# A Practical Second-Order Fault Attack against a Real-World Pairing Implementation

Johannes Blömer$^\dagger$, Ricardo Gomes da Silva$^*$, Peter Günther$^\dagger$, Juliane Krämer$^*$ and Jean-Pierre Seifert$^*$
$^*$*Technische Universität Berlin*
*Germany*
*Email:* {ricardo,juliane,jpseifert}@sec.t-labs.tu-berlin.de
$^\dagger$*University of Paderborn*
*Germany*
*Email:* {johannes.bloemer,peter.guenther}@upb.de

## Abstract

Several fault attacks against pairing-based cryptography have been described theoretically in recent years. Interestingly, none of these has been practically evaluated. We accomplish this task and prove that fault attacks against pairing-based cryptography are indeed possible and even practical — thus posing a serious threat. Moreover, we successfully conduct a second-order fault attack against an open source implementation of the eta pairing on an AVR XMEGA A1. We inject the first fault into the computation of the Miller Algorithm and apply the second fault to completely skip the final exponentiation. We introduce a low-cost setup that allows us to generate multiple independent faults in one computation. The setup implements these faults by clock glitches which induce instruction skips. With this setup we conducted the first practical fault attack against a complete pairing computation.

## Keywords

Pairing-Based Cryptography, Fault Attacks, eta Pairing.

## I. INTRODUCTION

Public-key cryptography is based on mathematical problems which are assumed to be hard. The secret information is protected by an attacker's inability to solve these problems. However, by inducing hardware or software faults into the computation of an algorithm and by analyzing the faulty result, an attacker might reveal that secret information without the need to solve the mathematical problem. Since fault attacks were first described in 1997 [1], they have been applied against various cryptographic algorithms [2] and became a standard tool to facilitate cryptanalysis. Nowadays, many techniques exist to induce faults, e.g., clock glitching, power glitching, and laser beams [3]. To thwart countermeasures against fault attacks, even two faults within one computation have been performed [4]. These attacks are often called second-order attacks [5].

We conducted the first practical fault attack against a real-world pairing implementation. Pairings are bilinear maps defined over groups on elliptic curves. Originally, they have been used for cryptanalytic techniques [6]. In 2001, however, they gained the research communities attention when they were used to realize identity-based encryption [7], [8]. Today, a wide range of different pairings is used [9] and several cryptographic protocols are based on pairings, e.g., attribute-based encryption [10], identity-based signatures [11], and key agreement protocols [12]. Moreover, pairings help to secure useful technologies such as wireless sensor networks [13], [14].

When we argue about attacks on pairings, we need to understand that most pairings are computed with the so-called Miller Algorithm, followed by a final exponentiation. In some cases like [15], the final exponentiation can be efficiently inverted but in general, both steps are considered hard to invert [16], [17]. This is different from other cryptographic primitives such as elliptic curve cryptography (ECC) with only one computational step. Here, a single fault is sufficient to reveal the secret [18]. Furthermore, in ECC the secret key is a scalar [19], while in pairing-based cryptography (PBC) it is an elliptic curve point [7]. Hence, attacks on ECC [18] can not simply be applied to PBC.

Previous results on fault attacks against pairing computations have two drawbacks. None of the proposed attacks against PBC have been practically evaluated on a real pairing implementation to date. Furthermore, the existing theoretical approaches use only a single fault to target either the Miller Algorithm, e.g. [20], [21], or the final exponentiation [22]. It is not clear how the two steps can be combined to break the complete pairing with a single fault. In [16], it was even argued that inverting pairings in one combined step does not seem feasible. Therefore, it is very natural to inject two faults in one pairing computation to facilitate the inversion of pairings.

*Our contribution:* We conducted the first practical fault attack against a real-world pairing implementation. We successfully realized a second-order fault attack against an open source implementation [23] of the eta pairing [24]. We skipped two instructions in the pairing computation. With the first fault we attacked the Miller Algorithm and with the second fault we completely skipped the final exponentiation. We show a general mathematical analysis for this type of attack and apply it to the concrete fault attack we conducted. Together with an automation of the analysis, this easily leads to the secret key: for the most cases we were able to reveal the secret key in a few minutes. This proves the claim that fault attacks on pairings are a serious threat. Moreover, we show that our mechanism of skipping instructions can be used to practically realize previous attacks. In order to perform general second-order attacks, we built a setup which precisely generates multiple clock glitches to skip specific instructions of the code.

*Remark on the eta pairing:* The eta pairing is no longer recommended for security applications [25]–[27]. It was important for us not to attack a self-made and tweaked implementation. For our target device, an XMEGA A1 from the Atmel AVR family, the eta pairing was the only publicly available implementation with acceptable performance. We emphasize that our attack is not at all restricted to this pairing and can be directly applied to other pairings.

*Organization:* The rest of this work is structured as follows: in Section II we present mathematical background information on pairings. In Section III, we discuss related work on fault attacks against pairings and categorize existing attacks into two distinct categories. In Section IV, we describe the low-cost setup that we used for the fault induction. In Section V, we describe how we used this setup to conduct a second-order fault attack against an open source pairing implementation. We resume the description of the second-order fault attack in Section VI by explaining how the faulty pairing computations can be analyzed to reveal the secret input point. Finally, we conclude in Section VII.

## II. BACKGROUND ON PAIRINGS

Let $E$ denote an elliptic curve that is defined over a finite field $\mathbb{F}_q$, where $q = p^m$ for some prime $p$ and $m \geq 1$. Based on the chord and tangent law [19], we define an additive group $(E, \oplus)$. With $[a]U$ we denote scalar multiplication of $U$ with $a \in \mathbb{Z}$. For $U, V \in E$, let $l_{U,V}$ denote a normalized equation of the line through $U$ and $V$. With $g_U$ we denote a normalized equation of the tangent line through $U$ at $E$. Hence, $l_{U,V}$ and $g_U$ represent the lines that occur while computing $U \oplus V$ and $[2]U$, respectively.

A pairing is an efficiently computable, non-degenerate bilinear map $e : \mathbb{G}_1 \times \mathbb{G}_2 \to \mathbb{G}_T$, where $\mathbb{G}_1$ and $\mathbb{G}_2$ are $r^{\text{th}}$ order subgroups of an elliptic curve $E$. In this work, we always assume $r$ to be prime. The group $\mathbb{G}_T$, which is a subgroup of $\mathbb{F}^*_{q^k}$, is also of order $r$. Here, $k$ is the so-called embedding degree, which is defined as the smallest integer $k$ such that $r$ divides $(q^k - 1)$. Most pairings $e(P, Q)$ on elliptic curves are computed by first computing the Miller function $f_{n,P}(Q)$ [28] followed by a final exponentiation to the power $z = (q^k - 1)/r$. For example with $P \in E(\mathbb{F}_q)$, the reduced tate pairing can be computed as $e(P,Q) = f_{n,P}(Q)^z$ [29]. Since the Miller function can be efficiently evaluated with the Miller Algorithm, cf. Algorithm 1, these two steps are often called Miller loop and final exponentiation [30].

For a detailed background on the arithmetic of elliptic curves and cryptographic pairings we refer to [19], [31].

In this work, we invert a pairing with the help of faults. We induce faults in the computation of $e(P, Q)$ and reveal the secret input point $Q$. Thus, the faults facilitate the mathematical cryptanalysis and the so-called first

**Algorithm 1** Miller Algorithm and final exponentiation

```
Require: $n = \sum_{j=0}^{t-1} n_j 2^j$ with $n_j \in \{0,1\}$ and $n_{t-1} = 1$, $P, Q \in E$
Ensure: $f_{n,P}(Q)$
$T \leftarrow P$, $f \leftarrow 1$
for $j = t-2$ .. $0$ do
    $f \leftarrow f^2 \cdot g_T(Q)/l_{[2]T,-[2]T}(Q)$
    $T \leftarrow [2]T$
    if $n_j = 1$ then
        $f \leftarrow f \cdot l_{T,P}(Q)/l_{T\oplus P,-(T\oplus P)}(Q)$
        $T \leftarrow T \oplus P$
    end if
end for
$f \leftarrow f^z$                                        ▷ final exponentiation
return $f$
```

argument pairing inversion problem (FAPI-1): given a point $P \in \mathbb{G}_1$ and a value $\gamma \in \mathbb{G}_T$, both chosen at random, find $Q \in \mathbb{G}_2$ such that $e(P,Q) = \gamma$ [16]. (FAPI-2 is the problem with P unknown and $Q \in \mathbb{G}_2$ chosen at random.) In the literature, FAPI-1 is usually split into two parts: the exponentiation inversion problem is, given $(P, z, \gamma)$, to compute the field element $\beta \in F_{q^k}^*$ such that $\beta^z = \gamma$ and $\beta = f_{n,P}(Q)$, where $Q \in \mathbb{G}_2$ is the solution of FAPI-1 for $(P, \gamma)$ [32]. The other part of FAPI-1 is the Miller inversion problem: given $(n, \beta, P)$ with $n \in \mathbb{N}, \beta \in F_{q^k}^*$ and $P \in \mathbb{G}_1$ chosen at random, compute the point $Q \in \mathbb{G}_2$ such that $f_{n,P}(Q) = \beta$, where $f_{n,P}(Q)$ is the output of the Miller loop for input $(n, P, Q)$.

## III. Existing work on Fault Attacks against Pairings

In recent years, several fault attacks against pairings have been proposed [15], [20]–[22], [33]–[35]. Most of them focus on the Miller Algorithm, while lately also an attack against the final exponentiation was published [22].

Some works contain categorizations of fault attacks, which help to structure this field and to classify known and new attacks. In [2], fault attacks were classified following the main components of a processor, regarding the precision of a fault an adversary is able to induce, and regarding the particular abstraction level on which a fault is exploited. Fault attacks have also been categorized as having three main effects on an algorithm: knock out a step in the computation, cause a loop to either end prematurely or run over, and to cause the data being operated on to be corrupted in some way [21]. In the same work, the authors also considered the locations that a data corruption fault can target in the Miller loop. Regarding fault attacks on pairing computations, faults were also described as corrupting precomputed values or parameters, inputs to the pairing, and intermediate values [33]. All these criteria are helpful to describe fault attacks on a high level, but they are not unambiguous: A fault which knocks out a step in the computation so that the loop runs over cannot be uniquely categorized in accordance with [21]. A fault in a program flow which alters the public input $P$ after some iterations of the loop and thus, also alters the intermediate values, cannot be uniquely categorized in accordance with [33].

### *Algebraic Categorization of Faults against the Miller Algorithm*

For the analysis of faulty computations, the physical realization of the fault attack is not relevant. Moreover, different physical faults or fault injection techniques may lead to the same effect on the algorithm. In our opinion, when talking about the effects a single fault can have on the Miller Algorithm, there are only two distinct categories. A fault can either be modeled as having modified the Miller bound $n$, or it can be modeled as having modified the Miller variable $f$.

**Modification of $n$:** In this category we classify all faults that can be modeled by a modification of the Miller bound $n$ to $n'$, cf. [15], [20], [33]–[35]. This includes the following interesting attacks:

- Modification of $n$ while loading the loop counter.
- Modification of $n$ to $n'$ directly in memory [20].
- Early termination of the Miller loop.
- Skipping of conditional if branches [34].
- Corruption of pointer to the Miller variable.

**Modification of $f$:** This category includes all faults which result in a modification of the Miller variable $f$, cf. [21], [33]. The Miller variable is updated during all iterations of the Miller loop. Thus, it can be modified during any iteration of the loop. Note that the actual fault does not have to alter $f$ directly, but, e.g., the intermediate point $T$, cf. Algorithm 1. However, this will result in a modified computation of $f$. This category includes the following interesting attacks:

- Disturb loading of $P$ or $Q$ during line computations.
- Skip update of point during line computations.
- Corrupt a field element directly in memory [21].
- Sign change fault attack [21].

All attacks from both categories can be realized with our setup from Section IV. We will present one practical example in Section V-B.

## IV. LOW-COST PLATFORM FOR MULTIPLE INSTRUCTION GLITCHES

In this section, we explain the fundamental setup that we used for our second-order fault attack. For this attack, we use instruction skip faults, i.e., transient faults which skip parts of the executed code. We generate these faults by means of clock glitching. In Section IV-A, we introduce our universal low-cost platform that generates clock glitches, and Section IV-B shows how clock glitches can be used to skip instructions.

### A. Low-Cost Glitching Platform

In this section, we detail our general setup for implementing CPU clock glitching. This is the mechanism of altering the code execution by clocking the CPU outside its specification for a short period of time. Our setup is similar to the setup of [36]. It is not specialized to attacks on pairings and can be used in other scenarios. It consists of three main components: the glitcher, the host system, and the target. A block diagram of the setup is shown in Figure 1, and Figure 2 shows a picture of our setup. The glitcher is used to generate the external clock for the target device. It is also used to generate the glitches on the clock signal. The host system is used to configure the glitcher and to acquire the output of the device under attack. The target executes the attacked program. We now describe the three components individually.

**Clock Glitcher:** For the hardware of the glitcher we use the DDK [38]. This is a security-focused low-cost open source development platform which consists of a field programmable gate array (FPGA) and an ARM CPU. The FPGA is used to perform the timing critical parts such as generation of the target's clock signal. The ARM CPU is mainly used to interface the FPGA with the host system. It implements a serial terminal that provides external control of the FPGA and an easy automation of the setup.

The glitcher uses two internal clocks: a low frequency clock at $f_l = 33$ MHz and a high frequency clock at $f_h = 99$ MHz. The FPGA implements a 32-bit timer that manages the timing of different events. The clock source of the timer is $f_l$. The glitcher provides a trigger input `gl_trig` to synchronize it with the target. Internally, this input is basically used to reset the timer. The main functionality of the glitcher is to generate a clock signal `gl_clk` for the target. This output can be switched between $f_l$ and $f_h$. A glitch is defined by three parameters, a timestamp $t$, a duration $d$, and a pattern $p$. When the timer reaches $t$, a glitch is generated by a synchronized

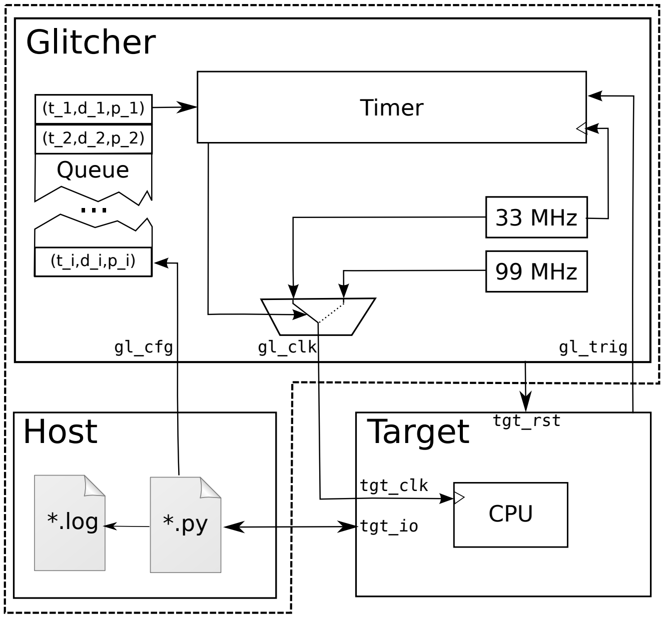

Figure 1. Simplified block diagram of our setup. The host configures the glitcher, which generates the glitches on the external clock of the target device. The target executes the program under attack.

switch from $f_l$ to $f_h$ for $d$ periods of $f_l$, i.e., $3 \cdot d$ periods of $f_h$. We implemented two glitch patterns. For $p = 1$, the high frequency clock $f_h$ is directly used to generate the glitch. For $p = 2$, the clock is gated during the second half of the $f_l$ clock period.

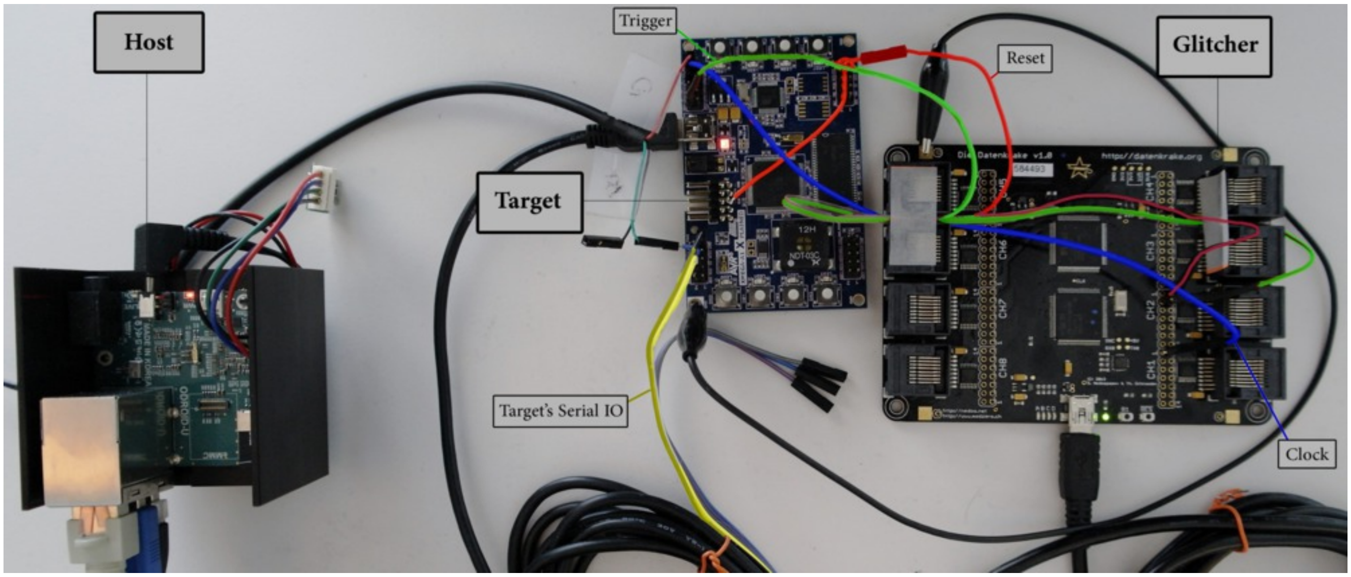

Figure 2. Practical Setup. The DDK (glitcher), located on the right, provides the clock (blue) and reset signal (red) to the XMEGA A1 (target), located in the center. The target also provides back to the DDK the trigger (green) indicating the beginning of the computation. Finally, the ODROID-U2 board [37] (host), which configures and monitors the other devices, can be seen on the left, to which both the target's serial IO (yellow) and the DDK's console (not shown) are connected to.

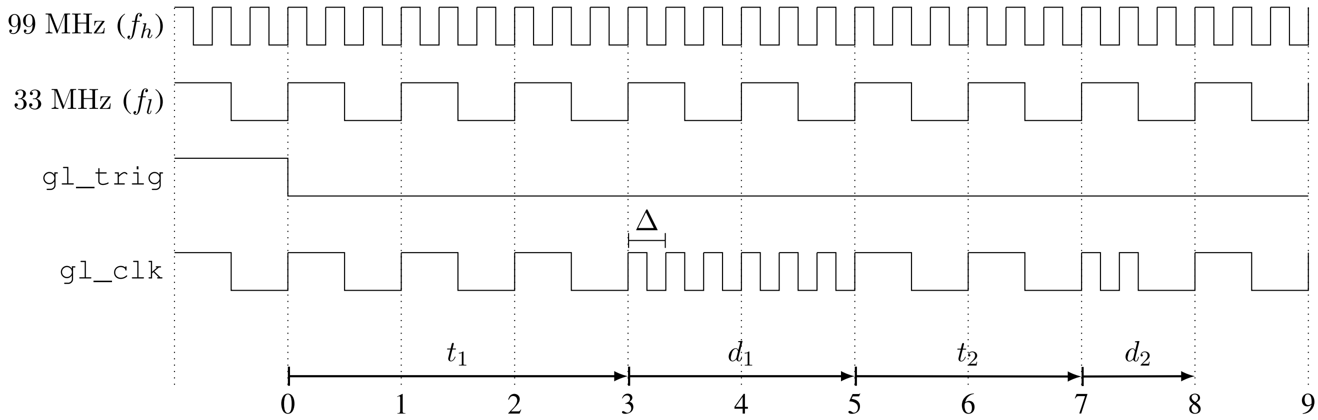

Figure 3. The figure shows the output `gl_clk` of the glitcher with two glitches. The first glitch is introduced with a delay of $t_1 = 3$ cycles of the 33 MHz clock, measured relatively to the trigger `gl_trig`. Its duration is $d_1 = 2$. With $p_1 = 1$, the 99 MHz clock is directly used to generate the glitch pattern. The second glitch is introduced with a delay of $t_2 = 2$ cycles of the 33 MHz clock, measured relatively to the first glitch. Its duration is $d_2 = 1$. With $p_2 = 2$, the 99 MHz clock is gated in the second half of the 33 MHz clock cycle. During a glitch, the delay between two consecutive positive clock edges is $\Delta = 1/f_h$.

It is crucial for our second-order attack to perform two synchronized glitches. Therefore, the glitcher implements a glitch queue. This queue can be filled with up to 256 triples $(t_1, d_1, p_1), \ldots, (t_{256}, d_{256}, p_{256})$. Then, for every element in the queue, the corresponding glitch is generated. For second-order attacks only two glitches need to be scheduled in the glitch queue. For more details on how the glitcher works, see [39]. To fill the queue, the glitcher's internal ARM CPU listens to the serial input at `gl_cfg`.

Figure 3 shows two glitches. The first glitch is introduced with a delay of $t_1 = 3$ cycles of the 33 MHz clock, measured relatively to the trigger `gl_trig`. Its duration is $d_1 = 2$. With $p_1 = 1$, the 99 MHz clock is directly used to generate the glitch pattern. The second glitch is introduced with a delay of $t_2 = 2$ cycles of the 33 MHz clock, measured relatively to the first glitch. Its duration is $d_2 = 1$. With $p_2 = 2$, the 99 MHz clock is gated in the second half of the 33 MHz clock cycle. During a glitch, the delay between two consecutive positive clock edges is $\Delta = 1/f_h$.

**Host System:** The host is a Linux-based system that configures the glitcher and automates the setup. It provides two serial IO lines. One is used to configure the glitcher while the other is used to receive the output from the target. The host system includes a Python [40] library to interface with the glitcher. For example, this allows an in-place analysis and logging of the target's output, followed by a direct reconfiguration for the next attacks. Another functionality provided by the host is to periodically execute a self-test routine for testing the integrity of the setup.

**Target:** For an automated reset of the target the glitcher controls the target's reset pin. Furthermore, the CPU of the target device is clocked with an external clock `tgt_clk`. We control the CPU clock by connecting `tgt_clk` to the glitcher output `gl_clk`.

For the concrete attack of this paper, we assume that the target generates a trigger on output `tgt_trig` before the computation of the target program is started. This signal is used to synchronize the target with the glitcher via `gl_trig`. Generating the trigger on the target is used to simplify the setup. In a real attack, it has to be generated by other means. For example, it could be derived from sniffing the targets IO to locate the command that initiates the attacked computation.

Finally, the IO of the target is connected to the host for initiating the attacked computation and for analysis of the computation's results.

### B. Instruction Skips

Clock glitches can generate instruction skip faults, instruction replacement faults, and data corruption faults on an AVR CPU [36], which is our target in the concrete attack described in Section V. Introducing faults by clock glitching is done by systematically overclocking the target device at defined instructions. Figure 3 shows a waveform of the target CPU clock. A glitch is introduced at $t_1 = 3$. If the difference $\Delta$ of two consecutive edges is outside of the functional range of the CPU circuit, there is a fair chance that the CPU computation gets disturbed. For example, the opcode of the current instruction may be altered to a non-existing opcode. An AVR CPU ignores invalid opcodes during program execution [36]. This results in *instruction skips*. Instruction skips by faults can be used to provoke very different effects on the execution of a concrete algorithm. In Section V and Section VI we will show how instruction skips can be used to attack a concrete pairing implementation.

## V. SECOND-ORDER FAULT ATTACK AGAINST THE ETA PAIRING

This section describes our concrete second-order fault attack against an open source pairing implementation. To conduct the attack, we use the setup that we explained in the previous section. This setup generates the required clock glitches which induce instruction skips.

In Section V-A, we first give an overview how we perform second-order attacks with the setup. We will split the attack into two phases, a profiling phase and a target phase. The profiling phase is only required once. We use it to learn relevant characteristics of the target implementation. Then, in the target phase, the attack can be performed against similar victim devices that store different secrets.

In Section V-B, we introduce our target device and explain our concrete attack on the pairing. Furthermore, we will explain how we were able to break the target implementation in a few minutes for most of the cases.

### A. Realization of Second-Order Fault Attacks against a Pairing Computation

We use the setup described in Section IV-A. This setup allows us to apply the instruction skip mechanism and log the output of the computation. We place the first fault during the execution of Algorithm 1 such that the cryptanalysis of the Miller inversion will be facilitated. The second fault will be introduced at line 10 to skip the procedure call to the final exponentiation.

To configure the glitcher from Section IV-A, the timing $t$, the duration $d$, and the pattern $p$ of the glitches are required. The timing depends on the secret argument of the pairing. Hence, the timing is a priori unknown to us, which makes it challenging to determine $t_1$ and $t_2$. Thus, we execute a profiling phase to find reasonable configurations $(t_1, d_1, p_1)$ and $(t_2, d_2, p_2)$ for the two glitches. We emphasize that once the profiling is completed, we do not need to repeat it when we attack new secrets on similar devices. Without loss of generality, we now assume that the second argument $Q \in \mathbb{G}_2$ is the secret point.

*1) Profiling Phase:* The profiling relies on two assumptions:

- The assembly code of the target pairing implementation is known to us.
- We are able to execute arbitrary *profiling code* on a *profiling device* similar to the target device.

Based on these assumptions, we first execute a modified pairing implementation on the profiling device. We modify the implementation in one or more of the following ways:

- We are able to compute the pairing for different values of $Q$ that are chosen by us.
- We implement triggers $T_1$ and $T_2$ on two external IO pins. Here, $T_1$ is raised immediately before the first target instruction and $T_2$ is raised immediately before the second target instruction.
- We implement an *emulation* mode that branches over the first target instruction from the assembly. This emulates successfully skipping the first target instruction.

These modifications allow us to determine $t_1$ and $t_2$, the timings of the two target instructions, in every computation of the modified pairing. Note that $t_2$ is measured relatively to $t_1$. To measure $t_2$ we use the emulation mode because we are interested in the delay for the case were the first fault has been successful. We execute the modified implementation for different secrets $Q$ chosen uniformly at random from $\mathbb{G}_2$. As a result, we obtain distributions for $t_1$ and $t_2$. Since these distributions are obtained over the random choices of $Q$, we will choose the parameter triples in the target phase according to them.

These steps of the profiling can be done either by an oscilloscope or by programming a special profiling mode into the FPGA of the glitcher. The profiling mode counts the number of clock cycles between `tgt_trig` and $T_1$, and between $T_1$ and $T_2$.

In the next step of the profiling, we determine useful combinations of the remaining glitching parameters $d_1$, $d_2$, $p_1$, and $p_2$. We do this by performing a large number of experiments where we use the glitcher to introduce glitches at $T_1$ and $T_2$ that are close to the target instructions. We use the fact that we know the values of $Q$ in the profiling phase. Hence, we can predict the output of the algorithm when successfully glitching either one or both of the target instructions. This allows us to identify successful tests and their respective parameters.

*2) Target Phase:* In the subsequent target phase, the actual target device with the unmodified code and the unknown secret is attacked. Therefore, we perform a sequence of experiments with different combinations of $(t_1, d_1, p_1)$ and $(t_2, d_2, p_2)$ until we are successful in skipping the two target instructions. We select the combinations and their order based on the results of the profiling phase.

### *B. Realization of our Concrete Second-Order Fault Attack against the eta Pairing*

For the concrete pairing implementation we used the RELIC toolkit [23]. It includes `C` implementations of finite field arithmetic, ECC, and PBC for different hardware platforms like Atmel's AVR family. The RELIC toolkit has also been used in TinyPBC for the implementation of PBC in wireless sensor networks [13]. To the best of our knowledge it is the only freely available implementation of PBC for AVR CPUs. In our concrete second-order fault attack, we targeted an AVR XMEGA A1 [41]. AVR controllers are also used in modern smart cards, while our version is freely programmable. A microcontroller from the AVR family was also analyzed in [36]. For our attack, we use RELIC version 0.3.5 without modifications of the source code. We compile the library with the `avr-gcc-4.8.2` toolchain and optimization level `-O1`[1]. The RELIC AVR default configuration defines the eta pairing [24] (function `pb_map_etats()`) as the standard pairing.

In our experiments both arguments $P$ and $Q$ are loaded from the internal memory. Loading the public argument from memory and not via the serial line helps to simplify the setup, but is not essential for the attack. Then $e(P, Q)$ is computed on the target device and the output is returned on the serial IO `tgt_io`.

We placed the first fault at line 9 of Algorithm 1 such that the `for` loop is terminated after the first iteration. The second fault was introduced at line 10 of Algorithm 1 to skip the procedure call to the final exponentiation. A successful attack gave us a faulty computation where the `for` loop was executed exactly once and the final exponentiation was not executed at all. In Section VI, we will show how this attack can be analyzed to obtain the secret argument of the pairing. To understand how we attack the end of the `for` loop, we refer to Table I. It shows how the compiler generates the end of the `for` loop. An instruction skip fault that removes the `rjmp` instruction in line 11 causes the loop to terminate immediately. Hence, it realizes our attack that modifies the Miller bound $n$.

*1) Profiling Phase:* In the first step, we estimated $t_1$, the clock cycle of the `rjmp` instruction. Therefore, we executed approximately 32,000 experiments with random choices of $Q$ and measured $t_1$ for each experiment. The distribution of $t_1$ is given in Table II. Then we determined $t_2$, the number of clock cycles from the `rjmp` to the `call` of the final exponentiation. We used the emulation mode of the profiling code. It allowed us to skip the `rjmp` instruction at $t_1$. We obtained a constant value of $t_2 = 28$. Here, $t_2$ is constant because if the first glitch was successful in leaving the `for` loop, the code executed between $t_1$ and $t_2$ is independent of the secret.

To select combinations of $d_1$, $d_2$, $p_1$ and $p_2$ for the target phase we injected approximately $40{,}000$ faults in less than 72 hours. Since we knew $Q$ during profiling, and hence also the values of $t_1$ and $t_2$, we were always able to

---

[1] If the RELIC library is compiled with optimization level `-O2`, the compiler replaces the function call to the final exponentiation by inline code. We currently work on an attack for `-O2`. Here, we will facilitate the exponentiation inversion by a fault during the computation of the final exponentiation and by an improved mathematical analysis.

Table I
ASSEMBLY OF END OF `FOR` LOOP GENERATED WITH `AVR-GCC`.

```
 call fb4_mul_dxs
.LVL43:
/*decrement loop counter LSB, MSB */
 subi r16,1
 sbc r17,__zero_reg__
 .loc 1 247 0 discriminator 2
 breq .+2
 /*jump to loop begin */
 rjmp .L2
.LBE2:
 .loc 1 486 0
/* clean stack*/
 subi r28,36
 sbci r29,-2
 out __SP_L__,r28
 out __SP_H__,r29
 pop r29
```

Table II
DISTRIBUTION OF THE EXECUTION TIME $t_1$ OF THE `RJMP` INSTRUCTION IN TABLE I, DEPENDING ON THE INPUT $Q$ OF ALGORITHM 1.

| $t_1$ in instruction cycles | occurrence | in % |
|---|---|---|
| 422,780 | 1 | $< 0.01$ |
| 424,515 | 1 | $< 0.01$ |
| 424,941 | 1 | $< 0.01$ |
| 427,731 | 1 | $< 0.01$ |
| 431,069 | 1 | $< 0.01$ |
| 581,804 | 3 | 0.01 |
| 581,903 | 28 | 0.08 |
| 582,001 | 7 | 0.02 |
| 582,002 | 590 | 1.66 |
| 582,100 | 30 | 0.08 |
| 582,101 | 1,763 | 4.95 |
| 582,111 | 1 | $< 0.01$ |
| 582,199 | 297 | 0.83 |
| 582,200 | 32,890 | 92.35 |

introduce the faults at the correct instructions. Regarding the two patterns $p_1$ and $p_2$ depicted in Figure 3, both produced good results. To be safe, we propose to use both in the target phase. For the duration of the glitches, we found $d_1 = 3$ or $d_1 = 5$ and $d_2 \leq 5$ as reasonable settings to use in the target phase.

*2) Target Phase:* Based on our results from the profiling shown in Table II we scheduled $t_1$ as $582{,}200 - i \cdot 99$ for $i \in \{0, \ldots, 5\}$.[2] If we did not succeed with one of these values, we fell back to a brute force search with $t_1 = 582{,}200 - i$ for $i = 1,2,3,\ldots$ until we were successful. We combined each value of $t_1$ with each combination of $d_1$, $d_2$, $p_1$, and $p_2$ that we determined in the profiling phase. For $t_2$ we added a small safety margin such that $t_2 \in \{26, \ldots, 30\}$. Furthermore, we repeated each combination for 10 times because even with the correct parameters, glitching is not always successful. Hence, for each value of $t_1$ we performed $2 \cdot 5 \cdot 2 \cdot 2 \cdot (30 - 25) \cdot 10 = 2{,}000$ experiments. For our setup, one test requires 7.5 seconds on average. This includes configuration of the glitcher, communication from target to host, and self-tests. Hence, we are able to perform more than 10,000 experiments

---

[2] We blame the occurrences at 582,199 as inaccurate and account them for the delay 582,200.

per day.

We will show in Section VI-B that we are able to efficiently determine from the target's output whether both instruction skips were successful or not. Furthermore, we will show that for a successful attack, we are able to efficiently compute the secret $Q$. Hence, once we detected the first successful experiment, we discarded all remaining experiments to start the next attack.

We repeated the attack for five different secrets, drawn uniformly at random from $\mathbb{G}_2$. We were always successful in skipping both instructions. The analysis of the experiments showed that for all secrets it occurred that $t_1$ was either 582,200 or 582,101. This is in line with the distribution in Table II. Hence, for each attack we required at most $2 \cdot 2{,}000$ experiments, whereas in the cases with $t_1 = 582{,}200$, much fewer experiments were required and it took us only minutes to be successful.

## VI. ANALYSIS OF FAULTY COMPUTATIONS

We now resume the description of the second-order fault attack by explaining the mathematical analysis which leads from the faulty computation to the secret key. We will first provide mathematical details of the attacked implementation and then give two examples, one for each category from Section III. With our setup from Section IV, we can realize any fault from both categories, i.e., all theoretical faults that have been presented so far. However, we concentrate on two examples to illustrate both categories.

The first example is the concrete attack from Section V. It illustrates the modification of the Miller loop bound $n$. The second example illustrates the modification of the Miller variable $f$. Both these analyses have already been described similarly, cf. [15], [20], [21], [33]. In both examples, we assume to know $P = (x_P, y_P)$, while $Q = (x_Q, y_Q)$ is secret. We induce the first fault during the computation of the Miller Algorithm and use the second fault to skip the function call to the final exponentiation. Thus, we do not have to solve the exponentiation inversion, but only a facilitated Miller inversion.

### *A. Mathematical Details of the Attacked Implementation*

We attacked an implementation of the eta pairing in characteristic 2 on supersingular elliptic curves. We decided to attack the eta pairing [24] despite current research results which indicate that it should no longer be used for security applications, cf. [25], [26]. This was due to the fact that the eta pairing is the default for AVR devices in the attacked RELIC library [23]. However, the attack can be easily applied to other pairings. The concrete implementation is very similar to the implementation proposed in [24, Section 6] and is presented in Algorithm 2.

The elliptic curve $E : y^2 + y = x^3 + x$ is defined over the finite field $\mathbb{F}_q$ with $q = 2^m$ and $m = 271$ in our implementation. For our case, i.e., $m = 7 \mod 8$, it holds that $\#E(\mathbb{F}_q) = 2^m + 2^{(m+1)/2} + 1$. We define the extension field $\mathbb{F}_{q^4} = \mathbb{F}_q(s, t)$ of degree 4, with $s^2 = s + 1$ and $t^2 = t + s$. Let $z = (q^4 - 1)/\#E(\mathbb{F}_q) = (2^{2m} - 1) \cdot (2^m - 2^{(m+1)/2} + 1)$, $n = 2^{(m+1)/2} + 1$, and $\psi(x,y) = (x + 1 + 1, y + sx + t)$. For input $P, Q \in E(\mathbb{F}_q)$ the eta pairing $\eta$ is then defined as

$$\eta(P, Q) = f_{n,-P}(\psi(Q))^z.$$

Because of the simple binary form of $n$, the main loop of Algorithm 1 mainly reduces to point doubling and squaring of field elements in $\mathbb{F}_{q^4}$, followed by one multiplication with $l_{[2^{(m+1)/2}](-P),-P}(\psi(Q))$ for the least significant bit of $n$. As in [24, Algorithm 3], the eta implementation computes the loop in reversed order in the RELIC library [23]. Therefore, $P' = \left[2^{(m-1)/2)}\right](-P)$ needs to be defined. Furthermore, the first loop is unrolled:

$$\begin{aligned} f_{n,-P}(\psi(Q)) = l_{[2]P',-P}(\psi(Q)) \cdot g_{P'}(\psi(Q)) \cdot \\ \prod_{j=1}^{(m-1)/2} g_{[2^{-j}]P'}(\psi(Q))^{2^j}. \end{aligned} \tag{1}$$

Algorithm 2 shows how the computation of (1) is implemented in the RELIC library.

**Algorithm 2** Implementation of $\eta(P,Q)$ on $E(\mathbb{F}_{2^m})$ for $m = 7 \mod 8$ and $E : y^2 + y = x^3 + x$.

```
Require: $P = (x_P, y_P), Q = (x_Q, y_Q) \in E$
Ensure: $\eta(P,Q)$
 1: $u \leftarrow x_P, v \leftarrow x_Q$
 2: $g \leftarrow u \cdot v + y_P + y_Q + 1 + (u + x_Q)s + t$
 3: $u \leftarrow x_P^2$
 4: $l \leftarrow g + v + u + s$
 5: $f \leftarrow g \cdot l$
 6: for $i = 1 \,..\, (m-1)/2$ do
 7:     $x_Q \leftarrow {x_Q}^2, y_Q \leftarrow {y_Q}^2$
 8:     $x_P \leftarrow \sqrt{x_P}, y_P \leftarrow \sqrt{y_P}$
 9:     $u \leftarrow x_P, v \leftarrow x_Q$
10:     $g \leftarrow u \cdot v + y_P + y_Q + 1 + (u + x_Q)s + t$
11:     $f \leftarrow f \cdot g$
12: end for
13: $f \leftarrow f^z$
14: return $f$
```

### B. *Example: Analysis after Modification of $n$*

Now, we analyze the output of our second-order attack from Section V. For the concrete RELIC implementation, the two instruction skip faults target the first execution of line 12 and the execution of line 13 of Algorithm 2. Hence, Table I shows the generated assembly for line 12 of Algorithm 2.

In an execution were both fault injections are successful, the `for` loop is executed exactly once and the final exponentiation is completely skipped. Since one loop is unrolled, this corresponds to an execution with two iterations of the loop in Algorithm 1, and a modification of $n$ from $2^{(m+1)/2} + 1$ to $2^2 + 1$. We see that our attack is in the category of faults that modify $n$. Let $\alpha$ be the output of the faulty computation $f'_{n,P}(\psi(Q))$. With (1) we obtain

$$\alpha = f'_{n,-P}(\psi(Q)) = l_{[2]P',-P}(\psi(Q)) \cdot g_{P'}(\psi(Q)) \cdot g_{[2^{-1}]P'}(\psi(Q))^2. \tag{2}$$

The following steps describe how we recover the secret input $Q$ of Algorithm 2 from $\alpha$.

1) **Algebraic Model of the Secret:** First, we define variables $x$ and $y$ representing the $x$-coordinate and the $y$-coordinate of the secret $Q$. Now we describe $Q$ as the root of a rational function. With (2) we define
$$f_P(x,y) := f'_{n,-P}(\psi(x,y)) - \alpha. \tag{3}$$
Since $f'_{n,-P}(\psi(x,y))$ is a product of four lines, $f_P(x,y)$ is of degree at most 4 in $x$ and $y$. In our case the secret is already defined over the strict subfield $\mathbb{F}_q$ of $\mathbb{F}_{q^4}$. We model this by considering $\mathbb{F}_{q^4}$ as an $k = 4$ dimensional vector space over $\mathbb{F}_q$. Then (3) can be re-written as four individual polynomials $f_P^{(1)}, \ldots, f_P^{(4)}$ over $\mathbb{F}_q$. This will reduce the computational complexity of the analysis in the next step.
2) **Computation of Candidates:** At this point, we define the variety $V_Q = V\left(f_P^{(1)}, \ldots, f_P^{(4)}\right) \cap E$ by a (possibly overdetermined) system of nonlinear multivariate equations. Since $Q \in V_Q$, we now compute all elements of $V_Q$ in this step. The complexity of this step mainly depends on the degrees of $f_P^{(1)}, \ldots, f_P^{(4)}$ and is reduced by using more equations than variables.
3) **Testing Candidates:** In the final step, we identify the secret from all elements in $V_Q$. To do this, we compute $\eta(P,Q')$ for the elements $Q' \in V_Q$. Each result is compared with $\eta(P,Q)$ that has been obtained from an error-free execution to identify the unique point $Q$.

Note that the case where $P$ is the secret can be handled analogously. The major difference is that we replace $f_P(x,y)$ from step 1 by a polynomial where $x$ and $y$ represent $\sqrt{x_P}$ and $\sqrt{y_P}$. From Algorithm 2 we see that the degree of $f_P(x,y)$ will now become at most $d = 7$. Due to the higher degree, the analysis will become more expensive.

Note that restricting to subfields as in step 1 can often be exploited. For example, it has been used in [21] and [32]. Indeed, the most common optimization for the implementation of pairings is to choose the first argument $P$ in $\mathbb{G}_1 \subseteq E(\mathbb{F}_q)$. Furthermore, for Type 1 pairings the second argument $Q$ is also $\mathbb{F}_q$-rational. For Type 3 pairings, $Q$ is defined in $\mathbb{G}_2 \subseteq E'(\mathbb{F}_{q^{k'}})$ where $E'$ is a degree $k'$ twist of $E$ and $k'$ divides $k$. For details on the selection of pairing-friendly curves we refer to [42].

As explained in Section V, many experiments fail in delivering the intended faults, i.e., in simultaneously skipping both target instructions. For a failed experiment, no candidate $Q'$ will pass step 3. Hence, in practice it is crucial to automate step 2 and step 3 for identifying the first successful experiment. We automated the analysis based on Sage [43], a free computer algebra system. Therefore, we re-implemented the eta pairing from the RELIC library in Sage. This implementation allows us to compute the pairing on arbitrary inputs $P$, $Q$, and $n$. Based on this implementation, we are able to automatically construct the multivariate polynomial (3) from step 1 for any value $\alpha$. Step 2 is an invocation of the `variety()` function on the ideal generated by $f_P^{(1)}, \ldots, f_P^{(4)}$ and $y^2 + y = x^3 + x$. This computation is based on Gröbner basis techniques. Hence, using five equations for only two variables accelerates this step. Finally, in step 3 we use the implementation of the pairing again, but evaluate it at the candidate points $Q'$ to identify $Q$.

Our non-optimized implementation requires less than one second for processing one faulty output $\alpha$. This is less time than the target device requires to compute the pairing. Hence, the mathematical computation is not critical for the performance of our attack.

### *C. Example: Analysis after Modification of $f$.*

For this example, we attack two computations of $\eta(P,Q)$. In both computations, the same input has to be used. During the first computation, we only use one fault and skip the final exponentiation. We denote the output with $\alpha_1$, i.e., $\alpha_1 = f_{n,-P}(\psi(Q))$. In the second computation, we also skip the final exponentiation. Prior to this fault, we induce another fault to skip an instruction which is involved in the update of the Miller variable $f$. In the general description of the Miller Algorithm, this corresponds to the lines 3, 4, 6 or 7 of Algorithm 1. In our concrete implementation, cf. Algorithm 2, also several instructions can be skipped to achieve a modification of $f$. For this example, we choose to illustrate the modification of $f$ by skipping the update of $u$ once. Thus, either line 3 or line 9 in any round of the `for` loop in Algorithm 2 can be skipped. We choose to skip line 3. We denote the second faulty output with $\alpha_2$, i.e., $\alpha_2 = f'_{n,-P}(\psi(Q))$. Since $\alpha_1$ and $\alpha_2$ are known, we also know $\alpha = \alpha_1/\alpha_2 \in \mathbb{F}_{q^4}$.

1) **Algebraic Model of the Secret:** The two values $\alpha_1$ and $\alpha_2$ have the same first factor $g$, which is computed in line 2, but differ in their factor $l$, which depends on $u$. Since $u$ depends on $x_P$ afterwards, which is not attacked itself in this scenario, all further factors of $\alpha_1$ and $\alpha_2$ which are computed during the `for` loop are equal. Thus, since all but the respective factors $l$ of $\alpha_1$ and $\alpha_2$ are equal, we receive the equation

$$\begin{aligned} & x_P \cdot x_Q + y_P + y_Q + 1 \\ & + (x_P + x_Q) \cdot s + t + x_Q + x_P^2 + s \\ = & \alpha \cdot [x_P \cdot x_Q + y_P + y_Q + 1 \\ & + (x_P + x_Q) \cdot s + t + x_Q + x_P + s], \end{aligned} \tag{4}$$

with all values except $x_Q$ and $y_Q$ known.

2) **Computation of Candidates:** The elliptic curve is defined by $E : y^2 + y = x^3 + x$. It gives us a second equation with root $Q$. By writing both $E$ and (4) as univariate polynomials in $y$ and using the theory of

resultants, we get a univariate polynomial in $x$ which has degree at most 3.

$$\begin{aligned}
&\operatorname{Res}(\alpha \cdot f'_{n,-P}(\psi(x,y)) - f_{n,-P}(\psi(x,y)), E) \\
= \; &(\alpha - 1)^2 \cdot (-x^3 - x) \\
&+ \Big[(\alpha - 1)(x_P \cdot x + x_P + x + y_P + 1 \\
&+ (x_P + x + 1) \cdot s + t) - x_P^2 + x_P\Big]^2 \\
&- \Big[(\alpha - 1)(x_P \cdot x + x_P + x + y_P + 1 \\
&+ (x_P + x + 1) \cdot s + t) - x_P^2 + x_P\Big] \cdot (\alpha - 1)
\end{aligned} \tag{5}$$

All roots of this polynomial are candidates for $x_Q$. For each of these candidates we evaluate $E$ and thereby get two candidates for the secret point $Q$.

3) **Testing Candidates:** Since we know the concrete implementation, we now compute $\eta(P, Q')$ for all candidates $Q'$ and compare the results with $\eta(P, Q)$, which has been obtained from an error-free execution. Since Equation 5 has degree at most 3 and $E$ has degree 2 in $y$, we have to test at most six candidates to identify the unique point $Q$.

Note that again, the roles of $x_Q$ and $y_Q$ can be switched. The resulting univariate polynomial in $y$ has at most degree 4, and we will then get three candidates for the secret point for each root. Thus, we have to test at most twelve candidates.

## VII. CONCLUSION

Several fault attacks against pairing-based cryptography have been published in the past. Interestingly, none of these have been practically evaluated. We accomplished this task and proved that fault attacks against pairing-based cryptography are indeed possible and are even practical — thus posing a serious threat. Moreover, we successfully conducted a practical second-order fault attack against an open source implementation of the eta pairing on an AVR XMEGA A1. We used this freely programmable chip to validate our attacks on a real-world smart card platform. On the basis of a new two-part categorization of all conceivable fault attacks against the underlying Miller Algorithm, we were able to reveal the secret point of a pairing in both categories.

For many pairing-based protocols, the output of the pairing can not directly be accessed by the attacker [44]. Hence, another direction of further research is how attacks on pairings can be applied to these protocols. However, as the first practical realization of fault against pairing-based cryptography, our results prove the requirement for further strong and efficient countermeasures. While generic countermeasures like checksums and redundant computations might also prevent fault attacks, they might be too expensive or not effective against all types of faults in the pairing-based context, as this turns out to be more complex than traditional cryptography. Our successful attacks highlight the demand for further research on how to protect against the complete skipping of the final exponentiation. Besides that, particularly the first and the last rounds of the Miller Algorithm have to be secured against fault attacks. Given that even RSA in CRT mode is still struggling with the Bellcore attack — after almost 20 years of intensive research — it is natural that the young field of pairing-based cryptography requires more research after our successful attack.

## ACKNOWLEDGMENT

This work was partially supported by the German Research Foundation (DFG) within the Collaborative Research Centre On-The-Fly Computing (SFB 901). J. K. gratefully acknowledges support by the Deutsche Telekom Stiftung. We thank Universidade Federal do Rio Grande do Sul and Technische Universität Berlin for allowing R.G.d.S. to do a dual-degree program. We also thank our colleague Dmitry Nedospasov who supported this work with productive ideas for the glitcher development.

# REFERENCES

[1] D. Boneh, R. A. DeMillo, and R. J. Lipton, "On the Importance of Checking Cryptographic Protocols for Faults," in *EUROCRYPT'97*, ser. Lecture Notes in Computer Science, vol. 1233. Springer Berlin Heidelberg, 1997, pp. 37–51.

[2] I. Verbauwhede, D. Karaklajic, and J.-M. Schmidt, "The Fault Attack Jungle - A Classification Model to Guide You," in *FDTC*, 2011, pp. 3–8.

[3] H. Bar-El, H. Choukri, D. Naccache, M. Tunstall, and C. Whelan, "The Sorcerer's Apprentice Guide to Fault Attacks," *Proceedings of the IEEE*, vol. 94, no. 2, pp. 370–382, Feb 2006.

[4] C. Kim and J.-J. Quisquater, "Fault Attacks for CRT Based RSA: New Attacks, New Results, and New Countermeasures," in *Information Security Theory and Practices. Smart Cards, Mobile and Ubiquitous Computing Systems*, ser. Lecture Notes in Computer Science, 2007, vol. 4462, pp. 215–228.

[5] E. Dottax, C. Giraud, M. Rivain, and Y. Sierra, "On Second-Order Fault Analysis Resistance for CRT-RSA Implementations." in *WISTP*, ser. Lecture Notes in Computer Science, vol. 5746. Springer Berlin Heidelberg, 2009, pp. 68–83.

[6] A. Menezes, S. Vanstone, and T. Okamoto, "Reducing Elliptic Curve Logarithms to Logarithms in a Finite Field," in *Proceedings of the Twenty-third Annual ACM Symposium on Theory of Computing*, ser. STOC '91, 1991, pp. 80–89.

[7] D. Boneh and M. Franklin, "Identity-Based Encryption from the Weil Pairing," *SIAM J. of Computing*, vol. 32, no. 3, pp. 586–615, 2003, extended abstract in Crypto'01.

[8] A. Shamir, "Identity-based Cryptosystems and Signature Schemes," in *Advances in Cryptology - CRYPTO 84*, ser. Lecture Notes in Computer Science, vol. 196. Springer-Verlag New York, Inc., 1985, pp. 47–53.

[9] D. F. Aranha, P. S. L. M. Barreto, P. Longa, and J. E. Ricardini, "The realm of the pairings," in *Selected Areas in Cryptography*, ser. Lecture Notes in Computer Science, vol. 8282. Springer Berlin Heidelberg, 2013, pp. 3–25.

[10] A. Sahai and B. Waters, "Fuzzy Identity-based Encryption," in *EUROCRYPT 2005*, ser. Lecture Notes in Computer Science, vol. 3494. Springer Berlin Heidelberg, 2005, pp. 457–473.

[11] F. Hess, "Efficient Identity Based Signature Schemes Based on Pairings," in *SAC 2002, LNCS 2595*. Springer Berlin Heidelberg, 2002, pp. 310–324.

[12] A. Joux, "A One Round Protocol for Tripartite Diffie-Hellman," in *Proceedings of the 4th International Symposium on Algorithmic Number Theory*, ser. ANTS-IV. Springer Berlin Heidelberg, 2000, pp. 385–394.

[13] L. B. Oliveira, D. F. Aranha, C. P. Gouvêa, M. Scott, D. F. Câmara, J. López, and R. Dahab, "TinyPBC: Pairings for Authenticated Identity-Based non-interactive Key Distribution in Sensor Networks," *Computer Communications*, vol. 34, no. 3, pp. 485 – 493, 2011, special Issue of Computer Communications on Information and Future Communication Security.

[14] L. B. Oliveira, D. F. Aranha, E. Morais, F. Daguano, J. López, and R. Dahab, "TinyTate: Identity-Based Encryption for Sensor Networks," *IACR Cryptology ePrint Archive*, 2007.

[15] D. Page and F. Vercauteren, "A Fault Attack on Pairing-Based Cryptography," *IEEE Transactions on Computers*, vol. 55, no. 9, pp. 1075–1080, 2006.

[16] S. D. Galbraith, F. Hess, and F. Vercauteren, "Aspects of Pairing Inversion," *IEEE Transactions on Information Theory*, vol. 54, no. 12, pp. 5719–5728, 2008.

[17] N. Kanayama and E. Okamoto, “Approach to Pairing Inversions Without Solving Miller Inversion,” *IEEE Transactions on Information Theory*, vol. 58, no. 2, pp. 1248–1253, 2012.

[18] I. Biehl, B. Meyer, and V. Müller, “Differential Fault Attacks on Elliptic Curve Cryptosystems,” in *Advances in Cryptology — CRYPTO 2000*, ser. Lecture Notes in Computer Science, vol. 1880, 2000, pp. 131–146.

[19] S. D. Galbraith, *Mathematics of Public Key Cryptography*. Cambridge University Press, 2012.

[20] N. E. Mrabet, “What about Vulnerability to a Fault Attack of the Miller’s Algorithm During an Identity Based Protocol?” in *Proceedings of the 3rd International Conference and Workshops on Advances in Information Security and Assurance*, ser. ISA ’09. Springer Berlin Heidelberg, 2009, pp. 122–134.

[21] C. Whelan and M. Scott, “The Importance of the Final Exponentiation in Pairings When Considering Fault Attacks,” in *Pairing*, ser. Lecture Notes in Computer Science, vol. 4575. Springer Berlin Heidelberg, 2007, pp. 225–246.

[22] R. Lashermes, J. Fournier, and L. Goubin, “Inverting the Final Exponentiation of Tate Pairings on Ordinary Elliptic Curves Using Faults,” in *CHES*, ser. Lecture Notes in Computer Science, vol. 8086. Springer Berlin Heidelberg, 2013, pp. 365–382.

[23] D. F. Aranha and C. P. L. Gouvêa, “RELIC is an Efficient LIbrary for Cryptography,” http://code.google.com/p/relic-toolkit/.

[24] P. S. L. M. Barreto, S. D. Galbraith, C. O’Eigeartaigh, and M. Scott, “Efficient pairing computation on supersingular abelian varieties,” *Des. Codes Cryptography*, vol. 42, no. 3, pp. 239–271, 2007.

[25] R. Granger, T. Kleinjung, and J. Zumbragel. (2014) Discrete Logarithms in the Jacobian of a Genus 2 Supersingular Curve over $GF(2^367)$. Accessed: 2014-01-30. [Online]. Available: https://listserv.nodak.edu/cgi-bin/wa.exe?A2=NMBRTHRY;23651c2.1401

[26] G. Adj, A. Menezes, T. Oliveira, and F. Rodríguez-Henríquez, “Computing Discrete Logarithms in $F_{3^{6*137}}$ and $F_{3^{6*163}}$ using magma,” *IACR Cryptology ePrint Archive*, 2014.

[27] R. Barbulescu, P. Gaudry, A. Joux, and E. Thomé, “A heuristic quasi-polynomial algorithm for discrete logarithm in finite fields of small characteristic,” in *EUROCRYPT 2014*, ser. Lecture Notes in Computer Science, vol. 8441. Springer Berlin Heidelberg, 2014, pp. 1–16.

[28] V. S. Miller, “The weil pairing, and its efficient calculation,” *J. Cryptology*, vol. 17, no. 4, pp. 235–261, 2004.

[29] M. Joye and G. Neven, Eds., *Identity-based cryptography*, ser. Cryptology and Information Security Series. IOS Press, 2008.

[30] S. Chang, H. Hong, E. Lee, and H.-S. Lee, “Reducing Pairing Inversion to Exponentiation Inversion using Non-degenerate Auxiliary Pairing,” *IACR Cryptology ePrint Archive*, 2013.

[31] I. F. Blake, G. Seroussi, and N. P. Smart, Eds., *Advances in Elliptic Curve Cryptography*, ser. London Mathematical Society Lecture Note Series. Cambridge University Press, 2005, no. 317.

[32] F. Vercauteren, “The Hidden Root Problem,” in *Proceedings of the 2nd international conference on Pairing-Based Cryptography*, ser. Pairing ’08. Springer Berlin Heidelberg, 2008, pp. 89–99.

[33] N. E. Mrabet, D. Page, and F. Vercauteren, “Fault Attacks on Pairing-Based Cryptography,” in *Fault Analysis in Cryptography*. Springer Berlin Heidelberg, 2012.

[34] K. Bae, S. Moon, and J. Ha, “Instruction Fault Attack on the Miller Algorithm in a Pairing-Based Cryptosystem,” in *IMIS*. IEEE, 2013, pp. 167–174.

[35] D. Page and F. Vercauteren, “Fault and Side-Channel Attacks on Pairing Based Cryptography,” *IACR Cryptology ePrint Archive*, 2004.

[36] 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 *FDTC*, 2011, pp. 105–114.

[37] ODROID-U2 Product Page. Accessed: 2014-05-29. [Online]. Available: http://hardkernel.com/main/products/prdt_info.php?g_code=G135341370451

[38] D. Nedospasov and T. Schroder, “Introducing Die Datenkrake: Programmable Logic for Hardware Security Analysis,” in *7th USENIX Workshop on Offensive Technologies*, 2013.

[39] R. G. da Silva, “Practical Analysis of Embedded Microcontrollers against Clock Glitching Attacks,” Bachelor’s Thesis, Technische Universität Berlin, 2014.

[40] Python Programming Language Homepage. Accessed: 2014-05-15. [Online]. Available: http://www.python.org

[41] ATxmega128A1 Product Page. Accessed: 2014-05-21. [Online]. Available: http://www.atmel.com/devices/atxmega128a1.aspx

[42] P. S. L. M. Barreto, B. Lynn, and M. Scott, “On the Selection of Pairing-Friendly Groups,” in *Selected Areas in Cryptography*, ser. Lecture Notes in Computer Science, vol. 3006. Springer Berlin Heidelberg, 2003, pp. 17–25.

[43] W. A. Stein. (2014) Sage Mathematics Software (Version 6.1). [Online]. Available: http://www.sagemath.org

[44] S. Chatterjee, K. Karabina, and A. Menezes, “Fault attacks on pairing-based protocols revisited,” *IACR Cryptology ePrint Archive*, 2014.