Faulty Clock Detection for Crypto Circuits Against Differential Fault Analysis Attack

Pei Luo and Yunsi Fei

Email:silenceluo@coe.neu.edu, yfei@ece.neu.edu
Department of Electrical and Computer Engineering
Northeastern University, Boston, MA 02115

Abstract. Clock glitch based Differential Fault Analysis (DFA) attack is a serious threat to cryptographic devices. Previous error detection schemes for cryptographic devices target improving the circuit reliability and cannot resist such DFA attacks. In this paper, we propose a novel faulty clock detection method which can be easily implemented either in FPGAs or integrated circuits to detect the glitches in system clock. Results show that the proposed method can detect glitches efficiently while needs very few system resource. It is also highly reconfigurable to tolerant clock inherent jitters, and will not involve complex design work for different processing technologies.

Keywords: Clock glitch detection, AES, differential fault analysis, side-channel attacks

1 introduction

Cryptosystems that provide encryption, decryption, and hashing functions have become the security engine of many critical systems and infrastructure. However, their implementations, whether in software or hardware, are vulnerable to side-channel attacks, which exploit parametric information about cryptographic operations including power consumption, timing information, and electromagnetic emanations to retrieve the secret key. Differential Fault Analysis (DFA) is a kind of side-channel attack, which intentionally injects transit faults into a cryptosystem and analyzes the difference between the correct and fault outputs to infer the key. DFA was first invented by Biham et. al. on the block cipher Data Encryption Standard (DES) [1]. It was later applied to AES and only two pairs of correct and faulty ciphertexts are required to break the cipher [2]. In [3], the fault model is relaxed to be a random fault anywhere in one of the four diagonals of the state matrix at the input of the eighth round of AES. It is shown that even if the fault induction corrupts two or three diagonals, two to four faulty ciphertexts are enough to retrieve the correct key.

The commonly used physical fault injection methods are clock glitching, supply voltage variation, and laser radiation. Clock glitching is the most practical and low-cost fault injection method, and therefore clock glitch based DFA is more controllable and becomes a real security threat. Work in [3] uses clock glitches to inject faults into cryptographic devices. Work in [4] also demonstrates the effectiveness of frequency injection attacks on both a microcontroller and a FPGA chip.

Some high-level protection methods for cryptographic systems employ error detection and correction codes or redundancy [5,6] to improve the reliability of circuits, but cannot resist clock glitch-based DFAs. Other protection methods try to detect the fault injection sources and therefore prevent DFAs [7].

In [7], a non-logic buffer-based delay chain is inserted, by monitoring the delay along the delay chain, a possible clock glitch based DFA can be detected. In [8], the authors design a circuit with a master counter and a slave counter clocked by the clock signal, and their results are compared to determine whether the difference is at least thus to detect the clock glitch.

In this work, we propose a new clock glitch detection method which makes good use of the existing modules in ICs and FPGAs. The proposed method can be easily implemented without considering the complex process technology like in [7]. Meanwhile, comparing with [8], our method is also highly reconfigurable and can be easily configured according to different requirements of precision and system variations. What's more, compare with previous methods, our scheme can make good use of the existing resources in FPGAs and can also be implemented efficiently in ASICs. Synthesis results show that our

proposed method has much lower resource overhead comparing with previous protection schemes like in [5,6,9].

The rest of the paper is organized as follows. In Section 2, we revisit the basics of DFA attacks and previous protection methods. In Section 3, we describe the proposed detection method and discuss its advantages over the previous schemes. In Section 4, we show the synthesis and simulation results of the proposed method, and compare it with previous protection schemes. Finally we conclude the paper in Section 5.

2 Background

In this section, we briefly introduce DFAs, and review the existing work on run-time error detection of crypto circuits.

2.1 Differential Fault Analysis Attacks

As fault analysis is dependent on the algorithm of cryptosystem, we take AES [10] as example. AES-128 algorithm has ten rounds of computation, with the first nine rounds consisting of four steps, SubByte, ShiftRow, MixColumn, and AddKey, and the last round just three steps - without MixColumn. Previous work has demonstrated that if faults can be injected between the MixColumn operation in round eight and SubByte operation in round ten, DFA based on this fault model can recover the last-round key with just two pairs of correct and faulty ciphertexts.

Fig. 1 depicts the general DFA process. Before introducing faults, the attacker first gets a pair of plaintext and correct ciphertext (P, C), where P is a randomly chosen plaintext and C is the corresponding ciphertext with the last round key as K_{10}. The attacker then injects faults into the cryptographic system and obtain a faulty ciphertex, C', under the same plaintext P. A fault caused by clock glitch that falls before the last round computation would result in a faulty input P'_{10} to the last round, instead of correct one P_{10}.

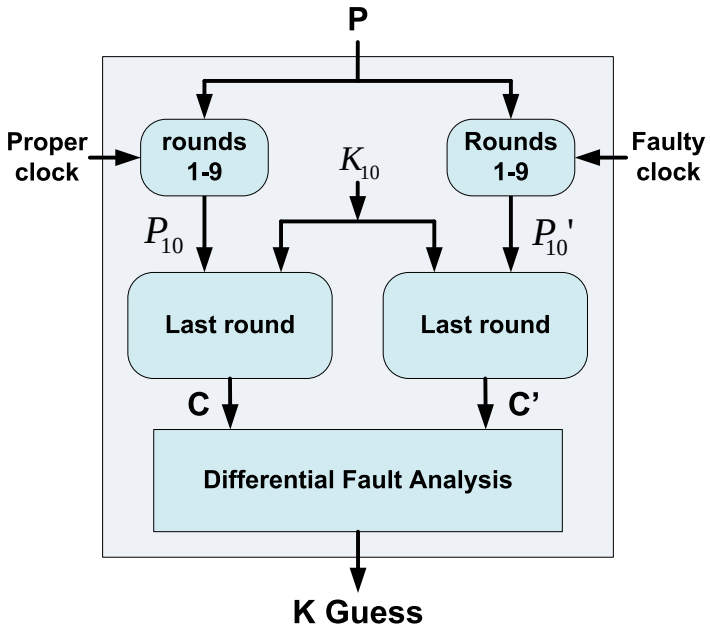

Fig. 1. Differential fault analysis model

By examining the difference between C and C', the attack can identify the location of the faulty byte in the 16-byte AES state, denoted as i. The attacker then makes an assumption of the last round key byte K_i^{10}. For any key guess K_i^{10}, the attacker can inversely compute the last round input P_{10} and P'_{10} from the ciphertext outputs.

$$\begin{cases} P_{10} = AES_{10}^{-1}(C, K_i^{10}) \\ P'_{10} = AES_{10}^{-1}(C', K_i^{10}) \end{cases}. \tag{1}$$

Certain characteristics of the difference between P_{10} and P'_{10} can be used to extract the correct K_i^{10}. For clock glitch fault injection, illegal clock will cause a setup-time violation since flip-flops are triggered before the output signals are fixed to a correct value. In [3], the authors show that faults may happen in either single bytes or multiple bytes, or even in multiple diagonals. They also demonstrate that relationship among faulty bytes can be used to recover the last round key bytes. Other fault injection methods would incur more complex fault models and there will be different characteristics among faulty bytes and such characteristics can be used for attacks [2,11].

2.2 DFA Countermeasures

Previous works against DFAs use redundancy or error detection codes to capture faults at run-time [5,6,9]. This kind of protection schemes can detect any injected faults inside the circuits when the results of the predictor based on the codes (or a redundant copy) and the compressor of the original copy do not match. The methods are useful when the faults fall on either the original circuit or the protection circuit, or the faults injected in both parts result in different errors. However, faults injected by clock glitches fall on the common input of these two parts and would affect the two circuits simultaneously to generate the same outputs. It is highly possible that the faults caused by clock glitches will escape these high-level error detection methods.

For the scheme proposed in [7], a non-logic buffer-based delay chain is inserted into the original design with propagation delay (T_d) predefined during design phase and $1/T_d$ indicates the upper bound of the clock frequency of the crypto circuit. If the circuit runs too fast (i.e. the applied clock frequency is faster than $1/T_d$), timing errors might occur. Thus by monitoring the result on this delay chain, a possible clock glitch based fault attack might be detected. For this design, the problem is that it involves complex design work for different process technology and crypto circuits, and the system frequency cannot be changed after production.

3 Proposed Clock Glitch Detection Method

In this section, we propose a new clock glitch detection method to resist glitch based DFAs. The main idea is that glitches make the clock irregular, and we design a circuit to monitor the clock pulses and detect such irregularity at run-time.

Denote the system clock used in the cryptographic circuit as *clock*. To measure the width of clock pulses, we introduce another higher-frequency clock source *clk*. The structure of the proposed scheme is shown in Fig. 2.

The *Width Counters* take *clock* as the input signal and *clk* as the triggering clock. The counters measure the width of the high pulse and low pulse of *clock* in number of *clk* cycles and store the results in two *Width Registers*, one for high and the other for low. Considering two consecutive cycles of *clock*, denote the *Width Counters'* result as n_0^H for the first logic 1, which means the number of cycles of *clk* while $clock = 1$, and define n_0^L the number of cycles of *clk* for $clock = 0$ in the first cycle. Similarly, use n_1^H and n_1^L to denote the width of the logic 1 and logic 0 of the following cycle, shown as in Fig. 3.

If the target *clock* and the reference *clk* are both stable, then:

$$\begin{cases} n_0^L = n_1^L \\ n_0^H = n_1^H \end{cases}. \tag{2}$$

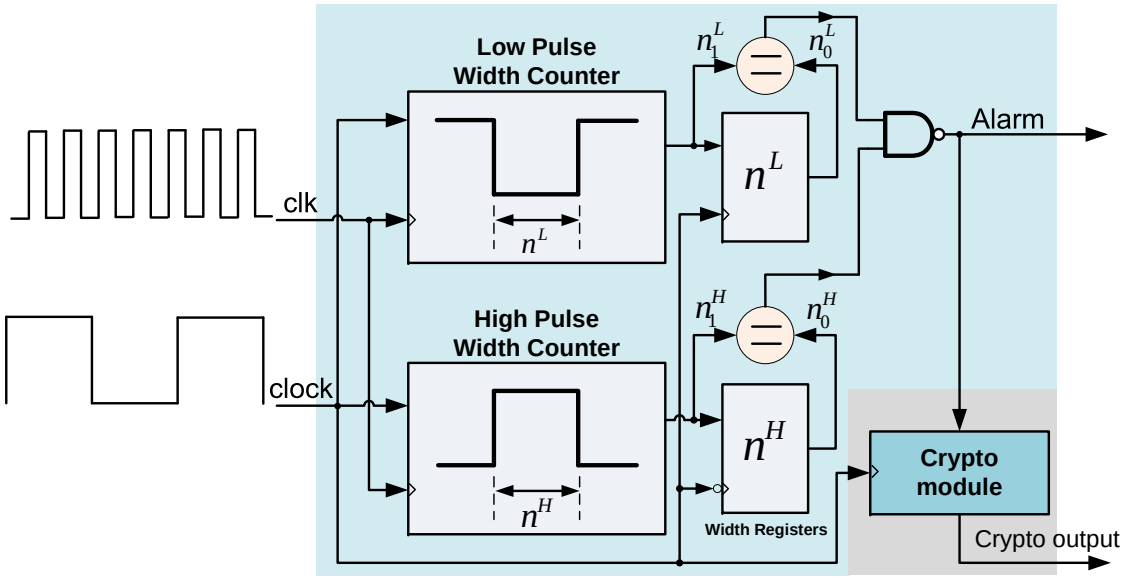

Fig. 2. The block diagram of the clock glitch detection circuit

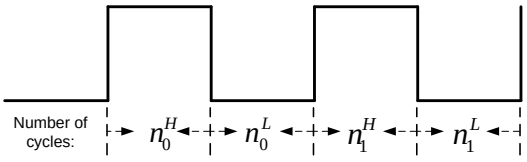

Fig. 3. Counter of the clock glitch detection circuit

For *clock* with 50% duty cycle, we have:

$$n_0^L = n_1^L = n_0^H = n_1^H. \tag{3}$$

If the clock is not constant, the width of logic '0' or logic '1' of *clock* is varying. Such difference between two consecutive clocks can be used to detect glitches in the system clock. If there is a glitch in *clock* shown as in Fig. 4. Then it's obvious that:

$$\begin{cases} n_0^L \neq n_1^L,\ n_1^L \neq n_2^L \\ n_0^H \neq n_1^H,\ n_1^H \neq n_2^H \end{cases}, \tag{4}$$

thus the glitch is detected and *Alarm* triggered.

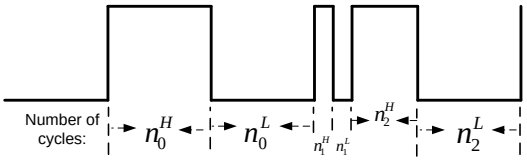

Fig. 4. When glitch happens in *clock*

3.1 Implementation of the Proposed Scheme

One key point of the design is to have a stable high frequency reference clock source *clk*. For FPGA and other embedded systems, clock generator and phase-locked loop (PLL) can be designed inside the devices in the design phase. An alternative method is to use existing resources. Take FPGA for example, a digital clock manager (DCM) or a PLL can be used to generate higher frequency with *clock* as the input clock. The Digital Clock Manager (DCM) primitive in Xilinx FPGA is used to implement delay locked loop, digital frequency synthesizer, digital phase shifter, or a digital spread spectrum [12]. The PLL in FPGA is used to generate multiple clocks with defined phase and frequency relationships to a given input clock [13,14].

For example, there are up to 12 DCM modules in Virtex 5 FPGAs and each can be used to generate utmost 32× higher frequency, and cascaded DCMs can be used to generate even higher frequency output [15]. Assume one DCM module is used to generate 32× frequency *clk*, then for stable *clock*, $n_0^L = n_1^L = n_0^H = n_1^H = 16$.

For PLL and DCM, if there is a glitch in *clock*, the glitch will be detected and meanwhile the frequency generation module will lose $LOCK$ and some operations are needed to reset the module [12,13,14]. The details will be explained and simulated in Section 4.

3.2 Advantage of the Proposed Scheme

Compared to the previous faulty clock detection schemes, our proposed scheme has several advantages. Firstly, it has a low resource requirement. It can be easily implemented in FPGAs and ICs, only another higher frequency clock source or a clock management module (PLL, etc) is needed. Secondly, the proposed scheme is easily configurable according to the precision requirement of the system. The method in [7] requires to insert a non-logic buffer-based delay chain to the circuits and the delay cannot be changed after production. The delay and the delay chain design are also technology dependent and therefore it increases the design difficulty and complexity. Thirdly, it is easy to control the frequency of reference clock *clk*, and higher precision can be achieved by using higher frequency *clk*. What's more, our scheme also has configurable threshold of clock jitter by controlling the comparator. Assume that the clock has slight variations and the width of *clock* is not constant, we set the comparator to tolerant different between n_0^L and n_1^L (and the difference between n_0^H and n_1^H) below λ, i.e., the variation is not caused by clock glitching and therefore is acceptable:

$$\begin{cases} |n_0^L - n_1^L| < \lambda \\ |n_0^H - n_1^H| < \lambda \end{cases}. \tag{5}$$

For instance, assume the frequency of *clk* is 32× of *clock*, we can set $\lambda = 2$ such that the variation of *clock* can be at most the width of 2×*clk*. Higher precision can be achieved by using higher frequency *clk* and λ can be chosen according to the sensitivity level requirement and clock jitter.

4 Implementation and simulation results

To verify the functionality of the proposed scheme, we use Virtex-5 FPGA and its internal DCM [12,13] to implement the proposed scheme . The target *clock* is running at 5 MHz and its duty cycle is 75%. We use DCM to generate the higher frequency clock *clk* with the frequency 160 MHz.

The simulation result is shown as in Fig. 5, with an glitch added in *clock*. The results show that the proposed scheme can detect glitches efficiently, with an alarm triggered when the glitch is detected. Meanwhile, the proposed scheme is very robust, which can recover the monitoring higher frequency clock rapidly after losing $LOCK$ of the target clock.

Details of the simulation result are shown as in Fig. 6. For the above simulation settings, $n_0^H = n_1^H = 24$ and $n_0^L = n_1^L = 8$ when there is no glitch. The register *pre_high_cnt*(n_0^H) holds value 24 and the value of *pre_low_cnt*(n_0^L) is 8 if the system clock is stable. When clock glitch happens, the register of the current result of n_1^H is 17 which is smaller than 24, and thus the glitch is detected.

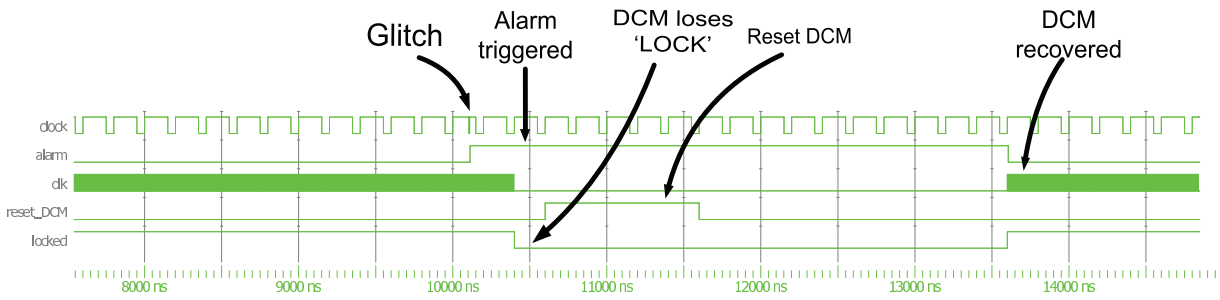

Fig. 5. Simulation result of the proposed scheme

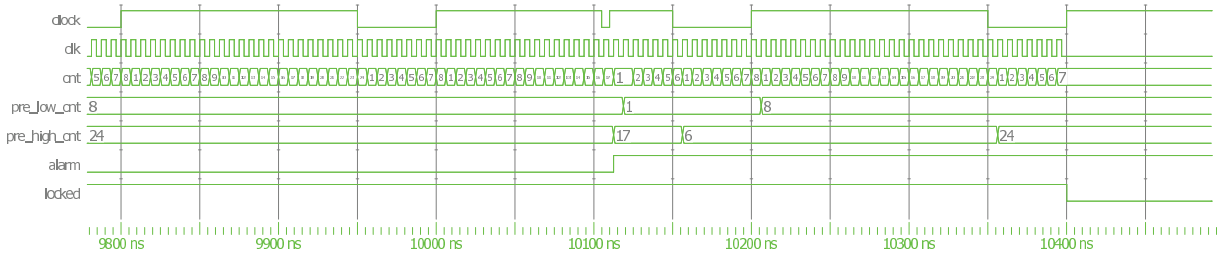

Fig. 6. Detailed results of the counter registers

To evaluate the implementation overheads of the proposed glitch detection scheme, we implement a glitch detection scheme based on the original AES using PLL in Verilog and synthesize it in Cadence Encounter RTL Compiler with the Nangate 45nm Opencell library version v2009_07. The designs were placed and routed using Cadence Encounter. The latency, the area overhead of the protection schemes were estimated using Concurrent Current Source (CCS) model under typical operation condition assuming a supply voltage of 1.1V and a temperature of 25 Celsius degree. The synthesis results are shown in Table 1.

Table 1. Overhead Evaluation of the Proposed Detection Method

Circuit	Area (um^2)	Power (mW)
$Original AES$	13987.9	20.4
Proposed	14392.5	22.8

The results show that the proposed glitch detection scheme incurs 2.9% area overhead and consumes 12% more power than the original AES implementation. Comparing with the scheme proposed in [7] with only about 0.47% area overhead, our proposed method consumes a little more area. But for FPGAs, our scheme can make good use of internal DCM and PLL modules. What's more, our method can be very easy to implement while the scheme in [7] involves complex design work for different FPGA families and IC process technologies.

5 Conclusion

In this paper, we propose a simple method to detect clock glitches and the results show that the proposed scheme can detect glitches in clock efficiently while needs much fewer resource overhead than the previous

protection schemes. Compared with previous schemes, the proposed scheme is designed specifically for clock glitch detection and it is highly reconfigurable. The proposed scheme involves no complex work in design phase for different process technology.

References

1. E. Biham and A. Shamir, "Differential fault analysis of secret key cryptosystems," in *Advances in CryptologyCRYPTO'97*. Springer, 1997, pp. 513–525.
2. G. Piret and J.-J. Quisquater, "A differential fault attack technique against SPN structures, with application to the AES and KHAZAD," in *Cryptographic Hardware and Embedded Systems-CHES 2003*. Springer, 2003, pp. 77–88.
3. D. Saha, D. Mukhopadhyay, and D. R. Chowdhury, "A diagonal fault attack on the Advanced Encryption Standard," *IACR Cryptology ePrint Archive*, vol. 2009, p. 581, 2009.
4. S. Skorobogatov, "Synchronization method for SCA and fault attacks," *Journal of Cryptographic Engineering*, vol. 1, no. 1, pp. 71–77, 2011.
5. M. Karpovsky, K. J. Kulikowski, and A. Taubin, "Differential fault analysis attack resistant architectures for the advanced encryption standard," in *Smart Card Research and Advanced Applications VI*. Springer, 2004, pp. 177–192.
6. C.-H. Yen and B.-F. Wu, "Simple error detection methods for hardware implementation of advanced encryption standard," *Computers, IEEE Transactions on*, vol. 55, no. 6, pp. 720–731, 2006.
7. H. Igarashi, Y. Shi, M. Yanagisawa, and N. Togawa, "Concurrent faulty clock detection for crypto circuits against clock glitch based DFA," in *Circuits and Systems (ISCAS), 2013 IEEE International Symposium on*. IEEE, 2013, pp. 1432–1435.
8. M. Rohleder, T. Koch, V. Litovtchenko, and T. Luedeke, "Clock glitch detection circuit," Dec. 29 2011, uS Patent App. 13/131,349. [Online]. Available: http://www.google.com/patents/US20110317802
9. M. Karpovsky, K. J. Kulikowski, and A. Taubin, "Robust protection against fault-injection attacks on smart cards implementing the advanced encryption standard," in *Dependable Systems and Networks, 2004 International Conference on*. IEEE, 2004, pp. 93–101.
10. P. FIPS, "197," *Advanced Encryption Standard (AES)*, vol. 26, 2001.
11. Y. Li, K. Sakiyama, S. Gomisawa, T. Fukunaga, J. Takahashi, and K. Ohta, "Fault sensitivity analysis," in *Cryptographic Hardware and Embedded Systems, CHES 2010*. Springer, 2010, pp. 320–334.
12. *DS485: Digital Clock Manager (DCM) Module.*
13. *DS622: Phase Locked Loop (PLL) Module (v2.00a).*
14. *Altera Phase-Locked Loop (Altera PLL) Megafunction User Guide.*
15. *UG190: Virtex-5 FPGA User Guide.*