Use of Simple Arithmetic Operations to Construct Efficiently Implementable Boolean functions Possessing High Nonlinearity and Good Resistance to Algebraic Attacks

Claude Carlet[1,2] and Palash Sarkar*[3]

[1]LAGA Laboratory, University of Paris 8, 93526 Saint-Denis, France
[2]University of Bergen, Norway
[3]Indian Statistical Institute, 203, B.T. Road, Kolkata, India 700108
Emails: claude.carlet@gmail.com, palash@isical.ac.in

January 12, 2025

Abstract

We describe a new class of Boolean functions which provide the presently best known trade-off between low computational complexity, nonlinearity and (fast) algebraic immunity. In particular, for $n \leq 20$, we show that there are functions in the family achieving a combination of nonlinearity and (fast) algebraic immunity which is superior to what is achieved by any other efficiently implementable function. The main novelty of our approach is to apply a judicious combination of simple integer and binary field arithmetic to Boolean function construction.
Keywords: Boolean function, nonlinearity, algebraic immunity, efficient implementation.

1 Introduction

The nonlinear filter model is a several decades old model for stream ciphers. This model consists of two components, namely a linear feedback shift register (LFSR) and a Boolean function which is applied to a subset of the bits of the LFSR. The sequence of outputs of the Boolean function on the successive states of the LFSR constitutes the keystream produced by the stream cipher.

The mathematical challenge for a Boolean function to be used in the filter model of stream ciphers is the following. Construct a large family (if possible an infinite one) of Boolean functions all of which are balanced and achieve a good combination of high nonlinearity and high algebraic resistance and further are efficient to implement. In [7], this design challenge was referred to as "the big single-output Boolean problem" (similar, in the domain of Boolean functions for stream ciphers, to the "big APN problem" in the domain of vectorial functions). In more concrete terms, Section 3.1.5 of [6] suggests that to resist algebraic attacks the number of variables should be at least 13, and further goes on to recommend that in practice "the number of variables will have to be near 20" which then creates a "problem of efficiency of the stream cipher."

There are several known constructions of families of Boolean functions which achieve some, but not all of the above properties. We discuss these families in details in Section 3. For the present, we

*Corresponding author.

briefly mention some of these families. The Carlet-Feng (CF) functions [8] are balanced, achieve optimal algebraic immunity (and also almost optimal fast algebraic immunity) and high nonlinearity, but are not efficient to implement (see [5] for a gate count estimate of CF functions). The hidden weight bit (HWB) function [4] is very efficient to implement and in [27] it was shown that the HWB function has good algebraic immunity, but the nonlinearity is too low. Subsequently, a sequence of works [28, 7, 22, 23] have generalised the HWB function to improve the nonlinearity while retaining the properties of good algebraic immunity and being efficient to implement. The trade-offs achieved by these works are not completely satisfactory.

In this paper, we revisit the above mentioned mathematical challenge for Boolean functions. We describe a family of functions as a solution to the problem. The functions are based on the HWB function. To improve the nonlinearity, we introduce post-processing and pre-processing steps. For the post-processing step, we first extend the HWB function to a vectorial function by extracting a few bits and then apply a highly nonlinear function to these bits. The number of extracted bits is small (in fact, a constant) and so it is feasible to apply a highly nonlinear function to these bits without affecting the efficiency of implementation. For the pre-processing step, we design a novel bijection from n-bit strings to n-bit strings. The bijection is constructed by a judicious combination of simple integer and binary field arithmetic. To the best of our knowledge, no previous work reported construction of Boolean functions based on a combination of integer and binary field arithmetic. Since all operations that we use are simple and efficient, the overall construction is also quite efficient.

The net effect of applying both the pre and post processing steps is a significant improvement of both nonlinearity and algebraic resistance over HWB without compromising on the issue of efficient implementation. Our experimental results show that for all $n \leq 20$, both nonlinearity and algebraic resistance of suitably chosen n-variable functions from the new family are substantially better than the corresponding values of n-variable functions from all previously known families [28, 7, 22, 23] that are efficient to implement. So our construction provides good solutions to the concrete problem highlighted in Section 3.1.5 of [6].

The paper is organised as follows. In Section 2 we describe the preliminaries. The relevant previous constructions are discussed in Section 3. The family of functions is described in Section 4. Section 5 concludes the paper.

2 Preliminaries

In this section, we introduce the notation and provide the definitions of the properties of Boolean functions that we consider in this work. For further details and more elaborate discussion on these issues we refer to [6].

The cardinality of a finite set S will be denoted by $\#S$. For a prime power q, $\mathbb{F}_q$ denotes the finite field of order q consisting of q elements. In particular, $\mathbb{F}_2$ denotes the finite field of two elements. For a positive integer n, $\mathbb{F}_2^n$ is the vector space of dimension n over $\mathbb{F}_2$. The addition operation over both $\mathbb{F}_2$ and $\mathbb{F}_2^n$ will be denoted by $\oplus$. Elements of $\mathbb{F}_2^n$ are considered to be n-bit binary strings.

For an n-bit binary string $\mathbf{x} = (x_1, \ldots, x_n)$, $\mathsf{wt}(\mathbf{x}) = \#\{i : x_i = 1\}$. Given two strings $\mathbf{x}$ and $\mathbf{y}$ of the same length, the distance between them, denoted $d(\mathbf{x}, \mathbf{y})$, is defined to be the number of places where $\mathbf{x}$ and $\mathbf{y}$ are unequal. Given $\mathbf{x} = (x_1, \ldots, x_n), \mathbf{y} = (y_1, \ldots, y_n) \in \mathbb{F}_2^n$, their inner product $\langle \mathbf{x}, \mathbf{y} \rangle$ is defined to be $\langle \mathbf{x}, \mathbf{y} \rangle = x_1 y_1 \oplus \cdots \oplus x_n y_n$. For an n-bit string $\mathbf{x}$, by $\mathsf{int}(\mathbf{x})$ we denote the unique integer $i \in \{0, \ldots, 2^n - 1\}$ whose n-bit binary representation is $\mathbf{x}$. Conversely, for $0 \leq i \leq 2^n - 1$, by $\mathsf{bin}_n(i)$ we denote the binary string given by the n-bit binary representation of i. The n-bit all-zero and all-one strings will be denoted as $\mathbf{0}_n$ and $\mathbf{1}_n$ respectively. For $\mathbf{x} = (x_1, \ldots, x_n), \mathbf{y} = (y_1, \ldots, y_n) \in \mathbb{F}_2^n$, we say

$\mathbf{x} \leq \mathbf{y}$ if $x_i \leq y_i$ for $i = 1, \ldots, n$.

An n-variable Boolean function f is a map $f : \mathbb{F}_2^n \to \mathbb{F}_2$. By $\mathsf{supp}(f)$ we denote the set $\{\mathbf{x} \in \mathbb{F}_2^n : f(\mathbf{x}) = 1\}$. The *weight* of f, denoted $\mathsf{wt}(f)$, is the size of $\mathsf{supp}(f)$, i.e. $\mathsf{wt}(f) = \#\mathsf{supp}(f)$. An n-variable function f is said to be *balanced* if $\mathsf{wt}(f) = 2^{n-1}$. An n-variable function f is uniquely represented by a binary string $f_0 \cdots f_{2^n-1}$, where for $i \in \{0, \ldots, 2^n - 1\}$, $f_i = f(\mathsf{bin}_n(i))$. Such a string representation of f is also called the *truth table representation* of f.

Algebraic normal form. An n-variable function f can be written as a multivariate polynomial in $\mathbb{F}_2[X_1, \ldots, X_n]/(X_1^2 \oplus X_1, \ldots, X_n^2 \oplus X_n)$ as follows. Let $\mathbf{X} = (X_1, \ldots, X_n)$. Then $f(X_1, \ldots, X_n) = \bigoplus_{\boldsymbol{\alpha} \in \mathbb{F}_2^n} a_{\boldsymbol{\alpha}} \mathbf{X}^{\boldsymbol{\alpha}}$, where $a_{\boldsymbol{\alpha}} \in \mathbb{F}_2$, and for $\boldsymbol{\alpha} = (\alpha_1, \ldots, \alpha_n)$, $\mathbf{X}^{\boldsymbol{\alpha}} = X_1^{\alpha_1} \cdots X_n^{\alpha_n}$. This representation is called the *algebraic normal form (ANF) representation* of f. The algebraic degree (or simply the degree) of f is defined to be $\mathsf{deg}(f) = \max\{\mathsf{wt}(\boldsymbol{\alpha}) : a_{\boldsymbol{\alpha}} = 1\}$. Functions of degree at most 1 are said to be affine functions. Affine functions having $a_{\mathbf{0}_n} = 0$ are said to be linear functions. It is known that if f is balanced, then $\mathsf{deg}(f) \leq n - 1$. A balanced function f with $\mathsf{deg}(f) = n - 1$ is said to have optimal degree.

The following equations relate the coefficients $a_{\boldsymbol{\alpha}}$ in the ANF of f to the truth table representation of f (see for example Pages 49 and 50 of [6]). For $\mathbf{x}, \boldsymbol{\alpha} \in \mathbb{F}_2^n$,

$$f(\mathbf{x}) = \bigoplus_{\boldsymbol{\beta} \leq \mathbf{x}} a_{\boldsymbol{\beta}} \quad \text{and} \quad a_{\boldsymbol{\alpha}} = \bigoplus_{\mathbf{z} \leq \boldsymbol{\alpha}} f(\mathbf{z}). \tag{1}$$

Nonlinearity and Walsh transform. For two n-variable functions f and g, the distance between them is denoted by $d(f, g)$ and is defined to be the distance between their truth table representations. The *nonlinearity* of an n-variable function f is denoted by $\mathsf{nl}(f)$ and is defined to be $\mathsf{nl}(f) = \min d(f, g)$, where the minimum is over all n-variable affine functions g.

The Walsh transform of an n-variable function f is a map $W_f : \mathbb{F}_2^n \to \mathbb{Z}$, where for $\boldsymbol{\alpha} \in \mathbb{F}_2^n$, $W_f(\boldsymbol{\alpha}) = \sum_{\mathbf{x} \in \mathbb{F}_2^n} (-1)^{f(\mathbf{x}) \oplus \langle \boldsymbol{\alpha}, \mathbf{x} \rangle}$. The function f is balanced if and only if $W_f(\mathbf{0}_n) = 0$. The nonlinearity of a function f is given by its Walsh transform as follows: $\mathsf{nl}(f) = 2^{n-1} - \frac{1}{2} \max_{\boldsymbol{\alpha} \in \mathbb{F}_2^n} |W_f(\boldsymbol{\alpha})|$.

A function f such that $W_f(\boldsymbol{\alpha}) = \pm 2^{n/2}$ for all $\boldsymbol{\alpha} \in \mathbb{F}_2^n$ is said to be a bent function [26]. Clearly such functions can exist only if n is even. The nonlinearity of an n-variable bent function is $2^{n-1} - 2^{n/2-1}$ and this is the maximum nonlinearity that can be attained by n-variable functions. By $\mathsf{LLB}(f)$ we will denote the *logarithm* (to base two) of the linear bias of the function f which is defined in the following manner: $\mathsf{LLB}(f) = \log_2 (1/2 - \mathsf{nl}(f)/2^n)$.

For a positive integer n, the covering radius bound CRB_n is defined to be $\mathsf{CRB}_n = 2^{n-1} - \lfloor 2^{n/2-1} \rfloor$. For an n-variable function f, we have $\mathsf{nl}(f) \leq \mathsf{CRB}_n$, where equality holds for bent functions. Let $\mathsf{LCRB}_n = \log_2 (1/2 - \mathsf{CRB}_n/2^n)$.

Algebraic resistance. The *algebraic immunity* of a function f, denoted by $\mathsf{AI}(f)$, is defined in the following manner [11, 24]: $\mathsf{AI}(f) = \min_{g \neq 0} \{\mathsf{deg}(g) : \text{ either } gf = 0, \text{ or } g(f \oplus 1) = 0\}$. For an n-variable function f, it is known [11] that $\mathsf{AI}(f) \leq \lceil n/2 \rceil$. So a function f has optimal AI if $\mathsf{AI}(f) = \lceil n/2 \rceil$. It was proved in [12] that a random n-variable function almost surely has AI at least $\lfloor n/2 - \log n \rfloor$.

Algebraic immunity quantifies the resistance of a function to algebraic attacks. In practice, it is also required to provide resistance to fast algebraic attack (FAA) [10]. Given an n-variable function f, let g be an n-variable function of degree e such that gf has degree d. If for small e, d is not too high then the function f is susceptible to an FAA. It is known [10] that for $e + d \geq n$, there exists functions g and h with $\mathsf{deg}(g) = e$ and $\mathsf{deg}(h) \leq d$ such that $gf = h$. Based on this observation, we provide the

following definition. For each $e \in \{1, \ldots, \mathsf{AI}(f) - 1\}$, let $d \leq n - 1 - e$ be the maximum integer such that there do not exist n-variable functions g and h with $\mathsf{deg}(g) = e$, $\mathsf{deg}(h) = d$ and $gf = h$. We call the list of all such pairs (e, d) as the *FAA-profile* of f.

A combined measure of resistance offered by a function f to both algebraic and fast algebraic attacks is defined to be *fast algebraic immunity (FAI)*:

$$\mathsf{FAI}(f) = \min\left(2\mathsf{AI}(f), \min_{g \neq 0}\{\mathsf{deg}(g) + \mathsf{deg}(fg) : 1 \leq \mathsf{deg}(g) < \mathsf{AI}(f)\}\right).$$

We have $\mathsf{FAI}(f) = \min(2\mathsf{AI}(f), \min\{e + d + 1\})$, where the second minimum is taken over all pairs (e, d) in the FAA-profile of f. Further, it is clear that for any function f, $1 + \mathsf{AI}(f) \leq \mathsf{FAI}(f) \leq 2\,\mathsf{AI}(f)$.

If $\mathsf{AI}(f) = \lceil n/2 \rceil$ and for each pair (e, d) in the FAA-profile of f, $e + d = n - 1$, then f is said to have perfect algebraic immunity (PAI) [21]. We introduce a relaxed version of the notion of optimal AI and PAI. We say that a function f has almost optimal AI if $\mathsf{AI}(f) \geq \lfloor n/2 \rfloor$ and f is said to have almost perfect FAA-profile if for each pair (e, d) in the FAA-profile of f, $e + d \geq n - 2$.

Remark 1 *There are known algorithms [1, 14, 13] for computing AI and FAI. The complexities of these algorithms are very high. For computing algebraic immunities we used the Boolean function library*[1] *of the SageMath software. On the computer resources available to us, it was not possible to do any computation related to algebraic resistance for functions on more than 20 variables.*

Implementation efficiency. The complexity of implementing a Boolean function is measured with respect to space and time. For example, a truth table representation of an n-variable Boolean function requires 2^n bits and can be computed at a single point in $O(1)$ time (assuming that a look-up into the truth table requires constant time which need not be true if n is large). More generally, we say that a Boolean function has an (S, T)-implementation if it can be implemented using S bits/gates and can be computed using T bit operations. In an asymptotic sense, we say that an infinite family of Boolean functions has an efficient implementation if any n-variable function in the family has an (S, T)-implementation where both S and T are bounded above by polynomials in n. From a concrete point of view, on the other hand, we will be interested in the concrete details of the implementation in terms of the actual number bits required to represent the function and the actual number of basic operations required to compute it.

Vectorial functions. For positive integers n and m, an (n, m)-vectorial Boolean function (also called an S-box) F is a map $F : \mathbb{F}_2^n \to \mathbb{F}_2^m$. If $m = 1$, then we get back a Boolean function. An (n, m)-vectorial Boolean function F can be written as $F = (f_1, \ldots, f_m)$, where each f_i, $i = 1, \ldots, m$, is an n-variable Boolean function. The f_i's are said to be the coordinate functions of F. For $\boldsymbol{\alpha} = (\alpha_1, \ldots, \alpha_m) \in \mathbb{F}_2^m$, let $F_{\boldsymbol{\alpha}} = \langle \boldsymbol{\alpha}, (f_1, \ldots, f_m) \rangle = \alpha_1 f_1 \oplus \cdots \oplus \alpha_m f_m$. Then $F_{\boldsymbol{\alpha}}$ is an n-variable Boolean function, and the $F_{\boldsymbol{\alpha}}$'s are called the component functions of F. For $n \geq m$, an (n, m)-vectorial function F is said to be balanced if for each $\boldsymbol{\beta} \in \mathbb{F}_2^m$, $\#F^{-1}(\boldsymbol{\beta}) = 2^{n-m}$. Equivalently, it is known that (see e.g. [6]) F is balanced if and only if all non-zero component functions of F are balanced.

Let F be an (n, m)-vectorial Boolean function and g be an m-variable Boolean function. The composition $g \circ F$ is an n-variable Boolean function given by $(g \circ F)(X_1, \ldots, X_n) = g(F(X_1, \ldots, X_n)) = g(f_1, \ldots, f_m)$. The Walsh transform of $f \circ F$ is the following [17]. For $\boldsymbol{\beta} \in \mathbb{F}_2^n$,

$$W_{f \circ F}(\boldsymbol{\beta}) \;=\; \frac{1}{2^m} \sum_{\boldsymbol{\alpha} \in \mathbb{F}_2^m} W_f(\boldsymbol{\alpha}) W_{F_{\boldsymbol{\alpha}}}(\boldsymbol{\beta}). \tag{2}$$

[1] `https://doc.sagemath.org/html/en/reference/cryptography/sage/crypto/boolean_function.html#sage.crypto.boolean_function.BooleanFunction.annihilator`

The following simple result follows from (2).

Proposition 1 *Let n and m be positive integers with $n \geq m$, and let F be a balanced (n, m)-vectorial function. Let f be an m-variable Boolean function. Then $f \circ F$ is balanced if and only if f is balanced.*

3 Relevant Previous Constructions

In this section, we briefly outline some previous relevant constructions.

Carlet-Feng (CF) functions. Any polynomial $a(x) = a_0 \oplus a_1 x \oplus \cdots + a_{n-1}x^{n-1} \in \mathbb{F}_2[x]$ is uniquely determined by the coefficient vector $\mathbf{a} = (a_{n-1}, \ldots, a_0) \in \mathbb{F}_2^n$. So the elements of $\mathbb{F}_2^n$ can be considered to be polynomials in $\mathbb{F}_2[x]$ of degree at most $n-1$. Let $\tau(x)$ be a primitive polynomial of degree n over $\mathbb{F}_2$. An n-variable CF-function is defined by its support which is the following set of polynomials of degrees at most $n-1$:

$$\{0, 1, x \bmod \tau(x), x^2 \bmod \tau(x), \ldots, x^{2^{n-1}-2} \bmod \tau(x)\}.$$

It was shown in [8] that such a Boolean function is balanced, has degree $n-1$ and AI $\lceil n/2 \rceil$. (This class of functions was earlier considered in [15] for showing the tightness of bounds on the algebraic immunity of vectorial functions and the nonlinearity was earlier studied in [3].) Further, it was shown in [21] that when n is one more than a power of two, such functions possess PAI. A lower bound on the nonlinearity of such functions was proved in [8]. For concrete values of n, the actual nonlinearities are much higher than the lower bound. Further, the nonlinearity depends on the choice of the primitive polynomial $\tau(x)$. We computed the nonlinearities of CF functions for certain values of n. The primitive polynomials that we used are given in Appendix A.

A drawback of the CF functions is that these are not very efficient to implement. Evaluating the value of a CF function on a particular input $a(x)$ amounts to computing i such that $a(x) \equiv x^i \bmod p(x)$. This is the discrete logarithm problem in $\mathbb{F}_{2^n}$. A truth table implementation of CF-functions requires $O(2^n)$ bits. Using polynomial space the discrete logarithm problem can be solved in asymptotically sub-exponential time. As a result, CF functions are unsuitable for fast and light weight implementations.

Hidden weight bit (HWB) functions. For $n \geq 1$, let $\mathsf{HWB}_n : \{0,1\}^n \to \{0,1\}$ be the hidden weight bit function [4] defined as follows. For $\mathbf{x} = (x_1, \ldots, x_n) \in \mathbb{F}_2^n$,

$$\mathsf{HWB}_n(\mathbf{x}) \quad = \quad x_{\mathsf{wt}(\mathbf{x})}, \tag{3}$$

where we assume that $x_0 = 0$. The HWB functions are clearly efficiently implementable. Cryptographic properties of HWB functions were studied in [27]. It was shown that the AI of HWB_n is at least $\lfloor n/3 \rfloor + 1$ and for n in the set $\{6, \ldots, 13\}$, the actual AI is either the lower bound or one more than the lower bound. For n in the set $\{6, \ldots, 13\}$, the FAA-profiles were reported in [27] and turned out to be significantly away from the profile of a PAI function.

The nonlinearity of HWB_n was shown to be $2^{n-1} - 2\binom{n-2}{\lceil (n-2)/2 \rceil}$. This value is quite low. So even though HWB functions are efficiently implementable, they do not possess sufficiently high nonlinearity for cryptographic applications. Concatenations of HWB functions have been studied in [28] producing functions with higher nonlinearities than the HWB functions, but still not high enough for use in practical systems.

Binary decision diagrams (BDD) have been used to propose attacks on stream ciphers [19, 20]. A positive feature of HWB functions is that these functions have high BDD complexity [4, 2, 18].

Generalised HWB (GHWB) functions. A generalisation of HWB functions was introduced in [7] with the goal of improving their nonlinearity and algebraic immunity while retaining the efficiency of implementation. The concrete results for $n = 13, 14, 15$ and 16 presented in [7] show that the AI of GHWB is almost optimal and is greater than the AI of HWB. There is also improvement in nonlinearity. This improvement, however, is not substantial and the obtained nonlinearities of GHWB functions are still not good enough for practical applications.

Cyclic weightwise functions. Another generalisation of the HWB function was made in [22]. Let $g_0, \ldots, g_n$ be n-variable functions. Using these $n+1$ functions, an n-variable weightwise function f is constructed as follows: for $\mathbf{x} \in \mathbb{F}_2^n$, $f(\mathbf{x}) = g_w(\mathbf{x})$, where $w = \mathsf{wt}(\mathbf{x})$. The function f is uniquely defined by the sequence of functions $(g_0, \ldots, g_n)$. Note that the function g_w is applied only to strings of weight w. In particular g_0 is applied only to the string $\mathbf{0}_n$.

Since implementing $n+1$ functions may be difficult in practice, the notion of *cyclic weightwise* functions was introduced in [22], where the functions g_i's are defined from a single n-variable function g as follows: $g_0 = g_1 = g$, and for $i \in \{2, \ldots, n\}$, g_i is defined to be $g_i(\mathbf{x}) = g(\mathbf{x} \ggg (i-1))$, where $\ggg$ is the cyclic right shift operator. The resulting function f is called a cyclic weightwise function, which we denote as $f = \mathsf{CW}_n(g)$. Lower bounds on the nonlinearities of $\mathsf{CW}_n(g)$ was obtained in [22] for the case when g is linear and for a particular quadratic function g. For the choice of $g(x_1, \ldots, x_n) = x_1 \oplus \left(\bigoplus_{i=1}^{\lfloor (n-1)/2 \rfloor} x_{2i}x_{2i+1}\right)$, actual nonlinearities, degrees and algebraic immunities of $\mathsf{CW}_n(g)$ were provided in [22]. These functions achieve both the highest nonlinearities and the highest algebraic immunities among all the functions presented in [22]. Cyclic weightwise Boolean functions possessing properties which improve upon the functions reported in [22] were described in [23].

Inverse map. Let $\rho(x) \in \mathbb{F}_2[x]$ be an irreducible polynomial of degree n. Then for any nonzero polynomial $a(x) \in \mathbb{F}_2[x]$ of degree at most $n-1$, there is a polynomial $b(x)$ also of degree at most $n-1$ such that $a(x)b(x) \equiv 1 \bmod \rho(x)$, i.e. $b(x) = a(x)^{-1} \bmod \rho(x)$. As in the case of the CF functions, we identify polynomials in $\mathbb{F}_2[x]$ of degrees at most $n-1$ with the elements of $\mathbb{F}_2^n$. We can then define an (n,n)-vectorial function $\mathsf{inv} : \mathbb{F}_2^n \to \mathbb{F}_2^n$ as follows: $\mathsf{inv}(\mathbf{0}_n) = \mathbf{0}_n$ and for any $a(x) \in \mathbb{F}_2[x]$ of degree at most $n-1$, $\mathsf{inv}(a(x)) = a(x)^{-1} \bmod \rho(x)$. This is the well known inverse map which was introduced to cryptography in [25]. A nonzero component function of inv is an n-variable Boolean function. Such functions are balanced and have degrees equal to $n-1$. Further, it is known [25, 9] that the nonlinearity of any non-zero component function is at least $2^{n-1} - 2^{n/2}$. The AI of such a function, however, is not good. It was shown in [16], that the AI is equal to $\lceil 2\sqrt{n} \rceil - 2$. From an implementation point of view, computing $a(x)^{-1} \bmod \rho(x)$ requires about $O(n^3)$ bit operations. For values of n which are relevant to the nonlinear filter model, it is not possible to make a lightweight and fast implementation of the inverse function.

4 Construction of Interval λ-HWB Functions

The HWB function is efficient to implement. Its major drawback, however, is its low nonlinearity. We provide two methods to improve the nonlinearity. The first using a post-processing and the second using a pre-processing.

4.1 Post-processing

One possible way to improve the cryptographic properties of the HWB function is to perform some post-processing of its output. Note that the HWB function produces a single bit of output. It is not meaningful to perform any post-processing on a single bit. So as a first step, we consider a vectorial version of the HWB function which produces more than one bit of output. Let r be the number of bits that are to be produced. The question then is how should these r bits be extracted. On an input $\mathbf{x} = (x_1, \ldots, x_n)$, the HWB function produces as output x_i, where i is the weight of $(x_1, \ldots, x_n)$. To extract r bits, we extract a window of r bits of $\mathbf{x}$ centered at x_i. This creates a difficulty if indices of the window fall outside the range $\{1, \ldots, n\}$. There are two ways to tackle this situation, namely the null and the cyclic boundary conditions. Let $\mathbf{x} = (x_1, \ldots, x_n)$ and suppose i is an integer which is not in $\{1, \ldots, n\}$. Under the *null boundary condition*, we define x_i to be 0, while under the *cyclic boundary condition*, we define x_i to be equal to x_j, where j is the unique integer in $\{1, \ldots, n\}$ such that $i \equiv j \bmod n$. From experimental results we find that the nonlinearities of the functions obtained using the cyclic boundary condition are more than the nonlinearities of the functions obtained using the null boundary condition. In view of this, we do not formally introduce the construction using the null boundary condition.

Given positive integers n and r with $r \leq n$, we define an (n, r)-vectorial function $\mathsf{HWB}_{n,r}$ as follows. For $\mathbf{x} \in \mathbb{F}_2^n$, let $w = \mathsf{wt}(\mathsf{x})$. Let $\ell = w - \lfloor r/2 \rfloor$ if r is odd and let $\ell = w - r/2 + 1$ if r is even. Then

$$\mathsf{HWB}_{n,r} \quad = \quad (x_\ell, x_{\ell+1}, \ldots, x_{\ell+r-1}) \quad \text{with cyclic boundary condition.} \tag{4}$$

Note that $\mathsf{HWB}_{n,1} = \mathsf{HWB}_n$. We have the following result regarding the balancedness of $\mathsf{HWB}_{n,r}$.

Proposition 2 *Let n and r be positive integers with $1 \leq r \leq n$. Then $\mathsf{HWB}_{n,r}$ is balanced.*

Proof: Let $\boldsymbol{\beta} \in \mathbb{F}_2^r$. We count the number of preimages of $\boldsymbol{\beta}$ under $\mathsf{HWB}_{n,r}$. For $\mathbf{x} \in \mathbb{F}_2^n$ with $w = \mathsf{wt}(\mathbf{x})$, suppose $\mathsf{HWB}_{n,r}(\mathbf{x}) = \boldsymbol{\beta}$. Then $(x_\ell, x_{\ell+1}, \ldots, x_{\ell+r-1}) = \boldsymbol{\beta}$, where $\ell = w - \lfloor r/2 \rfloor$ if r is odd and let $\ell = w - r/2 + 1$ if r is even. Let $k = \mathsf{wt}(\boldsymbol{\beta})$. Then $\#\{i \in \{1, \ldots, n\} \setminus \{\ell, \ldots, \ell + r - 1\} : x_i = 1\} = w - k$. So the number of $\mathbf{x}$'s such that $\mathsf{wt}(\mathbf{x}) = w$ and $(x_\ell, x_{\ell+1}, \ldots, x_{\ell+r-1}) = \boldsymbol{\beta}$ is equal to $\binom{n-r}{w-k}$. Consequently, the number of preimages of $\boldsymbol{\beta}$ under $\mathsf{HWB}_{n,r}$ is $\sum_{w=0}^{n} \binom{n-r}{w-k} = 2^{n-r}$, since $n - k \geq n - r$. □

Let λ be an r-variable Boolean function. We define an n-variable Boolean function λ-$\mathsf{HWB}_{n,r}$ in the following manner.

$$\lambda\text{-}\mathsf{HWB}_{n,r} \quad = \quad \lambda \circ \mathsf{HWB}_{n,r}. \tag{5}$$

So for $\mathbf{x} \in \mathbb{F}_2^n$, λ-$\mathsf{HWB}_{n,r}(\mathbf{x}) = \lambda(\mathsf{HWB}_{n,r}(\mathbf{x}))$.

Proposition 3 *Let λ be an r-variable Boolean function. Then λ-$\mathsf{HWB}_{n,r}$ is balanced if and only if λ is balanced.*

Proof: Proposition 2 shows that $\mathsf{HWB}_{n,r}$ is a balanced (n, r)-vectorial function. From Proposition 1 we have that the composition of a balanced (n, r)-vectorial function and an r-variable Boolean function λ is balanced if and only if λ is balanced. □

Let $\pi_1, \ldots, \pi_n$ be permutations of $\{1, \ldots, n\}$ and for $i = 1, \ldots, n$, let $P_i : \mathbb{F}_2^n \to \mathbb{F}_2^n$ be defined as $P_i(x_1, \ldots, x_n) = (x_{\pi(1)}, \ldots, x_{\pi(n)})$. Let g be an n-variable Boolean function and f be another n-variable Boolean function defined using g and $P_1, \ldots, P_n$ in the following manner: $f(\mathbf{x}) = g(P_w(\mathbf{x}))$, where $w = \mathsf{wt}(\mathbf{x})$. Proposition 4 of [22] shows that f is balanced if and only if g is balanced. Proposition 3

can be seen as a corollary of Proposition 4 of [22]. On the other hand, Proposition 4 of [22] itself can be seen as a corollary of Proposition 1 in the following manner. Given $P_1, \ldots, P_n$, define a bijection $S : \mathbb{F}_2^n \to \mathbb{F}_2^n$ by $S(\mathbf{x}) = P_w(\mathbf{x})$, where $w = \mathsf{wt}(\mathbf{x})$. Then $f = g \circ S$, and by Proposition 1, f is balanced if and only if g is balanced.

Proposition 4 *For any r-variable function λ, $(1 \oplus \lambda)$-$\mathsf{HWB}_{n,r} = 1 \oplus \lambda$-$\mathsf{HWB}_{n,r}$. More generally, for any invertible affine transformation $A : \mathbb{F}_2^r \to \mathbb{F}_2^r$, $\mathsf{nl}(\lambda \circ A \circ \mathsf{HWB}_{n,r}) = \mathsf{nl}(\lambda \circ \mathsf{HWB}_{n,r})$.*

The nonlinearity of λ-$\mathsf{HWB}_{n,r}$ is determined by the Walsh transform of λ-$\mathsf{HWB}_{n,r}$. In principle, using (2), the Walsh transform of λ-$\mathsf{HWB}_{n,r}$ can be determined from the Walsh transforms of λ and $\mathsf{HWB}_{n,r}$. So in principle, using (2), the nonlinearity of λ-$\mathsf{HWB}_{n,r}$ can be determined from the Walsh transforms of λ and $\mathsf{HWB}_{n,r}$. The form of (2), however, does not provide any easy method to identify conditions on the Walsh transform of λ such that the nonlinearity of λ-$\mathsf{HWB}_{n,r}$ is high. Faced with this scenario, we decided to search for choices of λ to determine the set of λ's having the highest possible nonlinearity. Since we are interested in balanced functions, using Proposition 3, we focused only on balanced λ's. Algorithm 1 describes our search strategy. It takes as input n, r and a list $\mathcal{S}$ of r-variable balanced functions and produces as output a set of functions λ such that the corresponding λ-$\mathsf{HWB}_{n,r}$ function has algebraic degree $n-1$ and as such, has maximal nonlinearity among all visited functions.

Algorithm 1: The search procedure for λ-$\mathsf{HWB}_{n,r}$.

Input: n, r and $\mathcal{S}$, where $\mathcal{S}$ is a subset of the set of all balanced r-variable functions

Output: A list $\mathcal{L}$ of r-variable functions such that for any $\lambda \in \mathcal{L}$, λ-$\mathsf{HWB}_{n,r}$ is balanced, has degree $n-1$ and $\lambda \in \mathrm{argmax}_{\mu \in \mathcal{S}}\, \mathsf{nl}(\mu\text{-}\mathsf{HWB}_{n,r})$

```
1  maxnl ← 0; L ← ∅
2  for λ ∈ S do
3      let f = λ-HWB_{n,r}
4      compute nl(f) and deg(f)
5      if deg(f) = n − 1 and maxnl < nl(f) then
6          maxnl ← nl(f); L ← {λ}
7      else
8          if deg(f) = n − 1 and maxnl = nl(f) then
9              L ← L ∪ {λ}
10 return L
```

Proposition 5 *For positive integers n and r with $1 \le r \le n$ and $\mathcal{S}$ a subset of balanced r-variable functions, let $\mathcal{L}$ be returned by Algorithm 1 on input n, r and $\mathcal{S}$. Then for any $\lambda \in \mathcal{L}$, λ-$\mathsf{HWB}_{n,r}$ is a balanced n-variable function having degree $n-1$. The time taken by Algorithm 1 is $O(\#\mathcal{S}\, n 2^n)$.*

Proof: Suppose $\mathcal{L}$ is the output of Algorithm 1. From Proposition 3 it follows that any $\lambda \in \mathcal{L}$ is balanced. From the algorithm, it directly follows that the degree is $n-1$.

For each λ in $\mathcal{S}$, the algorithm constructs the n-variable function λ-$\mathsf{HWB}_{n,r}$ and computes its nonlinearity and degree. So the time for each λ is $O(n2^n)$, and the total time is $O(\#\mathcal{S}\, n 2^n)$. □

If $\mathcal{S}$ is the set of all balanced r-variable functions, then the time required by Algorithm 1 is $O(\binom{2^r}{2^{r-1}} n 2^n)$. For $r = 2, 3$ and 4, and for $n = 13, \ldots, 20$, we have run Algorithm 1 with $\mathcal{S}$ to be the set of all r-variable balanced Boolean functions. (Note that for $r = 2$ the only balanced functions

are the non-constant affine functions.) A summary of our observations of these executions of Algorithm 1 are as follows.

1. For $n = 13, \ldots, 20$ and $r = 2$, for the λ's produced by Algorithm 1, the nonlinearities of λ-$\mathsf{HWB}_{n,2}$ are equal to the nonlinearities of the corresponding HWB_n. This though is not true in general. For example, for $n = 8$, taking $\lambda(X_1, X_2) = X_1 \oplus X_2$, the nonlinearity of λ-$\mathsf{HWB}_{8,2}$ is 92, while the nonlinearity of HWB_8 is 88.
2. For a fixed value of n, the nonlinearity of λ-$\mathsf{HWB}_{n,r}$ with λ produced by Algorithm 1 increases with the value of r.

For $r = 5$, the number of balanced r-variable functions is equal to $\binom{32}{16} \approx 2^{29.163}$. So if in Algorithm 1 we put $\mathcal{S}$ to be the set of all 5-variable balanced functions, then the time taken will be proportional to $n2^{n+29.163}$. On the computing resources available to use, for $n = 13$ this computation is barely feasible while it is out of our reach for $n = 20$. Accordingly, we decided to take $\mathcal{S}$ to be a proper subset of 5-variable balanced functions. The first condition that we imposed is to consider only functions having degree 4. This, however, does not significantly reduce the size of $\mathcal{S}$. Next we imposed the condition that along with degree 4, the functions should have nonlinearity 12, which is the maximum possible nonlinearity among all 5-variable balanced functions. This condition is motivated by our finding that for $r = 3$ and $r = 4$, the λ's which are returned by Algorithm 1 have the maximum possible nonlinearity among all balanced r-variable functions. The number of 5-variable functions having degree 4 and nonlinearity 12 is $1666560 \approx 2^{20.668}$. With $\#\mathcal{S} = 1666560$, it becomes feasible to run Algorithm 1 for $n = 13, \ldots, 20$ on our computers. The nonlinearities that are obtained are higher than the nonlinearities obtained for $r = 2, 3$ and 4. The following proposition states the results that we obtained.

Proposition 6 *Let $r = 5$. For $n = 13, \ldots, 20$, the maximum nonlinearities, along with the corresponding λ's and $1 \oplus \lambda$'s, achieved by balanced λ-$\mathsf{HWB}_{n,r}$ functions having degree $n - 1$, where λ runs over all 5-variable balanced functions having degree 4 and nonlinearity 12, are as follows.*

- $n = 13$, $\mathsf{nl}(\lambda\text{-}\mathsf{HWB}_{n,r}) = 3780$, *where* $\lambda, 1 \oplus \lambda \in \{\lambda_{5,1}, \lambda_{5,2}\}$.
- $n = 14$, $\mathsf{nl}(\lambda\text{-}\mathsf{HWB}_{n,r}) = 7572$, *where* $\lambda, 1 \oplus \lambda \in \{\lambda_{5,3}, \lambda_{5,4}\}$.
- $n = 15$, $\mathsf{nl}(\lambda\text{-}\mathsf{HWB}_{n,r}) = 15236$, *where* $\lambda, 1 \oplus \lambda \in \{\lambda_{5,1}, \lambda_{5,2}\}$.
- $n = 16$, $\mathsf{nl}(\lambda\text{-}\mathsf{HWB}_{n,r}) = 30526$, *where* $\lambda, 1 \oplus \lambda \in \{\lambda_{5,5}, \lambda_{5,6}\}$.
- $n = 17$, $\mathsf{nl}(\lambda\text{-}\mathsf{HWB}_{n,r}) = 61284$, *where* $\lambda, 1 \oplus \lambda \in \{\lambda_{5,1}, \lambda_{5,2}\}$.
- $n = 18$, $\mathsf{nl}(\lambda\text{-}\mathsf{HWB}_{n,r}) = 122758$, *where* $\lambda, 1 \oplus \lambda \in \{\lambda_{5,7}, \lambda_{5,8}\}$.
- $n = 19$, $\mathsf{nl}(\lambda\text{-}\mathsf{HWB}_{n,r}) = 246368$, *where* $\lambda, 1 \oplus \lambda \in \{\lambda_{5,9}, \lambda_{5,10}\}$.
- $n = 20$, $\mathsf{nl}(\lambda\text{-}\mathsf{HWB}_{n,r}) = 493476$, *where* $\lambda, 1 \oplus \lambda \in \{\lambda_{5,11}, \lambda_{5,12}\}$.

In the above, $\lambda_{5,i}$, $i = 1, \ldots, 12$, given by their 32-bit string representations are the following. (The ANFs of these functions are given in Appendix B.)

$$\lambda_{5,1} = 10111111010100010001101000001110, \quad \lambda_{5,2} = 10101000011010110100111001001110$$
$$\lambda_{5,3} = 10010011011000111011010111010000, \quad \lambda_{5,4} = 10000100110100111010100111110100$$
$$\lambda_{5,5} = 10101011011010110001101100011000, \quad \lambda_{5,6} = 10000101111110111000101000001110$$
$$\lambda_{5,7} = 11100001010111110000101001001110, \quad \lambda_{5,8} = 10101011001100100001111000011110$$
$$\lambda_{5,9} = 10001001010111110010110011101000, \quad \lambda_{5,10} = 10101011100100111011010100000110$$
$$\lambda_{5,11} = 01100010101011111110000111000100, \quad \lambda_{5,12} = 01100000110010110101011101001110$$

Efficiency of computing λ-HWB$_{n,5}$. The requirement is to compute the weight of one n-bit string and to compute the output of a 5-variable function. For $n \leq 20$, this is very efficient to do in both hardware and software.

Relation to Cyclic Weightwise Functions. Let $\ell_1 = 1 - \lfloor r/2 \rfloor$ if r is odd and let $\ell_1 = 1 - r/2 + 1$ if r is even. Define an n-variable function g, where for $(x_1, \ldots, x_n) \in \mathbb{F}_2^n$, $g(x_1, \ldots, x_n) = \lambda(x_\ell, x_{\ell+1}, \ldots, x_{\ell+r-1})$ with cyclic boundary condition. Let $g_0, g_1, \ldots, g_n$ be n-variable functions where $g_0 = g_1 = g$ and for $i \in \{2, \ldots, n\}$, $g_i(x_1, \ldots, x_n) = g((x_1, \ldots, x_n) \lll (i-1))$, where $\lll$ is the cyclic left shift operator. Then λ-HWB$_{n,r}$ is a weightwise function defined by the sequence of functions $(g_0, g_1, \ldots, g_n)$. Note that the notion of cyclic weightwise functions is defined using right cyclic shifts, whereas λ-HWB$_{n,r}$ is obtained from g using left cyclic shifts[2].

4.2 Pre-processing

The function λ-HWB$_{n,r}$ improves the properties of the HWB function by first extending the HWB function to a vectorial function and then applying λ to the output of the vectorial function. This constitutes a post-processing of the output of the HWB vectorial function.

To further improve the nonlinearity, we consider a pre-processing of the input to λ-HWB$_{n,r}$. In more details, we construct a nonlinear bijection $\phi : \mathbb{F}_2^n \to \mathbb{F}_2^n$, so that before applying λ-HWB$_{n,r}$ to an input $\mathbf{x} \in \mathbb{F}_2^n$, we first apply ϕ to $\mathbf{x}$ to obtain $\mathbf{y}$ and then apply λ-HWB$_{n,r}$ to $\mathbf{y}$.

The bijection ϕ combines integer and binary field arithmetic. Given $\mathbf{x} \in \mathbb{F}_2^n$, ϕ does the following: changes the representation of $\mathbf{x}$ to an element of $\mathbb{Z}_{2^n}$ and applies a bijection $\mathcal{B}$; changes back the representation to $\mathbb{F}_2^n$ and reverses the string (which is a linear operation over $\mathbb{F}_2^n$); again changes the representation to $\mathbb{Z}_{2^n}$ and applies $\mathcal{B}$; changes back representation to $\mathbb{F}_2^n$ and produces the output. Note that changing representations from $\mathbb{F}_2^n$ to $\mathbb{Z}_{2^n}$ and vice versa is simply a matter of considering the input to be either a binary string or a non-negative integer, and has no cost. The bijection $\mathcal{B}$ considers $\mathbb{Z}_{2^n}$ to be partitioned into intervals; determines the interval to which the integer representation i of $\mathbf{x}$ belongs, applies a simple permutation of the interval to i to obtain j which is then returned.

The above strategy for constructing ϕ was determined after a great deal of experimentation with combining simple integer and binary field arithmetic. For example, the bijection $\mathcal{B}$ is applied twice; our experiments show that applying $\mathcal{B}$ twice rather than once leads to noticeable increase in nonlinearity, but futher application of $\mathcal{B}$ does not provide significant gain in nonlinearity. We also considered various other approaches, but the nonlinearities achieved by such constructions were not found to be sufficiently high. The approach that we describe achieves good nonlinearity as well as good algebraic resistance as we report below.

Let $\mathbb{Z}_{2^n}$ be the set of integers modulo 2^n. We construct ϕ by mixing simple and fast operations over $\mathbb{Z}_{2^n}$ and $\mathbb{F}_{2^n}$ (conversions between the representations $\mathbb{F}_{2^n}$ and $\mathbb{Z}_{2^n}$ are done using the functions $\mathsf{int}(\mathbf{x})$ and $\mathsf{bin}_n(i)$, as described in Section 2.) The fact that each of the structures $\mathbb{Z}_{2^n}$ and $\mathbb{F}_{2^n}$ is complex with respect to the other is used in the so-called ARX cryptosystems.

The core of our construction of ϕ is based on the idea of partitioning $\mathbb{Z}_{2^n}$ into intervals. We first describe this partitioning strategy.

> *Partition of* $\mathbb{Z}_{2^n}$: Let $n \geq 2$ and $s < n$ be a positive integer. Let $0 \leq w_0, \ldots, w_{2^s-1} \leq 2^n - 1$ be integers such that $w_{k+1} = w_k + 2^{n-s} \bmod 2^n$. For $0 \leq k \leq 2^s - 1$, let $I_k = \{w_k, w_k + 1, \ldots, w_k + 2^{n-s} - 1\}$ where the elements of the set I_k are computed modulo 2^n.

[2]We were unaware of the paper [22] when we obtained the function λ-HWB$_{n,r}$. It is only later that we realised that λ-HWB$_{n,r}$ is a special case of (left) cyclic weightwise functions.

Proposition 7 *The collection of sets* $\{I_k\}$ *with* $k = 0, \ldots, 2^s - 1$ *forms a partition of* $\mathbb{Z}_{2^n}$.

Proof: Note that the number of I_k's is 2^s, and each I_k is a subset of $\mathbb{Z}_{2^n}$ containing 2^{n-s} elements. So to show the result it is sufficient to show that for $0 \leq k < \ell \leq 2^s - 1$, I_k and I_ℓ are disjoint. From the definition of the w_k's, we have $w_\ell = w_k + (\ell - k)2^{n-s} \bmod 2^n$. Suppose that I_k and I_ℓ have a non-empty intersection. Then there are integers a and b with $0 \leq a, b \leq 2^{n-s} - 1$ such that $w_k + a \equiv w_\ell + b \bmod 2^n$, i.e. $(\ell - k)2^{n-s} + (b - a) \equiv 0 \bmod 2^n$. Note that $1 \leq \ell - k \leq 2^s - 1$ and so $2^{n-s} \leq (\ell - k)2^{n-s} \leq 2^n - 2^{n-s}$. Further, $-2^{n-s} + 1 \leq b - a \leq 2^{n-s} - 1$. So $1 \leq (\ell - k)2^{n-s} + (b - a) \leq 2^n - 1$. Consequently, $(\ell - k)2^{n-s} + (b - a) \not\equiv 0 \bmod 2^n$, which is a contradiction. □

Proposition 8 *For* $n \geq 2$, $w_0 \in \mathbb{Z}_{2^n}$ *and positive integer* $s < n$, *define* $\mathcal{I}_{n,w_0,s} : \mathbb{Z}_{2^n} \to \mathbb{Z}_{2^s}$ *as follows.*

$$\mathcal{I}_{n,w_0,s}(i) \quad = \quad \begin{cases} \left\lfloor \frac{i - w_0}{2^{n-s}} \right\rfloor & \text{if } i \geq w_0, \\ \\ \left\lfloor \frac{i + 2^n - w_0}{2^{n-s}} \right\rfloor & \text{if } i < w_0. \end{cases} \tag{6}$$

Let $k = \mathcal{I}_{n,w_0,s}(i)$. *Then* $w_k = w_0 + k2^{n-s} \bmod 2^n$ *and* k *is the unique integer such that* i *is in* $I_k = \{w_k, w_k + 1, \ldots, w_k + 2^{n-s} - 1\}$.

Using the collection of intervals $\{I_k\}$, we define a bijection $\mathcal{B}$ of $\mathbb{Z}_{2^n}$. The idea is the following. Let $i \in \mathbb{Z}_{2^n}$. Then i is in one of the intervals I_k, and from i, the value of k can be found using Proposition 8. Suppose then that $i = w_k + a$, for some $a \in \mathbb{Z}_{2^{n-s}}$. Let $b = (2k + 1)a \bmod 2^{n-s}$. Since $2k + 1$ is odd, the map $a \mapsto (2k + 1)a \bmod 2^{n-s}$ is a bijection of $\mathbb{Z}_{2^{n-s}}$. So $b \in \mathbb{Z}_{2^{n-s}}$. Let $j = w_k + b$. We set $\mathcal{B}(i)$ to be equal to j. In the following result we provide a more formal description of the bijection $\mathcal{B}$.

Proposition 9 *For* $n \geq 2$, *positive integer* $s < n$ *and* $w_0 \in \mathbb{Z}_{2^n}$, *define* $\mathcal{B}_{n,w_0,s} : \mathbb{Z}_{2^n} \to \mathbb{Z}_{2^n}$ *as follows. For* $i \in \mathbb{Z}_{2^n}$, *the value of* $\mathcal{B}_{n,w_0,s}(i)$ *is determined by the following sequence of steps.*

1. $k \leftarrow \mathcal{I}_{n,w_0,s}(i)$;
2. $w_k \leftarrow w_0 + k2^{n-s} \bmod 2^n$;
3. $a \leftarrow (i - w_k) \bmod 2^n$;
4. $b \leftarrow a(2k + 1) \bmod 2^{n-s}$;
5. $j \leftarrow b + w_k \bmod 2^n$;
6. *set* $\mathcal{B}_{n,w_0,s}(i)$ *to be equal to* j.

The map $\mathcal{B}_{n,w_0,s}$ *defined above is a bijection.*

Proof: From Proposition 8, k is the unique integer such that i is in I_k. Since w_0 is given, w_k is uniquely determined by k and hence w_k is uniquely determined by i. So $a = i - w_k \bmod 2^n$ is an element of $\mathbb{Z}_{2^{n-s}}$, which is uniquely determined by i. Since $2k + 1$ is odd, the map $a \mapsto a(2k + 1) \bmod 2^{n-s}$ is a bijection from $\mathbb{Z}_{2^{n-s}}$ to itself. So b is in $\mathbb{Z}_{2^{n-s}}$ and is uniquely determined by a. Since $j = b + w_k \bmod 2^n$, b is uniquely determined by a, a itself is uniquely determined by i, and w_k is uniquely determined by i, it follows that j is also uniquely determined by i. This shows that $\mathcal{B}_{n,w_0,s}$ is an injection and hence a bijection. □

Given $\mathbf{x} = (x_1, x_2, \ldots, x_{n-1}, x_n) \in \mathbb{F}_2^n$, let $\mathsf{reverse}(\mathbf{x})$ denote the string $(x_n, x_{n-1}, \ldots, x_2, x_1)$, i.e. $\mathsf{reverse}(\mathbf{x})$ reverses the string $\mathbf{x}$. Using $\mathcal{B}$ and $\mathsf{reverse}$, we define a bijection ϕ from $\mathbb{F}_2^n$ to itself. The idea is the following. Given $\mathbf{x} \in \mathbb{F}_2^n$, change the representation to $i \in \mathbb{Z}_{2^n}$. Let $j = \mathcal{B}(i)$. Change the representation of j from $\mathbb{Z}_{2^n}$ to $\mathbb{F}_{2^n}$, use $\mathsf{reverse}$, and then change the representation back to $\mathbb{Z}_{2^n}$. Apply $\mathcal{B}$ once again and change the representation to $\mathbb{F}_{2^n}$ and produce as the output of ϕ. The description is made precise in the following result.

Proposition 10 *Given $n \geq 2$, positive integer $s < n$ and $w_0 \in \mathbb{Z}_{2^n}$, define a map $\phi_{n,w_0,s} : \mathbb{F}_2^n \to \mathbb{F}_2^n$ as follows. For $\mathbf{x} \in \mathbb{F}_2^n$, the following defines $\phi_{n,w_0,s}(\mathbf{x})$.*

$i \leftarrow \mathsf{int}(\mathbf{x})$; $j \leftarrow \mathcal{B}_{n,w_0,s}(i)$; $\mathbf{y} \leftarrow \mathsf{bin}_n(j)$;
$\mathbf{w} \leftarrow \mathsf{reverse}(\mathbf{y})$;
$i \leftarrow \mathsf{int}(\mathbf{w})$; $j \leftarrow \mathcal{B}_{n,w_0,s}(i)$; $\mathbf{z} \leftarrow \mathsf{bin}_n(j)$;
set $\phi_{n,w_0,s}(\mathbf{x})$ *to be equal to* $\mathbf{z}$.

The map $\phi_{n,w_0,s}$ described above is a bijection.

Using $\phi_{n,w_0,s}$ and λ-$\mathsf{HWB}_{n,r}$ we define a Boolean function $\mathsf{IntHWB}_{n,w_0,s,\lambda} : \mathbb{F}_2^n \to \mathbb{F}_2^n$ as follows.

$$\mathsf{IntHWB}_{n,w_0,s,\lambda} \quad = \quad \lambda\text{-}\mathsf{HWB}_{n,r} \circ \phi_{n,w_0,s} = \lambda \circ \mathsf{HWB}_{n,r} \circ \phi_{n,w_0,s}. \tag{7}$$

So for $\mathbf{x} \in \mathbb{F}_2^n$,

$$\mathsf{IntHWB}_{n,w_0,s,\lambda}(\mathbf{x}) \quad = \quad \lambda(\mathsf{HWB}_{n,r}(\phi_{n,w_0,s}(\mathbf{x}))). \tag{8}$$

One may note that the application of $\phi_{n,w_0,s}$ to $\mathbf{x}$ corresponds to a pre-processing of the input to λ-$\mathsf{HWB}_{n,r}$.

Efficiency. The parameters to the map $\mathsf{IntHWB}_{n,w_0,s,\lambda}$ are the integers $w_0 \in \mathbb{Z}_{2^n}$, $s < n$ and the r-variable function λ. The number of bits required to store w_0 is n and the number of bits required to store s is $\lceil \log_2 n \rceil$. Assuming that λ is stored in its truth table representation, $\mathsf{IntHWB}_{n,w_0,s,\lambda}$ requires $n + \lceil \log_2 n \rceil + 2^r$ bits to be stored. Note that we consider $r = 5$ and so $\mathsf{IntHWB}_{n,w_0,s,\lambda}$ has a very efficient space representation. Computing $\phi_{n,w_0,s}$ requires computing $\mathcal{B}_{n,w_0,s}$ twice and the reversal of an n-bit string. The computation of $\mathcal{B}_{n,w_0,s}$ requires the computation of $\mathcal{I}_{n,w_0,s}$ (which requires an integer subtraction, optionally an integer addition and a right shift operation to implement the quotient), four truncations to implement the modulo operations, one left shift operation to implement the multiplication by 2^{n-s}, two integer additions, a left shift and an increment to compute $2k+1$, one integer subtraction, and a single integer multiplication. In a hardware implementation, w_0 and s will be hardcoded into the circuit and λ will be implemented as a small combinational circuit. The other operations are simple arithmetic operations on n-bit quantities. In particular, for n around 20, we do not expect the size of the circuit implementing $\mathsf{IntHWB}_{n,w_0,s,\lambda}$ to be too large; a rough estimate is about a few hundred gates. So $\mathsf{IntHWB}_{n,w_0,s,\lambda}$ is a very efficiently implementable function.

Proposition 11 *Let $n \geq 2$, $w_0 \in \mathbb{Z}_{2^n}$, $s, r < n$ be positive integers, and λ be an r-variable function. Then $\mathsf{IntHWB}_{n,w_0,s,\lambda}$ is balanced if and only if λ is balanced.*

Proof: Since $\phi_{n,w_0,s}$ is a bijection, $\phi_{n,w_0,s} \circ \lambda$-$\mathsf{HWB}_{n,r}$ is balanced if and only if λ-$\mathsf{HWB}_{n,r}$ is balanced if and only if λ is balanced. □

The requirement is to choose w_0, s and λ in a manner so that $\mathsf{IntHWB}_{n,w_0,s,\lambda}$ has high nonlinearity. Since $\mathsf{IntHWB}_{n,w_0,s,\lambda}$ is constructed using the composition operator, using (2) the Walsh transform of $\mathsf{IntHWB}_{n,w_0,s,\lambda}$ can be expressed in terms of the Walsh transforms of $\phi_{n,w_0,s}$, $\mathsf{HWB}_{n,r}$ and λ. The resulting expression, however, does not provide guidance on how to choose the parameters of $\mathsf{IntHWB}_{n,w_0,s,\lambda}$ to ensure high nonlinearity. Further, we are also not aware of any other analytical method for ensuring that $\mathsf{IntHWB}_{n,w_0,s,\lambda}$ has high nonlinearity. In view of this, we decided to search for appropriate parameters so that $\mathsf{IntHWB}_{n,w_0,s,\lambda}$ has high nonlinearity. Letting $w_0 \in \mathbb{Z}_{2^n}$, $s \leq \lfloor n/2 \rfloor$ and λ to be a balanced r-variable function make the size of the parameter space $O(n2^n\binom{2^r}{2^{r-1}})$. For each selection of parameters

in this space, it is required to construct the function $\mathsf{IntHWB}_{n,w_0,s,\lambda}$ and compute its nonlinearity. This requires $O(n2^n)$ time. So the total time for the search becomes $O(n^2 2^{2n}\binom{2^r}{2^{r-1}})$. This is computationally infeasible. So we decided to fix $r = 5$ and consider the functions $\lambda_{5,i}$ corresponding to the values of n given by Proposition 6. This reduces the search time to $O(n^2 2^{2n})$. For $n = 13, \ldots, 20$ we were able to carry out this search. The search algorithm is given in Algorithm 2.

Algorithm 2: The search procedure for $\mathsf{IntHWB}_{n,w_0,s,\lambda}$.

Input: n, $\mathcal{L}$, where $\mathcal{L}$ is the list of $\lambda_{5,i}$ corresponding to n as given in Proposition 6
Output: A list $\mathcal{P}$ of triplets (λ, s, w_0).

```
1  maxnl ← 0; P ← ∅
2  for λ ∈ L do
3      for s in {1, ..., ⌊n/2⌋} do
4          for w_0 in Z_{2^n} do
5              let f = IntHWB_{n,w_0,s,λ}
6              compute nl(f) and deg(f)
7              if deg(f) = n − 1 and maxnl < nl(f) then
8                  maxnl ← nl(f); P ← {(λ, s, w_0)}
9              else
10                 if deg(f) = n − 1 and maxnl = nl(f) then
11                     P ← P ∪ {(λ, s, w_0)}
12 return P
```

The results of running Algorithm 2 for $n = 13, \ldots, 20$ are stated in the following proposition.

Proposition 12 *For $n = 13, \ldots, 20$ and λ is one of $\lambda_{5,i}$ given by Proposition 6, the maximum nonlinearities achieved by $\mathsf{IntHWB}_{n,w_0,s,\lambda}$ are as follows.*

1. *$n = 13$: for $s = 4$, $w_0 = 254$, $\mathsf{nl}(\mathsf{IntHWB}_{n,w_0,s,\lambda_{5,2}}) = 3952$.*
2. *$n = 14$: for $s = 5$, $w_0 = 13090$, $\mathsf{nl}(\mathsf{IntHWB}_{n,w_0,s,\lambda_{5,4}}) = 7974$.*
3. *$n = 15$: for $s = 7$, $w_0 = 21272$, $\mathsf{nl}(\mathsf{IntHWB}_{n,w_0,s,\lambda_{5,2}}) = 16062$.*
4. *$n = 16$:*
   for $s = 4$, $w_0 = 16699$, $\mathsf{nl}(\mathsf{IntHWB}_{n,w_0,s,\lambda_{5,5}}) = 32290$;
   for $s = 4$, $w_0 = 27429$, $\mathsf{nl}(\mathsf{IntHWB}_{n,w_0,s,\lambda_{5,5}}) = 32290$.
5. *$n = 17$: for $s = 4$, $w_0 = 105883$, $\mathsf{nl}(\mathsf{IntHWB}_{n,w_0,s,\lambda_{5,1}}) = 64834$.*
6. *$n = 18$: for $s = 5$, $w_0 = 118924$, $\mathsf{nl}(\mathsf{IntHWB}_{n,w_0,s,\lambda_{5,7}}) = 130042$.*
7. *$n = 19$: for $s = 5$, $w_0 = 200085$, $\mathsf{nl}(\mathsf{IntHWB}_{n,w_0,s,\lambda_{5,9}}) = 260606$.*
8. *$n = 20$: for $s = 5$, $w_0 = 353518$, $\mathsf{nl}(\mathsf{IntHWB}_{n,w_0,s,\lambda_{5,12}}) = 522046$.*

For $n = 13, \ldots, 19$, the algebraic immunities of the functions given in Proposition 12 could be computed on our servers, but for $n = 20$, the process exited abnormally and did not return the value of AI. The values of AI for $n = 13, \ldots, 19$ are stated in the following proposition.

Proposition 13 *The algebraic immunities of the functions in Proposition 12 are as follows.*

1. *$n = 13$: for $s = 4$, $w_0 = 254$, $\mathsf{AI}(\mathsf{IntHWB}_{n,w_0,s,\lambda_{5,2}}) = 6$.*
2. *$n = 14$: for $s = 5$, $w_0 = 13090$, $\mathsf{AI}(\mathsf{IntHWB}_{n,w_0,s,\lambda_{5,4}}) = 7$.*
3. *$n = 15$: for $s = 7$, $w_0 = 21272$, $\mathsf{AI}(\mathsf{IntHWB}_{n,w_0,s,\lambda_{5,2}}) = 7$.*

4. $n = 16$: *for* $s = 4$, $w_0 = 16699$, $\mathsf{AI}(\mathsf{IntHWB}_{n,w_0,s,\lambda_{5,5}}) = 8$;
   for $s = 4$, $w_0 = 27429$, $\mathsf{AI}(\mathsf{IntHWB}_{n,w_0,s,\lambda_{5,5}}) = 8$.
5. $n = 17$: *for* $s = 4$, $w_0 = 105883$, $\mathsf{AI}(\mathsf{IntHWB}_{n,w_0,s,\lambda_{5,1}}) = 9$.
6. $n = 18$: *for* $s = 5$, $w_0 = 118924$, $\mathsf{AI}(\mathsf{IntHWB}_{n,w_0,s,\lambda_{5,7}}) = 9$.
7. $n = 19$: *for* $s = 5$, $w_0 = 200085$, $\mathsf{nl}(\mathsf{IntHWB}_{n,w_0,s,\lambda_{5,9}}) = 9$.

Note that except for $n = 13, 15$ and 19, in all other cases the algebraic immunities are optimal, and for $n = 13, 15$ and 19, the algebraic immunities are one less than the optimal. We conjecture that the value of AI for the function in Proposition 12 for $n = 20$ is 10. This is based on our further study of algebraic immunities as discussed below.

To further understand the algebraic immunities of the functions in the class $\mathsf{IntHWB}_{n,w_0,s,\lambda}$, we conducted some more experiments. For $n = 13, \ldots, 19$, we fixed λ and s as in Proposition 13 and for 100 randomly chosen values of w_0, we constructed the function $\mathsf{IntHWB}_{n,w_0,s,\lambda}$ and computed its nonlinearity and algebraic immunity. For $n = 14, 16$ and 18, in all the 100 cases the algebraic immunities came out to be $n/2$, i.e. optimal. For $n = 13, 15, 17$ and 19, in all the 100 cases the algebraic immunities came out to be either $\lfloor n/2 \rfloor$ or $\lceil n/2 \rceil$. Letting a_1 and a_2 to be the number of cases where the algebraic immunities came out to be $\lfloor n/2 \rfloor$ and $\lceil n/2 \rceil$ respectively, we obtained $(a_1, a_2) = (70, 30), (65, 35), (73, 27), (62, 38)$ for $n = 13, 15, 17$ and 19 respectively. So the experiments provide evidence that for even n functions in the class $\mathsf{IntHWB}_{n,w_0,s,\lambda}$ have optimal algebraic immunity, while for odd n, functions in the class $\mathsf{IntHWB}_{n,w_0,s,\lambda}$ have either optimal or almost optimal algebraic immunity, with optimal algebraic immunity occuring for about 30% or more of the cases.

For $n = 17$, the function in Proposition 12 has optimal algebraic immunity. For $n = 13, 15$ and 19, the functions in Proposition 12 have algebraic immunity one less than the optimal. From the results of our above mentioned experiments with 100 random values of w_0, we provide examples of functions for $n = 13, 15$ and 19 with optimal algebraic immunity.

Example 1

- $n = 13$: *for* $s = 3$, $w_0 = 3204$,
  $\mathsf{nl}(\mathsf{IntHWB}_{n,w_0,s,\lambda_{5,7}}) = 3950$, $\mathsf{AI}(\mathsf{IntHWB}_{n,w_0,s,\lambda_{5,7}}) = 7$.
- $n = 15$: *for* $s = 4$, $w_0 = 51$,
  $\mathsf{nl}(\mathsf{IntHWB}_{n,w_0,s,\lambda_{5,7}}) = 16036$, $\mathsf{AI}(\mathsf{IntHWB}_{n,w_0,s,\lambda_{5,7}}) = 8$.
- $n = 19$: *for* $s = 5$, $w_0 = 471438$,
  $\mathsf{nl}(\mathsf{IntHWB}_{n,w_0,s,\lambda_{5,9}}) = 260502$, $\mathsf{AI}(\mathsf{IntHWB}_{n,w_0,s,\lambda_{5,7}}) = 10$.

Note that for $n = 13$, the nonlinearity of the above example is 3950, while the maximum nonlinearity reported in Proposition 12 is 3952. For $n = 15$, the nonlinearity of the above example is 16036, while the maximum nonlinearity reported in Proposition 12 is 16062. For $n = 19$, the nonlinearity of the above example is 260502, while the maximum nonlinearity reported in Proposition 12 is 260606. So for $n = 13, 15$ and 19, optimal AI can be obtained with a small decrease in nonlinearity.

To assess the resistance of the class of functions to fast algebraic attacks, we computed the FAA-profile for the functions given in Proposition 12 for $n = 13, 14, 15$ and 16 and also for the functions in Example 1. These are given below.

- FAA-profile for $\mathsf{IntHWB}_{n,w_0,s,\lambda_{5,2}}$ with $n = 13$, $s = 4$, $w_0 = 254$:
  $(1, 11), (2, 9), (3, 9), (4, 7), (5, 7)$.

- FAA-profile for $\mathsf{IntHWB}_{n,w_0,s,\lambda_{5,7}}$ with $n = 13$, $s = 3$, $w_0 = 3204$:
$(1, 10), (2, 9), (3, 9), (4, 7), (5, 7), (6, 6)$.

- FAA-profile for $\mathsf{IntHWB}_{n,w_0,s,\lambda_{5,4}}$ with $n = 14$, $s = 5$, $w_0 = 13090$:
$(1, 11), (2, 11), (3, 10), (4, 8), (5, 7), (6, 7)$.

- FAA-profile for $\mathsf{IntHWB}_{n,w_0,s,\lambda_{5,2}})$ with $n = 15$, $s = 7$, $w_0 = 21272$:
$(1, 13), (2, 11), (3, 11), (4, 9), (5, 9), (6, 7)$.

- FAA-profile for $\mathsf{IntHWB}_{n,w_0,s,\lambda_{5,7}}$ with $n = 15$, $s = 4$, $w_0 = 51$:
$(1, 13), (2, 11), (3, 10), (4, 9), (5, 8), (6, 7), (7, 7)$.

- FAA-profile for $\mathsf{IntHWB}_{n,w_0,s,\lambda_{5,5}}$ with $n = 16$, $s = 4$, $w_0 = 16699$:
$(1, 13), (2, 12), (3, 11), (4, 10), (5, 9), (6, 8), (7, 7)$.

- FAA-profile for $\mathsf{IntHWB}_{n,w_0,s,\lambda_{5,5}}$ with $n = 16$, $s = 4$, $w_0 = 27429$:
$(1, 13), (2, 12), (3, 11), (4, 10), (5, 9), (6, 8), (7, 7)$.

We find that almost perfect FAA-profile is achieved in all cases. Consequently, for all such functions f, $\mathsf{FAI}(f) \geq n - 1$. This indicates good resistance of these functions to fast algebraic attacks.

For $n = 17, \ldots, 20$, due to high memory requirement, it was not possible to compute the complete FAA-profiles for the functions in Proposition 12 and Example 1. Below we provide the partial FAA-profiles that could be computed.

- partial FAA-profile for $\mathsf{IntHWB}_{n,w_0,s,\lambda_{5,1}}$ with $n = 17$, $s = 4$, $w_0 = 105883$:
$(1, 14), (2, 14), (3, 13), (4, 12), (5, 11)$.

- partial FAA-profile for $\mathsf{IntHWB}_{n,w_0,s,\lambda_{5,7}}$ with $n = 18$, $s = 5$, $w_0 = 118924$:
$(1, 15), (2, 15), (3, 13), (4, 12)$.

- partial FAA-profile for $\mathsf{IntHWB}_{n,w_0,s,\lambda_{5,9}}$ with $n = 19$, $s = 5$, $w_0 = 200085$:
$(1, 16), (2, 15), (3, 14)$.

- partial FAA-profile for $\mathsf{IntHWB}_{n,w_0,s,\lambda_{5,9}}$ with $n = 19$, $s = 5$, $w_0 = 471438$:
$(1, 17), (2, 15), (3, 15)$.

- partial FAA-profile for $\mathsf{IntHWB}_{n,w_0,s,\lambda_{5,12}}$ with $n = 20$, $s = 5$, $w_0 = 353518$:
$(1, 17), (2, 16), (3, 15)$.

We observe that in all cases for (e, d) in the above partial FAA-profiles, the relation $e + d \geq n - 2$ holds and we conjecture that for any of these functions f, the relation $\mathsf{FAI}(f) \geq n - 1$ hold.

Remark 2 *From the experimental results we observe that for all the n-variable functions f of the type* IntHWB*, for which we were able to compute the algebraic immunities and the FAA-profiles, we have* $\mathsf{AI}(f) \geq \lfloor n/2 \rfloor$*, and* $\mathsf{FAI}(f) \geq n - 1$*. Further,* $\mathsf{AI}(f) = \lceil n/2 \rceil$ *in several of the cases. This suggests that functions of the type* IntHWB *provide good resistance to algebraic and fast algebraic attacks.*

	Table 7 of [22]		Table 5 of [23]		Proposition 12		CF [8]		cov rad bnd	
n	nl	LLB	nl	LLB	nl	LLB	nl	LLB	CRB_n	LCRB_n
13	3862	−5.13	–	–	3952	−5.83	3988	−6.25	4051	−7.51
14	7816	−5.45	7842	−5.55	7974	−6.23	8072	−7.09	8128	−8.00
15	15748	−5.69	–	–	16062	−6.67	16212	−7.57	16294	−8.51
16	31616	−5.83	31680	−5.91	32290	−7.10	32530	−8.11	32640	−9.00
17	–	–	–	–	64834	−7.54	65210	−8.65	65355	−9.50
18	–	–	–	–	130042	−7.99	130594	−9.10	130816	−10.00
19	–	–	–	–	260606	−8.41	261294	−9.27	261782	−10.50
20	–	–	–	–	522046	−8.87	523234	−9.96	523776	−11.00

Table 1: Comparison of nonlinearities achieved by $\mathsf{IntHWB}_{n,w_0,s,\lambda}$ with Table 7 of [22], Table 5 of [23], the CF functions and the covering radius bound.

n	Table 7 of [22]	Tables 6 and 7 of [23]	Proposition 13	CF [8]
13	(12,6)	–	(12,6)	(12,7)
14	(12,6)	(13,6)	(13,7)	(13,7)
15	(14,6)	–	(14,7)	(14,8)
16	(14,7)	(15,7)	(15,8)	(15,8)
17	–	–	(16,9)	(16,9)
18	–	–	(17,9)	(17,9)
19	–	–	(18,9)	(18,10)

Table 2: Comparison of degrees and algebraic immunities of $\mathsf{IntHWB}_{n,w_0,s,\lambda}$ functions with Table 7 of [22], Tables 6 and 7 of [23] and the CF functions.

4.3 Comparison

Among the previously known efficiently implementable functions HWB, GHWB [7] and the cyclic weightwise function [22, 23], the nonlinearities reported in Table 7 of [22] and Table 5 of [23] are the highest. So we compare the nonlinearities reported in Table 7 of [22] and Table 5 of [23] with those of $\mathsf{IntHWB}_{n,w_0,s,\lambda}$. To provide context, we also compare to the nonlinearites of the CF functions (even though the CF functions are not efficiently implementable) as well as to the values of the covering radius bound. We constructed the CF functions using the primitive polynomials in Appendix A and then computed their nonlinearities. For $n = 13$, the nonlinearity of the CF function that we obtained is higher than the nonlinearity reported in [7]. This is not surprising since the actual function and hence the value of the nonlinearity depends upon the actual primitive polynomial that is used. The comparison of nonlinearities is shown in Table 1. The comparison of degrees and algebraic immunities are shown in Table 2. Each entry of Table 2 is of the form (d, a), where d is the degree and a is the algebraic immunity. We note that the nonlinearities of the IntHWB functions reported in Proposition 12 are higher than the nonlinearities reported in Table 7 of [22] and Table 5 of [23]. For $n = 13$, the algebraic immunity of the IntHWB function given by Proposition 13 is equal to the algebraic immunity of the function reported in Table 7 of [22].

The LLB's of IntHWB functions are about 1.5 bits more than the LLB's of CF functions. While this may seem like a disadvantage, it is not actually so. Suppose a target value of LLB is fixed and the value is achieved by CF functions for a particular value of n. By choosing a higher value of n, the same value of LLB can be also be achieved by IntHWB functions. For example, choosing $n = 19$ we obtain a value of LLB which is lower than the value of the CF function for $n = 16$. Since IntHWB for $n = 19$ is very efficiently implementable, whereas the CF function with $n = 16$ is not, there is no loss in increasing the

number of variables.

4.4 Examples of $\mathsf{IntHWB}_{n,w_0,s,\lambda}$ for $n = 21$ to 30

It becomes very time consuming to run Algorithm 2 for n greater than 20. To obtain an idea of the nonlinearity achieved by $\mathsf{IntHWB}_{n,w_0,s,\lambda}$ for higher values of n we conducted some experiments. We fixed $s = 5$ and $\lambda = \lambda_{5,7}$ and constructed $\mathsf{IntHWB}_{n,w_0,s,\lambda}$ for a number of random choices of w_0. For $n = 21, \ldots, 24$, we chose 10000 values for w_0, while for $n = 25, \ldots, 30$, we chose 1000 values for w_0. For each $n = 21, \ldots, 30$, in the following example, we report the maximum nonlinearity that was achieved.

Example 2

- $n = 21$, $s = 5$, $w_0 = 1948971$: $\mathsf{nl}(\mathsf{IntHWB}_{n,w_0,s,\lambda_{5,7}}) = 1045280$.
- $n = 22$, $s = 5$, $w_0 = 223972$: $\mathsf{nl}(\mathsf{IntHWB}_{n,w_0,s,\lambda_{5,7}}) = 2092280$.
- $n = 23$, $s = 5$, $w_0 = 2179192$: $\mathsf{nl}(\mathsf{IntHWB}_{n,w_0,s,\lambda_{5,7}}) = 4187200$.
- $n = 24$, $s = 5$, $w_0 = 11878200$: $\mathsf{nl}(\mathsf{IntHWB}_{n,w_0,s,\lambda_{5,7}}) = 8378102$.
- $n = 25$, $s = 5$, $w_0 = 17211712$: $\mathsf{nl}(\mathsf{IntHWB}_{n,w_0,s,\lambda_{5,7}}) = 16761306$.
- $n = 26$, $s = 5$, $w_0 = 45478445$: $\mathsf{nl}(\mathsf{IntHWB}_{n,w_0,s,\lambda_{5,7}}) = 33530292$.
- $n = 27$, $s = 5$, $w_0 = 67070690$: $\mathsf{nl}(\mathsf{IntHWB}_{n,w_0,s,\lambda_{5,7}}) = 67070690$.
- $n = 28$, $s = 5$, $w_0 = 95163654$: $\mathsf{nl}(\mathsf{IntHWB}_{n,w_0,s,\lambda_{5,7}}) = 134157910$.
- $n = 29$, $s = 5$, $w_0 = 224553125$: $\mathsf{nl}(\mathsf{IntHWB}_{n,w_0,s,\lambda_{5,7}}) = 268332760$.
- $n = 30$, $s = 5$, $w_0 = 378168951$: $\mathsf{nl}(\mathsf{IntHWB}_{n,w_0,s,\lambda_{5,7}}) = 536691884$.

In Table 3, we compare the nonlinearities in Example 2 with those of the CF-function. Note that for $n = 21, \ldots, 30$, even though we were able to explore a very limited portion of the parameter space of IntHWB functions, the nonlinearities and the values of LLB that are achieved compare quite well to the corresponding values of the CF functions. In particular, the values of LLB for the IntHWB functions is at most about 2 more than those of the CF functions. As explained in Section 4.3, the main advantage of IntHWB functions is their very efficient implementation. So a target value of LLB can be cheaply achieved by increasing the value of n. While a CF function would achieve the same value of LLB for a smaller value of n, it would be much more efficient to implement an IntHWB function with a higher value of n. Further, based on our experiments for $n = 13$ to $n = 20$ we conjecture that even for $n > 20$, the IntHWB functions provide good resistance to algebraic attacks (see Remark 2).

5 Conclusion

We provided constructions of Boolean functions which are good solutions to the "the big single-output Boolean problem" proposed in [7]. The functions are built using simple arithmetic operations leading to these functions being efficient to implement.

For the functions that we propose we provide experimental results on the nonlinearity and algebraic resistance. A theoretical direction of work would be to prove results on nonlinearity and algebraic resistance for these functions. One major problem with doing this is that there are no good mathematical

	Example 2		CF [8]		cov rad bnd	
n	nl	LLB	nl	LLB	CRB_n	LCRB_n
21	1045280	−9.31	1046846	−10.24	1047852	−11.50
22	2092280	−9.75	2094936	−10.89	2096128	−12.00
23	4187200	−10.21	4190834	−11.24	4192856	−12.50
24	8378102	−10.64	8383446	−11.67	8386560	−13.00
25	16761306	−11.04	16769938	−12.17	16774320	−13.50
26	33530292	−11.44	33545384	−12.86	33550336	−14.00
27	67070690	−11.78	67097318	−13.50	67103072	−14.50
28	134157910	−12.13	134201202	−13.99	134209536	−15.00
29	268332760	−12.35	268409892	−14.36	268423871	−15.50
30	536691884	−12.55	536833704	−14.82	536854528	−16.00

Table 3: Comparison of nonlinearities achieved by the functions in Example 2 with those of CF functions and the covering radius bound.

techniques for analysing nonlinearity and algebraic resistance of functions built using a combination of integer arithmetic and arithmetic over $\mathbb{F}_2$. We hope that there will be future research focus on developing such techniques.

Another possible direction of work would be to extend our approach to vectorial functions. The ability to simultaneously produce several bits (instead of one) will lead to higher speed of keystream generation. On the other hand, producing more bits also provides the adversary with more information. For a vectorial function, it will be required to consider the nonlinearity and algebraic resistance of all non-zero component functions (i.e., linear combinations of the coordinate functions). The mathematical challenge is to ensure that each non-zero component function provides sufficient resistance to attacks. This makes the problem more difficult than the construction of Boolean functions. Again, we hope this problem will be addressed in the future.

Acknowledgement

We thank Pierrick Méaux for his comments on an earlier version of the paper. Deng Tang provided us with a program written by Simon Fischer which we have used for computing fast algebraic immunity. We thank both of them.

References

[1] Frederik Armknecht, Claude Carlet, Philippe Gaborit, Simon Künzli, Willi Meier, and Olivier Ruatta. Efficient computation of algebraic immunity for algebraic and fast algebraic attacks. In Serge Vaudenay, editor, *Advances in Cryptology - EUROCRYPT 2006, 25th Annual International Conference on the Theory and Applications of Cryptographic Techniques, St. Petersburg, Russia, May 28 - June 1, 2006, Proceedings*, volume 4004 of *Lecture Notes in Computer Science*, pages 147–164. Springer, 2006. 4

[2] Beate Bollig, Martin Löbbing, Martin Sauerhoff, and Ingo Wegener. On the complexity of the hidden weighted bit function for various BDD models. *RAIRO Theor. Informatics Appl.*, 33(2):103–116, 1999. 5

[3] Nina Brandstätter, Tanja Lange, and Arne Winterhof. On the non-linearity and sparsity of Boolean functions related to the discrete logarithm in finite fields of characteristic two. In Øyvind Ytrehus, editor, *Coding and Cryptography, International Workshop, WCC 2005, Bergen, Norway, March 14-18, 2005. Revised Selected Papers*, volume 3969 of *Lecture Notes in Computer Science*, pages 135–143. Springer, 2005. 5

[4] Randal E. Bryant. On the complexity of VLSI implementations and graph representations of Boolean functions with application to integer multiplication. *IEEE Trans. Computers*, 40(2):205–213, 1991. 2, 5

[5] Claude Carlet. Comments on "Constructions of cryptographically significant Boolean functions using primitive polynomials". *IEEE Trans. Inf. Theory*, 57(7):4852–4853, 2011. 2

[6] Claude Carlet. *Boolean Functions for Cryptography and Coding Theory*. Cambridge University Press, 2021. 1, 2, 3, 4

[7] Claude Carlet. A wide class of Boolean functions generalizing the hidden weight bit function. *IEEE Trans. Inf. Theory*, 68(2):1355–1368, 2022. 1, 2, 6, 16, 17

[8] Claude Carlet and Keqin Feng. An infinite class of balanced functions with optimal algebraic immunity, good immunity to fast algebraic attacks and good nonlinearity. In Josef Pieprzyk, editor, *Advances in Cryptology - ASIACRYPT 2008, 14th International Conference on the Theory and Application of Cryptology and Information Security, Melbourne, Australia, December 7-11, 2008. Proceedings*, volume 5350 of *Lecture Notes in Computer Science*, pages 425–440. Springer, 2008. 2, 5, 16, 18

[9] L. Carlitz and S. Uchiyama. Bounds for exponential sums. *Duke Math. J.*, 24(1):37–41, 1957. 6

[10] Nicolas T. Courtois. Fast algebraic attacks on stream ciphers with linear feedback. In Dan Boneh, editor, *Advances in Cryptology - CRYPTO 2003, 23rd Annual International Cryptology Conference, Santa Barbara, California, USA, August 17-21, 2003, Proceedings*, volume 2729 of *Lecture Notes in Computer Science*, pages 176–194. Springer, 2003. 3

[11] Nicolas T. Courtois and Willi Meier. Algebraic attacks on stream ciphers with linear feedback. In Eli Biham, editor, *Advances in Cryptology - EUROCRYPT 2003, International Conference on the Theory and Applications of Cryptographic Techniques, Warsaw, Poland, May 4-8, 2003, Proceedings*, volume 2656 of *Lecture Notes in Computer Science*, pages 345–359. Springer, 2003. 3

[12] F. Didier. A new upper bound on the block error probability after decoding over the erasure channel. *IEEE Trans. Inf. Theory*, 52(10):4496–4503, 2006. 3

[13] Frédéric Didier. Using wiedemann's algorithm to compute the immunity against algebraic and fast algebraic attacks. In Rana Barua and Tanja Lange, editors, *Progress in Cryptology - INDOCRYPT 2006, 7th International Conference on Cryptology in India, Kolkata, India, December 11-13, 2006, Proceedings*, volume 4329 of *Lecture Notes in Computer Science*, pages 236–250. Springer, 2006. 4

[14] Frédéric Didier and Jean-Pierre Tillich. Computing the algebraic immunity efficiently. In Matthew J. B. Robshaw, editor, *Fast Software Encryption, 13th International Workshop, FSE 2006, Graz, Austria, March 15-17, 2006, Revised Selected Papers*, volume 4047 of *Lecture Notes in Computer Science*, pages 359–374. Springer, 2006. 4

[15] Keqin Feng, Qunying Liao, and Jing Yang. Maximal values of generalized algebraic immunity. *Des. Codes Cryptogr.*, 50(2):243–252, 2009. 5

[16] Xiutao Feng and Guang Gong. On algebraic immunity of trace inverse functions on finite fields of characteristic two. *J. Syst. Sci. Complex.*, 29(1):272–288, 2016. 6

[17] Kishan Chand Gupta and Palash Sarkar. Toward a general correlation theorem. *IEEE Trans. Inf. Theory*, 51(9):3297–3302, 2005. 4

[18] Donald E. Knuth. *The Art of Computer Programming Volume 4, Fascicle 1: Bitwise Tricks and Techniques: Binary Decision Diagrams.* Addison-Wesley Professional, 2009. 5

[19] Matthias Krause. BDD-based cryptanalysis of keystream generators. In Lars R. Knudsen, editor, *Advances in Cryptology - EUROCRYPT 2002, International Conference on the Theory and Applications of Cryptographic Techniques, Amsterdam, The Netherlands, April 28 - May 2, 2002, Proceedings*, volume 2332 of *Lecture Notes in Computer Science*, pages 222–237. Springer, 2002. 5

[20] Matthias Krause and Dirk Stegemann. Reducing the space complexity of BDD-based attacks on keystream generators. In Matthew J. B. Robshaw, editor, *Fast Software Encryption, 13th International Workshop, FSE 2006, Graz, Austria, March 15-17, 2006, Revised Selected Papers*, volume 4047 of *Lecture Notes in Computer Science*, pages 163–178. Springer, 2006. 5

[21] Meicheng Liu, Yin Zhang, and Dongdai Lin. Perfect algebraic immune functions. In Xiaoyun Wang and Kazue Sako, editors, *Advances in Cryptology - ASIACRYPT 2012 - 18th International Conference on the Theory and Application of Cryptology and Information Security, Beijing, China, December 2-6, 2012. Proceedings*, volume 7658 of *Lecture Notes in Computer Science*, pages 172–189. Springer, 2012. 4, 5

[22] Pierrick Méaux and Yassine Ozaim. On the cryptographic properties of weightwise affine and weightwise quadratic functions. *Discrete Applied Mathematics*, 355:13–29, 2024. 2, 6, 7, 8, 10, 16

[23] Pierrick Méaux, Tim Seuré, and Deng Tang. The revisited hidden weight bit function. Cryptology ePrint Archive, Paper 2024/2022, 2024. 2, 6, 16

[24] Willi Meier, Enes Pasalic, and Claude Carlet. Algebraic attacks and decomposition of Boolean functions. In Christian Cachin and Jan Camenisch, editors, *Advances in Cryptology - EUROCRYPT 2004, International Conference on the Theory and Applications of Cryptographic Techniques, Interlaken, Switzerland, May 2-6, 2004, Proceedings*, volume 3027 of *Lecture Notes in Computer Science*, pages 474–491. Springer, 2004. 3

[25] Kaisa Nyberg. Differentially uniform mappings for cryptography. In Tor Helleseth, editor, *Advances in Cryptology - EUROCRYPT '93, Workshop on the Theory and Application of of Cryptographic Techniques, Lofthus, Norway, May 23-27, 1993, Proceedings*, volume 765 of *Lecture Notes in Computer Science*, pages 55–64. Springer, 1993. 6

[26] Oscar S. Rothaus. On "bent" functions. *J. Comb. Theory, Ser. A*, 20(3):300–305, 1976. 3

[27] Qichun Wang, Claude Carlet, Pantelimon Stanica, and Chik How Tan. Cryptographic properties of the hidden weighted bit function. *Discret. Appl. Math.*, 174:1–10, 2014. 2, 5

[28] Qichun Wang, Chik How Tan, and Pantelimon Stanica. Concatenations of the hidden weighted bit function and their cryptographic properties. *Adv. Math. Commun.*, 8(2):153–165, 2014. 2, 5

A Primitive Polynomials Used to Construct CF Functions

For $n = 13$ to 30, the following primitive polynomials were used in the construction of the CF functions.

$x^{13} \oplus x^4 \oplus x^3 \oplus x \oplus 1$
$x^{14} \oplus x^{12}x^{11} \oplus x \oplus 1$
$x^{15} \oplus x \oplus 1$
$x^{16} \oplus x^5 \oplus x^3 \oplus x^2 \oplus 1$
$x^{17} \oplus x^3 \oplus 1$
$x^{18} \oplus x^7 \oplus 1$
$x^{19} \oplus x^6 \oplus x^5 \oplus x \oplus 1$
$x^{20} \oplus x^3 \oplus 1$
$x^{21} \oplus x^2 \oplus 1$
$x^{22} \oplus x \oplus 1$
$x^{23} \oplus x^5 \oplus 1$
$x^{24} \oplus x^4 \oplus x^3 \oplus x \oplus 1$
$x^{25} \oplus x^3 \oplus 1$
$x^{26} \oplus x^6 \oplus x^2 \oplus x^1 \oplus 1$
$x^{27} \oplus x^5 \oplus x^2 \oplus x^1 \oplus 1$
$x^{28} \oplus x^3 \oplus 1$
$x^{29} \oplus x^2 \oplus 1$
$x^{30} \oplus x^{23} \oplus x^2 \oplus x^1 \oplus 1$

B ANFs of $\lambda_{5,i}$, $i = 1, \ldots, 12$

$$\begin{aligned}
&\lambda_{5,1}(X_1, X_2, X_3, X_4, X_5)\\
&\quad = \; X_1X_2X_3 \oplus X_1X_2X_4 \oplus X_1X_2 \oplus X_1X_3X_4X_5 \oplus X_1X_3 \oplus X_1X_5 \oplus X_1 \oplus X_3X_5\\
&\qquad \oplus X_4X_5 \oplus X_4 \oplus X_5 \oplus 1\\
&\lambda_{5,2}(X_1, X_2, X_3, X_4, X_5)\\
&\quad = \; X_1X_2X_3X_5 \oplus X_1X_2X_3 \oplus X_1X_2X_5 \oplus X_1X_3X_5 \oplus X_1 \oplus X_2X_3X_5 \oplus X_2X_3 \oplus X_2X_4X_5\\
&\qquad \oplus X_2X_4 \oplus X_3X_4X_5 \oplus X_3X_4 \oplus X_3X_5 \oplus X_4X_5 \oplus X_4 \oplus X_5 \oplus 1\\
&\lambda_{5,3}(X_1, X_2, X_3, X_4, X_5)\\
&\quad = \; X_1X_2X_3X_5 \oplus X_1X_2X_5 \oplus X_1X_3X_5 \oplus X_1X_3 \oplus X_1X_4X_5 \oplus X_1 \oplus X_2X_3X_4X_5 \oplus X_2X_4X_5\\
&\qquad \oplus X_2X_5 \oplus X_2 \oplus X_3X_4X_5 \oplus X_3X_4 \oplus X_3 \oplus X_4X_5 \oplus X_4 \oplus 1\\
&\lambda_{5,4}(X_1, X_2, X_3, X_4, X_5)\\
&\quad = \; X_1X_2X_3X_4 \oplus X_1X_2X_5 \oplus X_1X_2 \oplus X_1X_3X_4X_5 \oplus X_1X_4 \oplus X_1 \oplus X_2X_3X_4 \oplus X_2X_3\\
&\qquad \oplus X_2X_5 \oplus X_2 \oplus X_3X_4X_5 \oplus X_3X_5 \oplus X_3 \oplus 1\\
&\lambda_{5,5}(X_1, X_2, X_3, X_4, X_5)\\
&\quad = \; X_1X_2X_3X_5 \oplus X_1X_2X_3 \oplus X_1X_2X_5 \oplus X_1X_3X_5 \oplus X_1X_5 \oplus X_1 \oplus X_2X_3X_4 \oplus X_2X_4X_5\\
&\qquad \oplus X_2X_4 \oplus X_3X_4X_5 \oplus X_3X_4 \oplus X_3X_5 \oplus X_4X_5 \oplus X_4 \oplus X_5 \oplus 1\\
&\lambda_{5,6}(X_1, X_2, X_3, X_4, X_5)\\
&\quad = \; X_1X_2X_3 \oplus X_1X_2X_4 \oplus X_1X_2 \oplus X_1X_3X_4X_5 \oplus X_1X_3X_4 \oplus X_1X_4 \oplus X_1 \oplus X_2X_3X_4\\
&\qquad \oplus X_2X_3 \oplus X_2X_4 \oplus X_2 \oplus X_3X_4 \oplus X_3X_5 \oplus X_3 \oplus X_4X_5 \oplus 1\\
&\lambda_{5,7}(X_1, X_2, X_3, X_4, X_5)\\
&\quad = \; X_1X_2X_4 \oplus X_1X_2X_5 \oplus X_1X_2 \oplus X_1X_3X_4X_5 \oplus X_1X_3X_4 \oplus X_1X_3X_5 \oplus X_1X_4 \oplus X_3\\
&\qquad \oplus X_4X_5 \oplus X_4 \oplus X_5 \oplus 1\\
&\lambda_{5,8}(X_1, X_2, X_3, X_4, X_5)\\
&\quad = \; X_1X_2X_3X_5 \oplus X_1X_2X_3 \oplus X_1X_2X_5 \oplus X_1X_4X_5 \oplus X_1X_4 \oplus X_1X_5 \oplus X_1 \oplus X_2X_4X_5\\
&\qquad \oplus X_2X_4 \oplus X_3X_5 \oplus X_4X_5 \oplus X_4 \oplus X_5 \oplus 1\\
&\lambda_{5,9}(X_1, X_2, X_3, X_4, X_5)\\
&\quad = \; X_1X_2X_3X_4 \oplus X_1X_2X_3 \oplus X_1X_2X_4X_5 \oplus X_1X_2X_4 \oplus X_1X_2 \oplus X_1X_3X_4 \oplus X_1X_5 \oplus X_1\\
&\qquad \oplus X_2X_3X_4X_5 \oplus X_2X_4 \oplus X_2 \oplus X_3X_4 \oplus X_3X_5 \oplus X_4 \oplus X_5 \oplus 1\\
&\lambda_{5,10}(X_1, X_2, X_3, X_4, X_5)\\
&\quad = \; X_1X_2X_3X_4 \oplus X_1X_2X_3 \oplus X_1X_2X_4X_5 \oplus X_1X_2X_5 \oplus X_1X_3X_4 \oplus X_1X_4X_5 \oplus X_1 \oplus X_2X_3X_4X_5
\end{aligned}$$

$$\oplus X_2X_4X_5 \oplus X_2X_4 \oplus X_3X_4 \oplus X_3X_5 \oplus X_4X_5 \oplus 1$$

$\lambda_{5,11}(X_1, X_2, X_3, X_4, X_5)$

$$= X_1X_2X_3X_4 \oplus X_1X_2X_3X_5 \oplus X_1X_2X_3 \oplus X_1X_2X_4X_5 \oplus X_1X_2X_5 \oplus X_1X_3X_4X_5 \oplus X_1X_3X_5 \oplus X_1X_3 \oplus X_1X_5 \oplus X_1 \oplus X_2X_3X_4X_5 \oplus X_2X_4 \oplus X_2X_5 \oplus X_2 \oplus X_3X_5 \oplus X_4X_5 \oplus X_4 \oplus X_5$$

$\lambda_{5,12}(X_1, X_2, X_3, X_4, X_5)$

$$= X_1X_2X_3X_4 \oplus X_1X_2X_3X_5 \oplus X_1X_2X_4X_5 \oplus X_1X_3X_4X_5 \oplus X_1X_3X_5 \oplus X_1X_3 \oplus X_1X_4X_5 \oplus X_1X_4 \oplus X_1 \oplus X_2X_3X_4X_5 \oplus X_2X_3 \oplus X_2X_5 \oplus X_2 \oplus X_3X_4X_5 \oplus X_4X_5 \oplus X_4$$