# Clock Glitch Attacks in the Presence of Heating

Thomas Korak and Michael Hutter
Institute for Applied Information Processing and
Communications (IAIK), Graz University of Technology,
Inffeldgasse 16a, 8010 Graz, Austria
emails: {thomas.korak,michael.hutter}@iaik.tugraz.at

Barış Ege and Lejla Batina
Digital Security Group - ICIS,
Radboud University Nijmegen,
Heyendaalseweg 135, 6525 AJ Nijmegen, The Netherlands
emails: {b.ege,lejla}@cs.ru.nl

***Abstract*—Fault attacks have been widely studied in the past but most of the literature describes only individual fault-injection techniques such as power/clock glitches, EM pulses, optical inductions, or heating/cooling. In this work, we investigate combined fault attacks by performing clock-glitch attacks under the impact of heating. We performed practical experiments on an 8-bit AVR microcontroller which resulted in the following findings. First, we identified that the success rate of glitch attacks performed at an ambient temperature of $100\,^{\circ}\mathrm{C}$ is higher than under room temperature. We were able to induce more faults and significantly increase the time frame when the device is susceptible to glitches which makes fault attacks easier to perform in practice. Second, and independently of the ambient temperature, we demonstrate that glitches cause individual instructions to repeat, we are able to add new random instructions, and we identified that opcode gets modified such that address registers of individual instructions get changed. Beside these new results, this is the first work that reports results of combined glitch and thermo attacks.**

**Keywords: Fault attacks, temperature, heating, non-invasive, glitches, AVR, ATmega.**

## I. Introduction

Fault attacks pose a serious threat for cryptographic implementations. In the worst scenario, a single fault can reveal the entire secret key which has been shown to be feasible by many researchers in the last decade. There exist several techniques to inject faults, the most prominent techniques are to modify the power supply or the clock source by injecting spikes or glitches. Other methods have been proven even more powerful such as optical inductions that allow a precise localization of the fault injection, global and local EM pulses, or temperature variations. In this work, we first evaluate the impact to combine these techniques to improve the performance of practical fault attacks.

In principle, fault attacks can be either non-invasive, semi-invasive, or invasive. Non-invasive fault attacks do not require a modification of the targeted device. Variations in the supply voltage, the clock signal, or the temperature are used to force a faulty behavior during the calculation of the microcontroller. Semi-invasive attacks require the de-capsulation of the chip package to, for example, be able to inject optical inductions. Exposing the opened chip to an intense light source (e.g., laser beam, flash light) or using small needles to probe single wires on the metal layer of the chip are typical techniques used in the past. Invasive attacks make modifications in the chip (e.g., add additional wire connections, cutting wires, etc.) that can sometimes lead to the destruction of the device. Clock glitch and thermo attacks, which are considered in this paper are typically non-invasive and they do not require de-capsulation or modification of the device.

After injection of a fault, several analysis techniques can be applied to reveal the secret key, e.g., Differential Fault Analysis (DFA) [1], Collision Fault Analysis (CFA) [2], or Ineffective Fault Analysis (IFA) [3]. The decision, which analysis technique to use depends on various factors like which algorithm is going to be attacked or if the injected fault affects the program flow or the processed data. A very good and detailed overview of different kind of fault attacks can be found in the work of Bar-El, Choukri, Naccache, Tunstall, and Whelan [4]; also Verbauwhede, Karaklajic, and Schmidt [5] published a classification of current fault-injection techniques and countermeasures to prevent them.

In 2011, Balasch, Gierlichs, and Verbauwhede [6] performed an in-depth analysis of the effects of clock glitches on an 8-bit AVR microcontroller. They did not target a specific implementation but evaluated the influence of clock glitches on the instruction-execution pipeline of the microcontroller. Results show that the fetch as well as the execute stage of the pipeline can be affected by clock glitches. Our work extends their research in the sense that we additionally investigate the impact of high-temperature on the same platform (this has not been done before to the best of the authors knowledge) and we additionally provide significant research outcomes as summarized in the following.

In this paper, we present various non-invasive fault attacks on an AVR ATmega162 microcontroller. We performed clock-glitch fault attacks and evaluated the impact of high ambient temperature in respect to improve the performance of the attack. We injected several faults by varying the fault-injection time and shape at two different device temperatures: $25\,^{\circ}\mathrm{C}$ and $100\,^{\circ}\mathrm{C}$. We further analyzed the impact of the clock frequency and made the same experiments while the device was clocked with 10 MHz and 20 MHz. All attacks were performed using a low-cost custom-made FPGA board that allows injection of highly parameterized clock glitches. The obtained results and contributions of this paper can be summarized as follows:

- We show that with increased temperature, the device under attack gets more sensitive to clock-glitch attacks, i.e., we were able to inject faults that were not injected at room temperature.
- We also show that with increased temperature, the

[0]http://dx.doi.org/10.1109/FDTC.2014.20. The original article is available at http://ieeexplore.ieee.org/

time frame where the device is sensitive to glitches is getting larger which makes the device more susceptible to practical attacks.

- We further demonstrate that individual AVR instructions can be simply repeated by inducing glitches (independent of the ambient temperature). In contrast to related work in [6], we identify that the program counter is not incremented due to a glitch and that no instructions are skipped.
- We are able to insert new random instructions during the program flow without skipping other instructions.
- We can confirm the outcomes of [6] and show that with our setup we were able to change the opcode of individual instructions, e.g., changing an `ADD` instruction to a `MOV`, or to change the operand addresses, e.g., changing the operand address from register `R5` to `R14`.

The rest of the paper is structured as follows. In Section II, we give a brief overview on related work. In Section III we describe the used setup to inject faults and to perform heating experiments. Section IV describes the experiments and Section V presents the results of our work and discusses the details of the induced fault types. A discussion of the obtained results is given in Section VI and conclusions are drawn in Section VII.

## II. Related Work

A huge number of successful fault attacks have been reported during the last years. It figured out that hardware as well as software implementations of symmetric and asymmetric cryptographic primitives are vulnerable. Kömmerling et al. [7] recognized the threat of fault attacks already in 1999 and proposed some low-cost protection concepts in their work.

Many papers presented successful attacks by intentionally modifying the power supply of a device during cryptographic operations. For example, Choukri and Tunstall [8] presented an attack in 2005 where the number of rounds of round-based block ciphers can be reduced by injecting faults. They used power-supply glitches for that purpose. A similar fault-injection technique was applied by Schmidt and Herbst [9] who targeted an RSA implementation that makes use of the square-and-multiply algorithm. Selmane, Guilley, and Danger presented underpowering attacks in 2008 [10]. They caused timing violations to attack an AES implementation on a smart card.

There also exist related work on electromagnetic glitch attacks. Dehbaoui et al. [11], for example, presented an attack on AES in 2013. They injected transient faults by using electromagnetic pulses. For their fault injection, no physical access to the attacked device is required and they show the applicability by modifying the round counter of an AES implementation.

Clock glitches in particular have been exploited by Fukunaga et al. [12] in 2009 to attack a wide range of block ciphers implemented on a large-scale integrated circuit (LSI). They reduce the clock period to modify the internal state of the cipher caused by setup-time violations. A detailed description of the effects of clock glitches on integrated circuits is given in [13]. In that paper, the authors confirm their theoretical assumptions by attacking the AES block cipher implemented on an FPGA.

It has been shown in the past that tampering with the clock signal, the power supply voltage, or with electromagnetic pulses, faults can be injected that mainly cause timing violations in the digital circuit. Temperature fault attacks, in contrast, have shown to be effective against data-memory modifications. One of the first who demonstrated successful temperature attacks was Skorobogatov [14] who performed data-retention attacks on different SRAM chips in 2002. He decreased the ambient temperature of these chips by cooling the devices down to $-20\,^{\circ}\mathrm{C}$ and below. He showed that data gets somehow *frozen* and can be read out after some seconds after power down. Samyde, Skorobogatov, Anderson, and Quisquater made similar experiments published in the same year in [15]. Another similar experiment was done by Müller and Spreitzenbarth [16] in 2011. They developed a tool called FROST (forensic recovery of scrambled telephones), which allows to recover the RAM content of modern Android smart phones. The tool allows to retrieve disk encryption keys from RAM and the approach is comparable to cold boot attacks on PCs [17].

While low temperatures and cooling allows to increase the data-retention and remanence time, high temperatures and heating allows to change its content: Quisquater and Samyde [18] were one of the first who observed that high temperatures causes memory errors after hours of extensive heating. Govindavajhala and Appel [19] were able to induce errors into memories using a 50 watt spotlight clip-on lamp. By heating a device up to $100\,^{\circ}\mathrm{C}$, they were able to inject faults with a probability of $71.4\,\%$. Recently, Hutter and Schmidt [20] presented heating fault-attacks on an AVR microcontroller in 2014. They operated the device above the temperature specification ($> 125\,^{\circ}\mathrm{C}$). The authors verify the efficiency of this high-temperature attack by successfully attacking an RSA implementation.

The impact of temperature in combination with power or clock glitch attacks has not been analyzed in prior work. In this paper, we therefore answer the open research question if the sensitivity of glitch attacks gets effected by temperature and if yes, to which extend.

## III. The Fault Injection Setup

In this section, we describe our used fault-injection setup. It is based on a custom-made prototyping board consisting of a flexible Field Programmable Gate Array (FPGA). Afterwards, we give an overview about the heating process to let the device under attack operate in a higher temperature environment. Finally, we give a brief introduction to the targeted AVR family of microcontrollers and describe the basic architecture and instruction set.

### A. Fault Board for Clock Tampering

In order to inject faults during the computation of a microcontroller, we designed a custom-made Printed Circuit Board (PCB). This board consists of a *XILINX Spartan-6 XC6SLX45* FPGA and allows communicating with a PC over a

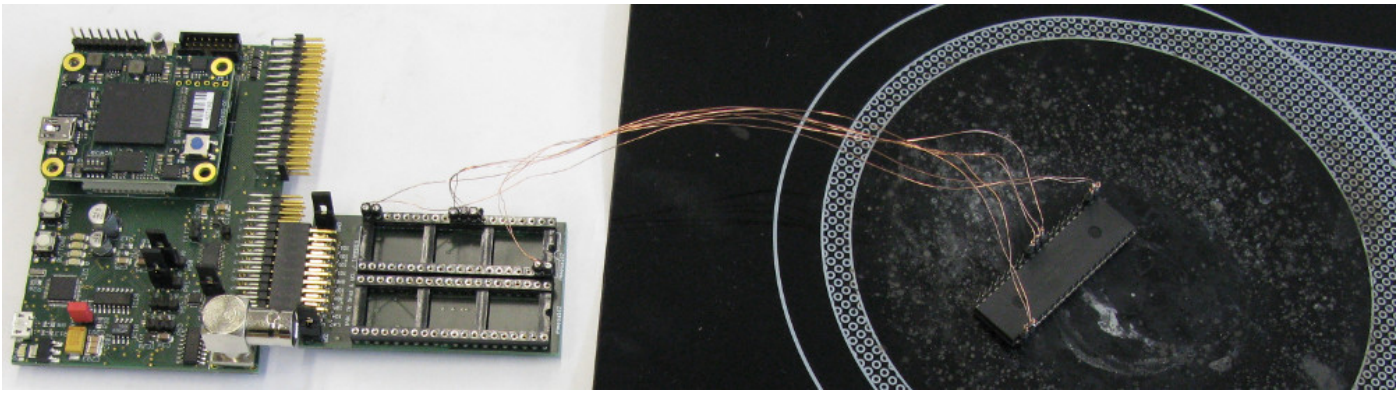

Fig. 1. The experimental setup: The fault board is located on the left side and the microcontroller is connected to it using thin copper wires. This allows to place the microcontroller in the middle of the heating plate.

USB-over-serial connection. It is equipped with many I/O pins that can be used to connect a wide range of microcontrollers or other FPGAs. Figure 1 shows the setup on the left side of the figure.

We mainly used two pins of the FPGA to generate clock glitches for the microcontroller. The first pin provides a *clock signal* that can be adapted by the FPGA (meaning that we are able to change the clock duration and edges individually). The second pin provides a *trigger signal* that indicates the starting point of the glitch injection. Next to these two pins, we used two power pins from our fault board to supply the microcontroller. We set the power supply to 3.3 Volts in our experiments which is within the normal specification range of the AVR.

Figure 2 shows the experimental setup as a block diagram. The clock and the trigger signals were captured by a digital storage oscilloscope, i.e., we used the *PicoScope 5203* from *Pico Technology* for these measurements. The oscilloscope and also the fault board is connected to a computer that runs Matlab. Via Matlab scripts we were able to automatically configure our fault board (e.g., setting different clock-glitch parameters) and to start and stop individual measurements of the clock and trigger signal.

In order to heat-up the microcontroller and to evaluate the influence of heating during clock-glitch injections, we placed the device on top of a heating plate. Our custom-made fault board is connected to this microcontroller via insulated copper wires. These wires had a diameter of 0.2 mm and allow to

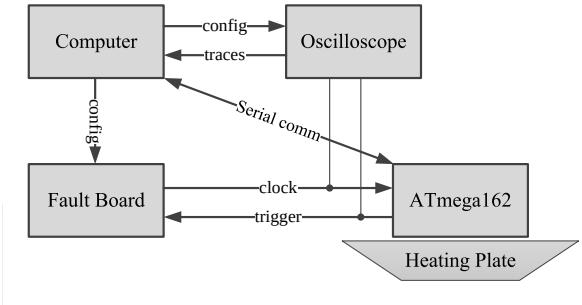


Fig. 2. Block diagram of the experimental setup.

place the microcontroller in the middle of a heating plate while our fault board keeps exempt from extensive heating. Figure 1 shows the setup on the right side of the figure.

**Clock-glitch generation.** We applied a similar approach like presented in [13], [21] in order to inject clock glitches into the target device. A block diagram of this clock-generation unit is shown in Figure 3. The clock generation works as follows. The hardware module takes as inputs a reference clock signal $clk$ and a glitch-enable signal $gl_{en}$. By using the reference clock signal $clk$, we are able to generate phase-shifted versions of it which we further denote by $clk_{s1}$ and $clk_{s2}$. For this, we used two Digital Clock Managers (DCMs) of the FPGA that provide these phase-shifting capabilities. Furthermore, we denote $gl_{act}$ the time when when the glitch is *active*. The $gl_{act}$ signal is generated using a two-input AND-gate. One input is the synchronized $gl_{en}$ signal, named $gl_{en,sync}$ while $clk_{s1}$ serves as the second input. As an output, the clock-generation unit provides a new clock signal, further denoted by $clk_{gl}$, that includes an inserted glitch.

The shape of the clock glitch can be parameterized by two values, i.e., $d_1$ and $d_2$. The first value $d_1$ represents the starting time when the glitch is inserted. The second value $d_2$ represents the ending time of the glitch. To be more exact, these two values represent the phase shifts of $clk_{s1}$ and $clk_{s2}$ and define the final *shape* of the inserted clock glitch. Figure 4 shows all involved signals and the final clock $clk_{gl}$ for a small value of $d_1$ (meaning that the clock glitch is started very early after a positive clock edge). Figure 5 shows the signals for a bigger value of $d_1$ (meaning that the clock glitch is started right before the end of a positive clock edge).

The figures also show the low times of the clock signal as denoted by $t_{low}$. By having a closer look at the two figures, it shows that the low times of the glitch-injected clock signal $clk_{gl}$ is different and depends on the parameter $d_1$. In particular, $t_{low,gl} = t_{low} - d_1$; so the low time becomes shorter the higher the value of $d_1$. This means that the negative clock edge becomes shorter the later the clock glitch is injected during the positive clock edge. Both the two parameters $d_1$ and $d_2$ and also the decreased low time $t_{low,gl}$ can be used to cause faulty computations during the computation of the microcontroller.

For the experiments, we used and evaluated the impact of

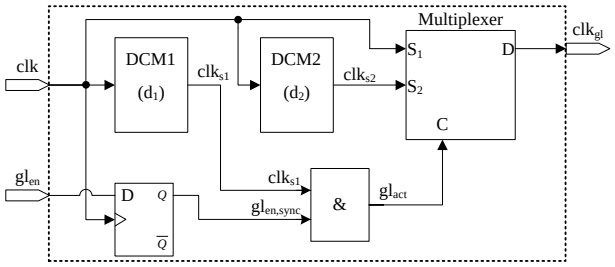


Fig. 3. Block diagram of the clock-generation unit using two Digital Clock Manager (DCM) blocks of the FPGA.

two different clock frequencies, i.e., 10 MHz ($T = 100$ ns) and 20 MHz ($T = 50$ ns). Figure 6 and Figure 7 show the measured clock signals for three different $[d_1, d_2]$ settings, once for a reference clock frequency of 10 MHz and once for a reference clock frequency of 20 MHz, respectively. These figures also show the relationship between $d_1$ and $t_{low,gl}$. The corresponding settings for $[d_1, d_2]$ are shown in the legends of these plots. Values for $d_2$ in the range of 7.0 ns and 50.0 ns were used for the setting of $f_{clk} = 10$ MHz. This equals glitch frequencies between 20 MHz and 142 MHz. For the setting of $f_{clk} = 20$ MHz, values for $d_2$ between 7.0 ns and 25.0 ns were used. The clock glitch is inserted after a trigger event on a predefined pin of the FPGA. Defining the number of clock cycles between the trigger event and the glitch insertion allows to precisely control the point in time when the clock glitch actually takes effect.

### B. Heating Plate with Temperature Measurement

For heating up the microcontroller, a laboratory heating plate from Schott instruments (SLK 1) was used. It does not allow to accurately control the temperature using a control system, but measuring the temperature and regulate the heating power figured out to be sufficient for the performed experiments. For the temperature measurements we have used a PT100 sensor element. Temperature sensors based on the PT100 sensor element are very common for industrial applications. According to the temperature, the resistance of the PT100 changes and at 0°C the resistance equals to 100 Ω. Several approaches in order to measure the resistance (or a proportional value like voltage drop or current) exist, depending on the intended accuracy. For the experiments in this work, we have used the resistance measurement function of a Fluke 111 TRUE RMS multimeter in order to acquire the resistance value. For that purpose the value shown on the multimeter has been subtracted by the resistance of the connection wires and with an online tool[1] the temperature value has been calculated.

*Remark:* During the experiments it figured out that the heating plate introduces electromagnetic interferences which lowers the signal quality. This did not influence our experiments, but if power measurements are performed in addition, we suggest to use a resistor-based heating element for heating up the device under test. Furthermore, this resistor-based heating element can be controlled very easily and the temperate can be adjusted more accurately than a heating plate. We used this resistor-based heating element to perform our final power measurements.

[1] http://www.thermibel.be/documents/pt100/conv-rtd.xml

### C. The Investigated Microcontroller - AVR ATmega162

We decided to analyze heating effects during clock glitch attacks on an ATmega162. The reason for that choice is that this microcontroller is commonly used especially in the field of embedded systems and has been widely investigated by the crypto-research community due to its ease of use, availability, and architecture documentation.

The ATmega162 is an 8-bit low-power microcontroller from Atmel. It is part of the AVR family and is based on a RISC architecture. The ATmega162 supports 131 instructions where most of them are single-cycle operations. The device can be clocked up to 8 MHz with an internal clock source or up to 16 MHz using an external clock (depending on the supply voltage). It provides 32 internal general-purpose registers (denoted by `R0` ... `R31`) that can be used by applications. Some of them are dedicated to special functions such as the registers `R0` and `R1` which store the result of a multiplication, or the sets (`R26`,`R27`), (`R28`,`R29`), and (`R30`,`R31`) which can be used for memory addressing purposes (they are referred to registers `X`, `Y`, and `Z` in the documentation). It further has a 1 kB of internal SRAM and 16 kB of programmable flash memory. Further information about the ATmega162 can be found in the datasheet [22].

Table I summarizes the parameters which are important for the further experiments. Note that the minimum clock signal low-time is of special interest because if it gets lower than 25 ns, one can tamper with the program counter of the device.

TABLE I. AVR ATMEGA162: EXTERNAL CLOCK DRIVE

| $V_{CC}$ | 2.7-5.5 V | | 4.5-5.5 V | | |
|---|---|---|---|---|---|
| Parameter | Min. | Max. | Min. | Max | Units |
| Clock Frequency | 0 | 8 | 0 | 16 | MHz |
| Clock Period | 125 | - | 62.5 | - | ns |
| High Time | 50 | - | 25 | - | ns |
| Low Time | 50 | - | 25 | - | ns |
| Period Change | - | 2 | - | 2 | % |

## IV. THE EXPERIMENTS

In the following, we describe the experiments in detail. First, we describe the written microcontroller program which was used to analyze the impact of clock glitches as well as temperature on the investigated instructions. Second, we explain the measurement process where we used a Matlab script to control the entire fault-injection process.

**The AVR microcontroller program.** At the beginning of the execution, some device-specific configurations are performed, including setting up the serial interface for communication with the control computer as well as setting the clock source to an external clock. After that, the program runs in a loop and waits for instructions from the measurement PC. Specific commands are used to select different initialization values for the registers and different instructions which should be affected by the clock glitch. After reception of a command (further denoted by $cmd$), the registers are initialized to known values, which enables us to evaluate the impact of an eventually occurring fault and guarantees a defined start

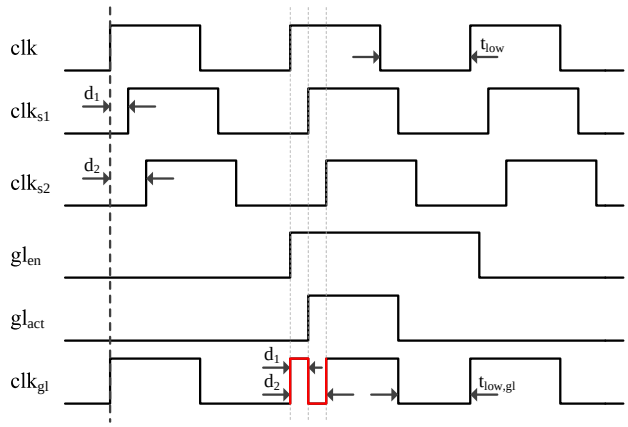


Fig. 4. Clock-glitch generation for a small value of $d_1$.

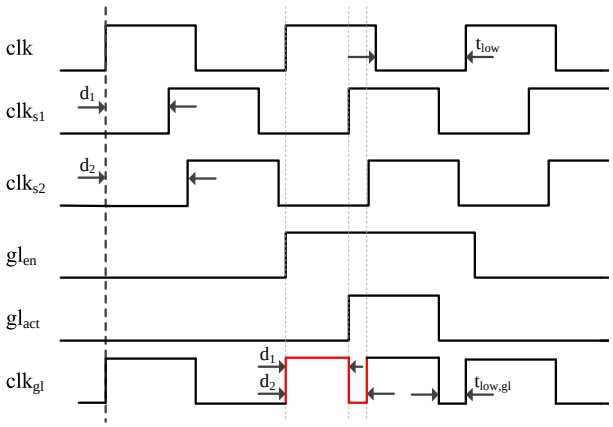


Fig. 5. Clock-glitch generation for a big value of $d_1$. Note that $t_{low,gl}$ is lower compared to $t_{low}$.

state. After rising a trigger pin and executing a fixed number of `NOP` instructions, the targeted instruction is executed. We have surrounded the targeted instruction with `NOP` instructions to avoid any side effects, introduced by the clock glitch, on the rest of the program flow. The fixed number of `NOP` instructions between the trigger event and the attacked instruction further allows us to precisely select the point in time the clock glitch should occur. At the end of the execution, the current register states are transferred to the control computer to evaluate the impact of the current glitch shape on the instruction.

**Evaluation process.** The following experiments are all performed for ambient temperatures of 25 °C (room temperature) and 100 °C, respectively. Note that the maximum temperature rating is specified to 125 °C in the datasheet, so we do not operate the device beyond its specifications. The whole procedure was automated using a MATLAB script in order to maximize the performance and minimize human interaction. The following values had to be defined before the script was started:

| | |
|---|---|
| $d_{1,start}$ | Start value for $d_1$ |
| $d_{1,end}$ | End value for $d_1$ |
| $d_2 - d_1$ | Shape of the inserted glitch (glitch duration) |
| $\Delta d_1$ | Step size for increasing $d_1$ |
| $N$ | Number of repititions for same glitch shape |
| $cmd$ | Command defining the targeted instruction |
| $Reg_{ref}$ | Reference register values for the current command |
| $f_{clk}$ | Clock frequency for the microcontroller |

For each parameter set $[d_1, d_2, cmd]$, the following procedure is then performed $N$ times:

1) Configure the fault board with the current clock glitch parameters $[d_1, d_2]$.
2) Arm the clock glitch function on the fault board to insert the clock glitch with the defined shape after a trigger event.
3) Send the command $cmd$ to the microcontroller.
4) Wait for the response of the microcontroller.
5) Compare the received register contents with the reference register contents $Reg_{ref}$ of the reference execution without clock glitch and store the values if there are deviations.

## V. Results

In this section, we present two main sets of results based on experiments when the AVR microcontroller is clocked with 10 MHz ($T = 100ns$) and 20 MHz ($T = 50ns$). Although the maximum clock frequency is documented as 16 MHz in the documentation of ATmega162, a slight overclocking to 20 MHz clock did not cause any faulty behavior to the operations when no clock glitch was present. The reason we did additional experiments with overclocking was that to push the device to its limits, and therefore making it more vulnerable to glitches.

Although we have done experiments with various instructions, the results summarized in this section are collected when injecting a clock glitch during the execution phase of the instruction `ADD R16, R5` (`R16` ← `R16+R5`). `R16` is the destination register and `R5` is the source register. For verification purposes, different source and destination registers were used for further experiments. It turned out that injecting a clock glitch while the instruction `ADD R16, R5` is executed, three different types of faults can be caused, depending on the configured glitch-shape parameters:

1) Inconsistent faults affecting the value of the destination register of an `ADD` instruction as well as one neighboring register.
2) Consistent faults modifying the executed instruction.
3) Consistent faults repeating the executed instruction.

In the following, we describe each fault type in a more detail and provide the experimental results.

### A. Inconsistent Faults

The first type of fault is generated when a clock glitch is introduced early in the positive clock-edge phase of the clock signal. Exemplary glitch shapes generating this type of faults are shown in the top plots of Figure 6 and Figure 7. In case where the device was clocked at 10 MHz (see Figure 6), an additional positive clock edge is inserted around 220 ns, 6.20 ns after the previous positive clock edge. In the case when the device is clocked at 20 MHz (see Figure 7), an additional positive clock edge is inserted around 95 ns, 5.75 ns after the previous positive clock edge. This type of fault not only sets

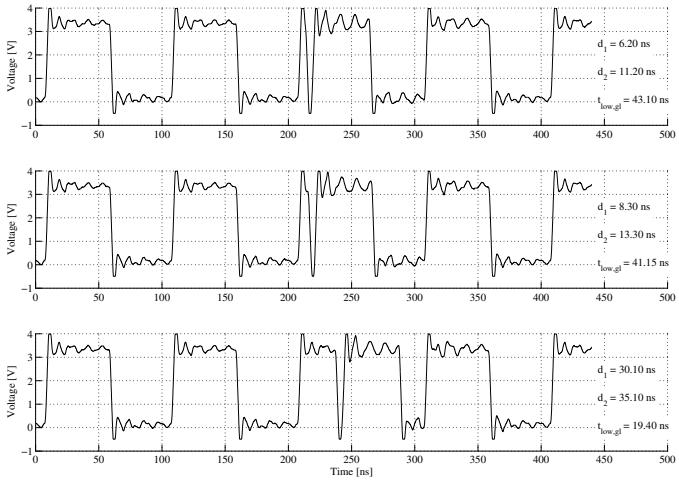


Fig. 6. Three different clock-glitch shapes generated with the fault board. The frequency of the reference clock $clk$ was set to 10 MHz (T = 100 ns) for generating this plot (glitch length = 5 ns).

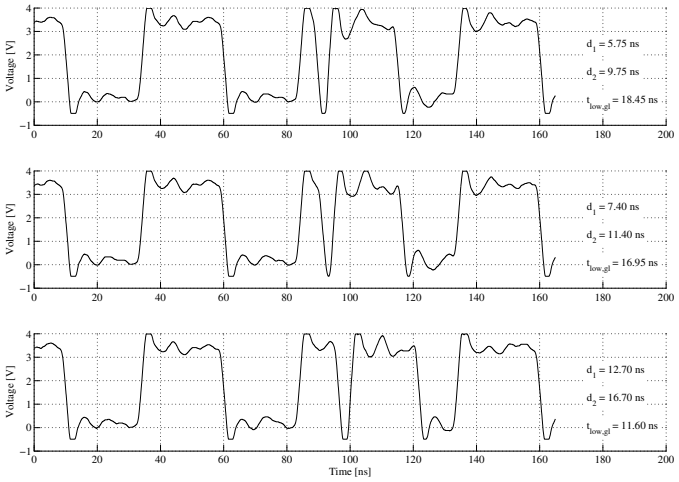


Fig. 7. Three different clock-glitch shapes generated with the fault board. The frequency of the reference clock $clk$ was set to 20 MHz (T = 50 ns) for generating this plot (glitch length = 4 ns).

the destination register (`R16`) to an incorrect value but also sets the next register (`R17`) in the register bank to an unpredictable value. Although the register `R17` gets cleared most of the time, our experiments also showed that the fault causes also other non-reproducible values such as 1, 17, 96, or 97. An interesting fact here is that this type of fault occurs only when the destination register is in the second half of the register bank (from register `R16...R31`), no fault is caused in registers `R0...R15`.

In order to further evaluate this inconsistent behavior, we slightly modified the attacked instruction. Different destination registers were used, all located in the upper half of the register bank. The results showed that for even destination registers (`R16`, `R18`, ...) the current and the *next* register are changed by the glitch in an unpredictable way. For odd destination registers (`R17`, `R19`, ...) the current and the *previous* register are changed by the glitch in an unpredictable way. This is interesting since there exist also a special AVR instruction, called `MOVW`, which is able to move two 8-bit values to other registers, i.e., it copies one register pair into another register pair. A pre-requisite of this instruction however is that the source and also the destination register addresses need to be even (which somehow corresponds to our findings that a glitch can cause modifications not only in a single register but also in a register pair which is due to the underlying hardware architecture and supported instruction set). Also the instructions `ADIW` (add immediate to word) and `SBIW` (subtract immediate from word) are potential candidates for instructions modifying two subsequent registers. A closer look on the description of these instructions, however, reveals that they only work for registers `R24...R31`.

Further evaluation showed that the glitch attack indeed causes to replace the addition by a `MOV` (or `MOVW`) instruction. Interestingly, it shows that also the following instruction is replaced by a `MOV`. We verified this fact by executing two subsequent `ADD` instructions and attacking the first one. For glitch shapes affecting two registers, the second `ADD` instruction was not executed. These results show that the opcode of two consecutive instructions is modified by a single clock glitch. A prerequisite for an executed `MOVW` instruction is that the values in the modified, subsequent registers also appear in two additional, subsequent registers in the register bank which should have been used as the source registers, which was not the case in our experiments. This observation as well as the replacement of the instruction after the attacked `ADD` instruction make the execution of two `MOV` instructions due to the clock glitch more likely than an execution of a `MOVW` instruction.

If the instruction before the attacked `ADD` instruction is not a `NOP` as it was the case in the previous experiments, different results for the similar glitch shape could be observed. We have changed the preceding `NOP` instruction by an `ADD` instruction, leading to the attack result described in Table II. The attacked `ADD` instruction at program counter value `n+1` is replaced by a `MOV` instruction with the source register equal to the destination register of the previous `ADD` instruction and the destination register equal to the destination register of the attacked `ADD` instruction. In the next clock cycle, the attacked `ADD` instruction is executed replacing the original instruction, in the current example a `NOP`.

TABLE II. MODIFICATIONS OF INSTRUCTIONS WHEN A CLOCK GLITCH FOR INCONSISTENT FAULTS IS INTRODUCED AT `N+1`.

| Prog. Counter | Execution without fault | Execution with fault |
|---|---|---|
| $n$ | ADD $R_{d,Add1}$, $R_{s,Add1}$ | ADD $R_{d,Add1}$, $R_{s,Add1}$ |
| $n+1$ | ADD $R_{d,Add2}$, $R_{s,Add2}$ | MOV $R_{d,Add2}$, $R_{d,Add1}$ |
| $n+2$ | NOP | ADD $R_{d,Add2}$, $R_{s,Add2}$ |

Summing up these first observations, it can be said that too many parameters (*previous instruction, next instruction, involved registers*) affect the outcome of the injected clock glitch. The faults result in un-predictable values so that the practical usage to target an implementation is somehow limited.

### B. Modified Instructions

The second type of fault is generated after the first type when increasing the glitch parameters which essentially shifts the glitch to the right within the clock cycle. Exemplary

glitch shapes generating this type of faults are shown in the middle plots of Figure 6 and Figure 7, for 10 MHz and 20 MHz clock, respectively. These type of faults can modify the executed instruction in an unpredictable but reproducible way. For instance, the instruction `ADD R16, R5` can be turned into a copy instruction: `MOV R16, R20`, or to another addition instruction with modified source register: `ADD R16, R14`. The resulting instructions caused by a clock glitch are summarized in Table III. The order of the instructions given in the table follows the same order that they appear when the glitch is moved to the right within the clock cycle. We have also added the opcodes of the resulting instructions to Table III. Comparing the opcodes of the modified instructions with the original one, the following statements can be made: Bits in the opcode can flip from 1 to 0 and from 0 to 1, three bits can be flipped at most, the destination register is never affected in our experiments.

TABLE III. THE RESULTING INSTRUCTIONS WHEN A CLOCK GLITCH IS INTRODUCED WHILE EXECUTING THE INSTRUCTION `ADD R16, R5`

| Instruction | Opcode |
|---|---|
| `ADD R16,R5` | `0000 1110 0000 0101` |
| `ADD R16,R4` | `0000 1110 0000 0110` |
| `ADD R16,R20` | `0000 1111 0000 0100` |
| `MOV R16,R4` | `0010 1110 0000 0100` |
| `ADD R16,R14` | `0000 1110 0000 1110` |
| `ADD R16,R12` | `0000 1110 0000 1100` |
| `ADD R16,R13` | `0000 1110 0000 1101` |

We verified these changes in the executed instructions by repeating the experiments with different sets of initialization values for the registers, and this type of faults turned out to be reproducible for a given glitch shape and a fixed temperature. The reproducible behavior of these faults suggest that with a glitch introduced at a certain time within the high part of the clock cycle, it is possible to force the instruction decoder to make a faulty computation in a reproducible way. However, the instruction is modified in a way that the destination register is kept unchanged. Either the instruction itself is changed or the source register, as shown in Table III. Our additional experiments using different destination registers also did not yield any modification to the executed instruction where the destination register gets modified. The glitch parameters that give these kinds of faults are shown in Figure 8 and Figure 9.

### C. Repeated Instructions

The third type of fault results in the instruction which is in the execution phase to be repeated and the rest of the instructions are run in the correct order. Glitch shapes leading to this kind of faults are shown in the bottom plots of Figure 6 and Figure 7. Reconstructing the internal workings of this kind of fault is not straightforward, since we investigate the device in a black box model and have no access to internal information such as: the value of the program counter or the output of the instruction decoder.

In our experiments we observed that for some particular glitch shapes (also visualized at the bottom plots of Figure 6 and Figure 7), we consistently observed that the result of the addition (58) was much lower than the expected value (213), while introducing the glitch in the execution phase of the instruction `ADD R16, R5`. Since the initial value of the register 5 is 101 and the initial value of the register 16 is 112, the expected result of the addition would be $101+112 \equiv 213 \pmod{2^8}$. If this addition was to be repeated, then the results would be

$$101 + 112 + 101 \equiv 314 \equiv 58 \pmod{2^8}.$$

It should be noted here that all arithmetic operations are in modulo $2^8$ since the Device under Test (DUT) is an 8-bit microcontroller. Therefore, this particular result of the addition made apparent that the current instruction was executed twice.

There are two main possible scenarios that can cause this kind of behavior: either the instruction which is in the pre-fetch phase is replaced with the instruction which is currently in the execution phase, or the program counter is not updated in the presence of such a clock glitch. To further investigate which type of scenario was realized in our experiments, we changed the test code from a single `ADD` instruction to two distinct `ADD` instructions as shown in the upper half of Table IV. This can verify if the program counter is updated or not depending on the final value of the register 16.

However, there is a subtle point here that is worth noting. If the program counter is updated and if the instruction which is already pre-fetched to the instruction register is executed independent of the current value of the program counter, then the program counter would have the value $n + 3$ after the execution of the second `ADD` operation, in turn skipping the instruction which should have been executed when program counter is $n + 2$. Therefore, adding two more instructions (a `CLR` instruction and an `LDI` instruction) in the test code enabled us to verify if the program counter is in fact updated and how the instruction register behaves in relation to the program counter. The complete test code is as shown in Table IV: First, two additions are performed, each having `R16` as destination. The first addition uses `R5` and the second `R21` as source registers. The other two instructions are a clear instruction, clearing the content of `R4` and a load instruction, writing `0xFF` to `R18`. Also on the right most column, the current value of register `R16` is shown to make it easier for the reader to follow the expected behavior of the microcontroller.

TABLE IV. ORDER OF INSTRUCTION CALLS TO TEST THE GLITCHES CAUSING TO REPEAT THE EXECUTED INSTRUCTION, WHERE THE VALUE OF `R5` IS 101 AND THE VALUE OF `R21` IS 117.

| Prog. Counter | Instruction | Value of `R16` |
|---|---|---|
| $n-1$ | `NOP` | 112 |
| $n$ | `ADD R16, R5` | 213 |
| $n+1$ | `ADD R16, R21` | 74 |
| $n+2$ | `CLR R4` | 74 |
| $n+3$ | `LDI R18, 0xFF` | 74 |
| $n+4$ | `NOP` | 74 |

This experiment confirmed that in fact no instruction is skipped (or replaced with a `NOP` instruction) and the first addition instruction is executed twice. This is only possible if the program counter is not updated and provided the fact that this type of fault happens whenever the negative time of the clock is very short, we believe that the program counter update on the DUT depends on the negative time of the clock signal. Our interpretation of the instruction modification is given in Table V. In the table, the first row shows the initial value of the register 16 before the glitch (112). In the following rows, the program flow and how the value of the register 16 is updated is shown. Further experiments showed that repeating the `ADD`

instruction is independent of the involved registers as well as the values stored in the registers.

TABLE V. INTERPRETATION OF A GLITCH RESULTING IN THE REPETITION OF AN INSTRUCTION, WHERE THE VALUE OF R5 IS 101 AND THE VALUE OF R21 IS 117.

| Prog. Counter | Instruction | Value of R16 |
|---|---|---|
| $n-1$ | NOP | 112 |
| $n$ | ADD R16, R5 | 213 |
| $n$ | ADD R16, R5 | 58 |
| $n+1$ | ADD R16, R21 | 175 |
| $n+2$ | CLR R4 | 175 |
| $n+3$ | LDI R18, 0xFF | 175 |
| $n+4$ | NOP | 175 |

Similarly to the case given in Table V, it can also happen that either the first or the second execution of the same instruction can get modified. In our experiments we observed values in $\{57, 66, 67\}$ which can only be explained by a repeated ADD instruction but either in the first or the second execution of the instruction is somehow misinterpreted in the instruction decoder and therefore resulting in a value much lower than the expected result.

## VI. THE INFLUENCE OF AMBIENT TEMPERATURE

Figure 8 and Figure 9 show the types of faults that can be caused for a given difference between the glitch parameters $d_1$ and $d_2$ when the microcontroller is clocked at 10 MHz, and 20 MHz respectively. In both figures, only the parameters which consistently produce a particular type of fault are plotted. On the vertical axis, different types of faults that can be generated are listed, and the horizontal axis is related to the timing of the glitch which is introduced within the clock cycle. For the experiments done in $25°$ Celsius, the results are plotted with a blue circle ($\circ$). To represent the results for the experiments that the device was heated to $100°$ Celsius, a red ($*$) is used. Note that a more detailed version of Figure 8 is given in Figure 12 in Appendix A for the interested reader.

By increasing the ambient temperature during clock-glitch injections, we made the following key observations:

1) We identified that essentially the same faults that can be caused during room temperature can be also caused at high temperatures. So there is no negative impact of higher temperatures in obtaining different fault types. However, we identified that the clock-glitch injection time is shifted and needs to be performed at a later instant of time within the targeted clock cycle. The reason for this is explained later in Section VI-A.
2) In some cases, e.g., in the case when the clock-glitch shape has a duration of $d_2 - d_1 = 5\,ns$, it shows that the sensitivity window, i.e., the time when the device is sensitive to clock-glitch injections, is getting larger with higher temperature. This means that glitch attacks under the presence of heating are easier to perform in practice because the time frame where the glitch causes faults is larger.
3) It is possible to consistently cause faults for certain glitch parameters (e.g.,: $d_2 - d_1 = 6\,ns$) when the device is running under high ambient temperature, which is not possible within room temperature though. That means that the success rate of clock-glitch attacks on that device is getting higher the higher the ambient temperature.
4) It should be noted that when the device was clocked at 20 MHz (results shown in Figure 9), the type of fault causing to repeat the executed instruction did not occur when the device was heated to $100°$ C. Although this behavior is visible when the difference between glitch parameters $d_2 - d_1 = 4\,ns$ in $25°$ C, our experiments did not yield this kind of fault when the device was heated and clocked at 20 MHz. The repeated instructions at this temperature were always modified in the second execution in the presence of a clock glitch at $100°$ C.
5) When the device is overclocked at 20 MHz, the longer glitch shapes, like $d_2 - d_1 = 6$ or $7\,ns$, result in a larger variety of faults when the device is heated up to $100°$ C. This behavior can be clearly seen in the bottom plots of Figure 9.

### A. Temperature Derating Factor

By analyzing Figures 8 and 9, it is clearly visible that the sensitivity window for inducing the faults is shifted to the right when the temperature is higher. We are now going to explain this effect with a simple example. Let us assume the simple synchronous circuit shown in Figure 10. The registers sample the data input $D$ at the positive clock edge and the same clock signal $CLK$ is provided to all registers. Between the transmitting registers $Reg_{TX}$ and the receiving registers $Reg_{RX}$ a combinational logic block is inserted. This combinational logic block has a propagation delay $t_{p,comb}$. This propagation delay defines the time after which the output has settled to a stable value in the worst case after an input change. A proportional relationship between $t_{p,comb}$ and the junction temperature exists, i.e., the higher the junction temperature is, the longer is the propagation delay of a combinational circuit. In industry, the *derating factor* $K_\Theta$ is used to describe the influence of the temperature on the speed of a circuit. In order to fully describe the impact of PTV (process, temperature, and voltage) variation on the speed of a circuit, derating factors describing the process ($K_P$) as well as the supply voltage ($K_V$) also exist. The nominal timing is multiplied with the product of $K_\Theta$, $K_P$, and $K_V$ to get the timing for a specific condition. More detailed information about the derating factors can be found, for example, in Chapter 12 (p. 590) in the book *Digital Integrated Circuit Design* by H. Kaeslin [23].

Several intermediate values ($IV_1$, $IV_2$, $IV_3$, ...) appear at the output of the combinational logic block before it settles to the stable value. If the receiving registers sample their input before the combinational block provides a stable value (due to a too high clock frequency or the insertion of a clock glitch to perform a fault attack), this consequently leads to wrong results. The intermediate values, which can be observed at the output of the combinational logic block depend on the previous input value $data_{in}(t-1)$ and the new input value $data_{in}(t)$. Each intermediate value can be observed for a specific time interval. With rising temperature, the speed of the combinational logic slows down as discussed above, so the temperature influences the signal-propagation time and therefore the fault-injection window when a glitch is effective or not. This fact is shown in the timing diagram in Figure 11. The

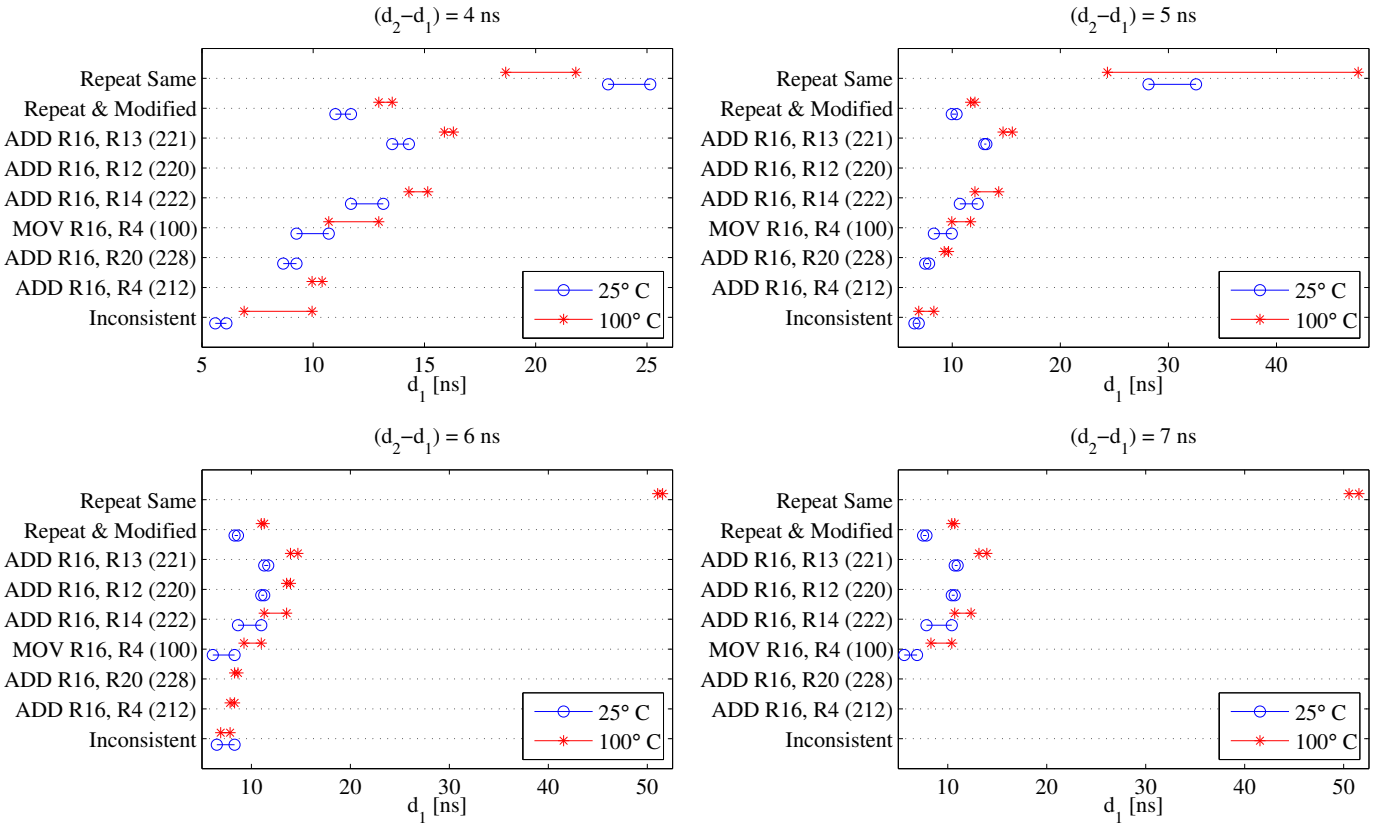


Fig. 8. Types of glitches generated depending on the glitch parameters used and the ambient temperature, while executing the instruction `ADD R16, R5`. The device is clocked at 10 MHz for these experiments.

proportional relationship between temperature and speed of the circuit increase the size of the signal-propagation intervals as well as shifts their position to the right. If a similar clock glitch is inserted at two different temperatures, the type of the fault is different if the receiving registers sample different intermediate values. This fact is also illustrated in Figure 11. By applying the modified clock signal $CLK_{gl}$, the receiving registers are forced to sample the output of the combinational logic block $data_{out}$ before it has settled to a stable value. For a temperature of $25°C$, $IV_3$ is sampled while for $100°C$, $IV_2$ is sampled.

## VII. Conclusion

In this paper, we aimed to answer the question if and how temperature effects the success rate of clock-glitch fault attacks on cryptographic implementations. As a target device, we evaluated the temperature impact on an 8-bit AVR ATmega162 microcontroller. In addition to this contribution, we present and discuss new fault types caused by clock glitches on the AVR. Our investigations showed that it is possible to repeat individual instructions, to insert new random instructions, and to change the opcode of instructions such that the operand address is changed to another value. We further demonstrated that with increased ambient temperature, clock-glitch attacks are getting more effective. This means that it is possible 1) to inject faults that were not injected during room temperature and 2) to increase the time frame when the device is sensitive to faults. The latter fact makes practical attacks more easy to perform since the device gets less sensitive to the exact fault-injection time.

As future work, we plan to investigate resistor-based heating elements to improve our setup. We further plan to increase the ambient temperature to the limits of the AVR microcontroller to evaluate the impact. As opposed to this, we also want to evaluate the impact of low-temperature attacks, meaning that we are interested in analyzing the impact of glitch attacks under the presence of cooling.

## Acknowledgements

The work presented in this article has been supported by the European Cooperation in Science and Technology (COST) Action IC1204 (Trustworthy Manufacturing and Utilization of Secure Devices - TRUDEVICE), by the European Commission through the FP7 program under project number 610436 (project MATTHEW), by the Technology Foundation STW (project 12624 - SIDES), and by the Netherlands Organization for Scientific Research NWO (project ProFIL 628.001.007).

## References

[1] E. Biham and A. Shamir, "Differential Fault Analysis of Secret Key Cryptosystems," in *Advances in Cryptology - CRYPTO '97, 17th Annual International Cryptology Conference, Santa Barbara, California, USA, August 17-21, 1997, Proceedings*, ser. LNCS, B. S. K. Jr., Ed., vol. 1294. Springer, 1997, pp. 513–525.

[2] L. Hemme, "A Differential Fault Attack Against Early Rounds of (Triple-) DES," in *Cryptographic Hardware and Embedded Systems-CHES 2004*. Springer, 2004, pp. 254–267.

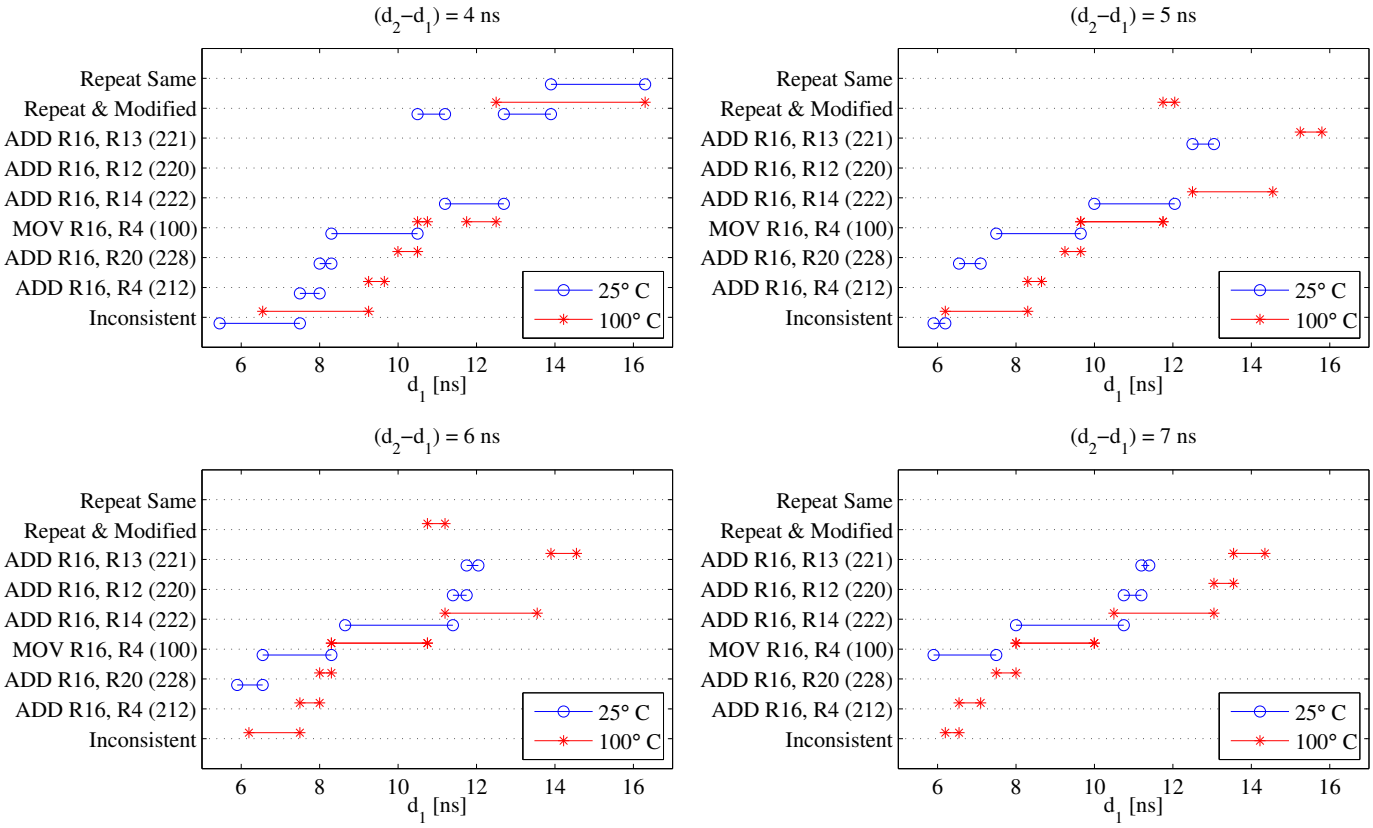


Fig. 9. Types of glitches generated depending on the glitch parameters used and the ambient temperature, while executing the instruction `ADD R16, R5`. The device is clocked at 20 MHz for these experiments.

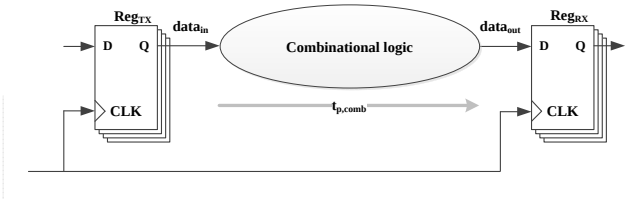


Fig. 10. A simple synchronous circuit with a combinational logic block between two storage elements. $t_{p,comb}$ equals the propagation delay of the combinational logic block.

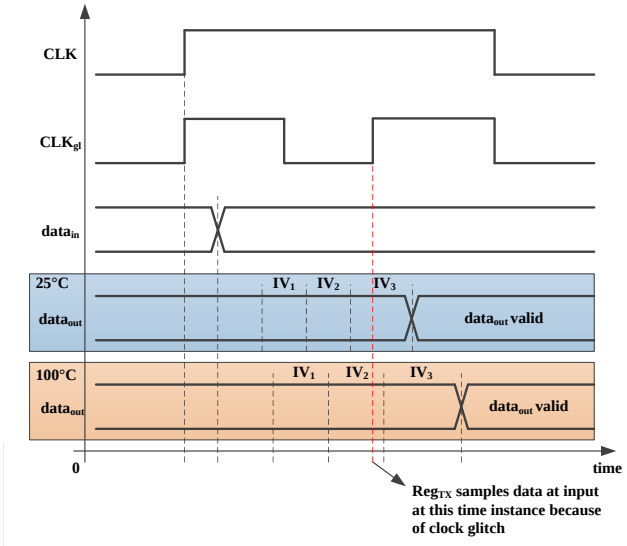


Fig. 11. Timing diagram showing the influence of temperature on the result of a clock glitch insertion. At different temperatures, different intermediate values are sampled by the receiving registers due to the changed timing.


[3] J. Blömer and J.-P. Seifert, "Fault Based Cryptanalysis of the Advanced Encryption Standard (AES)," in *Financial Cryptography, 7th International Conference, FC 2003, Guadeloupe, French West Indies, January 27-30, 2003, Revised Papers*, ser. LNCS, R. N. Wright, Ed., vol. 2742. Springer, January 2003, pp. 162–181.

[4] H. Bar-El, H. Choukri, D. Naccache, M. Tunstall, and C. Whelan, "The Sorcerer's Apprentice Guide to Fault Attacks," Cryptology ePrint Archive (http://eprint.iacr.org/), Report 2004/100, 2004. [Online]. Available: http://eprint.iacr.org/

[5] I. Verbauwhede, D. Karaklajic, and J.-M. Schmidt, "The Fault Attack Jungle - A Classification Model to Guide You," in *Workshop on Fault Diagnosis and Tolerance in Cryptography, FDTC 2011*, L. Breveglieri, S. Guilley, I. Koren, D. Naccache, and J. Takahashi, Eds. IEEE, 2011, pp. 3–8.

[6] J. Balasch, B. Gierlichs, and I. Verbauwhede, "An In-depth and Black-box Characterization of the Effects of Clock Glitches on 8-bit MCUs," in *Workshop on Fault Diagnosis and Tolerance in Cryptography, FDTC 2011*. IEEE, 2011, pp. 105–114.

[7] O. Kömmerling and M. G. Kuhn, "Design Principles for Tamper-Resistant Smartcard Processors," in *Proceedings of the 1st USENIX Workshop on Smartcard Technology (Smartcard '99), Chicago, Illinois, USA, May 1011, 1999*. McCormick Place South: USENIX Association,

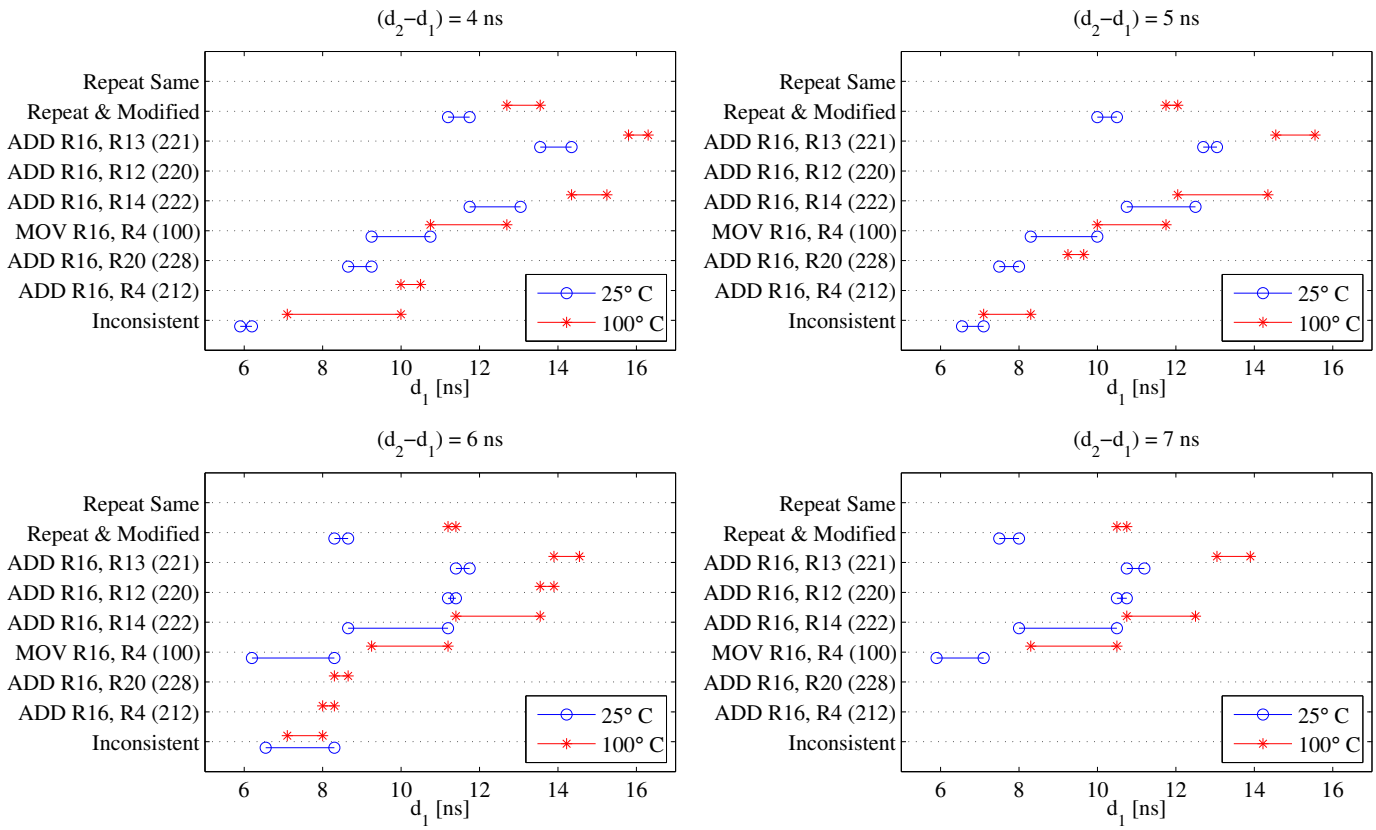


Fig. 12. Types of glitches generated depending on the glitch parameters used and the ambient temperature, while executing the instruction `ADD R16, R5`. The device is clocked at 10 MHz for these experiments.


May 1999, pp. 9–20, iSBN 1-880446-34-0.

[8] H. Choukri and M. Tunstall, "Round Reduction Using Faults," *Workshop on Fault Diagnosis and Tolerance in Cryptography, FDTC 2005*, vol. 5, pp. 13–24, 2005.

[9] J.-M. Schmidt and C. Herbst, "A Practical Fault Attack on Square and Multiply," in *Workshop on Fault Diagnosis and Tolerance in Cryptography, FDTC 2008*. IEEE, 2008, pp. 53–58.

[10] N. Selmane, S. Guilley, and J.-L. Danger, "Practical Setup Time Violation Attacks on AES," in *Dependable Computing Conference, 2008. EDCC 2008. Seventh European*. IEEE, 2008, pp. 91–96.

[11] A. Dehbaoui, A.-P. Mirbaha, N. Moro, J.-M. Dutertre, and A. Tria, "Electromagnetic Glitch on the AES Round Counter," in *COSADE 2013, Paris, France*. Springer, 2013, pp. 17–31.

[12] T. Fukunaga and J. Takahashi, "Practical Fault Attack on a Cryptographic LSI with ISO/IEC 18033-3 Block Ciphers," in *Workshop on Fault Diagnosis and Tolerance in Cryptography, FDTC 2011*. IEEE, 2009, pp. 84–92.

[13] M. Agoyan, J.-M. Dutertre, D. Naccache, B. Robisson, and A. Tria, "When Clocks Fail: On Critical Paths and Clock Faults," in *Smart Card Research and Advanced Application*. Springer, 2010, pp. 182–193.

[14] S. Skorobogatov, "Low temperature data remanence in static RAM," University of Cambridge Computer Laboratory, Tech. Rep., June 2002. [Online]. Available: http://www.cl.cam.ac.uk/techreports/UCAM-CL-TR-536.pdf

[15] D. Samyde, S. P. Skorobogatov, R. J. Anderson, and J.-J. Quisquater, "On a New Way to Read Data from Memory," in *IEEE Security in Storage Workshop (SISW02)*. IEEE Computer Society, 2002, pp. 65–69.

[16] T. Müller and M. Spreitzenbarth, "FROST - Forensic Recovery of Scrambled Telephones," in *ACNS 2013, Banff, AB, Canada.*, M. Jacobson, M. Locasto, P. Mohassel, and R. Safavi-Naini, Eds., vol. 7954, 2011, pp. 373–388. [Online]. Available: http://link.springer.com/chapter/10.1007%2F978-3-642-38980-1_23

[17] J. Halderman, S. D.Schoen, N. Heninger, W. Clarkson, W. Paul, J. A.Calandrino, A. J.Feldman, J. Appelbaum, and E. W.Felten, "Lest We Remember: Cold Boot Attacks on Encryption Keys," in *17th USENIX Security Symposium, San Jose, CA, July 2008*, 2008, pp. 45–60.

[18] J.-J. Quisquater and D. Samyde, "Eddy Current for Magnetic Analysis with Active Sensor," in *Conference on Research in SmartCards (E-Smart'02), Nice, France.* UCL, September 2002, pp. 185–194.

[19] S. Govindavajhala and A. W. Appel, "Using Memory Errors to Attack a Virtual Machine," in *IEEE Symposium on Security and Privacy, Proceedings of the 2003*, 2003, pp. 154–165. [Online]. Available: http://dl.acm.org/citation.cfm?id=830563

[20] M. Hutter and J.-M. Schmidt, "The Temperature Side-Channel and Heating Fault Attacks," in *CARDIS 2013, 12th, Berlin, Germany.*, ser. Lecture Notes in Computer Science, 2013, in press.

[21] S. Endo, T. Sugawara, N. Homma, T. Aoki, and A. Satoh, "An on-chip glitchy-clock generator and its applicataion to sage-error attack," in *COSADE 2011, Darmstadt, Germany*, ser. Workshop Proceedings COSADE 2011, 2011, pp. 175–182.

[22] Atmel Corporation, "ATmega 162/v Datasheet," 2003. [Online]. Available: http://www.atmel.com/Images/Atmel-2513-8-bit-AVR-Microntroller-ATmega162_Datasheet.pdf

[23] H. Kaeslin, *Digital Integrated Circuit Design – From VLSI Architectures to CMOS Fabrication*. Cambridge University Press, 2008, iSBN 978-0-521-88267-5.

## APPENDIX A
## DETAILED PLOT OF 10 MHZ FAULTS

Since the horizontal scale of Figure 8 is too wide because of the wide range of successful glitch parameters, a more detailed plot presenting results for smaller glitch parameters is given in Figure 12.