Microprocessors and Microsystems 45 (2016) 324–338

Contents lists available at ScienceDirect

Microprocessors and Microsystems

journal homepage: www.elsevier.com/locate/micpro

A new countermeasure against side-channel attacks based on hardware-software co-design

Ruben Lumbiarres-Lopez [a], Mariano Lopez-Garcia [a,*], Enrique Canto-Navarro [b]

[a] 1-Electronic Engineering, Universidad Politècnica de Cataluña, Avda. Victor Balaguer, 08800, Vilanova i la Geltrú, Spain
[b] 2-Enrique Cantó-Navarro, Universitat Rovira i Virgili, Automatics and Electronic Engineering, Avda. Països Catalans, Tarragona, Spain

ARTICLE INFO

Article history:
Received 18 December 2015
Revised 20 April 2016
Accepted 17 June 2016
Available online 5 July 2016

Keywords:
Countermeasure
Side-channel analysis
AES algorithm and hardware-software co-design

ABSTRACT

This paper aims at presenting a new countermeasure against Side-Channel Analysis (SCA) attacks, whose implementation is based on a hardware-software co-design. The hardware architecture consists of a microprocessor, which executes the algorithm using a false key, and a coprocessor that performs several operations that are necessary to retrieve the original text that was encrypted with the real key. The coprocessor hardly affects the power consumption of the device, so that any classical attack based on such power consumption would reveal a false key. Additionally, as the operations carried out by the coprocessor are performed in parallel with the microprocessor, the execution time devoted for encrypting a specific text is not affected by the proposed countermeasure. In order to verify the correctness of our proposal, the system was implemented on a Virtex 5 FPGA. Different SCA attacks were performed on several functions of AES algorithm. Experimental results show in all cases that the system is effectively protected by revealing a false encryption key.

1. Introduction

Since Kocher et al. [1], in the late 1990s, demonstrated the vulnerabilities of cryptographic devices, Side Channel Analysis (SCA) attacks have become the most significant threat related to the security of cryptographic algorithms. These attacks base their success on analyzing the leakage information that is mainly observable through the power consumption or the electromagnetic radiation (EM) emitted by a hardware device. The attack is feasible because either of these two quantities is related to the data being processed by the device, which depends on the value of the cryptographic key.

Once such weakness was revealed, part of the scientific community oriented their efforts in proposing countermeasures that provide resistance against SCA attacks. Although with some differences, almost all proposed solutions attempt to design systems in which the power consumption (or the EM) is independent of the data that they process. This objective is achieved either by providing systems featured with random power consumption or building devices in which such power is constant in each clock cycle. The latter approach, known as hiding, has usually been implemented at cell level based on the Dual-Rail Precharge (DRP) logic style. This style is tailored with signals represented by two complementary wires, in such a way that in every clock cycle only one switch per cycle is produced. Thus, during the pre-charge phase, both the direct and complementary wires are charged,whereas in the evaluation phase only one of them is discharged. Among the more significant proposals of this logic style can be found *Sense Amplifier Based Logic* (SABL) [2] and *Wave Dynamic Differential Logic* (WDDL) [3]. However, the main drawback of such DRP logic styles is that their success depends on the perfect balancing between the capacitive loads related to the complementary wires that form the overall circuit. This requirement implies including some constraints on the placement and routing steps. In contrast, the former approach, known as masking, has been implemented at both algorithm and cell levels. At cell level, the most relevant proposals are *Random Switching Logic* (RSL) [4], *Dual Random Switching Logic* (DRSL) [5] and *Masked Dual-Rail Precharge Logic* (MDPL) [6]. Masking Boolean approaches base their resistance against SCA attacks on concealing, by means of an exclusive OR operator, all intermediate values v with a random mask m. The masked values $v_m = (v \oplus m)$, which are actually being processed into the hardware device, are statistically independent with respect to v, so that the power consumption and the cryptographic key are completely uncorrelated. Thus, these logic styles are not affected by the imbalance existing between the routing capacitances of complementary wires. Furthermore, approaches based on hiding (i.e., SABL and WDDL) could be implemented in a smaller area than

* Corresponding author.
E-mail address: mariano.lopez@upc.edu (M. Lopez-Garcia).

http://dx.doi.org/10.1016/j.micpro.2016.06.009
0141-9331/

the one needed by the masked logic styles (i.e., MPDL, RSL, DRSL). Additionally, SABL, RSL and DRSL require designing specific cells for their implementation, whereas WDDL or MDPL allow designing such secure logic based on existing standard cells.

Moreover, it has been shown that in general the security of cell level implementations could be compromised due to the effect of the inter-wire capacitances [7] or the so-called *early propagation effect* [8,9]. As these vulnerabilities became known, the previous proposals were updated, including new measures that make systems more secure against most of these harmful effects. For instance, the original MDPL, which inherently is a glitch-free logic style based on majority-gates, was modified to support the early propagation effect (iMDPL) [10]. Other examples of improved DPR styles can be found in [11] and [12]. More recently, a new countermeasure termed SecLib has been proposed [13]. The early evaluation is prevented by designing specific cells based on two stages that avoid such effect. However, as stated by the authors, it also increases the cost in terms of area, delay and power consumption.

Masking is a countermeasure that can be also implemented at algorithm level. In [14], the authors proposed an implementation of the AES encryption algorithm using six independent masks. The algorithm was solved on an 8-bit microcontroller leading to an execution time twice that compared with the unmasked version. There are also some proposals for implementing hiding countermeasures on software. These approaches aim at introducing temporal jitter in the sequence of operations performed by the microprocessor. This way, the instant at which an effective attack might be produced is distributed over time following an unknown probability distribution function (misalignment of power traces). Some examples of these software countermeasures consist of introducing dummy cycles [15] or a random variation on the execution orders [16].

Other different publications aim at introducing noise to reduce the correlation between the processed data and the cryptographic key. Following this idea an interesting approach was proposed in [17], in which a noise generator correlated with the data that is being processed is included. However, the attack is only effective when the target is the function correlated with the introduced power noise. Additionally, the revealed key is not always the same and it depends on the number of traces captured and used to perform the attack.

As mentioned above, SCA attacks based their success on exploiting the existing dependence between the processed data and the power consumption (or EM). As data depend on the cryptographic key, from a statistical point of view it means that there exists a correlation between such a key and the consumed power. Theoretically, only the correct key is able to produce a correlation with a significant value, whereas the rest of the keys would generate a value close to zero. Countermeasures based on hiding or masking try to eliminate this correlation, in such a way that any SCA attack, performed on any possible key, does not produce any relevant result that could be distinguished among all others. In other words, all correlations between power consumption and guessed keys are equally likely and tend to zero.

The countermeasure proposed in this paper is completely different when compared with previous approaches. The mechanism for protecting the system consists in revealing a false key when a SCA attack is performed. This false key (or fake key) produces the highest correlation coefficient between the data processed and the power consumed by the hardware device. Thus, from the perspective of an attacker, the system behaves as an unprotected implementation that conceals the true key by producing a false positive. Note that, such implementation should be performed affecting as little as possible the power consumption trace (in amplitude and time) when compared with the original non-protected system.

Although the proposed countermeasure, termed faking, could be entirely implemented in software, the penalty on the execution time would be quite significant. In fact, including all additional calculations needed to conceal the real key, such execution time is almost doubled when compared with the non-protected version. Instead, the implementation presented in this paper is based on a hardware/software co-design. The system consists of a microprocessor which solves via software the classical Advanced Encryption Standard (AES) 128-bit cryptographic algorithm, and a coprocessor specifically designed for implementing the proposed countermeasure. The proposed architecture is intended for applications in which the main task performed by the microprocessor is to solve a specific processing from which a critical information is obtained. The encryption is necessary for storing this confidential data in an external device or for sending such information through a non-secure channel. For instance, the microprocessor could be used for analyzing a fingerprint image from which a confidential biometric feature is obtained and should be stored in an external memory. Although is out the scope of this paper, in applications where the encryption is the main task that should be performed, a complete hardware-implementation would be more suitable and faster. Regardless of the implementation chosen, hardware, hardware-software or pure software, the level of security for all of them is identical and only their features in terms of area and speed are different.

This paper is organized into five sections. Section 2 presents the fundamentals of the proposed countermeasure. The aim of Section 3 is to describe the internal architecture of the coprocessor and its main features. Section 4 presents the experimental results. Finally, Section 5 presents the conclusions

2. Fundamentals

2.1. Introduction

The structure of the AES 128-bit encryption algorithm is represented in Fig. 1 As the figure shows, the algorithm consists of four operations that are performed on a matrix of 16 bytes, termed state, in different rounds: *AddRoundKey (exclusive-OR), SubBytes, ShiftRows* and *Mixcolumns*. A general description about the principles of this cipher, including such four operations, can be found in [18,19].

Although in the proposal presented by *Kocher* the cryptographic key was found using the differential-of-means method, currently the most extended statistical method employed for this purpose is based on correlation [14]. This method consists of the following steps:

(a) The encryption algorithm is executed M times using a set of M different plain texts. For each one, a current trace is captured and stored for its subsequent processing.
(b) It is quite usual to choose as points to be attacked (target) the output of one of the four operations (inputs of the following points) involved in the AES algorithm, since their result (state) is normally written in a memory or register, which creates a distinguishable point at the captured power trace.
(c) A theoretical power model, which represents the consumption of the overall set of CMOS cells that form the circuit, should be chosen. Such a theoretical model is normally based on the Hamming distance (HD) or the Hamming weight (HW), that represents the value of a set of bits $v(t_k)$ related to the point to be attacked. Note that, if an intermediate value at instant (t_{k-1}) is $v(t_{k-1})$, then

$$HD(v(t_k)) = HW(v(t_{k-1}) \oplus v(t_k)) \quad (1)$$