Non-profiled Mask Recovery: the impact of Independent Component Analysis

Si Gao[1], Elisabeth Oswald[1], Hua Chen[2], and Wei Xi[3]

[1] University of Bristol
[2] Institute of Software, Chinese Academy of Sciences
[3] Southern Power Grid Science Research Institute

Abstract. As one of the most prevalent SCA countermeasures, masking schemes are designed to defeat a broad range of side channel attacks. An attack vector that is suitable for low-order masking schemes is to try and directly determine the mask(s) (for each trace) by utilising the fact that often an attacker has access to several leakage points of the respectively used mask(s). Good examples for implementations of low-order masking schemes are the based on table re-computations and also the masking scheme in DPAContest V4.2. We propose a novel approach based on Independent Component Analysis (ICA) to efficiently utilise the information from several leakage points to reconstruct the respective masks (for each trace) and show it is a competitive attack vector in practice.

Keywords: Side Channel Analysis, Masking, Independent Component Analysis

1 Introduction

Over the past decade, Side Channel Attacks (SCAs) have become a major threat for various cryptographic devices. Depending on the specific attacker model, most SCAs can be divided into two categories: profiled attacks and non-profiled attacks. In a profiled attack, the attacker (a priori) creates direct approximations of the device's leakage function, and uses these in an attack. This typically results in very efficient attacks but with the strong assumptions about the capabilities of the attacker. Non-profiled attacks only require a proportional (or weaker) approximation of the device's leakage model. The canonical example of such an attack is to approximate the device leakage with the Hamming weight of intermediate values, and utilise correlation as a distinguisher. Attacks in both categories often proceed via a divide and conquer strategy, which require (in the divide step) to explicitly guess partial keys. Consequently (in a known plaintext setting) such attacks are limited to first and last rounds of typical block cipher constructions.

In 2017, Gao et al. proposed a new non-profiled SCA based on Independent Component Analysis (ICA) [1]. Assuming the observed leakages follow the weighted Hamming weight model, the ICA based attack recovers the intermediate states without making any explicit key guesses. In their paper, the authors demonstrate several applications of this approach, including a new key-distinguisher, attacking the middle encryption rounds as well as reverse engineering. However, all previous discussions about ICA-based SCA focus on unprotected implementations. We are hence interested in investigating if ICA-based SCA can be useful to attack protected implementations.

For ICA-based SCA to work it is imperative to have access to several leakage points for some (targeted) intermediate value. In masked implementations, one can often observe leakages related to the manipulation of masks in the processor. Hence, ICA-based SCA could be a powerful tool for mask recovery in masked implementations, in particular optimised low-order masking schemes.

Our Contribution. In this paper, we explore the potential of ICA to compromise implementations of some (low order) masking schemes. Specifically, in table re-computation schemes, the multiple XORs in the re-computation process naturally provide multiple leakage observations for ICA. Compared with previous attacks, our ICA-based mask recovery finds the n-bit random masks with only n leakage points, whereas previous attacks take 2^n points. Experiments confirm that for smaller Sboxes ($n = 4$), ICA-based attack outperforms horizontal attakcs on smart card implementations. For the Rotating Sbox Masking (RSM) scheme, which is used in the DPAContest V4, our analysis proves that if the attacker chooses the leakages wisely, the random masks can be recovered as an approximate ICA problem. Although the mask recovery becomes less accurate, the following key recovery is hardly affected.

Paper Organization. In Section 2, we briefly review the targeted masking schemes as well as our primary tool—ICA. Section 3 analyzes the leakage behaviour of table re-computation schemes in details. As the XORs naturally provide multiple leakage observations, ICA enables the attacker to determine both the random masks and the secret key. We present another masking scheme—the masking scheme in DPAContest V4.2—in Section 4. Although this scheme computes the masked tables offline, the relevant random indexes in each round provide considerable leakages for ICA-based SCA. Impacts of this approach and conclusions are further presented in Section 5.

2 Preliminaries

2.1 Masking schemes

To date, masking is one of the most prevalent countermeasures for software implementations. In general, a masking scheme conceals the cryptographic intermediate states with random values. As a result, the data-dependent leakage no longer relates to the secret key. Previous studies proposed a variety of masking schemes, such as affine masking [2], polynomial masking [3] and inner product masking [4]. In this paper, we focus on Boolean masking, the most frequently implemented approach. In a Boolean masking scheme with d-shares, an intermediate state x is split into d shares $\left(x^{(1)}, x^{(2)}, ..., x^{(d)}\right)$ where $x^{(1)} \oplus x^{(2)} \oplus ... \oplus x^{(d)} = x$.

As each leakage point only depends on one $x^{(i)}$, the attacker cannot learn any useful information, unless they combine the leakages of all d shares.

For linear components P, implementing a Boolean masking is quite straightforward: as $P(x^{(1)} \oplus x^{(2)} \oplus ... \oplus x^{(d)}) = P(x^{(1)}) \oplus P(x^{(2)}) \oplus ... \oplus P(x^{(d)})$, simply applying P to all d shares gives the expected outputs. For non-linear components (Sboxes), things become tricker. In order to ensure the output shares satisfy $S(x^{(1)} \oplus x^{(2)} \oplus ... \oplus x^{(d)}) = S_1(x^{(1)}) \oplus S_2(x^{(2)}) \oplus ... \oplus S_n(x^{(d)})$, at least one of the masked Sbox S_i must be related to multiple input shares. Three proposals exist in previous studies [5]:

- **Compute the Sbox arithmetically.** In 2003, Ishai, Sahai and Wagner proposed a provably secure higher-order masking scheme for bit-wise AND [6]. Alternatively, the whole Sbox can be computed as a bunch of masked ANDs and masked NOTs. Compared with the unprotected implementations, this construction significantly increases the computation cost.
- **Table Re-computation.** In many look-up table schemes, the masked Sbox is computed as a look-up table [7,8,9]. In the first step, these schemes often generate a masked table using all the shares from $x^{(1)}$ to $x^{(d-1)}$. Then, the output shares $\left(y^{(1)}, y^{(2)}, ..., y^{(d)}\right)$ can be found by simply looking up $x^{(d)}$ in the masked table. The major drawback of this approach, is that the re-computation stage is not only costly, but also exploitable. For an n-bit Sbox, this procedure provides 2^n leakage points for each data share $x^{(i)}$. Thus, the attacker can collect all leakage points on the trace ("horizontally") and use a standard DPA style attack to recover $x^{(i)}$. For $n = 8$, this horizontal attack is actually quite efficient for software implementations [10,11].
- **Global look-up tables.** Alternatively, the masked table can also be computed offline [8]. In this case, a masked table is generated for each possible mask and stored in the data RAM/ROM. Considering the enormous memory cost, this approach is more suitable for smaller Sboxes (eg. 4-bit Sbox)[1]. For larger Sboxes, it often applies in Low-Entropy Masking Schemes (LEMS), such as the Rotating Sbox Masking (RSM) [13]. Instead of random masks, LEMS usually uses a precomputed set of constant masks, which significantly reduces the memory cost [13]. As a lightweight SCA countermeasure, it is LEMS's design philosophy to resist not all but a selection of important and powerful attacks [13]. Results from DPA Contest v4 and v4.2 are consistent with such statement: in the profiling case, the secret key can be found with only one trace [14].

2.2 Independent Component Analysis

Independent Component Analysis (ICA) [15] belongs to a class of problems called *Blind Source Separation* (BSS), which requires to separate a set of mixed signals, without the aid of information about the source signals or the mixing process. A common example is the *cocktail party problem* in which the challenge of a partygoer is to pick out a single conversation when in a noisy room.

[1] For specific processors, such implementation is not necessarily secure [12].

Suppose we have n simultaneous conversations (sources) $\mathbf{S} = \{s_1, s_2, ..., s_n\}$ going on in the party room. Microphones are placed in different positions, recording m mixtures (observations) of the original sources $\mathbf{Y} = \{y_1, y_2, ..., y_m\}$. Assuming the observation y_j is a linear mixture of all sources, we have

$$y_j = a_{j,0} + a_{j,1}s_1 + a_{j,2}s_2 + ... + a_{j,n}s_n$$

where $a_{j,i}$ stands for the real-valued coefficient. The overall mixing procedure can be written as

$$\mathbf{Y} = \mathbf{AS}$$

where $\mathbf{A}$ is called the *mixing matrix*. In signal processing, such statistical model is called *Independent Component Analysis* [15]. With additional multivariate Gaussian noise $\mathbf{N}$, the noisy ICA model is defined as

$$\mathbf{Y} = \mathbf{AS} + \mathbf{N}$$

The goal of ICA, is to recover the unknown sources $\mathbf{S}$ from the observation $\mathbf{Y}$, without knowing the mixing matrix $\mathbf{A}$ or the Gaussian noise $\mathbf{N}$ in advance.

2.3 ICA in Side Channel Analysis

Assuming the target device's leakage function is linear (in the bits of the intermediate values), recovering the secret intermediate values in SCA is quite similar to an ICA problem [1]. Specifically, when operating an n-bit intermediate state x, the data-dependent leakage can be written as

$$\mathrm{L}(x) = \alpha_0 + \alpha_1 x_1 + \alpha_2 x_2 + ... + \alpha_n x_n,\ \ \alpha_i \in \mathbb{R} \tag{1}$$

Here x_i represents the i-th bit [2] of $x$and L is a linear leakage function. This leakage function has the same form as one ICA observation (i.e. y_j in Sect. 2.2).

However for ICA we need more than a single observation. Suppose that the device not only computes x but also computes some other intermediate state $x' = x \oplus c$ (c is a constant) at some point. Then, the attacker can also learn the leakage of $L(x')$ [3]. Take $c = 00...01$ as an example, we have:

$$\begin{aligned} L(x') &= L(x \oplus 00...01) \\ &= \alpha_0 + \alpha_1 x_1 + \alpha_2 x_2 + ... + \alpha_n (x_n \oplus 1) \\ &= \alpha_0 + \alpha_1 x_1 + \alpha_2 x_2 + ... + \alpha_n (1 - x_n) \\ &= (\alpha_0 + \alpha_n) + \alpha_1 x_1 + \alpha_2 x_2 + ... - \alpha_n x_n \end{aligned}$$

It is not hard to see that such leakage can be regarded as the leakage from the same intermediate state x, but with a different linear leakage function L'. Thus,

[2] Throughout this paper, we always use subscript i as the i-th bit. Unlike traditional SCA, the intermediate state x here represents the random mask, which is not dependent on a key guess k .

[3] For simplicity, we assume all leakage share the same leakage function L. However, ICA does work with different L-s, as long as they are all linear combinations of x.

if the targeted implementation has some operand like $x \oplus c$, the attacker may be able to manipulate c to get multiple "observations"for the intermediate state x. Assuming the attacker can get enough observations (the number of observations $m \geq n$), in theory, he (or she) can solve the intermediate state x as an noisy ICA problem.

In practice, considering side channel leakage usually contains high level of noise, the authors also proposed an specific ICA algorithm for SCA. Due to the space limit, we omit further details: interested readers can find this part in [1].

Unlike other traditional SCAs, recovering x with ICA does not involve any key guess. As a consequence, ICA-based SCA serves as a perfect tool for SCA in the middle rounds or SCA-based reverse engineering [1]. Indeed, the authors already provide realistic experiments to verify their results on certain software implementations. On the other hand, as stated in [1], in many realistic circumstances, finding such XOR constant c might not be an easy task. For this reason, to date, the applications of ICA-based SCA are restricted to unprotected cryptographic implementations.

3 ICA-based Attack on a Table Re-computation Scheme

In the following, we analyse the potential application of ICA on a few masking schemes. Perhaps surprisingly, for some masking schemes, constructing multiple observations becomes much easier. The following two sections present two case studies: for each case study, we will review its mask computation, analyze its leakage and show how ICA-based SCA enables the recovery of the random masks. Comparison with previous attacks and experimental verifications are also provided in each case. We begin by studying a table re-computation scheme.

3.1 Table Re-computation Schemes

Considering the memory cost, masking schemes with global look-up table can hardly be applied to larger Sboxes (eg. the Sbox in AES). Thus, many masking schemes choose to generate the masked table online. In a d-shares table re-computation scheme, ($x^{(1)}$,$x^{(2)}$,...,$x^{(d-1)}$) is taken to the computation to create a masked table T. In the last step, the implementation simply looks up $x^{(d)}$ in T and returns $T(x^{(d)})$ as the output shares. To ensure its security against SCA, designers may also add some other procedures, such as refreshing T with fresh randomness after each table look-up [9]. Meanwhile, most masked table re-computations are rather similar: for clarity, we present a d-shares table re-computation procedure in Algorithm 1.

3.2 Previous Attacks

Note that in Algorithm 1, line 3 always produces 2^n leakages for each share. More specifically, assume the leakage function is L, the attacker learns the leakages of $(L(x^{(i)}), L(x^{(i)} \oplus 1), ..., L(x^{(i)} \oplus (2^n - 1)))$. As all these leakages depend on the

Algorithm 1 A d-shares table re-computation for an n-bit Sbox

```
Input: $x^{(1)},...,x^{(d)}$ such that $x = x^{(1)} \oplus x^{(2)} \oplus ... \oplus x^{(d)}$
Input: Shared table $T$ such that $\bigoplus_i T(u)^{(i)} = S(u)$
Output: Shared table $T$ such that $\bigoplus_i T\left(x^{(d)}\right)^{(i)} = S(x)$
1: for $i = 1$ to $d - 1$ do
2:     for all $u \in \{0,1\}^n$ do
3:         $T'(u) = T\left(u \oplus x^{(i)}\right)$
4:     end for
5:     $T = T'$
6: end for
```

same share $x^{(i)}$, the attacker can take a guess about $x^{(i)}$ and verify this guess with Correlation Power Analysis (CPA) [16]. Unlike traditional CPA which utilises a specific leakage point across many traces (i.e. a "vertical" attack), this attack utilizes all the 2^n leakages on the same trace (i.e. it is a "horizontal" attack). Having recovered the masks, key recovery is trivial: since all $d-1$ input shares (random masks) are already known, a traditional vertical CPA on the leakage of $x^{(d)}$ reveals the secret key. Previous studies proved that, for 8-bit Sboxes ($n = 8$), such "horizontal" attack is a serious threat for table re-computation schemes [10].

A common countermeasure for the horizontal attacks is to randomly shuffle the constant u in line 3. Since the computation follows some random order ($\varphi(0)$,$\varphi(1)$,...,$\varphi(2^n-1)$), $x^{(i)}$ alone can no longer determine all the 2^n leakages. However, for many smart card applications, generating and storing an n-bit random permutation φ in memory is far too expensive. Instead, they prefer to use some pseudo-random function φ that can be computed online. However, the computation of φ provides new leakages for the attacker. Tunstall et al. showed that the attacker can easily explore such leakages and recover the entire permutation φ [11]. Moreover, Bruneau et al. proposed a multi-variate attack which combines all 2^n leakages on one trace into a statistic that depends on $x^{(i)}$ [5]. As the combination is unordered, random shuffling does not affect the final statistic. Although $x^{(i)}$ cannot be recovered, the attacker finds the secret key through higher-order attacks, with the leakage of $x^{(i)}$ as well as this statistic.

3.3 ICA-based attack

Mask recovery The leakages that occur in table re-computation schemes are a perfect match for ICA. Specifically, each bit of the intermediate state x now becomes an independent binary source. Assuming the leakage function is linear, the attacker can always use the leakage of x as one observation for ICA. As stated previously, the leakage of $L(x \oplus c)$ can also be regarded as the leakages of x with a different leakage function. In other words, for table re-computation schemes, the attacker can always find 2^n independent observations through 2^n

XOR constants. In fact, ICA only needs n observations for a successful recovery. Taking noise into consideration, the formal model can be written as:

$$l = L(x^{(i)}) + \mathbf{N}$$

where $\mathbf{N}$ represents the random noise. As stated in Section 2.3, ICA-based SCA helps to recover the secret share $x^{(i)}$.

Key recovery Since all $d-1$ secret shares are already recovered, the following key recovery becomes trivial. Take the last round attack of AES for instance, assuming the corresponding ciphertext byte is c and related round key byte is k, we have

$$x^{(d)} = S^{-1}(c \oplus k) \oplus x^{(1)} \oplus x^{(2)} \oplus ... \oplus x^{(d-1)}$$

Since the attacker has the leakage of $x^{(d)}$, traditional CPA helps to determine the correct key guess for k, as long as the value of $x^{(1)} \oplus x^{(2)} \oplus ... \oplus x^{(d-1)}$ is given.

Comparison with previous attacks Compared with horizontal CPAs, our ICA-based mask recovery uses only n leakage samples. Since horizontal CPA takes guesses about $x^{(i)}$, it only applies to one certain trace. In other words, the sample size for horizontal CPA on table re-computation schemes is always 2^n. Previous studies showed that for $n = 8$, horizontal CPA works quite well with software implementations [10,11]. However, for smaller Sboxes (eg. $n = 4$), horizontal CPA becomes less effective [11]. This is not surprising though: as a non-profiled attack, CPA requires several traces to achieve a stable recovery. For our ICA-based mask recovery, smaller Sbox is hardly a problem. Since our approach uses only n leakage points, it works well even if $n = 2$. Meanwhile, the mask recovery in horizontal CPA is basically a one-dimensional attack: since each trace has different input shares (random masks), horizontal CPA only works on the horizontal axis. The following key recovery, on the other hand, only collects information on the vertical axis. In Bruneau et al.'s work [5], since the horizontal leakages are packed into one statistic, their attack mainly works on the vertical axis. On the contrary, our approach is essentially a two-dimensional attack. Both the multiple leakages on one trace ("horizonal") and the leakage model shared by all traces ("vertical") are taken into consideration. In some cases, this two-dimensional property becomes a limitation: if the target implementation uses random shuffling as a countermeasure, the frequently changing random order φ completely defeats our attack. Since the 2^n horizontal leakages in our attack are not packed together (like Bruneau et al.'s attack), this random order prevents our attack to explore the vertical information. However, such protection only works if the designers use a new φ for each encryption. If the random φ is fixed, our attack works exactly the same way: as ICA does not require to know the mixing matrix, we can recover $x^{(i)}$ without knowing φ. For easy comparison, we list the attacks mentioned above with 2-shares table re-computation schemes in Table 1[4].

[4] A v-variate attack means it takes v leakage samples in total.

Table 1. Comparison of attacks with 2-shares table re-computation schemes

	Variate	Fix shuffle	Random shuffle
horizontal CPA	$2^n + 1$		
Bruneau's attack	$2^{n+1} + 1$	✓	✓
our approach	$n + 1$	✓	

3.4 Experimental Validation

To show that our ICA-based attack works, we have implemented a 2-shares version of Coron's masking scheme [9] on an IC card with 8-bit microprocessor (Atmega163). The power consumption was measured with a PicoScope 3206D oscilloscope at a sampling rate of 1 GSa/s. The target cipher uses the 4-bit Sbox of PRESENT [17]. Since the previous studies already proved that horizontal CPA works well with 8-bit Sboxes, here we aim to test whether it still gives satisfying recovery with smaller Sboxes. Our entire trace set contains 200 traces, with 2 000 000 samples covering the Sbox computation in the last round. Results from both horizontal CPA and our ICA-based attack are presented in Figure 1.

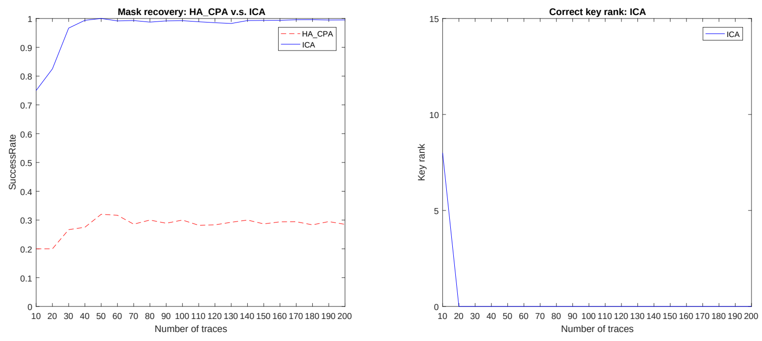

Fig. 1. Mask and key recovery: horizontal CPA v.s. ICA

Clearly the small 4-bit Sbox is an issue for horizontal CPA: as there are only 16 leakage samples on each trace, mask recovery becomes less reliable. In our experiments, only 30% of the random masks are successfully recovered. As most recovered masks are incorrect, further key recovery becomes less effective. On the other hand, our ICA-based mask recovery finds over 90% of the random masks correctly with only 40 traces Figure 1 shows such attack is quite efficient: the key recovery becomes stable after only 20 traces.

4 ICA-based Attack on DPAContest v4.2

As table re-computation schemes produce the leakages of $(x^{(i)}, x^{(i)} \oplus 1, ..., x^{(i)} \oplus 2^n - 1)$, recovering the random masks with ICA seems quite straightforward. In the following, let us consider a more subtle example: the masking scheme in DPAContest v4.2.

4.1 The Rotating Sbox Masking Scheme

Unlike typical table re-computation schemes, the masking scheme in DPAContest v4 uses global look-up tables, where the masked tables are pre-computed offline. As stated previously, for larger Sboxes (like AES), storing all possible masked tables is impossible for many commonly used encryption devices. Instead, DPAContest v4 uses Rotating Sbox Masking (RSM) [13], which uses a set of constant masks rather than completely random masks. More specifically, in the latest version (DPAContest v4.2) [18], the implementation uses the following mask set:

$$\begin{aligned} M[0:15] = \{&\mathrm{0x03}, \mathrm{0x0c}, \mathrm{0x35}, \mathrm{0x3a}, \mathrm{0x50}, \mathrm{0x5f}, \mathrm{0x66}, \mathrm{0x69}, \\ &\mathrm{0x96}, \mathrm{0x99}, \mathrm{0xa0}, \mathrm{0xaf}, \mathrm{0xc5}, \mathrm{0xca}, \mathrm{0xf3}, \mathrm{0xfc}\} \end{aligned}$$

Before any encryption, 16 masked tables (MS_i) are pre-computed and stored in memory:

$$\mathrm{MS}_i\,(x) = S(x \oplus M[i]) \oplus M[(i+1) \bmod 16]$$

In each encryption, the encryption device randomly pick a 16 elements offset array $O[0:15]$, where each $O[i]$ is a 4-bit random offset. According to the mask set, the initial 128 bit mask is

$$\mathrm{Mask}(0) = \{M[O[0]], M[O[1]], ..., M[O[15]]\}$$

At the end of one encryption round, each mask byte is "rotated" right for one position in the masking set. Thus, in the $(r+1)$-th round, the input mask is:

$$\mathrm{Mask}(r) = \{M[(O[0]+r)\mathrm{mod}16], M[(O[1]+r)\mathrm{mod}16], ..., M[(O[15]+r)\mathrm{mod}16]\}$$

Algorithm 2 describes the masked round function of AES-128 in detail.

In addition, considering the threat of higher-order SCA, random shuffling is applied to the first/last round. Since the Sbox computation order is not given, the attacker can hardly combine leakages from multiple traces and learn the secret key from conventional vertical SCA.

Algorithm 2 Masked round function of AES-128 in DPAContest v4.2

```
Input: masked input state $X = \{X[0], X[1], ..., X[15]\}$
Input: random mask index array $O = \{O[0], O[1], ..., O[15]\}$
Input: subkey $RK = \{RK[0], RK[1], ..., RK[15]\}$
Output: masked output state $X = \{X[0], X[1], ..., X[15]\}$
1: $X = X \oplus RK$                                        ▷ AddRoundKey
2: for $i = 0$ to 15 do
3:     $X_i = MS_{(O[i]+r) \bmod 16}(X[i])$                 ▷ Masked Sbox
4: end for
5: $X = \text{ShiftRow}(X)$
6: $X = \text{MixColumn}(X)$
7: $X = X \oplus \text{MixColumn}(\text{ShiftRow}(\text{Mask}(r+1))) \oplus \text{Mask}(r+1)$   ▷ Mask Compensation
```

4.2 Previous Attacks

Although there are many attacks in the hall of fame of DPAContest V4.2, fewer participants give detailed descriptions of their attacks. As a result, we can only present a brief overview of the current results. Apparently, profiling attacks work well with DPAContest v4.2. Most profiling attack recovers the secret key with a few traces, whereas the best one works with only one trace. On the other hand, most non-profiled attacks use much more traces. To date, the best non-profiled attack existed is due to Zeyi Liu et al [14]. According to the hall of fame, their attack takes only 14 traces, whereas all other non-profiled attacks need a few hundreds traces.

In theory, horizontal CPA still works for this scheme. Denote the 4-bit $O[0]$ as x, in each Sbox computation, the processor needs x to decide which masked table should be used. Algorithm 3 presents the assembly codes of the Sbox computation in DPAContest v4.2.

Algorithm 3 ASM codes of the masked Sbox computation in DPAContest v4.2

```
 1: ldi YH,hi8(__offset__)        ▷ point to the offset array location
 2: ldi YL,0x00
 3: ld offset, Y                  ▷ load offset $x$
 4: ldi ZH, hi8(aes_sbox0)
 5: add offset,I2                 ▷ $x = x + r$
 6: andi offset,0x0F              ▷ $x = x \ mod\ 16$
 7: add ZH, offset                ▷ Determine the masked table
 8: clr ZL
 9: mov ZL, ST11                  ▷ Table look up
10: clr ST11
11: lpm ST11, Z
```

As we can see in line 5-6, in the table look-up procedure, the attacker finds the leakage of $(x + r) \bmod 16$. Although the first/last round Sbox computation is shuffled, the rest 8 rounds in the middle still provide exploitable leakages. Specifically, the data-dependant leakages for round 2-9 can be written as $\{L((x+1) \bmod 16), L((x+2) \bmod 16), ..., L((x+8) \bmod 16)\}$. In this case, the attacker can guess x and verify his guess with horizontal CPA. Nonetheless, considering there are only 8 leakage samples available, recovering the random masks with horizontal CPA seems to be a difficult task.

4.3 ICA-based Attack

Apparently, applying ICA in this scheme is not as straightforward as table recomputation schemes. Following the previous construction, the random mask index x can be regarded as 4-bit binary sources. However, as the leakages here depend on $(x+r) \bmod 16$, the "XOR-constant "method [1] no longer provides multiple observations. Nonetheless, in round 9, we have

$$(x+8) \bmod 16 = x \oplus 8$$

As a result, the leakage of round 9 forms a valid ICA observation. Similarly, the Boolean function of $y = (x+4) \bmod 16$ can be written as:

$$\begin{aligned} y_1 &= x_1 \oplus x_2 \\ y_2 &= x_2 \oplus 1 \\ y_3 &= x_3 \\ y_4 &= x_4 \end{aligned}$$

Clearly, the least significant 3 bits have the same expressions as $x \oplus 4$. The only difference lies in the most significant bit y_1. Since ICA is a linear[5] procedure, the linear mixture of x can never express $x_1 \oplus x_2$. As a consequence, in ICA, the leakage of y_1 can be regarded as random noise. More specifically, in round 5,

$$\begin{aligned} l &= L(y) + \mathbf{N} \\ &= \alpha_0 + \alpha_1 y_1 + \alpha_2 y_2 + \alpha_3 y_3 + \alpha_4 y_4 + \mathbf{N} \\ &= \alpha_0 + \alpha_2 - \alpha_2 x_2 + \alpha_3 x_3 + \alpha_4 x_4 + \mathbf{N} + \alpha_1 (x_1 \oplus x_2) \\ &= L'(x) + \mathbf{N}' \end{aligned}$$

In other words, the leakages in round 5 can be regarded as a noisier observation of x with an equivalent leakage function where $\alpha_1 = 0$. Similar property holds for the leakages of $(x+2) \bmod 16$ and $(x+1) \bmod 16$, although the signal-to-noise-ratio (SNR) will be further reduced. As a result, attackers can recover the offset $O[0]$ with the leakages from round $(2, 3, 5, 9)$. With the random masks recovered, the following key recovery becomes much easier. Unlike the Sbox, the MixColumn computations in the first round are not shuffled. Therefore, attackers can explore the leakages of MixColumn and learn the secret key through conventional vertical SCA.

[5] Here linear means linear on real values, rather than GF_{2^n}.

4.4 Experimental Validation

We show how our ICA-based attack can be applied here with the EM traces provided by DPAContest [14]. In our experiments, the leakage of offset $O[0]$ appears not only in the Sbox computations, but also in the MixColumn computations. For better recovery, in each round, our ICA-based analysis takes both observations as its inputs. As a result, in the mask recovery stage, our analysis uses 8 observations to retrieve 4 sources. Even with these extra leakages, our mask recovery is not as good as the previous section. As we can see in Figure 2, the success rate for our ICA-based mask recovery is around 80%. Nonetheless, the following key-recovery proves that 80% accuracy is still good enough for key recovery: the correct key is almost determined after only 30 traces. On the other hand, in our experiment, 8 leakages can hardly support a horizontal CPA: only 10% of the recovered masks are correct and thus key recovery becomes infeasible.

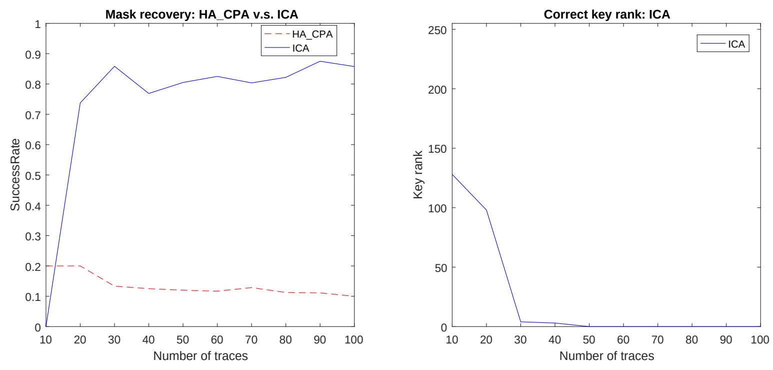

Fig. 2. Mask and key recovery: horizontal CPA v.s. ICA

5 Conclusion

In 2017, Gao et al. have proposed a novel side channel analysis based on independent component analysis (ICA) [1]. As this ICA-based SCA does not take a "guess-and-determine" procedure, this approach is quite useful for attacking the middle rounds or reverse engineering. However, previous work only studied unprotected implementations.

In this paper, we demonstrated the potential of ICA to defeat some masking schemes: table re-computation and the RSM masking scheme in DPAContest V4.2. Our analysis shows that, assuming the attacker can choose the leakage samples wisely, the random masks in both schemes can be effectively recovered. Compared with the previous attacks, our mask recovery requires fewer leakages. For masking scheme designers, our attack is another warning: horizontal attacks are indeed serious practical threats. If the same (or relevant) mask appears multiple times during the computation, the attacker may learn considerable information about the mask, even if it never mixes with any masked intermediate state.

6 Acknowledgements

This work has been funded in part by the National Key R&D Program of China(2018YFB0904900, 2018YFB0904901) and EPSRC under grant agreement EP/N011635/1 (LADA).

References

1. Gao, S., Chen, H., Wu, W., Fan, L., Cao, W., Ma, X.: My Traces Learn What You Did in the Dark: Recovering Secret Signals Without Key Guesses. In Handschuh, H., ed.: Topics in Cryptology — CT-RSA 2017: The Cryptographers' Track at the RSA Conference 2017, San Francisco, CA, USA, February 14-17, 2017, Proceedings. Springer International Publishing, Cham (2017) 363–378
2. von Willich, M.: A Technique with an Information-Theoretic Basis for Protecting Secret Data from Differential Power Attacks. In Honary, B., ed.: Cryptography and Coding: 8th IMA International Conference Cirencester, UK, December 17-19, 2001 Proceedings. Springer Berlin Heidelberg, Berlin, Heidelberg (2001) 44–62
3. Roche, T., Prouff, E.: Higher-order glitch free implementation of the AES using Secure Multi-Party Computation protocols. Volume 2. (2012) 111–127
4. Balasch, J., Faust, S., Gierlichs, B., Verbauwhede, I.: Theory and Practice of a Leakage Resilient Masking Scheme. In Wang, X., Sako, K., eds.: Advances in Cryptology — ASIACRYPT 2012: 18th International Conference on the Theory and Application of Cryptology and Information Security, Beijing, China, December 2-6, 2012. Proceedings. Springer Berlin Heidelberg, Berlin, Heidelberg (2012) 758–775
5. Bruneau, N., Guilley, S., Najm, Z., Teglia, Y.: Multi-variate High-Order Attacks of Shuffled Tables Recomputation. In Güneysu, T., Handschuh, H., eds.: Cryptographic Hardware and Embedded Systems – CHES 2015: 17th International Workshop, Saint-Malo, France, September 13-16, 2015, Proceedings. Springer Berlin Heidelberg, Berlin, Heidelberg (2015) 475–494
6. Ishai, Y., Sahai, A., Wagner, D.: Private Circuits: Securing Hardware against Probing Attacks. In Boneh, D., ed.: Advances in Cryptology — CRYPTO 2003: 23rd Annual International Cryptology Conference, Santa Barbara, California, USA, August 17-21, 2003. Proceedings. Springer Berlin Heidelberg, Berlin, Heidelberg (2003) 463–481
7. Messerges, T.S.: Securing the AES DPAContestFinalists Against Power Analysis Attacks. In Goos, G., Hartmanis, J., van Leeuwen, J., Schneier, B., eds.: Fast Software Encryption: 7th International Workshop, FSE 2000 New York, NY, USA, April 10-12, 2000 Proceedings. Springer Berlin Heidelberg, Berlin, Heidelberg (2001) 150–164
8. Prouff, E., Rivain, M.: A Generic Method for Secure SBox Implementation. In Kim, S., Yung, M., Lee, H.W., eds.: Information Security Applications: 8th International Workshop, WISA 2007, Jeju Island, Korea, August 27-29, 2007, Revised Selected Papers. Springer Berlin Heidelberg, Berlin, Heidelberg (2007) 227–244

9. Coron, J.S.: Higher Order Masking of Look-Up Tables. In Nguyen, P.Q., Oswald, E., eds.: Advances in Cryptology — EUROCRYPT 2014: 33rd Annual International Conference on the Theory and Applications of Cryptographic Techniques, Copenhagen, Denmark, May 11-15, 2014. Proceedings. Springer Berlin Heidelberg, Berlin, Heidelberg (2014) 441–458
10. Pan, J., den Hartog, J.I., Lu, J.: You Cannot Hide behind the Mask: Power Analysis on a Provably Secure S-Box Implementation. In Youm, H.Y., Yung, M., eds.: Information Security Applications: 10th International Workshop, WISA 2009, Busan, Korea, August 25-27, 2009, Revised Selected Papers. Springer Berlin Heidelberg, Berlin, Heidelberg (2009) 178–192
11. Tunstall, M., Whitnall, C., Oswald, E.: Masking Tables—An Underestimated Security Risk. In Moriai, S., ed.: Fast Software Encryption: 20th International Workshop, FSE 2013, Singapore, March 11-13, 2013. Revised Selected Papers. Springer Berlin Heidelberg, Berlin, Heidelberg (2014) 425–444
12. Kutzner, S., Nguyen, P.H., Poschmann, A., Wang, H.: On 3-share threshold implementations for 4-bit s-boxes. In: Constructive Side-Channel Analysis and Secure Design - 4th International Workshop, COSADE 2013, Paris, France, March 6-8, 2013, Revised Selected Papers. (2013) 99–113
13. 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: 2012 Design, Automation & Test in Europe Conference & Exhibition (DATE). 1173–1178
14. TELECOM ParisTech SEN research group: DPA contest v4. http://www.dpacontest.org/v4/
15. Hyvärinen, A., Oja, E.: Independent component analysis: algorithms and applications. Neural Networks **13** (2000) 411 – 430
16. Brier, E., Clavier, C., Olivier, F.: Correlation Power Analysis with a Leakage Model. In Joye, M., Quisquater, J.J., eds.: Cryptographic Hardware and Embedded Systems — CHES 2004. Volume 3156 of Lecture Notes in Computer Science. Springer Berlin Heidelberg (2004) pp. 16–29
17. Bogdanov, A., Knudsen, L.R., Leander, G., Paar, C., Poschmann, A., Robshaw, M.J.B., Seurin, Y., Vikkelsoe, C.: PRESENT: An Ultra-Lightweight Block Cipher. In Paillier, P., Verbauwhede, I., eds.: Cryptographic Hardware and Embedded Systems — CHES 2007: 9th International Workshop, Vienna, Austria, September 10-13, 2007. Proceedings. Springer Berlin Heidelberg, Berlin, Heidelberg (2007) 450–466
18. 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.: Security, Privacy, and Applied Cryptography Engineering: 4th International Conference, SPACE 2014, Pune, India, October 18-22, 2014. Proceedings. Springer International Publishing, Cham (2014) 201–218