Porting ROSITA to the ChipWhisperer for Broader Automatic Elimination of Power Analysis Leakages

Samuel Wong
University of Adelaide
kwok.y.wong@student.adelaide.edu.au

Chitchanok Chuengsatiansup
University of Adelaide
chitchanok.chuengsatiansup@adelaide.edu.au

Madura A. Shelton
University of Adelaide
madura.shelton@adelaide.edu.au

Yuval Yarom
University of Adelaide
yval@cs.adelaide.edu.au

ABSTRACT

With the advent of Internet-of-Things (IoT) devices in recent years, side-channel attacks have become a more relevant concern for embedded systems developers than ever. In particular, attacks abusing power analysis leakages continuously pose a significant security threat to cipher implementations, as they have the potential to reveal private keys and other secret information from the device. Since testing for side-channel leakages can be expensive and require specialised skills, which often cannot be afforded by IoT start-ups with limited resources, there exists a need for user-friendly tools that can automatically emulate, identify, and correct power analysis leakages in programs. While such tools exist, they are far and few between, and are often constrained to a singular platform. To fill this need, we present a way to expand Rosita, a code rewrite engine created by Shelton et al. (NDSS 2021), capable of automatically eliminating power analysis leakages from programs. We create a power model for the target on-board the ChipWhisperer, an open-source toolchain for embedded hardware security research. We show that power traces captured using the ChipWhisperer power model can be used and analysed by Rosita, thus broadening its ability to correct leakages on other platforms.

1 INTRODUCTION

Historically, side-channel attacks have existed for longer than computers have. A classic example of a side-channel attack predating computers is a polygraph test. By measuring and recording the blood pressure, pulse, breathing patterns, and other physiological indicators when a series of questions are asked, one may be able to gain insight into whether someone is telling the truth. This is effectively equivalent to a side-channel attack against the human brain. More generally, by observing unintended leakages of information from a given system, an attacker can recover secret information otherwise unobtainable by conventional means.

In computer security, side-channel attacks exploit indirect device signals such as cache accesses, timing information, and in particular, power consumption. Side-channel attacks in the form of power analysis have been proven to leak private keys from ciphers such as AES and more [9], compromising the security of devices that rely on them to protect the data of users. With the rapid expansion of the IoT market in the past decade, there is an ever-increasing amount of new embedded devices being developed by startups all across the globe, whose limited budgets are often incompatible with the high costs of evaluation labs for assessing side-channel leakages.

While many mitigation tools have been created for cache-based and time-based side-channel attacks [10], there remains a need for user-friendly tools capable of automatically eliminating power analysis leakages from programs across different hardware platforms. In general, elimination can be achieved by first emulating the supplied program, then detecting assembly instructions that cause power analysis leakages using a power model or the hardware description of the target processor [8], and rectifying the associated instructions such that the leak is no longer observable (through masking) or the correlation between the leak and the secret information is eliminated (through blinding). In 2021, Shelton et al. [14] presented Rosita, a code rewrite engine that uses an augmented version of the ELMO leakage emulator developed by McCann et al. [10] as a basis to emulate the micro-architectural characteristics, and subsequently identify code locations that are causing a leakage. Then, using selected rewrite rules, Rosita is able to rewrite the code such that the power leakage will no longer be observable by an attacker.

1.1 Motivation

Though Rosita can automatically eliminate power analysis leakages from programs, it is only able to do so on a limited number of platforms through the ELMO* leakage emulator. With the objective of allowing broader automatic elimination of leakages and subsequently helping improve the overall security of embedded hardware, we present ways to increase the number of supported platforms on Rosita.

To achieve this, we use the ChipWhisperer, an open-source toolchain for embedded hardware security research developed by O'Flynn and Chen [12] to capture power traces from the ARM Cortex-M0 based STM32F030F4P6 target on-board. The expected result of this project is a tool that needs to be capable of three functionalities. The tool should be able to interact with the ChipWhisperer toolchain to capture power traces from the target board as it executes the user-supplied program. Following on, the tool should bridge the traces to the ELMO* leakage emulator where it can test the program for leakages with respect to both the measured and generated power consumption through a newly created power model for the target. Finally, the detected leakages should be integrated into the Rosita workflow as described by Shelton et al. [14, p. 2]. This work provides future embedded systems developers with a user-friendly and streamlined solution to testing products for power analysis leakages and automatically eliminating such leakages before they make their way to production.

2 BACKGROUND

In this section, we aim to provide context for the various types of power analysis attacks commonly used against cryptographic applications (Sect. 2.1), the Test Vector Leakage Assessment methodology (Sect. 2.2), leakage emulators and evaluation of power models (Sect. 2.3), as well as automatic countermeasures against power analysis leakages (Sect 2.4).

2.1 Power analysis attacks

From a physics point of view, the power consumption of transistors in a processor depends on the current state they hold. An attacker capable of observing the power consumption over time in a cryptographic hardware device will be able to perform what is known as a power analysis attack. There exist many techniques in power analysis attacks. There is simple power analysis (SPA), where an attacker visually inspects power traces captured from a device or the power consumption graph plotted relative to time, such that it is possible to identify which instructions are being executed or loaded in the processor pipeline, based on previous profiling. A more sophisticated technique is differential power analysis (DPA), first described in 1998 by Kocher et al. [7]. The attacker collects a large number of power traces from a device leaking the Hamming weight during cryptographic operations. Then, using statistical analysis, the attacker derives the intermediate values used in cipher implementations, which allows the original secret keys to be recovered. Such attacks have been demonstrated to be successful against targets ranging from smart cards [7] to IEEE 802.15.4 nodes used in IoT devices [13].

2.2 Side-channel leakage assessment

With the threat of side-channel attacks becoming better known among vendors in the industry sector, greater attention has been put on developing standardised assessment programs, which evaluate the presence of leakages in cryptographic implementations for a given device under test. One such methodology is called the Test Vector Leakage Assessment (TVLA), first proposed by Goodwill et al. [6] in 2011. The TVLA seeks to be a systematised method for confirming and ruling out side-channel leakages of sensitive information, while being easy-to-apply and cost-effective, not requiring test operators to be exceptionally skilled in performing side-channel attacks. The TVLA methodology uses Welch's t-test to identify statistical differences between two sets of side-channel measurements. The test uses the means and variances of the two distributions to derive a statistic called the t-value. If one set of measurements were to be taken with fixed data, and another set with random data (a fixed-vs.-random test), a high t-value would indicate statistical differences between the two sets, implying that the device is emitting some form of data-dependent information in its operation through a side-channel [6].

When assessing implementations of cryptographic algorithms, there are two classes of tests to be followed: general and specific. General tests look to detect leakages that are functions of the input data or cipher key, while specific tests target specific intermediate steps of the algorithm that can be exploited to recover sensitive information, such as S-Box accesses in AES [9, 11]. It is important to note that the t-test statistic is used for supporting or rejecting a null hypothesis [15]. In other words, passing the TVLA does not guarantee resistance against all attacks, and a negative result does not necessarily imply the absence of leakages. Similarly, a positive result in a general test does not mean the leakages are directly and immediately exploitable, but rather it indicates the possibility for exploitation [3]. Despite these limitations, the confidence of the result can be improved through repeated testing with different data, which is why the TVLA method has remained useful as a simple and effective first-step evaluation tool for side-channel leakages [14].

2.3 Leakage emulators

In 2017, McCann et al. [10] proposed an emulator for power leakages for the ARM Cortex-M0 (ELMO). By profiling and reducing a set of 21 M0 instructions deemed relevant for cryptographic operations into five classes, an emulator could be modelled that provided insight into the leakage characteristics of the device. Similar efforts were made by Le Corre et al. [8] on using the HDL source code of the ARM Cortex-M3 to infer leakage properties of each stage of the processor

pipeline and creating the micro-architectural power simulator (MAPS). Though, these emulators were shown to be inaccurate and unable to detect all leakages. Gao and Oswald [4] first proposed the concept of completeness for power models used in leakage emulators in 2021, whereby a model is considered complete if and only if it captures all relevant state information for leakage assessment. As demonstrated in their analysis, neither emulator satisfied the completeness test. There have also been efforts made towards improving existing leakage emulators. In 2021, Gao et al. [5] demonstrated that it is feasible to build more accurate emulators by reverse-engineering the micro-architectural components of the target processor and analysing their leakages on a pipeline stage basis.

2.4 Automatic countermeasures

The elimination of side-channel leakages has often been an iterative process, involving manual work between testing the implementation for leakages, identifying the source of the leakages, masking the leakages such that they are no longer observable, and performing the test again to ensure the leakage has been correctly eliminated [14]. As the location and nature of power analysis leakages are dependent on not only the implementation but also the micro-architecture of the device under test [1], there currently exists no generalised or broad-spectrum countermeasure that is able to eliminate leakages on all hardware platforms. While it is possible to manually identify instruction locations that causes the leakage and masking it by every iteration, this method of elimination is inefficient and demands a significant amount of time, requiring the technician administering the evaluation to be skilled in not only carrying out side-channel attacks but also in applying leakage fixes for the specific implementation. Countermeasures in the form of code rewrite engines, such as Rosita by Shelton et al. [14], speeds up the process significantly by removing the human factor and automating the manual code patching that was previously required. Though, the efficiency of this automated elimination comes at the trade-off of a limited range in supported hardware platforms, requiring a linear regression model of the device's power leakage characteristics to be developed prior to performing any testing [10].

3 METHODOLOGY

3.1 Model building process

In order to emulate leakages found on the ChipWhisperer Nano with the ELMO emulator, we build a model of the target's power leakage characteristics by collecting power traces while the device executes different combinations of calibration instructions. The combinations include 21 instructions that the authors of ELMO consider to be of frequent use in typical cryptographic operations, which can be divided into five similarly-leaking instruction classes as defined by McCann et al. [10]. Each combination contains a sequence of three instructions, a sequence testing the `movs`, `str`, and `eors` instructions would resemble the following:

```
1 movs r0, r0
2 movs r0, r0
3 movs r0, r0
4 movs r0, r0
5 movs r0, r0
6 movs r2, r3
7 str r5, [r4]
8 eors r6, r7
```

Listing 1: Example model calibration code for `movs`, `str`, and `eors` instructions.

After the traces are collected, we process and trim the traces such that they only include the portions coinciding with our instructions of interest. Note that in Listing 1, a series of stationary `movs` from have been added before the instructions of interest as a fixed delay, this is to allow for easier trimming later in the process. Then, the traces will be analysed through linear regression, where the coefficients of the data representative of the bit values in the processor's Arithmetic Logic Unit are extracted [14], to be used by ELMO as a model for power consumption.

To aid in repeating the model building process, the authors of ELMO have also provided the relevant MATLAB functions for computing the coefficients, as well as pre-generated models of the STM32F0302 and NXP LPC1114 evaluation boards for testing[1].

3.2 Trace collection process

Power traces were collected using the scope onboard the ChipWhisperer Nano. The collection process is separated into two phases, a pre-capture preparation phase where all necessary objects are generated and compiled, and a capture phase where each calibration program is flashed on the device and traces are collected as the target processor executes.

In the preparation phase, we first generate the assembly instructions containing different sequence combinations for each of the five leakage classes. As the model for each leakage class will be built independently and will not be merged until the linear regression step, we split the assembly source

[1] Available at https://github.com/sca-research/ELMO.

output into separate directories to improve file organisation. We then use the AS assembler under the GNU ARM Embedded Toolchain to assemble each instruction combination along with the needed setup instructions and trigger function calls into small object files. These object files will be linked later with the C driver code in the capture phase. At the end of the preparation phase, we also compile the C driver program containing all necessary libraries, such as SimpleSerial for communication with the target, as well as the STM32F0 HAL driver for setting the trigger and other low-level functionalities.

We now turn our attention to the capture phase. For each sequence combination being tested, we copy (and overwrite if necessary) the object file containing the current combination of instructions to the ChipWhisperer project directory. We use the ARM toolchain's version of `gcc` to link the object under test with the driver program, and `objdump` to create a hexadecimal object file that can be installed on the target device. We then connect to the ChipWhisperer Nano using the supplied Python module[2], and install the calibration program on the target using the built-in STM32F0 programmer. After the program has successfully been flashed, we initialise the capture by resetting the trigger signal and flushing the target buffer. We arm the scope object such that capture will begin at the first rising edge of the trigger and send a predefined SimpleSerial command to the target to begin collecting power traces on the device. The traces are then saved as NumPy array files, which can be accessed later for the model building step of the process.

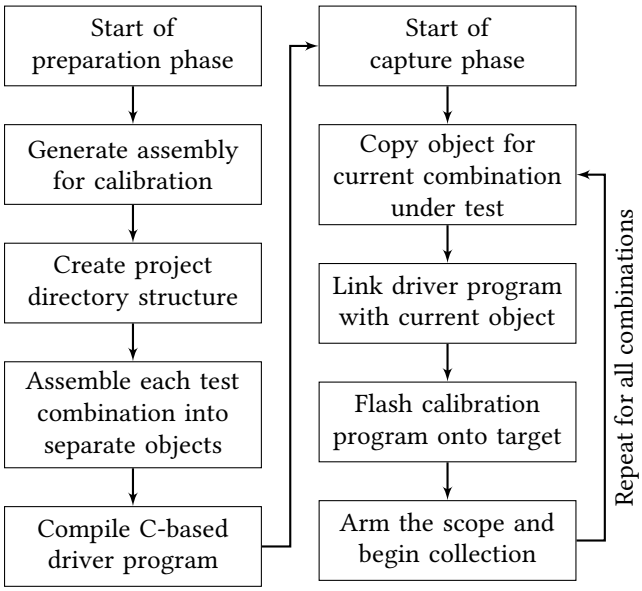

Figure 1: Preparation and capture phases of the trace collection process.

[2]Available at https://pypi.org/project/chipwhisperer/.

3.3 Workflow integration

In the Rosita workflow, we begin with a leaky implementation of a cipher and use ELMO to simulate the power leakage based on a model. Due to the modular design of the ELMO emulator [2], integration of a different power model into the Rosita workflow as defined by Shelton et al. [14] is rather trivial. Specifically, the linear regression coefficients obtained after the model building process can be used in the ELMO emulator for leakage detection in the context of the ChipWhisperer Nano with minimal configuration. We simply place the coefficient file in the ELMO source directory and update the path to the power model in the `elmodefines.h` header file before rebuilding the binary.

As the leakage analysis step in the Rosita workflow is entirely delegated to the ELMO emulator, a change in the power model, in theory, should not affect Rosita's ability to eliminate leakages through code rewrites. Though, the overall reduction in leakages may suffer if a low-quality model that fails to accurately reflect the power characteristics of the device were to be used, as such a model may lead to incomplete detection of leakages or false positives (i.e. incorrectly detecting leakages when there are none).

4 EXPERIMENTAL SETUP

4.1 Understanding the scope trigger

When performing power analysis attacks with a conventional oscilloscope, triggers are generally used for synchronising the voltage and time data, as well as for signalling the start and completion of a power trace. However, this is not the case with the ChipWhisperer Nano. Early testing with the device showed that while the trigger can be set to high to indicate the start of a trace by calling the built-in `trigger_high()`, the function `trigger_low()` does not stop the capture as expected and cannot be used to control the completion of a trace. In addition to this limitation, there currently exists no function for viewing the waveform of the trigger channel.

To better understand the behaviour of the trigger on the ChipWhisperer Nano, two probing wires were soldered each onto the GPIO7 pin (mislabelled as GPIO4 by the device manufacturer), which is used by the scope as the trigger signal, and ground pin on the target portion of the device. This allows us to observe the trigger channel on another oscilloscope with higher resolution and bandwidth. To investigate the trigger behaviour, we use a PicoScope 6404D with a Pico Technology TA046 differential probe connected to the oscilloscope, and monitor the signal as a test program with a known trigger pattern was run on the target.

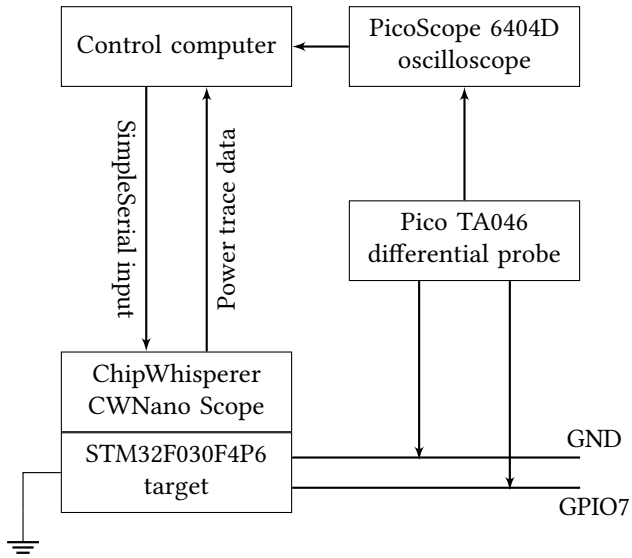

Figure 2: Trigger testing setup.

It became understood that, due to technical and likely cost constraints of the ChipWhisperer Nano, the trigger function defined in the scope object was designed to behave in a more unidirectional manner than conventional oscilloscope triggers. As normal, the scope will start the trace capture upon detecting a rising edge in the trigger signal. However, the trace will continue to be captured and only stop when the sample buffer reaches its maximum size as defined by the user. Hence, it is currently impossible to control the completion of a trace using a trigger signal.

With this limitation in mind, the correct order of using the trigger function on the ChipWhisperer Nano should be as follows:

```
trigger_setup()
<no-op delay>
trigger_low()
trigger_high()
<instructions of interest>
trigger_low()
```

Listing 2: Example code demonstrating the trigger function on the ChipWhisperer.

A delay is inserted between `trigger_setup()` and the start of the capture to allow a sufficient buffer such that the rising edge will consistently occur after all setup instructions have been completed. Without the delay, the scope may be unable to detect the rising edge and will return a timeout as a result of not receiving the signal to start the trace, causing the capture to fail entirely.

4.2 Capturing traces

To capture the calibration traces needed for building the power model for the ChipWhisperer Nano, we use the scope onboard the device to measure the power consumption of the STM32F030F4P6 target. Due to the synchronous nature of the ChipWhisperer scope [12], the required sampling rate is greatly reduced and high-quality traces can be obtained with the clock rate and sampling rate set at a 1:1 ratio. We sample every 133.3 ns at a clock rate of 7.5 MHz, with the sample buffer set to 100000 samples or the maximum size supported by the memory.

We perform the trace capture using version 5.6.1 of the ChipWhisperer Python module, on a ChipWhisperer Nano with firmware version 0.50.0, which supports improved SimpleSerial communication with the target. To ensure the produced trace is as close to the start of the instruction of interest as possible (excluding delay and setup), we also make a few modifications to the trigger setup shown in Listing 2, by directly incorporating the `trigger_high()` call adjacent to the sequence combination in the linked object, as so:

```
push {r0-r3}
bl trigger_high   // Start of trace
pop {r0-r3}
mov r0, r9
movs r0, r0
movs r0, r0
str r5, [r0,#8]
movs r0, r5
eors r0, r5
mov r0, r10
ldr r1, [r0,#24]  // ldr state set
movs r1, #0
movs r0, r0
movs r0, r0
movs r0, r0
movs r0, r0
movs r0, r0
eors r2, r3       // Insts under test
eors r4, r5
str r7, [r6]
mov r7, r7        // Delay
mov r7, r7
mov r7, r7
push {r0-r3}
bl trigger_low    // End of trace
pop {r0-r3}
```

Listing 3: Example assembly code demonstrating a closer trigger setup.

4.3 Optimising the capture

The trace collection process initially took over on average 1 hour and 30 minutes to complete for all five instruction classes. While not considered excessively time-consuming at first, the long capturing time quickly became a hindrance to the development process of the model, as any changes made to the calibration code would imply the capture would have to be redone from the beginning. To reduce the time spent on capturing, we first optimise the capture phase of the process by removing unused libraries and other residual ChipWhisperer project template files from the calibration program. This greatly lowered the complexity of the hexadecimal object file, speeding up the compilation step. The program's smaller size also subsequently sped up the flashing step, as the number of bytes required to be sent to the target is less, which adds to a total reduction in capture time of approximately 40 minutes, or a 44.4% decrease.

The process can be further optimised by moving the linking and flashing steps from the capture phase to the preparation phase. After profiling the automation code, it quickly became known that the most time-consuming step of the process is on programming the target with a newly linked hexadecimal object file, which from empirical testing, took on average 3.5 seconds per iteration to complete. To optimise this step, we compile the object file with all of the required combinations placed in separate functions before the capture phase and call different functions at run-time by communicating with the target using SimpleSerial. The effect of this is, since separate object files for each combination are no longer used, the device will only need to be programmed once, as opposed to 125 times per instruction class previously. By doing so, the total capture time can be massively reduced from taking 50 minutes to approximately only 14 minutes, leading to about a 72% decrease.

5 RESULTS

We now focus our attention on the experimental results obtained after performing the trace collection process outlined above, and the subsequent linear regression analysis for model building as detailed in Sect. 3.1 and by McCann et al. [10].

We first discuss the traces collected and the relation between the calibrating instruction and the specific location in the trace. We then present the leakages detected from a masked implementation of AES using the produced power model, comparing them with those detected using the ELMO-default STM32F0 Discovery Board power model. We then show that the detected leakages from our power model can be eliminated using the Rosita code rewrite engine, resulting in a fixed implementation of the cipher.

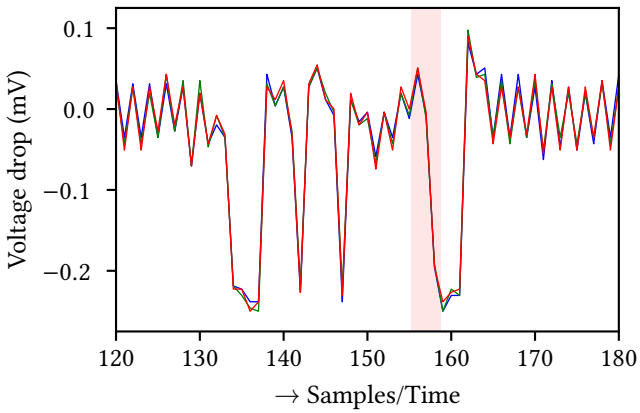

Figure 3: Power trace of calibration test 0 for `ldr`, with the instructions of interest highlighted.

The region containing the instructions of interest is highlighted as red in the figure above. As the synchronous design of the ChipWhisperer hardware allows traces to be captured at a clock and sampling frequency ratio of 1:1, we can directly equate the number of samples in the trace with the cycle count after the rising edge of the trigger signal. With this correlation in mind, we can observe that for this particular experimental setup, the calibration sequence of test 0 begins at cycle 156 and ends at cycle 158.

As the only differing instructions after setting the state for each test are the three calibration instructions, we can further confirm our location of instructions of interest by comparing the pattern of the voltage drop with another trace under the same instruction class. If the highlighted region were to be representative of some other instructions in the calibration process that are constant for each test, we should observe no change in trace pattern. To illustrate this, we now look at the trace of a more distanced test 38 for `ldr`.

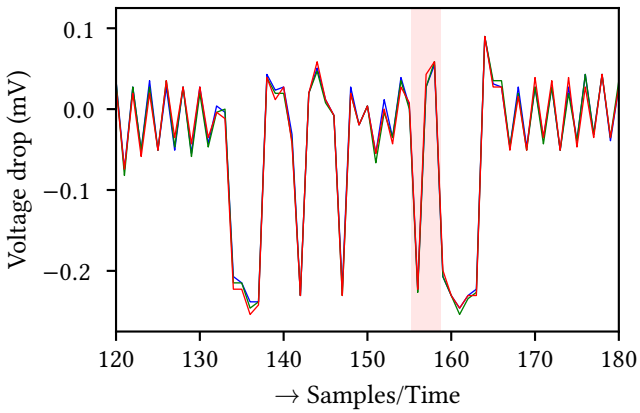

Figure 4: Power trace of calibration test 38 for `ldr`, with the instructions of interest highlighted.

We observe a difference in the power consumption between the two tests due to different calibration instructions between the two tests. In this case, test 0 had a sequence of combination `eors-eors-eors`, while test 38 had the combination `lsls-str-ldr`. With these three samples being the only major difference between the two traces, we are able to confirm that the highlighted region does correctly point to our instructions of interest.

Following the model building steps as described by McCann et al. [10], we compress the relevant power consumption values in the region to derive a single value for each test. These values are then packaged with the random inputs sent via SimpleSerial during the capture phase into a `.mat` binary data file. Using MATLAB, we perform the linear regression analysis on the compressed trace data and derive the coefficients, which can be exported into the desired model format for leakage emulation in ELMO and subsequently elimination with Rosita.

To evaluate the quality of the newly produced model, we compare the leakages detected from a masked implementation of AES (fixed-input) [14] using the ChipWhisperer Nano model, against those detected by the default ELMO model derived from the STM32F0 (30R8T6) Discovery Board. We will first focus on the *t*-test values emitted from Rosita using the default model. Note that leakages are defined as instructions corresponding to `t`-values exceeding the standard threshold of ±4.5 used in TVLA [6].

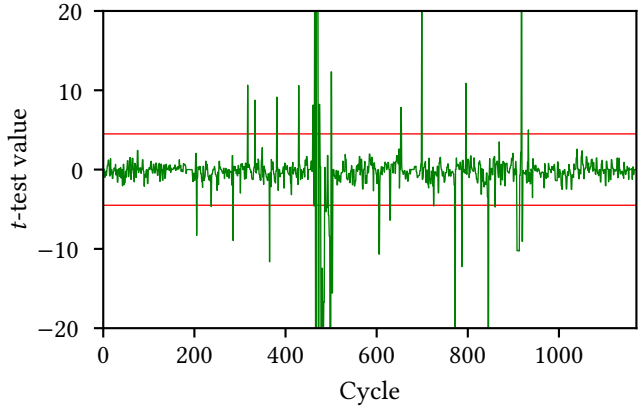

Figure 5: Leakage from masked implementation of AES using STM32F0 Discovery Board power model.

In the first round detection (i.e. before Rosita does any code rewrites) performed using the STM32F0 Discovery Board power model, we notice significant spikes in the *t*-test value starting from approximately cycle 465. There also exist other shorter span spikes surrounding cycles 770, 840, and 940. We keep these potential leakage locations in mind as we move on to those detected using the ChipWhisperer Nano power model for comparison.

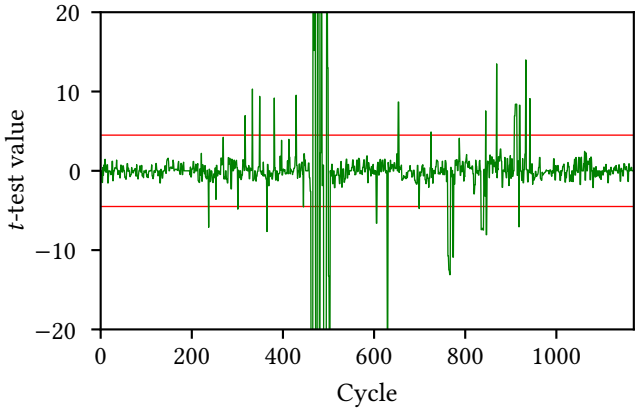

Figure 6: Leakage from masked implementation of AES using ChipWhisperer Nano power model.

From the figure above, we observe that ELMO was able to correctly emulate the same major leakages starting from cycle 465 using the ChipWhisperer Nano power model. It also correctly recognised the same leakage from about cycle 940. However, it failed to detect a few leakages near the middle of the round and is less sensitive to leakages with smaller data dependency when compared to the default model.

We then use Rosita to eliminate the above leakages. After three iterations of leakage detection and code rewrites, both implementations were able to be fixed with zero leakages being detected by each power model. We first present the `t`-test values from the fixed implementation of AES after the elimination process.

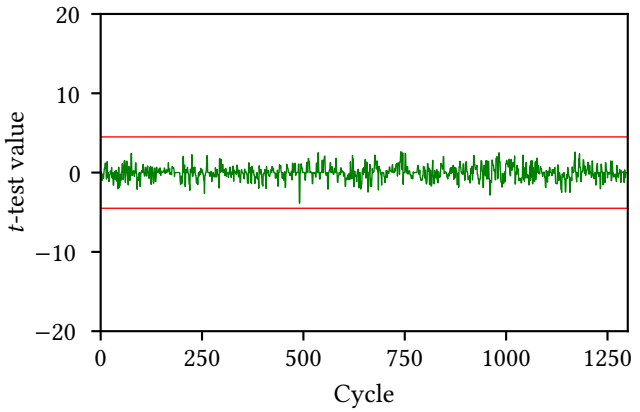

Figure 7: Fixed implementation of AES using STM32F0 Discovery Board power model.

As seen in the figure above, the `t`-test values remained within the threshold of ±4.5. While this does not imply the complete absence of leakages, this does suggest the previously detected leakages with the default model are now eliminated to a sufficient degree.

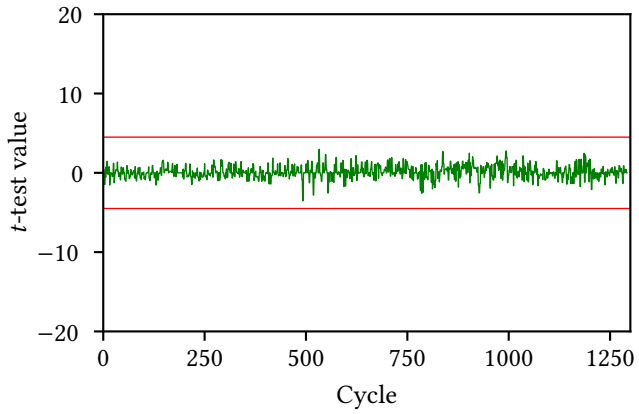

Figure 8: Fixed implementation of AES using ChipWhisperer Nano power model.

A similar case can be seen with the fixed implementation using the ChipWhisper Nano power model, indicating the successful elimination of the detected leakages.

As it stands currently, much remains to be done before the power model can be used for proper leakage mitigation. For one, the complexity and lengthiness of the trace collection process heavily limit the level of input randomness, which is necessary for an accurate power model. This can be remedied in the future with further optimisation of the collection process, specifically by removing the need to link and flash the calibration library for each test combination, thus increasing the number of traces per combination that can be feasibly captured within a set time frame.

In addition to the low input randomness, there are also data accuracy issues in the model building steps. Specifically, the specific points of interest containing the power consumption of calibration instructions on the traces may have not been trimmed correctly. This is led by the random input during calibration causing additional CPU cycles to be added when input-dependent instructions were being executed, prior to the three instructions of interest. In other words, some of the power consumption data used in the regression would have been incorrectly pointing to the no-op delay instructions instead, contributing to the overall error of the model.

6 CONCLUSION

In this work, we explored the possibility of expanding support for the Rosita code rewrite engine to ChipWhisperer Nano. By capturing power traces from the target device under test as sequences of calibration instructions are executed, then performing linear regression analysis on the collected traces, we built a power model for the power leakage characteristics of ChipWhisperer Nano.

Despite the accuracy limitations of the produced model, we demonstrated that it is possible to integrate a model of different microarchitecture into the Rosita workflow. Moreover, we showed that the power model we produced has been successful in identifying the main leakages in a masked implementation of AES, which could be eliminated by Rosita in three iterations to produce a fixed implementation of the cipher. While the trace collection methodology could be improved to allow for more complete mitigation of leakages, a better understanding of the ChipWhisperer capture process and its trigger functions has also been gained as a result of performing the model calibration, building upon the future work towards broadened automatic elimination of power leakages.

7 AVAILABILITY

A release of RositaWhisperer, containing the ChipWhisperer Nano power model built in this paper, the ChipWhisperer project files, plus all relevant trace collection and analysis scripts used can be downloaded from: https://github.com/SamillWong/RositaWhisperer.

REFERENCES

[1] Vipul Arora, Ileana Buhan, Guilherme Perin, and Stjepan Picek. 2021. A Tale of Two Boards: On the Influence of Microarchitecture on Side-Channel Leakage. (2021). Published: Cryptology ePrint Archive, Report 2021/905.

[2] E. Oswald, Valentina Banciu, Si Gao, Christos Tzotzadinis, and C. Whitmall. 2020. *White Paper: Security Testing Via Simulators.* Deliverable D2.6. REASSURE. 25 pages. http://reassure.eu/wp-content/uploads/2020/05/REASSURE_D26.pdf

[3] G Becker, J Cooper, E De Mulder, G Goodwill, J Jaffe, and G Kenworthy. 2015. *Test Vector Leakage Assessment (TVLA) Derived Test Requirements (DTR) with AES.* Technical Report. Cryptography Research Inc. 8 pages. https://www.rambus.com/wp-content/uploads/2015/08/TVLA-DTR-with-AES.pdf

[4] Si Gao and Elisabeth Oswald. 2021. A Novel Completeness Test and its Application to Side Channel Attacks and Simulators. (2021). Published: Cryptology ePrint Archive, Report 2021/756.

[5] Si Gao, Elisabeth Oswald, and Dan Page. 2021. Reverse Engineering the Micro-Architectural Leakage Features of a Commercial Processor. (2021). Published: Cryptology ePrint Archive, Report 2021/794.

[6] Gilbert Goodwill, Benjamin Jun, Joshua Jaffe, and Pankaj Rohatgi. 2011. A testing methodology for side channel resistance. Cryptography Research Inc., Todai-ji Cultural Center Nara, Japan, 15. https://csrc.nist.gov/csrc/media/events/non-invasive-attack-testing-workshop/documents/08_goodwill.pdf

[7] Paul Kocher, Joshua Jaffe, and Benjamin Jun. 1999. Differential Power Analysis. In *Advances in Cryptology — CRYPTO' 99.* Vol. 1666. Springer Berlin Heidelberg, Berlin, Heidelberg, 388–397. https://doi.org/10.1007/3-540-48405-1_25 Series Title: Lecture Notes in Computer Science.

[8] Yann Le Corre, Johann Großschädl, and Daniel Dinu. 2018. Micro-architectural Power Simulator for Leakage Assessment of Cryptographic Software on ARM Cortex-M3 Processors. In *Constructive Side-Channel Analysis and Secure Design*, Junfeng Fan and Benedikt Gierlichs (Eds.). Vol. 10815. Springer International Publishing, Cham, 82–98. https://doi.org/10.1007/978-3-319-89641-0_5 Series Title: Lecture Notes in Computer Science.

[9] Owen Lo, William J. Buchanan, and Douglas Carson. 2017. Power analysis attacks on the AES-128 S-box using differential power analysis (DPA) and correlation power analysis (CPA). *Journal of Cyber Security Technology* 1, 2 (April 2017), 88–107. https://doi.org/10.1080/23742917.2016.1231523

[10] David McCann, Elisabeth Oswald, and Carolyn Whitnall. 2017. Towards Practical Tools for Side Channel Aware Software Engineering: 'Grey Box' Modelling for Instruction Leakages. In *26th USENIX Security Symposium (USENIX Security 17).* USENIX Association, Vancouver, BC, 199–216. https://www.usenix.org/system/files/conference/usenixsecurity17/sec17-mccann.pdf

[11] National Institute of Standards and Technology. 2001. *Advanced encryption standard (AES).* Technical Report NIST FIPS 197. National Institute of Standards

and Technology, Gaithersburg, MD. NIST FIPS 197 pages. https://doi.org/10.6028/NIST.FIPS.197

[12] Colin O'Flynn and Zhizhang Chen. 2014. ChipWhisperer: An Open-Source Platform for Hardware Embedded Security Research. In *Constructive Side-Channel Analysis and Secure Design*, Emmanuel Prouff (Ed.). Vol. 8622. Springer International Publishing, Cham, 243–260. https://doi.org/10.1007/978-3-319-10175-0_17 Series Title: Lecture Notes in Computer Science.

[13] Colin O'Flynn and Zhizhang Chen. 2016. Power Analysis Attacks Against IEEE 802.15.4 Nodes. In *Constructive Side-Channel Analysis and Secure Design*, François-Xavier Standaert and Elisabeth Oswald (Eds.). Vol. 9689. Springer International Publishing, Cham, 55–70. https://doi.org/10.1007/978-3-319-43283-0_4 Series Title: Lecture Notes in Computer Science.

[14] Madura A. Shelton, Niels Samwel, Lejla Batina, Francesco Regazzoni, Markus Wagner, and Yuval Yarom. 2021. Rosita: Towards Automatic Elimination of Power-Analysis Leakage in Ciphers. In *Proceedings 2021 Network and Distributed System Security Symposium*. Internet Society, Virtual, 17. https://doi.org/10.14722/ndss.2021.23137

[15] François-Xavier Standaert. 2019. How (Not) to Use Welch's T-Test in Side-Channel Security Evaluations. In *Smart Card Research and Advanced Applications*, Begül Bilgin and Jean-Bernard Fischer (Eds.). Vol. 11389. Springer International Publishing, Cham, 65–79. https://doi.org/10.1007/978-3-030-15462-2_5 Series Title: Lecture Notes in Computer Science.