(X) = [(I_P) \mid (I_C) \mid (I_S)]\beta + \epsilon \tag{12}$$

**Comprehensive leakage model.** To consider the effect of operations and operand values, we make a leakage model 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[7]. Combining the ID with the instruction-based leakage model, we make a more `comprehensive leakage model`($L_{CH}$(X)) that models not only the instruction and corresponding operand values but also the linear interaction between them.

$$L_{CH}(X) = L_{ID}(X) + L_{INS}(X) \tag{13}$$

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

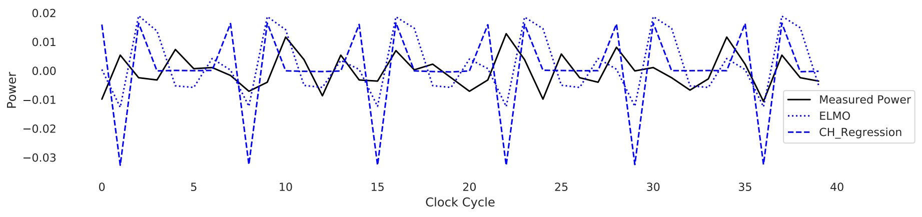


Fig. 9: Power trace for real measured compared to ELMO and $L_{CH}$(X)

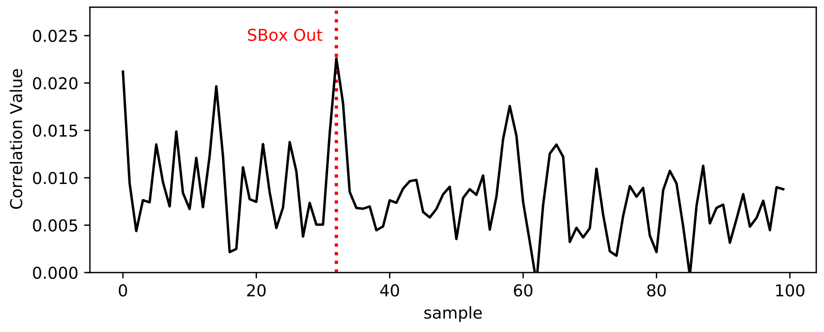
Figure 8, compares the model's performance in more detail with metrics such as AIC, BIC, $R^2$, and $R^2_{adj}$. As you can see in Fig 6, $R^2 \approx R^2_{adj}$ this is due to a large sample size ($\approx$ 16 Million) used for model fitting. Fig 9 illustrates a measured power trace vs. a simulation trace by ELMO and the CH model.
**Model selection.** We fit all the aforementioned leakage models in the ABBY data set, and Table 1 summarizes the result. Although the identity leakage

Table 1: Evaluation of leakage models

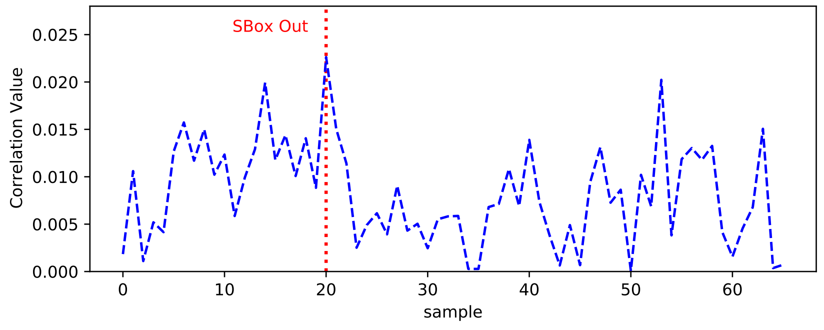
| Model | OHE($I_P$,$I_C$,$I_S$) | ID(Ops) | ID(Mem_Bus) | HW(Ops) | HW(Mem_Bus) | Rˆ2 adj | F-Stat[8] |
|---|---|---|---|---|---|---|---|
| $L_{HW}$(X) | | | | X | X | 0.30 | $6.92e+5$ |
| $L_{ID}$(X) | | X | X | | | 0.32 | $9.39e+5$ |
| $L_{INS}$(X) | X | | | | | 0.57 | $1.67e+5$ |
| $L_{CH}$(X) | X | X | X | | | 0.58 | $1.04e+5$ |

[7] 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

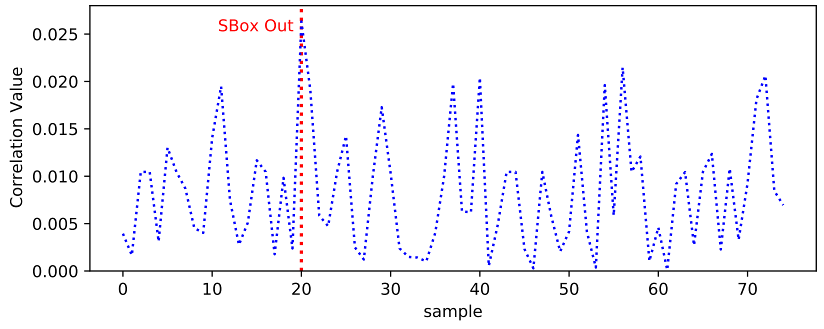
[8] Due to the large sample size($\approx 1.6e+7$) the F-test values are enormously greater than critical F-Value.



(a) Correlation based on the real measurement



(b) Correlation based on the CH model



(c) Correlation based on the ELMO model

Fig. 10: DPA attack based on correlation on a byte-masked AES

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 $R_a^2dj$. Albeit the comprehensive leakage model performance increased slightly with combining instructions and operand values, this model covers the leakage related to both operands and instructions.

## 6 Model evaluation

The main goal of any side-channel leakage simulator is to find the leaks and pinpoint their source. Differential power analysis is an effective, low-cost, and widely known side-channel attack. This attack uses variation in the electrical power consumption of the target and breaches device security by using statistical methods such as Pearson's correlation coefficient [19]. We applied a correlation-based DPA attack on an AESimplementation [9] to evaluate the ABBY-M0 dataset and the developed CH model. For the implementation of AES, we chose a first-order protected implementation by Yao et al.[27], which we refer to as *Byte-Masked AES*. We apply the attack on the first round S-box output. As proof that the CH model, developed using the ABBY-M0 dataset, captures the same leaks as ELMO, we notice how similar the correlation traces produced by ELMO and ABBY are as shown in Fig. 10. Furthermore, the correlation result for these simulators is comparable to the real measurement result of the target.

## 7 Conclusions and Future work

We propose ABBY, the first framework to automate the profiling of the microarchitectural layer. As a result, ABBY significantly reduces the human effort necessary to create transition-based 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 the DTW algorithm for trace annotation, which, although not optimal, seems to give satisfactory results. Using the ABBY-CM0 dataset we explored several leakage models ranging from traditional to transition-based, which include pipeline effects and instruction interaction. 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 that instruction-based leakage models such as ELMO and our comprehensive model are superior to traditional ones. When comparing the performance of ELMO with our comprehensive model, the results are very close, demonstrating the effectiveness of the ABBY framework.

In future work, our objective is to improve data generation, and investigate how targeted microarchitecture benchmarks such as [16] and optimizations of the model architecture can further enhance the performance of ABBY.

## References

1. Agrawal, D., Archambeault, B., Rao, J.R., Rohatgi, P.: The em side—channel(s). In: Kaliski, B.S., Koç, ç.K., Paar, C. (eds.) Cryptographic Hardware and Embedded Systems - CHES 2002. pp. 29–45. Springer Berlin Heidelberg, Berlin, Heidelberg (2003)
2. Ambrose, J.A., Aldon, N., Ignjatovic, A., Parameswaran, S.: Anatomy of differential power analysis for aes. In: 2008 10th International Symposium on Symbolic and Numeric Algorithms for Scientific Computing. pp. 459–466. IEEE (2008)
3. Bertoni, G., Zaccaria, V., Breveglieri, L., Monchiero, M., Palermo, G.: Aes power attack based on induced cache miss and countermeasure. In: International Conference on Information Technology: Coding and Computing (ITCC'05)-Volume II. vol. 1, pp. 586–591. IEEE (2005)
4. Bronchain, O., Hendrickx, J.M., Massart, C., Olshevsky, A., Standaert, F.X.: Leakage certification revisited: Bounding model errors in side-channel security evaluations. In: Boldyreva, A., Micciancio, D. (eds.) Advances in Cryptology – CRYPTO 2019. pp. 713–737. Springer International Publishing, Cham (2019)
5. Buhan, I., Batina, L., Yarom, Y., Schaumont, P.: Sok: Design tools for side-channel-aware implementations (2021)
6. Carmon, E., Seifert, J.P., Wool, A.: Photonic side channel attacks against rsa. In: 2017 IEEE International Symposium on Hardware Oriented Security and Trust (HOST). pp. 74–78. IEEE (2017)
7. Chakrabarti, A., Ghosh, J.K.: Aic, bic and recent advances in model selection. In: Bandyopadhyay, P.S., Forster, M.R. (eds.) Philosophy of Statistics, Handbook of the Philosophy of Science, vol. 7, pp. 583–605. North-Holland, Amsterdam (2011). https://doi.org/https://doi.org/10.1016/B978-0-444-51862-0.50018-6, https://www.sciencedirect.com/science/article/pii/B9780444518620500186
8. Corre, Y.L., Großschädl, J., Dinu, D.: Micro-architectural power simulator for leakage assessment of cryptographic software on ARM cortex-m3 processors. In: Fan, J., Gierlichs, B. (eds.) Constructive Side-Channel Analysis and Secure Design - 9th International Workshop, COSADE 2018, Singapore, April 23-24, 2018, Proceedings. Lecture Notes in Computer Science, vol. 10815, pp. 82–98. Springer (2018). https://doi.org/10.1007/978-3-319-89641-0_5, https://doi.org/10.1007/978-3-319-89641-0_5
9. Daemen, J., Rijmen, V.: Aes the advanced encryption standard. The Design of Rijndael **26** (01 2002)
10. Gao, S., Oswald, E.: A novel completeness test for leakage models and its application to side channel attacks and responsibly engineered simulators. In: Dunkelman, O., Dziembowski, S. (eds.) Advances in Cryptology – EUROCRYPT 2022. pp. 254–283. Springer International Publishing, Cham (2022)
11. Gao, S., Oswald, E., Page, D.: Reverse engineering the micro-architectural leakage features of a commercial processor. IACR Cryptol. ePrint Arch. p. 794 (2021), https://eprint.iacr.org/2021/794
12. Genêt, A., de Guertechin, N.L., Kaludjerovi'c, N.: Full key recovery side-channel attack against ephemeral sike on the cortex-m4. In: Bhasin, S., De Santis, F. (eds.) Constructive Side-Channel Analysis and Secure Design. pp. 228–254. Springer International Publishing, Cham (2021)
13. Hettwer, B., Gehrer, S., Güneysu, T.: Deep neural network attribution methods for leakage analysis and symmetric key recovery. In: Paterson, K.G., Stebila, D. (eds.) Selected Areas in Cryptography – SAC 2019. pp. 645–666. Springer International Publishing, Cham (2020)

14. Kocher, P.C.: Timing attacks on implementations of diffie-hellman, rsa, dss, and other systems. In: Koblitz, N. (ed.) Advances in Cryptology — CRYPTO '96. pp. 104–113. Springer Berlin Heidelberg, Berlin, Heidelberg (1996)
15. Lipp, M., Kogler, A., Oswald, D., Schwarz, M., Easdon, C., Canella, C., Gruss, D.: Platypus: Software-based power side-channel attacks on x86. In: 2021 IEEE Symposium on Security and Privacy (SP). pp. 355–371. IEEE (2021)
16. Marshall, B., Page, D., Webb, J.: Miracle: Micro-architectural leakage evaluation. Cryptology ePrint Archive, Report 2021/261 (2021), https://ia.cr/2021/261
17. McCann, D., Oswald, E., Whitnall, C.: Towards practical tools for side channel aware software engineering: 'grey box' modelling for instruction leakages. In: 26th USENIX Security Symposium (USENIX Security 17). pp. 199–216. USENIX Association, Vancouver, BC (Aug 2017), https://www.usenix.org/conference/usenixsecurity17/technical-sessions/presentation/mccann
18. Papagiannopoulos, K., Veshchikov, N.: Mind the gap: Towards secure 1st-order masking in software. In: Guilley, S. (ed.) Constructive Side-Channel Analysis and Secure Design. pp. 282–297. Springer International Publishing, Cham (2017)
19. Putra, S.D., Ahmad, A.S., Sutikno, S., Kurniawan, Y.: Attacking aes-masking encryption device with correlation power analysis. International Journal of Communication Networks and Information Security **10**(2), 397–402 (2018)
20. Sayakkara, A., Le-Khac, N.A., Scanlon, M.: A survey of electromagnetic side-channel attacks and discussion on their case-progressing potential for digital forensics. Digital Investigation **29**, 43–54 (2019)
21. Shelton, M.A., Chmielewski, Ł., Samwel, N., Wagner, M., Batina, L., Yarom, Y.: Rosita++: Automatic higher-order leakage elimination from cryptographic code. In: Proceedings of the 2021 ACM SIGSAC Conference on Computer and Communications Security. pp. 685–699 (2021)
22. Shelton, M.A., Samwel, N., Batina, L., Regazzoni, F., Wagner, M., Yarom, Y.: Rosita: Towards automatic elimination of power-analysis leakage in ciphers. IACR Cryptol. ePrint Arch. **2019**, 1445 (2019), https://eprint.iacr.org/2019/1445
23. Tiwari, V., Malik, S., Wolfe, A., Lee, M.T.: Instruction level power analysis and optimization of software. In: VLSI Design (1996)
24. Veshchikov, N., Guilley, S.: Use of simulators for side-channel analysis. In: 2017 IEEE European Symposium on Security and Privacy Workshops (EuroS PW). pp. 104–112 (2017). https://doi.org/10.1109/EuroSPW.2017.59
25. Veshchikov, N.: SILK: high level of abstraction leakage simulator for side channel analysis. In: Preda, M.D., McDonald, J.T. (eds.) Proceedings of the 4th Program Protection and Reverse Engineering Workshop, PPREW@ACSAC 2014, New Orleans, LA, USA, December 9, 2014. pp. 3:1–3:11. ACM (2014). https://doi.org/10.1145/2689702.2689706, https://doi.org/10.1145/2689702.2689706
26. Welch, D.: Thumbulator-Tool, available at https://github.com/dwelch67/thumbulator
27. Yao, Y., Yang, M., Patrick, C., Yuce, B., Schaumont, P.: Fault-assisted side-channel analysis of masked implementations. In: 2018 IEEE International Symposium on Hardware Oriented Security and Trust (HOST). pp. 57–64 (2018). https://doi.org/10.1109/HST.2018.8383891