Filtering-Based CPA: A Successful Side-Channel Attack Against Desynchronization Countermeasures

Karim M. Abdellatif[1,3], Damien Couroussé[2], Olivier Potin[1] and Philippe Jaillon[1]

[1]École Nationale Suprieure des Mines de Saint-Etienne, France

[2]Univ. Grenoble Alpes, F-38000 Grenoble, France; CEA, LIST, MINATEC Campus, F-38054 Grenoble, France

[3]Electrical Engineering Department, Faculty of Engineering, Minia University, Egypt

[1,2]firstname.lastname@emse.fr, firstname.lastname@cea.fr

ABSTRACT

Secure implementations against side channel attacks usually combine hiding and masking protections in software implementations. In this work, we focus on desynchronization protection which is considered as a hiding countermeasure. The idea of desynchronization is to obtain a non-predictable offset of the attacking point in terms of time dimension. For this purpose, we present exploiting pattern-recognition methods to filter interesting points for obtaining a successful side channel attack. Using this tool as a case study, we completely cancel the desynchronization effect of the CHES 2009/2010 countermeasure [2, 3]. Moreover, 25k traces are needed for a successful key recoveries in case of polymorphism-based countermeasure [4].

1. INTRODUCTION

Masking and hiding are two popular solutions to achieve better resistance against side-channel attacks, which are usually used in combination in security products. Masking is used to merge the sensitive (key dependent) data with random data (the mask) which is unknown to the attacker. Hiding is used to reduce the signal-to-noise ratio of leakage information in observation traces, hence requiring a greater number of observation (side channel traces) to recover the secret data. A common way to achieve hiding at the software level is to perform desynchronization between observation traces. Desynchronization countermeasures statically insert, in the functional code, a desynchronization routine which has unpredictable execution time from the attacker side as shown by [2, 3]. Another hiding countermeasure highlighted executing so-called dummy instructions: code sequences that are similar to the behavior of the protected code (e.g. the SubBytes routine), but on fake data was presented by [7]).

CS2 '17, January 24 2017, Stockholm, Sweden

DOI: http://dx.doi.org/10.1145/3031836.3031842

Random dummy loops were presented by [2, 3] in order to add random shiftings in the time dimension to be an effective countermeasure against side-channel attacks. In [4, 1], runtime code polymorphism was proposed as a generic protection against side channel attacks. The idea is to obtain new versions of the secured binary code on the device. Each version is functionally equivalent but has a different implementation. Hence, execution would lead to a different observation in terms of the power consumption. The authors of [4] invested this code transformation possibilities to obtain random register allocation, random instruction selection, instruction shuffling, and insertion of noise instructions, as a countermeasure against side channel attacks. **The idea of running new versions which have the same functionality but different power consumption raise this question, is it efficient to use code polymorphism for inserting random delays compared to previous work [2, 3] which was based on random dummy loops?** This open question was the motivation for this work to find a common attack for desynchronization countermeasures.

Our contribution: We present a channel attack on the design proposed by [4] and compare it with the previous work [2, 3]. Note that we use only the insertion of noise instructions in case of evaluating polymorphism-based countermeasure presented by [4]. The proposed attack depends on filtering interesting points in the traces before performing side channel attacks like CPA. We consider the countermeasure of [2, 3] as a reference for comparison in terms of the side channel evaluation (number of traces for a successful attack). Our attack successfully cancels the countermeasure proposed by [2, 3] and retrieve the secret key with a number of traces close to unprotected design (45-80 traces). In terms of [4], our attack shows that 25k traces are needed to break the key successfully.

The STM32VLDISCOVERY [8] evaluation kit was used in our setup. The board contains a Cortex-M3 core running at 24 MHz. It is not equipped with hardware security protections. The side-channel traces were obtained with a 2208A PicoScope, which features a 200 MHz bandwidth and a vertical resolution of 8 bits. An EM probe RF-U5-2 from Langer was used, and a PA 303 amplifier from Langer.

Section 2 and **Section 3** show the background of the countermeasures presented in [3, 4], respectively. **Section 4** proposes using interesting points for a successful CPA. **Sec-**

tion 5 presents the side channel evaluation of the two designs using interesting points. **Section 6** concludes this work.

2. CHES 2009/2010 COUNTERMEASURE

In this section, we present the countermeasure introduced at CHES 2009 (and improved at CHES 2010) by Coron and Kizhvatov [2, 3]. The random delays were inserted at 10 places per AES round (before AddRoundKey, before each 32-bit SubBytes block, before each 32-bit MixColumn block, and after the last MixColumn block).

The two proposals (CHES 2009/2010 countermeasures) highlight finding a good statistical distribution for the random lengths of the delays. CHES 2009 paper [2] analyzes floating point method. Its goal is to decrease the average length of random delays and increasing the variance. In the CHES 2010 paper [3], the authors noted a bad choice of parameters for the floating mean method can lead to security weaknesses. Therefore, they proposed an improved solution with a new criterion to evaluate the security of Random Delay Interrupt (RDI). In both cases, their implementations ran on an 8-bit Atmel AVR platform. The common between the two designs is using random dummy loops as a random delay.

Our implementation of CHES 2009/2010 countermeasure followed the same guidelines as in [2, 3] and was based on the AES-128 design [6]. We are interested in how the delays are inserted in the normal operation of AES instructions, and we don't focus on how much delay is inserted (we describe later that the proposed attack doesn't depend on the length of the random delay).

Algorithm 1 describes the execution of the SubBytes stage. The random delay, which is a **random number of dummy loops**, is inserted before/after each 4 s-boxes calculation (see [2]).

Algorithm 1 Calculation of SubBytes

1: **for** $i = 0$; $i < 16$; $i + 4$ **do**
2: Random()
3: $state(i + 0) = Sbox(state(i + 0))$;
4: $state(i + 1) = Sbox(state(i + 1))$;
5: $state(i + 2) = Sbox(state(i + 2))$;
6: $state(i + 3) = Sbox(state(i + 4))$;
7: **end for**

Two side channel attacks based on removing this random delay were reported in the literature [5, 9]. In [9], the authors identified the patterns of dummy operations by string matching and then removed the random delay from the obtained traces. Their attack was applied on Atmel AVR ATmega8 using power consumption. Hidden Markov Models (HMM) was presented by [5] as another attack. We can conclude that the two attacks [5, 9] were based on identifying the pattern of the random delay by correlation or by string matching, respectively.

3. POLYMORPHISM COUNTERMEASURE

The framework named COGITO [4] aims at providing a realistic solution to achieve code polymorphism in embedded systems. The capabilities are currently as follows: (1) Drive runtime code generation by a source of random numbers to produce polymorphic application components, (2) Use semantic equivalences at the instruction level to produce different (but functionally equivalent) instances of code sequences, (3) Shuffle the machine instructions and use register renaming during runtime code generation (4) Introduce useless instructions that do not interfere with the functionality of the generated code, so called *noise instructions*,(5) With limited memory consumption and fast code generation, so that it is applicable to very small computing units such as secure elements.

Figure 1: 16 s-box calculation of AES

For the sake of conciseness, we forward the reader to [4], which details the code generation framework and the polymorphic implementation of the AES design. As compared to [4], we attack here a polymorphic version that only uses the insertion of noise instructions, and none of the other polymorphic transformations. In our understanding, the insertion of noise instructions will have the greatest impact on the desynchronization of a leakage point in observation traces. Other polymorphic transformations, such as random register allocation and semantic equivalences, also have an effect on desynchronization, but to a lesser degree.

We emphasize on the fact that code polymorphism with runtime code generation is fundamentally different: the main source of temporal desynchronization comes from the insertion of so-called noise instructions. Noise instructions can be identical to real code instructions (e.g. memory loads and stores during the execution of the SubBytes, `xor` instructions during AddRoundKey), and are tightly weaved with the real code instructions thanks to runtime code generation. Hence, it is difficult to distinguish real and noise instructions As a result, the protected code presents better resistance to resynchronization attacks compared to [2, 3] as we will see later.

4. INTERESTING POINTS

In this section, we present the meaning of interesting points and how such points are very critical in terms of side channel attacks. We implemented AES-128 without any countermeasure as a case study to help us presenting interesting points. Fig. 1 shows the electromagnetic signal taken under executing the first SubBytes stage (16 s-boxes calculations) in the first round of AES.

The basic idea of CPA is : the key assumed is correct when the correlation is maximum between the model of the hamming weight/distance under this assumed key, and the real power consumption of the design. Therefore, interesting points are these points which give the maximum correlation in the trace.

We can realize that executing one s-box takes 5 clock cycles (2 load instructions and one store instruction as shown in Fig. 1). Note that pipelining allows executing 2 **load** in-

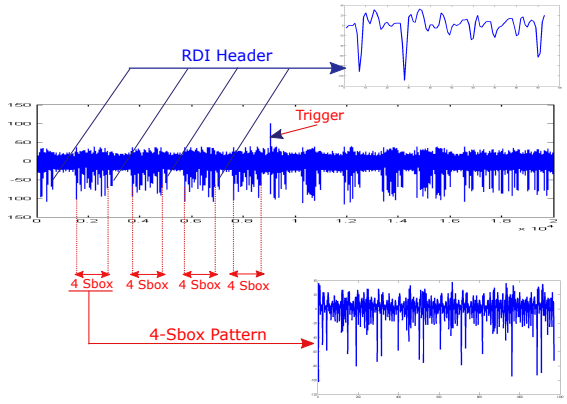

Figure 2: An electromagnetic trace of CHES 2009/2010

structions in 3 clock cycles but **store** instruction is executed in the last two cycles because of data dependency. Hence, applying CPA using the pattern of the last 2 clock cycles (during **store** instruction) will give the maximum correlation. This pattern indicates the power consumption while the new value of the s-box output is stored and this is exactly what a successful CPA looks for. We call this window ("store" instruction) as "**interesting points**".

5. SIDE CHANNEL ANALYSIS

In this section, we present the side channel analysis of the polymorphism-based countermeasure [4] compared to the countermeasure presented in [2, 3]. Our methodology uses correlation to detect interesting points inside the traces and not to remove the random delay like previous attacks.

5.1 CHES 2009/2010 Countermeasure [2, 3]

This countermeasure was targeting the random delay outside the interesting points in the trace (outside s-box calculation). Fig. 2 shows one trace from the countermeasure presented in [2, 3]. Note that the trigger starts with the beginning of SubBytes and terminates with the end of SubBytes calculation. We can conclude that the interesting points are inside (during) the calculation of 4 s-boxes and also before the Random Delay Interrupt (RDI) header.

We can filter interesting points by using the pattern of 4 s-boxes or RDI header. This is achieved by scanning the pattern of the 4 s-boxes or the RDI header on the trace (till the trigger position) by using cross correlation to assign the positions of the RDI header and the 4 s-boxes (see **Algorithm 2**). Fig. 3 shows the result of the cross correlation using the pattern of the 4 s-boxes (Note: the same will be in case of using the pattern of RDI header). It is clear that there are four maximum points, which means the existence of 4 grouped s-boxes.

After selecting interesting points that include the "store" instructions of the s-box, we can then perform CPA directly using these points as shown in Fig. 4. Fig. 5 shows the result of the CPA using the two cases of filtering. We can see that 85 and 45 traces are needed to detect the key in case of using the pattern of RDI headers and SubButes, respectively.

Algorithm 2 Selecting interesting points of [2, 3]

1: **Input** $Position_{Trigger}$, $Length_{Pattern}$, Ref, Trace
2: **Output** $Points_{Interest}$
3: **for** $i = 1 : Position_{Trigger} - Length_{Pattern}$ **do**
4: $Part_{Trace}$=$Trace(i : i + (Length_{Pattern} - 1))$;
5: $X(i) = Correlation(Ref, Part_{Trace})$;
6: **end for**
7: $Points_{Interest} = Max(X, 4)$;
8: Return $Points_{Interest}$

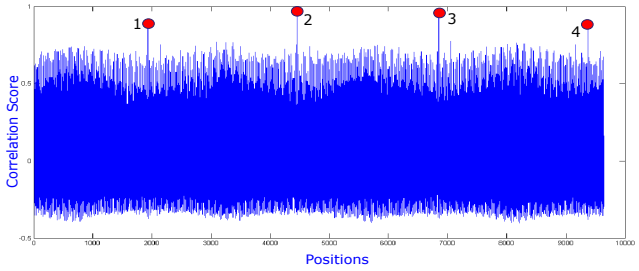

Figure 3: Cross Correlation score

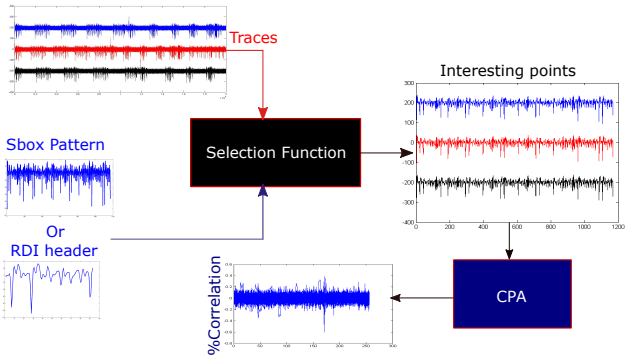

Figure 4: CPA-based selection function

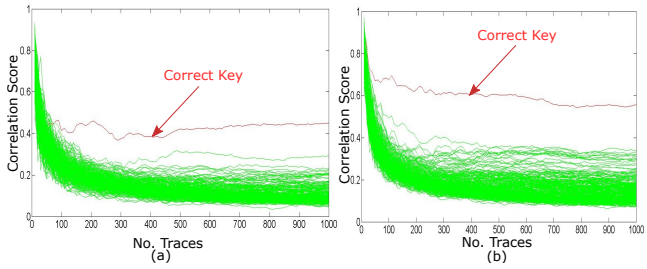

Figure 5: CPA result using interesting points, (a) with the RDI header pattern, (b) with the pattern of 4-sboxes

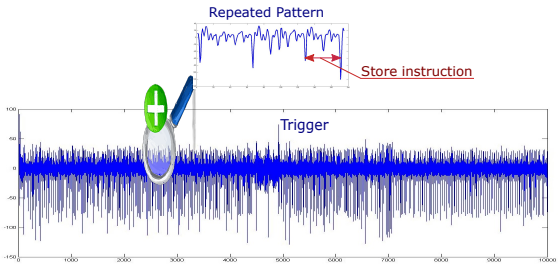

Figure 6: Electromagnetic trace of [4]

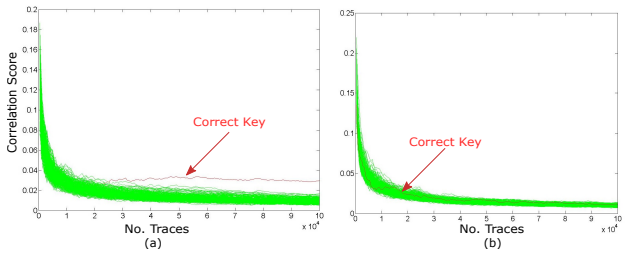

Figure 7: CPA result using interesting points of [4], (a) Using interesting points, (b) direct CPA

5.2 Polymorphism countermeasure [4]

In this part, we evaluate the countermeasure involving desynchronization with code polymorphism [4]. Fig. 6 shows an example of the electromagnetic traces during the execution of the first SubBytes stage. It presents the execution of 16 s-boxes separately under the countermeasure of **noise**. This figure illustrates that the electromagnetic trace is not clear like Fig. 2 to identify the calculation of the s-box (interesting points). To apply the methodology shown in Section 5.1 (Fig. 4) that filters interesting points, we have to do simple power analysis with more than one trace. The idea is to find a pattern which is repeated 16 times indicating 16 s-boxes calculation.

We found that the pattern shown in Fig. 6 is repeated 16 times. A part of this pattern is similar to 'store' instruction which identifies the interesting points in the calculation of the s-box. Therefore, we filter these patterns from the traces using the selection function (see Fig. 4 and **Algorithm 2**). Note that **Algorithm 2** selects four maximum peaks of correlation but in this case it should select sixteen peaks because of executing 16 s-boxes separately.

After selecting interesting points, CPA is then performed to detect the secret key. We computed the correlation coefficient on raw power traces using the Hamming weight power model to attack the first s-box. We found that 25k traces are sufficient to detect this key (see Fig. 7(a)). On the other hand, if direct CPA is performed on the design of [4], 100k traces are not sufficient to detect the key (see Fig. 7(b)).

It is clear that the presented design by [4] needs more traces compared to [2, 3] in order to obtain the secret key in case of applying CPA with/without filtering (see Table 1). On the other hand, Table 1 also shows the comparison between our presented attack and previous attacks in terms of the number of traces required for a successful attack.

Table 1: Comparison between attacks

Countermeasure	Attack	No. Traces
[2, 3]	direct CPA [2]	35000
[2, 3]	HMM [5]	100
[2, 3]	String matching [9]	50
[2, 3]	**RDI header detection**	**85**
[2, 3]	**S-box pattern detection**	**45**
[4]	**direct CPA**	**>100000**
[4]	**S-box pattern detection**	**25000**

6. CONCLUSION

In this paper, we presented the side channel attack on the designs proposed by [4] and [2, 3]. Filtering interesting points that indicate "store" instructions in the calculation of SubBytes was presented. The importance of such points was clear to perform a successful attack with more than 25k traces on the countermeasure of [4] and removing completely the countermeasure of [2, 3]. Compared to [2, 3], the polymorphism-based countermeasure required more traces in case of performing normal CPA or filtering-based CPA.

7. ACKNOWLEDGMENTS

This work was partially funded by the COGITO project, funded by the French National Research Agency (ANR) as part of the program Digital Engineering and Security (INS-2013), under grant agreement ANR-13-INSE-0006-01.

8. REFERENCES

[1] Agosta, G., Barenghi, A., and Pelosi, G. A code Morphing Methodology to Automate Power Analysis Countermeasures. In *Proceedings of the 49th Annual Design Automation Conference* (2012), ACM, pp. 77–82.

[2] Coron, J.-S., and Kizhvatov, I. An Efficient Method for Random Delay Generation in Embedded Software. In *Cryptographic Hardware and Embedded Systems-CHES 2009*. Springer, 2009, pp. 156–170.

[3] Coron, J.-S., and Kizhvatov, I. Analysis and Improvement of the Random Delay Countermeasure of CHES 2009. In *International Workshop on Cryptographic Hardware and Embedded Systems* (2010), Springer, pp. 95–109.

[4] Couroussé, D., Barry, T., Robisson, B., Jaillon, P., Potin, O., and Lanet, J.-L. Runtime Code Polymorphism as a Protection Against Side Channel Attacks. In *IFIP International Conference on Information Security Theory and Practice* (2016), Springer, pp. 136–152.

[5] Durvaux, F., Renauld, M., Standaert, F.-X., tot Oldenzeel, L. v. O., and Veyrat-Charvillon, N. Cryptanalysis of the CHES 2009/2010 Random Delay Countermeasure. *IACR Cryptology ePrint Archive 2012* (2012), 38.

[6] Heron, S. Advanced Encryption Standard (AES). *Network Security 2009*, 12 (2009), 8–12.

[7] Rivain, M., Prouff, E., and Doget, J. Higher-Order Masking and Shuffling for Software Implementations of Block Ciphers. In *Cryptographic Hardware and Embedded Systems-CHES 2009*. Springer, 2009, pp. 171–188.

[8] STmicroelectronics. UM0919 User Manual. http://www.st.com/content/ccc/resource/technical/document/user_manual/f3/16/fb/63/d6/3d/45/aa/CD00267113.pdf/files/CD00267113.pdf/jcr:content/translations/en.CD00267113.pdf.

[9] Strobel, D., and Paar, C. An Efficient Method for Eliminating Random Delays in Power Traces of Embedded Software. In *International Conference on Information Security and Cryptology* (2011), Springer, pp. 48–60.