Detecting Side Channel Vulnerabilities in Improved Rotating S-Box Masking Scheme—Presenting Four Non-profiled Attacks

Zeyi Liu[1,2,3], Neng Gao[1,2], Chenyang Tu[1,2(✉)], Yuan Ma[1,2], and Zongbin Liu[1,2]

[1] Data Assurance and Communication Security Research Center, Beijing, China
{liuzeyi,gaoneng,tuchenyang,mayuan,liuzongbin}@iie.ac.cn
[2] State Key Laboratory of Information Security,
Institute of Information Engineering, CAS, Beijing, China
[3] University of Chinese Academy of Sciences, Beijing, China

Abstract. Improved Rotating S-box Masking (RSM2.0 for short) is a well-known countermeasure designed and implemented by DPA Contest V4.2 committee to provide security protection for AES-128. By combining both 1st-order masking and shuffling techniques, improved RSM claims to offer at least non-profiled resistance for its software implementation and up to now no systematic research has been published to challenge such security claim yet. To study the practical security of RSM2.0 against non-profiled attacks, we first propose an analytical methodology to guide the detection of the exploitable vulnerabilities in RSM2.0. On the basis of the methodology, several potential flaws hidden in both the algorithm design and detailed implementation of RSM2.0 are discovered and we make use of them to design six attacking schemes in total, all of which belong to non-profiled attacks. Four representative attacks are eventually implemented and submitted to DPA Contest V4.2 for official evaluation and the results show that all the submitted attacks are both practical and feasible. Among them, the best attack scheme requires only 257 power traces to crack the complete 128-bit master key with 80% success rate. To further improve the security level of RSM2.0, we also discuss some possible strategies to eliminate or mitigate the threats proposed by us.

Keywords: Side-channel analysis · 1st-order masking schemes · Shuffling · Non-profiled attack · Second order CPA · DPA Contest V4.2

1 Introduction

Adding side channel resistances is indispensable for modern cryptographic devices to thwart the potential attack first proposed by Kocher et al. in [1].

C. Tu—The work is supported by a grant from the National High Technology Research and Development Program of China (863 Program, No. 2013AA01A214) and the National Basic Research Program of China (973 Program, No. 2013CB338001). Besides, the work is also supported by a grant from the National Natural Science Foundation of China (No. 61402470).

R. Avanzi and H. Heys (Eds.): SAC 2016, LNCS 10532, pp. 41–57, 2017.
https://doi.org/10.1007/978-3-319-69453-5_3

The basic idea of the attack is to collect the observable leakages derived from the operations of sensitive intermediate values and make use of them with the help of statistical methods to deduce the hidden secret in the devices, generally the cryptographic key. Masking and shuffling are two classic countermeasures most extensively studied to enhance the security level of cryptographic devices. By bringing in random numbers, masking schemes [2–4] divide each sensitive intermediate value into several individual parts while keeping each part random. This scheme cuts off the relationship between the hidden secret and the direct leakage from sensitive intermediate value, thus efficiently resisting the common statistical analysis methods in side channel areas such as [1,5,6]. On the other hand, shuffling schemes [7,8] provide the side channel protection from another perspective, namely time dimension. With the help of randomized index table, shuffling schemes either disorder the execution path of cryptographic algorithm or randomly insert the dummy operations, thus randomizing the leakage position of each sensitive intermediate value and putting up obstacles to most of the analysis methods that mainly rely on the constant leakage instant.

To counteract possible attacks against one single defense strategy and achieve a higher security level, combining both masking and shuffling has been a tendency [3,9–11] in the design of side channel countermeasures. Among them, improved RSM [11] is a most recent and well-known countermeasure proposed by DPA Contest V4.2 committee to provide security protection for AES-128. Improved from the original RSM [12], RSM2.0 updates the original masking strategy with newly introduced offset array and performs shuffled operations with the help of the shuffle array, thus aiming to counteract the existing non-profiled attacks proposed in V4.1. An attack is considered to be non-profiled when adversaries don't have the chance to build the precise leakage model of the device they target in a previous training phase, such as DPA [1], CPA [5] etc. Thus, compared with classic profiled method, such as template attack [13] or stochastic model [14], non-profiled attacks usually require more power trace for secret extraction but show a higher security risk.

The combined countermeasure in RSM2.0 shows its resistance to non-profiled attacks in the following way. On the one hand, one byte of offset index is superseded by an offset array of sixteen bytes which determine the mask usage for all state[1] bytes independently. By this means, the second order attack proposed by Zhou et al. [15] which combines S-box output with input mask m_i and the masked plaintext with mask m_{i+1} doesn't work anymore. It's also impossible to exploit the significant power difference when operating on mask 0xFF and 0x00 for offset recovery [16] or to perform two kinds of constructive collision attacks proposed also in [16] since the relationship of mask usage within the first and last round has been cut off. On the other hand, the employment of shuffle array completely disorders the predictable sequence of mask S-box execution both in the first and last round, causes changeable execution window of the concerned S-boxes and results in the obstacle to perform constant instant related attacks, such as 1st-order attack [17] or second order attack which relies on the leakage

[1] State denotes the basic data unit of 4 by 4 matrix as defined in standard AES-128.

preprocessing of two instantaneous moments. Furthermore, V4.2 committee also rewrites the implementation in assembly code and precharges the specific state registers before overwritten by new numbers. Thus the constructive first-order attack proposed in [18] by exploiting hamming distance leakage between the input and output of masked S-box doesn't work anymore.

According to the official evaluation, we are the first to launch the non-profiled attacks against RSM2.0, and this is also the first paper to systematically analyze the potential non-profiled vulnerabilities hidden in RSM2.0. Although some other attack schemes have also been submitted to official website [19], almost all of them belong to profiled schemes where attackers have to perform a training phase with large quantities of power traces in order to characterize the real leakage model of the targeted device. Such kind of attacks are capable to recover secret key within several power traces but require a stronger assumption for the abilities of the attackers. In this paper, we are only dedicated in the attacks of non-profiled type.

The contribution of this paper mainly lies in the following aspects. We make use of an analytical methodology to guide the search of exploitable flaws both in the algorithm design and implementation of RSM2.0. Then, on the basis of the discovered non-profiled flaws, we come up with several attack schemes, more precisely second order schemes and its variants. Four of the attacks are eventually implemented as examples to validate the usability of the flaws. Official results show that all of our uploaded attacks are both feasible and practical. And the best scheme require only 257 traces to recover the AES-128 master key with 80% global success rate(GSR), thus breaking the security claim of RSM2.0 for the first time. Furthermore, in order to eliminate or mitigate the threats proposed by us, some possible countermeasures are also discussed in this paper.

The rest of the paper is organized as follows. In Sect. 2 we review the detailed algorithm design of RSM2.0, especially the countermeasures newly added to prevent the enhanced implementation from some known attacks. Then, in Sect. 3, we first explain the analytical methodology used to restrict the range of vulnerability detection and then point out several potential flaws hidden in either the design or implementation of RSM2.0. Besides, the reasons of flaw generation are also clearly explained in this section. Afterwards, in Sect. 4, we show our practical attack processes together with the official evaluation results of our four exemplary attacks. Furthermore, the discussion of some possible countermeasures is presented in Sect. 5. Finally, we conclude our work in the last section of the article.

2 Improved RSM Scheme

In this section, the algorithm details of improved RSM are described explicitly. The description focuses on the mask usage and tracking in the algorithm flow and also on some important features newly brought in, including the shuffling countermeasure and the offset array. What's worth mentioning is that, for the simplicity of description, we omit "modulo 16" after all the addition operation used hereafter unless special explanation is made.

2.1 Algorithm Description

Mask array, offset array and shuffle array, denotes as M[], O() and Sf[] respectively, are three core components to build up the whole RSM2.0 scheme.

Mask array is designed to be a fixed and publicly known array of 16 bytes. The latest values are chosen meticulously with the goal to not only minimize the mutual information leakage [20] but also take side-channel indistinguishability [21] into consideration. We denote each individual value in the array as M[i], $i \in [0, 15]$, and the whole mask array can be specified as $\{0x03, 0x0c, 0x35, 0x3a, 0x50, 0x5f, 0x66, 0x69, 0x96, 0x99, 0xa0, 0xaf, 0xc5, 0xca, 0xf3, 0xfc\}$.

Offset array contains 16 four-bit random numbers which range from 0 to 15 and are refreshed in each encryption. It cooperates with Mask array to randomly and independently select mask values which provide the initial protection for the input state in each encryption round. That is, according to each offset byte, noted as O(i), $i \in [0, 15]$, mask M[O(i)+r] is picked out and later Xored with the sensitive input variable at the start of each encryption round, where $r \in [0, 9]$ represents the round index. We denote such input masks in each round as $Mask_r$, and it can be represented in the form of a mask state:

$$Mask_r = \begin{pmatrix} M[O(0)+r] & M[O(4)+r] & M[O(8)+r] & M[O(12)+r] \\ M[O(1)+r] & M[O(5)+r] & M[O(9)+r] & M[O(13)+r] \\ M[O(2)+r] & M[O(6)+r] & M[O(10)+r] & M[O(14)+r] \\ M[O(3)+r] & M[O(7)+r] & M[O(11)+r] & M[O(15)+r] \end{pmatrix}$$

Shuffle array is a new feature introduced in RSM2.0. It is refreshed trace by trace and kept secret to analysts as offset array does. What's different is that Sf[] is a random permutation of [0, 15] and is deployed to disorder the non-linear transformation of 16 S-boxes together with the subsequent linear layer operation, namely ShiftRows, both in the first and last round. In fact, two separate shuffle arrays are deployed which are defined as Sf0[] and Sf10[] by us to distinguish the position of their usage.

After the description of the three fundamental arrays, the round functions in RSM2.0 are explained below. Apart from the unchanged AddRoundKey(AR) function, the other round functions can be divided into two categories, namely the non-linear and linear layer functions.

- **Non-linear layer function:**
  The only one function that belongs to the non-linear layer is MaskedSubBytes(MS). Unlike standard SubBytes function in AES, MaskedSubBytes consists of sixteen different and reconstructed S-boxes corresponding to both the input mask and the output mask. Each masked S-box can be defined as $MaskedSubByte_{mm'}[x] = SubByte[x \oplus m] \oplus m'$, where m and m' represent the input and output mask byte respectively. Specifically in RSM2.0, m and m' are designed to be two successive masks in $M[]$. That is, each new S-box can be denoted as $MaskedSubByte_{M[i]M[i+1]}[]$ ($MaskedSubByte_i[]$ for short), $i \in [0, 15]$, and can be previously computed due to the already

known $M[]$. Thus the input mask state $Mask_r$ becomes traceable (switch to $Mask_{r+1}$) when going through the non-linear layer and such special and circular way of mask usage for S-box reconstruction is called the Rotating S-boxes Masking (RSM).

- **Linear layer functions:**
  Linear layer is composed of three functions in total, namely ShiftRows(SR) and MixColumns(MC), and also the additionally introduced MaskCompensation(MCP). On the one hand, the first three functions keep unchanged as in the standard AES. The only difference is that all of their inputs and outputs are protected with masks to randomize all the sensitive intermediate values in the practical encryption and more importantly, these masks are naturally traceable due to the linear property of all these functions.

  On the other hand, the MaskCompensation function is newly introduced to eliminate the derived output masks after the MixColumns function and simultaneously re-mask the intermediate variable with the input masks of the next round. To achieve this goal, the compensation mask, denoted as $MaskCompensation_r(MCP_r$ for short), $r \in [0, 8]$, should be first generated and can be expressed as:

$$MCP_r = MC(SR(Mask_{r+1})) \oplus Mask_{r+1}$$

  Then the MaskCompensation happening in the first nine rounds can be described in the following derivation process:

$$\begin{aligned}
&MC(SR(MS(K_r \oplus X_r \oplus Mask_r))) \oplus MCP_r \\
&= MC(SR(SubBytes(K_r \oplus X_r) \oplus Mask_{r+1})) \oplus MCP_r \\
&= MC(SR(SubBytes(K_r \oplus X_r))) \oplus MC(SR(Mask_{r+1})) \\
&\oplus MC(SR(Mask_{r+1})) \oplus Mask_{r+1} \\
&= MC(SR(SubBytes(K_r \oplus X_r))) \oplus Mask_{r+1}
\end{aligned}$$

  The only change in MaskCompensation happens in the last round where MixColumns is omitted and no next round input masks are needed due to the requirement of the unmasked and correct ciphertext output. Thus $MaskCompensation_9$ satisfies:

$$MaskCompensation_9 = ShiftRows(Mask_{10})$$

With the explanation of functions in both the linear layer and the non-linear layer, all the major process of RSM2.0 has been clearly demonstrated. And more detailed and complete algorithm of RSM2.0 is presented in Appendix 1.

2.2 Acquisition Platform and Measurements

The measurement of all the official power traces is completed on a 8-bit AVR microcontroller Atmega163 embedded in a smartcard. It contains 16 Kb of in-system programmable flash, 512 bytes of EEPROM, and 32 general purpose

working registers. The acquisition of traces is performed through a LeCroy WaveRunner 6100 A oscilloscope by the use of an EM probe. The sampling rate FS equals to 500 MS/s and the acquisition bandwidth is 200 MHz.

3 Detecting Non-profiled Vulnerabilities in RSM2.0

In this section, we explain the analytical methodology we comply with to lead the search of the non-profiled vulnerabilities and show the discovered exploitable leakages and the reason of their generation. Our discoveries validate that the improved countermeasure are far from perfect to counteract the type of non-profiled attacks. What's worse, some of the newly added defense mechanisms even directly result in the attacks presented in this section.

3.1 Analytical Methodology for Vulnerability Detection

Although 1st-order masking schemes can not resist 2nd-order attack theoretically, RSM2.0 puts targeted obstacles to such kind of attack. On the one hand, by protecting each state byte with an independent and randomly indexed mask, none of the two masked intermediates in the algorithm share the same mask part. On the other hand, the common attacking points, namely non-linear layer transformation, and also the subsequent ShiftRows operation are both implemented with shuffled order thus making it difficult to collect the power leakage from those parts.

In order to follow the traditional second order idea and achieve a better performance, we are not only expected to discover the new attacking point for mask elimination but also expected to bypass the shuffle countermeasure in order to avoid the costly *integrated-and-combined* strategy [9,22] for second order attack.

To accomplish the above target, we first perform the vulnerability detection process in both the algorithm design and the code implementation of RSM2.0. Our detection mainly comply with the following guidelines aiming at obtaining optimal attacking performance:

1. Restricting the range of detection in the intermediate values that contain only 8-bit subkey. Although the direct side channel attack against larger subkey block is possible [23], the expensive resource overheads, such as GPU acceleration and huge memory usage make it inefficient and unsuitable in the official evaluation platform. Thus, to acquire better attacking performance we focus on 8-bit subkey recovery at a time.
2. The predictable intermediates utilized by the attackers should be the results of the non-linear layer transformation. The characteristics of the non-linear transformation lead to the fact that each single bit guessing error of the target subkey would influence as much bits of the prediction result as possible. Thus, making it easier to distinguish the correct key from others with less power traces.
3. The security of the newly introduced MaskCompensation process should be taken into consideration additionally.

Based on the guidelines above, the vulnerability detection can be simply limited in the following sensitive regions (Fig. 1).

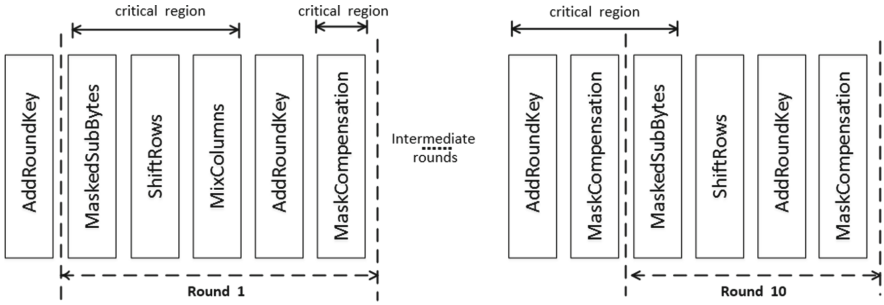


Fig. 1. Critical regions for vulnerability detection

3.2 Flaws in the Algorithm Design

Online Derivation of Compensation Mask. The first vulnerability we discover in the critical regions lies in the entire process of MCP_r derivation which must be implemented online in RSM2.0. The inevitable online feature is caused by the replacement of the offset index by the offset array $O()$. In the original RSM, the input mask state of round r, which we denote as $Mask_{idx,r}$, is uniquely determined by a 4-bit index denoted by idx. More accurately, such input mask state satisfies the following formula:

$$\begin{pmatrix} M[idx+0+r] & M[idx+4+r] & M[idx+8+r] & M[idx+12+r] \\ M[idx+1+r] & M[idx+5+r] & M[idx+9+r] & M[idx+13+r] \\ M[idx+2+r] & M[idx+6+r] & M[idx+10+r] & M[idx+14+r] \\ M[idx+3+r] & M[idx+7+r] & M[idx+11+r] & M[idx+15+r] \end{pmatrix}$$

Due to the fact that $M[]$ is a fixed mask array, there are only 16 possible values for $Mask_{idx,r}$ state. Therefore, the compensation mask state which is completely dependent on $Mask_{idx,r}$, is actually derived offline and stored previously for later use. However, the input mask state $Mask_r$ in RSM2.0 utilizes offset array $O()$ to index the selected mask for each input mask byte as stated in Sect. 2. Each element $O(i)$ in the offset array is independent and identically distributed, thus resulting in 16^{16} possible values for the $Mask_r$ state. In order to store all these compensation values derived from $Mask_r$, $16 * 16^{16}$ bytes of storage space is required which is unreachable in the embedded devices with constrained resources. Thus, online compensation mask derivation becomes indispensable and all of its related power consumption would be recorded in the power traces.

The other significant cause that leads to this online process further exploitable is the omission of shuffling protection during this stage. This vulnerability is serious since it means all the steps during the derivation of compensation mask, including the loading of $Mask_r$, the derivation for the first part

of compensation mask and the compensation mask generation, are processed at the constant time instants and leak the power consumptions corresponding to the original input $Mask_r$ or its intermediate state during the transformation. We briefly show all the available mask leakages in Fig. 2.

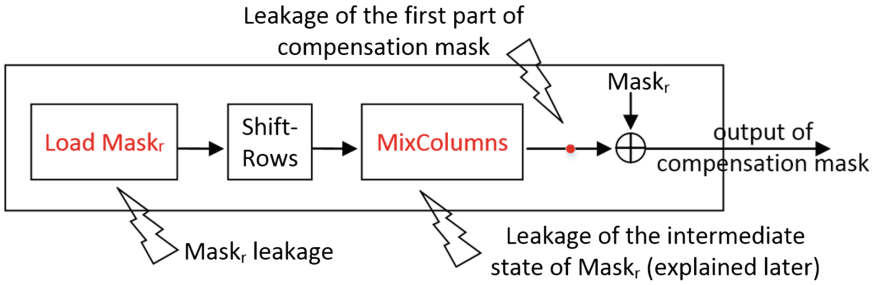


Fig. 2. Exploitable mask leakages in the derivation of compensation mask (Color figure online).

AddRoundKey Function Flaw. In addition to compensation mask derivation, AddRoundKey operation in RSM2.0 also lacks shuffling protection, thus causing potential second order threats. The exploitable loophole caused by this sequential process appears at the end of the ninth round, where the AddRoundKey function is performed right after the MixColumns of the current round. That is to say, each output byte of AddRoundKey is protected by the same mask as in MixColumns output, as shown in Fig. 3, where the X_i represents the input bytes of the ninth round and K_i, K_i' are the subkey bytes of the eighth and ninth round respectively. Special note is that such output masks (in red) are also generated as the first part of the compensation mask in the online mask derivation phase (the third leakage point in Fig. 2).

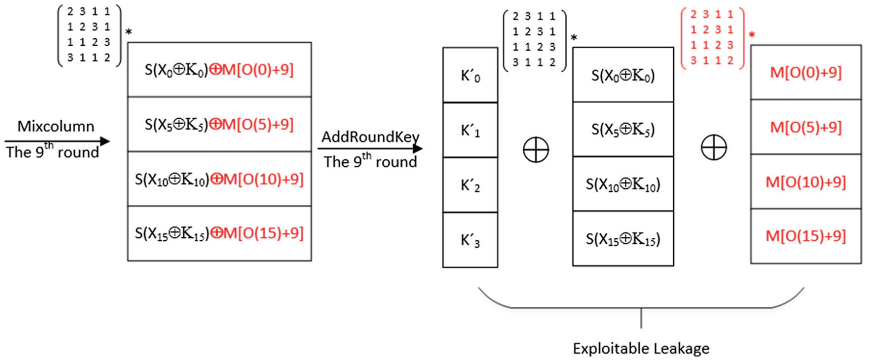


Fig. 3. Exploitable leakages in AddRoundKey, taking the first column as an example.

3.3 Flaws in the Implementation Level

Location of MaskCompensation. Flaws in the implementation level appear because of the inserted position of the online MaskCompensation. More precisely, in the source code of RSM2.0 implementation, the MaskCompensation of the current round starts operating after AddRoundKey has finished, just before the MaskedSubBytes of the next round, as shown in Fig. 4. This seemingly negligible implementation order does matter since all the masked variables after the ninth round AddRoundKey can be derived reversely from the known ciphertext by only guessing 8-bit subkey of the last round key, which is a proper subkey guessing space for side channel attacks. Besides, the MaskCompensation here would switch the protection masks of its output state to $Mask_9$ which has also leaked its power consumption at the start of the ninth round MaskCompensation (as shown in Fig. 2).

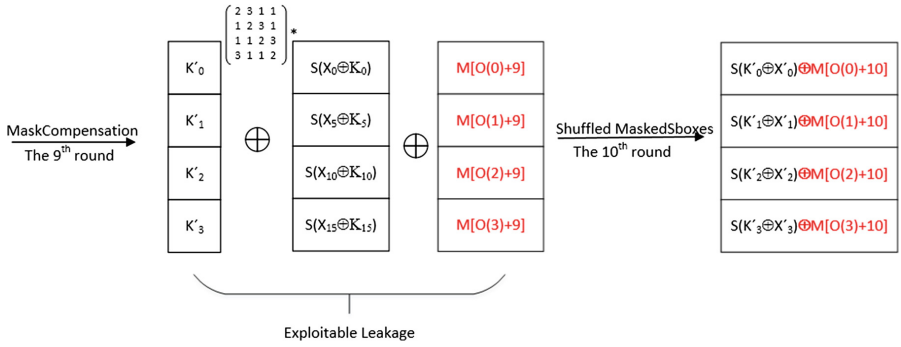


Fig. 4. Exploitable leakages at the output of MaskCompensation, taking the first column as an example.

Flaws in Linear Layer Function. More critical security flaws appear at linear layer. Since almost all of the proposed attacks against original RSM select the non-linear function, i.e. masked S-box, as their attacking point, designers of RSM2.0 pay too much attention to the protection of S-box execution while on the other hand ignore the potential security risk in the linear layer transformation.

The flaw we find in the linear layer appears in the process of MixColumns operation in the first encryption round. The essential reason that gives rise to this flaw actually lies in two aspects. The first one is that both the MixColumns function used for encryption and the MixColumns included in the compensation mask derivation stage share the same assembler code. The only difference is the input variables, namely $ShiftRows(X \oplus K \oplus Mask_1)$ and $ShiftRows(Mask_1)$ respectively. Also due to the linear characteristics of the MixColumns operation, such shared implementation implies that whenever a masked intermediate value is used or generated in the MixColumns for encryption, the corresponding mask itself, which is used for the protection of that intermediate value, will also appear in the exactly same position of the MixColumns for compensation mask

derivation. The second reason that makes the flaws further feasible is that the entire MixColumns code is implemented in a fixed sequence, which means that attackers are able to find out leakages in constant time instant, thus meeting the prerequisite of classic second order attacks.

4 Practical Attacks and Official Evaluation Results

With the clear explanation of all the exploitable vulnerabilities in the section above, six non-profiled attacks, more precisely second order attacks and the variants, can be launched. We selectively perform four of them as examples to validate the usability of the discovered flaws and show the effectiveness of our attack schemes by citing the official evaluation data.

The basic idea of classic second order attacks, as present in [22,24,25], is to preprocess the leakages from two parts of the power traces and both parts respectively correspond to the intermediate variables protected with the exact same mask. After preprocessing, the combined leakage is relevant to the unprotected intermediate value, and thus making the later correlation attack feasible. The performance of different preprocessing functions is studied in [26] and our attacks follow the preprocessing method of improved product combining proposed in this paper.

4.1 Second Order Attacks in the First Round[2]

We present two attacks in the first round and both of them make use of the leakages from the execution of shared MixColumns source code in the encryption and MCP_r derivation stage. To better understand the attacks, we briefly introduce the implementation approach of MixColumns in RSM2.0.

Suppose $(V_{0,j}, V_{1,j}, V_{2,j}, V_{3,j})^T$ is a column of 4 input bytes, which serves as a basic unit for MixColumns transformation. Then the output of the transformation, where $i \in [0,3]$, $j \in [0,3]$, can be formalized and recombined as:

$$\begin{aligned} V_{i,j} &= (2 * V_{i,j}) \oplus (3 * V_{(i+1)\%4,j}) \oplus V_{(i+2)\%4,j} \oplus V_{(i+3)\%4,j} \\ &= (2 * V_{i,j} \oplus 2 * V_{(i+1)\%4,j}) \oplus V_{(i+1)\%4,j} \oplus V_{(i+2)\%4,j} \oplus V_{(i+3)\%4,j} \\ &= V_{i,j} \oplus (V_{1,j} \oplus V_{2,j} \oplus V_{3,j} \oplus V_{4,j}) \oplus 2 * (V_{i,j} \oplus V_{(i+1)\%4,j}) \end{aligned}$$

The implementation follows the process of the calculation in the last line. That is, $V_{1,j} \oplus V_{2,j} \oplus V_{3,j} \oplus V_{4,j}$ is firstly generated and shared for all the bytes in column j. Then, to derive a specific byte $V_{i,j}$, the generation of $(V_{i,j} \oplus V_{(i+1)\%4,j})$ is subsequently completed. This value is later used as the input of the lookup table Xtime which is previously calculated to store 2*X (under $GF(2^8)$) in the position of X, where X ranges from 0 to 255. Thus, after going through Xtime table, $2*(V_{i,j} \oplus V_{(i+1)\%4,j})$ is acquired. Finally, by Xoring together $(V_{1,j} \oplus V_{2,j} \oplus V_{3,j} \oplus V_{4,j})$, $2*(V_{i,j} \oplus V_{(i+1)\%4,j})$ and the original value $V_{i,j}$ stored in the register, the newly updated $V_{i,j}$ comes into being.

[2] For the need of expression, all the "mod" operation would be explicitly added in this subsection.

Attacking Xtime Input. During the generation of the Xtime input ($V_{i,j} \oplus V_{(i+1)\%4,j}$), $V_{i,j}$ is first loaded into the register and simultaneously leaks instantaneous power consumption. When it happens in the encryption process, $V_{i,j}$ actually equals to one of the output variables of MaskedSubBytes function. Thus all the MaskedSubBytes outputs in the form of $S(X_i \oplus K_i) \oplus M[(O(i)+1)\%16], i \in [0, 15]$, leaks information. On the other hand, when it goes to MaskCompensation function, $V_{i,j}$ in fact represents one of the mask bytes in $Mask_1$ state. Thus, $M[(O(i) + 1)\%16]$ leaks its information as well. By preprocessing both of the leakages, the combined leakage would be relevant to the following intermediate variable:

$$(S(X_i \oplus K_i) \oplus M[(O(i) + 1)\%16]) \oplus M[(O(i) + 1)\%16] = S(X_i \oplus K_i), i \in [0, 15]$$

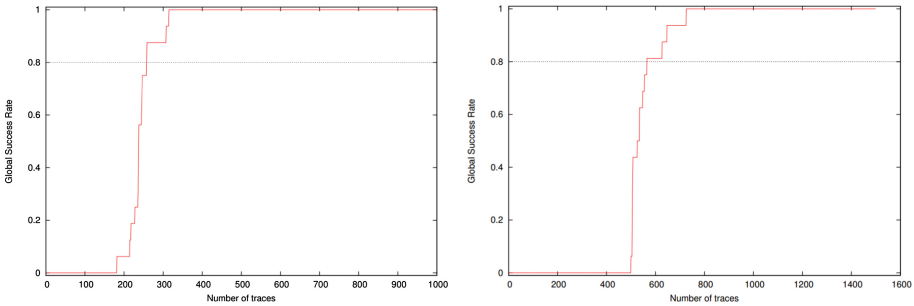
which is an unprotected and predictable value, appropriate for the traditional CPA attack.

Official evaluation results show that, the second order attack proposed here are both feasible and practical. Based on the published performance parameter, merely 258 traces are required (Fig. 5(a)) to recover all of the 128-bit master key with 80% success rate (the so-called 80% global success rate, GSR [27]) and only 210 traces would suffice to reduce the maximum partial guessing entropy(PGE [27]) under 10, which means that the remaining key guessing space is less than 10^{16}.

Optimized Chained Attack. Unlike the attack above, here we utilize the leakage exposed when generating each complete input byte of the Xtime, namely ($V_{i,j} \oplus V_{(i+1)\%4,j}$). After the same combination of the leakages in MixColumns of different stages, the united leakage would also be related to a predictable intermediate value which is involved with 16 bits of the master key. Taking the first column as an example, four predictable values are $S(X_0 \oplus K_0) \oplus S(X_5 \oplus K_5)$, $S(X_5 \oplus K_5) \oplus S(X_{10} \oplus K_{10})$, $S(X_{10} \oplus K_{10}) \oplus S(X_{15} \oplus K_{15})$ and $S(X_{15} \oplus K_{15}) \oplus S(X_0 \oplus K_0)$ respectively, where X_i and K_i are the ith plaintext and master key respectively. Direct attacks by guessing the first and the third predictable value or guessing the second and the fourth one could recover all the subkeys in this column but the key guessing space is $2 * 2^{16}$ in total.

In order to further reduce the computation overhead, we optimized the attack process in a chained way. That is, we first attack K_0 by using the leakages and method mentioned first in this subsection, then with the most probable guessing K_0 revealed, predictable value $S(X_0 \oplus K_0) \oplus S(X_5 \oplus K_5)$ is utilized to extract K_5 only. The same approach then goes on for the second predictable value, where this time, K_5 is fixed to the most probable value obtained in the last step and K_{10} becomes the only key which needs to be guessed in this step. Finally, K_{15} is also revealed in the same way. By this means, the key guessing space for extracting four subkeys in a column is now reduced to $4 * 2^8$ and the subkeys in other three columns can be inferred similarly as in the first column.

Figure 5(b) depicts the official evaluation result of such chained second-order attack. It shows that such method does work, and 565 traces are needed to uniquely determine the master key with 80% success rate(80% GSR).



(a) GSR for MixColumn input attack. (b) GSR for optimized chained attack.

Fig. 5. Official evaluation results of Global Success Rate, GSR.

Note also that the similar second order and chained attack can also be launched at the position of Xtime output, namely $2 * (V_{i,j} \oplus V_{(i+1)\%4,j})$, and the generation of shared $(V_{1,j} \oplus V_{2,j} \oplus V_{3,j} \oplus V_{4,j})$ respectively. The only difference lies in the form of the predictable intermediate value, which is selected and utilized by the attackers to infer the subkey. To avoid repetition, we don't describe them here anymore.

4.2 Second Order Attacks in the Ninth Round

Attacking MaskCompensation Output. The attack here utilizes the implementation level flaw explained in Sect. 3.3. Due to the inserted position of MaskCompensation, each individual output byte here contains only 8-bit subkey when derived from the ciphertext and is actually protected by the corresponding mask in $Mask_9$. Besides, $Mask_9$ has also been loaded byte by byte when generating the ninth round compensation masks as depicted in Fig. 2. Thus by combining the leakages from both positions, the preprocessed leakage has relevance to the following values which are predictable from the perspective of side channel opponents:

$$(Sbox^{-1}ShiftRows^{-1}[C \oplus K_L] \oplus Mask_9) \oplus Mask_9 = Sbox^{-1}ShiftRows^{-1}[C \oplus K_L]$$

where C refers to the output ciphertext state and K_L denotes the last round key. After 16 bytes of K_L is revealed completely, a reversed key expansion process of AES should be performed in order to fetch our final target, namely the 128-bit master key.

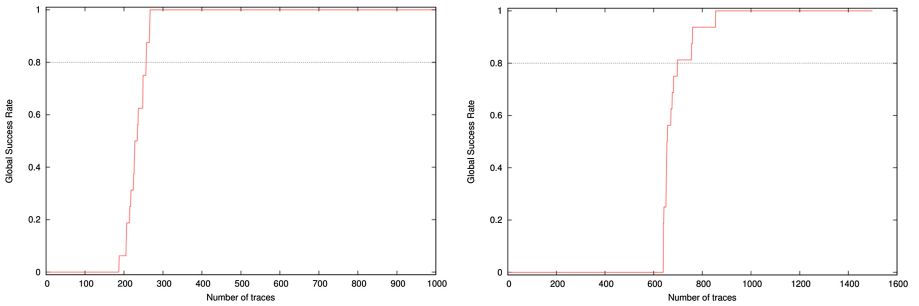
Figure 6(a) depicts the GSR tendency of this attack evaluated by DPA Contest committee. It shows that the attack by exploiting the implementation level flaw is of high efficiency that only 257 traces can meet the requirement for 80% GSR. Besides, to reduce the Max PGE under 10, only 205 traces are required.

Attacking AddRoundKey Output. The only difference between the output of the ninth round AddRoundKey and MaskCompensation lies in the masks for protection. For AddRoundKey function, these masks are obtained by transforming the input mask state of the ninth round, namely $Mask_8$, through MaskedSubBytes, ShiftRows and MixColumns in sequence. Thus, the AddRoundKey output in the ninth round can be derived from the ciphertext C:

$$Sbox^{-1}ShiftRows^{-1}[C \oplus K_L] \oplus MixColumns[ShiftRows[Mask_9]]$$

On the other hand, during the construction of the compensation mask in the current round, the first compensation part – $MixColumns[ShiftRows[Mask_9]]$ should be generated first as shown in Fig. 2.

Likewise, by performing second order attacks at both parts, the last round key K_L involved in the predictable variable $Sbox^{-1}ShiftRows^{-1}[C \oplus K_L]$ can be recovered and the master key would also be deduced instantly with the help of the key expansion process but in a reversed way. The official evaluation of GSR in Fig. 6(b) shows that such attack is also feasible in RSM2.0, and the number of traces needed to acquire 80% GSR is 698.



(a) GSR for MaskCompensation attack. (b) GSR for AddRoundKey output attack.

Fig. 6. Official evaluation results of Global Success Rate, GSR.

5 Discussion of Possible Countermeasures

In order to mitigate or even get rid of the non-profiled threats presented in this article, we propose the following coping strategies. All of the strategies follow the basic principle that both leakages exploited in a single second order attack should be protected individually by either eliminating the source of the leakage or by randomizing the position of its appearance.

1. **Adding resistance in the ninth round:** exchanging the order of the AddRoundKey and MaskCompensation function in the ninth round can be the first step to enhance the security level of RSM2.0. In fact, compared to the original attack aiming at the MaskCompensation output leakage, now attackers have to deduce each byte of hypothetical intermediate state, namely $Sbox^{-1}ShiftRows^{-1}[C \oplus K_L] \oplus K_P$, by searching 16-bit subkey space, where K_L and K_P represent the last and penultimate round key respectively.
   After exchanging, the output states of both functions are protected with $Mask_9$. In order to further prevent the second order threat, the sequential

operations of loading $Mask_9$, Xoring compensation masks and AddRoundKey execution should all be respectively shuffled. Fortunately, such protections could be easily added since every operation mentioned above deals with 16 state bytes independently.

2. **Adding resistance in the first round:** the shared MixColumns code in the compensation mask derivation and the original encryption part leads to our attacks in the first round. To plug such leakages, one possible method is to implement MixColumns in different ways. For example, alter the MixColumns implementation in the encryption part by following the process of the original transformation formula, namely $V_{i,j} = 2 * V_{i,j} \oplus 3 * V_{(i+1)\%4,j} \oplus V_{(i+2)\%4,j} \oplus V_{(i+3)\%4,j}$. Thus all of the chained attacks mentioned can be surely counteracted since the derived intermediates in different implementations don't share the same masks any more.

   To further prevent the exploitable leakages caused by the direct loading of MixColumns input bytes for calculating each output byte, the shuffled generation order for MixColumns output bytes could be considered as a possible solution.
3. **Special note is that:** no extra shuffle array(excluding Sf0[] and Sf10[]) is needed to perform the shuffling protection suggested above. The only thing we need to do is to distinguish the shuffle array used in the region of compensation mask derivation from that used to disorder the leakage of masked intermediate values. By this means, the pair of leakages utilized to perform certain second order attack mentioned in this paper may now come from 16^2 possible combinations of time instants, thus significantly increasing the difficulty for target leakage location and even causing huge overhead (number of power traces) when using advanced *integrated-and-combined* technique to accumulate the shuffled leakages as mentioned in [9,22].

6 Conclusion

This is the first paper to systematically analyze the non-profiled vulnerabilities in RSM2.0. To achieve the goal, we first propose the analytical methodology to guide the vulnerability detection and then scrutinize both the algorithm design and implementation details of the RSM2.0 countermeasure. Based on all the vulnerabilities newly found, several attacking schemes are proposed and four of them are finally implemented as examples and submitted for official evaluation.

The evaluation reports show that all of our proposed attacks are both feasible and practical. Thus, this study validates in the first time that the currently used countermeasures and implementation are still insufficient to provide RSM2.0 with non-profiled resistance.

To further fix the vulnerabilities described in this paper, we come up with several corresponding suggestions which either try to eliminate the second order leakages or to extend the protection region of shuffle countermeasure. These improvements are not unique but they can be considered as a general direction to set up obstacles for potential attackers and to further improve the security level of RSM2.0, especially against non-profiled attacks.

A Algorithm of Improved Rotating S-Boxes Masking

Algorithm 1. Improved rotating S-boxes masking scheme

Input:
16-bytes plaintext $X = [X0, X1, \ldots, X15]$;
One offset array: $O(i), i \in [0, 15]$,
(uniformly random, unknown);
Two shuffle arrays: $Sf0[i], Sf10[i], i \in [0, 15]$,
(uniformly random permutations, unknown);
Output:
16-bytes ciphertext $X = [X0, X1, \ldots, X15]$;

On-the-fly key expansion $RoundKey_r, r \in [0, 10]$,
$RoundKey_0 \leftarrow RoundKey_0 \oplus Mask_0$
$X = X \oplus RoundKey_0$
/*All rounds but the last one*/
for each $i \in [0, 8]$ **do**
  if $r = 0$ **then**
    for $i \in Sf0[0, 15]$ **do**
      $X_i = MaskedSubBytes_{O(i)+r}(X_i)$
      $X_i = SR(X_i)$
    end for
  else
    for $i \in [0, 15]$ **do**
      $X_i = MaskedSubBytes_{O(i)+r}(X_i)$
      $X_i = SR(X_i)$
    end for
  end if
  $X = MC(X)$
  /* AddRouondKey of the next round */
  $X = X \oplus RoundKey_{r+1}$
  $MCP_r = MC(SR(Mask_{r+1})) \oplus Mask_{r+1}$
  $X = X \oplus MCP_r$;
end for
/* Last round */
for $i \in Sf10[0, 15]$ **do**
  $X_i = MaskedSubBytes_{O(i)+9}(X_i)$
  $X_i = SR(X_i)$
end for
$X = X \oplus RoundKey_{10}$ /* Ciphertext unmasking */
$MCP_9 = SR(Mask_{10})$
$X = X \oplus MCP_9$;

References

1. Kocher, P., Jaffe, J., Jun, B.: Differential power analysis. In: Wiener, M. (ed.) CRYPTO 1999. LNCS, vol. 1666, pp. 388–397. Springer, Heidelberg (1999). doi:10.1007/3-540-48405-1_25
2. Coron, J.-S., Goubin, L.: On Boolean and arithmetic masking against differential power analysis. In: Koç, Ç.K., Paar, C. (eds.) CHES 2000. LNCS, vol. 1965, pp. 231–237. Springer, Heidelberg (2000). doi:10.1007/3-540-44499-8_18
3. Herbst, C., Oswald, E., Mangard, S.: An AES smart card implementation resistant to power analysis attacks. In: Zhou, J., Yung, M., Bao, F. (eds.) ACNS 2006. LNCS, vol. 3989, pp. 239–252. Springer, Heidelberg (2006). doi:10.1007/11767480_16
4. Rivain, M., Prouff, E.: Provably secure higher-order masking of AES. In: Mangard, S., Standaert, F.-X. (eds.) CHES 2010. LNCS, vol. 6225, pp. 413–427. Springer, Heidelberg (2010). doi:10.1007/978-3-642-15031-9_28
5. Brier, E., Clavier, C., Olivier, F.: Correlation power analysis with a leakage model. In: Joye, M., Quisquater, J.-J. (eds.) CHES 2004. LNCS, vol. 3156, pp. 16–29. Springer, Heidelberg (2004). doi:10.1007/978-3-540-28632-5_2
6. Gierlichs, B., Batina, L., Tuyls, P., Preneel, B.: Mutual information analysis. In: Oswald, E., Rohatgi, P. (eds.) CHES 2008. LNCS, vol. 5154, pp. 426–442. Springer, Heidelberg (2008). doi:10.1007/978-3-540-85053-3_27
7. Mangard, S., Oswald, E., Popp, T.: Power Analysis Attacks: Revealing the Secrets of Smart Cards. Springer Science & Business Media, Heidelberg (2008)
8. Veyrat-Charvillon, N., Medwed, M., Kerckhof, S., Standaert, F.-X.: Shuffling against side-channel attacks: a comprehensive study with cautionary note. In: Wang, X., Sako, K. (eds.) ASIACRYPT 2012. LNCS, vol. 7658, pp. 740–757. Springer, Heidelberg (2012). doi:10.1007/978-3-642-34961-4_44
9. Tillich, S., Herbst, C., Mangard, S.: Protecting AES software implementations on 32-bit processors against power analysis. In: Katz, J., Yung, M. (eds.) ACNS 2007. LNCS, vol. 4521, pp. 141–157. Springer, Heidelberg (2007). doi:10.1007/978-3-540-72738-5_10
10. Rivain, M., Prouff, E., Doget, J.: Higher-order masking and shuffling for software implementations of block ciphers. In: Clavier, C., Gaj, K. (eds.) CHES 2009. LNCS, vol. 5747, pp. 171–188. Springer, Heidelberg (2009). doi:10.1007/978-3-642-04138-9_13
11. Bhasin, S., Bruneau, N., Danger, J.-L., Guilley, S., Najm, Z.: Analysis and improvements of the DPA contest v4 implementation. In: Chakraborty, R.S., Matyas, V., Schaumont, P. (eds.) SPACE 2014. LNCS, vol. 8804, pp. 201–218. Springer, Cham (2014). doi:10.1007/978-3-319-12060-7_14
12. Nassar, M., Souissi, Y., Guilley, S., Danger, J.-L.: RSM: a small and fast countermeasure for AES, secure against 1st and 2nd-order zero-offset SCAs. In: Design, Automation & Test in Europe Conference & Exhibition (DATE 2012), pp. 1173–1178. IEEE (2012)
13. Chari, S., Rao, J.R., Rohatgi, P.: Template attacks. In: Kaliski, B.S., Koç, K., Paar, C. (eds.) CHES 2002. LNCS, vol. 2523, pp. 13–28. Springer, Heidelberg (2003). doi:10.1007/3-540-36400-5_3
14. Schindler, W., Lemke, K., Paar, C.: A stochastic model for differential side channel cryptanalysis. In: Rao, J.R., Sunar, B. (eds.) CHES 2005. LNCS, vol. 3659, pp. 30–46. Springer, Heidelberg (2005). doi:10.1007/11545262_3
15. Attacks in the v4.1 phase. http://www.dpacontest.org/v4/rsm_hall_of_fame.php

16. Kutzner, S., Poschmann, A.: On the security of RSM - presenting 5 first- and second-order attacks. In: Prouff, E. (ed.) COSADE 2014. LNCS, vol. 8622, pp. 299–312. Springer, Cham (2014). doi:10.1007/978-3-319-10175-0_20
17. Ye, X., Eisenbarth, T.: On the vulnerability of low entropy masking schemes. In: Francillon, A., Rohatgi, P. (eds.) CARDIS 2013. LNCS, vol. 8419, pp. 44–60. Springer, Cham (2014). doi:10.1007/978-3-319-08302-5_4
18. Moradi, A., Guilley, S., Heuser, A.: Detecting hidden leakages. In: Boureanu, I., Owesarski, P., Vaudenay, S. (eds.) ACNS 2014. LNCS, vol. 8479, pp. 324–342. Springer, Cham (2014). doi:10.1007/978-3-319-07536-5_20
19. Other attacks submitted in the officail website. http://www.dpacontest.org/v4/42_hall_of_fame.php
20. Nassar, M., Guilley, S., Danger, J.-L.: Formal analysis of the entropy/security trade-off in first-order masking countermeasures against side-channel attacks. In: Bernstein, D.J., Chatterjee, S. (eds.) INDOCRYPT 2011. LNCS, vol. 7107, pp. 22–39. Springer, Heidelberg (2011). doi:10.1007/978-3-642-25578-6_4
21. Carlet, C., Guilley, S.: Side-channel indistinguishability. In: Proceedings of the 2nd International Workshop on Hardware and Architectural Support for Security and Privacy, p. 9. ACM (2013)
22. Tillich, S., Herbst, C.: Attacking state-of-the-art software countermeasures—a case study for AES. In: Oswald, E., Rohatgi, P. (eds.) CHES 2008. LNCS, vol. 5154, pp. 228–243. Springer, Heidelberg (2008). doi:10.1007/978-3-540-85053-3_15
23. Moradi, A., Kasper, M., Paar, C.: Black-box side-channel attacks highlight the importance of countermeasures. In: Dunkelman, O. (ed.) CT-RSA 2012. LNCS, vol. 7178, pp. 1–18. Springer, Heidelberg (2012). doi:10.1007/978-3-642-27954-6_1
24. Oswald, E., Mangard, S., Herbst, C., Tillich, S.: Practical second-order DPA attacks for masked smart card implementations of block ciphers. In: Pointcheval, D. (ed.) CT-RSA 2006. LNCS, vol. 3860, pp. 192–207. Springer, Heidelberg (2006). doi:10.1007/11605805_13
25. Standaert, F.-X., Veyrat-Charvillon, N., Oswald, E., Gierlichs, B., Medwed, M., Kasper, M., Mangard, S.: The world is not enough: another look on second-order DPA. In: Abe, M. (ed.) ASIACRYPT 2010. LNCS, vol. 6477, pp. 112–129. Springer, Heidelberg (2010). doi:10.1007/978-3-642-17373-8_7
26. Prouff, E., Rivain, M., Bevan, R.: Statistical analysis of second order differential power analysis. IEEE Trans. Comput. **58**(6), 799–811 (2009)
27. Standaert, F.-X., Malkin, T.G., Yung, M.: A unified framework for the analysis of side-channel key recovery attacks. In: Joux, A. (ed.) EUROCRYPT 2009. LNCS, vol. 5479, pp. 443–461. Springer, Heidelberg (2009). doi:10.1007/978-3-642-01001-9_26