A FAULT INJECTION SIMULATOR FOR EMBEDDED PROCESSORS BASED ON RTOS

BINAPANI MOHAPATRA Raajdhani Engineering College, Bhubaneswar binapani@rec.ac.in

***Abstract*—** *The increasing risks that fault injection attacks pose to chip security have made it more crucial than ever to assess embedded systems' susceptibility to them. Since micro-controllers are less resilient to fault attacks than hardware-based cryptosystems, the task is especially crucial for them. This work provides an embedded high-level fault injection method based on a Real-Time Operating System (RTOS) and examines recent embedded fault injection simulators from the literature. The method seeks to be portable to 32-bit micro-controllers and embedded processors while remaining architecture-independent. The suggested method is modified for scheduled and event-based fault injection, and it mainly targets genuine fault attack scenarios on memory locations. To demonstrate the suggested mechanism, a Differential Fault Attack (DFA) was installed on a widely used ARM-based micro-controller running FreeRTOS. The increasing risks that fault injection attacks pose to embedded systems have made it more crucial than ever to assess an embedded system's susceptibility to them. The objective is also to emphasize the significance of physically safeguarding the memory and including data-specific countermeasures, as well as to effectively connect the embedded fault injection simulation mechanism to a computer-based cryptanalysis.*

***Keywords*—Cryptography; DFA; Fault Injection; Simulator; RTOS; ARM; Microcontroller; MATLAB**

I. INTRODUCTION AND BACKGROUND

In the Internet of Things era, personal and sensitive data exchanges have been made common between embedded systems. However this evolution must be accompanied by adequate security mechanisms. Depending on the level of secrecy of the data and the available resources, an embedded system may encrypt or decrypt data using software routines or rely on a distinct cryptographic hardware accelerator.

In fact, data that should be kept a secret is potentially subject to physical attacks where a malicious attacker tries to retrieve it partially or entirely (i.e. the secret key used to encrypt and decrypt). Physical attack aims to break security functionalities of any cryptographic scheme by targeting its implementations rather than trying to break its mathematical security which is generally unbreakable if recommended design parameters are used. There exist two main families of physical attacks: Side Channel Analysis (SCA) and Fault Analysis (FA). Side-Channel Analysis is a family of passive attacks comprising various types of attacks but mainly dominated by the power-monitoring attacks and electromagnetic attacks. The first consists in analysing power consumption of circuits while the second analyses their electromagnetic (EM) emissions. Over the years, several SCA techniques have been reported in the literature for power-monitoring attacks (Simple Power Analysis and Differential Power Analysis) [1], Electromagnetic Analysis Attacks [2], and Timing Attacks [3], etc. On the other side, fault attacks are active attacks which were first introduced by Boneh et *al.* on a microcontroller [4]. In a fault attack scenario, an attacker, with a physical access to a device, running a known program, tries to perturb its operation to induce faults using laser beam,voltage glitch, under powering, clock glitching, electromagnetic emissions, heating, etc., and then analyses theoutput to retrieve the secret data. Several fault attacks techniques exist and the most widely used technique is called the Differential Fault Analysis (DFA) [5]. This technique is based on comparing a certain number of faulty and fault-free outputs to derive information about the secret key. Research onFA techniques has been very active in both academic and industrial communities in the past twenty years and has revealed many exploitable design weaknesses for almost all cryptosystems families [6]. This has contributed to introducing new design practices to secure implementations against fault attacks for hardware designs [7] as well as software for embedded processors [8].

A. Fault Injection Attacks on Microcontrollers

Cryptographic software routines running on embedded processors and microcontrollers can integrate effective software countermeasures against SCA [9]. However, they are more vulnerable to FA [10] [11] [12] compared to cryptographic chips. In fact, the latter have a specific architecture with specifically designed countermeasures to FA. In addition, they are a black-box target from attacker's perspective. On the other side, software routines generally run on a known microcontroller's architecture where protection against fault injection attacks is limited to the microcontroller's default hardware countermeasures and the scheme-specific

software countermeasures [13]. Recent works, try to fill this gap by combining software SCA and FA countermeasures in general purpose microcontroller [14].

Developing tools and methods to evaluate vulnerabilities on embedded processors is a well-established field of research particularly for constrained devices. Memory is in particular subject to FA as it holds sensitive data and due to the fact that it can be accurately faulted using devices like laser [15] [16].

B. *Fault Injection Simulation*

While no standard testing approach can ensure resistance against all attacks, the physical fault injection is of great importance to characterize real fault effects on targeted chips. However, the cost of an efficient fault injection equipment is high (about 150,000 € for a standard laser fault injection platform [11]). In addition to that, the process is risky (the target chip may be damaged) and time-consuming. Furthermore, physical fault injection has low controllability and observability over faults and over the collected data which reduces its effectiveness. On the other side, because the effects of faults manifest themselves at the software level, faults have been modelled in the literature.

1) *Fault models*

Fault effects on microcontroller basically consist on tempering the value of a single or multiple bits. The fault distribution (number of bits) depends on the type of attack, the fault injection equipment and its accuracy. Fault targets either control or data flow.

- *Control flow :* To model a fault on the control flow (Program Memory), two fault models are commonly considered: (1) instruction corruption and (2) instruction skipping [17]. Based on bits tampering, the fault model size depends on the microcontroller's architecture.

- *Data flow:* In literature, data flow fault refers to fault on the processed data. Such faults can be modelled by memory corruption fault model with a granularity of bit, byte and multiple bytes [18].

With identified fault models, evaluation of robustness against fault attacks has been made easier and optimized under simulation. In fact, two families of simulation techniques have been developed to supplement the physical fault injection mechanism: Emulation-based techniques and Simulation-based techniques. Such techniques try to replicate the effect of the physical fault injection.

2) *Fault injection simulation techniques*

Emulation-based fault injection techniques are based on using targeted hardware implemented on FPGA instead of a computer-based simulation. This technique frees the simulation from assumptions on fault models and allows rapid attacks. Based on either reconfiguration [19] or instrumentation [20], those techniques combine the speed of physical fault injection and the flexibility of simulation. However, despite operating very closely to the real target, they remain physically different.

On the other side, Simulation-based fault injection techniques can be divided into three categories. First, the Full-software simulation, a technique that doesn't use specific target architecture and considers complex fault models associated with powerful attacks scenarios and where formal tools are generally used [21]. Second, the Hardware-aware simulation , a technique that relies on specific hardware models accuracy and needs large development effort and long simulation times [22][23]**.** The third category, the one on the scope of this paper, is known as the Software-Implemented Fault Injection (SWIFI) techniques. SWIFI techniques are a wide and diverse set of software mechanisms and tools dedicated for testing vulnerability to faults through software. SWIFI techniques are known to be flexible and to have good observability and controllability over injection of faults making them reliable solutions to evaluate the countermeasures against FA. SWIFI can be either used at compile-time or at run-time. For a broader review of fault injection techniques and tools, including SWIFI techniques, the reader may refer to the up-to-date surveys [24], [25] and [26].

Embedding a fault injection simulator allows simulating faults on the real target running real software and is advantageous over other simulation techniques as it releases the simulator from the assumption on the target model. The task is particularly challenging due to the limited software and hardware resources available in a chip to run the mechanism while providing a realistic fault injection. The realism of the fault attack simulation is also dependent on the fault model accuracy.

The aim of this work is to propose an embedded program-level, portable and run-time mechanism of fault injection simulation for embedded processors and microcontrollers. The mechanism takes advantage of an RTOS to manage a run-time attack scenario when associated with a computer-based cryptanalysis program in Matlab. The remainder of the paper is organized as follows: A review and discussion of recently embedded fault injection simulators and similar mechanisms from literature, is presented in section II, followed by the proposed mechanism in section III. A test case and results are given in section IV to validate the proposed approach. Finally, section V summarizes the paper and draws future works.

II. RELATED WORKS

Embedded Fault simulators are generally written inmachine language (i.e. assembler) but the higher levellanguage could still be used as a support. In [27], authors carried out a simulation of fault injections attack after experimentally characterizing fault effects on control flow (instruction skipping and instruction corruption fault models) on a 32-bit microcontroller (ARMv7 core). Close to the hardware level, the fault models proved to be realistic. A semi-manual embedded simulation process was applied in debugging mode using a specific program based on Keil UVSOCK library[28]. Due to writing protection on Flash Memory, the latter's content was shifted to RAM. The fault injection process needed frequent stops and restarts of the processor, which altered measuring correct latencies within thetarget and run-time fault injection simulation.

In [29], an architecture-specific fault injection attack strategy was presented. The attack consists of modifying a load

instruction to load externally controlled values into the Program Counter (PC). Authors target is a feature rich ARMv7-based SoC (1GHz, DDR3/400Mhz RAM, Gigabit Ethernet, etc.) in which an operating system (Ubuntu 12.04) was installed to simulate the fault injection. The corruption of the program's instructions, running on a shell code inside a Linux application, was done the function of flipped bits in a Python wrapper. The run-time simulation was very fast and fault injection results were immediately printed on the terminal. However, the embedded simulator largely depends onthe SoC and the OS features, making the reproduction of the same mechanism on lower range hardware yet to demonstrate.

In [30], authors presented an Embedded Fault Simulator (EFS) dedicated to smartcards. The simulator consists of two complementary modules: one, written in C, to integrate the EFS as a service in the smartcard OS, the other, in assembly, isthe fault injection mechanism. The reachable fault models of the EFS are: instruction skipping, instruction alternation and data modification with a granularity of bit, byte and word. TheEFS is highly configurable for each fault model. The injection mechanism basically relies on predetermined interruption routines triggered by the microcontroller's timer. The EFS was tested on an ARMv7-M architecture. Although having many exploitation possibilities, the EFS injection mechanism is architecture-dependent and relies on timer's availability within the target, limiting therefore its portability.

On the other side, high-level fault injection simulation is generally praised for its speed compared to low-level counterpart and benefits from using the programming language to inject faults. In [31], authors presented a methodology to secure any application with formally verified countermeasures at C-level automatically. To evaluate the efficiency of the proposed methodology, a computer-based simulation of a realistic C-level fault attack (jump attack), using a Python C parser was conducted. The simulation was much faster than equivalent assembly-level exhaustive jump fault attack on an AES encryption function. The latter took 3 weeks on ARMv7-M architecture, using Keil ARM-MDK compiler and Keil simulator. The countermeasures added upon C-level fault injection campaign enabled to defeat 60% of the attacks at the assembly level. The C-level fault model doesn't have the same fault coverage of assembler-level but the number of covered attacks-to-time (or to-test cases) ratio was much higher [32] and helpful to detect many weaknesses at source code level.

Simulating a fault attack generally requires three software components: A simulator of the target architecture, a fault injection mechanism, and a cryptanalysis program to provide the fault parameters (time, location, fault granularity, etc.) to the injection mechanism and process the received faulted outputs according to the chosen FA scenario.

In the reported works, few details were given on the software cryptanalysis process associated with the fault injection mechanism.

This could be explained by the fact that some works were limited to demonstrate the practicality of the approach without running related cryptanalysis. Also, this is due to the fact that the addressed attack models where control flow attack, which doesn't generally require processing several ciphers [33] [34]. However, for Data flow attacks, considering that the number of samples needed for a successful attack is not negligible, running an on-target data flow attack simulation requires efficient communication between the target-based injectionmechanism and the computer-based cryptanalysis program. While many works concentrated on simulating attacks on the control flow, data flow wasn't much addressed. I fact, despite being more complex and expensive to set up physically, compared to the control flow attack, the data flow attacks are still feasible using optical fault platform (laser), and recently using clock glitching [35] and voltage glitch [36] with different fault model granularities.

In this paper, an embedded fault injection simulation on the data flow was addressed. An ARM-based microcontroller (Cortex-M4 core) was used where an RTOS was embedded to manage the fault injection mechanism according to received parameters (fault location, corrupted data value, etc.) from a computer-based cryptanalysis program (in Matlab) applying an FA attack scenario.

III. Proposed Fault Injection Simulator

A. *Fault Injection Method*

A benefit from working on RTOS instead of developing all in application-level holds in the processing organization and portability of the code. In fact, several working tasks sharing access to data (writing or reading) can run through sharing mechanisms provided by the OS. In particular, FreeRTOS which is a class of RTOS designed to be small enough to run on a microcontroller although it is not limited to microcontroller applications. FreeRTOS, written in C, provides the core real time scheduling functionality, inter-task communication, timing and synchronization primitives [37]. Inaddition, unless low-level (assembly) calls are made in the program, the code will be portable between all supported architectures, hard core and soft core processors families (ARM Cortex-MX, Atmel AVR, Microchip PIC32MX, Free- scale, PowerPC Xilinx Microblaze, Altera NIOS II, etc.).

In this work, FreeRTOS was used on the targeted hardware, an STM32F4 MCU (ARM Cortex-M4 core), to provide run-time execution and flexibility to the fault injection mechanism and bridge it efficiently to the cryptanalysis program.

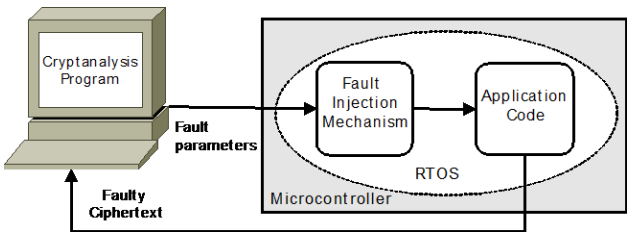

Fig. 1. Synoptic diagram of the FA platform

Because of the complex and time consuming computations involved in cryptanalysis, the latter is not embedded on the microcontroller and runs on a computer. Figure 1 shows a synoptic diagram of the complete Fault Analysis (FA)platform.

When setting up a fault injection environment, a Fault Injection Policy should be defined. Table 1 gives the fault injection policy followed in the FA platform.

TABLE I. FAULT INJECTION SIMULATION POLICY

Abstraction Level	C
Considered fault model(s)	Data corruption
Granularity	bit, byte
Fault location	Data flow
Injection time	Event and time triggered
Fault duration	Transient
Mean	Data replacement
Input data for the system	Random value

B. *Fault Injection and attack mechanism*

The simulator makes use of the RTOS to control the fault injection and associated attack. The simulator providesmemory data manipulation fault and temporal triggers. Those triggers are inserted with minimal modification on the target application, and there is no need for running it in debugging mode except the first run. In what follows, the steps used to set up the fault injection mechanism and run an attack are given.

1) Debug mode run: Check the sensitive data to be faulted (obtain memory addresses).

2) Golden run: It will serve to get the correct output for a reference plaintext.

3) Triggers insertion in the cryptographic code: The triggering code monitors a specific data depending on its value, the cryptographic code is suspended.

4) Fault parameters reception: data address, faultedvalue, etc. are received from the cryptanalysis program.

5) The fault Simulator starts the Application code (Cryptographic algorithm).

6) Application code suspension: The fault simulator suspends the Application code and injects a fault.

7) The cryptographic code is resumed.

8) The simulator sends the faulty outputs to the cryptanalysis program.

9) New fault parameters are received.

*10)*Check if the cryptanalysis program recovered the secret key, otherwise return to step 4.

C. *FreeRTOS threads management for fault injetion*

The FreeRTOS is a multitasking operating system using a scheduler to decide on the task to execute. At every interrupt from the system timer, the scheduler accord processing time to the highest priority task. In the proposed mechanism, the fault injection simulator was divided between three threads:

- Control Thread: Manages the communication with the cryptanalysis computer (fault parameters reception, faulty ciphertext sending, etc.).
- Injection Thread: The thread in charge of data corruption.
- Application Thread: Encapsulates the C code target of the fault injection.

The working of the simulator is based on a binary semaphore that synchronizes the three threads. The cryptographic code is encapsulated in the Application Thread. A representation of the working of the threads is depicted in Figure 2 where the steps 1-to-4 are explained as follows:

1) The Control Thread is the starting point of the fault injection, and is the thread with the highest priority. Upon receiving the fault parameters from a computer via UART, the Control Thread stores the fault parameters, releases the semaphore and suspends itself.

2) The Application Thread, having a lower priority, waits for the semaphore. Once obtained, Thread2 starts the cryptographic code. During its execution, the trigger monitors a change in a specific data and consequently releases the semaphore and suspends the Thread2. Furthermore, the extra code (monitor and suspend the thread) does not modify the targeted data location and allows resuming the Application execution from the suspended state.

3) The Thread3, i.e. the injection Thread, has the lowest Priority and obtains the semaphore after target code suspension. In this thread, the sensitive data is corrupted according to the received parameters. Then, Thread3 permutesthe priorities of Thread1 and Thread2 so that the latter obtains the next semaphore. Finally, Thread3 releases the semaphore and suspends itself.

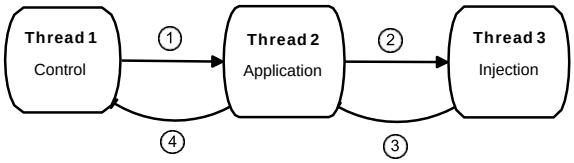

Fig. 2. Semaphore-based threads synchronization of the Simulator

4) The Thread2 resumes from where it was suspended with its sensitive data now corrupted. The calculations are done and a faulty ciphertext is generated. Then, the thread releases the semaphore and suspends itself.

At the next passage by the Control thread, the latter sends the faulty output to a computer via UART and waits for new parameters. Once received, the priorities of Thread1 and Thread2 have permuted again (original priorities are restored). Then, a new round of fault injection is started from step 1 until no new parameters are received meaning that the secret data (i.e. secret key) was successfully retrieved by the cryptanalysis program.

IV. Test Case and Analysis

The Simulator has been implemented on a development board (STM32F4 Discovery) build around the ARM Cortex-M4 processor. STM32 Family of microcontrollers features some integrity and safety mechanisms. In particular, for fault Injection attacks some hardware countermeasures exist like the Error Correction Code (ECC) and the Parity check. Both mechanisms, according to the constructor [38], ensure robust memory integrity and harden the protection against fault injection attacks. ECC protection is integrated with Flash memory controller while Parity Check is intended for the SRAM memory. However, such protections are only available in some chips (F0, F3, L0, L1 and L4 families). In another hand, software countermeasures against data corruption attack were successfully bypassed by multiple faults injections in [39]on an ARM Cortex-M3 using laser and in [35] on an ATmega163 microcontroller using clock glitching.

A. *Attack Scenario*

Dusart attack [34], a DFA attack on the popular and widely used Advanced Encryption Standard (AES) [40] was selected to be implemented in the platform. This attack demonstrates that using a fault on one byte anywhere between the 8th round MixColumn and 9th round MixColumn, an attacker would be able to retrieve the secret key using less than 50 faulty ciphertexts. The cryptanalysis program, i.e. the main part of the Dusart attack scenario, was written in Matlab based on the original algorithm [34]. As a target, an implementation of an AES-128 ECB encryption algorithm written in C and optimized for ARM architecture was used [41]. In AES-128, the "State" is a 4x4 array of coefficients in bytes (0-255) holding a portion of the data to be encrypted. The State goes through 4 transformations (SubBytes, ShiftRows, MixColumn and AddRoundKey) for 10 rounds (except the MixColumns operation which is not used in the 10th round) to generate the encrypted data.

In the Dusart attack, one of the bytes of the State array before the MixColumn transformation of the 9th round is replaced by a faulty value (Figure 3). The faulty byte will then be propagated by the MixColumn and spread over four bytes of the State. There is a linear relation between the four induced faults. For each byte, it is possible to find a set of possible value of induced fault, and then a set of possible values for the round key 10 (K_{10}). Finally, once K_{10} is found, it the entire secret key can be recovered.

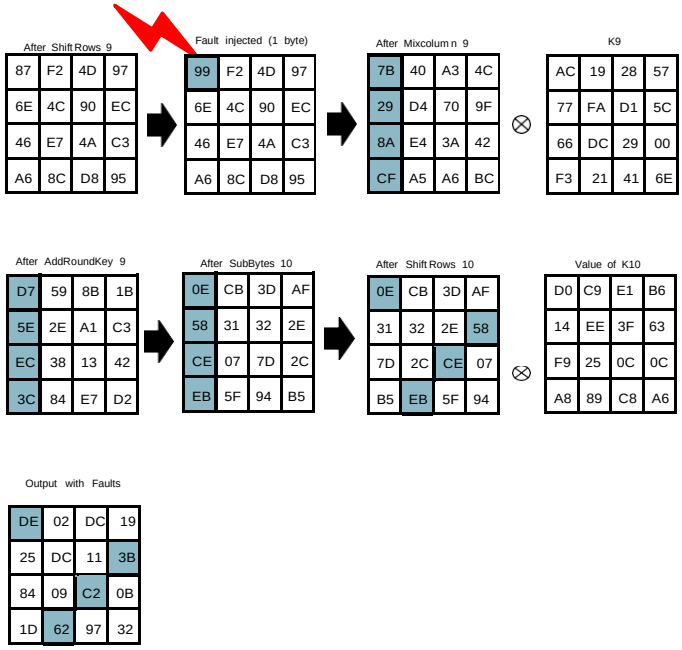

Fig. 3. Fault propagation in the State array in the Dusart attack[34]

According to the attack scenario, the State array is the target data of the fault injection. The Dusart attack on the FA platform was applied following the steps detailed in III.B. The attack simulation was performed using the setup that follows:

Computer-based cryptanalysis: A Matlab program ofDusart DFA running on an Intel i7-3770 at 3.40 GHz and connected to the microcontroller through a PL2303 UART Adapter.

Microcontroller: An STM32F407VG (Cortex-M4 ARM core). The fault injection trigger is a conditional statement on the round counter to inject faults in the 9th round and before the MixColumn transformation.

B. *Results and discussion*

Retrieving the correct 10th RoundKey required 50 random fault injections on array positions number: 1, 5, 9 and 13 and took 490 seconds (~8:12 min). The Fault mechanism, including the Application code, occupied 18 Kbytes of Flash memory and 14 Kbytes of RAM. This represents only 1.8 % of the totalflash memory and 7% of available RAM.

Table 2 shows a comparison of the proposed fault injection simulator with the similar fault injection mechanisms that were discussed in section II. The proposed simulator has the advantage of portability to different architectures and the very low memory overhead that comes with it. Also, using a high-level programming language brings a significant flexibility to the simulator though at the expense of covering control flow attacks which are generally modelled in assembler. One cryptanalysis algorithm was tested, but other algorithms targeting data flow are also applicable like those proposed by Giraud [33] and Tunstall [42], among others.

TABLE II. COMPARAISON OF THE EMBEDDED FAULT INJECTION SIMULATOR WITH SIMILIAR WORKS

Reference	Abstraction Level	Fault Target	Runtime attack	Granu-larity	Embedded	OS Support	Tool set	Target core	Architecture specific
[27]	Low Level[1]	Control flow	No (manual)	instruction	No	-	Program based on Keil UVSOCK library	ARMv7m Cortex-M3	Yes
[31]	High Level[2]	Control flow	Yes	C Line	No	-	Keil ARM-MDK compiler and simulator	ARM-v7m	No
[30]	High Level[2] Low Level[1]	Data flow, Control flow	Yes	Byte	Yes	Smart-Card OS	Embedded as an OS service.	ARMv7-M Cortex-M4	Yes (ASM part)
[29]	Low Level[1]	Control flow	No	instruction	Yes	Ubuntu 12.04	ARM Simulator (C+ Python + shellcode)	ARMv7-A	Yes
This Work	High Level[2]	Data flow	Yes	C variable	Yes	Free-RTOS	RTOS + Matlab Cryptanalysis	ARMv7-M Cortex-M4	No

[1]Assembler,[2]C Language

V. CONCLUSION

In this paper, a novel high-level embedded simulator for fault injection attacks on microcontrollers was proposed. The simulator relies on a real-time operating system (FreeRTOS) to accurately inject simple or multiple faults on data flow and carries out a complete attack scenario with the support of a computer-based cryptanalysis program. The proposed mechanism was tested for fault attack on data flow (Dusart attack) and can be applied to other attack scenarios. A number of improvements can still be made to the simulator like how to monitor and tamper sensitive data when using a proprietary code with a read-out protection. Another prospect of this work could be investigating on high level simulation of control flow fault injection attack with a realistic fault model. Similar to [30], combining FA attack with SCA to bypass the embedded hardware countermeasure can be investigated as well. A different perspective of this work could be in countermeasure integration. In fact, the OS can be used to integrate software countermeasures like dummy threads execution to mask the power traces or other physical signals that may leak exploitable information about the secret key.

REFERENCES

[1] P. Kocher, J. Jaffe, and B. Jun, "Differential Power Analysis," in Proceedings of the 19th Annual International Cryptology Conference on Advances in Cryptology, 1999, pp. 388–397.

[2] W. van Eck and Wim, "Electromagnetic radiation from video display units: An eavesdropping risk?," Computers & Security, vol. 4, no. 4, pp. 269–286, Dec. 1985.

[3] P. C. Kocher, "Timing Attacks on Implementations of Diffie-Hellman, RSA, DSS, and Other Systems," in Proc. of Advances in Cryptology (CRYPTO 1996), Lecture Notes in Computer Scicence 1109, 1996, pp. 104–113.

[4] D. Boneh, R. A. Demillo, and R. J. Lipton, "On the importance of checking cryptographic protocols for faults," in International Conference on the Theory and Applications of Cryptographic Techniques, 1997, vol. 1233, pp. 37–51.

[5] E. Biham and A. Shamir, "Differential fault analysis of secret key cryptosystems," in Proc. of Advances in Cryptology (CRYPTO 1997), Lecture Notes in Computer Scicence, 1997, vol. 1294, pp. 513–525.

[6] M. Joye and M. Tunstall, Fault Analysis in Cryptography. Berlin, Heidelberg: Springer Berlin Heidelberg, 2012.

[7] D. Karaklajic, J.-M. Schmidt, and I. Verbauwhede, "Hardware Designer's Guide to Fault Attacks," IEEE Transactions on Very Large Scale Integration (VLSI) Systems, vol. 21, no. 12, pp. 2295–2306, Dec. 2013.

[8] N. Theissing, D. Merli, M. Smola, F. Stumpf, and G. Sigl, "Comprehensive Analysis of Software Countermeasures against Fault Attacks," in Design, Automation & Test in Europe Conference, 2013, pp. 404–409.

[9] G. Agosta, A. Barenghi, G. Pelosi, and M. Scandale, "The MEET Approach: Securing Cryptographic Embedded Software Against Side Channel Attacks," IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems, vol. 34, no. 8, pp. 1320–1333, Aug. 2015.

[10] T. Korak and M. Hoefler, "On the effects of clock and power supply tampering on two microcontroller platforms," in Proceedings - 2014 Workshop on Fault Diagnosis and Tolerance in Cryptography, FDTC 2014, 2014, pp. 8–17.

[11] J. Breier and D. Jap, "Testing Feasibility of Back-Side Laser Fault Injection on a Microcontroller," in Proceedings of the WESS'15: Workshop on Embedded Systems Security - WESS'15, 2015, pp. 1–6.

[12] N. Moro, A. Dehbaoui, K. Heydemann, B. Robisson, and E. Encrenaz, "Electromagnetic fault injection: Towards a fault model on a 32-bit microcontroller," in Proceedings - 10th Workshop on Fault Diagnosis and Tolerance in Cryptography, FDTC 2013, 2013, pp. 77–88.

[13] N. Moro, K. Heydemann, A. Dehbaoui, B. Robisson, and E. Encrenaz, "Experimental evaluation of two software countermeasures against fault attacks," in Proceedings of the 2014 IEEE International Symposium on Hardware-Oriented Security and Trust, HOST 2014, 2014, pp. 112–117.

[14] J. Breier and X. Hou, "Feeding Two Cats with One Bowl: On Designing a Fault and Side-Channel Resistant Software Encoding Scheme," in RSA Conference Cryptographers' Track (CT-RSA 2017), 2017.

[15] M. Agoyan, J. M. Dutertre, A. P. Mirbaha, D. Naccache, A. L. Ribotta, and A. Tria, "How to flip a bit?," in Proceedings of the 2010 IEEE 16th International On-Line Testing Symposium, IOLTS 2010, 2010, pp. 235–239.

[16] B. Selmke, S. Brummer, J. Heyszl, and G. Sigl, "Precise Laser Fault injections into 90nm and 45nm SRAM-cells," in International Conference on Smart Card Research and Advanced Applications (CARDIS 2015), 2015, pp. 193–205.

[17] N. Moro, A. Dehbaoui, K. Heydemann, B. Robisson, and E. Encrenaz, "Electromagnetic fault injection: Towards a fault model on a 32-bit microcontroller," in Proceedings - 10th Workshop on Fault Diagnosis and Tolerance in Cryptography, FDTC 2013, 2013, pp. 77–88.

[18] M. Agoyan, J. M. Dutertre, A. P. Mirbahat, D. Naccache, A. L. Ribottat, and A. Tria, "Single-bit DFA using multiple-byte laser fault injection," in 2010 IEEE International Conference on Technologies for Homeland Security, HST 2010, 2010, pp. 113–119.

[19] L. Sterpone and M. Violante, "A new partial reconfiguration-based fault-injection system to evaluate SEU effects in SRAM-based FPGAs," in IEEE Transactions on Nuclear Science, 2007, vol. 54, no. 4, pp. 965– 970.

[20] S. A. Hwang, J. H. Hong, and C. W. Wu, "Sequential circuit fault simulation using logic emulation," IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems, vol. 17, no. 8, pp. 724–736, 1998.

[21] [21] M. Puys, L. Rivière, J. Bringer, and T. H. Le, "High-level simulation for multiple fault injection evaluation," in Data Privacy Management, Autonomous Spontaneous Security, and Security Assurance, 2015, vol. 8872, pp. 293–308.

[22] A. Papadimitriou, M. Tampas, D. Hely, V. Beroulle, P. Maistri, and R. Leveugle, "Validation of RTL laser fault injection model with respect to layout information," in Proceedings of the 2015 IEEE International Symposium on Hardware-Oriented Security and Trust, HOST 2015, 2015, pp. 78–81.

[23] S. Nimara, A. Amaricai, O. Boncalo, and M. Popa, "Multi-Level Simulated Fault Injection for Data Dependent Reliability Analysis of RTL Circuit Descriptions," Advances in Electrical and Computer Engineering, vol. 16, no. 1, pp. 93–98, 2016.

[24] M. Kooli and G. Di Natale, "A survey on simulation-based fault injection tools for complex systems," in Proceedings - 2014 9th IEEE International Conference on Design and Technology of Integrated Systems in Nanoscale Era, DTIS 2014, 2014, pp. 1–6.

[25] M. Kooli, A. Bosio, P. Benoit, and L. Torres, "Software testing and software fault injection," in 2015 10th International Conference on Design & Technology of Integrated Systems in Nanoscale Era (DTIS), 2015, pp. 1–6.

[26] R. Piscitelli, S. Bhasin, and F. Regazzoni, "Fault Attacks, Injection Techniques and Tools for Simulation," in Hardware Security and Trust, Cham: Springer International Publishing, 2017, pp. 27–47.

[27] N. Moro, "Securing assembly programs against attacks on embedded processors (Sécurisation de programmes assembleur face aux attaques visant les processeurs embarqués)," PhD Thesis, UPMC (France), 2014.

[28] Keil, "Application Note 198: Using the uVision Socket Interface," 2016. [Online]. Available: http://www.keil.com/appnotes/docs/apnt_198.asp. [Accessed: 01-May-2017].

[29] N. Timmers, A. Spruyt, and M. Witteman, "Controlling PC on ARM Using Fault Injection," in 2016 Workshop on Fault Diagnosis and Tolerance in Cryptography (FDTC), 2016, pp. 25–35.

[30] L. Rivière, J. Bringer, T. H. Le, and H. Chabanne, "A Novel Simulation Approach for Fault Injection Resistance Evaluation on Smart Cards," in 2015 IEEE 8th International Conference on Software Testing, Verification and Validation Workshops, ICSTW 2015 - Proceedings, 2015, pp. 1–8.

[31] J. F. Lalande, K. Heydemann, and P. Berthomé, "Software countermeasures for control flow integrity of smart card c codes," in 19th European Symposium on Research in Computer Security (ESORICS), 2014, vol. 8713 LNCS, no. PART 2, pp. 200–218.

[32] P. Berthome, K. Heydemann, X. Kauffmann-Tourkestansky, and J.-F. Lalande, "High Level Model of Control Flow Attacks for Smart Card Functional Security," in 2012 Seventh International Conference on Availability, Reliability and Security, 2012, pp. 224–229.

[33] C. Giraud, "DFA on AES," in Proceedings of the 4th international conference on Advanced Encryption Standard, 2004, pp. 27–41.

[34] P. Dusart, G. Letourneux, and O. Vivolo, "Differential Fault Analysis on A.E.S," in Applied Cryptography and Network Security, First International Conference, ACNS 2003. Kunming, China, October 16-19, 2003, Proceedings, 2003, vol. 2846, pp. 293–306.

[35] S. Endo, N. Homma, H. Yu-ichi, J. Takahashi, H. Fuji, and T. Aoki, "An Adaptive Multiple-Fault Injection Attack on Microcontrollers and a Countermeasure," IEICE Transactions on Fundamentals of Electronics, Communications and Computer Sciences, vol. E98.A, no. 1, pp. 171– 181, 2015.

[36] C. O'Flynn, "Fault Injection using Crowbars on Embedded Systems," Cryptology ePrint Archive, Report 2016/810, 2016.

[37] [37] "FreeRTOS." [Online]. Available: http://www.freertos.org/index.html. [Accessed: 20-Apr-2017].

[38] A. Programming, "Announcing Stack Overflow Documentation Strategies to develop STM32 in application programming," pp. 8–9, 2016.

[39] E. Trichina and R. Korkikyan, "Multi fault laser attacks on protected CRT-RSA," in Fault Diagnosis and Tolerance in Cryptography - Proceedings of the 7th International Workshop, FDTC 2010, 2010, pp. 75–86.

[40] NIST, "Announcing the Advanced Encryption Standard (AES)," Processing Standards Publication 197, 2001.

[41] Kokke, "Tiny AES128 in C," GitHub repository, 2016. [Online]. Available: https://github.com/kokke/tiny-AES128-C. [Accessed: 11-Nov-2016].

[42] M. Tunstall, D. Mukhopadhyay, and A. Subidh, "Differential Fault Analysis of the Advanced Encryption Standard using a Single Fault," in Proceedings of WISTP'11, 2011, pp. 224–233.