Exploiting Small Leakages in Masks to Turn a Second-Order Attack into a First-Order Attack

Alexander DeTrano
New York University
detrano@nyu.edu

Sylvain Guilley
ENST & Secure-IC S.A.S.
sylvain.guilley@enst.fr

Xiaofei Guo*
Security Center of Excellence,
Intel Corporation
xiaofei.rex.guo@intel.com

Naghmeh Karimi
Rutgers University
naghmeh.karimi@rutgers.edu

Ramesh Karri
New York University
rkarri@nyu.edu

ABSTRACT

Masking countermeasures, used to thwart side-channel attacks, have been shown to be vulnerable to mask-extraction attacks. State-of-the-art mask-extraction attacks on the Advanced Encryption Standard (AES) algorithm target S-Box re-computation schemes, but have not been applied to scenarios where S-Boxes are precomputed offline. We propose an attack targeting precomputed S-Boxes stored in non-volatile memory. Our attack targets AES implemented in software protected by a low entropy masking scheme and recovers the masks with 91% success rate. Recovering the secret key requires fewer power traces (in fact, by at least two orders of magnitude) compared to a classical second order attack. Moreover, we show that this attack remains viable in a noisy environment, or with a reduced number of leakage points.

Categories and Subject Descriptors

E.3 [**Data**]: Data encryption; K.6.5 [**Computing Milieux**]: Management of computing and information systems—*Physical security*

1. INTRODUCTION

Traditionally, a cryptographic algorithm was considered secure if it withstood classical linear and differential cryptanalysis. A side-channel attack exploits physical characteristics of a device in order to recover secret information, such as the encryption key. Power dissipation and electromagnetic (EM) emanation side-channel attacks are of particular concern because of their low implementation cost, ease of use, and effectiveness in extracting secret information [10]. Power analysis attacks work because the amount of power (or EM emanations) dissipated by a device is dependent on the data being processed. The Advanced Encryption Standard (AES) is the standard symmetric key encryption specified by National Institute of Standards and Technology (NIST) in FIPS 197, and is also included in ISO/IEC 18033-3:2010. It is widely used in electronic systems such as automated teller machines, telecommunications, and virtual private networks. Traditional cryptanalysis cannot break AES. However, if AES is not carefully implemented, side-channel attacks can leak the secret key [10, 3, 2].

1.1 Related Work

Masking variables is a well-known countermeasure [16, 18, 15, 5] to protect against side-channel attacks. Sensitive variables are concealed by random variables. Masking comes in a variety of flavors, however we consider only the *Boolean* type in this paper. Boolean masking splits a sensitive variable x into a number $(d+1)$ of shares by the exclusive-or (XOR) operation $x = x_0 \oplus \ldots \oplus x_d$. Each share is processed independently so that the measured leakage depends on some random value, rather than the sensitive information. A first-order masking scheme uses one mask, whereas a d^{th}-order masking scheme uses d masks. A $(d+1)^{th}$-order attack targets the manipulation of $d+1$ manipulated variables that jointly depend on a secret value (e.g., the $d+1$ masks). A d^{th}-order masking scheme can be broken by a $(d+1)^{th}$-order attack [11]. Masking strategies can also be classified according to the amount of entropy used; intuitively, the more entropy in the set of masks, the more secure the implementations are against side-channel analysis. *Full Entropy Masking Schemes* (FEMS) draw masks from the entire mask set to conceal sensitive information [20]. In the case of AES, each plaintext byte is masked, and so each mask can take on all 256 values from $\mathbb{F}_2^8$. *Low Entropy Masking Schemes* (LEMS) instead draw masks from a reduced mask set, a strict subset of $\mathbb{F}_2^8$ [20, 7].

Masking the non-linear portions of AES, i.e., the S-Boxes can be costly. The masked S-Boxes can be calculated on-the-fly for each encryption [16], securely precomputed before encryption begins [8], or generated offline and stored in ROM/RAM [12]. The S-Box precomputation scheme suits

*The views expressed in this paper solely belong to the authors and do not in anyway reflect the views of Intel Corporation. The countermeasures described in this paper were implemented in experimental hardware and software environments. The authors of this paper have not explored the potential applicability of these countermeasures to commercially available hardware and software.

HASP '15, June 13 2015, Portland, OR, USA
Copyright is held by the owner/author(s). Publication rights licensed to ACM.
ACM ISBN 978-1-4503-3483-9/15/06 ...$15.00.
DOI: http://dx.doi.org/10.1145/2768566.2768573.

AES, because the 16 S-Boxes are the same (unlike DES, for instance). However, the S-Box precomputation method significantly increases total encryption time. The masked S-Box is typically recalculated for every encryption and this S-Box recomputation can be as long as the entire AES operation, if not longer. For instance, the authors in [11] describe an AES implementation that takes twice as long to encrypt a plaintext versus the equivalent unprotected version; 33% of the run-time is spent calculating the masked S-Box. The frequent reuse of the mask during the S-Box precomputation allows for horizontal attacks (deemed horizontal because multiple points along a single power trace are analyzed [4]), which exploit the high multiplicity of samples (namely, 256) to recover the mask [14, 19].

Computing offline the entire set of masked S-Boxes (256 for a FEMS) alleviates the extra runtime issue of S-Box recomputation, but requires at least 64 kilobytes of memory which is beyond the capacity of embedded systems such as smartcards. LEMS offers a trade-off between complexity and security. The space required for a LEMS using 16 masks out-of-256 masks is that needed to store 16 S-Boxes (namely 4 kilobytes of storage). Removing the need for a lengthy masked S-Box precomputation, we notice that LEMS are less prone to attacks such as [14, 19]. Additional masks (as in high-order masking schemes) increase the complexity and area overhead of the design, since these extra masks have to be stored in memory or calculated at some point in time. Therefore first-order masking schemes are the mainstream protection.

1.2 Contribution and Outline

Efficient first-order masking schemes (FEMS using S-Box precomputation or LEMS such as Rotating S-Box Masking [12]) reuse the same mask several times, typically at each S-Box call, therefore a horizontal power analysis attack on 16 leakage points can reveal the mask. We show that the state-of-the-art mask extraction attack [19] on S-Box precomputation can be retargeted towards a masked AES implementation. Indeed, the attack presented in [19, 14] is the core idea of this paper. At the time of writing, a similar attack was published on the DPA Contest website[6] by Nakai et al. We want to stress that both works were performed independent of each other. We therefore add value by exploring the attack parameters in order to gain a deeper understanding of the strength of the attack. We show that the attack can succeed even in the presence of noise: tiny information on the mask can be extracted, enabling a first-order attack in a second pass. We find that this type of attack outperforms a classical second-order attack with respect to number of traces needed to recover the key.

The rest of the paper is organized as follows. Section 2 proposes the mask recovery attack, and validates it using publicly available data. Section 3 discusses the attack results and attack parameters, compares the attack with a state-of-the-art second-order attack [17] in noisy environments, and proposes a countermeasure. Section 4 concludes the paper.

2. PROPOSED MASK RECOVERY ATTACK

We describe the implemented countermeasure, power analysis, and the proposed attack.

2.1 Rotating S-Box Masking

A first-order masking countermeasure called *Rotating Sbox*

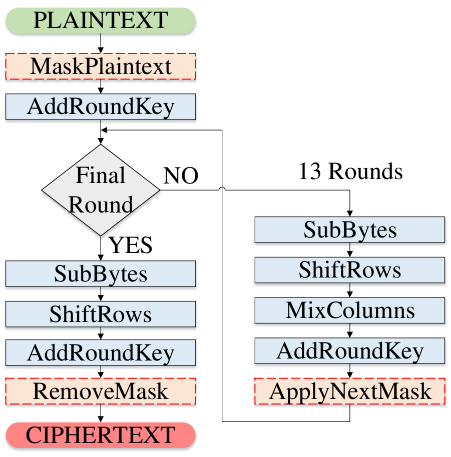


Figure 1: AES-256 with the Rotating S-Box Masking (RSM) protection. RSM is a low entropy masking scheme. The dashed boxes represent the operations added by RSM to AES.

Masking (RSM) [12] is shown in Figure 1. The dotted boxes represent the additional steps added to AES-256 by RSM. RSM is a Boolean-additive LEMS and uses a total of 16 public-knowledge masks, $m_{0-15} \in \mathcal{M} \subset \mathbb{F}_2^8$, one for each byte of plaintext. At the start of each encryption, a random offset $j \in [0 \dots 15]$ is drawn. The offset can be thought of as the number of positions to cyclically left-rotate the base set of masks, $\mathcal{M}_0$. The set of masks with offset j is denoted as $\mathcal{M}_j$, e.g., if the offset $j = 0$, then the masks are deployed in the following order $\mathcal{M}_0 = m_0, m_1, m_2, \dots, m_{14}, m_{15}$. Thus, only 16 possibilities exist for the the order of the masks, since a shift greater than 15 simply wraps around. The masks are then XORed with the plaintext, and this result is XORed with the first round key. The S-Box is replaced by 16 masked S-Boxes, where each S-Box corresponds to an offset. This avoids the penalty of the lengthy S-Box recomputation that other masking schemes utilize (except masking schemes with S-Box secure calculation [18, 5]). *ShiftRows* is unchanged since the underlying data is not modified. The *MixColumns* operation is a special masked version. Afterwards, the next-round masks are applied while simultaneously removing the current-round masks, and the offset value is incremented. It is important to stress that the data never appear unmasked.

2.2 Power Analysis

A generic power (or EM) analysis attack has the following five steps [11].

1) **Measure the power consumption (or EM) of a device as it encrypts (*resp.* decrypts) a number of plaintexts (*resp.* ciphertexts).** We used EM traces provided by the DPA Contest V4 [6], as detailed in Section 2.3.
2) **Choose an intermediate result of the target algorithm to attack.** Normally a part of the algorithm that operates on the key is attacked. However, we wish first to recover the mask set, so we target the loading of the masks, as described in Section 2.5.
3) **Calculate the intermediate results for all secret**

hypotheses. In this case, there are 16 possibilities for the mask set, shown in matrix $\mathbf{M}$ in Section 2.6.

4) **Apply a hypothetical power model to the calculated intermediate results**. We used the Hamming weight power model, as described in Section 2.6.
5) **Compare the measured power consumption to the hypothetical power consumption to determine the secret key (or a small part of the key)**. This is explained in more detail in Section 2.6.

This attack is performed in two stages: (1) the pre-processing mask recovery stage and (2) CPA attack to recover the key. The basic idea is to recover an estimate of the masks from each power trace, and then launch a horizontal (attacking many samples from a single trace) CPA attack against the 16 possible combinations of the mask. Recovering the masks allows us to undo the countermeasure so that we can correctly predict some intermediate value, e.g., the S-Box output. Thus a second CPA attack, vertical (attacking the same time instance across many traces) this time, reveals the key. Both stages are first-order attacks.

2.3 Experimental Setup

The AES-256 RSM is implemented on an Atmel ATMega-163 smartcard connected to a SASEBO-W board [6]. EM traces were captured using a Langer EM near-field probe RF-U 5-2, sampled at 500 MS/s by a Lecroy Waverunner 6100A oscilloscope.

Algorithm 1: Mask Recovery

```
input  : Window when masking is thought to occur $W$
         A single power trace $t$
         Length of masking operation $\Delta$
         Mask matrix $\mathbf{M}$
output: The mask set $\mathcal{M}_g$ and the mask offset $g$
1  $\tau \leftarrow$ ChooseSamples(W) ;                    // leakage Detection
2  $r \leftarrow 0$ ;                          // row index for subtrace matrix V
3  for $i \in \tau$ do
4      for $j \leftarrow 0$ to 15 do
5          $lkg \leftarrow t(i + j \cdot \Delta)$ ;     // measured leakage at sample i
           for byte j
6          $\mathbf{V}[r, j] \leftarrow lkg$ ;                // build subtrace matrix
7      end
8      $r \leftarrow r + 1$ ;                               // increment row index
9  end
10 $\mathbf{H} \leftarrow hw(\mathbf{M})$ ;                          // mask Hamming weight
11 return $g \leftarrow \arg\max \hat{\rho}(\mathbf{V}, \mathbf{H})$ ;     // recover the mask offset
12 return $\mathcal{M}_g \leftarrow \mathbf{M}[:, g]$ ;                      // guessed mask set
```

2.4 Leakage Detection

We use Normalized Inter-class Variance (NICV) [1], which is an analysis of variance (ANOVA) F-Test, to identify leakage in power traces. The NICV relies on publicly available information (such as known plaintexts or ciphertexts). Let T be the set of power traces, and X be the corresponding set of plaintext bytes. The NICV is calculated as $NICV = \frac{\mathrm{Var}(\mathbf{E}[T|X])}{\mathrm{Var}(T)}$ where, $0 \leq |NICV| \leq 1$. Figure 2 shows the NICV calculated for each plaintext byte using 10,000 traces, and reveals useful information to the attacker. With knowledge of the algorithm, he can distinguish when different operations take place. The 16 peaks in Figure 2(a) from samples 0 to 75,000 suggest the *AddRoundKey* operation, while the second set of 16 peaks beginning at sample point 10^5 signifies the *SubBytes* operation. An attacker can use this knowledge to extract leakage samples that belong to a certain operation.

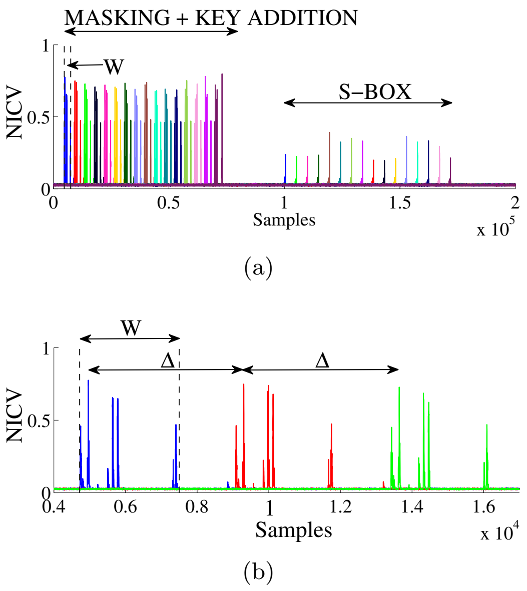


Figure 2: NICV for each plaintext byte over 10,000 **traces. (a) AES operations are identifiable (b) NICV for the first 3 bytes of plaintext. Each byte exhibits similar characteristics, which implies the operation taking place a number times, but each time processing different data.**

The attacker now has a rough idea of the time frame when each operation takes place, and can even determine the amount of time to process each byte by examining Δ, the distance between the peaks in Figure 2(b). Figure 2(a) shows that each plaintext byte is operated on only once before it enters the S-Box, i.e. there is only one time interval when leakage occurs for each plaintext byte before the S-Box. Therefore the plaintext loading, masking operation, and *AddRoundKey* must all take place within the same time interval. Moreover, the order and morphology of each NICV curve tells the attacker that the same set of operations is applied 16 times in a row, beginning with byte 0 and ending with byte 15. Consequently, the attacker now has an idea about the mask order.

2.5 Extract Leaky Samples

The attacker then chooses a window W of width Δ, and extracts possible candidates for the time samples when each mask is loaded. The attacker can use the NICV (or some other leakage detection tool [1] such as SOSD or SOST) to minimize the amount of points he will attack by considering only leakage measurements above a certain threshold (determined empirically), or he can simply attack every point in the window. The attacker selects τ samples to attack from a single power trace, and stores their leakage measurements, v, into the first column of the $\tau \times 16$ matrix $\mathbf{V}$. Each column $\mathbf{V}$ is then filled in by extracting the leakage measurement exactly Δ samples from the previous measurement.

$$\mathbf{V} = \begin{bmatrix} t_0 & t_0+\Delta & t_0+2\Delta & \cdots & t_0+15\Delta \\ t_1 & t_1+\Delta & t_1+2\Delta & \cdots & t_1+15\Delta \\ \vdots & \vdots & \vdots & \ddots & \vdots \\ t_{\tau-1} & t_{\tau-1}+\Delta & t_{\tau-1}+2\Delta & \cdots & t_{\tau-1}+15\Delta \end{bmatrix}.$$

2.6 Recover the Mask Offset

The next step is to launch a modified CPA attack on the sub-traces in $\mathbf{V}$. Since we do not know in which order the masks were loaded, we guess every combination, as shown in the 16×16 matrix $\mathbf{M} = [\mathcal{M}_0 \ldots \mathcal{M}_{15}]^\top$. Each column of $\mathbf{M}$ corresponds to an offset applied to the base set of masks $\mathcal{M}_0$, where

$$\mathbf{M} = \begin{bmatrix} m_0 & m_1 & m_2 & \cdots & m_{15} \\ m_1 & m_2 & m_3 & \cdots & m_0 \\ \vdots & \vdots & \vdots & \ddots & \vdots \\ m_{15} & m_0 & m_1 & \cdots & m_{14} \end{bmatrix}.$$

We apply a Hamming weight power model $hw(\cdot)$ to the mask matrix $\mathbf{M}$, which is generally a good model for microprocessors [11, 9]. The hypothetical power consumption is $\mathbf{H} = hw(\mathbf{M})$. The next step is to compare the modeled power consumption with the measured power consumption. If we assume the power model to be linear, e.g., Hamming weight or Hamming distance, a natural choice for the attack is the correlation coefficient. Correlation power analysis (CPA) evaluates the amount of correlation between a set of measured power traces T and a model of the key-dependent device leakage, L [3], and is calculated for every time sample. Pearson's correlation coefficient is calculated as $\rho(T, L) = \text{cov}(T, L)/(\sigma_T \sigma_L)$, however this can be difficult (or impossible) to compute, and so we instead use an estimate $\hat{\rho}$ (where $|\hat{\rho}| \leq 1$) which is calculated as $\frac{\sum_{i=0}^{n-1}(t_i - \overline{t_i})(l_i - \overline{l_i})}{\sqrt{\sum_{i=0}^{n-1}(t_i - \overline{t_i})^2 \sum_{i=0}^{n-1}(l_i - \overline{l_i})^2}}$ for the set of traces T (containing n traces t_i) and hypothetical power model L, containing n hypothetical power consumption values l. Wrong guesses for the key will have correlations close to 0, while the correct guess will have $|\hat{\rho}|$ close to 1 (assuming the power model is accurate). We calculate $\hat{\rho}(\mathbf{V}, \mathbf{H})$, which leads to 16 correlation coefficients. Each correlation coefficient corresponds to a mask offset. By choosing the location where the max $\hat{\rho}(\mathbf{V}, \mathbf{H})$ occurs, we can guess the offset. Using the offset guess, we can predict the S-Box output and deploy a CPA attack to recover the key.

3. RESULTS

This attack is feasible since the device leaks the Hamming weight of the masks when they are loaded from memory. Once the masks are recovered, extracting the key is straightforward. Our attack requires 10.1 traces to fully recover the key, while an attack on an unprotected implementation requires 9.9 traces and can be considered as a lower bound regarding the number of traces. Our attack is close to that bound; the reason that we need slightly more traces is because we do not always correctly guess the offset. Comparing our offset guesses with the actual mask offsets, we were able to successfully guess the offset 91% of the time. Recall the estimation error of the mean in a Bernoulli process is $p\,(1-p)\,/n_{\text{rep}}$, where $p = 0.91$ and n_{rep} is the number of repetitions, namely $n_{\text{rep}} = 10{,}000$. The success rate is estimated over $10{,}000$ traces with accuracy $\approx 10^{-5}$.

3.1 Mask Recovery Success Rate

Figure 4(a) shows the success rate of recovering the mask for various signal-to-noise ratios (SNR). The probability of correctly guessing the offset at random is 1/16, or 6.25%: we exceed this value for all SNR $> 2^5$ (i.e., $\sigma_{\text{noise}} > 30$). Therefore, using our method is preferred to naively guessing for most noise levels.

3.2 Tweaking the Algorithm Parameters

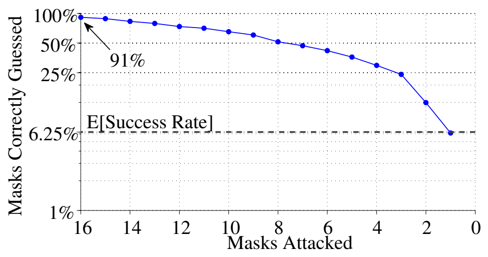


Figure 3: Mask recovery success rate as a function of number of masks attacked.

We examine how the algorithm parameters affect the mask recovery success rate. If only one mask (out of a possible 16) is attacked, the success rate equates to the expected value for naively guessing the mask. Indeed, with 1 mask, there is no "rotation" possible, hence the mask is "horizontally indistinguishable". Thus an attacker gains no advantage by attacking only 1 sample, since the extra computation time does not lead to an increase in success rate. However, attacking 2 masks, i.e., $\{m_0, m_1\}$, allows the pair to be distinguished with 11% success rate, slightly outperforming naive guessing. The success rate increases linearly as the number of masks increases, demonstrating the positive relationship between mask entropy and number of masks attacked.

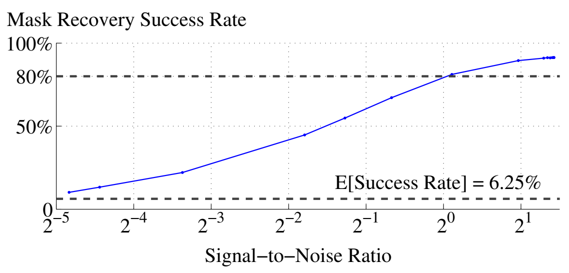
The attacker can also vary the width of the window where he suspects the masking operation to occur. Enlarging the window linearly increases the computational effort, i.e., increasing the width by n samples, leads to an attack complexity of $\mathcal{O}(n)$. Compare this to a second-order attack, where an increase in n samples requires $\frac{n(n-1)}{2}$ calculations [13].

3.3 Comparison with State-of-the-Art in the Presence of Noise

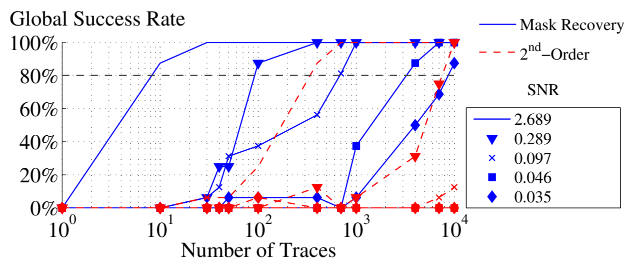
Noise increases the difficulty of carrying out a successful power attack, i.e., an attacker is required to measure more power traces. Common sources of noise include electronic noise from other circuit components, measurement errors, and clock jitter [11, 9]. Most of the noise in cryptographic devices can be approximated by a normal distribution $\sim \mathcal{N}(0, \sigma^2)$ [11]. In order to determine the influence of noise on our attack, we artificially corrupt the power traces by introducing additive white Gaussian noise $\sim \mathcal{N}(0, \sigma^2)$.

We compare our attack with a state-of-the-art second-order attack, namely the bivariate attack using a centered product as combination function [17]. This type of attack is ideal for software first-order masking schemes, and was proven to be optimal in the presence of noise [17].

Figure 4(b) shows the evolution of global success rate (GSR) as a function of number of traces attacked and signal-to-noise ratio (SNR). GSR is the probability to recover the full key. We define an attack as being successful if GSR $\geq 80\%$ [?]; conversely, we define a failed attack if the GSR fails to reach 80% within 10^5 traces. The best-case attack scenario is SNR $= 2.689$, i.e., no artificial noise is added. The best-case mask recovery attack requires 10 traces to succeed, whereas the best-case second-order attack does not succeed until 300 traces. The mask recovery attack is more resilient to noise since for a given number of power traces, the success rate will be higher for all SNR's. Regardless of the noise level, our mask recovery attack (empirically)



(a)



(b)

Figure 4: (a) Mask recovery success rate for 10^5 traces (b) Global success rate (GSR) vs. number of traces for different noise levels. The mask recovery attack outperforms the second-order attack by at least two orders of magnitude at every SNR.

reveals the key faster than a traditional bivariate attack.

The mask recovery attack outperforms the second-order attack by about two orders of magnitude for SNR ≥ 0.289. The second-order attack fails for SNR < 0.289, whereas the mask recovery attack succeeds for $0.035 \leq$ SNR ≤ 2.689. The lower performance of the second-order attack can be attributed to the leakage combination function. Indeed, by combining multiple leakages, the noise is amplified [17]. By choosing an optimal prediction function, the noise amplification can be minimized, but many more traces must be analyzed for a successful attack as shown in Figure 4(b).

4. CONCLUSION

We demonstrated how to recover a set of masks used in an AES-RSM software implementation. Our attack outperforms a traditional bivariate attack by two orders of magnitude, and can succeed even in heavy noise. We show how the attack parameters affect the success rate, namely that attacking just 2 (out of 16) yields a better mask recovery success rate versus naive guessing. It is not enough to say an implementation is first-order (or second-order, etc.) secure. Indeed, we showed that the countermeasure (genuine RSM implemented naively in software on a smartcard) can only defend against traditional first-order attacks.

5. REFERENCES

[1] S. Bhasin, J.-L. Danger, S. Guilley, and Z. Najm. Side-channel leakage and trace compression using normalized inter-class variance. In *HASP 2014*, pages 7:1–7:9, New York, NY, USA, 2014. ACM.

[2] A. Bogdanov. Multiple-differential side-channel collision attacks on AES. In E. Oswald and P. Rohatgi, editors, *CHES 2008*, volume 5154 of *LNCS*, pages 30–44. Springer, Heidelberg, 2008.

[3] E. Brier, C. Clavier, and F. Olivier. Correlation power analysis with a leakage model. In M. Joye and J.-J. Quisquater, editors, *CHES 2004*, volume 3156 of *LNCS*, pages 16–29. Springer, Heidelberg, 2004.

[4] C. Clavier, B. Feix, G. Gagnerot, M. Roussellet, and V. Verneuil. Horizontal correlation analysis on exponentiation. In *ICICS 2010*, pages 46–61, 2010.

[5] J.-S. Coron. Higher order masking of look-up tables. In P. Nguyen and E. Oswald, editors, *EUROCRYPT 2014*, volume 8441 of *LNCS*, pages 441–458.

[6] DPA Contest. `http://www.dpacontest.org`. [Online; accessed 30-May-2014].

[7] V. Grosso, F.-X. Standaert, and E. Prouff. Low Entropy Masking Schemes, Revisited. In *CARDIS 2013*, LNCS. Springer, nov 2013. Berlin, Germany.

[8] C. Herbst, E. Oswald, and S. Mangard. An AES smart card implementation resistant to power analysis attacks. In J. Zhou, M. Yung, and F. Bao, editors, *ACNS*, volume 3989 of *LNCS*, pages 239–252. Springer, Heidelberg, 2006.

[9] P. Kocher, J. Jaffe, B. Jun, and P. Rohatgi. Introduction to differential power analysis. *Journal of Cryptographic Engineering*, pages 5–27, 2011.

[10] P. C. Kocher, J. Jaffe, and B. Jun. Differential power analysis. In *CRYPTO 1999*, pages 388–397, London, UK, UK, 1999. Springer-Verlag.

[11] S. Mangard, E. Oswald, and T. Popp. *Power Analysis Attacks: Revealing the Secrets of Smart Cards.* Springer, 2008. ISBN 0-387-30857-1.

[12] M. Nassar, Y. Souissi, S. Guilley, and J.-L. Danger. RSM: A small and fast countermeasure for AES, secure against 1st and 2nd-order zero-offset SCAs. In *DATE 2012*, pages 1173–1178, 2012.

[13] E. Oswald, S. Mangard, C. Herbst, and S. Tillich. Practical Second-order DPA Attacks for Masked Smart Card Implementations of Block Ciphers. CT-RSA 2006, pages 192–207, Berlin, Heidelberg, 2006. Springer-Verlag.

[14] J. Pan, J. den Hartog, and J. Lu. You Cannot Hide behind the Mask: Power Analysis on a Provably Secure S-Box Implementation. In H. Youm and M. Yung, editors, *Information Security Applications*, volume 5932 of *LNCS*, pages 178–192. Springer, 2009.

[15] E. Prouff, C. Giraud, and S. Aumônier. Provably Secure S-Box Implementation Based on Fourier Transform. In L. Goubin and M. Matsui, editors, *CHES 2006*, volume 4249 of *LNCS*, pages 216–230. Springer, Heidelberg, 2006.

[16] E. Prouff and M. Rivain. A Generic Method for Secure SBox Implementation. In S. Kim, M. Yung, and H.-W. Lee, editors, *Information Security Applications*, volume 4867 of *LNCS*, pages 227–244. Springer, Heidelberg, 2007.

[17] E. Prouff, M. Rivain, and R. Bevan. Statistical analysis of second order differential power analysis. *IEEE Trans. on Computers*, 58(6):799–811, 2009.

[18] M. Rivain and E. Prouff. Provably secure higher-order masking of AES. In S. Mangard and F.-X. Standaert, editors, *CHES 2010*, volume 6225 of *LNCS*, pages 413–427. Springer, Heidelberg, 2010.

[19] M. Tunstall, C. Whitnall, and E. Oswald. Masking tables - an underestimated security risk. In *FSE 2013*, pages 425–444, 2013.

[20] X. Ye and T. Eisenbarth. On the Vulnerability of Low Entropy Masking Schemes. In A. Francillon and P. Rohatgi, editors, *CARDIS 2013, Berlin, Germany, November 27-29, 2013*, volume 8419 of *LNCS*, pages 44–60. Springer, 2013.