



Hochschule für Angewandte Wissenschaften Hamburg
Hamburg University of Applied Sciences

Bachelorthesis

Dominik Björn Zürner

A Demonstrator for Optical Fault Injection Attacks

Dominik Björn Zürner
A Demonstrator for Optical Fault Injection Attacks

Bachelorthesis based on the study regulations
for the Bachelor of Science degree programme
Informations- und Elektrotechnik
at the Department of Information and Electrical Engineering
of the Faculty of Engineering and Computer Science
of the Hamburg University of Applied Sciences

Supervising examiner : Prof. Dr. Heike Neumann
Second Examiner : Dr.-Ing. Wolfgang Tobergte

Day of delivery: March 23, 2018

Dominik Björn Zürner

Title of the Bachelorthesis

A Demonstrator for Optical Fault Injection Attacks

Keywords

Hardware security, fault injection, semi-invasive, optical fault attacks, backside, micro-controllers, embedded systems, RSA, RSA-CRT BellCoRe-Attack

Abstract

This thesis describes the design of a demonstrator for optical fault injection attacks. To do this, a microcontroller was examined for vulnerabilities for this kinds of attacks. Afterwards a demonstrator based on an cryptographic algorithm was created.

Dominik Björn Zürner

Titel der Arbeit

Ein Demonstrator für die Fehlerinjektion durch Lichtattacken

Stichworte

Hardware-Sicherheit, Seitenkanalattacken, Mikrocontroller, eingebettete Systeme, RSA, RSA-CRT, BellCoRe-Attacke

Kurzzusammenfassung

Diese Arbeit beschreibt die Entwicklung eines Demonstrators für die Fehlerinjektion durch Lichtattacken. Hierfür wird zuerst ein Microcontroller auf die Anfälligkeit für diese Art von Attacken untersucht. Anschließend wird auf Basis eines kryptografischen Algorithmus ein Demonstrator entwickelt.

Contents

List of Figures	6
1. Introduction	8
2. Background	10
2.1. Integrated Circuits	10
2.2. Attacks on Secure Devices	12
2.2.1. Side-Channel Analysis	13
2.2.2. Fault Injection	13
2.2.3. Physical Attacks	14
2.2.4. Optical Fault Injection	16
2.2.5. Effects of Fault Attacks	19
2.3. RSA	20
2.3.1. Signing Messages	22
2.3.2. RSA-CRT	22
2.3.3. BellCoRe Attack	23
2.4. Countermeasures	24
3. Practical Attack	28
3.1. Device Architecture	28
3.2. Decapsulation	30
3.3. Target Analysis	32
3.4. Setup and Experiments	34
3.4.1. Experiment 1: Finding a Vulnerable Spot	35
3.4.2. Experiment 2: Fault Behavior of the Flash Controller	38
3.4.3. Experiment 3: Influence of the Light Intensity	40
3.5. Conclusion	42
4. Demonstrator	43
4.1. Use Case	43
4.2. Implementation	45
4.2.1. Microcontroller	47
4.2.2. Host Application	50

4.3. Outcomes	53
5. Conclusion	56
Bibliography	57
A. Appendix	61

List of Figures

1.1. User interface of the old demo	9
2.1. Cross-section of semiconductor die under a scanning electron microscope . .	10
2.2. Cross section of IC package	11
2.3. Modification of an IC using FIB	15
2.4. Cross-section of a CMOS Inverter	16
2.5. Schematic of a CMOS Inverter	17
2.6. Riscure Laser Station 2 [Ris18]	18
2.7. Instruction corruption	19
2.8. Encrypted message transfer using RSA	20
2.9. Active shield on an Atmel ATAES132	24
2.10. Simple power analysis on square and multiply algorithm	26
2.11. Block diagram of a RSA decryption method with verification [Tob11]	27
3.1. Pipeline of the ARM Cortex-M3	29
3.2. X-ray image of an STM32F103RBT6 with QFP package	30
3.3. JetEtch Pro Decapsulation System by Nisene Technology	31
3.4. Decapsulated Microcontroller	31
3.5. Die of STM32F103RBT6 with annotations	32
3.6. Backside of STM32F103RBT6 with annotations	33
3.7. Setup for experiments with xenon strobe light	34
3.8. Reflect photons on the leadframe	36
3.9. Covered chips using tape	37
3.10. Flowgraph for flash-tester.py	38
3.11. Relationship between distances of the strobe-light and the occurred faults . .	40
4.1. Applied authentication scheme using RSA-CRT	45
4.2. Flowchart of the microcontroller application	48
4.3. UART frame format for Client-to-Host messages	49
4.4. Flow graph of the host applications main loop	50
4.5. Different status indicators of the battery monitor	51
4.6. Page 2 with communication display and statistic	52
4.7. Page 3 with key extraction and cloning	52

4.8. Both boards of the demonstrator in 3D printed cases	54
4.9. Complete demonstrator kit in case	55

1. Introduction

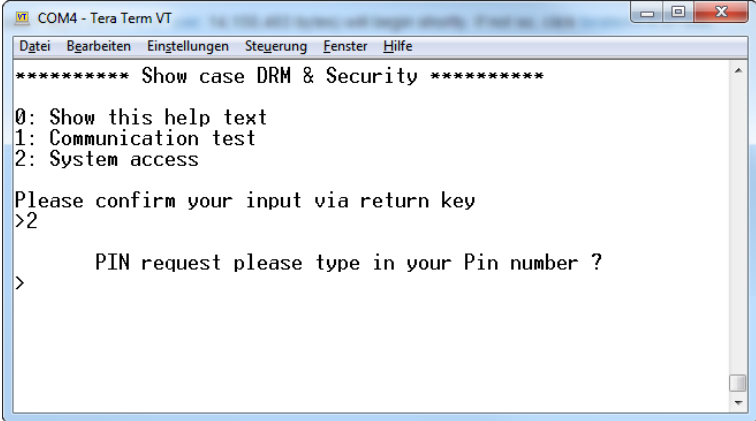
Embedded systems are part of everyday life in today's world. They are used in household appliances like washing machines and freezers but also in security-critical areas like access control, cars, and smart cards. This growing trend is expected to continue with the Internet of Things (IoT). Robust and mathematically secure cryptographic algorithms ensure security of such devices. However, these methods have only been tamperproof against mathematical attacks, not against attacks that target the implementation of these algorithms.

Security flaws in hardware are even more dangerous than in software, since they typically cannot be patched. Chips on embedded devices are usually easy to access, which makes many more attack vectors possible. Hardware attacks can be used to manipulate the behavior of a system or to obtain secret information like a cryptographic key. This is done by monitoring and analyzing side channels like the power consumption, injection faults, or a combination of both.

The objective of this thesis is to design a new demonstrator for NXP Semiconductors Germany GmbH on the topic of optical fault injection attacks. The main goal is to provide an understanding about the threads and impacts of these types of fault attacks. The demonstrator must be comprehensible to a broad-based target group. Customers and visitors should be convinced about the importance of using secure components in security-related products. The demonstrator will be also used for in-house training for employees with a security background.

NXP already has a demo based on a microcontroller with 8051 architecture and command line user interface. The new demonstrator is expected to have a clear graphical user interface (GUI). NXP should be able to ship the demo to different sites, so it should be portable and fit into a small case. Also, a previously uninvolved person must be able to set it up by following a short user guide. The demonstrator must be reliable and reproducible. Five working demos are required. One main challenge is to make the faults occur relatively quickly. In the real world, fault attacks are usually completely automated, and it might be days or weeks before an exploitable fault occurs

The 8051 architecture was invented in the 1980s, but it is still used in a variety of today's microcontrollers. However, the architecture is now considered obsolete. Therefore the more modern ARM Cortex-M3 architecture was chosen for this demo. Today, 130 billion Advanced



```
COM4 - Tera Term.VT
Datei Bearbeiten Einstellungen Steuerung Fenster Hilfe
***** Show case DRM & Security *****
0: Show this help text
1: Communication test
2: System access
Please confirm your input via return key
>2
    PIN request please type in your Pin number ?
>
```

Figure 1.1.: User interface of the old demo

RISC Machines (ARM) processor have been sold, and 90% of all mobile devices are powered by an ARM processor [Lim17]. They are used in a wide range of applications, such as motor drives, application control, handheld devices, equipment, personal computer (PC) peripherals, and medical equipment.

This thesis starts in chapter 2 with background information about integrated circuits that is essential in understanding the subsequent introduction to attacks on secure devices. Chapter 3 describes the process of preparing the target and finding a vulnerability on the chip. Chapter 4 examines the design of the built demonstrator for optical fault injection attacks.

2. Background

2.1. Integrated Circuits

An integrated circuit (IC) is a set of electronic circuits consisting of parts such as resistors, capacitors, and transistors. These circuits are implemented on a small piece of semiconductor material called die. Semiconductors are usually composed of silicon. These chips are much smaller, faster, and cheaper than those constructed of discrete components [Wik18a].

ICs are structured in several layers. Drain and source regions of transistors are diffused into the semiconductor material. Isolations for the gates are implemented by depositing and etching silicon nitride using masking. The structural size of regions are as small as a dozen nanometers. On the top, several metal layers interconnect the different electronic components to a circuit. Security ICs also have several metal layers as shields. Metal shields are further described in Chapter 2.4. Figure 2.1 illustrates the cross-section of a semiconductor die under a scanning electron microscope (SEM) with the different metal layers marked as M1-M6.

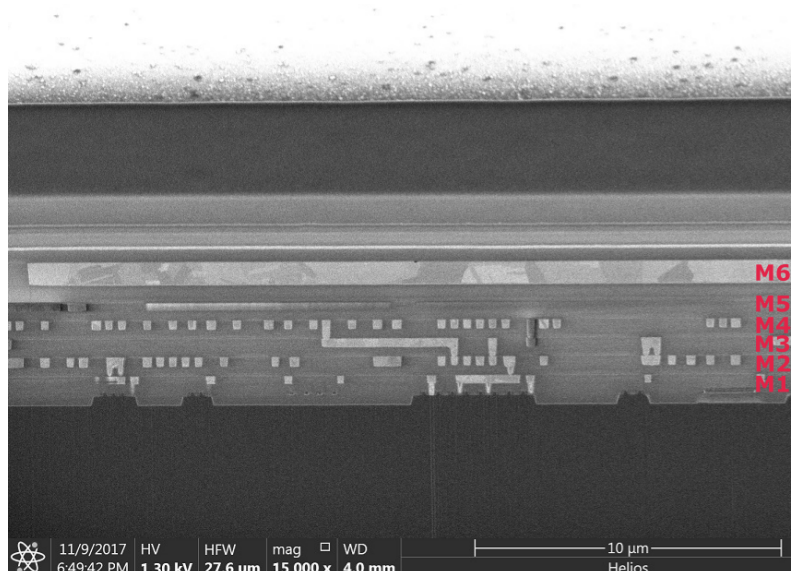


Figure 2.1.: Cross-section of semiconductor die under a scanning electron microscope

ICs are encapsulated in an enclosure, also known as a package. This packaging protects the die from damage and environmental influences. The die is glued onto a leadframe. Bondwires are used to connect the pads on the die with the individual pins of the leadframe. In the 1980s, chip packages were composed of metal or ceramic, but now epoxy resins are typically used.

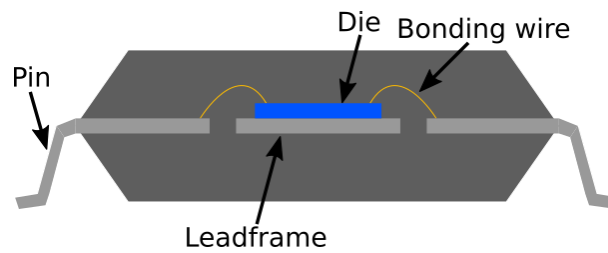


Figure 2.2.: Cross section of IC package

2.2. Attacks on Secure Devices

The first type of hardware attack was likely discovered in 1943, when an engineer at Bell Labs found a flaw in a cryptographic teletype [Age07]. Rather than attacking theoretical weaknesses in the algorithms, the information could be obtained by monitoring radio frequency (RF) emissions - a so-called side channel attack. During the late 1990s, the scientific community became aware of hardware attacks as a method to compromise cryptographic systems [Koc96].

Evaluating the security of embedded devices requires a wide knowledge of hardware and software. A system normally can only be as secure as its weakest component. Five [Sko05] major attack categories on embedded devices have been defined as follows:

Software attacks exploit vulnerabilities found in the protocols and cryptographic algorithms or their implementation.

Fault generation uses abnormal environmental conditions to cause malfunctions in the processor to exploit them.

Eavesdropping monitors a variety of information provided indirectly by an embedded device through supply and interface connections and RF transmission. This information can have statistical correlation to secret keys or similar elements.

Reverse engineering understands operations, functions, timing, and signal paths of the semiconductor.

Microprobing¹ accesses the chip surface directly to observe, manipulate, or interface with the chip.

These attacks can also be combined. Attackers could use eavesdropping on side channels to see how far a program has processed. Attackers can use this information to inject a fault at a certain location in the program. For example, a password authentication could be bypassed this way. The fault could be injected during the comparison between the correct password and the one provided by the user.

Hardware attacks can be classified with following attributes:

Invasive involves decapsulation of the chip to reverse engineer, probe, or modify it.

Semi-invasive also involves decapsulation but without making direct contact to the die.

Non-invasive includes analysis of side-channels like timing, power consumption, and electromagnetic fields. This type also includes glitching on power or clock supply.

¹Physical attacks would be a better description as it also involves other attacks.

Local includes precise attacks on deliberate areas of the chip that implement a certain function.

Global relates to global parameters such as voltage and clock and results in random impacts. Adversaries are not in charge of which functions of the chip they are attacking.

2.2.1. Side-Channel Analysis

Non-invasive attacks are also known as side-channel attacks. They refer to a variety of information emitted indirectly by an embedded device. This information, such as time delays, power consumption, and electromagnetic emission, can be statistically related to the program flow and be used to restore secret information, as in [Age07], or for further attacks. Eavesdropping on these side-channels can be done without modifying the hardware. The device usually will not be damaged permanently.

General side-channel attacks include:

Simple power analysis monitors variations in the power consumption of the device. The power consumption varies with the executed instructions of the controller. Therefore, by recognizing patterns of a cryptographic algorithm, secret keys can be extracted - for instance, in the multiply-and-square algorithm of an RSA [Neu16].

Differential power Analysis occurs in two phases. First, a large amount of power traces is taken during the cryptographic algorithm. Secondly, the data is analyzed with a variety of statistical methods. In this way, secrets can be obtained from measurements, which contain too much noise for simple power analysis.

2.2.2. Fault Injection

A fault attack describes an active attack that aims to inject errors into a target device. These errors can be accomplished by several tampering means. Typical target circuits are central processing unit (CPU) registers, memories, or program counters. Depending on the precision of the applied technique, the effects can vary from flipping certain single bits to random values in several bytes.

Common fault attacks include the following:

Clock glitching. This attack induces faults by sudden and short increases of the clock signal. Depending on gate length and internal clock distribution, the manufacturer specifies a maximum clock frequency. When the clock signal is too fast, the flip-flops are

triggered before the input signal is stable, resulting in a metastable state. In this manner, the executed instructions are disturbed or prevented, as there is not enough time to complete them before the next clock cycle occurs. After the glitch, the processor operates normally. Combining this attack with simple power analysis offers the opportunity to launch the attack at certain points in the code. Clock glitching attacks are simple to perform, as no modification of the chip itself or specialized hardware is needed. [SGD08]

Power glitching. This attack is similar to clock glitching. The fault is induced by abrupt changes in the supply voltages. Voltage glitches modify the timing properties of CMOS-logic, causing faults like instruction skipping. Furthermore, memories need a stable power supply to operate correctly. Sensitive amplifiers are used to read from flash memory, and high-voltage pumps are used to write to it. A reduced supply voltage might result in incorrect data being read or written to the memory. Timing and duration of the glitch must be adjusted until the desired fault is achieved. [BECN⁺04]

Temperature attack. An IC is only specified for a specific temperature range. Outside this range, proper function is not guaranteed. Conceivable effects can be the random modification of RAM cells through heating. The temperature can vary to a point where writing to memory works but reading does not [BECN⁺04]. Another attack aims on DRAM memory. If a device is turned off, the contents of DRAM vanish gradually over a period of seconds to minutes. This process can be slowed down by cooling the chip to around -50°C . Contents like cryptographic keys can then be read out. [HSH⁺08].

Electromagnetic-Pulse. This attack uses electromagnetic pulses to influence memory cells or general functions. An active coil is used to create a magnetic field that induces Eddy currents on the surface near the conducting materials. Using this technique precisely, local attacks can be carried out. The equipment needed for this attack is relatively cheap, and no decapsulation of the IC is needed. The attacker does need to know the basic layout of the die to position the coil.

Optical fault. A semi-invasive fault injection attack is carried out through light pulses or high-intensity lasers. This type of attack is described in more detail in section 2.2.4.

2.2.3. Physical Attacks

A few more hardware attacks must be mentioned. The following invasive attacks are used to modify the chip or to directly interface with it:

Focused ion beam. A focused ion beam (FIB) can be used directly to modify an integrated circuit. It can be used to add or cut electrical connections to make individual elements

like bus lines accessible for measurements. Figure 2.3 illustrates modifications done with a FIB on (1) a new connection between two traces and (2) a cut through a trace.

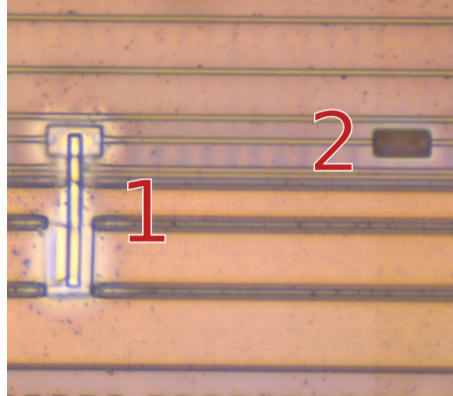


Figure 2.3.: Modification of an IC using FIB

Microprobing. Needles are placed on traces of the chip to eavesdrop internal signals. [Sko17]

Reverse engineering. Photons are emitted by switching transistors. These photons can be detected and used as a side channel. Because of the metal layer on the front side, these attacks would be carried out on the backside. Invasive modification of the IC, like thinning of the die, can be required. [Sch14]

2.2.4. Optical Fault Injection

During nuclear tests in the 1950s, abnormal behavior was observed on semiconductors [Wik18b]. These problems also occurred in space and aerospace electronics caused by cosmic radiation. The generic term for effects caused by ionizing particles striking through sensitive electronic devices is Single Event Effect (SEE). An important subcategory of SEE is Single Event Upsets (SEU), which describes a state change of an electronic circuit. These events are also called soft errors, as they are only temporary and do not physically damage the device. In the mid-1960s, pulsed lasers were used to simulate these effects on semiconductors [Hab65]. The single board computer Raspberry Pi 2, released in 2015, is also affected by SEUs. The computer crashes if it is photographed with a xenon flash. The voltage regulator on the board has a chip scale package. The die is mounted directly onto the printed circuit Board (PCB) upside down without any enclosure. Through a small gap between die and PCB, the photon could penetrate the silicon [Rak16]. In 2002 [SA⁺02] published the first optical fault injection on the SRAM of a microcontroller.

When an ionizing particle like a photon hits a part of a semiconductor, it is called the photoelectric effect. If the photons strike the silicon of a device with a higher energy than the bandgap, electron-hole pairs are generated along the light beam's path. Usually all of these pairs recombine, and no effect on the IC is noticeable. The most sensitive regions of an IC are reversed biased pn junctions. If the photon hits the junction, like in figure 2.4, the holes and electrons cannot recombine because of the electric field. The holes drift to the p-zone and the electrons to the n-zone. This process, called funnelling, leads to a fast transient current through the struck junction. Then, diffusion is where the current decreases slowly until all charges are collected, recombined, or diffused away by the junction area [WA08].

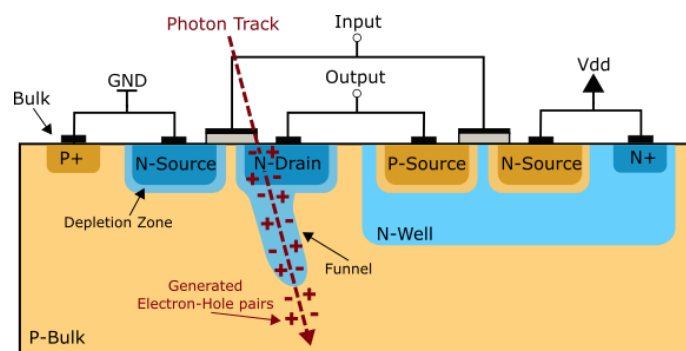


Figure 2.4.: Cross-section of a CMOS Inverter

Inside a logic circuit, this process can cause an SEU, which means that the state of a circuit changes unexpectedly. Figure 2.5 illustrates the schematic of the CMOS Inverter in Figure 2.4. The input of the inverter is High and the output state is Low. Hence, the PMOS transistor

is On and connected to the positive supply voltage Vdd. The NMOS transistor is Off. If a photon hits the drain (marked red) of the reversed biased NMOS, as illustrated in Figure 2.4, a transient current flows through the struck transistor. As a result, the restoring transistor sources a current to compensate the particle-induced current and induces a voltage drop at its drain [Rec10]. This action leads to a temporary change of the output from High to Low. When the input of the inverter is High, the vulnerable area is the source of the closed PMOS transistor.

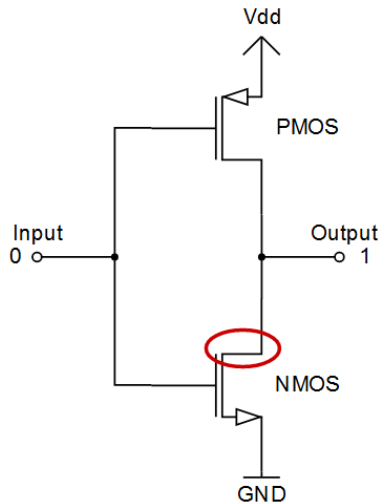


Figure 2.5.: Schematic of a CMOS Inverter

Optical fault injections can be carried out either from the front or back of the die. Frontside attacks on silicon can be implemented with wavelengths from $500nm - 800nm$, depending on the chip's design. Backside attacks on modern chips are more practical due to the increasing number of metal layers on the chip itself. The metal layers make the target denser for a light beam. Wavelengths for backside attacks reach from $950 - 1050nm$ [Ana14]. For wavelengths over $1100nm$, silicon becomes transparent and the photons pass through without being absorbed. With backside attacks, the transistor can be reached directly through the silicon on which the transistors were built.

Optical fault attacks can be executed with various light sources with the appropriate wavelength. A xenon² photo-flash lamp of a camera is the simplest form of this technique, as it only requires cheap equipment and a decapsulated chip. The light of a photoflash contains the wavelengths necessary for front- and backside attacks. However, without any further equipment, only precise global attacks can be executed. Therefore, it is not detectable which areas and function of the chip are attacked. As a result, effects can be extremely random and not reproducible, or multiple errors can take the chip into a non-operational state.

²Light-emitting diode (LED) flashes lack the necessary near-infrared light spectrum [KBCK13].

A more advanced setup uses lasers in combination with an optical microscope to obtain extremely focused light beams. Depending on the optics, the spot size of a laser can be as small as $1\mu m$, making it possible to attack certain parts of a circuit without disrupting the whole chip. Lasers can also help in understanding what caused the fault.

Figure 2.6 illustrates a commercially available setup for the evaluation of optical fault attacks. It is equipped with a motorized XY-Stage. The setup can be used for stepping over a chip's surface and automatically testing various spots for vulnerability. Two individual lasers offer the opportunity to attack two locations at the same time. An infrared camera can be used to navigate over the chip during backside attacks.



Figure 2.6.: Riscure Laser Station 2 [Ris18]

2.2.5. Effects of Fault Attacks

The following effects of fault attacks are possible on a microcontroller:

Data/code modification affects stored data in the memory, which particularly. This applies in particular to SRAM or FLASH memory. This can variate from unpredictable values at various locations to flipping specific bits or bytes.

Register modification directly affects CPU registers. Especially in a PC, this modification offers an attacker the opportunity to jump to any location in the code and control the program flow.

CPU execution corruption changes the instruction of an opcode. Depending on the precision of an attack, this corruption can skip an instruction or change it to a legit random or specific value.

Instruction	Destination Address	15	12	11	cond	8	7	imm8	0
1. BLE	0x080004ca	1	1	0	1	1	1	0	1
2. B	0x080004ca	1	1	1	1	1	1	0	1
3. BLE	0x0800048a	1	1	0	1	1	0	1	1

Figure 2.7.: Instruction corruption

Figure 2.7 illustrates an example of instruction corruption. This example uses a Thumb2 16-bit conditional branch. These branches are conditional on the status register. Bit 15-12 of the opcode encodes the instruction as conditional branch, bit 11-8 encodes the condition, and bit 7-0 encodes the offset from the PC to the destination address. Instruction (1) is a branch if less or equal instruction, which is common at the end of a loop just after a comparison of the loop counter. The modification of a few bits can cause a completely different behavior of the program. Instruction (2) changes from branch if less or equal to branch always by flipping a single bit in the condition field. Instruction (3) changes the address field to jump to a different address in the code. An adversary can use these effects to either bypass access or right control verification. The attacker can also generate faulty encryptions or signatures, from which secret keys can be extracted.

2.3. RSA

RSA is an asymmetric encryption and signature scheme. It was first published in 1978 [RSA78] and is named after its inventors Ronald L. Rivest, Adi Shamir, and Leonard Adleman. RSA, through its asymmetry, includes a public key for encryption and a different key, called the private key, for decryption. It is not feasible to obtain the private key from the public key in a reasonable amount of time. Thus, for $m^{k_e} = c \pmod n$, where m is the plain text, k_e the encryption key and c the cipher, there is a decryption key k_d so that $c^{k_d} \pmod n = m$.

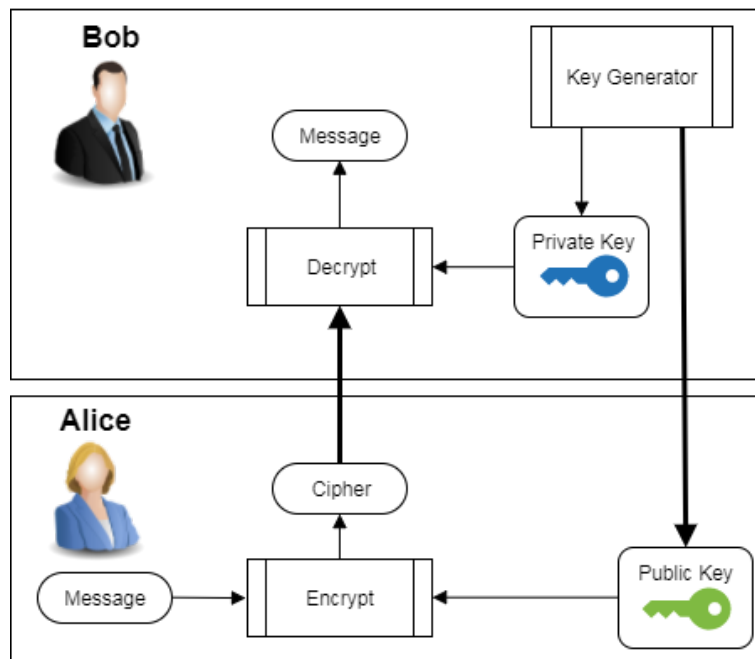


Figure 2.8.: Encrypted message transfer using RSA

For example, if Alice wants to send an encrypted message to Bob, he first needs to generate a key pair. The scheme of this system is illustrated in Figure 2.8. To create a key pair, two large prime numbers p and q are multiplied. The product $n = p * q$ is called the modulus and is part of the public key. In addition $\phi(n) = (p - 1) * (q - 1)$ is calculated, where ϕ is the amount of integers that are relatively prime and smaller to n .

The second part of the public key is the public exponent e . Usually, the values $e = 3$, $e = 17$ or $e = 2^{16} + 1$ are chosen because of the low hamming weight of the number, which results in a faster computation of the algorithm 1.

The private key d is determined by $ed \equiv 1 \pmod{\phi(n)}$ [Neu16].

Alice can now encrypt a message by using Bob's public key:

$$c := m^e \pmod n \quad (2.1)$$

Bob can decrypt the message using his private key, which only he is aware of:

$$c^d \pmod n = m \quad (2.2)$$

The size of the modulus n is the key size. The security depends on the difficulty of factorizing the product of large prime numbers. To date, no mathematical procedure exists for factorizing a large n , without knowing p or q , within a reasonable time. . But considering progress in cryptanalysis, the recommended key length for RSA will be extended. The German federal office for information security (BSI) recommends a key length of 2,000 bits until the year 2022, when the key size should be increased again [fSid18].

Implementation

As depicted in formula 2.1 and 2.2, exponentiations are required for RSA. As the exponents are several hundred digits long, these computations would take a long time in a naive implementation. A 32-bit CPU can usually only multiply two 16-bit numbers in one instruction cycle. As the size of the key and the messages is usually much longer, an algorithm is required to perform exponentiations.

Algorithm 1 depicts a square and multiply algorithm, an efficient method for exponentiating by squaring. The exponent k is used in its binary form.

Algorithm 1: Square and Multiply Algorithm

Input : x, k, n

Output: $result = x^k \pmod n$

```

1  $result = 1$ 
2 for  $i = Len(k)..0$  do
3    $result = result^2 \pmod n$ 
4   if  $k_i = 1$  then
5      $result = result * x \pmod n$ 
6   end
7 end

```

2.3.1. Signing Messages

RSA can also be used to sign a message. Suppose Alice wants to send message to Bob. Alice can calculate the signature of message $s = m^d \bmod n$, using her private key. She then can send the signature and unencrypted message to Bob. Bob can then use the public key to verify the message. Therefore, he can apply $s^e \bmod n$ using Alice's public key. If the result matches the actual message, he knows that the message could only have been sent by Alice, as long as she is the only person who is in possession of her private key.

2.3.2. RSA-CRT

Decryption or signing can be made even more efficient by using the Chinese Remainder Theorem (CRT). The receiver of the message, in possession of p and q , can calculate $c^d \bmod p$ and $c^d \bmod q$. Using these intermediate results, the receiver can calculate $c^d \bmod n$ by using the CRT. As p and q are much smaller than n , this approach is much faster than calculating $c^d \bmod n$ directly.

Algorithm 2: RSA-CRT Algorithm

Input : c, d, p, q, n

Output: $m = c^d \bmod n$

- 1 $d_p = d \bmod (p - 1)$
 - 2 $d_q = d \bmod (q - 1)$
 - 3 Find u and v with $1 = u * p + v * q$ using the extended Euclidean algorithm
 - 4 $sig_p = c^{d_p} \bmod p$
 - 5 $sig_q = c^{d_q} \bmod q$
 - 6 $m = u * p * sig_p + v * q * sig_q$
-

2.3.3. BellCoRe Attack

The BellCoRe attack was published in 1997 by Boneh, DeMillo and Lipton from Bell Communications Research Inc. also known as BellCoRe [BDL97]. The attack uses fault injection to provoke an error in a RSA-CRT algorithm. The result of this faulty signature can lead to the leakage of the private key. As depicted in Algorithm 2, the computation for $sig = c^d \bmod n$ is split into two parts, sig_p and sig_q . The attack is based on injecting a fault during the computation of these partial results and leaving the other one correct.

Let $\hat{sig} = u * p * sig_p + v * q * \hat{sig}_q$ be a signature with the faulty \hat{sig}_q .

A subtraction of a correct signature with a faulty one produces:

$$sig - \hat{sig} = (u * p * sig_p + v * q * sig_q) - (u * p * sig_p + v * q * \hat{sig}_q) = v * q * (sig_q - \hat{sig}_q)$$

It is most likely, that $sig_q - \hat{sig}_q$ is relatively prime with p and q . The term $1 = u * p + v * q$ implies $v * q \equiv 1 \pmod p$, which means $v * q$ cannot be a multiple of p . As q is a factor of n , the computation of the greatest common divisor of $sig - \hat{sig}$ and n will reveal q .

$$GCD(sig - \hat{sig}, n) = q$$

The modulus n can now be easily factorized. To summarize, the private key of a RSA cryptosystem can be determined by using a correct signature and one where a fault happened during the computation of only one of the both partial signatures. It even doesn't matter how often a fault happens during the computation of sig_q .

2.4. Countermeasures

Secure embedded systems are critical for financial institutions, retailers, and governments. Therefore, techniques have been developed to provide protection against hardware attacks. Different certifications, like the ISO/IEC 15408, also known as the Common Criteria, evaluate and audit security properties of IT products. There are two approaches to prevent attacks. One is to prevent attacks from occurring in the first place by making the job of an adversary harder. The other is to detect attacks and react in an appropriate way. For example, a device could delete all its secret data after a certain amount of monitored tampering attempts. Since no countermeasures can prevent all attacks, various techniques are combined. An adequate trade-off between efficiency and security must be chosen.

Common countermeasures include the following:

Metal shields on the top layer of the die are used to deny access to the circuits and to detect intrusions. Their purpose is to prevent all types of semi- and full-invasive attacks, including probing, fault attacks, and optical fault injection. They also hamper reverse engineering. These shields can be implemented as passive, and they measure the capacitive load of the mesh. However, passive shields can be defeated as they must tolerate some variations in the quantities to be measured. On active shields, random bit sequences are put on the input of the trace and compared for faults on the output. Figure 2.9 illustrates an active shield on top of a secure IC. The randomized layout makes it harder to guess the pattern bypass of the shield [BCD⁺12]. It is impossible to find any hints about the structures underneath it with a microscope. At the marked spot, a trace of the shield was cut using the FIB. The chip was not operational afterward.

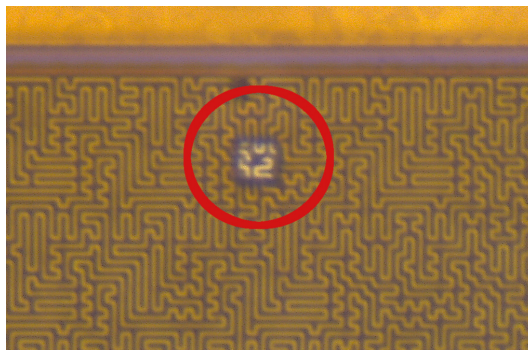


Figure 2.9.: Active shield on an Atmel ATAES132

Tamper detectors for voltage, temperature, frequency, and light are used to detect various fault attacks.

Randomized chip design is another way to make reverse engineering of the chip harder.

The key is to renounce the use hard-coded macros during the Very-Large-Scale Integration (VLSI) design and to use decorrelated glue logic instead. Some manufacturers are tempted to use the same already laid-out and certified crypto processor block on various products. However, this design makes it easier to identify these function blocks, and hardware vulnerability found in the design can be applied to all other products with little effort.

Encrypted memory prevents reverse engineering of program data, as readout protections of ROMs in the microcontroller can be bypassed [OT17]. Encrypted RAM prevents live forensics of the system.

Software countermeasures are used against a large variety of attacks. They involve double computation and verifying results as an effective detection against fault attacks. Also, the time redundancy of a software function can be improved to reduce leaked information about the program flow-side channels, decreasing the ability to use timing attacks.

As already mentioned, the security of an embedded systems relies on hardware and software design. Various techniques during software development can be applied to decrease the risk of side-channel and fault attacks.

Software countermeasures for RSA

Algorithm 1 is vulnerable for simple power analysis. For $d_i = 1$, the multiplication in line 5 is executed. For $d_i = 0$, the multiplication is not executed and results in shorter computation time. As a consequence, the exponent d is determinable in the power consumption profile on Figure 2.10 using a timing attack.

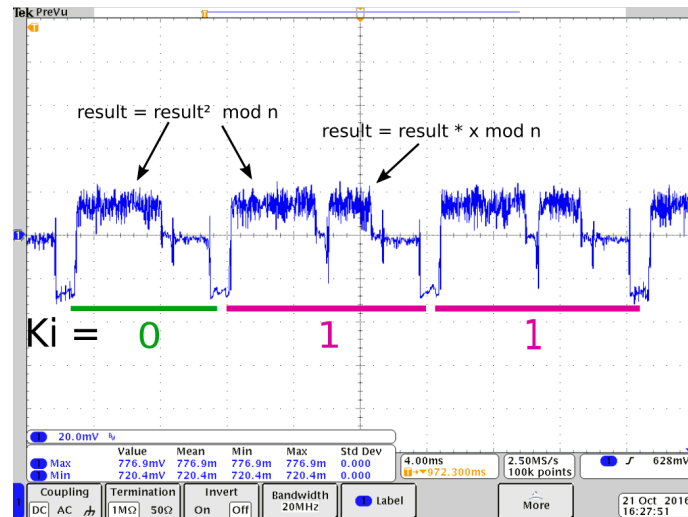


Figure 2.10.: Simple power analysis on square and multiply algorithm

In [Tob11], an enhanced RSA decryption method, with software countermeasures against timing and fault attacks, is described. This vulnerability is resolved in Figure 2.11 at block (44) by adding an identical computation for the case $d_i = 0$. As a result, the program takes the same amount of time for both states. This additional computation is later used for verifying the result against faults.

M_n at (44) executes the exponentiation with the complement of d . So (44) can be written as:

$M_n = C^{d^{-1}} \bmod N$, where the complement d^{-1} can be replaced by:

$d^{-1} = 2^L - 1 - d$, where L is the number of bits of d .

The verification at (45) calculates the product:

$$y = C * M * M_n \bmod N$$

$$y = C * C^d * C^{2^L - 1 - d}$$

$$y = C^{2^L}$$

The result is compared with the auxiliary variable x , which holds the value $x = C^{2^L}$ after L cycles. Any fault during the computation will mean that x is not equal to the product y .

The algorithm can be improved more by initializing M_n with the value of C . This way the multiplication with C at (45) is not needed.

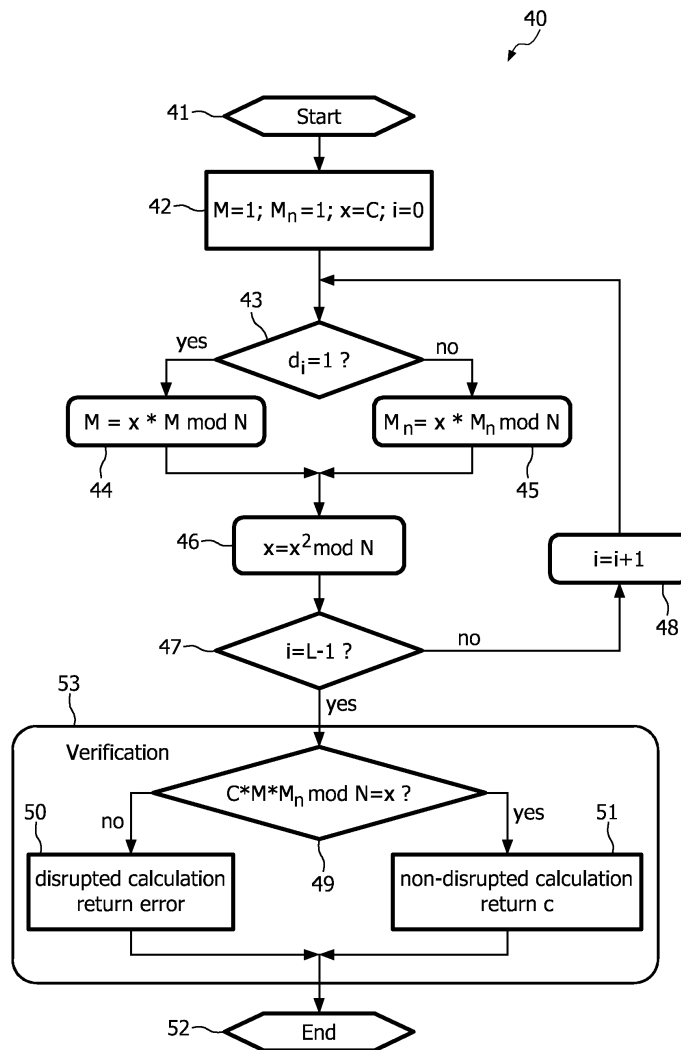


Figure 2.11.: Block diagram of a RSA decryption method with verification [Tob11]

3. Practical Attack

This chapter describes details of attacks. Although the principles of optical fault injections were discussed in chapter 2, optical fault injections are far too complex to guess or simulate their behavior during an attack. Therefore, the target architecture and the silicon die have been analyzed and the fault behavior investigated in different experiments. Similar attacks have been successfully executed by [SA⁺02] on a PIC16F84. The authors used a photo-flash mounted on a microscope, and, with a magnification of 1500x, they were able to change single bits in the SRAM. [GGS17] depicted an attack on an ARM Cortex-M0 controller using a low-cost setup. A photo-flash was mounted on a 3D printed frame with a lens and a motorized XY-stage to implement local attacks from the chip's backside. [Nac16] demonstrated a successful BellCoRe attack on the same processor family.

3.1. Device Architecture

The STM32F103RBT6 microcontroller is part of the STM32F1 processor family, which consists of a ARM Cortex-M3 32-bit RISC core. This IP-Core is part of the Cortex-M family by Advanced RISC Machines Ltd. The STM32F103 is realized as a System on a Chip (SOC). It also contains 128 Kbytes of Flash memory and 20 Kbytes of SRAM.

The ARM Cortex-M3 is based on a modified Harvard Architecture with separate storage and buses for instructions and data. This structure allows simultaneous access to both storages. However, instructions and data storage are accessible over the same address space. The Cortex-M3 uses a subset of the Thumb-2 instruction set. Thumb-2 extends the 16-bit Thumb-1 instruction set with additional 32-bit instructions. The 16- and 32-bit instructions can be used together in the code and ensure higher code density and performance.

This processor has 16 CPU Registers, R0-R15. R0-R12 are general purpose registers, which can be used for data storage and are much faster than all other memory. The stack pointer (SP) R13 is used as a pointer to the active stack. The link register (R14) is used to store the return value for the program counter when a subroutine is called. R15 is the actual program counter (PC). It stores the addresses of currently executed instructions.

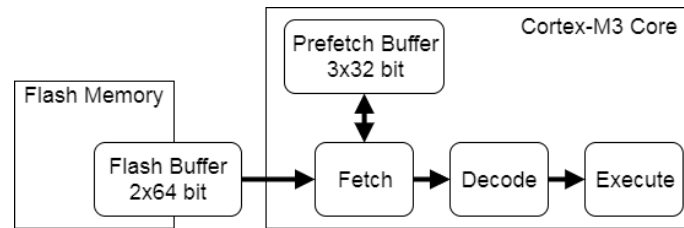


Figure 3.1.: Pipeline of the ARM Cortex-M3

The in Figure 3.1 illustrated ARM-v7M architecture implements a three-stage processor pipeline. All three stages-fetch, decode, and execute-operate simultaneously during a clock cycle. For example, during execution, a second set of instructions can be decoded, and another can be fetched. During the fetch state, the program data is returned from the instruction memory. Up to two instructions can be fetched during a cycle because most of them are 16 bits wide. The decode state reads the immediate values out of the instruction code and forwards them to the registers. The instruction then is executed during the execution state. When an instruction takes several clock cycles to be executed, the pipeline stalls. The instruction prefetch unit also has an instruction buffer. The buffer has a capacity of 3x32 bits. Thus, it can store up to 6x16-bit or 3x32-bit instructions. Instructions can be fetched several cycles ahead before they enter the decode stage.

The internal flash memory holds the program code. It has a bandwidth of 64 bit per block. Program data for the prefetch unit of the core is buffered in a additional 2x64 bit flash buffer. The buffer reads in 64 bit with a single read. This enables faster CPU execution, as the CPU fetches one word at a time with the next word already available in the prefetch buffer.

The STM32F1 already has some features to harden it against environmental influences. All modules of the chip are fed by an embedded voltage controller. Therefore, voltage rails for the CPU core and memory cannot be disrupted directly. A programmable voltage detector constantly compares the input voltage with a configurable value. An interrupt is triggered when the supply is over or below the threshold. An internal temperature sensor can be used to monitor the temperature of the device. A clock security system can be activated to detect errors in the external oscillator and generate an interrupt. However, these systems are designed to protect against faults caused by environmental conditions. They might be insufficient to protect against precise fault attacks.

In addition, the processor has several fault exceptions. These exceptions cover such usage faults as attempted execution of undefined instructions, division by zero, and invalid loading of the PC. They also cover memory faults, which occur when the CPU attempts to read or write at invalid addresses. The microcontroller also has a debugger. Among other things, it supports processor halt, single-step, processor core register access, hardware and software breakpoints, and full system memory access.

3.2. Decapsulation

To gain access to the die of an IC, its enclosure needs to be opened. This process is called decapsulation. There are many different approaches depending whether the front or back of the chip should be opened. A further difference is whether the chip should remain intact or if only the die itself is needed for analysis. Many labs use red-fuming nitric acid (HNO_3) as standard procedure for decapsulation. It etches away the epoxy but leaves the die and bondwires, which are made out of gold, intact. For newer ICs, copper bondwires are used more frequently as they are cheaper than gold. In this case, [Age14] recommends using a 9:1 or 2:1 mixture of HNO_3 and sulfuric acid (H_2SO_4). For thicker packages like PDIP, it is recommended to mill a small cavity as preparation. Chemical etching will then be faster and more effective.

For the target chip, this optional step is not needed. The STM32F103RBT6 comes in a QFP package that is only 1.6mm thick. The materials declaration form [STM17] indicates that gold bondwires are used; hence HNO_3 can be used. First, the IC was scanned with an X-ray to determine the actual size and position of the die. The die can be detected in the center of Figure 3.2, lying on the leadframe. Also, the bondwires, connecting the die with the pins, are recognizable.

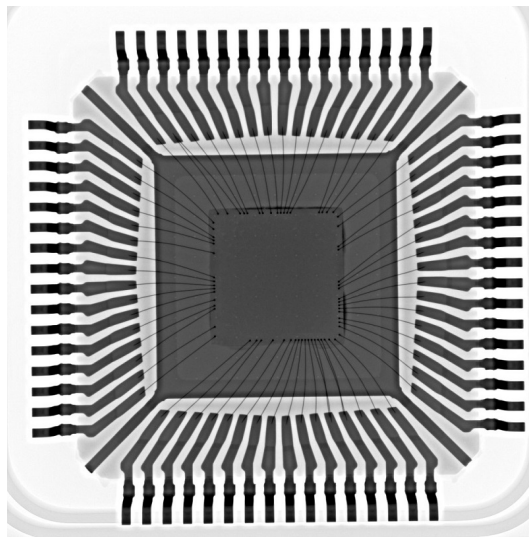


Figure 3.2.: X-ray image of an STM32F103RBT6 with QFP package

An automated chemical decapsulation system is used for the etching process. This process allows faster, safer, and more reproducible results. A suitable Teflon mask, depending on the die size, is chosen, and the IC is placed into the machine. Etch time and temperature are

selected and the automatic etch process is started. Figure 3.3 displays the machine under a fume hood.

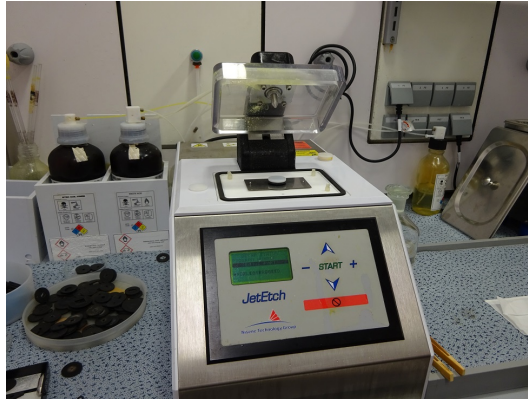


Figure 3.3.: JetEtch Pro Decapsulation System by Nisene Technology

After etching, the remaining acid in the hole is washed away with acetone. An additional step is to clean residues of the die surface using a semiconductor cleaning agent with assistance from an ultrasonic cleaner. Afterward, the whole package is cleaned again with isopropyl alcohol. The last step is to blow-dry the specimen to avoid corrosion. Figure 3.4 displays the final result with the IC fully exposed and the bondwires and pins intact.

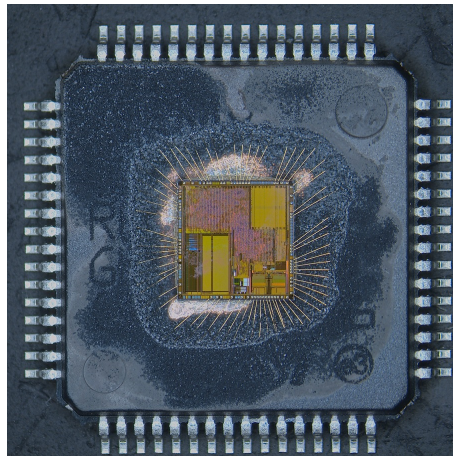


Figure 3.4.: Decapsulated Microcontroller

3.3. Target Analysis

After the die is exposed, the basic components of the microcontroller can be located. This view can be helpful in gathering more information about the chip, especially for optical fault injection attacks to find potential targets. Figure 3.5 depicts a composite image of the die made with a microscope. It is fairly easy to locate the memories because of their consistent pattern. The bottom left identifies as the 128 kByte flash memory. On the top right of the die is the SRAM with 20 kByte capacity. At the bottom right of the die, analog functions are identifiable by their unique structure. Capacitors and resistors for the internal power supply fill a large amount of space. These blocks are usually designed and tested once and used on all ICs of the same family. The remaining surface of the die is populated by glue logic, including the ARM core and all digital parts of peripherals such as timers, clock management, and bus drivers. The digital circuits are described by a hardware description language (HDL) and synthesized automatically. The synthesizer places standard cells such as AND and OR and then automatically routes connections between them. Because of this process, these circuits are not distinguishable without comprehensive reverse engineering.

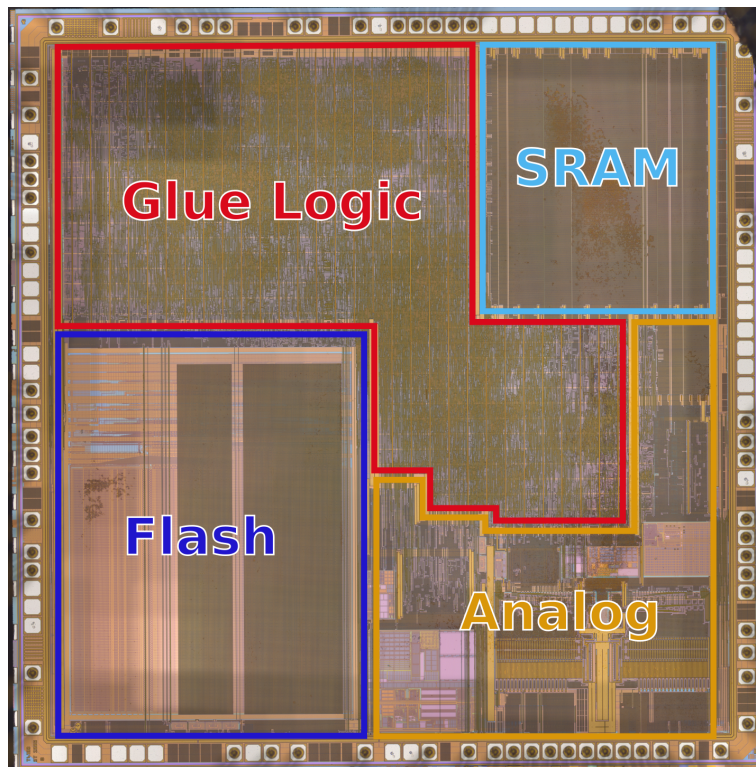


Figure 3.5.: Die of STM32F103RBT6 with annotations

As mentioned in 2.2.4 silicon is transparent for infrared light. Due to this factor, images

from the backside of the chip can be made with an infrared camera. Since there are no metal layers on the backside, more components can be revealed. To gain access to the backside, the chip was completely decapsulated and removed from the remaining lead frame. Afterward, the die was placed with the backside up under a microscope. The microscope was equipped with an infrared light source and a high-density near-field infrared camera. The camera picked up the light reflected by the die. Figure 3.6 depicts a complete backside image of the device under test (DUT). The flash controller, which had been hidden by the metal layers on Figure 3.5, is now clearly visible.

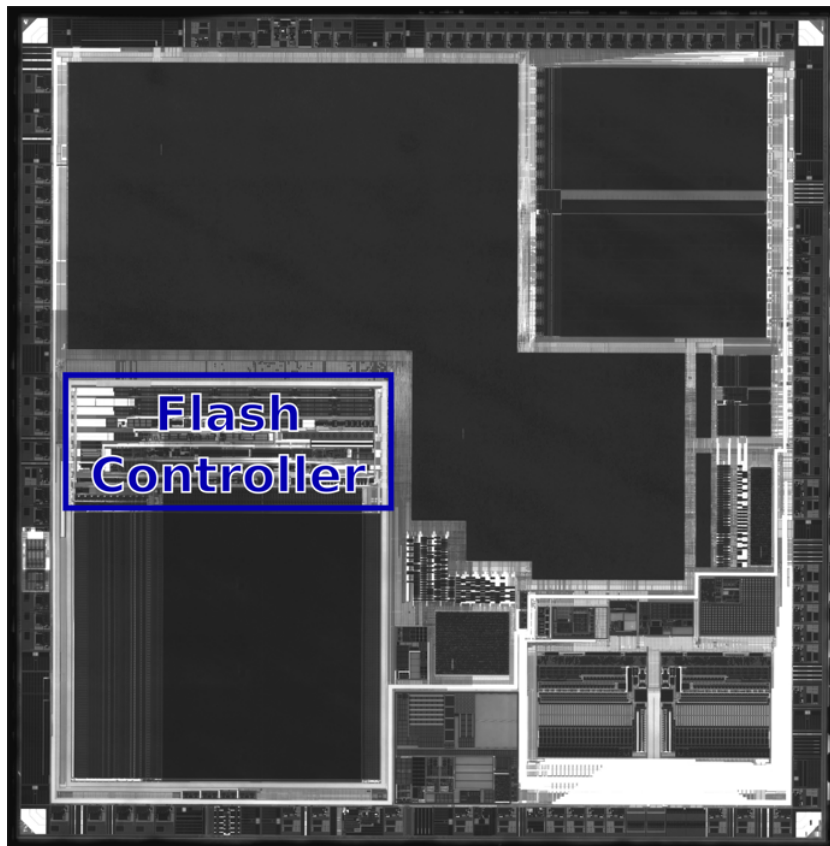


Figure 3.6.: Backside of STM32F103RBT6 with annotations

3.4. Setup and Experiments

The experiments are executed with a xenon strobe light. The only adjustable setting is the flash frequency. To ensure constant conditions, the strobe light is mounted on an aluminium profile. A measuring scale is attached to the profile to measure the distance between the strobe light and the IC. This setup allows the intensity to be adjusted by varying the distance.

The microcontroller on a NUCLEO-F103RB development board was desoldered and replaced with a decapsulated one. To communicate with the board, the microcontroller is connected to the PC via USB. This way, a UART and the debugger interface are accessible. The onboard ST-Link debugger on all development boards was upgraded with Seggers J-Link firmware [Gmb18a]. This debugger runs faster and has more capabilities. All programs are developed using Seggers Embedded Studio [Gmb18b]. The GCC Compiler Version 6.3.1 is used.



Figure 3.7.: Setup for experiments with xenon strobe light

3.4.1. Experiment 1: Finding a Vulnerable Spot

To gain a basic understanding of whether the controller is even sensitive to frontside optical fault attacks, a simple test program was written in C. Algorithm 3 starts by sending a reset notifier using the UART. Afterwards, it will increment a loop counter and an additional counter until they both reach a value of 1 million. The results of both counters are then sent to the PC using UART with a baud rate of 115,200 baud. The external crystal is not populated on the development board. Therefore, the internal RC-Oscillator with a frequency of 8 MHz is used for the system clock. No additional libraries are used in order to keep the code as short as possible. The more the processor stays inside the actual counting loop, the greater the probability that a fault will hit this loop. In this way, faults can be narrowed to a few instructions of the program code. Presumably, a fault would affect an incorrect result of the counter value.

All unused flash memory was written with a branch to the address of itself. In case the PC is set to a random value in flash, the processor will remain in this loop, making it easy to trace this type of fault with the debugger. A handler for the hard fault interrupt sends out the Configurable Fault Status Register (CFSR) and Hard Fault Status Register (HFSR), providing some insights about the failures. Afterward, the handler stays in an endless loop. Thus, the microcontroller must be reset manually. These hard faults are considered as failed attacks since they are recognized by the chip.

On a proper device, line 9 would never be executed because of the while-loop. This message will indicate that the program broke out of the loop. Because of the compiler optimization, line 1 and two 2 cannot be replaced by while (1), as the compiler would otherwise ignore the unreachable code in line 9. This factor is still the case when using the lowest optimization level in ARM GCC. The total duration for a complete counting loop and output of the data takes around 2 seconds.

Algorithm 3: Testprogram

```
1 UartSend("Reset") a = 1
2 while a == 1 do
3     cnt = 0
4     for i = 0..1000000 do
5         cnt++
6     end
7     UartSend(i,cnt)
8 end
9 UartSend("Outbreak")
```

At this point, all attempted attacks resulted in a reset of the controller. This reset was still the case with maximum distance between the strobe light and the IC, even with use of neutral density filters on the IC that reduced the light intensity of the flash up to 99.9%. A problem with global optical attacks is that analog circuits are fairly sensitive to light. Figure 3.5 shows that parts of the analog circuit are not covered by metal layers, which makes them even more sensitive to these attacks. Referring to the pinout the PLLs for the clock signals, the reset handler and likely the internal voltage regulator might be located there. Optical fault attacks with lasers would not cause such problems, as their spots are smaller. They only affect local areas of the chip.

To prevent the controller from resetting during the attacks, different methods of masking the die were tested. Paint has previously been used to protect parts of a memory against ultraviolet (UV) radiation. It can be applied using a microscope and a single hair of a brush. A very steady hand is required to ensure that no bondwires are damaged. Some paints were transparent for non-determinable wavelength, which disrupted the IC. Furthermore, photons that impact next to the die can reflect and strike the silicon from the side, as depicted in Figure 3.8.

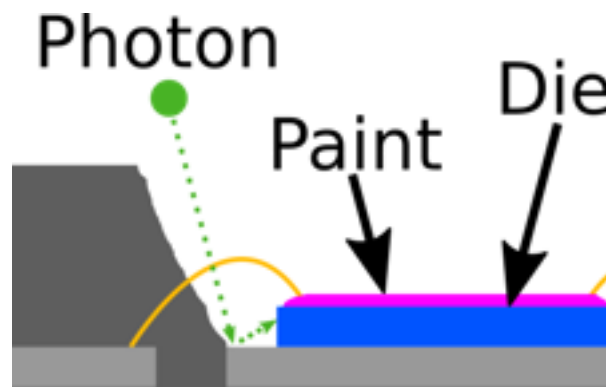


Figure 3.8.: Reflect photons on the leadframe

A more hassle-free method is to use electric tape. It is completely lightproof and can be stuck on the remaining package surface. Therefore, the chance of harming the bondwires is reduced. Masks can be cut with a scalpel. Afterward, they can be placed very precisely under a microscope. Figure 3.9a depicts an applied mask where the analog circuits are covered. Different areas of the chip were covered during this test. The flash controller was the vulnerable part of the chip. This finding could be confirmed with the help of the backside image in Figure 3.6. To reduce the number of unusable faults the mask in Figure 3.9b, which only exposes the area of the flash controller, was made. Tests where only the SRAM or the Core logic were exposed proved unsuccessful.

Different error patterns occurred during the successful fault injections against Algorithm 3. In many cases, cnt and i had the same, too-low value. This result indicates that the counting

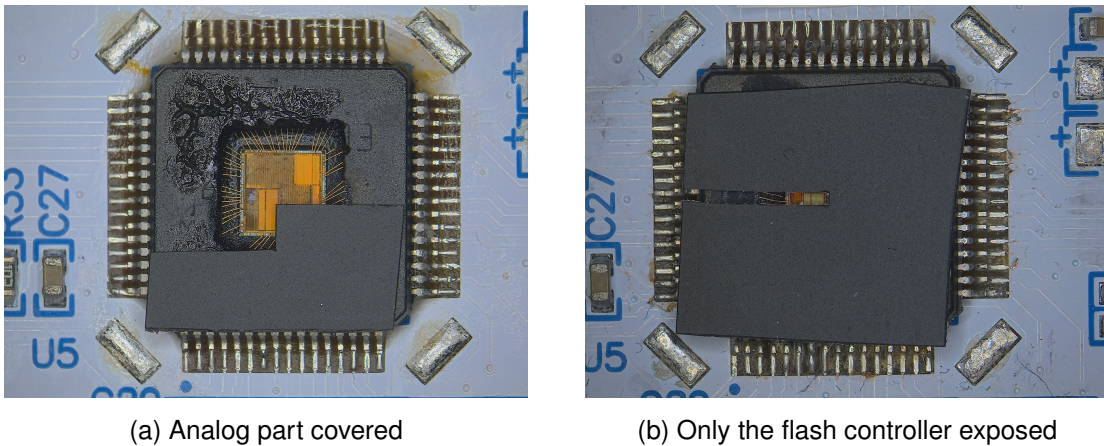


Figure 3.9.: Covered chips using tape

loop was left too early. Other cases had an unaffected i and a much too high or too low cnt . In this case, the incrementation instruction of cnt may have been disturbed. As explained in Chapter 2.2.5, only a single bit of a conditional branch has to be flipped to change its behavior. A breakout of the infinite loop occurred only once. The relevant code for the while loop is only 3 assembly instructions long, making it unlikely to be struck.

The contents of the fault registers during the hard fault exceptions were also evaluated. Most of the time, the exceptions were thrown because the memory tried to access invalid addresses. Another common exception was that the processor was attempting to execute illegal instructions. An illegal instruction is an opcode that is not defined in the instruction set. Overall, the produced faults were well-reproducible.

3.4.2. Experiment 2: Fault Behavior of the Flash Controller

This experiment is meant to examine the effects that the fault injection has on the flash controller. The faults in Chapter 3.4.1 were injected during the program execution. No writing or erasing of the flash was performed during the test program. Also, the faults were only temporary, with no data in memory changing permanently. This finding suggests that only the reading of the flash was affected by the attack. The illegal instruction exceptions suggest that the chip received corrupted instructions from the flash memory.

To test this theory, the flash memory was completely written with a consistent test pattern. It was then read out using the debugger. The flash controller was attacked during the reading. Afterward, the original flash contents were compared to the dumps. The attack was automated using the python program `flash-tester.py`. A python library [Inc18] was used as interface to the SEGGER J-Link debugger. The flash was dumped in and compared to the reference image in a constant loop. Corrupted memory contents were released.

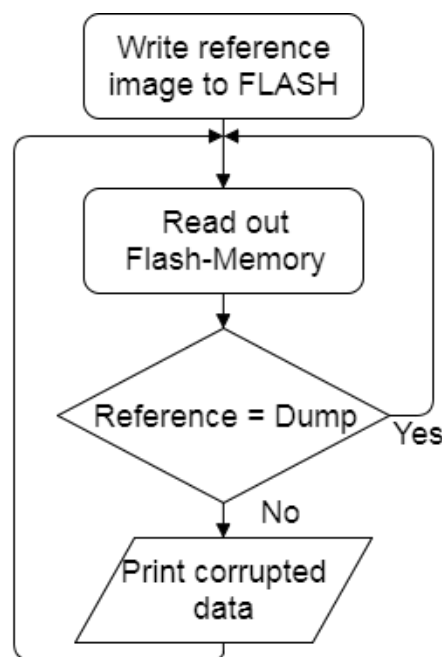


Figure 3.10.: Flowgraph for `flash-tester.py`

As expected, the attacks resulted in corrupted data in the dumps. Depending on the frequency of the flash, one or two small areas in the flash were affected. The locations were at different addresses during the test. The corrupted areas ranged between 16 bits and several words, depending on the distance between DUT and flash. Different test patterns demonstrated that the corruption resulted in setting bits to zero. The case of a bit modification from 0 to 1 never occurred. In most cases, the data was completely set to 0. This result means

that the CPU will load 0x0000 as its instructions, which will be decoded as "mov r0, r0" and is equal to an no operation (NOP) instruction. As the data corruption never occurred at the same address, and only the flash controller was exposed to the light, it is unlikely that the flash cells themselves were flipped.

3.4.3. Experiment 3: Influence of the Light Intensity

During the experiments, we observed that the distance between the DUT and the flash had a significant influence on the error rate. It seemed that a higher rate of hard fault exceptions occur when the light intensity increases. The portable demonstrator will not have a stand, but it is still important to gain the highest fault rate possible. Therefore, we need to further examine this observation.

As already mentioned, the flash intensity was not adjustable. Therefore, it was varied by changing the distance between the flash and the DUT. The strobe light was constantly run with a frequency of 1-2 Hz. For every distance, 50 loops of the program were attacked and logged. The python program "flash_test.py" was written for automated evaluation. Using the pySerial library, the incoming UART data was fetched in and the results were parsed. The total amount of runs, unaffected runs, injected faults, hard faults, and values of the counter were logged into a .csv file.

The experiment proved that the intensity of the strobe light has a significant influence on the fault behavior. Figure 3.11 illustrates the relationship between the distances of the flash and the DUT and their effects on the chip.

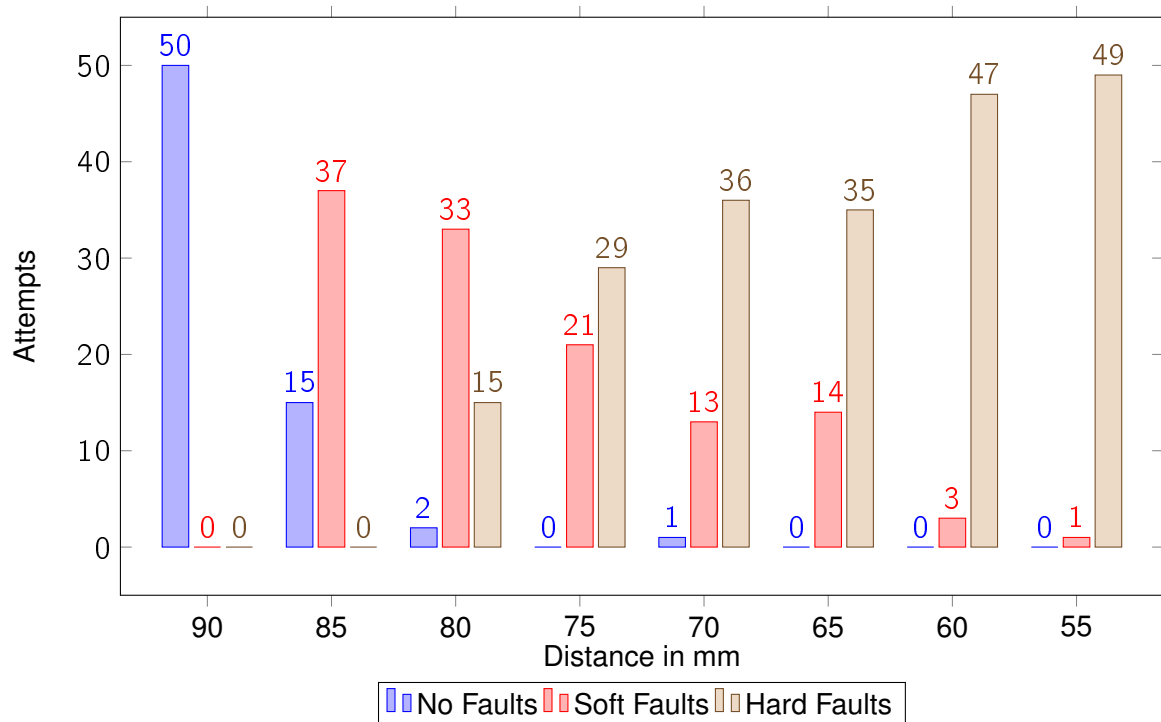


Figure 3.11.: Relationship between distances of the strobe-light and the occurred faults

Attacks with a distance of 90 mm or more were unsuccessful. Distances lower than 90 mm had a rapidly increasing number of incorrect counter values. By further decreasing the distance, more hard faults occurred. Below 55 mm, the chip was not able to operate properly anymore.

3.5. Conclusion

The experiments demonstrate that even on semiconductors with several metal layers, frontside attacks are still possible. Vulnerable spots can be found and attacked with the cheapest equipment. Advanced equipment was used for the backside images, although locating the flash controller is also possible from the front side. After a vulnerable spot was found, simple test programs were created to investigate the fault behavior. These insights are useful to plan attacks on security functions.

It was proven that the flash controller is disrupted, leading to wrong results during the readout of the program instructions. Most of the affected opcodes were turned to zero, and therefore skipped by the controller. This result offers the opportunity to manipulate conditional checks, cryptographic algorithms, and simple software countermeasures. It would be even more effective in combination with external triggers, which induct the fault at the exact moment of a conditional check.

Following assumptions can be made for the design of the demonstrator:

- Frontside attacks are possible. Therefore, only simple modifications on the board are required.
- For proper results, the distance between the photo-flash and the board must be kept at around 8-9 cm.
- Long algorithms are more suitable to attack in a demo than in conditional checks.

4. Demonstrator

This chapter is about the actual design of the demonstrator for optical fault injection attacks. As part of the "Kryptographie in Software und Hardware" lecture at HAW Hamburg, experience has already been gained in attacking a RSA-CRT implementation with power glitching on a similar device. Ultimately, it has not been possible to perform successful attacks. The previous experiments have demonstrated that long algorithms are a promising target for local optical fault attacks. With its long computation times, the RSA-CRT offers a much-larger surface for triggered attacks than conditional checks in a password interrogation. The BellCoRe attack is well-known and efficient. Therefore, a use case for a BellCoRe attack on a RSA-CRT implementation has been chosen.

The audience might not have the required knowledge to understand the exact processes of fault injection and the BellCoRe attack. For these people, the demo has been greatly simplified, so only the effects and impacts can be demonstrated to them.

4.1. Use Case

The use case for this demo is based on secure authentication schemes. These solutions are implemented in medical test equipment, printer cartridges, battery packages, and accessories like charging cables. They are meant to protect manufacturers and consumers from fraudulent products. Counterfeit batteries for mobile devices might not have the required safety and protection circuits. In 2017, smartphone batteries distributed through AT&T's insurance program were recalled. Some of the batteries were counterfeit and showed abnormal behavior, which could lead to overheating [Com17]. In 2004, LG recalled batteries for mobile phones because they lacked the necessary over-voltage protection circuits [Com04].

Many authentication ICs and battery-management ICs with authentication mechanisms integrated are on the market. For the authentication schemes, symmetric or asymmetric cryptography is used.

On symmetric systems, the host and the client possess the same secret key. The host sends a challenge to the client, and both encrypt the challenge with the secret key. The client sends

back the encrypted challenge in response. The host compares both ciphers. If they match, the host can be assured that the client is in possession of the right secret key, and is therefore authentic.

Asymmetric schemes are based on asymmetric cryptography. The host has the public key and the client the private key. The client will sign the received challenge using the private key and send it back as the response. The host will verify the response using the public key. The client will authenticate if the result matches with the plain challenge. Usually these schemes are combined with a hashing algorithm. They produce a condensed representation called a message digest for a message. Most systems use a random challenge generator to make replay attacks harder. The advantage of an asymmetric scheme is that secure key storage is only needed for the client side. However, symmetric schemes are less expensive and easier to implement.

4.2. Implementation

This demo represents the simplified case of a battery authentic system. A connected battery needs to be authenticated to be accepted by the host. The authentication IC of the battery is represented by the microcontroller on the evaluation board. If the board is connected to the computer, the host application will send a challenge to the device. The IC will sign the challenge using the private key, stored in the flash memory. Afterward, it sends back the result to the host. The host will apply the RSA-CRT with the public key to the received response. Then, the result is compared with the previously sent challenge. A matching result means that the device is authentic, and the host application will accept the battery. Otherwise, a warning is presented to the user that the user has connected an unauthorized battery. To simplify the implementation, no random challenge generator and hashing algorithm will be used. Instead, the host application will always send the same challenge. This setup also simplifies the BellCoRe attack because one correct and one false response for the same challenge is needed. The applied scheme is depicted in Figure 4.1.

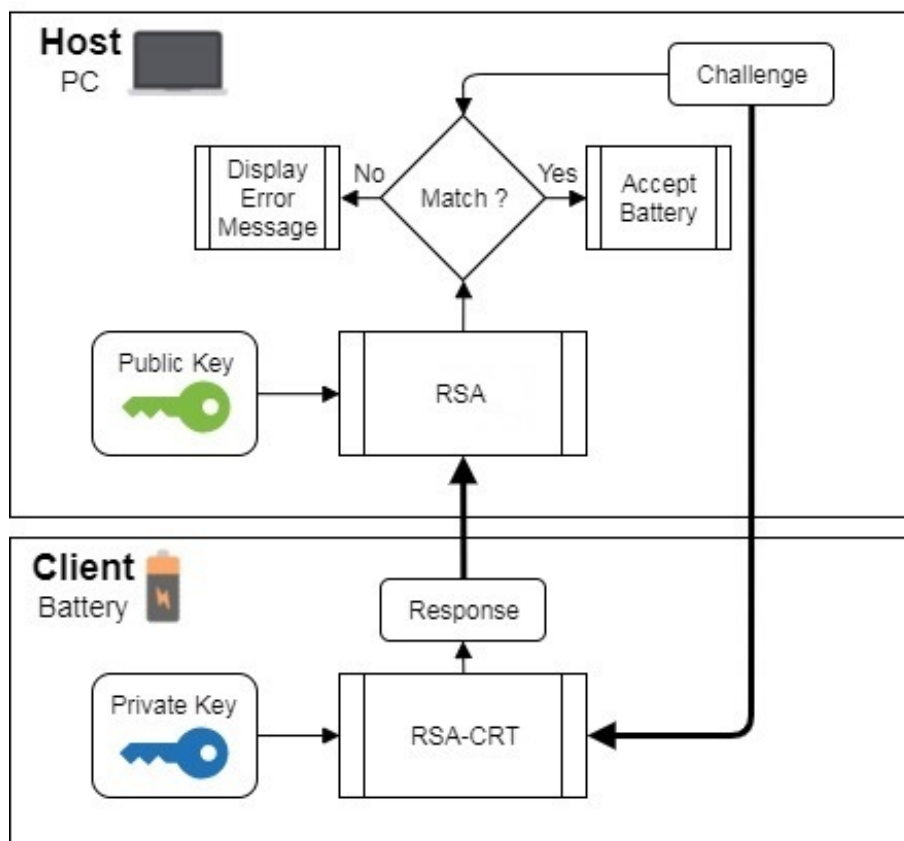


Figure 4.1.: Applied authentication scheme using RSA-CRT

A second board represents a counterfeit battery. This board is not in possession of the secret key; therefore, it cannot answer with valid responses to the challenges. Thus, the host application will not accept the connected device and will provide a warning.

The faults are injected with a detachable flash for cameras, which has the advantage of being more portable as it is supplied by batteries. The microcontroller on the board, which represents the original battery, is decapsulated. Faults can be injected to compromise the RSA-CRT implementation. The obtained private key can then be sent to a counterfeited device to make it act as a verified unit. The board is connected to the computer using USB, establishing a UART connection.

4.2.1. Microcontroller

As already mentioned, this demo is based on the BellCoRe Attack on an RSA decryption. The RSA-CRT was implemented according to Chapter 2.3.2. RSA requires calculations with several hundred-digits- long numbers. The Cortex-M3 lacks any process to do these calculations in hardware. Therefore, the FLINT¹ long-number arithmetic library was used. The FLINT library stores the values as CLINT objects, which are variable-length arrays of digits. The size of the values is only limited by the RAM. The computations in this library are performed in an inner loop for each element of the object. These loops are extremely vulnerable to the previously discovered effects of fault injections. If instructions are skipped during the computation loop, the program will proceed with the wrong values without noticing it.

Due to the poor performance of the microcontroller, the RSA calculation for a 256-bit key requires an average of 500 ms. Therefore, only 256-bit keys are used. On the other hand, this long computation time is essential to be able to hit the algorithm with the injected faults.

The flowchart in Figure 4.2 illustrates the implementation of the client application in the microcontroller. The program starts with initialization routines. The USART2 interface is used for communication with the host over the USB to UART bridge. The RSA key values are saved as strings in the program memory. During the initialization, they are read from flash converted into a CLINT object using the `str2clint_l` function.

During the main loop, the program will first check if the new challenge flag is set. If so, it will overwrite the challenge string with the value in the buffer². This step is necessary to ensure that the challenge is not overwritten by the UART interrupt during the response calculation or data output. Subsequently, the corresponding response to the challenge will be calculated using the RSA-CRT function.

The data will be sent to the host using the UART interface. A simple frame layout, depicted in Figure 4.3, is used to parse with the host application. The frame is sent as a string with a header section (orange) at the beginning, followed by up to two data elements (blue) of variable length. The single elements are divided by space characters. Each frame ends with a new line character (red).

The UART receive interrupt is used to read in new challenges from the host. When the UART receives data, the interrupt is generated. The handler will check if a new char is available in the FIFO, and it is then read into the buffer. The newline character "\n" represents the end of a string. The program will then set the new challenge flag to true. A new private key can

¹This library is part of [Wel01].

²Although this function will not be used for the demonstrator

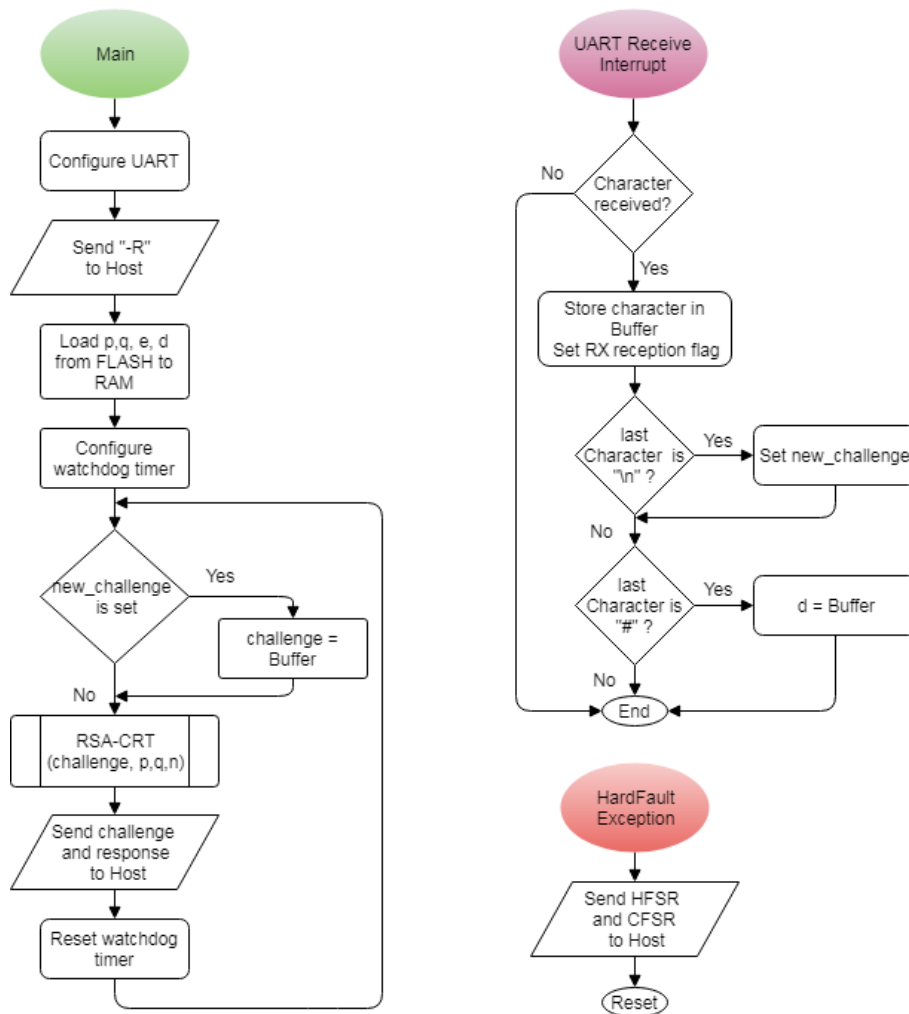


Figure 4.2.: Flowchart of the microcontroller application

be configured by sending a hashtag character at the end of the string. This key will only be stored in volatile memory to make the demo easy to reset.

The hard fault handler is programmed to reset the chip after sending out the error message. In this way, the host application can count these failures, and the chip does not get stuck during the presentation. A watchdog timer is implemented to improve reliability. This timer resets after a programmed amount of time if the counter is not reset. This step prevents the processor from getting stuck without causing a fault exception. These additions to the code are essential because usually only under 17% of the attacks on this RSA-CRT implementation are successful, with 20-40% of the attempts resulting in hard fault exceptions or other program crashes. If manual resetting would be required, the demo would become impractical.



Figure 4.3.: UART frame format for Client-to-Host messages

Implementing Countermeasures

As an additional feature, the hardened multiply and square algorithm described in Chapter 2.4 is implemented. If a fault is detected, the "Fault Detected" notifier is sent to the host and the device will reset. The countermeasures can be activated with a jumper on pin PC7 to ground.

Extracting the private key was still possible with the countermeasures implemented. To obtain a comparison, 1,000 runs with each version were monitored. On the straightforward implementation, an exploitation fault was accomplished on 17.1% of the rounds, compared to the hardened version, where only 0.4% exploitable faults occurred and 194 faults have been detected. On an actual application, the device could refuse operation after a certain amount of errors have occurred.

Most likely the verification of the results are compromised by a glitch. At position 49 on Figure 2.11, the program decides whether to proceed or to abort depending on the result of the verification. If this conditional check is somehow disrupted, the RSA-CRT algorithm will continue with wrong results of the exponentiation, leading to a faulty and compensable output.

This result underlines that software measures alone are not sufficient to secure embedded devices. Only the joint implementation of software and hardware countermeasures can ensure a high level of security.

4.2.2. Host Application

The host application for the demo is written in python. Using the tkinter library, a graphical user interface (GUI) was created. To make the user interface clearer and less distracting, it is divided into three different pages. The mainloop depicted in Figure 4.4 handles the communication with the connected board in the background. It sends out the challenge, and the incoming UART messages are parsed depending on the header field. The received responses are verified, and if they are wrong, a BellCoRe attack is attempted. All results are displayed in the user interface. To reduce CPU load, the program sleeps 2 ms before each loop. For the RSA calculation and the BellCoRe attack, a small library named CryptTools.py was written. It contains a straightforward implementation for the RSA, BellCoRe attack, and support functions. The GUI is automatically updated during the idle times.

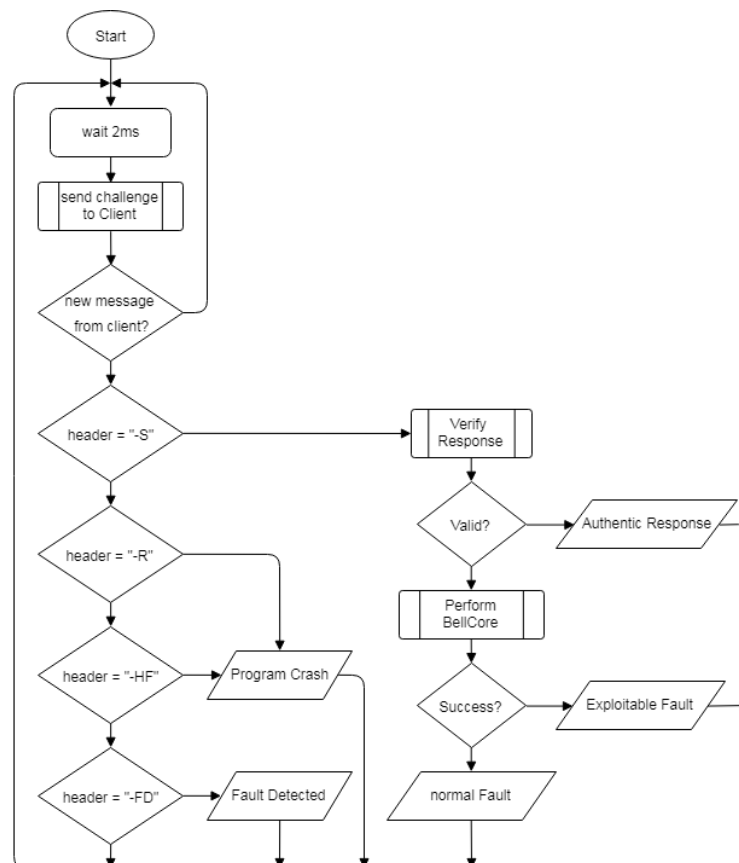


Figure 4.4.: Flow graph of the host applications main loop

Parameters like the RSA public key and the serial ports are saved inside a configuration file named config.ini. PyInstaller [Tea17] is used to generate a Windows executable and run the application on any Windows computer without python installed. To make the demo easier

to use during a presentation, all buttons are mapped to keyboard keys. The three different pages of the GUI are described in the following sections.

Battery Monitor

Page 1 contains a battery monitor. It serves as an introduction to the use case. The battery monitor indicates if a battery is connected, and if it is authentic or not.

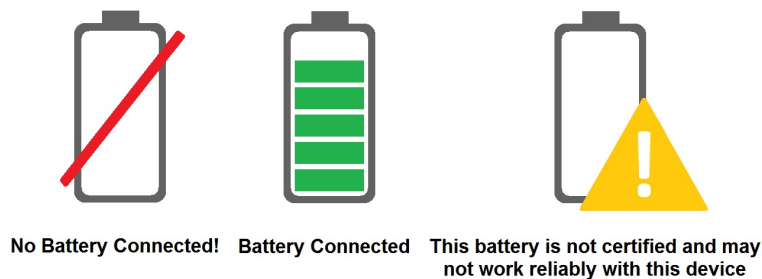


Figure 4.5.: Different status indicators of the battery monitor

Main Page

On page 2, the communication between host and client are charted. Challenges and responses are constantly updated in two lists. Correct challenge-and-response pairs are highlighted with a green background. Faulty signatures are highlighted in yellow. In case of a faulty signature, the program executes the BellCoRe attack and attempts to calculate the private key. An exploitable faulty signature is highlighted with a red background color.

As an additional feature, a simple statistics function is implemented. By activating the function, a configurable number of rounds will be logged. The number of rounds, correct responses, faults and exploitable faults, program crashes, and detected faults are listed in a table. There are modes for the normal straightforward and for the implementation with countermeasures, making it possible to compare both.

BellCoRe and Cloning

On page 3, all exploitable responses are listed. Any of these responses can be selected, and the BellCoRe attack will be applied to it. The private key will then be indicated. By pressing the clone button, the obtained private key will be sent to the counterfeit device.

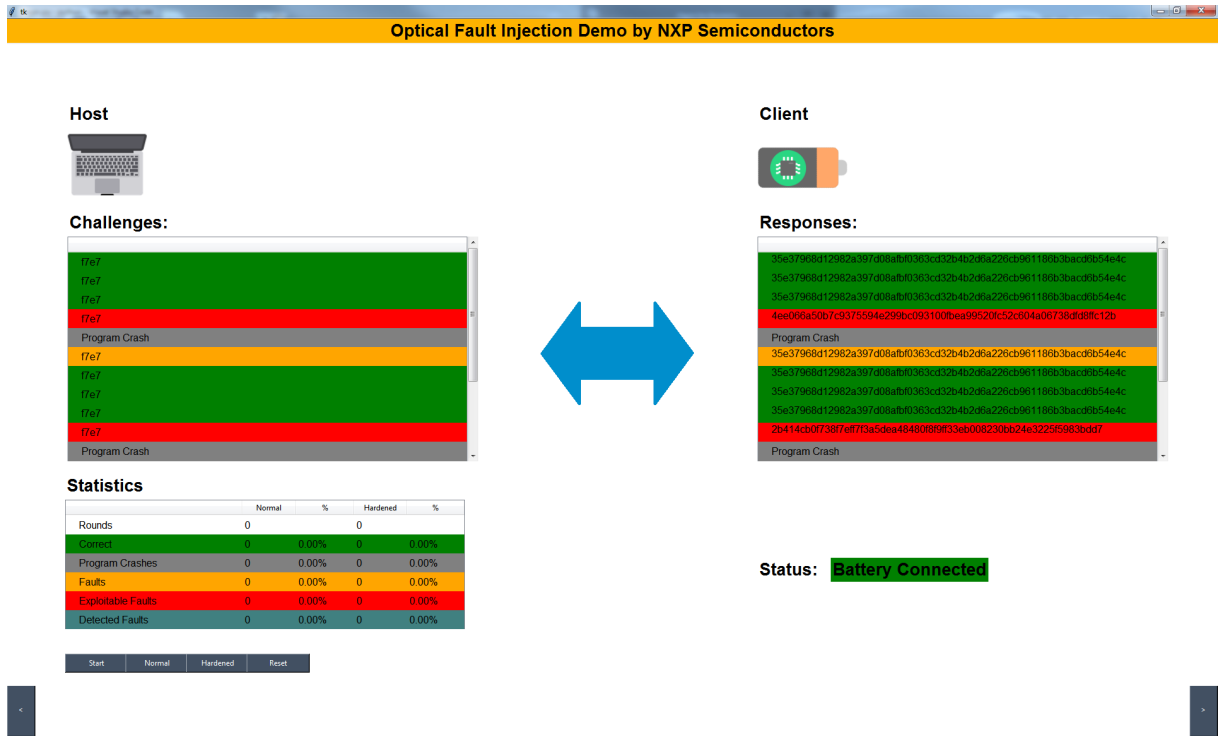


Figure 4.6.: Page 2 with communication display and statistic

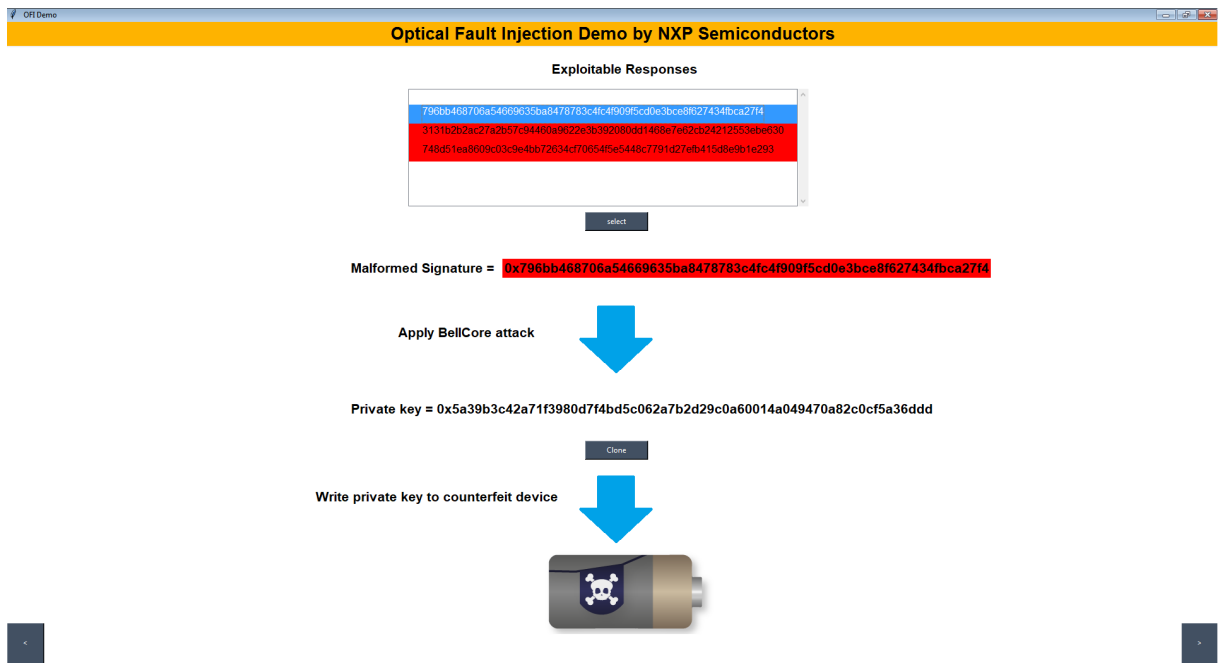
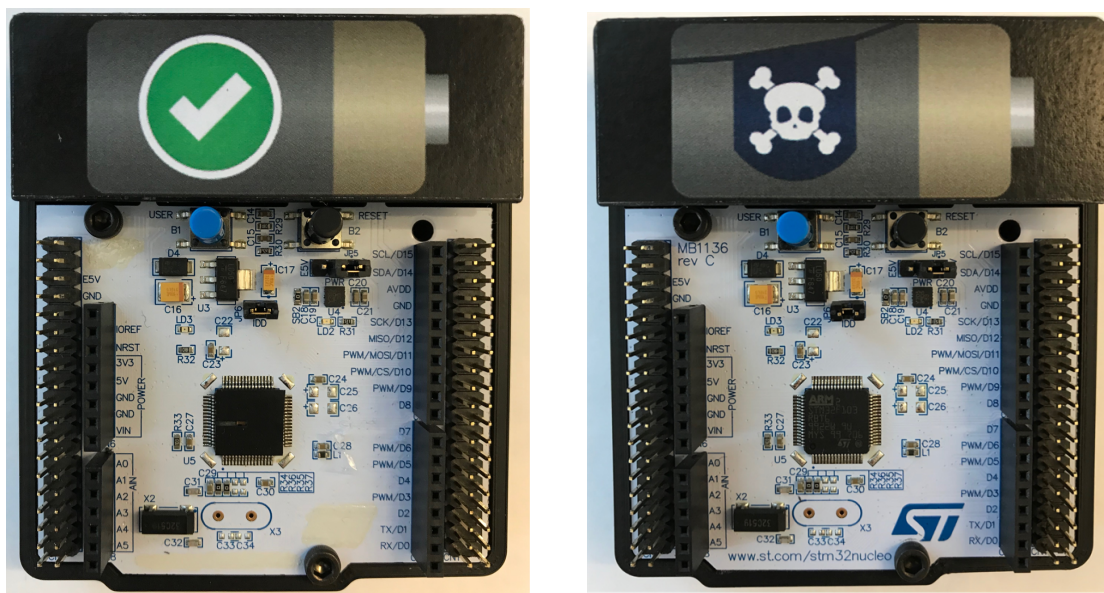


Figure 4.7.: Page 3 with key extraction and cloning

4.3. Outcomes

Overall, five units of the demo were created. Their reliability was proven several times in presentations during the design process. The demonstrator is packed in a hard case to make it portable. Instead of strobe lights, external photo-flashes are used. The benefits are battery supply and easier handling. Cases for the boards were made using a 3-D printer. To illustrate the two boards as original and counterfeit, a small panel was designed and printed. The panels have a small picture on them and can be plugged onto the boards. A short video of the demonstration is included in the appendix. Instructions, host application, and source code are available in each kit on a USB flash drive. Each demo kit contains the following items:

- A development board with decapsulated chip representing the authentic battery
- An unmodified development board representing the counterfeit battery
- A photo-flash
- A decapsulated chip for viewing purposes
- A USB mini-cable
- A USB stick with instructions, the host application, binary files, and source code
- A ruler with a marking to help align the photo-flash



(a) Original device

(b) Counterfeit device

Figure 4.8.: Both boards of the demonstrator in 3D printed cases



Figure 4.9.: Complete demonstrator kit in case

5. Conclusion

A demonstrator for optical fault injection attacks was built for this bachelor thesis. In building the demonstrator, a Cortex-M3 microcontroller was examined for vulnerabilities. After de-capsulating the chips, the surfaces of the die were analyzed for potential targets. By using electronic tape, the local fault attacks could be performed with cheap equipment. Simple test programs have been used to find the flash controller as the vulnerable spot. This finding proves once more that these types of attacks are feasible with the simplest equipment and little prior knowledge. Hardware manufacturers need to be aware that security-related applications on standard microcontrollers are absolutely not recommended. Another threat for a company is that their intellectual property can be stolen from these devices.

The knowledge derived during the experiments was used to design a portable, easy-to-reproduce, and cheap demonstrator. Effects of the attacks are perfectly suited for an BellCoRe attack on an RSA-CRT implementation, which is well known in the cryptography community. To be able to teach people without knowledge about this attack and the threats and impacts of fault attacks, a use case around the BellCoRe attack was arranged. In addition, a software countermeasure was implemented.

Further work can use the photo flash lamp to perform triggered attacks. Also interesting could be using a STM32F105RB controller. It is a pin-compatible controller of the same production family and with the same flash size. However, this controller has the embedded trace macrocell (ETM) in its core. This feature can be used in addition to an advanced tracing debugger. It could be possible to trace the modified instructions executed by processor after a successful attack on the flash buffer.

The vulnerable part of the flash controller might be the reading charge pumps. Charge pumps on flash memory are known as a weak point [NGSJ99]. Attempts have been made to locate the charge pumps to perform more local attacks on them. Therefore, a chip was encapsulated on the backside, and a hole was milled into a development board. In this way, the flash controller could be further investigated using photo emission microscopy. At this point, it was not possible to determine the charge pumps.

Bibliography

- [Age07] National Security Agency. Tempest: A signal problem. <https://www.nsa.gov/news-features/declassified-documents/cryptologic-spectrum/assets/files/tempest.pdf>, 2007. [Online; accessed 21-March-2018].
- [Age14] European Space Agency. Escc basic specification no. 25300 - decapsulation of plastic encapsulated semiconductor devices. <http://escies.org/escs-specs/published/25300.pdf>, 2014. [Online; accessed 21-March-2018].
- [Ana14] Nikolas Athanasios Anagnostopoulos. Optical fault injection attacks in smart card chips and an evaluation of countermeasures against them. University of Twente - EIT ICT Labs Master School, 2014.
- [BCD⁺12] S. Briais, J. M. Cioranescu, J. L. Danger, S. Guilley, D. Naccache, and T. Porteboeuf. Random active shield. In *2012 Workshop on Fault Diagnosis and Tolerance in Cryptography*, pages 103–113, Sept 2012.
- [BDL97] Dan Boneh, Richard A. DeMillo, and Richard J. Lipton. On the importance of checking cryptographic protocols for faults. In Walter Fumy, editor, *Advances in Cryptology — EUROCRYPT '97*, pages 37–51, Berlin, Heidelberg, 1997. Springer Berlin Heidelberg.
- [BECN⁺04] Hagai Bar-Ei, Hamid Choukri, David Naccache, Michael Tunstall, and Claire Whelan. The sorcerer's apprentice guide to fault attacks. *IACR Cryptology ePrint Archive*, 2004.
- [Com04] United States Consumer Product Safety Commission. Verizon wireless announce recall of counterfeit cell phone batteries. <https://www.cpsc.gov/Recalls/2004/cpsc-verizon-wireless-announce-recall-of-counterfeit-cell-phone-batteries>, 2004. [Online; accessed 21-March-2018].
- [Com17] United States Consumer Product Safety Commission. Fedex supply chain recalls cellphone batteries that could lead to fire and burn hazards (recall alert).

- <https://www.cpsc.gov/Recalls/2017/fedex-supply-chain-recalls-cellphone-batteries-recall-alert>, 2017. [Online; accessed 21-March-2018].
- [fSidl18] Bundesamt für Sicherheit in der Informationstechnik. Kryptographische verfahren: Empfehlungen und schlüssellängen. https://www.bsi.bund.de/SharedDocs/Downloads/DE/BSI/Publikationen/TechnischeRichtlinien/TR02102/BSI-TR-02102.pdf?__blob=publicationFile, 2018. [Online; accessed 21-March-2018].
- [GGS17] Oscar M. Guillen, Michael Gruber, and Fabrizio De Santis. Low-cost setup for localized semi-invasive optical fault injection attacks - how low can we go? In *Constructive Side-Channel Analysis and Secure Design - 8th International Workshop, COSADE 2017, Paris, France, April 13-14, 2017, Revised Selected Papers*, pages 207–222, 2017.
- [Gmb18a] SEGGER Microcontroller GmbH. Converting st-link on-board into a j-link. <https://www.segger.com/products/debug-probes/j-link/models/other-j-links/st-link-on-board/>, 2018. [Online; accessed 21-March-2018].
- [Gmb18b] SEGGER Microcontroller GmbH. The leading cross platform ide-embedded studio. <https://www.segger.com/products/development-tools/embedded-studio/>, 2018. [Online; accessed 21-March-2018].
- [Hab65] D. H. Habing. The use of lasers to simulate radiation-induced transients in semiconductor devices and circuits. *IEEE Transactions on Nuclear Science*, 12(5):91–100, Oct 1965.
- [HSH⁺08] J. Alex Halderman, Seth D. Schoen, Nadia Heninger, William Clarkson, William Paul, Joseph A. Cal, Ariel J. Feldman, and Edward W. Felten. Least we remember: Cold boot attacks on encryption keys. 2008.
- [Inc18] Square Inc. Python library for device debugging/programming via j-link. <https://github.com/square/pylink>, 2018. [Online; accessed 21-March-2018].
- [KBCK13] Felix Kimme, Peter Brick, Sangam Chatterjee, and Tran Quoc Khanh. Optimized flash light-emitting diode spectra for mobile phone cameras. In *Applied Optics Vol. 52*, pages 8779–8788. OSA-Publishing, 2013.
- [Koc96] Paul C. Kocher. Timing attacks on implementations of diffie-hellman, rsa, dss, and other systems. In *Proceedings of the 16th Annual International Cryptology*

- Conference on Advances in Cryptology, CRYPTO '96*, pages 104–113, London, UK, UK, 1996. Springer-Verlag.
- [Lim17] ARM Limited. Q3 2017 roadshow slides. https://www.arm.com/-/media/global/company/investors/PDFs/Arm_SB_Q3_2017_Roadshow_Slides_Final.pdf?revision=ccb28439-9cd5-42ed-a152-0f78d18c1370&la=en, 2017. [Online; accessed 21-March-2018].
- [Nac16] David Naccache. Side-channel and fault analysis in the presence of countermeasures: Tools, theory and practice. These de Doctorat de l'Université de recherche Paris Sciences et Lettres PSL Research University, 2016.
- [Neu16] Heike Neumann. Kryptographie skript 1.2. Kryptographie in Software und Hardware, 2016.
- [NGSJ99] D. N. Nguyen, S. M. Guertin, G. M. Swift, and A. H. Johnston. Radiation effects on advanced flash memories. *IEEE Transactions on Nuclear Science*, 46(6):1744–1750, Dec 1999.
- [OT17] Johannes Obermaier and Stefan Tatschner. Shedding too much light on a microcontroller's firmware protection. In *11th USENIX Workshop on Offensive Technologies (WOOT 17)*, Vancouver, BC, 2017. USENIX Association.
- [Rak16] Paul Rako. Xenon death flash for the raspberry pi 2. <https://www.edn.com/design/systems-design/4441502/Xenon-death-flash-for-the-Raspberry-Pi-2>, 2016. [Online; accessed 21-March-2018].
- [Rec10] Paolo Rech. Soft errors induced by neutrons and alpha particles in systems on chips. Università di PADOVA, 2010.
- [Ris18] Riscure. Laser station 2 - fast, accurate and predictable multi-in-time optical glitching for front side and back side laser attacks. <https://www.riscure.com/product/laser-station-2/>, 2018. [Online; accessed 21-March-2018].
- [RSA78] R. L. Rivest, A. Shamir, and L. Adleman. A method for obtaining digital signatures and public-key cryptosystems. *Commun. ACM*, 21(2):120–126, February 1978.
- [SA⁺02] Sergei P Skorobogatov, Ross J Anderson, et al. Optical fault induction attacks. In *CHES*, volume 2523, pages 2–12. Springer, 2002.

- [Sch14] Alexander Schlösser. Lumineszenz heißer elektronen in siliziumstrukturen alphonischer seitenkanal. Technische Universität Berlin, Fakultät II - Mathematik und Naturwissenschaften, 2014.
- [SGD08] N. Selmane, S. Guilley, and J. L. Danger. Practical setup time violation attacks on aes. In *2008 Seventh European Dependable Computing Conference*, pages 91–96, May 2008.
- [Sko05] Sergei P. Skorobogatov. Semi-invasive attacks. University of Cambridge Computer Laboratory, 2005.
- [Sko17] S. Skorobogatov. How microprobing can attack encrypted memory. In *2017 Euromicro Conference on Digital System Design (DSD)*, pages 244–251, Aug 2017.
- [STM17] STMicroelectronics. Materials declaration form. http://www.st.com/content/ccc/resource/quality_and_reliability/quality_certificate/material_declaration/group3/5b/4c/24/10/8f/0b/4a/7f/DM00421222/files/S25W_410XXXX_signed.pdf/jcr:content/translations/en.S25W_410XXXX_signed.pdf, 2017. [Online; accessed 21-March-2018].
- [Tea17] PyInstaller Development Team. PyInstaller: Freeze (package) python programs into stand-alone executables. <https://www.pyinstaller.org/>, 2005-2017.
- [Tob11] W. Tobergte. Decryption method, 2011. US Patent 8,065,531.
- [WA08] F. Wang and V. D. Agrawal. Single event upset: An embedded tutorial. In *21st International Conference on VLSI Design (VLSID 2008)*, pages 429–434, Jan 2008.
- [Wel01] Michael Welschenbach. *Kryptographie in C und C++*. Springer-Verlag, Berlin Heidelberg New York, 2001.
- [Wik18a] Wikipedia. Integrated circuit — Wikipedia, the free encyclopedia. <http://en.wikipedia.org/w/index.php?title=Integrated%20circuit&oldid=831271392>, 2018. [Online; accessed 21-March-2018].
- [Wik18b] Wikipedia. Single event upset — Wikipedia, the free encyclopedia. <http://en.wikipedia.org/w/index.php?title=Single%20event%20upset&oldid=813362152>, 2018. [Online; accessed 21-March-2018].

A. Appendix

The appendix is located on a CD-ROM. It can be inspected at the auditors Prof. Dr. Heike Neumann and Dr.-Ing. Wolfgang Tobertge.

Declaration

I declare within the meaning of section 25(4) of the Examination and Study Regulations of the International Degree Course Information Engineering that: this Bachelor report has been completed by myself independently without outside help and only the defined sources and study aids were used. Sections that reflect the thoughts or works of others are made known through the definition of sources.

Hamburg, March 23, 2018

City, Date

sign