Constant-Time Higher-Order Boolean-to-Arithmetic Masking

Michael Hutter and Michael Tunstall

Cryptography Research,
425 Market Street, 11th Floor, San Francisco,
CA 94105, United States
{michael.hutter,michael.tunstall}@cryptography.com

Abstract. Converting a Boolean mask to an arithmetic mask, and vice versa, is often required in implementing side-channel resistant instances of cryptographic algorithms that mix Boolean and arithmetic operations. In this paper, we describe a method for converting a Boolean mask to an arithmetic mask that runs in constant time for a fixed order. We propose explicit algorithms for a second-order secure Boolean-to-arithmetic mask conversion that uses 24 instructions and for a third-order secure mask conversion that uses 56 instructions. We show that our solution is more efficient than previously proposed methods for any choice of masking-scheme order, typically by several orders of magnitude.

Keywords: Side-channel analysis, higher-order DPA, mask switching, countermeasures, Boolean-to-arithmetic mask conversion, SHA.

1 Introduction

Differential Power Analysis (DPA) was introduced as a means of extracting cryptographic keys by Kocher, Jaffe, and Jun [1] in 1999, who noted that the power consumption of a device was dependent on the operations being performed, and the value of the operands used. They showed that one could acquire the power consumption over time while a device was computing a cryptographic algorithm, and analyze the acquisitions to determine the cryptographic key. Subsequently, it was shown that the same analyses could be conducted by exploiting other side channels, e.g., the changes in the electromagnetic field around a microprocessor [2,3,4].

A typical DPA attack involves acquiring a series of acquisitions while a device is operating on varying inputs and analyzing the power traces by comparing what occurred at the same point in time in each trace. The simplest analysis is to choose one bit of an intermediate state and divide the set of acquisitions depending on the value of this bit, make two mean traces and subtract one trace from the other point-by-point. A significant difference should be visible in the trace corresponding to where this intermediate state was created by the device. This is typically referred to as a *first-order* analysis, as each point in the output trace is dependent on the same point in time in the acquisitions. If two (or more)

points in each trace are combined, we refer to as a *second-order* (or *higher-order*) analysis.

To prevent the side-channel analyses of a cryptographic implementation, one would typically apply a random mask to the input such that operating on the masked data is indistinguishable from random data. A common masking technique is Boolean masking, where an input word gets masked by a random value. All operations are then performed using the Boolean-masked data. However, there exist many cryptographic algorithms that require both Boolean and arithmetic operations, such as integer addition, e.g., SHA-1, SHA-2, Blake, ChaCha, Skein, IDEA, RC6, etc. Masked versions of these algorithms therefore require changing Boolean masks into arithmetic masks, and vice versa, which we refer to as "Boolean-to-arithmetic" and "Arithmetic-to-Boolean" mask conversions, respectively.

In 2001, Goubin proposed an efficient constant-time method for Boolean-to-arithmetic mask conversion [5]. His method is secure against first-order analysis, but does not resist second-order attacks. The solutions in the literature use *recursive* methods [6,7], where the missing carry bits are calculated using a masked-adder structure, or Look-up Table based methods [8,9], that perform pre-computations and store intermediates in memory. It has also been suggested that higher-order versions of Boolean-to-arithmetic mask conversion cannot be done in constant time [8].

In this paper, we present novel algorithms for higher-order secure Boolean-to-arithmetic mask conversion. All proposed methods run in constant time and are independent on the input word size. In particular, we present a second-order secure algorithm that requires only 24 instructions and a third-order secure algorithm that requires only 56 instructions. Furthermore, we provide a generalized algorithm that provides side-channel resistance for masking schemes of any higher order. Our n-th order secure algorithm is significantly faster than the best recursive method in the literature [6] for any order, often by several orders of magnitude.

Outline. The paper is organized as follows. In Section 2, we describe the Boolean-to-arithmetic mask conversion problem and discuss previous work. In Section 3 we present a novel constant-time algorithm to perform a secure second-order Boolean-to-Arithmetic mask conversion, and generalize it to higher orders in Section 4. Conclusions are drawn in Section 5.

2 Boolean-to-Arithmetic Masking

In this paper we shall consider operations available in a typical microprocessor with registers of a fixed bit length. Specifically, we shall consider values that are in the field $(\mathbb{Z}_{2^k}, \oplus, +)$ where $k \in \mathbb{Z}_{\geq 0}$ is the bit length of the registers used, $\oplus$ is a bitwise XOR operation and $+$ is integer addition. Other operations are available in a typical microprocessor, but are not relevant to the algorithms described in this paper.

We define the problem of changing a Boolean mask into an arithmetic mask as follows:

Definition 1 (*Boolean-to-Arithmetic Mask Conversion Problem*). *Given $x' = x \oplus r$, where $x, r \in (\mathbb{Z}_{2^k}, \oplus, +)$, as a Boolean masked secret x and r is a random value taken from $\mathbb{Z}_{2^k}$, we wish to be able to compute $x'' = x + s$, with $s \in (\mathbb{Z}_{2^k}, \oplus, +)$ where $k \in \mathbb{Z}_{\geq 0}$, without revealing any information on x through some side channel. Where x'' is the arithmetically masked secret x and s is a random value taken from $\mathbb{Z}_{2^k}$.*

One naïve approach would be to perform the conversion directly by simply removing the Boolean mask and by adding an arithmetic mask afterwards, i.e.,

$$(x' \oplus r) + s = ((x \oplus r) \oplus r) + s = x + s = x'' ,$$

using the notation given in Definition 1. This, however, would manipulate x directly, allowing an attacker to use side-channel analysis to determine that a hypothesized value of x is manipulated during the mask conversion. Hence, one needs to use an algorithm where all intermediates are statistically independent of the secret x.

Definition 1 generalizes to higher-order masking schemes as follows:

Definition 2 (*Higher-Order Boolean-to-Arithmetic Mask Conversion Problem*). *Assuming a masking scheme of order n. Then, given $x' = x \oplus r_1 \oplus \ldots \oplus r_n$, where $x, r_i \in (\mathbb{Z}_{2^k}, \oplus, +)$, $k \in \mathbb{Z}_{\geq 0}$ for $i \in \{1, \ldots, n\}$, as a Boolean masked secret x and n a random values, r_i for $i \in \{1, \ldots, n\}$, taken from $\mathbb{Z}_{2^k}$, we wish to compute $x'' = x + s_1 + \ldots + s_n$, with $s_i \in (\mathbb{Z}_{2^k}, \oplus, +)$ for $i \in \{1, \ldots, n\}$, without revealing any information on x through some side channel. Where x'' is the arithmetically masked secret x and s_i, for $i \in \{1, \ldots, n\}$, are random values taken from $\mathbb{Z}_{2^k}$.*

Higher-order mask conversion methods require that the masks used for the arithmetically masked output are not related to the Boolean masked input to avoid any side-channel leakage. If we consider, without loss of generality, a second-order secure Boolean-to-arithmetic mask conversion that uses the same input masks r_1 and r_2 to mask the output, information would leak through the carries generated from the arithmetic masks. For ease of expression, we shall consider an attacker able to XOR two intermediate states together in a second-order side-channel attack (a very rough approximation of a second-order side-channel attack, we refer the interested reader to Mangard et al. [10] for a more detailed discussion). If an attacker can combine the input x' and the output x' using some side channel they obtain the following:

$$\begin{aligned} x' \oplus x'' &= (x \oplus r_1 \oplus r_2) \oplus (x + r_1 + r_2) \\ &= (x \oplus r_1 \oplus r_2) \oplus ((x \oplus r_1 \oplus c_1) \oplus r_2 \oplus c_2) \\ &= c_1 \oplus c_2, \end{aligned}$$

where c_1 and c_2 represent the carries produced in the additions $x + r_1$ and $(x + r_1) + r_2$, respectively, as an XOR difference. That is, $c_i = (x + r_i) \oplus x \oplus r_i$ for $i \in \{1, 2\}$. We note that c_1 and c_2 are dependent on x and could be used to conduct a side-channel attack.

To avoid this source of higher-order leakage, the output of the mask conversion needs to be masked with values that are independent of the input Boolean masks. This can be achieved through re-freshing the masks during the conversion, either once or periodically, as required [6].

In the following, we describe some of the methods for mask conversion that have been presented in the literature.

2.1 Goubin's Method

Goubin proposed an efficient method of converting a Boolean mask to an arithmetic mask at CHES 2001 [5]. His method requires a constant number of instructions, is resistant to first-order side-channel analysis and, at the time of writing, remains the most efficient algorithm known.

The essential observation of Goubin was that the function

$$\Phi_{\mathbb{Z}}(a, b) : \mathbb{Z}^2 \longrightarrow \mathbb{Z} : a, b \longmapsto (a \oplus b) + b \tag{1}$$

is affine over $\mathbb{F}_2$, from which it follows that $(\Phi(a, b) \oplus \Phi(a, 0))$ is *linear* for any value of b. Trivially, we note the same function is valid in the field $(\mathbb{Z}_{2^k}, \oplus, +)$, for any $k \in \mathbb{Z}_{\geq 0}$, and in the remainder of this paper we shall consider the function:

$$\begin{aligned} \Phi(a, b) \; : \; (\mathbb{Z}_{2^k}, \oplus, +)^2 &\longrightarrow (\mathbb{Z}_{2^k}, \oplus, +) \\ a, b &\longmapsto (a \oplus b) + b \end{aligned} \tag{2}$$

for some $k \in \mathbb{Z}_{\geq 0}$.

Taking the notation from Definition 1, for some arbitrary k in $\mathbb{Z}_{\geq 0}$, the above allows one to mask the computation of $\Phi(x', r) = (x' \oplus r) + r$ with an additional random value $\gamma \in \mathbb{Z}_{2^k}$. We recall $x, r \in (\mathbb{Z}_{2^k}, \oplus, +)$ and $x' = x \oplus r$. Then,

$$\Phi(x', \gamma \oplus r) = (x' \oplus (\gamma \oplus r)) + (\gamma \oplus r)\,, \tag{3}$$

which can be followed by an unmasking step using

$$\Phi(x', \gamma) = (x' \oplus \gamma) + \gamma\,. \tag{4}$$

Hence, a secure Boolean-to-arithmetic mask conversion can be performed using the following relationship:

$$\begin{aligned} x'' &= x' \oplus \Phi(x', \gamma) \oplus \Phi(x', \gamma \oplus r) \\ &= x' \oplus [(x' \oplus \gamma) + \gamma] \oplus [(x' \oplus (\gamma \oplus r)) + (\gamma \oplus r)] \end{aligned} \tag{5}$$

where, following the notation in Definition 1, $s = r$, i.e., $x'' = x + r$. One can implement this conversion using 7 instructions (2 additions and 5 XOR operations), as described by Goubin, and is recalled in Algorithm 1.

Goubin then proceeds to give a proof of the following:

Algorithm 1: First-order Secure Boolean-to-Arithmetic Masking

Input: $x' = x \oplus r$, the mask r, a random integer γ, where $x, r, \gamma \in (\mathbb{Z}_{2^k}, \oplus, +)$

Output: $x'' = x + r$

1 $t \leftarrow x' \oplus \gamma$
2 $t \leftarrow t + \gamma$
3 $t \leftarrow t \oplus x'$
4 $\gamma \leftarrow \gamma \oplus r$
5 $z \leftarrow x' \oplus \gamma$
6 $z \leftarrow z + \gamma$
7 $z \leftarrow z \oplus t$

return z

Lemma 1. *An implementation of Algorithm 1 is resistant to first-order side-channel analysis.*

Proof. From Algorithm 1, we can obtain the list of intermediate values $V_0, \ldots, V_6$ that appear during the computation of (5):

$$V_0 = \gamma \qquad V_4 = [(x' \oplus \gamma) + \gamma] \oplus x'$$
$$V_1 = \gamma \oplus r \qquad V_5 = x' \oplus \gamma \oplus r$$
$$V_2 = x' \oplus \gamma \qquad V_6 = (x' \oplus \gamma \oplus r) + (\gamma \oplus r)$$
$$V_3 = (x' \oplus \gamma) + \gamma$$

If we suppose that γ is uniformly distributed on $\mathbb{Z}_{2^k}$, for some arbitrary $k \in \mathbb{Z}_{\geq 0}$, it is easy to see that:

- the values V_0, V_1, V_2, and V_5 are uniformly distributed on $\mathbb{Z}_{2^k}$.
- the distributions of V_3, V_4, and V_6 are dependent on x' but not on r. □

We note that this proof holds in the field $(\mathbb{Z}_{2^k}, \oplus, +)$, for any $k \in \mathbb{Z}_{\geq 0}$, but not in $\mathbb{Z}$ since the carry produced by the most significant bits of x combined with the arithmetic mask will depend on x.

2.2 Recursive Methods

One can also convert a Boolean masked value into an arithmetically masked value using an *addition* operation, which generates the required carries that can then be applied to the Boolean masked input value bit-by-bit. The first application was proposed by Goubin [5] as a means of converting an arithmetic mask to a Boolean mask (a topic beyond the scope of this paper), and a similar technique was described by Golić in 2007 who proposed using the same method for Boolean-to-arithmetic mask conversion in hardware [11]. Both conversion methods have a complexity of $\mathcal{O}(n)$ with regard to the bit length of the inputs because all n bits of the input word are processed individually.

Another hardware-oriented design was proposed by Schneider et al. [7], who presented a conversion method based on a Carry Look-ahead Adder (CLA) structure which reduces the complexity to $\mathcal{O}(\log n)$. They adopted a threshold implementation [12,13] approach to avoid first and second-order side-channel leakage.

Recursive software implementations were proposed by, for example, Karroumi et al. They described a method adding two Boolean masked values in $\mathcal{O}(n)$ time [14]. Coron et al. [15] were the first to propose the use of Carry Look-ahead Adders in software, thus reducing the complexity to $\mathcal{O}(\log n)$. Both works made use of masked AND operations, as defined by Trichina [16] and Ishai et al. [17], respectively.

2.3 Higher-Order Boolean-to-Arithmetic Masking

Coron et al. [6] proposed a method for higher-order Boolean-to-arithmetic mask conversion (see Definition 2) at CHES 2014. Their algorithm calculates carries *recursively* and is built on masked AND and XOR operations that are resistant to higher-order side-channel analysis. Using these secure operations, one can construct an adder resistant to higher-order side-channel analysis with which one can also convert an arithmetic mask to a Boolean mask (the latter topic being beyond the scope of this paper). The authors reported that their fastest h-th order Boolean-to-arithmetic mask conversion has a minimum time complexity of $\mathcal{O}((2\,h+1)^2 n)$, with regard to the bit length of the inputs n.

The first look-up table-based conversion algorithm that resists second-order attacks was proposed by Vadnala and Großschädl in 2013 [8], where, to achieve the desired level of resistance, the algorithm adopts the generic second-order secure S-box implementation of Rivain et al. [18]. Using this method, following the notation in Definition 2, one computes $x_i + r$ for fixed r, where $x_i \in \{0, \ldots, 2^k\}$, and then chooses the correct masked output from all the possible values generated. However, a table with 2^k entries is required which is problematic if k is not small.

An improved version was proposed by Vadnala and Großschädl in 2015 [9], where an input k-bit word would be split into p words with smaller bit widths of $\ell \leq 8$ bits. The conversion is then done on each word individually, and the results combined. Their final solution has a time complexity of $\mathcal{O}(2^{\ell+2}p)$ and a memory requirement of $\mathcal{O}(2^{\ell+2}(\ell+2))$.

3 Constant-Time Second-Order Boolean-to-Arithmetic Mask Conversion

In this section, we present a novel method to perform second-order secure Boolean-to-arithmetic mask conversion whose time complexity is independent of the input-word size. Following the notation in Definition 2, we consider a Boolean masked input $x' = x \oplus r_1 \oplus r_2$, where $x, r_1, r_2 \in (\mathbb{Z}_{2^k}, \oplus, +)$, and an arithmetically masked output $x'' = x + s_1 + s_2$, where $s_1, s_2 \in (\mathbb{Z}_{2^k}, \oplus, +)$.

3.1 Definitions

We recall (2), defined over the field $(\mathbb{Z}_{2^k}, \oplus, +)$, for any $k \in \mathbb{Z}_{\geq 0}$

$$\begin{aligned} \Phi(a,b) \,:\, (\mathbb{Z}_{2^k}, \oplus, +)^2 &\longrightarrow (\mathbb{Z}_{2^k}, \oplus, +) \\ a, b &\longmapsto (a \oplus b) + b \end{aligned} \tag{6}$$

for any $k \in \mathbb{Z}_{\geq 0}$. We shall also use the function

$$\begin{aligned} \bar{\Phi}(a,b) \,:\, (\mathbb{Z}_{2^k}, \oplus, +)^2 &\longrightarrow (\mathbb{Z}_{2^k}, \oplus, +) \\ a, b &\longmapsto (a \oplus b) - b \end{aligned} \tag{7}$$

for any $k \in \mathbb{Z}_{\geq 0}$. While subtraction is not a field operation, we shall use it as a convenient way of expressing the addition with the additive inverse of an operand. Similar to Φ, Goubin notes that

$$x - r = x' \oplus \bar{\Phi}(x', \gamma) \oplus \bar{\Phi}(x', \gamma \oplus r)\,, \tag{8}$$

using the notation in Definition 1, and that $\bar{\Phi}$ is also affine over $\mathbb{F}_2$ [5].

3.2 The Algorithm

Our conversion method consists of three steps.

1. We compute $x + (r_1 \oplus r_2 \oplus \alpha)$ for some random $\alpha \in \mathbb{Z}_{2^k}$.
2. We compute $s_1 - (r_1 \oplus r_2 \oplus \alpha)$ for some random $s_1 \in \mathbb{Z}_{2^k}$.
3. Add the results of Steps 1 and 2 to s_2, a random value taken from $\mathbb{Z}_{2^k}$, resulting in $x + s_1 + s_2$.

We describe these steps in detail below.

Step 1: We consider Goubin's solution to the first-order Boolean-to-arithmetic mask conversion (5),

$$x + r = (x \oplus r) \oplus \Phi(x \oplus r, \gamma) \oplus \Phi(x \oplus r, \gamma \oplus r)\,.$$

Let $r = r_1 \oplus r_2$ and $\gamma = \gamma_1 \oplus \gamma_2$, where $r_1, r_2, \gamma_1, \gamma_2 \in \mathbb{Z}_{2^k}$, then

$$\begin{aligned} x + (r_1 \oplus r_2) = (x \oplus r_1 \oplus r_2) &\oplus \Phi(x \oplus r_1 \oplus r_2, \gamma_1 \oplus \gamma_2) \\ &\oplus \Phi(x \oplus r_1 \oplus r_2, \gamma_1 \oplus \gamma_2 \oplus r_1 \oplus r_2)\,, \end{aligned} \tag{9}$$

or, more succinctly, using the notation from Definition 2,

$$x + (r_1 \oplus r_2) = x' \oplus \Phi(x', \gamma_1 \oplus \gamma_2) \oplus \Phi(x', \gamma_1 \oplus \gamma_2 \oplus r_1 \oplus r_2)\,. \tag{10}$$

Given that Φ is affine over $\mathbb{F}_2$, we can split the first Φ operation giving,

$$x + (r_1 \oplus r_2) = \Phi(x', \gamma_1) \oplus \Phi(x', \gamma_2) \oplus \Phi(x', \gamma_1 \oplus \gamma_2 \oplus r_1 \oplus r_2)\,. \tag{11}$$

If one were to compute $x+(r_1 \oplus r_2)$ using the above, a second-order side-channel attack would be possible for same reason that we require the input and output mask to be different. That is, the combined leakage of the input x' and $x+(r_1 \oplus r_2)$ will depend on x (see Section 2).

To overcome this problem, we apply an extra Boolean mask, $\alpha \in \mathbb{Z}_{2^k}$, to x' as follows:

$$(x \oplus \alpha) + (r_1 \oplus r_2) = \Phi(x' \oplus \alpha, \gamma_1) \oplus \Phi(x' \oplus \alpha, \gamma_2) \oplus \Phi(x' \oplus \alpha, \gamma_1 \oplus \gamma_2 \oplus r_1 \oplus r_2)\,. \tag{12}$$

However, $(x \oplus \alpha) + (r_1 \oplus r_2)$ is not useful but can be modified given that Φ is affine over $\mathbb{F}_2$, resulting in

$$\begin{aligned} x + (r_1 \oplus r_2 \oplus \alpha) = &\Phi(x' \oplus \alpha, \gamma_1) \oplus \Phi(x' \oplus \alpha, \gamma_2) \\ &\oplus \Phi(x' \oplus \alpha, \gamma_1 \oplus \gamma_2 \oplus r_1 \oplus r_2 \oplus \alpha)\,, \end{aligned} \tag{13}$$

which will produce a result that prevents second-order leakage.

The order that (13) is computed is important to avoid combining masks that would allow a second-order side-channel attack. However, this is quite straightforward and will not be detailed here.

Step 2: The second step is another Boolean-to-arithmetic mask conversion to securely compute $s_1 - (r_1 \oplus r_2 \oplus \alpha)$, where s_1 represents one of the two output masks. For this purpose, one can use the first-order secure Boolean-to-arithmetic mask conversion defined in (5), where we define $s'_1 = s_1 \oplus (r_1 \oplus r_2 \oplus \alpha)$ as the Boolean masked input and $s''_1 = s_1 - (r_1 \oplus r_2 \oplus \alpha)$ as the arithmetically masked output of the following conversion. Then, given (5), we have

$$s''_1 = s'_1 \oplus \bar{\Phi}(s'_1, \delta) \oplus \bar{\Phi}(s'_1, \delta \oplus r_1 \oplus r_2 \oplus \alpha)\,, \tag{14}$$

where δ is a random value taken from $\mathbb{Z}_{2^k}$. If we let $\delta = r_1$, then

$$s''_1 = s'_1 \oplus \bar{\Phi}(s'_1, r_1) \oplus \bar{\Phi}(s'_1, r_2 \oplus \alpha)\,, \tag{15}$$

and, given that $\bar{\Phi}$ is affine over $\mathbb{F}_2$, this can be rewritten as

$$s_1 - (r_1 \oplus r_2 \oplus \alpha) = \bar{\Phi}(s'_1, r_1) \oplus \bar{\Phi}(s'_1, r_2) \oplus \bar{\Phi}(s'_1, \alpha)\,. \tag{16}$$

Equation (15) requires a total of 7 XORs and 2 additions, whereas Equation (16) requires 5 XORs and 3 additions. Thus, the first equation might be attractive for hardware implementations in cases where additions are more expensive than XOR operations.

Step 3: Given (13), (15) and (16), one can generate another random $s_2 \in \mathbb{Z}_{2^k}$ and the desired arithmetically masked value x'' can be computed as follows:

$$\begin{aligned} x'' &= (s_2 + (x + (r_1 \oplus r_2 \oplus \alpha))) + (s_1 - (r_1 \oplus r_2 \oplus \alpha)) \\ &= x + s_1 + s_2\,. \end{aligned}$$

Algorithm 2: Second-order Secure Boolean-to-Arithmetic Masking.

Input: $x' = x \oplus r_1 \oplus r_2$ with $x, r_1, r_2 \in \mathbb{Z}_{2^k}$ and random numbers $\gamma_1, \gamma_2, \alpha, s_1, s_2 \in \mathbb{Z}_{2^k}$ for some $k \in \mathbb{Z}_{\geq 0}$

Output: $x'' = x + s_1 + s_2$

1 $z \leftarrow \gamma_1 \oplus r_1$
2 $z \leftarrow z \oplus \gamma_2$
3 $z \leftarrow z \oplus r_2$
4 $u \leftarrow x' \oplus z$
5 $z \leftarrow z \oplus \alpha$
6 $u \leftarrow u + z$
7 $v \leftarrow x' \oplus \gamma_1$
8 $v \leftarrow v \oplus \alpha$
9 $v \leftarrow v + \gamma_1$
10 $w \leftarrow x' \oplus \gamma_2$
11 $w \leftarrow w \oplus \alpha$
12 $w \leftarrow w + \gamma_2$
13 $z \leftarrow u \oplus v$
14 $z \leftarrow z \oplus w$
15 $z \leftarrow z + s_1$
16 $w \leftarrow \alpha \oplus r_2$
17 $u \leftarrow s_2 \oplus r_1$
18 $u \leftarrow u - w$
19 $w \leftarrow w \oplus s_2$
20 $v \leftarrow w \oplus r_1$
21 $w \leftarrow w - r_1$
22 $u \leftarrow u \oplus v$
23 $u \leftarrow u \oplus w$
24 $z \leftarrow z + u$
return z

3.3 Implementation Details

Algorithm 2 shows the second-order secure Boolean-to-arithmetic mask conversion described above, which requires 24 instructions.

We note that the combination of random values used as masks often involves combinations of values that are uniformly and non-uniformly distributed over $\mathbb{Z}_{2^k}$. The latter caused by the use of Boolean and arithmetic operations to make the conversion. For example, Line 17 in Algorithm 2 sets t to $(\alpha \oplus s_1) - s_1$, where α is defined as uniformly distributed over $\mathbb{Z}_{2^k}$, whereas the combination of the two operations involving s_1 will mean that the effect of s_1 on t is not uniform. This is advantageous to an attacker since combining a point where t is manipulated with a point where a combination of x and α, or a combination of x, α and s_1, would both provide an effective second-order side-channel attack.

Lemma 2. *An implementation of Algorithm 2 is resistant to second-order side-channel analysis.*

Proof. From Algorithm 2, we can obtain the list of intermediate values $V_0, \ldots, V_{31}$ that appear during the computation, including all inputs and the output:

$V_0 = x' = x \oplus r_1 \oplus r_2$
$V_1 = x'' = x + s_1 + s_2$
$V_2 = r_1$
$V_3 = r_2$
$V_4 = s_1$
$V_5 = s_2$
$V_6 = \gamma_1$
$V_7 = \gamma_2$
$V_8 = \alpha$
$V_9 = r_1 \oplus \gamma_1$
$V_{10} = r_1 \oplus \gamma_1 \oplus \gamma_2$
$V_{11} = r_1 \oplus r_2 \oplus \gamma_1 \oplus \gamma_2$
$V_{12} = r_1 \oplus r_2 \oplus \gamma_1 \oplus \gamma_2 \oplus \alpha$
$V_{13} = x' \oplus \gamma_1$
$V_{14} = x' \oplus \gamma_1 \oplus \alpha$
$V_{15} = (x' \oplus \gamma_1 \oplus \alpha) + \gamma_1$
$V_{16} = x' \oplus \gamma_2$
$V_{17} = x' \oplus \gamma_2 \oplus \alpha$
$V_{18} = (x' \oplus \gamma_2 \oplus \alpha) + \gamma_2$
$V_{19} = x \oplus \gamma_1 \oplus \gamma_2$
$V_{20} = (x \oplus \gamma_1 \oplus \gamma_2) + (r_1 \oplus r_2 \oplus \gamma_1 \oplus \gamma_2 \oplus \alpha)$
$V_{21} = [(x' \oplus \gamma_1 \oplus \alpha) + \gamma_1] \oplus [(x' \oplus \gamma_2 \oplus \alpha) + \gamma_2]$
$V_{22} = x + (r_1 \oplus r_2 \oplus \alpha)$
$V_{23} = x + (r_1 \oplus r_2 \oplus \alpha) + s_2$

$V_{24} = \alpha \oplus s_1$
$V_{25} = (\alpha \oplus s_1) - s_1$
$V_{26} = (\alpha \oplus r_1)$
$V_{27} = (\alpha \oplus r_1) - r_1$
$V_{28} = (\alpha \oplus r_2)$
$V_{29} = (\alpha \oplus r_2) - r_2$
$V_{30} = [(\alpha \oplus s_1) - s_1] \oplus [(\alpha \oplus r_1) - r_1]$
$V_{31} = s_1 - (r_1 \oplus r_2 \oplus \alpha)$

We shall assume that the random values input to the algorithm, $\gamma_1, \gamma_2, s_1, s_2$, and α, are uniformly distributed over $\mathbb{Z}_{2^k}$, for some arbitrary $k \in \mathbb{Z}_{\geq 0}$. We shall consider an attacker able to combine two intermediate states. One of these states will have to contain x where the variables that act to mask x are the variables that contribute an odd number of times. The variables that contribute an even number of times will, at best, introduce a mask with a non-uniform distribution and not contribute to protecting x. For example, consider V_{15} where r_1, r_2 and α occur an odd number of times and will mask x, but γ_1 occurs an even number of times and will not prevent an attack. An attack is successful if one of the following conditions is met for the second intermediate state.

1. The combination of all the distinct variables combined in one intermediate state are equal to the variables used to masks x, i.e., a non-uniformly distributed variable can be used to reveal a secret but not protect it.
2. The combination of all the variables contributing to an intermediate state with an odd frequency are equal to the variables used to masks x, i.e., the uniformly distributed variables can still be used to reveal a secret even when combined with a non-uniformly distributed variable.

We note an attacker could choose the empty set for the second state, which includes the possibility of a first-order side channel attack.

We list the intermediate states given above that do not contain x, where we list two sets: the first is the number of distinct variables contributing to the intermediate state and the second is the number of variables that contribute with an odd frequency.

$V_2 : \{r_1\}, \{r_1\}$
$V_3 : \{r_2\}, \{r_2\}$
$V_4 : \{s_1\}, \{s_1\}$
$V_5 : \{s_2\}, \{s_2\}$
$V_6 : \{\gamma_1\}, \{\gamma_1\}$
$V_7 : \{\gamma_2\}, \{\gamma_2\}$
$V_8 : \{\alpha\}, \{\alpha\}$
$V_9 : \{r_1, \gamma_1\}, \{r_1, \gamma_1\}$
$V_{10} : \{r_1, \gamma_1, \gamma_2\}, \{r_1, \gamma_1, \gamma_2\}$
$V_{11} : \{r_1, r_2, \gamma_1, \gamma_2\}, \{r_1, r_2, \gamma_1, \gamma_2\}$
$V_{12} : \{r_1, r_2, \gamma_1, \gamma_2, \alpha\}, \{r_1, r_2, \gamma_1, \gamma_2, \alpha\}$
$V_{24} : \{\alpha, s_1\}, \{\alpha, s_1\}$
$V_{25} : \{\alpha, s_1\}, \{\alpha\}$
$V_{26} : \{\alpha, r_1\}, \{\alpha, r_1\}$
$V_{27} : \{\alpha, r_1\}, \{\alpha\}$
$V_{28} : \{\alpha, r_2\}, \{\alpha, r_2\}$
$V_{29} : \{\alpha, r_2\}, \{\alpha\}$
$V_{30} : \{\alpha, s_1, r_1\}, \{\varnothing\}$
$V_{31} : \{s_1, r_1, r_2, \alpha\},, \{s_1, r_1, r_2, \alpha\}$

We also list the same information for the intermediate states that include x.

$V_0 : \{r_1, r_2\}, \{r_1, r_2\}$
$V_1 : \{s_1, s_2\}, \{s_1, s_2\}$
$V_{13} : \{r_1, r_2, \gamma_1\}, \{r_1, r_2, \gamma_1\}$
$V_{14} : \{r_1, r_2, \gamma_1, \alpha\}, \{r_1, r_2, \gamma_1, \alpha\}$
$V_{15} : \{r_1, r_2, \gamma_1, \alpha\}, \{r_1, r_2, \alpha\}$
$V_{16} : \{r_1, r_2, \gamma_2\}, \{r_1, r_2, \gamma_2\}$
$V_{17} : \{r_1, r_2, \gamma_2, \alpha\}, \{r_1, r_2, \gamma_2, \alpha\}$
$V_{18} : \{r_1, r_2, \gamma_2, \alpha\}, \{r_1, r_2, \alpha\}$
$V_{19} : \{\gamma_1, \gamma_2\}, \{\gamma_1, \gamma_2\}$
$V_{20} : \{\gamma_1, \gamma_2, r_1, r_2, \gamma_1, \gamma_2, \alpha\}, \{r_1, r_2, \alpha\}$
$V_{22} : \{r_1, r_2, \alpha\}, \{r_1, r_2, \alpha\}$
$V_{23} : \{r_1, r_2, \alpha\}, \{r_1, \alpha\}$

We have a special case with V_{21}, where the contribution from x has a non-uniform distribution protected by other values with non-uniform distributions. Hence, we define the sets associated with V_{21} as

$$V_{21} : \{r_1, r_2, \gamma_1, \alpha\}, \{\varnothing\}$$

where we note that the empty set used in the second set does not imply an attack is trivial since x is not uniformly distributed.

For each of $V_1, V_{13}, V_{14}, V_{15}, V_{16}, V_{17}, V_{18}, V_{19}, V_{20}, V_{22}, V_{23}$ affected by x, we consider the uniformly distributed variables affecting each intermediate state (the second listed set) and V_{21} where we consider the first set. There does not exist another set with the same combination of variables that would allow an instance of x to be unmasked permitting a side-channel attack.

Hence, Algorithm 2 is resistant to first and second-order side-channel analysis. □

4 Higher-Order Boolean-to-Arithmetic Masking

To generalize the algorithm described in Section 3, we consider an n-th order Boolean masking scheme, for $n > 2$, that masks the secret value x with random masks $r_1, \ldots, r_n$. That is, we wish to take $x' = x \oplus \bigoplus_{i=1}^{n} r_i$ and compute $x'' = x + \sum_{i=1}^{n} s_i$ without allowing any n-th order leakage to occur (see Definition 2). As above, we shall use the functions Φ and $\bar{\Phi}$, as defined in Section 3.1.

4.1 The Algorithm

Our conversion method consists of three steps.

1. We compute $(x + (\alpha \oplus \bigoplus_{i=1}^{n} r_i)) + \sum_{i=1}^{n-2} s_i$ for some random values $\alpha, s_i \in \mathbb{Z}_{2^k}$ for $i \in \{1, \ldots, n-2\}$.
2. We compute $\left(\bigoplus_{i=1}^{n-1} \kappa_i\right) - (\alpha \oplus \bigoplus_{i=1}^{n} r_i)$ for some random values $\kappa_i \in \mathbb{Z}_{2^k}$ for $i \in \{1, \ldots, n-1\}$.
3. We combine the results of Steps 1 and 2, and a further Boolean-to-arithmetic mask conversion, resulting in $x + \sum_{i=1}^{n} s_i$ for some random values $s_i \in \mathbb{Z}_{2^k}$ for $i \in \{1, \ldots, n\}$.

We describe these steps in detail below.

Step 1: We consider Goubin's solution to the first-order Boolean-to-arithmetic mask conversion (5).

$$x + r = (x \oplus r) \oplus \Phi(x \oplus r, \gamma) \oplus \Phi(x \oplus r, \gamma \oplus r)\,.$$

Let $r = r_1 \oplus \ldots \oplus r_n$ and $\gamma = \gamma_1 \oplus \ldots \oplus \gamma_n$, where $r_1, \ldots, r_n, \gamma_1, \ldots, \gamma_n \in \mathbb{Z}_{2^k}$, then following the reasoning given in Section 3.2, we can state

$$x + \bigoplus_{i=1}^{n} r_i = x' \oplus \Phi\left(x', \bigoplus_{i=1}^{n} \gamma_i\right) \oplus \Phi\left(x', \bigoplus_{i=1}^{n} \gamma_i \oplus r_i\right)\,. \tag{17}$$

Given that Φ is affine over $\mathbb{F}_2$, we can split the first Φ operation giving,

$$x + \bigoplus_{i=1}^{n} r_i = ((n \wedge 1)\, x') \oplus \left(\bigoplus_{i=1}^{n} \Phi(x', \gamma_i) \right) \oplus \Phi \left(x', \bigoplus_{i=1}^{n} \gamma_i \oplus r_i \right) . \tag{18}$$

where $\wedge$ is a logical-AND operation. That is, we require an XOR with x' only when n is odd.

To prevent second-order leakage caused by the combination of the input x' and the output of (18), we apply an extra Boolean mask, $\alpha \in \mathbb{Z}_{2^k}$, following the reasoning given in Section 3, i.e.,

$$\begin{aligned} x + \left(\alpha \oplus \bigoplus_{i=1}^{n} r_i \right) = ((n \wedge 1)\, (x' \oplus \alpha)) \oplus \left(\bigoplus_{i=1}^{n} \Phi(x' \oplus \alpha, \gamma_i) \right) \\ \oplus \Phi \left(x' \oplus \alpha, \alpha \oplus \bigoplus_{i=1}^{n} \gamma_i \oplus r_i \right) , \end{aligned} \tag{19}$$

where we compute $\Phi(x' \oplus \alpha, \gamma_i)$, for $i \in \{1, \ldots, n\}$, as

$$x', \alpha, \gamma_i \longmapsto ((x' \oplus \gamma_i) \oplus \alpha) + \gamma_i$$

to avoid any second-order leakage caused by combining $(x' \oplus \alpha)$ with the output of (19).

However, the computation would still cause a higher-order leak, i.e., when x', α, and (19) get combined. Thus, we are required to add extra masks to prevent this leakage, and we use $(n-2)$ masks μ_i, for $i \in \{1, \ldots, n-2\}$, as follows:

$$\begin{aligned} \left(x + \left(\alpha \oplus \bigoplus_{i=1}^{n} r_i \right) \right) \oplus \bigoplus_{i=1}^{n-2} \mu_i = ((n \wedge 1)\, (x' \oplus \alpha)) \oplus \left(\bigoplus_{i=1}^{n-2} \Phi(x' \oplus \alpha, \gamma_i) \oplus \mu_i \right) \\ \oplus \Phi(x' \oplus \alpha, \gamma_{n-1}) \oplus \Phi(x' \oplus \alpha, \gamma_n)) \oplus \Phi \left(x' \oplus \alpha, \alpha \oplus \bigoplus_{i=1}^{n} \gamma_i \oplus r_i \right) . \end{aligned} \tag{20}$$

The result is then passed through a function that will perform a Boolean-to-arithmetic mask conversion resistant to $(n-2)$-th order side-channel attacks to produce

$$\left(x + \left(\alpha \oplus \bigoplus_{i=1}^{n} r_i \right) \right) + \sum_{i=1}^{n-2} s_i \, , \tag{21}$$

where s_i, for $i \in \{1, \ldots, n-2\}$, are the output masks required in the output of Step 3. Note that in case of a third-order Boolean-to-arithmetic mask conversion, one can directly use one of the output masks, i.e., $s_1 = \mu_1$, because the first-order Boolean-to-arithmetic mask conversion does not require that the input and output masks are different (see Section 2).

We note that using a function that will perform a Boolean-to-arithmetic mask conversion resistant to an $(n-2)$-th order side-channel attack will call a function that is resistant to an $(n-4)$-th order side-channel attack etc. That is, until a first or second-order resistant algorithm is required (see Sections 2.1 and 3, respectively).

Step 2: In the second step, we can perform another Boolean-to-arithmetic mask conversion to securely compute $\left(\bigoplus_{i=1}^{n-1}\kappa_i\right) - (\alpha \oplus \bigoplus_{i=1}^{n} r_i)$, for some random values $\kappa_i \in \mathbb{Z}_{2^k}$, for $i \in \{1,\ldots,n-1\}$. In which we view the combination of any elements of $\{\kappa_1,\ldots,\kappa_{n-1}\}$ as secret and $(r_1 \oplus \ldots \oplus r_n \oplus \alpha)$ as the mask, where any combination of any elements of $\{r_1 \oplus \ldots \oplus r_n \oplus \alpha\}$ is also secret. We can use an $(n-1)$-th-order version of Step 1, as follows

$$\left(\bigoplus_{i=1}^{n-1}\kappa_i\right) - \left(\alpha \oplus \bigoplus_{i=1}^{n} r_i\right) = ((\neg n \wedge 1)\,\beta) \oplus \bigoplus_{i=1}^{n-1}\left(\bar{\Phi}\left(\beta,\delta_i\right)\right) \oplus \bar{\Phi}\left(\beta, \alpha \oplus r_n \oplus \bigoplus_{i=1}^{n-1}\delta_i \oplus r_i\right) \tag{22}$$

where $\beta = \left(\alpha \oplus r_n \oplus \bigoplus_{j=1}^{n-1}\kappa_j \oplus r_j\right)$ and δ_i are random values taken from $\mathbb{Z}_{2^k}$ for $i \in \{1,\ldots,n-1\}$. We note that the order in which operands are treated is particularly important. For example, α and r_n cannot be combined but need to be XORed separately with the result of the XOR sum they are combined with. Likewise, the terms of the XOR sums need to be computed separately, i.e., $\bigoplus_{i=1}^{n-1}\delta_i \oplus r_i = (\delta_1 \oplus r_1) \oplus (\delta_2 \oplus r_2) \oplus \ldots \oplus (\delta_{n-1} \oplus r_{n-1})$.

Step 3: Summing (21) with the left-hand sides of (22) and the remaining output masks s_i with $i \in \{n-1,\ldots,n\}$, we have

$$\left(x + \left(\alpha \oplus \bigoplus_{i=1}^{n} r_i\right) + \sum_{i=1}^{n-2} s_i\right) + \left(\bigoplus_{i=1}^{n-1}\kappa_i - \alpha \oplus \bigoplus_{i=1}^{n} r_i\right) + \sum_{i=n-1}^{n} s_i = x + \sum_{i=1}^{n} s_i + \bigoplus_{i=1}^{n-1}\kappa_i. \tag{23}$$

One can then generate other random values $\ell_1,\ldots,\ell_{n-2} \in \mathbb{Z}_{2^k}$ and compute

$$\kappa_{n-1} \oplus \bigoplus_{i=1}^{n-2}\ell_i \oplus \kappa_i\,, \tag{24}$$

followed by an $(n-2)$-th order Boolean-to-arithmetic mask conversion where $\bigoplus_{i=1}^{n-1}\kappa_i$ is the secret and $\bigoplus_{i=1}^{n-2}\ell_i$ are the Boolean masks to produce $\sum_{i=1}^{n-2}\ell_i + \bigoplus_{i=1}^{n-1}\kappa_i$. Note that the elements of $\{\kappa_1,\ldots,\kappa_{n-1}\}$ cannot be combined without

causing leakage. Finally, we subtract $\sum_{i=1}^{n-2} \ell_i + \bigoplus_{i=1}^{n-1} \kappa_i$ from (23) and add all elements of $\{1, \ldots, \ell_{n-2}\}$, which gives us the arithmetically masked output, i.e.,

$$\left(x + \sum_{i=1}^{n} s_i + \bigoplus_{i=1}^{n-1} \kappa_i \right) - \left(\sum_{i=1}^{n-2} \ell_i + \bigoplus_{i=1}^{n-1} \kappa_i \right) + \sum_{i=1}^{n-2} \ell_i = x + \sum_{i=1}^{n} s_i \,. \tag{25}$$

Again, we note that the order in which the operations in the above step is computed is important to not combine masks that will produce leakage that could be exploited by a side-channel attack. However, this is quite straightforward and, other than the cases mentioned above, will not be detailed here (see Algorithm 3 for an example of a third-order resistant mask conversion).

Complexity
Each of the steps described above, without the use of Boolean-to-arithmetic mask conversions of a lower order, will have a linear increase in time complexity with regard to the order of the side-channel resistance. That is, have time complexity $\mathcal{O}(n)$. The recursive calls to Boolean-to-arithmetic mask conversions of a lower order will increase the time complexity to $\mathcal{O}(n^2)$.

4.2 Implementation Details

Algorithm 3 shows a third-order secure Boolean-to-arithmetic mask conversion as an example of the method described above, which requires 56 instructions. We give a proof of security in Appendix B.

4.3 Comparison

Table 1 compares the performance of our proposed method with related work. We consider the work of Coron et al. [6] who proposed a higher-order secure Boolean-to-arithmetic algorithm; we do not consider LUT-based methods as they would require a pre-compuation phase and additional memory (see Section 2). We estimated the operation count of all methods by considering all necessary operations excluding the generation of random numbers, loop-instruction overheads, and variable initialization. Appendix A provides more details about the calculation and also compares the requirements on randomness.

Table 1 clearly shows that our solution is faster than Coron et al.'s method for all considered register widths and security orders, often by several orders of magnitude. Our methods also require fewer random values by at least one magnitude as shown in Table 3 (see Appendix A).

5 Conclusions

In this paper, we present Boolean-to-arithmetic mask conversion methods that can be computed in constant time for a masking scheme of a given order. Our proposed methods have a complexity of $\mathcal{O}(n^2)$ with regard to the security order

Algorithm 3: Third-order Secure Boolean-to-Arithmetic Masking.

Input: $x' = x \oplus r_1 \oplus r_2 \oplus r_3$ with $x, r_1, r_2, r_3 \in \mathbb{Z}_{2^k}$ and random numbers $\gamma_1, \gamma_2, \gamma_3, \alpha, \delta_1, \delta_2, \kappa_1, \kappa_2, s_1, s_2, s_3 \in \mathbb{Z}_{2^k}$ for some $k \in \mathbb{Z}_{\geq 0}$

Output: $x'' = x + s_1 + s_2 + s_3$

1	$t \leftarrow \gamma_1 \oplus r_1$	**20**	$z \leftarrow z \oplus v$	**39**	$v \leftarrow v - \delta_2$
2	$t \leftarrow t \oplus \gamma_2$	**21**	$z \leftarrow z \oplus x'$	**40**	$w \leftarrow \kappa_1 \oplus \delta_1$
3	$t \leftarrow t \oplus r_2$	**22**	$z \leftarrow z \oplus w$	**41**	$w \leftarrow w \oplus \delta_2$
4	$t \leftarrow t \oplus \gamma_3$	**23**	$z \leftarrow z \oplus t$	**42**	$w \leftarrow w \oplus \kappa_2$
5	$t \leftarrow t \oplus r_3$	**24**	$t \leftarrow z \oplus r_2$	**43**	$t \leftarrow t \oplus w$
6	$u \leftarrow t \oplus \alpha$	**25**	$t \leftarrow t + r_2$	**44**	$w \leftarrow w - t$
7	$t \leftarrow x' \oplus t$	**26**	$t \leftarrow t \oplus z$	**45**	$t \leftarrow u \oplus v$
8	$t \leftarrow t + u$	**27**	$u \leftarrow r_2 \oplus s_1$	**46**	$t \leftarrow t \oplus w$
9	$u \leftarrow x' \oplus \gamma_1$	**28**	$z \leftarrow z \oplus u$	**47**	$z \leftarrow z + s_2$
10	$u \leftarrow u \oplus \alpha$	**29**	$z \leftarrow z + u$	**48**	$z \leftarrow z + t$
11	$u \leftarrow u + \gamma_1$	**30**	$z \leftarrow z \oplus t$	**49**	$v \leftarrow s_3 \oplus \kappa_2$
12	$v \leftarrow x' \oplus \gamma_2$	**31**	$t \leftarrow \kappa_1 \oplus r_1$	**50**	$w \leftarrow v \oplus \kappa_1$
13	$v \leftarrow v \oplus \alpha$	**32**	$t \leftarrow t \oplus \kappa_2$	**51**	$u \leftarrow v - \kappa_1$
14	$v \leftarrow v + \gamma_2$	**33**	$t \leftarrow t \oplus r_2$	**52**	$u \leftarrow u \oplus w$
15	$w \leftarrow x' \oplus \gamma_3$	**34**	$t \leftarrow t \oplus \alpha$	**53**	$v \leftarrow s_3 \oplus \kappa_1$
16	$w \leftarrow w \oplus \alpha$	**35**	$t \leftarrow t \oplus r_3$	**54**	$v \leftarrow v - \kappa_2$
17	$w \leftarrow w + \gamma_3$	**36**	$u \leftarrow t \oplus \delta_1$	**55**	$w \leftarrow u \oplus v$
18	$z \leftarrow \alpha \oplus u$	**37**	$u \leftarrow u - \delta_1$	**56**	$z \leftarrow z + w$
19	$z \leftarrow z \oplus s_1$	**38**	$v \leftarrow t \oplus \delta_2$		**return** z

and are independent of the input-word size. We present explicit algorithms for a second-order secure mask conversion that requires 24 instructions, i.e., a multiple of 3.4 compared to the instruction count of Goubin's method (7 instructions), and a third-order secure mask conversion that requires 56 instructions, i.e., a multiple of 8 compared to Goubin's method. We also describe a generic conversion method for masking schemes of any higher order. All methods offer a better performance that the state-of-the-art by at least one order of magnitude and also require fewer random values also by at least one order of magnitude.

References

1. Kocher, P., Jaffe, J., Jun, B.: Differential Power Analysis. In Wiener, M.J., ed.: CRYPTO '99. Volume 1666 of LNCS., Springer, Heidelberg (1999) 388–397
2. Agrawal, D., Archambeault, B., Rao, J.R., Rohatgi, P.: The EM Side-channel(s). In Kaliski Jr., B.S., Koç, Ç.K., Paar, C., eds.: CHES 2003. Volume 2523 of LNCS., Springer, Heidelberg (2003) 29–45
3. Gandolfi, K., Mourtel, C., Olivier, F.: Electromagnetic Analysis: Concrete Results. In Koç, C.K., Naccache, D., Paar, C., eds.: CHES 2001. Volume 2162 of LNCS., Springer, Heidelberg (2001) 251–261
4. Quisquater, J.J., Samyde, D.: ElectroMagnetic Analysis (EMA): Measures and Counter-Measures for Smart Cards. In Attali, I., Jensen, T.P., eds.: E-smart 2001. Volume 2140 of LNCS., Springer, Heidelberg (2001) 200–210

Table 1. Operation count for different Boolean-to-arithmetic mask conversion methods up to a security order of 8.

$B \rightarrow A$	Security Order							
Conversion	1	2	3	4	5	6	7	8
Goubin's method	7	-	-	-	-	-	-	-
Coron et al. (8 bits)	-	909	1,369	1,962	2,619	3,372	4,189	5,171
Coron et al. (16 bits)	-	1,781	2,681	3,842	5,131	6,612	8,221	10,155
Coron et al. (32 bits)	-	3,525	5,305	7,602	10,155	13,092	16,285	20,123
Coron et al. (64 bits)	-	7,013	10,553	15,122	20,203	26,052	32,413	40,059
Our proposal	-	24	56	115	197	331	513	763

5. Goubin, L.: A Sound Method for Switching between Boolean and Arithmetic Masking. In Koç, Ç.K., Naccache, D., Paar, C., eds.: CHES 2001. Volume 2162 of LNCS., Springer, Heidelberg (2001) 3–15
6. Coron, J., Großschädl, J., Vadnala, P.K.: Secure Conversion between Boolean and Arithmetic Masking of Any Order. In Batina, L., Robshaw, M., eds.: CHES 2014. Volume 8731 of LNCS., Springer, Heidelberg (2014) 188–205
7. Schneider, T., Moradi, A., Güneysu, T.: Arithmetic Addition over Boolean Masking—Towards First- and Second-Order Resistance in Hardware. In Malkin, T., Kolesnikov, V., Lewko, A.B., Polychronakis, M., eds.: ACNS 2015. Volume 9092 of LNCS., Springer, Heidelberg (2015) 559–578
8. Vadnala, P.K., Großschädl, J.: Algorithms for Switching between Boolean and Arithmetic Masking of Second Order. In Gierlichs, B., Guilley, S., Mukhopadhyay, D., eds.: SPACE 2013. Volume 8204 of LNCS., Springer, Heidelberg (2013) 95–110
9. Vadnala, P.K., Großschädl, J.: Faster Mask Conversion with Lookup Tables. In Mangard, S., Poschmann:, A.Y., eds.: COSADE 2015. Volume 9064 of LNCS., Springer, Heidelberg (2015) 207–221
10. Mangard, S., Oswald, E., Popp, T.: Power Analysis Attacks — Revealing the Secrets of Smart Cards. Springer (2007)
11. Golić, J.D.: Techniques for Random Masking in Hardware. IEEE Transactions on Circuits and Systems **54**(2) (2007) 291–300
12. Nikova, S., Rechberger, C., Rijmen, V.: Threshold Implementations Against Side-Channel Attacks and Glitches. In Ning, P., Qing, S., Li, N., eds.: ICICS 2006. Volume 4307 of LNCS., Springer, Heidelberg (2006) 529–545
13. Nikova, S., Rijmen, V., Schläffer, M.: Secure Hardware Implementation of Non-linear Functions in the Presence of Glitches. Journal of Cryptology **24**(2) (2011) 292–321
14. Karroumi, M., Richard, B., Joye, M.: Addition with Blinded Operands. In Prouff, E., ed.: COSADE 2014. Volume 8622 of LNCS., Springer, Heidelberg (2014) 41–55
15. Coron, J., Großschädl, J., Tibouchi, M., Vadnala, P.K.: Conversion from Arithmetic to Boolean Masking with Logarithmic Complexity. In Leander, G., ed.: FSE 2015. Volume 8731 of LNCS., Springer, Heidelberg (2015) 130–149
16. Trichina, E.: Combinational Logic Design for AES SubByte Transformation on Masked Data. IACR Cryptology ePrint Archive **2003** (2003) 236

17. Ishai, Y., Sahai, A., Wagner, D.: Private Circuits: Securing Hardware against Probing Attacks. In Boneh, D., ed.: CRYPTO 2003. Volume 2729 of LNCS., Springer, Heidelberg (2003) 463–481
18. Rivain, M., Dottax, E., Prouff, E.: Block Ciphers Implementations Provably Secure Against Second Order Side Channel Analysis. In Nyberg, K., ed.: FSE 2008. Volume 5086 of LNCS., Springer, Heidelberg (2008) 127–143

A Complexity Calculation

We estimate the costs for Coron et al.'s higher-order Boolean-to-arithmetic mask conversion method [6] as follows. For a single masked AND (`SecAnd`) operation [6, Section 3] we estimate the number of required instructions to be

$$2 \cdot (n+1) \cdot n + 25,$$

with n being the security order. Furthermore, we estimate the higher-order secure masked addition function (`SecAddGoubin`) as defined in [6, Section 3.2] to be

$$(2 \cdot (n+1) \cdot n + 26 + n) + (k-1) \cdot [2 \cdot (n+1) \cdot n + 27] + (2 \cdot (n+1)),$$

where k represents the bit-width of the operands. The `Expand` function has an estimated complexity of $2 \cdot (n+1)$ and the `FullXor` function requires $2 \cdot n + n$.

Using these estimations, we calculated the total operation count for higher-order Boolean-to-arithmetic mask conversion as defined in [6, Section 5] for register sizes of 8, 16, 32, and 64 bits and provide the results in Table 1.

A.1 Performance Details and Comparison

We provide more performance details on our method in comparison with related work. Table 2 lists the number of required instructions in terms of arithmetic and Boolean operations up to a security order of 8. Note that Coron et al.'s solution [6] does not require arithmetic operations, we therefore refer to the total instruction count given in Table 1.

Table 2. Number of required arithmetic and Boolean operations of our proposed method up to a security order of 8.

$B \rightarrow A$	**Security Order**							
Conversion	1	2	3	4	5	6	7	8
Arithmetic operations	2	8	14	31	46	83	116	193
Boolean operations	5	16	42	84	151	248	397	570

Furthermore, we list the number of required random variables to perform a Boolean-to-arithmetic mask conversion in Table 3. We estimate the number of

random variables according to [6] and, for simplicity, we do not consider optimization techniques such as re-using random inputs or common sub-expression elimination. Furthermore, we do not apply optimization techniques for our proposed method for security order 4 and above (for lower security orders, we give the same number of required random variables from the explicit algorithms proposed in this paper). For completeness, we also list the number of required random variables for the first-order mask conversion method proposed by Goubin [5].

Table 3. Comparison of required number of random variables.

$B \rightarrow A$	**Security Order**							
Conversion	1	2	3	4	5	6	7	8
Goubin's method	1	-	-	-	-	-	-	-
Coron et al. (8 bits)	-	66	127	221	331	465	615	806
Coron et al. (16 bits)	-	122	239	421	635	897	1,191	1,566
Coron et al. (32 bits)	-	234	463	821	1,243	1,761	2,343	3,086
Coron et al. (64 bits)	-	458	911	1,621	2,459	3,489	4,647	6,126
Our proposal	-	5	11	27	44	81	120	199

B Security Proof for Algorithm 3

Lemma 3. *An implementation of Algorithm 3 is resistant to third-order side-channel analysis.*

Proof. From Algorithm 3, we can obtain the list of intermediate values $V_0, \ldots, V_{70}$ that appear during the computation, including all inputs and the output:

$V_0 = x' = x \oplus r_1 \oplus r_2 \oplus r_3$
$V_1 = x'' = x + s_1 + s_2 + s_3$
$V_2 = r_1$
$V_3 = r_2$
$V_4 = r_3$
$V_5 = s_1$
$V_6 = s_2$
$V_7 = s_3$
$V_8 = \gamma_1$
$V_9 = \gamma_2$
$V_{10} = \gamma_3$
$V_{11} = \alpha$
$V_{12} = \delta_1$
$V_{13} = \delta_2$
$V_{14} = \kappa_1$
$V_{15} = \kappa_2$
$V_{16} = \gamma_1 \oplus r_1$
$V_{17} = \gamma_1 \oplus \gamma_2 \oplus r_1$
$V_{18} = \gamma_1 \oplus \gamma_2 \oplus r_1 \oplus r_2$
$V_{19} = \gamma_1 \oplus \gamma_2 \oplus \gamma_3 \oplus r_1 \oplus r_2$
$V_{20} = \gamma_1 \oplus \gamma_2 \oplus \gamma_3 \oplus r_1 \oplus r_2 \oplus r_3$
$V_{21} = \gamma_1 \oplus \gamma_2 \oplus \gamma_3 \oplus r_1 \oplus r_2 \oplus r_3 \oplus \alpha$
$V_{22} = x' \oplus \gamma_1 \oplus \gamma_2 \oplus \gamma_3 \oplus r_1 \oplus r_2 \oplus r_3$
$V_{23} = (x' \oplus \gamma_1 \oplus \gamma_2 \oplus \gamma_3 \oplus r_1 \oplus r_2 \oplus r_3)$
$\quad +(\gamma_1 \oplus \gamma_2 \oplus \gamma_3 \oplus r_1 \oplus r_2 \oplus r_3 \oplus \alpha)$
$V_{24} = x' \oplus \gamma_1$
$V_{25} = x' \oplus \gamma_1 \oplus \alpha$
$V_{26} = (x' \oplus \gamma_1 \oplus \alpha) + \gamma_1$

$$
\begin{aligned}
V_{27} &= x' \oplus \gamma_2 \\
V_{28} &= x' \oplus \gamma_2 \oplus \alpha \\
V_{29} &= (x' \oplus \gamma_2 \oplus \alpha) + \gamma_2 \\
V_{30} &= x' \oplus \gamma_3 \\
V_{31} &= x' \oplus \gamma_3 \oplus \alpha \\
V_{32} &= (x' \oplus \gamma_3 \oplus \alpha) + \gamma_3 \\
V_{33} &= [(x' \oplus \gamma_1 \oplus \alpha) + \gamma_1] \oplus \alpha \\
V_{34} &= [(x' \oplus \gamma_1 \oplus \alpha) + \gamma_1] \oplus \alpha \oplus s_1 \\
V_{35} &= [(x' \oplus \gamma_1 \oplus \alpha) + \gamma_1] \oplus \alpha \oplus s_1 \\
&\quad \oplus [(x' \oplus \gamma_2 \oplus \alpha) + \gamma_2] \\
V_{36} &= [(x' \oplus \gamma_1 \oplus \alpha) + \gamma_1] \oplus \alpha \oplus s_1 \\
&\quad \oplus [(x' \oplus \gamma_2 \oplus \alpha) + \gamma_2] \oplus x' \\
V_{37} &= [(x' \oplus \gamma_1 \oplus \alpha) + \gamma_1] \oplus \alpha \oplus s_1 \\
&\quad \oplus [(x' \oplus \gamma_2 \oplus \alpha) + \gamma_2] \oplus x' \\
&\quad \oplus [(x' \oplus \gamma_3 \oplus \alpha) + \gamma_3] \\
V_{38} &= [x + (r_1 \oplus r_2 \oplus r_3 \oplus \alpha)] \oplus s_1 \\
V_{39} &= [x + (r_1 \oplus r_2 \oplus r_3 \oplus \alpha)] \oplus s_1 \oplus r_2 \\
V_{40} &= ([x + (r_1 \oplus r_2 \oplus r_3 \oplus \alpha)] \oplus s_1 \oplus r_2) + r_2 \\
V_{41} &= [([x + (r_1 \oplus r_2 \oplus r_3 \oplus \alpha)] \oplus s_1 \oplus r_2) \\
&\quad + r_2] \oplus ([x + (r_1 \oplus r_2 \oplus r_3 \oplus \alpha)] \oplus s_1) \\
V_{42} &= r_2 \oplus s_1 \\
V_{43} &= [x + (r_1 \oplus r_2 \oplus r_3 \oplus \alpha)] \oplus r_2 \\
V_{44} &= ([x + (r_1 \oplus r_2 \oplus r_3 \oplus \alpha)] \oplus r_2) + (r_2 \oplus s_1) \\
V_{45} &= x + (r_1 \oplus r_2 \oplus r_3 \oplus \alpha) + s_1 \\
V_{46} &= \kappa_1 \oplus r_1 \\
V_{47} &= \kappa_1 \oplus \kappa_2 \oplus r_1 \\
V_{48} &= \kappa_1 \oplus \kappa_2 \oplus r_1 \oplus r_2 \\
V_{49} &= \kappa_1 \oplus \kappa_2 \oplus r_1 \oplus r_2 \oplus \alpha \\
V_{50} &= \kappa_1 \oplus \kappa_2 \oplus r_1 \oplus r_2 \oplus r_3 \oplus \alpha \\
V_{51} &= \kappa_1 \oplus \kappa_2 \oplus r_1 \oplus r_2 \oplus r_3 \oplus \alpha \oplus \delta_1 \\
V_{52} &= (\kappa_1 \oplus \kappa_2 \oplus r_1 \oplus r_2 \oplus r_3 \oplus \alpha \oplus \delta_1) - \delta_1 \\
V_{53} &= \kappa_1 \oplus \kappa_2 \oplus r_1 \oplus r_2 \oplus r_3 \oplus \alpha \oplus \delta_2 \\
V_{54} &= (\kappa_1 \oplus \kappa_2 \oplus r_1 \oplus r_2 \oplus r_3 \oplus \alpha \oplus \delta_2) - \delta_2 \\
V_{55} &= \kappa_1 \oplus \delta_1 \\
V_{56} &= \kappa_1 \oplus \delta_1 \oplus \delta_2 \\
V_{57} &= \kappa_1 \oplus \kappa_2 \oplus \delta_1 \oplus \delta_2 \\
V_{58} &= \alpha \oplus r_1 \oplus r_2 \oplus r_3 \oplus \delta_1 \oplus \delta_2 \\
V_{59} &= (\kappa_1 \oplus \kappa_2 \oplus \delta_1 \oplus \delta_2) \\
&\quad - (\alpha \oplus r_1 \oplus r_2 \oplus r_3 \oplus \delta_1 \oplus \delta_2) \\
V_{60} &= [(\kappa_1 \oplus \kappa_2 \oplus r_1 \oplus r_2 \oplus r_3 \oplus \alpha \oplus \delta_1) - \delta_1] \\
&\quad \oplus [(\kappa_1 \oplus \kappa_2 \oplus r_1 \oplus r_2 \oplus r_3 \oplus \alpha \oplus \delta_2) - \delta_2] \\
V_{61} &= (\kappa_1 \oplus \kappa_2) + r_1 \oplus r_2 \oplus r_3 \oplus \alpha \\
V_{62} &= x + (r_1 \oplus r_2 \oplus r_3 \oplus \alpha) + s_1 + s_2 \\
V_{63} &= x + s_1 + s_2 + (\kappa_1 \oplus \kappa_2) \\
V_{64} &= s_3 \oplus \kappa_2 \\
V_{65} &= s_3 \oplus \kappa_1 \oplus \kappa_2 \\
V_{66} &= (s_3 \oplus \kappa_2) - \kappa_1 \\
V_{67} &= [(s_3 \oplus \kappa_2) - \kappa_1] \oplus (s_3 \oplus \kappa_1 \oplus \kappa_2) \\
V_{68} &= s_3 \oplus \kappa_1 \\
V_{69} &= (s_3 \oplus \kappa_1) - \kappa_2 \\
V_{70} &= s_3 - (\kappa_1 \oplus \kappa_2)
\end{aligned}
$$

We assume that the inputs $\gamma_1, \gamma_2, \gamma_3, s_1, s_2, s_3, \delta_1, \delta_2, \kappa_1, \kappa_2$, and α, are uniformly distributed over $\mathbb{Z}_{2^k}$, for some arbitrary $k \in \mathbb{Z}_{\geq 0}$. We shall consider an attacker able to combine three intermediate states. One of these states will have to contain x where the variables that act to mask x are the variables that contribute an odd number of times. The variables that contribute an even number of times will, at best, introduce a mask with non-uniform distribution and not contribute to protecting x. An attack is successful if one of the following conditions is met for the other two intermediate states.

1. The combination of all the distinct variables combined in two intermediate states (i.e., the union of the two sets) are equal to the variables used to masks x.
2. The combination of all the variable contributing to two intermediate states with an odd frequency are equal to the variables used to masks x (i.e., the union of the two sets minus the intersection).
3. A combination of one each set of variables where we consider all the distinct variables combined in an intermediate state and another where we consider variables contributing to an intermediate state with an odd frequency, where combined variables are equal to the variables used to masks x.

We note an attacker could choose the empty set for the second and/or third state, which includes the possibility of a first or second-order side channel attack.

We list the intermediate states given above that do not contain x, where we list two sets: the first is the number of distinct variables contributing to the intermediate state and the second is the number of variables that contribute with an odd frequency.

$V_2 : \{r_1\}, \{r_1\}$
$V_3 : \{r_2\}, \{r_2\}$
$V_4 : \{r_3\}, \{r_3\}$
$V_5 : \{s_1\}, \{s_1\}$
$V_6 : \{s_2\}, \{s_2\}$
$V_7 : \{s_3\}, \{s_3\}$
$V_8 : \{\gamma_1\}, \{\gamma_1\}$
$V_9 : \{\gamma_2\}, \{\gamma_2\}$
$V_{10} : \{\gamma_3\}, \{\gamma_3\}$
$V_{11} : \{\alpha\}, \{\alpha\}$
$V_{12} : \{\delta_1\}, \{\delta_1\}$
$V_{13} : \{\delta_2\}, \{\delta_2\}$
$V_{14} : \{\kappa_1\}, \{\kappa_1\}$
$V_{15} : \{\kappa_2\}, \{\kappa_2\}$
$V_{16} : \{\gamma_1, r_1\}, \{\gamma_1, r_1\}$
$V_{17} : \{\gamma_1, \gamma_2, r_1\}, \{\gamma_1, \gamma_2, r_1\}$
$V_{18} : \{\gamma_1, \gamma_2, r_1, r_2\}, \{\gamma_1, \gamma_2, r_1, r_2\}$
$V_{19} : \{\gamma_1, \gamma_2, \gamma_3, r_1, r_2\}, \{\gamma_1, \gamma_2, \gamma_3, r_1, r_2\}$
$V_{20} : \{\gamma_1, \gamma_2, \gamma_3, r_1, r_2, r_3\}, \{\gamma_1, \gamma_2, \gamma_3, r_1, r_2, r_3\}$
$V_{21} : \{\gamma_1, \gamma_2, \gamma_3, r_1, r_2, r_3, \alpha\}, \{\gamma_1, \gamma_2, \gamma_3, r_1, r_2, r_3, \alpha\}$
$V_{42} : \{r_2, s_1\}, \{r_2, s_1\}$
$V_{46} : \{\kappa_1, r_1\}, \{\kappa_1, r_1\}$
$V_{47} : \{\kappa_1, \kappa_2, r_1\}, \{\kappa_1, \kappa_2, r_1\}$
$V_{48} : \{\kappa_1, \kappa_2, r_1, r_2\}, \{\kappa_1, \kappa_2, r_1, r_2\}$
$V_{49} : \{\kappa_1, \kappa_2, r_1, r_2, \alpha\}, \{\kappa_1, \kappa_2, r_1, r_2, \alpha\}$
$V_{50} : \{\kappa_1, \kappa_2, r_1, r_2, r_3, \alpha\}, \{\kappa_1, \kappa_2, r_1, r_2, r_3, \alpha\}$
$V_{51} : \{\kappa_1, \kappa_2, r_1, r_2, r_3, \alpha, \delta_1\}, \{\kappa_1, \kappa_2, r_1, r_2, r_3, \alpha, \delta_1\}$
$V_{52} : \{\kappa_1, \kappa_2, r_1, r_2, r_3, \alpha, \delta_1\}, \{\kappa_1, \kappa_2, r_1, r_2, r_3, \alpha\}$
$V_{53} : \{\kappa_1, \kappa_2, r_1, r_2, r_3, \alpha, \delta_2\}, \{\kappa_1, \kappa_2, r_1, r_2, r_3, \alpha, \delta_2\}$
$V_{54} : \{\kappa_1, \kappa_2, r_1, r_2, r_3, \alpha, \delta_2\}, \{\kappa_1, \kappa_2, r_1, r_2, r_3, \alpha\}$
$V_{55} : \{\kappa_1, \delta_1\}, \{\kappa_1, \delta_1\}$
$V_{56} : \{\kappa_1, \delta_1, \delta_2\}, \{\kappa_1, \delta_1, \delta_2\}$
$V_{57} : \{\kappa_1, \kappa_2, \delta_1, \delta_2\}, \{\kappa_1, \kappa_2, \delta_1, \delta_2\}$
$V_{58} : \{\alpha, r_1, r_2, r_3, \delta_1, \delta_2\}, \{\alpha, r_1, r_2, r_3, \delta_1, \delta_2\}$
$V_{59} : \{\kappa_1, \kappa_2, \delta_1, \delta_2, \alpha, r_1, r_2, r_3\}, \{\kappa_1, \kappa_2, \alpha, r_1, r_2, r_3\}$
$V_{60} : \{\kappa_1, \kappa_2, r_1, r_2, r_3, \alpha, \delta_1, \delta_2\}, \{\varnothing\}$
$V_{61} : \{\kappa_1, \kappa_2, r_1, r_2, r_3, \alpha\}, \{\kappa_1, \kappa_2, r_1, r_2, r_3, \alpha\}$
$V_{64} : \{s_3, \kappa_2\}, \{s_3, \kappa_2\}$
$V_{65} : \{s_3, \kappa_1, \kappa_2\}, \{s_3, \kappa_1, \kappa_2\}$
$V_{66} : \{s_3, \kappa_1, \kappa_2\}, \{s_3, \kappa_1, \kappa_2\}$
$V_{67} : \{s_3, \kappa_1, \kappa_2\}, \{\varnothing\}$
$V_{68} : \{s_3, \kappa_1\}, \{s_3, \kappa_1\}$
$V_{69} : \{s_3, \kappa_1, \kappa_2\}, \{s_3, \kappa_1, \kappa_2\}$
$V_{70} : \{s_3, \kappa_1, \kappa_2\}, \{s_3, \kappa_1, \kappa_2\}$

We also list the same information for the intermediate states that include x.

$V_0 : \{r_1, r_2, r_3\}, \{r_1, r_2, r_3\}$
$V_1 : \{s_1, s_2, s_3\}, \{s_1, s_2, s_3\}$
$V_{22} : \{r_1, r_2, r_3, \gamma_1, \gamma_2, \gamma_3\}, \{\gamma_1, \gamma_2, \gamma_3\}$
$V_{23} : \{r_1, r_2, r_3, \gamma_1, \gamma_2, \gamma_3, \alpha\}, \{r_1, r_2, r_3, \alpha\}$
$V_{24} : \{r_1, r_2, r_3, \gamma_1\}, \{r_1, r_2, r_3, \gamma_1\}$
$V_{25} : \{r_1, r_2, r_3, \gamma_1, \alpha\}, \{r_1, r_2, r_3, \gamma_1, \alpha\}$
$V_{26} : \{r_1, r_2, r_3, \gamma_1, \alpha\}, \{r_1, r_2, r_3, \alpha\}$
$V_{27} : \{r_1, r_2, r_3, \gamma_2\}, \{r_1, r_2, r_3, \gamma_2\}$
$V_{28} : \{r_1, r_2, r_3, \gamma_2, \alpha\}, \{r_1, r_2, r_3, \gamma_2, \alpha\}$
$V_{29} : \{r_1, r_2, r_3, \gamma_2, \alpha\}, \{r_1, r_2, r_3, \alpha\}$
$V_{30} : \{r_1, r_2, r_3, \gamma_3\}, \{r_1, r_2, r_3, \gamma_3\}$
$V_{31} : \{r_1, r_2, r_3, \gamma_3, \alpha\}, \{r_1, r_2, r_3, \gamma_3, \alpha\}$
$V_{32} : \{r_1, r_2, r_3, \gamma_3, \alpha\}, \{r_1, r_2, r_3, \alpha\}$
$V_{33} : \{r_1, r_2, r_3, \gamma_1, \alpha\}, \{r_1, r_2, r_3\}$
$V_{34} : \{r_1, r_2, r_3, \gamma_1, \alpha, s_1\}, \{r_1, r_2, r_3, s_1\}$
$V_{36} : \{r_1, r_2, r_3, \gamma_1, \gamma_2, \alpha, s_1\}, \{r_1, r_2, r_3, \alpha, s_1\}$
$V_{37} : \{r_1, r_2, r_3, \gamma_1, \gamma_2, \gamma_3, \alpha, s_1\}, \{r_1, r_2, r_3, \alpha, s_1\}$
$V_{38} : \{r_1, r_2, r_3, \alpha, s_1\}, \{r_1, r_2, r_3, \alpha, s_1\}$
$V_{39} : \{r_1, r_2, r_3, \alpha, s_1\}, \{r_1, r_3, \alpha, s_1\}$
$V_{40} : \{r_1, r_2, r_3, \alpha, s_1\}, \{r_1, r_2, r_3, \alpha, s_1\}$
$V_{43} : \{r_1, r_2, r_3, \alpha\}, \{r_1, r_3, \alpha\}$
$V_{44} : \{r_1, r_2, r_3, \alpha, s_1\}, \{r_1, r_2, r_3, \alpha, s_1\}$
$V_{45} : \{r_1, r_2, r_3, \alpha, s_1\}, \{r_1, r_2, r_3, \alpha, s_1\}$
$V_{62} : \{r_1, r_2, r_3, \alpha, s_1, s_2\}, \{r_1, r_2, r_3, \alpha, s_1, s_2\}$
$V_{63} : \{s_1, s_2, \kappa_1, \kappa_2\}, \{s_1, s_2, \kappa_1, \kappa_2\}$

We have special cases with V_{35} and V_{41}, where the contribution from x has a non-uniform distribution protected by other values with non-uniform distributions. Hence, we define the sets associated with V_{35} and V_{41} as

$$V_{35} : \{r1, r2, r3, \gamma_1, \gamma_2, \alpha, s_1\}, \{\varnothing\}$$
$$V_{41} : \{r_1, r_2, r_3, \alpha, s_1\}, \{\varnothing\}$$

where we not that the empty set used in the second set does not imply an attack is trivial since x is not uniformly distributed.

For each of V_0, V_1, V_{22}, V_{23}, V_{24}, V_{25}, V_{26}, V_{27}, V_{28}, V_{29}, V_{30}, V_{31}, V_{32}, V_{33}, V_{34}, V_{36}, V_{37}, V_{38}, V_{39}, V_{40}, V_{43}, V_{44}, V_{45}, V_{62}, V_{63} affected by x, the uniformly distributed variables affecting each intermediate state (the second listed set) and V_{35}, V_{41} where we consider the first set. There does not exist another combination of two sets, following the attacks enumerated above, with the same combination of variables that would allow an instance of x to be unmasked permitting a side-channel attack.

Hence, Algorithm 3 is resistant to first, second and third-order side-channel analysis. □

C Implementation Considerations

All algorithms described in this paper have the property that all calculated intermediates (and also higher-order combinations thereof) are statistically independent of the secret value x. In the past, it has however been shown that the claimed security order of those algorithms is usually lower when they are directly applied in software or hardware. The reason for this, for example in case of a software implementation, lies in the fact that intermediate values are often unintentionally combined by the underlying hardware architecture. One Typical cause of leaks is where intermediate values of the algorithm, which are stored in some registers, get overwritten with other intermediate results of the algorithms. Thus, the Hamming distance of both intermediates will leak information. Other reasons for leakage are the combination of internal signals that depend on two or more intermediate values which are either stored in registers or currently (or previously) used in operations in the processor's datapath. First-order secure algorithms will therefore often show first-order leakage in practice. The same holds true for higher-order secure algorithms which resistance level has shown to be actually lower than claimed.

Direct applications of secure algorithms in hardware require similar care when implemented. Integrated circuits in CMOS, for example, have the property that many gates make output transitions several times per clock cycle. Such transitions (or often called *glitches*) contain information about the secret value, even though all intermediates have been carefully masked at the algorithm level. State-of-the-art countermeasures try to get rid of those physical effects by applying (additional) masking or hiding techniques on either the gate level (e.g., using secure logic styles such as dual-rail logic etc.) or algorithm level (e.g., using secret sharing and multi-party computation such as threshold implementations).

Implementation of secure algorithms that have been proven secure, for example, that every calculated intermediate is statistically independent of the secret, can, therefore, not be automatically considered "secure". However, such algorithms are important to be able to construct secure systems. The proposed algorithms in this paper can be used in combination with other countermeasures in order to guarantee resistance at the claimed security order.