Ming *et al. Cybersecurity* (2021) 4:14
https://doi.org/10.1186/s42400-021-00082-w

Cybersecurity

RESEARCH Open Access

A secure and highly efficient first-order masking scheme for AES linear operations

Jingdian Ming[1,2], Yongbin Zhou[1,2*], Huizhong Li[1,2] and Qian Zhang[1,2]

Abstract

Due to its provable security and remarkable device-independence, masking has been widely accepted as a noteworthy algorithmic-level countermeasure against side-channel attacks. However, relatively high cost of masking severely limits its applicability. Considering the high tackling complexity of non-linear operations, most masked AES implementations focus on the security and cost reduction of masked S-boxes. In this paper, we focus on linear operations, which seems to be underestimated, on the contrary. Specifically, we discover some security flaws and redundant processes in popular first-order masked AES linear operations, and pinpoint the underlying root causes. Then we propose a provably secure and highly efficient masking scheme for AES linear operations. In order to show its practical implications, we replace the linear operations of state-of-the-art first-order AES masking schemes with our proposal, while keeping their original non-linear operations unchanged. We implement four newly combined masking schemes on an Intel Core i7-4790 CPU, and the results show they are roughly 20% faster than those original ones. Then we select one masked implementation named RSMv2 due to its popularity, and investigate its security and efficiency on an AVR ATMega163 processor and four different FPGA devices. The results show that no exploitable first-order side-channel leakages are detected. Moreover, compared with original masked AES implementations, our combined approach is nearly 25% faster on the AVR processor, and at least 70% more efficient on four FPGA devices.

Keywords: Side-Channel Attacks (SCAs), Masking scheme, Advanced Encryption Standard (AES), Linear operations

Introduction

Side-Channel Attacks (SCAs) exploit physical leakages (e.g. running time (Kocher 1996), power consumption (Kocher et al. 1999) and electromagnetic radiations (Quisquater and Samyde 2001)) of a cryptographic implementation to recover the corresponding secrets. During the past two decades, SCAs have been proved to be serious threats to practical security of cryptographic devices, like CPUs, GPUs, smart cards, ASICs and FPGAs. The recent Meltdown and Spectre (Prout et al. 2018) attacks are typical examples of utilizing cache running time to steal secret data. As a consequence, countermeasures against SCAs must be developed and applied to protect the secret information. In fact, the necessity of protection against SCAs has been reached a broad consensus in the industry. There have been international standards that require cryptographic modules to protect against SCAs. For example, FIPS 140-3 (FIPS Publication 140-3 2019) and ISO/IEC 17825:2016 (JTC 2016), which both are security standards for cryptographic modules, claim that cryptographic modules with high level security should concern with the mitigation of SCAs. Obviously, designing secure (symmetrical) cryptography implementations against SCAs has become one of the most important hot spot in physical security.

Among different kinds of SCAs, first-order attacks, such as first-order differential power analysis (DPA) (Kocher et al. 1999) and first-order correlation power analysis (CPA) (Schneider and Krishnamoorthy 1996), are widely used due to their low costs. Thus, they are the first into considerations when guarding devices against SCAs.

*Correspondence: zhouyongbin@iie.ac.cn
[1]State Key Laboratory of Information Security,Institute of Information Engineering, Chinese Academy of Sciences, Beijing, China
[2]School of Cyber Security, University of Chinese Academy of Sciences, Beijing, China

Springer Open

© The Author(s). 2021 **Open Access** This article is licensed under a Creative Commons Attribution 4.0 International License, which permits use, sharing, adaptation, distribution and reproduction in any medium or format, as long as you give appropriate credit to the original author(s) and the source, provide a link to the Creative Commons licence, and indicate if changes were made. The images or other third party material in this article are included in the article's Creative Commons licence, unless indicated otherwise in a credit line to the material. If material is not included in the article's Creative Commons licence and your intended use is not permitted by statutory regulation or exceeds the permitted use, you will need to obtain permission directly from the copyright holder. To view a copy of this licence, visit http://creativecommons.org/licenses/by/4.0/.

ISO/IEC 17825:2016 specifies the use of Test Vector Leakage Assessment (TVLA) framework as a certain measure to assess whether an implementation of symmetrical cryptography is vulnerable to first-order attacks or not. According to the highest security level, namely ISO/IEC 17825 level-4, there should be at least 100,000 traces collected to test whether a specific implementation is resistant to them.

Due to its provable security and remarkable device-independence, masking (Rivain et al. 2009) has been one of the most wide-adopted countermeasures against SCAs (Duc et al. 2019). Specifically, masking countermeasure randomizes the subtle dependency between a sensitive intermediate values and its corresponding side-channel leakages by splitting the sensitive intermediate into several shares. Usually, it is called d^{th}-order masking when the sensitive intermediate is split into d+1 shares to thwart d^{th}-order SCAs (Ishai et al. 2003). Namely, a first-order masked implementation should be able to protect against first-order attacks. However, the main drawback of a masked implementation is that its cost is relatively high compared with the unprotected counterpart. In order to make masking countermeasure more practical, it is worthwhile to reduce its cost as much as possible.

Since relatively high complexity of non-linear operations, most studies focus on improving non-linear operations for masking schemes (Rivain and Prouff 2010). Specifically, for symmetric cryptographic algorithms such as AES, most studies focus on increasing the S-box efficiency by improving the look-up table (LUT) operations (Coron 2014; Coron et al. 2018) or mending polynomial computation over the finite field (Coron et al. 2014). However, the security and efficiency of linear masking schemes (For sake of simplicity, the masking scheme of linear operations is called linear masking scheme in the rest of this paper) are widely ignored. In this paper, we focus on the security and efficiency of masked AES linear operations, and we find linear part should not be ignored. As for security, we find that there are serious flaws in MixColumns operations in some masking schemes because correlative masks are adopted in each round. As for efficiency, to figure out which part of operations account more in whole cryptographic computation, we evaluate the running time proportions of non-linear and linear operations in state-of-the-art masking schemes. Due to the primacy of first-order security, we evaluate those in four different first-order masking schemes firstly, and these masking schemes are described in details as follows.

SP. SP (Schramm and Paar 2006) scheme is designed to be secure at any order d. This scheme is now regarded to be first-order and second-order secure, as an third-order attack against it was shown in (Coron et al. 2007). Additionally, they claimed that it is sufficient to use a single 8-bit mask for the entire masked AES algorithm.

ASCAD. ASCAD (Prouff et al. 2018) is a public dataset for the study of side-channel analysis, which is also implemented by a first-order masking AES scheme. So we use the same name to represent this masking scheme in this paper.

RSMv1. RSM (Nassar et al. 2012) is a low entropy masking scheme, which aims at keeping performances and complexity close to unprotected AES design while being as robust against first-order SCAs as other masking in cryptographic implementations. In this paper, we use RSMv1 to denote the first vision of RSM. Note that this masking scheme was adopted by DPA Contest v4.1 as a study case for side-channel security. Moreover, it has been proven in (Veshchikov and Guilley 2017) that the linear part of RSMv1 has a first-order flaw, and similar flaw also happens in SP scheme.

RSMv2. RSM was revisited in (Bhasin et al. 2014), and we use RSMv2 to denote this improved RSM scheme. Actually, RSMv2 was adopted by DPA Contest v4.2, and is one of the most popular first-order masking schemes.

All benchmarkings have been done on an Intel Core i7-4790 CPU. The results are shown in Fig. 1.

Moreover, we evaluate the running time proportions of non-linear and linear operations in three higher order masking schemes (proposed in (Rivain and Prouff 2010; Coron 2014; Coron et al. 2018), and denoted by **RP**, **Cor** and **CRZ** respectively in this paper) with d=1 and d=2 in same experimental setting. The results are shown in Fig. 2.

It can be seen that linear operations account for at least 70% in these four first-order masking schemes, while they account for less than 20% in second-order masking schemes. Note that complexity of non-linear part grows much faster than that of linear part with increasing d, so linear operations would account for lower in whole computations when masking order d gets higher than two. Considering that first-order security is primacy in masking countermeasure and linear operations are relatively important in first-order masking, in this paper we mainly

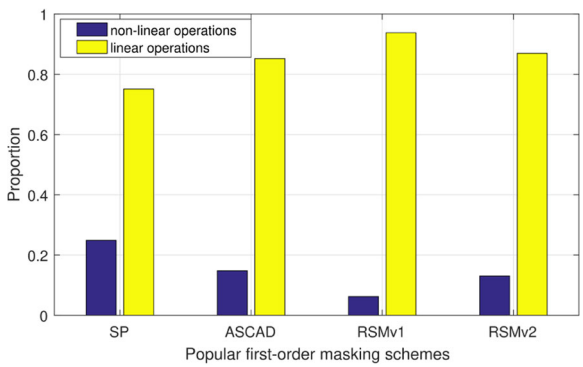


Fig. 1 Running time proportion of non-linear and linear operations in AES first-order masking schemes

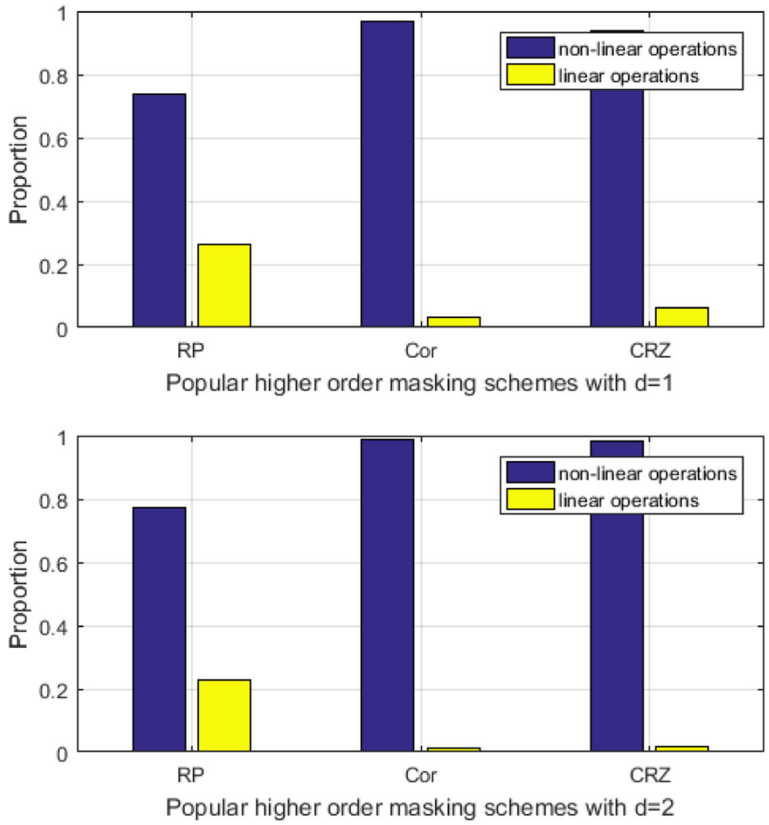


Fig. 2 Running time proportion of non-linear and linear operations in AES higher order masking schemes

focus on the optimization of linear operations to make first-order masking countermeasure secure and more efficient.

Our Contribution. In this paper, we find that AES linear operations are widely ignored in state-of-the-art first-order masking countermeasures. Specifically, these masking schemes are insecure when correlative masks are adopted in each round, and inefficient because some linear operations are called redundantly. Consequently, we propose a secure and highly efficient first-order linear masking scheme for AES, then heuristically prove its security and critically analyze its efficiency. In fact, this linear masking scheme can be combined with existing non-linear masking schemes, which not only maintains the original security level, but also further improves the implementation efficiency. As a concrete illustration, we apply it to four state-of-the-art AES masking schemes on an Intel Core i7-4790 CPU by replacing their linear operations with our proposal. The results demonstrate that by combining our proposal, these popular masking schemes are approximately 20% faster compared to the original ones. Furthermore, we combine a well-studied masking scheme named RSMv2 with our proposal, then evaluate its practical security, and carry out a detailed performance comparison on five different embedded devices. The results of security evaluation verify that there is no available first-order leakage in both software and hardware implementations. In addition, the results of efficiency evaluation show that the implementations combined our proposal are nearly 25% faster than existing implementations on AVR ATMega163 Processor, and the efficiency is improved by more than 70% on four different FPGA devices.

The rest of our paper is organized as follows. Sec. 2 reviews the basic knowledge of masking schemes for AES, and Sec. 3 illustrates the issues of the masking schemes in their linear operations. In Sec. 4, we propose our new first-order masking scheme for AES linear operations and analyze its security and efficiency in theory. Then in Sec. 5 we evaluate the performance of our proposal on different implementations. Finally, Sec. 6 concludes the paper.

Preliminary

In this section, we review the existing masking schemes for AES. In masking schemes, each sensitive value is split into $d+1$ shares $x_0, x_1 \cdots x_d$, and their relation can be expressed as Eq.(1).

$$x = x_0 \perp x_1 \cdots \perp x_d, \tag{1}$$

where x means the sensitive value while x_i means i-th split share, and $\perp$ denotes the mask operation. In the rest

of the paper, we shall consider that $\perp$ is the exclusive-or (XOR) operation denoted by $\oplus$. Usually, the d shares $\{x_1, \cdots, x_d\}$ called masks are randomly picked up and the $\{x_0\}$ called the masked value is processed such that it satisfied Eq. (1). Recently, most studies focus on designing the masked S-box with $d+1$ shares, such as d^{th}-order masking scheme based on LUT (Coron 2014), based on polynomial computation over the finite field (Coron et al. 2014) and so on. But the designing for linear operations is widely ignored. As illustrated above, the costs of linear operations account much more than that of non-linear operations in first-order masked implementations. Thus in this paper, we focus on first-order masking scheme, and accordingly each sensitive value is split into two shares (d=1). Recently, a common masking approach for linear operations is to compute each share successively (Schramm and Paar 2006; ParisTech 2015; Coron et al. 2014), as shown in Fig. 3. First of all, 16 S-boxes are processed respectively, and the input 8-bit value of each S-box is split into 2 shares, so each S-box is fed with 2*8 bits. Next, all 16*8 bits of each share are fed to two linear operations, because $F(x_0 \oplus x_1) = F(x_0) \oplus F(x_1)$ if F is a linear function. Thus, it is obvious that this masking scheme for linear operations will cost 2 times as much as unprotected AES implementation.

In this paper, to distinguish the two shares and sensitive value clearer, x_0, x_1 and x are expressed as ST, M and V respectively. Then we consider the masked AES implementation, while a general masked AES-128 implementation is shown in Algorithm 1, with the following linear operations:

AddRoundKey. AddRoundKey is a Boolean operation, which XOR the state data with round keys. As for Boolean masking schemes, the round keys need to do XOR with only one share (generally ST), so this operation costs the same in masked implementation with unprotected one.

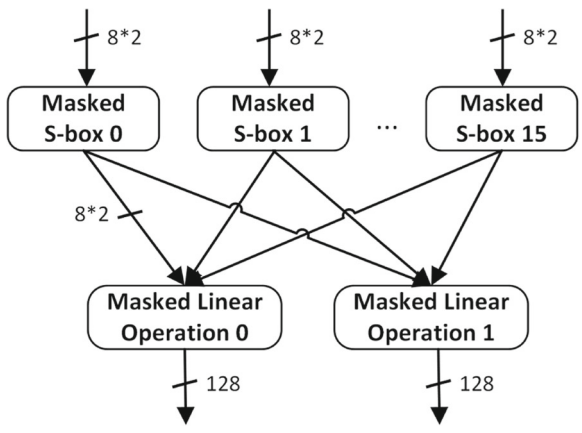


Fig. 3 The process of existing first-order masking scheme in one AES round

Algorithm 1 A general first-order Masked AES-128 implementation.

Input: 16 bytes *plain*[16] and *key*[16] for MixColumns
Output: 16 bytes *cipher*[16]

1: $W[176] \leftarrow$ KeyExpansion(key)
2: $M[16] \leftarrow$ MaskGen()
3: $ST \leftarrow M \oplus plain$
4: **for** i = 0 to 9 **do**
5: $ST \leftarrow$ AddRoundKey($ST, W(i*16 : i*16+15)$)
6: $[ST, M] \leftarrow$ Remask(ST, M) (optional)
7: $[ST, M] \leftarrow$ MSbox(ST, M)
8: $[ST, M] \leftarrow$ Remask(ST, M) (optional)
9: $ST \leftarrow$ ShiftRows(ST)
10: $M \leftarrow$ ShiftRows(M)
11: $ST \leftarrow$ MixColumns(ST)
12: $M \leftarrow$ MixColumns(M)
13: $[ST, M] \leftarrow$ Remask(ST, M) (optional)
14: **end for**
15: $[ST, M] \leftarrow$ Remask(ST, M) (optional)
16: $[ST, M] \leftarrow$ MSbox(ST, M)
17: $[ST, M] \leftarrow$ Remask(ST, M) (optional)
18: $ST \leftarrow$ ShiftRows(ST)
19: $M \leftarrow$ ShiftRows(M)
20: $ST \leftarrow$ AddRoundKey($ST, W(160 : 175)$)
21: $cipher \leftarrow ST \oplus M$

Remask. Remask is not a necessary operation for all first-order masking schemes. Specifically, remask operation refreshes the two shares ST and M without changing $ST \oplus M$ (it is different with mask refresh, since the new M may be not generated randomly). For example, the masking scheme in (Prouff et al. 2018) needs to remask before and after *MSbox*, which corresponds to the line 6, 8, 15 and 17 in Algorithm 1. And the masking schemes in (Nassar et al. 2012; Bhasin et al. 2014) need to remask after MixColumns, which corresponds to the line 13 in Algorithm 1.

ShiftRows. ShiftRows is necessary for AES. In the existing first-order masking schemes, ShiftRows operation executes once for each share in one round, as shown in line 9 and line 10 in Algorithm 1.

MixColumns. MixColumns is also necessary for masked AES schemes. MixColumns is usually expressed as several sequences of multiplications by 2 over $\mathbb{F}_{2^8}$ and XOR operations. Specifically, it can be expressed as Eq. (2).

$$\begin{cases} ST_{4i} \leftarrow 2ST_{4i} \oplus 3ST_{4i+1} \oplus ST_{4i+2} \oplus ST_{4i+3} \\ ST_{4i+1} \leftarrow ST_{4i} \oplus 2ST_{4i+1} \oplus 3ST_{4i+2} \oplus ST_{4i+3} \\ ST_{4i+2} \leftarrow ST_{4i} \oplus ST_{4i+1} \oplus 2ST_{4i+2} \oplus 3ST_{4i+3} \\ ST_{4i+3} \leftarrow 3ST_{4i} \oplus ST_{4i+1} \oplus ST_{4i+2} \oplus 2ST_{4i+3} \end{cases}, \quad (2)$$

where $i \in \{0, 1, 2, 3\}$. Actually, MixColumns can be implemented as Eq. (2) directly, which is an original and trivial implementation but is still adopted by (Coron 2014; Coron et al. 2018). Then it was improved in (Fang 2009), which is shown as Algorithm 2. The $GF256MUL2(\cdot)$ operation denotes the multiplication by 2 over $\mathbb{F}_{2^8}$. Actually, this improved MixColumns module has been adopted in (Nassar et al. 2012; Bhasin et al. 2014; Prouff et al. 2018). In original MixColumns, the $GF256MUL2$ operations for $ST[4i + j]$ and $ST[4i + (j + 1)_{mod\,4}]$ need to compute respectively, so each ST_i in Eq. (2) needs 2 $GF256MUL2$ and 4 XOR (3 XOR expressed in equation and one for $3\times ST[4i + (j + 1)_{mod\,4}]$). Since the linear operations are called twice in first-order masking scheme, there are totally 32×2 $GF256MUL2$ and 64×2 XOR in original MixColumns. Comparatively, it can be counted that there are 16 $GF256MUL2$ and 60 XOR in Algorithm 2, then totally 16×2 $GF256MUL2$ and 60×2 XOR for improved MixColumns in first-order masking scheme.

Algorithm 2 Improved computation of one MixColumns using GF256MUL function.

Input: 16 bytes intermediates ST[16] for MixColumns
Output: 16 bytes intermediates OT[16] for next step
1: **for** i = 0 to 3 **do**
2: $t \leftarrow ST[4i] \oplus ST[4i+1] \oplus ST[4i+2] \oplus ST[4i+3]$
3: **for** j = 0 to 3 **do**
4: $tmp \leftarrow ST[4i+j] \oplus ST[4i+(j+1)_{mod\,4}]$
5: $OT[4i+j] \leftarrow GF256MUL2(tmp) \oplus ST[4i+j] \oplus t$
6: **end for**
7: **end for**

However, there are security flaws and redundant calls in the linear operations, which will be explained in next section.

Security and efficiency of linear operations

In this section, we illustrate the flaws of existing masked AES implementations in linear operations. Then we show a protected algorithm as an example, which satisfied first-order secure but not efficient. This example could help to understand our proposal described in next section.

In first-order masking schemes, each sensitive intermediate should be randomized by at least one randomness. Certainly, the output of a linear operation (e.g. XOR) should also be randomized, so we have Eq. (3).

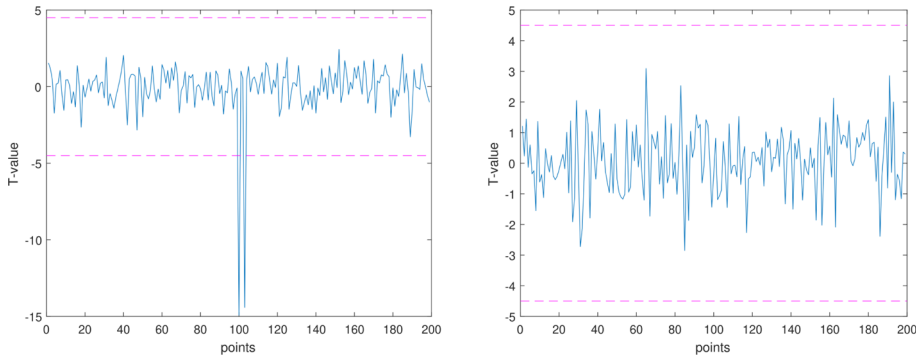
$$\begin{aligned} ST_0 \oplus ST_1 &= (V_0 \oplus M_0) \oplus (V_1 \oplus M_1) \\ &= (V_0 \oplus V_1) \oplus (M_0 \oplus M_1). \end{aligned} \tag{3}$$

Two sensitive intermediates V_0 and V_1 are randomized by masks M_0 and M_1 separately, while masked values are denoted by ST_0 and ST_1. Then, the output $(V_0 \oplus V_1)$ is randomized by $(M_0 \oplus M_1)$. If these masks are correlative (e.g. in SP scheme, $M_0 = M_1$), then $(V_0 \oplus V_1)$ are not randomized enough and this masking scheme cannot reach first-order security. Similarly, in a masked AES scheme, these linear operations cannot reach first-order security while the masks in each round are correlative. Specifically, in MixColumns module, the different masked values (ST_i) get XOR without any new random number introduced. Thus if these intermediates are protected by correlative masks, the new intermediate (their XOR results) may no longer be protected.

It can be verified by simulated experiments. We simulated the leakages of one loop in Algorithm 2 using a very popular leakage model (Ming et al. 2020), which can be expressed by $L = HW(x) + N$, where L denotes the leakages and $HW(x)$ denotes Hamming weight of the intermediate x. N denotes Gaussian noise and we use σ to denote its standard deviation. σ is set to 2 in all simulated experiments. We simulated 200 points totally, and 19 points among them are corresponding to $3+4\times 4=19$ operations in one loop in Algorithm 2 while other points are simulated by Gaussian noise. We launch t-test (Gilbert Goodwill et al. 2011) to detect the first-order leakages on these simulated points, and the results are shown in Fig. 4a. It can be seen that first-order leakages are really obvious. Additionally, this flaw is also found in RSMv1. In this scheme, third bit of masks after XOR operation are totally the same, so the third bit of output is not randomized. Utilizing this unprotected bit, first-order leakages can also be detected, which are shown in Fig. 5b. Similar results were illustrated in (Veshchikov and Guilley 2017), which utilized this unprotected bit to launch first-order attack on DPA contest v4 dataset, and successfully recovered the secret key. However, they thought this attack worked due to improper mask set, but the implementation with their revisited mask sets is still first-order attackable (Ming et al. 2020).

We show one protected example to make our proposal easier to understand. Specially, we generate a new randomness s and introduce it to line 2 and line 4 in Algorithm 2 to protected sensitive intermediates, and the protected example is shown in Algorithm 3. It is easy to verify that the inputs and outputs of Algorithm 3 are the same as those of Algorithm 2, since randomness s is cancelled out before output.

It is also trivial to verify the security of Algorithm 3. We denote by I the set of intermediate variables that processed during an execution of Algorithm 3, and we denote by EI the set of extra variables due to accumulation in an intermediate variable. Table 1 lists these variables of Algorithm 3. In order to prove Algorithm 3 is secure against first-order SCA, we need to show that all variables are protected at least by one randomness. Specifically, I_1 and I_2 are straightforwardly secure. I_3, I_4 and I_5 are protected by s



(a) Alg. 2 in SP scheme (b) Alg. 3 in SP scheme

Fig. 4 T-test on simulated leakages of Algorithm 2 and Algorithm 3 in SP scheme

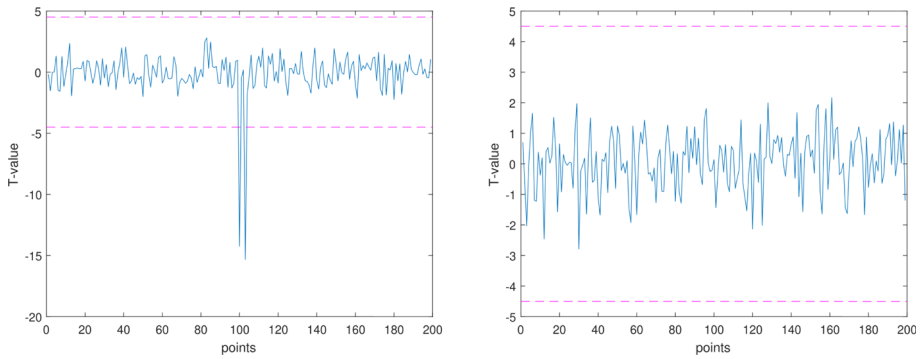
Algorithm 3 One example of a protected MixColumns implementation.

Input: 16 bytes intermediates ST[16] for MixColumns
Output: 16 bytes intermediates OT[16] for next step

1: $s \xleftarrow{\$} \mathbb{F}_2^8$
2: **for** i = 0 to 3 **do**
3: $t \leftarrow ST[4i] \oplus GF256MUL2(s) \oplus ST[4i+1] \oplus ST[4i+2] \oplus ST[4i+3]$
4: **for** j = 0 to 3 **do**
5: $tmp \leftarrow ST[4i+j] \oplus s \oplus ST[4i+(j+1)_{mod\,4}]$
6: $OT[4i+j] \leftarrow GF256MUL2(tmp) \oplus ST[4i+j] \oplus t$
7: **end for**
8: **end for**

or $GF256MUL2(s)$. Note that if t (*resp. tmp*) is not cleared to zero before overwriting with $ST[4i]$ (*resp.* $ST[4i+j]$), there will be two sets of extra variables, which are denoted by EI_1 and EI_2. The extra variables in EI_1 and EI_2 are related to the leakages function. For example, the extra variable in EI_1 can be denoted by $t \oplus ST[4(i+1)]$ in Hamming distance leakage function. Since t (*resp. tmp*) is still protected by s (*resp.* $GF256MUL2(s)$) which is independent with ST, the two sets EI_1 and EI_2 are secure as well. Note that the output OT might be attackable since s is removed. However, if the 16 masks $M[16]$ are the same, OT will be still protected well. Because the masks M are also removed in line 3 and line5, so $GF256MUL2(tmp)$ and t are only protected by s, which is removed in line 6. Then OT will be under the same protection of the masks of ST, namely M.

In addition, we simulate the points for each operation in protected example with 16 same masks and detect their first-order leakages. There are 200 simulated points while 24 points among them are corresponding to 24 operations in one loop in Algorithm 3. The detection results for protected SP and protected RSMv1 are shown in Figs. 4b and 5b respectively. It can be seen that no first-order leakages detected after protection. And our proposal described in next section is an improvement from this example.



(a) Alg. 2 in RSMv1 scheme (b) Alg. 3 in RSMv1 scheme

Fig. 5 T-test on simulated leakages of Algorithm 2 and Algorithm 3 in RSMv1 scheme

Table 1 Intermediate and extra variables of Algorithm 3

l	l_l	Steps
1	$s, GF256MUL2(s)$	1,3,5
2	ST	3,5,6
3	$t, tmp, GF256MUL2(tmp)$	3,5,6
4	$ST[i] \oplus s, \bigoplus_i ST[i] \oplus GF256MUL2(s)$	3,5,6
5	$GF256MUL2(tmp) \oplus ST[i]$	6
l	El_l	Steps
1	$< t, ST[4(i+1)] >$	3
2	$< tmp, ST[4i+(j+1)] >$	5

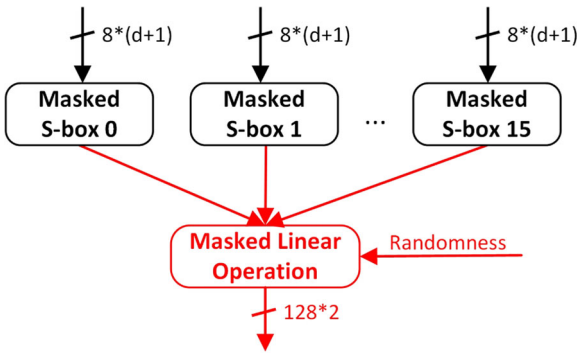


Fig. 6 The process of our proposed first-order masking scheme in one AES round

On the other hand, these linear operations usually become inefficient because some linear operations are called repeatedly and redundantly. Only considering the necessary operations, the ShiftRows and MixColumns are called for each shares, as shown in line 9, line 10 and line 11, line 12 in Algorithm 1. Clearly, calling linear operations repeatedly is surely satisfied with correctness for a linear function, since $F(ST \oplus M)$ equals to $F(ST) \oplus F(M)$ for a linear function $F(\cdot)$. However, the masks M are randomly picked up, and linear operations for M seem wasteful. Thus, it is important to design a secure linear algorithm without changing the masks M while ST is updated, then the linear operations for masks M can be saved.

Linear masking scheme

In this section, we propose a secure and highly efficient first-order linear masking scheme for AES, then analyze its efficiency and prove its security.

Core idea

In fact, designing two linear operations is a waste of resources for masked AES implementation. Therefore, we introduce the idea that: reduce two masked linear operations in Fig. 3 to one while extra randomness is induced to maintain the security level, as shown in Fig. 6. Consequently, the efficiency on linear operation can be considerably increased by a factor of two.

Following this core idea, we propose our first-order linear masking scheme.

First-order linear masking scheme

As shown above, there are 3 necessary modules of the linear operations in AES: AddRoundKey, ShiftRows and MixColumns. Among them, AddRoundKey and ShiftRows have been secure and efficient. As for AddRoundKey, only one share needs to do XOR with corresponding round key, which cost the same with unprotected AES. As for ShiftRows, we can just reorder the computation of 16 S-boxes (Bhasin et al. 2014) to save half costs. However, designing the masked MixColumns module is the most complex and significant for improving the linear operation since it costs the highest in these modules.

The masked MixColumns is shown as Algorithm 4, which is called SEMixColumns in this paper to distinguish it from others. In order to combine two MixColumns into one while maintaining the same security, new generated random numbers (line 1) are needed to protect the intermediates, then part of masks can be removed. Finally, these random numbers s are removed in line 10 while this line is protected by the masks of ST_{4i+j}, and the outputs of Algorithm 4 (16 bytes $OT[16]$) are still protected by M, so the MixColumns for M in Algorithm 1 (line 12) can be saved.

Algorithm 4 Secure and efficient proposal of one MixColumns (called SEMixColumns in this paper).

Input:
16 bytes intermediates ST[16] for MixColumns
16 bytes masks M[16] for MixColumns
Output: 16 bytes intermediates OT[16] for next step

1: $s \xleftarrow{\$} \mathbb{F}_2^8$
2: **for** i = 0 to 3 **do**
3: $r_0 \xleftarrow{\$} M[4i] \oplus M[4i+1]$
4: $r_1 \xleftarrow{\$} M[4i+1] \oplus M[4i+2]$
5: $r_2 \xleftarrow{\$} M[4i+2] \oplus M[4i+3]$
6: $r_3 \xleftarrow{\$} M[4i+3] \oplus M[4i]$
7: $t \leftarrow ST[4i] \oplus GF256MUL2(s) \oplus ST[4i+1] \oplus ST[4i+2] \oplus ST[4i+3] \oplus r_1 \oplus r_3$
8: **for** j = 0 to 3 **do**
9: $tmp \leftarrow ST[4i+j] \oplus s \oplus ST[4i+(j+1)_{mod\,4}] \oplus r_j$
10: $OT[4i+j] \leftarrow GF256MUL2(tmp) \oplus ST[4i+j] \oplus t$
11: **end for**
12: **end for**

Table 2 Comparison of the costs on linear operations of one round

Cost	Number of XOR			Number of *GF*256*MUL*2	Number of Shift
Operation	AddRoundKey	MixColumns	Remask	MixColumns	ShiftRows
Original Implementation	16	64× 2	0/32/64/96	32× 2	12× 2
Improved Implementation	16	60× 2	0/32/64/96	16× 2	12× 2
Our Proposal	16	120	0/32/64/96	20	12× 2

Thus, line 11 and line 12 in Algorithm 1 are replaced by "$ST \leftarrow$ SEMixColumns(ST, M)" in our masked AES algorithm, while other parts are all the same. Note that the ST should be cleared to zero before overwriting with OT, otherwise there will be extra leakages corresponding to the sensitive intermediate $GUL2MUL(tmp) \oplus t$. One MixColumns module is saved by the optimization, although SEMixColumns costs more since new variables are introduced. It can be counted that there are totally 20 $GF256MUL2$ and 120 XOR in Algorithm 4. Since Algorithm 4 is only called once in masked AES implementation, it costs less than other algorithm in sum. In order to make clearer comparison of the costs on linear operations, we list the required number of basic operations (XOR, Shift and $GF256MUL2$) of these linear implementations in one AES round, as shown in Table 2.

It can be seen in Table 2 that our proposal is advantaged particularly on the number of $GF256MUL2$. Our proposal only cost 20 $GF256MUL2$ in each AES round. So the implementation for $GF256MUL2$ certainly affects the efficiency gain though our proposal. Moreover, Table 2 is only a theoretical comparison, the practical improvements may be different more or less due to different implementation targets.

Security analysis

In this section, we prove that our proposed masking scheme can actually reach first-order security against SCAs.

First of all, other linear parts except MixColumns have been proven to reach first-order security. As for MixColumns, both original implementation (Coron et al. 2018) and efficient implementation (Veshchikov and Guilley 2017) will be threaten by first-order attacks if the masks in each round are correlative, which has been demonstrate in Sec. 3. So we need to prove that our proposed SEMixColumns (Algorithm 4) is able to protect against first-order attacks, namely each sensitive value is protected by a random number.

We heuristically prove the first-order security of Algorithm 4 with the approach similar to that in Sec. 3. We also denote by I the set of intermediate variables that processed during an execution of Algorithm 4, and we denote by EI the set of extra variables due to accumulation in an intermediate variable. Table 3 lists these variables of Algorithm 4. Specifically, I_1, I_3, I_4, I_5 and I_7 are the same as the sets in Table 3, which have been proved to be secure. And I_2 and I_6 are also protected by s or $GF256MUL2(s)$. Since random numbers s are removed in line 10 while ST is protected by the masks M, the outputs I_8 are still protected by M. In addition, if t (*resp. tmp*) is not cleared to zero before overwriting with $ST[4i]$ (*resp.* $ST[4i+j]$), there will also be two set of extra variables, which are denoted by EI_1 and EI_2 in Table 3. Similarly, EI_1 and EI_2 are protected by s or $GF256MUL2(s)$ as well.

In summary, all sensitive intermediates in linear operations are still under protection, which proves that our proposal for linear operations can theoretically achieve first-order security.

More efficient implementation in certain cases

Our proposal as shown in Algorithm 4 can be optimized further in certain cases. Specifically, if Eq. (4) is satisfied, the masks in each column after ShiftRows are equal (the same color in Fig. 7), then r_0, r_1, r_2 and r_3 in Algorithm 4 equal to 0. Therefore, there is no need for the XOR operations related with these intermediates, which can save an additional 40 XOR per round.

$$\begin{cases} m_i = m_{i+1 \ (mod\,16)}, & if \ \ i \ (mod\,4) = 1 \\ m_i = m_{i+5 \ (mod\,16)}, & others \end{cases} \tag{4}$$

However, the optimized method cannot be adopted by all first-order masking schemes since: (1) this method

Table 3 Intermediate and extra variables of Algorithm 4

l	I_l	Steps
1	$s, GF256MUL2(s)$	1,7,9
2	r_0, r_1, r_2, r_3	3,4,5,6,7,9
3	ST, OT	7,9,10
4	$t, tmp, GF256MUL2(tmp)$	7,9,10
5	$ST[i] \oplus s, \bigoplus_i ST[i] \oplus GF256MUL2(s)$	7,9
6	$t \oplus r_3, tmp \oplus r_j$	7,9
7	$GF256MUL2(tmp) \oplus ST[i]$	10
8	OT	10
l	EI_l	Steps
1	$< t, ST[4(i+1)] >$	7
2	$< tmp, ST[4i+(j+1)] >$	9

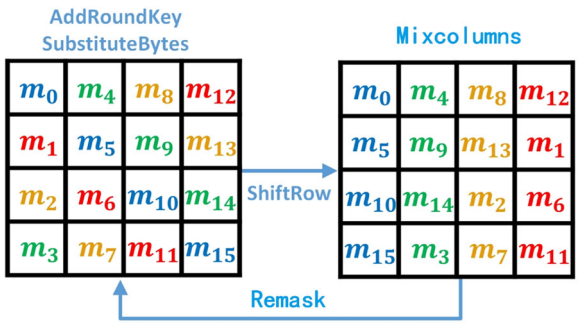


Fig. 7 The masks in our first-order masking scheme

requires the masks to satisfied Eq. (4) (m_i denotes the i-th mask of the 16 bytes masks M[16]), so it may be incompatible with some first-order masking schemes with similar type restrictions (e.g. the masking scheme RSMv1 requires the masks to be a regular sequence, which is not compatible with this more efficient implementation). (2) after the Mixcolumns, it is necessary to remask to ensure continuous implementation process (as shown in Fig. 7). So the masking scheme should contain the remask operation after Mixcolumns (line 13 in Algorithm 1), or this optimized method would not work for the scheme duo to extra XOR introduced.

Experiment

In this section, we combine our linear masking algorithm with different first-order masking schemes, and evaluate their practical efficiency in a CPU implementation. Next, we choose the scheme named RSMv2, which have been studied well because of DPA Contest v4.2, as our main target. Specifically, RSMv2 is implemented in both software and hardware, then its side-channel security and practical efficiency are evaluated later.

Evaluation of running time on CPU

A masked AES implementation is consist of two parts: non-linear part and linear part, so the speedup of our proposal toward whole AES execution time is also affected by the implementations of non-linear part. Generally speaking, if the non-linear part is well-designed and efficient, the improvement of our proposal will get more remarkable. In this experiment, we combine our proposal with four state-of-the-art first-order masking schemes, which have been introduced in details.

These schemes are implemented on an Intel Core i7-4790 CPU running at 3.60GHz. Considering that the function $GF256MUL2$ also affects speedup of our proposal, we implement $GF256MUL2$ by two methods: (1) Computation (slower but save memories). Rotate the 8-bits input to the left by 1 bit, then judge whether the most significant bit equals to 1. If so, then XOR this intermediate with $0x1b$. (2) LUT (faster but need extra memories). Look up the outputs in a stored table. The results of speedup in different first-order masking schemes are shown in Table 4. It can be seen that our proposal can get approximately 20% speedup on these masking schemes. Moreover, for each masking scheme, the speedup of our proposal is greater while $GF256MUL2$ is achieved by computation, because computation based $GF256MUL2$ requires more running time than LUT based one, which lead to more cost savings by using our proposal.

Moreover, we discuss about how the compiler treats these software implementations, and the details are shown in Appendix.

Evaluation of embedded implementations

Considering that RSMv2 has been studied well because of DPA Contest v4.2 and its software implementation has been published in (ParisTech 2015), we select this popular first-order masking scheme as our study object. And in our experiments, we replaced the linear operations by our more efficient proposal while keeping the non-linear operations unchanged.

As for software implementation, we adopt the similar RSMv2 implementation as used in DPA Contest v4.2. We programmed the hex-files on a FunCard with an Atmel ATmega 163 micro-processor, which is the suggested platform from DPA Contest v4.2. The implementation codes of RSMv2 scheme can be also downloaded from the DPA Contest v4.2 (ParisTech 2015), which uses C Language on connection between the processor and computer, and uses assembly language on cryptographic implementation. To take a fair comparison, we only use assembly language to change the linear operations. Note that $GF256MUL2$ is implemented by LUT in the published codes, and we implement this function in the same way.

Compared to the RSMv2 implementation, the code size of our proposal gets similar. Actually both of these two hex-files are 42KB. As for execution time, our proposal gets obviously faster. The FunCard is connected with computer by SASEBO-W board, and we measured the running time of RSMv2 and our proposal with a trigger in codes and an oscilloscope. Since ATmega 163 micro-processor is running at 8MHz, it is easy to get the results on the software implementation in cycles, which are show as Table 5. The results show that using our proposal the masked AES is nearly 25% faster than before on the software implementation.

As for hardware implementations, the RSMv2 scheme in (Bhasin et al. 2014) and our proposal are both implemented in Kintex-7, Virtex-6, Virtex-5 and Spartan-3E FPGA devices, the consumed resources for masked AES are shown in Table 6. Since these AES implementations take 128-bit inputs and outputs, the data size is 128 bit. And the latency of these implementations is 14 cycles.

Table 4 Speedup in popular first-order masking schemes on an Intel Core i7-4790 CPU running at 3.60GHz

Scheme	Linear Operatons	First-Order Security	$GF256MUL2$[1]	Time [us]	Speedup[2]
Unprotected AES	Original Implementation	No	Computation	3.178	–
			LUT	2.160	
	Improved Implementation	No	Computation	2.596	
			LUT	2.067	
SP	Original Implementation	No	Computation	3.606	–
			LUT	2.525	
	Improved Implementation	No	Computation	2.917	
			LUT	2.454	
	Our Proposal	Yes	Computation	3.12	
			LUT	2.460	
ASCAD	Original Implementation	Yes	Computation	6.377	–
			LUT	4.307	
	Improved Implementation	Yes	Computation	5.143	
			LUT	4.169	
	Our Proposal	Yes	Computation	4.168	23.39%
			LUT	3.539	17.80%
RSMv1	Original Implementation	No	Computation	6.315	–
			LUT	4.389	
	Improved Implementation	No	Computation	5.407	
			LUT	4.161	
	Our Proposal	Yes	Computation	4.254	
			LUT	3.613	
RSMv2	Original Implementation	Yes	Computation	6.651	–
			LUT	4.610	
	Improved Implementation	Yes	Computation	5.496	
			LUT	4.388	
	Our Proposal	Yes	Computation	4.483	22.6%
			LUT	3.796	15.6%
	More Efficient Proposal	Yes	Computation	4.381	25.45%
			LUT	3.668	19.63%

The function *GF256MUL2* can be implemented by different methods, which affects the speedup of our proposal. 'Computation' in this line means that *GF256MUL2* is implemented by shift and XOR operations, 'LUT' means it is implemented by look-up table.
While getting the speedup for each masking scheme, we compare our proposal with first-order secure and the most efficient implementation

The delay column represent the longest path between registers. Note that non-linear part is implemented using fixed BRAM costs, which is a common and popular implementation (Nassar et al. 2012). As for GF256MUL2, it is implemented by computation on FPGA. Above all, it can be seen that our proposal greatly outperforms the RSMv2 scheme in efficiency on these four different FPGA devices, and the efficiency are all increased by more than 70%.

- On Kintex-7 device, which is built on 28-nm process technology, our proposal achieves better performance with 70.78% efficiency gain compared to existing implementation.
- Virtex-6 is also built on 28-nm process technology, and the performance on it gets worse (more delays and LUTs) than that on Virtex-6 device. But our proposal still achieves better performance with

Table 5 Speedup on an ATmeaga 163 micro-processor running at 8MHz

Number of cycles		Speedup
RSMv2	Our Proposal	
27,001	21,287	26.84%

Table 6 Comparison on different hardware implementations for RSMv2 and our proposal

Design Library	Scheme	Delay	Throughput		Area		Combined Gain
		[nsec]	[Gbps]	Gain	[LUTs]	Gain	
Kintex-7 28nm XC7K70T	RSMv2	2.916	3.135	31.07%	1599	30.32%	70.78%
	Our Proposal	2.225	4.109		1227		
Virtex-6 28nm XC6VCX75T	RSMv2	4.000	2.286	27.73%	1646	34.15%	71.35%
	Our Proposal	3.131	2.920		1227		
Virtex-5 65nm XC5VLX20T	RSMv2	4.240	2.156	32.65%	1700	36.11%	80.60%
	Our Proposal	3.197	2.860		1249		
Spartan-3E 90nm XC3S1600E	RSMv2	9.003	1.016	28.18%	2501	38.10%	74.38%
	Our Proposal	7.134	1.282		1811		

71.35% efficiency gain, which is similar to the result on Kintex-7 device.

- Virtex-5 is for integration at 56-nm, but the performance on it is similar with that on Virtex-6, and our proposal achieves better performance with 80.60% efficiency gain, which is the most in these four devices.
- Spartan-3E is built on 90-nm process technology. On Spartan-3E device, our proposal can also achieve better performance with 74.38% efficiency gain.

An interesting finding is that the delays and LUTs on Spartan-3E are much more than other three devices. We think it is because Kintex-7, Virtex-6 and Virtex-5 are 6-input LUTs based FPGA, while Spartan-3E devices is 4-input LUTs based one. Therefore, the implementation on Spartan-3E requires more LUTs (almost 50% than others), and accordingly needs more running time.

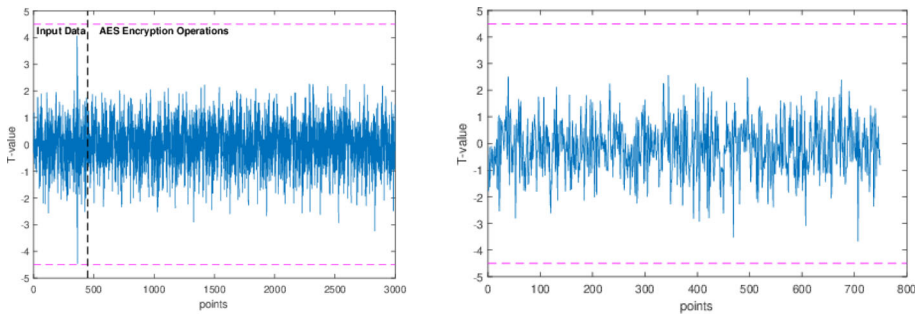
From the results of these experiments, our proposal for linear operations of AES masking schemes greatly improves the efficiency of masked AES on both hardware and software cryptographic implementations.

Side-channel security

To evaluate the practical security of our proposal, we launch t-tests (Gilbert Goodwill et al. 2011) and first-order CPA (Rivain 2008) to both software and hardware implementations respectively. The traces for software implementation are collected from an Atmel ATmega 163 micro-processor on SASEBO-W board with an Agilent DSO9104A oscilloscope. The sampling rate is set to be 20MS/s, and the collected traces are compressed with a step of 4 points. The traces for hardware implementation are collected from a Xilinx Kintex-7 family device XC7K160T-1FBG676 embeded on SAKURA-X board with also an Agilent DSO9104A oscilloscope. The sampling rate is set to be 1GS/s.

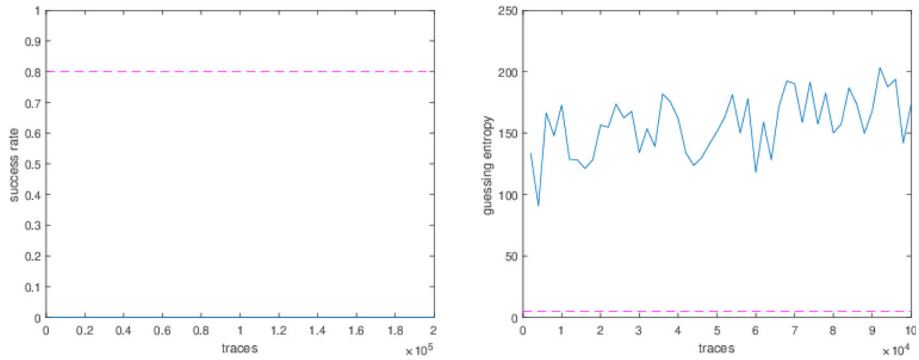
We use leakage detection test to evaluate whether any first-order side-channel leakages existing in our proposal. Specifically, We apply first-order t-test on both software implementation and hardware implementation. For software implementation, we totally collect 200,000 traces, each trace has 12,000 points and is compressed to 3,000 points. Considering that FPGA implementation is faster and gets more noise, we totally collect 560,000 traces, each trace has 750 points. The results of first-order t-tests are shown in Fig. 8. It is obvious that the first-order leakages of two implementations are always lower than ± 4.5, which means our proposal has no exploitable first-order leakages on these two implementations.

In Fig. 8a, the leakages near No.400 point get more obvious than other points. Actually, it is the operations for data



(a) Software implementation (b) Hardware implementation

Fig. 8 T-test on software and hardware RSMv2 implementation after combining our proposal



(a) Success rate (b) Guessing entropy

Fig. 9 First-order attacks on software RSMv2 implementation after combining our proposal

input (reading plaintexts), so these points cannot be used to recover the secret key, and the following attack results also demonstrate it. We used 200,000 traces to launch a first-order attack on the software implementation, and used 560,000 traces to launch first-order attack on hardware implementations. Figures 9 and 10 show the success rate and guessing entropy results (Rivain 2008) for these two implementations. We can see that the success rate always equals to zero in both implementations, and there is no downward trend in the guess entropy results.

Above all, all experimental results show that our proposal can actually provide first-order security to the software and hardware implementations.

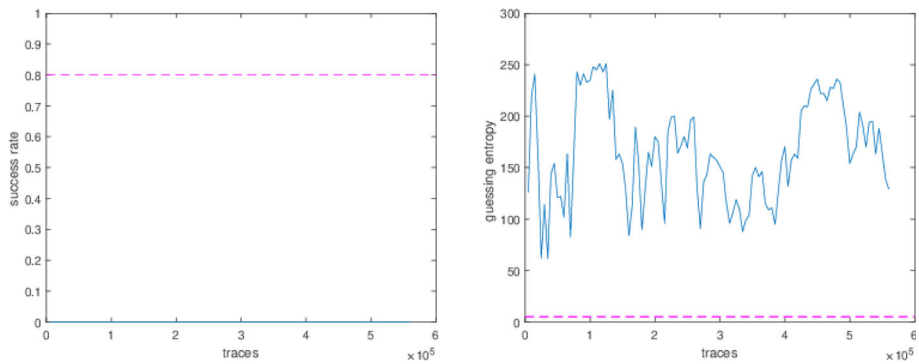
Application in higher order masking schemes

Our proposal performs well in first-order masking schemes in different implementations. However, we do not think it can achieve similar performance in higher order masking implementations.

We have shown the running time proportion of linear operations in different AES higher order masking implementations in Fig. 2, and we find that the significance of the linear part progressively decreases with the increase of the masking order. Take **CRZ** (Coron et al. 2018) higher order masking scheme as an example, which is state-of-the-art provably secure masked implementation for AES at any order d. When the order of **CRZ** masking scheme exceeds two, linear operations account for less than 3% of the total AES implementation. In fact, the evaluations on other higher order masking schemes get similar results. Therefore, we think that it is of little meaning to improve the efficiency of AES higher order masking schemes by optimizing their linear operations. Namely, it should be still focused on non-linear operations to improve the higher order masking schemes.

Conclusion

In this paper, we point out that there are some design defects in the linear operation of state-of-the-art masked AES implementations. These defects not only decrease the security level of some popular first-order masking schemes, but also reduce their implementation efficiency. In order to remedy these defects, we propose a first-order linear masking scheme with inspired performance, then



(a) Success rate (b) Guessing entropy

Fig. 10 First-order attacks on hardware RSMv2 implementation after combining our proposal

prove its security and analyze its efficiency. Specifically, this masking scheme is designed for AES linear operations, so it can be combined with any existing masked AES scheme to further improve the implementation efficiency. To demonstrate its practical security and efficiency, we combine our proposal with four different state-of-the-art first-order masking schemes on an Intel Core i7-4790 CPU. The results show that the four masking schemes can get roughly 20% faster than before by using our proposal. Moreover, we select a well-studied masking scheme named RSMv2, and implement it on five different embedded devices. The results show that by using our proposal, the masking implementations achieve much higher efficiency without exploitable first-order leakages.

Appendix

To figure out how the compiler treats the software implementation, we optimize the implementations with the help of "-O3" option. Then we measure the performance of all popular first-order masking schemes on an Intel Core i7-4790 CPU again. The results are shown in Table 7. It can be seen that the speedup is not impressive under this environment.

We additionally take ARM as embedded software target. Specifically, we evaluate the performance of unprotected AES and four masking schemes on a STM32F4 MCU based on ARM Cortex-M4. Thanks to the simulator in Keil uVision5, the performance results of all the schemes are straightforwardly in cycles, as shown in Table 8. It can

Table 7 Speedup in popular first-order masking schemes on an Intel Core i7-4790 CPU with "-O3" compiler option running at 3.60GHz

Scheme	Linear Operations	First-Order Security	$GF256MUL2$[1]	Time [us]	Speedup[2]
Unprotected AES	Original Implementation	No	Computation	3.178	–
			LUT	0.362	
	Improved Implementation	No	Computation	0.400	
			LUT	0.362	
SP	Original Implementation	No	Computation	0.587	–
			LUT	0.523	
	Improved Implementation	No	Computation	0.562	
			LUT	0.533	
	Our Proposal	Yes	Computation	0.528	
			LUT	0.599	
ASCAD	Original Implementation	Yes	Computation	0.847	–
			LUT	0.734	
	Improved Implementation	Yes	Computation	0.793	
			LUT	0.732	
	Our Proposal	Yes	Computation	0.775	2.32%
			LUT	0.693	5.63%
RSMv1	Original Implementation	No	Computation	0.775	–
			LUT	0.666	
	Improved Implementation	No	Computation	0.734	
			LUT	0.660	
	Our Proposal	Yes	Computation	0.665	
			LUT	0.593	
RSMv2	Original Implementation	Yes	Computation	0.969	–
			LUT	0.840	
	Improved Implementation	Yes	Computation	0.909	
			LUT	0.845	
	Our Proposal	Yes	Computation	0.893	1.79%
			LUT	0.819	3.17%
	More Efficient Proposal	Yes	Computation	0.809	12.36%
			LUT	0.795	6.29%

Table 8 Speedup in popular first-order masking schemes on an Cortex-M4 using simulator

Scheme	Linear Operations	First-Order Security	$GF256MUL2$[1]	Time [us]	Speedup[2]
Unprotected AES	Original Implementation	No	Computation	3.178	–
			LUT	14,683	
	Improved Implementation	No	Computation	14,116	
			LUT	0.362	
SP	Original Implementation	No	Computation	25,177	–
			LUT	18,661	
	Improved Implementation	No	Computation	21,001	
			LUT	17,959	
	Our Proposal	Yes	Computation	22,909	
			LUT	18,922	
ASCAD	Original Implementation	Yes	Computation	41,894	–
			LUT	28,862	
	Improved Implementation	Yes	Computation	33,542	
			LUT	27,458	
	Our Proposal	Yes	Computation	28,214	18.88%
			LUT	24,056	14.14%
RSMv1	Original Implementation	No	Computation	45,048	–
			LUT	32,016	
	Improved Implementation	No	Computation	36,696	
			LUT	30,612	
	Our Proposal	Yes	Computation	31,368	
			LUT	27,210	
RSMv2	Original Implementation	Yes	Computation	46,995	–
			LUT	33,963	
	Improved Implementation	Yes	Computation	38,643	
			LUT	32,559	
	Our Proposal	Yes	Computation	33,315	15.99%
			LUT	29,157	11.67%
	More Efficient Proposal	Yes	Computation	33,279	16.12%
			LUT	29,148	11.70%

be seen that our proposal is nearly 15% faster on the typical 32-bit processor, which is not as much as the benefis on 64-bit Intel processor. Namely, the performance gains are different due to implemented compiler.

Acknowledgments
The authors would like to thank the Xinkuan Qiu for his valuable comments.

Authors' contributions
JM and YZ proposed the first-order AES masking scheme, and drafted the manuscript. HL participated in problem discussions and improvements of the manuscript. QZ implemented and benchmarked the proposed scheme in FPGA devices. All authors read and approved the final manuscript.

Authors' information
All authors (Email: mingjingdian, zhouyongbin, lihuizhong, zhangqian@iie.ac.cn) are with State Key Laboratory of Information Security, Institute of Information Engineering, Chinese Academy of Sciences, Beijing 100093, China, and School of Cyber Security, University of Chinese Academy of Sciences, Beijing 100049, China. Yongbin Zhou is the corresponding author of this paper.

Funding
This work is supported in part by National Natural Science Foundation of China (No.61632020, No.U1936209 and No.62002353) and Beijing Natural Science Foundation (No.4192067).

Availability of data and materials
Not applicable.

Competing interests
The authors declare that they have no competing interests.

Received: 2 October 2020 Accepted: 10 February 2021
Published online: 02 June 2021

References

Bhasin S, Bruneau N, Danger J, Guilley S, Najm Z (2014) Analysis and improvements of the DPA contest v4 implementation. In: Chakraborty RS, Matyas V, Schaumont P (eds). Security, Privacy, and Applied Cryptography Engineering. SPACE 2014. Lecture Notes in Computer Science, vol 8804. Springer, Cham. https://doi.org/10.1007/978-3-319-12060-7_14

Coron J (2014) Higher order masking of look-up tables. In: Nguyen PQ, Oswald E (eds). Advances in Cryptology - EUROCRYPT 2014. EUROCRYPT 2014. Lecture Notes in Computer Science, vol 8441. Springer, Berlin, Heidelberg. https://doi.org/10.1007/978-3-642-55220-5_25

Coron J, Prouff E, Rivain M (2007) Side channel cryptanalysis of a higher order masking scheme. In: Paillier P, Verbauwhede I (eds). Cryptographic Hardware and Embedded Systems - CHES 2007. CHES 2007. Lecture Notes in Computer Science, vol 4727. Springer, Berlin, Heidelberg. https://doi.org/10.1007/978-3-540-74735-2_3

Coron J, Roy A, Vivek S (2014) Fast evaluation of polynomials over binary finite fields and application to side-channel countermeasures. In: Batina L, Robshaw M (eds). Cryptographic Hardware and Embedded Systems - CHES 2014. CHES 2014. Lecture Notes in Computer Science, vol 8731. Springer, Berlin, Heidelberg. https://doi.org/10.1007/978-3-662-44709-3_10

Coron J, Rondepierre F, Zeitoun R (2018) High order masking of look-up tables with common shares. IACR Trans Cryptogr Hardw Embed Syst 1:40–72. https://doi.org/10.13154/tches.v2018.i1.40-72

Duc A, Faust S, Standaert F (2019) Making masking security proofs concrete (or how to evaluate the security of any leaking device), extended version. J Cryptol 32(4):1263–1297. https://doi.org/10.1007/s00145-018-9277-0

Fang J (2009) Mixcolumn round transformation optimization and improvement in the aes algorithm. Microcomput Inf 25(21):49–51

FIPS Publication 140-3 (2019) Security Requirements for Cryptographic Modules. The National Institute of Standards and Technology. https://nvlpubs.nist.gov/nistpubs/FIPS/NIST.FIPS.140-3.pdf

Gilbert Goodwill BJ, Jaffe J, Rohatgi P, et al. (2011) A testing methodology for side-channel resistance validation. In: NIST non-invasive attack testing workshop, vol 7. NIST. pp 115–136. https://csrc.nist.gov/csrc/media/events/non-invasive-attack-testing-workshop/documents/08_goodwill.pdf

Ishai Y, Sahai A, Wagner DA (2003) Private circuits: Securing hardware against probing attacks. In: Boneh D (ed). Advances in Cryptology - CRYPTO 2003. CRYPTO 2003. Lecture Notes in Computer Science, vol 2729. Springer, Berlin, Heidelberg. https://doi.org/10.1007/978-3-540-45146-4_27

JTC I (2016) Iso/iec 17825:2016 information technology - security techniques - testing methods for the mitigation of non-invasive attack classes against cryptographic modules. https://www.iso.org/standard/60612.html

Kocher P (1996) Timing Attacks on Implementations of Diffie-Hellman, RSA, DSS, and Other Systems. In: Koblitz N (ed). Advances in Cryptology - CRYPTO '96. CRYPTO 1996. Lecture Notes in Computer Science, vol 1109. Springer, Berlin, Heidelberg. https://doi.org/10.1007/3-540-68697-5_9

Kocher PC, Jaffe J, Jun B (1999) Differential power analysis. In: Wiener M (ed). Advances in Cryptology ? CRYPTO? 99. CRYPTO 1999. Lecture Notes in Computer Science, vol 1666. Springer, Berlin, Heidelberg. https://doi.org/10.1007/3-540-48405-1_25

Ming J, Zhou Y, Cheng W, Li H, Yang G, Zhang Q (2020) Mind the balance: Revealing the vulnerabilities in low entropy masking schemes. IEEE Trans Inf Forensics Secur 15:3694–3708. https://doi.org/10.1109/TIFS.2020.2994775

Nassar M, Souissi Y, Guilley S, Danger J (2012) RSM: A small and fast countermeasure for aes, secure against 1st and 2nd-order zero-offset scas. In: 2012 Design, Automation & Test in Europe Conference & Exhibition, DATE 2012, Dresden, Germany, March 12-16, 2012. pp 1173–1178. https://doi.org/10.1109/DATE.2012.6176671

ParisTech T (2015) Dpa contest v4.2. documentation. http://www.dpacontest.org/v4/42_doc.php. Accessed 27 Aug 2015

Prouff E, Strullu R, Benadjila R, Cagli E, Dumas C (2018) Study of deep learning techniques for side-channel analysis and introduction to ASCAD database. IACR Cryptol ePrint Arch 53. http://eprint.iacr.org/2018/053

Prout A, Arcand W, Bestor D, Bergeron B, Byun C, Gadepally V, Houle M, Hubbell M, Jones M, Klein A, Michaleas P, Milechin L, Mullen J, Rosa A, Samsi S, Yee C, Reuther A, Kepner J (2018) Measuring the impact of spectre and meltdown. In: 2018 IEEE High Performance Extreme Computing Conference, HPEC 2018, Waltham, MA, USA, September 25-27, 2018. pp 1–5. https://doi.org/10.1109/HPEC.2018.8547554

Quisquater J, Samyde D (2001) Electromagnetic analysis (EMA): measures and counter-measures for smart cards. In: Attali I, Jensen T (eds). Smart Card Programming and Security. E-smart 2001. Lecture Notes in Computer Science, vol 2140. Springer, Berlin, Heidelberg. https://doi.org/10.1007/3-540-45418-7_17

Rivain M (2008) On the exact success rate of side channel analysis in the gaussian model. In: Selected Areas in Cryptography, 15th International Workshop, SAC 2008, Sackville, New Brunswick, Canada, August 14-15, Revised Selected Papers. pp 165–183. https://doi.org/10.1007/978-3-642-04159-4_11

Rivain M, Prouff E (2010) Provably secure higher-order masking of AES. In: Mangard S, Standaert FX (eds). Cryptographic Hardware and Embedded Systems, CHES 2010. CHES 2010. Lecture Notes in Computer Science, vol 6225. Springer, Berlin, Heidelberg. https://doi.org/10.1007/978-3-642-15031-9_28

Rivain M, Prouff E, Doget J (2009) Higher-order masking and shuffling for software implementations of block ciphers. In: Clavier C, Gaj K (eds). Cryptographic Hardware and Embedded Systems - CHES 2009. CHES 2009. Lecture Notes in Computer Science, vol 5747. Springer, Berlin, Heidelberg. https://doi.org/10.1007/978-3-642-04138-9_13

Schneider PH, Krishnamoorthy S (1996) Effects of correlations on accuracy of power analysis - an experimental study. In: Proceedings of the 1996 International Symposium on Low Power Electronics and Design, 1996, Monterey, California, USA, August 12-14, 1996. pp 113–116. https://doi.org/10.1109/LPE.1996.547490

Schramm K, Paar C (2006) Higher order masking of the AES. In: Pointcheval D (ed). Topics in Cryptology - CT-RSA 2006. CT-RSA 2006. Lecture Notes in Computer Science, vol 3860. Springer, Berlin, Heidelberg. https://doi.org/10.1007/11605805_14

Veshchikov N, Guilley S (2017) Implementation flaws in the masking scheme of DPA contest v4. IET Inf Secur 11(6):356–362. https://doi.org/10.1049/iet-ifs.2016.0475

Publisher's Note

Springer Nature remains neutral with regard to jurisdictional claims in published maps and institutional affiliations.

Submit your manuscript to a SpringerOpen journal and benefit from:

- ► Convenient online submission
- ► Rigorous peer review
- ► Open access: articles freely available online
- ► High visibility within the field
- ► Retaining the copyright to your article

Submit your next manuscript at ► springeropen.com