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 and has quadratic complexity as the security order increases. A significant improvement over previous work that has exponential complexity. 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 second-order secure algorithm is at least an order of magnitude faster and our third-order secure algorithm is more than twice as fast as other algorithms in the literature.

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

1 Introduction

Differential Power Analysis (DPA) was introduced as a means of extracting cryptographic keys by Kocher, Jaffe, and Jun [16] 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 [1,10,25].

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 the addition of integers, e.g., SHA-2 [22], ChaCha [5], Blake [2], Skein [9], IDEA [17], RC6 [27], 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 [12]. His method is secure against first-order analysis, but does not resist second-order attacks. The solutions in the literature use *recursive* methods [7,28], where the missing carry bits are calculated using a masked-adder structure, or Look-up Table based methods [30,31], 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 [30].

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. Our algorithms are significantly faster than the best recursive methods in the literature [7].

This paper is an extended version of previous work first published on the IACR's eprint server in 2016 [13]. After a weakness was identified in our third, and higher-order, algorithms [8], the algorithms were updated and are presented here. Our second-order secure algorithm remains secure, and is at least one order of magnitude faster than previous work. The algorithms we present were tested using exhaustive simulation, inspired by the state-of-the-art Strong Non-Interference (SNI) notion. Proofs based on this notion are also provided. We show that our corrected algorithms are, at a minimum, about twice as fast as related work—including recent work [8].

In addition to the new contributions, we highlight the importance of the number of random numbers when comparing algorithms. Many algorithms give an artificially low instruction count hiding a large number of required random numbers. In comparing our work with other published works we take into account the number of random values required to implement an algorithm by considering the number of instructions required to compute an Xorshift random number

generator [21] used by Coron's explicit implementation [8]. We demonstrate that our work is about twice as fast as Coron's in all cases. Moreover, our algorithms have quadratic complexity with regard to the security order, whereas Coron's algorithms have exponential complexity.

The paper is organized as follows. In Section 2, we describe the general 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. In Section 5, we compare our work with other algorithms in the literature, and discusses implementation considerations in both software and hardware in Section 6. Conclusions are drawn in Section 7.

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 (*The problem of converting Boolean to arithmetic masks*). *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 can be generalized to higher-order masking schemes as follows:

Definition 2 (*The problem of converting Boolean to arithmetic masks of higher order*). *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. [19] for a more detailed discussion). If an attacker can combine the input x' and the output x'' using some side-channel information 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, $c_1 = (x + r_1) \oplus x \oplus r_1$ and $c_1 = (x + r_1 + r_2) \oplus x \oplus r_1 \oplus r_2$. 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 [7].

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 [12]. 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 $\mathbb{F}_2$. It follows that $(\Phi(a, b) \oplus \Phi(a, 0))$ is *linear* for any $b \in \mathbb{Z}$. 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.

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

Goubin then proceeds to give a proof of the following:

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):

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

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 [12] 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. [28], 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 [23,24] 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 [15]. Coron et al. [6] 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 [29] and Ishai et al. [14], respectively.

2.3 Higher-Order Boolean-to-Arithmetic Masking

Coron et al. [7] proposed a method for conducting a 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 [30], where, to achieve the desired level of resistance, the algorithm adopts the generic second-order secure S-box implementation of Rivain et al. [26]. 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 [31], 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 $\mathbb{F}_2$ [12].

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),

$$x + r = (x \oplus r) \oplus \Phi(x \oplus r, \gamma) \oplus \Phi(x \oplus r, \gamma \oplus r)\,. \tag{9}$$

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

$$\begin{aligned} x + (r_1 \oplus r_2) = (x \oplus r_1 \oplus r_2) &\oplus \Phi(x \oplus r_1 \oplus r_2, \gamma_1 \oplus \gamma_2) \\ &\oplus \Phi(x \oplus r_1 \oplus r_2, \gamma_1 \oplus \gamma_2 \oplus r_1 \oplus r_2)\,, \end{aligned} \tag{10}$$

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

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

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

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

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:

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

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

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

If we consider (14), 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. In

both cases, the combination will be dependent on the carry bits produced by the arithmetic operations, as all variables occur an even number of times. The effect is similar to that seen if masks are not changed in a higher-order conversion algorithm, as discussed in Section 2. We can, prevent this by applying a Boolean mask $s_1 \in \mathbb{Z}_{2^k}$, giving:

$$\begin{aligned}(x + (r_1 \oplus r_2 \oplus \alpha)) \oplus s_1 = \Phi(x' \oplus \alpha, \gamma_1) \oplus \Phi(x' \oplus \alpha, \gamma_2) \\ \oplus \Phi(x' \oplus \alpha, \gamma_1 \oplus \gamma_2 \oplus r_1 \oplus r_2 \oplus \alpha) \oplus s_1 \,. \end{aligned} \tag{15}$$

The order that (15) 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

$$x + (r_1 \oplus r_2 \oplus \alpha) + s_1 \tag{16}$$

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

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

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

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

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

$$s_2 - (r_1 \oplus r_2 \oplus \alpha) = \bar{\Phi}(s_2', r_1) \oplus \bar{\Phi}(s_2', r_2) \oplus \bar{\Phi}(s_2', \alpha) \,. \tag{19}$$

Equation (18) requires a total of 7 `XOR`s and 2 additions, whereas Equation (19) 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 (16) and (19), 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. We made some effort to reduce the number of instructions and random values required without affecting the level of security or the sequence of steps. That is, we compute the steps given at the beginning of Section 3.2. The details of the steps given above remain unmodified for clarity.

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 $z \leftarrow \gamma_1 \oplus r_1$	**12** $w \leftarrow w + \gamma_2$	**23** $w \leftarrow \alpha \oplus r_2$
2 $z \leftarrow z \oplus \gamma_2$	**13** $z \leftarrow r_2 \oplus s_1$	**24** $u \leftarrow s_2 \oplus r_1$
3 $z \leftarrow z \oplus r_2$	**14** $u \leftarrow u \oplus r_2$	**25** $u \leftarrow u - w$
4 $u \leftarrow x' \oplus z$	**15** $u \leftarrow u \oplus v$	**26** $w \leftarrow w \oplus s_2$
5 $z \leftarrow z \oplus \alpha$	**16** $u \leftarrow u \oplus w$	**27** $v \leftarrow w \oplus r_1$
6 $u \leftarrow u + z$	**17** $v \leftarrow u \oplus s_1$	**28** $w \leftarrow w - r_1$
7 $v \leftarrow x' \oplus \gamma_1$	**18** $v \leftarrow v + r_2$	**29** $u \leftarrow u \oplus v$
8 $v \leftarrow v \oplus \alpha$	**19** $w \leftarrow u \oplus z$	**30** $u \leftarrow u \oplus w$
9 $v \leftarrow v + \gamma_1$	**20** $v \leftarrow v \oplus w$	**31** $z \leftarrow z + u$
10 $w \leftarrow x' \oplus \gamma_2$	**21** $w \leftarrow u + z$	**return** z
11 $w \leftarrow w \oplus \alpha$	**22** $z \leftarrow v \oplus w$	

We prove the security of Algorithm 2 using the probing model proposed by Ishai, Sahai, and Wagner [14], 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. [4] 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 $n \geq 2\,t + 1$ 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. □

This was validated by implementing a simulator and verifying that the distributions of each 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 $I \cup J \cup \{a_1, \ldots, a_{31}\}$ (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. We chose the field $\mathbb{Z}_{2^4}$ to be small enough to make verification trivial and large enough that carries are propagated as they would be in an arbitrarily large field.

Remark 1. We note our proof is somewhat different to the proofs described by Barthe et al. [4]. 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 simpler to model the random values as inputs to determine whether the distribution of each intermediate state is identical for all values that the masked input can take.

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):

$$x + r = (x \oplus r) \oplus \Phi(x \oplus r, \gamma) \oplus \Phi(x \oplus r, \gamma \oplus r)\,. \tag{20}$$

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

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

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

$$x + \bigoplus_{i=1}^{n} r_i = ((n \wedge 1)\, x') \oplus \left(\bigoplus_{i=1}^{n} \Phi(x', \gamma_i)\right) \oplus \Phi\left(x', \bigoplus_{i=1}^{n} \gamma_i \oplus r_i\right)\,, \tag{22}$$

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

$$\begin{aligned} x + \left(\alpha \oplus \bigoplus_{i=1}^{n} r_i\right) = {} & ((n \wedge 1)\,(x' \oplus \alpha)) \\ & \oplus \left(\bigoplus_{i=1}^{n} \Phi(x' \oplus \alpha, \gamma_i)\right) \oplus \Phi\left(x' \oplus \alpha, \alpha \oplus \bigoplus_{i=1}^{n} \gamma_i \oplus r_i\right)\,, \end{aligned} \tag{23}$$

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 (23).

However, the computation would still cause a higher-order leak, i.e., when x', α, and (23) get combined. Thus, we are required to add extra masks to prevent this leakage, and we use μ_i for $i \in \{1, \ldots, n-1\}$ and also ξ_i for $i \in \{1, \ldots, n-2\}$, as follows:

$$\begin{aligned} \left(x + \left(\alpha \oplus \bigoplus_{i=1}^{n} r_i\right)\right) \oplus \bigoplus_{i=1}^{n-1} \mu_i = {} & \\ & \bigoplus_{i=1}^{n-2} \xi_i \oplus ((n \wedge 1)\,(x' \oplus \alpha)) \oplus \left(\bigoplus_{i=1}^{n-1} \Phi(x' \oplus \alpha, \gamma_i) \oplus \mu_i\right) \\ & \oplus \Phi(x' \oplus \alpha, \gamma_n)) \oplus \Phi\left(x' \oplus \alpha, \alpha \oplus \bigoplus_{i=1}^{n} \gamma_i \oplus r_i\right) \oplus \bigoplus_{i=1}^{n-2} \xi_i\,. \end{aligned} \tag{24}$$

Note that the masks ξ_i are used to protect the intermediate values of (24) 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 to replace the Boolean operation with an arithmetic operation, i.e.,

$$x + \left(\alpha \oplus \bigoplus_{i=1}^{n} r_i\right) + \bigoplus_{i=1}^{n-1} \mu_i. \tag{25}$$

Note that the first-order conversion requires sufficient additional masks (δ_i for $i = 1...n-1$) to not cause any leakage, i.e.,

$$x + \left(\alpha \oplus \bigoplus_{i=1}^{n} r_i\right) + \bigoplus_{i=1}^{n-1} \mu_i = ((n-1) \wedge 1)\,\zeta) \oplus \left(\bigoplus_{i=1}^{n-1} \Phi(\zeta, \delta_i)\right) \oplus \Phi\left(\zeta, \bigoplus_{i=1}^{n-1} \delta_i \oplus \mu_i\right), \tag{26}$$

where $\zeta = (x + (\alpha \oplus \bigoplus_{i=1}^{n} r_i)) \oplus \bigoplus_{i=1}^{n-1} \mu_i$.

Step 2: In the second step, we wish to compute

$$\left(\alpha \oplus \bigoplus_{i=1}^{n} r_i\right) + \bigoplus_{i=1}^{n-1} \mu_i + \bigoplus_{i=1}^{n-1} \kappa_i\,, \tag{27}$$

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

$$\epsilon = \alpha \oplus \bigoplus_{i=1}^{n} r_i \oplus \bigoplus_{i=1}^{n-1} \kappa_i\,, \tag{28}$$

then, given (22), we can compute

$$\left(\alpha \oplus \bigoplus_{i=1}^{n} r_i\right) + \bigoplus_{i=1}^{n-1} \kappa_i = (((n-1) \wedge 1)\,\epsilon) \oplus \left(\bigoplus_{i=1}^{n-1} \Phi(\epsilon, \beta_i)\right) \oplus \Phi\left(\epsilon, \bigoplus_{i=1}^{n-1} \kappa_i \oplus \beta_i\right), \tag{29}$$

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., $\bigoplus_{i=1}^{n} r_i \oplus \bigoplus_{i=1}^{n-1} \kappa_i = (\kappa_1 \oplus r_1) \oplus (\kappa_2 \oplus r_2) \oplus \ldots \oplus (\kappa_{n-1} \oplus r_{n-1}) \oplus r_n$.

As a next step, we add $n-1$ additional masks, as in Step 1, i.e., we add random values μ_i for $i \in \{1, \ldots, n-1\}$ as follows:

$$\left(\alpha \oplus \bigoplus_{i=1}^{n} r_i + \bigoplus_{i=1}^{n-1} \kappa_i\right) \oplus \bigoplus_{i=1}^{n-1} \mu_i = (((n-1) \wedge 1)\, \epsilon) \\ \oplus \left(\bigoplus_{i=1}^{n-1} \Phi(\epsilon, \beta_i) \oplus \mu_i\right) \oplus \Phi\left(\epsilon, \bigoplus_{i=1}^{n-1} \kappa_i \oplus \beta_i\right). \tag{30}$$

Finally, we can perform a first-order secure Boolean-to-arithmetic mask conversion (similar as described in Step 1) to replace the single Boolean operation with an arithmetic operation giving

$$\left(\alpha \oplus \bigoplus_{i=1}^{n} r_i + \bigoplus_{i=1}^{n-1} \kappa_i\right) + \bigoplus_{i=1}^{n-1} \mu_i, \tag{31}$$

which equals to (27).

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 where $n = 4$ 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.,

$$\bigoplus_{i=1}^{n-1} \kappa_i + \sum_{i=1}^{n} s_i\,. \tag{32}$$

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

$$x + \left(\alpha \oplus \bigoplus_{i=1}^{n} r_i\right) + \bigoplus_{i=1}^{n-1} \mu_i + \bigoplus_{i=1}^{n-1} \kappa_i + \sum_{i=1}^{n} s_i\,. \tag{33}$$

Then subtracting the output of Step 2 results in

$$x + \sum_{i=1}^{n} s_i\,. \tag{34}$$

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 call to Boolean-to-arithmetic mask conversions of a lower order will increase the time complexity to $\mathcal{O}\left(n^2\right)$.

4.2 Implementation Details

Algorithm 3 gives an explicit implementation of a third-order secure Boolean-to-arithmetic mask conversion as an example of the method described above, which requires 74 instructions.

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 γ_1, γ_2, γ_3, β_1, β_2, δ_1, δ_2, κ_1, κ_2, α, μ_1, μ_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	$z \leftarrow \kappa_1 \oplus r_1$	**26**	$w \leftarrow u \oplus v$	**51**	$z \leftarrow z \oplus \gamma_1$;
2	$z \leftarrow z \oplus \kappa_2$	**27**	$u \leftarrow x' \oplus \gamma_1$	**52**	$z \leftarrow z \oplus x'$
3	$z \leftarrow z \oplus r_2$	**28**	$u \leftarrow u \oplus \alpha$	**53**	$u \leftarrow z \oplus \delta_1$
4	$z \leftarrow z \oplus r_3$	**29**	$u \leftarrow u + \gamma_1$	**54**	$u \leftarrow u + \delta_1$
5	$z \leftarrow z \oplus \alpha$	**30**	$u \leftarrow u \oplus \gamma_1$	**55**	$v \leftarrow z \oplus \delta_2$
6	$w \leftarrow z \oplus \beta_1$	**31**	$u \leftarrow u \oplus \mu_1$	**56**	$v \leftarrow v + \delta_2$
7	$u \leftarrow w + \beta_1$	**32**	$v \leftarrow x' \oplus \gamma_2$	**57**	$u \leftarrow u \oplus v$
8	$u \leftarrow u \oplus \mu_1$	**33**	$v \leftarrow v \oplus \alpha$	**58**	$v \leftarrow \delta_1 \oplus \mu_2$
9	$v \leftarrow z \oplus \beta_2$	**34**	$v \leftarrow v + \gamma_2$	**59**	$v \leftarrow v \oplus \delta_2$
10	$v \leftarrow v + \beta_2$	**35**	$u \leftarrow u \oplus v$	**60**	$v \leftarrow v \oplus \mu_1$
11	$u \leftarrow u \oplus v$	**36**	$v \leftarrow x' \oplus \gamma_3$	**61**	$z \leftarrow z \oplus v$;
12	$v \leftarrow w \oplus \kappa_1$	**37**	$v \leftarrow v \oplus \alpha$	**62**	$z \leftarrow z + v$
13	$v \leftarrow v \oplus \beta_2$	**38**	$v \leftarrow v + \gamma_3$	**63**	$z \leftarrow z \oplus u$;
14	$v \leftarrow v \oplus \kappa_2$	**39**	$u \leftarrow u \oplus \alpha$	**64**	$v \leftarrow \kappa_1 \oplus s_1$
15	$w \leftarrow v \oplus z$	**40**	$u \leftarrow u \oplus v$	**65**	$u \leftarrow v + \kappa_2$
16	$v \leftarrow v + w$	**41**	$z \leftarrow \gamma_1 \oplus r_1$	**66**	$u \leftarrow u \oplus v$
17	$v \leftarrow v \oplus \mu_2$	**42**	$z \leftarrow z \oplus \gamma_2$	**67**	$u \leftarrow u \oplus \kappa_2$
18	$w \leftarrow u \oplus v$	**43**	$z \leftarrow z \oplus r_2$	**68**	$v \leftarrow \kappa_2 \oplus s_1$
19	$z \leftarrow r_3 \oplus \mu_1$	**44**	$z \leftarrow z \oplus \gamma_3$	**69**	$v \leftarrow v + \kappa_1$
20	$z \leftarrow z \oplus \mu_2$	**45**	$z \leftarrow z \oplus r_3$	**70**	$u \leftarrow u \oplus v$
21	$u \leftarrow w \oplus r_3$	**46**	$v \leftarrow z \oplus \alpha$	**71**	$u \leftarrow u + s_2$
22	$u \leftarrow u + r_3$	**47**	$z \leftarrow x' \oplus z$	**72**	$u \leftarrow u + s_3$
23	$v \leftarrow w \oplus z$	**48**	$z \leftarrow z + v$	**73**	$z \leftarrow z + u$
24	$v \leftarrow v + z$	**49**	$z \leftarrow z \oplus u$	**74**	$z \leftarrow z - w$
25	$u \leftarrow u \oplus w$	**50**	$z \leftarrow z \oplus \mu_2$		**return** z

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

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. □

Our proof was validated by implementing a simulator and verifying that the distributions of each a_i, for $1 \leq i \leq 74$ is 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 and triplets of elements in $I \cup J \cup \{a_1, \ldots, a_{74}\}$ (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. In this case the choice of the field $\mathbb{Z}_{2^4}$ is more important as a large field size could not be tested in a reasonable amount of time.

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 SNI proofs [4].

More concretely, to conduct a search of this algorithm efficiently the only combinations where at least one element in the elements chosen from the set $I \cup J \cup \{a_1, \ldots, a_{74}\}$ is dependent on x. That is, any combination where none of the elements were computed from variables dependent on x they can be safely discarded. For elements in $\{a_1, \ldots, a_{74}\}$ the simplest known expression using elements of $I \cup J$ was taken by either following the sequence of instructions, or from the equations above. These expressions were used to generate a C source file to analyze that combination, which was then complied with the `-Ofast` flag using gcc. Thus, 1.29×10^5 C source files were automatically created and compiled. Executing the resulting binaries required 48 CPU cores running for twelve weeks.

We test our algorithm as one block because, to date, breaking up our algorithm into gadgets that can be independently verified and creating an algorithm equivalent to those we propose has a prohibitive cost in performance [8]. That is, Coron's algorithm has exponential complexity, with regard to the security order, compared to the quadratic complexity of the algorithm presented in this paper.

5 Comparison

Table 1 compares the performance of our method with previous work. We consider the work of Coron et al. [7] who proposed a high-order secure Boolean-to-arithmetic algorithm in 2014. We also discuss the recent results from Coron [8] and compare it with our results. We do not consider LUT-based methods as they would require a pre-computation phase and additional memory (see Section 2).

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

$B \rightarrow A$	Security Order							
Conversion	1	2	3	4	5	6	7	8
Goubin's method (2001) [12]	7	-	-	-	-	-	-	-
Coron et al. (2014) - 8 bits [7]	-	909	1,369	1,962	2,619	3,372	4,189	5,171
Coron et al. (2014) - 16 bits [7]	-	1,781	2,681	3,842	5,131	6,612	8,221	10,155
Coron et al. (2014) - 32 bits [7]	-	3,525	5,305	7,602	10,155	13,092	16,285	20,123
Coron et al. (2014) - 64 bits [7]	-	7,013	10,553	15,122	20,203	26,052	32,413	40,059
Hutter-Tunstall (2016) [13][a]	-	31	56	115	197	331	513	763
Coron (2017) [8]	-	-	155	367	803	1,687	N/A	7,039
Our proposal	-	**31**	**74**	**123**	**242**	**386**	**557**	**753**

[a] The original algorithms have a security flaw from the third order upwards which were corrected in this version of the paper.

For this comparison, 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 [7] as follows. For a single masked `AND` (`SecAnd`) operation [7, 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 [7, Section 3.2] to be

$$\begin{aligned}&(2{\cdot}(n+1) \cdot n + 26 + n)\\ &\quad + (k-1) \cdot [2 \cdot (n+1) \cdot n + 27] + (2 \cdot (n+1)),\end{aligned}$$

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 a higher-order Boolean-to-arithmetic mask conversion as defined in [7, 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 from [7] for all considered register widths and security orders, often by several orders of magnitude. Compared to [8], who adopts our original idea from [13], our solution is faster for any security order. Specifically, our third-order secure algorithm is more than twice as fast, and the difference increases to more than an order of magnitude by the eight order. Our second-order secure algorithm is at least one order of magnitude faster than previous work.

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

$B \rightarrow A$ Conversion	Security Order							
	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

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 (2001) [12]	1	-	-	-	-	-	-	-
Coron et al. (2014) - 8 bits [7]	-	66	127	221	331	465	615	806
Coron et al. (2014) - 16 bits [7]	-	122	239	421	635	897	1,191	1,566
Coron et al. (2014) - 32 bits [7]	-	234	463	821	1,243	1,761	2,343	3,086
Coron et al. (2014) - 64 bits [7]	-	458	911	1,621	2,459	3,489	4,647	6,126
Hutter-Tunstall (2016) [13]	-	5	11	27	44	81	120	199
Coron (2017) [8]	-	11	32	77	170	359	740	1,505
Our proposal	-	**5**	**15**	**26**	**42**	**59**	**81**	**104**

5.1 Performance details

Table 2 lists the number of required instructions for our proposed algorithms in terms of arithmetic and Boolean operations up to a security order of 8. As a reference, we also list Goubin's solution in the first-order case, and our solution in the other cases.

In Table 3, we list the number of required random variables to perform a Boolean-to-arithmetic mask conversion. Many mask-conversion proposals give an artificially low instruction count hiding a large number of required random numbers. We list the number of random values required to compute Goubin's original method [12], Coron et al.'s method [7], our original solution from 2016 [13], and Coron's recent approach [8]. We compare these numbers with the number of random values required by our proposed algorithm. For the second and third-order algorithms, we give the number of random values required by our explicit algorithms, where some random values are used to fulfill several purposes.

The results show that our solution requires significantly fewer random variables when compared to all other algorithms in the literature. In particular, when compared to [8], we require almost half the number of random values for the second-order case, and more than half the number random number is all other cases, increasing to an order of magnitude by the eighth order.

Finally, we provide a comparison of all solutions that takes into account the number of randomness required. Table 4 lists the total instruction count for

Table 4. Total operation count including the generation of random numbers using Marsaglia's Xorshift PRNG [21] from Coron's the open-source *C* implementation.

$B \rightarrow A$	Security Order							
Conversion	1	2	3	4	5	6	7	8
Coron (2017) [8]	-	110	315	752	1,653	3,482	N/A	14,564
Our proposal	-	56	149	253	452	681	962	1,273
Factor	-	**×1.9**	**×2.1**	**×2.9**	**×3.6**	**×5.1**	N/A	**×11.4**

the algorithms. We added the number of instructions listed in Table 1 with the number of instructions required to generate all random numbers that are required for the mask conversion. For this purpose, we consider a practical implementation of a Pseudo Random Number Generator (PRNG), like the Marsaglia's Xorshift RNG used by Coron [8]. We consider only the `XOR` instructions, since modern processors are often able to perform shift operations for free. We also did not consider typical overheads that are caused by function calls or loading constants from memory. Therefore we assume that generating a random number takes five instructions in the comparison given in Table 4.

By considering the cost of generating random values, we show that our proposal is about twice as fast as the next best algorithm for lower orders and increases to more than ten times faster for the eighth order case and above.

6 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 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 [3].

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 [20]. 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 [18]) or algorithm level (e.g., using secret sharing and multi-party computation such as threshold implementations [23]).

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 combined 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.

7 Conclusions

In this paper, we present Boolean-to-arithmetic mask conversion methods that can be computed in constant time for a masking scheme of second and third order. We present explicit algorithms for a second-order secure mask conversion that requires 31 instructions, and a third-order secure mask conversion that requires 74 instructions. Our second-order secure algorithm is at least one order of magnitude faster than previous work; and our corrected algorithm for third order is more than twice as fast.

Our algorithms are shown to be secure under the SNI model, although we treat our algorithms as a single gadget, rather than break it into smaller gadgets that can be independently verified. Attempts at achieving this have a large performance cost [8], i.e., exponential complexity with regard to the security order, compared to the quadratic complexity of the algorithm presented in this paper. An efficient method for that would allow one to break our algorithms into gadgets is left for future research. Also, given the resources required to validate the SNI proof for Algorithm 3, a similar proof for orders greater than three is also left for future research.

References

1. D. Agrawal, B. Archambeault, J. R. Rao, and P. Rohatgi. The EM Side-channel(s). In B. S. Kaliski Jr., Ç. K. Koç, and C. Paar, editors, *CHES 2003*, volume 2523 of *LNCS*, pages 29–45. Springer, Heidelberg, 2003.
2. J.-P. Aumasson, L. Henzen, W. Meier, and R. C.-W. Phan. SHA-3 Proposal BLAKE, December 2010. https://131002.net/blake.
3. J. Balasch, B. Gierlichs, V. Grosso, O. Reparaz, and F.-X. Standaert. On the Cost of Lazy Engineering for Masked Software Implementations. In M. Joye and A. Moradi, editors, *CARDIS 2014*, volume 8968 of *LNCS*, pages 64–81. Springer, Heidelberg, 2014.
4. G. Barthe, S. Belaïd, F. Dupressoir, P.-A. Fouque, B. Grégoire, P.-Y. Strub, and R. Zucchini. Strong non-interference and type-directed higher-order masking. In E. R. Weippl, S. Katzenbeisser, C. Kruegel, A. C. Myers, and S. Halevi, editors,

ACM Conference on Computer and Communications Security 2016, pages 116–129. Springer, Heidelberg, 2016.
5. D. J. Bernstein. Chacha, a variant of salsa20, 2008.
6. J.-S. Coron, J. Großschädl, M. Tibouchi, and P. K. Vadnala. Conversion from Arithmetic to Boolean Masking with Logarithmic Complexity. In G. Leander, editor, *FSE 2015*, volume 8731 of *LNCS*, pages 130–149. Springer, Heidelberg, 2015.
7. J.-S. Coron, J. Großschädl, and P. K. Vadnala. Secure Conversion between Boolean and Arithmetic Masking of Any Order. In L. Batina and M. Robshaw, editors, *CHES 2014*, volume 8731 of *LNCS*, pages 188–205. Springer, Heidelberg, 2014.
8. Jean-Sebastien Coron. Higher-order Conversion from Boolean to Arithmetic Masking. *IACR Cryptology ePrint Archive*, 2017:252, 2017.
9. N. Ferguson, S. Lucks, B. Schneier, D. Whiting, M. Bellare, T. Kohno, J. Callas, and J. Walker. The Skein Hash Function Family, October 2010. http://www.skein-hash.info.
10. K. Gandolfi, C. Mourtel, and F. Olivier. Electromagnetic Analysis: Concrete Results. In C. K. Koç, D. Naccache, and C. Paar, editors, *CHES 2001*, volume 2162 of *LNCS*, pages 251–261. Springer, Heidelberg, 2001.
11. J. Dj. Golić. Techniques for Random Masking in Hardware. *IEEE Transactions on Circuits and Systems*, 54(2):291–300, 2007.
12. L. Goubin. A Sound Method for Switching between Boolean and Arithmetic Masking. In Çetin Kaya Koç, David Naccache, and Christof Paar, editors, *CHES 2001*, volume 2162 of *LNCS*, pages 3–15. Springer, Heidelberg, 2001.
13. Michael Hutter and Michael Tunstall. Constant Time Higher-Order Boolean-to-Arithmetic Masking. Cryptology ePrint Archive, Report 2016/1023/20161222:183711, 2016. https://eprint.iacr.org/2016/1023/20161222:183711.
14. Y. Ishai, A. Sahai, and D. Wagner. Private Circuits: Securing Hardware against Probing Attacks. In D. Boneh, editor, *CRYPTO 2003*, volume 2729 of *LNCS*, pages 463–481. Springer, Heidelberg, 2003.
15. M. Karroumi, B. Richard, and M. Joye. Addition with Blinded Operands. In E. Prouff, editor, *COSADE 2014*, volume 8622 of *LNCS*, pages 41–55. Springer, Heidelberg, 2014.
16. P. Kocher, J. Jaffe, and B. Jun. Differential Power Analysis. In M. J. Wiener, editor, *CRYPTO '99*, volume 1666 of *LNCS*, pages 388–397. Springer, Heidelberg, 1999.
17. X. Lai and J. L. Massey. A Proposal for a New Block Encryption Standard. In I. Damgård, editor, *Workshop on the Theory and Application of Cryptographic Techniques*, volume 473 of *LNCS*, pages 389–404. Springer, Heidelberg, 1990.
18. A. J. Leiserson, M. E. Marson, and M. A. Wachs. Gate-level masking under a path-based leakage metric. In L. Batina and M. Robshaw, editors, *CHES 2014*, volume 8731 of *LNCS*, pages 580–597. Springer, Heidelberg, 2014.
19. S. Mangard, E. Oswald, and T. Popp. *Power Analysis Attacks — Revealing the Secrets of Smart Cards*. Springer, 2007.
20. S. Mangard, T. Popp, and B. M. Gammel. Side-channel leakage of masked CMOS gates. In A. Menezes, editor, *CT-RSA 2005*, volume 3376 of *LNCS*, pages 351–365. Springer, Heidelberg, 2005.
21. G. Marsaglia. Xorshift RNGs. *Journal of Statistical Software*, 8(14):1–6, 2003.
22. National Institute of Standards and Technology (NIST). FIPS-180-4: Secure Hash Standard, August 2015. http://csrc.nist.gov/publications/fips/fips180-4.

23. S. Nikova, C. Rechberger, and V. Rijmen. Threshold Implementations Against Side-Channel Attacks and Glitches. In P. Ning, S. Qing, and N. Li, editors, *ICICS 2006*, volume 4307 of *LNCS*, pages 529–545. Springer, Heidelberg, 2006.
24. S. Nikova, V. Rijmen, and M. Schläffer. Secure Hardware Implementation of Nonlinear Functions in the Presence of Glitches. *Journal of Cryptology*, 24(2):292–321, 2011.
25. J.-J. Quisquater and D. Samyde. ElectroMagnetic Analysis (EMA): Measures and Counter-Measures for Smart Cards. In I. Attali and T. P. Jensen, editors, *E-smart 2001*, volume 2140 of *LNCS*, pages 200–210. Springer, Heidelberg, 2001.
26. M. Rivain, E. Dottax, and E. Prouff. Block Ciphers Implementations Provably Secure Against Second Order Side Channel Analysis. In K. Nyberg, editor, *FSE 2008*, volume 5086 of *LNCS*, pages 127–143. Springer, Heidelberg, 2008.
27. R. L. Rivest, M.J.B. Robshaw, R. Sidney, and Y.L. Yin. The RC6 Block Cipher. ftp://ftp.rsasecurity.com/pub/rsalabs/rc6/rc6v11.pdf, August 1998.
28. T. Schneider, A. Moradi, and T. Güneysu. Arithmetic Addition over Boolean Masking—Towards First- and Second-Order Resistance in Hardware. In T. Malkin, V. Kolesnikov, A. B. Lewko, and M. Polychronakis, editors, *ACNS 2015*, volume 9092 of *LNCS*, pages 559–578. Springer, Heidelberg, 2015.
29. E. Trichina. Combinational Logic Design for AES SubByte Transformation on Masked Data. *IACR Cryptology ePrint Archive*, 2003:236, 2003.
30. P. K. Vadnala and J. Großschädl. Algorithms for Switching between Boolean and Arithmetic Masking of Second Order. In B. Gierlichs, S. Guilley, and D. Mukhopadhyay, editors, *SPACE 2013*, volume 8204 of *LNCS*, pages 95–110. Springer, Heidelberg, 2013.
31. P. K. Vadnala and J. Großschädl. Faster Mask Conversion with Lookup Tables. In S. Mangard and A. Y. Poschmann:, editors, *COSADE 2015*, volume 9064 of *LNCS*, pages 207–221. Springer, Heidelberg, 2015.