Higher Order Masking of Look-up Tables

Jean-Sebastien Coron

University of Luxembourg

October 26, 2013

Abstract. We describe a new algorithm for masking look-up tables of block-ciphers at any order, as a countermeasure against side-channel attacks. Our technique is a generalization of the classical randomized table countermeasure against first-order attacks. We prove the security of our new algorithm against t-th order attacks in the usual Ishai-Sahai-Wagner model from Crypto 2003; we also improve the bound on the number of shares from $n \geq 4t+1$ to $n \geq 2t+1$ for an adversary who can adaptively move its probes between successive executions. Our algorithm has the same time complexity $\mathcal{O}(n^2)$ as the Rivain-Prouff algorithm for AES, and its extension by Carlet *et al.* to any look-up table. In practice for AES our algorithm is less efficient than Rivain-Prouff, which can take advantage of the special algebraic structure of the AES Sbox; however for DES our algorithm performs slightly better.

1 Introduction

Side-Channel Attacks. An implementation of a cryptographic algorithm on some concrete device, such as a PC or a smart-card, can leak additional information to an attacker through the device power consumption or electro-magnetic emanations, enabling efficient key-recovery attacks. One of the most powerful attack is the Differential Power Analysis (DPA) [KJJ99]; it consists in recovering the secret-key by performing a statistical analysis of the power consumption of the electronic device, for several executions of a cryptographic algorithm. Another powerful class of attack are template attacks [CRR02]; a template is a precise model for the noise and expected signal for all possible values of part of the key; the attack is then carried out iteratively to recover successive parts of the key.

Random Masking. A well-known countermeasure against side-channel attacks consists in masking all internal variables with a random r, as first suggested in [CJRR99]. Any internal variable x is first masked by computing $x' = x \oplus r$, and the masked variable x' and the mask r are then processed separately. An attacker trying to analyze the power consumption at a single point will obtain only random values; therefore, the implementation will be secure against first-order DPA. However, a first-order masking can be broken in practice by a second-order side channel attack, in which the attacker combines information from two leakage points [Mes00]; however such attack usually requires a larger number of power consumption curves, which can be unfeasible in practice if the number of executions is limited (for example, by using a counter). For AES many countermeasures based on random masking have been described, see for example [HOM06].

More generally, one can split any variable x into n boolean shares by letting $x = x_1 \oplus \cdots \oplus x_n$ as in a secret-sharing scheme [Sha79]. The shares x_i must then be processed separately without leaking information about the original variable x. Most block-ciphers (such as AES or DES) alternate several rounds, each containing one linear transformation (or more), and a non-linear transformation. A linear function $y = f(x)$ is easy to compute when x is shared as $x = x_1 \oplus \cdots \oplus x_n$, as it suffices to compute $y_i = f(x_i)$ separately for every i. However securely computing a non-linear function $y = S(x)$ with shares is more difficult and is the subject of this paper.

The Ishai-Sahai-Wagner Private Circuit. The theoretical study of securing circuits against an adversary who can probe its wires was initiated by Ishai, Sahai and Wagner in [ISW03]. The goal is to protect a cryptographic implementation against side-channel attacks in a provable way. The authors consider an adversary who can probe at most t wires of the circuit. They showed how to transform any boolean circuit C of size $|C|$ into a circuit of size $\mathcal{O}(|C| \cdot t^2)$ that is perfectly secure against such adversary.

The Ishai-Sahai-Wagner (ISW) model is relevant even in the context of power attacks. Namely the number of probes in the circuit corresponds to the attack order in a high-order DPA. More precisely, if a circuit is perfectly secure against t probes, then combining t power consumption points as in a t-th order DPA will reveal no information to the adversary. To obtain useful information about the key the adversary will have to perform an attack of order at least $t+1$. The soundness of higher-order masking in the context of power attacks was first demonstrated by Chari *et al.* in [CJRR99], who showed that in a realistic leakage model the number of acquisitions to recover the key grows exponentially with the number of shares. Their analysis was recently extended by Prouff and Rivain in [PR13]. The authors proved that the information obtained by observing the *entire* leakage of an execution (instead the leakage of the n shares of a given variable) can be made negligible in the masking order. This shows that the number of shares n is a sound security parameter for protecting an implementation against side-channel attacks.

To protect against an adversary with at most t probes, the ISW approach consists in secret-sharing every variable x into n shares x_i where $n = 2t+1$, that is $x = x_1 \oplus x_2 \oplus \cdots \oplus x_n$ where $x_2, \ldots, x_n$ are uniformly and independently distributed bits. An adversary probing at most $n-1$ variables clearly does not learn any information about x. Processing a NOT gate is straightforward since $\bar{x} = \bar{x}_1 \oplus x_2 \oplus \cdots \oplus x_n$; therefore it suffices to invert the first share x_1. To process an AND gate $z = xy$, one writes:

$$z = xy = \left(\bigoplus_{i=1}^{n} x_i\right) \cdot \left(\bigoplus_{i=1}^{n} y_i\right) = \bigoplus_{1 \le i,j \le n} x_i y_j \qquad (1)$$

and the cross-products $x_i y_j$ are processed and recombined without leaking information about the original inputs x and y. More precisely for each $1 \le i < j \le n$ one generates random bits $r_{i,j}$ and computes $r_{j,i} = (r_{i,j} \oplus x_i y_j) \oplus x_j y_i$; the n shares z_i of $z = xy$ are then computed as $z_i = x_i y_i \oplus \bigoplus_{j \ne i} r_{i,j}$. Since there are n^2 such cross-products, every AND gate of the circuit is expanded to $\mathcal{O}(n^2) = \mathcal{O}(t^2)$ new gates in the circuit.

The authors also describe a very convenient framework for proving the security against any set of t probes. Namely proving the security of a countermeasure against first-order attacks ($t = 1$) is usually straightforward, as it suffices to check that every internal variable has the uniform distribution (or at least a distribution independent from the secret-key). Such approach can be extended to second-order attacks by considering pairs of internal variables (as in [RDP08]); however it becomes clearly unfeasible for larger values of t, as the number of t-uples to consider would grow exponentially with t. Alternatively the ISW framework is simulation based: the authors prove the security of their construction against a adversary with at most t probes by showing that any set of t probes can be perfectly simulated without the knowledge of the original input variables (such as x, y in the AND gate $z = xy$). In [ISW03] this is done by iteratively generating a subset I of indices of the input shares that are sufficient to simulate the t probes; then if $|I| < n$ the corresponding input shares can be perfectly simulated without knowing the original input variable, simply by generating independently and uniformly distributed bits. In the ISW construction every probe adds at most two indices in I, so we get $|I| \le 2t$ and therefore $n \ge 2t+1$ is sufficient to achieve perfect secrecy against a t-limited adversary. A nice property of the ISW framework is that the technique easily extends from a single gate to the full circuit: it suffices to maintain a global subset of indices I that is iteratively constructed from the t probes as in a single gate.

The Rivain-Prouff Countermeasure. The Rivain-Prouff countermeasure [RP10] was the first provably secure higher-order masking scheme for the AES block-cipher. Namely, all previous masking schemes were secure against first-order or second-order attacks only. The classical randomized table countermeasure [CJRR99] is secure against first-order attacks only. The Schramm and Paar countermeasure [SP06] was designed to be secure at any order n, but an attack of order 3 was shown in [CPR07]. An alternative countermeasure based on table recomputation and provably secure against second-order attacks was described in [RDP08], but no extension to any order is known. The Rivain-Prouff countermeasure was therefore the first masking scheme for AES secure for any order $t \ge 3$.

The Rivain-Prouff countermeasure is an adaptation of the previous ISW construction to software implementations, working in the AES finite field $\mathbb{F}_{2^8}$ instead of $\mathbb{F}_2$. Namely the non-linear part of the AES

Sbox can be written as $S(x) = x^{254}$ over $\mathbb{F}_{2^8}$, and as shown in [RP10] such monomial can be evaluated with only 4 non-linear multiplications (and a few linear squarings). These 4 multiplications can be evaluated with n-shared input using the previous technique based on Equation (1), by working over the field $\mathbb{F}_{2^8}$ instead of $\mathbb{F}_2$. In order to achieve resistance against an attack of order t, the Rivain-Prouff algorithm also requires at least $n = 2t + 1$ shares (see [CPRR13] for a more detailed analysis).

The Rivain-Prouff countermeasure was later extended by Carlet *et al.* to any look-up table [CGP$^+$12]. Namely using Lagrange interpolation any Sbox with k-bit input can be written as a polynomial

$$S(x) = \sum_{i=0}^{2^k-1} \alpha_i \cdot x^i$$

over $\mathbb{F}_{2^k}$, for constant coefficients $\alpha_i \in \mathbb{F}_{2^k}$. The polynomial can then be evaluated with n-shared multiplications as in the Rivain-Prouff countermeasure. The authors of [CGP$^+$12] describe two techniques for optimizing the evaluation of $S(x)$ by minimizing the number of non-linear multiplications: the cyclotomic method and the parity-split method; the later method is asymptotically faster and requires $\mathcal{O}(2^{k/2})$ multiplications. Therefore the Carlet *et al.* countermeasure with n shares has time complexity $\mathcal{O}(2^{k/2} \cdot n^2)$, where $n \geq 2t + 1$ to ensure resistance against t-th order attacks.

Extending the randomized table countermeasure. Our new countermeasure is completely different from the Rivain-Prouff countermeasure and its extension by Carlet *et al.*. Namely it is essentially based on table recomputations and does not use multiplications over $\mathbb{F}_{2^k}$. To illustrate our technique we start with the classical randomized table countermeasure, secure against first order attacks only, as first suggested in [CJRR99]. The Sbox table $S(u)$ with k-bit input is first randomized in RAM by letting

$$T(u) = S(u \oplus r) \oplus s$$

for all $u \in \{0,1\}^k$, where $r \in \{0,1\}^k$ is the input mask and $s \in \{0,1\}^k$ is the output mask.[1] To evaluate $S(x)$ from the masked value $x' = x \oplus r$, it suffices to compute $y' = T(x')$, as we get $y' = T(x') = S(x' \oplus r) \oplus s = S(x) \oplus s$; this shows that y' is indeed a masked value for $S(x)$. In other words the randomized table countermeasure consists in first re-computing in RAM a temporary table with inputs shifted by r and with masked outputs, so that later it can be evaluated on a masked value $x' = x \oplus r$ to obtain a masked output.

A natural generalization at any order n would be as follows: given as input $x = x_1 \oplus \cdots \oplus x_n$ we would start with a randomized table with inputs shifted by x_1 only, and with $n-1$ output masks; then we would incrementally shift the full table by x_2 and so on until x_{n-1}, at which point the table could be evaluated at x_n. More precisely one would initially define the randomized table

$$T(u) = S(u \oplus x_1) \oplus s_2 \oplus \cdots \oplus s_n$$

where $s_2, \ldots, s_n$ are the output masks, and then progressively shift the randomized table by letting $T(u) \leftarrow T(u \oplus x_i)$ for all u, iteratively from x_2 until x_{n-1}. Eventually the table would have all its inputs shifted by $x_1 \oplus \cdots \oplus x_{n-1}$, so as previously one could evaluate $y' = T(x_n)$ and obtain $S(x)$ masked by $s_2, \ldots, s_n$.

What we have described above is essentially the Schramm and Paar countermeasure [SP06]. However as shown in [CPR07] this is insecure. Namely consider the table $T(u)$ after the last shift by x_{n-1}; at this point we have $T(u) = S(u \oplus x_1 \oplus \cdots \oplus x_{n-1}) \oplus s_2 \oplus \cdots \oplus s_n$ for all u. Now assume that we can probe $T(0)$ and $T(1)$; we can then compute $T(0) \oplus T(1) = S(x_1 \oplus \cdots \oplus x_{n-1}) \oplus S(1 \oplus x_1 \oplus \cdots \oplus x_{n-1})$, which only depends on $x_1 \oplus \cdots \oplus x_{n-1}$; therefore it suffices to additionally probe x_n to leak information about $x = x_1 \oplus \cdots \oplus x_{n-1} \oplus x_n$; this gives an attack of order 3 only for any value of n; therefore the countermeasure can only be secure against second-order attacks.

The main issue with the previous countermeasure is that the *same* masks $s_2, \ldots, s_n$ were used to mask all the inputs of $S(u)$, so one can exclusive-or any two lines of the randomized table and remove all the

[1]One can also take $s = r$. For simplicity we first assume that the Sbox has both k-bit input and k-bit output.

output masks. A natural fix is to use *different* masks for every $S(u)$, so one would write initially:

$$T(u) = S(u \oplus x_1) \oplus s_{u,2} \oplus \cdots \oplus s_{u,n}$$

for all $u \in \{0,1\}^k$, and as previously one would iteratively shift the table by $x_2, \ldots, x_{n-1}$, and also the masks $s_{u,i}$ separately for each i. The previous attack is thwarted because the lines of $S(u)$ are now masked with different set of masks. Eventually one would read $T(x_n)$, which would give $S(x)$ masked by $s_{x_n,2}, \ldots, s_{x_n,n}$.

Our new Countermeasure. Our new countermeasure is based on using independent masks as above, with additionally a refresh of the masks between every successive shifts of the input. Since the above output masks $s_{u,j}$ are now different for all lines u of the table, we actually have a set of n randomized tables, as opposed to a single randomized table in the original Schramm and Paar countermeasure. Perhaps more conveniently one can view every line u of our randomized table as a n-dimensional vector of elements in $\{0,1\}^k$, and write for all inputs $u \in \{0,1\}^k$:

$$T(u) = (s_{u,1},\, s_{u,2}, \ldots,\, s_{u,n})$$

where initially each vector $T(u)$ is a n-boolean sharing of the value $S(u \oplus x_1)$. The vectors $T(u)$ of our randomized table are then progressively shifted for all $u \in \{0,1\}^k$, first by x_2 and so on until x_{n-1}, as in the original Schramm and Paar countermeasure. Eventually the evaluation of $T(x_n)$ gives a vector of n output shares that corresponds to $S(x)$.

To refresh the masks between successive shifts we can generate a random n-sharing of 0, that is $a_1, \ldots, a_n \in \{0,1\}^k$ such that $a_1 \oplus \cdots \oplus a_n = 0$ and we xor the vector $T(u)$ with $(a_1, \ldots, a_n)$, independently for every u. More concretely one can use the RefreshMasks procedure from [RP10], which consists given $y = y_1 \oplus y_2 \oplus \cdots \oplus y_n$ in xoring both y_1 and y_i with $tmp \leftarrow \{0,1\}^k$, iteratively from $i = 2$ to n. In summary our new countermeasure is essentially the Schramm and Paar countermeasure with independent output masks for every line of the SBOX table, and with mask refreshing after every shift of the table; we provide a full description in Section 3.1.[2]

We show that our new countermeasure is secure against any attack of order t in the ISW model, with at least $n = 2t + 1$ shares. The proof works as follows. Assume that there are at most $n - 3$ probes; then it must be the case that at least one of the $n - 2$ shifts of the table by x_i and subsequent mask refreshings are not probed at all. Since the corresponding mask refreshings are not probed, we can perfectly simulate any subset of $n - 1$ shares at the output of those mask refreshings. Therefore we can perfectly simulate all the internal variables up to the x_{i-1} shift by knowing $x_1, \ldots, x_{i-1}$, and any subset of $n - 1$ shares after the x_i shift by knowing $x_{i+1}, \ldots, x_n$. Since the knowledge of x_i is not needed in the simulation, the full simulation can be performed without knowing the original input x, which proves the security of our countermeasure.[3]

Note that it does not matter how the mask refreshing is performed; the only required property is that after a (non-probed) mask refreshing any subset of $n - 1$ shares among the n shares have independent and uniform distribution; such property is clearly satisfied by the RefreshMasks procedure from [RP10] recalled above. We stress that in the argument above only the mask refreshings corresponding to one of the x_i shift are assumed to be non-probed (which must be the case because of the limited number of probes), and that all the remaining mask refreshings can be freely probed by the adversary, and correctly simulated.

The previous argument only applies when the Sbox evaluation is considered in isolation. When combined with other operations (in particular Xor gates), we must actually apply the same technique (with the I subset) as in [ISW03], and we obtain the same bound $n \geq 2t + 1$ for the number of shares, as in the Rivain-Prouff countermeasure (see [CPRR13]).

[2]The mask refreshing is necessary to prevent a different attack. Assume that we probe the first component of $T(0)$ for the initial configuration of the table $T(u)$, and we again probe the first component of $T(0)$ when the table $T(u)$ has eventually been shifted by $x_2 \oplus \cdots \oplus x_{n-1}$. If $x_2 \oplus \cdots \oplus x_{n-1} = 0$ then without mask refreshing those two probed values must be the same; this leaks information about $x_2 \oplus \cdots \oplus x_{n-1}$, and therefore it suffices to additionally probe x_1 and x_{n-1} to have an attack of order 4 for any n.

[3]The previous argument could be extended to the optimal number of probes $n - 1$ by considering the initial sharing of $S(u \oplus x_1)$ and by adding a final mask refreshing after the evaluation of $T(x_n)$, as actually done in Section 3.1.

Asymptotic complexities. With respect to the number n of shares, our new countermeasure has the same time complexity $\mathcal{O}(n^2)$ as the Rivain-Prouff and Carlet *et al.* countermeasures. However for a k-bit input table, our basic countermeasure has complexity $\mathcal{O}(2^k \cdot n^2)$ whereas the Carlet *et al.* countermeasure has complexity $\mathcal{O}(2^{k/2} \cdot n^2)$, which is better for large k.

In Section 3.3 we describe a variant of our countermeasure for processors with large register size, with the same time complexity $\mathcal{O}(2^{k/2} \cdot n^2)$ as the Carlet *et al.* countermeasure, using a similar approach as in [RDP08]. Our variant consists in packing multiple Sbox outputs into a single register, and performing the table recomputations at the register level first. For example for DES we can pack 8 output 4-bit nibbles into a single 32-bit register; in that case the running time is divided by a factor 8. We stress that our variant does *not* consist in putting multiple shares of the same variable into a single register, as reading such register would reveal many shares at once, and thereby decrease the number of probes t required to break the countermeasure.

Note that our countermeasure has memory complexity $\mathcal{O}(n)$, instead of $\mathcal{O}(n^2)$ for the Rivain-Prouff countermeasure as described in [RP10]. However we show in Appendix C that the memory complexity of the Rivain-Prouff countermeasure can be reduced to $\mathcal{O}(n)$, simply by computing the variables in a different order; this extends to the Carlet *et al.* countermeasure. We summarize in Table 1 the time and memory complexities of the two countermeasures.

Countermeasure	Time complexity	Memory complexity
Carlet *et al.* [CGP+12]	$\mathcal{O}(2^{k/2} \cdot n^2)$	$\mathcal{O}(2^{k/2} \cdot n)$
Our countermeasure	$\mathcal{O}(2^k \cdot n^2)$	$\mathcal{O}(2^k \cdot n)$
Our countermeasure (large register)	$\mathcal{O}(2^{k/2} \cdot n^2)$	$\mathcal{O}(2^{k/2} \cdot n)$

Table 1. Time and memory complexities, for a k-bit input table masked with n shares and secure against any attack at order t, with $2t + 1 \leq n$. The memory complexity for large register size is expressed in number of registers.

Protecting a full Block-Cipher. We show how to integrate our countermeasure into the protection of a full block-cipher against t-th order attacks. We consider two models of security. In the *restricted* model, the adversary always probes the same t intermediate variables for different executions of the block-cipher. In the *full* model the adversary can change the position of its probes adaptively between successive executions; this is essentially the ISW model for stateful circuits.

The restricted model is relevant in practice because in a t-th order DPA attack, the statistical analysis is performed on a fixed set of t intermediate variables for all executions. In both models the key is initially provided in shared form as input, with n shares. In the full model it is necessary to re-randomize the shares of the key between executions, since otherwise the adversary could recover the key by moving its probes between successive executions; obviously this re-randomization of shares must also be secure against a t-th order attack.

We show that $n \geq 2t + 1$ is sufficient to achieve security against t-th order attacks in both models. In particular, this improves the bound $n \geq 4t + 1$ from [ISW03] for stateful circuits.[4] We get an improved bound because for every execution we use both an initial re-randomization of the key shares (before they are used to evaluate the block-cipher) and a final re-randomization of the key shares (before they are given as input to the next execution), whereas in [ISW03] only a final re-randomization was used. With the same technique we can obtain the same improved bound in the full model for the Rivain-Prouff countermeasure and its extension by Carlet *et al.*.

Note that in the full model the bound $n \geq 2t + 1$ is actually optimal. Namely as noted in [ISW03] the adversary can probe t of the key shares at the end of one execution and then another t of the key shares at the beginning of the next execution, hence a total of $2t$ key shares of the same n-sharing of the secret-key. Hence $n \geq 2t + 1$ shares are necessary.

[4]In [ISW03] the bounds are $n \geq 2t + 1$ for stateless circuits and $n \geq 4t + 1$ for stateful circuits.

Practical Implementation. Finally we have performed a practical implementation of our new countermeasure for both AES and DES, using a 32-bit architecture so that we could apply our large register variant. For comparison we have also implemented the Rivain-Prouff countermeasure for AES and the Carlet *et al.* countermeasure for DES; for the latter we have used the technique from [RV13], in which the evaluation of a DES Sbox requires only 7 non-linear multiplications. We summarize the result of our practical implementations in Section 5. We obtain that in practice for AES our algorithm is less efficient than Rivain-Prouff, which can take advantage of the special algebraic structure of the AES Sbox; however for DES our algorithm performs slightly better. Our implementation is publicly available [Cor13].

2 Definitions

In this section we first recall the Ishai-Sahai-Wagner (ISW) framework [ISW03] for proving the resistance of circuits against probing attacks. In [RP10] Rivain and Prouff describe an adaptation of the ISW model for software implementations. We follow the same approach and describe two security models: a restricted model in which the adversary always probes the same t intermediate variables (which is essentially the model considered in [RP10]), and a full model in which the t probes can be changed adaptively between executions (which is essentially the ISW model for stateful circuits).

2.1 The Ishai-Sahai-Wagner Framework

Privacy for Stateless Circuits. A *stateless* circuit over $\mathbb{F}_2$ is a directed acyclic graph whose sources are labeled with the input variables, sinks are labeled with output variables, and internal vertices stand for function gates. A stateless circuit can be *randomized*, if it additionally contains *random gates*; every such gate has no input, and its only output at each invocation of the circuit is a uniform random bit.

A *t-limited adversary* can probe up to t wires in the circuit, and has unlimited computational power. A stateless circuit C is called (perfectly) secure against such adversary, if the distribution of the probes can be efficiently and perfectly simulated, without access to the internal wires of C. For stateless circuits one assumes that the inputs and outputs of the circuit must remain private. For example in a block-cipher the input key must remain private. To prevent the adversary for learning the inputs and outputs one uses an *input encoder* I and an *output decoder* O, whose internal wires cannot be probed. Additionally, the inputs of I and the outputs of O are also assumed to be protected against probing. However, the outputs of I and the inputs to O can be probed. Finally the *t-private stateless transformer* (T, I, O) maps a stateless circuit C into a (randomized) stateless circuit C', such that C' is secure against the t-limited adversary, and $O \circ C' \circ I$ has the same input-output functionality as C.

We illustrate this property with a simple circuit C computing a single xor gate $z = x \oplus y$. The input encoder I would first generate independent uniform bits $x_2, \ldots, x_n$, and let $x_1 = x \oplus x_2 \oplus \cdots \oplus x_n$. Note that x_1 is also a uniform random bit, and $x = x_1 \oplus \cdots \oplus x_n$. If the adversary can only read $t = n - 1$ of the n wires $x_1, \ldots, x_n$, then the adversary's behavior can be efficiently simulated by an adversary who cannot probe any of these wires. Namely any t of these wires are independently and uniformly distributed random bits; therefore the probed values can be simulated by picking t independent random bits. Note that this does not reveal any information about x; however $n = t + 1$ wires are sufficient to recover x. The input encoder I would proceed similarly with $y = y_1 \oplus \cdots \oplus y_n$. The modified circuit C' would compute the wires $z_i = x_i \oplus y_i$ for all $1 \le i \le n$. Eventually the output decoder O would compute $z = z_1 \oplus \cdots \oplus z_n$. Since all the shares x_i, y_i are manipulated separately, the modified circuit C' is clearly secure against an adversary with at most $t = n - 1$ probes.

The ISW Scheme. In [ISW03] the author showed how to transform any boolean circuit C of size $|C|$ into a circuit of size $\mathcal{O}(|C| \cdot t^2)$ that is perfectly secure against a t-limited adversary. The approach in [ISW03] consists in secret-sharing every variable x into n shares x_i where $n = 2t + 1$, that is $x = x_1 \oplus x_2 \oplus \cdots \oplus x_n$ where $x_2, \ldots, x_n$ are uniformly and independently distributed bits. To protect any circuit it is sufficient to show how to process a NOT gate and a AND gate.

Processing a NOT gate is straightforward since $\bar{x} = \bar{x}_1 \oplus x_2 \oplus \cdots \oplus x_n$; therefore it suffices to invert the first share x_1. Processing a AND gate $z = xy$ is more complicated. One writes:

$$z = xy = \left(\bigoplus_{i=1}^{n} x_i\right) \cdot \left(\bigoplus_{i=1}^{n} y_i\right) = \bigoplus_{1 \le i,j \le n} x_i y_j$$

and the cross-products $x_i y_j$ are processed and recombined without leaking information about x and y. More precisely for each $1 \le i < j \le n$ one generates random bits $r_{i,j}$ and computes $r_{j,i} = (r_{i,j} \oplus x_i y_j) \oplus x_j y_i$; the n shares z_i of $z = xy$ are then computed as $z_i = x_i y_i \oplus \bigoplus_{j \neq i} r_{i,j}$. Since there are n^2 such cross-products, every AND gate of the circuit is expanded to $\mathcal{O}(n^2) = \mathcal{O}(t^2)$ new gates in the circuit. The authors prove the security of their construction against a t-limited adversary by showing that any set of t probes can be perfectly simulated without knowing the internal wires of the circuit, for $n \ge 2t + 1$.

Extension to Stateful Circuits. The ISW model and construction can be extended to stateful circuits, that is a circuit containing memory cells. In the stateful model the inputs and outputs are known to the attacker and one does not use the input encoder I and output decoder O. For a block-cipher the secret key sk would be originally incorporated in a shared form sk_i inside the memory cells of the circuit; the key shares sk_i would be re-randomized after each invocation of the circuit. The authors show that for stateful circuits $n \ge 4t + 1$ shares are sufficient for security against a t-limited adversary; we refer to [ISW03] for more details.

2.2 Security Model for Software Implementations

In [RP10] Rivain and Prouff describe an adaptation of the ISW model for software implementations of encryption algorithms. They consider a *randomized encryption algorithm* $\mathcal{E}$ taking as input a plaintext m and a randomly shared secret-key sk and outputting a ciphertext c, with additional access to a random number generator. More precisely the secret-key sk is assumed to be split into n shares $sk_1, \ldots, sk_n$ such that $sk = sk_1 \oplus \cdots \oplus sk_n$ and any $(n-1)$-uple of sk_i's is uniformly and independently distributed. Instead of considering the internal wires of a circuit, they consider the intermediate variables of the software implementation. This approach seems well suited for proving the security of our countermeasure; in principle one could write our countermeasure with randomized table as a stateful circuit and work in the ISW model for stateful circuits, but that would be less convenient.

In the following we describe two different models of security. In the *restricted* model the adversary provides a message m as input and receives $c = \mathcal{E}_{sk}(m)$ as output. The adversary can run $\mathcal{E}_{sk}$ several times, but she always obtain the same set of t intermediate variables that she can freely choose before the first execution. In the *full* model, the adversary can adaptively change the set of t intermediate variables between executions. In both models the shares sk_i of the secret-key sk are initially incorporated in the memory cells of the block-cipher implementation. We say that a randomized encryption algorithm is secure against t-th order attack (in the restricted or full model) if the distribution of any t intermediate variables can be perfectly simulated without the knowledge of the secret-key sk. This implies that anything an adversary $\mathcal{A}$ can do from the knowledge of t intermediate variables, another adversary $\mathcal{A}'$ can do the same without the knowledge of those t intermediate variables. Note that since $\mathcal{A}$ initially provides the message m and receives the ciphertext c, we can consider that both m and c are public and given to the simulator.

Note that in the full model it is necessary to re-randomize in memory the shares sk_i of the key, since otherwise the adversary could recover sk by moving its probes between successive executions; obviously this re-randomization of shares must also be secure against a t-th order attack.

3 Our New Algorithm

3.1 Description

In this section we describe our new algorithm for computing $y = S(x)$ where

$$S : \{0,1\}^k \to \{0,1\}^{k'}$$

is a look-up table with k-bit input and k'-bit output. Our new algorithm takes as input $x_1, \ldots, x_n$ such that $x = x_1 \oplus \cdots \oplus x_n$ and must output $y_1, \ldots, y_n$ such that $y = S(x) = y_1 \oplus \cdots \oplus y_n$, without leaking information about x. Our algorithm uses two temporary tables T and T' in RAM; both have k-bit input and a vector of n elements of k'-bit as output, namely

$$T, T' : \{0,1\}^k \to (\{0,1\}^{k'})^n$$

Given a vector $\boldsymbol{v} = (v_1, \ldots, v_n)$ of n elements, we write $\oplus(\boldsymbol{v}) = v_1 \oplus \cdots \oplus v_n$. We denote by $T(u)[j]$ and $T'(u)[j]$ the j-th component of the vectors $T(u)$ and $T'(u)$ respectively, for $1 \leq j \leq n$. In practice the two tables can be implemented as 2-dimensional arrays of elements in $\{0,1\}^{k'}$. We use the same RefreshMasks procedure as in [RP10].

Algorithm 1 Masked computation of $y = S(x)$

```
Input: $x_1, \ldots, x_n$ such that $x = x_1 \oplus \cdots \oplus x_n$
Output: $y_1, \ldots, y_n$ such that $y = S(x) = y_1 \oplus \cdots \oplus y_n$
 1: for all $u \in \{0,1\}^k$ do
 2:     $T(u) \leftarrow \big(S(u), 0, \ldots, 0\big) \in (\{0,1\}^{k'})^n$          ▷ $\oplus\big(T(u)\big) = S(u)$
 3: end for
 4: for $i = 1$ to $n-1$ do
 5:     for all $u \in \{0,1\}^k$ do
 6:         for $j = 1$ to $n$ do $T'(u)[j] \leftarrow T(u \oplus x_i)[j]$          ▷ $T'(u) \leftarrow T(u \oplus x_i)$
 7:     end for
 8:     for all $u \in \{0,1\}^k$ do
 9:         $T(u) \leftarrow \mathsf{RefreshMasks}\big(T'(u)\big)$          ▷ $\oplus\big(T(u)\big) = S(u \oplus x_1 \oplus \cdots \oplus x_i)$
10:     end for
11: end for          ▷ $\oplus\big(T(u)\big) = S(u \oplus x_1 \oplus \cdots \oplus x_{n-1})$ for all $u \in \{0,1\}^k$.
12: $(y_1, \ldots, y_n) \leftarrow \mathsf{RefreshMasks}\big(T(x_n)\big)$          ▷ $\oplus\big(T(x_n)\big) = S(x)$
13: return $y_1, \ldots, y_n$
```

Algorithm 2 RefreshMasks

```
Input: $z_1, \ldots, z_n$ such that $z = z_1 \oplus \cdots \oplus z_n$
Output: $z_1, \ldots, z_n$ such that $z = z_1 \oplus \cdots \oplus z_n$
 1: for $i = 2$ to $n$ do
 2:     $tmp \leftarrow \{0,1\}^{k'}$
 3:     $z_1 \leftarrow z_1 \oplus tmp$
 4:     $z_i \leftarrow z_i \oplus tmp$
 5: end for
 6: return $z_1, \ldots, z_n$
```

It is easy to verify the correctness of Algorithm 1. We proceed by induction. Assume that at Line 4 for index i we have for all inputs $u \in \{0,1\}^k$:

$$\oplus\big(T(u)\big) = S(u \oplus x_1 \oplus \cdots \oplus x_{i-1}) \tag{2}$$

The assumption clearly holds for $i = 0$, since initially we have $\oplus\big(T(u)\big) = S(u)$ for all inputs $u \in \{0,1\}^k$. Assuming that (2) holds for index i at Line 4, after the shifts performed at Line 6 we have for all inputs $u \in \{0,1\}^k$,

$$\oplus\big(T'(u)\big) = \oplus\big(T(u \oplus x_i)\big) = S\big((u \oplus x_i) \oplus x_1 \oplus \cdots \oplus x_{i-1}\big) = S(u \oplus x_1 \oplus \cdots \oplus x_i)$$

and therefore the assumption holds at Step $i+1$. At the end of the loop we have therefore

$$\oplus\big(T(u)\big) = S(u \oplus x_1 \oplus \cdots \oplus x_{n-1})$$

for all $u \in \{0,1\}^k$, and then $\oplus\big(T(x_n)\big) = S(x_n \oplus x_1 \oplus \cdots \oplus x_{n-1}) = S(x)$ which gives $y_1 \oplus \cdots \oplus y_n = S(x)$ as required. This proves the correctness of Algorithm 1.

Note that a NAND gate can be implemented as a 2-bit input, 1-bit output look-up table; therefore Algorithm 1 can be used to protect any circuit, with the same complexity $\mathcal{O}(n^2)$ as the ISW construction.

3.2 Security Proof

The following Lemma proves the security of our countermeasure against t-th order attacks, for any t such that $2t+1 \leq n$. Given a subset $I \subset [1,n]$ of indices we denote $x_{|I} := (x_i)_{i \in I}$.

Lemma 1. *Let $(x_i)_{1 \leq i \leq n}$ be the input shares of Algorithm 1 and let t be such that $2t < n$. For any set of t intermediate variables, there exists a subset $I \subset [1,n]$ of indices such that $|I| \leq 2t < n$ and the distribution of those t variables can be perfectly simulated from the shares $x_{|I}$. The output shares $y_{|I}$ can also be perfectly simulated from $x_{|I}$.*

Proof. Given a set of t intermediate variables $v_1, \ldots, v_t$ probed by the adversary, we construct a subset $I \subset [1,n]$ of indices such that the distribution of those t variables can be perfectly simulated from $x_{|I}$. We call *Part i* the computation performed within the main for loop for index i for $1 \leq i \leq n-1$, that is from Line 5 to Line 10 of Algorithm 1; similarly we call *Part n* the computation performed at Line 12. We do not consider the intermediate variables from Line 2, as they can be perfectly simulated without the knowledge of x.

The proof intuition is as follows. Every intermediate variable v_h is identified by its "line" index i corresponding to the Part in which it appears, with $1 \leq i \leq n$, and by its "column" index j corresponding to the j-th component of the vector in which it appears; for any such intermediate variable v_h both indices i and j are added to the subset I (except for x_i and the *tmp* variables within RefreshMasks for which only i is added). The crucial observation is the following: if $i \notin I$, then no intermediate variable was probed within Part i of Algorithm 1; in particular the *tmp* variables within the corresponding RefreshMasks were not probed. Therefore we can perfectly simulate the outputs of the RefreshMasks function which have "column" index $j \in I$, by generating uniform and independent elements in $\{0,1\}^{k'}$, as long as $|I| < n$. This means that for $i \notin I$ we can perfectly simulate all variables $T(u)[j]$ for $j \in I$ in Line 9. Considering now Part i for which $i \in I$, since we know x_i we can still perfectly simulate all intermediate variables with "column" index $j \in I$ (including also the *tmp* variables within RefreshMasks), which includes by definition of I all the intermediates variables v_h. Therefore all intermediate variables v_h can be perfectly simulated as long as $|I| < n$, which gives the condition $2t < n$.

Formally the procedure for constructing the set I is as follows:

1. We start with $I = \emptyset$.
2. For any intermediate variable v_h:
   (a) If $v_h = x_i$ or $v_h = u \oplus x_i$ at Line 6, then add i to I.
   (b) If $v_h = T(u \oplus x_i)[j]$ or $v_h = T'(u)[j]$ at Line 6 in Part i, then add both i and j to I.
   (c) If $v_h = T'(u)[j]$ or $v_h = T(u)[j]$ at Line 9 in Part i, then add both i and j to I.
   (d) If $v_h = tmp$ for any tmp within RefreshMasks in Part i (either at Line 9 or 12), then add i to I.
   (e) If $v_h = x_n$ at Line 12, then add n to I.
   (f) If $v_h = T(x_n)[j]$ or $v_h = y_j$ at Line 12, then add both n and j to I.

This terminates the description of the procedure for constructing the set I. Since any intermediate variable v_h adds at most two indices in I, we must have $|I| \leq 2t < n$.

We now show how to complete a perfect simulation of all intermediate variables v_h using only the values $x_{|I}$. We proceed by induction. Assume that at the beginning of Part i we can perfectly simulate all variables $T(u)[j]$ for all $j \in J$ and all $u \in \{0,1\}^k$. This holds for $i = 1$ since initially we have $T(u) = (S(u), 0, \ldots, 0)$ which does not depend on x.

We distinguish two cases. If $i \notin I$ then no *tmp* variable within the RefreshMasks in Part i has been probed. Therefore we can perfectly simulate all intermediate variables $T(u)[j]$ for $j \in I$ at the output of RefreshMasks at Line 9, or similarly all y_j for $j \in I$ at the output of RefreshMasks at Line 12 when $i = n$, as long as $|I| < n$. Formally this can be proven as follows. Let j^* be such that $j^* \notin I$. Since the internal variables of the RefreshMasks are not probed, we can redefine RefreshMasks where the randoms *tmp* are accumulated inside z_{j^*} instead of z_1. Since $j^* \notin I$ we have that z_{j^*} is never used in the computation of any variable v_h, and therefore every variables z_j for $j \in I$ is masked by a random *tmp* which is used only once.

Therefore at the output of RefreshMasks the variables $T(u)[j]$ for $j \in I$ can be perfectly simulated for all $u \in \{0,1\}^k$, simply by generating uniform and independent values.

If $i \in I$ then knowing x_i we can perfectly simulate all intermediate variables with column index $j \in I$ in Part i. Namely our induction hypothesis states that at the beginning of Part i the variables $T(u)[j]$ for all $j \in J$ can already be perfectly simulated. Knowing x_i we can therefore propagate the simulation for all variables with column index j and perfectly simulate $T(u \oplus x_i)[j]$, $T'(u)[j]$ and the resulting $T(u)[j]$ at Line 9, and similarly the variables y_j at Line 12 if $i = n$; in particular the *tmp* variables within RefreshMasks are simulated exactly as in the RefreshMasks procedure.

Since in both cases we can perfectly simulate all intermediate variables $T(u)[j]$ for $j \in I$ at the end of Part i, the induction hypothesis holds for $i+1$; therefore it holds for all $1 \le i \le n$. From the reasoning above we can therefore simulate all intermediate variables in Part i with column index j such that $i, j \in I$; by definition of I this includes all intermediate variables v_h, and all output shares $y_{|I}$; this proves Lemma 1. □

3.3 A Variant for Processors with large Register

With respect to the number n of shares, our new countermeasure has the same time complexity $\mathcal{O}(n^2)$ as the Rivain-Prouff and Carlet *et al.* countermeasures. However for a k-bit input table, our algorithm has complexity $\mathcal{O}(2^k \cdot n^2)$ whereas the Carlet *et al.* countermeasure has complexity $\mathcal{O}(2^{k/2} \cdot n^2)$ only.

In this section we describe a variant of our countermeasure with the same complexity as Carlet *et al.*, but for processors with large enough register size ω bits, using a similar approach as in [RDP08, Section 3.3]. We assume that a read/write operation on such register takes unit time. In this variant the k'-bit outputs of the Sbox are first packed into words of $\omega = \ell \cdot k'$ bits, where ℓ is assumed to be a power of two. For example, for a DES Sbox with $k = 6$ input bits and $k' = 4$ output bits, on a $\omega = 32$ bits architecture we can pack $\ell = 8$ output 4-bit nibbles into a 32-bit word.

Formally, starting from the original Sbox $S : \{0,1\}^k \to \{0,1\}^{k'}$ we define a new Sbox S' with k_1-bit input and $\omega = \ell \cdot k'$ bits output

$$S'(a) = S(a \,\|\, 0^{k_2}) \,\|\, \cdots \,\|\, S(a \,\|\, 1^{k_2})$$

where $k = k_1 + k_2$ and $k_2 = \log_2 \ell$. To compute $S(x)$ for $x \in \{0,1\}^k$, we proceed in two steps:

1. Write $x = a\|b$ for $a \in \{0,1\}^{k_1}$ and $b \in \{0,1\}^{k_2}$, and compute $z = S'(a) = S(a\|0^{k_2})\| \cdots \|S(a\|1^{k_2})$
2. Viewing z as a k_2-bit input, k'-bit output table, compute $y = S(x) = z(b)$.

We show in Appendix A how to adapt our countermeasure from Algorithm 1 to compute $y = S(x)$ in the two-step process above, taking as input the shares x_i. Since the new table S' has size $2^k/\ell$ elements instead of 2^k, the complexity of the first step becomes $\mathcal{O}(2^k/\ell \cdot n^2)$. Similarly the table at the second step contains ℓ elements, which gives a complexity $\mathcal{O}(\ell \cdot n^2)$. Therefore the total complexity of our variant countermeasure is $\mathcal{O}((2^k/\ell + \ell) \cdot n^2)$. If we have large enough register size ω so that we can take $\ell = \omega/k' = 2^{k/2}$, then the complexity of our variant countermeasure becomes $\mathcal{O}(2^{k/2} \cdot n^2)$, the same complexity as the Carlet *et al.* countermeasure.[5,6]

4 Higher Order Masking of a Full Block-Cipher

In this section we show how to integrate our countermeasure into a full block-cipher. We consider a block-cipher with the following operations: Xor operation $z = x \oplus y$, linear (or affine) transform $y = f(x)$, and

[5]Since the Carlet *et al.* countermeasure is based on computing in the field $\mathbb{F}_{2^k}$, it is unclear how the the Carlet *et al.* countermeasure could benefit from larger register sizes; so it seems that its complexity remains $\mathcal{O}(2^{k/2} \cdot n^2)$ even for large register size.

[6]Note that for DES with 32-bit registers we can take the optimum $\ell = 2^{6/2} = 8$. However for AES the optimum $\ell = 2^{8/2} = 16$ would require 128-bit registers.

look-up table $y = S(x)$. This covers both AES and DES block-ciphers. We show how to apply high-order masking to these operations, in order to protect a full block-cipher against t-th order attacks.[7]

Xor operation. We consider a Xor operation $z = x \oplus y$. Taking as input the shares x_i and y_i such that $x = x_1 \oplus \cdots \oplus x_n$ and $y = y_1 \oplus \cdots \oplus y_n$, it suffices to compute the shares $z_i = x_i \oplus y_i$.

Linear operation. We consider a linear operation $y = f(x)$. Taking as input the shares x_i such that $x = x_1 \oplus \cdots \oplus x_n$, it suffices to compute the shares $y_i = f(x_i)$ separately.

Table Look-up. A table look-up $y = S(x)$ is computed using our previous Algorithm 1.

Input Encoding. Given x as input, we first encode x as $x_1 = x$ and $x_i = 0$ for $2 \leq i \leq n$. Secondly we let $(x_1, \ldots, x_n) \leftarrow \mathsf{RefreshMasks}(x_1, \ldots, x_n)$.

Output Decoding. Given $y_1, \ldots, y_n$ as input, we compute $y = y_1 \oplus \cdots \oplus y_n$ using Algorithm 3 below.

Algorithm 3 Shares recombination

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

Key Shares Refreshing. As mentioned in Section 2.2 we must re-randomize the key shares between successive executions of the block-cipher in order to achieve security in the full model. Using Algorithm 4 below we perform both an initial Key Shares Refreshing (before the shares sk_i are used to evaluate the block-cipher), and a final Key Shares Refreshing (before the key shares sk_i are stored for the next execution).[8]

Algorithm 4 Key Shares Refreshing

```
Input: $sk_1, \ldots, sk_n$ such that $sk = sk_1 \oplus \cdots \oplus sk_n$
Output: $sk_1, \ldots, sk_n$ such that $sk = sk_1 \oplus \cdots \oplus sk_n$
1: for $i = 1$ to $n$ do $(sk_1, \ldots, sk_n) \leftarrow \mathsf{RefreshMasks}(sk_1, \ldots, sk_n)$
2: return $sk_1, \ldots, sk_n$
```

This terminates the description of our randomized encryption algorithm. The following theorem proves the security of the randomized encryption scheme defined above in the full model, under the condition $n \geq 2t + 1$; we give the proof in Appendix B. This improves the bound $n \geq 4t + 1$ from [ISW03] for stateful circuits. We stress that any set of t intermediate variables can be probed by the adversary, including variables in the input encoding, output decoding, and key shares refreshing; that is, no operation is assumed to be leak-free.

Theorem 1. *The randomized encryption scheme defined above achieves t-th order security in the full model for $n \geq 2t + 1$.*

Remark 1. The input encoding operation need not be randomized by $\mathsf{RefreshMasks}$; this is because the input x is public and given to the simulator, who can therefore perfectly simulate the initial shares $x_{|I}$ for any subset $I \subset [1, n]$. Moreover in the restricted model the key shares refreshing is not necessary. In practice we can keep both operations as their time complexity is only $\mathcal{O}(n)$ and $\mathcal{O}(n^2)$ respectively.

[7]Xor is a linear operation, so one could consider the linear operation $y = f(x)$ only, but it seems more convenient to consider the Xor operation separately.

[8]Note that for both algorithms 3 and 4 the $\mathsf{RefreshMasks}$ procedure must be applied with the tmp randoms generated with the appropriate bit-size (instead of k').

Remark 2. We stress that the secret key sk must be initially provided with randomized shares, since sk is secret and not given to the simulator; in other words it would be insecure for the randomized block-cipher to receive sk as input and perform the initial input encoding on sk by himself.

Remark 3. In the output decoding operation we perform a series of n mask refreshing before computing y. This is to enable a correct simulation of the intermediate variables c at Line 3 in case they are probed by the adversary.

5 Practical Implementation

We have performed a practical implementation of our new countermeasure for both AES and DES, using a 32-bit architecture so that we could apply our large register variant. More precisely we could pack $\ell = 4$ output bytes for AES, and $\ell = 8$ output 4-bit nibbles for DES. For comparison we have also implemented the Rivain-Prouff countermeasure [RP10] for AES and the Carlet *et al.* countermeasure [CGP+12] for DES; for the latter we have used the technique from [RV13], in which the evaluation of a DES Sbox requires only 7 non-linear multiplications. The performances of our implementations are summarized in Table 2. We use the bound $n = 2t + 1$ for the full model of security (which implies security in the restricted model).

	t	n	Time (ms)	Penalty
AES, unmasked			0.0018	1
AES, Rivain-Prouff	1	3	0.092	50
AES, our countermeasure	1	3	0.80	439
AES, Rivain-Prouff	2	5	0.18	96
AES, our countermeasure	2	5	2.2	1205
AES, Rivain-Prouff	3	7	0.31	171
AES, our countermeasure	3	7	4.4	2411
AES, Rivain-Prouff	4	9	0.51	276
AES, our countermeasure	4	9	7.3	4003

	t	n	Time (ms)	Penalty
DES, unmasked			0.010	1
DES, Carlet *et al.*	1	3	0.47	47
DES, our countermeasure	1	3	0.31	31
DES, Carlet *et al.*	2	5	0.78	79
DES, our countermeasure	2	5	0.59	59
DES, Carlet *et al.*	3	7	1.3	129
DES, our countermeasure	3	7	0.90	91
DES, Carlet *et al.*	4	9	1.9	189
DES, our countermeasure	4	9	1.4	142

Table 2. Comparison of secure AES and DES implementations, in C on a MacBook Air running on a 1.86 GHz Intel processor.

We obtain that in practice for AES our algorithm is an order of magnitude less efficient than Rivain-Prouff, which can take advantage of the special algebraic structure of the AES Sbox; however for DES our algorithm performs slightly better than the Carlet *et al.* countermeasure. Note that this holds for a 32-bit architecture; on a 8-bit architecture the comparison could be less favorable. The source code of our implementations is publicly available [Cor13].

One could think that because of the large penalty factors the countermeasures above are unpractical. However in some applications the block-cipher evaluation can be only a small fraction of the full protocol (for example in a challenge-response authentication protocol), and in that case a penalty factor of say 100 for a single block-cipher evaluation may be acceptable.

References

[CGP+12] Claude Carlet, Louis Goubin, Emmanuel Prouff, Michaël Quisquater, and Matthieu Rivain. Higher-order masking schemes for s-boxes. In *FSE*, pages 366–384, 2012.

[CJRR99] Suresh Chari, Charanjit S. Jutla, Josyula R. Rao, and Pankaj Rohatgi. Towards sound approaches to counteract power-analysis attacks. In *CRYPTO*, 1999.

[Cor13] Jean-Sebastien Coron. `https://github.com/coron/htable/`, 2013.

[CPR07] Jean-Sebastien Coron, Emmanuel Prouff, and Matthieu Rivain. Side channel cryptanalysis of a higher order masking scheme. In *CHES*, pages 28–44, 2007.

[CPRR13] Jean-Sebastien Coron, Emmanuel Prouff, Matthieu Rivain, and Thomas Roche. Higher-order side channel security and mask refreshing. In *FSE*, 2013.

[CRR02] Suresh Chari, Josyula R. Rao, and Pankaj Rohatgi. Template attacks. In *CHES*, pages 13–28, 2002.

[HOM06] Christoph Herbst, Elisabeth Oswald, and Stefan Mangard. An AES smart card implementation resistant to power analysis attacks. In *ACNS*, pages 239–252, 2006.

[ISW03] Yuval Ishai, Amit Sahai, and David Wagner. Private circuits: Securing hardware against probing attacks. In *CRYPTO*, pages 463–481, 2003.

[KJJ99] Paul C. Kocher, Joshua Jaffe, and Benjamin Jun. Differential power analysis. In *CRYPTO*, pages 388–397, 1999.

[Mes00] Thomas S. Messerges. Using second-order power analysis to attack dpa resistant software. In *CHES*, pages 238–251, 2000.

[PR13] Emmanuel Prouff and Matthieu Rivain. Masking against side-channel attacks: A formal security proof. In *EUROCRYPT*, pages 142–159, 2013.

[RDP08] Matthieu Rivain, Emmanuelle Dottax, and Emmanuel Prouff. Block ciphers implementations provably secure against second order side channel analysis. In *FSE*, pages 127–143, 2008.

[RP10] Matthieu Rivain and Emmanuel Prouff. Provably secure higher-order masking of AES. In *CHES*, pages 413–427, 2010.

[RV13] Arnab Roy and Srinivas Vivek. Analysis and improvement of the generic higher-order masking scheme of FSE 2012. In *CHES*, pages 417–434, 2013.

[Sha79] Adi Shamir. How to share a secret. *Commun. ACM*, 22(11):612–613, 1979.

[SP06] Kai Schramm and Christof Paar. Higher order masking of the AES. In *CT-RSA*, pages 208–225, 2006.

A A Variant for Processors with Large Registers

In this section we describe a variant of Algorithm 1 for processors with large register size ω bits. We assume that a read/write operation on such register takes unit time. As previously the goal is to compute $y = S(x)$ where

$$S : \{0,1\}^k \to \{0,1\}^{k'}$$

is a look-up table with k-bit input and k'-bit output.

Under the variant the k'-bit outputs of S are first packed into words of $\omega = \ell \cdot k'$ bits, where ℓ is assumed to be a power of two. Formally, we define a new Sbox S' with k_1-bit input and $\omega = \ell \cdot k'$ bits output

$$S'(a) = S(a \,\|\, 0^{k_2}) \,\|\, \cdots \,\|\, S(a \,\|\, 1^{k_2})$$

for $a \in \{0,1\}^{k_1}$ where $k = k_1 + k_2$ and $k_2 = \log_2 \ell$. To compute $S(x)$ for $x \in \{0,1\}^k$, we proceed in two steps:

1. Write $x = a\|b$ for $a \in \{0,1\}^{k_1}$ and $b \in \{0,1\}^{k_2}$, and compute $z = S'(a) = S(a\|0^{k_2})\|\cdots\|S(a\|1^{k_2})$
2. Viewing z as a k_2-bit input and k'-bit output table, compute $y = S(x) = z(b)$.

We must show how to compute $y = S(x)$ in the two steps above when the input x is shared with n shares x_i. In the first step we proceed as in Algorithm 1, except that the new table S' has a k_1-bit input instead of a k-bit input, and $\omega = \ell \cdot k'$-bit output instead of k'-bit output. Note that the table S' contains $2^{k_1} = 2^{k-k_2} = 2^k/\ell$ elements instead of 2^k for the original S. Since we assume that a read/write operation on a ω-bit register takes unit time, the complexity of the first step is now $\mathcal{O}(2^k/\ell \cdot n^2)$. Note that S and S' take the same amount of memory in RAM; in the first step of our countermeasure we can achieve a speed-up by a factor ℓ because we are moving ℓ blocks of k' bits at a time inside registers of size $\omega = \ell \cdot k'$ bits.

The second step requires a slight modification of Algorithm 1. Namely we must view the output z from Step 1 as a look-up table with k_2-bit input and k'-bit output. However this output z is now obtained in shared form, namely we get shares $z_1, \ldots, z_n$ such that $z = z_1 \oplus \cdots \oplus z_n$, whereas in Algorithm 1 the look-up table $S(x)$ is a public table. This is not a problem, as we can simply keep this table in shared form when initializing the $T(u)$ table at Line 2 of Algorithm 1. More precisely in the second step we can initialize the table $T(u)$ with:

$$T(u) = (z_1(u), \ldots, z_n(u)) \in (\{0,1\}^{k'})^n$$

for all $u \in \{0,1\}^{k_2}$, and we still have $\oplus\big(T(u)\big) = z(u)$ for all u as required. Since the second step uses a table of size $2^{k_2} = \ell$ elements, its complexity is $\mathcal{O}(\ell \cdot n^2)$.

The full complexity of our variant countermeasure is therefore $\mathcal{O}((2^k/\ell+\ell)\cdot n^2)$. If we have large enough register size ω so that we can take $\ell=\omega/k'=2^{k/2}$, then the complexity of our variant countermeasure becomes $\mathcal{O}(2^{k/2}\cdot n^2)$, the same complexity as the Carlet *et al.* countermeasure.[9]

The following Lemma shows that our variant countermeasure achieves the same level of security as Algorithm 1; the proof is essentially the same as the proof of Lemma 1 and is therefore omitted.

Lemma 2. *Let $(x_i)_{1\le i\le n}$ be the input shares of the above countermeasure for large register size, and let t be such that $2t<n$. For any set of t intermediate variables, there exists a subset $I\subset[1,n]$ of indices such that $|I|\le 2t<n$ and the distribution of those t variables can be perfectly simulated from the x_i's with $i\in I$. The output shares $y_{|I}$ can also be perfectly simulated from $x_{|I}$.*

B Proof of Theorem 1

As mentioned in introduction, a nice property of the ISW framework is that the simulation technique easily extends from a single gate to the full circuit: it suffices to maintain a global subset of indices I that is iteratively constructed from the t probes as in a single gate. Therefore to prove Theorem 1 we proceed in two steps: in the first step we explain how the subset I is constructed for each of the elementary operations from Section 4; in the second step we show how to derive a proof of security for the full block-cipher, first in the restricted model and then in the full model.

B.1 Security of Elementary Operations

For the Xor operation and the Linear operation, the construction of the subset I is straightforward.

Lemma 3. *Let $(x_i)_{1\le i\le n}$ and $(y_i)_{1\le i\le n}$ be the input shares of the Xor operation. For any set of t intermediate variables, there exists a subset $I\subset[1,n]$ of indices such that $|I|\le 2t$ and the distribution of those t variables can be perfectly simulated from $x_{|I}$ and $y_{|I}$. The output shares $z_{|I}$ can also be perfectly simulated from $x_{|I}$ and $y_{|I}$.*

Proof. For any probed variable x_i or y_i or $z_i=x_i\oplus y_i$, we add i to I. We get $|I|\le t$. Any such variable can be simulated by knowing the values $x_{|I}$ and $y_{|I}$. □

Lemma 4. *Let $(x_i)_{1\le i\le n}$ be the input shares of the Linear operation. For any set of t intermediate variables, there exists a subset $I\subset[1,n]$ of indices such that $|I|\le 2t$ and the distribution of those t variables can be perfectly simulated from $x_{|I}$. The output shares $y_{|I}$ can also be perfectly simulated from $x_{|I}$.*

Proof. For any probed variable x_i or $y_i=f(x_i)$ or any intermediate variable in the computation of $f(x_i)$, we add i to I. We get $|I|\le t$. Any such variable can be simulated by knowing the values $x_{|I}$. □

The look-up table operation is computed using Algorithm 1, and the construction of I is given in the proof of Lemma 1.

For the Initial Encoding operation we don't need to include any index i in I; namely the input x is assumed to be public and given to the simulator.

Lemma 5. *Let x be the public input of the Initial Encoding operation. For any set of t intermediate variables, the distribution of those t variables can be perfectly simulated from x. All output shares x_i can be perfectly simulated.*

Proof. The initial shares x_i with $x_1=x$ and $x_i=0$ for $2\le i\le n$ can be computed from x. The variables in RefreshMasks can then be perfectly simulated. As noted previously this initial RefreshMasks is actually unnecessary. □

[9] In principle the same procedure could be applied recursively, and the total complexity would become $\mathcal{O}(k\cdot n^2)$ instead of $\mathcal{O}(2^{k/2}\cdot n^2)$ for large enough registers. However since the look-up table input size k is usually small ($k\le 8$) this is unlikely to make a difference in practice.

In the Output Decoding operation of Algorithm 3 we first perform a series of n mask refreshing before computing y. This is to enable a correct simulation of the intermediate variables c at Line 3 in case they are probed by the adversary. Note that the output y is assumed to be public and given to the simulator. Heuristically this series of n mask refreshing seems necessary; otherwise to correctly simulate the intermediate variables c one would need to know all the shares y_i, which does not seem possible in the simulation.

Lemma 6. *Let $(y_i)_{1 \le i \le n}$ be the input shares of the Output Decoding operation. For any set of t intermediate variables, there exists a subset $I \subset [1, n]$ of indices such that $|I| \le 2t$ and the distribution of those t variables can be perfectly simulated from $y_{|I}$ and $y = y_1 \oplus \cdots \oplus y_n$.*

Proof. 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 denote by i^* the index of this RefreshMasks. Since we know $y = y_1 \oplus \cdots \oplus y_n$, we can therefore 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 c for computing y.

In summary before the i^* RefreshMasks knowing 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. This proves Lemma 6. □

Finally we show how to construct the subset I for the Key Shares Refreshing from Algorithm 4. The following lemma is straightforward and will be used in the restricted model of security only.[10]

Lemma 7. *Let $(sk_i)_{1 \le i \le n}$ be the input shares of Algorithm 4. For any set of t intermediate variables, there exists a subset $I \subset [1, n]$ of indices such that $|I| \le 2t$ and the distribution of those t variables can be perfectly simulated from the values $sk_{|I}$. The output shares $sk'_{|I}$ can be perfectly simulated from $sk_{|I}$*

Proof. If a variable sk_j in a RefreshMasks operation is probed, we add j to I. We get $|I| \le t \le 2t$, and we can perfectly simulate those t intermediate variables from the input values $sk_{|I}$, as well as the output shares $sk'_{|I}$. □

B.2 Security of the full Block-Cipher in the Restricted Model

In this section we prove the security of the full block-cipher against t-th order attack in the restricted model, using essentially the same reasoning as in [ISW03]. We first restrict ourselves to a single execution of the randomized block-cipher.

For the full block-cipher we compute a global subset $I \subset [1, n]$ of indices by examining each n-shared operation in the randomized block-cipher. We have seen in the previous section that in each n-shared operation a probe adds at most 2 indices in I. Since a total of t intermediate variables in the full block-cipher can be probed by the adversary, the size of the set I will still be bounded by $2t$. The simulation is then performed as in the proof of Lemma 1, from the input of the randomized block-cipher to the output. The input shares $sk_{|I}$ of the block-cipher can be perfectly simulated as long as $|I| < n$, by generating uniform and independent values. Moreover we have shown in the previous lemmas that the output shares $y_{|I}$ of all operations can always be simulated given the input shares with indices in I. Therefore we can maintain the invariant that for each n-shared operation g, the shares of the inputs to g with indices in I are perfectly simulated. Inductively the values of all t intermediate variables probed by the adversary can therefore be perfectly simulated.

The previous reasoning is easily extended to multiple executions of the block-cipher in the restricted model of security. We can see these multiple executions as a single execution in which the shares sk_i of the secret-key are initially provided as input. Namely the shares sk_i used in one execution are applied the initial and final Key Shares Refreshing procedures in this same execution and then used as input for the next

[10] As mentioned previously the Key Shares Refreshing operation is actually unnecessary in the restricted model.

execution. In the "unwound" execution the adversary can obtain up to t intermediate variables in each of the concatenated sub-executions Q. Since in the restricted model these t intermediate variables are the same in each sub-execution Q, we can keep the same subset I of indices for the "unwound" execution and the simulation proof can proceed as before. This proves the security of our construction in the restricted model of security, for $n \geq 2t + 1$.

B.3 Security in the Full Model, for $n \geq 4t + 1$

For ease of exposition we first prove the security in the full model under the stronger bound $n \geq 4t + 1$. For this we prove a different lemma for the Key Refreshing Operation; instead of considering the output shares $sk'_{|I}$ as in Lemma 7 we prove a stronger property: we show that *any* subset J of at most $2t$ output shares from Algorithm 4 can be perfectly simulated.

Lemma 8. *Let $(sk_i)_{1 \leq i \leq n}$ be the input shares of Algorithm 4, where $n \geq 4t+1$. For any set of t intermediate variables, there exists a subset $I \subset [1, n]$ of indices such that $|I| \leq 2t$ and the distribution of those t variables can be perfectly simulated from the values $sk_{|I}$. Moreover for any subset $J \subset [1, n]$ such that $|J| \leq 2t$ we can perfectly simulate the output shares $sk'_{|J}$.*

Proof. The subset I is constructed as previously. If a variable sk_j in a RefreshMasks operation is probed, we add j to I. We get $|I| \leq t \leq 2t$. Note that we use the weaker bound $|I| \leq 2t$ because when considering the full block-cipher the subset I will be constructed from all n-shared operations, instead of only the Key Shares Refreshing operation; this includes Algorithm 1 in which a single probe can add 2 indices in I, which gives the bound $|I| \leq 2t$.

Since $t < n$ there exists at least one RefreshMasks operation that is not probed at all, corresponding to index i^*. Since this i^*-th RefreshMasks operation is not probed at all, we can perfectly simulate any set of $n - 1 \geq 4t$ shares as output of this i^*-th RefreshMasks. We can therefore simulate the shares corresponding to the (at most) $2t$ indices in J, and also the shares corresponding to the (at most) $2t$ indices in I. Note that the two subsets I and J are not necessarily disjoint; however the simulation requires $|J \cup I| < n$; this condition is satisfied since $|J \cup I| \leq |J| + |I| \leq 2t + 2t \leq 4t < n$. Note that the condition $n \geq 4t + 1$ is necessary for our simulation to work; if we only had $n \geq 2t + 1$ then we could not necessarily simulate the shares from both J and I, as we could have $|J \cup I| = n$ and be unable to simulate the output shares without knowing the secret-key sk. The previous simulation after the i^*-th RefreshMasks is then propagated to the output of Algorithm 4.

In summary we can perfectly simulate any intermediate variable sk_j for $j \in I$, knowing the input shares $sk_{|I}$. By definition of I this covers all the probed intermediate variables. Moreover any subset J of $2t$ output shares can be perfectly simulated. This proves the lemma. □

We now turn to the proof of Theorem 1 under the stronger condition $n \geq 4t + 1$. In the full model of security the adversary can adaptively change the t probed intermediate variables between each of the sub-executions Q. Therefore we obtain a different subset I of indices for each sub-execution Q. Assume that we know the input shares $sk^{in}_{|I}$ for the set of indices I of a given sub-execution Q. We can therefore perfectly simulate the t intermediate variables probed by the adversary in this sub-execution Q. Consider now the next execution Q', in which the simulation of the t intermediate variables requires the knowledge of the input shares $sk^{in}_{|I'}$ for a possibly different subset I'. Recall that in each sub-execution Q the key shares are applied the initial and final Key Share Refreshing procedures and then provided as input to the next sub-execution Q'. We can now use the property shown in Lemma 8: any subset J of at most $2t$ output variables sk^{out}_i from the final Key Share Refreshing procedure of sub-execution Q can be perfectly simulated. We can then "connect" the two executions as follows. Since $|I'| \leq 2t$ we can take $J = I'$, which enables to perfectly simulate the output shares $sk^{out}_{|J}$ which are the same as the input shares $sk^{in}_{|I'}$ of the next sub-execution Q'. Therefore we can maintain the invariant that in each sub-execution Q with subset I the shares $sk^{in}_{|I}$ as input to sub-execution Q are perfectly simulated. The simulation can then proceed as before from one sub-execution to the next; inductively the values of all t intermediate variables probed by the adversary can be perfectly simulated in each sub-execution. Note that the previous property from Lemma 8 requires the stronger bound $n \geq 4t+1$ instead of $n \geq 2t+1$.

Note that under this stronger bound $n \geq 4t+1$ we did not need the initial Key Shares Refreshing operation, which is actually unnecessary. However this initial Key Shares Refreshing operation will be required to obtain the improved bound $n \geq 2t+1$.

B.4 Security in the Full Model for $n \geq 2t+1$

We now prove a lemma specifically for the initial Key Shares Refreshing operation.

Lemma 9 (Initial Key Shares Refreshing). *Let $(sk_i)_{1 \leq i \leq n}$ be the input shares of Algorithm 4, where $n \geq 2t+1$. For any set of t intermediate variables, there exists subsets $I, I_2 \subset [1,n]$ of indices with $|I| \leq 2t$ and $|I_2| \leq t$ such that the distribution of those t variables can be perfectly simulated from the values $sk_{|I_2}$, and the distribution of the output shares $sk'_{|I}$ can be perfectly simulated.*

Proof. Since $t < n$ there exists at least one RefreshMasks operation that is not probed at all, corresponding to index i^*.

The subset I_2 is constructed as follows. If a variable sk_j in a RefreshMasks operation of index $i < i^*$ is probed, we add j to I_2. We get $|I_2| \leq t$.

The subset I is constructed as follows. If a variable sk_j in a RefreshMasks operation of index $i > i^*$ is probed, we add j to I. We get $|I| \leq t \leq 2t$.

Since the i^*-th RefreshMasks operation is not probed at all, we can perfectly simulate any set of $n-1 \geq 2t$ shares as output of this i^*-th RefreshMasks. We can therefore perfectly simulate the shares corresponding to the (at most) $2t$ indices in I. This simulation can be propagated to the output shares $sk'_{|I}$ of Algorithm 4.

In summary before the i^*-th RefreshMasks we can perfectly simulate any intermediate variable sk_j for $j \in I_2$, knowing the input shares $sk_{|I_2}$. And after the i^*-th RefreshMasks we can perfectly simulate any intermediate variable sk_j for $j \in I$. By definition of I_2 and I this covers all the probed intermediate variables. This proves the lemma. □

Our second lemma concerns the final Key Shares Refreshing. As previously instead of considering the output shares $sk'_{|I}$ for the subset I as in Lemma 7, we prove a stronger property, namely that the output shares $sk'_{|J}$ can be perfectly simulated for *any* subset $J \subset [1,n]$ such that $|J| \leq t$.

Lemma 10 (Final Key Shares Refreshing). *Let $(sk_i)_{1 \leq i \leq n}$ be the input shares of Algorithm 4, where $n \geq 2t+1$. For any set of t intermediate variables, there exists a subset $I \subset [1,n]$ of indices such that $|I| \leq 2t$ and the distribution of those t variables can be perfectly simulated from the values $sk_{|I}$. Moreover for any subset $J \subset [1,n]$ such that $|J| \leq t$ we can perfectly simulate the output values $sk'_{|J}$.*

Proof. The proof is similar to the proof of Lemma 9. Since $t < n$ there exists at least one RefreshMasks operation that is not probed at all, corresponding to index i^*.

The subset I is constructed as follows. If a variable sk_j in a RefreshMasks operation of index $i < i^*$ is probed, we add j to I. We get $|I| \leq t \leq 2t$.

We also construct a subset I_3 as follows. If a variable sk_j in a RefreshMasks operation of index $i > i^*$ is probed, we add j to I_3. We get $|I_3| \leq t$

Since the i^*-th RefreshMasks operation is not probed at all, we can perfectly simulate any set of $n-1 \geq 2t$ shares as output of this i^*-th RefreshMasks. We can therefore perfectly simulate the shares corresponding to the (at most) t indices in I_3, and the (at most) t indices in J. Note that the two subsets I_3 and J are not necessarily disjoint; however the simulation requires $|J \cup I_3| < n$; this condition is satisfied since $|J \cup I_3| \leq |J| + |I_3| \leq t + t \leq 2t < n$. The previous simulation after the i^*-th RefreshMasks is then propagated to the output of Algorithm 4.

In summary before the i^*-th RefreshMasks we can perfectly simulate any intermediate variable sk_j for $j \in I$, knowing the input shares $sk_{|I}$. And after the i^*-th RefreshMasks we can perfectly simulate any intermediate variable sk_j for $j \in I_3$. By definition of I_3 and I this covers all the probed intermediate variables. Moreover any subset J of at most t output variables sk'_i can be perfectly simulated. This proves the lemma. □

We can now terminate the proof of Theorem 1, using essentially the same reasoning as in the previous section.

In a given sub-execution Q the input key shares sk_i^{in} as provided as input to the initial Key Shares Refreshing operation, which outputs the shares sk_i^{med} which are used by the block-cipher operations. The key shares sk_i^{med} are then given as input to the final Key Shares Refreshing operation, which outputs the shares sk_i^{out}. These shares are then given as input to the next sub-execution Q'.

In the full model of security the adversary can adaptively change the t probed intermediate variables between each of the sub-executions Q. Therefore we obtain a different subset I of indices for each sub-execution Q. Moreover from Lemma 9 for each sub-execution Q we have a subset of indices I_2 with $|I_2| \leq t$ such that the knowledge of input key shares $sk^{in}_{|I_2}$ is necessary for the simulation of the probed variables in the initial Key Refreshing operation. Assume that we know the input shares $sk^{in}_{|I_2}$; from Lemma 9 one can then perfectly simulate the output shares $sk^{med}_{|I}$ of this initial Key Refreshing Operation. These outputs shares $sk^{med}_{|I}$ can then be used to perfectly simulate the probed variables in the main block-cipher operations. Then from Lemma 10 knowing $sk^{med}_{|I}$ one can also perfectly simulate the probed variables in the final Key Shares Refreshing operation. The output shares sk_i^{out} are then provided as input to the next sub-execution Q'. Since the simulation of this next sub-execution Q' requires the knowledge of $sk^{in}_{|I'_2}$, from Lemma 10 we can again "connect" the two executions: it suffices to take $J = I'_2$ (which is possible since by Lemma 9 we have $|I'_2| \leq t$), which enables to perfectly simulate the shares $sk^{out}_{|J}$ which are then the same as the input shares $sk^{in}_{|I'_2}$, which are then perfectly simulated for the next sub-execution Q'. The simulation can then proceed as before from one sub-execution to the next; inductively the values of all t intermediate variables probed by the adversary can be perfectly simulated in each sub-execution. This terminates the proof of Theorem 1.

C Secure Multiplication with Linear Memory Complexity

In this section we show that the SecMult algorithm from [RP10] can be computed with $\mathcal{O}(n)$ memory instead of $\mathcal{O}(n^2)$. We first recall the original SecMult algorithm from [RP10], and then describe our variant SecMult' with $\mathcal{O}(n)$ memory complexity.

Algorithm 5 SecMult - masked multiplication over $\mathbb{F}_{2^k}$

Input: shares $a_1, \ldots, a_n$ such that $a = a_1 \oplus \cdots \oplus a_n$, and shares $b_1, \ldots, b_n$ such that $b = b_1 \oplus \cdots \oplus b_n$

Output: shares c_i such that $c_1 \oplus \cdots \oplus c_n = ab$

```
 1: for $i = 1$ to $n$ do
 2:     for $j = i + 1$ to $n$ do
 3:         $r_{i,j} \leftarrow \mathbb{F}_{2^k}$
 4:         $r_{j,i} \leftarrow (r_{i,j} \oplus a_i b_j) \oplus a_j b_i$
 5:     end for
 6: end for
 7: for $i = 1$ to $n$ do
 8:     $c_i \leftarrow a_i b_i$
 9:     for $j = 1$ to $n$, $j \neq i$ do $c_i \leftarrow c_i \oplus r_{i,j}$
10: end for
11: return $c_1, \ldots, c_n$
```

Algorithm 6 SecMult' - masked multiplication over $\mathbb{F}_{2^k}$, linear memory

Input: shares $a_1, \ldots, a_n$ such that $a = a_1 \oplus \cdots \oplus a_n$, and shares $b_1, \ldots, b_n$ such that $b = b_1 \oplus \cdots \oplus b_n$

Output: shares c_i such that $c_1 \oplus \cdots \oplus c_n = ab$

```
 1: for $i = 1$ to $n$ do
 2:     $c_i \leftarrow a_i b_i$
 3: end for
 4: for $i = 1$ to $n$ do
 5:     for $j = i + 1$ to $n$ do
 6:         $s \leftarrow \mathbb{F}_{2^k}$                              ▷ $s = r_{i,j}$
 7:         $s' \leftarrow (s \oplus a_i b_j) \oplus a_j b_i$         ▷ $s' = r_{j,i}$
 8:         $c_i \leftarrow c_i \oplus s$
 9:         $c_j \leftarrow c_j \oplus s'$
10:     end for
11: end for
12: return $c_1, \ldots, c_n$
```