Techniques for Random Masking in Hardware

Jovan Dj. Golić*

Abstract

A new technique for Boolean random masking of the logic AND operation in terms of NAND logic gates is presented and its potential for masking arbitrary cryptographic functions is pointed out. The new technique is much more efficient than a previously known technique, recently applied to AES. It is also applied for masking the integer addition. In addition, new techniques for the conversions from Boolean to arithmetic random masking and vice versa are developed. They are hardware oriented and do not require additional random bits. Unlike the previous, software-oriented techniques showing a substantial difference in the complexity of the two conversions, they have a comparable complexity being about the same as that of one integer addition only. All the techniques proposed are in theory secure against the first-order differential power analysis on the logic gate level. They can be applied in hardware implementations of various cryptographic functions, including AES, (keyed) SHA-1, IDEA, and RC6.

Key words. Digital circuits, logic circuits, Boolean functions, cryptography, arithmetic circuits, random masking, side-channel attacks, power analysis

1 Introduction

Cryptographic functions dealing with secret keys such as, for example, block ciphers or message authentication codes can be implemented in software or hardware on microelectronic data-processing devices such as integrated circuit chip (smart) cards. During the execution of a cryptographic algorithm, sensitive data depending on the secret key is being processed, being sent over the internal links, and being stored in the internal memories of the device. It is known that even for tamper-resistant chips, where the underlying integrated circuit is protected by special physical measures, this sensitive information may leak out through various side channels, such as measurements of timing, power consumption, and electromagnetic

*The author is with Access Network and Terminals System Design, Telecom Italia Lab, Telecom Italia, Via G. Reiss Romoli 274, 10148 Turin, Italy, Fax: +39 011 228 7140, Phone: +39 011 228 5572, Email: jovan.golic@tilab.com.

radiation as well as monitoring of signals by microprobing. The objective of side-channel attacks is to recover the secret key by using the leaked information. Power analysis attacks [13] are powerful as they do not require expensive resources and as most implementations without countermeasures incorporated are vulnerable to such attacks. The (first-order) differential power analysis (DPA) attacks are particularly interesting as they use a simple statistical technique that is almost independent of the implementation of the cryptographic algorithm.

The basis of power analysis attacks are elementary computations within the cryptographic device that depend on a part of the secret key and on the known input and/or output information. If the power consumption corresponding to these elementary computations depends on the values being computed, then the power consumption curves contain information about the secret key which may be feasible to extract by statistical techniques. In particular, in DPA attacks, one guesses the involved part of the secret key, computes the values of a chosen intermediate variable, classifies the power curves according to these values, and then computes and compares the average values of the classified curves. Both software and hardware implementations are potentially vulnerable.

1.1 Previous Work on Random Masking

A general method for counteracting power analysis attacks on the algorithmic level is to randomize the computations depending on the secret key by masking the original data with random masks and by modifying the computations accordingly. This method in fact originated as data splitting [3], [11], where the intermediate data is split into a number of shares and the computation is then performed on the shares. Masking [16] is essentially similar to data splitting into two shares [11], with a difference that one does not have to perform duplicate computations on the resulting shares. The masking operation combining the data with a random mask is typically adapted to the mathematical operations used in the cryptographic algorithm [16], because the required modifications in the computations are then minimized. More precisely, assume that in some elementary computation in the algorithm, inputs x and y are combined together into an output z by using a group operation $\circ$ according to $x \circ y = z$. As using any group operation for masking is sufficient to perfectly randomize the data, assume that x and y are randomized by using the purely random (uniformly distributed) and mutually statistically independent masks r_x and r_y through $\circ$, respectively. Then the computation does not have to be modified, as $(x \circ r_x) \circ (y \circ r_y) = z \circ (r_x \circ r_y)$. Note that most random masking techniques proposed for public-key cryptosystems are essentially of this type. Very frequently used group operations on n-bit words in cryptographic algorithms are the bitwise XOR, denoted as $\oplus$, and the integer addition and subtraction modulo 2^n, which are denoted just as $+$ and $-$, respectively, if the module is clear from the context. The corresponding random masking techniques are commonly called Boolean and arithmetic masking, respectively.

If $z = f(x, y)$, for an arbitrary function f, and if we want to obtain a masked output $z' = z \circ r_z$ from masked inputs $x' = x \circ r_x$ and $y' = y \circ r_y$, then the function f has to be modified into a new function f' determined by $z' = f'(x', y', r_x, r_y, r_z) = f(x' \circ r_x, y' \circ r_y) \circ r_z$. The problem is how to compute f' securely, that is, in a way resistant to DPA. To this end, f' can also depend on additional (dummy) random masking variables. The resistance to DPA of a masked algorithm computing f' can be controlled by a theoretical condition that each variable in this algorithm should be statistically independent of the (unmasked) input information. This condition is implicitly rather than explicitly respected in previous works on random masking in software, on the word level. In addition, for software implementations, f' should be expressed in terms of word operations available on common processors such as logic and arithmetic operations. Consequently, in a masked cryptographic algorithm, only the elementary computations different from the underlying group operation $\circ$ have to be modified. The only known general technique for doing this in software is based on precomputed lookup tables stored in RAM. It consists in precomputing a lookup table for f', given the values of r_x, r_y, and r_z, and then in reading the values from this table according to the masked inputs x' and y'. If the bit size of the inputs is too large, then it may be possible to apply the technique to smaller input blocks.

For many cryptographic functions, such as AES [6], the RAM space required for masking the nonlinear parts, that is, the S-boxes, may be impractical for constrained microelectronic devices such as smart cards. As for the solutions not based on precomputed and stored lookup tables, the multiplicative masking technique [1] for the AES S-boxes is shown to be vulnerable to the zero-value based DPA in [8], whereas the embedded multiplicative masking technique [8], which avoids this problem, does not provide ideal security.

In many algorithms, the bitwise XOR and the modular integer addition or subtraction, along with other Boolean and integer operations, are combined together for cryptographic security. The best-known examples are the widely used cryptographic hash function SHA-1 [18] and the block ciphers IDEA [14] and RC6 [22]. Note that SHA-1 incorporates a secret key if it is used for message authentication. In such algorithms, it is convenient to use both group operations for random masking. Therefore, there is a need to convert between the two corresponding masks in a secure way. Namely, given an n-bit data word x and an n-bit purely random masking word r, the problem is to compute securely $x+r$ from $x \oplus r$ and vice versa. Alternatively, another, related problem is treated in the literature as it is closer to data splitting, namely, to compute securely $x-r$ from $x \oplus r$ and vice versa. The two problems are here referred to as the mask addition and mask subtraction problems, respectively.

The first solution to the problem is proposed in [16], but in [4] both conversions are shown to be potentially vulnerable to a more sophisticated power analysis attack, due to the fact that the intermediate variables were not all fully randomized as binary words. Under a certain power consumption condition, this attack may be regarded as a sort of word-based DPA attack. New solutions for both conversions are proposed in [12]. They are essentially word, that is, software oriented and according to them it appears that the conversion from arithmetic to Boolean masking is inherently much more difficult than the conversion in the opposite direction. More precisely, the solution for the conversion from Boolean to arithmetic masking requires 7 n-bit word operations and an auxiliary n-bit random masking word, namely, 5 bitwise XOR operations and 2 subtractions modulo 2^n. The solution for the conversion from arithmetic to Boolean masking is much less efficient and requires $5(n+1)$ n-bit word operations and an auxiliary n-bit random masking word. For comparison, note that the direct conversion of the masks could be achieved by only two n-bit word operations, namely, one XOR and one addition or subtraction modulo 2^n, but is not computationally secure.

Another software-oriented solution for the conversion from arithmetic to Boolean masking, which requires precomputation and RAM storage of certain lookup tables and some auxiliary random masking bits, is proposed in [5] and further improved in [19]. It is generally more efficient than the corresponding solution from [12], depending on the processor word size, but remains much less efficient than the solution from [12] for the conversion in the opposite direction. Consequently, the known software-oriented mask conversion techniques are inefficient to be implemented in hardware.

Note that apart from using the mask conversion algorithms, it is also necessary to mask the nonlinear Boolean and integer operations that exist in a given cryptographic algorithm. For example, in (keyed) SHA-1 it is necessary to mask the logic AND operation for 32-bit words with respect to the Boolean (XOR) mask. Due to the data splitting paradigm, it is tempting, but incorrect, to conclude [5] that two word operations suffice for achieving this. To avoid using a number of precomputed and stored lookup tables, one has to mask the bitwise logic AND operation directly.

If a cryptographic algorithm is implemented in hardware, by a digital integrated circuit, then to prevent the (first-order) DPA attack, in theory it is necessary and sufficient to ensure that every elementary computation involving the secret information and performed by a logic gate is randomized. More precisely, the *secure computation condition* to be satisfied is that the output (binary) value of each logic gate in the protected hardware design should have the same probability distribution for each fixed value of the secret key and input information. In other words, the output value of each logic gate should be statistically independent of the secret key and input information. The randomness is provided by purely random masks, which should ideally be refreshed for every new input data to be processed by the cryptographic function considered. As far as resistance to DPA is concerned, random masking bits can be used repeatedly, but their number should be large enough in order to prevent more sophisticated power analysis attacks targeting the outputs of several logic gates jointly.

With an objective to mask lookup table hardware implementations of Boolean functions, a technique based on masking the MUX logic gate is proposed in [17], without defining the secure computation condition explicitly. It essentially consists in replacing each MUX gate in the original lookup table by a masked MUX gate which consists of three MUX gates. The gate count is thus tripled, while the delay is doubled. This technique can directly be used for masking the block ciphers by masking the lookup tables of S-boxes implemented in ROM, but generally requires a large gate count.

A general concept of random masking on the logic gate level and several techniques for masking the AND and OR logic gates are proposed in [15] and [10], including the secure computation condition, which is also explicitly formulated in [9]. Some other techniques for random masking of logic gates are introduced in [7], but are flawed as the secure computation condition is not respected. The techniques for masking the S-box of AES recently described in [23], [20], and [2] are all essentially based on the technique [15], presented in [10], for random masking of the logic AND operation.

1.2 Main Objectives and Results

The objective of this paper is to introduce new techniques for random masking of cryptographic algorithms in hardware, on the logic gate level, where the techniques should satisfy the secure computation condition described above. A secure computation on the word level in software generally does not imply a secure computation on the bit level in hardware. In practice, the secure computation condition on the bit level is necessary for providing resistance to DPA attacks and is also likely to be sufficient, although individual logic gates do not achieve their final (random) values simultaneously and in the transition stage their output values may vary (randomly) and may depend on their previous inputs. This effect is also present in software implementations and, in fact, generally makes the power analysis of non-masked implementations more difficult, especially so for logic circuit implementations in hardware.

The techniques proposed can be classified into Boolean masking techniques, which apply to arbitrary algorithms, and techniques for the conversion between Boolean and arithmetic masking, which apply to algorithms containing both Boolean and integer arithmetic operations. They are all bit based and hence suitable for direct hardware implementation by logic circuits. Boolean masking techniques can also be implemented on the word level, in software.

According to [10], two techniques for Boolean (XOR) masking of the logic AND and OR operations, namely, the XOR-based and MUX-based technique are pointed out. It is shown that both of them can be securely implemented by using NAND gates only and that the latter is much more efficient than the former. An arbitrary logic circuit, composed of XOR, NOT, AND, and OR gates, can thus be masked by using the techniques for masking the AND and OR gates, where the masked circuit can be obtained by replacing the AND and OR gates by masked AND and OR gates, respectively, by keeping the XOR and NOT gates intact, and by distributing or adapting the masking bits appropriately. The distribution of masking bits can be automatized, but this is a separate topic, not treated in this paper.

An important example is a logic circuit [24] for the S-box of AES [6], which consists of AND, XOR, and NOT gates, and the MUX-based technique thus yields a much more efficient solution than the XOR-based technique [23]. Another important example is the bitwise logic AND operation in (keyed) SHA-1. The XOR-based technique is also applicable for masking the multiplication operation in any ring structure such as a finite field or a ring of integeres modulo a positive integer. For example, this can be used in IDEA and RC6.

If a round of an iterative cryptographic algorithm contains both Boolean and integer arithmetic operations, like in (keyed) SHA-1, IDEA, and RC6, then mask conversion techniques are useful for providing the protection of hardware implementations against power analysis and other side-channel attacks. The techniques proposed for the secure conversions from arithmetic to Boolean random masking and vice versa, for both mask addition and mask subtraction, do not require additional random masking bits and are both equally efficient in terms of the gate count of the corresponding logic circuits. The gate count is roughly the same as that for one addition modulo 2^n of two n-bit words, while the depth (i.e., the delay) of the logic circuit for the conversion from arithmetic to Boolean masking is about one half of the depth of the logic circuit for the conversion from Boolean to arithmetic masking and is roughly the same as that for one addition modulo 2^n of two n-bit words.

The technique for the conversion from Boolean to arithmetic masking can also be used for the secure hardware computation of the arithmetic masking operation $x + r$, where x is a secret n-bit word and r is an n-bit purely random mask. Note that if the masked value is computed directly in terms of the carry bits, then the computation is not secure on the logic gate level as the carry bits are dependent on x and are thus not fully randomized.

The mask conversion techniques are especially effective if a round of an iterative cryptographic algorithm contains a number of integer arithmetic operations in a row, which is the case in (keyed) SHA-1, IDEA, and RC6. Alternatively, for hardware implementations, instead of using the mask conversion techniques, one can mask the integer arithmetic operations directly by using the proposed techniques for masking the AND logic gate. Therefore, a logic circuit for masking the addition of two integers modulo 2^n is also provided, and can be of separate interest. It thus turns out that this approach is effective if a small number (e.g., 1 to 3) of additions modulo 2^n are used in a row. For (keyed) SHA-1, where this number is equal to 4, the alternative approach is hence less effective.

The rest of the paper is organized as follows. Techniques for Boolean masking of the logic AND operation and their applications are treated in Section 2. Techniques for the conversions from Boolean to arithmetic masking and vice versa, for mask addition, are introduced in Sections 3 and 4, respectively. Mask conversion techniques for mask subtraction are presented Section 5. In Section 6, a technique for Boolean masking of the addition of two integers is described and usefulness of mask conversion techniques is then demonstrated. Conclusions are given in Section 7.

2 Masking Logic AND Operation

For the Boolean operations, we adopted the usual notation: $\oplus$ for XOR or addition modulo 2, $\wedge$ for AND, $\vee$ for OR, and $\bar{\ }$ for NOT, whereas for the MUX logic operation of two data inputs x and y and a control input c we use $\mathrm{MUX}(x, y; c) = \bar{c} \wedge x \vee c \wedge y$. Note that $x \oplus y = \bar{x} \wedge y \vee x \wedge \bar{y} = \mathrm{MUX}(y, \bar{y}; x)$. As usual, $\wedge$ has the priority over $\oplus$ and $\vee$. The same notation is used when the operations are applied bitwise, on the word level.

A masked AND logic gate, with respect to the Boolean (XOR) mask, operating on masked inputs $x' = x \oplus r_x$ and $y' = y \oplus r_y$ and producing the masked output $z' = x \wedge y \oplus r_z$, should implement the masked AND operation

$$z' \; = \; x' \wedge' y' \; = \; (x' \oplus r_x) \wedge (y' \oplus r_y) \oplus r_z \tag{1}$$

by a logic circuit in which the outputs of all logic gates are computed securely. As explained in Section 1, the secure computation condition means that the output value of every elementary bit-based computation in the algorithm, that is, the output of every elementary logic gate in the corresponding logic circuit, should have the same probability distribution for each fixed value of the data input, provided that the involved masking bits are uniformly distributed and mutually statistically independent.

According to [10], two solutions to this problem are pointed out. One solution [15] consists in applying the distributive property to (1) and in grouping the terms appropriately to obtain

$$z' \; = \; z \oplus r_z = (((r_z \oplus (r_x \wedge r_y)) \oplus (r_x \wedge y')) \oplus (r_y \wedge x')) \oplus (x' \wedge y') \tag{2}$$

in which all the computations are secure if r_x, r_y, and r_z are uniformly distributed and mutually statistically independent, see Fig. 1. The corresponding logic circuit consists of 4 AND and 4 XOR gates. The computations are secure as for each fixed value of (x, y), the two inputs to each AND gate are uniformly distributed and mutually statistically independent, whereas, due to r_z, one of the inputs to each XOR gate is uniformly distributed and statistically independent of the other input. Other similar expressions can also be derived.

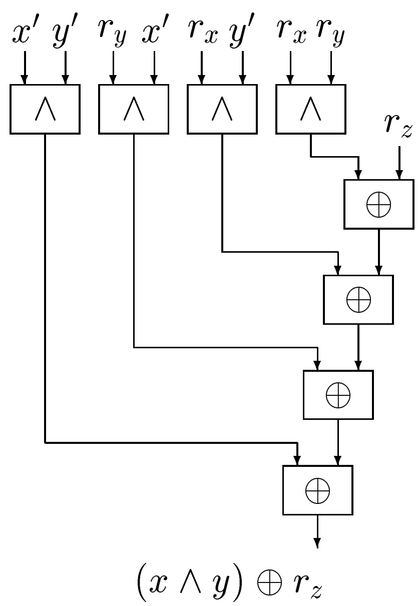

Figure 1: An XOR-based circuit for masking the AND gate.

The other solution uses a technique [17] for masking a MUX logic gate. The masked MUX gate is a cascade connection of the SWITCH gate and the MUX gate, the SWITCH gate being controlled by the control masking bit and the MUX gate being controlled by the masked control bit, where $\mathrm{SWITCH}(x, y; c) = (\mathrm{MUX}(x, y; c), \mathrm{MUX}(y, x; c))$. It is assumed that the two input masking bits and the output masking bit are all the same and that the control masking bit is statistically independent of them. A direct application of this technique to the lookup table implementation of the AND function requires four MUX gates, one of which is an XOR gate, in order to implement the masked AND gate.

However, if we allow for the output masking bit to be the same as one of the two input masking bits, then only three MUX gates suffice to compute securely the masked AND gate, and the number of masking bits needed is reduced from three to two. The new technique consists in using the expression $z = x \wedge y = \mathrm{MUX}(0, x; y)$ for the AND function and in masking the MUX function. We thus obtain

$$\begin{aligned} z' = z \oplus r_x &= \mathrm{MUX}\left(\mathrm{MUX}(r_x, x'; r_y), \mathrm{MUX}(x', r_x; r_y); y'\right) \\ &= \bar{y}' \wedge (\bar{r}_y \wedge r_x \vee r_y \wedge x') \vee y' \wedge (r_y \wedge r_x \vee \bar{r}_y \wedge x'), \end{aligned} \tag{3}$$

see Figs. 2 and 3. The logic circuit shown in Fig. 3 consists of 6 AND, 3 OR, and 2 NOT gates. All the involved computations are secure.

More precisely, for each fixed value of (x, y), the output bit of each MUX gate in Fig. 2 (i.e., the output bit of each OR gate in Fig. 3) is uniformly distributed as each of its two data input bits is uniformly distributed and statistically independent of the control bit, whereas the output bit of each AND gate in Fig. 3 has the probability 1/4 of being equal to 1 as its two input bits are uniformly distributed and mutually statistically independent.

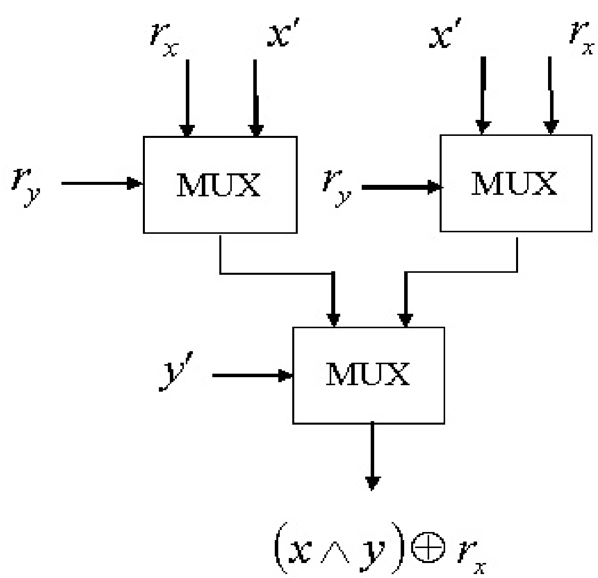

Figure 2: A MUX-based circuit for masking the AND gate.

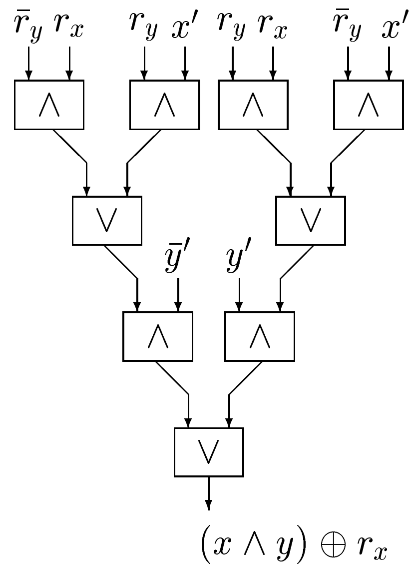

Figure 3: An (AND, OR, NOT)-based circuit for masking the AND gate.

Other equivalent circuits can be obtained similarly. Logic circuits for masking the OR gate can be obtained, for example, by applying the duality principle to (2) and (3), respectively. When a number of 2-input AND gates are connected together, assuming that the input y' is used for the connection, the depth per masked AND gate is 1 AND and 2 XOR gates for the first solution and 1 AND, 1 OR, and 1 NOT gate for the second solution.

Another solution can be derived from the MUX-based circuit for masking the AND gate, by implementing the MUX gates in terms of NAND gates. This solution is practically important since NAND gates are suitable for implementation in CMOS transistor technology. It is known that it takes 4 CMOS transistors to implement one NAND or one NOR gate. One MUX gate can be implemented by a circuit composed of 4 NAND gates, whose depth is 2 NAND gates with respect to two data inputs and 3 NAND gates with respect to the control input, assuming, for simplicity, that a NOT gate is implemented as a NAND gate (in fact, it takes only two CMOS transistors to implement a NOT gate).

Similarly, yet another solution can be derived from the XOR-based circuit for masking the AND gate, by implementing the XOR and AND gates in terms of NAND gates. It is known that one XOR gate can be implemented by a circuit composed of 4 NAND gates, whose depth is 3 NAND gates with respect to both inputs, whereas one AND gate is, for simplicity, assumed to be implemented by two NAND gates.

It follows that all the computations remain secure on the NAND logic gate level, in both the circuits. In terms of NAND gates, the MUX-based solution is *superior* to the XOR-based solution. Namely, the gate counts are then 12 versus 24 NAND gates, and the depths are 4 versus 12 NAND gates, respectively. Here, we did not count the delays of 1 NAND gate for computing $\bar{r}_y$ and 2 NAND gates for computing $r_x \wedge r_y$, respectively, as these terms are data-independent and can hence be precomputed. When a number of 2-input AND gates are connected together, then the depths per masked gate reduce to 3 NAND gates and 8 NAND gates, respectively.

However, for the word operations, such as the bitwise AND operation in SHA-1 implemented in software, the XOR-based solution is better with regard to common instruction sets, as (2) and (3) can be implemented in 8 and 11 processor cycles, respectively. The solution based on (2) is also applicable for masking the multiplication in an arbitrary field or ring structure, e.g., for modular integer multiplication in IDEA or RC6.

The proposed techniques can be used for masking an arbitrary logic circuit, composed of XOR, AND, OR, and NOT gates, where only AND and OR gates have to be effectively masked, and the masking bits have to be distributed or adapted appropriately. In particular, they can be applied for masking the logic circuit [24] for the S-box of AES, which consists of XOR, NOT, and AND gates. The circuit is obtained by using the composite field representation of $GF(2^8)$ based on quadratic extensions, to represent one inversion in $GF(2^8)$ in terms of one inversion and a number of additions and multiplications in $GF(2^4)$ and, further, by reducing the operations in $GF(2^4)$ to additions and multiplicatons in $GF(2)$. Such a solution using the XOR-based masking technique is described in [23], without referring to the secure computation condition. Accordingly, the solution using the novel MUX-based masking technique is much more efficient, as the the number of NAND gates per masked AND gate is halved and the depth is reduced about three times!

The solutions from [20] and [2] for masking the S-box of AES in hardware are essentially the same, whereas the former is also implemented [21], but does not invoke the secure computation condition. They also use the composite field representation of GF(2^8) based on quadratic extensions and are in spirit similar to the solution from [23]. The only essential difference is that the resulting multiplications in GF(2^4) and/or GF(2^2) are masked directly by using (2) over the respective subfields. More precisely, in [2], an equivalent form of (2) is used instead.

3 Conversion from Boolean to Arithmetic Masking

In this section, an algorithm for the conversion from Boolean to arithmetic masking, with respect to the mask addition, is proposed. In mathematical terms, given an n-bit data word $x = x_{n-1}x_{n-2}\cdots x_1x_0$ and an n-bit purely random masking word $r = r_{n-1}r_{n-2}\cdots r_1r_0$, the problem considered is to compute securely $x + r$ from $x \oplus r$, where the addition is modulo 2^n.

Let $x' = x'_{n-1}x'_{n-2}\cdots x'_1x'_0 = x \oplus r$ and $x'' = x''_{n-1}x''_{n-2}\cdots x''_1x''_0 = x + r$, where the least significant bit has index 0. Then according to the well-known school method for integer addition with carry, we have

$$x''_i \;=\; x_i \oplus r_i \oplus c_{i-1} \;=\; x'_i \oplus c_{i-1}, \quad 0 \le i \le n-1 \tag{4}$$

where $c_{-1} = 0$ and

$$c_{i-1} \;=\; x_{i-1} \wedge r_{i-1} \vee c_{i-2} \wedge (x_{i-1} \oplus r_{i-1}) \;=\; \bar{x}'_{i-1} \wedge r_{i-1} \vee c_{i-2} \wedge x'_{i-1}, \quad 1 \le i \le n-1. \tag{5}$$

Further, in (5), if $i = 1$, then we substitute $c_{i-2} = 0$, and if $2 \le i \le n-1$, then, due to (4), we substitute $c_{i-2} = x'_{i-1} \oplus x''_{i-1}$. Thus we obtain $c_0 = \bar{x}'_0 \wedge r_0$ and

$$c_{i-1} \;=\; \bar{x}'_{i-1} \wedge r_{i-1} \vee x'_{i-1} \wedge \bar{x}''_{i-1}, \quad 2 \le i \le n-1. \tag{6}$$

Now, by substituting c_{i-1} in (4), after an additional algebraic manipulation, we finally get the recursive equations

$$x''_0 \;=\; x'_0 \tag{7}$$

$$x''_1 = \bar{x}'_0 \wedge (\bar{x}'_1 \wedge r_0 \vee x'_1 \wedge \bar{r}_0) \vee x'_0 \wedge x'_1 \tag{8}$$

$$x''_i = \bar{x}'_{i-1} \wedge (\bar{x}'_i \wedge r_{i-1} \vee x'_i \wedge \bar{r}_{i-1}) \vee x'_{i-1} \wedge (\bar{x}'_i \wedge \bar{x}''_{i-1} \vee x'_i \wedge x''_{i-1}), \quad 2 \le i \le n-1. \tag{9}$$

In equations (8) and (9), one can recognize the underlying structures of MUX and XOR gates. Accordingly, they can also be put in a more insightful form

$$x''_1 = \mathrm{MUX}(x'_1 \oplus r_0, x'_1; x'_0) \tag{10}$$

$$x''_i = \mathrm{MUX}(x'_i \oplus r_{i-1}, x'_i \oplus \bar{x}''_{i-1}; x'_{i-1}), \quad 2 \le i \le n-1. \tag{11}$$

Equations (8) and (9) can be implemented by a logic circuit composed of elementary blocks shown in Fig. 4, for $2 \le i \le n-1$, whereas for $i = 1$, the block is simplified by formally setting $x''_0 = 1$. The involved logic gates are AND, OR, and NOT gates, which are all elementary, and NOT gates are not shown for simplicity.

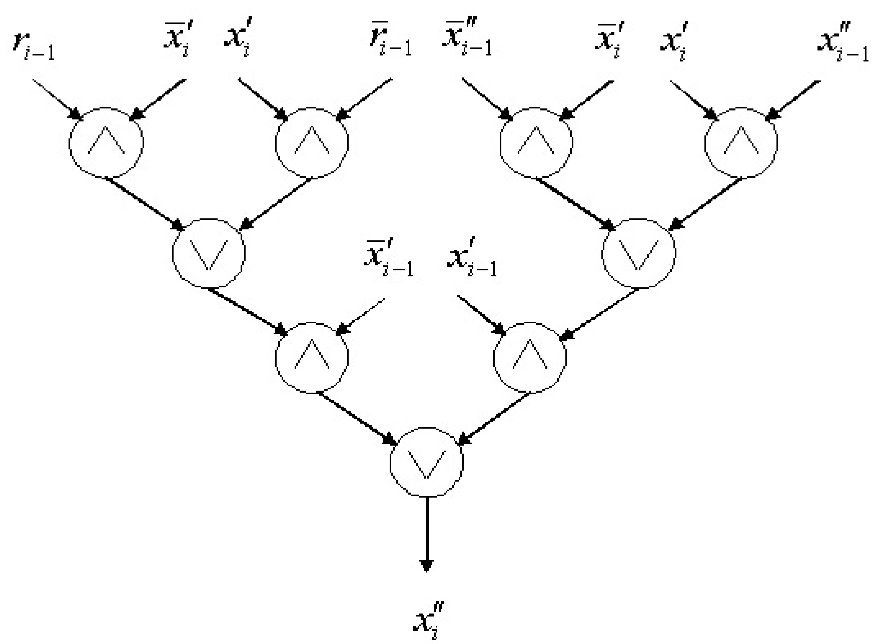

Figure 4: A block for conversion from Boolean to added arithmetic mask.

The elementary block for the logic circuit composed of MUX and XOR gates according to (10) and (11) is shown in Fig. 5. Despite some similarity with the logic circuit from Fig. 2, the logic circuit from Fig. 5 is essentially different.

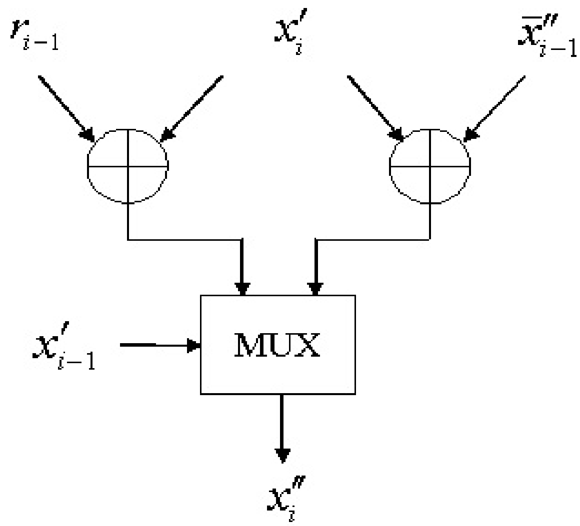

Figure 5: A MUX-based block for conversion from Boolean to added arithmetic mask.

All the computations in (10) and (11) as well as in (8) and (9) are secure, that is, the output value of each logic gate in the corresponding logic circuit has the same probability distribution for every fixed value of the input x. Unlike the previously proposed techniques, it is interesting that no additional masking bits are required. The desired randomization is provided by x'_i, that is, by the underlying masking bit r_i, which is statistically independent of r_{i-1}, x''_{i-1}, and x'_{i-1}.

More precisely, assume that the input x has an arbitrary fixed value. Then the output bit of each XOR gate in Fig. 5 (i.e., the output bit of each corresponding OR gate in Fig. 4) is uniformly distributed as its two input bits are uniformly distributed and mutually statistically independent. The output bit of the MUX gate in Fig. 5 (i.e., the output bit of the corresponding OR gate in Fig. 4) is uniformly distributed as each of its two data input bits is uniformly distributed and statistically independent of the control bit, whereas the output bit of each AND gate in Fig. 4 has the probability 1/4 of being equal to 1 as its two input bits are uniformly distributed and mutually statistically independent.

The computations remain secure even when MUX, XOR, and NOT gates from Fig. 5 are implemented in terms of NAND gates. In fact, the XOR gate with one input negated (XNOR) can also be securely implemented by using 3 NAND gates and 1 NOR gate.

The gate count of all the circuits is equivalent to $3n-4$ MUX gates and the depth is about $2n-2$ MUX gates. For comparison, note that the school method for integer addition with carry has an equivalent gate count of about $3n-2$ MUX gates and depth of about n MUX gates. The proposed technique can also be adapted to other methods for integer addition.

4 Conversion from Arithmetic to Boolean Masking

In this section, an algorithm for the conversion from arithmetic to Boolean masking, with respect to the mask addition, is proposed. In mathematical terms, given an n-bit data word $x = x_{n-1}x_{n-2}\cdots x_1x_0$ and an n-bit purely random masking word $r = r_{n-1}r_{n-2}\cdots r_1r_0$, the problem considered is to compute securely $x \oplus r$ from $x + r$, where the addition is modulo 2^n. We use the same notation as in Section 3.

We start from the equations (10) and (11) developed in Section 3. The main point is that (11) can be inverted by switching x_i' and x_i'' and keeping everything else the same, and the same is true for (10) and x_1' and x_1''. Accordingly, we thus obtain

$$x_1' = \mathrm{MUX}(x_1'' \oplus r_0, x_1''; x_0') \tag{12}$$

$$x_i' = \mathrm{MUX}(x_i'' \oplus r_{i-1}, x_i'' \oplus \bar{x}_{i-1}''; x_{i-1}'), \quad 2 \le i \le n-1. \tag{13}$$

In terms of AND, OR, and NOT gates, we equivalently have

$$x_0' = x_0'' \tag{14}$$

$$x_1' = \bar{x}_0' \wedge (\bar{x}_1'' \wedge r_0 \vee x_1'' \wedge \bar{r}_0) \vee x_0' \wedge x_1'' \tag{15}$$

$$x_i' = \bar{x}_{i-1}' \wedge (\bar{x}_i'' \wedge r_{i-1} \vee x_i'' \wedge \bar{r}_{i-1}) \vee x_{i-1}' \wedge (\bar{x}_i'' \wedge \bar{x}_{i-1}'' \vee x_i'' \wedge x_{i-1}''), \quad 2 \le i \le n-1. \tag{16}$$

Equations (15) and (16) can be implemented by a logic circuit composed of elementary blocks shown in Fig. 6, for $2 \leq i \leq n-1$, whereas for $i = 1$, the block is simplified by formally setting $x_0'' = 1$. The elementary block for the logic circuit composed of MUX and XOR gates according to (12) and (13) is shown in Fig. 7.

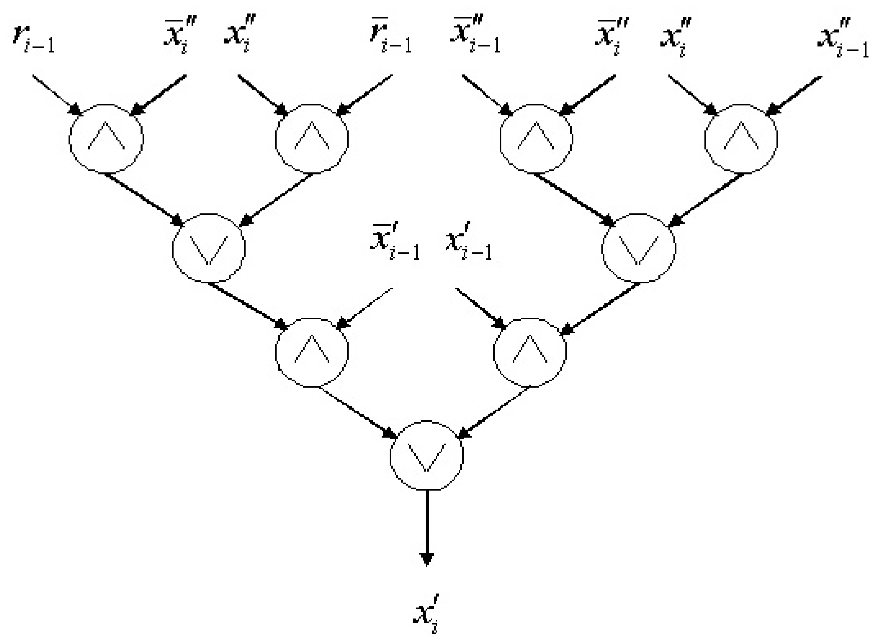

Figure 6: A block for conversion from added arithmetic to Boolean mask.

All the computations in (12) and (13) as well as in (15) and (16) are secure, where the desired randomization is provided by x_i'', that is, by the underlying masking bit r_i, which is statistically independent of r_{i-1}, x_{i-1}'', and x_{i-1}'. As in Section 3, no additional masking bits are required. As in Fig. 5, the computations remain secure if MUX, XOR, and NOT gates from Fig. 7 are implemented in terms of NAND and NOR gates.

The gate count of all the circuits is equivalent to $3n - 4$ MUX gates and the depth is now reduced to about n MUX gates. The depth reduction is due to the fact that the values x_{i-1}'' are already available.

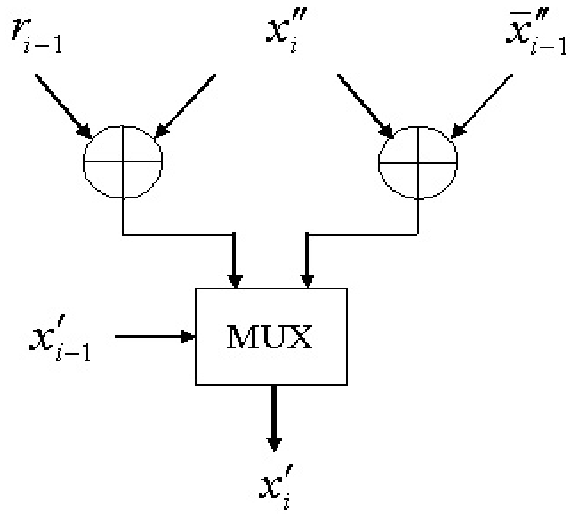

Figure 7: A MUX-based block for conversion from added arithmetic to Boolean mask.

5 Conversions for Mask Subtraction

In this section, the mask conversion problems in which the mask is being subtracted from instead of added to data are considered. The two proposed algorithms are for the conversion from Boolean to arithmetic masking and vice versa, respectively. In mathematical terms, given an n-bit data word $x = x_{n-1}x_{n-2}\cdots x_1x_0$ and an n-bit purely random masking word $r = r_{n-1}r_{n-2}\cdots r_1r_0$, the problems considered are to compute securely $x'' = x - r$ from $x' = x \oplus r$ and vice versa, where the subtraction is modulo 2^n. The starting point is the well-known expression for the additive inverse modulo 2^n, namely, $-r \equiv \bar{r} + 1 \pmod{2^n}$.

Accordingly, by substituting $\bar{r}$ for r and by setting the initial carry as $c_{-1} = 1$, we obtain the following recursive equations for the conversion from Boolean to arithmetic masking

$$x''_0 = x'_0 \tag{17}$$

$$x''_1 = x'_0 \wedge (\bar{x}'_1 \wedge r_0 \vee x'_1 \wedge \bar{r}_0) \vee \bar{x}'_0 \wedge x'_1 \tag{18}$$

$$x''_i = x'_{i-1} \wedge (\bar{x}'_i \wedge r_{i-1} \vee x'_i \wedge \bar{r}_{i-1}) \vee \bar{x}'_{i-1} \wedge (\bar{x}'_i \wedge x''_{i-1} \vee x'_i \wedge \bar{x}''_{i-1}), \quad 2 \leq i \leq n-1 \tag{19}$$

and, in terms of MUX and XOR operations,

$$
\begin{aligned}
x''_1 &= \mathrm{MUX}(x'_1 \oplus r_0, x'_1; \bar{x}'_0) && (20)\\
x''_i &= \mathrm{MUX}(x'_i \oplus r_{i-1}, x'_i \oplus x''_{i-1}; \bar{x}'_{i-1}), \quad 2 \le i \le n-1. && (21)
\end{aligned}
$$

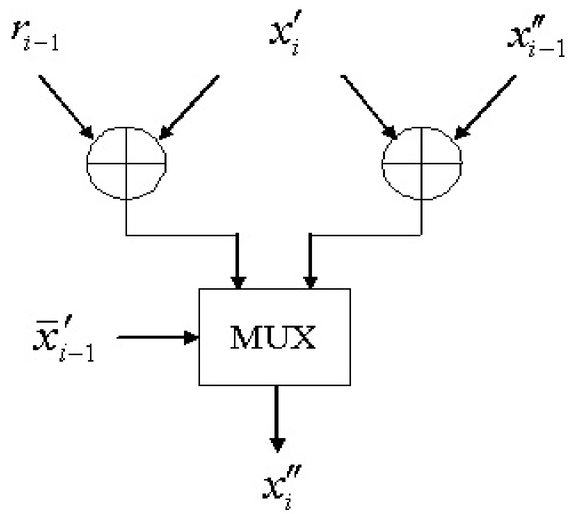

Figure 8: A MUX-based block for conversion from Boolean to subtracted arithmetic mask.

From these equations, similarly as in Section 4, we obtain the following recursive equations for the conversion from arithmetic to Boolean masking

$$
\begin{aligned}
x'_0 &= x''_0 && (22)\\
x'_1 &= x'_0 \wedge (\bar{x}''_1 \wedge r_0 \vee x''_1 \wedge \bar{r}_0) \vee \bar{x}'_0 \wedge x''_1 && (23)\\
x'_i &= x'_{i-1} \wedge (\bar{x}''_i \wedge r_{i-1} \vee x''_i \wedge \bar{r}_{i-1}) \vee \bar{x}'_{i-1} \wedge (\bar{x}''_i \wedge x''_{i-1} \vee x''_i \wedge \bar{x}''_{i-1}), \quad 2 \le i \le n-1 && (24)
\end{aligned}
$$

and, in terms of MUX and XOR operations,

$$
\begin{aligned}
x'_1 &= \mathrm{MUX}(x''_1 \oplus r_0, x''_1; \bar{x}'_0) && (25)\\
x'_i &= \mathrm{MUX}(x''_i \oplus r_{i-1}, x''_i \oplus x''_{i-1}; \bar{x}'_{i-1}), \quad 2 \le i \le n-1. && (26)
\end{aligned}
$$

For both conversions, the resulting logic circuits are analogous to those defined for the mask addition and have the same gate count and depth, respectively. The corresponding elementary blocks in terms of MUX and XOR gates are shown in Figs. 8 and 9, respectively.

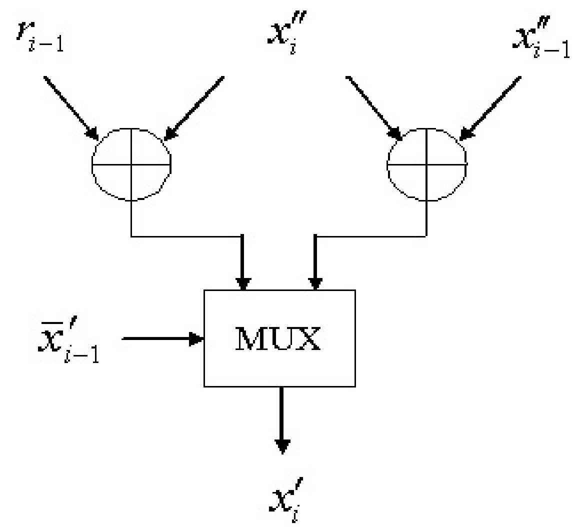

Figure 9: A MUX-based block for conversion from subtracted arithmetic to Boolean mask.

All the involved elementary computations are secure for essentially the same reasons as for the mask addition. When MUX and XOR gates are implemented in terms of NAND gates, all the computations remain secure. Consequently, the gate count is equivalent to $3n-4$ MUX gates for both conversions, whereas the depth is about $2n-2$ and n MUX gates for the two conversions, respectively.

6 Masking Integer Addition and Comparison

Let $x = x_{n-1}x_{n-2}\cdots x_1x_0$ and $y = y_{n-1}y_{n-2}\cdots y_1y_0$ be two given n-bit data words, and let $r_x = r_{x,n-1}r_{x,n-2}\cdots r_{x,1}r_{x,0}$ and $r_y = r_{y,n-1}r_{y,n-2}\cdots r_{y,1}r_{y,0}$ be the corresponding statistically independent purely random masking words to be used as Boolean masks. Let $z = x + y$, where the addition is modulo 2^n, and let r_z be the output Boolean mask, which can be related to r_x and r_y or statistically independent of them. The problem considered is to compute securely $(x+y) \oplus r_z$ from $x \oplus r_x$ and $y \oplus r_y$. Let $x' = x \oplus r_x$, $y' = y \oplus r_y$, and $z' = z \oplus r_z$.

According to the well-known school method for integer addition with carry, we have

$$z_i \;=\; x_i \oplus y_i \oplus c_{i-1}, \quad 0 \le i \le n-1 \tag{27}$$

where $c_{-1} = 0$ and

$$c_{i-1} \;=\; x_{i-1} \wedge y_{i-1} \oplus c_{i-2} \wedge (x_{i-1} \oplus y_{i-1}), \quad 1 \le i \le n-1. \tag{28}$$

Here, in comparison with (5), $\vee$ is replaced by $\oplus$ as it is easier to mask. Equations (27) and (28) essentially define the one-bit full adder.

In light of Section 2, the solution consists in replacing the AND operation in (28) at two places by the masked AND operation by using (3), and in distributing the masking bits so that the relation between the output and input masks is relatively simple and that the depth of the masked circuit is as small as possible. As argued in Section 2, the solution using (3) is much more efficient than that using (2), which is proposed in [15]. Let c_i' denote the masked carry bit c_i where the masking bit is to be determined, $0 \le i \le n-2$. We thus obtain

$$z_i' \;=\; x_i' \oplus y_i' \oplus c_{i-1}', \quad 0 \le i \le n-1 \tag{29}$$

where $c_{-1}' = 0$,

$$c_0' \;=\; \mathrm{MUX}\,(\mathrm{MUX}(r_{y,0}, y_0'; r_{x,0}), \mathrm{MUX}(y_0', r_{y,0}; r_{x,0}); x_0'), \tag{30}$$

and for $2 \le i \le n-1$

$$\begin{aligned} c_{i-1}' \;=\; & \mathrm{MUX}\,(\mathrm{MUX}(r_{x,i-1}, x_{i-1}'; r_{y,i-1}), \mathrm{MUX}(x_{i-1}', r_{x,i-1}; r_{y,i-1}); y_{i-1}') \\ & \oplus \mathrm{MUX}\,(\mathrm{MUX}(r_{x,i-1} \oplus r_{y,i-1}, x_{i-1}' \oplus y_{i-1}'; r_{y,i-2}), \\ & \qquad \mathrm{MUX}(x_{i-1}' \oplus y_{i-1}', r_{x,i-1} \oplus r_{y,i-1}; r_{y,i-2}); c_{i-2}'). \end{aligned} \tag{31}$$

To minimize the depth, c_{i-2}' is used as the control bit of the MUX gate for masking the right-hand AND operation in (28), so that the corresponding output masking bit is equal to

$r_{x,i-1} \oplus r_{y,i-1}$. As the control bit of the MUX gate for masking the left-hand AND operation is chosen to be y'_{i-1} in (31), the output masking bit for the masked left-hand AND gate is equal to $r_{x,i-1}$. As a consequence, the masking bit for c'_{i-1} is then equal to $r_{y,i-1}$, for $2 \leq i \leq n-2$. For $i = 1$, the right-hand AND operation does not effectively exist in (28), the control bit of the MUX gate for masking the left-hand AND operation is chosen to be x'_{i-1} in (30), and the masking bit for c'_{i-1} is then also $r_{y,i-1}$. The output masking bits are then purely random and are given as $r_{z,i} = r_{x,i} \oplus r_{y,i} \oplus r_{y,i-1}$, $0 \leq i \leq n-1$, where formally $r_{y,-1} = 0$. If desired, the output mask can be adapted to $r_x \oplus r_y$ by using additional $n-1$ XOR operations. All the involved computations are secure.

Altogether, the gate count required is about $10n - 14$ MUX gates and the depth is about $2n - 1$ MUX gates. In comparison with integer addition, the gate count is slightly more than tripled, while the depth is doubled. The proposed technique can also be applied to other methods for integer addition, for example, to the so-called Wallace trees, where the one-bit full adders are arranged in a tree to reduce the total depth.

We now compare the two masking approaches, that is, the mask conversion with integer addition and the masked integer addition without the mask conversion, taking (keyed) SHA-1 for example. In each iteration of the compression function of SHA-1, one has to compute 4 integer additions. One of the 5 operands that is a constant requires the masked integer addition, but not the conversion from Boolean to arithmetic masking. Accordingly, the gate counts of the basic implementations of the two approaches are then equivalent to about 9 versus 13 integer additions, respectively, and the depths are both equivalent to about 6 integer additions. Here, we did not count operations involving only the masking bits as they are data-independent and can hence be precomputed. In particular, some masks can be adapted to be the same in every iteration of SHA-1. This demonstrates that the mask conversion techniques are also important for hardware implementations. In general, the second approach can be more effective if the number of consecutive integer additions is relatively small.

7 Conclusions

It is shown that arbitrary logic circuits can be randomly masked, in a way theoretically secure against the first-order DPA on the logic gate level, by using a new algorithm for Boolean masking of the logic AND operation in terms of NAND gates. The new algorithm is considerably more efficient than the previously known algorithm, recently applied to AES. In particular, the gate count and the delay of the corresponding logic circuit are reduced two and three times, respectively. An algorithm for Boolean masking of the integer addition is also derived.

A new method for the secure conversion between Boolean and arithmetic random masking is introduced. The developed mask conversion algorithms do not require auxiliary random masking bits and are significantly more efficient than the previously known algorithms when applied in hardware, especially for the conversion from arithmetic to Boolean masking.

The new random masking algorithms are practically important for protecting hardware implementations of cryptographic algorithms against power analysis and other side-channel attacks. They can also be used in the hardware design of arithmetic and logic units for secure processors.

References

[1] M.-L. Akkar and C. Giraud, "An implementation of DES and AES, secure against some attacks," Cryptographic Hardware and Embedded Systems - CHES 2001, *Lecture Notes in Computer Science*, vol. 2162, pp. 309-318, 2001.

[2] J. Blömer, G. Merchan, and V. Krummel, "Provably secure masking of AES," Selected Areas in Cryptography - SAC '04, *Lecture Notes in Computer Science*, to appear.

[3] S. Chari, C. Jutla, J. Rao, and P. Rohatgi, "Towards sound approaches to counteract power-analysis attacks," Advances in Cryptology - CRYPTO '99, *Lecture Notes in Computer Science*, vol. 1666, pp. 398-412, 1999.

[4] J.-S. Coron and L. Goubin, “On Boolean and arithmetic masking against differential power analysis,” Cryptographic Hardware and Embedded Systems - CHES 2000, *Lecture Notes in Computer Science*, vol. 1965, pp. 231-237, 2000.

[5] J.-S. Coron and A. Tchulkine, “A new algorithm for switching from arithmetic to Boolean masking,” Cryptographic Hardware and Embedded Systems - CHES 2003, *Lecture Notes in Computer Science*, vol. 2779, pp. 89-97, 2003.

[6] J. Daemen and V. Rijmen, *The Design of Rijndael: AES - The Advanced Encryption Standard*. Berlin: Springer-Verlag, 2002.

[7] B. Gammel, F. Klug, and O. Kniffler, “Arithmetic unit and method for carrying out an arithmetic operation with coded operands,” WIPO PCT patent No. WO 03/060691 A2, July 24, 2003 (see also German patent No. DE10201449C1, Aug. 14, 2003).

[8] J. Dj. Golić and C. Tymen, “Multiplicative masking and power analysis of AES,” Cryptographic Hardware and Embedded Systems - CHES 2002, *Lecture Notes in Computer Science*, vol. 2523, pp. 198-212, 2002.

[9] J. Dj. Golić, “DeKaRT: A new paradigm for key-dependent reversible circuits,” Cryptographic Hardware and Embedded Systems - CHES 2003, *Lecture Notes in Computer Science*, vol. 2779, pp. 98-112, 2003.

[10] J. Dj. Golić and R. Menicocci, “Universal masking on logic gate level,” *Electronics Letters*, vol. 40(9), pp. 526-527, Apr. 2004.

[11] L. Goubin and J. Patarin, “DES and differential power analysis - The duplication method,” Cryptographic Hardware and Embedded Systems - CHES ’99, *Lecture Notes in Computer Science*, vol. 1717, pp. 158-172, 1999.

[12] L. Goubin, "A sound method for switching between Boolean and arithmetic masking," Cryptographic Hardware and Embedded Systems - CHES 2001, *Lecture Notes in Computer Science*, vol. 2162, pp. 3-15, 2001.

[13] P. Kocher, J. Jaffe, and B. Jun, "Differential power analysis," Advances in Cryptology - CRYPTO '99, *Lecture Notes in Computer Science*, vol. 1666, pp. 388-397, 1999.

[14] X. Lai and J. Massey, "A proposal for a new block encryption standard," Advances in Cryptology - EUROCRYPT '90, *Lecture Notes in Computer Science*, vol. 473, pp. 389-404, 1991.

[15] R. Menicocci and J. Pascal, "Elaborazione crittografica di dati digitali mascherati", Italian patent pending MI2003A001375, July 2003.

[16] T. Messerges, "Securing the AES finalists against power analysis attacks," Fast Software Encryption - FSE 2000, *Lecture Notes in Computer Science*, vol. 1978, pp. 150-164, 2001.

[17] T. Messerges, E. Dabbish, and L. Puhl, "Method and apparatus for preventing information leakage attacks on a microelectronic assembly," US patent No. US 6,295,606 B1, Sept. 25, 2001.

[18] National Institute of Standards and Technology, "Secure Hash Standard," Federal Information Processing Standards Publication 180-1, 1995.

[19] O. Neiße and J. Pulkus, "Switching blindings with a view towards IDEA," Cryptographic Hardware and Embedded Systems - CHES 2004, *Lecture Notes in Computer Science*, vol. 3156, pp. 230-239, 2004.

[20] E. Oswald, S. Mangard, and N. Pramstaller, "Secure and efficient masking of the AES - a mission impossible?," *Cryptology ePrint Archive*, Report 2004/134, Jun. 2004, available at http://eprint.iacr.org/.

[21] N. Pramstaller, F.K. Gürkaynak, S. Haene, H. Kaeslin, N. Felber, and W. Fichtner, "Towards an AES crypto-chip resistant to differential power analysis," *Proceedings of European Solid-State Circuits Conference - ESSCIRC 2004*, Leuven, Belgium, pp. 307-310, Sept. 2004.

[22] R. L. Rivest, M. J. B. Robshaw, R. Sydney, and Y. L. Yin, "The RC6 block cipher," v1.1, Aug. 1998, available at http://www.rsasecurity.com/rsalabs/rc6.

[23] E. Trichina and T. Korkishko, "Small size, low power, side-channel-immune AES coprocessor," *presented at the 4. Conference on the Advanced Encryption Standard (AES)*, Bonn, Germany, May 2004 (see also E. Trichina, "Combinational logic design for AES subbyte transformation on masked data," *Cryptology ePrint Archive*, Report 2003/236, Nov. 2003, available at http://eprint.iacr.org/).

[24] J. Wolkerstorfer, E. Oswald, and M. Lamberger, "An ASIC implementation of the AES SBoxes," CT-RSA 2002, *Lecture Notes in Computer Science*, vol. 2271, pp. 67-78, 2002.