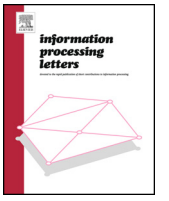
Information Processing Letters 182 (2023) 106395

Contents lists available at ScienceDirect

Information Processing Letters

journal homepage: www.elsevier.com/locate/ipl



Domain-oriented masked bit-parallel finite-field multiplier against side-channel attacks

José L. Imaña [a,*], Siemen Dhooghe [b]

[a] Department of Computer Architecture and Automation, Complutense University, 28040 Madrid, Spain
[b] ESAT/COSIC, Department of Electrical Engineering, Katholieke Universiteit Leuven, B-3001 Leuven-Heverlee, Belgium

ARTICLE INFO

Article history:
Received 25 October 2022
Received in revised form 6 February 2023
Accepted 31 March 2023
Available online 5 April 2023

Keywords:
Side-channel analysis (SCA)
Cryptography
Domain-oriented masking
Finite field
Multiplier

ABSTRACT

Side-Channel Analysis (SCA) constitutes a serious threat to the security of implemented cryptosystems. In SCA, the attacker can obtain information leakage from a device executing cryptographic algorithms by means of the measure of side-channels such as power consumption, electromagnetic radiation and execution time. For this reason, effective countermeasures against SCA are indispensable in implemented cryptographic devices. The use of masking schemes (in which intermediate computations are independent from the sensible input data) constitutes the most effective approach to achieve resistance against physical attacks. Among the different masking methods proposed for hardware, domain-oriented masking is one of the most promising due to its lower implementation costs, level of security and glitch resistance. In this paper, a new bit-parallel first-order domain-oriented masked finite field multiplier is presented which incorporates the addition of fresh random values without increasing the computation delay. Explicit expressions for the computation of the new masked multiplier for the binary extension field used in the *Advanced Encryption Standard* (AES) are also given.

1. Introduction

Side-channel analysis is a class of attacks in which the attacker can obtain information leakage via physical side-channels such as power consumption or electromagnetic radiation. The method was first proposed in 1999 by Kocher et al. [1]. The most promising method to obtain resistance against Side-Channel Analysis (SCA) is to make sensitive computations independent from the processed data by using *masking schemes* as originally proposed by Goubin and Patarin [2] and Chari et al. [3]. A common feature of these approaches is the requirement of random values in order to mask the data being processed. However, the naive implementation of the method is vulnerable against glitches.

A masking scheme immune against glitches is the *threshold implementation* (TI) [4], [5] method, where a variable is split into a number of *shares* and three important properties (*correctness*, *non-completeness* and *uniformity*) must be fulfilled. There exist many hardware implementations protected against SCA that are based on TI [6]. However, the extension of TI to higher protection orders has shown to be vulnerable and is very costly (due to the high number of shares and the amount of fresh random bits needed) [7].

A *domain-oriented* masking (DOM) method was introduced in [8] that can be implemented for arbitrary protection orders without being vulnerable to glitches. In the DOM methodology, each share of a variable is associated with a share *domain*, where all shares of a domain are independent from the shares of other domains. *Inner-*

* Corresponding author.
E-mail addresses: jluimana@ucm.es (J.L. Imaña), siemen.dhooghe@esat.kuleuven.be (S. Dhooghe).

https://doi.org/10.1016/j.ipl.2023.106395

domain and *cross-domain* products are computed in the DOM method, which combine shares within one domain and from different domains, respectively. DOM always uses the minimum number of shares and a lower number of required fresh random bits, therefore providing lower implementation costs without compromising security compared to TI.

Since masking schemes impose significant delays to the computation, due to the use of registers for security purposes rather than pipelining purposes, the reduction of the latency of masking schemes has been a popular research field. For example, the work by Groß et al. [9] allows for trade-offs between the number of register stages (i.e. latency of the design) and the number of shares (i.e. area of the design).

In this paper, a new bit-parallel first-order domain-oriented masked finite field $\mathbb{F}_{2^m}$ multiplier is presented that incorporates the addition (XOR) of fresh random values into the computation of the *cross-domain* products without increasing the computation delay and without significantly increasing the area cost. Explicit expressions for the computation of the new masked multiplier for the binary extension field used in the *Advanced Encryption Standard* (AES) are also given.

2. Background

Let $f(y) = \sum_{i=0}^{m} f_i y^i$ be an irreducible polynomial of degree m over the binary field $\mathbb{F}_2 = \{0, 1\}$ and let α be a root of $f(y)$. Any element X[1] of the binary extension field $\mathbb{F}_{2^m}$ can be represented in the polynomial basis $\{1, \alpha, \ldots, \alpha^{m-1}\}$ as $X = \sum_{i=0}^{m-1} x_i \alpha^i$, with $x_i \in \mathbb{F}_2$. Although the classic polynomial basis multiplication over $\mathbb{F}_{2^m}$ requires a multiplication of polynomials followed by a reduction modulo the irreducible polynomial $f(y)$, several methods combining these two steps using a *product matrix* have been proposed in the literature [10], [11]. In [12], a method for the construction of multipliers over $\mathbb{F}_{2^m}$ generated by the irreducible polynomial $f(y)$ was given. In [12], in order to compute the product $C = A \cdot B$, with $A, B \in \mathbb{F}_{2^m}$, the functions $\mathbf{S_i}$ $(1 \le i \le m)$ and $\mathbf{T_i}$ $(0 \le i \le m-2)$ were defined. These functions are given by the addition (XOR) of terms $x_k = (a_k b_k)$ and $z_i^j = (a_i b_j \oplus a_j b_i)$, where $a_i, b_i \in \mathbb{F}_2$ are the coordinates of A and B, respectively. The expressions of $\mathbf{S_i}(x_k, z_i^j)$ and $\mathbf{T_i}(x_k, z_i^j)$ were given in [12] as

$$\mathbf{S_i} = x_p \oplus \bigoplus_{h=0}^{p-1} z_h^{i-h-1}, \mathbf{T_i} = x_q \oplus \bigoplus_{j=1}^{r-(i+1)} z_{i+j}^{m-j}, \quad (1)$$

where $p = \lfloor i/2 \rfloor$ and $q = (\lceil m/2 \rceil + \lfloor i/2 \rfloor)$. The term x_p only appears for i *odd* and x_q only appears for m and i *even* or for m and i *odd*. In this case, $r = q$. Otherwise, the term x_q does not appear and $r = (\lceil m/2 \rceil + \lceil i/2 \rceil)$. The coordinates of the product $C = A \cdot B$ can be computed as the sum of some of these $\mathbf{S_i}$ and $\mathbf{T_i}$ functions depending on the irreducible polynomial selected for the field [12], [13].

[1] For any $X \in \mathbb{F}_{2^m}$, lowercase x_i represents the i^{th} bit of X.

It can be observed that the terms given in (1) can be implemented using AND gates (for the products $a_i b_j$) and binary-trees of XOR gates. Furthermore, the depth (number of levels) of XOR binary-trees are different for the $\mathbf{S_i}$ and $\mathbf{T_i}$ terms. The sum of products expressions of $\mathbf{S_i}$ and $\mathbf{T_i}$ terms for $\mathbb{F}_{2^8}$ using (1) are given in Table 1. From these expressions it must be noted that, for example, the term $\mathbf{S_8}$ requires the addition of eight product terms, so a binary-tree of seven 2-input XOR gates with depth three is needed for the implementation (in fact, for the addition of v terms, $\lceil log_2 v \rceil$ levels of 2-input XOR gates are needed). However, $\mathbf{T_4}$ requires the addition of three product terms, so it can be implemented with a binary-tree of two XOR gates with depth two. Table 1 shows the complexities of $\mathbf{S_i}$ and $\mathbf{T_i}$ terms for $\mathbb{F}_{2^8}$, where #AND and #XOR represent the number of 2-input AND and XOR gates, respectively, and #AND-l and #XOR-l represent the number of 2-input AND and XOR levels, respectively, for each term.

In *masking* schemes, computations are made independent from the data being processed. In order to do that, a sensitive data X[2] is split into $d+1$ shares (where d is called the *masking order*) in such a way that $X = X_1 \oplus X_2 \oplus \ldots \oplus X_d$. It has been proven that the complexity of a successful side-channel attack against a masked implementation increases exponentially with the masking order [3]. For this reason, the design of efficient masking schemes is an important research topic. Intermediate signals in a given masking scheme need to be statistically independent of all unshared inputs and outputs, which often requires the addition of fresh random shares to the intermediate results. While linear functions over the binary extension field $\mathbb{F}_{2^m}$ can be easily implemented, the implementation of nonlinear operations is quite difficult. For this reason, $\mathbb{F}_{2^m}$ masked multipliers can be used to compare different masking schemes.

3. Domain-oriented masking

In this section, we review the domain-oriented masking (DOM) scheme by Groß et al. [8]. More specifically, we review the multiplier which takes two independently distributed share vectors. We note that the multiplier working on dependently distributed shares is insecure against higher-order attacks as noted by Moos et al. [14].

In DOM, each share of a variable is associated with a share *domain*. For example, if a variable X has two shares X_1 and X_2, each share is associated with the domains 1 and 2, respectively. In order to achieve d^{th}-order security, a domain-oriented implementation uses $d+1$ shares for each variable with, therefore, $d+1$ domains. The idea of the domain-oriented methodology is that the shares of all domains are independent from the shares of other domains. This implies that each coordinate function (be it over $\mathbb{F}_2$ or $\mathbb{F}_{2^m}$) is independent of one share of each input (or more for higher-order security). Thus, the multiplier is (higher-order) non-complete as defined in [4], [5]. For example, if the variables $X = X_1 \oplus X_2$ and $Y = Y_1 \oplus Y_2$ (first-order security) and a function has as inputs X_1 and Y_1 (domain

[2] For any data X in masking, uppercase X_i represents the i^{th} share of X.

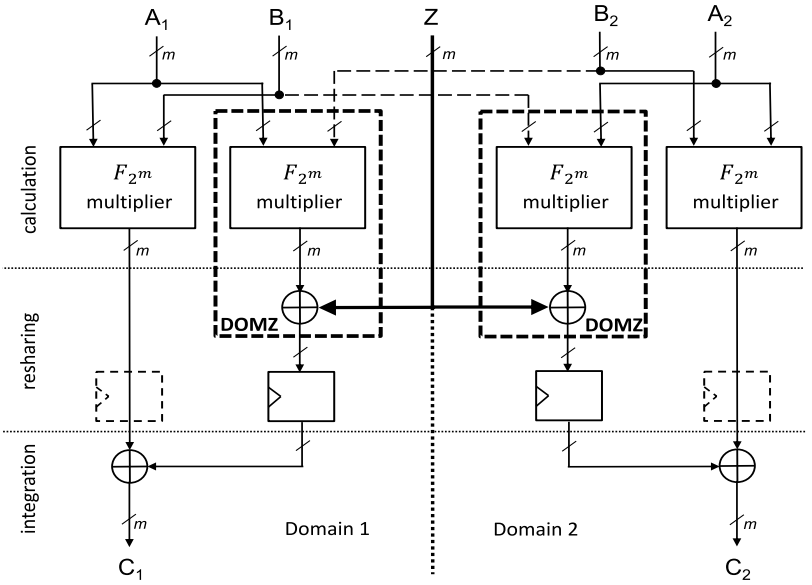


Fig. 1. First-order domain-oriented multiplier.

1), then all intermediate values computed by the function are independent of the original variables X and Y because the shares X_2 and Y_2 (belonging to domain 2) are not part of the function. For linear functions, the independence of domains only requires the use of shares belonging to one domain, but for non-linear functions, shares from different domains need to be used. In order to maintain the independence of the shares in the different domains, the domain-oriented approach adds a fresh random share Z and uses registers to prevent glitch propagation among domains.

For d^{th}-order security, a domain-oriented masked multiplier was given in [8]. In order to compute the product $C = A \cdot B$, with $A, B \in \mathbb{F}_{2^m}$, using d^{th}-order security, the shares $A = A_1 \oplus A_2 \oplus \ldots \oplus A_d$, $B = B_1 \oplus \ldots \oplus B_d$ and $C = C_1 \oplus \ldots \oplus C_d$, with $A_i, B_i, C_i \in \mathbb{F}_{2^m}$, $i = 1, 2, \ldots d$, are considered. The first-order secure domain-oriented masked multiplier given in [8] is shown in Fig. 1.

The domain-oriented masked multiplier in Fig. 1 consists of two share domains, in such a way that the operands $A = A_1 \oplus A_2$ and $B = B_1 \oplus B_2$ are fed to the multiplier by means of their shares A_1, B_1 (domain 1) and A_2, B_2 (domain 2), while that the output product is given by the two shares $C = C_1 \oplus C_2$. The sharings for A and B need to be uniformly random and independent from each other.

In order to compute the output shares, the DOM multiplier given in [8] performs the three steps *calculation*, *resharing* and *integration*. In the *calculation* step, the $\mathbb{F}_{2^m}$ multiplications $A_1 \cdot B_1$, $A_1 \cdot B_2$, $A_2 \cdot B_1$ and $A_2 \cdot B_2$ must be performed. In [8], the products $A_1 \cdot B_1$ and $A_2 \cdot B_2$ are denoted *inner-domain* and the products $A_1 \cdot B_2$ and $A_2 \cdot B_1$ are denoted *cross-domain*. Inner-domain products only combine shares within one domain, while that cross-domain products combine shares from different domains. The integration of cross-domain products into a given domain is achieved in the *resharing* step, where a fresh random value Z is added (XOR) to the cross-domain products, therefore becoming statistically independent from the other values. The inclusion of this resharing XOR in the output of cross-domain products introduces an additional delay for the computation of the domain-oriented multiplication. In order to minimize the number of fresh shares and therefore reduce the overload, in [8] the same fresh share Z is used for the resharing of cross-domain products in first-order security (see Fig. 1). Furthermore, registers are included in the last part of the resharing step to prevent glitch propagation (the dotted registers in Fig. 1 are optional and are only required for pipelining). In the last *integration* step, the reshared cross-domain products are added to the corresponding inner-domain products for each domain, obtaining the two shares of the $\mathbb{F}_{2^m}$ product $C = C_1 \oplus C_2$. The security of this domain-oriented multiplier against first-order attacks is due to each domain including only inner-domain products and cross-domain products that have been reshared with a fresh random share that is only used once in each domain.

As shown in Fig. 1, the domain-oriented first-order masked multiplier requires four $\mathbb{F}_{2^m}$ multipliers for the computation of the inner-domain and cross-domain products. Furthermore, a fresh random value $Z \in \mathbb{F}_{2^m}$ is added to a cross-domain finite field multiplier in the *resharing* step. This resharing XOR introduces an additional delay in the overall computation. In next sections, we introduce a new finite field $\mathbb{F}_{2^m}$ multiplier where the resharing XOR is embedded in the finite field multiplier, in such a way that the new $\mathbb{F}_{2^m}$ bit-parallel multiplier performs the field operation $C = A \cdot B \oplus Z$, without increasing the number of XOR levels. We name this new domain-oriented $\mathbb{F}_{2^m}$ module as DOMZ-multiplier. Fig. 1 shows with a thick dotted box the resharing XOR and the $\mathbb{F}_{2^m}$ multiplier which combination constitutes the new DOMZ-multiplier.

Table 1
Expressions and complexities of $\mathbf{S_i}$ and $\mathbf{T_i}$ terms for $\mathbb{F}_{2^8}$.

			#AND	#XOR	#AND-l	#XOR-l
$\mathbf{S_1}$	x_0	a_0b_0	1	0	1	0
$\mathbf{S_2}$	z_0^1	$(a_0b_1 \oplus a_1b_0)$	2	1	1	1
$\mathbf{S_3}$	$x_1 \oplus z_0^2$	$a_1b_1 \oplus (a_0b_2 \oplus a_2b_0)$	3	2	1	2
$\mathbf{S_4}$	$z_0^3 \oplus z_1^2$	$(a_0b_3 \oplus a_3b_0) \oplus (a_1b_2 \oplus a_2b_1)$	4	3	1	2
$\mathbf{S_5}$	$x_2 \oplus z_0^4 \oplus z_1^3$	$a_2b_2 \oplus (a_0b_4 \oplus a_4b_0) \oplus (a_1b_3 \oplus a_3b_1)$	5	4	1	3
$\mathbf{S_6}$	$z_0^5 \oplus z_1^4 \oplus z_2^3$	$(a_0b_5 \oplus a_5b_0) \oplus (a_1b_4 \oplus a_4b_1) \oplus (a_2b_3 \oplus a_3b_2)$	6	5	1	3
$\mathbf{S_7}$	$x_3 \oplus z_0^6 \oplus z_1^5 \oplus z_2^4$	$a_3b_3 \oplus (a_0b_6 \oplus a_6b_0) \oplus (a_1b_5 \oplus a_5b_1) \oplus (a_2c_4 \oplus a_4c_2)$	7	6	1	3
$\mathbf{S_8}$	$z_0^7 \oplus z_1^6 \oplus z_2^5 \oplus z_3^4$	$(a_0b_7 \oplus a_7b_0) \oplus (a_1b_6 \oplus a_6b_1) \oplus (a_2b_5 \oplus a_5b_2) \oplus (a_3b_4 \oplus a_4b_3)$	8	7	1	3
$\mathbf{T_0}$	$x_4 \oplus z_1^7 \oplus z_2^6 \oplus z_3^5$	$a_4b_4 \oplus (a_1b_7 \oplus a_7b_1) \oplus (a_2b_6 \oplus a_6b_2) \oplus (a_3b_5 \oplus a_5b_3)$	7	6	1	3
$\mathbf{T_1}$	$z_2^7 \oplus z_3^6 \oplus z_4^5$	$(a_2b_7 \oplus a_7b_2) \oplus (a_3b_6 \oplus a_6b_3) \oplus (a_4b_5 \oplus a_5b_4)$	6	5	1	3
$\mathbf{T_2}$	$x_5 \oplus z_3^7 \oplus z_4^6$	$a_5b_5 \oplus (a_3b_7 \oplus a_7b_3) \oplus (a_4b_6 \oplus a_6b_4)$	5	4	1	3
$\mathbf{T_3}$	$z_4^7 \oplus z_5^6$	$(a_4b_7 \oplus a_7b_4) \oplus (a_5b_6 \oplus a_6b_5)$	4	3	1	2
$\mathbf{T_4}$	$x_6 \oplus z_5^7$	$a_6b_6 \oplus (a_5b_7 \oplus a_7b_5)$	3	2	1	2
$\mathbf{T_5}$	z_6^7	$(a_6b_7 \oplus a_7b_6)$	2	1	1	1
$\mathbf{T_6}$	x_7	a_7b_7	1	0	1	0

4. New domain-oriented masked $\mathbb{F}_{2^m}$ multiplier

In this section, we optimize the calculation of the DOM multiplier. Such an optimization does not alter the security of the scheme as the same mathematical equations are still calculated. Namely, each coordinate function of the optimization remains non-complete as defined by Nikova et al. [4].

As given in Section 3, the domain-oriented multiplication requires a *resharing* step in which a fresh random value $Z \in \mathbb{F}_{2^m}$ is added to a cross-domain finite field multiplier. This addition (XOR) introduces an additional delay to the overall computation. It can be observed that this resharing XOR can be embedded in the previously given multiplier, in such a way that a new $\mathbb{F}_{2^m}$ bit-parallel domain-oriented multiplier performing the field operation $C = A \cdot B \oplus Z$, without increasing the number of XOR levels, can be given. We provide the first-order variant of the new domain-oriented module and name it DOMZ-multiplier. The resharing XOR and the $\mathbb{F}_{2^m}$ multiplier that are combined to constitute the new DOMZ-multiplier are shown with a thick dotted box in Fig. 1.

As shown in Section 2 (Table 1), the $\mathbf{S_i}$ and $\mathbf{T_i}$ terms present a different number of XOR levels for their implementation. Using the associative property of the exclusive-OR, a 2-input *resharing* XOR can be associated to one of the $\mathbf{S_i}$ or $\mathbf{T_i}$ terms (with the lowest number of XOR levels) that contribute to the computation of each product coordinate. For the product coordinate c_i, the corresponding 2-input *resharing* XOR has the bit z_i as one of its inputs, and one of the $\mathbf{S_i}$ or $\mathbf{T_i}$ terms with lowest XOR levels appearing in the expression of c_i as the other input. This association of the *resharing* XOR to one of the $\mathbf{S_i}$ or $\mathbf{T_i}$ terms with the lowest number of XOR levels constitutes the general strategy for the construction of the DOMZ-multiplier. It must be noted that the lowest number of XOR levels correspond with those $\mathbf{S_i}$'s with lowest subindices and with $\mathbf{T_i}$'s with highest subindices. In the event that there were two $\mathbf{S_i}$ or $\mathbf{T_i}$ terms with the same number of levels, the *resharing* XOR could be arbitrarily associated to any of them. This strategy can be applied to any binary extension field multiplier over $\mathbb{F}_{2^m}$ (with $m \geq 3$) generated by different irreducible polynomials when $\mathbf{S_i}$ and $\mathbf{T_i}$ functions are used for the implementation [12], [13], [15].

Table 2
Coordinates c_i of the product for $f(y) = y^8 + y^4 + y^3 + y + 1$.

c_0	=	$\mathbf{S_1} \oplus \mathbf{T_0} \oplus \mathbf{T_4} \oplus \mathbf{T_5}$;
c_1	=	$\mathbf{S_2} \oplus \mathbf{T_0} \oplus \mathbf{T_1} \oplus \mathbf{T_4} \oplus \mathbf{T_6}$;
c_2	=	$\mathbf{S_3} \oplus \mathbf{T_1} \oplus \mathbf{T_2} \oplus \mathbf{T_5}$;
c_3	=	$\mathbf{S_4} \oplus \mathbf{T_0} \oplus \mathbf{T_2} \oplus \mathbf{T_3} \oplus \mathbf{T_4} \oplus \mathbf{T_5} \oplus \mathbf{T_6}$;
c_4	=	$\mathbf{S_5} \oplus \mathbf{T_0} \oplus \mathbf{T_1} \oplus \mathbf{T_3} \oplus \mathbf{T_6}$;
c_5	=	$\mathbf{S_6} \oplus \mathbf{T_1} \oplus \mathbf{T_2} \oplus \mathbf{T_4}$;
c_6	=	$\mathbf{S_7} \oplus \mathbf{T_2} \oplus \mathbf{T_3} \oplus \mathbf{T_5}$;
c_7	=	$\mathbf{S_8} \oplus \mathbf{T_3} \oplus \mathbf{T_4} \oplus \mathbf{T_6}$;

This new technique is illustrated in Subsection 4.1, where we show the new DOMZ-multiplier for the *type I irreducible pentanomial* $f(y) = y^8 + y^4 + y^3 + y + 1$ used in the Advanced Encryption Standard (AES) [16].

4.1. DOMZ-multiplier for $f(y) = y^8 + y^4 + y^3 + y + 1$

Type I irreducible pentanomials [17] are defined as $f(y) = y^m + y^{n+1} + y^n + y + 1$, with $2 \leq n \leq \lfloor \frac{m}{2} \rfloor - 1$. These pentanomials are used in a wide number of important applications, such as AES. In [12], general expressions for the computation of the product $C = A \cdot B$ using $\mathbf{S_i}$ and $\mathbf{T_i}$ terms for *type I irreducible pentanomials* were given. The coordinates of the product for the AES irreducible pentanomial $f(y) = y^8 + y^4 + y^3 + y + 1$ are given in Table 2 [12].

From the raw additions of $\mathbf{S_i}$ and $\mathbf{T_i}$ terms given in Table 2, the lowest-delay expressions for the computation of the product coordinates are shown in Table 3, where the number of XOR levels given in Table 1 have been used in order to perform the additions achieving the lowest number of XOR levels. The parentheses used in Table 3 represent the addition of terms that must be done to achieve the lowest-delay. Using these expressions and the complexities given in Table 1, Fig. 2(a) shows the implementation of the most complex coordinate c_3 of the AES multiplier, where the triangles represent the binary-trees of XOR gates for each $\mathbf{S_i}$ and $\mathbf{T_i}$ terms (with the number inside each triangle representing the number of XOR levels). From the expressions in Table 3 and from Fig. 2, we have that the number of 2-input XOR gates needed for the

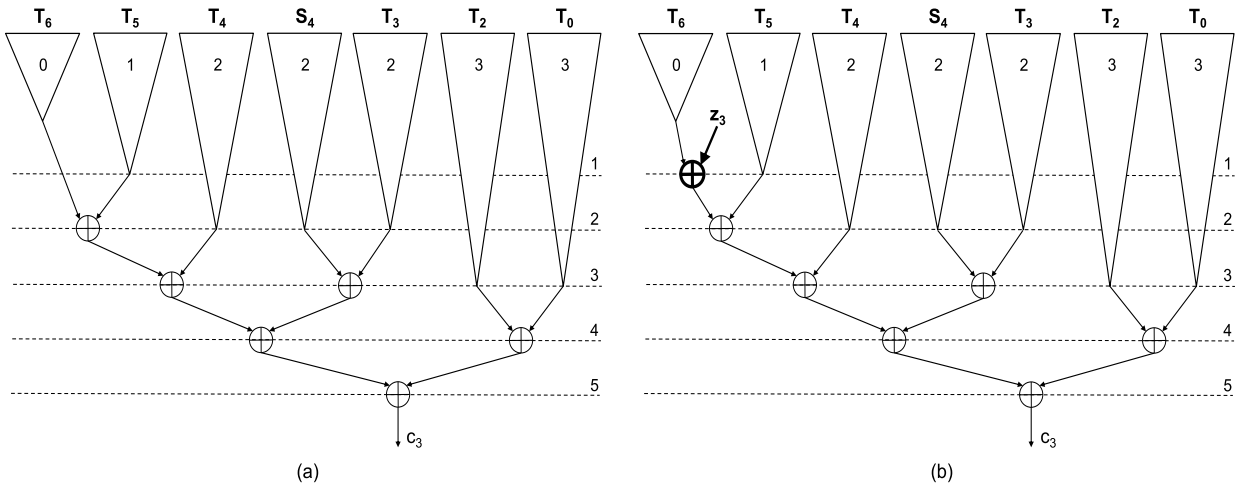


Fig. 2. (a) Construction of coordinate c_3 of AES multiplier with the lowest number of XOR levels. (b) Construction of coordinate c_3 of AES DOMZ-multiplier with maintenance of the lowest number of XOR levels.

Table 3
Coordinates c_i of AES multiplier with lowest delay.

c_0	=	$((\mathbf{S_1} \oplus \mathbf{T_5}) \oplus \mathbf{T_4}) \oplus \mathbf{T_0}$;
c_1	=	$((\mathbf{T_6} \oplus \mathbf{S_2}) \oplus \mathbf{T_4}) \oplus (\mathbf{T_0} \oplus \mathbf{T_1})$;
c_2	=	$(\mathbf{T_5} \oplus \mathbf{S_3}) \oplus (\mathbf{T_1} \oplus \mathbf{T_2})$;
c_3	=	$(((\mathbf{T_6} \oplus \mathbf{T_5}) \oplus \mathbf{T_4}) \oplus (\mathbf{S_4} \oplus \mathbf{T_3})) \oplus (\mathbf{T_0} \oplus \mathbf{T_2})$;
c_4	=	$((\mathbf{T_6} \oplus \mathbf{T_3}) \oplus \mathbf{T_1}) \oplus (\mathbf{S_5} \oplus \mathbf{T_0})$;
c_5	=	$(\mathbf{T_4} \oplus \mathbf{T_2}) \oplus (\mathbf{S_6} \oplus \mathbf{T_1})$;
c_6	=	$(\mathbf{T_5} \oplus \mathbf{T_3}) \oplus (\mathbf{S_7} \oplus \mathbf{T_2})$;
c_7	=	$(\mathbf{T_6} \oplus \mathbf{T_4}) \oplus (\mathbf{S_8} \oplus \mathbf{T_3})$;

Table 4
Coordinates c_i of the new AES DOMZ-multiplier.

c_0	=	$(((\mathbf{S_1} \oplus z_0) \oplus \mathbf{T_5}) \oplus \mathbf{T_4}) \oplus \mathbf{T_0}$;
c_1	=	$(((\mathbf{T_6} \oplus z_1) \oplus \mathbf{S_2}) \oplus \mathbf{T_4}) \oplus (\mathbf{T_0} \oplus \mathbf{T_1})$;
c_2	=	$((\mathbf{T_5} \oplus z_2) \oplus \mathbf{S_3}) \oplus (\mathbf{T_1} \oplus \mathbf{T_2})$;
c_3	=	$((((\mathbf{T_6} \oplus z_3) \oplus \mathbf{T_5}) \oplus \mathbf{T_4}) \oplus (\mathbf{S_4} \oplus \mathbf{T_3})) \oplus (\mathbf{T_0} \oplus \mathbf{T_2})$;
c_4	=	$(((\mathbf{T_6} \oplus z_4) \oplus \mathbf{T_3}) \oplus \mathbf{T_1}) \oplus (\mathbf{S_5} \oplus \mathbf{T_0})$;
c_5	=	$((\mathbf{T_4} \oplus z_5) \oplus \mathbf{T_2}) \oplus (\mathbf{S_6} \oplus \mathbf{T_1})$;
c_6	=	$((\mathbf{T_5} \oplus z_6) \oplus \mathbf{T_3}) \oplus (\mathbf{S_7} \oplus \mathbf{T_2})$;
c_7	=	$((\mathbf{T_6} \oplus z_7) \oplus \mathbf{T_4}) \oplus (\mathbf{S_8} \oplus \mathbf{T_3})$;

construction of a coordinate c_i given by the addition of w $\mathbf{S_i}$ and $\mathbf{T_i}$ terms is $w-1$. Furthermore, the parenthesized additions given in Table 3 assure the achievement of the lowest delay.

For the construction of the DOMZ-multiplier, it can be observed in Fig. 2(a) that the different levels of XOR gates needed for each $\mathbf{S_i}$ and $\mathbf{T_i}$ terms can be used to embed the *resharing* XOR gates and associate them to those terms with the lowest number of XOR levels without increasing the delay. Fig. 2(b) shows the implementation of the coordinate c_3 of the AES multiplier with the inclusion of the *resharing* XOR gate corresponding with the z_3 coordinate of the fresh random value $Z \in \mathbb{F}_{2^8}$. This *resharing* XOR is associated with the term $\mathbf{T_6}$ that is the one with the lowest number of XOR levels (0 levels) in the expression for the computation of c_3 coordinate. It can be observed in Fig. 2(b) that the inclusion of the additional XOR associated with $\mathbf{T_6}$ does not increment the delay (number of XOR levels). Table 4 shows the coordinates c_i of the new AES DOMZ-multiplier where the above strategy has been used. The parentheses in Table 4 represent the addition of terms that must be done to achieve the lowest-delay expressions. From the expressions given in Table 4, it can be observed that only eight 2-input XOR gates are needed to embed the *resharing* XOR required by the domain-oriented $\mathbb{F}_{2^8}$ multiplication into the cross-domain $\mathbb{F}_{2^8}$ DOMZ-multiplier (shown in Fig. 1 with thick dotted box) performing $A \cdot B \oplus Z$ operation.

4.2. Complexity analysis and comparison with other $\mathbb{F}_{2^8}$ multipliers

In order to determine the theoretical complexity of the DOMZ-multiplier given in Table 4, the complexities of the $\mathbf{S_i}$ and $\mathbf{T_i}$ terms must be known. These functions are given as *binary trees* of 2-input XOR gates with a lower level of 2-input AND gates (for the products $a_i b_j$ of the coordinates of A and B). It can be proven [12] that the number of 2-input AND gates, 2-input XOR gates and delay of $\mathbf{S_i}$ and $\mathbf{T_i}$ terms are i, $i-1$, $T_{AND} + \lceil log_2(2\lceil \frac{i}{2} \rceil) \rceil) T_{XOR}$ and $m-i-1$, $m-i-2$, $T_{AND} + \lceil log_2(m-i-1) \rceil) T_{XOR}$, respectively, where T_{AND} and T_{XOR} stand for the delay of 2-input AND and XOR gates, respectively. The terms with highest complexities are $\mathbf{S_m}$ and $\mathbf{T_0}$. The total contribution of $\mathbf{S_i}$ and $\mathbf{T_i}$ to the *space* complexity is m^2 AND and $(m-1)^2$ XOR gates. Table 1 shows the complexities of $\mathbf{S_i}$ and $\mathbf{T_i}$ terms for $\mathbb{F}_{2^8}$, where #AND and #XOR represent the number of 2-input AND and XOR gates, respectively, and #AND-l and #XOR-l represent the number of 2-input AND and XOR levels, respectively, for each term. It can be observed that the total contribution of these terms to the $\mathbb{F}_{2^8}$ DOMZ-multiplier *space* complexity is 64 AND and 49 XOR gates.

From Table 4, the addition of 5, 6, 5, 8, 6, 5, 5 and 5 $\mathbf{S_i}/\mathbf{T_i}$ terms and z_i coordinates of fresh random value Z for the c_0, c_1, c_2, c_3, c_4, c_5, c_6 and c_7 product coordinates, respectively, are needed. Therefore, the number of XOR gates needed for each product coordinate c_i, $i = 0, \ldots, 7$, is 4, 5, 4, 7, 5, 4, 4 and 4, respectively, whose sum corresponds

Table 5
Theoretical complexities of $A \cdot B \oplus Z$ over $\mathbb{F}_{2^8}$.

	#AND	#XOR	Delay
[19]	64	85	$T_{AND} + 10T_{XOR}$
[20]	52	115	$T_{AND} + 8T_{XOR}$
[12]	64	79	$T_{AND} + 7T_{XOR}$
[17]	64	86	$T_{AND} + 7T_{XOR}$
[10]	64	80	$T_{AND} + 7T_{XOR}$
[15]	64	105	$T_{AND} + 6T_{XOR}$
DOMZ-mult.	64	86	$T_{AND} + 5T_{XOR}$

with a total of 37 2-input XOR gates. It is important to note that in order to have the lowest delay, no *subexpression sharing* has been used in Table 4. *Subexpression sharing* corresponds with subexpressions that could be shared among different product coordinates c_i, therefore reducing the area complexity. For example, the addition $\mathbf{T_4} \oplus \mathbf{T_6}$ in Table 2 appears in the product coordinates c_1, c_3 and c_7, so it could be shared in order to reduce the number of XOR gates.

The overall space and time complexity of the bit-parallel DOMZ-multiplier for AES $\mathbb{F}_{2^8}$ is given in Table 5. To the knowledge of the authors, no similar bit-parallel $\mathbb{F}_{2^m}$ multipliers performing $A \cdot B \oplus Z$ have been proposed in the literature. For this reason, Table 5 includes the complexities of $\mathbb{F}_{2^8}$ finite field multipliers for type-I pentanomial $f(y) = y^8 + y^4 + y^3 + y + 1$ followed by an additional level of eight XOR gates in parallel (to perform the addition of the fresh random $Z \in \mathbb{F}_{2^8}$). From Table 5, it can be observed that the DOMZ-multiplier here presented can compute the operation $A \cdot B \oplus Z$ with only five levels of 2-input XOR gates, therefore obtaining a reduction of 16.7% in the number of XOR levels in comparison with the result of [15]. Furthermore, the reduction in the number of XOR gates is 18.1% with respect to [15]. It must be noted that in [18], an LFSR (*Linear-Feedback Shift Register*)-based $\mathbb{F}_{2^8}$ multiplier performing $A \cdot B \oplus Z$ was presented. However, as this is a sequential architecture (different from the bit-parallel one here considered) it has not been included in Table 5 for fair comparison.

5. Conclusion

Domain-oriented masking is one of the most promising schemes against physical attacks due to its low implementation costs, level of security and glitch resistance. In this paper, a new bit-parallel first-order domain-oriented masked finite field (DOMZ) multiplier performing $A \cdot B \oplus Z$ in $\mathbb{F}_{2^m}$ has been presented. The DOMZ-multiplier incorporates the addition of fresh random values without increasing the computation delay. Explicit expressions for the computation of the new masked multiplier for the binary field $\mathbb{F}_{2^8}$ generated by the type-I pentanomial $f(y) = y^8 + y^4 + y^3 + y + 1$ used in AES have been also given. However, a full implementation of the AES is outside the scope of this work and would be interesting future work. From the comparison with other structures performing $A \cdot B \oplus Z$, it can be observed that the new DOMZ-multiplier here presented can compute this operation with only five levels of 2-input XOR gates, therefore obtaining a reduction of 16.7% in the number of XOR levels in comparison with other alternatives. The practical impact of the new proposed multiplier could depend on the target device selected for the implementation. The reduction of the number of XOR levels in DOMZ-multiplier could reduce the latency in a pipelined implementation for a given cryptosystem. We note that in masking, registers must be placed to secure against glitches. As a result, all masking implementations are highly pipelined. As mentioned previously, we will consider as an interesting future work the use of the proposed DOMZ-multiplier for a full implementation of AES in order to check the improvement given by the proposed multiplier.

Declaration of competing interest

The authors declare that they have no known competing financial interests or personal relationships that could have appeared to influence the work reported in this paper.

Data availability

Data will be made available on request.

Acknowledgements

The authors would like to thank the reviewers for their valuable comments to improve the quality of the paper.

The work of José L. Imaña was supported by grant PID2021-123041OB-I00 funded by MCIN/AEI/ 10.13039/ 501100011033 and by "ERDF A way of making Europe", and by the CM under grant S2018/TCS-4423. The work of Siemen Dhooghe was supported by a PhD Fellowship from the Research Foundation – Flanders (FWO).

References

[1] P. Kocher, J. Jaffe, B. Jun, Differential power analysis, in: Advances in Cryptology CRYPTO 1999, in: LNCS, vol. 1666, 1999, pp. 388–397.
[2] L. Goubin, J. Patarin, Des and differential power analysis the duplication method, in: Cryptographic Hardware and Embedded Systems, in: LNCS, vol. 1717, 1999, pp. 158–172.
[3] S. Chari, C. Jutla, J. Rao, P. Rohatgi, Towards sound approaches to counteract power-analysis attacks, in: Advances in Cryptology CRYPTO 1999, in: LNCS, vol. 1666, 1999, pp. 398–412.
[4] S. Nikova, C. Rechberger, V. Rijmen, Threshold implementations against side-channel attacks and glitches, in: Information and Communications Security, in: LNCS, vol. 4307, 1999, pp. 529–545.
[5] B. Bilgin, B. Gierlichs, S. Nikova, V. Nikov, V. Rijmen, Higher-order threshold implementations, in: Advances in Cryptology ASIACRYPT 2014, in: LNCS, vol. 8874, 2014, pp. 326–343.
[6] B. Bilgin, B. Gierlichs, S. Nikova, V. Nikov, V. Rijmen, Trade-offs for threshold implementations illustrated on aes, IEEE Trans. Comput.-Aided Des. Integr. Circuits Syst. 34 (7) (2015) 1188–1200.
[7] O. Reparaz, B. Bilgin, S. Nikova, B. Gierlichs, I. Verbauwhede, Consolidating masking schemes, in: Advances in Cryptology CRYPTO 2015, in: LNCS, vol. 9215, 2015, pp. 764–783.
[8] H. Gross, S. Mangard, T. Korak, Domain-oriented masking: compact masked hardware implementations with arbitrary protection order, in: ACM Workshop on Theory of Implementation Security TIS 2016, 2016, p. 3.
[9] H. Groß, R. Iusupov, R. Bloem, Generic low-latency masking in hardware, IACR Trans. Cryptogr. Hardw. Embed. Syst. 2 (2018) 121.
[10] A. Reyhani-Masoleh, M. Hasan, Low complexity bit parallel architectures for polynomial basis multiplication over $gf(2^m)$, IEEE Trans. Comput. 53 (8) (2004) 945–959.

[11] S. Bayat-Sarmadi, M. Farmani, High-throughput low-complexity systolic Montgomery multiplication over $gf(2^m)$ based on trinomials, IEEE Trans. Circuits Syst. II, Express Briefs 62 (4) (2015) 377–381.
[12] J. Imaña, R. Hermida, F. Tirado, Low complexity bit-parallel multipliers based on a class of irreducible pentanomials, IEEE Trans. Very Large Scale Integr. (VLSI) Syst. 14 (12) (2006) 1388–1393.
[13] J. Imaña, Efficient polynomial basis multipliers for type ii irreducible pentanomials, IEEE Trans. Circuits Syst. II, Express Briefs 59 (11) (2012) 795–799.
[14] T. Moos, A. Moradi, T. Schneider, F. Standaert, Glitch-resistant masking revisited or why proofs in the robust probing model are needed, IACR Trans. Cryptogr. Hardw. Embed. Syst. 2019 (2) (2019) 256–292, https://doi.org/10.13154/tches.v2019.i2.256-292.
[15] J. Imaña, Fast bit-parallel binary multipliers based on type-I pentanomials, IEEE Trans. Comput. 67 (6) (2018) 898–904.
[16] Advanced encryption standard, Tech. Rep., 2001.
[17] F. Rodríguez-Henríquez, Ç.K. Koç, Parallel multipliers based on special irreducible pentanomials, IEEE Trans. Comput. 52 (2003) 1535–1542.
[18] F. Wegener, A. Moradi, Yet another size record for aes: a first-order sca secure aes s-box based on $gf(2^8)$ multiplication, in: 17th Smart Card Research and Advanced Application Conference CARDIS 2018, in: LNCS, vol. 11389, 2019, pp. 111–124.
[19] T. Zhang, K. Parhi, Systematic design of original and modified Mastrovito multipliers for general irreducible polynomials, IEEE Trans. Comput. 50 (7) (2001) 734–749.
[20] S.-M. Park, K.-Y. Chang, D. Hong, C. Seo, New efficient bit-parallel polynomial basis multiplier for special pentanomials, Integration 47 (2014) 130–139.