Characterising a CPU Fault Attack Model via Run-Time Data Analysis

Martin S. Kelly
Information Security Group
Smart Card Centre
Royal Holloway, University of London
Egham, TW20 0EX
United Kingdom.
Email: Martin.Kelly.2014@live.rhul.ac.uk

Keith Mayes
Information Security Group
Smart Card Centre
Royal Holloway, University of London
Egham, TW20 0EX
United Kingdom.
Email: Keith.Mayes@rhul.ac.uk

John F. Walker
DNV GL Ltd.
Crescent House
46 Priestgate
Peterborough, PE1 1LF
United Kingdom.
Email: john.walker@dnvgl.com

***Abstract*—Effective software defences against errors created by fault attacks need to anticipate the probable error response of the target micro-controller. The range of errors and their probability of occurrence is referred to as the Fault Model. Software defences are necessarily a compromise between the impact of an error, its likelihood of occurrence, and the cost of the defence in terms of code size and execution time. In this work we first create a fault insertion system and then use it to demonstrate a technique for precisely triggering and capturing individual error responses within a running micro-controller. This enables a more realistic calibration of a micro-controller's fault model. We apply the system to a representative micro-controller and the results show that error insertion is far more predictable than anticipated, and is consistent over a wide range of experimental tolerances. This observation undermines some widely deployed software defences recommended for fault attack protection.**

I. Introduction

Fault injection is a semi-invasive attack technique used to induce behavioural errors in semi-conductor integrated circuits and the risks posed by such errors in both execution and calculation have been known for a considerable time. The classic Bellcore Attack [5] against RSA, and its logical extension [4] to general cryptographic calculations, highlighted the vulnerability of cryptographic algorithms to fault analysis. Likewise fault tolerant computing is highly relevant within safety critical systems and interest here predates cryptographers' concerns [13].

Faults may be injected into an executing micro-controller (μP) through a variety of mechanisms. Perturbations in a chip's power and clock supply [2], Photo-electric effects from lasers or white light [17] and localised intense EM fields [12]. Manufacturers of secured μPs have integrated various hardware defences into their devices such as physical shields and light sensitive detection circuitry [11] & [18] in order to make attacks difficult to perform. Duplicate circuitry can be added to detect errors within individual logic paths [3] and even duplicate processors [8] can be used. These features are very effective, but techniques are evolving to circumvent these physical defences [20]. Furthermore, while these hardware defences are widely utilised in the current generation of smart card μPs they have yet to make their way into devices aimed at the cost-sensitive consumer electronics market and the rapidly growing *Internet of Things* market.

Software defences are therefore an unavoidable requirement for systems that need to resist attack. To complement the hardware defences, or lack thereof, programmers add redundant code to their algorithms to verify critical operations. Double checking of calculations [14] and monitoring of flow control [7] are typical of the techniques employed. The whole spectrum of defensive techniques applicable to μPs has been analysed [19], drawing conclusions on the relative effectiveness and overheads of the various methods.

Appropriate defensive code depends on the accuracy of the fault model, which by necessity makes assumptions about the nature of injected errors. To improve the accuracy of fault models we need to examine real faults from physical silicon while it executes under normal conditions: i.e. not single stepping via JTAG or similar debug hardware. The inherent difficulty in trying to categorise the nature of such faults has been highlighted by [6]. Namely that the test program is equally vulnerable to the induced errors and it is difficult to tell whether the program failed, data was corrupted or reference data was corrupted.

This paper first describes how we overcame the reported difficulties. First by creating a laser fault insertion calibration system and then using it to precisely trigger and capture individual error responses within a running micro-controller. By controlling the fault stimulus, both spatially and temporally, we have analysed the behaviour of individual μP instructions, identified their modes of failure, and gained additional insight into the efficacy of defensive code. The experimental system was used to precisely characterise dynamic fault behaviour, which led to surprisingly consistent and predictable results that have an impact on defensive coding techniques

Section II provides a brief overview of the background and motivation for the work which led to development of the test rig. A narrative of experimental methods and results forms Section III while the implications of the results are evaluated in Section IV. Conclusions are summarised in Section V, along with suggestions for future work.

II. Background

The initial stimulus for this investigation was contradictory advice received from two different penetration testing laboratories, during the code-review of several payment card applications. The contradiction arose from the recommended approach to double checking a security variable before performing a protected action. One lab recommended that state variables should be tested both positively and negatively, as illustrated by Method A in figure 1, whereas the other was equally assertive in arguing that a state be tested twice before an action was taken, see Method B in figure 1. The conflicting advice was driven by reviewers' respective fault models, which were themselves based on experience or instinct. The proponent of Method A considered the risk of influencing a jump; insisting on both positive and negative confirmation to ensure the program flow to the sensitive function involved both taken and not taken branches. The proponent of Method B perceived the main risk as being data corruption during the variable read; by insisting that the variable be read twice its value is re-confirmed before a sensitive action was taken. No published literature could be found to favour either view and we were concerned that that code could satisfy a formal review yet still be vulnerable. To investigate this issue we created and used a practical test system that was sufficiently precise and controllable to profile the behaviour of individual instructions whilst under laser attack.

```
C source                              | Compiled output
--------------------------------------+---------------------------
volatile signed char bState;          | bState:    .byte 1
--------------------------------------+---------------------------
// -- Method A --                     | MethodA:
// Test and test not.                 |     lds   r24,bState
                                      |     cpi   r24,STRUE
void MethodA(void) {                  |     brne  L4
  if (bState == STRUE) {              |     lds   r24,bState
    if (bState != STRUE) {            |     cpi   r24,STRUE
      Trap();                         |     breq  L3
    }                                 |     rcall Trap
    DoSecretStuff();                  |     rjmp  L4
    return;                           | L3: rcall DoSecretStuff
  }                                   | L4:
}                                     |
--------------------------------------+---------------------------
// -- Method B --                     | MethodB:
// Test and test again.               |     lds   r24,bState
                                      |     cpi   r24,STRUE
void MethodB(void) {                  |     brne  L2
  if (bState == STRUE) {              |     lds   r24,bState
    if (bState == STRUE) {            |     cpi   r24,STRUE
      DoSecretStuff();                |     brne  L1
      return;                         |     rcall DoSecretStuff
    }                                 |     rjmp  L2
    Trap();                           | L1: rcall Trap
  }                                   | L2:
}                                     |
```

Fig. 1: Protected Method Code

III. Experimental Method

In our test rig we use a hardware interrupt synchronised with, but delayed from, the fault stimulus. By making the interrupt service routine responsible for delivering results off the card for analysis, the attacked features are no longer part of the detection or reporting system. While the process is slow it is easily automated.

The first step in the experimental approach was to define the steps in a test method that would generate the results we were interested in. These steps are summarised below, and they were used to define the requirements for the test rig and the μP under test.

1) Reset the μP and preset all registers and memory with known values.
2) Setup the required state for the chosen experiment.
3) Start a timer that will initiate two actions.
    a) Trigger the laser at a precise time in the future.
    b) Cause a system interrupt shortly after the laser pulse.
4) Execute our test program while timing the laser to strike the instruction under investigation.
5) In the interrupt service routine, dump the μP state to the host workstation for off chip analysis.
6) Wait for a reset and start again.

For practical testing, we needed a μP typical of those used in smart-cards but without the additional hardware defences associated with smart cards that would complicate the investigation. Techniques for bypassing defences exist [9] & [20], demonstrating that choosing a defenceless chip does not invalidate the resulting data. It was also desirable to have a μP that was representative of a smart-card, so we selected the Atmel ATtiny841 [1] for this study. The AVR core is widely used in smart cards and is being actively marketed as an IoT component. Other CPUs could have been considered, but the high quality of the documentation and unrestricted access to both development samples and tools made the AVR an ideal candidate.

A. Test Rig

The chip was first removed from its packaging, washed, attached to a new carrier and its electrical connections re-bonded. A small circuit board was built to support the re-mounted chip. Besides providing the power, clock and communications required to program and run the chip this board generates the trigger signals for the laser and facilitates the collection of experimental results. The timer circuit runs at 40 MHz and is divided by four to provide the μP's clock. This provides precise and repeatable laser triggering at quarter cycle intervals during μP execution.

A simple experiment confirmed the correct operation of test circuit. This was achieved by programming the target to initialise and then repeatedly increment a register. The last instruction executed before the interrupt was identified from within the interrupt service routine. Working back from this point, we could identify the instruction that was executing when the laser fired (See the annotations in figure 5) Four runs of this experiment with increasing delays confirmed the phase relationship of the timer quarter-cycles against the the μP clock. This process calibrated the zero of the timer and enabled accurate synchronisation in the main set of experiments.

The circuit board was mounted on a computer controlled X-Y axis microscope stage and a workstation application was written to exercise the tests. The test rig could be programmed to move to any position on the surface of the μP and execute a test script. Scripts specific to each μP instruction were created.

B. Initial Probing

For the initial set of experiments we divided the chip surface into a 100×100 squares, corresponding to a $20\,\mu\text{m}$ grid. The laser exposure footprint was set to match this grid separation, corresponding to approximately 12 track widths as observed on the chip's surface. The size of the laser spot being comparable to [20]'s observations.

It is already understood that with very finely focussed lasers it is possible control individual transistors and to fully control the state of a one bit memory cell [16]. Here, by using a wider laser spot we demonstrate a more practical attack scenario as many of the individual points of interest are obscured by the metal tracks in the top layers of the chip. Each experiment was repeated with laser pulses at differing quarter cycle intervals from one quarter before the observed instruction up until the start of the following instruction, giving 6 samples for a single cycle instruction or 10 samples for a 2 cycle instruction. Each experiment was executed 4 times to look for consistency in the observed effect. This results in either $240,000$ or $400,000$ samples per instruction studied. The typical time needed for an experimental test run was roughly 48 hours.

The tests were grouped to simplify the comparison of similar behaviour.

1) No Operation Tests: provide an opportunity to observe changes to the μP relating solely to instruction fetch with none of the additional complications relating to register transfers or arithmetic calculations. Multiple tests were performed with flags in the status register initialized to differing states before the target NOP instruction was executed.

2) Single Register Update Tests: are characterised by a Read-Modify-Write cycle affecting only one register and a subset of the status flags. Here the R0 register was initialized to different values before INC and DEC instructions were were targeted. The initialization values were chosen so that the expected result generated all possible combinations of flag states.

3) Arithmetic Tests: are binary operations taking input from two sources and sometimes overwriting the first with an updated value. Here the ADD, SUB and CP instructions were tested with registers initialized to generates all possible flag states.

4) Memory Access Tests: carried out simple data transfers between registers and memory. LD and ST instructions were tested while they transferred positive, negative and zero values between memory and registers.

5) Branching Tests: aimed to test the vulnerability of branch decision logic. All possible variations of Carry and Zero flag states were initialized and the BRCC, BRCS, BRNE and BREQ conditional branches tested whilst under attack.

Table I shows a summary of the results of these experiments. The nature of the errors is discussed below.

TABLE I
LASER INDUCED ERRORS

Test Set	Samples	Errors	Unique	Repeat[a]
No Operation	480000	4175	1589	61.9%
Single Register	1920000	14320	6175	56.9%
Arithmetic	3360000	28923	12562	56.6%
Memory	2400000	12873	5368	58.3%
Branching	3200000	22845	8683	62.0%
	11360000	83136	34377	58.6%

[a] $Repeat = (Errors - Unique)/Errors$.

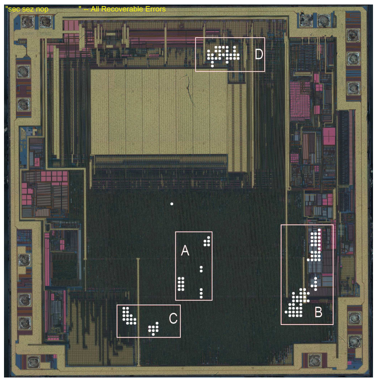

Fig. 2: Error Locations

Figure 2 is a composite map showing all areas on the chip where errors were induced during the NOP tests.

Through analysis of the retrieved data we have identified four categories of error.

1) Total Crash: In some cases it was impossible to recover the μP state after the laser pulse. Clearly corruption had occurred and the μP's interrupt mechanism failed to deliver the information to the host workstation. These errors were rare but where they occurred they were usually repeatable. This behaviour was most frequently observed in the zones labelled A & B in Figure 2.

2) Widespread non-fatal corruption: Multiple registers became simultaneously corrupted; most frequently in zone B. More registers are corrupted than could possibly be achieved via program execution and we assume the register contents are being simultaneously altered. The general behaviour was often repeatable but the erroneous values assigned to the registers had no obvious pattern.

3) Memory corruption: In some instances the memory became corrupted, but with no corruption to the μP state. This behaviour was associated with zone C and appears similar to the effect studied in detail by [17] where RAM was modified directly.

4) Status Register: Here the flags register was corrupted without any other errors induced in the μP. These errors occurred exclusively in Zone D. This was a surprising result, given the stability of the general purpose registers and is discussed further below.

Conspicuous by their absence were errors relating to the current register being updated. These proved very rare under our test conditions, which was surprising as modification during update had been anticipated to be a significant threat. Likewise corruption of data during transfer from memory to register had been considered a potent threat. We see little evidence here but acknowledge that more investigation is needed here because the initial test set only considered transfer to and from RAM.

The most significant result was the high degree of repeatability of the errors. In many cases all erroneous results for a test at a specific location and time were identical.

C. Error Repeatability

Zone D was re-scanned while varying the aperture and the power of the laser. This zone was chosen because a high proportion of all tests exhibited the same effect. Namely flag corruption with no other side effect. For this set of tests we targeted the CMPI instruction, with input variables to generate all possible combinations of status flags. This time we took 10 samples at each location and time interval; the smaller scanned area made this practical. In total $180,000$ sample were taken for each power and aperture setting.

The laser pulse lasts approximately $4\,\mathrm{ns}$. In low power setting our laser emits up to $2\,\mathrm{mJ}$ per pulse. We varied the power from 10% through to 30% and the aperture from 20% through to 60%. The microscope lens has 35% transmission ratio at our chosen wavelength of $532\,\mathrm{nm}$ (green). The results are shown in table II.

These results show that, while total error counts vary, the likelihood of getting a repeat error is both high and remarkably consistent across the full range of apertures and energy levels. The results for flag corruption remained consistent too.

For each power and aperture setting we graphed the error count, see figure 3 for a single example. From this set of graphs, too numerous to show here, we have seen that, sensitive areas yield many errors while adjacent areas show no errors at all; resulting in distinct peaks in the graphs. We also see that our erroneous flag result is strongly correlated with the third quarter phase of the instruction cycle.

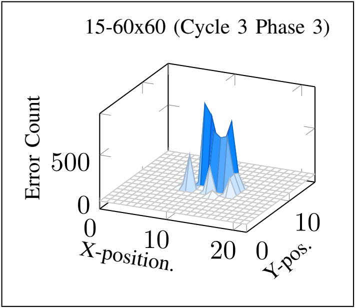

Fig. 3: Errors counts Zone D.

D. Testing the Observed Flag Vulnerability

As individual instructions, branches appeared to be resistent to manipulation but the apparent ability to manipulate flags implied it would be possible to influence branches. By hitting the comparison operation with the laser we would expect flag corruption and a subsequent branch to misbehave. The code in figure 4 was executed with inputs of STRUE, SFALSE and an illegal value of 00H. It was struck with the laser at all possible timing intervals. This was repeated 10 times at all locations in the previously identified Zone D.

Within the result data we expected to find examples of branches erroneously taken and erroneously skipped. The surprise result was we found numerous errors where a branch had been skipped and none where a branch had been taken. Furthermore we observed no flag corruption. Clearly the branch instruction was misbehaving but it was not misdirected by faulty flags as had been anticipated.

Closer observation of the results showed some of the MOV instructions had been skipped. Timing of the laser pulse associated to skipped branches and MOVs was such that it had occurred during the execution of the proceeding instruction. This strongly suggests that instructions are not misexecuting but that the prefetch of the next instruction is being corrupted.

Similar results have observed by [15] on an ARM μP where the cache failed to update, resulting in the re-execution of the previous cache contents. We repeated our test with a series of increment register instructions and observed missing updates as expected but no examples of the additional updates that would be expected if the previously fetched instruction was being re-executed. This also indicates that our initial extensive scanning of the chip surface, instructions and injection times was flawed. All instructions were examined while they pre-fetched a NOP and this result suggest many of the errors we previously observed were due to the misbehaviour of the subsequent misfetched NOP rather than that of the instruction under test.

```
                           ; Timing
     nop                   ; 1.0-1.1
     nop                   ; 1.2-2.1
     cpi  r25,SFALSE       ; 2.2-3.1
     breq L_False          ; 3.2-4.1/5.1
     cpi  r25,STRUE        ; 4.2-5.1
     breq L_True           ; 5.2-6.1/7.1
     rjmp L_Trap           ; 6.2-8.1
...
  L_Weird:                 ; Note: mov
     mov  r0,r25           ; used here
     mov  r1,r25           ; because it
     mov  r2,r25           ; does not
     mov  r3,r25           ; update any
     rjmp L_Weird          ; flags.
...
  L_True:
     mov  r10,r25          ; 7.2-8.1
     mov  r11,r25
     mov  r12,r25
     mov  r13,r25
     rjmp L_True
..
  L_False:
     mov  r16,r25          ; 5.2-6.1
     mov  r17,r25
     mov  r18,r25
     mov  r19,r25
     rjmp L_False
...
  L_Trap:
     mov  r4,r25           ; 9.2-10.2
     mov  r5,r25
     mov  r6,r25
     mov  r7,r25
     rjmp L_Trap
```

Fig. 4: Branch Test Code

Figure 5 shows the probable scenario where a single instruction is skipped.

IV. IMPLICATION

We have shown that the effects of injected errors are far from random and are in fact frequently and consistently repeatable. We have shown that this effect remains consistent across a wide rage of laser apertures and intensities. In particular, wide apertures and low power offer effective and practical attacks that could be mounted using low cost equipment. This brings into question the assertion that "performing two faults on two instructions separated by a few clock cycles is hardly feasible." [10].

TABLE II
ERRORS BY POWER AND APERTURE.

	Aperture[a]														
	20%			30%			40%			50%			60%		
Power[b]	µJ[c]	N[d]	R[e]	µJ	N	R	µJ	N	R	µJ	N	R	µJ	N	R
10%	2.8	0	0%	6.3	101	85%	11.2	1842	93%	17.5	8142	97%	25.2	9020	95%
15%	4.2	2642	95%	9.5	3729	95%	16.8	6934	96%	26.3	7848	96%	37.8	10219	95%
20%	5.6	3640	96%	12.6	4952	92%	22.4	6604	95%	35.0	5518	95%	50.4	8362	95%
25%	7.0	4127	95%	15.8	5803	96%	28.0	5676	95%	43.8	8208	96%	63.0	10773	96%
30%	8.4	4543	96%	18.9	5899	96%	33.6	4797	94%	52.5	8738	96%	75.6	9690	97%

[a] Aperture diameter as a percentage of 40 µm [b] Laser power setting as a percentage of 2 mJ per pulse. [c] Energy delivered to the chip surface after masking by the aperture and losses in the microscope optics. [d] Total count of errors from 180, 000 samples taken within Zone D. [e] Percentage of errors that are duplicates for a single location and stimulus timing. Where $repeatability = (Errors - UniqueErrors)/Errors$.

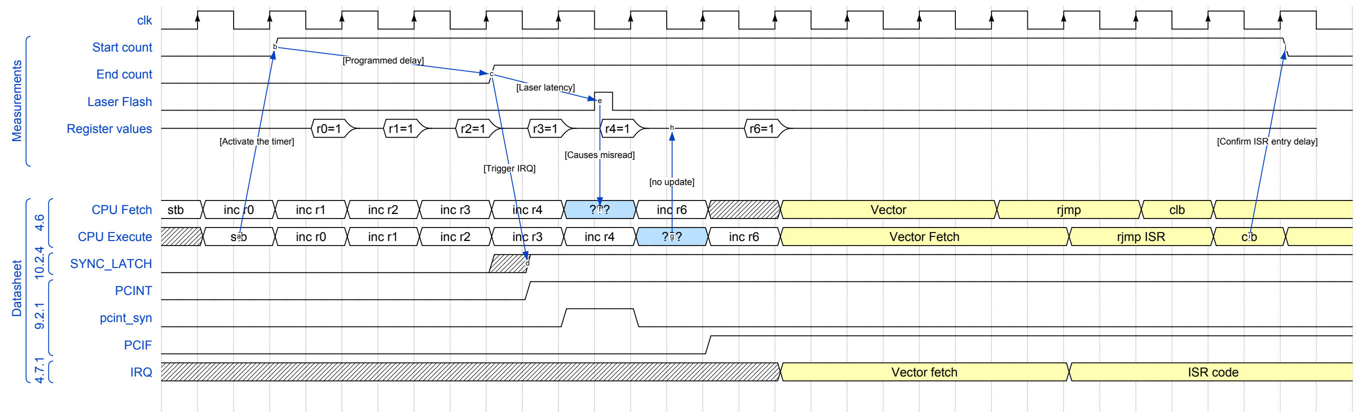

Fig. 5: Signal Timing

In the light of the above we have reevaluated the initial advice that prompted this study. Figure 1 shows the two code samples and their corresponding assembler output. Our results here suggest that the original Method A provides a better defence than its replacement. In principle the strongest approach is to place critical processing at the destination of branches, making it unreachable when branches are skipped. Additionally, placing the defended code at low addresses and the tests that invoke it at higher addresses will reduce the risk of repeated instruction skips from falling through in to it. This is a simple practical approach that can be achieved through source-code and control of the compiler's optimisation settings. This technique will also be robust on a CPU with an instruction cache or pipeline, where [15] & [21] have demonstrated that consecutive instructions may be skipped or misinterpreted after a single error.

A. Reevaluation of Historic Code

We were fortunate to have access to defensive code that had been re-used within certified EMV cards, a National ID card scheme, and a Java Card OS implementation. Looking only at the software implemented defences, and hypothetically considering the implications of the same source code being targeted at the μP studied here, we noted three common areas of vulnerability.

1) State variable: double testing as described above is used in all of the applications. The Global Platform card manager implemented within the JavaCard OS shows many examples of this sub-optimal defensive code; for example card life-cycle state and secure session state influence the availability of commands and in all cases the restricted code will be reached by skipping the branch.

2) Waymarkers or Sentinel values: are used in the three most recent projects to record execution of subroutines and to detect skipped or misexecuted code. In all cases where a test is performed the code branches to a trap function when the state is not as expected. It would be significantly stronger if execution branched when the state was as expected and falling through to the trap by default. This of course would have the disadvantage of making the source code untidy to the verge of being unreadable.

3) Variable redundancy: widely used in the JavaCard project. Here, for example, multiple loop counters and termination conditions are used to control loops, ensuring either the correct number of repetitions or detection and trapping for inconsistent states. In the compiler generated code used for sanity checking of loop termination we repeatedly see conditional branches to the trap condition.

V. Conclusions

The test rig and error state recovery mechanism has proved itself to be a flexible and effective tool, providing insights and information not available from simulations. Here we have demonstrated its effectiveness in characterizing the risk of skipped instructions and in particular branch operations. It can be used to quickly identify vulnerable areas of a chip and quantify the likelihood of generating reproducible, and therefore exploitable, errors. Generating a more complete, realistic and accurate fault model for a chip will simplify the task of application development and testing.

We can confidently dispel the myth that error effects are random. If an error can be induced then it can probably be reproduced. Therefore repeating tests offers limited advantages and defences that rely on the combinatorial effects of low probabilities are therefore flawed.

We have confirmed long held suspicions that instruction skipping is a significant threat. By demonstrating that equally reproducible results can be obtained from low power and wide aperture, we challenge suggestions that double fault injection is prohibitively expensive or difficult [10]. In fact it appears to be practical with low budget equipment; an observation we intend to demonstrate next. Paradoxically our expensive laser equipment cannot generate pulses at short intervals whereas readily available low-cost solid state diodes can do this.

More work is required to characterise the failures observed in other parts of the chip and to confirm the observation that register and RAM access is harder to adversely influence. Similarly the ease of corruption of the instruction pre-fetch suggests that other aspects of memory access may be vulnerable and our test examples did not cover these. We also intend to confirm our initial assumption that the technique is equally applicable when using an infra-red laser from the rear side of the chip. The low-power and wide aperture results suggest this was a safe initial assumption.

While we have concentrated on branch operations the failure mechanism affects all instructions. It is therefore obvious that, on this chip, all calculations are vulnerable to skipped instruction errors and that any answers, particularly cryptographic results, must be verified before being disclosed or acted upon.

The work here has been exclusively performed on one version of the AVR micro-controller. We believe the techniques described are equally applicable to other μP architectures, which we intend to verify in future work.

References

[1] Atmel Corporation. *ATtiny841 Datasheet – 8-bit AVR Microcontroller with 4/8K Bytes In-System Programmable Flash*, 05 2014. Rev. 8495H.

[2] C. Aumüller, P. Bier, W. Fischer, P. Hofreiter, and J.-P. Seifert. Fault attacks on rsa with crt: Concrete results and practical countermeasures. In *International Workshop on Cryptographic Hardware and Embedded Systems*, pages 260–275. Springer, 2002.

[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*, 94(2):370–382, 2006.

[4] E. Biham and A. Shamir. Differential fault analysis of secret key cryptosystems. In B. S. K. Jr., editor, *Advances in Cryptology - CRYPTO '97, 17th Annual International Cryptology Conference, Santa Barbara, California, USA, August 17-21, 1997, Proceedings*, volume 1294 of *Lecture Notes in Computer Science*, pages 513–525. Springer, 1997.

[5] D. Boneh, R. A. DeMillo, and R. J. Lipton. On the importance of checking cryptographic protocols for faults (extended abstract). In W. Fumy, editor, *Advances in Cryptology - EUROCRYPT '97, International Conference on the Theory and Application of Cryptographic Techniques, Konstanz, Germany, May 11-15, 1997, Proceeding*, volume 1233 of *Lecture Notes in Computer Science*, pages 37–51. Springer, 1997.

[6] L. Dureuil, M.-L. Potet, P. de Choudens, C. Dumas, and J. Clédière. From code review to fault injection attacks: Filling the gap using fault model inference. In *International Conference on Smart Card Research and Advanced Applications*, pages 107–124. Springer, 2015.

[7] O. Goloubeva, M. Rebaudengo, M. Reorda, and M. Violante. Soft-error detection using control flow assertions. In *Defect and Fault Tolerance in VLSI Systems, 2003. Proceedings. 18th IEEE International Symposium on*, pages 581–588, Nov 2003.

[8] Infineon Technologies AG. *Whitepaper. Integrity Guard — The newest generation of digital security technology.*, 09 2012. Rev. 4.12.

[9] O. Kömmerling and M. G. Kuhn. Design principles for tamper-resistant smartcard processors. In *Proceedings of the USENIX Workshop on Smartcard Technology on USENIX Workshop on Smartcard Technology*, WOST'99, pages 2–2, Berkeley, CA, USA, 1999. USENIX Association.

[10] N. Moro, K. Heydemann, E. Encrenaz, and B. Robisson. Formal verification of a software countermeasure against instruction skip attacks. *Journal of Cryptographic Engineering*, 4(3):145–156, 2014.

[11] NXP Semiconductors N.V. *Shortform Datasheet. SmartMX2 P40 family P40C012/040/072 — Product short data sheet, Company Public*, 04 2015. Rev. 3.0, 262830.

[12] R. Omarouayache, J. Raoult, S. Jarrix, L. Chusseau, and P. Maurine. Magnetic microprobe design for em fault attack. In *Electromagnetic Compatibility (EMC EUROPE), 2013 International Symposium on*, pages 949–954. IEEE, 2013.

[13] D. L. Palumbo and R. W. Butler. A performance evaluation of the software-implemented fault-tolerancecomputer. *Journal of Guidance, Control, and Dynamics*, 9(2):175–180, 1986.

[14] G. Reis, J. Chang, N. Vachharajani, R. Rangan, and D. August. Swift: software implemented fault tolerance. In *Code Generation and Optimization, 2005. CGO 2005. International Symposium on*, pages 243–254, March 2005.

[15] L. Riviere, Z. Najm, P. Rauzy, J.-L. Danger, J. Bringer, and L. Sauvage. High precision fault injections on the instruction cache of armv7-m architectures. In *Hardware Oriented Security and Trust (HOST), 2015 IEEE International Symposium on*, pages 62–67. IEEE, 2015.

[16] C. Roscian, A. Sarafianos, J.-M. Dutertre, and A. Tria. Fault model analysis of laser-induced faults in sram memory cells. In *Fault Diagnosis and Tolerance in Cryptography (FDTC), 2013 Workshop on*, pages 89–98. IEEE, 2013.

[17] S. P. Skorobogatov and R. J. Anderson. Optical fault induction attacks. In B. S. K. Jr., Ç. K. Koç, and C. Paar, editors, *Cryptographic Hardware and Embedded Systems - CHES 2002, 4th International Workshop, Redwood Shores, CA, USA, August 13-15, 2002, Revised Papers*, volume 2523 of *Lecture Notes in Computer Science*, pages 2–12. Springer, 2002.

[18] STMicroelectronics N.V. *Shortform Datasheet. ST23ZC08 — Secure microcontroller with enhanced security.*, 03 2012. ID 019021 Rev 2.

[19] N. Theißing, D. Merli, M. Smola, F. Stumpf, and G. Sigl. Comprehensive analysis of software countermeasures against fault attacks. In E. Macii, editor, *Design, Automation and Test in Europe, DATE 13, Grenoble, France, March 18-22, 2013*, pages 404–409. EDA Consortium San Jose, CA, USA / ACM DL, 2013.

[20] J. G. Van Woudenberg, M. F. Witteman, and F. Menarini. Practical optical fault injection on secure microcontrollers. In *Fault Diagnosis and Tolerance in Cryptography (FDTC), 2011 Workshop on*, pages 91–99. IEEE, 2011.

[21] B. Yuce, N. F. Ghalaty, H. Santapuri, C. Deshpande, C. Patrick, and P. Schaumont. Software fault resistance is futile: Effective single-glitch attacks. In *Fault Diagnosis and Tolerance in Cryptography (FDTC), 2016 Workshop on*, pages 47–58. IEEE, 2016.