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 31 instructions and for a third-order secure mask conversion that uses 74 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-2 [5], ChaCha [6], Blake [7], Skein [8], IDEA [9], RC6 [10], 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 [11]. His method is secure against first-order analysis, but does not resist second-order attacks. The solutions in the literature use *recursive* methods [12,13], where the missing carry bits are calculated using a masked-adder structure, or Look-up Table based methods [14,15], 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 [14].

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 31 instructions and a third-order secure algorithm that requires only 74 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 [12] 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. Section 5 discusses implementation considerations in both software and hardware. Conclusions are drawn in Section 6.

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. [16] 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 [12].

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 [11]. 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 $\gamma$, 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. $\square$

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 [11] 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 [17]. 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. [13], 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 [18,19] 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 [20]. Coron et al. [21] 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 [22] and Ishai et al. [23], respectively.

2.3 Higher-Order Boolean-to-Arithmetic Masking

Coron et al. [12] 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 [14], where, to achieve the desired level of resistance, the algorithm adopts the generic second-order secure S-box implementation of Rivain et al. [24]. 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 [15], 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 $s_1, s_2 \in (\mathbb{Z}_{2^k}, \oplus, +)$.

In the following, we try to express the ideas behind the mask conversion as clearly as possible. This leads to many instances where an expedient expression will leak if implemented as shown because of the ordering of operations. We highlight some of the issues in the text but do not attempt to enumerate all the possible leaks that could be caused by incorrectly ordering operations.

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 $\mathbb{F}_2$ [11].

3.2 The Algorithm

Our conversion method consists of three steps.

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

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

If we consider (13), we note that the combination of $x + (r_1 \oplus r_2 \oplus \alpha)$ with either $\Phi(x' \oplus \alpha, \gamma_1)$ or $\Phi(x' \oplus \alpha, \gamma_2)$ will not be statistically independent of x. We can, prevent this by applying a Boolean mask $s_1 \in \mathbb{Z}_{2^k}$, giving:

$$\begin{aligned} (x + (r_1 \oplus r_2 \oplus \alpha)) \oplus s_1 = & \Phi(x' \oplus \alpha, \gamma_1) \oplus \Phi(x' \oplus \alpha, \gamma_2) \oplus \\ & \Phi(x' \oplus \alpha, \gamma_1 \oplus \gamma_2 \oplus r_1 \oplus r_2 \oplus \alpha) \oplus s_1\,. \end{aligned} \tag{14}$$

The order that (14) 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.

Then, given $(x + (r_1 \oplus r_2 \oplus \alpha)) \oplus s_1$, one can apply Goubin's first-order Boolean to arithmetic mask conversion, as described in Algorithm 1, which will produce

$$x + (r_1 \oplus r_2 \oplus \alpha) + s_1 \tag{15}$$

without any first or second-order leakage.

Step 2: The second step is another Boolean-to-arithmetic mask conversion to securely compute $s_2 - (r_1 \oplus r_2 \oplus \alpha)$, where s_2 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 $s_2' = s_2 \oplus (r_1 \oplus r_2 \oplus \alpha)$ as the Boolean masked input and $s_2'' = s_2 - (r_1 \oplus r_2 \oplus \alpha)$ as the arithmetically masked output of the following conversion. Then, given (5), we have

$$s_2'' = s_2' \oplus \bar{\Phi}(s_2', \delta) \oplus \bar{\Phi}(s_2', \delta \oplus r_1 \oplus r_2 \oplus \alpha)\,, \tag{16}$$

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

$$s_2'' = s_2' \oplus \bar{\Phi}(s_2', r_1) \oplus \bar{\Phi}(s_2', r_2 \oplus \alpha)\,, \tag{17}$$

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

$$s_2 - (r_1 \oplus r_2 \oplus \alpha) = \bar{\Phi}(s_2', r_1) \oplus \bar{\Phi}(s_2', r_2) \oplus \bar{\Phi}(s_2', \alpha) . \qquad (18)$$

Equation (17) requires a total of 7 `XOR`s and 2 additions, whereas Equation (18) requires 5 `XOR`s 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: We can now compute the desired arithmetically masked value x'' by combining the output of (15) and (18), i.e.,

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

3.3 Implementation Details

Algorithm 2 shows the second-order secure Boolean-to-arithmetic mask conversion described above, which requires 31 instructions.

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 $z \leftarrow r_2 \oplus s_1$
14 $u \leftarrow u \oplus r_2$
15 $u \leftarrow u \oplus v$
16 $u \leftarrow u \oplus w$
17 $v \leftarrow u \oplus s_1$
18 $v \leftarrow v + r_2$
19 $w \leftarrow u \oplus z$
20 $v \leftarrow v \oplus w$
21 $w \leftarrow u + z$
22 $z \leftarrow v \oplus w$
23 $w \leftarrow \alpha \oplus r_2$
24 $u \leftarrow s_2 \oplus r_1$
25 $u \leftarrow u - w$
26 $w \leftarrow w \oplus s_2$
27 $v \leftarrow w \oplus r_1$
28 $w \leftarrow w - r_1$
29 $u \leftarrow u \oplus v$
30 $u \leftarrow u \oplus w$
31 $z \leftarrow z + u$
   return $z$
```

We prove the security of Algorithm 2 using the probing model proposed by Ishai, Sahai, and Wagner [23], where we seek to show that it is secure for up to two probes. For this, we use the refined model proposed by Barthe et al. [25] where we make use of the t-SNI (Strong Non-Interference) construction, with t being the security order. This allows us to prove that an algorithm is only vulnerable to a side-channel attack of order n, where $n \geq t + 1$ (rather than $n \geq 2t + 1$ required by Ishai et al.).

Lemma 2. (2-SNI of Algorithm 2) *Let $\{x', r_1, r_2\}$ be the input shares of Algorithm 2, and $\{x'', s_1, s_2\}$ be the output shares for any set of t intermediate variables and any subset $|k| \leq t_k$ of output shares such that $t + t_k \leq 2$, there exists a subset I of intermediate variables with $|I| \leq 2$, such that the distribution of those t intermediate variables, and the output shares can be perfectly simulated from $\{x', r_1, r_2\}$.*

Proof. We construct two sets $I = \{x', r_1, r_2\}$ and $J = \{\gamma_1, \gamma_2, \alpha, s_1, s_2\}$ corresponding to the input shares and the random values required, respectively. We denote a_i, for $1 \leq i \leq 31$, as the intermediate values in Algorithm 2, the definition of which means that is easy to see that each a_i can be perfectly simulated from the input shares and/or the required random values. That is, any internal variable within Algorithm 2 can be perfectly simulated from a subset of elements from I and/or J. $\square$

This was validated by implementing a simulator and verifying that the distributions of all a_i, for $1 \leq i \leq 31$ are identical for all values of $x \in \mathbb{Z}_{2^4}$, without loss of generality. Likewise, the simulator also verified the joint distribution of all possible combinations of pairs of elements in $\{x', r_1, r_2\} \cup \{\gamma_1, \gamma_2, \alpha, s_1, s_2\} \cup \{a_1, \ldots, a_{31}\}$ (i.e., the union of the set of inputs, required random values, and intermediate states) are identical for all values of $x \in \mathbb{Z}_{2^4}$, without loss of generality. Thus, demonstrating that Algorithm 2 is resistant to first and second order side-channel analysis.

Remark 1. We note our proof is somewhat different to the proofs described by Barthe et al. [25]. Typically, one would seek to model intermediate states as random values to ease the computation complexity of the verification. However, the combination of Boolean and arithmetic operations makes this difficult, and it is better to model the random values as inputs to determine whether the intermediate states are identical for all input.

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).

In the following, we try to express the ideas behind the mask conversion as clearly as possible. This leads to many instances where an expedient expression will leak if implemented as shown because of the ordering of operations. We highlight some of the issues in the text but do not attempt to enumerate all the possible leaks that could be caused by incorrectly ordering operations.

4.1 The Algorithm

Our conversion method consists of four steps.

1. We compute $x + (\alpha \oplus \bigoplus_{i=1}^{n} r_i) + \bigoplus_{i=1}^{n-1} \mu_i$ for some random values $\alpha, \mu_i \in \mathbb{Z}_{2^k}$.
2. We compute $(\alpha \oplus \bigoplus_{i=1}^{n} r_i) + \bigoplus_{i=1}^{n-1} \mu_i + \bigoplus_{i=1}^{n-1} \kappa_i$ for some random values $\kappa_i \in \mathbb{Z}_{2^k}$.
3. We compute $\bigoplus_{i=1}^{n-1} \kappa_i + \sum_{i=1}^{n} s_i$ for all output masks $s_i \in \mathbb{Z}_{2^k}$.
4. We combine the results of Steps 1, 2, and 3 to obtain $x + \sum_{i=1}^{n} s_i$.

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

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

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

$$x + \bigoplus_{i=1}^{n} r_i = ((n \wedge 1)\, x') \oplus \left(\bigoplus_{i=1}^{n} \Phi(x', \gamma_i)\right) \oplus \Phi\left(x', \bigoplus_{i=1}^{n} \gamma_i \oplus r_i\right)\,. \tag{20}$$

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 (20), we apply an extra Boolean mask, $\alpha \in \mathbb{Z}_{2^k}$, following the reasoning given in Section 3, i.e.,

$$x + \left(\alpha \oplus \bigoplus_{i=1}^{n} r_i\right) = ((n \wedge 1)\,(x' \oplus \alpha)) \oplus \left(\bigoplus_{i=1}^{n} \Phi(x' \oplus \alpha, \gamma_i)\right) \oplus \Phi\left(x' \oplus \alpha, \alpha \oplus \bigoplus_{i=1}^{n} \gamma_i \oplus r_i\right)\,, \tag{21}$$

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

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

to avoid any second-order leakage caused by combining $(x' \oplus \alpha)$ with the output of (21).

However, the computation would still cause a higher-order leak, i.e., when x', α, and (21) get combined. Thus, we are required to add extra masks to prevent this leakage, and we use ν_i for $i \in \{1, \ldots, n-1\}$ and also ξ_i for $i \in \{1, \ldots, n-2\}$, as follows:

$$\begin{aligned}\left(x + \left(\alpha \oplus \bigoplus_{i=1}^{n} r_i\right)\right) \oplus \bigoplus_{i=1}^{n-1} \nu_i = \\ &\bigoplus_{i=1}^{n-2} \xi_i \oplus ((n \wedge 1)\,(x' \oplus \alpha)) \oplus \left(\bigoplus_{i=1}^{n-1} \Phi(x' \oplus \alpha, \gamma_i) \oplus \nu_i\right) \\ &\oplus \Phi(x' \oplus \alpha, \gamma_n)) \oplus \Phi\left(x' \oplus \alpha, \alpha \oplus \bigoplus_{i=1}^{n} \gamma_i \oplus r_i\right) \oplus \bigoplus_{i=1}^{n-2} \xi_i \,.\end{aligned} \tag{22}$$

Note that the masks ξ_i are used to protect the intermediate values of (22) as there is not a secure way of ordering these terms without causing leakage. The masks, therefore, need to be interleaved with the computation and removed at the end.

The result is then passed through a function that will perform a Boolean-to-arithmetic mask conversion of order $n-1$ to replace the Boolean operation with an arithmetic operation, giving

$$x + \left(\alpha \oplus \bigoplus_{i=1}^{n} r_i\right) + \bigoplus_{i=1}^{n-1} \mu_i \tag{23}$$

where, μ_i, for $i \in \{1, \ldots, n-1\}$, are random values that replace the input masks as required by Definition 2.

Step 2: In the second step, we wish to compute $(\alpha \oplus \bigoplus_{i=1}^{n} r_i) + \bigoplus_{i=1}^{n-1} \mu_i + \bigoplus_{i=1}^{n-1} \kappa_i$, for some random values $\kappa_i \in \mathbb{Z}_{2^k}$. In which, we view the combination of any elements of $\{\kappa_1, \ldots, \kappa_{n-1}\}$, $\{\mu_1, \ldots, \mu_{n-1}\}$, and, likewise, the combination of any elements of $(r_1 \oplus \ldots \oplus r_n)$ as secret. Let $\delta = \alpha \oplus \bigoplus_{i=1}^{n} r_i \oplus \bigoplus_{i=1}^{n-1} \kappa_i$, then, given (20), we can compute

$$\left(\alpha \oplus \bigoplus_{i=1}^{n} r_i\right) + \bigoplus_{i=1}^{n-1} \kappa_i = (((n-1) \wedge 1)\,\delta) \oplus \left(\bigoplus_{i=1}^{n-1} \Phi(\epsilon, \beta_i)\right) \oplus \Phi\left(\epsilon, \bigoplus_{i=1}^{n-1} \kappa_i \oplus \beta_i\right), \tag{24}$$

where β_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, the terms of the XOR sums need to be computed separately, i.e., $\bigoplus_{i=1}^{n} r_i \oplus \bigoplus_{i=1}^{n-1} \kappa_i = (\kappa_1 \oplus r_1) \oplus (\kappa_2 \oplus r_2) \oplus \ldots \oplus (\kappa_{n-1} \oplus r_{n-1}) \oplus r_n$.

We further need to add additional masks to avoid higher-order leakage. As in Step 1, we use $(n-1)$ masks, i.e., we random values ζ_i for $i \in \{1, \ldots, n-1\}$

as follows:

$$\left(\alpha \oplus \bigoplus_{i=1}^{n} r_i + \bigoplus_{i=1}^{n-1} \kappa_i\right) \oplus \bigoplus_{i=1}^{n-1} \zeta_i = (((n-1) \wedge 1)\, \delta) \oplus \left(\bigoplus_{i=1}^{n-1} \Phi(\delta, \beta_i) \oplus \zeta_i\right) \oplus \Phi\left(\delta, \bigoplus_{i=1}^{n-1} \kappa_i \oplus \beta_i\right) . \tag{25}$$

Finally, we can perform a Boolean-to-arithmetic mask conversion of order $n-1$ to replace the Boolean operation with an arithmetic operation giving

$$\left(\alpha \oplus \bigoplus_{i=1}^{n} r_i + \bigoplus_{i=1}^{n-1} \kappa_i\right) + \bigoplus_{i=1}^{n-1} \ell_i \tag{26}$$

where, ℓ_i, for $i \in \{1, \ldots, n-1\}$, are random values that replace the input masks as required by Definition 2.

Step 3: In the third step, we wish to compute $\bigoplus_{i=1}^{n-1} \kappa_i + \sum_{i=1}^{n} s_i$, for some random values to be used as output masks $s_i \in \mathbb{Z}_{2^k}$, for $i \in \{1, \ldots, n\}$. This can be achieved by conducting an $(n-2)^{th}$-order secure Boolean-to-arithmetic mask conversion (e.g., using Algorithm 1 when $n = 3$ or Algorithm 2 when $n = 4$ etc.) using the input $\bigoplus_{i=1}^{n-1} \kappa_i \oplus \bigoplus_{i=1}^{n-2} \lambda_i$, λ_i (for $i \in \{1, \ldots, n-2\}$) are random values, resulting in $\bigoplus_{i=1}^{n-1} \kappa_i + \sum_{i=1}^{n-2} s_i$. Note that we choose the output masks of the lower-order conversion to be used as output masks for the order we are considering.

Then one can add s_{n-1} and s_n to get the desired result, i.e.,

$$\bigoplus_{i=1}^{n-1} \kappa_i + \sum_{i=1}^{n} s_i \, . \tag{27}$$

Step 4: We add the output of each step. Adding the output of Step 1 to the output of Step 3, produces

$$x + \left(\alpha \oplus \bigoplus_{i=1}^{n} r_i\right) + \bigoplus_{i=1}^{n-1} \mu_i + \bigoplus_{i=1}^{n-1} \kappa_i + \sum_{i=1}^{n} s_i \, . \tag{28}$$

Then subtracting the output of Step 2 results in

$$x + \sum_{i=1}^{n} s_i \, . \tag{29}$$

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 74 instructions.

As previously, we proceed with a 3-SNI proof:

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
       $\gamma_1, \gamma_2, \gamma_3, \beta_1, \beta_2, \delta_1, \delta_2, \kappa_1, \kappa_2, \alpha, \mu_1, \mu_2, s_1, s_2, s_3 \in \mathbb{Z}_{2^k}$ for some $k \in \mathbb{Z}_{\geq 0}$
Output: $x'' = x + s_1 + s_2 + s_3$
1  $z \leftarrow \kappa_1 \oplus r_1$
2  $z \leftarrow z \oplus \kappa_2$
3  $z \leftarrow z \oplus r_2$
4  $z \leftarrow z \oplus r_3$
5  $z \leftarrow z \oplus \alpha$
6  $w \leftarrow z \oplus \beta_1$
7  $u \leftarrow w + \beta_1$
8  $u \leftarrow u \oplus \mu_1$
9  $v \leftarrow z \oplus \beta_2$
10 $v \leftarrow v + \beta_2$
11 $u \leftarrow u \oplus v$
12 $v \leftarrow w \oplus \kappa_1$
13 $v \leftarrow v \oplus \beta_2$
14 $v \leftarrow v \oplus \kappa_2$
15 $w \leftarrow v \oplus z$
16 $v \leftarrow v + w$
17 $v \leftarrow v \oplus \mu_2$
18 $w \leftarrow u \oplus v$
19 $z \leftarrow r_3 \oplus \mu_1$
20 $z \leftarrow z \oplus \mu_2$
21 $u \leftarrow w \oplus r_3$
22 $u \leftarrow u + r_3$
23 $v \leftarrow w \oplus z$
24 $v \leftarrow v + z$
25 $u \leftarrow u \oplus w$
26 $w \leftarrow u \oplus v$
27 $u \leftarrow x' \oplus \gamma_1$
28 $u \leftarrow u \oplus \alpha$
29 $u \leftarrow u + \gamma_1$
30 $u \leftarrow u \oplus \gamma_1$
31 $u \leftarrow u \oplus \mu_1$
32 $v \leftarrow x' \oplus \gamma_2$
33 $v \leftarrow v \oplus \alpha$
34 $v \leftarrow v + \gamma_2$
35 $u \leftarrow u \oplus v$
36 $v \leftarrow x' \oplus \gamma_3$
37 $v \leftarrow v \oplus \alpha$
38 $v \leftarrow v + \gamma_3$
39 $u \leftarrow u \oplus \alpha$
40 $u \leftarrow u \oplus v$
41 $z \leftarrow \gamma_1 \oplus r_1$
42 $z \leftarrow z \oplus \gamma_2$
43 $z \leftarrow z \oplus r_2$
44 $z \leftarrow z \oplus \gamma_3$
45 $z \leftarrow z \oplus r_3$
46 $v \leftarrow z \oplus \alpha$
47 $z \leftarrow x' \oplus z$
48 $z \leftarrow z + v$
49 $z \leftarrow z \oplus u$
50 $z \leftarrow z \oplus \mu_2$
51 $z \leftarrow z \oplus x'$
52 $z \leftarrow z \oplus \gamma_1$;
53 $u \leftarrow z \oplus \delta_1$
54 $u \leftarrow u + \delta_1$
55 $v \leftarrow z \oplus \delta_2$
56 $v \leftarrow v + \delta_2$
57 $u \leftarrow u \oplus v$
58 $v \leftarrow \delta_1 \oplus \mu_2$
59 $v \leftarrow v \oplus \delta_2$
60 $v \leftarrow v \oplus \mu_1$
61 $z \leftarrow z \oplus v$;
62 $z \leftarrow z + v$
63 $z \leftarrow z \oplus u$;
64 $v \leftarrow \kappa_1 \oplus s_1$
65 $u \leftarrow v + \kappa_2$
66 $u \leftarrow u \oplus v$
67 $u \leftarrow u \oplus \kappa_2$
68 $v \leftarrow \kappa_2 \oplus s_1$
69 $v \leftarrow v + \kappa_1$
70 $u \leftarrow u \oplus v$
71 $u \leftarrow u + s_2$
72 $u \leftarrow u + s_3$
73 $z \leftarrow z + u$
74 $z \leftarrow z - w$
   return $z$
```

Lemma 3. (3-SNI of Algorithm 3) *Let $\{x', r_1, r_2, r_3\}$ be the input shares of Algorithm 3, and $\{x'', s_1, s_2, s_3\}$ be the output shares for any set of t intermediate variables and any subset $|k| \leq t_k$ of output shares such that $t + t_k \leq 3$, there exists a subset I of intermediate variables with $|I| \leq 3$, such that the distribution of those t intermediate variables, and the output shares can be perfectly simulated from $\{x', r_1, r_2, r_3\}$.*

Proof. We construct two sets $I = \{x', r_1, r_2, r_3\}$ and $J = \{\gamma_1, \gamma_2, \gamma_3, \beta_1, \beta_2, \delta_1, \delta_2, \kappa_1, \kappa_2, \alpha, \mu_1, \mu_2, s_1, s_2, s_3\}$ corresponding to the input shares and the random values required, respectively. We denote a_i, for $1 \leq i \leq 74$, as the intermediate values in Algorithm 3, the definition of which means that is easy to see that each a_i can be perfectly simulated from the input shares and/or the required random values. That is, any internal variable within Algorithm 3 can be perfectly simulated from a subset of elements from I and/or J. □

This was validated by implementing a simulator and verifying that the distributions of all a_i, for $1 \leq i \leq 74$ are identical for all values of $x \in \mathbb{Z}_{2^4}$, without

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

$\boldsymbol{B \to A}$	**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	-	31	74	123	242	386	557	753

loss of generality. Likewise, the simulator also verified the joint distribution of all possible combinations of pairs and triplets of elements in $\{x', r_1, r_2, r_3\} \cup \{\gamma_1, \gamma_2, \gamma_3, \beta_1, \beta_2, \delta_1, \delta_2, \kappa_1, \kappa_2, \alpha, \mu_1, \mu_2, s_1, s_2, s_3\} \cup \{a_1, \ldots, a_{74}\}$ (i.e., the union of the set of inputs, required random values, and intermediate states) are identical for all values of $x \in \mathbb{Z}_{2^4}$, without loss of generality. Thus, demonstrating that Algorithm 3 is resistant to first, second, and third-order side-channel analysis.

The number of inputs required for Algorithm 3 would seem to be too large for an efficient exhaustive search through all the possible sources of third-order leakage. However, we note that in simulating individual operations only a subset of the inputs or random values are required. The effect is similar to the use of gadgets in t-SNI proofs [25], but the use of arithmetic operations means that there is often interference between different parts of the algorithm. Detecting were this occurs to allow one to break such algorithms into gadgets, is left for future research.

4.3 Comparison

Table 1 compares the performance of our proposed method with related work. We consider the work of Coron et al. [12] 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.

We estimate the costs for Coron et al.'s higher-order Boolean-to-arithmetic mask conversion method [12] as follows. For a single masked `AND` (`SecAnd`) operation [12, 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 [12, 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 [12, Section 5] for register sizes of 8, 16, 32, and 64 bits and provide the results in Table 1. It 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.

Performance details. Table 2 lists the number of required instructions in terms of arithmetic and Boolean operations up to a security order of 8. We list Goubin's solution in the first-order security case, and our solution in the other cases. Note that Coron et al.'s solution [12] 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* → *A	**Security Order**							
Conversion	1	2	3	4	5	6	7	8
Arithmetic operations	2	8	18	28	42	56	74	110
Boolean operations	5	23	56	95	200	330	483	643

Required number of random variables. 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 [12] 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 [11].

5 Implementation Considerations

All algorithms described in this paper have the property that all calculated intermediates (and relevant higher-order combinations thereof) are statistically independent of the secret value x. In the past, it has been shown that the claimed

Table 3. Comparison of required number of random variables.

B $\rightarrow$ ***A*** **Conversion**	**Security Order**							
	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	15	26	42	59	81	104

security order of those algorithms is usually lower when they are directly applied in software or hardware. For example, in a software implementation intermediate values are often unintentionally combined by the underlying hardware architecture. One typical cause of leakage is where intermediate values of the algorithm, which are stored in some registers, get overwritten with other intermediate results of the algorithms. Other sources of leakage include the combination of internal signals that depend on two or more intermediate values which are either stored in registers (register interferences) or currently (or previously) used in operations in the processor's datapath. Hence, implementations of first-order side-channel resistant algorithms may show first-order leakages in practice, and the same holds true for higher-order secure algorithms whose resistance level has shown to be actually lower than claimed [26].

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 (*glitches*) contain information about the secret value, even though all intermediates have been carefully masked at the algorithm level [27]. State-of-the-art countermeasures try to get rid of those physical effects by applying (additional) countermeasures at the gate level (e.g., using secure logic styles such as dual-rail logic [28]) or algorithm level (e.g., using secret sharing and multi-party computation such as threshold implementations [18]).

Naïve implementation of algorithms that have been proven secure—in the sense that every calculated intermediate is statistically independent of the secret value—can, therefore, not be automatically considered resistant to side-channel analysis. However, the algorithms proposed in this paper can be combination with other countermeasures in order to guarantee resistance at the claimed security order. We do not provide any further details here since the countermeasures required will vary from one platform to another.

6 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 n and are independent of the input-word size. We present explicit algorithms for a second-order secure mask conversion that requires 31 instructions, i.e., a multiple of about 4 compared to the instruction count of Goubin's method (7 instructions), and a third-order secure mask conversion that requires 74 instructions, i.e., a factor of about 10 more expensive 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 than 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
5. National Institute of Standards and Technology (NIST): FIPS-180-4: Secure Hash Standard (August 2015) http://csrc.nist.gov/publications/fips/fips180-4.
6. Bernstein, D.J.: Chacha, a variant of salsa20 (2008)
7. Aumasson, J.P., Henzen, L., Meier, W., Phan, R.C.W.: SHA-3 Proposal BLAKE (December 2010) https://131002.net/blake.
8. Ferguson, N., Lucks, S., Schneier, B., Whiting, D., Bellare, M., Kohno, T., Callas, J., Walker, J.: The Skein Hash Function Family (October 2010) http://www.skein-hash.info.
9. Lai, X., Massey, J.L.: A Proposal for a New Block Encryption Standard. In Damgård, I., ed.: Workshop on the Theory and Application of Cryptographic Techniques. Volume 473 of LNCS., Springer, Heidelberg (1990) 389–404
10. Rivest, R.L., Robshaw, M., Sidney, R., Yin, Y.: The RC6 Block Cipher. ftp://ftp.rsasecurity.com/pub/rsalabs/rc6/rc6v11.pdf (August 1998)
11. 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
12. Coron, J.S., 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

13. 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
14. 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
15. 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
16. Mangard, S., Oswald, E., Popp, T.: Power Analysis Attacks — Revealing the Secrets of Smart Cards. Springer (2007)
17. Golić, J.D.: Techniques for Random Masking in Hardware. IEEE Transactions on Circuits and Systems **54**(2) (2007) 291–300
18. 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
19. 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
20. 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
21. Coron, J.S., 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
22. Trichina, E.: Combinational Logic Design for AES SubByte Transformation on Masked Data. IACR Cryptology ePrint Archive **2003** (2003) 236
23. 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
24. 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
25. Barthe, G., Belaïd, S., Dupressoir, F., Fouque, P.A., Grégoire, B., Strub, P.Y., Zucchini, R.
26. Balasch, J., Gierlichs, B., Grosso, V., Reparaz, O., Standaert, F.X.: On the Cost of Lazy Engineering for Masked Software Implementations. In Joye, M., Moradi, A., eds.: CARDIS 2014. Volume 8968 of LNCS., Springer, Heidelberg (2014) 64–81
27. Mangard, S., Popp, T., Gammel, B.M.: Side-channel leakage of masked CMOS gates. In Menezes, A., ed.: CT-RSA 2005. Volume 3376 of LNCS., Springer, Heidelberg (2005) 351–365
28. Leiserson, A.J., Marson, M.E., Wachs, M.A.: Gate-level masking under a path-based leakage metric. In Batina, L., Robshaw, M., eds.: CHES 2014. Volume 8731 of LNCS., Springer, Heidelberg (2014) 580–597