Efficient and Provably Secure Methods for Switching from Arithmetic to Boolean Masking

Blandine Debraize

Gemalto, 6 rue de la Verrerie, 92197 Meudon Cedex, France
blandine.debraize@gemalto.com

Abstract. A large number of secret key cryptographic algorithms combine Boolean and arithmetic instructions. To protect such algorithms against first order side channel analysis, it is necessary to perform conversions between Boolean masking and arithmetic masking. Louis Goubin proposed in [7] an efficient method to convert from Boolean to arithmetic masking. However the conversion method he also proposed in [7] to switch from arithmetic to Boolean is less efficient and could be a bottleneck in some implementations. Two faster methods were proposed in [3] and [11], both using precomputed tables. We show in this paper that the algorithm in [3] is bugged, and propose an efficient correction. Then, we propose an alternative to the algorithm in [11] with a valuable timing/memory tradeoff. This new method offers better security in practice and is well adapted for 8-bit architectures in terms of time performance (3.3 times faster than Goubin's algorithm for one single conversion).

Key Words: side channel analysis, differential power analysis, Boolean masking, arithmetic masking, conversion from arithmetic to Boolean masking.

1 Introduction

In 1999, the concept of Differential Power Analysis (DPA) was introduced in [9] by Paul Kocher. It consists in retrieving information about the secret key of an algorithm by analyzing the power consumption curves generated by the device in which the algorithm is implemented, during its execution. It was extended to some other techniques like CPA (Correlation Power Analysis), and EMA (Electromagnetic Analysis), based on similar principles. All these attacks relying on physical leakage of an electronic device are more generically called side channel analysis.

Countermeasures were soon developed to thwart these attacks. The most commonly used method, initially proposed in [2] and [8], consists in splitting all key-dependant intermediate variables processed during the execution of the algorithm into several shares. The value of each share,

considered independently from the other ones, is randomly distributed and independent of the value of the secret key: thefefore, the power leakage of one share does not reveal any secret information. It is shown in [2] that the number of power curves needed to mount an attack grows exponentially with the number of shares. When only two shares are used, the method comes to masking all intermediate data with random. In this case it is said that the implementation is protected against first order DPA. For algorithms that combine Boolean and arithmetic operations, two different kinds of masking must be used: Boolean masking and arithmetic masking. A large number of algorithms have this property: all software oriented finalists of the eSTREAM stream cipher competition [5], some other stream ciphers like Snow 2.0 [4] and Snow 3G, the block cipher IDEA [10], and several hash function designs used for HMAC constructions. The security of DPA-protected implementations of such ciphers strongly depends on the security of conversions between arithmetic and Boolean masking in both directions.

Two secure conversion algorithms (one for each direction) were proposed by Goubin in [7], but the arithmetic to Boolean method of [7] is quite slow and can be a bottleneck in some implementations. Then a second arithmetic to Boolean algorithm using two precomputed tables was proposed by Jean-Sébastien Coron and Alexei Tchulkine in [3]. Finally, an extension of the method of [3] was proposed by Olaf Neiße and Jürgen Pulkus in [11], allowing to reduce memory consumption.

In this paper we first recall the mechanisms of these three methods, showing that the Coron-Tchulkine algorithm is not correct in most cases. Then we propose a modification of Coron-Tchulkine's algorithm, correcting the bug and improving time performance. We also propose a new fast and secure arithmetic to Boolean conversion technique. Finally we give some performance comparisons between all methods.

2 Definitions and previous work

The masking technique introduced in [2] and [8] consists in splitting each intermediate variable that appears in the cryptographic algorithm, using a secret sharing scheme. Therefore, an attacker must analyze multiple point distributions, which requires a number of power curves exponential in the number of shares. To protect implementations against first order DPA, this technique has to be applied with two shares.

For algorithms that combine Boolean and arithmetic functions, two kinds of masking are used:

1. *Boolean masking*: $x' = x \oplus r$
2. *Arithmetic masking*: $x' = x - r \bmod 2^K$.

Here $\oplus$ is the exclusive or. The variable x refers to the secret intermediate data, r to the random value used to obtain the masked data x', these three data having size K.

The conversion algoritms from one masking to another must also be secure against side channel analysis. This means that all intermediate variables must be independent of the secret data.

2.1 First secure method

In [7] Louis Goubin proposed an efficient method to convert a Boolean masking into an arithmetic masking, relying on the fact that the function $f_{x'}(r) = (x' \oplus r) - r$ is affine in r over the field with two elements.

An algorithm converting from arithmetic to Boolean masking was also proposed in [7], based on the following recursion formula:

$$(A + r) \oplus r = u_{K-1}, \text{ where: } \begin{cases} u_0 = 0, \\ \forall k \geq 0, u_{k+1} = 2[u_k \wedge (A \oplus r) \oplus (A \wedge r)]. \end{cases}$$

But this method is less efficient than from Boolean to arithmetic, as the number of operations is linear in the size of the intermediate data.

2.2 Coron-Tchulkine method

In [3], Jean-Sébastien Coron and Alexei Tchulkine proposed a second method to convert from arithmetic to Boolean masking. This method is based on the use of two precomputed tables. Let us recall its principle: two tables G and C are generated during the precomputation phase of the algorithm. Both tables have size 2^k, where k is the size of the processed data; the value of k is typically 4 or 8. For example if $k = 4$, a 32-bit variable is divided into 8 nibbles: the algorithm works then in 8 steps, each step processing one nibble of the 32-bit variable. Table G converts a nibble from arithmetic to Boolean masking, while Table C manages carries coming from the modular addition. Indeed, let us consider a masked data x' splitted into n nibbles $x'_{n-1}||...||x'_i||...||x'_0$: each value $x_i = x'_i + r$ can be possibly more than 2^k. In this case the carry must be added to the nibble x'_{i+1} before its conversion. As the carry value is correlated to the secret data, it must be masked. Therefore, for each input x'_i, the table

C outputs the carry value c masked by the addition of a random k-bit value γ. Both Tables G and C can be described as follows:

Algorithm 2.1: Precomputation of tables

Table G generation	Table C generation
Input: a nibble size k	Input: a k-bit value r.
1. Generate a random k-bit r	1. Generate a random k-bit γ
2. For $A = 0$ to $2^k - 1$ do	2. For $A = 0$ to $2^k - 1$ do
$G[A] = (A + r) \oplus r$	$\mathrm{C[A]} \leftarrow \begin{cases} \gamma, \text{ if } A + r < 2^k \\ \gamma + 1 \bmod 2^k, \text{ if } A + r \geq 2^k \end{cases}$
3. Output G and r.	3. Output C and γ.

Finally the conversion phase can be described by the following algorithm, where the symbol $||$ means concatenation:

Algorithm 2.2: Conversion of a $(n \cdot k)$-bit variable

Input: (A, R) such that $x = A + R \bmod 2^{n \cdot k}$
and r, γ generated during precomputation phase

1. For $i = 0$ to $n - 1$ do
2. Split A into $A_h||A_l$ and R into $R_h||R_l$ such that A_l and R_l have size k
3. $A \leftarrow A - r \bmod 2^{(n-i) \cdot k}$
4. $A \leftarrow A + R_l \bmod 2^{(n-i) \cdot k}$
5. if $i < n - 1$ do
6. $A_h \leftarrow A_h + C[A_l] \bmod 2^{(n-i-1) \cdot k}$
7. $A_h \leftarrow A_h - \gamma \bmod 2^{(n-i-1) \cdot k}$
8. $x'_i \leftarrow G[A_l] \oplus R_l$
9. $x'_i \leftarrow x'_i \oplus r$
10. $A \leftarrow A_h$ and $R \leftarrow R_h$
11. Output $x' = x'_{n-1}||...||x'_i||...||x'_0$

Let us specify that the value A_h and A_l are updated at the same time as A: the value A is splitted into A_h and A_l throughout all the algorithm. This remark is true for all conversion algorithms of this paper.

But this algorithm is actually not correct in case $n > 2$. Indeed, let us suppose that the following propositions are true together:

1. $n > 2$,
2. γ takes the value $2^k - 1$,
3. The carry equals 1.

Then in the first iteration of the loop of Algorithm 2.2 (i.e. when $i = 0$), the size of A_h is greater than k. In this case the value $A_h + C[A_l] - \gamma$ does not equal $A_h + 1$. Thus the table C generation must be modified to obtain an algorithm outputting always the correct value.

In Section 3, we propose a method combining correction and time performance improvement of Coron-Tchulking method.

2.3 Neiße-Pulkus method

A third method was proposed in 2004 by Olaf Neiße and Jürgen Pulkus in [11]. As Coron-Tchulkine algorithm, it is based on the precomputation of tables. The principle is first to store the values of each possible nibble updated in the new masking mode in a 2^k-entry table, as Table G of Section 2.2. The carry is also stored in a 2^k-entry table C, but contrary to Coron-Tchulkine method, it is here stored unmasked. Tables G and C can be possibly combined in one table to reduce RAM space requirements.

The carry is masked during conversion step by the fact that sometimes the direct value of the intermediate variable is processed and sometimes its complement is processed. A random bit z generated at the beginning of each conversion step decides if the complement is used or not.

The method is based on the fact that, for any l-bit value x and its complement $\bar{x}$, the equation $x + \bar{x} + 1 = 2^l$ holds. Thus for a bit z, if $\tilde{x}$ denotes x when $z = 0$, and $\bar{x}$ when $z = 1$, we obtain $\tilde{x} = x - z \bmod 2^l$. And for two l-bit values x_1 and x_2 it can easily proved that $\widetilde{x_1 + x_2} = \widetilde{x_1} + \widetilde{x_2} + z \bmod 2^l$.

Let us take the notations of Section 2.2 (Algorithm 2.2): a random k-bit mask r used as input and output mask of Table G, and (A, R) such that $x = A + R \bmod 2^{n \cdot k}$. Then $\tilde{x} - (r||\cdots||r) = \tilde{A} + \tilde{R} - (r||\cdots||r) + z$, where each nibble of $\tilde{x} - (r||\cdots||r)$ is taken as input of the table, the output being the corresponding nibble of $\tilde{x} \oplus (r||\cdots||r)$. Thus the bit z must be added to the intermediate variable A at the beginning of each conversion step. At the end, as $\tilde{A} \oplus \tilde{R} = A \oplus R$, the correct result is then obtained.

The main principle of the conversion algorithm of [11] can be summarized by Algorithm 2.3.

Security of the method. The authors of [11] claim that their algorithm is resistant against DPA. From a DPA-only point of view, the value of the bit $C[A_l]$ (line 9 of Algorithm 2.4) is indeed independent of the value of the secret data, due to the fact that for a k-bit value w, the number of k-bit values r such that $w + r \geq 2^k$ is w, and the number of k-bit values r such that $\bar{w} + r \geq 2^k - 1$ is $2^k - w - 1$, inducing a constant number $2^k - 1$ of possible r contributing to a non-zero carry.

But in practice this algorithm may pose a security problem, as the value $-z \bmod 2^{n \cdot k}$ is manipulated several times during one conversion step: as z is a random bit, this value is either `0` or `0xFF...FF`. It could be distinguished by the attacker in some context, using SPA techniques. With this information, the attacker could mount a DPA attack, using the

fact that the carries are then unmasked. This implies that the behavior of the component in terms of power and electromagnetic leakage must be studied very carefully before choosing this conversion method.

Algorithm 2.3: Conversion of a $(n \cdot k)$-bit variable

Input: (A, R) such that $x = A + R \bmod 2^{n \cdot k}$ and r, generated during precomputation phase

1. Generate a random bit z
2. $Z \leftarrow -z \bmod 2^{n \cdot k}$
3. $A \leftarrow (A \oplus Z) - (r||\cdots||r) \bmod 2^{n \cdot k}$
4. $R \leftarrow R \oplus Z$
5. For $i = 0$ to $n - 1$ do
6. Split A into $A_h||A_l$ and R into $R_h||R_l$ such that A_l and R_l have size k
7. $A \leftarrow A + R_l \bmod 2^{(n-i) \cdot k}$
8. if $i < n - 1$ do
9. $A_h \leftarrow A_h + C[A_l] \bmod 2^{(n-i-1) \cdot k}$
10. $x'_i \leftarrow G[A_l] \oplus R_l$
11. $A \leftarrow A_h$ and $R \leftarrow R_h$
12. Output $x' = (x'_{n-1}||...||x'_i||...||x'_0) \oplus (r||\cdots||r)$

The final method proposed in [11] is slightly different from Algorithm 2.3, as a technique is added in [11] to reduce memory consumption (it is recalled in Section 4.2). The use of this technique implies a decrease in the speed of the algorithm. As our paper mainly focuses on time performance, we propose in Appendix C another modification of the algorithm in order to reach maximal speed. This algorithm is used for the performance tests described at section 5.

3 Correction and improvement of Coron-Tchulkine method

We show in Appendix A that the immediate possible corrections of Coron-Tchulkine method keeping the size of the carry's mask γ to k bits are not first order DPA resistant: the size of γ must be at least $(n-1) \cdot k$ to obtain complete independence of intermediate data from the secret key.

Let us remark that both the information provided by Table G of Coron-Tchulkine method (update of the nibble in the new masking mode) and the information of Table C (additively masked carry) can be summarized in one unique table T whose outputs have size $n \cdot k$:

Algorithm 3.1: Table T generation

1. Generate a random k-bit r and a random $(n \cdot k)$-bit γ
2. For $A = 0$ to $2^k - 1$ do
   $T[A] = ((A + r) \oplus r) + \gamma \bmod 2^{n \cdot k}$
3. Output T, r and γ

The number of entries of the table is 2^k, and the size of each entry is $\frac{n \cdot k}{8}$ bytes. For typical values $k = 8$ and $n = 4$, the memory consumption is doubled here compared to the adaptation of Neiße-Pulkus method proposed in Appendix C.

The resulting conversion algorithm is as follows:

Algorithm 3.2: Conversion of a $(n \cdot k)$-bit variable

Input: (A, R) such that $x = A + R \bmod 2^{n \cdot k}$
and r, γ generated during precomputation phase

1. For $i = 0$ to $n - 1$ do
2. Split A into $A_h||A_l$ and R into $R_h||R_l$, such that A_l and R_l have size k
3. $A \leftarrow A - r \bmod 2^{(n-i) \cdot k}$
4. $A \leftarrow A + R_l \bmod 2^{(n-i) \cdot k}$
5. $A \leftarrow A_h||0 + T[A_l] \bmod 2^{n \cdot k}$
6. $A \leftarrow A - \gamma \bmod 2^{n \cdot k}$
7. $x'_i \leftarrow A_l \oplus R_l$
8. $x'_i \leftarrow A_l \oplus r$
9. $A \leftarrow A_h$ and $R \leftarrow R_h$
10. Output $x' = x'_0||...||x'_i||...||x'_{n-1}$

Here, as T's outputs have the same size as the processed data, if the value $A+r$ is greater than 2^k during the precomputation of T, the $(k+1)^{\text{th}}$ least significant bit of $T[A]$ is automatically set to 1 before being masked by the addition of γ. During the conversion algorithm, the carry is added to the current variable (line 5) at the same time as the nibble A_l is updated.

The use of one table instead of two is clearly an advantage in terms of time performance. But it is still possible to reduce the execution time of the conversion algorithm by moving some instructions out of the loop. These improvements are described in Appendix B.

4 New Method

In terms of performance, for a 16-bit or an 8-bit processor, the drawback of the method described in Section 3 is the fact that the size of the manipulated data is the same as the size of the intermediate data of the algorithm. Indeed, the typical size for intermediate data is 32 bits: the time of the conversion algorithm is then multiplied by 2 with a 16-bit processor, and by 4 with an 8-bit processor. In this section we propose a method more appropriate for processors whose registers have size smaller than the intermediate data of the algorithm.

4.1 Principle

Let us remark that using precomputed tables to keep data masked during the algorithm execution comes to treating masked information as memory address information. As a carry is a 1-bit information, our goal in this section is to apply this principle to 1-bit information.

Let us suppose that instead of being masked arithmetically as proposed in Sections 2.2 and 3, carries are protected by Boolean masks. The protection comes then to adding by exclusive or a random bit to the carry value. For example, if we call ρ such a random bit, a 2-entry table C can be generated during the precomputed step in the following way:

Algorithm 4.1: Table C generation
1. Generate a random bit ρ and a random $(n \cdot k)$-bit value λ
2. $C[\rho] = \lambda$
3. $C[\rho \oplus 1] = \lambda + 1 \bmod 2^{n \cdot k}$
4. Output C and λ.

Now let us suppose that a carry c, protected by the Boolean mask ρ, is manipulated during the conversion algorithm. Thus the masked value $b = c \oplus \rho$ can be used in the following way to add the carry c to the value A_h in a secure way:

Algorithm 4.2: Carry addition
Inputs: – a value A_h (masked arithmetically), – a carry bit b (c masked in a Boolean way) – C, λ generated during precomputation phase
1. $A_h = A_h + C[b] \bmod 2^{n \cdot k}$
2. $A_h = A_h - \lambda \bmod 2^{n \cdot k}$
3. Output A_h

It is easy to convince oneself about:

1. *The correctness of the method*: whatever the value of ρ is, the value $C[b]$ is equal to the carry c added to λ modulo $2^{n \cdot k}$.
2. *The resistance against order 1 DPA*: all processed intermediate variables are independent of the unmasked values. The masked carry is treated as information about the address of a RAM location. This address is independent from the value of the carry, as it changes from one execution to the other.

In next Section, we propose a method combining the information of both the nibble to be updated and of the carry masked by exclusive or, using this combination as an address in a conversion table.

4.2 Algorithm

In this section a new method for switching from arithmetic to Boolean masking is proposed. As the method described in Section 3, it requires the precomputation of one table T whose outputs must contain information about both the nibble updated in the new masking mode and the next carry bit. Here the output of T is directly the value $(A+r+c) \oplus r$, where c is the carry resulting from the previous addition.

The precomputed table T has the following properties:

- The carry value is masked by exclusive or with a random bit.
- During conversion phase, the choice of the address in the table not only depends on the value of the nibble but also on the value of the maksed previous carry. This implies T has size 2^{k+1}. The needed amount of memory is then doubled compared to Neiße-Pulkus method, and is the same compared to the correction of Coron-Tchulkine method for typical values $k = 8$ and $n = 4$.

The value of the random bit used to mask carries decides during precomputation step if the values of the addresses of nibbles of type $(A+r) \oplus r$ are greater or less than the addresses of nibbles of type $(A+r+1) \oplus r$. And during conversion step, the value of the masked carry is used to compute the address of the next nibble to be loaded from the table.

The method is outlined in Algorithms 4.3 and 4.4. Here again, all processed variable are independent of secret data, inducing resistance of the algorithm against first order DPA.

Algorithm 4.3: Table T generation

1. Generate a random k-bit r and a random bit ρ
2. For $A = 0$ to $2^k - 1$ do
   $T[\rho||A] = (A + r) \oplus (\rho||r)$
   $T[(\rho \oplus 1)||A] = (A + r + 1) \oplus (\rho||r)$
3. Output T, r and ρ

If $k = 8$, the time of the conversion phase is optimized. But in this case the size of the output data of the table is $k + 1 = 9$ bits. This implies that this data needs two bytes to be stored, and the size of the table in RAM is then 1024 bytes. This amount of memory is possible today on many embedded components, but could still be too large in some cases. As in [11], the fact that the Boolean masking of a secret data $x'_b = x \oplus r$ and the arithmetic masking of the same data $x'_a = x - r \mod 2^k$ have always the same least significant bit can be used to reduce the size of the

precomputed table. Thus storing the least significant bit of $(A + r) \oplus r$ or of $(A + r + 1) \oplus r$ is not necessary. The place of this useless bit can be taken by the carry bit. The resulting algorithm is then slower but the needed amount of memory is reduced by half. In our method it is then 512 bytes.

Algorithm 4.4: Conversion of a $n \cdot k$-bit variable

Input: (A, R) such that $x = A + R \bmod 2^{n \cdot k}$,
r, ρ generated during precomputation phase

1. $A \leftarrow A - (r||...||r||...||r) \bmod 2^{n \cdot k}$
2. $\beta \leftarrow \rho$
3. For $i = 0$ to $n - 1$ do
4. Split A into $A_h||A_l$ and R into $R_h||R_l$, such that A_l and R_l have size k.
5. $A \leftarrow A + R_l \bmod 2^{(n-i) \cdot k}$
6. $\beta||x'_i \leftarrow T[\beta||A_l]$
7. $x'_i \leftarrow x'_i \oplus R_l$
8. $A \leftarrow A_h$ and $R \leftarrow R_h$
9. Output $x' = (x'_0||...||x'_i||...||x'_{n-1}) \oplus (r||...||r||...||r)$

5 Performance Tests

In this section we propose some performance comparisons between Goubin's method and the three methods based on precomputed tables. The versions chosen for the tests are the ones that are optimized in terms of time performance:

- Algorithms C.1 and C.2 (Appendix C) for the modified Neiße-Pulkus method (named Mod. N.-P. in Tables 1, 2 and 3).
- Algorithms B.1 and B.2 (Appendix B) for the correction of Coron-Tchulkine method (named Mod. C.-T. in Tables 1, 2 and 3).
- Algorithms 4.3 and 4.4 (Section 4.2) for the new method proposed in this paper.

We first chose to perform C implementations of these algorithms and test them on 8051 architectures: one 8-bit and one 16-bit microprocessors. Table 1 and Table 2 summarize the performance comparison results for both components. These results are given in clock cycles numbers, computed with the help of a simulation tool. The size of the data to be converted from arithmetic to Boolean masking is 32 bits, as it is the most common size for intermediate data of cryptographic algorithms. For the table-based algorithms, two nibble sizes were tested: $k = 4$ and $k = 8$. The size of the precomputed tables in RAM are given in number of bytes.

We also performed performance comparison tests for the same algorithms in ARM assembler on a 32-bit 26 MHz microprocessor. In Table 3 the time results of these tests are given in microseconds.

Table 1. Smart card 8-bit microprocessor

	Goubin's method	Mod. N.-P.		Mod. C.-T.		New method	
		$k = 4$	$k = 8$	$k = 4$	$k = 8$	$k = 4$	$k = 8$
Precomputation time	10325	2562	40274	18589	109391	3166	93007
Conversion time	39213	15479	9208	13969	7060	11720	6111
Table size	0	16	512	64	1024	32	1024

Table 2. Smart card 16-bit microprocessor

	Goubin's method	Mod. N.-P.		Mod. C.-T.		New method	
		$k = 4$	$k = 8$	$k = 4$	$k = 8$	$k = 4$	$k = 8$
Precomputation time	86	377	3734	921	5933	439	5174
Conversion time	934	558	308	512	274	445	257
Table size	0	16	512	64	1024	32	1024

The generation of random numbers is required by all methods. For Goubin's algorithm (see [7]), the size of the random value is 32 bits, and only one such random word is necessary for each execution. For this reason, the time of this generation is set in precomputation step. Thus we remark that, depending on the chip, the time of the generation of the random values is generally not negligible for these conversion algorithms. This also explains the time difference between the precomputation steps of the Neiße-Pulkus method and of the Coron-Tchulkine method (one random byte is generated in N.-P. against four in C.-T. method).

From the figures given in Table 1 and Table 2, we remark that the new method is more efficient on these microcontrollers than the improved correction of Coron-Tchulkine algorithm. This confirms the fact that the

Table 3. Smart card 32-bit microprocessor

	Goubin's method	Mod. N.-P.		Mod. C.-T.		New method	
		$k = 4$	$k = 8$	$k = 4$	$k = 8$	$k = 4$	$k = 8$
Precomputation time	15.1	9.6	156.2	25.5	188.8	12.1	180.3
Conversion time	32.9	12.9	10.3	12.1	8	14.9	9.2
Table size	0	16	512	64	1024	32	1024

new method is better adapted to 8-bit and 16-bit architectures. On the 8-bit architecture, the conversion step of the new method is the fastest.

Choosing $k = 4$, the improved Neiße-Pulkus and the new method are both faster than Goubin's algorithm on all architectures, even for a single conversion, with a small amount of needed memory. Neiße-Pulkus method is about twice faster on the 32-bit microcontroller, and the new method about three times faster on the 8-bit microcontroller.

6 Conclusion

In this paper we sought the fastest methods for switching from arithmetic to Boolean masking. We first analyzed the two known methods [3, 11] based on precomputed tables: we showed that the algorithm proposed in [3] is not correct and proposed an improved correction. We also proposed a new method, which is well adapted for 8-bit architectures in terms of time preformance. As the correction of [3], it offers better security against side channel analysis in practice than the algorithm in [11].

References

1. Jean-Philippe Aumasson, Luca Henzen, Willi Meier, and Raphael.C.-W Phan. SHA-3 proposal BLAKE. Submission to the SHA-3 Competition. Available at `http://csrc.nist.gov/groups/ST/hash/sha-3/`.
2. Suresh Chari, Charanjit S. Jutla, Josyula R. Rao, and Pankaj Rohatgi. Towards sound approaches to counteract power-analysis attacks. In *CRYPTO*, volume 1666 of *Lecture Notes in Computer Science*, pages 398–412. Springer, 1999.
3. Jean-Sébastien Coron and Alexei Tchulkine. A new algorithm for switching from arithmetic to boolean masking. In *CHES*, volume 2779 of *Lecture Notes in Computer Science*, pages 89–97. Springer, 2003.
4. Patrik Ekdahl and Thomas Johansson. A new version of the stream cipher snow. In *Selected Areas in Cryptography*, volume 2595 of *Lecture Notes in Computer Science*, pages 47–61. Springer, 2002.

5. eSTREAM. ECRYPT Stream Cipher Project, IST-2002-507932. Available at `http://www.ecrypt.eu.org/stream/`.
6. Niels Ferguson, Stefan Lucks, Bruce Schneier, Doug Whiting, Mihir Bellare, Tadayoshi Kohno, Jon Callas, and Jesse Walker. The Skein hash function family. Available at `http://csrc.nist.gov/groups/ST/hash/sha-3/`.
7. Louis Goubin. A sound method for switching between boolean and arithmetic masking. In *CHES*, volume 2162 of *Lecture Notes in Computer Science*, pages 3–15. Springer, 2001.
8. Louis Goubin and Jacques Patarin. Des and differential power analysis (the "duplication" method). In *CHES*, volume 1717 of *Lecture Notes in Computer Science*, pages 158–172. Springer, 1999.
9. Paul C. Kocher, Joshua Jaffe, and Benjamin Jun. Differential power analysis. In *CRYPTO*, volume 1666 of *Lecture Notes in Computer Science*, pages 388–397. Springer, 1999.
10. Xuejia Lai and James L. Massey. A proposal for a new block encryption standard. In *EUROCRYPT*, volume 473 of *Lecture Notes in Computer Science*, pages 389–404. Springer, 1990.
11. Olaf Neiße and Jürgen Pulkus. Switching blindings with a view towards idea. In *CHES*, volume 3156 of *Lecture Notes in Computer Science*, pages 230–239. Springer, 2004.

Appendices

A Security weaknesses of two immediate modifications of Coron-Tchulkine method

Two immediate modifications of the method proposed in [3] seems possible. We call these two possible tables C' and C'':

Algorithm A.1: Carry table C' generation	**Algorithm A.2:** Carry table C'' generation
Input: a random r of k bits. 1. Generate a random k-bit γ such that $\gamma < 2^k - 1$ 2. For $A = 0$ to $2^k - 1$ do C'[A] $\leftarrow \begin{cases} \gamma, \text{ if } A + r < 2^k \\ \gamma + 1, \text{ if } A + r \geq 2^k \end{cases}$ 3.Output C' and γ.	Input: a random r of k bits. 1. Generate a random k-bit γ 2. For $A = 0$ to $2^k - 1$ do C"[A] $\leftarrow \begin{cases} \gamma, \text{ if } A + r < 2^k \\ \gamma + 1, \text{ if } A + r \geq 2^k \end{cases}$ 3. Output C'' and γ.

But both corrections imply that some manipulated data are not completely decorrelated from the value of the secret data. Indeed, the least significant bit of the output of Table C' is correlated to the corresponding carry bit. Let us compute the correlation coefficient.

Using the notations from Section 2.2, let us call b_0 the least significant bit of the output of Table C', and c the corresponding carry value.

Let us define a set χ:

$$\begin{aligned}\chi = \{\ (A, r, \gamma) : A &\in \mathbb{N} \text{ such that } A < 2^n, \\ r &\in \mathbb{N} \text{ such that } r < 2^n, \\ \gamma &\in \mathbb{N} \text{ such that } \gamma < 2^n - 1\}\end{aligned}$$

The correlation coefficient between b_0 and c conditionned on the subset χ is as follows:

$$\begin{aligned}\mathcal{C}or\,(b_0\ ,\ c \mid \chi) &= |\mathrm{P}(b_0(x) = c(x) \mid x \in \chi) - \mathrm{P}(b_0(x) \neq c(x) \mid x \in \chi)| \\ &= \left| \frac{1}{\#\chi} \sum_{x \in \chi} (-1)^{b_0(x) \oplus c(x)} \right|\end{aligned}$$

Actually the value $b_0(x) \oplus c(x)$ neither depends on the value of A nor on the value of r, but only depends on the value of the least significant bit of γ. We call this bit γ_0. As 2^{k-1} times out of $2^k - 1$, $\gamma_0 = 0$, and $2^{k-1} - 1$ times out of $2^k - 1$, $\gamma_0 = 1$, we have:

$$\begin{aligned}\mathcal{C}or\,(b_0\ ,\ c \mid \chi) &= \left| \frac{1}{2^k - 1} \sum_{x \in \chi} (-1)^{\gamma_0(x)} \right| \\ &= \left| \frac{1}{2^k - 1} \left(\sum_{\gamma < 2^k - 1, \gamma_0 = 0} (-1)^0 + \sum_{\gamma < 2^k - 1, \gamma_0 = 1} (-1)^1 \right) \right| \\ &= \left| \frac{1}{2^k - 1} \left(2^{k-1} - (2^{k-1} - 1) \right) \right| = \frac{1}{2^k - 1}\end{aligned}$$

If $k = 4$, the correlation coefficient is then $\frac{1}{15}$.

It could be shown in a similar way that the correlation coefficient between the most significant bit of the output of Table C'' (the $(k+1)^{\text{th}}$ bit) and the carry bit has value $\frac{31}{256}$.

Many other modifications of carry Table C are possible, each one corresponding to a specific interval in which the random carry's mask γ takes its values. In each of these cases it can be shown in a way similar as above that if this interval is smaller than $[0, 2^{(n-1)\cdot k} - 1]$, a correlation exists between the output of the table and the carry bit.

In case the interval is $[0, 2^{(n-1)\cdot k} - 1]$, as the addition of Table C's output is performed modulo at most $2^{(n-1)\cdot k}$ (line 6 of Algorithm 2.2), the precomputed addition "γ+carry" can be also performed modulo $2^{(n-1)\cdot k}$ without the lack of correctness of the initial Coron-Tchulkine method.

B Time Performance improvement of Algorithm 3.2

To reduce the execution time of Algorithm 3.2, some instructions can be set out of the loop. Three of them can be removed from the loop without weakening security:

- The arithmetic masking with the random r (line 3) can be performed before the loop.
- The subtraction of the value γ (line 6) can be performed before the loop (inducing a modification of the precomputed table T).
- The Boolean unmasking with the random r (line 8) can be performed after the loop.

Indeed, in case these instructions are moved out of the loop, all nibbles of A but one remain masked with the initial mask R during the execution of the algorithm, and the lasting nibble is masked by the random value r. All intermediate variables are then independent of secret data throughout the execution.

Some extra calculations must be performed during precomputation step, allowing to minimize the cost of the subtraction of γ during conversion step. The improved version of the method is then as follows:

Algorithm B.1: Table T generation

1. Generate a random k-bit r and a random $((n-1)\cdot k)$-bit γ
2. Compute $\Gamma = \sum_{i=1}^{k-1} 2^{i\cdot k} \cdot \gamma \bmod 2^{n\cdot k}$
3. $\gamma' \leftarrow \gamma || r$
4. For $A = 0$ to $2^k - 1$ do
   $T[A] = \left(A + \gamma' \bmod 2^{n\cdot k}\right) \oplus r$
5. Output T, r and Γ

Algorithm B.2: Conversion of a $n \cdot k$-bit variable

Input: (A, R) such that $x = A + R \bmod 2^{n \cdot k}$
and r, E generated during precomputation phase

1. $A \leftarrow A - (r||...||r||...||r) \bmod 2^{n \cdot k}$
2. $A \leftarrow A - \Gamma \bmod 2^{n \cdot k}$
3. For $i = 0$ to $n - 1$ do
4. Split A into $A_h||A_l$ and R into $R_h||R_l$, such that A_l and R_l have size k
5. $A \leftarrow A + R_l \bmod 2^{(n-i) \cdot k}$
6. $A \leftarrow A_h||0 + T[A_l] \bmod 2^{n \cdot k}$
7. $x'_i \leftarrow A_l \oplus R_l$
8. $A \leftarrow A_h$ and $R \leftarrow R_h$
9. Output $x' = (x'_0||...||x'_i||...||x'_{n-1}) \oplus (r||...||r||...||r)$

C Time Performance improvement of Neiße-Pulkus method.

The principle of the method described in [11] is a direct extension of the Coron-Tchulkine method. Therefore the two precomputed tables of [11] can be generated at the same time in one unique table exactly the same way as proposed in section 3 for Coron-Tchulkine method.
The adapted version of Neiße-Pulkus method is then as follows:

Algorithm C.1: Table T generation

1. Generate a random k-bit r
2. For $A = 0$ to $2^k - 1$ do
   $T[A] = (A + r) \oplus r$
3. Output T and r

Algorithm C.2: Conversion of a $n \cdot k$-bit variable

Input: (A, R) such that $x = A + R \bmod 2^{n \cdot k}$
and r generated during precomputation phase

1. Generate a random bit z
2. $Z \leftarrow -z \bmod 2^{n \cdot k}$
3. $A \leftarrow (A \oplus Z) - (r|| \cdots ||r) \bmod 2^{n \cdot k}$
4. $R \leftarrow R \oplus Z$
5. For $i = 0$ to $n - 1$ do
6. Split A into $A_h||A_l$ and R into $R_h||R_l$, such that A_l and R_l have size k
7. $A \leftarrow A + R_l \bmod 2^{(n-i) \cdot k}$
8. $A \leftarrow A_h||0 + T[A_l] \bmod 2^{n \cdot k}$
9. $x'_i \leftarrow A_l \oplus R_l$
10. $A \leftarrow A_h$ and $R \leftarrow R_h$
11. Output $x' = (x'_0||...||x'_i||...||x'_{n-1}) \oplus (r||...||r||...||r)$

Let us remark that the instructions inside the loop are the same in algorithm B.2 and in algorithm C.2. Concerning memory consumption, for typical values $k = 8$ and $n = 4$, the size of T's outputs is here 9 bits: the required amount of memory is then 512 bytes. It is reduced by half compared to the improved Coron-Tchulkine algorithm of Appendix B.