REASSURE

Project ID: 731591, Horizon 2020
`http://www.reassure.eu`

REASSURE

Deliverable D2.4

Report on Instruction Level Profiling

Editor:	E. Oswald (BRI)
Deliverable nature:	R
Dissemination level: (Confidentiality)	PU
Delivery date:	June 30, 2018
Version:	1.0
Total number of pages:	17
Keywords:	evaluation, automation, profiling, side-channels

Horizon 2020
European Union funding
for Research & Innovation

This project has received funding from the European Union's Horizon 2020 research and innovation programme under grant agreement No. 731591.

Executive summary

This document reports on interim progress of the REASSURE consortium towards methodologies to create representative instruction-level profiles of microprocessors of at least medium complexity.

We begin by reviewing possible approaches to modelling leakage, ranging from detailed transistor level simulations derived from back-annotated netlists, through 'profiled' models fitted to sample data, to simulations based on simplifying assumptions about typical device behaviour. We identify the requirements for our planned REASSURE simulator, which will need to be able to process source code as input and should be flexible with respect to the (profiled or unprofiled) models by which it produces leakage predictions.

Next we summarise a range of existing tools from industry and academia, including one (ELMO) which we argue best sets the precedent that we plan to follow for REASSURE. We explain the modelling procedure used in the construction of ELMO and report on the results of our own model-building endeavour using data from an alternative implementation of the same board (an ARM Cortex M0). In particular, we show that there are broad similarities between the implementations, but some differences that will require attention if models are expected to port from one to another.

The next steps for REASSURE will be to explore improvements to the model building procedure, as well as its extension to more complex devices, in order to eventually release our own simulator tools to the wider side channel research and evaluation community.

List of authors

Company	Author
BRI	E. Oswald
BRI	C. Whitnall
BRI	S. Gao

Other contributions gratefully received from

Company	Author
NXP	V. Verneuil

Revision history

Revision number	Date	Comment
1.0	June 2018	Final Public Version

Contents

1 Introduction

The overarching goal of work package (WP) 2 is to create flexible evaluation methods that are usable by non-domain experts. This can be achieved by addressing a number of objectives. The first objective is to research shortcut formulae that enable early statements about the security of an implementation against some side-channel attacks. The second objective is to develop some generic leakage assessment methods and to assess the feasibility of automating some evaluation steps/techniques such that they could be offered as a remote (e.g. cloud-based) service. The third objective is to create a robust statistical methodology to characterise existing devices with the aim of building accurate leakage simulators. The fourth objective is to research the types of statements that can meaningfully be made following the automated analysis of simulated leakages (i.e. the combination of the second and third goal).

This deliverable is concerned with the third of these objectives. For this purpose we first consider simulation approaches and survey existing simulators focusing on their pros and cons, in particular with regards to their integration into a tool chain that is usable by non-experts. Then we consider in more detail a specific approach which we deem to be promising for our goal.

2 Leakage Simulation Approaches

Simulation and emulation are well known principles in the context of developing (complex) systems. There is a considerable overlap between the terms simulation and emulation in general because both refer to the idea of imitating some observable behaviour of a system. Within the context of computer science, the word emulation tends to be used when referring to the imitation of some higher-level feature. In particular, in the context of programming an instruction set emulator of a target device produces the same behaviour as that device on an instruction/register level, without necessarily reproducing the concrete implementation of instructions on that device. In contrast, the term simulation is used when referring to the imitation of some lower-level behaviour. For instance there exist a number of power simulation tools that are based on using analytical models of gates together with some characterisation of the underlying technology.

A leakage simulator will, as we argue in this deliverable, most likely require device emulation combined with power/EM simulation (we do not consider the simulation of timing or cache behaviour in the context of REASSURE). Device emulation is a well developed area, and there are a number of emulators for different architectures available (for instance, [19, 13, 2, 29]). Leakage simulation is less well developed and we will focus on this aspect in the rest of this section. For this purpose we look at how leakage simulation could be achieved at the different architectural levels.

2.1 Simulations at different levels

2.1.1 Transistor level simulation

Any circuit is eventually mapped to a network of transistors. The power consumption of such transistors can be modelled via known (differential) equations, but these require information about the technology that a chip will be built in. SPICE (Simulation Program with Integrated Circuit Emphasis) [18] is an open source analogue electronic circuit simulator. It is the “ancestor” of many different commercial tools. SPICE takes a (back annotated) netlist of transistors as input. A netlist describes how the transistors in a circuit are connected. Back annotation refers to the process of “augmenting” a netlist with more accurate information (such as delays of various sorts). Although the analogue modelling of individual transistors enables a very accurate description of their power behaviour, more complex differential effects that may occur as a result of the placement of components on a circuit cannot be captured.

2.1.2 Gate level simulation

Gate level simulations are also based on (back annotated) netlists. The greater the amount of place and route information which is available and included (via back annotation) into the netlist, the better the resulting power simulation. For the simulation, the number of transitions in each gate is counted and weighted, according to the information in the netlist. The sum over all weighted transitions is then an approximation for the instantaneous power of the circuit. If there is no information about how to "translate" a transition to the power consumption, then one can simply count the number of transitions (with equal weights). This is often referred to as a "toggle count model".

2.1.3 Behavioural level simulation

The two previous levels are the lowest at which a circuit can be described. The next level up is often referred to as the behavioural description level. On this level, there is no information about the placement of circuit elements and the routing of signals between them. One only has access to the behavioural description of the components. This description may be in the form of low level machine code or micro code, instruction/assembly code, or higher-level code (such as C).

There have been a number of works by Tiwari et al. [26, 27, 14] on the topic of estimating the average power consumption over a set time (in other words, the energy) early on in the design process. Because energy estimation research was mainly concerned with small and thus less complex devices, for which assembly instructions directly map to machine instructions which are not further decoded into micro instructions, these works tended to emphasise models build at the instruction/assembly level. The current interest in making energy consumption information available to the software developer in order to facilitate more energy efficient code has prompted further recent work in this area [10, 9, 20, 17]. By and large energy estimates are derived by instrumenting the recording of the average power of instructions (or sequences of instructions) using some lab setup with an actual device.

In the context of side channel analysis researchers recognised the value of simulations early on and many papers produce results based on simulated traces. For such simulations, researchers use a higher level description of a cipher (often in C) and then apply either a standard power model (e.g. Hamming weight, Hamming distance) or a profiled power model (e.g. the power modelled via a Gaussian distribution with parameters estimated from the corresponding instruction/operation of a real device). Evidently, using a standard power model only gives a rough estimate, whilst utilising a profile acquired from an actual implementation implies that the analysis is specific to a particular piece of code/sequence of instructions and thus cannot be applied to arbitrary sequences of code.

2.2 Requirements for a REASSURE simulator

The clear goal of WP2 of REASSURE is to do the necessary technical work to develop tools (in WP3) that are accessible to non-domain experts in the context of developing more secure software implementations of cryptography. To facilitate the uptake of our tools, we aim to release (wherever possible) some open source prototypes. Whilst the focus of this deliverable is on the methodology to determine the actual leakage models for a simulator, these need to fit into the simulator tool itself. Hence before settling on any methodology for the creation of leakage models, we need to spell out more precisely what matters for the REASSURE simulator. With this in mind, there are a number of technical requirements for a leakage simulation tool, which we first list and then discuss in more detail:

- it must accept inputs (i.e. source code) that are suitably close to what is actually executed on the target platform;
- it must include some standard analysis techniques that require minimal user interaction;
- it must be capable of relating identified leaks to the inputs (i.e. the source code);
- it must be modular (different emulators, leakage models, trace formats, etc).

2.2.1 Suitable inputs

In a typical software development flow, a program that is supplied in a high level language is compiled down into some lower level language that maps more or less immediately to actual machine instructions. In the context of implementing side channel resilient cryptography, the standard approach is to code everything in an assembly language to avoid potentially detrimental optimisations of the compiler. In this context, the implementation of countermeasures is then a very manual process that requires a side channel aware expert developer. Clearly a more elegant and accessible approach would be to enable developers to write in a higher level language, and then utilise the compiler to create side channel resilient code. To achieve this, the compiler would need some information about the leakage properties of the device that is to run the high level code it compiles. With reference to our previous discussion of simulation on different levels, it seems that the correct level to choose in our context is behavioural simulation: we cannot assume access to the netlist of the processor. Because we will focus on processors with medium complexity we can assume that assembly instructions translate more or less immediately to machine instructions. As a consequence our envisioned focus (as sketched in the REASSURE Description of Work) of using assembly code seems a good candidate as input.

2.2.2 Basic leakage analysis

To help non-expert software developers a simulator needs to not only produce accurate simulated traces but also have some basic tools to evaluate those traces or indeed the input code itself. The latter point could be tightly integrated with the compiler itself and is not immediately related to leakage simulation. Hence we will consider this at a later stage and focus for now on the components that tie in directly with the leakage simulation part. There are two basic tools that one can use for evaluating leakage traces. In a known key setting one can use standard statistical tests such as a correlation analysis (utilising the available leakage profiles) to determine specific key dependent leakages in traces. Such an approach is called specific because one has to choose a particular target intermediate value. Doing this could be automated (e.g. one could extract all explicit intermediate variables from the assembly code and choose them as target intermediate values). One could also use a non-specific approach where the statistical distance between a set of leakage traces associated with a fixed input and a set associated with randomly varying inputs is computed. Both approaches feature in the literature of leakage detection, which we have outlined in more detail in deliverable 2.2.

2.2.3 Relating leaks to source code

Because it is not currently feasible (given the state of the art) to let a compiler do all the work that is required to make a piece of code side channel resilient, our simulator should support the next best option: to report found leaks to the software developer in some useful way. A realistic and meaningful option for this task is to link leaks (identified in simulated traces) back to the code that gave rise to them. If we were to build the simulator using assembly instructions as input (as suggested before), then each point in a leakage trace would be associated with an instruction, and would thus satisfy this requirement. It is somewhat harder to propagate this information upwards towards higher level languages such as C because the compiler may make significant changes to C code in its compilation down to assembly.

2.2.4 Modularity

There are a number of device emulators available (both commercial as well as open source). Furthermore there are a number of different devices, featuring different leakage models. Finally, to enable potentially further or more sophisticated trace analysis it should be possible to write out simulated traces in some widely used trace formats. This means that it would be desirable to have a reasonably modular design for the overall tool.

3 Existing Leakage Simulators

The appeal of leakage simulation was evident already during the early days of side channel research: when proposing a novel analysis method it is necessary to demonstrate its efficiency across many different use cases. Instrumenting many different devices (and for a range of different signal-to-noise ratios) is cumbersome and time consuming, and thus researchers were motivated to look at alternatives such as running simulations on different (artificially generated leakage models) and by adding different amounts of Gaussian noise.

Another area of interest for simulations was the evaluation of hardware implementations. In the context of hardware it is even more crucial to make statements early on in the design, because of the cost implications of having to tape out multiple test devices for evaluations. Consequently, the development and use of tools to estimate instantaneous power at the transistor and gate level was of considerable interest; a previous EU funded project called SCARD devoted some time to this task [1].

We now run through a list of simulators that we were able to find information for and review their pros and cons, asking in each instance whether their leakage simulation approach (among other characteristics) is suitable for the purposes of REASSURE.

3.1 Industrial Tools

3.1.1 PinPas

Pinpas (Program Inferred Power Analysis Simulator) [5] is one of the earliest tools to be written specifically as a simulator for use in side channel analysis. It was written in Java and aimed to simulate simple leakage from small 8-bit microcontrollers. The leakage model itself was chosen *a priori* (specifically, it was the Hamming weight) rather than being derived from any concrete device. It was not open source and is no longer available.

3.1.2 Tools from the SCARD project

Within the context of the EU funded project SCARD a number of hardware simulation approaches were investigated and implemented. Two prototype power estimators were implemented that both operate at the gate level. The first one was a stand-alone tool based on value change dumps (VCDd), which are output files of typical hardware tool chains; the other was directly embedded into Mentor's ModelSim via the PLI interface [1].

Within the project they also developed a methodology to model the effect of the IC package and bonding wires on the power consumption. A follow up paper by Kirschbaum and Popp [12] attempted to evaluate the quality of the toggle-count power models by presenting a visual comparison of real and simulated traces as well as a comparison of DPA results. Their conclusion was that the models were good. The scope of the comparison was however extremely limited both in terms of the circuits they compared and the (lack of) statistical methodology.

3.1.3 Riscure Inspector

The Inspector toolbox [22] contains some functionality for high level power simulation: given a high level piece of code, and a power model, a tool will produce traces based on applying the given leakage function to intermediates that are specified in the high level code. It does not take information about an actual target architecture into account.

3.1.4 esDynamic

esDynamic is a set of tools to enable the analysis of cryptographic implementations [6]. Included is the ability to emulate binary code on a number of platforms and to perform some standard side channel analysis on these emulations. There does not seem to be any actual characterisation of the emulated platforms included.

3.1.5 Virtualyzr

The Virtualyzr tool from Secure-IC is a HDL simulator [24]. There is no detailed public information available for it, but from the website one can gather that it can be used with various types of descriptions that are available within a typical hardware design flow.

3.2 Academic tools

Thuillet et al. [25] report on a simulator that is similar to PinPas and is designed for the analysis of smart cards. It takes in a program in a high level language and works using a standard power model.

Debande et al. [4] emphasise the importance of (and the complexities involved in) deriving realistic leakage models empirically. They fit linear models in function of the state bits and state transitions using the techniques of linear regression.

Gagnerot [8] reports on writing a simulator for side channel and fault attacks as part of his PhD thesis. The simulator was designed for a 16-bit RISC architecture under an NDA. It was based on using standard power models (HW and HD).

Reparaz [21] suggests to essentially use a power simulation with some leakage detection testing to spot problems in masked implementations as an alternative to more costly verification tools.

In Bristol we have used an 8051 emulator together with a standard power model (HW) for years in our Applied Security unit. Every student receives a compiled binary that includes a 'secret key' and can then interact with the leakage simulator to generate side channel traces to perform attacks.

ELMO [16] is a simulator that takes in the Thumb assembly of an algorithm and maps the data flow via power models that have been provided especially and are intended to be specific to a processor architecture: in the paper, a considerable amount of effort is inveseted in explaining how such models can be efficiently obtained. In particular, the featured approach is based on model building rather than model estimation (i.e. rather than beginning with a fixed model and simply estimating the parameters, one explores a variety of candidate model configurations and makes explicit decisions about the functional form based on the statistical significance or otherwise of tested alternatives). The underlying M0 emulator is open source [29], and thus the tool is also available as open source [15].

Veshchikov proposes a number of tools in his PhD thesis; these are called Silk, Ascold and Savrasca [28], each working on a different level of abstraction. Savrasca can take in custom models and is capable of dealing with an AVR architecture because it is essentially based on SimulAVR.

MAPS is the most recent tool [3]. It also focuses on an M class processor from ARM. Since the development of ELMO, ARM has enabled researchers to get access to a (semi) obfuscated RTL description of the M0 and M3 processor. MAPS uses this information to model the pipeline of an M3, which is an important source of HD leakage. The emulator is written from scratch in a mix of C and C++ and uses standard power models (HW, HD).

4 The REASSURE Modelling Approach

Nearly all of the existing simulators consist of an already available instruction set emulator (for a specific class of devices), which is used to extract intermediate values that are then passed into a standard power model. Thereby, for each generated intermediate value, a leakage point is created. The vector of all such leakage points for a given set of inputs forms a single simulated leakage trace.

Using a standard leakage model such as the Hamming weight risks ignoring relevant leaks that are present in a real device. This risk would be small on a device with a very simple architecture, for instance a typical 8-bit micro-controller. On any more complex device, that features a pipeline or has some instruction caches, the risk would be considerably greater.

Toggle-count models such as that used in the SCARD project are not feasible for commodity processors because access to the full processor description is not available. For the M0/M3, ARM provides the 'DesignStart' licences. These make the RTL description available: all information that is required for simulation is available as well as sufficient detail and documentation about the interfaces. However, many internals are obfuscated and thus any netlist based power simulation may have to be performed for the entire RTL, which would be too slow. The MAPS simulator utilised the availability of some information about the M0/M3 to model the pipeline behaviour only. Hence it only models the behaviour of one specific component of the processor, which is not satisfactory for our purposes.

The approach taken for ELMO focused on model building in what the authors described as a 'grey box' setting, in which exact and detailed access to the chip architecture is not available, but partial knowledge is put to good use. Different candidate models, informed by known characteristics of the device, are fitted and compared using statistical tests in order to learn the correct form of the model (rather than simply obtaining parameter estimates for a given assumed form). Thus, without relying on precise architectural details, it is possible to produce models that are nonetheless far more sophisticated than simple Hamming weights or Hamming distances. The original research also takes into account board and process effects.

Given our goals, the methodology of ELMO seems the most appropriate for REASSURE. An open question, though, is whether a model derived from a specific instance of an architecture (the ELMO paper uses an implementation of an M0 by ST) can be assumed to hold true for another instance of the same architecture (i.e. an implementation of an M0 by a different manufacturer).

After reviewing the ELMO model building methodology, we discuss the findings of our own application of this same methodology to an NXP implementation of the M0, and how the resulting models compare with those obtained previously.

4.1 Building functional power models

The models integrated into ELMO are built according to a 'grey box' approach – that is, they do not rely on detailed knowledge of the architecture (which, as is the case for Cortex-M devices, is usually not easy to obtain) but they do take advantage of basic features that are publicly available. For example, Figure 1, redrawn from [7], depicts the architectural components of an ARM core in simplified form. The CPU comprises an arithmetic-logic unit (ALU), a hardware multiplier, and a (barrel) shifter. Two buses feed from the register banks into the ALU, one of which also connects to some data in/out registers, while a third connects the ALU output back to the register banks.

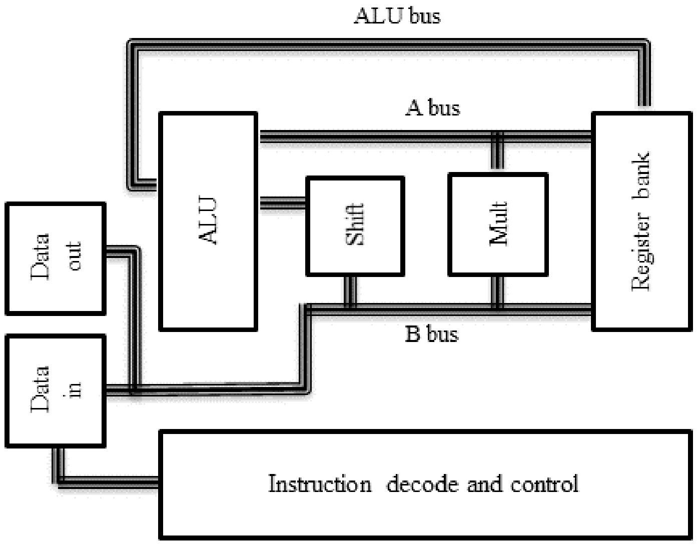

Figure 1: Simplified ARM CPU architecture (redrawn from [7]; re-used from [16] with permission) for a 3-stage pipeline architecture.

The models are built, tested and modified in an incremental manner using the statistical techniques of linear regression and the F-test, which we now briefly review:

Linear regression (first proposed in the context of side channel profiling by Schindler et al. in 2005, to our knowledge [23]) entails writing down an expression of the form $\mathbf{y} = \mathbf{X}\boldsymbol{\beta} + \boldsymbol{\varepsilon}$ where $\mathbf{y}$ is a vector of observed values of the dependent variable (that is, the quantity one seeks to predict), $\boldsymbol{X}$ is a matrix comprising of corresponding column vectors of independent (a.k.a. explanatory) variables, $\boldsymbol{\beta}$ is the vector of (unknown) coefficients that must be learned from the data, and $\boldsymbol{\varepsilon}$ is a vector of error terms which is modelled as (typically Gaussian) noise. 'Fitting' the model entails estimating $\boldsymbol{\beta}$, for example by minimising the sum of squares of the difference between the model predictions and the raw data (i.e. the residuals; this popular method is called 'ordinary least squares' (OLS)).

The F-test is a statistical hypothesis test which can be used to decide whether or not (subsets of) the explanatory variables on the right hand side of the model equation really do (jointly) influence the dependent variable on the left hand side enough to be worth including. Suppose we have two models A and B, such that B has the same explanatory variables as A plus some additional variables that we would like to test (i.e. A is 'nested' within B). The F-statistic is computed via the residual sums of squares (RSS) of each model: $F = \frac{\left(\frac{\text{RSS}_A - \text{RSS}_B}{p_B - p_A}\right)}{\left(\frac{\text{RSS}_B}{n - p_B}\right)}$, where p_A, p_B are the numbers of parameters in models A and B and n is the sample size. Under the *null hypothesis* that the new terms have no effect (as long as it reasonably holds that the residuals are independent and normally distributed with a common variance), F has an F-distribution with $(p_B - p_A, n - p_B)$ degrees of freedom. This then provides a criterion for deciding whether or not the new terms should be included:

if, for a given significance level[1] (the ELMO paper opts for $\alpha = 5\%$ throughout) the observed value of F is larger than the 'critical value' of the $F_{p_B - p_A, n - p_B}$ distribution, then we conclude that the tested terms contribute significantly to the model and should be included. If not, we say that there is insufficient evidence to reject the null hypothesis of no effect, and leave the tested terms out of the model. Note that when the number of new terms tested is one, the F-test is exactly equivalent to the t-test.

Using linear regression to build models for side channel leakages typically entails putting side channel measurements on the left hand side and explanatory variables summarising the algorithmic processes associated with those measurements (e.g. intermediate data, current and adjacent instructions) on the right hand side.

To build models for ELMO, the power consumption of the device is measured (with fine granularity) during the repeated execution of known sequences of assembly code on different (randomly generated) input data. The clock cycle corresponding to the target instruction is identified by inspection and compressed to a column vector (in [16] the index with the maximum mean is chosen; another option might be to sum all the points in the cycle) – this then forms the (univariate) dependent variable. The candidate explanatory variables are derived from the assembly sequences and known inputs associated with each observation of the power consumption. The strategy that follows is geared around simplifying wherever possible in order to facilitate as much *meaningful* complexity as possible given the data and computational resources available.

To this end, it is desirable, early on in the process, to restrict the number of distinct instructions needing to be profiled. If this can be done without ignoring important features of the excluded instructions, it becomes more feasible to eventually build models that take previous and subsequent instructions into account, as it reduces the number of possible *triplets* for which data need to be sampled and analysed. McCann et al. [16] begin with a set of 23 instructions deemed most essential for cryptographic applications and build separate linear regression models for each these in terms of their 32-bit operands and their bit-wise transitions from the preceding operands. They then perform a hierarchical clustering analysis on the OLS-estimated coefficients and find that the best arrangement (according to the silhouette index for cluster quality) corresponds to a grouping of instructions into 5 categories: ALU operations, shifts, loads, stores and multiply. This arrangement is strikingly aligned with what might be intuitively expected by inspection of Figure 1. Thus McCann et al. are able to empirically confirm that, rather than profile all 23 instructions, it is reasonable to profile a single representative from each group, increasing the potential for other forms of model complexity.

Having chosen `eors`, `lsls`, `str`, `ldr` and `muls` as the five representative instructions for the group of 23, the authors then explore various incremental extensions to their basic data-dependent models. Intuition about the architecture is used to guide the order and grouping in which additional variables are tested (via the F-test) for their contribution to the leakage; those which are found (jointly) significant are included from then on.[2] Requiring evidence for the relevance of new explanatory variables helps towards the goal of a sensible trade-off between comprehensiveness and complexity in the final models.

Since ELMO is designed to simulate side-channel leakage, with a particular interest in the extent to which it 'leaks' information about the supposedly secret data flow, McCann et al. are primarily interested in candidate explanatory variables which interact with the data in order to exacerbate the data-dependency of the power consumption. Interaction terms are incorporated in linear regression models by estimating a coefficient for the contribution of the *product* of two or more individual terms. Note that it is advisable to fit all corresponding individual terms even if one is not interested in those directly, as interaction effects fitted on their own may operate as proxies for the missing main effects, thereby appearing more influential than they are.

The variables which appear in the final models for the ST Cortex M0 board are:

- Bits of operand 1 and 2.
- Bit-wise transitions between operands 1 and 2 and the operands to the previous instruction.
- Previous instruction (`lsls`, `str`, `ldr` or `muls`, fitted as an indicator variable with `eors` as the baseline category).

[1]That is, the rate of false positives deemed acceptable under the circumstances.

[2]This is deemed preferable to automated procedures such as stepwise regression [11], which are prone to over-fitting (and to understating the uncertainty of the finalised models, unless care is taken), and which bypass intuition to the point where the resulting configurations are unlikely to be readily interpretable.

- Subsequent instruction (`lsls`, `str`, `ldr` or `muls`, fitted as an indicator variable with `eors` as the baseline category).
- Previous instruction multiplied by the Hamming weight of operand 1, and likewise for operand 2.
- Subsequent instruction multiplied by the Hamming weight of operand 1, and likewise for operand 2.
- Previous instruction multiplied by the Hamming distance between operand 1 and operand 1 of the previous instruction, and likewise for operand 2.
- Subsequent instruction multiplied by the Hamming distance between operand 1 and operand 1 of the previous instruction, and likewise for operand 2.
- `lsls` and `muls` models only: Pair-wise interactions between bits of operand 1, and likewise for operand 2.

The models also all include an intercept term. However, because the different instructions within clusters have different level power consumptions, and because ELMO is geared towards simulating a *proportional approximation* of the data-dependent leakage (rather than a level-accurate approximation which is much less likely to be robust between boards), the intercept term is not provided to ELMO. Omitting it is equivalent to re-locating the leakage around zero, and does not impact on the features of interest.

With this approach as our starting point, the REASSURE project is working to better understand the scope of the particular models derived by McCann et al. [16], to refine and extend the form of the models where possible, and to re-fit the models for other devices of interest exhibiting sufficiently different leakage behaviours. We begin by repeating the analysis of [16] for an alternative implementation of the Cortex M0.

4.2 Comparing the functional models of an ST M0 and an NXP M0

The original ELMO paper [16] focused on an implementation of the Cortex M0 produced by ST Microelectronics. In order to explore the extent to which the resulting models generalise to other versions of the same board, we reproduce the model building process using measurements acquired from an NXP LPC1114FN28.

The 23 instructions profiled by McCann et al. separated neatly into five intuitive groupings (ALU / Shifts / Loads / Stores / Multiply) according to their cluster analysis of the ST board leakages [16, Table 5]. The same analysis of those 23 instructions on the NXP board produced a near but not exact match: ALU instructions on immediate values (along with `cmp`) formed two additional subgroups distinct from regular ALU instructions, and `strb` formed a group on its own distinct from the other store instructions.

The functional form of the data-dependent leakage for each representative instruction is depicted in Figure 2. Each line represents the coefficients on the bits and bit flips of the two 32-bit operands, as estimated via ordinary least squares. Comparing this with the equivalent analysis for the ST board (Figure 3, re-used with permission from [16, Figure 7]) again reveals broad similarity with some differences – the most substantial being the inversion of the roles of operands 1 and 2 in the `muls` instruction. Use of the ST-based model to simulate an NXP board would ideally take this difference into account in order to avoid a misleading impression.

A particular challenge of modelling the NXP board is that the location and length of the leakage associated with the 23 different instructions varies considerably. Figure 4, for example, shows the power consumption as the `and` and the `orr` instructions execute. The observable activity begins earlier in the case of the former, and seems to spread over a larger number of clock cycles. This increases the difficulty of compressing the instruction leakages to a single, representative point for the purposes of estimating the ELMO-style models.

However, the different instructions are not the only source of timing variation. One of the most intriguing observations made during our analysis so far is that the mere code layout can sometimes affect the actual execution time. By way of specific example, if we copy the same code twice, the first time starting with the address “1be” and the second starting with “1c4” (see first row of Table 1), the actual running time would be 5 cycles in the first case and 7 cycles in the second. This timing difference has been verified with both the SysTick timer and with traces-based leakage analysis.

Since none of the documents from ARM or NXP explicitly explain the cause of this behaviour, we are obliged to resort to guesswork. From our experiments, it seems that these extra 2 cycles appear whenever the code gadget comes across some address ending with hex “c”. A reasonable guess would be that, when executing

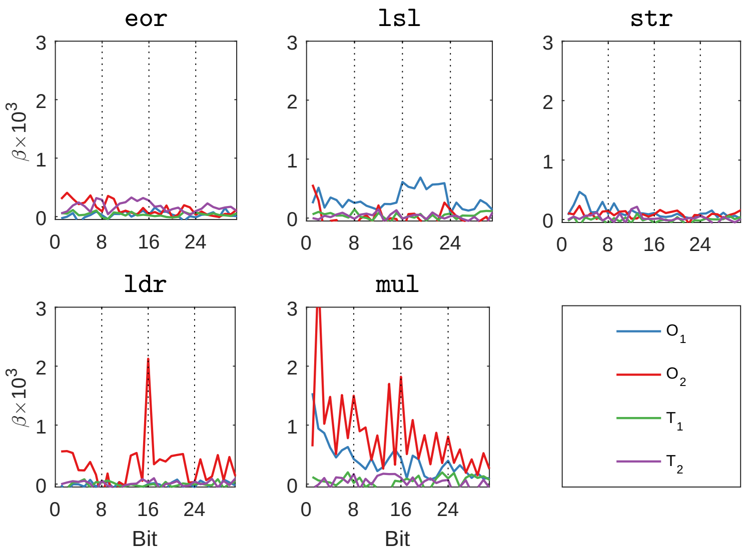

Figure 2: Estimated coefficients on the model terms for each chosen representative M0 instruction executing on the NXP board. Equivalent to the analysis of the ST board presented in [16, Figure 7].

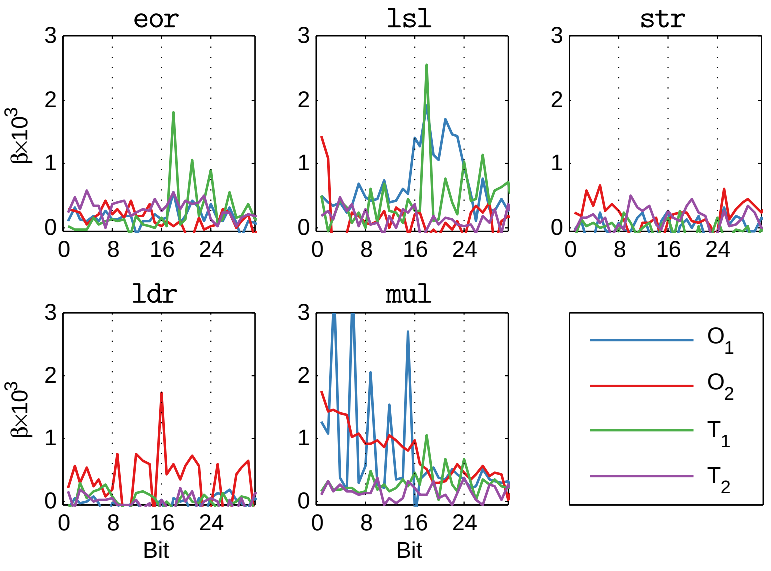

Figure 3: Estimated coefficients on the model terms for each chosen representative M0 instruction executing on the ST board. Re-used with permission from [16, Figure 7].

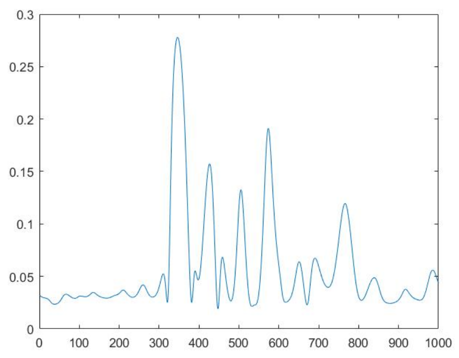
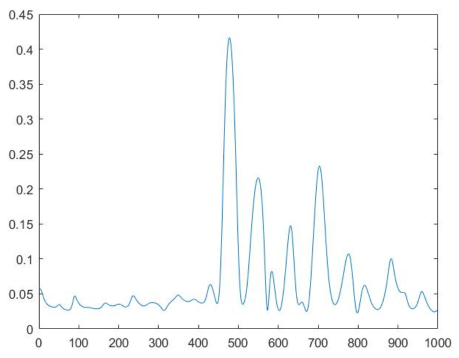

Figure 4: Measured power consumption as `and` and `orr` instructions execute.

5 Cycles	**7 Cycles**
1be : str r1, [r0, #0]	1c4 : str r1, [r0, #0]
1c0 : b.n 0x1c8	1c6 : b.n 0x1ce
1c2 : nop	1c8 : nop
1c4 : nop	1ca : nop
1c6 : nop	1cc : nop
1c8 : str r2, [r0, #0]	1ce : str r2, [r0, #0]

Table 1: Examples of code sequences which take different numbers of clock cycles to execute (as counted via SysTick) depending on the address alignment.

an instruction in address "c", since ARM M0 still fetches the instruction in a 32-bit manner, the CPU will come across some obstacle that stalls the pipeline for 2 cycles. The main suspect is the Flash, which is known to be slower than RAM. In other words, if the next instruction line lies in Flash, the CPU may stall the pipeline for 2 cycles and fetch a new instruction line (16 bytes) to RAM. We further verify our guess by forcing the core to pre-fetch these two gadgets to RAM: if it is running from RAM, both gadgets take 5 cycles to finish, which is consistent with both ARM and NXP's documents.

From a side channel perspective, these extra 2 cycles leak considerable information about the execution flow. In cases where there is a branch (although cryptographic implementations try to avoid these), no matter how carefully balanced the code is, the execution time still reveals which branch is chosen. The situation might be worse if the attacker can observe the power consumption: as we can see in Figure 5, there is a distinguishable peak on the trace which indicates when this "instruction pre-fetch" happens. Even with a low-cost oscilloscope, an attacker can now analyse this roadmap and track back to each executed instruction via the ASM code. Although we do not have a concrete example of a possible attack, to some extent, such "pre-fetch peaks" make the executed code transparent to attackers.

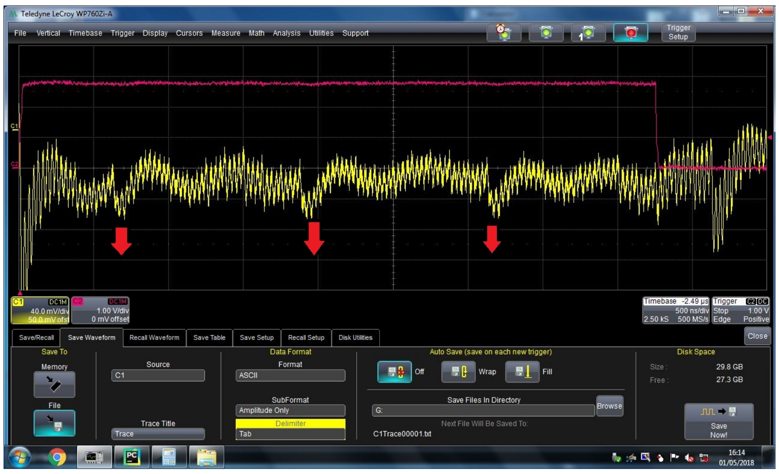

Figure 5: A screenshot from the oscilloscope showing that the instruction pre-fetch leaves a peak on the power trace.

4.3 Conclusions

Of all the options known to us, we conclude that the methodology used to produce instruction-level models for the ELMO simulator [16] is the best-suited for the purposes of the REASSURE project. However, our exploratory analysis reveals that even simple devices can vary by manufacturer, implying that (at the very least) some manufacturer-specific information will be required in order to port a model between boards, if not (at worst) a whole new model. We also observe the potential for considerable variation in execution flow and memory access which are likely to make it more challenging to pinpoint instructions in the traces produced by some devices. Questions about how to proceed with the model building as the number of distinct instruction clusters increases (as observed for the NXP) or when the device or its leakage prove more complex in other ways (for example, in the case of a longer pipeline) also remain open and in need of addressing. It is part of the ongoing work of the REASSURE project to explore these challenges as we prepare our own simulator tools for release to the wider side channel research and evaluation community.

References

[1] M. J. Aigner, S. Mangard, F. Menichelli, R. Menicocci, M. Olivieri, T. Popp, G. Scotti, and A. Trifiletti. Side channel analysis resistant design flow. In *International Symposium on Circuits and Systems (ISCAS 2006), 21-24 May 2006, Island of Kos, Greece*. IEEE, 2006.

[2] ARM Limited. ARMulator: ARM Instruction Set Simulator. `https://sourceforge.net/projects/armulator/`, accessed June 2018., 2015.

[3] Y. L. Corre, J. Großschädl, and D. Dinu. Micro-architectural power simulator for leakage assessment of cryptographic software on ARM cortex-m3 processors. In J. Fan and B. Gierlichs, editors, *Constructive Side-Channel Analysis and Secure Design – 9th International Workshop, COSADE 2018, Singapore, April 23-24, 2018, Proceedings*, volume 10815 of *LNCS*, pages 82–98. Springer, 2018.

[4] N. Debande, M. Berthier, Y. Bocktaels, and T.-H. Le. Profiled model based power simulator for side channel evaluation. Cryptology ePrint Archive, Report 2012/703, 2012.

[5] J. den Hartog, J. Verschuren, E. P. de Vink, J. de Vos, and W. Wiersma. PINPAS: A tool for power analysis of smartcards. In *International Conference on Information Security (SEC2003)*, volume 250 of *IFIP Conference Proceedings*, pages 453–457. Kluwer, 2003.

[6] eshard. esDynamic. `https://www.eshard.com/product/esdynamic/`, accessed June 2018.

[7] S. Furber. *ARM System-on-Chip Architecture*. Addison Wesley, 2000.

[8] G. Gagnerot. Étude des attaques et des contre-mesures assoccées sur composants embarqués. PhD thesis, Université de Limoges, 2013.

[9] K. Georgiou, S. X. de Souza, and K. Eder. The iot energy challenge: A software perspective. *IEEE Embedded Systems Letters*, 8 2017.

[10] K. Georgiou, S. Kerrison, Z. Chamski, and K. Eder. Energy transparency for deeply embedded programs. *TACO*, 14(1):8:1–8:26, 2017.

[11] R. R. Hocking. The Analysis and Selection of Variables in Linear Regression. *Biometrics*, 32(1):1–49, 1976.

[12] M. Kirschbaum and T. Popp. *Evaluation of Power Estimation Methods Based on Logic Simulations*, pages 45–51. Verlag der Technischen Universität Graz, 2007.

[13] J. Larus. SPIM: A MIPS32 Simulator. `http://spimsimulator.sourceforge.net/`, accessed June 2018.

[14] M. T.-C. Lee, V. Tiwari, S. Malik, and M. Fujita. Power analysis and minimization techniques for embedded dsp software. *IEEE Transactions on Very Large Scale Integration (VLSI) Systems*, 5(1):123–135, March 1997.

[15] D. McCann. ELMO. `https://github.com/bristol-sca/ELMO`, 2017.

[16] D. McCann, E. Oswald, and C. Whitnall. Towards practical tools for side channel aware software engineering: 'Grey box' modelling for instruction leakages. In E. Kirda and T. Ristenpart, editors, *26th USENIX Security Symposium, USENIX Security 2017, Vancouver, BC, Canada, August 16-18, 2017.*, pages 199–216. USENIX Association, 2017.

[17] J. Morse, S. Kerrison, and K. Eder. On the limitations of analyzing worst-case dynamic energy of processing. *ACM Trans. Embed. Comput. Syst.*, 17(3):59:1–59:22, 2018.

[18] L. W. Nagel and D. Pederson. SPICE (Simulation Program with Integrated Circuit Emphasis). Technical Report UCB/ERL M382, EECS Department, University of California, Berkeley, Apr 1973.

[19] P. Maydell et al. QEMU. `http://wiki.qemu.org/`, 2014.

[20] J. Pallister, S. Kerrison, J. Morse, and K. Eder. Data dependent energy modeling for worst case energy consumption analysis. In S. Stuijk, editor, *Proceedings of the 20th International Workshop on Software and Compilers for Embedded Systems, SCOPES 2017, Sankt Goar, Germany, June 12-13, 2017*, pages 51–59. ACM, 2017.

[21] O. Reparaz. Detecting flawed masking schemes with leakage detection tests. In T. Peyrin, editor, *Fast Software Encryption*, pages 204–222, Berlin, Heidelberg, 2016. Springer Berlin Heidelberg.

[22] Riscure. Inspector SCA Security Tool. `https://www.riscure.com/security-tools/inspector-sca/`.

[23] W. Schindler, K. Lemke, and C. Paar. A Stochastic Model for Differential Side Channel Cryptanalysis. In J. Rao and B. Sunar, editors, *CHES 2005*, volume 3659 of *LNCS*, pages 30–46. Springer Berlin / Heidelberg, 2005.

[24] Secure-IC. Virtualyzr. `http://www.secure-ic.com/solutions/virtualyzr/`, accessed June 2018.

[25] C. Thuillet, P. Andouard, and O. Ly. A smart card power analysis simulator. In *Proceedings of the 12th IEEE International Conference on Computational Science and Engineering, CSE 2009*, pages 847–852. IEEE Computer Society, 2009.

[26] V. Tiwari, S. Malik, and A. Wolfe. Power analysis of embedded software: a first step towards software power minimization. *IEEE Transactions on Very Large Scale Integration (VLSI) Systems*, 2(4):437–445, Dec 1994.

[27] V. Tiwari, S. Malik, A. Wolfe, and M. T.-C. Lee. Instruction level power analysis and optimization of software. *VLSI Signal Processing*, 13(2-3):223–238, 1996.

[28] N. Veshchikov. Use of Simulators for Side-Channel Analysis: Leakage Detection and Analysis of Cryptographic Systems in Early Stages of Development. PhD thesis, Université Libre de Bruxelles, 2017.

[29] D. Welch. Thumbulator. `https://github.com/dwelch67/thumbulator.git/`, 2014.