University of West Bohemia
Faculty of Applied Sciences
Department of Computer Science and Engineering

Hardware Side-Channel Attacks on Safety Critical Devices

PhD Study Report

Enrico Pozzobon

It should be Plzeň, 2020

Technical report No. DCSE/TR-2020-03
Distribution: public

Contents

1 Introduction

Side-channel attack indicates any attack that is used to break a security algorithm without exploiting a vulnerability intrinsic to the algorithm itself, but rather by using the information leaked by its physical implementation. Some examples of information that can be leaked from an algorithm implementation are execution time, power consumption, electromagnetic radiations and even sound.

A fault injection attack instead attempts to break a security measure by injecting faults in the environment where the algorithm is executed, bringing the hardware outside of its operating conditions.

In modern software engineering, the need for secure algorithms using strong cryptography is well understood and most developers are expected to produce secure software. Though that is not always the case, it is overall true that over the past decade, a stronger focus was put on software security.

With the advancements in the security domain, the attacks also became more sophisticated, leading to the increase in popularity of side channel analysis and fault injections as attack methodology. Side channel and fault injection attacks are often the only practical way to break a correctly implemented security algorithm. Because of the rising importance of these attacks, it is crucial for a software engineer to be able to evaluate the risks originating not only from the algorithm, but also by its implementation and by the hardware that is executing it; to understand whether it is leaking information or it is vulnerable to fault injection.

2 Side Channel Analysis

This section describes which types of side channels can leak secret information even from a well designed algorithm. This list should be considered as an overview, and aims to show the channels that have been studied and exploited the most.

2.1 Timing

An attacker can obtain side channel information by examining how much time it takes for the target device to complete a computation. Timing attacks on some cryptographical functions have been studied already in 1996 in [1].

2.1.1 Memcmp Timing Attack

The simplest example about timing attacks is the standard C function `memcmp` (or equivalently `strcmp`). This function compares sequentially the bytes in two chunks of memory (or strings) for a specified length (at most N at a time, where N depends on the word size of the Central Processing Unit (CPU) architecture and on the alignment of the two memory chunks) and returns 0 if all the bytes are identical. If a difference is found, `memcmp` returns immediately a non-zero value. By examining how long it takes for a `memcmp` invocation to return, an attacker can estimate the number of identical bytes in the beginning of the two strings.

This so-called `memcmp` timing attack is especially useful if the attacker has control on one of the two strings and is able to extract information about the contents of the other, secret string. For example, suppose that the `memcmp` is comparing the user input and a secret password, and only allows the user to proceed if the input is equal to the password. Now, if the password is L bytes long, and the bytes can be any value from 0 to 255, it would take up to 256^L attempts to guess the password by exhaustive search. If the attacker is able to precisely measure the time between the submission of the password and the reception of the "login failed" message (and assuming optimal conditions for the attack, e.g. the two strings in memory are misaligned and have to be compared byte by byte), he will be able to guess each byte of the password in at most 256 attempts, by checking which of the 256 values results in a longer execution time of the comparison. This leads to the cracking of the password in just $256 \cdot L$ attempts. Figure 1 shows how it is possible to easily measure

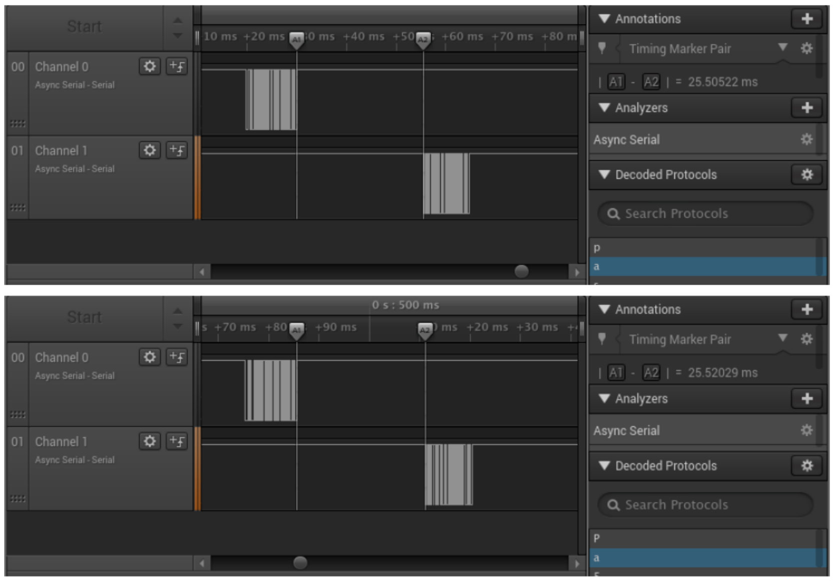

Figure 1: Logic Analyzer traces for two different passwords which are verified by `memcmp` on a microcontroller. Channel 0 is the serial TX line where the password is entered, and Channel 1 is the serial RX line where the result of the comparison is received. Notice that the time between query and response increases by 15μs by providing the correct initial byte (uppercase 'P' instead of a lowercase 'p').

the time of each comparison by using a logic analyzer on the communication lines of a device.

Luckily, the `memcmp` timing attack is easily fixed by implementing a version which executes in constant time like `CRYPTO_memcmp` [2] from the OpenSSL library which doesn't use conditional statements.

2.1.2 Spectre

Other than from the execution time of a function, timing side channel information can also be extracted from internal behaviours of a processor, such as cache misses. When an attacker is able to execute code on the victim's machine hardware, even though it is running in a different container or virtual machine, he can extract precise time measurement using the functions provided by the system, and use it to detect cache misses events. The Spectre vulnerability [3], released in 2017, consists in exploiting the speculative

```
1 if (index < simpleByteArray.length) {
2     index = simpleByteArray[index | 0];
3     index = (((index * 4096)|0) & (32*1024*1024-1))|0;
4     localJunk ^= probeTable[index|0]|0;
5 }
```

Figure 2: Javascript listing used to perform the Spectre exploit in a browser. Initially, the branch predictor is trained with `index=0` to always fulfill the `if` condition. Then, the value of `index` is set to the offset of the target memory location relative to `simpleByteArray`, causing it to fail the check on line 1, while lines 2, 3 and 4 still get speculatively executed. Finally, the attacker can check which index of `probeTable` was put in the cache by measuring the time it takes to read from `probeTable`. This reveals the value of `simpleByteArray[index]` for any value of `index`, even when it points to memory outside of the javascript sandbox.

execution feature of a processor to load arbitrary segments of memory into the cache, and then using accurate timing information to check whether a cache miss happened.

The attacker can load segments pointed by memory he has no access to, and then guess which segment was put into the cache by measuring whether a cache miss happens or not when trying to load other segments into the cache. This results in the attacker being able to read bytes from memory which is not accessible to his process.

For example, Spectre allows a javascript advert running in a web page to read all the data contained in the browser process, even outside of the javascript sandbox, including passwords and session cookies.

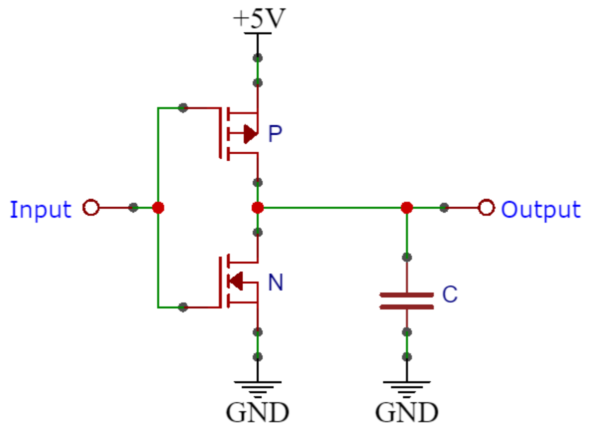

Figure 3: Typical schematic of a CMOS inverter used as an example to study the leakage of a logic gate. When the input is switched high, current flows from the supply through the P transistor into the load capacitor C. When the input switches to low, current flows from the load capacitor to ground through the transistor N. When the input stays constant, the output also stays constant, and there is no current flow.

2.2 Power Usage

Every operation inside an electronic device requires energy to be executed. This includes usage of the memory bus, mathematical calculations, usage of peripherals, amongst others. By examining the amount of current flowing from the power supply of the device into the processor, it is possible to leak sensitive data even from cryptography algorithms that are normally considered "secure".

The energy necessary to switch a bit on the output of a CMOS logic gate is often cited as the reason for information leakage, since this happens for every register that is written during the execution of a program. In particular, the Memory Data Register (MDR) is often considered the leakiest part of any MicroController Unit (MCU) since it is connected to very long silicon traces that connect to every word in the memory of the device, and therefore needs a high amount of energy in order to transfer the word to and from the memory.

As the amount of energy necessary to flip a bit from 0 to 1 and from 1 to

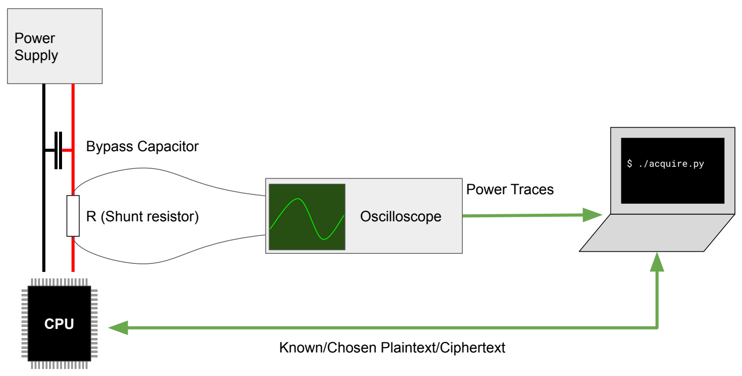

Figure 4: Attack Setup for a precise power monitoring attack. Notice that the shunt resistor is placed as close as possible to the target processor (possibly after the bypass capacitors), and a differential probe is connected across the shunt resistor.

0 is higher than the energy spent to keep a bit to its previous value, we can say that the energy necessary to write a word in a register is proportional to the hamming distance between the previous value and the new value.

2.2.1 Attack Setup

Most power analysis attacks are invasive and as such are typically used on embedded devices to extract cryptographic information. This is because it is necessary to place a current measurement device between the power supply and the MCU in order to obtain high quality power traces.

Typically, the power trace of the target processor is cut and a shunt resistor is placed in series with it. There is no fixed value to choose for the resistance of the shunt, but it should be chosen to be as large as possible to make current measurements easier, but small enough that the processor still operates correctly. For example, on a low power ATmega328P, the shunt can be 120Ω, while on a more powerful MPC5748G it should be around 1Ω. To remove an undesired low-pass effect form the capture, as many bypass capacitors should be removed from the Printed Circuit Board (PCB) as possible while keeping the device stable and the shunt resistor should be put after them, as close as possible to the target chip, as shown in Figure 4.

It is good practice to power the device using an external power supply with

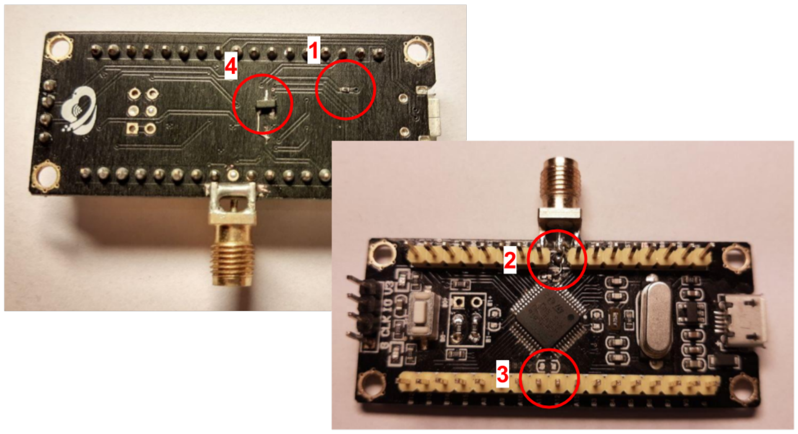

Figure 5: Modifications made to a small STM32F103C8T6 development board ("black pill") in order to make power monitoring side-channel attacks easier: the power supply trace was cut and a shunt resistor was placed in its path (1), an SMA connector was soldered to the V_{CC} pin (2), and all bypass capacitors were removed from the PCB (3). An N-channel MOSFET was also installed for crowbar glitching attacks (4), described in section 3.1.1.

low noise characteristics, as usually the switching power supplies included in most devices have a high amount of ripple that would negatively affect the measurements. After the shunt resistor has been put in place, a differential probe from an oscilloscope is placed across it. Alternatively, if the power supply is stable enough, it is sufficient to connect a single probe to the supply pin of the device (V_{DD}/V_{CC}) and then set the oscilloscope in AC coupling mode in order to discard the DC offset of the power supply.

In order to allow the capture of as much information as possible, high bandwidth connections should be used to connect the oscilloscope to the target device. Figure 5 shows an SMA connector soldered on a development board to allow high bandwidth traces to be acquired.

2.2.2 Simple Power Analysis

The simplest form of power analysis is leaking bits directly from visual analysis of the power trace. This is particularly useful on algorithms that operate on individual bits sequentially, since the contribution of each bit is visible

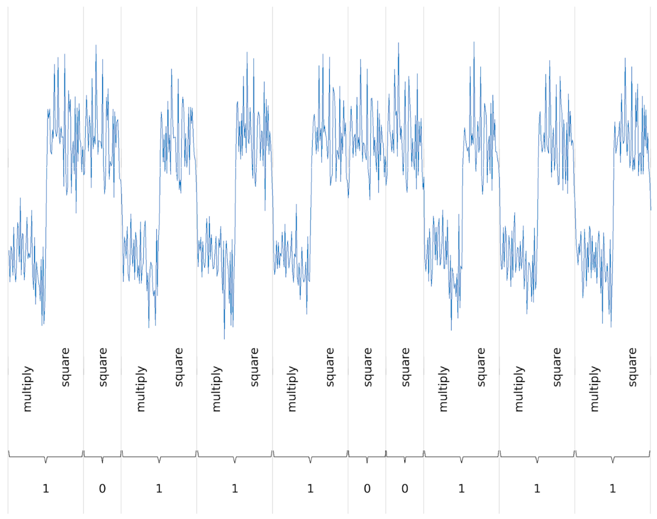

Figure 6: Simple Power Analysis of a RSA power trace (detail). When a bit of the secret exponent is 1, both the “multiply” and the “square” steps need to be executed. When the bit of the secret exponent is 0, only the “square” step is executed. Since the “square” and “multiply” operations have different power signatures, it is possible to recover all bits of the secret exponent.

at a separate moment in time in the power trace. One example of such an algorithm is the square-and-multiply operation used in RSA, for which the multiply step is only executed when the examined bit of the exponent is 1 and not when it is 0.

2.2.3 Differential Power Analysis

More refined power consumption analysis can be executed by capturing multiple traces of the same algorithm being executed with different input data, and then comparing the differences between the traces. This process is defined as Differential Power Analysis (DPA) [4].

One example usage of DPA is breaking secure symmetric encryption, like Advanced Encryption Standard (AES). In order to do this, an attacker first needs to collect a large number of power traces of the target device performing AES encryptions with the same key but with multiple different

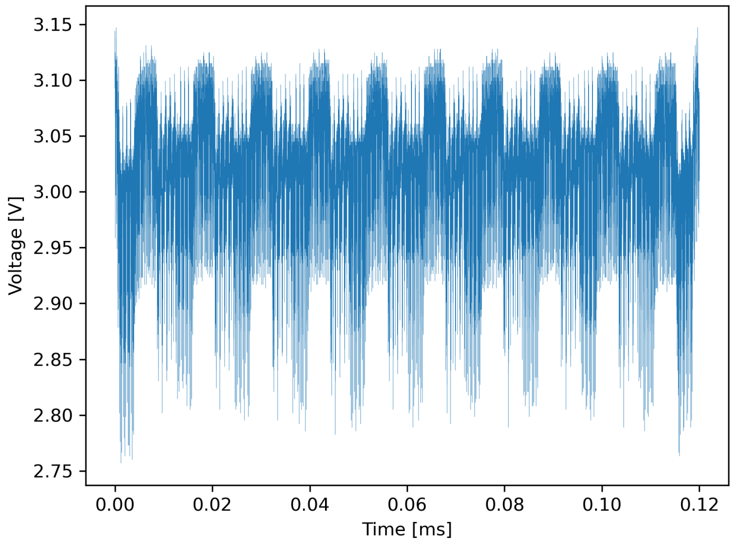

Figure 7: Power trace acquired from running TinyAES with a 128-bit key on an STM32F103C8T6, with the 10 rounds clearly visible.

plaintexts, possibly uniformly random. It is also necessary that the attacker knows either the ciphertext or the plaintext associated with each trace he acquired.

Trace Alignment

When executing DPA, it is important to align the different traces in a way that the target algorithm is at the same time offset in all the traces (also called synchronization). This alignment can be achieved with signal processing techniques, such as cross-correlation of the traces.

In some traces, the target algorithm might be interrupted by either a preemptive scheduler or any hardware interrupt. It can be useful to be able to identify these defective traces and delete or fix them.

When the clock of the target processor changes over time, it is also necessary to compensate this by "stretching" the power trace, for example using resampling. This can happen both in high end targets (e.g. the Intel Turbo Boost technology increases the clock dynamically) and on cheaper ones (e.g. many MCU allow operation using an internal oscillator, which changes its resonating frequency depending on the temperature).

Choice of an Intermediate Value and Point of Interest

Once the traces are aligned, the attacker needs to choose a sample or a range of samples which leaks some material which can be used to recover the secret key. As explained earlier, instructions that operate on the MDR are particularly "leaky" and therefore "memory store" and "memory load" instructions are preferred targets.

The sample where the attack will be performed is called the Point of interest (PoI), and should be sampled at the point in time when some intermediate value of the cryptographic computation is stored or loaded from memory.

The target intermediate value should be the result of a combination between some constant data unknown to the attacker (the secret key) and some variable data which is known to the attacker (e.g. the ciphertext in a known ciphertext attack). When possible, the target intermediate value should be chosen to be the output of a nonlinear function, in a way that small errors in the estimation of the key would be easily detectable with large changes in the power consumption. As an example, the output of an Exclusive OR (XOR) operation is a bad choice for a target intermediate value, because an error in a single bit of a byte always results in a difference in power utilization of 1/8 (assuming the power consumption is proportional to the hamming distance of the original value in the register and the new value). On the other hand, the nonlinear `SubBytes` operation of AES is a good choice because an error of a single bit in the input leads to random and uniformly distributed hamming weights in the output.

When attacking AES, the attacked intermediate value is usually the output of the first `SubBytes` operation of the encryption when performing a known plaintext attack, or of the decryption when performing a known ciphertext attack. Assuming a known plaintext attack, and remembering that the output of the first `SubBytes` of an AES encryption is:

$$intermediate = \texttt{SubBytes}(\texttt{AddRoundKey}(plaintext)) \tag{1}$$

We can write the expression for every byte i of the intermediate state:

$$intermediate[\texttt{i}] = \texttt{sbox}[plaintext[\texttt{i}] \oplus \texttt{key}[\texttt{i}]] \tag{2}$$

There are different approaches for finding the moment in time when the target intermediate value is written to memory (and therefore the PoI), for example using timing information and emulating the execution time of each instruction of the target algorithm (if the software is available to the attacker); by visual inspection of the trace and knowledge of the working of the algorithm; by correlating each "time slice" of all the traces with the data known to the attacker; or simply by exhaustive search.

Differential Power Analysis using Least Significant Bit (LSB)

For performing DPA, the attacker needs to group the traces in two sets according to some power utilization model and a selection function, and compute the Difference of Means (DoM) between the two sets. A simple selection function to use for this is the LSB model, which takes the last bit of some intermediate value of the attacked algorithm. The LSB selection function simply consists in performing an AND operation between the target intermediate value and 1, which for the attack on AES introduced before is:

$$LSB(intermediate[\mathtt{i}]) = 1 \wedge \mathtt{sbox}[plaintext[\mathtt{i}] \oplus \mathtt{key}[\mathtt{i}]] \tag{3}$$

This selection function assumes that the initial value of the register where the target intermediate value is written (usually MDR) is the same for every trace.

For any hypothesis k for byte $\mathtt{key}[\mathtt{i}]$, we can group the traces into two sets: one for which the LSB of the intermediate byte is 0, and the other for which it is 1. For every possible value k from 0 to 255, the attacker computes the value of $LSB(intermediate[\mathtt{i}])$ and uses it to split the traces in two sets, of which he computes the DoM. This can be represented as the following binary matrix H, for which every element $H_{d,k}$ represents the group (either 0 or 1) that trace d belongs to given that the key byte is k:

$$H_{d,k} = 1 \wedge \mathtt{sbox}[plaintext_d[\mathtt{i}] \oplus k] \tag{4}$$

Then, given D being the number of traces and $T_{d,t}$ being the value of trace d at sample t, the difference of mean matrix R can be represented as:

$$R_{k,t} = \frac{\sum_{d=1}^{D} T_{d,t} \cdot H_{d,k}}{\sum_{d=1}^{D} H_{d,k}} - \frac{\sum_{d=1}^{D} T_{d,t} \cdot \left(1 - H_{d,k}\right)}{\sum_{d=1}^{D} \left(1 - H_{d,k}\right)} \tag{5}$$

Then, the correct value of the examined key byte will be:

$$\hat{k} = \arg\max_{k} \left(\left| R_{k,t_{PoI}} \right| \right) \tag{6}$$

because for wrong key bytes, the incorrect contributions of the LSB terms in the DoM will average out to zero, while when k is the correct hypothesis, the DoM will be maximized.

When attacking AES-128 (with a 128-bit, 16-byte key), it is sufficient to repeat this computation 16 times on the first round of the encryption or decryption algorithm. When attacking AES-192 or AES-256, the first 16 bytes of the key are leaked in the same way, while further bytes have to be leaked from the successive rounds, keeping in mind to include the key schedule algorithm into the calculation of the intermediate values.

Figure 8 and 9 show the empirical results of trying to perform a DPA attack on TinyAES [5] with a 128 bit key on a STM32F103C8T6 MCU using a Rigol DS1054Z oscilloscope sampling at 500 Msps, in a known plaintext scenario (the attacker knows the plaintext and wants to extract the key). Both figures show results when trying to extract the first byte of the key, but the results are similar for all bytes.

2.2.4 Correlation Power Analysis

Correlation Power Analysis (CPA) differs from DPA in that the Pearson Correlation Coefficient (PCC) is used to choose one of the hypothesized key bytes instead of a simple difference, allowing to use much more accurate power utilization models instead of binary models like the LSB used before. Unlike DPA, CPA does not require splitting the traces in two groups and considering the differences between the entire groups [6].

The first steps for performing a CPA attack are identical to a DPA attack, meaning that the acquisition of power traces, their alignment, and the choice of a target intermediate value and point of interest are performed in the same way.

The attacker computes an expected power utilization at the PoI for each trace and for each hypothesised value of the key byte, usually by using the Hamming Weight of the value or its Hamming Distance from the previous contents of the register. When attacking one key byte of AES, this leads to a $D \times 256$ matrix H, where D is the number of traces and 256 is obviously the number of possible values that the examined key byte can take. For

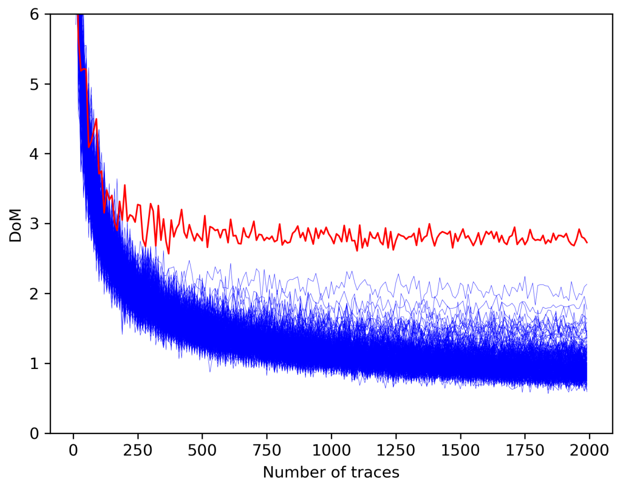

Figure 8: Maximum value of DoM of the groups created from the LSB selection function with different number of traces. The correct key byte is highlighted in red. The correct key byte starts becoming recognisable with 300 traces.

the previously considered AES known plaintext attack, using the Hamming Weight function HW as power utilization model, H for the i-th byte of the key is:

$$H_{d,k} = HW\big(\texttt{sbox}[plaintext_d[\texttt{i}] \oplus k]\big) \tag{7}$$

Finally, assuming L is the length of each trace, and T is the $D \times L$ matrix where each trace is a row, the PCC is computed between each column of T and column of H, resulting in the new matrix R which is $256 \times L$:

$$R_{k,t} = \frac{Cov(H,T)}{\sigma_H \cdot \sigma_T} = \frac{\sum\limits_{d=1}^{D} \left(H_{d,k} - \overline{H_k}\right) \cdot \left(T_{d,t} - \overline{T_t}\right)}{\sqrt{\sum\limits_{d=1}^{D} \left(H_{d,k} - \overline{H_k}\right)^2 \cdot \sum\limits_{d=1}^{D} \left(T_{d,t} - \overline{T_t}\right)^2}} \tag{8}$$

The correct value of the key byte k is then obtained by:

$$\hat{k} = \arg\max_{k} \left(\left|R_{k,t_{PoI}}\right|\right) \tag{9}$$

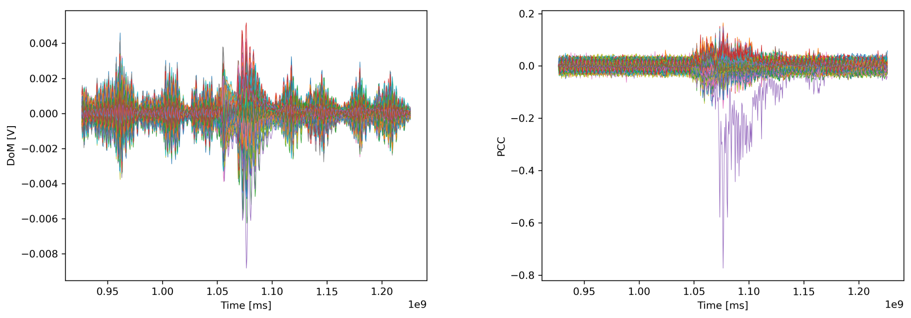

Figure 9: The Matrix $R_{k,t}$ for both DPA (left) and CPA (right) seen as several plots in the time domain, with each of the 256 lines representing one value of k, and the index t in the x-axis. Notice that the purple line shows a much larger magnitude of the correlation compared to the others, indicating that the value of k associated with that line is the correct one. Notice that the line has negative correlation because the acquired traces measured the voltage at the V_{CC} pin, which is inversely proportional to the current across the shunt resistor. The difference of the two plots also shows the superiority of the CPA compared to the DPA, since the line associated to the correct key is much more recognisable in the plot to the right.

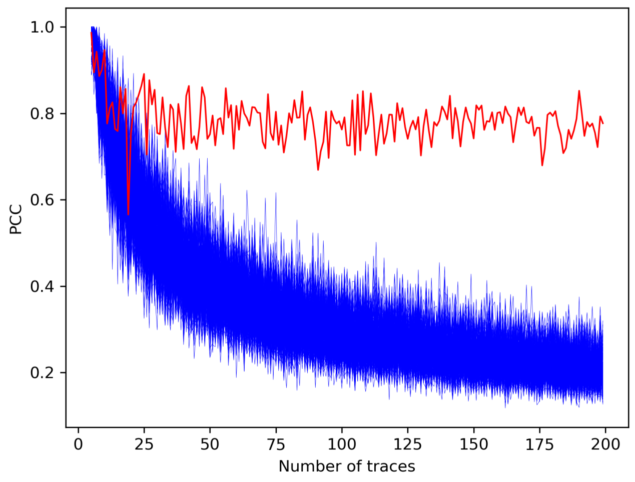

Figure 10: Maximum value of the PCC, using the same traces used in figure 8. The attack extracts the correct key byte univocally already with 40 traces, showing that CPA requires around one eight of the traces to work when compared to DPA.

because for wrong key bytes, the incorrect Hamming Weights will not correlate with the actual power usage at the PoI, while when k is the correct hypothesis, the correlation is maximised. When attacking AES, the correlation in CPA often works so well that the only sample that will correlate in any relevant way is the PoI, so it is possible to skip the selection of the PoI entirely and just assume the correct leaked key byte to be:

$$\hat{k} = \arg\max_{k} \left(\max_{t} \left(\left| R_{k,t} \right| \right) \right) \tag{10}$$

Compared to DPA, CPA correctly guesses the bytes of the key with a much lower number of traces, and needs less manual intervention to find the PoI sample in the traces. The comparison between DPA and CPA can be seen on Figures 8 and 10, which were realised from the same acquisitions, but show that CPA can be successful with a much lower amount of collected data.

2.3 Electromagnetic Emissions

The high frequency switching of logic gates in a MCU also causes information to be leaked in the form of electromagnetic radiation. A magnetic field is generated every time a charged particle (e.g. electron) travels through a conductor, and it is possible to capture this magnetic field with accurate probes. The usage of this side channel to eavesdrop is referred to as Van Eck phreaking, and it can also be used to leak secret cryptographic material.

Similar to what was described for power consumption leakages, different signal processing techniques can be used to analyse the captured data, with either simple electromagnetic attacks Simple ElectroMagnetic Attack (SEMA) looking to extract secret bits by analysing a single captured trace, or Differential ElectroMagnetic Attack (DEMA) which operate of the differences of multiple traces captured from the same algorithm. These attack techniques use the exact same mathematical functions used by power analysis attacks (Simple Power Analysis (SPA), DPA and CPA), since the magnetic field is proportional to the current which itself is proportional to the power consumption of the device.

Electromagnetic side-channel attacks can be made in a completely non-invasive way by probing completely outside of the device, or they can be executed closer to the silicon die by physically opening the processor package. Depending on the distance from the target (and therefore the degree of invasiveness), data can be leaked more or less easily from the electromagnetic field.

An example of non-invasive electromagnetic side-channel attack were the first Van Eck phreaking attacks which were performed on computer Cathode Ray Tube (CRT) monitors. These emit large amounts of radiation in normal operation and are therefore easy to eavesdrop on. Even from behind a wall and in another room, it is possible to reconstruct the image emitted by a CRT monitor, making this form of side-channel attack an efficient way to spy on the victim. Modern LCD computer monitors switch a much smaller amount of current, limiting the amount of radiations leaked, but the attack is still feasible as demonstrated by [7].

On the other end of the invasiveness spectrum, it is possible to use microscopic magnetic probes on the bare silicon die of an Integrated Circuit (IC), in such a way to extract magnetic field traces out of specific areas of the processor, like for example the memory, the power supply circuitry, or some cryptographic hardware acceleration peripheral.

2.4 Acoustic Emissions

Acoustic Cryptanalysis is also a possible attack used to steal secret information by analysing the sound waves coming from the target device, generated by the mechanical vibrations of several components.

Such attacks can be performed in a non-invasive and undetectable way either by using microphones in the proximity of the victim (such as using a smartphone sitting on the same desk as the victim computer) or from a distance using a laser microphone [8].

2.4.1 Acoustic Keyboard Snooping

One simple example of acoustic cryptanalysis is the recovery of a password from the sound of a user typing it onto a keyboard, which can be achieved through the small differences in sound of each key [9].

2.4.2 Acoustic Cryptanalysis

Similar to electromagnetic radiation, operation of a processor can also produce acoustic emission, for example due to the microphonic effects of ceramic bypass capacitors, or due to the vibrations of the power supply coils. In these cases, the emitted sounds are often related to the power consumption of the device, and therefore the same simple and differential power analysis signal processing can be used on these sound signals.

The sounds emitted by the vibrations of the components can be outside of the human hearing range, and were demonstrated by [10] to allow leakage of cryptographic material even from 4 meters away from the target device, but generally the bandwidth of this side channel is much lower than other techniques.

2.5 Optical

Side channel information can also be leaked optically through photons. One simple example of this are LED indicators, like hard drive activity indicators, the lighting patterns of which can directly leak information about what activity is being executed on the PC.

2.5.1 Photonic Emissions

Picosecond imaging circuit analysis (PICA) devices are able to detect photons which are emitted by transistors when a logic gate changes value, leading effective optical attacks on naked silicon dies [11, 12].

2.5.2 Microimaging of Decapped Chips Emissions

Another way secret information can leak out of a processor "optically" is through visual inspection of the silicon layers, using hydrofluoric acid to strip one layer at a time and a microscope to obtain high resolution photos of the logic gates [13]. While this attack is destructive, it allows an attacker to reverse engineer the hardware of the device and visually read out the bits of mask Read-only Memorys (ROMs) and flash memories.

2.5.3 Optical RAM extraction

By combining microimaging and Data Remanence (explained in the next section), it is also possible to visually extract bits out of individual RAM cells, as done by [14] using a laser microscope.

2.6 Data Remanence

Data remanence can happen whenever the hardware fails to ensure that data is truly no longer accessible after it was marked as deleted by the software.

2.6.1 Data Remanence on Persistent Storage

Obviously, if an attacker can get in possession of a persistent storage device like a hard drive or a flash memory, he can read out the entire content and any sensitive information stored within. Less obviously, data that was believed to be deleted can still be extracted using data recovery techniques. Firstly, data that is deleted from the file system is still present in the partition, and extracting a low level dump of the partition can recover the data. Data that is overwritten in magnetic storage (hard drives, tape backups) still leaves a trace of the previous contents which can be detected with a more sensitive read head. A page deleted from a flash memory is often just relocated by the wear-leveling algorithm of the device, and the data is therefore still present in one of the pages that were marked as “empty”. One simple and effective way to avoid the persistent storage of a stolen device to be extracted is to encrypt the storage drive and either let the user memorize the decryption key or store it in a Trusted Platform Module (TPM). It is important to remember that encryption is not completely solving the problem, as it is just “moving” the concern from securing the sensitive data itself to securing the decryption key, and the decryption key could still be leaked using, for example, a Cold Boot Attack.

2.6.2 Cold Boot Attacks

While storing unencrypted sensitive information in a hard drive or other forms of persistent storage is well understood to be a bad practice and is easy to avoid by the usage of encrypted storage, the same can’t be said about volatile RAM. In most situations, it is necessary to store some secret key in RAM before using it in a cryptographic algorithm, and this can lead to the leakage of that key to an attacker if a cold boot attack is performed.

A cold boot attack involves the attacker interrupting the supply of power to the target device at the moment in time where some secret key is stored in RAM, and then quickly bringing the temperature of the RAM chips down to a temperature where they retain the data even though they are not connected to power anymore. In fact, while Static RAM (SRAM) necessitates a power supply to maintain the data, and Dynamic RAM (DRAM) even requires the data to be periodically refreshed in order to avoid losing it, the time that it takes for the data to disappear without power grows longer the lower the temperature of the device is.

Once the temperature is brought down, the attacker will then connect the RAM chips to another device to extract the contents, or boot some custom software on the device under attack which will read out the memory without overwriting it (hence the name of the attack). This attack can be used to break a Full Disk Encryption (FDE) setup by reading out the part of RAM where the decryption key for the disk is stored.

The solution for Cold Boot Attacks is to use register-based key storage to store a symmetric key to use for hardware accelerated encryption of the entire RAM, but this requires the hardware to support RAM encryption.

3 Fault Injection Attacks

While side channel analysis focuses on extracting information from the hardware and software side-effects of an algorithm implementation, fault injection attacks try to disrupt the state of execution of the algorithm by making use of many of the same side channels. This usually involves the attacker bringing the target device outside of its operating range, for example changing the supply to provide a lower voltage than what is specified in the datasheet.

While it is trivial to break or render temporarily unusable a device (denial of service) with physical access, fault injection attacks are usually done in a controlled fashion to purposefully break security algorithms and get access to secret information within a device. One example usage of a fault injection attack on a MCU would be to disable the debug interface protection to be able to extract the firmware over a censored JTAG connection.

While this chapter examines only a few of the channels used for fault injection, operating the target device outside of any of its operating conditions could lead to faults which are desirable by an attacker. It is also possible that combining multiple faults leads to a successful attack when the individual faults were useless (e.g. a voltage spike fault only achieving the desired effect when the target device is below a specific temperature).

3.1 Glitching

A glitching setup involves physically connecting to the electric circuit of the target device and injecting anomalies in the power supply or other electric traces, or using electromagnetic fields to inject charges in the conductors inside a processor.

3.1.1 Crowbar Glitching

Crowbar glitching is one of the easiest forms of fault injection, and consists in connecting one of the device's power supply lines to ground in a specific time interval. The "crowbar" circuit which shorts the power supply is usually a simple N-channel MOSFET controlled by an Field-Programmable Gate Array (FPGA) which accurately activates and deactivates the crowbar at specified points in time after a "trigger" signal [15].

Crowbar glitching is particularly effective at stopping the target processor from correctly loading some contents of the volatile memory or persistent

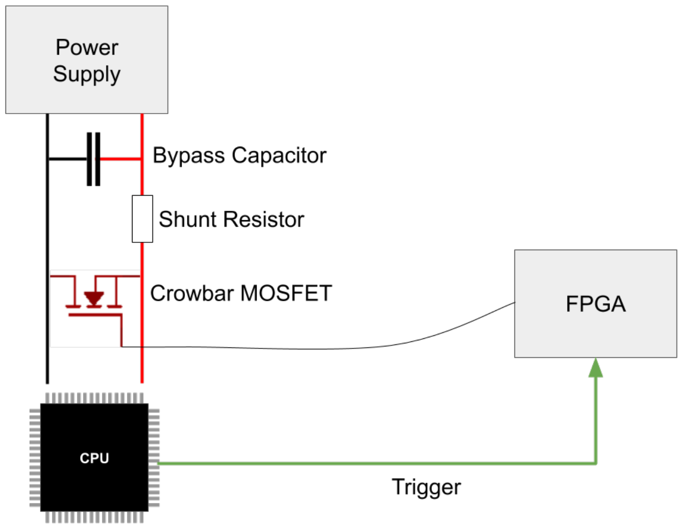

Figure 11: Example attack setup for a crowbar glitching attack. Like in a power analysis setup, it is desirable to place the circuitry necessary for the attack between the bypass capacitors and the target processor, to reduce the filtering effect.

storage. One typical example usage of crowbar glitching is preventing the fetch of a branch assembly instruction. When successful, the processor loads a NOP instruction instead and prevents the branch from being executed, thus allowing the attacker to bypass a security check. Another usage of crowbar glitches is preventing the load of some configuration from the storage, like an authentication key, so that the processor loads a zero key instead.

The target preparation and attack setup for a crowbar glitch is similar to what is done for power analysis, meaning the power supply trace is cut between the anode of the bypass capacitors and the V_{CC} pin of the target processor, and a shunt resistor is placed on the cut trace. The crowbar MOSFET is soldered as close as possible to the target processor with its source connected to ground and its drain connected to the V_{CC} pin. The shunt resistor is optional but it prevents the capacitor from “compensating” the voltage spike introduced by the crowbar. Figure 11 illustrates a schematic of the setup, while figure 5 shows an implementation of that schematic.

To understand when to activate the crowbar, a rough knowledge of the algorithm that is getting executed and its behaviour in time is necessary, but simple and differential power analysis can usually help this. This is done by giving different inputs to the algorithm and observing the differences in power usage (as it is done in DPA) to understand where the input is processed.

Once the attacker has a rough guess on when in time the crowbar glitch needs to be injected, he can then proceed with a search of the timing parameters of the glitch (offset from trigger and duration). If the glitch causes the processor to reset, this indicates that the glitch duration was too long, and it should be reduced, on the other hand, if the processor keeps working without any anomaly, it indicates that the duration was too short. For estimating the correct time offset from the trigger of when to activate the crowbar, the attacker can use exhaustive search combined with some sort of feedback from the I/O or from trace analysis to understand if the glitch was activated too early or too late. Further parameter search strategies for glitching are explored in [16].

Crowbar glitching can be especially effective against MCUs which use multiple power supply rails, like the MPC57xx family of devices which has 4 separate power supply domains: Core supply, Flash supply, Low power supply and Analog to Digital Converter (ADC) supply. In such a device, the attacker can choose to inject a glitch in one of the power domains without affecting the others, allowing him to e.g. glitch a fetch from flash while letting the execution of the code continue normally.

While the setup shown for crowbar glitching only allows the attacker to inject a voltage spike to 0V, more arbitrary waveforms are possible with more complex setups, for example using a complementary MOSFETs setup to make the raising edge of the voltage spike faster, or using a P-channel MOSFET to send a high voltage spike instead of a low voltage one. More general glitches like these are usually referred to as power glitching.

3.1.2 Clock Glitching

Clock glitching involves injecting electrical glitches in the clock line of a processor instead of its power supply line, causing extra raising and falling edges to be received by the processor, thus violating the maximum clock frequency specified by the datasheet for a brief time. This kind of attack is not effective on processors that make use of internal oscillators or produce an internal clock out by multiplying the external clock using PLL.

[17] shows the results of performing a clock glitching attack on an ATMega163 MCU by injecting a single clock period of duration $T_g \leq 57ns$ on

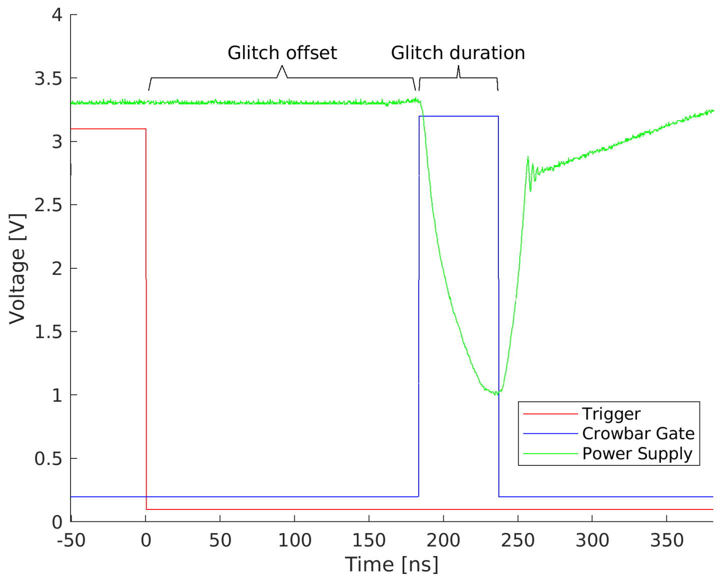

Figure 12: Plot of an oscilloscope trace of a crowbar glitch. The red line represents the trigger signal which is used as an input to the FPGAs for timing the activation of the glitch. The blue line represents the output of the FPGAs which is connected to the gate of the crowbar N-channel MOSFETs. Finally, the green line is the supply voltage as seen by the oscilloscope. The shape of the glitch is dependent on the characteristics of the MOSFETs and of the supply circuitry.

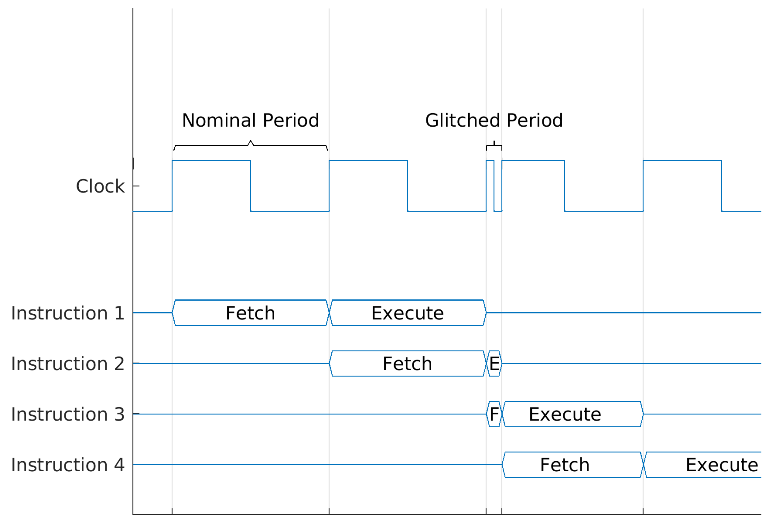

Figure 13: Example of clock glitch, together with the execution flow of the attack presented in [17]. The ATmega163 uses a 2-stage pipeline, where the fetch of the next instruction is executed at the same time as the execution of the current one. Introducing a clock glitch in the third period like shown would disrupt the fetch of the opcode for instruction 3, causing the wrong opcode to be executed in the fourth period.

a processor whose minimum nominal clock period is $T_n = 125ns$. On the abnormally short clock period, the processor tends to execute a different instruction than the one specified in the code with the executed opcodes exhibiting a stuck-at-fault pattern, meaning the instruction that is actually executed is encoded by an opcode where some bits are stuck at the value of the previously executed opcode.

A typical usage of a clock glitch is for an attacker to “skip” a specific instruction, for example by replacing a branch instruction that stops access to some protected function of the firmware with a useless opcode.

3.1.3 Electromagnetic fault injection

Electromagnetic Fault Injection (EMFI) is the technique of using electromagnetic radiation to inject faults in a target hardware, which is possible without making modifications to the circuitry or even touching the target device itself. Typically, it is performed by pulsing a large electric current through an inductor in the close proximity of the target chip, which induces currents within the circuit on the PCB and also inside the IC itself. Even when the induced currents within the target are small, they can still be enough to change the voltage on the gate of a transistor and therefore alter the flow of a program or the state of some memory or register.

3.1.4 Laser fault injection

The same induction of current used in EMFI can also be achieved by using light directly on a silicon die. The target die must be first decapped, then a laser can be aimed precisely and glitches can be injected directly at the individual transistors. [18] shows how it is possible to set the individual bits on data that is being fetched from memory on a relatively modern ARM Cortex-M3 MCU.

A simpler form of fault injection using light instead of a laser is described on [19], which describes how the readout protection eFuses of a PIC18F1320 MCU can be reset simply by shining an ultraviolet light source at the right angle on the bare die at the right angle while protecting the program flash with electric tape. The same article also shows that flash memory, just like old Erasable Programmable Read-Only Memorys (EPROMs), can also be erased using UV light.

```
1 code1a:
2       mov (X), %eax
3       mov (Y), %ebx
4       clflush (X)
5       clflush (Y)
6       mfence
7       jmp code1a
```

Figure 14: Simple x86 assembly code snippet that causes the row hammer effect [21]. X and Y are assumed to be memory locations in the two adjacent rows to the memory row containing the target data. This code consists in a loop (line 1, 7) that continuously reads data from locations X and Y (lines 2, 3) and then flushes the cache (lines 4, 5, 6) to make sure the next loop will fetch X and Y from memory again instead of the cache.

3.2 Fault injection via software

Up until now, the shown kinds of fault injection required an hardware attack setup that would bring the target outside of its absolute maximum rating limits in some ways. However, fault injection on the hardware level has been shown to also be possible from software.

3.2.1 Row hammer

Row hammer is an exploit that allows a malicious process to cause changes in memory areas which it is not allowed to write. This is achieved by rapidly activating rows that are adjacent to the target row, doing which induces a small amount of electric charge causing some cells to flip despite not being written to.

For the attack to work, the attacker needs to be able to execute low-level code on the target machine. In particular, the most barebone x86 assembly snippet that can induce errors to happen in adjacent memory rows is shown in Figure 18. If the attack is successful, it can lead to privilege escalation [20], because it allows a process to write to memory that the kernel did not give it permission to change.

The reason why this attack works is the organization of how bits are stored physically in a DRAM memory. Memory cells are packed extremely close to each other, and operate on low amounts of current to reduce heat and save energy, which results in low noise margins. Memory cells are physically organized in rows and columns, where each row is activated by the address decoder and then each column reads (or writes) a single bit. The value of each memory cell is stored as a voltage across a capacitor and needs to be periodically refreshed because of the leakage of the capacitor. The row hammer attack causes the voltage on a memory cell to leak faster than expected by inducing currents from activating the adjacent memory rows. While it would be easy to assume Error-Correcting Code (ECC) memory to be immune to this attack, it was shown that the row hammer effect indeed affects ECC memory too.

This vulnerability is extremely difficult to exploit in a real world attack because it requires the malicious process to successfully allocate two rows of memory exactly adjacent to the row which contains the target data, and because it can cause a flip in any of the bits of the row and not only the target one. Nevertheless, it demonstrates how an hardware fault can be injected from the software, without the need of physical access and possibly on a service which allows code execution to an attacker by design (for example a Virtual Private Server (VPS) on a shared host machine).

3.2.2 CLKscrew

In order to achieve a long battery life, modern smartphones adapt the frequency and voltage of their processors to match the required performance; this is possible thanks to the Dynamic voltage and frequency scaling (DVFS) registers that the hardware exposes to the software. Setting these registers to certain values which are not recommended by the manufacturer, it is possible to inject glitches in the hardware, causing anything from software crashes to violations of a Trusted Execution Environment (TEE), as demonstrated by the CLKscrew attack [22].

Effectively, setting an high frequency with a low voltage is similar to performing a clock glitching attack without hardware access, because each frequency requires a determined minimum voltage to operate stably, below which electrical signals can't activate logic gates and flip-flops fail to latch properly.

While it does not need any physical alterations to the hardware, the CLKscrew attack needs a malicious process running with elevated permissions in the smartphone operating system in order to write to the DVFS registers and disable the interrupts to achieve a precise timing. However, this attack was performed successfully to leak secrets from the ARM Trust-

Zone TEE which are supposed to be completely inaccessible from even the operating system.

4 Conclusion

As evidenced in the examples presented, security exploitation of software and hardware implementations can lead to serious consequences even in the absence of vulnerabilities in the algorithms used. This is especially true where computers are used in safety critical environments, such as the automotive field or industrial controllers.

While some software vulnerabilities can easily be fixed with software updates within days from the discovery of the bug, this is not always the case for devices which are not connected to the internet and therefore can't be updated remotely. Furthermore, correcting hardware vulnerabilities is impossible without physically replacing components, which comes at a great cost both in labor and in downtime of the system. For example, if a hardware vulnerability is found in an immobilizer system in a car (responsible for making it impossible to start a car without the correct keyfob), the manufacturer would need to recall all cars of that series and replace the device or suffer the consequence of having unhappy customers with an insecure car. If a hardware vulnerability is found in a safety-critical system of the car, like the Anti-lock braking system (ABS), it can be used by a malicious individual to hurt the owner.

To combat the exploitation of side channel information leakage and fault injection, it is necessary to discover these vulnerabilities before releasing a product to the market, by developing tools and methodologies for the assessment of the security of an implementation. After that, mitigations both hardware and software can be developed to render these attacks impossible or at the very least harder to execute.

4.1 Aims of Doctoral Thesis

- Evaluation of already existing side-channel attacks when performed on safety-critical devices (e.g. automotive and power grid ECUs).
- Development of new methodologies and tools attest the vulnerability to Side-Channel Attacks of an algorithm running on a safety-critical processor.
- Research of software mitigations to limit the effectiveness of side-channel attack and fault injection from the hardware layer.

4.2 Title of Doctoral Thesis

Hardware Side-Channel Attacks on Safety Critical Devices.

Glossary

AC Alternated Current. 7

DC Direct Current. 7

JTAG Joint Test Action Group, a standard for debugging electronic devices. 21

LCD Liquid Crystals Display. 16

LED Light Emitting Diode. 18

MOSFET Metal-Oxide-Semiconductor Field-Effect Transistor. 7, 21–24

NOP No-OPeration, an assembly instruction that idles for one cycle. 22

PC Personal Computer. 18

PLL Phase-Locked Loop. 23

RSA RSA (Rivest–Shamir–Adleman) public cryptography algorithm. 8

RX Receive. 3

SMA SubMiniature version A connector, typically used in high bandwitdh applications. 7

TX Transmit. 3

UV Ultra-violet. 26

Acronyms

ABS Anti-lock braking system. 30

ADC Analog to Digital Converter. 23

AES Advanced Encryption Standard. 8, 10–13, 15

CMOS Complementary Metal-Oxide Semiconductor. 5

CPA Correlation Power Analysis. 12, 14–16

CPU Central Processing Unit. 2

CRT Cathode Ray Tube. 16

DEMA Differential ElectroMagnetic Attack. 16

DoM Difference of Means. 11–13

DPA Differential Power Analysis. 8, 9, 11, 12, 14–16, 23

DRAM Dynamic RAM. 20, 27

DVFS Dynamic voltage and frequency scaling. 28

ECC Error-Correcting Code. 28

EMFI Electromagnetic Fault Injection. 25, 26

EPROM Erasable Programmable Read-Only Memory. 26

FDE Full Disk Encryption. 20

FPGA Field-Programmable Gate Array. 21, 24

IC Integrated Circuit. 16, 26

LSB Least Significant Bit. 11–13

MCU MicroController Unit. 5, 6, 9, 12, 16, 21, 23, 26

MDR Memory Data Register. 5, 10, 11

PCB Printed Circuit Board. 6, 7, 26

PCC Pearson Correlation Coefficient. 12, 13, 15

PICA Picosecond imaging circuit analysis. 18

PoI Point of interest. 10–12, 15

RAM Random Access Memory. 18–20

ROM Read-only Memory. 18

SEMA Simple ElectroMagnetic Attack. 16

SPA Simple Power Analysis. 16

SRAM Static RAM. 19

TEE Trusted Execution Environment. 28, 29

TPM Trusted Platform Module. 19

VPS Virtual Private Server. 28

XOR Exclusive OR. 10

References

[1] Paul Kocher. Timing attacks on implementations of diffie-hellman, rsa, dss, and other systems. *Advances in Cryptology — CRYPTO '96*, 1996.

[2] *OpenSSL* `CRYPTO_memcmp`. Constant time memory comparison.

[3] P. Kocher, J. Horn, A. Fogh, D. Genkin, D. Gruss, W. Haas, M. Hamburg, M. Lipp, S. Mangard, T. Prescher, M. Schwarz, and Y. Yarom. Spectre attacks: Exploiting speculative execution. In *2019 IEEE Symposium on Security and Privacy (SP)*, pages 1–19, 2019.

[4] Paul Kocher, Joshua Jaffe, Benjamin Jun, and Pankaj Rohatgi. Introduction to differential power analysis. *J. Cryptographic Engineering*, 1:5–27, 04 2011.

[5] *Tiny AES C*. A small and portable implementation of the AES ECB, CTR and CBC encryption algorithms written in C.

[6] Eric Brier, Christophe Clavier, and Francis Olivier. Correlation power analysis with a leakage model. *Proc of Cryptographic Hardware and Embedded Systems*, 3156:16–29, 08 2004.

[7] Markus Kuhn. Electromagnetic eavesdropping risks of flat-panel displays. *LNCS*, 3424, 07 2004.

[8] Andrea Barisani and Daniele Bianco. Sniffing keystrokes with lasers/-voltmeters. Black Hat USA 2009, 2009.

[9] Li Zhuang, Feng Zhou, and J. D. Tygar. Keyboard acoustic emanations revisited. *ACM Trans. Inf. Syst. Secur.*, 13(1), November 2009.

[10] Daniel Genkin, Adi Shamir, and Eran Tromer. Acoustic cryptanalysis. *Journal of Cryptology*, 30, 02 2016.

[11] J. Ferrigno and M. Hlavac. When aes blinks: Introducing optical side channel. *Information Security, IET*, 2:94 – 98, 10 2008.

[12] Alexander Schlösser, Dmitry Nedospasov, Juliane Krämer, Susanna Orlic, and Jean-Pierre Seifert. Simple photonic emission analysis of aes. *Journal of Cryptographic Engineering*, 3, 04 2013.

[13] Adam Laurie and Zac Franken. Decapping chips the easy hard way. DEF CON 21, 2012.

[14] David Samyde, Sergei Skorobogatov, Ross Anderson, and Jean-Jacques Quisquater. On a new way to read data from memory. *Proceedings of the First International IEEE Security in Storage Workshop*, pages 65–69, 12 2002.

[15] Colin O'Flynn. Fault injection using crowbars on embedded systems. *IACR Cryptology ePrint Archive*, 2016:810, 2016.

[16] Rafael Carpi, Stjepan Picek, Lejla Batina, Federico Menarini, Domagoj Jakobovic, and M. Golub. Glitch it if you can: Parameter search strategies for successful fault injection. In *CARDIS 2013: Smart Card Research and Advanced Application*, volume 8419, pages 236–252, 06 2014.

[17] Josep Balasch, Benedikt Gierlichs, and Ingrid Verbauwhede. An in-depth and black-box characterization of the effects of clock glitches on 8-bit mcus. pages 105–114, 09 2011.

[18] Brice Colombier, Alexandre Menu, Jean-Max Dutertre, Pierre-Alain Moellic, Jean-Baptiste Rigaud, and Jean-Luc Danger. Laser-induced single-bit faults in flash memory: Instructions corruption on a 32-bit microcontroller. pages 1–10, 05 2019.

[19] Andrew Huang. Hacking the pic 18f1320. 2007.

[20] Mark Seaborn and Thomas Dullien. Exploiting the dram rowhammer bug to gain kernel privileges. 2015. Google Project Zero.

[21] Yoongu Kim, Ross Daly, Jeremie Kim, Chris Fallin, Ji Hye, Lee, Donghyuk Lee, Chris Wilkerson, Konrad Lai, and Onur Mutlu. Flipping bits in memory without accessing them: An experimental study of dram disturbance errors. *Proceedings - International Symposium on Computer Architecture*, 06 2014.

[22] Adrian Tang, Simha Sethumadhavan, and Salvatore Stolfo. Clkscrew: Exposing the perils of security-oblivious energy management. 08 2017.