Binary Permutation Polynomial Inversion and Application to Obfuscation Techniques

Lucas Barthelemy[abd]
lbarthelemy@quarkslab.com

Ninon Eyrolles[a]
neyrolles@quarkslab.com

Guenaël Renault[bce]
guenael.renault@upmc.fr

Raphaël Roblin[bd]
raph.roblin@gmail.com

[a]Quarkslab, Paris, France
[b]Sorbonne Universités, UPMC Univ Paris 06, F-75005, Paris, France
[c]CNRS, UMR 7606, LIP6, F-75005, Paris, France
[d]UPMC Computer Science Master Department, SFPN Course
[e]Inria, Paris Center, PolSys Project

Keywords

Obfuscation; Permutation Polynomial

ABSTRACT

Whether it is for constant obfusation, opaque predicate or equation obfuscation, Mixed Boolean-Arithmetic (MBA) expressions are a powerful tool providing concrete ways to achieve obfuscation. Recent papers [22, 1] presented ways to mix such a tool with permutation polynomials modulo 2^n in order to make the obfuscation technique more resilient to SMT solvers. However, because of limitations regarding the inversion of such permutations, the set of permutation polynomials presented suffers some restrictions. Those restrictions allow several methods of arithmetic simplification, decreasing the effectiveness of the technique at hiding information. In this work, we present general methods for permutation polynomials inversion. These methods allow us to remove some of the restrictions presented in the literature, making simplification attacks less effective. We discuss complexity and limits of these methods, and conclude that not only current simplification attacks may not be as effective as we thought, but they are still many uses of polynomial permutations in obfuscation that are yet to be explored.

1. INTRODUCTION

Obfuscation techniques often use invertible functions to hide meaningful components of the program, especially in data-flow obfuscation. A common example is the *encoding* of variables or constants with affine functions [4], also used in cryptographic white-box setups [3]. A natural generalization of an affine function is a polynomial and if it is bijective, it is called *permutation polynomial*.

SPRO'16, October 28 2016, Vienna, Austria

ISBN 978-1-4503-4576-7/16/10...$15.00

DOI: http://dx.doi.org/10.1145/2995306.2995310

Permutation polynomials are the subject of many research efforts in mathematics [12, 13, 21]. The well-studied Dickson polynomials [8, 15] provide examples of permutation polynomials. Most of the results on these polynomials assume that their coefficients are in a finite field.

In the cryptographic context, applications of permutation polynomials appeared as a generalization of the RSA cryptosystem or for designing symmetric protocols [10, 14, 11, 7]. Only little work focuses on permutation polynomial with coefficients in the ring of integer modulo 2^n [6, 16], which represents the main application of such polynomials in the context of software obfuscation. In the context of error correcting code applied to mobile communications, permutation polynomials modulo 2^n appear as a central object to design interleavers [9, 19, 18].

To the best of our knowledge, papers related to applications of permutation polynomials modulo 2^n present results on the characterization of such polynomials or study subfamilies for which the inverse polynomial is given thanks to a closed-form formula.

This is particularly the case for the paper by Rivest [16] which provides a characterization of all the permutation polynomials modulo 2^n and the one by Zhou et al. [22] which uses a (strict) subfamily for which a closed-form formula is known for the inverse. Using a subfamily is a problem in the context of obfuscation since such polynomials will have an intrinsic form which can be recognized more easily and thus help the reverser. The attack proposed in [1] relies on this property.

Our main objective in this paper is to provide algorithms and implementations for the computation of the inverse of any permutation polynomial modulo 2^n. Thus, obfuscation designer can stop relying on permutation polynomials with specific form (and so easily breakable).

Before presenting our contributions, we recall some theoretical results which will be used in the sequel.

1.1 Binary Permutation Polynomials

In the sequel, we will be mainly interested in bijective functions of $\mathbb{Z}_{2^n} = \mathbb{Z}/2^n\mathbb{Z}$ the integer ring modulo 2^n. More precisely, we will consider objects as defined in the following:

DEFINITION 1. *Let A be a finite ring. A map $f : A \to A$ is said to be a* polynomial function *if there exists a polyno-*

mial P in $A[x]$ such that $\forall a \in A : f(a) = P(a)$. If, moreover f is bijective, this function and the polynomial P are said to be a permutation polynomial *of A. When $A = \mathbb{Z}_{2^n}$ we will call them* binary permutation polynomial*.*

In [16] Rivest provides the following result which characterizes binary permutation polynomials.

THEOREM 1 ([16]). *Let $A = \mathbb{Z}_{2^n}$ with $n \geq 2$ and let $P(x) = a_0 + a_1 x + \cdots + a_d x^d$ be a polynomial with integral coefficients. The polynomial P represents a function f of A which is a binary permutation polynomial if and only if a_1 is odd, $(a_2 + a_4 + a_6 + \dots)$ is even, and $(a_3 + a_5 + a_7 + \dots)$ is even.*

It is worth to note that the function f in the preceding theorem can be represented by many different polynomials. More precisely, we have the following results.

PROPOSITION 1. *Let A be a finite ring. The ring $\mathcal{F}(A)$ of polynomial functions on A is isomorphic to the quotient ring $A[X]/I$ where I is the polynomial ideal defined by $I = \{P \in A[x] : \forall a \in A, P(a) = 0\}$. In particular, the group $\mathcal{P}(A)$ (for the composition) of permutation polynomials of A is a subset of this quotient ring.*

In the case where A is a finite field, the structure of $\mathcal{F}(A)$ can be identified (since every function is polynomial) [13], but in the case of a finite ring it is far from easy. When A is a finite ring, there are many papers providing theoretical results on $\mathcal{F}(A)$ or $\mathcal{P}(A)$ (e.g. structure, isomorphism, cardinality) but only few reports on algorithmic aspects. In the rest of this paper, we will address the problem of efficiently computing a polynomial representing the inverse of a given binary permutation polynomial and consider some applications to obfuscation.

1.2 MBA Obfuscation using Binary Permutation Polynomials

In [22], Zhou et al. present a constant obfuscation based on two components: *Mixed Boolean-Arithmetic* (MBA) expressions and binary permutation polynomials. They present their own subset of binary permutation polynomials for which they provide a closed-form formula for inversion. For a given degree $m > 0$, they define the following subset of $\mathcal{P}(\mathbb{Z}_{2^n})$:

$$P_m(\mathbb{Z}_{2^n}) = \left\{ \sum_{i=0}^{m} a_i x^i \mid a_1 \text{ is odd and } a_i^2 = 0 \text{ for } i > 1 \right\}$$

and prove the following theorem.

THEOREM 2 ([22]). *The set $P_m(\mathbb{Z}_{2^n})$ is a permutation group under the functional composition operator $\circ$. The inverse of an element $f(x) = \sum_{i=0}^{m} a_i x^i \in P_m(\mathbb{Z}_{2^n})$ is given by $g(x) = \sum_{j=0}^{m} b_j x^j$ where each coefficient can be computed by: $b_m = -a_1^{-m-1} a_m$, $b_1 = a^{-1} - a_1^{-1} \sum_{j=2}^{m} j a_0^{j-1} A_j$, $b_0 = -\sum_{j=1}^{m} a_0^j b_j$ and*

$$b_k = -a_1^{-k-1} - a_1^{-1} \sum_{j=k+1}^{m} \binom{n}{r} a_0^{j-k} A_j, m - 1 \geq k \geq 2$$

where $A_m = -a_1^{-m} a_m$, and A_k are recursively defined by

$$A_k = -a_1^{-k} a_k - \sum_{j=k+1}^{m} \binom{j}{k} a_0^{j-k} A_j, \quad \text{for } 2 \leq k < m.$$

From this set, Zhou et al. propose an obfuscation technique based on the composition of two binary permutation polynomials $P(x)$ and $Q(x) = P^{-1}(x)$, and an m-variable MBA identity $E = 0$ with multiple non-zero terms. For a certain constant $K \in \mathbb{Z}_{2^n}$, one has $K = Q(P(K)) = Q(E + P(K))$. Expanding the expression yields an obfuscated expression containing m variables and yet having always K as an output value.

While this work provides an interesting way of using binary permutation polynomials in obfuscation, it presents two drawbacks: firstly, only a small subset of $\mathcal{P}(\mathbb{Z}_{2^n})$ is used and secondly, public work describing attacks of this technique [1] was recently published.

1.3 Attack of MBA Obfuscation

In [1], Biondi et al. describe three approaches to attack the MBA-based constant obfuscation: albegraic simplification, SMT solver-based deobfuscation, and synthesis-based deobfuscation. The algebraic simplification method is of first interest for us, since it is strongly based on the form (see Theorem 2) of the set of polynomials defined by Zhou et al. As stated by the authors of the attack, the recovering of both polynomials P and Q is based on the fact that the coefficients of $Q(x) = \sum_{j=0}^{m} b_j x^j$ are all even except for b_1. It is indeed stressed in [1] that, should polynomials from another family that the one of Theorem 2 be used to obfuscate, the reduction attack presented would fail.

In Section 2.3, we present how to compute the inverse of any binary permutation polynomial as characterized by Rivest, thus enlarging the family of potential obfuscating polynomials to be used in the MBA-based obfuscation. More precisely, we present the following contributions.

1.4 Contributions

In this paper, we assess the feasability of binary permutation polynomials inversion with two known methods: Lagrange interpolation and Newton's inversion algorithm. We show in Section 2.1 that using a ring extension, it is possible to use Lagrange interpolation in order to compute the inverse of a restrained set of permutation polynomials, namely the one defined by Expression (4) (this set includes the one defined in [22]). As Lagrange interpolation does not provide a method to inverse all binary permutation polynomials, and has a high complexity due to the fact that it operates on a ring extension, we detail in Section 2.3 how Newton's inversion method can be used in the context of binary permutation polynomials.

Through the use of the zero ideal as mentioned in Proposition 1 and in Section 2.2.1, we define a ring of reduced permutation polynomials, and we adapt Newton's method for compositional inversion on this particular ring. As opposed to the method used in [22] and Lagrange interpolation, this new method successfully inverts all the binary permutation polynomials. Moreover, the complexity of Newton's method proved to be far better than Lagrange interpolation since computations are made on integer modulo 2^n, and thus the cost of multiplication is far less than the one in the ring extension used during Lagrange interpolation.

Both the zero ideal and the algorithm for permutation polynomial inversion provide new possibilities for obfuscation, as well as for working with polynomials in $\mathbb{Z}_{2^n}$. In particular, it offers resilience of the technique presented by Zhou et al. from the algebraic attack in [1].

2. ON INVERTING BINARY PERMUTATION POLYNOMIALS

In this section, we introduce two methods to invert permutation polynomials in $\mathbb{Z}_{2^n}[x]$. In contrary to the formula given in [22] that allows to invert a thin part of all the permutation polynomials, we present two general algorithms that efficiently inverse more permutation polynomials, one of them even inverting all polynomials as characterized by Rivest in [16].

We first introduce an interpolation method and show its limitations, then present a more efficient inversion method based on a modular functional Newton approach. Even if the first method is far more costly in term of performances than the second one, its exposition greatly helps to understand what is the best strategy for inverting permutation polynomials.

2.1 Inversion Through Interpolation

In order to identify a polynomial given as a black box, it is usual to proceed by interpolation [23, 5]. The main problem here is to find enough sample points. In this section, we present a method allowing this computation in $\mathbb{Z}_{2^n}[x]$.

Thorough this article, we consider applications $\mathbb{Z}_{2^n} \to \mathbb{Z}_{2^n}$ which are defined by evaluation of a polynomial $f(x)$. We will use the same notation for the polynomial and the corresponding function. In particular, X will denote the identity application: $x \xrightarrow{X} x \quad \forall x \in \mathbb{Z}_{2^n}$.

2.1.1 Lagrange Interpolation in a Modular Ring

Let us consider a function φ taking as entry an integer in $\mathbb{Z}_{2^n}$ and supplying the polynomial evaluation of f at this point:

$$x \xrightarrow{\varphi} f(x)$$

For each point evaluated, we can predict the result of φ^{-1} without its knowledge:

$$x \xrightarrow{\varphi} y \quad \Leftrightarrow \quad y \xrightarrow{\varphi^{-1}} x$$

Let us assume that φ^{-1} can be defined as the evaluation of an (unknown) polynomial f^{-1}. If, moreover, the polynomial f is taking values in a finite field and f^{-1} is of degree d_{inv}, then from $d_{inv}+1$ tuples $(y_i = f(x_i), x_i)$ one can find f^{-1} by computing a Lagrange interpolation:

$$f^{-1}(x) = \sum_{j=0}^{d_{inv}} x_j \prod_{i \neq j} \frac{(x - y_i)}{(y_j - y_i)} \tag{1}$$

In the case of binary permutation polynomials, the Expression (1) is not always computable, as half of the elements in $\mathbb{Z}_{2^n}$ are zero-divisors. Thus, by considering the determinant of the Vandermonde's matrix V_f of f

$$\det(V_f) = \prod_{0 \leq i < j \leq d_{inv}} (y_j - y_i) \tag{2}$$

which is equal to the denominator of Expression (1), the computation of the Lagrange interpolation is possible if and only if $\det(V_f)$ is inversible.

However, it can be shown that if the y_i are in the ring $\mathbb{Z}_{2^n}$, then as soon as $d_{inv} \geq 2$, the product $\prod_{0 \leq i < j \leq d_{inv}} (y_j - y_i)$ is not invertible. Therefore, Lagrange interpolation is not applicable as usual on this particular ring.

2.1.2 Ring Extension

The impossibility to use Lagrange interpolation comes from the fact that there is no invertible combination of three or more elements in $\mathbb{Z}_{2^n}$ such that the product in Expression (2) is invertible. To bypass this problem, the base ring needs to be extended. The approach is similar to field extension, i.e. to extend the base ring $\mathbb{Z}_{2^n}$ with a new symbolic element ω, which defines the bigger ring $\mathbb{Z}_{2^n}[\omega]$ as the quotient of $\mathbb{Z}_{2^n}[x]$ by an irreducible polynomial P of degree m to which ω is a root. The resulting ring is $\mathbb{Z}_{2^n}[\omega] = \mathbb{Z}_{2^n}[x]/P$ and will be noted as $\mathbb{Z}_{2^n}^m$. We will show how such a sufficiently big extension allows Lagrange interpolation in our context. These extensions generally are called Galois rings.

To know if the application of a Lagrange interpolation is possible in such an extension, the progression of the number of invertible elements is of first interest. The following fundamental result from [20] gives a formal definition of Galois rings which will help us recognize invertible elements.

THEOREM 3. *In the Galois ring $\mathbb{Z}_{2^n}[\omega]$, there exists a nonzero element α of multiplicative order $2^m - 1$ which is a root of a monic basic primitive polynomial $h(x)$ in $\mathbb{Z}_{2^n}[x]$ of degree m and dividing $x^{2^m-1} - 1$. The element α verifies that*

$$\mathbb{Z}_{2^n}[\alpha] = \{a_0 + a_1\alpha + \cdots + a_{m-1}\alpha^{m-1} : a_0, a_1, \ldots, a_{m-1} \in \mathbb{Z}_{2^n}\}$$

is isomorphic to $\mathbb{Z}_{2^n}[\omega]$.

This theorem states that any Galois ring extension can be defined by an element α which is primitive. This means that any element of the extension can be written with powers of α as explained in the following Theorem.

THEOREM 4. *With the same notations as in Theorem 3. Let the set*

$$\mathcal{T} = \{0, 1, \alpha, \ldots, \alpha^{2^m-2}\},$$

then any element $e \in \mathbb{Z}_{2^n}[\alpha]$ can be written uniquely as

$$e = a_0 + a_1 \times 2 + \cdots + a_{n-1} \times 2^{n-1}$$

where $a_0, a_1, \ldots, a_{n-1} \in \mathcal{T}$. Moreover, e is invertible if and only if $a_0 \neq 0$.

This representation is called 2-adic representation. If we take a look at this representation modulo 2, we have:

$$e = a_0 \mod 2$$

Therefore, in order to have $a_0 \neq 0$, we need $e \neq 0 \mod 2$. When e is represented as a polynomial in ω, to achieve such a condition, e needs to have at least one odd coefficient (since all even coefficients become null modulo 2). Thus we can characterize the invertible elements of $\mathbb{Z}_{2^n}^m$ as elements with at least one odd coefficient, while elements with only even coefficients are zero-divisors.

In order to compute Lagrange interpolation, we need $d_{inv}+1$ elements y_i such that the product

$$\prod_{0 \leq i < j \leq d_{inv}} (y_j - y_i) \tag{3}$$

is invertible.

In order to obtain a set of elements such that the product of their difference is invertible (as in Expression (3)), for any

couple of elements within that set, there must be at least one degree on which their coefficients have different parity (so that the difference between those two coefficients is odd). Note that the value of said coefficients is not important, only their parity. Thus, let us consider the elements of $\mathbb{Z}_{2^n}^m$ with coefficients in $\{0,1\}$. The number of elements for which the product of differences are invertible is the number of distinct elements with m coefficients in $\{0,1\}$, i.e. a set of elements for which the result of Expression (3) is of size at most 2^m.

To compute Lagrange interpolation, we need a set of size at least d_{inv} elements. Thus, when choosing the extension degree m, we need:

$$m > \log_2(d_{inv})$$

Note that n does not appears in this inequality. However, the following section will demonstrate that d_{inv} needs to be estimated, and that estimation depends on n.

The problem is that we are using extended Lagrange interpolation to invert a polynomial $f(x)$, therefore we do not generate the set $\{y_0, \ldots, y_{d_{inv}}\}$ but $\{x_0, \ldots, x_{d_{inv}}\}$ on which f will be evaluated on. Predicting the invertibility of

$$\prod_{0 \le i < j \le d_{inv}} (f(x_j) - f(x_i))$$

is a non-trivial problem. But by knowing the number of invertible elements in $\mathbb{Z}_{2^n}^m$, the probability that the matrix V_f is invertible when randomly drawing a set E of d_{inv} elements can be computed. Let $Pr(V_{f,E}^{-1})$ be the probability that the Vandermonde matrix determinant of the polynomial f evaluated on the random set of elements E is invertible. In other words:

$$E = \{x_0, \ldots, x_{d_{inv}}\}$$
$$Pr(V_{f,E}^{-1}) = Pr\Big(\prod_{0 \le i < j \le d_{inv}} (f(x_j) - f(x_i)) \text{ is invertible}\Big)$$

Assuming that the permutation polynomial f is injective within the extended ring, the following property holds:

Proposition 2. *The probability to obtain an invertible Vandermonde matrix when randomly drawing d_{inv} elements in $\mathbb{Z}_{2^n}^m$ is:*

$$\begin{aligned} Pr(V_{f,E}^{-1}) &= 1 - \overline{Pr(V_{f,E}^{-1})} \\ &= 1 - \frac{2^{(n-1)m}}{2^{nm}} = 1 - \frac{1}{2^m} \end{aligned}$$

Therefore that probability increases proportionally to m.

Regarding complexity, Lagrange interpolation has a complexity quadratic in the number of points we interpolate on. But those points are now elements of $\mathbb{Z}_{2^n}^m$, meaning computing on those points is equivalent to computing on polynomials of degree m. Thus we achieve a complexity of:

$$O((d_{inv}\mathcal{M}(m))^2) \approx O((d_{inv}m^2)^2)$$

We need to find a balance between the cost of computing $det(V_f)$ with evaluation points E and that of performing Lagrange interpolation.

This allows us to perform a Lagrange interpolation successfully. Moreover, $\mathbb{Z}_{2^n}^m$ being extended from $\mathbb{Z}_{2^n}$, if f^{-1} exists in $\mathbb{Z}_{2^n}$, then the interpolation should produce a result in $\mathbb{Z}_{2^n}$ (i.e. all coefficients in α are null). We will see in Sections 2.1.3 and 2.2.3 that this is not always the case, due to non-uniqueness of compositional identity.

2.1.3 Reachable Polynomials and Complexity

While the complexity of the extended Lagrange interpolation is far worse than the closed-form expression provided in [22], it successfully inverts all polynomials of the set provided in this paper.

Let us consider an intermediary set between the one defined in [22] and the full set of permutation polynomials as characterized in [16]:

$$f(x) = \sum_{i=0}^{d} a_i x^i \begin{cases} a_1 \text{ is odd} \\ a_i \text{ is even } \forall i \ge 2 \end{cases} \quad (4)$$

The extended Lagrange interpolation also offers the possibility of inverting such polynomials, but the alleged inverse degree has to be greatly increased. With our own implementation of extended Lagrange, we determined empirically that d_{inv} needs to be approximately equal to $d \times n$. As recalled above the Lagrange interpolation is quadratic in the number of elements interpolated on, that are themselves elements of $\mathbb{Z}_{2^n}^m$. Thus, we obtain a complexity of $O((dn)^2 m^4)$ for such a computation when $d_{inv} = d \times n$.

Using this Lagrangian strategy our goal of inverting the full set of permutation polynomials is not achieved but provide more inversions than the close formula given in [22]. Unfortunately, the result of the extended Lagrange interpolation remains in $\mathbb{Z}_{2^n}^m$ regardless of the number of points interpolated on.

A main drawback of ring extension, apart from the increase in complexity, is that it requires to work on the ring $\mathbb{Z}_{2^n}[X]$ as opposed to $\mathcal{P}(A)$ as described in Section 2.1.2. The next section will develop why this prevents the inverse computation of all permutation polynomials as characterized by Rivest.

2.2 On Polynomial Ideals

Addressing the problem of d_{inv} alleged huge value requires to focus on the problem of *polynomial simplification*, meaning that for a given polynomial f, one asks for a polynomial g (if it exists) verifying:

$$g(x) = f(x) \mod 2^n \quad \forall x \in \mathbb{Z}_{2^n} \text{ with } \deg(g) < deg(f)$$

In [17], a method for polynomial simplification using polynomial ideal is presented: in this section, we adapt their method to our case. An ideal of $\mathbb{Z}_{2^n}[x]$ generated by the finite sequence of polynomials $\{F_0, F_1, \ldots, F_s\}$ is the set of all its $\mathbb{Z}_{2^n}[x]$-linear combinations:

$$\langle F_0, \ldots, F_s \rangle = \{\sum_{i=0}^{s} G_i F_i : G_i \in \mathbb{Z}_{2^n}[x]\}$$

2.2.1 Zero Ideal on $\mathbb{Z}_{2^n}^*$ and $\mathbb{Z}_{2^n}$

The method presented in [17] is defined in the ring of polynomials with coefficients in $\mathbb{Z}_{2^n}^*$, the ring of inversible elements in $\mathbb{Z}_{2^n}$. The authors define an ideal I^* generated by a finite set $\{P_1^*, \ldots, P_{d_{n+1}^*}^*\}$ of polynomials which evaluate globally to 0 as a function taking values in $\mathbb{Z}_{2^n}^*$. Thus the ideal I^* verifies:

$$P \in I^* \iff P(x) = 0 \mod 2^n \qquad \forall x \in \mathbb{Z}_{2^n}^*$$

The authors define this set of polynomials in the following

way (we choose here to use positive roots):

$$P_i^*(x) = 2^{n-i-t_i} \prod_{j=0}^{i} (x - 2j - 1)$$

where t_i is the largest integer ℓ such that 2^ℓ divides $i!$. The integer d_n^* is defined as the largest integer i such that $n-i-t_i$ is positive. In particular, $P_0^* = 2^n$ and $P_{d_{n+1}^*}^* = (x-1)(x-3)\cdots(x-2d_n^*+1)$.

Let us recall that, in a field, for a variable a to be root of a polynomial, $(x-a)$ must divide that polynomial. However, in a general ring, zero-divisors provide an alternative way of producing 0.

When x is an odd integer, the value $\prod_{j=1}^{i}(x-2j-1)$ is a product of i consecutive even numbers. Therefore, as explained in [17], it is divisible by $2^i i!$ and so it is divisible by 2^{i+ti}. Multiplying this polynomial by 2^{n-i-t_i} results in 2^n which is equal to zero in $\mathbb{Z}_{2^n}$.

From The P_i^* generating the ideal I^* on $\mathbb{Z}_{2^n}^*$, we deduce the set of polynomials P_i generating the ideal I in $\mathbb{Z}_{2^n}[x]$ of all the polynomials evaluating to 0 for any value of x in $\mathbb{Z}_{2^n}$, i.e. I is the *zero ideal* of $\mathbb{Z}_{2^n}[x]$ (see Proposition 1). The definition of P_i is derived from P_i^*, by considering even roots, so that if x is even, it still induces a product of i consecutive even numbers:

$$P_i = 2^{n-i-t_i} \prod_{j=0}^{2i-1} (x - j)$$

Note that P_0 is equal to 2^n and the degree of P_i is increasing with i. The index of the polynomial P_i with highest degree is defined as follows. Let i_{max} be the greatest integer ℓ such that $n-(\ell-1)-t_{\ell-1} > 0$. Two cases are possible, i.e $n - i_{max} - t_{i_{max}}$ equals 0 or is negative, but the polynomial $P_{i_{max}}$ is always defined as

$$P_{i_{max}} = \prod_{j=0}^{2i_{max}-1} (x - j)$$

2.2.2 Polynomial Simplification

The generating set $\{P_o, \ldots, P_{i_{max}}\}$ of the zero ideal I provides an algorithmic method to compute in $\mathbb{Z}_{2^n}[x]$ modulo I. In particular, for a given f, this let us compute a polynomial g of smallest degree in the class $\{f + h \ : \ h \in I\}$ of f modulo I. Such a computation is called *polynomial simplification*. As stated in Proposition 1, any polynomial g in the class of f will define the same function on $\mathbb{Z}_{2^n}$, thus the simplification procedure let us define a binary permutation polynomial with the smallest polynomial as possible.

We now describe this simplification method. First, note that for all i, the polynomial P_i is the product of a monic polynomial (with a leading coefficient equals to 1) and a constant equal to a power of 2 (which we will refer as exp_i in the sequel). Moreover, the more i increases (therefore the degree of P_i), the smaller exp_i gets. From these remarks, one can simplify a given permutation polynomial f by using the following procedure. If exp_i divides the leading coefficient a_d of $f(x) = \sum_{i=0}^{d} a_i x^i$ and $deg(P_i) < d$, let us define g the permutation interpreted as the polynomial:

$$g(x) = f(x) - (\frac{a_d}{exp_i})(x^{d-deg(P_i)})P_i$$

Then g is in the same class as f modulo I and $deg(g) < d$ (i.e g is a reduction of f modulo I).

The process continues with the new polynomial g while the conditions on the leading coefficient and the degree are true. At the end, one obtains the reduction of f modulo I with the smallest degree.

To resume the previous discussion, the permutation polynomial set $\mathcal{P}(\mathbb{Z}_{2^n})$ is now seen as a quotient sub-group of $\mathbb{Z}_{2^n}[X]/I$ as described in the introduction. For each class in $\mathcal{P}(\mathbb{Z}_{2^n})$, one can choose, as a representative, a polynomial of degree smaller than the ones of the P_i. Hence, these representatives have a degree smaller than $d_n = 2 \times i_{max}$ the degree of $P_{i_{max}}$ and thus, the set $\mathcal{P}(\mathbb{Z}_{2^n})$ is finite.

We then deduce the following central result.

THEOREM 5. *The ideal $I = \langle P_0, \ldots, P_{i_{max}} \rangle$ verifies*

$$\text{for } f, g \in \mathbb{Z}_{2^n}[X] \text{ and } h \in I$$
$$f = g - h \iff f(x) = g(x) \mod 2^n \quad \forall x \in \mathbb{Z}_{2^n}$$

Moreover any binary permutation polynomial can be equally represented by a polynomial of degree at most $d_n = Deg(P_{i_{max}})$.

We do not provide a complete proof of this result, it will be detailed in an extended version of this paper.

2.2.3 Identity Polynomials in $\mathbb{Z}_{2^n}$

Simplification is not the only result induced by the zero ideal. Let us define the set $\mathcal{I}_{2^n}$:

$$\mathcal{I}_{2^n} = \{x + h(x)\} \quad \forall h \in I$$

This set defines a class of polynomials which are, as functions, equivalent to X. Therefore, for a given polynomial f, if there exists g such as $g \circ f \in \mathcal{I}_{2^n}$, g is equivalent to f^{-1}.

This is an important result as it hints the reason why Lagrange interpolation struggles to invert some polynomials. Lagrange interpolation searches for a polynomial verifying $g(f(x)) = x \mod 2^n$, but as we shown, there are other choices producing a compositional inverse. That is to say, x (the polynomial) is not the only permutation polynomial defining X (the function). This means that:

$$g(f(x)) = x \mod 2^n \quad \forall x \in \mathbb{Z}_{2^n} \iff g \circ f = X$$

Let g be a polynomial such as $g(f(x)) = x + h(x)$ with $h \in I$, we witnessed, through experimentation, that replacing x in the Lagrange formula by $x + h(x)$ produced a valid g. The problem with such an approach is that, unlike $\mathcal{P}(\mathbb{Z}_{2^n})$, $\mathcal{I}_{2^n}$ is not finite. Therefore predicting h before interpolation did not seem reliable in practice.

However, other inversion methods do not present such restrictions. Newton's method for compositional inversion is one of them.

2.3 Inversion Through Newton's Method

In this section we present an adaption for the binary permutation polynomials of the Newton's method for the computation of the compositional inverse as presented in [2].

2.3.1 Iterative Inversion in Fields

Newton's method (also known as Newton-Raphson method) aims to find successively better approximations of the roots of a function f on some set E. It is based on the assumption that, near a point x_0 in the vicinity of a root, a function's behavior is almost that of its tangent line:

$$f(x) \approx f(x_0) + f'(x_0)(x - x_0) \quad \forall x \in E$$

Therefore, one can approximate a better root by finding x such that:

$$f(x_0) + f'(x_0)(x - x_0) = 0$$
$$x = x_0 - \frac{f(x_0)}{f'(x_0)}$$

As long as f is derivable and that its derivative evaluation in some x_i is invertible, one can define the following iteration:

$$x_{i+1} = x_i - \frac{f(x_i)}{f'(x_i)}$$

Where x_{i+1} is a better root approximation than x_i. Therefore, if a root exists, the algorithm should converge to it (they are some exceptions in which the algorithm is "trapped" in a cycle, but we never encountered this problem while testing this algorithm in $\mathbb{Z}_{2^n}$).

This method can be adapted for other purposes, for example:

$$x_{i+1} = x_i - \frac{f(x_i) - 7}{f'(x_i)}$$

should converge to x such as $f(x) = 7$.

The formula linked to our problematic is that of compositional inverse:

$$g_{i+1} = g_i - \frac{f \circ (g_i - X)}{f' \circ g_i}$$

The main problem with this formula is the arithmetic inversion of $f'(g_i)$. But it is possible to use the following simplification:

$$f \circ g = X$$
$$(f \circ g)' = (f' \circ g) \times g' = 1$$
$$g' = \frac{1}{f' \circ g}$$
$$g_{i+1} = g_i - g_i' \times (f \circ (g_i - X))$$

This method was defined on a field, and in the following section we address the subject of its use in $\mathbb{Z}_{2^n}$.

2.3.2 The $\mathbb{Z}_{2^n}$ Case

In order to assess the correctness of the method in $\mathbb{Z}_{2^n}$, we use the two properties for Newton's method to correctly compute a compositional inverse [2]:

$$f(0) = 0$$
$$f'(0) \text{ is invertible}$$

The second condition is respected as $f'(0) = a_1$ which is always invertible, regardless of the polynomial we study.

In order to check for the first property, let us consider the polynomial:

$$r(x) = \sum_{i=1}^{d} a_i x^i$$

we have $f = r + a_0$, meaning f produces the same permutation as r with an additional shift. Therefore, if we invert r, we have the following equality:

$$r^{-1} \circ f = f \circ r^{-1} = a_0 + r \circ r^{-1} = a_0 + X$$
$$(r^{-1} \circ f) - a_0 = X$$

The polynomial $r(x)$ having no constant coefficient, this means $r^{-1} - a_0$ is inverse of f. However, $r(0) = 0$, thus r is invertible through Newton's method (in practice, Newton's method in $\mathbb{Z}_{2^n}$ converges while $f(0) \neq 0$ most of the time).

Therefore, if there exists g such as $g \circ f = X$, Newton's method should converge to g. We tried to use this method on the set of polynomials which were invertible through Lagrange interpolation and the method succeeded every time.

However, the full set of permutation polynomials as characterized in [16] still remains out of reach for Newton's method as it is. But, as mentioned in Section 2.2.3, this results from the fact that:

$$g(f(x)) = x \mod 2^n \quad \forall x \in \mathbb{Z}_{2^n} \iff g \circ f = x$$

Let us take a second look at the formula presented in the previous section:

$$g_{i+1} = g_i - \frac{f \circ (g_i - X)}{f' \circ g_i}$$

The problem is that x is not the only polynomial equivalent to the identity function X modulo the zero ideal I. Thus if $g(f(x)) = x + h(x)$ with $h \in I$ not equal to 0, this formula would become:

$$g_{i+1} = g_i - \frac{f \circ (g_i - x - h(x)))}{f' \circ g_i}$$

However, we will see in the next section an alternative way of computing on reduced polynomials, which does not assume the knowledge of h.

2.3.3 The Full Set is Reachable

Unlike Lagrange interpolation, Newton's method does not require the use of an extended ring. Thus, one can compute Newton's method in the ring of reduced polynomials $\mathcal{F}(\mathbb{Z}_{2^n})$ where:

$$g(f(x)) = x \mod 2^n \quad \forall x \in \mathbb{Z}_{2^n} \iff g(f) = x$$

Using Newton's method in the ring of reduced polynomials consists in reducing modulo I the result of each iteration (see Section 2.2.2). Then, during the computation, if a polynomial g equivalent to x appears it would be reduced to x. Thus, the problem emphasized above is now solved.

We tried to inverse all the permutation polynomials as characterized in [16]. Not only does it succeed in inverting all the polynomials, but the fact that the degrees of the reduced polynomials are bounded prevents unnecessary increase of the size of the objects during composition and multiplication.

As far as computation goes, each iteration requires a composition and a multiplication (along with some additions and subtractions). In theory, Newton's method converge to a solution with quadratic speed (see [2]). Meaning to inverse a polynomial, the number of iterations required would be squared in the degree of its inverse. This degree being bounded by n, we get at most $\sqrt{n}$ iterations.

Considering a quadratic cost for polynomial multiplication and composition, an overall complexity of $O(d_n^2 \times \sqrt{n})$ is achieved, where d_n is a bound on the degree of a reduced polynomial (see Section 2.2.2). This complexity is far better than the one of the Lagrange interpolation as described above.

Note that the cost of polynomial reduction does not change this complexity estimation. Actually. if we consider consider

the product of two polynomials of degree d_{max}, the resulting polynomial before reduction is then of degree $2 \times d_{max}$. Reducing that polynomial to a polynomial of degree d_{max} requires d_{max} polynomial subtractions, and therefore presents a complexity of $O(d_{max}^2)$. On the same principle, further reduction from this polynomial of degree d_{max} to a polynomial of smaller degree will at most be quadratic in the degree, which is the same complexity as the other computations performed (multiplications and compositions) during each iterations of Newton's inversion.

2.3.4 On the degree of the inverse

An interesting result which could follow this study is the relation between the degree of a permutation polynomial and the degree of its minimal inverse (let us recall that in contrary to the set defined in [22], the degree of a permutation polynomial is not necessary the same as its inverse).

Because the polynomial set defined in [22] does not present such a problem, our hypothesis is that this difference in degree is tight to the difference between this set and the full characterization given in [16].

Permutation polynomials presented in [22] are polynomials $\sum_i a_i x^i$ such that:

$$2^{\frac{n}{2}} | a_i \qquad \forall i \geq 2$$

Therefore each coefficient a_i with $i \geq 2$ has a very high 2-adic valuation.

The set of all the permutation polynomials as characterized by Rivest does not present such a restriction: it includes polynomials with coefficients of small 2-adic valuation. In the opposite, those polynomials results in high degree differences between a polynomial and its inverse. We leave these observations as an open problem that will be studied in a future work.

3. RESULTS AND APPLICATIONS

3.1 Experimental Results

We have implemented all these methods in `Python` and `C` language. In this section we report on their performances and applications. Experimental results presented in Figures 1 and 2 were realized on a *Lenovo T430* with an $Intel\ Core\ i7-3520M\ CPU\ @\ 2.90GHz \times 4$.

3.1.1 Performances

For the results presented in Figures 1, we uniformely drawn permutation polynomials of various degrees and mesured inversion time.

As stated in Section 2.3.3, Newton's iterative method for computing a compositional inverse on $\mathbb{Z}_{2^n}$ converges in $\sqrt{n}$ iterations, regardless of the degrees of f and f^{-1} (while the cost of performing one iteration does depend on the degree). However, our experimental results tend to show a convergence in $\log_2(n)$ iterations. Newton's method is known to show such convergence rates in some very peculiar cases (see [2]). We have yet to formalize the fact that our problem belongs to one of those, this will be part of our following work on this topic.

Moreover, as one can see on Figure 1 the complexity of the Newton's method seems to depend linearly in the bound on the degree of the given polynomial than quadratically. This behavior may be explained by the simplification method using the zero ideal presented in Section 2.2.2. This reduction allows us to bound the degree of the intermediate polynomials, reducing costs even further. This bound being reached extremely rapidly by the degree of our intermediate polynomial (composition and multiplication induce a rapid growth of its degree), the cost of increasing the degree of the initial polynomial only affects the operations of composition and multiplication linearly. We also plan to study more precisely, in an extended version of this paper, the tight complexity of the Newton's method in this context.

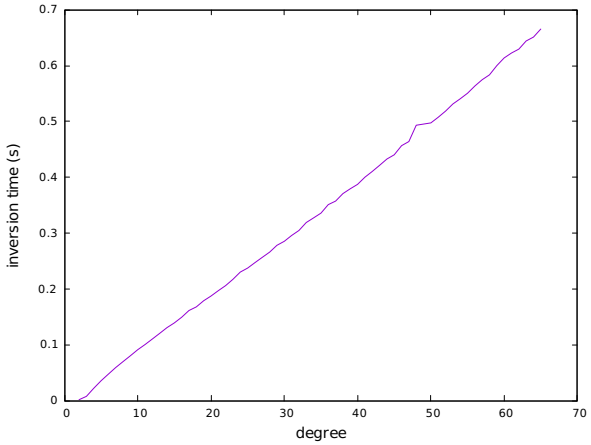

Figure 1: Invertion time using Newton's method on 64 bits.

3.2 Applications in Obfuscation

In this section, we assess the use of our contributions in the field of program obfuscation, whereas by using permutation polynomial inversion, or identity polynomials as presented in Section 2.2.3.

The cost of obfuscation can be determined either during obfuscation time or during execution. We focus more on the latter, since obfuscation usually has limitations regarding the performance overhead it produces, but more rarely on the time needed to actually obfuscate the input program.

3.2.1 Using Permutation Polynomial Inversion

The obfuscation technique presented in [22] aims at hiding a constant K with a permutation polynomial f, its inverse f^{-1} and an MBA expression E equivalent to zero, with the following process:

$$K = f^{-1}(E + f(K))$$

Thus what will appear in the obfuscated code is a polynomial in the variables of E. Obviously, this method could be used with any non-trivial expression E equivalent to zero, not only MBA expressions.

Our results regarding binary permutation polynomial inversion allow any designer to use any of these permutations for this type of obfuscation. This offer both more diversity in the obfuscation technique, and resilience to algebraic attacks such as presented in [1].

For example, the binary permutation polynomial:

$$f = 195907858x^3 + 727318528x^2 + 3506639707x + 6132886$$

could be used to obfuscate a constant K. However 195907858 is only divisible by 2 while 727318528 is divisible by 2^{16}.

Therefore, when trying to simplify f with the technique presented in [1], the factor 2^{16} is supposed to divide both coefficients a_3 and a_2, and in fact, it only divides a_3. This polynomial thus provides a counter example to the attack proposed in [1].

More generally, f and f^{-1} can be used to encode variables or constants. For constants, this means computing the value $f(K)$ during obfuscation, storing this value in the program instead of K, and using $f^{-1}(f(K))$ during execution. In the same manner, one can replace every writing occurrences of a variable x with $f(x)$ and all reading instructions of $f(x)$ with $f^{-1}(f(x))$. This process is quite common in obfuscation as it prevents the value of x from appearing in memory, but is more costly since it requires two evaluations of a permutation polynomial (f and f^{-1}) compared to the encoding of a constant ($f(K)$ is computed during obfuscation). This also means that during the obfuscation process, encoding a variable requires to check for the degrees of both f and f^{-1} for performances issues, while constant encoding only requires a small degree for f^{-1}.

Regarding encoding of variables or constants, the use of permutation polynomials does not provide much more security compared to affine functions, as much as it provides diversity. Indeed, we showed in Section 2.3 that inverting permutation polynomials is not a difficult problem, as Newton's method efficiently invert all binary permutation polynomials. Nevertheless, encoding with such polynomials contributes to the diversity of the obfuscation technique, thus increasing the difficulty to detect it by reverse engineering.

3.2.2 *Using Polynomials of the Zero Ideal I*

From the zero ideal I we defined in Section 2.2.1, we can use polynomials in I to design two obfuscation techniques. Let us consider a polynomial $h \in I$, then $h(x) = 0 \mod 2^n \quad \forall x \in \mathbb{Z}_{2^n}$. Therefore h can be used to produce an *opaque* zero, i.e. a non-trivial null expression. This could be used for example as an opaque predicate (see [4]), or in replacement of the MBA expression in Zhou et al. [22] obfuscation:

$$K = f^{-1}(h(x) + f(K))$$

Another use for the polynomial of this ideal is to obfuscate arithmetic expressions, namely polynomials in $\mathbb{Z}_{2^n}[x]$. Let us consider a polynomial $g(x)$ to obfuscate, one could write it $(h(x) + 1) \times g(x)$ and expand the product to hide the details of g. It is also possible to add dependencies to other variables of the obfuscated program by using $h(z)$ for some other variable z of the context: adding dependencies increases the resilience of the obfuscation technique by making the analysis of the program more complex.

On the other hand, once one example of the obfuscation scheme has been reversed, its weakness is that the analyst only needs to identify the variable z not influencing the output of the computation, and remove any term in z. Keeping the same input for h and g (e.g. $(h(x) + 1) \times g(x)$) removes this weakness, but quotienting by the ideal will remove $h(x)$.

In term of performance, we conducted some experiments to benchmark our approach. In Figure 2, one can see the ratio between execution time of an obfuscated polynomial modulo 2^{64} and execution time of an unobfuscated one in function of the degree of h. We conducted our test on numerous polynomials $h \in I$ drawn uniformly. We chose polynomials g in $\mathbb{Z}_{2^n}[X]$, drawn uniformly as well. As one can see, this ratio is quite small, it never exceeds 10 even for polynomial h of degree up to 64. All in all, we think this approach can make analysis more difficult with a negligible impact on the efficiency of the program.

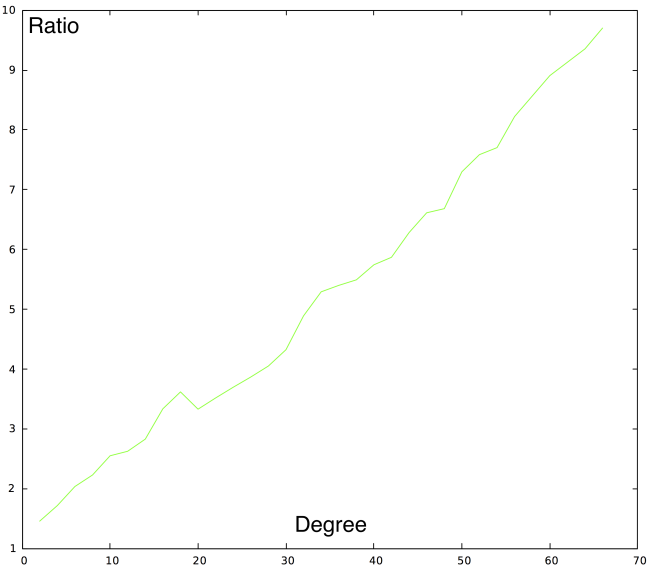

Figure 2: Obfuscation overhead ratio depending on the degree of h.

4. CONCLUSION

In this paper, we have investigated two methods for inverting binary permutation polynomials. These methods allow us to expand the set of permutation polynomials usable for obfuscation purposes. Increasing the variety of those polynomials makes the elaboration of a simplification method harder. Moreover, we introduced some notions (zero ideal) that can be used to improve obfuscation methods using permutation polynomials.

Future works will regard the application of our result to polynomial permutation modulo p^n with $p > 2$ as well as multivariate permutation polynomials. In particular, this would require to adapt the zero ideal formula to different rings.

5. REFERENCES

[1] F. Biondi, S. Josse, A. Legay, and T. Sirvent. Effectiveness of Synthesis in Concolic Deobfuscation. Preprint, Dec. 2015.

[2] A. Bostan. Algorithmes rapides pour les polynômes, séries formelles et matrices. In *Actes des Journées Nationales de Calcul Formel*, Les cours du CIRM, tome 1, numéro 2, pages 75–262, Luminy, France, 2010.

[3] S. Chow, P. Eisen, H. Johnson, and P. C. V. Oorschot. White-Box Cryptography and an AES Implementation. In *Proceedings of the Ninth Workshop on Selected Areas in Cryptography (SAC 2002*, pages 250–270. Springer-Verlag, 2002.

[4] C. Collberg, C. Thomborson, and D. Low. A Taxonomy of Obfuscating Transformations. Technical Report 148, University of Auckland, 1997.

[5] J. Dumas. On newton-raphson iteration for multiplicative inverses modulo prime powers. *IEEE Trans. Computers*, 63(8):2106–2109, 2014.

[6] J. Gomez-Calderon and G. L. Mullen. Galois rings and algebraic cryptography. *Acta Arith.*, 59(4):317–328, 1991.

[7] I. Gupta, L. Narain, and C. V. Madhavan. Cryptological applications of permutation polynomials. *Electronic Notes in Discrete Mathematics*, 15:91 –, 2003. Proceedings of the R.C. Bose Centenary Symposium on Discrete Mathematics and Applications.

[8] S. Hong, X. Qin, and W. Zhao. Necessary conditions for reversed dickson polynomials of the second kind to be permutational. *Finite Fields and Their Applications*, 37:54–71, 2016.

[9] J. Lahtonen, J. Ryu, and E. Suvitie. On the degree of the inverse of quadratic permutation polynomial interleavers. *IEEE Trans. Inform. Theory*, 58(6):3925–3932, 2012.

[10] R. Lidl and W. B. Müller. *Permutation Polynomials in RSA-Cryptosystems*, pages 293–301. Springer US, Boston, MA, 1984.

[11] R. Lidl and W. B. Müller. Permutation polynomials in RSA-cryptosystems. In *Advances in cryptology (Santa Barbara, Calif., 1983)*, pages 293–301. Plenum, New York, 1984.

[12] G. L. Mullen. Permutation polynomials over finite fields. In *Finite fields, coding theory, and advances in communications and computing (Las Vegas, NV, 1991)*, volume 141 of *Lecture Notes in Pure and Appl. Math.*, pages 131–151. Dekker, New York, 1993.

[13] G. L. Mullen and H. Niederreiter. The structure of a group of permutation polynomials. *J. Austral. Math. Soc. Ser. A*, 38(2):164–170, 1985.

[14] W. B. Müller and W. Nöbauer. Some remarks on public-key cryptosystems. *Studia Sci. Math. Hungar.*, 16(1-2):71–76, 1981.

[15] L. Qu and C. Ding. Dickson polynomials of the second kind that permute z_m. *SIAM J. Discrete Math.*, 28(2):722–735, 2014.

[16] Ronald L. Rivest. Permutation Polynomials Modulo 2w. *Finite Fields and Their Applications*, 7(2):287–292, Apr. 2001.

[17] Z. S. Smile Markovski, Danilo Gilgoroski. Polynomial functions on the units of z_{2^n}. *Quasigroups and Related Systems*, 18:59–82, 2010.

[18] J. Sun and O. Y. Takeshita. Interleavers for turbo codes using permutation polynomials over integer rings. *IEEE Trans. Inform. Theory*, 51(1):101–119, 2005.

[19] O. Y. Takeshita. Permutation polynomial interleavers: an algebraic-geometric perspective. *IEEE Trans. Inform. Theory*, 53(6):2116–2132, 2007.

[20] Z. Wan. *Lectures on Finite fields and Galois rings*. World scientific, New Jersey, River Edge, 2003.

[21] L. Wang. On permutation polynomials. *Finite Fields and Their Applications*, 8(3):311 – 322, 2002.

[22] Y. Zhou, A. Main, Y. X. Gu, and H. Johnson. Information Hiding in Software with Mixed Boolean-Arithmetic Transforms. In *8th International Workshop in Information Security Applications (WISA'07)*, pages 61–75, 2007.

[23] R. Zippel. Interpolating polynomials from their values. *J. Symb. Comput.*, 9(3):375–403, 1990.