Contents lists available at ScienceDirect

Microprocessors and Microsystems

journal homepage: www.elsevier.com/locate/micpro

Side-channel leakage models for RISC instruction set architectures from empirical data

Hermann Seuschek[a,*], Stefan Rass[b]

[a] Institute for Security in Information Technology, Technische Universität München, Germany
[b] Institute of Applied Informatics, Alpen-Adria Universität Klagenfurt, Austria

ARTICLE INFO

Article history:
Received 30 October 2015
Accepted 11 January 2016
Available online 3 February 2016

Keywords:
Side-channel attack
Leakage Model
Applied cryptography
Processor architecture
Internet of Things

ABSTRACT

Side-channel attacks are currently among the most serious threats for embedded systems. Popular countermeasures to mitigate the impact of such attacks are masking schemes, where secret intermediate values are split in two or more values by virtue of secret sharing. In case of unwanted correlations between different registers inside the CPU, the shared secret may leak through a side-channel. This problem is particularly evident on low cost embedded systems, such as nodes for the Internet of Things (IoT), where cryptographic algorithms are often implemented in pure software on a reduced instruction set computer. On such an architecture, all data manipulation operations are carried out on the contents of the CPU's register file. This means that all intermediate values of the cryptographic algorithm pass through the register file. Towards avoiding unwanted leakages, special care has to be taken in the mapping of the registers to intermediate values of the algorithm. In this work, we describe an empirical study that reveals effects of unintended unmasking of masked intermediate values and thus leaking secret values. The effects are abstracted to the level of the instruction set architecture on a RISC CPU. Furthermore, we discuss the possibility to have the compiler thwart such leakages.

1. Introduction

Starting with the seminal paper from Kocher et al. [15], the field of side-channel attacks emerged to a mature and active research topic. As opposed to classical cryptanalysis where the theoretical concepts of cryptographic algorithms are analyzed, side-channel attacks deal with the security properties of *implementations* of these algorithms [17]. These implementations leak information of their internal state in terms of timing behavior, power consumption, electromagnetic radiations and others. Such leaked information can be exploited by using different techniques to infer information on cryptographic secrets such as keys. Side-channel attacks are known to be very efficient and often easy to perform.

The way how internal information is leaked is modeled by a leakage function L of the form usually being,

$$L(\cdot) = L_d(\cdot) + N,$$

where $L_d\ (\cdot)$ is the deterministic part depending on the internal state and N is random noise. Balasch et al. [6] categorize the deterministic part of the leakage function by two properties: The leakage function can be *value-based* or *transition-based* and it can be *generic* or *specific*.

Value-based leakage functions model the leakage of a variable by assuming that the variable value is leaked directly. Generic value-based leakage functions map possible values to the set of leaked physical quantities. For specific value-based leakage functions the cardinality of the set of leaked physical quantities is smaller, as it is the case for Hamming weight leakage. For transition-based leakage functions the leaked value depends on the current value and the previous value of a variable. Generic transition-based leakage functions map possible values to the set of leaked physical quantities. Table 1 shows the differences in these four settings.

Throughout the experimental part of this work (Section 3), we will consider a *specific* leakage that is the *Hamming weight* $\mathrm{hw}\ (\cdot)$ and the Hamming distance $\mathrm{hd}\ (\cdot,\cdot)$ of a register content or combinations of different register contents. Note that for two bitstrings x, y, we have $\mathrm{hd}(x, y) = \mathrm{hw}(x \oplus y)$, where $\oplus$ denotes the bitwise XOR. This justifies the aggregation of two register values $(v_{\text{before}}, v_{\text{after}})$ into $v_{\text{before}} \oplus v_{\text{after}}$ whenever transition-based leakage functions in terms of Hamming distance are being used.

We let $\mathcal{V}$ denote the set of all possible register contents. The leakage function maps concrete values thereof to a physical (measured) leakage value within the set $\mathcal{L}$. The cryptanalysis proceeds

* Corresponding author. Tel.: +498928928253.
E-mail addresses: hermann.seuschek@tum.de (H. Seuschek), stefan.rass@aau.at (S. Rass).

http://dx.doi.org/10.1016/j.micpro.2016.01.004
0141-9331/© 2016 Elsevier B.V. All rights reserved.

Table 1
Categorization of leakage functions according to Balasch et al. [6].

	Value-based	Transition-based
Generic	$L_d : \mathcal{V} \to \mathcal{L}$ $v_{\text{current}} \mapsto \ell$ $\lvert\mathcal{V}\rvert = \lvert\mathcal{L}\rvert$	$L_d : \mathcal{V} \times \mathcal{V} \to \mathcal{L}$ $(v_{\text{before}}, v_{\text{after}}) \mapsto \ell$ $\lvert\mathcal{V}\rvert = \lvert\mathcal{L}\rvert$
Specific	$L_d : \mathcal{V} \to \mathcal{L}$ $v_{\text{current}} \mapsto \ell$ $\lvert\mathcal{V}\rvert > \lvert\mathcal{L}\rvert$	$L_d : \mathcal{V} \times \mathcal{V} \to \mathcal{L}$ $(v_{\text{before}}, v_{\text{after}}) \mapsto \ell$ $\lvert\mathcal{V}\rvert > \lvert\mathcal{L}\rvert$

by backward inference from the measured leakage to a hypothesis on the register content. The set of hypotheses is marked by a star, such as $v*$ being a hypothesis based on the leakage ℓ caused by processing the register content v.

In order to mitigate side-channel vulnerabilities, many countermeasures have been proposed [17]. These countermeasures can be divided into two principal classes, which are *hiding* and *masking* actions. Hiding countermeasures aim to reduce the dependence of the externally available physical quantities and the internal values. Masking countermeasures do not reduce the leakage of individual variables, but split sensitive variables in two or more parts (secret sharing). Computations are carried out on the individual shares. This is only easy to implement for linear operations; non-linear operations require special treatment. Besides this issue, masking countermeasures must assure that the masks and masked values are *processed in a separate way* and that a random source is available for generating the masks. Nevertheless, masking countermeasures are popular for software implementations of cryptographic algorithms. Implementing masking countermeasures in software can hardly be done in higher level languages because the compiler may apply masks too late or remove the mask at all for optimization if it classifies the mask as redundancy. Therefore masking should be done within the assembly representation and calls for highly experienced and skilled designers. This way of design is time consuming and expensive, which gave rise to designated research on tools that support designers in the implementation of cryptographic algorithms in a side-channel resilient fashion.

2. Related work

Bayrak et al. [7,8] present a framework for automatic application of side-channel countermeasures. Their approach is divided into two steps: first, an implementation of a cryptographic algorithm is analyzed regarding its side-channel leakage per machine instruction. In the second step, based on the analysis results, countermeasures are applied to sequences of instructions. The authors demonstrated a significant reduction of the side-channel leakage while the resulting overhead for the countermeasures is lower than for a global application of the countermeasures.

Moss et al. [18] show how a compiler could be modified to insert a first order masking in an implementation of a cryptographic algorithm. The authors implemented a prototype compiler for a domain specific language, which is capable of generating ARM code. The paper presents practical analysis results that prove the effectiveness of the method. Additionally, Agosta et al. in [2–4], present methods to randomize the control flow of cryptographic software during runtime, while preserving the code's functionality. A similar approach was proposed by Bayrak et al. [10] with the difference that shuffling of independent instructions is done by a hardware unit residing between the CPU and the instruction memory. Beyond the addition of this unit, no modification of the hardware architecture is required; only the compiler has to add annotations in the machine code to tell the shuffler which commands can be shuffled without altering the semantic of the code. Another approach proposed by Bayrak et al. [11] is based on modifying the clock tree and adding jitter to the clock of a synchronous circuit. This complicates the alignment of measurements for side-channel attacks, thus reducing the likelihood of the attack being successful. By setting proper constraints, commercial synthesis tools can handle such designs in a straightforward way.

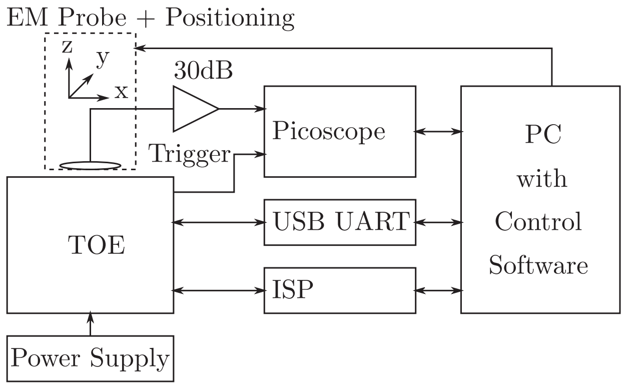

Fig. 1. Side-channel measurement setup.

Agosta et al. [1] present a security oriented data flow analysis (SDFA) framework for analyzing the data flow in an algorithm and for applying side-channel countermeasures at nodes of the flow graph where the input data depends on a selectable number of secret key bits.

Bayrak et al. [9] present the tool *Sleuth* that verifies the correct implementation of side-channel countermeasures by transferring program code to a data flow graph representation and formulating a satisfiability (SAT) problem to check if masking was applied in a correct way. The resulting SAT problem can be solved with state-of-the-art SAT solvers.

Recently, Balasch et al. [6] claimed that research on the design automation for side-channel security can rely on theoretical foundations for side-channel countermeasures. Our work is intended to substantiate this opinion.

This paper is an extension to Seuschek et al. [20] were unexpected leakage of values stored in the register file was described. We will give an explanation of the observed behavior based on the characteristics of the circuits in the data path of the examined CPU.

3. Experimental analysis of a CPU

3.1. Measurement setup

Measurements were carried out with the setup depicted in Fig. 1. The EM emanations are gathered with a *Langer RF-B 3-2* near field probe which detects vertical emanations with an resolution of approximately 2 mm. The near field probe can be moved in x-, y- and z-direction by using a modified Velleman K8200 3D printer. The minimum step size in x- and y-direction is 0.015 mm and 0.781 μm in z-direction. This resolution should even allow localized side-channel attacks [14] when using a finer probe. Our application of the positioning system is to find a location with high side-channel leakage. This is achieved by moving the near field probe on top of the CPU and determining the Pearson correlation like in a correlation power analysis (CPA). It turns out, that the highest correlation coefficients are observed above the bonding wires carrying the supply current.

The probe signal is amplified using a *Langer PA 303* high bandwidth amplifier with 30 dB gain. The output of the amplifier is fed into a *PicoScope 6402C* configured with a sampling rate of 1.25 $\frac{\text{GS}}{\text{s}}$

and an analog bandwidth of 250 MHz. The acquired measurement traces are stored in structured binary files in the *HDF5* format [21] for later analysis.

For our experiments we use two different targets of evaluation (TOEs). The primary TOE is a Papilio One 500K[1] board with a build-on Xilinx Spartan 3 XC3S500E FPGA. This FPGA is configured with the Atmel AVR clone by Lepetenok [16] and modifications from Gassett [13] which is clocked by a 32 MHz crystal oscillator. This clock is halved by the digital clock manager on the FPGA in order to derive the CPU clock. For comparisons with the genuine AVR implementation, we use self-build circuit board featuring an *Atmel ATmega32* micro controller unit (MCU) clocked by a 7.3728 MHz crystal oscillator. The execution of a code sequence Π on the TOE can be triggered by a command transmitted by a UART interface. Additionally, the UART interface allows to fill the register file with arbitrary values before the execution of Π and reading of the registers afterwards.

Controlling the overall side-channel analysis setup including probe positioning, TOE programming, TOE control, and oscilloscope control as well as the data analysis is done by a Python-based software framework. The framework allows the fully automated analysis of the leakage behavior of different instructions and register configurations.

3.2. The examined CPU architecture

The reduced instruction set computer (RISC)[19] is a CPU architecture designed in the nineteen-eighties in Berkeley. The idea started a paradigm shift in the design of CPUs. These days, it was a common practice to make CPUs more complex by integrating an increasing number of complex instructions (complex instruction set computer, CISC). As opposed to this, RISC aimed to simplify the hardware design of CPUs by reducing the functionality implemented in hardware, allowing simpler instruction decoding and memory interfacing. The simple memory interface manifests in the fact that almost all instructions operate on data stored in the register file. The register file offers a relatively large number of registers while memory accesses are handled by `load` and `store` instructions.

Due to the register-centric data path, the register file with the arithmetic logic unit (ALU) is an interesting study object for side-channel leakage on architecture level.

The examined MCU is a member of a simple yet powerful family of 8 bit RISC MCUs. The architecture follows the modified Harvard model, which means that although there are separated memories for code and data, it is possible to read data from the code memory. The relatively low maximum clock frequency of the included central processing unit (CPU) makes caches unnecessary. Most instructions are executed in a single clock cycle with the exception of branching and load/store. The CPU pipeline has two stages, one for instruction *fetch* and the second one for the *execution*.

The register file consists of 32 registers, each of which is 8 bit wide. The registers `R26:R27`, `R28:R29`, and `R30:R31` alternatively treated as 16 bit registers (`X`, `Y`, `Z`) for instructions requiring 16 bit addresses as arguments.

Fig. 2 gives a simplified overview of the data path formed by the register file and the ALU. Some multiplexers are omitted in the diagram to focus on the interaction between the two modules of interest. It is important to note that most of the signals are combined in a purely combinatorial way. Only when it comes to writing to a register, the data in the corresponding register is updated with the clock edge. The ALU is also a combinatorial circuit which

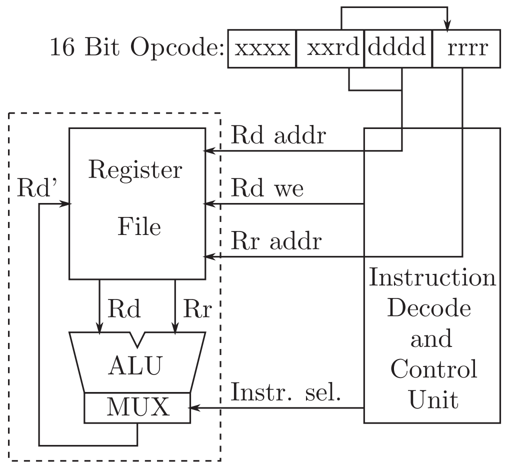

Fig. 2. Simplified view on the AVR data path.

performs all implemented computation on the input values in parallel. At the output the required result is selected by a multiplexer which is controlled by the instruction decoder.

As shown in Fig. 2 the selection of the registers is performed by two 5 bit wide address buses. These buses are configured by the instruction decoder in accordance to the opcode. For the AVR instruction set 10 bits out of the 16 bits are reserved for selecting registers. After instruction fetching, the bits directly applied to the buses without any decoding step for most instructions.

3.3. Identifying leakage behavior

Seuschek et al. [20] observed an unexpected side-channel behavior of code sequences implemented on a CPU. Values stored in some combinations of registers leak the Hamming distance of the corresponding stored values. This could lead to unintentional unmasking in implementations of cryptographic algorithms applying masking countermeasures. Initially, this effect was quite surprising, and towards an explanation, we will first look at the underlying mechanisms involved in the data path by carrying out experiments and formalizing the findings.

In the following, we will denote the *i*th register as `Ri` when we refer to the *physical entity*, while writing R_i to mean the *value* stored in register `Ri`. Consequently, the variables $R_0, \ldots, R_{31}$ represent the contents of the (physical) register file `R0...R31`. According to the AVR instruction set manual [5] we denote the destination and first source register as `Rd` and the second source registers as `Rr`

As explained before, individual registers `Ri` in the register file are selected by an internal address bus. Due to different delays for each individual signal on the address bus, the register selection multiplexers select different registers until the address value on the input of the register file has been settled. This causes glitches with the consequence that, depending on the desired register, several register values R_i are exposed to the internal data buses between the register file and the ALU. Subsequently the Hamming distance of the consecutive exposed register values is leaked.

Some instructions perform a register selection, even if the data words stored in the particular registers are not used by the instruction. In this case the register selection bits in the opcode are used to encode different information but this bits are still exposed to the address buses. This causes signal transitions in the data path which cause the leakage of register values even if the values are not used for an actual computation. This effect can be observed when the `NOP` instruction is executed on the AVR CPU.

[1] http://www.papilio.cc.

This instruction is often used for side-channel analysis of short instruction sequences to remove pipelining effects [17]. For the AVR architecture the opcode of NOP is 0x0000. This means that value R_0 is exposed to both ALU inputs Rd and Rr. As a consequence the NOP instructions in observed programs lead to severe leakage of value R_0 which was observed and briefly discussed in [20].

We analyzed the leakage behavior of different registers during execution of CPU instructions. To build a model for this behavior, we carried out experiments with different programs Π. This programs were executed $n = 15000$ times with different random bytes stored in the register file. For this, our software framework retrieves uniformly distributed random 8 bit values $u^{(k,j)} \in \{0, 1\}^8$, before the kth execution of Π with $0 \leq k < n$. The values are sent to the MCU using a UART interface where they are stored in the registers Rj with $0 \leq j < 26$. We only considered registers R0...R25 for these experiments, since the remaining registers R26...R31 are reserved for storing 16 bit addresses. This makes a total of 26 uniformly random values being stored in the register file before each execution of Π. Afterwards a general purpose input output (GPIO) pin is set using the set bit instruction (sbi) to trigger the oscilloscope. Now the program Π, which performs operations on the random register values R_j, is executed. After finishing Π the trigger pin is cleared using the clear bit instruction cbi and the modified register values R_j with $0 \leq j < 26$ are returned using the UART interface. The n executions of Π make up a collection of electromagnetic emanations (leakage vectors). Each of these is a time-series $(\boldsymbol{\ell}^{(t)})_{t \in T}$ over the discrete time steps $t \in T = \{t_{\text{start}} < t_2 < t_3 < \cdots < t_{\text{finish}}\}$.

We use specific transition- and value-based leakage functions to map the random register contents to a leakage hypothesis $L_d : ((R_0, \ldots, R_{25})_{before}, (R_0, \ldots, R_{25})_{after}) \mapsto \ell^*$ and $L_d : (R_0, R_1, \ldots, R_{25}) \mapsto \ell^*$. The actual functions we utilized are the Hamming distance model (i.e. the number of flipped bits) with $\ell^* = \text{hd}(v_{before}, v_{after}) = \text{hw}(v_{before} \oplus v_{after})$ and the Hamming weight of a register value v (i.e. the number of bits set) with $\ell^* = \text{hw}(v)$. While the leakage of most CMOS devices can be well approximated by Hamming distance, often the Hamming weight model also gives good results. This is because of the fact, that Hamming weight is a special case of Hamming distance where the initial value is constant $v_{before} = 0$. For our experiments, where before each iteration fresh random data was stored in the register file, no Hamming weight leakage could be observed.

To mount our analysis, we evaluate the Pearson correlation between the vector of leakage hypotheses $\boldsymbol{\ell}^*$ and the vector of all n leakages $\boldsymbol{\ell}_\Pi(t)$ that were measured at time instance t in program Π. To this end, we computed the Pearson correlation coefficient $\rho(t)$ at time t. This is basically a correlation power analysis (CPA) [12] without key hypotheses, because the "intermediate value" is not depending on an unknown secret key. As a result, we get a vector of correlation coefficients for each program Π in conjunction with a leakage hypothesis ℓ^* denoted by $\boldsymbol{\rho}_{\Pi,\ell^*}(t) = \text{corr}(\boldsymbol{\ell}^*, \boldsymbol{\ell}_\Pi(t))$.

3.3.0.1. Leakage caused by selection of both registers. This first experiment is to examine the leakage of a program sequence Π^{both} which does not alter the internal state of the CPU with the exception of the change in the program counter:

```
mov r0, r0
mov r15, r15
mov r0, r0
```

The code sequence was selected intentionally to force many bits (in this case four) of both register address buses to change. We start with setting all bits to *zero* (mov r0, r0). Afterwards the four least significant bits are set to *one* (mov r15, r15). The most significant bit is omitted because of the special use of R26 to R31 as memory addressing registers. In a last step the program sets the register address buses again to *zero*.

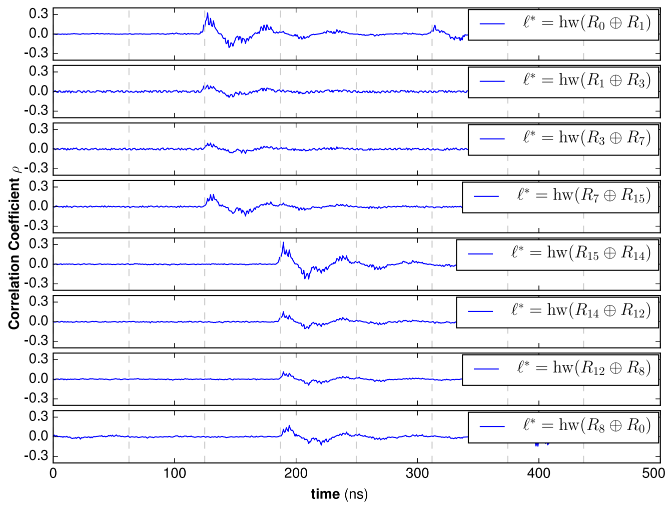

Fig. 3. Correlations $\rho(t)$ between measured EM emanation during execution of Π^{both} and different leakage assumptions ℓ^* for the FPGA implementation of the AVR.

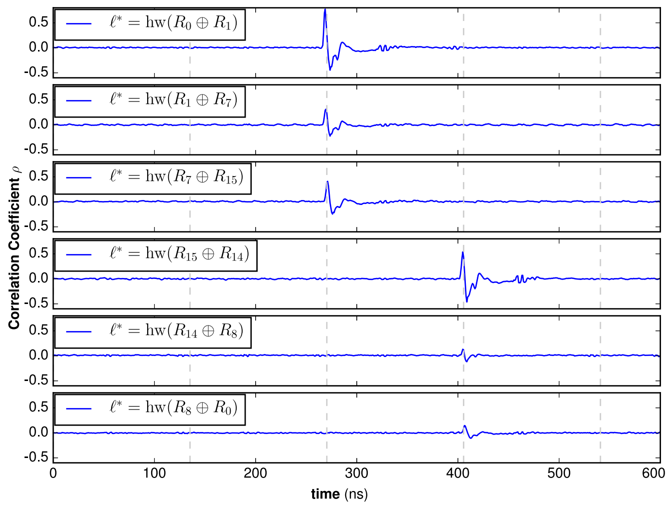

Fig. 4. Correlations $\rho(t)$ between measured EM emanation during execution of Π^{both} and different leakage assumptions ℓ^* for the AVR implementation from Atmel.

First, it was observed that the Hamming distance $\ell^* = \text{hd}(R_0, R_{15})$ and Hamming weight $\ell^* = \text{hw}(R_j)$ with $j \in \{0, 15\}$ do not lead to any noticeable correlations for both TOEs. This clearly shows that there is no direct transition from R0 to R15 and that the devices do not leak register content through Hamming weight.

Figs. 3 and 4 depict the correlation between different alternative leakage assumptions and the captured side-channel emanations for the two available implementations of the AVR architecture. It can be seen that the differing signal timing of the register address bus bits leads to the selection of several registers from R0 up to R15 and back to R0 in a sequence. The sequence is determined by the signal timing of the CPU implementation with the consequence that different implementations may lead to different sequences. In our experiments we observed this kind of behavior. For the FPGA implementation of the AVR core, the sequence of selected Registers is R0 $\rightarrow$ R1 $\rightarrow$ R3 $\rightarrow$ R7 $\rightarrow$ R15 $\rightarrow$ R14 $\rightarrow$ R12 $\rightarrow$ R8 $\rightarrow$ R0. Fig. 3 shows the corresponding correlation traces for the Hamming distance of the consecutive register values.

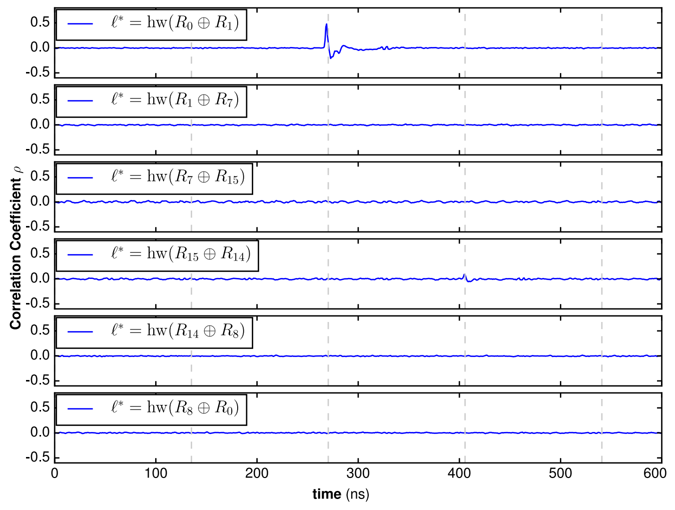

Fig. 5. Correlations $\rho(t)$ between measured EM emanation during execution of Π^{first} and different leakage assumptions ℓ^* for the AVR implementation from Atmel.

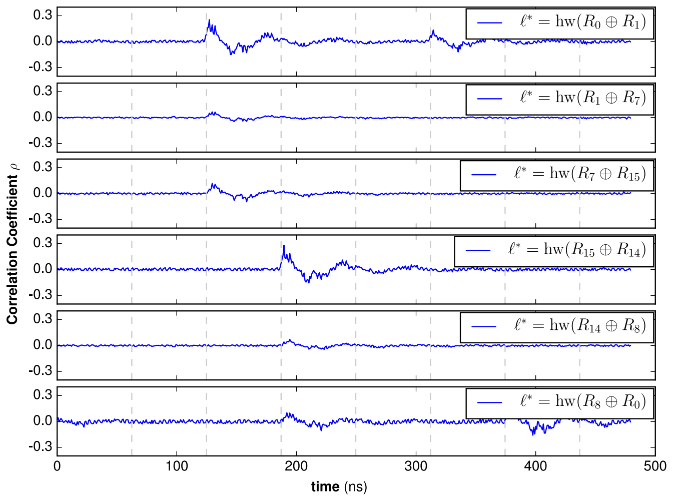

Fig. 6. Correlations $\rho(t)$ between measured EM emanation during execution of Π^{second} and different leakage assumptions ℓ^* for the FPGA implementation of the AVR.

The genuine AVR implementation from Atmel shows a slightly different leakage behavior. Although the timing of the bus signal is quite similar, the timing of two signals cannot be differentiated with our measurement setup. The observed register selection sequence is R0 → R1 → R7 → R15 → R14 → R8 → R0. Fig. 4 gives an overview of the analysis results for this sequence.

In the following we will compare the contribution of both the first parameter of the assembler instruction Rd and the second one Rr to the leakage of the register values.

3.3.0.2. Leakage caused by selection of first register. The leakage of the first register parameter Rd of the assembly instructions is analyzed using the following code sequence Π^{first}:

```
mov r0,r0
mov r15, r16
mov r0, r0
```

Here again, in the first step the two register selection buses are set to zero. In the next step the four least significant address bits of the first and the most significant bit of the second register selection bus are set to one (`mov r15, 16`). For the first bus we should observe the same sequence of register selections as in the first experiment using Π^{both}. The second bus should directly select R16 after R0 and finally select R0 again because only one address bit is affected.

The FPGA implementation of the AVR shows almost the same leakage behavior as in the experiment before shown in Fig. 3. The main difference is the reduced correlation coefficient which shows that the contribution to the leakage of the first parameter is lower than for the second one. Surprisingly the behavior of the AVR implementation is completely different as shown in Fig. 5. Here the expected register sequence does not lead to observable correlations.

3.3.0.3. Leakage caused by selection of second register. Finally, we analyze the leakage behavior of the second assembly instruction parameter Rr with the code sequence Π^{second}:

```
mov r16, r0
mov r16, r15
mov r16, r0
```

For this experiment we set the first parameter (destination register) to R16 with the result that this register is the only one which is altered by the code sequence. Additionally, there are no read accesses to this register even not by glitches in the data path. The second parameter is initially set to R0, then to R15, and finally to R0. This should lead to the register selection sequences as observed before.

In the initial analysis of the measurements obtained from the FPGA no correlations for the transitions R1 → R3 and R3 → R7 where observed. This led to the assumption that here, like in the genuine AVR implementation, a transition from R1 to R7 occurs. The results depicted in Fig. 6 confirm this assumption. This means that the selection of the first and the second register lead to different behavior.

The results for the genuine AVR implementation are almost the same as depicted in Fig. 4 which clearly shows that the selection of the second parameter has the highest influence on this CPU implementation.

Based on our observations we develop a formal model in Section 4 to describe and identify plausible register selection sequences based on available measurements.

4. Modeling the observed leakage behavior

The timing behavior of the register selection process can be modeled using a graph G_L where the set of nodes $V = \{\mathrm{R0}, \mathrm{R1}, \ldots\}$ represents the available registers and the set of edges $E = \{(\mathrm{R0}, \mathrm{R1}), (\mathrm{R0}, \mathrm{R2}), \ldots\}$ the possible direct transitions between two registers. According to our experiments, each of this transitions cause Hamming distance leakage of the values stored in corresponding registers. Fig. 7 exemplary shows such a graph for 16 registers and single bit transitions between the selected registers. The edges of the shortest simple paths (i.e., a path without loops) between two nodes represent all transitions, and therefore candidates for leaked Hamming distances. Only one of the possible paths is a critical one which causes leakage. This path can be identified by computing the correlations between measured time series and the possible transitions for each node. The path which causes the highest correlations is then assumed to be the critical one. Fig. 7 highlights the path between register R0 and R15 which was identified as the critical one in our experiments with the genuine AVR implementation from Atmel.

For our FPGA based AVR implementation we observed that two bits of the register selection bus change in a way that we cannot detect Hamming distance leakage for single bit transitions. In this case we skip a node in the graph. The behavior can be modeled by addition additional edges to the graph that represents two

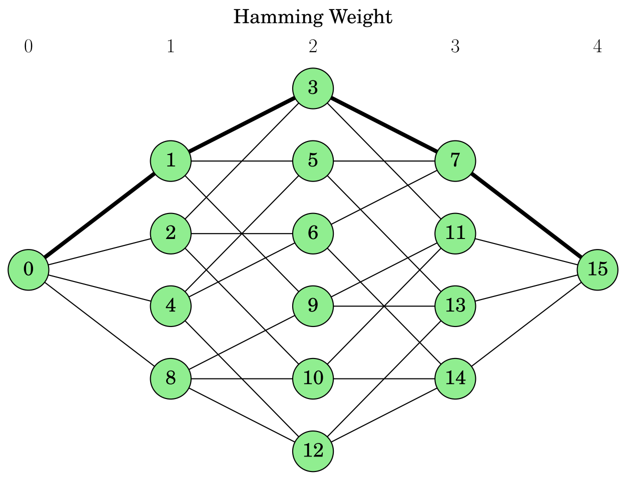

Fig. 7. Graph representing the transitions of selected registers when only one bit on the register selection bus changes at a time.

changing bits at a time. Even more complex timing behavior can be modeled this way.

Any hypothesis on the register selection sequence (regardless of whether it has been obtained from maximal correlation search or by pure enumeration of all possible sequences in the graph G_L), can be put to a statistical test to confirm its validity. We applied the following procedure to narrow down the set of candidate paths in G_L as follows: let π_h be an arbitrary among the paths in G_L from 0 to the target address (15 in our case). The hypothesis that we shall test is the assumption of the intermediate register sequence being given by π_h. That is, from π_h, we derive a set of Hamming distances between consecutive registers being addressed along the path. For example, the path $\pi_h = 0 \to 1 \to 3 \to 7 \to 15$ would give the Hamming distances $\mathrm{hw}(R_0 \oplus R_1)$, $\mathrm{hw}(R_1 \oplus R_3)$, $\mathrm{hw}(R_3 \oplus R_7)$ and $\mathrm{hw}(R_7 \oplus R_{15})$ (cf. Fig. 3 for example). Let the hypothesis π_h consist of m different registers $R_1, \ldots, R_m$, where R_m is the register used in the instruction (for addressing R15 along the aforementioned sequence π_h, we would have $m = 4$ terms, i.e., the length of the path from R0 to R15 in G_L, minus one). At any time-step t of the measurement (time series), we fit a linear model $\ell(t) = \sum_{i=1}^{m-1} \alpha_i \cdot \mathrm{hw}(R_i \oplus R_{i+1}) + N$ to the measured leakage $\ell(t)$, where the unknown α-coefficients are determined by a least-squares algorithm, and the error term N is assumed with a Gaussian distribution. Following a conventional analysis of variance, we consider the influence of a term $\mathrm{hw}(R_i \oplus R_{i+1})$ as *significant*, if the linear model obtained by removing the term provides a statistically different (i.e., worse) explanation for the leakage, than the full model that includes the term.

In this way, we obtain significance indicators for all Hamming distances that are relevant according to our hypothesis π_h, at all time steps. If our hypothesized sequence π_h were right, then the terms $\mathrm{hw}(R_i \oplus R_{i+1})$ (for $i = 1, 2, \ldots, m$) should become significant for the first time exactly in the order that the addressing sequence (according to the hypothesis π_h) prescribes. More specifically, we can easily compute for each term $\mathrm{hw}(R_i \oplus R_{i+1})$ the earliest time $t = t_i$ when this term has gained significant influence to the leakage $\ell(t)$ in the time series. Assuming the correctness of the hypothesis π_h, the so-obtained time-steps should form a monotone sequence, i.e., we should observe $t_1 \leq t_2 \leq \cdots \leq t_{m-1}$.

We implemented the whole procedure in the R language[2] to validate the approach, with the following result: taking the measurement data obtained by the execution of Π^{both} on our FPGA platform, the test rejected all candidate sequences except for $\pi_1 = 0 \to 1 \to 3 \to 7 \to 15$ and $\pi_2 = 0 \to 1 \to 3 \to 11 \to 15$. The first one has independently been confirmed by the maxima of correlations depicted in Fig. 3 (exhibiting the same order of influences of the terms along the time series), whereas the second seems to be an artifact that – in a practical analysis – would call for a deeper look using the second test (based on correlations), as it did in our experiments.

Unfortunately, we were unable to validate the assumption on the Gaussian distribution in the linear model (the respective Shapiro–Wilk tests rejected the hypothesis on the distribution of the error term N), so that the linear model is actually *not* a good fit for the leakage measurement. Thus, a future direction of refinement could be looking for more accurate models for the error term. Indeed, the linear model (at least the systematic part of it) appears to be a valid candidate, since the additivity of EM radiations caused by accessing different bit-lines is directly found in the model (as the terms are added there too). Also, even though different bit-lines leak differently, the α-constants in the model would account for this. For the error term, the proper distribution remains somewhat unclear and needs further analysis. One approach (up to testing along future work) could be based on considering the error term as Skellam distributed (given that the Hamming distances are counting data, and that the EM radiation rises and diminishes with bits being set or reset). On the bright side, however, the Skellam distribution can somehow be approximated by a Gaussian distribution (asymptotically), thus underpinning the model as a good (though not perfect) tool to identify potential addressing sequences. Our empirical findings confirm this intuition quite nicely.

5. Countermeasures

In general, the task is to minimize the leakage specific behavior caused by the micro-architecture of a CPU. Especially the Hamming distance leakage of register pairs which is unexpected from a software developers perspective is addressed. The results of the analysis already suggest a simple countermeasure. Depending on the possible register transitions defined by the internal hardware timings, storing critical value pairs have to be prevented. Most importantly, the registers should be allocated in a way that avoids the assignment of masked secrets and masks to these pairs.

As the edges of our example graph G_L represent all possible transitions between selected registers, the adjacency matrix of the graph nicely visualizes register pairs which are prone to leak the Hamming distance of values stored in them. Fig. 8 visualizes the matrix for single bit and dual bit transitions as a two-dimensional image where the edges are represented by colored pixels. As a result, the pixels mark the critical register pairs which have to be prevented. It can be observed that for single bit transitions, the matrix is very sparse and there are still 208 out of $16 \cdot 15 = 240$ register pairs available for secure use. By adding all possible dual bit transitions the amount is reduced, but there are still 184 register pairs left.

As we have shown, the proper selection of registers can reduce the vulnerability to side-channel attacks. It cannot be ruled out, that this was taken into account by designers unintentionally before by changing assembly code in a trial and error approach when side-channel leakage was observed. Beyond this strategy, our results are intended to help software developers to understand the leakage mechanisms on the instruction set architecture level. Most importantly, the results should bring further the development of design tools which perform the register allocation in a way that the side-channel leakage is minimized. The adjacency matrix of G_L can for example be used to guide the compiler backend or as

[2] http://www.r-project.org.

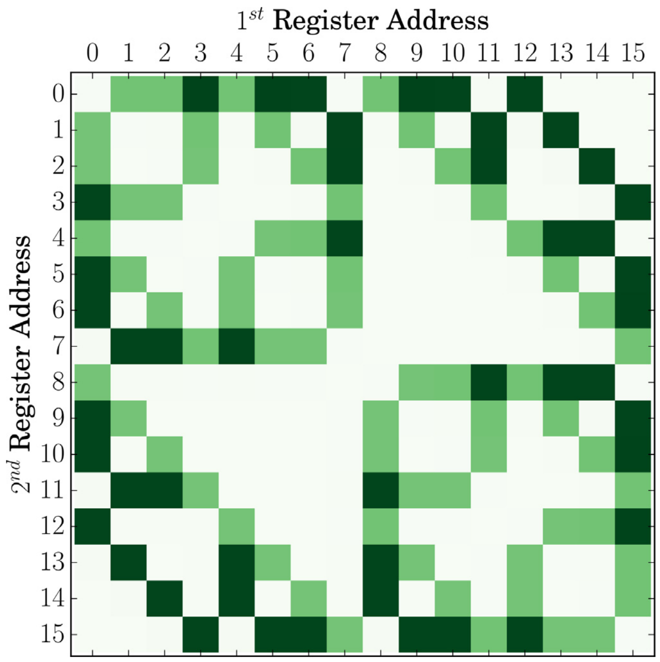

Fig. 8. Image representing critical register pairs under the assumptions that only one bit (lighter color) and two bits (darker color) of the register selection bus change at a time.

a separate tool, in doing the register allocation on assembly code level.

6. Outlook

Our experiments show hidden side-channel leakage on the instruction set architecture level, which can be discovered by statistical analysis. We presented a model to deduct the mutual influences between registers, which can be used to guide a compiler in the assignment of registers in a way so that secrets do not leak the Hamming distance between certain register combinations. This effect is even more to be avoided, since our experiments showed that leakage can occur even though a register is never touched from a software perspective.

As a conclusion, the (tool-aided) design and implementation of cryptographic software that is side-channel resilient must not restrict attention to the instruction level, but has to consider hardware characteristics too. Hidden leakage behavior, e.g. induced by the underlying micro-architecture, has to be sought and modeled in a way that enables a treatment on the instruction level. Nevertheless, not all leakage phenomena are fully understood, which calls for additional experiments along subsequent and follow-up work. One such piece of future work is the investigation of the leakage behavior of different CPU architectures. We currently started with experiments using two different implementations of the Atmel AVR architecture. This allows a better understanding of the observed phenomena and a comparison of different implementations. The FPGA implementation allowed the explorations of the micro-architecture which was also useful for the analysis of the genuine AVR implementation from Atmel.

The main contribution here is the report on leakage phenomena caused by hardware characteristics, so as to support the development of advanced tools to help designers of side-channel hardened cryptographic software. Towards this goal, the *automatic application* of the proposed countermeasures, e.g., the proper selection of registers, is perhaps one of the most promising areas of future investigations.

Acknowledgments

This work has been partly funded by the German Federal Ministry of Education and Research, grant number 01IS13020A (SIBASE). Furthermore, the authors want to thank Siemens AG for the financial support and the DSD 2015 committee for selecting this work for publication in the TRUDEVICE special issue of the MICPRO ELSEVIER journal.

References

[1] G. Agosta, A. Barenghi, M. Maggi, Pelosi, Compiler-based side channel vulnerability analysis and optimized countermeasures application, in: IEEE, 2013.
[2] G. Agosta, A. Barenghi, G. Pelosi, A code morphing methodology to automate power analysis countermeasures, in: Proceedings of the 49th Annual Design Automation Conference, DAC '12, ACM, New York, NY, USA, 2012, pp. 77–82, doi:10.1145/2228360.2228376.
[3] G. Agosta, A. Barenghi, G. Pelosi, M. Scandale, A multiple equivalent execution trace approach to secure cryptographic embedded software, in: Proceedings of the 51st Annual Design Automation Conference on Design Automation Conference, DAC'14, ACM, New York, NY, USA, 2014, doi:10.1145/2593069.2593073.
[4] G. Agosta, A. Barenghi, G. Pelosi, M. Scandale, The MEET approach: securing cryptographic embedded software against side channel attacks, IEEE Trans. Comput.-Aided Des. Integr. Circ. Syst. 34 (8) (2015) 1320–1333, doi:10.1109/tcad.2015.2430320.
[5] Atmel AVR 8-bit Instruction Set, Atmel Corporation, 2014.
[6] J. Balasch, B. Gierlichs, V. Grosso, O. Reparaz, F.-X. Standaert, On the Cost of Lazy Engineering for Masked Software Implementations. Report 2014/413, Cryptology ePrint ArchiveJune 2014.
[7] A. Bayrak, F. Regazzoni, D. Novo Bruna, P. Brisk, F. Standaert, P. Ienne, Automatic application of power analysis countermeasures, IEEE Trans. Comput. 64 (2) (2015) 329–341, doi:10.1109/tc.2013.219.
[8] A.G. Bayrak, F. Regazzoni, P. Brisk, O.X. Standaert, P. Ienne, A first step towards automatic application of power analysis countermeasures, in: Design Automation Conference (DAC), 2011 48th ACM/EDAC/IEEE, 2011, pp. 230–235.
[9] A.G. Bayrak, F. Regazzoni, D. Novo, P. Ienne, Sleuth: automated verification of software power analysis countermeasures, in: G. Bertoni, J.-S. Coron (Eds.), Cryptographic Hardware and Embedded Systems - CHES 2013, *Lecture Notes in Computer Science*, Springer Berlin Heidelberg, 2013, pp. 293–310, doi:10.1007/978-3-642-40349-1_17.
[10] A.G. Bayrak, N. Velickovic, P. Ienne, W. Burleson, An architecture-independent instruction shuffler to protect against side-channel attacks, ACM Trans. Archit. Code Optim. 8 (4) (2012), doi:10.1145/2086696.2086699.
[11] A.G. Bayrak, N. Velickovic, F. Regazzoni, D. Novo, P. Brisk, P. Ienne, An EDA-friendly protection scheme against side-channel attacks, in: Proceedings of the 2013 Design, Automation & Test in Europe Conference & Exhibition (DATE), 2013, IEEE, 2013, pp. 410–415, doi:10.7873/date.2013.093.
[12] E. Brier, C. Clavier, F. Olivier, Correlation power analysis with a leakage model, in: M. Joye, J.-J. Quisquater (Eds.), Cryptographic Hardware and Embedded Systems - CHES 2004, *Lecture Notes in Computer Science*, 3156, Springer Berlin Heidelberg, 2004, pp. 16–29, doi:10.1007/978-3-540-28632-5_2.
[13] J. Gassett, Arduino Soft Core, 2013, https://github.com/GadgetFactory/Arduino-Soft-Core (accessed 09.02.2016).
[14] J. Heyszl, S. Mangard, B. Heinz, F. Stumpf, G. Sigl, Localized electromagnetic analysis of cryptographic implementations, in: O. Dunkelman (Ed.), Topics in Cryptology CT-RSA 2012, *Lecture Notes in Computer Science*, 7178, Springer Berlin / Heidelberg, 2012, pp. 231–244, doi:10.1007/978-3-642-27954-6_15.
[15] P. Kocher, J. Jaffe, B. Jun, Differential Power Analysis, in: M.J. Wiener (Ed.), Proceedings of CRYPTO'99, *Lecture Notes in Computer Science*, 1666, Springer-Verlag, 1999, pp. 388–397.
[16] R. Lepetenok, AVR Core, 2012, http://opencores.com/project,avr_core (accessed 09.02.2016).
[17] S. Mangard, E. Oswald, T. Popp, Power Analysis Attacks: Revealing the Secrets of Smart Cards (Advances in Information Security), 2007, Springer, 2007.
[18] A. Moss, E. Oswald, D. Page, M. Tunstall, Compiler Assisted Masking, in: E. Prouff, P. Schaumont (Eds.), Cryptographic Hardware and Embedded Systems CHES 2012, *Lecture Notes in Computer Science*, 7428, Springer Berlin Heidelberg, 2012, pp. 58–75, doi:10.1007/978-3-642-33027-8_4.
[19] D.A. Patterson, C.H. Sequin, RISC I: a reduced instruction set VLSI computer, in: Proceedings of the 8th Annual Symposium on Computer Architecture, in: ISCA '81, IEEE Computer Society Press, Los Alamitos, CA, USA, 1981, pp. 443–457.
[20] H. Seuschek, S. Rass, Side-channel leakage models for risc instruction set architectures from empirical data, in: Proceedings of the Euromicro DSD/SEAA, Funchal, Madeira, Portugal, 2015.
[21] The HDF Group, Hierarchical Data Format, version 51997-2015, http://www.hdfgroup.org/HDF5/.

Hermann Seuschek is a doctoral candidate in electrical engineering at the Institute for Security in Information Technology, Technische Universität München (TUM), Germany. He graduated with a Dipl.-Ing. degree (equivalent to master) in 2005. Before he joined his current position, he worked for several years for Siemens Corporate Technology in the field of applied cryptography and embedded systems security. His research interests include secure hardware/software co-design of embedded systems with a focus on architecture, development tools, and side-channel analysis.

Stefan Rass graduated with a double master degree in mathematics and computer science from the Alpen-Adria Universität Klagenfurt (AAU) in 2005. He received a Ph.D degree in mathematics in 2009, and habilitated on applied computer science and system security in 2014. His research interests include applied system security, as well as complexity theory, statistics, decision theory and game-theory. He authored numerous papers related to security and applied statistics and decision theory in security. Furthermore, he participated in various nationally and internationally funded research projects. Currently, he is an associate professor at the AAU, teaching courses on theoretical computer science, complexity theory, security and cryptography.