Secure Conversion Between Boolean and Arithmetic Masking of Any Order

Jean-Sébastien Coron, Johann Großschädl, and Praveen Kumar Vadnala

University of Luxembourg,
Laboratory of Algorithmics, Cryptology and Security (LACS),
{jean-sebastien.coron,johann.groszschaedl,praveen.vadnala}@uni.lu

Abstract. An effective countermeasure against side-channel attacks is to mask all sensitive intermediate variables with one (or more) random value(s). When a cryptographic algorithm involves both arithmetic and Boolean operations, it is necessary to convert from arithmetic masking to Boolean masking and vice versa. At CHES 2001, Goubin introduced two algorithms for secure conversion between arithmetic and Boolean masks, but his approach can only be applied to first-order masking. In this paper, we present and evaluate new conversion algorithms that are secure against attacks of any order. To convert masks of a size of k bits securely against attacks of order n, the proposed algorithms have a time complexity of $\mathcal{O}(n^2k)$ in both directions and are proven to be secure in the Ishai, Sahai, and Wagner (ISW) framework for private circuits. We evaluate our algorithms using HMAC-SHA-1 as example and report the execution times we achieved on a 32-bit AVR microcontroller.

Keywords: Side-channel analysis (SCA), higher-order SCA, arithmetic masking, Boolean masking, provably secure masking, HMAC-SHA-1

1 Introduction

Side-Channel Attacks. Traditionally, cryptographic algorithms are designed under the premise that a system can only be attacked in a black-box way, even though in practice this assumption is not necessarily true. An attacker may be able to obtain some partial information about the secret key(s) through means that were originally not anticipated by the system designer. A typical example are the so-called side-channel attacks, which can be mounted by measuring the power consumption [14], EM radiations [8], or execution time [13] of a cryptosystem, or by observing its response to fault injection [1]. It is widely accepted that these attacks are very powerful and can completely break a system.

Masking is a common countermeasure against side-channel attacks. Boolean masking, firstly suggested in [3, 10], consists in splitting every sensitive variable x in two shares x' and r, where $x' = x \oplus r$ and r is a randomly generated value [3]. The two shares are manipulated separately according to the cryptographic algorithm. Such modification is straightforward for linear functions, which can be computed separately on these two shares. For non-linear functions, such as

SBOXes, the usual technique consists in pre-computing a randomized SBOX in RAM for every new execution of the algorithm [3].

First-order Boolean masking is vulnerable to a second-order attack in which the adversary combines information about the two shares x' and r; such attacks are feasible in practice (see [18]). Boolean masking can actually be extended to any number of shares, e.g. when using n shares, an implementation should be resistant against t-th order attacks, in which the adversary combines leakage information from $t < n$ variables. It was shown in [3] that, under a reasonable power leakage model, the overall number of executions required to recover the secret key grows exponentially with the number of shares.

At CHES 2010, Rivain and Prouff [20] proposed an algorithm to protect the AES against t-th order attacks, based on the Ishai-Sahai-Wagner construction [11]. Their basic idea is to write the AES round transformations as operations in the field $\mathrm{GF}(2^8)$ and mask additions and multiplications. This approach can be extended to any SBOX by considering the polynomial representation of the SBOX, which can be computed using Lagrange polynomial interpolation over a finite field [2]. Rivain et al introduced in [19] a table re-computation method to protect any SBOX from second-order attacks. The classical randomized table countermeasure, secure against first-order attacks, has recently been extended to work against t-th order attacks [5].

Security Model. We definitely aim for countermeasures against side-channel attacks that can be proven secure in a reasonable model of side-channel leakage (i.e. we will not be satisfied with heuristic “ad-hoc” countermeasures). Perhaps the simplest such model is the probing attack model proposed by Ishai, Sahai and Wagner (ISW) at CRYPTO 2003 [11] (see Subsection 2.2). They initiated the theoretical study of securing circuits against an adversary who can probe its wires. In this model, the attacker is allowed to access at most t wires of the circuit, but he should not be able to learn anything about the secret key. The authors show that any circuit C can be transformed into a new circuit of size $\mathcal{O}(t^2 \cdot |C|)$ that is resistant against such an adversary. The approach is based on secret-sharing every variable x into n shares x_i with $x = x_1 \oplus x_2 \cdots \oplus x_n$, and processing the shares in a way so that no information about the initial variable x can be learned by any t-limited adversary, for $n \geq 2t + 1$.

In recent years, numerous papers on provable security against side-channel attacks have been published in the literature, forming the rapidly emerging field of leakage-resilient cryptography. Building upon the leakage model introduced by Micali and Reyzin [16] and on the bounded retrieval model [6, 7], the leakage resilience model assumes that the adversary has the ability to repeatedly learn arbitrary functions of the secret key, as long as the total number of bits leaked to the adversary is bounded by some parameter L. This is a very strong security notion because an attacker can choose arbitrary leakage functions; only the amount of leaked information is bounded. In particular, it is more general than the ISW probing model [11], in which the attacker has only access to a limited number of physical bits computed in the circuit.

However, cryptosystems proven secure in the most general leakage-resilient model are often too inefficient for practical use. In practice, one typically has to design a countermeasure against side-channel attacks for an existing algorithm (such as AES or HMAC-SHA-1) instead of devising a completely new algorithm based on the principles of leakage-resilient cryptography. The main advantage of the ISW probing model is that it can potentially lead to relatively practical designs. Another benefit is its interplay with resistance against power analysis attacks. Namely, if a given algorithm is proven resistant against t probes in the ISW model, then (at least) $t + 1$ measurements in a power acquisition must be combined to obtain the key. As shown in [3], the number of power acquisitions required to recover the key grows exponentially with t. This means that, even if a real probing attack would be physically impossible or too costly, it makes sense to obtain countermeasures with the largest possible value of t since this translates into an (exponentially in t) increasing level of security against power attacks. In this paper, we mainly work in the ISW model.

Proving the resistance of a countermeasure against a single-probe attack (or a first-order attack) is usually straightforward since it suffices to show that all intermediate variables are uniformly distributed (or, at least, that their distribution is independent from the secret key) as in this case a single probe reveals no information to the attacker. To prove resistance against t probes, one should *a priori* consider every possible t-tuple of variables and show that their joint distribution is independent from the secret key. This approach has been used to prove the security of algorithms against second-order attacks [19]. However, as the number of such t-tuples grows exponentially with t, this analysis becomes unfeasible, even for small values of t. To work around this problem, Ishai, Sahai and Wagner introduced in [11] a very practical simulation framework in which one shows how to simulate any set of t wires probed by the adversary from a subset of the input shares of the transformed circuits. Since any proper subset of these input shares can be simulated without knowledge of the input values in the original circuit, a perfect simulation of the t probed wires is possible. We follow the same approach in this paper.

Boolean vs Arithmetic Masking. Boolean masking is widely-used countermeasure for cryptographic algorithms that use only linear operations over the field $\mathbb{F}_2$ and non-linear SBOXes (e.g. DES and AES). However, if an algorithm includes arithmetic operations (such as IDEA [15], RC6 [4], and SHA-1 [17]), a masking scheme that is compatible with the arithmetic operation must be used [3]. For example, if $x_3 = x_1 + x_2$ must be computed securely, we can mask both x_1 and x_2 arithmetically by writing $x_1 = A_1 + r_1$ and $x_2 = A_2 + r_2$ for some random values r_1 and r_2. Then, instead of computing the sum x_3 directly, we can add the two shares separately, which results again in two arithmetic shares for $x_3 = (A_1 + A_2) + (r_1 + r_2)$. Note that throughout this paper all additions and subtractions are performed modulo 2^k for some k.

Besides IDEA, RC6 and SHA-1, there exist many other algorithms that execute both arithmetic (e.g. modular addition) and logical operations. Examples include ARX-based block ciphers like XTEA and Threefish, the SHA-3 finalists Blake and Skein, as well as all four stream ciphers from the e-Stream software portfolio. Hence, techniques to protect both kinds of operation are of practical importance. One approach to achieve this is to use appropriate masking (corresponding to the operation) and convert between the maskings whenever it is necessary. Of course, this requires that the mask conversion itself is also secure against first-order (resp. higher-order) attacks. Another idea is to use only one kind of masking (either Boolean or arithmetic) and employ secure algorithms to perform the needed operations directly on the shares. While there exist some papers about the first method, the second approach has, surprisingly, not been studied in detail. The decision whether to apply the conversion or not depends on the target cryptographic algorithm. For HMAC-SHA-1, the second method yields more efficient implementations, as we will show in this paper.

Our Contribution. Currently, there exists no practical conversion technique that works for masking of order two or higher. The present paper attempts to fill this gap. We introduce the first conversion algorithms between Boolean and arithmetic masking that are secure against t-th order attacks (instead of first-order only). We start with the problem of how to apply arithmetic operations directly on Boolean shares and present an algorithm for secure addition modulo 2^k with n shares (where $n \geq 2t+1$) that has a complexity of $\mathcal{O}(n^2 k)$. Then, we introduce algorithms to convert from Boolean to arithmetic masking and vice versa, again with a complexity of $\mathcal{O}(n^2 k)$ in both directions. These algorithms are proven secure in the Ishai, Sahai and Wagner (ISW) framework for private circuits [11].

We apply our countermeasures to protect HMAC-SHA-1 against second and third-order attacks. We implemented and evaluated all our masking schemes on a 32-bit AVR processor. Based on a detailed performance analysis, we identify the most efficient algorithms in practice for different levels of security.

2 Previous Work

2.1 First-order Conversion: Goubin's Algorithms

In this section, we firstly recall Goubin's algorithm for conversion from Boolean to arithmetic masking and vice versa [9]. Goubin's conversion algorithms are proven secure against first-order attacks only; thus, we restrict our attention to first-order masking. For Boolean masking, we can write $x = x' \oplus r$, where r is a randomly generated k-bit value, while for arithmetic masking, we can write $x = A + r \bmod 2^k$ (as mentioned previously, all additions and subtractions are performed modulo 2^k for some parameter k).

Boolean to Arithmetic Conversion. The Boolean to arithmetic conversion method of Goubin [9] is based on the following function $\Psi_{x'}(r) : \mathbb{F}_{2^k} \rightarrow \mathbb{F}_{2^k}$

$$\Psi_{x'}(r) = (x' \oplus r) - r$$

Theorem 1 (Goubin [9]). *The function $\Psi_{x'}(r)$ is affine over $\mathbb{F}_2$.*

Due to this affine property, the conversion from Boolean to arithmetic masking is fairly straightforward. Given x' and r so that $x = x' \oplus r$, we have to compute A so that $x = A + r$, which can be done as follows:

$$A = (x' \oplus r) - r = \Psi_{x'}(r) = \Psi_{x'}(r \oplus r_2) \oplus (\Psi_{x'}(r_2) \oplus \Psi_{x'}(0))$$

where r_2 is a random element of $\mathbb{F}_{2^k}$. This conversion method is clearly secure against first-order attacks because the left term $\Psi_{x'}(r \oplus r_2)$ is independent from r (and, therefore, independent from x), and the right term $\Psi_{x'}(r_2) \oplus \Psi_{x'}(0)$ is also independent from r and x. Note that this technique is very efficient since it requires only a constant number of operations (independent of k).

Arithmetic to Boolean Conversion. Goubin also introduced a technique to convert from arithmetic to Boolean masking, secure against first-order attacks [9]. Unfortunately, his arithmetic-to-Boolean conversion is more costly than the conversion in the other direction since its time complexity is $\mathcal{O}(k)$ for registers of k bits. It is based on the following theorem; we denote by $2x$ the multiplication of x by 2 modulo 2^k.

Theorem 2 (Goubin [9]). *If we denote $x' = (A + r) \oplus r$, we also have $x' = A \oplus u_{k-1}$, where u_{k-1} is obtained from the following recursion formula:*

$$\begin{cases} u_0 = 0 \\ \forall i \geq 0, u_{i+1} = 2[u_i \wedge (A \oplus r) \oplus (A \wedge r)] \end{cases}$$

Since this iterative computation of u_i contains only logical XOR and AND operations, it can be easily protected against first-order leakage. We refer to [9] for further details. Recently, Karroumi et al applied this method to obtain a first-order secure addition on Boolean shares directly [12].

2.2 The Ishai, Sahai and Wagner Framework

In this subsection we describe the framework of Ishai, Sahai and Wagner (ISW) [11] for proving the security against an adversary observing at most t variables within a circuit. We will use this framework in Section 4 and in Appendix A to prove the security of our conversion algorithms.

A *stateless* circuit over $\mathbb{F}_2$ can be defined as a directed acyclic graph whose sources and sinks are input and output variables, respectively, while its vertices are Boolean gates [5]. Such a stateless circuit can be augmented with random-bit gates to form a *randomized circuit*. As stated in [11], a *random-bit gate* has no input and produces as output a uniformly random bit at each new invocation of the circuit. A t-limited adversary can probe up to t wires in the circuit, and has unlimited computational power. Given a stateless circuit C, we must transform it into a new circuit C' that can resist such an adversary. However, this is only possible if the inputs and outputs of the new circuit C' are hidden since an input of C might contain some secret-key bits and by probing these bits the adversary can obtain information about the secret key. Therefore, we allow the use of a randomized *input encoder* I and *output decoder* O, whose wires can not be probed by the adversary. Both I and O should be independent from the circuit C being transformed.

Definition 1. *Let T be an efficiently computable, deterministic function mapping a stateless circuit C to a stateless circuit C', and let I, O be input and output decoder, respectively. (T, I, O) is said to be a t-private stateless transformer if it satisfies soundness and privacy, defined as follows:*

- *Soundness: C and $O \circ C' \circ I$ have identical input-output functionality.*
- *Privacy: the values of any t wires of C' can be efficiently simulated without access to any wire of C'.*

In our conversion algorithms we will often work with k-bit variables (for some fixed parameter k) instead of single bits; in this case probing one such variable will automatically reveal its k-bit value instead of a single bit. Clearly, this can only make the adversary stronger.

The ISW framework also includes definitions for stateful circuits, i.e. circuits with memory gates. As shown in [11], achieving privacy for stateful circuits is easy once privacy has been achieved in the stateless model. Thus, we focus on the stateless model in our work. We recall the main theorem from [11] below.

Theorem 3 (Ishai, Sahai, Wagner [11]). *There exists a perfectly t-private stateless transformer (T, I, O) such that T maps any stateless circuit C of size $|C|$ and depth d to a randomized stateless circuit of size $\mathcal{O}(n^2 \cdot |C|)$ and depth $\mathcal{O}(d \log t)$, where $n = 2t + 1$.*

Privacy for Stateless Circuits. For an arbitrary circuit C the corresponding circuit C' is constructed by maintaining the following invariant: for each wire in the circuit C, there are n wires in C', which add up to the value on the wire in C. Without loss of generality, any circuit C can be represented using NOT and AND gates only. Thus, if we can transform these two gates, the whole circuit is transformable. It is easy to transform a NOT gate using the following simple relation: If $x = x_1 \oplus x_2 \oplus \cdots \oplus x_n$ then $\mathrm{NOT}(x) = \mathrm{NOT}(x_1) \oplus x_2 \oplus \cdots \oplus x_n$. To transform AND gates, the authors present an elegant solution, which is shown in Algorithm 1.

3 Secure Addition on Boolean Shares

In this section, we describe algorithms that can be used to perform an addition (or a subtraction) on the Boolean shares directly, thereby eliminating the need to convert masks from one form to the other. Formally, given n Boolean shares

Algorithm 1 SecAnd

```
Input: $(x_i)$ and $(y_i)$ for $1 \le i \le n$
Output: $(z_i)$ for $1 \le i \le n$, with $\bigoplus_{i=1}^{n} z_i = \bigoplus_{i=1}^{n} x_i \wedge \bigoplus_{i=1}^{n} y_i$
1: for $i = 1$ to $n$ do
2:     for $j = i + 1$ to $n$ do
3:         $r_{i,j} \leftarrow \mathsf{rand}(1)$
4:         $r_{j,i} \leftarrow (r_{i,j} \oplus (x_i \wedge y_j)) \oplus (x_j \wedge y_i)$
5:     end for
6: end for
7: for $i = 1$ to $n$ do
8:     $z_i = x_i \wedge y_i$
9:     for $j = 1$ to $n$ do
10:        if $i \neq j$ then
11:            $z_i \leftarrow z_i \oplus r_{i,j}$
12:        end if
13:    end for
14: end for
```

of $x = x_1 \oplus \cdots \oplus x_n$ and $y = y_1 \oplus \cdots \oplus y_n$, we need to compute n Boolean shares of $z = z_1 \oplus \cdots \oplus z_n$ satisfying the relation $z = x + y$, i.e.

$$z_1 \oplus \cdots \oplus z_n = (x_1 \oplus \cdots \oplus x_n) + (y_1 \oplus \cdots \oplus y_n)$$

We propose two algorithms to solve this problem based on the ISW method.

3.1 First Variant

The first solution is obtained by transforming the k-bit addition circuit into a circuit of XOR and AND gates so that the the ISW technique can be applied directly [11]. A modular addition of two k-bit variables x and y can be defined recursively as $(x + y)^{(i)} = x^{(i)} \oplus y^{(i)} \oplus c^{(i)}$, where

$$\begin{cases} c^{(0)} = 0 \\ \forall i \ge 1, c^{(i)} = (x^{(i-1)} \wedge y^{(i-1)}) \oplus (x^{(i-1)} \wedge c^{(i-1)}) \oplus (c^{(i-1)} \wedge y^{(i-1)}) \end{cases} \tag{1}$$

Here, $x^{(i)}$ denotes the i-th bit of variable x, with $x^{(0)}$ being the least significant bit. Since this recursion formula involves solely XOR and AND operations, we can simply use the ISW approach from [11] to protect it against attacks of any order. The resulting algorithm is shown in Algorithm 2.

Initially, there will be no carry; therefore, we set all n shares of the carry to zero (Step 1). Next, we compute the carries for the remaining bits through the formula given in Equation (1). The loop runs from 0 to $k - 2$ only, since the carry from the last bit does not need to be computed in a modular addition. In Step 8 we apply an XOR operation on the two inputs x_i, y_i and the carry c_i to obtain the n shares corresponding to $x + y \bmod 2^k$. The algorithm SecAnd has a time complexity of $\mathcal{O}(n^2)$ and, as a consequence, the full algorithm has a time

Algorithm 2 SecAdd

```
Input: $(x_i)$ and $(y_i)$ for $1 \leq i \leq n$
Output: $(z_i)$ for $1 \leq i \leq n$, with $\bigoplus_{i=1}^{n} z_i = \bigoplus_{i=1}^{n} x_i + \bigoplus_{i=1}^{n} y_i$
1: $(c_i^{(0)})_{1\leq i\leq n} \leftarrow 0$                                   ▷ Initially carry is zero
2: for $j = 0$ to $k-2$ do                                                          ▷ Compute carry bit by bit
3:     $(xy_i^{(j)})_{1\leq i\leq n} \leftarrow \mathsf{SecAnd}((x_i^{(j)})_{1\leq i\leq n}, (y_i^{(j)})_{1\leq i\leq n})$     ▷ $x^{(j)} \wedge y^{(j)}$
4:     $(xc_i^{(j)})_{1\leq i\leq n} \leftarrow \mathsf{SecAnd}((x_i^{(j)})_{1\leq i\leq n}, (c_i^{(j)})_{1\leq i\leq n})$     ▷ $x^{(j)} \wedge c^{(j)}$
5:     $(yc_i^{(j)})_{1\leq i\leq n} \leftarrow \mathsf{SecAnd}((y_i^{(j)})_{1\leq i\leq n}, (c_i^{(j)})_{1\leq i\leq n})$     ▷ $y^{(j)} \wedge c^{(j)}$
6:     $(c_i^{(j+1)})_{1\leq i\leq n} \leftarrow (xy_i^{(j)})_{1\leq i\leq n} \oplus (xc_i^{(j)})_{1\leq i\leq n} \oplus (yc_i^{(j)})_{1\leq i\leq n}$
7: end for
8: $(z_i)_{1\leq i\leq n} \leftarrow (x_i)_{1\leq i\leq n} \oplus (y_i)_{1\leq i\leq n} \oplus (c_i)_{1\leq i\leq n}$     ▷ $z = x + y = x \oplus y \oplus c$
9: return $(z_i)_{1\leq i\leq n}$
```

complexity of $\mathcal{O}(n^2k)$. Algorithm 2 has to perform AND and XOR operations only. Due to the ISW scheme, we already know that such a circuit is protected from attacks of order t, where $n \geq 2t+1$. This proves the following theorem and shows the security of Algorithm 2 in the ISW model.

Theorem 4. *Let $(x_i)_{1\leq i\leq n}$ and $(y_i)_{1\leq i\leq n}$ be the input shares of Algorithm 2 and let $2t < n$. For any set of t intermediate variables, there exists a subset $I \subset [1,n]$ of indices such that $|I| \leq n-1$, whereby the shares $x_{|I}$ and $y_{|I}$ can perfectly simulate those t intermediate variables as well as the output shares $z_{|I}$.*

3.2 Second Variant

The second approach is based on the recursion from Goubin's theorem (Theorem 2), which uses the relation $x + y = x \oplus y \oplus u_{k-1}$, where u_{k-1} is obtained from the following recursion formula:

$$\begin{cases} u_0 = 0 \\ \forall i \geq 0, u_{i+1} = 2[u_i \wedge (x \oplus y) \oplus (x \wedge y)] \end{cases}$$

Algorithm 3 represents the solution based on Goubin's formula to compute the addition. Here, the function SecAnd is called with arguments of a size of k bits instead of 1-bit arguments as in Algorithm 2. In this setting, the ISW scheme has to be adapted as follows: (i) all 1-bit variables defined over $\mathbb{F}_2$ are replaced by k-bit variables defined over $\mathbb{F}_{2^k}$; (ii) the 1-bit XOR operations are replaced by k-bit XOR operations; and (iii) the 1-bit AND operations are replaced by k-bit AND operations. This extension still preserves the security of the original scheme. Note that this method has been used before in the higher-order secure masking technique for AES proposed by Rivain and Prouff [20].[1]

[1] In the Rivain-Prouff masking scheme, the AND operations over $\mathbb{F}_2$ were replaced with multiplications over $\mathbb{F}_{2^k}$ instead of AND operations over $\mathbb{F}_{2^k}$.

Algorithm 3 SecAddGoubin

```
Input: $(x_i)$ and $(y_i)$ for $1 \le i \le n$
Output: $(z_i)$ for $1 \le i \le n$, with $\bigoplus_{i=1}^{n} z_i = \bigoplus_{i=1}^{n} x_i + \bigoplus_{i=1}^{n} y_i$
 1: $(w_i)_{1\le i\le n} \leftarrow \mathsf{SecAnd}((x_i)_{1\le i\le n}, (y_i)_{1\le i\le n})$        ▷ $\omega = x \wedge y$
 2: $(u_i)_{1\le i\le n} \leftarrow 0$        ▷ Initialize shares of $u$ to zero
 3: $(a_i)_{1\le i\le n} \leftarrow (x_i)_{1\le i\le n} \oplus (y_i)_{1\le i\le n}$        ▷ $a = x \oplus y$
 4: for $j = 1$ to $k-1$ do
 5:     $(ua_i)_{1\le i\le n} \leftarrow \mathsf{SecAnd}\big((u_i)_{1\le i\le n}, (a_i)_{1\le i\le n}\big)$
 6:     $(u_i)_{1\le i\le n} \leftarrow (ua_i)_{1\le i\le n} \oplus (w_i)_{1\le i\le n}$
 7:     $(u_i)_{1\le i\le n} \leftarrow 2(u_i)_{1\le i\le n}$        ▷ $u \leftarrow 2(u \wedge a \oplus \omega)$
 8: end for
 9: $(z_i)_{1\le i\le n} \leftarrow (x_i)_{1\le i\le n} \oplus (y_i)_{1\le i\le n} \oplus (u_i)_{1\le i\le n}$        ▷ $z = x + y = x \oplus y \oplus u$
10: return $(z_i)_{1\le i\le n}$
```

The time complexity of Algorithm 3 is still $\mathcal{O}(n^2k)$. However, in practice, this algorithm will be faster for two reasons: (i) the number of calls to the function SecAnd inside the loop is reduced from three to one, and (ii) all the operations are directly performed on the k-bit variables instead of single bits, thus there is no need to perform bit manipulations. Similar to Algorithm 2, it is easy to see that the security of Algorithm 3 follows from the original ISW scheme.

Theorem 5. *Let $(x_i)_{1\le i\le n}$ and $(y_i)_{1\le i\le n}$ be the input shares of Algorithm 3 and let $2t < n$. For any set of t intermediate variables, there exists a subset $I \subset [1,n]$ of indices such that $|I| \le n-1$, whereby the shares $x_{|I}$ and $y_{|I}$ can perfectly simulate those t intermediate variables as well as the output shares $z_{|I}$.*

4 Secure Arithmetic to Boolean Masking for any Order

In this section, we describe two new algorithms for conversion from arithmetic to Boolean masking of any order. That is, given n arithmetic shares with the property $x = A_1 + \cdots + A_n$, our algorithms output the corresponding Boolean shares satisfying $x = x_1 \oplus \cdots \oplus x_n$, secure against attacks of order t, where $2t \le n-1$. We describe in Section 5 the algorithm for secure conversion in the other direction, i.e. from Boolean to arithmetic masking.

We first present a straightforward algorithm with complexity $\mathcal{O}(n^3k)$, where n and k are the number of shares and the register size, respectively. Then, we give an improved algorithm with a complexity of $\mathcal{O}(n^2k)$. Internally, both algorithms use the secure addition function we described in Section 3. Though it is more efficient in practice to perform secure addition directly on Boolean shares (due to the overhead of converting between the masks twice), such conversion algorithms may still be useful, e.g. when the required number of conversions is lower than the required number of secure additions.[2]

[2] For HMAC-SHA-1, it is more efficient to perform secure addition directly on the Boolean shares, as we will show later.

4.1 A Simple Algorithm with Complexity $\mathcal{O}(n^3k)$

We first describe a simple approach for converting from arithmetic to Boolean masking with complexity $\mathcal{O}(n^3k)$. Assume that a sensitive variable x is shared among n arithmetic masks as follows:

$$x = A_1 + \cdots + A_n \tag{2}$$

We separately re-share each of the arithmetic shares A_i $(1 \le i \le n)$ into n random Boolean shares $x_{i,j}$ $(1 \le j \le n)$ so that $A_i = x_{i,1} \oplus \cdots \oplus x_{i,n}$. Hence, the sensitive variables x is now given as:

$$x = (x_{1,1} \oplus \cdots \oplus x_{1,n}) + \cdots + (x_{n,1} \oplus \cdots \oplus x_{n,n}) \tag{3}$$

For each arithmetic share A_i $(1 \le i \le n)$, such re-sharing can be accomplished by generating $x_{i,j}$ independently at random for $2 \le j \le n$ and letting $x_{i,1} = A_i \oplus x_{i,2} \oplus \cdots \oplus x_{i,n}$. We then sequentially add the A_i's using their n-Boolean shared representation $A_i = \bigoplus_{j=1}^{n} x_{i,j}$. For this, we use either the SecAdd or the $\mathsf{SecAddGoubin}$ algorithm from Section 3. Eventually, we get the final result x in Boolean form as

$$x = z_1 \oplus \cdots \oplus z_n \tag{4}$$

Since each of the $n-1$ calls to SecAdd has a complexity of $\mathcal{O}(n^2k)$, the overall complexity of the arithmetic to Boolean conversion is $\mathcal{O}(n^3k)$.

Theorem 6. *Let $(A_i)_{1 \le i \le n}$ be the input shares of the previous algorithm and let $2t < n$. For any set of t intermediate variables, there exists a subset $I \subset [1, n]$ of indices such that $|I| \le 2t < n$, whereby the shares $A_{|I}$ can perfectly simulate those t intermediate variables as well as the output shares $z_{|I}$.*

Proof. We show how to simulate any set of t probes, for $2t < n$. We firstly consider the initial re-sharing of the arithmetic shares A_i $(1 \le i \le n)$. At first, the set I is empty. If there is a probe in the re-sharing of A_i, we add the index i to I. Then, we consider the second part of the algorithm, starting from Equation (3) to the final result given by Equation (4). This second part is essentially an iteration of a circuit obtained through the ISW transform. Therefore, by applying the ISW methodology, we can simply continue with the construction of the subset I, so that any probe in this second part, and any of the output shares $z_{|I}$, can be perfectly simulated by knowing the inputs $x_{i,j}$ for $j \in I$ and for all $1 \le i \le n$; moreover, we know from the ISW methodology that $|I| \le 2t < n$.

For any $i \notin I$, since the re-sharing of A_i is not probed, we can perfectly simulate the $x_{i,j}$ for $j \in I$ without knowing A_i. Namely, since $|I| \le 2t < n$, the $x_{i,j}$ for $j \in I$ form a proper subset of n shares, and we can perfectly simulate such a subset without knowing A_i by generating the values independently and uniformly at random. For $i \in I$, we can simulate the $x_{i,j}$ in the same way as in the "real" circuit because we know the input A_i. Therefore, as required, we can perfectly simulate the $x_{i,j}$ for $j \in I$ and all $1 \le i \le n$.

In summary, the t probes as well as the output shares $z_{|I}$ can be perfectly simulated from the knowledge of the input shares $A_{|I}$, where $|I| \le 2t < n$. □

It is easy to observe that one can improve the complexity of this algorithm by using fewer shares at the beginning. In particular, Equation (3) contains a total of n^2 shares, while only n are necessary. Therefore, at the beginning, we use only two shares for every A_i instead of n shares. Then, we build a tree where at each layer the number of additive terms is divided by two, while the number of Boolean shares within an additive term is doubled. In this way, the overall number of shares remains n or $2n$ at each level, and so the complexity becomes $\mathcal{O}(n^2k)$ instead of $\mathcal{O}(n^3k)$. We provide a complete description below.

4.2 Our New Arithmetic to Boolean Conversion Algorithm

In this section, we describe our new algorithm for converting from arithmetic to Boolean masking with a complexity of $\mathcal{O}(n^2k)$. Our algorithm is best described recursively. Assume that we already found an algorithm $\mathcal{A}_{n/2}$ for converting a set of $n/2$ arithmetic shares A_i into $n/2$ Boolean shares x_i such that

$$A_1 + \cdots + A_{n/2} = x_1 \oplus \cdots \oplus x_{n/2}.$$

Now, given as input a variable x represented with n arithmetic shares A_i:

$$x = A_1 + \cdots + A_n$$

we can first apply algorithm $\mathcal{A}_{n/2}$ separately on the two halves to get

$$\begin{aligned} x &= (A_1 + \cdots + A_{n/2}) + (A_{n/2+1} + \cdots + A_n) \\ &= \ (x_1 \oplus \cdots \oplus x_{n/2}) \ + \quad (y_1 \oplus \cdots \oplus y_{n/2}) \end{aligned}$$

We now apply a simple expansion step, in which the $n/2$ shares x_i and y_i are each expanded to n shares. This can be done by randomly splitting every share x_i into $x_i = x'_{2i-1} \oplus x'_{2i}$ and similarly for $y_i = y'_{2i-1} \oplus y'_{2i}$. We obtain:

$$x = (x'_1 \oplus \cdots \oplus x'_n) + (y'_1 \oplus \cdots \oplus y'_n)$$

Then, we apply the n-Boolean addition circuit SecAdd or SecAddGoubin from Section 3 to obtain x represented with n Boolean shares $x = z_1 \oplus \cdots \oplus z_n$ as required.

We now show that the algorithm has a complexity of $\mathcal{O}(n^2k)$. For the sake of simplicity, we assume that n is a power of two. Let T_i be the execution time of $\mathcal{A}_i$, which takes i arithmetic shares as input. We proceed by induction, based on the assumption that $T_i \leq c \cdot i^2$ for all $i \leq n/2$ and some constant c. When running algorithm $\mathcal{A}_n$ with n shares, one first applies $\mathcal{A}_{n/2}$ on both halves, and then executes the expansion step (with $3n$ steps). Finally, the SecAdd algorithm is performed, which gives:

$$T_n \leq 2T_{n/2} + 3n + c' \cdot n^2 \leq 2c \cdot (n/2)^2 + 3n + c' \cdot n^2$$

for some constant c', such that the execution time of SecAdd with n shares is $\leq c' \cdot n^2$. We get:

$$T_n \leq (c/2 + 3 + c') \cdot n^2$$

Algorithm 4 ConvertA→B

```
Input: $(A_i)$ for $1 \le i \le n$
Output: $(z_i)$ for $1 \le i \le n$, with $\bigoplus_{i=1}^{n} z_i = \sum_{i=1}^{n} A_i$
1: If $n = 1$ then return $A_1$
2: $(x_i)_{1\le i\le n/2} \leftarrow \mathsf{ConvertA{\to}B}\big((A_i)_{1\le i\le n/2}\big)$
3: $(x'_i)_{1\le i\le n} \leftarrow \mathsf{Expand}\big((x_i)_{1\le i\le n/2}\big)$    ▷ $\bigoplus_{i=1}^{n} x'_i = \bigoplus_{i=1}^{n/2} x_i = \sum_{i=1}^{n/2} A_i$
4: $(y_i)_{1\le i\le n/2} \leftarrow \mathsf{ConvertA{\to}B}\big((A_i)_{n/2+1\le i\le n}\big)$
5: $(y'_i)_{1\le i\le n} \leftarrow \mathsf{Expand}\big((y_i)_{1\le i\le n/2}\big)$    ▷ $\bigoplus_{i=1}^{n} y'_i = \bigoplus_{i=1}^{n/2} y_i = \sum_{i=n/2+1}^{n} A_i$
6: $(z_i)_{1\le i\le n} \leftarrow \mathsf{SecAdd}\big((x'_i)_{1\le i\le n},\ (y'_i)_{1\le i\le n}\big)$
7: return $(z_i)_{1\le i\le n}$    ▷ $\bigoplus_{i=1}^{n} z_i = \bigoplus_{i=1}^{n} x'_i + \bigoplus_{i=1}^{n} y'_i = \sum_{i=1}^{n} A_i$
```

Algorithm 5 Expand

```
Input: $x_i$ for $1 \le i \le n$
Output: $y_i$ for $1 \le i \le 2n$ with $\bigoplus_{i=1}^{2n} y_i = \bigoplus_{i=1}^{n} x_i$
1: $(r_i)_{1\le i\le n} \leftarrow \mathsf{Rand}(k)$
2: $(y_{2i})_{1\le i\le n} \leftarrow (x_i \oplus r_i)_{1\le i\le n}$
3: $(y_{2i+1})_{1\le i\le n} \leftarrow (r_i)_{1\le i\le n}$
4: return $(y_i)_{1\le i\le 2n}$
```

Hence, it suffices to fix the constant c so that $3 + c' \le c/2$ to get $T_n \le c \cdot n^2$ as required to prove the result. A formal description of our new conversion method can be found in Algorithm 4, which, in turn, uses the expansion step specified in Algorithm 5. The following theorem confirms that Algorithm 4 is secure in the ISW framework.

Theorem 7. *Let $(A_i)_{1\le i\le n}$ be the input shares of Algorithm 4. For any set of t intermediate variables and any k output shares, there exists a subset $I \subset [1, n]$ of indices such that $|I| \le k + 2t$, where the shares $A_{|I}$ can perfectly simulate those t intermediate variables as well as the output shares $x_{|I}$.*

Proof. We first prove the following property of the Expand method.

Lemma 1. *In Algorithm 5, a set of k outputs ($k \le 2n$) and t probes ($t \le n$) can be perfectly simulated using at most $\lfloor k/2 \rfloor + t$ inputs.*

Proof of Lemma 1. We proceed by induction. When $n = 1$, the algorithm gets only x as input and outputs $(x \oplus r, r)$ for a uniformly random r. Now, we have to distinguish between the following two cases: there is no probe ($t = 0$), and there is at least one probe ($t \ge 1$).

In the latter case, i.e. there is at least one probe (for x, or r, or $x \oplus r$), then $t \ge 1$ and the probe can be perfectly simulated by using the input x and generating r uniformly at random. This will also perfectly simulate both outputs. As a consequence, for $t = 1$ and any k with $0 \leq k \leq 2$, we can perfectly simulate the t probes and the k outputs using at most $1 \leq \lfloor k/2 \rfloor + t$ inputs.

We now assume that there are no probes ($t = 0$). If no output needs to be simulated (i.e. $k = 0$), then knowledge of the input x is not required. If only a single output must be simulated ($k = 1$), where either $y_1 = x \oplus r$ or $y_2 = r$ has to be simulated, such output can be perfectly simulated by generating a random number uniformly, without knowing x. Finally, if $k = 2$, then one input is required. Therefore, for any k with $0 \leq k \leq 2$, the number of required inputs is always at most $\lfloor k/2 \rfloor + t$.

For $n > 1$, let us consider the i-th sub-circuit and denote the number of outputs to be simulated by k_i and the number of probes by t_i for $1 \leq i \leq n$. Based on the above arguments, the total number of inputs needed for the simulation is then at most

$$\sum_{i=1}^{n} \lfloor k_i/2 \rfloor + t_i \leq \lfloor k/2 \rfloor + t,$$

which finally proves the Lemma. □

The proof of Theorem 7 is obtained via induction on the number of shares n. We assume that the result holds for $n/2$ and prove that it holds for n. We distinguish among 5 sets of probes:

- The t_A probes for the Secure Addition subroutine (Line 6 of Algorithm 4).
- The t_{EL} and t_{ER} probes for the left and right Expand circuit, respectively (lines 3 and 5 of Algorithm 4).
- The t_{CL} and t_{CR} probes for the left and right Arithmetic to Boolean conversion circuit, respectively (lines 2 and 4 of Algorithm 4).

From the security proof of the SecAnd algorithm given in [11], we know that a set of k outputs and t_A probes can be simulated using a subset of $k + 2t_A$ inputs in each of the two input shares x'_i and y'_i. Therefore, the property also holds for the SecAdd algorithm.

According to Lemma 1, a set of $k + 2t_A$ outputs and t_{EL} (resp. t_{ER}) probes can be simulated using at most $\lfloor (k + 2t_A)/2 \rfloor + t_{EL} = \lfloor k/2 \rfloor + t_A + t_{EL}$ inputs (resp. $\lfloor k/2 \rfloor + t_A + t_{ER}$ inputs). Since the result is assumed to hold for $n/2$, the $\lfloor k/2 \rfloor + t_A + t_{EL}$ outputs and the t_{CL} probes of the left conversion can be simulated using at most $\lfloor k/2 \rfloor + t_A + t_{EL} + 2t_{CL}$ inputs. An upper bound of the number of inputs for the right conversion can be derived in the same way. As a consequence, the total number of required inputs is at most $k + 2t$ according to the following equation

$$\begin{aligned}|I| &\leq \lfloor k/2 \rfloor + t_A + t_{EL} + 2t_{CL} + \lfloor k/2 \rfloor + t_A + t_{ER} + 2t_{CR} \\ &\leq k + 2(t_A + t_{EL} + t_{ER} + t_{CL} + t_{CR}) \\ &\leq k + 2t,\end{aligned}$$

which proves Theorem 7. □

5 From Boolean to Arithmetic Masking of Any Order

We now present a new algorithm for converting in the other direction, i.e. from Boolean to arithmetic masking, again with a complexity of $\mathcal{O}(n^2k)$. Algorithm 6 specifies our arithmetic-to-Boolean conversion in detail.

Algorithm 6 Conversion from Boolean to Arithmetic Masking

```
Input: $(x_i)$ for $1 \le i \le n$
Output: $(A_i)$ for $1 \le i \le n$, with $\sum_{i=1}^{n} A_i = \bigoplus_{i=1}^{n} x_i$
1: $(A_i)_{1 \le i \le n-1} \leftarrow \mathsf{Rand}(k)$
2: $(A'_i)_{1 \le i \le n-1} \leftarrow (-A_i)_{1 \le i \le n-1},\ A'_n \leftarrow 0$
3: $(y_i)_{1 \le i \le n} \leftarrow \mathsf{ConvertA{\to}B}\big((A'_i)_{1 \le i \le n}\big)$        $\triangleright \bigoplus_{i=1}^{n} y_i = \sum_{i=1}^{n} A'_i = -\sum_{i=1}^{n-1} A_i$
4: $(z_i)_{1 \le i \le n} \leftarrow \mathsf{SecAdd}\big((x_i)_{1 \le i \le n}, (y_i)_{1 \le i \le n}\big)$        $\triangleright \bigoplus_{i=1}^{n} z_i = \bigoplus_{i=1}^{n} x_i + \bigoplus_{i=1}^{n} y_i$
5: $A_n \leftarrow \mathsf{FullXor}\big((z_i)_{1 \le i \le n}\big)$        $\triangleright A_n = \bigoplus_{i=1}^{n} z_i = \bigoplus_{i=1}^{n} x_i - \sum_{i=1}^{n-1} A_i$
6: return $(A_i)_{1 \le i \le n}$.        $\triangleright \sum_{i=1}^{n} A_i = \bigoplus_{i=1}^{n} x_i$
```

We use the same randomized XOR method as in [5] to compute $A_n \leftarrow \bigoplus_{i=1}^{n} z_i$; we recall this method in Algorithm 7. The randomized XOR method, in turn, uses Algorithm 8 (which was first proposed by Rivain and Prouff [20]) to refresh the masks.

Algorithm 7 FullXor

```
Input: $y_1, \dots, y_n$
Output: $y$ such that $y = y_1 \oplus \cdots \oplus y_n$
1: for $i = 1$ to $n$ do $(y_1, \dots, y_n) \leftarrow \mathsf{RefreshMasks}(y_1, \dots, y_n)$
2: return $y_1 \oplus \cdots \oplus y_n$
```

Algorithm 8 RefreshMasks

```
Input: $z_1, \dots, z_n$ such that $z = z_1 \oplus \cdots \oplus z_n$
Output: $z_1, \dots, z_n$ such that $z = z_1 \oplus \cdots \oplus z_n$
1: for $j = 2$ to $n$ do
2:     $tmp \leftarrow \mathsf{Rand}(k)$
3:     $z_1 \leftarrow z_1 \oplus tmp$
4:     $z_j \leftarrow z_j \oplus tmp$
5: end for
6: return $z_1, \dots, z_n$
```

The following theorem proves the security of Algorithm 6 in the ISW model; the proof is provided in Appendix A.

Theorem 8. *Let $(x_i)_{1 \le i \le n}$ be the input shares of Algorithm 6. For any set of t intermediate variables with $2t < n$, there exists a subset $I \subset [1, n]$ of indices such that $|I| \le 2t$, whereby the shares $x_{|I}$ can perfectly simulate those t intermediate variables as well as the output shares $A_{|I}$.*

6 Implementation Results

We have implemented all the solutions proposed in this paper on a 32-bit AVR microcontroller for security level $t = 2, 3$. We then applied all these techniques to HMAC-SHA-1 and compared the running time with respect to an unmasked implementation. Table 1 gives the running time of the addition and conversion algorithms along with the number of calls to the rand function for security level $t = 2, 3$. As expected, the addition algorithms using Goubin's theorem (i.e. the second variant presented in Section 3.2) outperform the first variant (given in Section 3.1). Therefore, we applied the second variant to implement the secure conversion algorithms.

Table 1. Execution times of all algorithms (in thousands of clock cycles) for $t = 2, 3$ and the number of calls to the rand function

Algorithm	Time	rand
second-order addition		
Algorithm 2	87	1240
Algorithm 3	26	320
second-order conversion		
Algorithm 4	54	484
Algorithm 6	81	822
third-order addition		
Algorithm 2	156	2604
Algorithm 3	46	672
third-order conversion		
Algorithm 4	121	1288
Algorithm 6	162	1997

HMAC-SHA-1. The hash function SHA-1 operates on blocks of 512 bits and produces a 160-bit message digest. Each message block is divided into 16 words of 32-bits each, which are extended to produce 64 further words (i.e. the total number of words is 80). The main loop contains 80 iterations corresponding to each of these 80 words. In order to protect HMAC-SHA-1 against side-channel attacks, we follow two different approaches, which are summarized below.

In the first approach, we use Boolean masking and perform secure addition on Boolean shares directly whenever required. Every iteration of the main loop requires four 32-bit additions, which amounts in a total of 320 additions for 80 iterations. Moreover, five additions have to be performed at the end to update the state. So, in total, 325 secure additions need to be carried out per message block.

In the other approach, we use Boolean masking and convert it to arithmetic masking wherever necessary. In this case, we need four Boolean to arithmetic conversions and one arithmetic to Boolean conversion per iteration, yielding a total of 400 conversions for 80 iterations. Additionally, we need 10 conversions to update the result, i.e. a total of 410 conversions per block are required. The execution times of both approaches are summarized in Table 2.

Table 2. Execution times of second and third-order secure masking (in thousands of clock cycles) and performance penalty compared to an unmasked implementation of HMAC-SHA-1

Algorithm	Time	Penalty
HMAC-SHA-1	104	1
second-order addition		
Algorithm 2	57172	549
Algorithm 3	17847	171
second-order conversion		
Algorithm 4, 6	62669	602
third-order addition		
Algorithm 2	106292	987
Algorithm 3	31195	299
third-order conversion		
Algorithm 4, 6	127348	1224

7 Conclusions

In this paper, we addressed the problem of secure conversion between Boolean and arithmetic masking for any order. By applying the ISW framework and Goubin's results for first-order conversion, we developed two algorithms of the same asymptotic complexity to securely add Boolean shares. We then described novel conversion algorithms between Boolean and arithmetic masking that are provably secure at any order. Practical experiments based on HMAC-SHA-1 as case study show that, in the case of second and third-order security, using Boolean masking and performing secure addition on Boolean shares directly is more efficient than converting between Boolean and arithmetic masking. Even though the proposed algorithms entail a massive performance penalty, they can still be practically useful for applications like challenge-response authentication where only a single block of data needs to be encrypted.

References

1. D. Boneh, R. A. DeMillo, and R. J. Lipton. On the importance of checking cryptographic protocols for faults (extended abstract). In *EUROCRYPT*, pages 37–51, 1997.
2. C. Carlet, L. Goubin, E. Prouff, M. Quisquater, and M. Rivain. Higher-order masking schemes for S-Boxes. In *FSE*, pages 366–384, 2012.
3. S. Chari, C. S. Jutla, J. R. Rao, and P. Rohatgi. Towards sound approaches to counteract power-analysis attacks. In *CRYPTO*, 1999.
4. S. Contini, R. L. Rivest, M. J. B. Robshaw, and Y. L. Yin. Improved analysis of some simplified variants of RC6. In *FSE*, 1999.
5. J.-S. Coron. Higher order masking of look-up tables. In *EUROCRYPT*, pages 441–458, 2014.
6. G. D. Crescenzo, R. J. Lipton, and S. Walfish. Perfectly secure password protocols in the bounded retrieval model. In *TCC*, pages 225–244, 2006.
7. S. Dziembowski. Intrusion-resilience via the bounded-storage model. In *TCC*, pages 207–224, 2006.
8. K. Gandolfi, C. Mourtel, and F. Olivier. Electromagnetic analysis: Concrete results. In *CHES*, pages 251–261, 2001.
9. L. Goubin. A sound method for switching between Boolean and arithmetic masking. In *CHES*, pages 3–15, 2001.
10. L. Goubin and J. Patarin. DES and differential power analysis (the "duplication" method). In *CHES*, pages 158–172, 1999.
11. Y. Ishai, A. Sahai, and D. Wagner. Private circuits: Securing hardware against probing attacks. In *CRYPTO*, pages 463–481, 2003.
12. M. Karroumi, B. Richard, and M. Joye. Addition with blinded operands. In *COSADE*, 2014.
13. P. C. Kocher. Timing attacks on implementations of Diffie-Hellman, RSA, DSS, and other systems. In *CRYPTO*, pages 104–113, 1996.
14. P. C. Kocher, J. Jaffe, and B. Jun. Differential power analysis. In *CRYPTO*, pages 388–397, 1999.
15. X. Lai and J. L. Massey. A proposal for a new block encryption standard. In *EUROCRYPT*, pages 389–404, 1990.
16. S. Micali and L. Reyzin. Physically observable cryptography (extended abstract). In *TCC*, pages 278–296, 2004.
17. NIST. Secure hash standard. In *Federal Information Processing Standard, FIPA-180-1*, 1995.
18. E. Oswald, S. Mangard, C. Herbst, and S. Tillich. Practical second-order DPA attacks for masked smart card implementations of block ciphers. In *CT-RSA*, pages 192–207, 2006.
19. M. Rivain, E. Dottax, and E. Prouff. Block ciphers implementations provably secure against second order side channel analysis. In *FSE*, pages 127–143, 2008.
20. M. Rivain and E. Prouff. Provably secure higher-order masking of AES. In *CHES*, pages 413–427, 2010.

A Proof of Theorem 8

We recall the following Lemma from [5] (with $|I| \leq t$ instead of $|I| \leq 2t$) and its proof.

Lemma 2. *Let $(y_i)_{1 \le i \le n}$ be the input shares of the* FullXor *algorithm. For any set of t intermediate variables, there exists a subset $I \subset [1, n]$ of indices such that $|I| \le t$ and the distribution of those t variables can be perfectly simulated from $y_{|I}$ and $y = y_1 \oplus \cdots \oplus y_n$.*

Proof of Lemma 2. We first consider the series of n RefreshMasks. If any variable y_j is probed inside any of the RefreshMasks, we add j to I.

Moreover since $t < n$, there must be at least one RefreshMasks that is not probed at all; let i^* be the index of this RefreshMasks. Since we know $y = y_1 \oplus \cdots \oplus y_n$, we can perfectly simulate *all* the shares $(y_i)_{1 \le i \le n}$ after this i^*-th RefreshMasks. Therefore we can perfectly simulate all y_i's until the last RefreshMasks, and all intermediate variables for computing $y = y_1 \oplus \cdots \oplus y_n$.

In summary before the i^* RefreshMasks, with the knowledge of the input shares $y_{|I}$, we can perfectly simulate all intermediate variables y_j for $j \in I$, and after the i^* RefreshMasks we can perfectly simulate all intermediate variables. Finally the *tmp* variables are simulated as in the real circuit. This proves Lemma 2. □

From Lemma 2, the set of t_1 probes in the FullXor circuit computing $A_n = \bigoplus_{i=1}^{n} z_i$ can be simulated from A_n and at most t_1 inputs z_i. From the previous lemmas, those t_1 inputs z_i and the t_2 probes in the remaining circuit can be perfectly simulated using $x_{|I}$, for $I \subset [1, n]$, where $|I| \le t_1 + 2t_2$. If $t_1 > 0$ we add n to I; we still have $|I| \le 2t$ where $t = t_1 + t_2$.

It remains to show how we can simulate A_n, as this is required for the simulation in Lemma 2 if $t_1 > 0$, or if $t_1 = 0$ and $n \in I$, since we must simulate all outputs $A_{|I}$. We select an arbitrary $i_0 \notin I$ such that $i_0 \neq n$; this is possible since in both cases we have $n \in I$ and $|I| \le 2t < n$. We have:

$$A_n = \left(x - \sum_{\substack{i=1 \\ i \neq i_0}}^{n-1} A_i \right) - A_{i_0}$$

Since $i_0 \notin I$ the variable A_{i_0} does not enter in any computation of the simulation. Since in the real circuit A_{i_0} is generated uniformly at random, we can simulate A_n by generating a uniform random value. This proves Theorem 8.