Functional Integrated Circuit Analysis

Dmitry Nedospasov*, Jean-Pierre Seifert
Security in Telecommunications,
Dept. of Software Engineering and Theoretical Computer Science,
Technische Universität Berlin,
Berlin, Germany
{dmitry,jpseifert}@sec.t-labs.tu-berlin.de

Alexander Schlösser*, Susanna Orlic
Optical Technologies,
Institute for Optics and Atomic Physics,
Technische Universität Berlin,
Berlin, Germany
{schloesser,orlic}@opttech.tu-berlin.de

* These authors contributed equally to this work

***Abstract*—This work introduces a novel, automated methodology for performing functional analysis of integrated circuits (ICs), such as microcontrollers and smart cards. By selectively executing code on a given chip, the resulting optical emission images yield critical information about the chip's functional layout. Automation of the code-generation allows us to generate and process hundreds of test cases that access specific elements of the IC. Subsequently, by correlating the executed code with the images, we are able to locate and identify functional elements of the chip's design, such as memory layout and important registers. This methodology provides an efficient way to isolate potential points of interest and thus significantly reduces the amount of effort required to mount attacks. We present exemplary results for a common microcontroller.**

I. Introduction

Today, vendors of integrated circuits (ICs) are faced with ever-growing threats to the security of their designs. As a result, many manufacturers have simply increased the complexity of their designs and achieved security through obscurity. To protect intellectual property and reduce the potential attack surface, many chips now implement non-standard attack-resistant logic styles and protective metalization layers, making functional analysis of secure ICs a tedious process. Attacks are commonly classified into three categories, based on the amount of sample preparation required to perform the attack. Non-invasive attacks, such as power analysis [1], have the advantage of not requiring sample preparation at all. However, the impact of such attacks can be greatly reduced with more resilient software implementations, i.e., with software countermeasures, such as dummy rounds. This work focuses on the steps necessary to perform more advanced attacks that are far more difficult to prevent in software, but which also require more extensive sample preparation and a greater understanding of the underlying hardware. To perform such invasive and/or semi-invasive attacks, an attacker must first identify points of interest (POIs) in a given hardware implementation, which the attacker can subsequently exploit. Often this requires the attacker first partially reverse-engineer a chip, i.e., perform functional analysis of the IC to identify potential POIs. To prevent such attacks outright, many secure ICs now implement what are referred to as active meshes, current-carrying interconnects that can alert the CPU that the integrity of the IC has been compromised [2]. Especially in the case of fully-invasive attacks, an attacker is thus forced to first circumvent the mesh before beginning analysis.

This work introduces a new methodology that is not impeded by the complexities of modern ICs. To circumvent the mesh completely, we perform semi-invasive backside analysis, which requires little or no preparation beyond decapsulation of the chip. Backside analysis, is performed through the substrate of the IC. By using a combination of hardware and software we are able to perform black box functional analysis of programmable microcontrollers and smartcards independent of the logic style used. Furthermore, our setup allows for complete automation of the code generation and acquisition process. This methodology does not require expensive equipment, nor does it require time-intensive sample preparation, which is the case for many other state of the art analysis methods. Our methodology is able to identify several important POIs, such as, the exact position of bits in memory, important registers or parts of the execution logic or exclude entire areas of the chip. For this reason, when used in combination with electromagnetic side-channel analysis or optical fault injection, for example, the amount of effort that must go into searching for potential target areas on the IC can be greatly reduced if not eliminated entirely.

The main contributions of this paper are as follows:

- A software generation framework for performing functional analysis by targeting specific parts of a processor's logic, making it possible to identify logic, the layout of memory and the location of critical registers.
- An inexpensive setup for efficient, automated backside optical functional analysis of ICs.
- Detailed description of the key features of the proposed setup and the software generation framework.
- Practical results against a common microcontroller, the Atmel ATMega328p.

The rest of this paper is structured as follows: Section II presents the challenges of performing optical emission analysis and also compares our methodology to the state of the art of optical emission analysis. The experimental setup used in this work is detailed in Section III. Section IV presents the practical results, where we were able to identify instruction specific logic, determine the structure and layout of the chip's

internal SRAM and identify individual registers including the status registers used in conditional branching. Finally, Section V concludes this work and several directions for future work are presented in Section VI.

II. Background and Related Work

Observation of light emissions in silicon devices has been a common tool in IC failure analysis since 1955 [3]. As carriers are accelerated by electric fields they gain kinetic energy, which is then released via radiative transitions, generating photons. In CMOS transistors this hot-carrier luminescence takes place at the drain edge where the source-drain electric field is most intense and predominantly in n-type transistors as electrons are more easily accelerated than holes [4]. As a result, optical emissions of CMOS logic show a data-dependent behavior similar, but not equal to power consumption. In the case of a standard CMOS-inverter, the vast majority of photons are generated when the input switches from 0 to 1 and a current flows through the n-type transistor. The photon generation rate is governed primarily by the supply voltage and the switching frequency of the transistors under observation.

Since multiple interconnect layers prevent generated photons from leaving the IC on the frontside, observation has to be conducted from the backside. In this case emitted photons have to pass through the silicon substrate, which is highly absorptive for wavelengths shorter than the bandgap energy, leaving only near infrared (NIR) photons for analysis. In the failure analysis community this effect has been used for many different purposes, primarily to detect and characterize manufacturing faults. The technology of choice to perform such spatially and temporally resolved measurements via backside analysis is Picosecond Imaging Circuit Analysis (PICA), which is based on infrared-sensitive gated multi channel plates. One of the first works to PICA in a security application was [5]. In [5] the authors utilize PICA to attack the `AddRoundKey` operation of AES [6]. In [7] the authors use PICA to develop methods for detecting malicious additions to an IC. However, the cost and complexity of PICA equipment makes it a non-viable choice in real world attacks. Instead of using PICA, [8] demonstrated a low cost alternative, an Si-CCD, which is also capable of recovering the photonic emissions from the silicon substrate via backside analysis. In [9], backside optical fault induction is used to study the effects on on-chip memories. The idea that optical attacks can also be used to partially reverse-engineer a chip is also introduced in [8] and [9].

Since our methods are also based on functional analysis from the backside, they can be considered semi-invasive. Backside analysis using our methodology can be carried out with just a low-cost Si-CCD. In contrast to [9], instead of using a laser to identify interesting areas of a chip, we developed a methodology based on selectively executing code on the chip. By creating loops that are short in length we can maximize the emissions of certain parts of the chip and thus identify their function. This can eliminate the tedious exhaustive search that is the basis of many attacks by identifying potential target areas on the chip directly by studying the optical emission images. Though we present results for the `AVR` architecture and specifically the ATmega328p, this methodology can also be easily applied to other architectures.

III. Experimental Setup

This section covers the design of the hardware and software that make up the experimental setup.

A. Hardware

The hardware setup consists of a Si-CCD camera directly connected to interchangeable microscope objectives and two perpendicularly arranged linear stages onto which a custom printed circuit board (PCB) is mounted with the device under test (DUT).

The optic's design has been kept to an absolute minimum to maximize system throughput and analysis efficiency. We use finite-conjugate reflection type objectives with gold plated mirrors. This way 85% of the captured light reaches the camera; Absorption of NIR photons, common in glass objectives and tube lenses is bypassed. We also use a special CCD sensor type that has become usable for scientific applications in recent years. Back illuminated deep depletion sensors feature the highest NIR quantum efficiency in silicon detectors yet. At 900 nm, over 90% of the impeding photons are detected. To avoid dark current and readout noise the sensor and preamp are thermoelectrically cooled to -70°C and shift and readout rates are customizable.

The DUT is placed into a cavity in the middle of the PCB and soldered upside down. For our experiments, the PCB consisted of the ATmega328p, which was provided an external 16 MHz clock from a standard quartz oscillator on the PCB, and a connection to the measurement PC. To allow a sufficient amount of photons to pass through the absorptive substrate the IC package is opened and backside polished to a substrate thickness of approximately 50 μm. Though we chose to prepare samples using an automated polishing machine, in [10] a lower-cost method for sample preparation is presented.

In addition to an increased software loop frequency, substrate thinning can greatly improve the acquisition time.

For the purpose of functional analysis more elaborate and expensive techniques like silicon immersion lenses, InGaAs cameras or even multi-channel plates have therefore become unnecessary.

B. Software

To efficiently test input sets of several hundred subroutines the chip must be programmed in such a way that it executes a new subroutine on every boot. Thus, the code generation framework must implement the following features: (1) In order to keep track, the state must be saved to non-volatile memory. Specifically in our implementation, an integer is saved to the chip's EEPROM and is incremented at every boot. (2) Since the amount of subroutines is of variable length, the control code must also be generated automatically. Our

implementation generated wrapper files written in C containing a switch-case that receives the integer from EEPROM as input and calls a different subroutine based on the value of the integer. (3) Finally, the subroutines themselves must also be automatically generated. Our framework generates an assembly file that contains the entire set of subroutines to be executed by the chip.

By passing multiple arguments to the framework a test set of arbitrary size can be specified. The scripts generate all the necessary wrapper files and subroutines. Thus, the chip can easily be programmed with a workload of several hundred test cases and left to run autonomously with no additional user input. The measurement PC resets the chip (causing it to execute the next subroutine), triggers the camera to take an image, saves the image and repeats the process until images are taken for the entire test set. The only real limitation for the size of the test set is the size of the program memory. Though our control code was written in C and included several libraries from `avr-glibc`, if memory constraints were an issue, the program control code could be rewritten more efficiently in assembly. Furthermore, it is possible to generate the subroutines on chip to further overcome limited program memory. However, this would increase the complexity of the program code and was not further investigated in this work. This methodology can also be adapted for debug interfaces such as JTAG, which provide ways to change program code on the fly, without reseting the chip.

IV. Practical Results

In this section we provide a case study of several representative exemplary test cases that demonstrate what is possible when selectively executing code, which targets specific parts of a CPU.

A. Identifying logic

Figure 1 demonstrates an obvious first choice for functional emission analysis. In this endless loop in `avr` assembly a relative jump is performed to the same address:

```
1  .global infloop
2  infloop: rjmp infloop ; rjmp to self
```

This image provides an important baseline for all other images, since all of the loops will contain at least one relative jump. This also means that these emissions will for the most part also be present in all other images. Since we are not interested in the emission of the `rjmp` instruction this image serves as a reference to which we can subsequently compute difference images.

In addition, this image demonstrates several other important characteristics of emission images. Non-logic circuits on a given IC do not necessarily switch with the clock-frequency. Certain non-logic circuits on the chip, such as the sense amplifiers and charge pumps, constantly have a current flowing through them and also constantly emit photons. Such circuits are easily identifiable because of their characteristic emissions in the resulting images and position relative to the synthesized

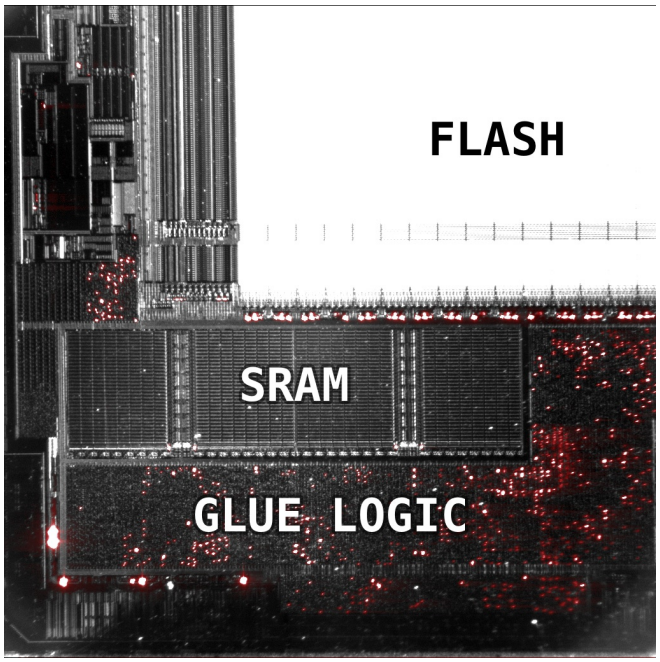


Fig. 1. Reflective light image of the chip's layout with an overlay of the optical emissions. The labels designate the basic functional groups of the IC that can be identified within the reflective light image. Within the overlaid emisisons, all the larger emission points along the perimeter of the glue logic are non-logic.

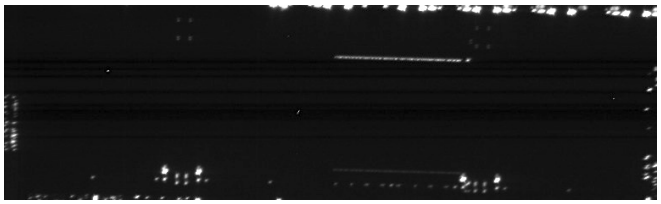
logic of the CPU, see Figure 1. In this fashion, by observing the optical emission images of a given subroutine, entire inactive regions and in certain cases active non-logic regions can be identified, thus reducing the search area.

B. Memory Map

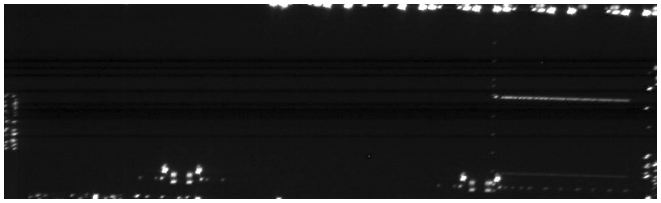
Determining the layout of memory is often an important first step for laser fault injection. In [11], the authors performed this task by introducing optical faults to the SRAM array. Instead, this analysis can be performed more efficiently by simply observing the photonic emissions of memory access to different parts of the address space. For our analysis we wrote a subroutine that received two parameters, a value and an address to which the value is written.

```
1  ; first  parameter: r25:r24 - addr
2  ; second parameter: r22 - value
3  .global memmap
4  memmap: movw r26,r24 ; move addr to X
5  loop:   st   X,r22 ; write value to [X]
6          rjmp loop
```

The ATMega328p has 2 Kilobytes of SRAM, which is mapped to `0x100` to `0x8FF` [12]. By performing memory access to the entire address space of the chip, as is shown in Figures 2(a) and 2(b), we were able to build up a complete map of the addresses of the memory lines within SRAM. The memory map of the address lines starting from `0x100` to `0x880` in 16 steps of 0x80, is shown in Figure 2(c). In most



(a) Memory access to 0x580



(b) Memory access to 0x700

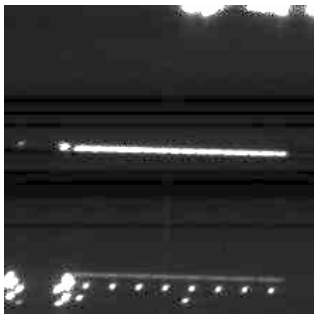
(c) Memory map of addresses within the SRAM

Fig. 2. Optical emission images of memory access to different memory lines of the SRAM

implementations the stack is set to `RAMEND`, i.e., `0x8FF` for the ATmega328p. If an attacker wished to manipulate stack variables, he would likely target this very area. Note, the unexpected location of `RAMEND`, i.e., the position of `0x880` relative to the position of `0x100`, in Figure 2(c).

Although it may be sufficient for an attacker to simply know the address of a given memory line in SRAM, we decided to determine the layout of every byte within a memory line. To determine the layout of the bytes within a given memory line we used the same subroutine as we did to map the memory lines, but instead accessed 16 consecutive bytes, i.e. `0x300`, `0x301...0x30F`. We used the store command, `st`, which allowed us to observe the column selectors and determine the positions of a byte within a line. Examples for memory access to `0x300` and `0x304` are shown in Figures 3(a) and 3(b),

(a) Memory access to 0x300



(b) Memory access to 0x304

Fig. 3. Memory access to different bytes within a single line of SRAM. Note the position of the 8 column selectors at the bottom of the memory bank.

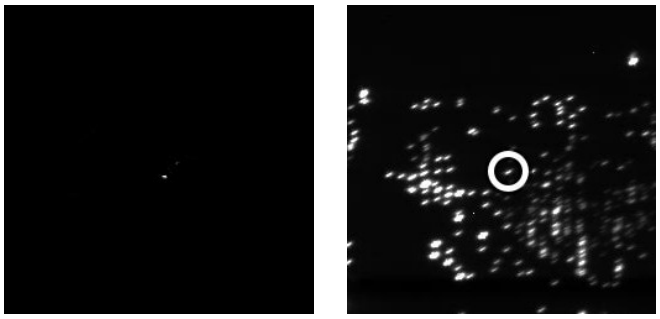
respectively.

Finally, after having determined the positions of the individual bits of each byte within the memory line, we determined the byte order within each memory line. To determine the byte order we observed the optical emissions resulting from storing bytes that have a Hamming weight of 1, see Figure 4.

Fig. 4. Byte order of a memory line. Emissions generated by storing the value `0x02` (00000010_2) to address `0x300` in SRAM. The optical emission pattern generated by a '0' is from the top-left to the bottom-right, whereas a '1' has emissions from the bottom-left to the top-right. Note the only '1' in the memory line is at the position of '`BIT1`'. '`LSB`' is the position of the first bit of the first byte in the memory line and '`BIT1`' is the position of the second bit of the first byte in the memory line.

C. Identifying Branching Logic

In [7], the authors proposed computing differences between two images to detect malicious alterations. Instead, we used a similar technique to compute the differences between images in which we selectively targeted different parts of the CPU's



(a) Difference image of Z-flag (b) Z-flag in the emissions of cpi

Fig. 5. Emissions of the SREG Z-Flag. Figure 5(a) shows the isolated emissions after computing the difference to the emissions of all other SREG flags. Figure 5(b) shows that the emissions in Figure 5(a) are also present in the emissions of the cpi loop.

logic. By iteratively computing the differences between a single input image and *multiple images* that target other parts of the CPU's logic, it is possible to identify emissions that uniquely belong to the logic targeted by the code in the input image. In this fashion, it is possible to identify specific execution logic and even registers.

We demonstrate that it is possible to identify individual status registers that are used in conditional branching. The `avr` architecture specifies an 8-bit status register (`SREG`). The `SREG` is made up of 8 individual flags: `C`, `Z`, `N`, `V`, `S`, `H`, `T`, `I`. Specifically, we will focus on the zero flag, `Z`-flag, as this is what is tested for the *branch-if-not-equal* instruction (`brne` in `avr` assembly). For each flag, `avr` assembly includes a separate set and clear instruction, i.e., `sez` and `clz` to set and clear the `Z`-flag.

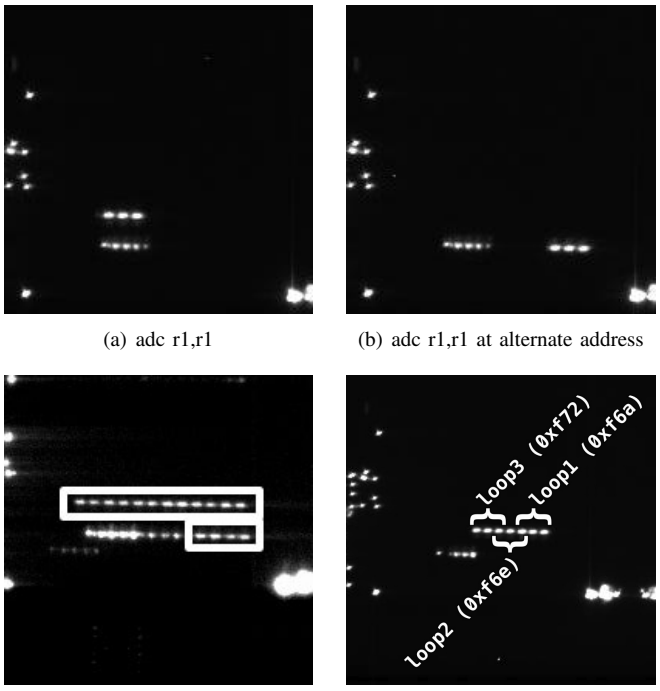
By computing the differences of the `setclrz` subroutine and the corresponding subroutines that set and cleared the `C`, `N`, `V`, `S`, `H`, `T`, `I` flags, respectively, we were able to isolate several points of emission on the whole chip, see Figure 5(a).

```
1  .global setclrz
2  setclrz: sez
3           clz
4           rjmp setclrz
```

We were then able to verify that these emissions are also present in the image taken of the following loop, see Figure 5(b):

```
1  .global cpiloop
2  cpiloop: ldi  r24,0x10
3  loop:    cpi  r24,0x10
4           rjmp loop
```

This loop performs *compare-with-immediate*, subsequently setting the Z-flag because the value in `r24` is equal to the immediate value. We could differentiate between loops that set Z-flag and loops, which did not by observing this single point of emission.



(a) adc r1,r1 (b) adc r1,r1 at alternate address

(c) Address-dependent emissions (d) Overlay of 3 contiguous loops

Fig. 6. Optical emissions of address-dependent logic. Figures 6(a) and 6(b) show the only differences in emissions between two identical loops at different memory locations. Figure 6(c) highlights the 16 emissions that exhibited an address-dependency to the executed code. Figure 6(d) shows an overlay of the optical emissions of three contiguous subroutines.

D. Identifying Execution Logic

One more goal that we wished to achieve was to identify areas of execution logic that could be attacked in order to change the control flow of the program by skipping, changing or corrupting instructions. Certain architectures such as the `AVR` architecture of the ATMega328p do not include exception vectors in the case that an undefined instruction is executed. Such architectures are thus particularly vulnerable to, for example, optical fault attacks that can succeed simply by selectively corrupting opcodes as they are executed. By generating multiple, identical subroutines that are contiguous in the program code, it is possible to isolate address-dependent logic. Specifically, we were able to identify part of the chip, directly adjacent to the flash (the program memory), that exhibited a dependency to the address of the executed instructions.To isolate such emissions, it is necessary to execute identical subroutines so that the only difference between the subroutines is the physical address of the executed code, see Figures 6(a) and 6(b).

Observing the emissions for multiple identical subroutines enabled us to identify all 16 points of the chip that exhibit address-dependent emissions, see Figure 6(c). By analyzing the emissions of the following code, it is possible to understand the nature of the emissions and their address dependency:

```
1       ldi r17, 0x01 ; 0xf6a - loop1
2       rjmp .-4      ; 0xf6c
3       ldi r17, 0x02 ; 0xf6e - loop2
4       rjmp .-4      ; 0xf70
5       ldi r17, 0x04 ; 0xf72 - loop3
6       rjmp .-4      ; 0xf74
```

The code contains 3 loops generated by the framework at the following addresses in the program code: `0xf6a`, `0xf6e`, `0xf72`. Each opcode is 16-bit in length, and each loop is two opcodes long. The ATmega328p datasheet [12] states that the next instruction is fetched from program memory in parallel while the current instruction is executed. This means while the `rjmp` is being executed at address `0xf6c`, the next instruction, `ldi r17, 0x02` at `0xf6e` is being fetched. Hence, each loop's control flow results in 3 16-bit instructions being fetched from program memory. The ATmega328p's flash is structured in 16, 16-bit columns, i.e., each column is one 16-bit opcode wide. For this reason, the emissions are likely part of the column control logic of the flash. Each of the 16 points corresponds to one column in flash. Thus when 2 consecutive opcodes are executed in a loop this results in 3 column emissions, since the first opcode of the following loop is also fetched, see Figures 6(a) and 6(b). This was also verified for loops of varying length, varying positions within the program code and with code containing 32-bit opcodes. Given such information and by targeting this area, an attacker can potentially corrupt, introduce concurrency to the process of fetching opcodes or overwrite opcodes with others during execution.

V. Conclusion

Since our methodology employs backside analysis, additional layers of metalization, such as active meshes, have no effect on the backside emissions of the IC. As feature sizes continue to shrink, transistor voltages drop. Lower voltages generally result in fewer photons. Furthermore, in a real world scenario where an attacker cannot execute his own code, acquiring even a single image can be made infeasible by implementing delays and thus reducing loop frequencies. For this reason, it is also important to consider the control flow of an implementation. A busy-waiting-based implementation may reveal memory contents that an interrupt-based implementation would not.

In this work we presented our methodology for performing automated functional analysis of ICs. Though our analysis focuses on the `AVR` architecture and specifically the ATmega328p, it is also applicable to other architectures and ICs. Much of the analysis can even be performed on platforms and execution environments of much greater complexity. JavaCard, for example, has been shown to be susceptible to software exploits. Because of how short it is, our code is well suited as a payload in complex execution environments. Also, since functional integrated circuit analysis can be performed through the backside it is not impeded by the active meshes of modern secure ICs.

Many state of the art attacks assume an exhaustive search is necessary to identify potential targets on the chip. Functional integrated circuit analysis provides a far more efficient way to find potential points of interests. The exhaustive search can be greatly reduced, if not eliminated entirely, since our methodology can even identify single points within the chip's logic. We provide practical examples of how our methodology was applied to identify the exact memory layout, status registers used in conditional branching and logic used to fetch opcodes from program memory.

VI. Future work

We are currently in the process of adapting many of the findings as a basis for cryptographic attacks. For example, software implementations that implement the AES SBOX in program memory can be defeated by monitoring access to the flash column logic. Alternatively, if the AES SBOX is stored in SRAM, monitoring access to the row and column selectors is sufficient to defeat the implementation. Finally, status registers, such as the Z-Flag, can be monitored to reveal the key during the `AddRoundKey` operation of AES.

VII. Acknowledgements

The authors would like to acknowledge the support of the German Federal Ministry of Education and Research and the Helmholtz Research School on Security Technologies. We would also like to thank the Semiconductor Devices research group at TU Berlin for sample preparation and colleagues Collin Mulliner and Christoph Bayer for their insightful comments during the paper writing process.

References

[1] S. Mangard, E. Oswald, and T. Popp, *Power Analysis Attacks - Revealing the Secrets of Smartcards*, 1st ed. Springer, Mar. 2007.

[2] W. Rankl and W. Effing, *Smart Card Handbook*, fourth edition ed. Wiley, 2010.

[3] R. Newman, "Visible Light from a Silicon p-n Junction," *Physical Review*, vol. 100, pp. 700–703, Oct. 1955.

[4] J. Bude, N. Sano, and A. Yoshii, "Hot-carrier luminescence in Si," *Physical Review B*, vol. 45, no. 11, p. 5848, 1992.

[5] J. Ferrigno and M. Hlaváč, "When AES blinks: introducing optical side channel," *IET Information Security*, vol. 2, no. 3, p. 94, 2008.

[6] "Advanced Encryption Standard (AES)," Nov. 2001.

[7] P. Song, F. Stellari, D. Pfeiffer, J. Culp, A. Weger, A. Bonnoit, B. Wisnieff, and M. Taubenblatt, "MARVEL — Malicious alteration recognition and verification by emission of light," *Hardware-Oriented Security and Trust (HOST), 2011 IEEE International Symposium on*, pp. 117–121, 2011.

[8] S. Skorobogatov, "Using Optical Emission Analysis for Estimating Contribution to Power Analysis," *Fault Diagnosis and Tolerance in Cryptography, FDTC 2009*, pp. 111–119, 2009.

[9] ——, "Optical Fault Masking Attacks," *Fault Diagnosis and Tolerance in Cryptography, FDTC 2010*, pp. 23–29, 2010.

[10] K. Nohl, D. Evans, Starbug, and H. Plötz, "Reverse-engineering a cryptographic RFID tag," in *Proceedings of the 17th USENIX Security Symposium*. USENIX Association, Jul. 2008.

[11] S. Skorobogatov and R. Anderson, "Optical fault induction attacks," *Cryptographic Hardware and Embedded Systems, CHES 2002*, pp. 31–48, 2003.

[12] Atmel Coorporation, "Atmel ATmega328P Datasheet," http://www.atmel.com/dyn/resources/prod_documents/doc8271.pdf, May 2011.