Microprocessors and Microsystems 103 (2023) 104945

Contents lists available at ScienceDirect

Microprocessors and Microsystems

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

Hardware architecture of Dillon's APN permutation for different primitive polynomials

José L. Imaña [a,*], Nikolay Kaleyski [b], Lilya Budaghyan [b]

[a] Department of Computer Architecture and Automation, Faculty of Physics, Complutense University, 28040 Madrid, Spain
[b] Selmer Center, Department of Informatics, University of Bergen, 5020 Bergen, Norway

ARTICLE INFO

Keywords:
Almost perfect nonlinear (APN)
Block cypher
S-box
Primitive polynomial
Hardware architecture

ABSTRACT

Cryptographically strong functions used as S-boxes in block cyphers are fundamental for the cypher's security. Their representation as lookup tables is possible for functions of small dimension. For larger dimensions, this is infeasible, especially if resources are limited. An alternative is representing the function as a polynomial over a finite field, and constructing a circuit evaluating this polynomial. We study how the choice of primitive polynomial affects the efficiency of hardware implementations. We take Dillon's permutation on 6 bits (the only known permutation in even dimension from the cryptographically optimal Almost Perfect Nonlinear functions) as a relevant example, and present hardware architectures, polynomial representations and hardware theoretical complexities for all primitive polynomials of degree six. To compare the efficiency, we report on results obtained from FPGA (Field Programmable Gate Array) implementations. To the best of our knowledge, no similar study has been given in the literature. We observe that using the primitive trinomial $y^6 + y + 1$ reduces the number of 2-input XOR gates up to 11.7% and the number of XOR gates $\times$ Delay metrics up to 13.2% with respect to the worst-case implementation. Therefore, the choice of primitive polynomial can profoundly impact the efficiency of such an implementation, and should be carefully considered.

1. Introduction

Block cyphers are cryptographic primitives that provide confidentiality by allowing the encryption and decryption of messages once a secret key has been shared. They are crucial to the design of many cryptographic protocols, and are thus critical for providing security in communication. Designing secure block cyphers is not an easy task, since they need to be resilient against any kind of cryptanalytic attack that an attacker might potentially employ. Furthermore, block cyphers must be designed in such a way that encryption and decryption are efficient in terms of time and memory, since cyphers may be used to process large amounts of data in practice.

The design of virtually all modern block cyphers revolves around one or more highly non-linear transformations combined with linear components. Since linear functions behave in a predictable way, it is the non-linear part of the cypher that provides its security; and it is the properties of this nonlinear part that can be used to measure the weakness or resilience to various cryptanalytic attacks. The nonlinear transformations are typically referred to as *substitution boxes*, or *S-boxes*. An S-box is a function that takes a sequence of n bits as input, and produces a sequence of m bits as output; for this reason, they are also called (n, m)*-functions*. One of the most important cases of such functions is when $n = m$, i.e. when the input sequence of bits is replaced with another sequence of the same length.

Differential [1] and linear [2] cryptanalysis are two of the most efficient attacks against block cyphers. Specific design criteria for S-boxes have been formulated that characterize how resilient they are against these attacks. The resistance of an S-box to differential and linear attacks is quantified by the *differential uniformity* [3] and the *nonlinearity* of the S-box, respectively. In order to resist differential attacks, the differential uniformity is required to be as low as possible; while in order to resist linear attacks, the nonlinearity of the S-box should be high. In the case when $n = m$, the functions having the best possible differential uniformity are called *almost perfect nonlinear (APN)*, while those having the best possible nonlinearity are called *almost bent (AB)*. Furthermore, it can be shown that any AB function is APN [4]. Unfortunately, AB functions exist only when n is odd, while in practice some of the most natural cases involve an even number of bits.

Besides being cryptographically strong (in terms of e.g. the differential uniformity and nonlinearity), it is frequently desirable, or

* Corresponding author.
E-mail addresses: jluimana@ucm.es (J.L. Imaña), Nikolay.Kaleyski@uib.no (N. Kaleyski), Lilya.Budaghyan@uib.no (L. Budaghyan).

https://doi.org/10.1016/j.micpro.2023.104945
Received 26 January 2023; Received in revised form 13 July 2023; Accepted 3 October 2023
Available online 6 October 2023

necessary, for S-boxes to be bijective. For instance, a generic construction of block cyphers called a Substitution Permutation Network (SPN) requires the S-box used to be bijective. The Rijndael cypher, implemented as the Advanced Encryption Standard (AES) by the National Institute of Standards and Technology (NIST) [5], is one of the most secure and widely used block cyphers today, and its design is based on an SPN. Rijndael has a bijective $(8, 8)$-function at the core of its design; surprisingly enough, this function is not APN so that its differential uniformity is not optimal. The reason for this is that, at the time of writing, no bijective APN functions on 8 bits are known. Moreover, the existence of bijective APN functions on an even number of bits larger than 6 is currently one of the most important open problems in the theory of cryptographic Boolean functions.

In contrast, optimal permutations with an odd number of variables have been known for years [6], but it was long believed that APN permutations on an even number of variables do not exist (in fact, it was proven [7,8] that APN permutations with four variables do not exist). However, Dillon et al. [9] found an APN permutation on six variables, and this is currently the only known example of an APN permutation with an even number of variables (up to equivalence).

APN permutations on an even number of bits are clearly of significant practical interest. For example, Dillon's permutation has been used in the design of the lightweight authenticated encryption algorithm FIDES [10]. Finding an APN permutation on 8 bits would allow us to construct an even more secure version of the Rijndael cypher; and knowing APN permutations on a larger number of variables would allow the construction of strong cyphers operating with larger blocks of data, which will becomes necessary in the future when technological advances make the currently available designs insecure.

While APN functions and permutations are mostly studied as mathematical objects and represented as e.g. polynomials over finite fields, their use in practice frequently requires them to be implemented in hardware. For small dimensions, it is possible to simply use a lookup table containing all values of the function. Since the size of this lookup table grows exponentially with the dimension, this clearly becomes problematic for functions on a larger number of bits, and especially in the case of a resource-constrained environment. For this reason, it is important to consider other hardware implementations that may be more efficient in terms of memory requirements [11]. A natural possibility is to make a hardware implementation performing finite field arithmetic and to have it evaluate the function (as a polynomial over the corresponding finite field) for a give input. However, there are multiple choices of a primitive polynomial with which to construct a finite field of a given dimension, each leading to a different univariate representation of the function. It is then important to know whether the choice of primitive polynomial affects the complexity of such a hardware implementation, and if so, by how much. As our results will demonstrate, while the cryptographic properties of a given function remain the same regardless of the choice of primitive polynomial, the complexity with which it can be computed in practice highly depend on the concrete implementation.

In this paper, we present a hardware architecture for Dillon's permutation implemented using the six different primitive polynomials of the finite field $\mathbb{F}_{2^6}$. Dillon's permutation is a natural choice for a function on which to study the effects of the choice of the primitive polynomial, since it is an APN function (and thus cryptographically optimal from the point of view of resistance to differential cryptanalysis) as well as bijective (since some constructions of cryptographic cyphers, such as substitution-permutation networks, require the S-box to be a permutation). We also give the corresponding univariate representations for the different polynomials. In order to compare the different representations, we give the hardware theoretical complexities and report on results obtained from FPGA implementations. To the best of our knowledge, no similar study has been given in the literature so far. From the theoretical and experimental results, we observe that the implementation of Dillon's permutation using the primitive trinomial $f(y) = y^6+y+1$ presents the highest reduction in complexity (compared to the worst case), as measured by the number of 2-input XOR gates and in the number of XOR gates × Delay metrics. Therefore the use of the primitive polynomial $f(y) = y^6 + y^4 + y^3 + y + 1$ with which the function is typically represented does not provide the best results from the point of view of a hardware implementation. We stress that Dillon's permutation is used here merely as one example of a cryptographically relevant function in low dimension, and the methodology and hardware architecture presented here can be generalized to other cryptographic functions, in such a way that univariate polynomial representations for different primitive polynomials could be considered and similar architectures to the one here proposed could also be determined.

Table 1
Dillon permutation $g(x)$ in hexadecimal.

	w																
		0	1	2	3	4	5	6	7	8	9	A	B	C	D	E	F
	0	00	04	05	0F	19	26	2B	03	2A	1A	10	29	0D	2F	12	20
v	1	25	36	33	2C	37	22	30	3B	24	17	0E	0B	18	32	3A	34
	2	23	07	31	3E	0A	0C	3C	08	01	02	39	16	1B	13	15	28
	3	35	2D	1C	11	3D	06	1D	21	1E	09	1F	27	3F	14	2E	38

The paper is organized as follows. Section 2 introduces the fundamental concepts used throughout the paper. Dillon's permutation, its univariate representations with respect to the different primitive polynomials of $\mathbb{F}_{2^6}$, and related analysis are given in Section 3. Section 4 presents the hardware architecture of Dillon's permutation and the description of its different components. Theoretical complexity analysis of the hardware architecture for the different primitive polynomials is given in Section 5. Section 6 gives FPGA implementation results and discussion. Finally, Section 7 concludes the paper with a summary of our observations and some potential directions for future work.

2. Notation and preliminaries

Let $\mathbb{F}_2 = \{0, 1\}$ be the finite field with two elements; we will also refer to it as the *binary field*, and to any extension field $\mathbb{F}_{2^n}$ as a *binary extension field*. Let $f(y) = \sum_{i=0}^{m} f_i y^i$ be a monic irreducible polynomial of degree m over $\mathbb{F}_2$, where $f_i \in \mathbb{F}_2$ for $i = 0, 1, \ldots, m$. All elements of the binary extension field $\mathbb{F}_{2^m}$ can be represented in the *standard basis* $\Omega = \{1, p, \ldots, p^{m-1}\}$, where p is a root of $f(y)$. Any element $x \in \mathbb{F}_{2^m}$ can be represented in Ω as $x = \sum_{i=0}^{m-1} x_i p^i = (1, p, \ldots, p^{m-1}) \cdot (x_0, \ldots, x_{m-1})^T$, with $x_i \in \mathbb{F}_2$, where $(x_0, \ldots, x_{m-1})$ are the coordinates of x with respect to the standard basis.

Vectorial Boolean functions, or (n, m)-functions, are mappings between the vector spaces $\mathbb{F}_2^n$ and $\mathbb{F}_2^m$ for some positive integers n and m [12]. Practically all modern block cyphers include one or more (n, m)-functions as their only nonlinear components, and so these functions are of critical importance in cryptography. Due to the natural identification of the vector space $\mathbb{F}_2^m$ with the binary extension field $\mathbb{F}_{2^m}$, we can also see (n, m)-functions as mappings between the finite fields $\mathbb{F}_{2^n}$ and $\mathbb{F}_{2^m}$. When $n = m$, any (m, m)-function can be uniquely expressed as a polynomial in the form $g(x) = \sum_{i=0}^{2^m-1} a_i x^i$, for $a_i \in \mathbb{F}_{2^m}$, called the *univariate representation* of g. We note that, while any two binary extension fields $\mathbb{F}_{2^m}$ of the same degree m are isomorphic, the polynomial representing an (m, m)-function depends on the concrete binary extension field $\mathbb{F}_{2^m}$, i.e. on the choice of primitive polynomial.

The *algebraic degree* of g is the largest *binary weight* (number of 1's in the binary representation) of an exponent i with $a_i \neq 0$ in the univariate representation. Functions of algebraic degree at most 1 are called *affine*. Functions of algebraic degree 2 and 3 are called *quadratic* and *cubic*, respectively. A *linear* function is an affine function g that satisfies $g(0) = 0$.

Given an (m, m)-function g, the *derivative* of g in direction $a \in \mathbb{F}_{2^m}$ is given by $D_a g(x) = g(x+a)+g(x)$. The *differential uniformity* of g (denoted by Δ_g) is the largest value of $\Delta_g(a, b)$ among all $a \neq 0$ and b from $\mathbb{F}_{2^m}$, where $\Delta_g(a,b)$ is the number of solutions x to the equation $D_a g(x) = b$. The lower the differential uniformity of a given function, the better its security against differential cryptanalysis [1], which is one of the most efficient attacks that can be employed against block cyphers. The lowest possible value that Δ_g can take for any (m,m)-function g is $\Delta_g = 2$, and if $\Delta_g = 2$, then g is called *almost perfect nonlinear (APN)*. For this reason, the study and construction of APN functions are of great importance for the design of secure block cyphers. Examples of APN constructions can be found in [13,14], and as general references we can cite [15,16].

When used in practice for the construction of block cyphers, it is often desirable for APN functions to possess other beneficial properties, such as being a permutation. The latter is a necessary condition for an (m,m)-function to be used as an S-box in a Substitution Permutation Network, or SPN.

3. Dillon's permutation

The existence of APN (m,m)-permutations for even m (known as the "big APN problem") is one of the most important open questions in the area; it was long believed that an APN function on an even number of bits cannot be a permutation. Dillon disproved this in [9] by constructing an APN permutation of $\mathbb{F}_{2^6}$. Dillon's permutation is therefore of great theoretical and practical importance because it is the only known APN permutation for even dimension so far (efforts are being made recently [17] to find more APN permutations in dimension six). Furthermore, it has been used in the design of the lightweight authenticated encryption algorithm FIDES [10]. However, it is currently unknown whether APN permutations exist in even dimensions greater than 6, and so the *Big APN Problem* [18] on the existence of APN permutations of even dimension greater than six remains unsolved.

3.1. Univariate polynomial representations

The univariate polynomial representation of Dillon's APN permutation $g(x)$, with $x \in \mathbb{F}_{2^6}$, generated by the *Magma* computational algebra software [19] is given in Eq. (1), where p is *Magma*'s default primitive element, which is a root of the primitive polynomial $f(y) = y^6 + y^4 + y^3 + y + 1$ over the binary field $\mathbb{F}_2$:

$$\begin{aligned} g(x) = {} & p^{36}x^{60} + p^{44}x^{58} + p^{40}x^{57} + p^{55}x^{56} + \\ & p^{26}x^{54} + p^{23}x^{53} + p^{36}x^{52} + p^{23}x^{51} + p^{17}x^{50} + \\ & p^{54}x^{49} + p^{14}x^{48} + p^{21}x^{46} + p^{53}x^{45} + p^{21}x^{44} + \\ & p^{7}x^{43} + p^{57}x^{42} + p^{8}x^{41} + p^{10}x^{40} + p^{12}x^{39} + \\ & p^{20}x^{38} + p^{52}x^{37} + p^{46}x^{36} + p^{27}x^{35} + p^{44}x^{34} + \\ & p^{18}x^{33} + p^{57}x^{32} + p^{28}x^{30} + p^{44}x^{29} + p^{42}x^{28} + \\ & p^{26}x^{27} + p^{20}x^{26} + p^{10}x^{25} + p^{45}x^{24} + x^{23} + \\ & p^{7}x^{22} + p^{57}x^{21} + p^{21}x^{20} + p^{22}x^{19} + p^{6}x^{17} + \\ & p^{8}x^{16} + p^{43}x^{15} + p^{42}x^{13} + p^{47}x^{12} + p^{56}x^{11} + \\ & p^{38}x^{10} + p^{36}x^{8} + p^{47}x^{7} + p^{4}x^{6} + p^{8}x^{5} + \\ & p^{23}x^{4} + p^{39}x^{3} + p^{52}x^{2} + p^{59}x. \end{aligned} \tag{1}$$

A look-up table of this permutation obtained for this first univariate representation is given in Table 1. Here v and w are given in hexadecimal. For example, if the input x is given in hexadecimal as $x = vw = 2C$ then $g(2C) = 1B$.

We note that the above univariate representation was generated using *Magma*'s default primitive pentanomial $f(y) = y^6 + y^4 + y^3 + y + 1$ with which Dillon's APN permutation is typically represented [18,20]. However, for the binary finite field $\mathbb{F}_{2^6}$ there are five other primitive polynomials (trinomials and pentanomials). They are $f(y) = y^6 + y^5 + y^4 + y + 1$, $f(y) = y^6 + y^5 + y^3 + y^2 + 1$, $f(y) = y^6 + y^5 + 1$, $f(y) = y^6 + y + 1$ and $f(y) = y^6 + y^5 + y^2 + y + 1$ so other univariate representations of Dillon permutation can be given using these primitive polynomials for $\mathbb{F}_{2^6}$.

Table 2
Analysis of polynomial representations.

		$\#x^j p$	$\#p^j$	Monic x^j
$y^6+y^4+y^3+y+1$	P1	52	33	x^{23}
$y^6+y^5+y^4+y+1$	P2	54	38	x^{24}
$y^6+y^5+y^3+y^2+1$	P3	53	38	x^{45}
y^6+y^5+1	T4	56	36	–
y^6+y+1	T5	54	37	–
$y^6+y^5+y^2+y+1$	P6	51	32	$x^3, x^{12}, x^{18}, x^{44}$

The six univariate polynomial representations are given in Appendix A where they have been named $P1$, $P2$, $P3$, $T4$, $T5$ and $P6$ depending on the type of the primitive polynomial (pentanomial or trinomial) used for generating them. The default primitive polynomial of *Magma* is denoted by $P1$.

The procedure for obtaining the univariate representations is the following: we first construct the look-up table in Table 1 by evaluating the univariate polynomial $P1$ representing Dillon's permutation on all inputs $x \in \mathbb{F}_{2^6}$, and converting the finite field elements to binary vectors (representing the coordinates of the finite field elements with respect to the standard basis). We then choose some other primitive polynomial, convert the binary vectors expressed in Table 1 to finite field elements, and use Lagrange interpolation to reconstruct the polynomial representing the function.

3.2. Analysis of polynomial representations

The expressions given in Appendix A consist of terms of the form $p^i x^j$, where p is a root of the primitive polynomial $f(y)$ under consideration. Some of the terms are monic, i.e. of the form x^j, and their presence may affect the efficiency of the hardware implementation since they are easier to evaluate. The exponents j have algebraic up to 4, which is to be expected as Dillon's permutation (and therefore any representation of it) is known to have algebraic degree 4. Nonetheless, not all exponents of algebraic degree 4 (or less) appear with non-zero coefficients. As we shall see, the presence or absence of different terms has a profound impact on the cost of implementing the function in hardware. Another reason for this is that, for a given representation, some coefficients p^i are common to several terms $p^i x^j$ for different values of j, so expressions of the form $p^i(x^{j_1} + x^{j_2} + \cdots) = p^i v_i$ can be obtained, which allows to cut down the complexity of the implementation.

Following Appendix A, Table 2 shows the number of different powers of x that are multiplied by p^j terms (represented by $\#x^j p$ in Table 2), the number of different powers of p (represented by $\#p^j$) and the monic x^j terms for the different univariate representations. Since the algebraic degree is 4, the powers x^{31}, x^{47}, x^{55} and x^{59} (of algebraic degree 5) do not appear in the expressions. Furthermore, the expression $P1$ includes neither x^9, x^{14} nor x^{18}; $P2$ does not include x^{35}; $T5$ does not include x^{48} and $P6$ does not include x^{56}. These differences among the univariate polynomial representations influence the hardware implementation complexity, as we illustrate in Section 4; the effect is quite substantial.

4. Hardware architecture of Dillon's permutation

Following the analysis in Section 3.2, the hardware architecture for Dillon's permutation given by any of the univariate representations in Appendix A is shown in Fig. 1 which shows the different modules needed for the computation. The 6-bit input x is represented in the finite field $\mathbb{F}_{2^6}$ generated by the six different primitive polynomials. The powers of x are computed by the method of square and multiply. In order to do this, the successive squares of x are computed in the *Generator of* x^{2^i} module shown in Fig. 1, the combined products of these squares are then computed in the *Generator of* $x^h = x^{2^i} \cdot x^{2^j}$ module, and the powers x^k (given by the multiplication of some of

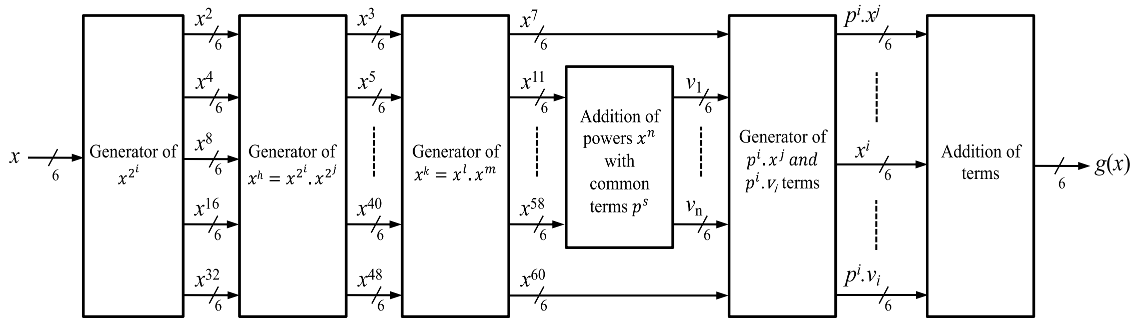


Fig. 1. Hardware architecture of Dillon's permutation.

the previously computed powers) are given by the *Generator of* $x^k = x^l \cdot x^m$ module presented in Fig. 1, where these two last modules use $\mathbb{F}_{2^6}$ multipliers in parallel for the different primitive polynomials. As detailed in Section 3.2, the addition of some powers $v_i = (x^{j_1} + x^{j_2} + \cdots)$, is computed in the *Addition of powers* x^n *with common terms* p^s module. The products $p^i x^j$ and $p^i v_j$ are computed in the *Generator of* $p^i \cdot x^j$ *and* $p^i \cdot v_j$ *terms* module. Finally, the sum (XOR) of x^i, $p^i x^j$ and $p^i v_i$ is computed in the *Addition of terms* module in Fig. 1.

The following subsections describe the implementation of the individual modules in detail.

4.1. Generator of x^{2^i}

This module computes the powers x^2, x^4, x^8, x^{16} and x^{32} modulo the primitive polynomial $f(y)$. These powers are easily computed in the finite field $\mathbb{F}_{2^6}$ due to the fact that x^{2^i}, for $i = 1, 2, \ldots, 5$, is $x^{2^i} = x_5 \cdot p^{5\cdot 2^i} + x_4 \cdot p^{4\cdot 2^i} + x_3 \cdot p^{3\cdot 2^i} + x_2 \cdot p^{2\cdot 2^i} + x_1 \cdot p^{1\cdot 2^i} + x_0 \cdot p^{0\cdot 2^i}$ and the powers of p are reduced using the corresponding primitive polynomial.

For example, for $T5$ with primitive polynomial $f(y) = y^6 + y + 1$, we have $x^8 = x_5 p^{40} + x_4 p^{32} + x_3 p^{24} + x_2 p^{16} + x_1 p^8 + x_0$, where p^{40}, p^{32}, p^{24}, p^{16} and p^8 must be reduced modulo $f(y)$. Since p is a primitive element, we have that $p^6 = p+1$, so $p^{40} = p^5 + p^3 + p^2 + p + 1$, $p^{32} = p^3 + 1$, $p^{24} = p^4 + 1$, $p^{16} = p^4 + p + 1$ and $p^8 = p^3 + p^2$, and substituting this into the expression for x^8 leads us to

$$\begin{aligned} x^8 = (x_5)p^5 + (x_3 + x_2)p^4 + \\ (x_5 + x_4 + x_1)p^3 + (x_5 + x_1)p^2 + \\ (x_5 + x_2)p + (x_5 + x_4 + x_3 + x_2 + x_0). \end{aligned} \tag{2}$$

Therefore the coordinates of the successive powers of x are given as the XORs of the coordinates of x (depending on the reduction modulo $f(y)$). The hardware architecture of x^8 for the primitive trinomial $f(y) = y^6 + y + 1$ is given in Fig. 2.

4.2. Generator of $x^h = x^{2^i} \cdot x^{2^j}$

This module computes the powers x^3, x^5, x^6, x^9, x^{10}, x^{12}, x^{17}, x^{18}, x^{20}, x^{24}, x^{33}, x^{34}, x^{36}, x^{40} and x^{48} by means of the parallel multiplication of some of the terms x, x^2, x^4, x^8, x^{16} and x^{32}. For example, $x^{36} = x^4 \cdot x^{32}$, where this product is implemented with a multiplier over $\mathbb{F}_{2^6}$ using the corresponding primitive polynomial. Multipliers selected for the implementation are described in Section 4.7.

4.3. Generator of $x^k = x^l \cdot x^m$

The remaining powers x^k (up to the maximum value x^{60} in all primitive polynomials except $T5$) are computed by the parallel multiplication of some of the terms given by the above module (including the input x). For example, $x^{53} = x^{20} \cdot x^{33}$, where x^{20} and x^{33} are generated in the previous module. The product is also implemented using a multiplier over $\mathbb{F}_{2^6}$ for the corresponding primitive polynomial. Multipliers selected for the implementation are described in Section 4.7.

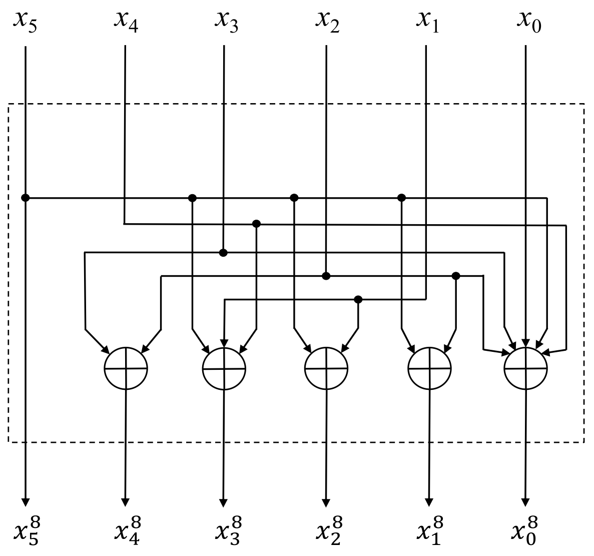


Fig. 2. Hardware architecture of x^8 for primitive trinomial $f(y) = y^6 + y + 1$.

4.4. Addition of powers x^n with common terms p^s

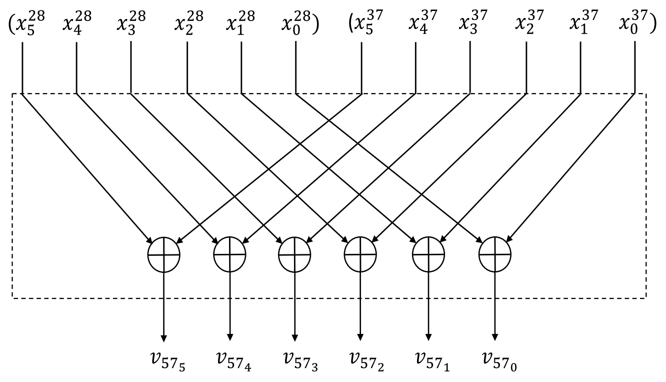
As discussed in Section 3.2, some terms p^i are common to several products $p^i x^j$ for different values of j, so expressions of the form $p^i(x^{j_1} + x^{j_2} + \cdots) = p^i v_i$ can be obtained. Therefore, the sums $v_i = (x^{j_1} + x^{j_2} + \cdots)$ are computed in this module.

Appendix B shows the additions of powers x^n with common terms p^s for the different primitive polynomials. For example, for $T5$ the following expressions can be found in the univariate representation (see Appendix B): $p^7(x^{26}+x^{39}) = p^7 v_7$, $p^{21}(x^6+x^{17}+x^{44})$, $p^{24}(x^4+x^{12})$, $p^{26}(x^{27}+x^{40})$, $p^{28}(x^{11}+x^{15})$, $p^{32}(x^8+x^{41})$, $p^{34}(x^2+x^7+x^{52})$, $p^{37}(x^{14}+x^{32}+x^{58})$, $p^{39}(x^{16}+x^{49})$, $p^{42}(x^{42}+x^{51})$, $p^{43}(x^{35}+x^{54})$, $p^{45}(x^{21}+x^{50})$, $p^{49}(x^{24}+x^{56})$ and $p^{57}(x^{28}+x^{37}) = p^{57} v_{57}$. The use of these expressions allows us to reduce the complexity of the implementation. The sum of such powers x^n is simply performed as the bitwise XOR of the corresponding coordinates. The architecture of the addition of powers $v_{57} = (x^{28} + x^{37})$ for the primitive trinomial $T5$ is given in Fig. 3.

4.5. Generator of $p^i \cdot x^j$ and $p^i \cdot v_i$ terms

The univariate representations given in Appendix A involve the addition of terms $p^i \cdot x^j$ and $p^i \cdot v_i$, where $v_i = (x^{j_1} + x^{j_2} + \cdots)$ as given in previous subsection, and $x^j, v_j \in \mathbb{F}_{2^6}$. In general, the term $p^i \cdot A$, with $A \in \mathbb{F}_{2^6}$, will be $p^i \cdot A = p^i \cdot (a_5 p^5 + a_4 p^4 + a_3 p^3 + a_2 p^2 + a_1 p + a_0) = (a_5 p^{5+i} + a_4 p^{4+i} + a_3 p^{3+i} + a_2 p^{2+i} + a_1 p^{1+i} + a_0 p^i)$, where the powers of p are reduced using the corresponding primitive polynomial.

For example, for $P1$ with primitive polynomial $f(y) = y^6 + y^4 + y^3 + y + 1$, we have the term $p^6 \cdot A = (a_5 p^{11} + a_4 p^{10} + a_3 p^9 + a_2 p^8 + a_1 p^7 + a_0 p^6)$ where



$(x_5^{28}\ x_4^{28}\ x_3^{28}\ x_2^{28}\ x_1^{28}\ x_0^{28})$
$(x_5^{37}\ x_4^{37}\ x_3^{37}\ x_2^{37}\ x_1^{37}\ x_0^{37})$
v_{57_5}
v_{57_4}
v_{57_3}
v_{57_2}
v_{57_1}
v_{57_0}

Fig. 3. Addition of powers $v_{57} = (x^{28} + x^{37})$ for primitive trinomial $T5$.

p^{11}, p^{10}, p^9, p^8, p^7 and p^6 must be reduced modulo $f(y)$. Since p is a primitive element, we have that $p^6 = p^4+p^3+p+1$, so $p^{11} = p^5+p^4+p^3+1$, $p^{10} = p^5 + p^4 + 1$, $p^9 = p^5 + p^4 + p^2 + 1$, $p^8 = p^5 + p^4 + p^2 + p + 1$, $p^7 = p^5 + p^4 + p^2 + p$, and $p^6 = p^4 + p^3 + p + 1$, and substituting this into the expression for $p^6 \cdot A$ gives Eq. (3):

$$\begin{aligned} p^6 \cdot A = (a_5 + a_4 + a_3 + a_2 + a_1)p^5+ \\ (a_5 + a_4 + a_3 + a_2 + a_1 + a_0)p^4+ \\ (a_5 + a_0)p^3 + (a_3 + a_2 + a_1)p^2+ \\ (a_2 + a_1 + a_0)p + (a_5 + a_4 + a_3 + a_2 + a_0). \end{aligned} \tag{3}$$

Therefore, the coordinates of $p^i \cdot A$ are given as the XORs of the coordinates of A (depending on the reduction modulo the primitive polynomial $f(y)$). The hardware architecture of $p^6 \cdot A$ for the primitive pentanomial $f(y) = y^6 + y^4 + y^3 + y + 1$ is given in Fig. 4.

4.6. Addition of x^i, $p^i \cdot x^j$ and $p^i \cdot v_i$ terms

The final addition of the terms x^i, $p^i \cdot x^j$ and $p^i \cdot v_i$ appearing in the univariate expressions for Dillon's permutation given in Appendix A is simply performed as the bitwise XOR of the corresponding coordinates.

4.7. Multipliers over $\mathbb{F}_{2^6}$

Multipliers over the finite field $\mathbb{F}_{2^6}$ for the different primitive polynomials (trinomials and pentanomials) are needed in the second and third modules given in Sections 4.2 and 4.3, respectively, to compute terms of the form $x^{2^i} \cdot x^{2^j}$ and $x^l \cdot x^m$.

Several $GF(2^m)$ multipliers have been proposed in the literature [21–24]. In this work, we have used the method presented in [23,24] for the construction of $\mathbb{F}_{2^m}$ multipliers for primitive trinomials and pentanomials, respectively, which exhibit lowest delay with a balanced area×time complexity compared to other similar approaches. In [23,24], in order to compute the product $C = A \cdot B$, with $A, B \in \mathbb{F}_{2^6}$, the functions $\mathbf{S_i}$ $(1 \le i \le m)$ and $\mathbf{T_i}$ $(0 \le i \le m-2)$ given by the addition of terms $x_k = (a_k b_k)$ and $z_i^j = (a_i b_j + a_j b_i)$ were defined, with $a_i, b_i \in \mathbb{F}_2$ being the coordinates of A and B, respectively. These functions were given in [23] as $\mathbf{S_i} = x_p + \sum_{h=0}^{p-1} z_h^{i-h-1}$ and $\mathbf{T_i} = x_q + \sum_{j=1}^{r-(i+1)} z_{i+j}^{m-j}$, where $p = \lfloor i/2 \rfloor$, $q = (\lceil m/2 \rceil + \lfloor i/2 \rfloor)$, the term $x_p = a_p b_p$ only appears for i *odd* and x_q only appears for m and i *even* or for m and i *odd*, where $r = q$. Otherwise, i.e., for m *odd* and i *even* or for m *even* and i *odd*, the term x_q does not exist and $r = (\lceil m/2 \rceil + \lceil i/2 \rceil)$. For example, using the above expressions, the terms $\mathbf{S_i}$ and $\mathbf{T_i}$ for $\mathbb{F}_{2^6}$ are: $\mathbf{S_1} = x_0 = a_0 b_0$, $\mathbf{S_2} = z_0^1 = (a_0 b_1 + a_1 b_0)$, $\mathbf{S_3} = x_1 + z_0^2 = a_1 b_1 + (a_0 b_2 + a_2 b_0)$, $\mathbf{S_4} = z_0^3 + z_1^2 = (a_0 b_3 + a_3 b_0) + (a_1 b_2 + a_2 b_1)$, $\mathbf{S_5} = x_2 + z_0^4 + z_1^3 = a_2 b_2 + (a_0 b_4 + a_4 b_0) + (a_1 b_3 + a_3 b_1)$, $\mathbf{S_6} = z_0^5 + z_1^4 + z_2^3 = (a_0 b_5 + a_5 b_0) + (a_1 b_4 + a_4 b_1) + (a_2 b_3 + a_3 b_2)$, $\mathbf{T_0} = x_3 + z_1^5 + z_2^4 = a_3 b_3 + (a_1 b_5 + a_5 b_1) + (a_2 b_4 + a_4 b_2)$, $\mathbf{T_1} = z_2^5 + z_3^4 = (a_2 b_5 + a_5 b_2) + (a_3 b_4 + a_4 b_3)$, $\mathbf{T_2} = x_4 + z_3^5 = a_4 b_4 + (a_3 b_5 + a_5 b_3)$,

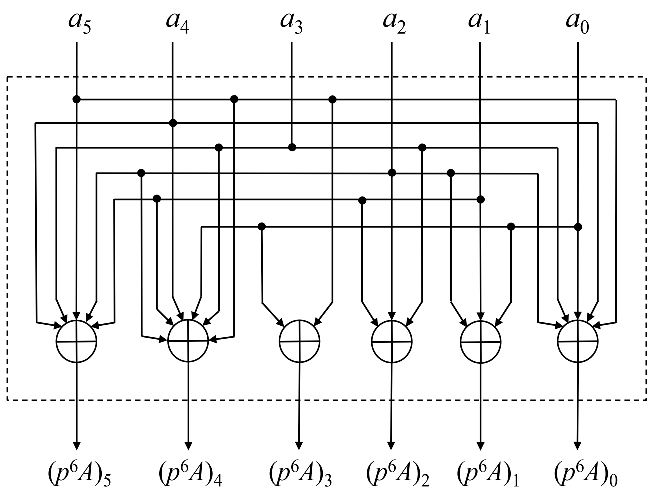


Fig. 4. Hardware architecture of $p^6 \cdot A$ for $f(y) = y^6 + y^4 + y^3 + y + 1$.

$\mathbf{T_3} = z_4^5 = (a_4 b_5 + a_5 b_4)$, $\mathbf{T_4} = x_5 = a_5 b_5$. The coordinates of the product $C = A \cdot B$ can be computed as the sum of some of these terms.

The specific coordinate expressions of the product $C = A \cdot B$ over $\mathbb{F}_{2^6}$, given in terms of the $\mathbf{S_i}$ and $\mathbf{T_i}$ functions, for the different primitive polynomials are given in Appendix C.

5. Theoretical complexity analysis

Area and time theoretical complexities of the different permutation architectures can be obtained from the complexities of the previously described modules. Area complexity corresponds to the number of 2-input AND and XOR gates. Time complexity is determined by the maximum number of 2-input AND and XOR gates that a signal must traverse from input to output, and is given in terms of T_A and T_X (gate delay of a 2-input AND and 2-input XOR gate, respectively).

Theoretical complexities of the permutation architectures can be calculated by first determining the complexities of the $\mathbb{F}_{2^m}$ multipliers. From the expressions of $\mathbf{S_i}$ and $\mathbf{T_i}$ terms given in Section 4.7 and the coordinate expressions of the product given in Appendix C, it can be observed that the number of 2-input AND gates is given by m^2, so in this case the number of AND gates needed is 36 for each multiplier. Furthermore, the number of 2-input XOR gates is 45, 44, 40, 45, 35 and 42 for the $P1$, $P2$, $P3$, $T4$, $T5$ and $P6$ multipliers, respectively (where the expressions including XORs with multiple inputs have been converted to equivalent 2-input XOR gates). Finally, it can be observed that the maximum theoretical delay of the multipliers is given by $T_A + 5T_X$ for $P1$, $P2$, $P3$, $T4$, $P6$ and $T_A + 4T_X$ for $T5$.

5.1. Area complexity

From the previous description of the different modules of the architecture, it can be observed that all the modules (except the second and third ones involving multiplication in $\mathbb{F}_{2^6}$) are given as the sum (XORs) of the coordinates of their inputs. Therefore, the number of 2-input AND gates is given by these two modules that compute the terms $x^{2^i} \cdot x^{2^j}$ and $x^l \cdot x^m$. The module that generates the terms of the form $x^{2^i} \cdot x^{2^j}$ computes the powers x^3, x^5, x^6, x^9, x^{10}, x^{12}, x^{17}, x^{18}, x^{20}, x^{24}, x^{33}, x^{34}, x^{36}, x^{40} and x^{48} by means of the multiplication of some of the terms x, x^2, x^4, x^8, x^{16} and x^{32}. This module is the same for all primitive polynomials (that need 15 multipliers) except for $T5$ (which does not include x^{48}, and therefore needs 14 multipliers). The module generating the terms $x^k = x^l \cdot x^m$ (up to the maximum value x^{60} in all primitive polynomials except $T5$) are computed by the product of some of the terms given by the previous module (including the input x). The number of multipliers thus depend on the specific univariate

Table 3
Theoretical complexities for different primitive polynomials.

		#AND	#XOR	Delay
$y^6+y^4+y^3+y+1$	P1	1764	3003	$2T_A+24T_X$
$y^6+y^5+y^4+y+1$	P2	1764	3000	$2T_A+24T_X$
$y^6+y^5+y^3+y^2+1$	P3	**1728**	2757	$2T_A+24T_X$
y^6+y^5+1	T4	1800	3044	$2T_A+23T_X$
y^6+y+1	T5	**1728**	**2484**	$2T_A+\mathbf{22}T_X$
$y^6+y^5+y^2+y+1$	P6	1764	2832	$2T_A+23T_X$

Table 4
Theoretical complexities for different primitive polynomials using subexpressions sharing for multiplication.

		#AND	#XOR	Delay
$y^6+y^4+y^3+y+1$	P1	1764	2709	$2T_A+24T_X$
$y^6+y^5+y^4+y+1$	P2	1764	2804	$2T_A+24T_X$
$y^6+y^5+y^3+y^2+1$	P3	**1728**	2661	$2T_A+24T_X$
y^6+y^5+1	T4	1800	2544	$2T_A+25T_X$
y^6+y+1	T5	**1728**	**2484**	$2T_A+\mathbf{22}T_X$
$y^6+y^5+y^2+y+1$	P6	1764	2685	$2T_A+25T_X$

expression for the selected primitive polynomial. For the above two modules, the overall number of multipliers needed for $P1$, $P2$, $P3$, $T4$, $T5$ and $P6$ are 49, 49, 48, 50, 48 and 49, respectively, where each multiplier needs 36 AND gates. Table 3 shows the theoretical number of 2-input AND gates needed for each univariate expression using the different primitive polynomials.

With respect to the number of 2-input XOR gates for each architecture, it can be observed that the first module generator of x^{2^i} only involves XOR gates for the implementation whose number can be deduced from the corresponding expressions for each polynomial. For example, the number of 2-input XOR gates needed for the computation of x^8 for the polynomial $T5$ given in Fig. 2 is 9 XORs. The total number of 2-input XOR gates needed in this first module for $P1$, $P2$, $P3$, $T4$, $T5$ and $P6$ are 51, 45, 30, 47, 29 and 51, respectively. As previously described, the second and third modules generating the terms $x^{2^i}\cdot x^{2^j}$ and $x^l\cdot x^m$, respectively, need a total number of 49, 49, 48, 50, 48 and 49 multipliers for $P1$, $P2$, $P3$, $T4$, $T5$ and $P6$, respectively. Using the complexities of the $\mathbb{F}_{2^6}$ multipliers previously given, we have that the total number of XOR gates needed for these two modules for $P1$, $P2$, $P3$, $T4$, $T5$ and $P6$ are 2205, 2156, 1920, 2250, 1680 and 2058, respectively. Next, it can be observed that the module performing the addition of terms of the form x^n common to an element p^s needs 114, 90, 90, 120, 102 and 114 XOR gates for $P1$, $P2$, $P3$, $T4$, $T5$ and $P6$, respectively. The computation of the 2-input XOR gates for the generator module of the terms $p^i\cdot x^j$ and $p^i\cdot v_i$ gives a total number of 435, 481, 489, 417, 457 and 399 XOR gates for $P1$, $P2$, $P3$, $T4$, $T5$ and $P6$, respectively. Finally, the last module performing the addition of the terms x^i, $p^i\cdot x^j$ and $p^i\cdot v_i$ appearing in the univariate expressions of Dillon's permutation for $P1$, $P2$, $P3$, $T4$, $T5$ and $P6$ needs a total number of 198, 228, 228, 210, 216 and 210 XOR gates, respectively. Table 3 shows the total theoretical number of 2-input XOR gates needed for each univariate expression using the different primitive polynomials.

From Table 3, we can see that the lowest number of 2-input AND gates corresponds to the expressions given by $P3$ and $T5$, while the lowest number of 2-input XOR gates corresponds to the univariate expression given by $T5$. Furthermore, it is important to note that trinomial $T4$ exhibits the highest number of AND and XOR gates among the different primitive polynomials.

5.2. Time complexity

From the previous descriptions of the modules, it can be observed that the first module generator of x^{2^i} has a maximum delay of $3T_X$ for polynomials $P1$, $P2$, $P3$, $T5$ and a delay of $2T_X$ for polynomials $T4$ and $P6$. The delay of the modules generating the terms $x^{2^i}\cdot x^{2^j}$ and $x^l\cdot x^m$ is given by the delay of the $\mathbb{F}_{2^6}$ multiplier (T_A+5T_X for $P1$, $P2$, $P3$, $T4$, $P6$ and T_A+4T_X for $T5$). The next module performing the addition of the terms of the form x^n common to an element p^s presents a maximum delay of $2T_X$ for all primitive polynomials. The generator module of the terms $p^i\cdot x^j$ and $p^i\cdot v_i$ also needs a maximum delay of $3T_X$ in all cases. Finally, the last module performing the addition of terms x^i, $p^i\cdot x^j$ and $p^i\cdot v_i$ has a maximum delay of $6T_X$ for all primitive polynomials. Table 3 shows the theoretical delay for each univariate expression using the different primitive polynomials, where it can be observed that the lowest delay corresponds to the univariate expression given by primitive trinomial $T5$ and the highest delay is obtained by $P1$, $P2$ and $P3$ polynomials.

5.3. Subexpressions sharing

As previously shown, the theoretical complexity of the hardware architectures is highly dependent on the complexity of the multiplier, so the expressions in Appendix C for $\mathbb{F}_{2^6}$ multipliers could be further refined in order to obtain a reduction of the area complexity. It can be observed that for the different primitive polynomials, some sums of terms of the form $\mathbf{T_i}$ can be shared among the different product coordinates. For example, for the primitive pentanomial $P1$, the sum $(\mathbf{T_1}+\mathbf{T_2})$ can be found in the product coordinates c_1, c_2, c_4 and c_5, while the sum $(\mathbf{T_3}+\mathbf{T_4})$ is found in the coordinates c_0, c_4 and c_5. Therefore, *subexpressions sharing* can be used to reduce the theoretical area complexity of the implementations. The new expressions for the coordinates of the $\mathbb{F}_{2^6}$ multipliers using intermediate variables for subexpressions sharing are given in Appendix C, where we note that for the primitive trinomial $T5$ no sharing can be performed.

From the new expressions given in Section 4.7 for $\mathbb{F}_{2^6}$ multipliers, it can be observed that the number of 2-input AND gates is m^2, so the number of AND gates is 36 for each multiplier. Furthermore, the number of 2-input XOR gates is 39, 40, 38, 35 and 39 for the $P1$, $P2$, $P3$, $T4$ and $P6$ multipliers, respectively ($T5$ does not apply *subexpressions sharing*). Finally, we note that the use of subexpressions sharing only modifies the theoretical delays for the multipliers using the polynomials $T4$ and $P6$. In these two cases, the theoretical delay is T_A+6T_X, whereas the delay without sharing is T_A+5T_X.

The application of these complexities of $\mathbb{F}_{2^6}$ multipliers to the different permutation architectures gives the new theoretical complexities shown in Table 4, where a reduced number of XOR gates is obtained for all polynomials (except for $T5$). However, the delay complexity for $T4$ and $P6$ increases by a factor of $2T_X$ with respect to the delay given in Table 3. In conclusion, the architecture given by $T5$ presents the lowest area and time complexities, while that the highest number of AND gates corresponds to $T4$ and the highest number of XOR gates corresponds to the $P2$ pentanomial.

6. FPGA implementations

In order to further compare the different univariate expressions of Dillon's permutation given in Appendix A, we have developed FPGA implementations of the hardware architecture given in Section 4. The architectures have been described in VHDL (Very High Speed Integrated Circuit Hardware Description Language), synthesized and implemented on Xilinx FPGA Artix-7 XC7A12T-3-CPG238 using VIVADO 2021.2. Experimental post-place and route results using the expressions for the $\mathbb{F}_{2^6}$ multipliers shown in Appendix C are given in Table 5. We note that each of the six implementations fits in 6 LUTs (Lookup Tables), so in order to show the differences between the different implementations, we have included the optimized number of 2-input XOR gates given by the synthesizer in Table 5. The synthesizer supplies the total number of XOR gates inferred by the tool, including XORs with multiple inputs and with several bits each. These XORs have been converted to equivalent 2-input XOR gates in order to have a fair comparison of the different implementations. We also note that the synthesis tool does not provide the number of AND gates used in the design.

Table 5
Experimental complexities for different primitive polynomials.

		#XOR	%	Delay (ns)	%	#XOR ×Delay	%
$y^6+y^4+y^3+y+1$	P1	2860	98.4	5.542	98.1	15 850.12	96.9
$y^6+y^5+y^4+y+1$	P2	2865	98.6	5.633	99.7	16 138.55	98.7
$y^6+y^5+y^3+y^2+1$	P3	2603	89.6	5.625	99.5	14 641.88	89.6
y^6+y^5+1	T4	2906	100.0	5.626	99.6	16 349.16	100.0
y^6+y+1	T5	**2356**	**81.1**	**5.536**	**98.0**	**13042.82**	**79.8**
$y^6+y^5+y^2+y+1$	P6	2707	93.2	5.651	100.0	15 297.26	93.6

Table 6
Experimental complexities for different primitive polynomials using subexpressions sharing.

		#XOR	%	Delay (ns)	%	#XOR ×Delay	%
$y^6+y^4+y^3+y+1$	P1	2566	96.1	5.631	100.0	14 449.15	96.1
$y^6+y^5+y^4+y+1$	P2	2669	100.0	5.631	100.0	15 029.14	100.0
$y^6+y^5+y^3+y^2+1$	P3	2507	93.9	5.625	99.9	14 101.87	93.8
y^6+y^5+1	T4	2406	90.1	5.625	99.9	13 533.75	90.0
y^6+y+1	T5	**2356**	**88.3**	**5.536**	**98.3**	**13042.82**	**86.8**
$y^6+y^5+y^2+y+1$	P6	2560	95.9	5.542	98.4	14 187.52	94.4

In Table 5, #XOR represents the optimized number of 2-input XOR gates given by the synthesizer, $Delay$ corresponds to the delay (in nanoseconds) needed by each architecture to compute the output of Dillon's permutation, and $\#XOR \times Delay$ is the product of the number of 2-input XOR gates and the delay (less is better). The columns % represents the percentages for the results (with respect to the highest values) corresponding to the #XOR, $Delay$ and $\#XOR \times Delay$ metrics. From Table 5, we can see that the implementation of the permutation architecture for the primitive trinomial $f(y) = y^6+y+1$ presents the best results in the number of 2-input XORs (with a reduction of up to 18.9% with respect to the implementation of $T4$), delay (with a reduction of 2% with respect of $P6$) and in $\#XOR \times Delay$ (with a reduction of 20.2% with respect to $T4$). The implementation results given in Table 5 also show that the use of *Magma*'s default primitive polynomial $f(y) = y^6+y^4+y^3+y+1$ ($P1$) does not provide the best results from a point of view of a hardware implementation. Furthermore, the worst results obtained for implementation ($\#XOR \times Delay$) of Dillon's permutation expressions correspond to the primitive trinomial $f(y) = y^6+y^5+1$ ($T4$).

We obtained the above experimental results using the coordinates given in Appendix C for the $\mathbb{F}_{2^6}$ multipliers. However, the complexity of the hardware implementation is highly dependent on the complexity of the multiplier, so the expressions given in Appendix C for the multipliers using *subexpressions sharing* can be used to reduce the area complexity of the implementations. Table 6 shows the experimental post-place and route results obtained using subexpressions sharing for the $\mathbb{F}_{2^6}$ multipliers. As in the previous case, each of the six implementations fits in 6 LUTs, so the number of 2-input XOR gates given by the synthesizer is included in Table 6. It can be observed that the implementation of the permutation architecture for the primitive trinomial $f(y) = y^6+y+1$ still presents the best results in the number of 2-input XORs (with a reduction of up to 11.7% with respect to the implementation of $P2$), delay (with a reduction of 1.7% with respect to $P1$ and $P2$) and in $\#XOR \times Delay$ (with a reduction of 13.2% with respect to $P2$). The experimental results given in Table 6 still show that the use of the default primitive polynomial $f(y) = y^6+y^4+y^3+y+1$ ($P1$) does not provide the best results from the point of view of a hardware implementation. In this case, the worst results obtained in terms of the ($\#XOR \times Delay$) metric correspond to the use of the primitive pentanomial $f(y) = y^6+y^5+y^4+y+1$ ($P2$).

We can observe that there are differences between the theoretical complexities given in Table 3 and Table 4 and the corresponding FPGA experimental complexities given in Table 5 and Table 6, respectively. These differences are due to the optimizations performed by the synthesizer and the mapping to the CLBs (Configurable Logic Blocks) of the FPGA. In both the theoretical and experimental complexities, the permutation architecture for the primitive trinomial $f(y) = y^6+y+1$ presents the best results.

Table 7
Experimental complexities for different primitive polynomials using different multipliers.

Polynomial		#XOR	Delay (ns)	#XOR ×Delay
P1	[21]	2719	5.647	15 354.19
	[22]	2615	**5.626**	14 711.99
	[24]	**2566**	5.631	**14449.15**
P2	[21]	2718	**5.617**	15 267.01
	[22]	2718	5.632	15 307.78
	[24]	**2669**	5.631	**15029.14**
P3	[21]	2555	**5.623**	14 366.76
	[22]	**2507**	5.633	14 121.93
	[24]	**2507**	5.625	**14101.87**
T4	[21]	2756	5.542	15 273.75
	[22]	2556	5.645	14 428.62
	[23]	**2406**	**5.625**	**13533.75**
T5	[21]	**2356**	5.632	13 268.99
	[22]	**2356**	5.645	13 299.62
	[23]	**2356**	**5.536**	**13042.82**
P6	[21]	2609	5.617	14 654.75
	[22]	**2560**	5.632	14 417.92
	[24]	**2560**	**5.542**	**14187.52**

As previously mentioned, the complexity of the hardware implementation is highly dependent on the complexity of the multiplier. Table 7 shows the comparison of the experimental results obtained for different primitive polynomials using different $\mathbb{F}_{2^6}$ multipliers. In Table 7, the results obtained in Table 6 for the multipliers given in [23,24] using subexpressions sharing are compared with the results obtained using the two-step classic $\mathbb{F}_{2^m}$ multiplication method (polynomial multiplication and reduction modulo an irreducible polynomial) given in [21] and the efficient matrix–vector multiplication method, also using subexpression sharing, given in [22]. It can be observed in Table 7 that the results obtained by [23,24] for trinomials and pentanomials, respectively, present the lowest number of XOR gates, the lowest delay for three of the six polynomials, and the lowest $\#XOR \times Delay$ values in all cases. Furthermore, Table 7 also shows that the implementation of the permutation architecture for the primitive trinomial $f(y) = y^6+y+1$ presents the best results in the number of 2-input XORs, delay and in $\#XOR \times Delay$.

From the above results, we can conclude that the choice of the primitive polynomial can have a huge impact on the area and delay of FPGA implementations. Dillon's permutation works over $\mathbb{F}_{2^6}$ and it is small enough to fit in 6 LUTs, but for APN permutations working on larger dimensions the selection of the primitive polynomial can profoundly impact the efficiency of the FPGA implementation.

In order to further compare the different univariate expressions, Table 8 gives the area (transistor count) and delay estimates for the different polynomials, where some STMicroelectronics real circuits have

Table 8
Area-delay estimate for the different primitive polynomials.

		Trans.	Delay (ns)	A×D
$y^6+y^4+y^3+y+1$	P1	25980	300	7794.0
$y^6+y^5+y^4+y+1$	P2	26598	300	7979.4
$y^6+y^5+y^3+y^2+1$	P3	25410	300	7623.0
y^6+y^5+1	T4	25236	312	7873.6
y^6+y+1	T5	**24504**	**276**	**6763.1**
$y^6+y^5+y^2+y+1$	P6	25944	312	8094.5

been used. They are high-speed CMOS (Complementary Metal Oxide Semiconductor) gates fabricated by silicon gate C^{2}MOS (Clocked CMOS) technology, with balanced propagation delays, high speed and low power dissipation. The typical propagation delay t_{PD} has been considered to ensure a fair comparison. The circuits used are M74HC08 (AND gate with $t_{PD} = 6$ ns) and M74HC86 (XOR gate with $t_{PD} = 12$ ns). For the estimate of the number of CMOS transistors used in the designs, we used the traditional counts (6 transistors for 2-input AND gate and 6 for 2-input XOR gate). In Table 8, the theoretical number of AND gates and the theoretical delay using subexpressions sharing (given in Table 4), and the experimental number of XOR gates given in Table 6 for subexpressions sharing have been used, where the $Trans.$ column represents the number of transistors and the $A\times D$ column ($Area\times Delay$) is given in *transistors* × *microseconds*. From Table 8, we can observe that the estimation of the permutation architecture for the primitive trinomial $f(y) = y^6+y+1$ presents again the best results in the number of transistors (with a reduction of up to 7.9% with respect to $P2$), delay (with a reduction of 11.5% with respect to $T4$ and $P6$) and in $Area \times Delay$ (with a reduction of 16.4% with respect to $P6$).

7. Conclusion

Modern block cyphers typically use vectorial Boolean functions (*S-boxes* or *substitution boxes*) as their only nonlinear components, and so the security of the encryption depends on the properties of the *S-box*. APN functions are optimal with respect to their resistance to differential cryptanalysis. Frequently, it is necessary or desirable for an S-box to be bijective, but to date very few APN permutations are known. Dillon's permutation in dimension 6 is the only known APN permutation in an even dimension that was originally generated with the primitive pentanomial $f(y) = y^6+y^4+y^3+y+1$ for the binary finite field $\mathbb{F}_{2^6}$, and so is a good candidate for studying how the choice of the primitive polynomial affects the complexity of the hardware implementation. In this paper, we have presented a hardware architecture for Dillon's permutation and implemented it in FPGA for the six different primitive polynomials that can be used to construct $\mathbb{F}_{2^6}$. We have also given the corresponding univariate representations and hardware theoretical complexities for the implementations corresponding to the different primitive polynomials. To the best of our knowledge, no similar study has been given in the open literature. From the FPGA experimental results, the implementation of Dillon permutation using the primitive trinomial $f(y) = y^6 + y + 1$ presents reductions in the number of 2-input XOR gates of up to 11.7% and in the #XOR × Delay metrics of up to 13.2% with respect to the implementation with the primitive polynomial using the highest number of XORs. Therefore the use of the default primitive polynomial $f(y) = y^6+y^4+y^3+y+1$ does not provide the best results from a point of view of a hardware implementation. More generally, we see that the choice of primitive polynomial can have a huge impact on the efficiency of the implementation.

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.

Acknowledgments

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

The work of J.L. Imaña was supported in part by grant PID2021-123041OB-I00 funded by MCIN/AEI/10.13039/501100011033 and by "ERDF A way of making Europe", and by the Comunidad de Madrid, Spain under grant S2018/TCS-4423. The work of L. Budaghyan was supported by Trond Mohn Foundation under Grant "Construction of Optimal Boolean Functions" and by the Research Council of Norway under Grant 314395.

Appendix A. Univariate expressions for dillon permutation

The univariate expressions for Dillon permutation using different primitive polynomials are as follows, where p is a primitive element.

A.1. *Primitive pentanomial P1:* $f(y) = y^6 + y^4 + y^3 + y + 1$

$$g(x) = p^{36}x^{60} + p^{44}x^{58} + p^{40}x^{57} + p^{55}x^{56} + p^{26}x^{54} + p^{23}x^{53} + p^{36}x^{52} + p^{23}x^{51} + p^{17}x^{50} + p^{54}x^{49} + p^{14}x^{48} + p^{21}x^{46} + p^{53}x^{45} + p^{21}x^{44} + p^{7}x^{43} + p^{57}x^{42} + p^{8}x^{41} + p^{10}x^{40} + p^{12}x^{39} + p^{20}x^{38} + p^{52}x^{37} + p^{46}x^{36} + p^{27}x^{35} + p^{44}x^{34} + p^{18}x^{33} + p^{57}x^{32} + p^{28}x^{30} + p^{44}x^{29} + p^{42}x^{28} + p^{26}x^{27} + p^{20}x^{26} + p^{10}x^{25} + p^{45}x^{24} + x^{23} + p^{7}x^{22} + p^{57}x^{21} + p^{21}x^{20} + p^{22}x^{19} + p^{6}x^{17} + p^{8}x^{16} + p^{43}x^{15} + p^{42}x^{13} + p^{47}x^{12} + p^{56}x^{11} + p^{38}x^{10} + p^{36}x^{8} + p^{47}x^{7} + p^{4}x^{6} + p^{8}x^{5} + p^{23}x^{4} + p^{39}x^{3} + p^{52}x^{2} + p^{59}x.$$

A.2. *Primitive pentanomial P2:* $f(y) = y^6 + y^5 + y^4 + y + 1$

$$g(x) = p^{11}x^{60} + p^{24}x^{58} + p^{40}x^{57} + p^{35}x^{56} + p^{47}x^{54} + p^{42}x^{53} + p^{9}x^{52} + p^{34}x^{51} + p^{50}x^{50} + p^{7}x^{49} + p^{25}x^{48} + p^{22}x^{46} + p^{37}x^{45} + p^{7}x^{44} + p^{26}x^{43} + p^{38}x^{42} + p^{31}x^{41} + p^{48}x^{40} + p^{39}x^{39} + p^{11}x^{38} + p^{21}x^{37} + p^{9}x^{36} + p^{45}x^{34} + p^{30}x^{33} + p^{37}x^{32} + p^{32}x^{30} + p^{26}x^{29} + p^{23}x^{28} + p^{45}x^{27} + p^{2}x^{26} + p^{38}x^{25} + x^{24} + p^{38}x^{23} + p^{40}x^{22} + p^{58}x^{21} + p^{38}x^{20} + p^{18}x^{19} + p^{49}x^{18} + p^{60}x^{17} + p^{32}x^{16} + p^{16}x^{15} + p^{40}x^{14} + p^{22}x^{13} + p^{29}x^{12} + p^{10}x^{11} + p^{12}x^{10} + p^{4}x^{9} + p^{19}x^{8} + p^{59}x^{7} + p^{58}x^{6} + p^{28}x^{5} + p^{32}x^{4} + p^{53}x^{3} + p^{17}x^{2} + p^{30}x.$$

A.3. *Primitive pentanomial P3:* $f(y) = y^6 + y^5 + y^3 + y^2 + 1$

$$g(x) = p^{53}x^{60} + p^{12}x^{58} + p^{61}x^{57} + p^{10}x^{56} + p^{41}x^{54} + p^{60}x^{53} + p^{12}x^{52} + p^{44}x^{51} + p^{36}x^{50} + p^{19}x^{49} + p^{36}x^{48} + p^{33}x^{46} + x^{45} + p^{22}x^{44} + p^{56}x^{43} + p^{38}x^{42} + p^{16}x^{41} + p^{48}x^{40} + p^{28}x^{39} + p^{22}x^{38} + p^{2}x^{37} + p^{22}x^{36} + p^{13}x^{35} + p^{58}x^{34} + p^{29}x^{33} + p^{26}x^{32} + p^{60}x^{30} + p^{27}x^{29} + p^{31}x^{28} + p^{20}x^{27} + p^{31}x^{26} + p^{54}x^{25} + p^{44}x^{24} + p^{55}x^{23} + p^{42}x^{21} + p^{51}x^{20} + p^{30}x^{19} + p^{8}x^{18} + p^{38}x^{17} + p^{31}x^{16} + p^{32}x^{14} + p^{11}x^{13} + p^{38}x^{12} + p^{22}x^{11} + p^{45}x^{10} + p^{16}x^{9} + p^{17}x^{8} + p^{21}x^{7} + p^{50}x^{6} + p^{18}x^{5} + p^{21}x^{4} + p^{27}x^{3} + p^{2}x^{2} + p^{59}x.$$

A.4. *Primitive trinomial T4:* $f(y) = y^6 + y^5 + 1$

$$g(x) = p^{16}x^{60} + p^{32}x^{58} + p^{8}x^{57} + p^{60}x^{56} + p^{60}x^{54} + p^{34}x^{53} + p^{2}x^{52} + p^{40}x^{51} + p^{7}x^{50} + p^{53}x^{49} + p^{30}x^{48} + p^{27}x^{46} + p^{6}x^{45} + p^{26}x^{44} + p^{53}x^{43} + p^{54}x^{42} + p^{27}x^{41} + p^{54}x^{40} + p^{51}x^{39} + p^{40}x^{38} + p^{32}x^{37} + p^{61}x^{36} + p^{8}x^{35} + p^{58}x^{34} + p^{37}x^{33} + p^{27}x^{32} + p^{29}x^{30} + p^{58}x^{29} + p^{48}x^{28} + p^{23}x^{27} + p^{8}x^{26} + p^{23}x^{25} + p^{35}x^{24} + p^{62}x^{23} + p^{62}x^{22} + p^{13}x^{21} + p^{27}x^{20} + p^{22}x^{19} + p^{41}x^{18} + p^{22}x^{17} + p^{33}x^{16} + p^{47}x^{15} + p^{5}x^{14} + p^{56}x^{13} + p^{37}x^{12} + p^{38}x^{11} + p^{22}x^{10} + p^{6}x^{9} + p^{47}x^{8} + p^{4}x^{7} + px^{6} + p^{53}x^{5} + p^{51}x^{4} + p^{14}x^{3} + p^{28}x^{2} + p^{15}x.$$

A.5. *Primitive trinomial T5:* $f(y) = y^6 + y + 1$

$$g(x) = p^{37}x^{58} + p^{23}x^{57} + p^{49}x^{56} + p^{43}x^{54} + p^{29}x^{53} + p^{34}x^{52} + p^{42}x^{51} + p^{45}x^{50} + p^{39}x^{49} + p^{62}x^{46} + p^{48}x^{45} + p^{21}x^{44} + p^{30}x^{43} + p^{42}x^{42} + p^{32}x^{41} + p^{26}x^{40} + p^{7}x^{39} + p^{44}x^{38} + p^{57}x^{37} + p^{27}x^{36} + p^{43}x^{35} + px^{34} + p^{55}x^{33} + p^{37}x^{32} + p^{50}x^{30} + p^{36}x^{29} + p^{57}x^{28} + p^{26}x^{27} + p^{7}x^{26} + p^{9}x^{25} + p^{49}x^{24} + p^{47}x^{23} + p^{3}x^{22} + p^{45}x^{21} + p^{19}x^{20} + p^{33}x^{19} + p^{14}x^{18} + p^{21}x^{17} + p^{39}x^{16} + p^{28}x^{15} + p^{37}x^{14} + p^{51}x^{13} + p^{24}x^{12} + p^{28}x^{11} + p^{2}x^{10} + p^{8}x^{9} + p^{32}x^{8} + p^{34}x^{7} + p^{21}x^{6} + p^{56}x^{5} + p^{24}x^{4} + p^{54}x^{3} + p^{34}x^{2} + p^{4}x.$$

A.6. Primitive pentanomial P6: $f(y) = y^6 + y^5 + y^2 + y + 1$

$g(x) = p^{61}x^{60} + p^{34}x^{58} + p^{45}x^{57} + p^{55}x^{54} + p^{61}x^{53} + p^{15}x^{52} + p^{11}x^{51} + p^{55}x^{50} + p^{12}x^{49} + p^{16}x^{48} + p^{52}x^{46} + p^{60}x^{45} + x^{44} + p^{25}x^{43} + p^{11}x^{42} + p^{31}x^{41} + p^{41}x^{40} + p^{39}x^{39} + p^{52}x^{38} + p^{12}x^{37} + p^{49}x^{36} + p^{41}x^{35} + p^{31}x^{34} + p^{21}x^{33} + p^{16}x^{32} + p^{41}x^{30} + p^{34}x^{29} + p^{47}x^{28} + p^{61}x^{27} + p^{35}x^{26} + p^{19}x^{25} + p^{61}x^{24} + p^{43}x^{23} + p^{24}x^{22} + p^{59}x^{21} + p^{48}x^{20} + p^{41}x^{19} + x^{18} + p^{40}x^{17} + p^{7}x^{16} + p^{38}x^{15} + p^{47}x^{14} + p^{24}x^{13} + x^{12} + p^{39}x^{11} + p^{2}x^{10} + p^{28}x^{9} + p^{28}x^{8} + p^{50}x^{7} + p^{6}x^{6} + p^{30}x^{5} + p^{15}x^{4} + x^{3} + p^{56}x^{2} + p^{47}x.$

Appendix B. Addition of powers x^n with common terms p^s

The addition of powers x^n with common terms p^s for the different primitive polynomials are as follows.

B.1. Pentanomial P1: $f(y) = y^6 + y^4 + y^3 + y + 1$

$p^7(x^{22} + x^{43}) = p^7 v_7$, $p^8(x^5 + x^{16} + x^{41})$, $p^{10}(x^{25} + x^{40})$, $p^{20}(x^{26} + x^{38})$, $p^{21}(x^{20} + x^{44} + x^{46})$, $p^{23}(x^4 + x^{51} + x^{53})$, $p^{26}(x^{27} + x^{54})$, $p^{36}(x^8 + x^{52} + x^{60})$, $p^{42}(x^{13} + x^{28})$, $p^{44}(x^{29} + x^{34} + x^{58})$, $p^{47}(x^7 + x^{12})$, $p^{52}(x^2 + x^{37})$ and $p^{57}(x^{21} + x^{32} + x^{42}) = p^{57}v_{57}$.

B.2. Pentanomial P2: $f(y) = y^6 + y^5 + y^4 + y + 1$

$p^7(x^{44}+x^{49}) = p^7 v_7$, $p^9(x^{36}+x^{52})$, $p^{11}(x^{38}+x^{60})$, $p^{22}(x^{13}+x^{46})$, $p^{26}(x^{29}+x^{43})$, $p^{30}(x+x^{33})$, $p^{32}(x^4+x^{16}+x^{30})$, $p^{37}(x^{32}+x^{45})$, $p^{38}(x^{20}+x^{23}+x^{25}+x^{42})$, $p^{40}(x^{14} + x^{22} + x^{57})$, $p^{45}(x^{27} + x^{34})$ and $p^{58}(x^6 + x^{21}) = p^{58}v_{58}$.

B.3. Pentanomial P3: $f(y) = y^6 + y^5 + y^3 + y^2 + 1$

$p^2(x^2 + x^{37}) = p^2 v_2$, $p^{12}(x^{52} + x^{58})$, $p^{16}(x^9 + x^{41})$, $p^{21}(x^4 + x^7)$, $p^{22}(x^{11} + x^{36} + x^{38} + x^{44})$, $p^{27}(x^3 + x^{29})$, $p^{31}(x^{16} + x^{26} + x^{28})$, $p^{36}(x^{48} + x^{50})$, $p^{38}(x^{12} + x^{17} + x^{42})$, $p^{44}(x^{24} + x^{51})$ and $p^{60}(x^{30} + x^{53}) = p^{60}v_{60}$.

B.4. Trinomial T4: $f(y) = y^6 + y^5 + 1$

$p^6(x^9+x^{45}) = p^6 v_6$, $p^8(x^{26}+x^{35}+x^{57})$, $p^{22}(x^{10}+x^{17}+x^{19})$, $p^{23}(x^{25}+x^{27})$, $p^{27}(x^{20} + x^{32} + x^{41} + x^{46})$, $p^{32}(x^{37} + x^{58})$, $p^{37}(x^{12} + x^{33})$, $p^{40}(x^{38} + x^{51})$, $p^{47}(x^8+x^{15})$, $p^{51}(x^4+x^{39})$, $p^{53}(x^5+x^{43}+x^{49})$, $p^{54}(x^{40}+x^{42})$, $p^{58}(x^{29}+x^{34})$, $p^{60}(x^{54} + x^{56})$ and $p^{62}(x^{22} + x^{23}) = p^{62}v_{62}$.

B.5. Trinomial T5: $f(y) = y^6 + y + 1$

$p^7(x^{26} + x^{39}) = p^7 v_7$, $p^{21}(x^6 + x^{17} + x^{44})$, $p^{24}(x^4 + x^{12})$, $p^{26}(x^{27} + x^{40})$, $p^{28}(x^{11} + x^{15})$, $p^{32}(x^8 + x^{41})$, $p^{34}(x^2 + x^7 + x^{52})$, $p^{37}(x^{14} + x^{32} + x^{58})$, $p^{39}(x^{16} + x^{49})$, $p^{42}(x^{42} + x^{51})$, $p^{43}(x^{35} + x^{54})$, $p^{45}(x^{21} + x^{50})$, $p^{49}(x^{24} + x^{56})$ and $p^{57}(x^{28} + x^{37}) = p^{57}v_{57}$.

B.6. Pentanomial P6: $f(y) = y^6 + y^5 + y^2 + y + 1$

$p^{11}(x^{42} + x^{51}) = p^{11}v_{11}$, $p^{12}(x^{37} + x^{49})$, $p^{15}(x^4 + x^{52})$, $p^{16}(x^{32} + x^{48})$, $p^{24}(x^{13} + x^{22})$, $p^{28}(x^8 + x^9)$, $p^{31}(x^{34} + x^{41})$, $p^{34}(x^{29} + x^{58})$, $p^{39}(x^{11} + x^{39})$, $p^{41}(x^{19} + x^{30} + x^{35} + x^{40})$, $p^{47}(x + x^{14} + x^{28})$, $p^{52}(x^{38} + x^{46})$, $p^{55}(x^{50} + x^{54})$ and $p^{61}(x^{24} + x^{27} + x^{53} + x^{60}) = p^{61}v_{61}$.

Appendix C. Coordinate expressions for $\mathbb{F}_{2^6}$ multipliers

Coordinates of the product $C = A \cdot B$ over $\mathbb{F}_{2^6}$, given in terms of the $\mathbf{S_i}$ and $\mathbf{T_i}$ functions, are as follows, where $(c_5, c_4, c_3, c_2, c_1, c_0)$, with $c_i \in \mathbb{F}_2$, are the coordinates of C.

C.1. Primitive pentanomial P1: $f(y) = y^6 + y^4 + y^3 + y + 1$

$c_0 = \mathbf{S_1}+\mathbf{T_0}+\mathbf{T_2}+\mathbf{T_3}+\mathbf{T_4}$, $c_1 = \mathbf{S_2}+\mathbf{T_0}+\mathbf{T_1}+\mathbf{T_2}$, $c_2 = \mathbf{S_3}+\mathbf{T_1}+\mathbf{T_2}+\mathbf{T_3}$, $c_3 = \mathbf{S_4}+\mathbf{T_0}$, $c_4 = \mathbf{S_5}+\mathbf{T_0}+\mathbf{T_1}+\mathbf{T_2}+\mathbf{T_3}+\mathbf{T_4}$, $c_5 = \mathbf{S_6}+\mathbf{T_1}+\mathbf{T_2}+\mathbf{T_3}+\mathbf{T_4}$.

Using subexpressions sharing (with intermediate variables $\mathbf{v_0} = \mathbf{T_1} + \mathbf{T_2}$, $\mathbf{v_1} = \mathbf{T_3} + \mathbf{T_4}$, $\mathbf{v_2} = \mathbf{v_0} + \mathbf{v_1}$): $c_0 = \mathbf{S_1} + \mathbf{T_0} + \mathbf{T_2} + \mathbf{v_1}$, $c_1 = \mathbf{S_2} + \mathbf{T_0} + \mathbf{v_0}$, $c_2 = \mathbf{S_3} + \mathbf{v_0} + \mathbf{T_3}$, $c_3 = \mathbf{S_4} + \mathbf{T_0}$, $c_4 = \mathbf{S_5} + \mathbf{T_0} + \mathbf{v_2}$, $c_5 = \mathbf{S_6} + \mathbf{v_2}$.

C.2. Primitive pentanomial P2: $f(y) = y^6 + y^5 + y^4 + y + 1$

$c_0 = \mathbf{S_1}+\mathbf{T_0}+\mathbf{T_1}+\mathbf{T_3}+\mathbf{T_4}$, $c_1 = \mathbf{S_2}+\mathbf{T_0}+\mathbf{T_2}+\mathbf{T_3}$, $c_2 = \mathbf{S_3}+\mathbf{T_1}+\mathbf{T_3}+\mathbf{T_4}$, $c_3 = \mathbf{S_4} + \mathbf{T_2} + \mathbf{T_4}$, $c_4 = \mathbf{S_5} + \mathbf{T_0} + \mathbf{T_1} + \mathbf{T_4}$, $c_5 = \mathbf{S_6} + \mathbf{T_0} + \mathbf{T_2} + \mathbf{T_3} + \mathbf{T_4}$.

Using subexpressions sharing (with intermediate variables $\mathbf{v_0} = \mathbf{T_3} + \mathbf{T_4}$, $\mathbf{v_1} = \mathbf{T_1} + \mathbf{v_0}$, $\mathbf{v_2} = \mathbf{T_0} + \mathbf{T_2}$): $c_0 = \mathbf{S_1} + \mathbf{T_0} + \mathbf{v_1}$, $c_1 = \mathbf{S_2} + \mathbf{v_2} + \mathbf{T_3}$, $c_2 = \mathbf{S_3} + \mathbf{v_1}$, $c_3 = \mathbf{S_4} + \mathbf{T_2} + \mathbf{T_4}$, $c_4 = \mathbf{S_5} + \mathbf{T_0} + \mathbf{T_1} + \mathbf{T_4}$, $c_5 = \mathbf{S_6} + \mathbf{v_2} + \mathbf{v_0}$.

C.3. Primitive pentanomial P3: $f(y) = y^6 + y^5 + y^3 + y^2 + 1$

$c_0 = \mathbf{S_1}+\mathbf{T_0}+\mathbf{T_1}+\mathbf{T_2}$, $c_1 = \mathbf{S_2}+\mathbf{T_1}+\mathbf{T_2}+\mathbf{T_3}$, $c_2 = \mathbf{S_3}+\mathbf{T_0}+\mathbf{T_1}+\mathbf{T_3}+\mathbf{T_4}$, $c_3 = \mathbf{S_4} + \mathbf{T_0} + \mathbf{T_4}$, $c_4 = \mathbf{S_5} + \mathbf{T_1}$, $c_5 = \mathbf{S_6} + \mathbf{T_0} + \mathbf{T_1}$.

Using subexpressions sharing (with intermediate variable $\mathbf{v_0} = \mathbf{T_0} + \mathbf{T_1}$): $c_0 = \mathbf{S_1} + \mathbf{v_0} + \mathbf{T_2}$, $c_1 = \mathbf{S_2} + \mathbf{T_1} + \mathbf{T_2} + \mathbf{T_3}$, $c_2 = \mathbf{S_3} + \mathbf{v_0} + \mathbf{T_3} + \mathbf{T_4}$, $c_3 = \mathbf{S_4} + \mathbf{T_0} + \mathbf{T_4}$, $c_4 = \mathbf{S_5} + \mathbf{T_1}$, $c_5 = \mathbf{S_6} + \mathbf{v_0}$.

C.4. Primitive trinomial T4: $f(y) = y^6 + y^5 + 1$

$c_0 = \mathbf{S_1} + \mathbf{T_0} + \mathbf{T_1} + \mathbf{T_2} + \mathbf{T_3} + \mathbf{T_4}$, $c_1 = \mathbf{S_2} + \mathbf{T_1} + \mathbf{T_2} + \mathbf{T_3} + \mathbf{T_4}$, $c_2 = \mathbf{S_3} + \mathbf{T_2} + \mathbf{T_3} + \mathbf{T_4}$, $c_3 = \mathbf{S_4} + \mathbf{T_3} + \mathbf{T_4}$, $c_4 = \mathbf{S_5} + \mathbf{T_4}$, $c_5 = \mathbf{S_6} + \mathbf{T_0} + \mathbf{T_1} + \mathbf{T_2} + \mathbf{T_3} + \mathbf{T_4}$.

Using subexpressions sharing (with intermediate variables $\mathbf{v_3} = \mathbf{T_3} + \mathbf{T_4}$, $\mathbf{v_2} = \mathbf{T_2} + \mathbf{v_3}$, $\mathbf{v_1} = \mathbf{T_1} + \mathbf{v_2}$, $\mathbf{v_0} = \mathbf{T_0} + \mathbf{v_1}$): $c_0 = \mathbf{S_1} + \mathbf{v_0}$, $c_1 = \mathbf{S_2} + \mathbf{v_1}$, $c_2 = \mathbf{S_3} + \mathbf{v_2}$, $c_3 = \mathbf{S_4} + \mathbf{v_3}$, $c_4 = \mathbf{S_5} + \mathbf{T_4}$, $c_5 = \mathbf{S_6} + \mathbf{v_0}$.

C.5. Primitive trinomial T5: $f(y) = y^6 + y + 1$

$c_0 = \mathbf{S_1} + \mathbf{T_0}$, $c_1 = \mathbf{S_2} + \mathbf{T_0} + \mathbf{T_1}$, $c_2 = \mathbf{S_3} + \mathbf{T_1} + \mathbf{T_2}$, $c_3 = \mathbf{S_4} + \mathbf{T_2} + \mathbf{T_3}$, $c_4 = \mathbf{S_5} + \mathbf{T_3} + \mathbf{T_4}$, $c_5 = \mathbf{S_6} + \mathbf{T_4}$.

C.6. Primitive pentanomial P6: $f(y) = y^6 + y^5 + y^2 + y + 1$

$c_0 = \mathbf{S_1} + \mathbf{T_0} + \mathbf{T_1} + \mathbf{T_2} + \mathbf{T_3}$, $c_1 = \mathbf{S_2} + \mathbf{T_0} + \mathbf{T_4}$, $c_2 = \mathbf{S_3} + \mathbf{T_0} + \mathbf{T_2} + \mathbf{T_3}$, $c_3 = \mathbf{S_4} + \mathbf{T_1} + \mathbf{T_3} + \mathbf{T_4}$, $c_4 = \mathbf{S_5} + \mathbf{T_2} + \mathbf{T_4}$, $c_5 = \mathbf{S_6} + \mathbf{T_0} + \mathbf{T_1} + \mathbf{T_2}$.

Using subexpressions sharing (with intermediate variables $\mathbf{v_0} = \mathbf{T_0} + \mathbf{T_2}$, $\mathbf{v_1} = \mathbf{T_3} + \mathbf{v_0}$): $c_0 = \mathbf{S_1} + \mathbf{T_1} + \mathbf{v_1}$, $c_1 = \mathbf{S_2} + \mathbf{T_0} + \mathbf{T_4}$, $c_2 = \mathbf{S_3} + \mathbf{v_1}$, $c_3 = \mathbf{S_4} + \mathbf{T_1} + \mathbf{T_3} + \mathbf{T_4}$, $c_4 = \mathbf{S_5} + \mathbf{T_2} + \mathbf{T_4}$, $c_5 = \mathbf{S_6} + \mathbf{T_1} + \mathbf{v_0}$.

References

[1] E. Biham, A. Shamir, Differential cryptanalysis of DES-like cryptosystems, J. Cryptol. 4 (1) (1991) 3–72.
[2] M. Matsui, Linear cryptoanalysis method for DES cipher, in: T. Helleseth (Ed.), Advances in Cryptology, Vol. 765, EUROCRYPT, Springer, Berlin, 1994, pp. 386–397.
[3] K. Nyberg, Differentially uniform mappings for cryptography, in: T. Helleseth (Ed.), Advances in Cryptology, Vol. 765, EUROCRYPT, Springer, Berlin, 1994, pp. 55–64.
[4] F. Chabaud, S. Vaudenay, Links between differential and linear cryptanalysis, in: Workshop on the Theory and Application of Cryptographic Techniques, Springer, Berlin, Heidelberg, 1994, pp. 356–365.
[5] Advanced encryption standard, in: FIPS, Vol. 197, 2001.
[6] K. Nyberg, L.R. Knudsen, Provable security against differential cryptanalysis, in: E.F. Brickell (Ed.), Advances in Cryptology, CRYPTO, Springer, Berlin, 1993, pp. 566–574.
[7] X.-D. Hou, Affinity of permutations on $\mathbb{F}_2^n$, Discrete Appl. Math. 154 (2006) 313–325.
[8] M. Calderini, M. Sala, I. Villa, A note on APN permutations in even dimension, Finite Fields Appl. 46 (2017) 1–16.
[9] J.F. Dillon, APN polynomials: an update, in: International Conference on Finite Fields and Applications Fq9, 2009.
[10] B. Bilgin, A. Bogdanov, M. Knezevic, F. Mendel, Q. Wang, Fides: Lightweight Authenticated Cipher with Side-Channel Resistance for Constrained Hardware, CHES 2013, in: LNCS 8086, 2013, pp. 142–158.
[11] J.L. Imaña, L. Budaghyan, N. Kaleyski, Decomposition of dillon's APN permutation with efficient hardware implementation, in: WAIFI 2022, in: LNCS 13638, 2023, pp. 250–268.
[12] C. Carlet, Boolean functions for cryptography and error correcting codes, in: Boolean Models and Methods in Mathematics, Computer Science, and Engineering, Cambridge Univ. Press, 2010, pp. 257–397, Ch. 8.
[13] L. Budaghyan, C. Carlet, Classes of quadratic APN trinomials and hexanomials and related structures, IEEE Trans. Inform. Theory 54 (5) (2008) 2354–2357.

[14] L. Budaghyan, T. Helleseth, N. Kaleyski, A new family of APN quadrinomials, IEEE Trans. Inform. Theory 66 (11) (2020) 7081–7087.
[15] L. Budaghyan, Construction and Analysis of Cryptographic Functions, Springer, 2015.
[16] C. Carlet, Boolean Functions for Cryptography and Coding Theory, Cambridge University Press, Cambridge, 2021.
[17] M. Calderini, On the EA-classes of known APN functions in small dimensions, Cryptogr. Commun. 12 (2020) 821–840.
[18] K. Browning, J. Dillon, M. McQuistan, A. Wolfe, An APN permutation in dimension six, in: Finite Fields: Theory and Applications FQ9, Vol. 518, 2010, pp. 33–42.
[19] Magma Computational Algebra System, Computational Algebra Group, University of Sydney, http://magma.maths.usyd.edu.au/magma/.
[20] L. Perrin, A. Udovenko, A. Biryukov, Cryptanalysis of a theorem: Decomposing the only known solution to the big APN problem, in: 36th International Cryptology Conference, CRYPTO 2016, in: LNCS 9815, 2016, pp. 93–122.
[21] F. Rodríguez-Henríquez, N. Saqib, A. Díaz-Pérez, Ç.K. Koç, Cryptographic Algorithms on Reconfigurable Hardware, Springer, New York, 2006.
[22] A. Reyhani-Masoleh, A. Hasan, Low complexity bit parallel architectures for polynomial basis multiplication over $GF(2^m)$, IEEE Trans. Comput. 53 (8) (2004) 945–959.
[23] J.L. Imaña, J.M. Sánchez, F. Tirado, Bit-parallel finite field multipliers for irreducible trinomials, IEEE Trans. Comput. 55 (5) (2006) 520–533.
[24] J.L. Imaña, Efficient polynomial basis multipliers for type II irreducible pentanomials, IEEE Trans. Circuits Syst. II Express Briefs 59 (11) (2012) 795–799.

José L. Imaña received the M.Sc. and Ph.D. degrees in physics from Complutense University, Madrid, Spain, in 1989 and 2003, respectively. He was an Electronic Design Engineer at the Madrid Institute of Technology, Spain. He is currently with the Department of Computer Architecture and Automation at Complutense University, where he was promoted to an Associate Professor with tenure in 2006. He has been the promoter and cofounder of the International Workshop on the Arithmetic of Finite Fields (WAIFI). His research interests include algorithms and VLSI architectures for computations in finite fields, computer arithmetic, cryptographic hardware and post-quantum cryptography.

Nikolay Kaleyski received the bachelor's and master's degrees in theoretical computer science from Charles University, Prague, in 2014 and 2016, respectively, and the Ph.D. degree from the Selmer Centre, University of Bergen, in 2021. In addition to his research activities, he actively reviews articles for several international journals and takes part in outreach and propagational activities for the Selmer Centre and the Department of Informatics, University of Bergen, where he is currently tenure track Associate Professor. His research interests include classes of cryptographically optimal functions over finite fields, including APN functions, AB functions, and planar functions, as well as mathematical constructions and related computational and algorithmic questions.

Lilya Budaghyan received the Ph.D. degree from the University of Magdeburg, Germany, in 2005, and the Habilitation degree from the University of Paris 8, France, in 2013. She is currently a Professor and the Head of the Selmer Center in Secure Communication, Department of Informatics, University of Bergen, Norway. She also conducted her research at Yerevan State University, Armenia; the University of Trento, Italy; and Telecom ParisTech, France. Her research interests include cryptographic boolean functions and discrete structures and their applications. Since 2018, she has been a member of the Norwegian Academy of Technological Sciences (NTVA). She was a recipient of the Trond Mohn Foundation Award in 2016, the Young Research Talent Grant from the Norwegian Research Council in 2014, a Post-Doctoral Fellowship Award from the Foundation of Mathematical Sciences of Paris in 2012, and the Emil Artin Junior Prize in Mathematics in 2011.