Methodology for EM Fault Injection: Charge-based Fault Model

Haohao Liao
Department of Electrical and Computer Engineering
University of Waterloo
Waterloo, Canada N2L 3G1
haohao.liao@uwaterloo.ca

Catherine Gebotys
Department of Electrical and Computer Engineering
University of Waterloo
Waterloo, Canada N2L 3G1
cgebotys@uwaterloo.ca

***Abstract*—Recently electromagnetic fault injection (EMFI) techniques have been found to have significant implications on the security of embedded devices. Unfortunately there is still a lack of understanding of EM fault models and countermeasures for embedded processors. For the first time, this paper proposes an extended fault model based on the concept of critical charge and a new EMFI backside methodology based on over-clocking. Results show that exact timing of EM pulses can provide reliable repeatable instruction replacement faults for specific programs. An attack on AES is demonstrated showing that the EM fault injection requires on average less than 222 EM pulses and 5.3 plaintexts to retrieve the full AES key. This research is critical for ensuring embedded processors and their instruction set architectures are secure and resistant to fault injection attacks.**

Keywords—side channel, fault injection, EM, fault model, embedded processor security

I. INTRODUCTION

Security is increasingly widespread in many embedded devices. The devices themselves must store cryptographic keys and be designed to support secure boots, secure processing, etc. In addition to correct implementation, the security must be resistant to physical attacks. For example electromagnetic waves maybe focused onto the device (using EM pulses) causing a change in some computation or data within the device (referred to as an EM fault injection attack). Although research has examined fault injection attacks for several decades, there remains limited understanding of the fault models of embedded processors and how countermeasures can be designed. Fault injection attacks have important consequences on embedded systems and thus quantization of the threat is important. For EM fault injection attacks, the number of EM pulses required to launch an attack would also be useful to quantize. In theory if x fault injections may reveal the complete key and on average y EM pulses are required before a fault injection is possible, then the secret key lifetime must support on average less than xy invocations of the cryptographic algorithm to prevent a successful attack. Since an increasing number of devices are flipchip packaged, the susceptibility of EMFI to the backside is also important.

II. PREVIOUS RESEARCH

Fault injection attacks include power or clock glitching, electromagnetic pulse injection, laser injection, and others. In general electromagnetic fault injection attacks are lower cost and have fewer requirements other than access to chip or de-capsulated die. Some earlier fault injection techniques included constant under-voltaging [5] which produced reset faults in memory loads independent of address values. This research was likely the first to suggest instruction replacement fault types. Researchers in [7,8] have suggested that forward body biasing injection (FBBI) would have better spatial resolution than that achieved with an EM pulse on the backside of the IC, however only approximately 2% of faults were exploitable. Researchers in [1] have proposed a timing fault model where the coupling between the EM pulse and power-ground network temporarily reduces the power supply, increases the delay, and finally causes a setup time constraint violation. Later in [2] a sampling fault model was proposed suggesting the EM pulse temporarily alters the voltage of several nodes in the circuit during some time and after this time interval, the chip quickly recovers its original state, suggesting the faults should be injected near the clock edge to affect values latched into registers and memory. Research in [3] observed bit set faults and suggested metastability as a cause during instruction loads from flash. Other researchers have examined EM fault injection effects on clock glitching [15], DRAM chips [13], or using harmonic EM fault injection [16]. Real attacks using EM fault injection have been demonstrated on secure boot functionality indicating instruction replacement faults [17]. Most cryptographic countermeasures assume data is faulted with zeros or random values [6] or instruction skip faults [10] accounting for only a limited set of observed fault injections [11]. There remains a limited in-depth analysis of processor based faults, including limited quantization of attack difficulty in terms of number of EM pulses required. Additionally there remains a lack of understanding of fault models and fault types. In this paper a fault model will be defined as the mechanism explaining how a fault may occur and fault types refer to specific instructions (faulty instruction replacement) which model the faulty behavior of the targeted fault-injected instruction.

III. EXPERIMENTAL METHODOLOGY AND RESULTS

The experimental setup and methodology along with empirical results on an embedded microcontroller are described in this section. The proposed charge-based fault model derived empirically along with in-depth analysis and quantization of the faulty instruction replacement and an AES attack are presented.

A. EMFI Setup and Methodology

Experimental setup was based on a low cost desktop CNC machine (approx. $2.5K USD), an EM pulse generator and EM probe. The CNC machine was retrofitted to hold the EM probe and enabled accurate xyz placement with a resolution of 12.7 um. The CNC machine also provided automated backside de-capsulation of the processor. The EM pulse generation system used was the EMV Langer Burst power station 202 [12] (EM pulse has approximate 2ns rise time). The EM probe tip (approx. 0.5um) was placed approximately 0.8mm above the exposed backside die surface of the processor. The device under analysis is a backside de-capsulated PIC16F687 (chosen due to the simple 14-bit opcodes and DIP package with die size approximately 2.3mm X 2.6mm). The PIC16F687 utilizes 4 Q clock cycles per instruction cycle. The instruction cycles utilize a 2-stage pipeline (fetch and execute instruction stages). The PIC16F687 was (over)clocked with an external 52MHz clock (supplied by a function generator) and was used to generate an external trigger to the EM system. The EM pulses utilized for fault injection were 500V single positive pulses, unless otherwise stated. All programs were written in assembly language to enable detailed timing analysis. Due to the lack of a high speed debugger for the processor, empirical analysis was performed using assembly code design to confirm the presence and details of injected faults. Parameters analyzed for all experiments included timing of EM pulse, % of rounds which had a specific fault and statistics on the number of EM pulses required to inject the first fault. Initially the EM pulse injection was scanned over a test code sequence in 10ns and shorter intervals in order to find specific EM pulse times where faults were successful. The fault injection experiments utilized a master python script (to control/synchronize the CNC machine, function generator, EM pulse generator and PIC programmer) in addition to assembly programs which executed a set number of rounds. In each round a loop of code was responsible for generating an output trigger signal and executing a series of instructions, one of which was the target of the fault injection. The subsequent instructions in the loop after the target instruction were designed to handle fault detection and analysis.

Faults in both data and control were detected in software in each loop through several iterations of assembly design until all faults were accountable. Generally data-based faults were detected from the data values written or not written to sRAM, registers or status bits. For example, if the target instruction were to store data from a register (or through ALU) to memory, the program would initially examine the register and memory data values (or status bits). If an unexpected data value was found, the program would jump to a fault handling routine. The fault handling routine would place a flag data value in memory and then process the fault. Flag data was utilized to check for control faults. For example if data was not written as expected and flag data was not set, yet the program executed the fault handling, it was likely a control type of fault was injected. In this case further flags were placed into the code to help trace the control path of the faulty instruction.

The best placement of the EM probe relative to the die was found using a memory writing program synchronized with the CNC machine. After every 10 rounds, the probe was moved to a different *xy* location, gradually scanning over the entire area of the die. During each round fault injections and other parameters were recorded. Initially the probe tip was placed approximately over a corner of the die. Then, for the first scan over the die, a low resolution was used to find a sensitive area where faults are more likely to be injected. Fig. 1. illustrates this coarse grained shmoo plot over the die region using different magnitudes of EM pulse voltages. Next, a more detailed scan with a higher resolution was performed on the sensitive region identified in the first scan. The best probe location (based on large number of faults with low standard deviations) was utilized for all subsequent experiments.

Further analysis of the memory write program, indicated that the fault (a single bit reset) was actually related to the prefetch of the instruction which followed the memory write, specifically *movlw*, which loaded an immediate data value into a temporary working register, the *w* register. This was empirically confirmed by inserting several *nop*s in between the memory write and the *movlw* (and adjusting the trigger signal accordingly so that the EM pulse occurred during the prefetch of *movlw*). Faults were largely successfully injected near a specific Q cycle edge in the instruction prefetch stage (likely the end of the Q4 cycle due to 50ns typical off-chip delay [9], likely corrupting the instruction register load during the start of following execution stage [9]). Unfortunately this could not be verified beyond our experimental results since it is well known that contents of the instruction register are not available.

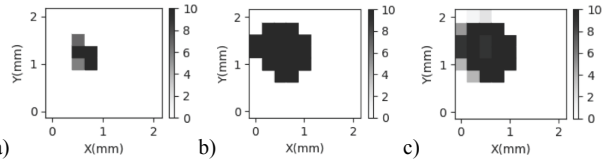

Fig. 1. Coarse grained shmoo plots of # rounds with fault injection of *movlw 0x00* over die using 70V a), 150V b), and 500V c) EM pulse voltages.

B. Charge-based Model

It was empirically observed, similar to [2], that fault injection was possible only during a specific timing window near a specific clock edge. However unlike [2], when the clock frequency was reduced, no fault injection was possible throughout the clock period. Further experimentation indicated that some lower clock frequencies could only be made susceptible to fault injection with a suitable reduction in *Vdd*, as shown in Table I. Note all explored combinations of *Vdd* and clock frequency were fully functional without fault injection. The delays in columns *D1* to *D4* of Table I represent the delay between the respective clock edge and the EM pulse. A positive delay means the EM pulse occurs before the specific clock edge. For example, *Dx* is the delay between the EM pulse and a rising or falling clock edge *x*, where $x = 1,2,3,4$ are consecutive falling, rising, falling, rising edges of a clock respectively (representing 1.5 clock cycles). In the first row of Table I, the EM pulse which occurs 3.7ns before the first falling clock edge *D1*, produces a fault at 52MHz, 5V (see first column). However, in row 2, when the EM pulse occurs again at 3.7ns before the same first falling clock edge *D1*, no fault occurs at 46.02MHz, 5V. Similarly for the next 3 rows where the EM pulse has the same delay distance from the other

underlined clock edges, with the slower clock frequencies no fault is injected at 5V. Next, for each EM pulse timing, the supply voltage, *Vdd,* was continually reduced by 0.1V until the fault could be injected, as shown in the second column of Table I, empirically demonstrating that at lower clock frequencies only with lower power supply can EM faults be injected.

A possible theoretical explanation for the observed behavior can be obtained through examining the charge-based model. Consider a circuit node which under normal operating conditions charges from a '0' to a '1'. At 5V with slower clock frequencies, a maximum amount charge is built up on the capacitor at the circuit node. Although the EM pulse can modify the charge (via noise or power-ground network), it cannot reduce a sufficient amount of charge from accumulating on the node to cause a '0' to occur (when latched into a flipflop). When operating with a high clock frequency (e.g. overclocking) at 5V, the capacitors are charged just as quickly but for a shorter clock period duration, hence the total charge accumulated on the circuit node is less (and may only be just over the threshold to reach a '1' , also referred to as low swing signals). When EM pulses are applied to this node, since the capacitor is not fully charged, the EM pulse injection is able to disturb a sufficient amount of charge to change the bit value (when latched into a flipflop), such that the '1' threshold is not reached and thus the bit becomes a '0'. The amount of charge displacement required to change the bit value is much less than the case where a slower clock frequency is utilized. When slower clock frequencies are employed with a reduction in *Vdd*, the capacitance is charged slower (due to lower *Vdd*), and hence has less charge at the end of the clock cycle. Thus when the EM pulse is applied, again less charge needs to be disturbed in order to prevent the bit from reaching the '1' (when latched into a flipflop). In effect the proposed extended fault model is based on the minimum amount of charge required to change the normal behavior of a circuit at a sensitive node. This amount of charge is often referred to as critical charge in the field of single event upsets [4]. At low clock frequencies with reduced *Vdd* or at high clock frequencies, the critical charge is smaller and thus the circuit is more susceptible to EM fault injection.

C. Instruction Replaced Faults

Table II illustrates the replaced faulty instructions (column 2) which replaced the target instruction (column 1) as a result of successful fault injection. A total of 340 rounds were utilized in the experiments of Table II. The time between the EM pulse and the end of the appropriate Q cycle was 15.23ns for all instructions except for first 2 rows of table which utilized delays of 12.15ns and 16ns. For example, in the first row, the target instruction was *movlw 0x80* and the EM pulse was 12.15ns from the end of the appropriate Q cycle. The EM pulse caused the least significant bit of the data value pointed to by the file select register (indirect address) to be reset (to 0). The only instruction to achieve this was *bcf INDF,0.* This is completely unrelated to the function of the target instruction (which was loading the value *0x80* into the *w* register) thus further supporting the notion that likely the fault is affecting the load of the instruction register. For a targeted instruction in a specific code sequence, a small variation of the EM pulse timing often led to quite different faulty instructions, as seen in rows 1 and 2 of Table II.

TABLE I. FAULT INJECTION (FI) VS CLOCK FREQUENCY AND VDD

FI/5V	FI/Vdd	Clk.Freq (MHz)	Delay (ns)			
			D1	*D2*	*D3*	*D4*
Yes	Yes/5V	52	3.7	13.3	22.9	32.5
No	Yes/4.2V	46.02	3.7	14.6	25.4	36.3
No	Yes/4.2V	46.18	2.5	13.3	24.1	35
No	Yes/4.4V	46.33	1.3	12.1	22.9	33.7
No	Yes/4.4V	46.48	0.3	11	21.8	32.5

In the last 4 rows where the targeted instruction is *xorwf 0x20, f* again different faulty instructions were found even though the delay of the EM pulse with respect to the prefetch of the target instrution was the same. In case *a1* and *a2*, the target instruction resided at the same address, and the exact same code sequence was used except for the immediate 3 instructions before the target instruction. In case *a1* the 3 instructions preceding the target instruction are 2 *nop*'s and *movf 0x30,w* , whereas in case *a2* the target instruction is preceded by *movf, xorwf,f* and *movf 0x30,w* type instructions. Similarly for cases *b1* and *b2* the target instruction is preceded by the same two sequences as in *a1* and *a2* respectively, but this code is embedded within a complete AES program, not a test program. In the last column of Table II, one can see that in all cases, except in one case, the bits are more likely to be reset and not set. In other words the effect of the EM pulse is mostly likely to cause opcode bits to transition from 1 to 0.

In all cases the faulty instruction opcodes appeared to be uncorrelated to opcode values from preceding instructions. Further experimentation revealed that address bits within the opcode (typically *b6..b0*) were also not susceptible to fault injection. Negative EM pulses also injected faults but only after increasing the delay to 25.23ns. The variance of fault injections was also quantized, indicating that standard deviations were under 2%. For example in row 3 of table II, the *NOP* replaced instruction occurred *56.5% +/- 2.8% (2std).* The averge number of EM pulses before the first *NOP* fault was injected was at minimum 1 EM pulse and at most 11 EM pulses.

D. AES attack

The complete AES code was utilized to launch a fault injection attack to retrieve an AES key byte. The attack assumes the correct and faulty ciphertext is available, so the attacker knows if a fault has been injected or not, but the plaintext is not known (random). The *xorwf 0x20,f (w xor f=>f)* instruction of the last round, which computes the exclusive or of key byte *n* (stored in register *w*) with the AES state (initially stored in memory at address *f*) was targeted. Note that if faulty instruction *movwf 0x20 (w=>f)* occurs key byte *n* (*k*) will be output in the faulty ciphertext byte. Otherwise if faulty instruction *nop* occurs, the AES state (*s*)

will be output in the faulty ciphertext byte. In this later case the result of the exclusive-or of this faulty ciphertext (*s*) with the correct ciphertext (= [*s* xor *k*]) will reveal the AES key byte (= *s* xor [*s* xor *k*]). Since either faulty instruction will reveal the key byte, the attack maintains a list of pairs, one per fault injection, where each pair is [faulty-ciphertext-byte , exclusive-or]. The list grows until two pairs have either the same faulty ciphertext byte or same exclusive-or value, in which case the potential correct key byte has been found. We denote this as a potential correct key byte since there remains a small probability that an incorrect key may be found (since there are only 256 possible byte values). Empirical results of fault injection attacks on the complete AES algorithm executing on the processor were obtained. Using this attack, out of a total of 16 different AES keys, 11 keys were correctly identified.

TABLE II. CODE SEQUENCE EFFECTS ON FAULT INJECTION AT 52MHZ

Target Instr	Statistics		
	Replaced faulty instruction	***Frequency Occurence***	***Opcode Bits reset[set]***
movlw 0x80 [d1]	*bcf INDF, 0*	98.4%	b13, b7
movlw 0x80 [d1]	*movwf INDF* *goto 0x80*	56.4% 42.3%	b13,b12 b12[b11]
xorwf 0x20,f [a1]	*iorwf 0x20,w* *NOP* *iorwf 0x20,f*	40.3% 57.9% 1.8%	b9,b7 b10,b9,b7 b9
xorwf 0x20,f [b1]	*subwf 0x20,f* *movwf 0x20*	65.8% 34.1%	b10 b10,b9
xorwf 0x20,f [a2]	*iorwf 0x20,w* *nop* *xorwf 0x20,w*	60% 35% 2.6%	b9,b7 b10,b9,b7 b7
xorwf 0x20,f [b2]	*iorwf 0x20,f*	98.7%	b9

[d1] Delay of 12.15ns and 16ns for 1st/2nd row. [a1,a2,b1,b2] Instruction embedded within diff. code seq.

The number of EM pulses required to obtain a single correct AES 128-bit key varied from 134 to 453, with an average of 222 and standard deviation of 81. The statistics shown in Table II rows *b1* and *b2*, indicated that possibly with a different implementation of AES, the faulty instruction of the targeted *xorwf 0x20,f* may also have been *iorwf 0x20,f* with 98.7% frequency. In this case an AES attack would also be possible, by monitoring the faulty ciphertext generated from executing different plaintexts, recording bits of the targeted ciphertext byte which become '0'. In both attack cases and unlike previous research, the attack does not make assumptions that the fault model is a bit flip, random byte/word, or instruction skip.

IV. DISCUSSIONS AND CONCLUSIONS

It is well known that as technologies progress the critical charge also reduces. Hence it is possible that the EM pulse was not strong enough to disturb the larger critical charge at slower clock frequencies. The *xorwf,f* instruction researched in this paper was shown to have a significant impact on the AES security application. Although previous research had observed the key byte appearing in the faulty ciphertext, only 2 out of 16 key bytes had this type of fault [1] and the suggested attack algorithm did not include handling of the *nop* faulty instruction. In section III.D a complete attack on an AES key byte in a real implementation of AES was performed. Since the number of plaintexts required was extremely low, it is expected that with higher EM pulse voltage or more accurate EM pulse timing, the number of EM pulses used in the attack may be reduced further. Attacks using fault injection of other instructions, such as the *movlw* instruction, could likely also be exploited in applications such as secure boot where the corresponding faulty instruction *goto* (2nd row of Table II) could potentially skip authentication checks during secure boot. For the first time an extended charge-based fault model is presented for backside EMFI and validated empirically. Unlike previous research, the IC backside is shown to be very susceptible to localized and repeatable EM fault injection. For the first time it was shown that a real implementation of AES can be attacked in a simple manner. This research is important for improving resilience and countermeasures for fault injection attacks which are critical for preventing future EM fault injection attacks. The authors would like to thank Mustafa Faraj for useful discussions and development of some of the tools used in these experiments. The research is supported by funding in part by grants from NSERC and XtremeEDA.

REFERENCES

[1] A.Dehbaoui, J.-M. Dutertre, B. Robisson, and A. Tria, "Electromagnetic transient faults injection on a hardware and a software implementations of AES," in *Fault Diagnosis and Tolerance in Cryptography (FDTC), 2012 Workshop on, pp. 7-15, IEEE, 2012.*

[2] S.Ordas, L. Guillaume-Sage, and P. Maurine, "Electromagnetic fault injection: the curse of flip-flops*," Jnl of Crypt Eng*, pp. 1-15, 2016.

[3] N.Moro, et.al. "Electromagnetic fault injection: towards a fault model on a 32-bit microcontroller," in *FDTC 2013, pp. 77-88, IEEE, 2013.*

[4] P.Shivakumar, M. Kistler, S. W. Keckler, D. Burger and L. Alvisi, "Modeling the effect of technology trends on the soft error rate of combinational logic," *Proc. Intl Conf on Dependable Sys and Networks*, Washington, DC, USA, 2002, pp. 389-398.

[5] A.Barenghi et. al., "Low Voltage Fault Attacks to AES and RSA on General Purpose Processors," eprint iacr 130/2010, 2010.

[6] P.Rauzy, S.Guilley, "A formal proof of countermeasures against fault injection attacks on CRT-RSA," eprint IACR 506/2013, 2013.

[7] K.Tobich, et.al., "Voltage spikes on the substrate to obtain timing faults", Proc. of the 16th Euromicro Conf. on Dig. Sys.Des., 2013.

[8] P.Maurine, "Techniques for EM Fault Injection: Equipments and experimental results", in *FDTC 2012*, Belgium, 2012, pp.3-4.

[9] Microchip "PIC16F631/677/685/687/689/690 20-pin flash-based, 8-bit CMOS microcontrollers, DS40001262F Microchip, 2015.

[10] A.Barenghi et.al. "Countermeasures against faut attacks on software implemented AES", Proc. of *WESS 2010,* ACM, 2010, pp.1-10.

[11] N.Moro et.al. "Experimental evaluation of two software countermeasures against fault attacks" IEEE Proc. of *HOST 2014*, 2014.

[12] EMV Langer Burst power station 202 and ICI HH500-15 LEFT pulse magnetic field source EM probe, http://www.langer-emv.de

[13] L.Riviere et.al. "High precision fault injections on the instruction cache of ARMv7-M architectures" IEEE *HOST 2015*.

[14] A.Cui, R.Housley "BADFET: Defeating modern secure boot using second-order pulsed electomagnetic fault injection" Usenix workshop on offensive technologies, *WOOT, 2017*.

[15] M.Ghodrati "Thwarting electromagnetic fault injection attack utilizing timing attack countermeasure" MASc Thesis, Advisor P.Schaumont, Dec 2017.

[16] A.Boyer et.al. "Evaluation of the Near-Field Injection Method at integrated circuit level" EMC Europe 2014, 2014, pp.1-6.

[17] N.Timmers, Spruyt A., Witteman M. "Controlling PC on ARM using fault injection" in *Fault Diagnosis and Tolerance in Cryptography (FDTC), 2016 Workshop on, IEEE, 2016.*