A Tale of Two Boards: On the Influence of Microarchitecture on Side-Channel Leakage

Vipul Arora[1,2], Ileana Buhan[2], Guilherme Perin[3], and Stjepan Picek[3]

[1] Riscure B.V, Delft University of Technology
[2] Radboud University, The Netherlands

Abstract. Advances in cryptography have enabled the features of confidentiality, security, and integrity on small embedded devices such as IoT devices. While mathematically strong, the platform on which an algorithm is implemented plays a significant role in the security of the final product. Side-channel attacks exploit the variations in the system's physical characteristics to obtain information about the sensitive data. In our scenario, a software implementation of a cryptographic algorithm is flashed on devices from different manufactures with the same instruction set configured for identical execution. To analyze the influence of the microarchitecture on side-channel leakage, we acquire thirty-two sets of power traces from four physical devices. While we notice minor differences in the leakage behavior for different physical boards from the same manufacturer, our results confirm that the difference in microarchitecture implementations of the same core will leak different side-channel information. We also show that TVLA leakage prediction should be treated with caution as it is sensitive to both false positives and negatives.

Keywords: Microarchitecture · Side-channel leakage

1 Introduction

The question we ask in this work is both simple and practically relevant for an embedded system developer assigned to implement an existing cryptographic algorithm on a microcontroller. The developer is free to choose any microcontroller meeting the project's functional requirements, e.g., ARM Cortex M0, a popular choice in the IoT industry. Our developer has several options for a *given core* from the diverse SoC range offered by different manufacturers.

Devices supporting a similar instruction set architecture (ISA) vary in design depending on the implementation choices. The ISA represents an abstraction of the underlying hardware implementation, known as *the microarchitecture* [9]. Figure 1 shows the relation between the ISA and the microarchitecture. The ability to separate the ISA design from the microarchitecture was a significant step in the development of modern computing, granting functional compatibility while allowing for flexibility in the implementation. As the choices made during the ISA implementation significantly impact the final product's performance, the microarchitecture implementation is considered a trade secret, and details are typically not available in the public domain.

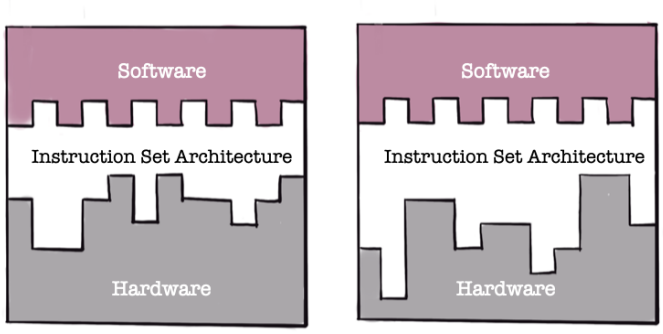

Fig. 1: We refer to the ISA implementation as the *microarchitecture*, which is manufacturer-specific and considered a trade secret. The illustration is inspired by [13].

All other things being equal, our developer would like to choose the microarchitecture implementation, which minimizes the side-channel leakage. Concretely, the question relevant to our embedded system developer is:

> *Given the choice between two implementations of the same core, how significant is the difference in side-channel leakage?*

Two devices designed with an ARM Cortex M0 core from the same family, the same ISA, and different vendors are selected for this study. To detect the source of differences between different implementations, we took special care to synchronize the traces between two devices for identical execution. We labeled the time samples in the trace with the executed instruction to identify and explain, where possible, the source of difference.

Contributions. We present a methodology for comparing software implementations across devices with the same instruction set and comment on the influence of microarchitecture implementation on side-channel leaks. We compare the manufacturing variability between different physical devices from the same manufacturer. To reveal the effects of the microarchitecture implementation, we compare devices from different manufacturers. We contrast the accuracy of leakage detection techniques with the "real" leaks obtained by profiling for the evaluation. We show that leakage detection techniques are prone to false positives and false negatives, and its results should be treated with caution.

2 Related Works

The results presented in this paper have a wider application than the practical relevance for our embedded system engineer. First, to the area of side-channel leakage simulators, which face the problem of portability across devices. For example, both ELMO [10] and [11], were created specifically for an ARM Cortex

M0 STM32F0 (30R8T6) device. If the microarchitecture implementation significantly impacts the side-channel leakage, the simulator needs to be retrained when the target is an ARM Cortex M0 `NRFf51` board. The creation of sophisticated gray-boxed leakage models, required for accurate side-channel simulators requires the inclusion of microarchitecture information. While we know that reverse engineering the microarchitecture of commercial processors is possible [4, 10], the effort is intensive.

Second, our results have an application to the area of deep learning for SCA, where training and attacking across different physical boards using the same model is possible but requires a special training procedure [3]. Golder et al. extended the previous work and explored the cross-device perspective for a large number of devices (3) [5]. Bhasin et al. showed that portability makes the deep learning attacks more difficult as the deep learning algorithms will easily overfit [1]. To prevent this, the authors proposed the multiple device model setting. Van der Valk et al. aimed to analyze the portability problem from the AI explainability perspective and discussed the overspecialization phenomenon. Overspecialization denotes the situation when a machine learning attack does not overfit when using the test set from the same device (as when not considering portability), but it overfits when attacking a different device [14]. Wu et al. provide a workaround for the multiple device model where ablation can reduce the overfitting effect [15]. Zhang et al. investigated the difficulty of profiling attacks when considering homogeneous (same devices) and heterogeneous settings (different devices) [16].Another challenge for profiled attacks is that the collection of side-channel traces becomes less reliable after a long period. Consequently, some trend noise must be added to the side-channel traces due to temperature and environmental conditions evolution over time. Heuser et al. characterized this effect and proved that trend noise drastically impedes SCA [7]. Similar findings are confirmed by Cao et al. [2].

3 Background

ARM Cortex M0. The Cortex M0 is a 32-bit RISC processor developed by ARM that implements version v6M of the ARM instruction set [8]. It is one of the most widely used embedded devices due to an efficient instruction set and affordable development costs with comprehensive development tools and support. The Cortex M0 has a Harvard architecture with both 16-bit (THUMB) and 32-bit instructions and a 32-bit data path. It does not include a data cache or memory management unit (MMU) but comes with a prefetch buffer. The ARM6 has 37 registers, consisting of thirty-one 32-bit general-purpose registers and six additional status registers. The instruction set determines the functional capabilities of a processor by specifying the list of all supported instructions.

Test Vector Leakage Assessment (TVLA) [6] is one of the most popular leakage detection methods due to its simplicity and relative effectiveness. It is based on statistical hypothesis tests and comes in two flavours: *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 taken as evidence for the presence of a leak.

Key rank estimate is a commonly used metric in SCA for assessing the performance of an attack. It is performed in a known key scenario and returns the rank of the correct key candidate in the sorted score vector of all key candidates. The key ran estimate is related to the success rate curve [12], which shows the evolution of the correct key candidate as more traces are added. There are two differences compared to the success rate: first, key ranking is performed on a fixed set of traces, whereas the success rate is performed on a variable set of traces to capture the evolution of the correct key candidate; second, key ranking can be performed for all samples in the trace, whereas the success rate is typically shown for one sample. The result of the key rank estimate is affected by the number of traces used for analysis. If leaks are present, key rank converges towards the first position as more traces are added.

4 Experimental Setup

Target devices. We selected for this study two ARM Cortex-M0 cores as comprehensive literature is available, and the Cortex M0 has found wide application in embedded and IoT devices:

1. `STM32` Discovery is a development board from ST Microelectronics for the `STM320f051` device, which consists of an on-board MCU interface enabling easy flashing and debugging using STLink over USB. The development board also offers a PPI port that connects a current probe to measure the current consumption. On inspection of the `STM32` board's schematic, we observe that the MCU interface and the target device share the same power source. The target MCU is powered by an external 3V3 supply from the current measurement port using the USB port. The coupling capacitors attached to the power pins of the target MCU are removed; they act as a low pass filter on the input power supply to the target MCU. We used two `STM32` boards for our experiments, and we refer to them as $\texttt{STM}_\texttt{A}$and $\texttt{STM}_\texttt{B}$.
2. `NRFf51` is a SoC designed for Bluetooth Low Energy applications based on Cortex M0 running at 16MHz. The `NRFf51` development kit also offers a current measurement port, and we found no coupling capacitors to the power line circuitry. The target MCU is already isolated from the interface MCU when the board is powered externally using 3V3, so no hardware modifications are required. We used two `NRFf51` boards for our experiments, and we refer to them as $\texttt{NRF}_\texttt{A}$and $\texttt{NRF}_\texttt{B}$.

Measurement setup. Throughout this paper, we maintain the same experimental setup, shown Figure 2. We use Riscure's Inspector SCA toolchain [3] for

[3] `https://www.riscure.com/security-tools/inspector-sca`

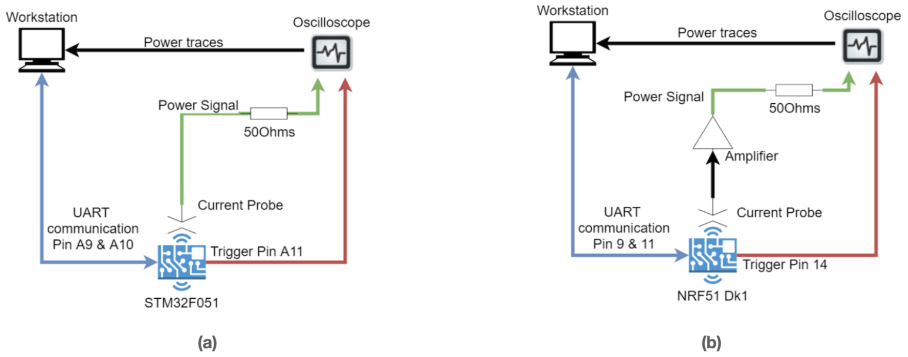

Fig. 2: Acquisition setup for (a) `STM32` and (b) `NRFf51` boards. For both, pin A9 and A10 are used for UART Rx and Tx, respectively. For connecting the trigger signal, we use pin A11 for the `STM32`and pin A14 for the `NRFf51` boards. For both, the signal from the current probe is attached to the oscilloscope through a 50Ω impedance. For the `NRFf51` board, the signal from the current probe is passed through a signal amplifier.

acquisition and signal processing. Furthermore, we use a Picoscope 3000 and a Risc-CP189 current probe. An important requirement for our setup is that both boards execute the same instructions in sync. Since the two boards have a different startup script for configuration and execution, we took special care to ensure the code between the triggers is identical for both targets. The same compiler was used to generate the binary files, and we compared the disassembled code on both boards to verify that the execution is identical. For a consistent toolchain, the software projects for both devices were created and compiled using Kiel Vision 5. An unmasked implementation of AES-128[4] was flashed on both target boards. The execution sequence is:

1. On boot/reset, a startup code runs on both target devices, which sets the system and peripheral clocks. While the `NRFf51` device works at a fixed clock speed of 16Mhz, the `STM32` device supports operation over a wide clock frequency. The startup code sets the clock frequency to 16MHz.
2. Core and UART drivers are initialized.
3. System tick interrupt is disabled.
4. Control enters the main function, AES object with a preset key is initialized.
5. Enter an infinite loop, repeat the steps below:
   (a) Receive 16 bytes of data over UART.
   (b) Set *trigger pin low*, which signals the oscilloscope to start recording.
   (c) 16 bytes of data are encrypted.
   (d) Encrypted ciphertext is returned over UART.

Synchronization of traces. To provide an accurate analysis of the observed effects, we want to align the traces with clock cycle accuracy. We compared the accuracy of the trigger signal from the oscilloscope to the recorded traces to find

[4] `https://github.com/ARMmbed/mbedtls`

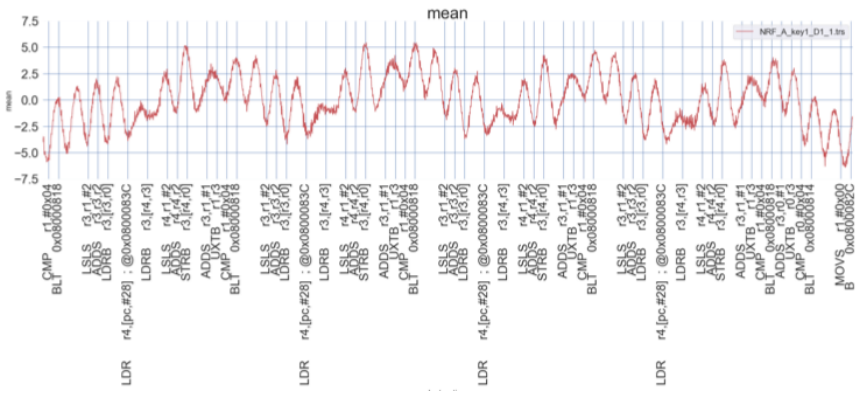

Fig. 3: An example of labeled traces.

the level of drift. Using the disassembly of C code, the assembly code line that sets the trigger pin low is found to have exact timing. We use the number of cycles it takes the program flow to enter the encrypt function, and we use it to identify the start of the encrypt function in the recorded traces.

Adding instruction labels. We used the ARM process simulator in Keil MDK version 5[5] to record the execution trace of instructions. The tool outputs a CSV file with disassembly code and with the execution time for each instruction. We use the execution trace information to link the instruction labels to their power trace segments. An example of the results of combining power traces with instruction labels obtained from the execution trace is shown in Figure 3. The example presents the acquired power trace immediately after acquisition and up until add_round_key operation on the first four bytes. Unless otherwise mentioned, for the rest of the experiments we use the power trace corresponding to the Listing 1.1. To confirm the correctness of the labeling, we also visually verified that repeated instruction sequences show a similar power consumption.

Trace sets. We collected a total of 32 trace sets, 2500 traces each, from the four physical boards ($\texttt{STM}_\texttt{A}$, $\texttt{STM}_\texttt{B}$, $\texttt{NRF}_\texttt{A}$, $\texttt{NRF}_\texttt{B}$) available. Half of the traces are provided with a fixed 16-byte plaintext, and half have 16-byte random plaintext. We used two different keys, key_1 and key_2, for the encryption and two different values for the fixed input D_1 and D_2. As the TVLA methodology [6] specifies performing a repetition to verify the results, the trace sets are labeled by 1 or 2, representing two repetitions. Figure 4 shows an overview of the collected traces.

5 A Closer Look at the Implementation

The raw traces from the `STM32` and `NRFf51` board are shown in Figure 5. A quick visual comparison confirms that the power consumption for the two devices differs significantly. The operations performed are based on repetitive patterns

[5] https://www2.keil.com/mdk5/docs

```
1   CMP r1, #0x04
2   BLT 0x08000818
3   LSLS r3,r1,#2
4   ADDS r3,r3,r2
5   LDRB r3,[r3,r0]
6   LDR r4,[pc,#28] : @0x0800083C
7   LDRB r3,[r4,r3]
8   LSLS r4,r1,#1
9   ADDS r4,r4,r2
10  STRB r3,[r4,r0]
11  ADDS r3,r1,#1
12  UXTB r1,r3
13  CMP r1,#0x04
14  BLT 0x08000818
15  LSLS r3, r1,#2
16  ADDS r3,r3,r2
17  LDRB r3,[r3,r0]
18  LDR r4, [pc,#28]: @0x0800083C
19  LDRB r3,[r4,r3]
20  LSLS r4,r1,#1
21  ADDS r4,r4,r2
22  STRB r3,[r4,r0]
23  ADDS r3,r1,#1
```

Listing 1.1: Code sequence captured during the experiments.

that can be distinctly identified for both devices.

S-box leakage. To understand how the devices are leaking, we isolate the samples corresponding to the S-box computation in round one, as shown in Figure 5. Using the Hamming Weight (HW) leakage model, we profile the targets. We select all 16 bytes of the S-box and correlate the intermediate values with the selected samples. We rank the probability of leak for all the possible key-byte combinations. With this approach, we relate observable leaks at each time sample index with the probability of the correct key byte leaking to an attacker. The results are shown in Figure 6.

For the `STM32` device, Figure 6 (top), we observe that key data leaks strongly while the subsequent byte is loaded, which seems evidence for data-overwrite leaks from registers. A small section of leaks is observed again when a key element from the same group is operated upon. This can relate to how key data is stored in subsequent memory locations, and memory access loads more than 1-byte data on the bus. This effect can be due to 4-byte memory access in Cortex M0; the old key bytes are also sent on the bus due to a word size of 4 bytes. (i.e. we observe leak of k[0][0] when operations are performed on k[0][1], leak of k[0][0], k[0][1] when operation are performed on k[0][2]; similarly We observe leak of k[0][0], k[0][1], k[0][2] when operation are performed on k[0][3]).

		STM Board			NRF Board	
		Fixed set-1	Fixed set-2		Fixed set-1	Fixed set-2
Board A	Key 1	STM_A_key1_D1_1 STM_A_key1_D1_2	STM_A_key1_D2_1 STM_A_key1_D2_2	Key 1	NRF_A_key1_D1_1 NRF_A_key1_D1_2	NRF_A_key1_D2_1 NRF_A_key1_D2_2
	Key 2	STM_A_key2_D1_1 STM_A_key2_D1_2	STM_A_key2_D2_1 STM_A_key2_D2_2	Key 2	NRF_A_key2_D1_1 NRF_A_key2_D1_2	NRF_A_key2_D2_1 NRF_A_key2_D2_2
Board B	Key 1	STM_B_key1_D1_1 STM_B_key1_D1_2	STM_B_key1_D2_1 STM_B_key1_D2_2	Key 1	NRF_B_key1_D1_1 NRF_B_key1_D1_2	NRF_A_key1_D2_1 NRF_A_key1_D2_2
	Key 2	STM_B_key2_D1_1 STM_B_key2_D1_2	STM_B_key2_D2_1 STM_B_key2_D2_2	Key 2	NRF_B_key2_D1_1 NRF_B_key2_D1_2	NRF_B_key2_D2_1 NRF_B_key2_D2_2

Fig. 4: Overview trace sets. The nomenclature is class_board_key_data_repetition. For example, a trace set with the name NRF_B_key2_D1_1 means it was collected from $\texttt{NRF}_\texttt{B}$ board, key K_2 is used for encryption, D_1 is provided as fixed input, and 1 is the repetition cycle.

.

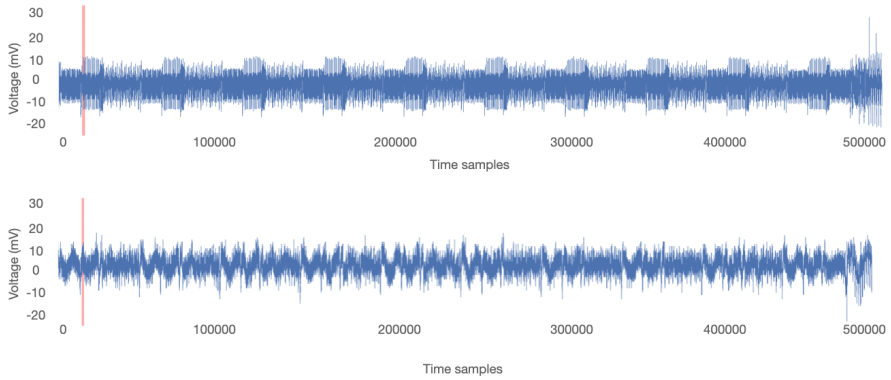

Fig. 5: The raw power trace for the $\texttt{STM}_\texttt{A}$ device (top) and $\texttt{NRF}_\texttt{A}$ device (bottom). The highlighted section marks the 1st round of the S-box operation on the first byte of data. The selection starts at index 14 910 and has a length 1 235 samples. This section of trace has been used for the evaluation in Sections 6 and 7.

Similarly, the results for the `NRFf51` device are presented in Figure 6 (bottom). We observe that the correct key intermediate is leaking consistently after the first time it is read from memory, and key data is leaking when operations are performed on byte data stored in subsequent memory locations. Memory access read 4 bytes of consecutive data from the provided memory address. Comparing the leaks across the two devices, we note that data-overwrite leaks are observed at similar trace sections. The key bytes start to leak subsequent to the `STR` instructions and leak while the next byte data is loaded by `LDR` instructions. We surmise that the contrasting behavior is a result of the difference in microarchitecture implementation. The `NRFf51` device is a low power board; memory access consumes significant power and impacts dynamic power consumption. The choice of memory technology will impact the leaks observed from the board.

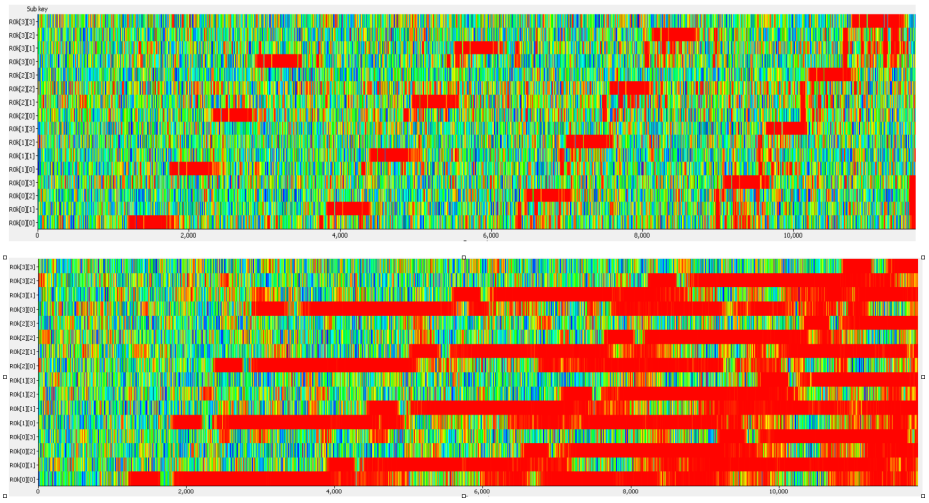

Fig. 6: (To be viewed in colors.) Key rank results for the `STM32` (top) and `NRFf51` (bottom) devices. The selection captures the s-box operation. We perform key ranking on all 16 bytes in round 1. The red color indicates strong leaks, where the correct key candidate is ranked in the first position, whereas other colors represents weak leaks.

For the remainder of the report, we select the leaks from the S-box operation on 1 byte of data (Byte 1). To maintain uniformity in the analysis, the same trace section will be used for all comparisons.

6 The Influence of Manufacturing Variability

Manufactured silicon chips have variations due to the raw material used or due to variations in the manufacturing process. Non-uniform etching can introduce inconsistencies in transistors' depletion layer, which will affect the leakage current generated on switching. Inconsistencies are spread out across peripherals at the microarchitecture level, which means that each physical device will have its power fingerprint resulting from the accumulation of these effects.

This section explores the manufacturing differences between boards from the same manufacturer, namely, the differences in side-channel leaks from $\texttt{STM}_\texttt{A}$ vs. $\texttt{STM}_\texttt{B}$, and $\texttt{NRF}_\texttt{A}$ vs. $\texttt{NRF}_\texttt{B}$. These results are useful for putting in perspective the results obtained in Section 7. The devices are prepared with similar hardware modifications and flashed with the same binary, keeping the key and input data parameters identical, as described in Section 5.

The result of the TVLA test for both $\texttt{STM}_\texttt{A}$ and $\texttt{STM}_\texttt{B}$, Figure 7 (a), shows a similar shape for the leakage. Additionally, the repetition of the test (with different inputs and keys) shown with a dotted line confirms these results. Key rank results in Figures 7 (b) and (c) show leaks over a wider section of power traces for both devices compared to the results predicted by TVLA. Key rank

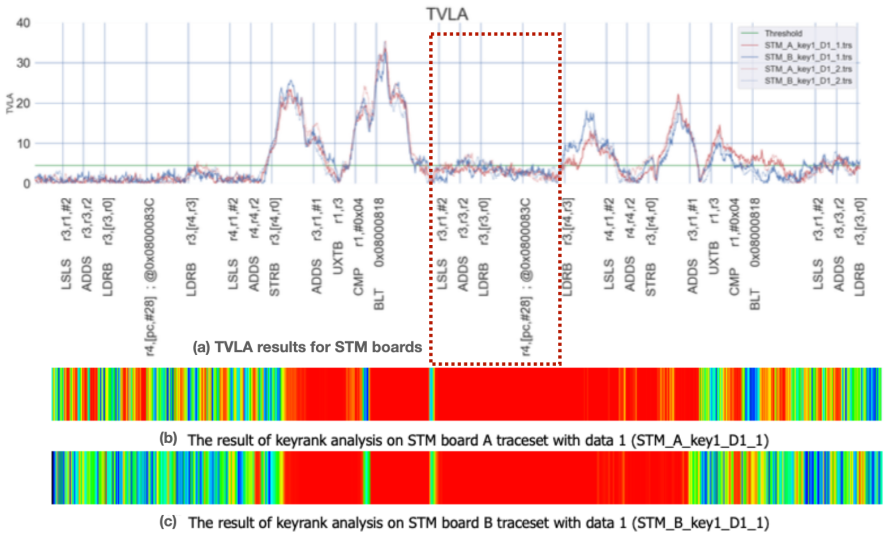

Fig. 7: (To be viewed in color.) Results of data leakage comparison for $\texttt{STM}_\texttt{A}$ and $\texttt{STM}_\texttt{B}$. From the top: (a) TVLA traces, (b) key rank results for $\texttt{STM}_\texttt{A}$, (c) key rank results for $\texttt{STM}_\texttt{B}$. The red rectangle indicates the area of the trace where TVLA shows false negative (profiling indicates leakage).

results for the $\texttt{STM}_\texttt{A}$ device show leaks at the beginning of the trace, a behavior not seen in the key ranking results for the $\texttt{STM}_\texttt{B}$ device. These leaks are probably caused by manufacturing defects in $\texttt{STM}_\texttt{A}$. Gaps in leakage are observed for both boards during the execution of `UXTB` and `BLT` instructions, which can be sourced from effects in the physical layer. We observe gaps in the results of key rank analysis during the `ADDS` and the `BLT` instruction, which is consistent for both boards. We could attribute this effect to operations being implemented at the hardware level, which mask the leakage of key data at those locations.

The TVLA results shown in Figure 8 (a) indicate a very similar trend for both `NRFf51` boards. Key rank analysis results for the two boards are shown in Figure 8 (c) and (d). While we note a slight variation between the leakage $\texttt{NRF}_\texttt{A}$ and $\texttt{NRF}_\texttt{B}$, the overall trend is similar. However, we note a significant difference between the leakage predicted by TVLA, which indicates both false positive and false negative leakage.

To summarize, we confirm that the manufacturing process may create slight differences between the leaks in the different physical devices we examined, but the overall trend seems consistent. Based on the experimental results, we conclude that there are significant differences between the leakage predicted by TVLA and the ground truth as indicated by profiling the targets.

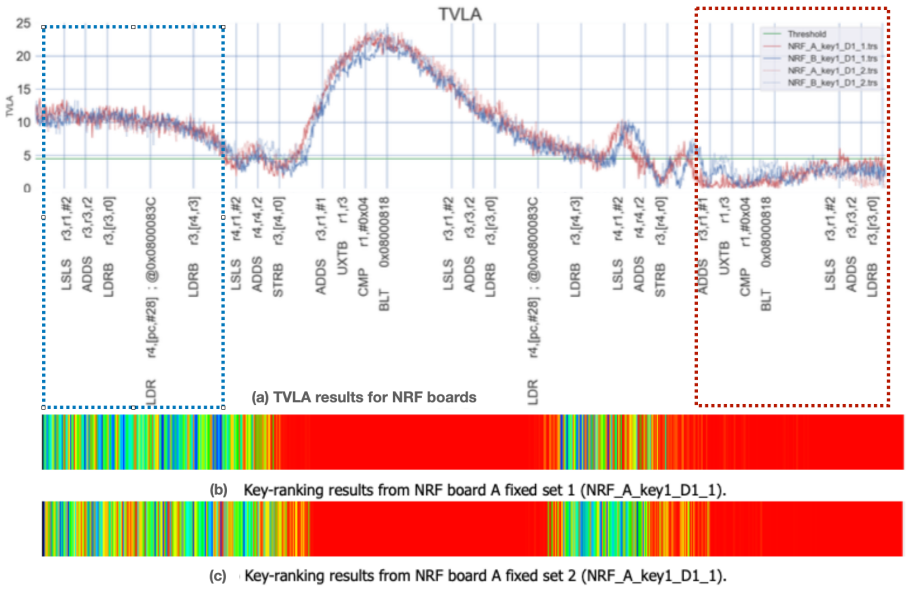

Fig. 8: (To be viewed in color.) Results of data leakage comparison for $\texttt{NRF}_\texttt{A}$ and $\texttt{NRF}_\texttt{B}$. From the top: (a) TVLA traces, (b) key rank for $\texttt{NRF}_\texttt{A}$, (c) key rank for $\texttt{NRF}_\texttt{B}$. The blue rectangle indicates the area of the trace where TVLA shows false positive (leakage while profiling indicates no leakage). The red rectangle indicates the area of the trace, where TVLA shows false negative (no leakage while profiling clearly indicates leakage).

7 The Influence of Microarchitectural Implementation

The ARM Cortex-M0 microprocessor has a three-stage pipeline, which means that there can be up to three instructions implemented in the fetch, decode, and execute stages of the pipeline. Memory access greatly impacts the dynamic power, so the effect of memory instructions is significant and can be diffused to be visible while other instructions are executed. Furthermore, while we know the instruction executed at every clock cycle, we note that the power trace consists of a *cumulative effect* from all pipeline stages of the processor.

When porting code to a device with a similar hardware architecture, the grouping of instructions in pipeline stages will probably be also similar. However, the magnitude/contribution of leaks from different pipeline stages may vary for different devices. Additionally, as microarchitectural implementation choices are not public, the best we can do for describing the difference in side-channel leakage between the `STM32` and `NRFf51` boards is a plausible explanation.

7.1 Power Profiles

Mean traces. Fixed vs. random mean plots comparing the two devices are presented in Figure 9. The power traces from `STM32` devices have a higher power

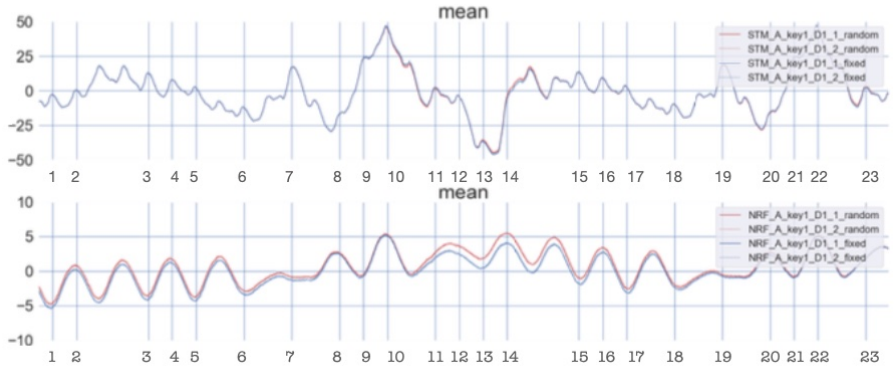

Fig. 9: Fixed vs. random mean trace plot for `STM32` (top) and `NRFf51` (bottom). The y-axis shows the power consumption, and the x-axis represents time. The numbers on the x-axis are the instruction being executed, see Listing 1.1.

consumption compared to the traces obtained from the `NRFf51` device, as evident from the scale of the y-axis. `STM32` is designed for general-purpose IoT applications, whereas the `NRFf51` is a low energy device with a current consumption of 2mA. The low power of the `NRFf51` device makes it more sensitive to noise.

Comparing the mean trace plots from both devices, we observe the fixed vs. random lines deviating at the same power trace sections. In Figure 9, this is visible in the difference between the red (random set) and blue (fixed set) lines for both plots. The deviations reveal sections of code with a dynamic power component. If the underlying data is changed in the code section, we will observe fluctuations in the specific section of power traces. We repeated the experiments with a different value for the input data to verify that the observations are not incidental. We distinguish between the two repetitions by presenting results with a solid and a dotted line in the power profile comparison plots. Repeating artifacts are observed for `LDR` (labeled 6 and 18) and `STRB` (labeled 10 and 22) instructions in power trace, confirming the correct labeling of traces with instructions. In the case of the `NRFf51` devices, the effects of individual instructions are not as prominent and are difficult to distinguish visually.

From the mean plots of fixed (blue) vs. random (red) execution for both the device, we observe both the sets exhibit a similar trend though they differ along certain sections of the traces. In Figure 9 (top), we notice that the deviation between the random and fixed sets is visible only following the `STRB r3,[r4,r0]` (labeled 10) instruction until the `BLT` branch (labeled 14). In Figure 9 (bottom), we can distinctly see the execution trace of fixed as well as random set. The interesting observation is that the distance between the mean of two sets increases substantially after the `STRB r3,[r4,r0]` (labeled 10) instruction and then slowly decreases up until the `BLT` instruction (labeled 14). From the results in the pre-

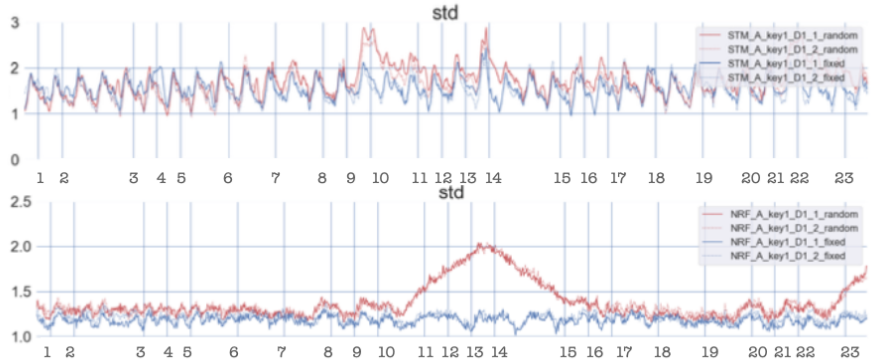

Fig. 10: Fixed vs. random standard deviation trace plot for `STM32` (top) and `NRFf51` (bottom). The y-axis shows the power consumption, and the x-axis represents time. The numbers on the x-axis are the instruction being executed, see Listing 1.1.

vious section, we know that these are the locations where leaks are observed. The software implementation seems to show evidence of data overwrite leaks at similar sections across devices of both classes.

Standard deviation. The operands influence the power consumption due to the toggling of bits when new data is loaded. An increase in the standard deviation of the random set is observed where the power consumption depends on the underlying data. The standard deviation of the fixed set provides us with a base level for executing a set of operations with constant data.

Standard deviation plots in Figure 10 show that the standard deviation for the fixed set consistently varies for every clock cycle. This behavior is consistent for the fixed sets for all boards and repetitions. The increase in standard deviation for the random sets provides evidence of leaks, and interestingly these are observed at similar trace sections for both the `STM32` and `NRFf51` devices. For the `STM32` traces, the deviation in random vs. fixed plot occurs near the `ADDS r4,[r4,r2]` (labeled 9), `STRB r3,[r4,r0]` (labeled 9) and `BLT 0x08000818` (labeled 14) instructions where variance of random set is visibly higher in comparison to the fixed set. In the case of `NRFf51` boards, the variance of a random set is higher compared to the fixed set for all sections of the trace. Following the execution of the `STRB r3,[r4,r0]` instruction (labeled 10), the variance of random set increases until the `CMP r1,#0x04` instruction (labeled 13) where it peaks and goes down until `LSLS r3,r1,#2` instruction (labeled 15) where the operation on next byte starts.

7.2 Data Leakage

TVLA results. Figure 11 shows the TVLA results for the $\texttt{STM}_\texttt{A}$ (red line) and the $\texttt{NRF}_\texttt{A}$(blue line) devices. A green line represents the threshold value of 4.5. The plot shows that the side-channel leaks for the two devices differ significantly.

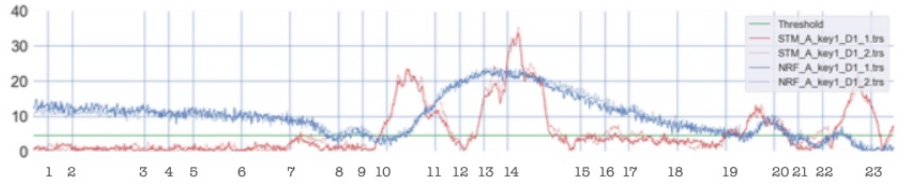

Fig. 11: TVLA results for the $\texttt{STM}_\texttt{A}$ and $\texttt{NRF}_\texttt{A}$ devices. The numbers on the x-axis are the instructions being executed, see Listing 1.1.

For the $\texttt{STM}_\texttt{A}$, the TVLA value rises above the 4.5 threshold at `STRB r3,[r4,r0]` instruction, goes down at `UXTB r1, r3` instruction and rises again covering `CMP r1,#0x04`(labeled 13) and `BLT 0x08000818` (labeled 14) instructions. For the $\texttt{NRF}_\texttt{A}$ device, the TVLA results show leakage for almost all instructions (labeled 1-21). However, as seen in the previous section, the TVLA results need to be considered with caution.

Key rank analysis results have been added as a transparent layer over the TVLA for both boards in Figure 12.

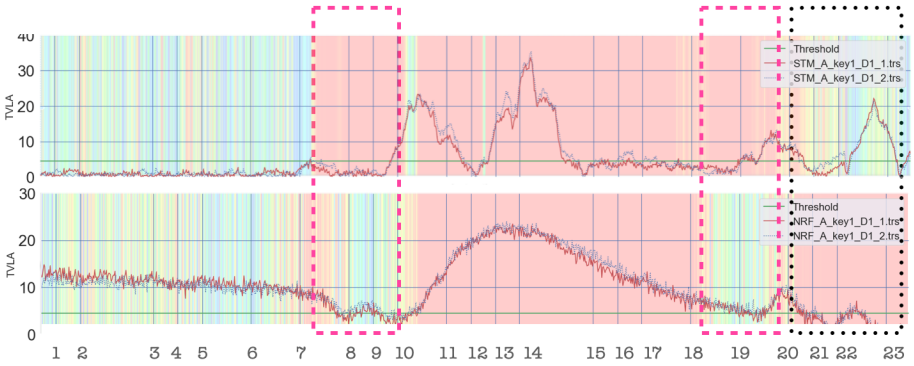

Fig. 12: (To be viewed in colors.) Overlay of the key rank estimate on the TVLA results for $\texttt{STM}_\texttt{A}$ device(top) and $\texttt{NRF}_\texttt{A}$ device (bottom). Red regions represent the index locations where the correct key is ranked first. The rectangles highlight differences in leakage between the two boards. The numbers on the x-axis are the instruction being executed, see Listing 1.1.

For the $\texttt{STM}_\texttt{A}$ device, the correct key data starts leaking Figure 12 (top) from the `LDRB r3, [r4,r3]` instruction (labeled 7) until the `STRB r3,[r4,r0]` instruction (labeled 22). Our hypothesis for the leak observed during the `LDRB r3,[r4,r3]` instructions (labeled 7) is that the key byte is being loaded on the bus. The key byte also leaks while the arithmetic instructions are being

performed, at `ADDS r4,r4,r2` instruction (labeled 9). We believe this is an effect of the three-stage pipeline: while the `ADDS` instruction is being executed the data is being pre-fetched for the `STRB` instruction.

We see that the correct key byte continues to leak in the subsequent instructions even though no operations are being performed directly on the key data. In the analyzed s-box implementation, the loop operates on 4 bytes, four times to operate on a total 16 bytes of data; the check for the loop occurs at `CMP r1,#0x04` instruction(labeled 13). The check compares the relative value of R1 to #0x04 and branches to the next instruction if the R1 value is less than 4.

The subsequent instructions `LSLS` and `ADDS` compute the relative index from which the next key data is to be loaded by the `LDRB` instruction, which is when the leak of key data stops. We find this to be an interesting behavior since the data stops leaking when the data in the memory bus-A is overwritten by new data. We do not have an explanation for the gaps in the resulting leaks for the `STRB` (labeled 10) and `UXTB` (labeled 12).

Figure 12 (bottom) shows the key rank analysis for the $\texttt{NRF}_\texttt{A}$ device. The correct key byte starts leaking at `STRB, r3,[r4,r0]` instruction (labeled 10), and leaks until `LDR r4,[pc,#28]` instruction (labeled 18). The leaks observed in the $\texttt{NRF}_\texttt{A}$ device seem to have a strong effect on the dynamic power, and its effects seem diffused, showing up while other instructions are being executed. We can infer that the correct key byte is on the bus after the `STRB` instruction (labeled 10), which leaks over the subsequent instructions as the data is being overwritten. An interesting behavior observed in $\texttt{NRF}_\texttt{A}$boards is the leak of the first key byte when the operations are being performed on the next byte of data, due to register overwrites from `LDR` instruction.

To summarize, we confirm that the influence of microarchitecture implementation has a significant effect on the leakage behavior of the two boards we analyzed. The results for the $\texttt{NRF}_\texttt{A}$ device differ from the key rank results on $\texttt{STM}_\texttt{A}$ device showing an additional leak of arithmetic (LSLS) instruction (labeled 15). The results for the $\texttt{STM}_\texttt{A}$ and the $\texttt{NRF}_\texttt{A}$ devices show a similar trend subsequent to the `STRB` instruction.

8 Conclusions and Future Work

Our results show that while the power traces collected from the boards of the two manufacturers have very different visual profiles, some instruction sequences leak in the same way, which can be explained by the similar pipeline executions of instructions for both cores. To answer whether the microarchitecture impacts side-channel leakage, we first investigate the influence of manufacturing variation. While we observe differences between physical boards, the trend for side-channel leakage for the two boards we investigated is similar. When comparing the side-channel leak between different chips, we see clear evidence of leakage behavior that we attribute to microarchitecture implementation differences.

In terms of the impact on the design of side channel simulators, our results show that the existence of a generic simulator, e.g., for an ARM-Cortex M0, is

improbable. Differences in microarchitecture, such as differences in memory implementation or other functional optimisations, require that a simulator which predicts side channel leakage needs to be trained for different silicon implementations. For the portability of templates between different core implementations, we extrapolate that the differences in microarchitecture will be a deciding factor.

We compared TVLA, probably the first choice of leakage assessment technique, with the leakage obtained by profiling. Despite its simplicity and based on the differences observed in our results, we would caution our embedded system developer against using TVLA alone to determine leakage behavior and suggest using key ranking as a more robust, albeit more effort-intensive technique.

References

1. Bhasin, S., Chattopadhyay, A., Heuser, A., Jap, D., Picek, S., Shrivastwa, R.R.: Mind the portability: A warriors guide through realistic profiled side-channel analysis. In: 27th Annual Network and Distributed System Security Symposium, NDSS 2020, San Diego, California, USA, February 23-26, 2020. The Internet Society (2020), https://www.ndss-symposium.org/ndss2020/
2. Cao, Y., Zhou, Y., Yu, Z.: On the negative effects of trend noise and its applications in side-channel cryptanalysis. IACR Cryptology ePrint Archive **2013**, 102 (2013), http://eprint.iacr.org/2013/102
3. Das, D., Golder, A., Danial, J., Ghosh, S., Raychowdhury, A., Sen, S.: X-deepsca: Cross-device deep learning side channel attack. In: 2019 56th ACM/IEEE Design Automation Conference (DAC). pp. 1–6 (2019)
4. Gao, S., Oswald, E., Page, D.: Reverse engineering the micro-architectural leakage features of a commercial processor. Cryptology ePrint Archive, Report 2021/794 (2021), https://eprint.iacr.org/2021/794
5. Golder, A., Das, D., Danial, J., Ghosh, S., Sen, S., Raychowdhury, A.: Practical approaches toward deep-learning-based cross-device power side-channel attack. IEEE Transactions on Very Large Scale Integration (VLSI) Systems **27**(12), 2720–2733 (2019). https://doi.org/10.1109/TVLSI.2019.2926324
6. Goodwill, G., Jun, J., P.Rohatgi: A testing methodology for side channel resistance validation. NIST non-invasive attack testing workshop (2018)
7. Heuser, A., Kasper, M., Schindler, W., Stöttinger, M.: A New Difference Method for Side-Channel Analysis with High-Dimensional Leakage Models. In: Dunkelman, O. (ed.) CT-RSA. Lecture Notes in Computer Science, vol. 7178, pp. 365–382. Springer (2012)
8. Limited, A.: Arm v6-m architecture reference manual. Tech. rep., ARM Limited (ARM DDI 0419E (ID070218) 2018)
9. Marshall, B., Page, D., Webb, J.: Miracle: Micro-architectural leakage evaluation. Cryptology ePrint Archive, Report 2021/261 (2021), https://eprint.iacr.org/2021/261
10. McCann, D., Oswald, E., Whitnall, C.: Towards practical tools for side channel aware software engineering: 'grey box' modelling for instruction leakages. In: USENIX Security Symposium. pp. 199–216 (2017)
11. Shelton, M.A., Samwel, N., Batina, L., Regazzoni, F., Wagner, M., Yarom, Y.: Rosita: Towards automatic elimination of power-analysis leakage in ciphers. In: NDSS (2021)

12. Standaert, F.X., Malkin, T.G., Yung, M.: A unified framework for the analysis of side-channel key recovery attacks. In: Annual international conference on the theory and applications of cryptographic techniques. pp. 443–461. Springer (2009)
13. Stokes, J.: Inside the Machine, An illustrated Introduction to Microprocessors and Computer Architecture. No startch press/ars technica library (2007)
14. van der Valk, D., Picek, S., Bhasin, S.: Kilroy was here: The first step towards explainability of neural networks in profiled side-channel analysis. In: Bertoni, G.M., Regazzoni, F. (eds.) Constructive Side-Channel Analysis and Secure Design. pp. 175–199. Springer International Publishing, Cham (2021)
15. Wu, L., Won, Y.S., Jap, D., Perin, G., Bhasin, S., Picek, S.: Explain some noise: Ablation analysis for deep learning-based physical side-channel analysis. Cryptology ePrint Archive, Report 2021/717 (2021), https://eprint.iacr.org/2021/717
16. Zhang, F., Shao, B., Xu, G., Yang, B., Yang, Z., Qin, Z., Ren, K.: From homogeneous to heterogeneous: Leveraging deep learning based power analysis across devices. In: 2020 57th ACM/IEEE Design Automation Conference (DAC). pp. 1–6 (2020). https://doi.org/10.1109/DAC18072.2020.9218693