Microprocessors and Microsystems 62 (2018) 50–60

Contents lists available at ScienceDirect

Microprocessors and Microsystems

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

Reactive side-channel countermeasures: Applicability and quantitative security evaluation

Giovanni Agosta, Alessandro Barenghi*, Gerardo Pelosi, Michele Scandale

Politecnico di Milano. Department of Electronics, Information and Bioengineering, DEIB, Piazza Leonardo da Vinci, 32. 20133, Italy

ARTICLE INFO

Keywords:
Applied cryptography
Embedded systems security
Computer security
Automated countermeasure application
Reactive countermeasures

ABSTRACT

The security of cryptographic implementations running on embedded systems is threatened by side-channel attacks. Such attacks retrieve a secret key from a computing device observing the information leaking on unintended channels such as the energy consumed during a computation. The vast majority of the countermeasures proposed against such attacks aims at preventing the attacker from exploiting fruitfully the information leaking on the side-channel either altering it or hiding it within a higher noise envelope. Whilst all these countermeasures provide a quantitative security margin against an attacker, they do not provide an indication of having been successfully overcome, thus forsaking the possibility of taking a reactive action upon an eventual security breach. In an effort to propose a reactive countermeasure, we describe our proposal suggesting the introduction of redundant computations employing fixed fake keys (a.k.a. *chaffs*) to pollute the leaked information with plausible albeit deceitful one. We provide an in depth analysis of the proposed approach, highlighting the constraints to its effective applicability, and the boundary conditions which allow its employment for the securization of a system. We detail the attacker model considered, and the reactive security margin provided by the proposed scheme, highlighting the extent of the realizability of a reactive countermeasure, given the nature of the side-channel information. To provide experimental backing to our analysis, effectiveness and efficiency results on the Advanced Encryption Standard (AES) cipher implementation as well as lightweight block ciphers implementations running on an ARM Cortex-M4 processor are shown.

1. Introduction

In recent years, the use of cryptographic primitives became essential for embedded systems due to their widespread adoption in several security-sensitive domains such as health-care, automotive and industrial control. Modern cryptographic systems are designed to withstand both protocol-level and mathematical cryptanalyses; as a consequence, attackers often focus on the analysis of the side effects of the computation performed by the device. There is a vast corpus of academic and industrial literature proving how the physical access to an embedded device may enable the recovery of sensitive information (typically, the secret key employed in a cryptographic algorithm), which is otherwise supposed to be hidden, through exploiting the side-channel leakages of the underlying computing platform [1–4].

Among the possible cryptanalyses employing the side-channel leakage, Differential Power Analysis (DPA) and Differential Electromagnetic Analysis (DEMA) are very well known [3,5,6]. Both analyses follow a common work-flow: first they record a measurement of either the power consumption or the electro-magnetic (EM) emissions of the target device (the measurements being known as *power-* or *EM-traces*) for a large number of runs with different input values. Subsequently, they select an intermediate operation of the algorithm employing a part of the secret key, and compute a consumption/emissions prediction for every possible value of the secret key portion, according to a model of the triggered switching activity (e.g., the Hamming weight of the output of the operation). Finally, the predicted consumption/emission values are statistically matched against each sample of the recorded power/EM traces to assess which key hypothesis yields the prediction fitting best the actual measurements. In this fashion, the secret key can be recovered, one part at time, even if the relevant information is stored within the device in a non accessible way.

DPAs and DEMAs have been proven practically viable against production grade implementations of ciphers, both in hardware and software, even with an inexpensive equipment. A significant amount of research effort has been directed to devise effective and efficient countermeasures. The most adopted strategies to design

* Corresponding author.
E-mail addresses: giovanni.agosta@polimi.it (G. Agosta), alessandro.barenghi@polimi.it (A. Barenghi), gerardo.pelosi@polimi.it (G. Pelosi), michele.scandale@polimi.it (M. Scandale).

https://doi.org/10.1016/j.micpro.2018.07.001
Received 26 March 2018; Accepted 4 July 2018

countermeasures focus on the principles of *masking* [3,7–9], *hiding* [3], and *morphing* [10–12] and aim at reducing the effective side-channel leakage. Masking aims at invalidating the link between the predicted hypothetical emission/power consumption values (bound to the selected intermediate operation) and the actual values processed by the cryptographic primitive. In a masked implementation, each sensitive intermediate value is concealed through splitting it in a number of shares, which are then separately processed. Hence, the cryptographic primitive is modified to correctly process each share and recombine them only at the end of the computation. Hiding methods aim at concealing the relation between the emission/power consumption of the device and the operations performed by the cryptographic primitive. In software cipher implementations, these strategies are based on execution flow randomization via instructions rescheduling (e.g., permuting the sequence of accesses to look-up tables) and/or on inserting random delays made of dummy operations. The morphing technique prevents an attacker from being able to construct a reliable model of the side-channel behavior of the device, changing how a cryptographic primitive is computed at each execution of the algorithm. The first proposed technique to achieve this combines the implementation of the chosen cryptographic primitive with a polymorphic engine which dynamically re-writes the binary code of the sensitive instructions to be protected, at run-time [10]. This strategy enables the generation of many different versions of the protected code at the designer's will, at each run of the cipher, thus preventing any attacker both from recognizing the exact point in time where the selected intermediate operation is executed, and from understanding how such operation is actually computed. The former effect can be classified as hiding-in-time, while the latter one prevents the formulation of a proper consumption model. Tackling the issue of platforms where the code memory is not writable, the approach described in [11] picks at random among a set of semantically equivalent code fragments to obtain different execution paths at each run of the same protected algorithm implementation.

All the aforementioned countermeasure strategies aim at either reducing the actual side-channel leakage or preventing its exploitation altogether, as the prime and only way to hinder side-channel attacks. None of the aforementioned strategies actually leave the leakage intact and blend it within a crowd of plausible, albeit fake, pieces of information. By contrast, it is commonplace, and a well established practice, in the network and system security communities, to provide intentionally vulnerable, worthless targets to the adversaries [13]. The purpose of such targets is to act as a red herring for adversaries, enabling the detection of malicious intentions towards the networks and hosts owned by the legitimate users (e.g., intrusion attempts). Other notable examples of using decoy resources to detect security breaches is found in the practice of deploying either phony credentials (such as credit card numbers) to discover their theft, or fabricated documents to act as bait for possible inside adversaries aiming at violating the system's usage policy.

The focus of this work is to provide an in depth description and security analysis of the reactive defense strategy presented in [12], against a side-channel attacker who has complete knowledge of the details of a software implementation of a block cipher primitive, and is trying to exfiltrate the secret key exploiting the information leakage during the decryption of a ciphertext. The attacker is assumed to have no means of access to the output of the decryption on the device, can only observe the actions performed by the attacked security system, and should not be able to distinguish an incorrectly decrypted plaintext from a correct one without interacting with the system itself. Practical application scenarios include keyless entry systems, physical authentication systems based on the transmission of an encrypted cryptographic token, or broadcast authentication schemes for wireless sensor networks. The *chaff countermeasure*, proposed in [12], swarms the attacker with dummy side-channel leakages among which the real one is blended. This allows the system designer to detect side-channel attacks whenever the adversary employs an incorrect value to produce forged encrypted content to be fed into the attacked system. This, in turn, allows a prompt response to the breach attempt before the attack succeeds, a much welcome feature in domains such as automotive, sensor networks and industrial control. We note that, since the fake leakage is not distinguishable from the real one, the security of the proposed defense strategy is not altered even when pitted against profiled cryptanalyses, as the fake leakage will provide the same information of the correct one to them. This is in contrast with leakage suppression techniques proposed in the current state-of-the-art (e.g., masking and hiding), where the defender attempts at hindering the exploitation of the leaked information raising the required technical effort to lead the attack.

We describe the application scenario of such a countermeasure, pointing out the applicative context and attacker model, and providing a hint of how the recently proposed *Honey Encryption* scheme [14,15] may be employed to broaden the number of viable application scenarios where the chaff countermeasure can be applied; thus, expanding and improving the discussion with respect to the one in [12]. We report how to automatically apply the proposed countermeasure to software cipher implementations, and implement it as a transparent compiler pass in the LLVM compiler suite. Subsequently, we provide a new and detailed security analysis, highlighting the extent of the protection provided by the use of *chaff*-keys both against a single side-channel attack, and against a cascade of attacks. To practically ground our security analysis, we report experimental results on execution time and code size overheads due to the introduction of chaff-based countermeasures on a range of block ciphers, suitable to the use case scenario for which the chaff countermeasure is proposed. We provide results on the AES cipher, as it is a widely used standard employed also in current high security keyless entry systems (e.g., NXP ACTIC-4G [16], Atmel ATA5795C [17]), and was analyzed in [12], albeit with a lower number of chaff keys. We extend our experimental campaign, with respect to the one in [12], considering the computing requirements and code sizes of the lightweight ciphers XTEA [18], SPECK 128/128 [19], SIMON 128/128 [19], PRESENT and CLEFIA. PRESENT and CLEFIA are the ISO standard lightweight ciphers [20].

The rest of the article is organized as follows: in Section 2 we point out the scenario and the applicability conditions of the chaff-based countermeasure, as well as the attacker model. In Section 3 we highlight the property of a chaff-protected implementation and describe the methodology and tool to automatically apply the countermeasure. In Section 4 we detail the security analysis of the chaff-based approach, while in Section 5 we provide our experimental evaluation, highlighting the advantages of applying the chaff-countermeasure to lightweight ciphers. Finally, in Section 6 we draw our conclusions.

2. Scenario

The scenario where the reactive chaff-based countermeasure is deployed is the one where the system manufacturer desires the deployed equipment to take an active stance against side-channel adversaries through reacting to an attack attempt, e.g., signal to the data owner the malicious action or wipe the secret key in the device.

The typical information flow of the scenario where it is possible to put into being a reactive countermeasure is the one represented in Fig. 1. In this setting, a message produced in a safe environment and encrypted by means of a symmetric cipher, is sent to the device targeted by the side-channel attacker. The encrypted content is sent over a transport layer, which is assumed to be completely open to the attacker's eavesdropping actions. The message confidentiality during transmission is provided by the symmetric encryption layer, while unintentional transmission errors are prevented by an error correction code applied to the encrypted message. The content is decrypted by the deployed device into which the secret key is stored by the manufacturer in a non-extractable fashion. This decryption action is the actual target of side-channel attacks, with the final purpose of employing the derived