Algorithms for switching between block-wise and arithmetic masking

Evgeny Alekseev[1][0000−0002−1279−0359] and Andrey Bozhko[1,2][0000−0003−1424−608X]

[1] CryptoPro LLC, Moscow, Russia
{alekseev, bozhko}@cryptopro.ru
[2] Lomonosov Moscow State University, Moscow, Russia

Abstract. The task of ensuring the required level of security of information systems in the adversary models with additional data obtained through side channels (a striking example of implementing threats in such a model is a differential power analysis) has become increasingly relevant in recent years. An effective protection method against side-channel attacks is masking all intermediate variables used in the algorithm with random values. At the same time, many algorithms use masking of different kinds, for example, Boolean, byte-wise, and arithmetic; therefore, a problem of switching between masking of different kinds arises. Switching between Boolean and arithmetic masking is well studied, while no solutions have been proposed for switching between masking of other kinds. This article recalls the requirements for switching algorithms and presents algorithms for switching between block-wise and arithmetic masking, which includes the case of switching between byte-wise and arithmetic masking.

Keywords: side-channel attacks · masking techniques · byte-wise masking · mask switching.

1 Introduction

The concept of Differential Power Analysis was first presented by Paul Kocher et al. in 1999 in [1]. The efficiency of the method, which consists in obtaining information about the secret key of a cryptographic algorithm by analyzing the power consumption of the device during multiple executions of the algorithm, turned out to be extraordinarily high. DPA can be applied to a large variety of symmetric and asymmetric algorithms. Soon some other techniques, based on analyzing the physical leakage of the device, were proposed. This class of attacks is usually referred to as side-channel attacks.

With the development of side-channel attacks, some countermeasures were proposed. In [2] and [3] the most commonly used method was introduced. It consists in separating all key-dependent intermediate variables during the computation of the algorithm into several independent shares. Since these shares are viewed as independent and random, the leakage of a few shares (less than a

number of all shares) doesn't reveal any information. When only two shares are used, the method is equivalent to masking all sensitive variables with random values.

In some cryptographic algorithms (for example, IDEA [4], RC6 [5], SPECK [6]), different kinds of masking are used – Boolean, byte-wise, arithmetic, etc. Therefore, a problem of securely switching between masking of different kinds arises. Since Boolean and arithmetic maskings are the most commonly used, switching between these kinds of masking is well-studied with many efficient and secure algorithms proposed [7,8,9]. However, some high-efficient implementations of cryptographic algorithms require different kinds of masking – for example, in SIMD implementation of Magma [10] byte-wise masking is used. Therefore, algorithms for switching between less common kinds of masking are needed.

In this paper we first introduce a new kind of masking – block-wise masking, which generalizes Boolean, arithmetic, and byte-wise masking. Then we recall the main requirements for switching algorithms and some ideas used in Boolean-arithmetic switching algorithms. Finally, we propose secure algorithms for switching between byte-wise and arithmetic masking.

2 Definitions

The masking technique was first proposed in [2] and [3] and is designed to protect against so called first-order attacks (described, for example, in [1]). It consists in splitting each sensitive variable into two shares – a protected data and a mask. Depending on which the algorithm is being protected, different kinds of masking are used. The two most common are:

1. *Boolean masking*: $X = x \oplus r$;
2. *Arithmetic masking*: $A = x + r \bmod 2^n$,

where variable x is the protected data, r is the random mask, n is some natural number and $\oplus$ is the exclusive or operation. Hereafter "$\bmod 2^n$" will be omitted when the context is clear.

However, some implementations of cryptographic algorithms make the use of less common kinds of masking. One of them is

3. *Byte-wise masking*: $O = x \boxplus_8 r$,

where $\boxplus_8$ refer to byte-wise addition – each operand is treated as a sequence of bytes, which are summed independently modulo 2^8. We will also denote byte-wise subtraction by $\boxminus_8$.

All these kinds of masking can be generalized by the following definition.

Definition 1. *Let $n = l \cdot k$, where $k, l \in \mathbb{N}$. Let us denote by a_i the i-th rightmost block of bit length k of a variable $a = a_{l-1} \| a_{l-2} \| \dots \| a_0$, where $\|$ means concatenation. Then the block-wise masking (ADDk) of $x \in \mathbb{Z}_{2^n}$ with the mask $r \in \mathbb{Z}_{2^n}$ is the following value:*

$$B = (x_{l-1} + r_{l-1}) \| (x_{l-2} + r_{l-2}) \| \dots \| (x_0 + r_0),$$

where the addition of variables x_i and r_i is the addition modulo 2^k. Similarly to byte-wise addition and subtraction we will denote block-wise addition and subtraction by $\boxplus_k$ and $\boxminus_k$ respectively (we will omit the index when context is clear in the sequel).

Thus, Boolean masking, byte-wise masking and arithmetic masking can be viewed as block-wise masking with $k = 1$, $k = 8$ and $k = n$ respectively.

3 Switching between different kinds of masking

When different kinds of masking are used in one cryptographic algorithm the problem of securely switching between them arises. A switching algorithm has to calculate the value of the data under the mask of one kind by its value under the mask of the other kind and the value of the mask itself. The switching algorithm has to meet the following criteria:

1. All intermediate variables must be independent of the data to be masked. In other words, the algorithm has to be secure against side-channel attacks.
2. The algorithm has to run in constant time and memory.
3. The algorithm has to be correct and efficient.

The first criterion is caused by the following fundamental hypothesis, as formulated in [7], for the applicability of the Differential Power Analysis (and most of the other side-channel attacks) to a certain cryptographic algorithm.

Fundamental hypothesis ([7]). There exists an intermediate variable, that appears during the computation of the algorithm, such that knowing a few key bits (in practice less than 32 bits) allows us to decide whether two inputs (respectively two outputs) give or not the same value for this variable.

Let us consider the 'naive' algorithm for switching from Boolean masking to arithmetic. We have $X = x \oplus r$ and we want to obtain $A = x + r$. The straightforward method is to compute the following expression: $A = (X \oplus r) + r$. This method is, surely, correct since $(X \oplus r) + r = (x \oplus r \oplus r) + r = x + r$, which equals A by definition. However, when we compute $X \oplus r$, we get x in plain, which contradicts the first criterion and allows us to successfully mount the DPA attack.

The second 'naive' way for switching from Boolean masking to arithmetic is the following. We could have first calculated $X' = X + r$ and then $A' = X' \oplus r$. This algorithm is rather secure as x here is always proceeded under some random mask, however, it is easy to see, that $A' \neq A = x + r$, since $(X + r) \oplus r = (x \oplus r + r) \oplus r \neq (x \oplus r \oplus r) + r$. For similar reasons, we can't just xor (or arithmetically add) some temporary random value to hide computations and then just xor (or subtract) it in the end.

First secure methods for switching between Boolean and arithmetic masking were proposed by Goubin in [7]. The algorithm for switching from Boolean to

arithmetic masking is highly efficient and uses the fact that function $\Phi_X(r) = (X \oplus r) - r \bmod 2^n$ is affine over $GF(2)$. The algorithm for switching from arithmetic to Boolean masking was also proposed by the author, but it's less efficient as the number of operations is linear in the size of the intermediate data.

The second method for switching from arithmetic to Boolean masking was proposed by Jean-Sébastien Coron and Alexei Tchulkine in [11]. The method introduces an approach based on pre-computed tables. It was further developed by Nyiße and Pulkus in [12], but it may be vulnerable to SPA techniques in some contexts, as shown in [8]. Furthermore, in [8] Debraize shows, that the Coron-Tchulkine algorithm is bugged and proposes an efficient correction, as well as the new method with timing/memory tradeoff, also based on pre-computed tables. Higher-order switching algorithms were proposed and improved in [9,13,14,15].

Since Boolean and arithmetic maskings are the most common, switching between them was the focus of previous works and no secure switching algorithm between other kinds has not been yet proposed.

4 Switching between block-wise and arithmetic masking

In this section we propose two algorithms – one for switching from arithmetic to block-wise masking and the other one for switching in another direction. Both algorithms make use of pre-computed tables and develop ideas from [8]. In this section we fix some $n, k \in \mathbb{N}$, such that $n = l \cdot k, l \in \mathbb{N}$, where n is the size of the masked data (and 2^n is the modulus for arithmetic masking) and k is the size of the block in Definition 1.

4.1 Preparations. Pre-computed tables

Let us consider a byte-wise masked variable $B = B_{l-1} \| \dots \| B_0 = x_{l-1} \| \dots \| x_0 \boxplus r_{l-1} \| \dots \| r_0$. Here each block is processed independently and the value of the block B_i is the sum of x_i and r_i modulo 2^k. With the arithmetic masking the situation is different. Arithmetic addition can also be carried out block-by-block, however, it is possible, that the sum $x_i + r_i$ is greater than or equal to 2^k. In this case we have to add a carry to the next block x_{i+1} (that addition, in its turn, may trigger a carry addition to the next block if $x_{i+1} = 2^k - 1$). If we manage carries correctly, incrementing the follow-up x_{i+1} with a carry when needed, then each A_i will be equal to $x_i + r_i \bmod 2^k$. The goal of the pre-computed tables constructed in that section is to securely manage the carry addition (or subtraction) in both conversion algorithms.

An algorithm in Figure 1 generates two tables — T and G. The first one is table T of size 2^k bits. It determines by the value s of the sum modulo 2^k of some number $a \in \mathbb{Z}_{2^k}$ and a fixed $m \in \mathbb{Z}_{2^k}$ whether their arithmetic sum is greater than or equal to 2^k. In other words, $T[s]$ is equal to 1 if we need to add a carry to the next block. Each entry of table T is masked with a random bit b.

nk_table()

1 : $\gamma \xleftarrow{\mathcal{U}} \{0,1\}^n$
2 : $b \xleftarrow{\mathcal{U}} \{0,1\}$
3 : $m \xleftarrow{\mathcal{U}} \{0,1\}^k$
4 : **for** $s = 0 \dots 2^k - 1$ **do** :
5 : $\quad T[s] \leftarrow b \oplus (s < m)$
6 : $G[b] \leftarrow \gamma$
7 : $G[b \oplus 1] \leftarrow \gamma + 2^k$
8 : **return** T, G, γ, m

Fig. 1. Generation of tables T and G

Table G has just two entries, both of size n, and is of more technical purpose. It 'unmasks' the value from T and returns either the carry arithmetically masked with a random value $\gamma \in \mathbb{Z}_{2^n}$ or just the random value.

The tables generation requires 3 random generations and $2 \cdot 2^k + 2$ elementary operations.

4.2 Switching from arithmetic to block-wise masking

An algorithm for switching from arithmetic to block-wise masking is outlined in Figure 2. The conversion is carried out block-by-block, starting from the rightmost one. The algorithm gets a value $A = x + r$ and a mask r as an input and outputs a value $B = x \boxplus_k r$.

ADDnADDk(A, r)

1 : $T, G, \gamma, m \leftarrow nk\_table()$
2 : **for** $i = 0 \dots (l-1)$ **do** :
3 : $\quad A \leftarrow A + m$
4 : $\quad A \leftarrow A - r_i$
5 : $\quad A \leftarrow A - G[T[A_0]]$
6 : $\quad A \leftarrow A + \gamma$
7 : $\quad B_i \leftarrow A_0 \boxplus r_i$
8 : $\quad B_i \leftarrow B_i \boxminus m$
9 : $\quad A \leftarrow A_{l-1-i} \| \dots \| A_1$
10 : **return** $B = B_{l-1} \| \dots \| B_1 \| B_0$

Fig. 2. The algorithm for switching from arithmetic to block-wise masking

After generating the tables and random temporary masks γ and m with the algorithm $nk\_table()$ from Figure 1 conversion proceeds in the following way. Let us go through the first iteration of the cycle. First of all (lines 3 and 4), a k-bit temporary mask m is arithmetically added to A and the rightmost block r_0 of the long-term mask is subtracted from A to 'unmask' the rightmost block of x. At this point:

$$A = (x_{l-1} \| \dots \| x_1 \| x_0) + (r_{l-1} \| \dots \| r_1 \| m).$$

The second step (lines 5 and 6) is to subtract from A a carry, which may have appeared then we added the temporary mask m arithmetically, subtraction is performed under the mask γ from table G to hide the process. After subtracting the carry the rightmost block of x is masked with m in a block-wise way:

$$A = ((x_{l-1} \| \dots \| x_1 \| x_0) + (r_{l-1} \| \dots \| r_1 \| 0)) \boxplus (0 \| \dots \| 0 \| m).$$

Finally, we can compute a corresponding block of the block-wise masked B (lines 7 and 8) — we add to the rightmost block of A the rightmost block of long-term mask r_0 modulo 2^k and subtract the temporary mask m: $B_0 = A_0 \boxplus r_i \boxminus m = x_0 \boxplus m \boxplus r_i \boxminus m = x_0 \boxplus r_0$. The last step (line 9) is to cut off the rightmost block of A, which we have just remasked, and to proceed with the next iteration of the cycle. Subsequent blocks are processed in exactly the same way.

The algorithm requires $2 \cdot l$ calls to the pre-computed tables and $7 \cdot l$ elementary operations.

4.3 Switching from block-wise to arithmetic masking

$ADDkADDn(B, r)$

1 : $T, G, \gamma, m \leftarrow nk\_table()$
2 : **for** $i = (l-1) \dots 0$ **do** :
3 : $\quad A \leftarrow A \| B_i$
4 : $\quad A_0 \leftarrow A_0 \boxplus m$
5 : $\quad A_0 \leftarrow A_0 \boxminus r_i$
6 : $\quad A \leftarrow A + G[T[A_0]]$
7 : $\quad A \leftarrow A - \gamma$
8 : $\quad A \leftarrow A + r_i$
9 : $\quad A \leftarrow A - m$
10 : **return** A

Fig. 3. The algorithm for switching from block-wise to arithmetic masking

An algorithm for switching from block-wise to arithmetic masking is outlined in Figure 3. The algorithm gets the block-wise masked value B and the mask

r as an input and outputs the arithmetically masked value B. Analogically to the previous algorithm, the process is carried out block-by-block, but starting from the leftmost block. At the beginning of each iteration the processed block of byte-wise masked B is concatenated as the rightmost block to A (at the beginning A is initialized with an empty string). After that the conversion is carried out analogically to the conversion in the previous algorithm but in the opposite direction.

The algorithm is of the same computational complexity as the $ADDnADDk$ algorithm.

4.4 Security and the pre-computed tables usage

The algorithms are resistant to first-order attacks since all the intermediate variables are always masked with a random value – either with the long-term mask or with one of the random values, generated on the pre-computation stage.

The pre-computed tables are usually re-computed with the beginning of the cryptographic algorithm and may be used several times during its execution. Moreover, both proposed algorithms for switching between block-wise and arithmetic masking are designed in a way, that they both can use the same pre-computed tables during one execution of the cryptographic algorithm.

5 Conclusion

In this paper we introduced block-wise masking – a new kind of masking, which can be viewed as a generalization for more common kinds. We recalled requirements for switching algorithms and provided an overview of existing algorithms. Finally, with the new kind of masking the secure and fairly efficient method for switching between block-wise and arithmetic masking was proposed.

References

1. Kocher P., Jaffe J., Jun B. (1999) Differential Power Analysis. In: Wiener M. (eds) Advances in Cryptology — CRYPTO' 99. CRYPTO 1999. Lecture Notes in Computer Science, vol 1666, pp. 388–397. Springer, Berlin, Heidelberg. https://doi.org/10.1007/3-540-48405-1_25
2. Goubin L., Patarin J. (1999) DES and Differential Power Analysis The "Duplication" Method. In: Koç Ç.K., Paar C. (eds) Cryptographic Hardware and Embedded Systems. CHES 1999. Lecture Notes in Computer Science, vol 1717, pp. 158-172. Springer, Berlin, Heidelberg. https://doi.org/10.1007/3-540-48059-5_15
3. Chari S., Jutla C.S., Rao J.R., Rohatgi P. (1999) Towards Sound Approaches to Counteract Power-Analysis Attacks. In: Wiener M. (eds) Advances in Cryptology — CRYPTO' 99. CRYPTO 1999. Lecture Notes in Computer Science, vol 1666, pp. 398–412. Springer, Berlin, Heidelberg. https://doi.org/10.1007/3-540-48405-1_26
4. Lai X., Massey J.L. (1991) A Proposal for a New Block Encryption Standard. In: Damgård I.B. (eds) Advances in Cryptology — EUROCRYPT' 90. EUROCRYPT 1990. Lecture Notes in Computer Science, vol 473, pp. 389-404. Springer, Berlin, Heidelberg. https://doi.org/10.1007/3-540-46877-3_35

5. Contini S., Rivest R., Robshaw M., Yin Y.L. (1999) Improved analysis of some simplified variants of RC6. In: Fast Software Encryption, 6th International Workshop, FSE' 99, Rome, Italy, March 24-26, 1999, Proceedings, pp. 1–15. Springer. https://doi.org/10.1007/3-540-48519-8_1
6. Beaulieu, R., Shors D., Smith J., Treatman-Clark S., Weeks S., Wingers L. (2013) The SIMON and SPECK families of lightweightblock ciphers. IACR Cryptology ePrint Archive, 2013:404.
7. Goubin L. (2001) A Sound Method for Switching between Boolean and Arithmetic Masking. In: Koç Ç.K., Naccache D., Paar C. (eds) Cryptographic Hardware and Embedded Systems — CHES 2001. CHES 2001. Lecture Notes in Computer Science, vol 2162, pp. 3–15. Springer, Berlin, Heidelberg. https://doi.org/10.1007/3-540-44709-1_2
8. Debraize B. (2012) Efficient and Provably Secure Methods for Switching from Arithmetic to Boolean Masking. In: Prouff E., Schaumont P. (eds) Cryptographic Hardware and Embedded Systems – CHES 2012. CHES 2012. Lecture Notes in Computer Science, vol 7428, pp. 107–121. Springer, Berlin, Heidelberg. https://doi.org/10.1007/978-3-642-33027-8_7
9. Bettale, L., Coron, J.-S., Zeitoun, R. (2018) Improved High-Order Conversion From Boolean to Arithmetic Masking. In: IACR Transactions on Cryptographic Hardware and Embedded Systems, 2018(2), pp. 22–45. https://doi.org/10.13154/tches.v2018.i2.22-45
10. Dolmatov, V., Baryshkov, D. (2020) GOST R 34.12-2015: Block Cipher "Magma", RFC 8891 https://doi.org/10.17487/RFC8891
11. Coron J.-S., Tchulkine A. (2003) A New Algorithm for Switching from Arithmetic to Boolean Masking. In: Walter C.D., Koç Ç.K., Paar C. (eds) Cryptographic Hardware and Embedded Systems – CHES 2003. CHES 2003. Lecture Notes in Computer Science, vol 2779, pp. 89–97. Springer, Berlin, Heidelberg. https://doi.org/10.1007/978-3-540-45238-6_8
12. Neiße O., Pulkus J. (2004) Switching Blindings with a View Towards IDEA. In: Joye M., Quisquater JJ. (eds) Cryptographic Hardware and Embedded Systems - CHES 2004. CHES 2004. Lecture Notes in Computer Science, vol 3156, pp. 230-239. Springer, Berlin, Heidelberg. https://doi.org/10.1007/978-3-540-28632-5_17
13. Coron, J.-S. (2017) High-order conversion from Boolean to arithmetic masking, In: Fischer W., and Homma N. (eds) Cryptographic Hardware and Embedded Systems – CHES 2017. CHES 2017. Lecture Notes in Computer Science, vol. 10529, pp. 93–114. Springer. https://doi.org/10.1007/978-3-319-66787-4_5
14. Coron J.-S., Großschädl J., Vadnala P.K. (2014) Secure Conversion between Boolean and Arithmetic Masking of Any Order. In: Batina L., Robshaw M. (eds) Cryptographic Hardware and Embedded Systems – CHES 2014. CHES 2014. Lecture Notes in Computer Science, vol 8731, pp. 188–205. Springer, Berlin, Heidelberg. https://doi.org/10.1007/978-3-662-44709-3_11
15. Hutter, M., Tunstall, M. (2019) Constant-time higher-order Boolean-to-arithmetic masking. Journal of Cryptographic Engineering, 9, pp. 173–184. https://doi.org/10.1007/s13389-018-0191-z