University of Puerto Rico
Río Piedras Campus
Faculty of Natural Sciences
Department of Mathematics

On The Properties and Construction of Boolean Bent and Near-Bent Functions, and Their Applications to Error-Correcting Codes for NASA Deep-Space

By

José W. Velázquez Santiago

A THESIS SUBMITTED IN PARTIAL FULFILLMENT
OF THE REQUIREMENTS FOR THE DEGREE OF MASTER
OF PHILOSOPHY IN APPLIED MATHEMATICS AT THE UNIVERSITY
OF PUERTO RICO, RÍO PIEDRAS CAMPUS

May 16^{th}, 2022

APPROVED BY THE MASTER THESIS
COMMITTEE
IN PARTIAL FULFILLMENT OF THE REQUIREMENTS
FOR THE DEGREE OF
MASTER OF PHILOSOPHY IN APPLIED MATHEMATICS
AT THE UNIVERSITY OF PUERTO RICO

ADVISOR:

Heeralal Janwa, Ph.D.
University of Puerto Rico, Río Piedras

READERS:

Moises Delgado, Ph.D.
University of Puerto Rico, Cayey

Puhua Guan, Ph.D.
University of Puerto Rico, Rio Piedras

Abstract of M.S. Thesis Presented to the Graduate School
of the University of Puerto Rico, Río Piedras Campus in Partial Fulfillment of the
Requirements for the Degree of Master of Philosophy in Applied Mathematics

On The Properties and Construction of Boolean Bent and Near-Bent Functions, and Their Applications to Error-Correcting Codes for NASA Deep-Space

By

José W. Velázquez Santiago

May, 2022

A THESIS SUBMITTED IN PARTIAL FULFILLMENT
OF THE REQUIREMENTS FOR THE DEGREE OF MASTER
OF PHILOSOPHY IN APPLIED MATHEMATICS AT THE UNIVERSITY
OF PUERTO RICO, RÍO PIEDRAS CAMPUS

Abstract

In this investigation, we research the properties of highly nonlinear vectorial Boolean functions in m variables and their connections to good error-correcting codes. We focus on "bent" and "near-bent" functions, which achieve maximum nonlinearity for m even and odd, respectively. These functions $f : \mathbb{F}_{2^m} \to \mathbb{F}_{2^k}$ are defined via their Walsh-Hadamard spectrum meeting certain conditions. For the Boolean function cases, Gold (1968), Kasami (1971) and later Dillon (1999) and Dobbertin (1999) have showed properties under which near-bent functions derived from almost-bent functions are near-bent. These are Boolean functions in m variables of the form $Tr(x^d)$. We relate the constructed functions to corresponding error-correcting cyclic codes as per Janwa and Wilson's work on "Hyperplane sections of Fermat varieties in P^3 in char. 2 and some applications to cyclic codes" in 1993. We use the defining set of a cyclic code with "y" roots of the form $\{1, d_1, d_2, \ldots, d_{y-1}\}$ to construct the codes computationally. The entries of the defining set correspond to the exponents of the Boolean power functions considered. The conditions needed for these functions to be bent and near-bent are similar to the conditions needed to construct two-error-correcting codes through the defining set. The main exponents considered for the construction of these functions are the Gold and Kasami-Welch exponents ($2^l + 1, 2^{2l} - 2^l + 1$ respectively). We use cyclotomic coset analysis modulo $2^m - 1$ on the Gold and Kasami-Welch exponents used for these functions. We identify theorems related to the distribution of the Gold and Kasami-Welch exponents in the cyclotomic cosets. These theorems are then used to present a new proof of results by Yoshiara on the enumeration of non CCZ-equivalent Gold and Kasami-Welch trace Boolean

near-bent functions. These theorems consider slightly different restrictions on the exponents to the ones considered by Yoshiara.

Furthermore, we analyze and generalize theorems on the Gold and Kasami-Welch bent and near-bent functions. We identify a pattern in the relationship between exponents that led to near-bent/almost-bent (AB) functions. Various authors have studied and generalized the conditions under which $Tr(\alpha^i x^d)$ is a bent function where d is the Gold exponent . We present a conjecture on the exponents of α that lead to non-bent functions (and consequently, those that do). This is based on computational analysis of bent functions constructed through the algorithms we present. Various algorithms are constructed that generate these functions, and tables are obtained, which are used to establish our theorems and conjectures. Tables with the Gold bent function construction exception cases for up to 24 variables as well as some Kasami-Welch functions in six and 12 variables that support these conjectures are showed.

We also compare the developed functions and codes to well-known theorems and conjectures presented by Ding (2016), McGuire (2004), Calderbank (1984), Goethals (1979), and others on the weight distribution of the dual codes. Three weight dual codes are known to be associated with two-error-correcting codes. The distribution of the weights of the dual of the codes over $\mathbb{F}_{2^m}$ generated by the method above is conjectured to have the form $[2^{m-1} - a$, 2^{m-1}, $2^{m-1} + a]$ as presented by McGuire. We generated codes for up to 13 variables, and all codes satisfied this weight distribution. Ding compiled a list of theorems on the exact weight distribution of these codes. We algorithmically applied these theorems to the codes constructed and found some codes that do not meet any of the criteria. However, these codes did meet the symmetric weight distribution criteria. An equivalence analysis was done for these codes to identify them with codes from known theorems. We further study and classify cyclic codes in two, three and four roots based on their weight distributions. These are constructed by using combinations of APN/AB and bent exponents.

An LDPC Code analysis approach was implemented to codes constructed from the selected functions/codes in the work above. Bayesian belief propagation analysis over networks produced by these codes was done via Tanner graphs constructed from these codes, and analysis was done to determine the generated codes' coding gain. The high code rates are ideal for very strict bandwidth requirements. We utilize Neal's algorithm [53] to transmit encoded messages by utilizing our proposed codes via bent and near-bent functions. These codes are our unique results, and they show comparable performance to protograph based codes, Quasi-cyclic based codes, Turbo codes, and AR4JA codes. Our codes have improved or competitive performance for the SNR values in the range $[0, 0.75]$ and relative coding gain improvements of over 0.50 dB.

ACKNOWLEDGMENTS

I would like to thank professor Heeralal Janwa for his guidance on this thesis and Professor Moises Delgado for introducing me to the world of research in mathematics.
This research was supported by NASA PR Space Grant No. 80NSSC20M0052 and NNX15AI11H

Contents

Abstract 3
ACKNOWLEDGMENTS 6
1. Introduction 9
1.1. Boolean Functions 9
1.2. Error-Correcting Codes 12
1.3. Cyclotomic Coset Analysis of Cyclic Codes and Boolean Functions 13
1.4. Low-Density-Parity-Check Codes 14
2. Construction of Bent and Near-Bent Functions 16
2.1. Construction of Bent Functions 16
2.2. Construction of Near-Bent functions 18
3. Error-Correcting Code Construction and Almost-Bent/APN Exponents 20
3.1. Error-Correcting Codes and Minimum Distance Computations 20
4. Cyclotomic Coset Analysis of the Gold and Kasami-Welch Exponents 25
4.1. CCZ and Cyclotomic equivalence of Boolean Functions 26
4.2. Distribution of the Gold and Kasami-Welch Exponents in a Cyclotomic Coset 27
4.3. New proof for Yoshiara's result on the CCZ-equivalence of Gold and Kasami-Welch Near-Bent Functions 47
5. Improvement on Dillon and Dobbertin's Theorem on the Construction of Bent Gold and Kasami-Welch Functions 58
5.1. Gold Bent Function Construction 58
5.2. Further Results and Conjectures for the Gold Case 61
5.3. Kasami-Welch Bent Function Construction 67
6. Weight Distribution of Dual Cyclic Codes and Some Conjectures 68
6.1. Symmetric Weight Distribution of Generated Codes 68
6.2. Ding Weight Distribution Tables 69
6.3. Weight Distribution Classes of Cyclic Codes 71
7. LDPC Code Algorithms and Next-Generation NASA Code Construction Through Bent and Near-Bent Functions 77
7.1. Algorithms used 78
7.2. Methodology 80
7.3. SNR Performance Improvement Comparisons to Other Codes 82
References 85
8. Appendix:Tables 89
9. Appendix:Figures 118
10. Appendix:Algorithms 129

LIST OF SYMBOLS

$\mathbb{Z}$, set of integers

$\mathbb{F}_{2^m}$, $GF(2^m)$ field of order 2^m

$\mathbb{F}_2^m$, m-dimensional vector space of 2 elements

$\mathbb{F}_{2^m}^*$, multiplicative group (without the zero element)

$\mathcal{C}$, Linear Code

$[n, n-m, d(C)]$, binary linear code $\mathcal{C}$ of block length n, dimension m and minimum distance $d(\mathcal{C})$

R, rate of the code $\mathcal{C}$

$G_{\mathcal{C}}$, generator matrix of the code $\mathcal{C}$

$H_{\mathcal{C}}$, parity check matrix of the code $\mathcal{C}$

$C(a)$, 2-cyclotomic coset that contains the integer a

SNR, Signal-to-noise Ratio a

BER, Bit error rate

$(m, n) = e$, greatest common divisor of m, n is e

$Tr_k^m(x)$, the trace function on x from $\mathbb{F}_{2^m} \to \mathbb{F}_{2^k}$

$Tr(x)$, the trace function on x from $\mathbb{F}_{2^m} \to \mathbb{F}_2$

APN, Almost Perfect-Nonlinear

AB, Almost-bent function

CHAPTER 1 Background

1. Introduction

1.1. **Boolean Functions.** A vectorial Boolean function of size k in m variables (denoted as an (m, k) Boolean function) is defined as $f : \mathbb{F}_{2^m} \to \mathbb{F}_{2^k}$ [15]. The study of these functions has been a central focus in the development of many mathematical branches, such as the development of error-correcting codes and cryptographic codes. In 1993, Janwa and Wilson [41] showed that the conditions needed to construct two-error-correcting cyclic codes matched the conditions needed for (m, m) Boolean functions to be APN. These results were later generalized by Rodier et al. [57]. The high nonlinearity of these functions is also important for cryptographic applications, as they have high resistance to differential cryptanalytic attacks [19]. Maximizing the nonlinearity property of Boolean functions has been studied since Rothaus proposed the concept of "bent" functions in 1976 [59]. Rothaus defined a bent function in m even number of variables f as a function from $\mathbb{F}_{2^m}$ to $\mathbb{F}_2$ where the Fourier coefficients of $(-1)^{f(x)}$ are ± 1. Other equivalent definitions of these functions have been proposed, an important one being the Boolean functions in m (even) variables that achieve maximum nonlinearity [13]. However, the Walsh-Transform definition is used in this work to computationally verify bent-ness:

Definition 1 (Carlet [15] Boolean Bent Function)**.** Let f be a Boolean function in m (even) variables. Then, f is bent if and only if:

$$|\hat{F}(\omega)| = 2^{\frac{m}{2}} \ \forall \omega \in \mathbb{F}_{2^m} \ .$$

Where the Walsh-Hadamard transform is defined as:

Definition 2 (Carlet [15] Walsh-Hadamard Transform)**.** The Walsh-Hadamard transform of a Boolean function f in m variables at the value $\omega \in F_{2^m}$ is given by:

$$\hat{F}(\omega) = \sum_{x \in \mathbb{F}_{2^m}} (-1)^{\omega * x + f(x)}. \tag{1}$$

A similar definition is given for near-bent functions, where the number of variables m is odd.

Definition 3 (Carlet [15] Boolean Near-Bent Function)**.** Let f be a Boolean function in m (odd) variables. Then, f is near-bent if:

$$|\hat{F}(\omega)| \in = \{0, 2^{\frac{m+1}{2}}\} \ \forall \omega \in \mathbb{F}_{2^m} \ .$$

The nonlinearity these functions achieve is $2^{m-1} - 2^{\frac{m}{2}-1}, 2^{m-1} - 2^{\frac{m-1}{2}}$ respectively. The concept introduced by Rothaus was then generalized to the vectorial Boolean case by Nyberg in 1991 [55].

Vectorial bent functions are those (m, k) (m even) Boolean functions who have maximal nonlinearity [15]. A vectorial Boolean function f is bent if for every $\alpha \in \mathbb{F}^*_{2^m}$ each of its component functions $Tr(\alpha f)$ is also bent [16]. The component functions are defined as:

Definition 4 (Carlet [15] Component Function)**.** For a vectorial Boolean function $F : \mathbb{F}_{2^m} \to \mathbb{F}_{2^k}, 1 \leq k \leq m$, its component functions are denoted by:

$$Tr(v * F(x)), v \in \mathbb{F}_{2^k}^* \tag{2}$$

These trace functions are defined as follows:

Definition 5 (Gong [72] Trace Function)**.** For m and k integers such that $k|m$, the trace of an element $x \in \mathbb{F}_{2^m}$ is denoted by the vectorial Boolean function

$$Tr_k^m(x) : \mathbb{F}_{2^m} \to \mathbb{F}_{2^k}$$

such that:

$$Tr_k^m(x) = x + x^q + \cdots + x^{q^{l-1}}, q = 2^k, l = \frac{m}{k} \tag{3}$$

When $k = 1$, we write Tr(x).

Nyberg proved that these functions only exist whenever $k \leq \frac{m}{2}$ [55]. For the odd variable case, AB functions are defined as those from $\mathbb{F}_{2^m} \to \mathbb{F}_{2^m}$ that have maximal nonlinearity [15]. They are directly related to the near-bent functions, as a (m, m) function is AB if and only if each of its component functions is near-bent. This is easy to see as the nonlinearity of the AB functions must be maximal ($2^{m-1} - 2^{\frac{m-1}{2}}$), which is only possible if all its component functions have the same nonlinearity. AB functions offer optimal resistance to linear cryptanalysis, which leads to optimal resistance to differential cryptanalysis, although the converse is not always true. That is to say, AB functions are APN, but not all APN functions are AB [10]. Linear and differential cryptanalysis resistance are defined as follows:

Definition 6 (Canteaut [10] Measure of linear crypt-analysis resistance)**.** For a function $f : \mathbb{F}_{2^m} \to \mathbb{F}_{2^m}$, the linear cryptanalysis resistance is measured by:
$\lambda_f(a, b) = |\#\{x \in \mathbb{F}_{2^m}, a \cdot x + b \cdot f(x) = 0\} - 2^{m-1}|$
$\cdot$ is the dot product on $\mathbb{F}_{2^m}$ and $a, b \in F_{2^m}$
When $\lambda_f(a, b) = 2^{\frac{m-1}{2}}$ f is called AB. [10]

Definition 7 (Canteaut [10] Measure of differential cryptanalysis resistance)**.** For a function $f : \mathbb{F}_{2^m} \to \mathbb{F}_{2^m}$, $a, b \in \mathbb{F}_{2^m}, a \neq 0$, the differential cryptanalysis resistance is measured by:
$\delta(a, b) = \#\{x \in \mathbb{F}_{2^m} | f(x + a) + f(x) = b\}$.
When $\delta(a, b) = 2$ f is APN.

Definition 8 (Nyberg [56] Differential Uniformity)**.** For a function $f : \mathbb{F}_{2^m} \to \mathbb{F}_{2^m}$, $a, b \in \mathbb{F}_{2^m}, a \neq 0$, the differential uniformity is defined as:
$\max_{a \neq 0, b} \delta(a, b)$
where $\delta(a, b) = \#\{x \in \mathbb{F}_{2^m} | f(x + a) + f(x) = b\}$

Power functions	Exponent	Conditions	Exponent Type
Gold	2^l+1	$(l,m)=1$	APN/AB
Kasami-Welch	$2^{2l}-2^l+1$	$(l,m)=1$	APN/AB
Niho (even)	$2^r+2^{\frac{r}{2}}-1$	m odd, r even	APN/AB
Niho (odd)	$2^r+2^{\frac{3r+1}{2}}-1$	m odd, r odd	APN/AB
Inverse	2^m-2	m odd	APN
Dobbertin	$2^{\frac{4m}{5}}+2^{\frac{3m}{5}}+2^{\frac{2m}{5}}+2^{\frac{m}{5}}-1$	m divisible by 5	APN
Welch	2^r+3	m odd	APN/AB
Nyberg	$2^{2l}+2^l-1$	$4l\equiv 3 \pmod m$	APN

TABLE 1. Exponents for the power function $f(x)=x^d$ over $\mathbb{F}_{2^m}$ such that f is APN or AB. When m is odd, $m=2r+1$ [28, 32, 46]

Janwa and Wilson in [41] considered the Gold and Kasami-Welch exponents and when these led to APN functions and two-error-correcting codes. These exponents were proven to be the only ones such that the monomial power Boolean function $f(x)=x^d$ is exceptional APN [39]. The Gold and Kasami-Welch exponents are also used in well-known constructions for bent and near-bent trace Boolean functions. The construction of the near-bent functions $Tr(x^{2^l+1}), Tr(x^{2^{2l}-2^l+1})$ was studied by various authors such as Kasami, Dillon and Dobbertin [44, 25, 21]. The bent functions of the form $Tr(\alpha x^{2^l+1}), Tr(\alpha x^{2^{2l}-2^l+1})$, for $\alpha \in \mathbb{F}_{2^m}^*$ a primitive element in the field, have been studied by various authors such as Dillon and Dobbertin in [22], Charpin [17], Hu [40], Khoo [45] and Ma [48]. A recurring condition needed for these functions to be bent/near-bent is that $(m,l)=1$, which is the same condition needed for the corresponding vectorial Boolean functions to be APN and their cyclic codes (as defined in [41]) to be two-error-correcting.

The classification of bent and near-bent functions is an important problem. The current enumeration of bent functions is not exact, and known bounds for this could still be improved, as suggested by Carlet in [14]. Similarly, finding a tighter lower bound is an open problem [15]. In this work, we are interested in the nonlinearity and differential uniformity properties of these functions, and the most general form of equivalence between Boolean functions that preserve these properties is the CCZ-equivalence [11]. Yoshiara showed in [71] an equivalent notion of equivalence for Boolean APN power functions:

Theorem 1 (Yoshiara [71])**.** Let $f_d(x)=x^d, f_e(x)=x^e$ be power APN functions on $\mathbb{F}_{2^m}$, with $m\geq 3$. Then f_d, f_e are CCZ-equivalent if and only if 1) $e\equiv d2^a \pmod{2^m-1}$ for some integer $a\in[0,m-1]$ or 2) m is odd and $ed\equiv 2^a \pmod{2^m-1}$ for an integer $a\in[0,m-1]$.

From this theorem, the concept of "Cyclotomic-equivalent" Boolean functions is considered.

Definition 9 (Budaghyan [5] Cyclotomic equivalence)**.** Let $f(x)=x^k$, and $g(x)=x^l$, $f,g:\mathbb{F}_{2^m}\to\mathbb{F}_{2^m}$, the functions are cyclotomic-equivalent if there exists an integer $0\leq a<m$ such that $l\equiv k2^a \pmod{2^m-1}$ or $kl\equiv 2^a \pmod{2^m-1}$, with $(k,2^m-1)=1$.

1.2. **Error-Correcting Codes.** In 1947, Richard Hamming constructed the first error-correcting codes, dubbed "Hamming Codes" [34]. The purpose of error-correcting codes is to add redundancy to the information to detect and correct errors. Linear error-correcting codes are vector sub spaces of dimension k a field [42]. These are codes of block length n, dimension $k = n-m$ and a minimum distance $d(C)$. Thus, we can construct codewords as an element $x \in \mathbb{F}_2^n$. The minimum distance of a codes is related to its error-correcting capability by the relation $d = 2(t)+1$, where t is the number of errors the code can correct [42]. This is defined as the minimum number of entries in which any two codewords in the code can differ [42]. For linear codes, this value is the same as the minimum weight of the code (minimum number of nonzero entries a nonzero codeword in the code can have). This can be easily seen as $d(x,y) = D(C) = d(x-y,0) = w(x-y)$ for $x, y \in C$. Thus, you can find a codeword with weight the same as the minimum distance. Alternatively, if you had a codeword "z" with weight less than the minimum distance then you would have: $w(z) = d(z,0) < D(C)$ which is a contradiction. Linear codes can be represented through a $(m)xn$ (where $m = n-k$) parity check matrix H such that $x \in C$ if and only if $Hx^T = 0$ [42]. In this work, we consider the construction of cyclic codes. These are linear codes such that if $x = (a_0, a_1, \ldots, a_{n-1})$ is a codeword, then so is $y = (a_{n-1}, a_0, \ldots, a_{n-2})$ $a \in \mathbb{F}_2^m$. Define α as a primitive element in the field $\mathbb{F}_{2^m}$ (that is, an element of order 2^m-1) and define $m_i(x)$ as the minimal polynomial of α^i over $\mathbb{F}_2$. We can define a cyclic code (with two roots) of length 2^m-1 by constructing the generator polynomial $g(x) = m_1(x)m_d(x)$. We will denote these two root codes as C_m^d [41]. We call $\{d_0, d_1, d_2, \ldots, d_{y-1}\}$ the defining set of the cyclic code with y roots whose generator polynomial is $g(x) = m_{d_0}(x)m_{d_1}(x)m_{d_2}(x)\ldots m_{d_{y-1}}(x)$. Janwa and Wilson in [41] construct a parity check matrix for two root cyclic codes. Let $\alpha \in \mathbb{F}_{2^m}$ be a primitive element of the field and f some vectorial Boolean power function from $\mathbb{F}_{2^m}$ to $\mathbb{F}_{2^m}$, then, a parity check matrix of a two root cyclic code is:

$$\text{H'} = \begin{bmatrix} \alpha^{2^m-2} & \alpha^{2^m-3} & \ldots & \alpha^1 & \alpha^0 \\ f(\alpha^{2^m-2}) & f(\alpha^{2^m-3}) & \ldots & f(\alpha^1) & f(\alpha^0) \end{bmatrix}$$

Janwa and Wilson showed conditions for which this construction led to two-error-correcting codes [41]. They considered the highly nonlinear functions $f(x) = x^{2^l+1}, f(x) = x^{2^{2l}-2^l+1}$ (the Gold and Kasami-Welch functions respectively). To verify if this matrix corresponds to a two-error-correcting code, the linear combination of any four columns of H' must not equal the 0 vector. This is because otherwise there is a codeword of weight four, and hence the minimum distance is less than 5. We can consider this as a system of equations with two equations and four variables (where a column in H' = $(\alpha^i, f(\alpha^i))^T = (X, f(X))^T$). Thus, we get the following system of equations:

$$\begin{gathered} \text{X} + \text{Y} + \text{Z} + \text{W} = 0 \\ \text{f(X)} + \text{f(Y)} + \text{f(Z)} + \text{f(W)} = 0 \end{gathered}$$

We get:

$$\begin{gathered} \text{X} + \text{Y} + \text{Z} = \text{W} \\ \rightarrow \text{f(X)} + \text{f(Y)} + \text{f(Z)} + \text{f(X} + \text{Z} + \text{Y)} = 0 \end{gathered}$$

Note that for the second equation, the "trivial" solutions (X = Y, X = Z, Y = Z) are not possible as X,Y,Z correspond to distinct columns of the matrix H. As such we can factor out these solutions and study:

$$\phi_{f(X,Y,Z)} = \frac{f(X)+f(Y)+f(Z)+f(X+Z+Y)}{(X+Y)(X+Z)(Z+Y)} = 0$$

as per Janwa and Wilson [41] and later Ferard et al [26], the APN property of the function f is equivalent to verifying if the rational points of the algebraic surface X in a 3-dimensional space defined by $\phi_f(X, Y, Z)$ are all in the surface made by the planes X+Y = 0, X+Z=0, Y+Z = 0. That is, the APN property is equivalent to verifying the existence of the two-error-correcting code given $f = x^d$ where d is the Gold or Kasami-Welch exponent. Janwa and Wilson showed that this is true when $(m, l) = 1$. Rodier later generalized this problem for a general function f [57]. This construction can be extended for more roots. We can construct a cyclic code corresponding to a defining set: $\{d_0, d_1, d_2, \ldots, d_{y-1}\}$ for $\alpha \in F_{2^m}$ a primitive element in the field and $f_{d_i} = x^{d_i}$:

$$\text{H'} = \begin{bmatrix} \alpha^{2^m-2} & \alpha^{2^m-3} & \cdots & \alpha^1 & \alpha^0 \\ f_1(\alpha^{2^m-2}) & f_1(\alpha^{2^m-3}) & \cdots & f_1(\alpha^1) & f_1(\alpha^0) \\ f_{d_1}(\alpha^{2^m-2}) & f_{d_1}(\alpha^{2^m-3}) & \cdots & f_{d_1}(\alpha^1) & fd_1(\alpha^0) \\ . & . & \cdots & . & . \\ . & . & \cdots & . & . \\ . & . & \cdots & . & . \\ f_{d_{y-1}}(\alpha^{2^m-2}) & f_{d_{y-1}}(\alpha^{2^m-3}) & \cdots & f_{d_{y-1}}(\alpha^1) & f_{d_{y-1}}(\alpha^0) \end{bmatrix}$$

1.3. **Cyclotomic Coset Analysis of Cyclic Codes and Boolean Functions.** In this work, we focus on the construction of cyclic codes. Cyclic codes are those such that if $x = (a_0, a_1, \ldots, a_{n-1})$ is a codeword, then so is $y = (a_{n-1}, a_0, \ldots, a_{n-2})$, $a \in \mathbb{F}_2$. Define ω as a primitive element in the field $\mathbb{F}_{2^m}$ and define $m_i(x)$ as the minimal polynomial of ω^i over $\mathbb{F}_2$. We can define a cyclic code (with two roots) of length $2^m - 1$, dimension at least $2^m - 1 - 2m$ and minimum distance d(C) by constructing the generator polynomial $g(x) = m_1(x)m_d(x)$[41]. We denote these two root codes as C_m^d. The weight of a codeword $x = (a_0, a_1, \ldots, a_{n-1})$ is given by $\{\#i|a_i \neq 0\}$.

As mentioned in the previous subsection, we denote $\{d_0, d_1, d_2, \ldots, d_{y-1}\}$ the defining set of the cyclic code with y roots whose generator polynomial is $g(x) = m_{d_0}(x)m_{d_1}(x)m_{d_2}(x) \ldots m_{d_{y-1}}(x)$. Janwa and Wilson have shown in [41] that for a cyclic code in two roots (defining set $\{1, d_1\}$), if $f(x) = x^{d_1}$ satisfies (l,m) = 1 (and thus is APN), then the code is two-error-correcting. This same condition is present for the construction of near-bent functions of the form $Tr(x^{d_1})$ for d_1 corresponding to the Gold or Kasami-Welch exponents [15]. It is known that if $f(x) = x^d$, then $f(x)$ is AB if and only if $Tr(x^d)$ is near-bent [15]. Furthermore, if $f(x)$ is AB, then it is APN [10]. We note the following, based on definition 5, the Gold and Kasami-Welch near-bent functions in m variables have the form:

$$Tr(x^d) = x^d + x^{2d} + x^{4d} + \cdots + x^{(2^{m-1})d} \tag{4}$$

Consider the following definition for the 2-cyclotomic cosets (mod $2^m - 1$) from [69]:

Definition 10 (Wong [70] 2-Cyclotomic Coset of a)**.** Consider $a \in Z_{2^m-1}$, the 2-cyclotomic coset of size $j \pmod{2^m-1}$ that contains a $(C(a))$ is of the form:

$$\{a, 2a, 4a, \ldots, 2^{j-1}a\} \tag{5}$$

where j is some divisor of m and $2^j a \equiv a \pmod{2^m-1}$.

We refer to this as the cyclotomic coset of a $(C(a))$ for the rest of this paper. We call the smallest positive integer in $C(a)$ the cyclotomic coset representative of a and denote it by $CR(a)$.

$$Tr(x^d) = Tr(x^{2d}) = \cdots = Tr(x^{(2^{m-1})d}). \tag{6}$$

Now consider the following for the minimal polynomial of α^{d_i} of degree j_2 over $\mathbb{F}_2$:

$$m_{d_i}(x) = a_0 + a_1x^1 + a_2x^2 + \ldots x^{j2}. \tag{7}$$

Since the coefficients of the polynomial are 0s or 1s, if α^{d_i} is a root, then so is any α^y such that $y \in C(d_i)$ (say $y = \alpha^{2^k d_i}$, k an integer such that $0 < k < m$):

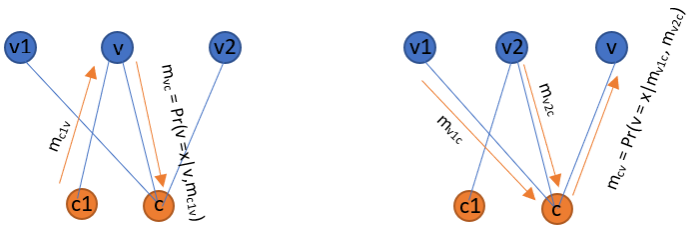
$$\begin{aligned} m_{d_i}(\alpha^{2^k d_i}) &= a_0 + a_1(\alpha^{2^k d_i})^1 + a_2(\alpha^{2^k d_i})^2 + \ldots (\alpha^{2^k d_0})^{j_2} \\ &= a_0 + a_1(\alpha^{d_i})^{2^k} + a_2(\alpha^{d_i})^{2*2^k} + \ldots (\alpha^{d_i})^{j_2*2^k} \\ &= (a_0 + a_1(\alpha^{d_i}) + a_2(\alpha^{d_i})^2 + \ldots (\alpha^{d_i})^{j_2})^{2^k} = (0)^{2^k} = 0 \end{aligned}$$

From [29] we can establish a clear relationship between the cyclotomic cosets and the roots of minimal polynomials. It is known that $X^{2^m} - X$ over $\mathbb{F}_{2^m}$ is the product of all minimal polynomials over $\mathbb{F}_2$ whose degree divides m. From corollary 3 in [29] we have that $X^{2^m-1} - 1 = \sum_s (m_s(X))$ which iterates over a set of cyclotomic coset representatives. Thus, the size of the cyclotomic cosets must be a divisor of m.

With these results, we reduce the number of exponents needed to construct the cyclic codes and the associated functions. We choose the smallest exponent of the cyclotomic coset (called the cyclotomic coset representative) for these purposes. We do this for the Gold and Kasami-Welch exponents by studying the cyclotomic cosets that contain these exponents for $0 < l < m$. Given this, we determine the number of non-equal Gold near-bent functions and determine an upper bound for the number of non-equal Kasami-Welch near-bent functions of the form $Tr(x^d)$.

1.4. **Low-Density-Parity-Check Codes.** Error-correction is an important concept for NASA deep-space applications. For NASA spacecraft, information is sent through millions of miles in space. Space behaves like an additive white Gaussian noise (AWGN) channel (and is thus used by multiple authors to test their codes) and thus introduces noise to the information sent [1, 9, 62]. Multiple coding schemes have been implemented to provide reliable communications for NASA deep-space applications. McEliece in [50] discussed the implementation of Reed-Solomon (RS) codes for NASA implementations. These have been later utilized in various NASA missions, such as the Mariner 9, Voyager, and Galileo spacecrafts [50, 68]. However, further research has been done on the efficiency and implementation of these codes, and newer codes that can further approach the Shannon channel capacity have been studied. These are the Turbo codes [37]. These codes have fast decoding and good performance for deep-space conditions. However, more recently, state of the art by NASA's

Jet Propulsion Laboratory (JPL) involves the use of LDPC codes. These are constructed by using the parity check matrix of a code as the adjacency matrix of a sparse Tanner (bipartite) graph [60]. These graphs are such that the nodes are divided into two classes that are not interconnected with each other. One class of nodes corresponds to the variable nodes and the other to the check nodes. LDPC codes have fast linear-time decoding through Bayesian belief propagation over sparse networks. Belief propagation works by passing probabilities (beliefs) along the nodes through the edges that connect them [62]. The message passed from a message node v to a check node c is the probability that v has a certain value given the observed value of v and all the values communicated to v in the previous round from check nodes other than c. On the other hand, the message passed from c to v is the probability that v has a certain value given all the messages passed to c in the previous rounds from message nodes other than v.



In the first round, the check node sends along the outgoing edges their log likelihoods conditioned on the observed values. If the channel is a binary symmetric code channel with error probability p, then the first message sent to all the check nodes adjacent to a message node is $ln(1-p)-ln(p)$ if the node value is 0 , and the negative of this if it is 1. In all subsequent rounds of the algorithm, a check node c sends to an adjacent message node v a likelihood according to the equation $ln(L(x_1+x_2+x_3+\dots|y_1,y_2,\dots)) = \ln((\frac{1+\prod_{i=1}^{l}(tanh(\frac{l_i}{2}))}{1-\prod_{i=1}^{l}(tanh(\frac{l_i}{2}))})$. A message node v sends to a check node its log likelihood conditioned on its observed value and on the incoming log likelihoods from adjacent check nodes other than c using relationship the relationship $ln(L(x|y_1,y_2,\dots)) = \sum_{i=1}^{d} ln(L(x|y_i))$. Denote $m_{vc}^{(l)}$ as the message passed from message node v to check node c at the l-th round. Define $m_{cv}^{(l)}$ similarly. At round 0, $m_{vc}^{(0)}$ is the log likelihood of the message node v conditioned on its observed value, which is independent of c. Denote this value by m_v. then the updated expectation for the messages under belief propagation are given by:

$$m_{vc}^{(l)} = m_v \text{ if } l = 0$$
$$m_{vc}^{(l)} = m_v + \sum_{c' \in c_v - \{c\}} m_{c'v}^{(l-1)} \text{ if } l > 0$$
$$m_{cv}^{(l)} = ln(\frac{1+\prod_{v' \in v_c - \{v\}} tanh(\frac{m_{v'c}^{(l)}}{2})}{1-\prod_{v' \in v_c - \{v\}} tanh(\frac{m_{v'c}^{(l)}}{2})}) \text{ [62]}$$

The analysis done by the algorithm traverses the edges of the graph. For small number of edges, if we set a number for the iterations, then these edges are traversed a constant

number of times. The number of edges is given by the degrees of the message node. Thus, the algorithm uses a number of operations that is linear in the number of message nodes [62].

LDPC codes compare favorably against Turbo codes [36]. Results show that they outperform Turbo codes for high code rates. According to Calzolari [9] next-generation NASA coding schemes require large coding gain, high spectral efficacy, and low complexity decoding. Coding gain is the difference between the $\frac{E_b}{N_0}$ (SNR) required to achieve a given BER in a coded system and the $\frac{E_b}{N_0}$ required to achieve the same BER in an uncoded system. According to Heegard [37], by the late 1960s, an improvement of 1 dB in coding gain is the equivalent of $1,000,000 in development and launch costs. For 2013, the value of a 1 dB improvement for Deep Space Networks is about $80,000,000. This is a significant cost for future missions, and as such, any improvements over current standards improve the cost-efficiency of deep-space missions. The spectral efficiency η is the average number of information bits transmitted per two-dimensional signaling interval of duration T. It is also measured as a ratio between the data rate, and the available bandwidth [9]. This is measured in bits/second/hertz. For low data rates and large available bandwidth, the spectral efficiency is not a problem, and code rates of $\frac{1}{2}, \frac{1}{6}$ have been considered for those parameters in traditional deep-space missions [9]. However, for scenarios such as Mars/Lunar Missions require high data rates (hundreds of megabits/second), while Mars exploration scenarios require medium data rates (tens of megabits/second). These scenarios also require codes that work at very low SNR values [9]. It is known that the code rate is inversely proportional to bandwidth expansion [9]. As such, codes with very high code rates are sought for specific bandwidth efficiency constraints [9]. In this work, we consider the application of LDPC code algorithms to codes from the parity check matrices of cyclic codes in two roots. These codes are constructed as Janwa and Wilson do in [41] via the utilization of highly nonlinear functions. We consider Boolean bent and near-bent functions as our highly nonlinear functions for this construction. We propose the following construction:

Proposition 1. Codes that meet NASA criteria for next-generation channel decoding

Consider the set of near-bent exponents $S := \{j | j \notin C(i) \forall i \in S, i \neq j\}$, α a primitive element in $\mathbb{F}_{2^m}$, $f_i(x) = x^{s_i}, s_i \in S$. Consider the $2m$ x n matrix of the form:

$$\text{H'} = \begin{bmatrix} \alpha^{2^m-2} & \alpha^{2^m-3} & \cdots & \alpha^1 & \alpha^0 \\ f_i(\alpha^{2^m-2}) & f_i(\alpha^{2^m-3}) & \cdots & f_i(\alpha^1) & f_i(\alpha^0) \end{bmatrix}$$

Then, the resulting code will be a length $n = 2^m - 1$, k $\geq 2^m - 1 - 2m$ and d(C) = 5. The graph constructed by utilizing this matrix as its adjacency matrix will correspond to a code that meets at least two of NASA criteria for the next-generation channel decoding for $m \geq 7$ and small SNR values.

Our implementation and results leading to this proposition are discussed in Section 7.

2. Construction of Bent and Near-Bent Functions

2.1. **Construction of Bent Functions.** In this investigation, we construct algorithms that generate a Boolean function $g : \mathbb{F}_{2^m} \to \mathbb{F}_2$ in m variables of the Gold and Kasami-Welch

type as constructed by Dillon and Dobbertin in [22]. These exponents are known for their corresponding APN power functions ($f : \mathbb{F}_{2^m} \to \mathbb{F}_{2^m}$) of the form $f(x) = x^d$. The exponents are defined as $d = 2^l + 1, 2^{2l} - 2^l + 1$ respectively, with (l,m) = 1 a necessary condition for these functions to be APN and the construction of two-error-correcting codes as per Janwa and Wilson [41]. To generate these functions, we construct algorithms (which will be shown in the following subsections) that define the Boolean functions and iterate over distinct Gold and Kasami-Welch exponents over the field $\mathbb{F}_{2^m}$. We utilize definitions 1, 2 and 5.

Dillon and Dobbertin in [22] analyzed the Fourier coefficients of $(-1)^{Tr(\lambda x^d)}$ and obtained conditions under which the Kasami-Welch and Gold Boolean functions are bent:

Proposition 2 ([22] Dillon and Dobbertin Gold Bent Functions)**.** Let L = $\mathbb{F}_{2^m}$ and let (l,m) = 1. For:

$$\lambda \in L^*, s^{\lambda}_{2^l+1}(x) = Tr(\lambda x^{2^l+1}) \text{ and } \rho^{\lambda}_{2^l+1}(x) = (-1)^{s^{\lambda}_{2^l+1}(x)}$$

if m is even and $\lambda \in L^*$ is not a cube, then:

$$\hat{\rho}_{2^l+1}(\alpha) = \pm 1, \ \forall \alpha \in L,$$

i.e $s^{\lambda}_{2^l+1}(x)$ is bent.

Theorem 2 ([22] Dillon and Dobbertin Kasami-Welch Bent Functions)**.** Let $L = \mathbb{F}_{2^m}$, where m is even but not divisible by six and let $K = \mathbb{F}_{2^2}$. Let $d = 2^{2l} - 2^l + 1$, (l,m) = 1. For:

$$\lambda \in K^*, s^{\lambda}_d = Tr(\lambda x^d) \text{ and let } \rho^{\lambda}_d = (-1)^{s^{\lambda}_d} = \phi(\lambda x^d)$$

then:

$$1) \text{ If } \lambda \neq 1, \text{then } s^{\lambda}_d \text{ is bent i.e } \hat{\rho}^{\lambda}_d(\beta) = \pm 1, \forall \beta \in L$$

$$2) \text{ If } \ \lambda = 1, \text{then } \hat{\rho}_d \text{ takes just three values } \{-2, 0, 2\}, \text{ and}$$

$$\text{supp } \hat{\rho}_d = L \setminus K^* \{\Gamma(\theta) : Tr_{L/K}(\theta^{-d}) \neq 0\}$$

where $\Gamma(z) = T(z^{2^{3k}+1})/z^{2^{2k}(2^k+1)}$

and $\ T(z) = z + z^{2^k} + z^{2^{2k}}$.

Later, this was re-stated in [15] by reducing the divisibility criteria to m not divisible by three and $\lambda \in \mathbb{F}_{2^m}$ a non-cube. To study these functions, we take α as a primitive element in $\mathbb{F}_{2^m}$ and consider $Tr(\alpha^i x^d)$ where we iterate over i for each Gold and Kasami-Welch exponent. We state the following lemma as an observation from this notation:

Lemma 1 (Primitive Element Representation of a k-th Power Element in $\mathbb{F}_{2^m}$)**.** Let $\alpha \in \mathbb{F}_{2^m}$ be a primitive in the field and β a k-th power of another element in the field. Then, $\beta \equiv \alpha^{ik} \pmod{2^m - 1}$ where $i, k \in \mathbb{Z}$.

Proof: Let $\beta \equiv \gamma^k \pmod{2^m - 1} \in \mathbb{F}_{2^m}$. Since α is a primitive element in the field, then one of its powers will be equivalent to γ. Let us say $\alpha^i \equiv \gamma \pmod{2^m - 1}$. Then we have: $\beta \equiv \gamma^k \equiv (\alpha^i)^k \equiv \alpha^{ik} \pmod{2^m - 1}$

Thus, if $\beta \in \mathbb{F}_{2^m}$ is a k-th power of another element, then you can represent it as α^j where j is a multiple of k.

□

This lemma is utilized in Subsections 5.1 and 5.3 for analyzing the results of our algorithms and constructing our tables and conjectures.

To study these functions, we consider the following construction. Let $\alpha \in \mathbb{F}_{2^m}$ be a primitive element. Then we construct the function $Tr(\alpha^i x^d)$ for $0 < i < 2^m - 1$ and d one of the exponents under consideration. We iterate over all possible values of i for a set exponent d. According to Carlet, α^i must be a non-cube and m not divisible by three for the Kasami-Welch case to be bent. As per Lemma 1, this would mean that whenever i is not a multiple of three, then the function is bent (given that the other two conditions are met). This observation leads us to divide the set of exponents of α into sets of multiples. Specifically, multiples of a number that divide the order of the field. For multiples of 3, it is easy to see that 3 divides the order of $\mathbb{F}_{2^m}$ as m is even and thus, $2^m \equiv 1 \pmod 3 \rightarrow 2^m - 1 \equiv 0 \pmod 3$. This significantly decreases the number of computations, as otherwise, we are considering $2^m - 2$ functions for every exponent d. The analysis of these functions is done in section 5.

2.2. **Construction of Near-Bent functions.** Near-bent functions are defined for m (odd) number of variables. There are known constructions of near-bent functions, namely the Gold and Kasami-Welch functions $Tr(x^{2^l+1}), Tr(x^{2^{2l}-2^l+1})$ [31, 44, 25, 21]. As mentioned previously, near-bent functions are related to AB functions in that the component functions of the AB function are near-bent. For an AB function $F : \mathbb{F}_{2^m} \rightarrow \mathbb{F}_{2^m}$, $F(x) = x^d$, then $Tr(\lambda x^d)$ is near-bent for $\lambda \in \mathbb{F}_{2^m}^*$ [15]. Specifically, for $\lambda = 1$, we have that $Tr(x^d)$ must be near-bent. This brings the question as to which exponents lead to AB (and thus, near-bent) functions. In Table 1 a list of known APN and AB exponents is showed. Here we observe that while all AB functions are APN, not all APN functions are AB. In 1999, Canteaut established a theorem that determines when an APN function is also AB [10]. First, they define the 2^l divisibility of a code:

Definition 11 (Canteaut [10] 2^l divisibility of a cyclic code)**.** A binary cyclic code C is said to be 2^l divisible if the weight of any codeword in the code is divisible by 2^l. [10]

Theorem 3 (Canteaut [10] Conditions for APN function to be almost-bent)**.** Let m be an odd integer and $f : \mathbb{F}_{2^m} \rightarrow \mathbb{F}_{2^m}$ with $\lambda_f \neq 2^{m-1}$. Then f is almost-bent if and only if f is APN and $C_m^{d\perp}$ (dual code of C_m^d) is $2^{\frac{m-1}{2}}$ divisible.

Janwa and Wilson considered the APN property of Boolean functions in their 1993 paper [41]. The minimum distance of the cyclic codes depends on the APN property. However, AB functions achieve maximum nonlinearity by definition, whereas APN function do not necessarily. Thus, we expect some differences in the cyclic codes constructed from APN functions that are not also AB. In Section 6 we discuss the weight distribution of the codes generated via the application of APN/AB/Bent functions for cyclic codes with two, three and four roots and note the impact of functions being APN but not AB.

We construct algorithms that generate a Boolean function $g : \mathbb{F}_{2^m} \rightarrow F_2$ in m variables of the form $Tr(x^d)$. We iterate over a pre-generated list of exponents (APN, AB, bent) and verify if the functions are near-bent. To do this we use the following algorithm:

Algorithm 1. [67]

```
def NearBent(f):
  dim = f.nvariables()
  w = f.walsh_hadamard_transform(), k = (dim +1)/2
  for i in range(2^dim):
    if abs(w[i]) != 0 and abs(w[i]) != 2^k:
      return "Not Near-Bent"
    else:
      i = i
  return "Near-Bent"
```

The input is a Boolean function "f" in m variables and the algorithm determines its Walsh-Hadamard transform spectrum. The Boolean function input is done via the BooleanFunction(f(x)) command, where $f(x)$ is a polynomial over $\mathbb{F}_{2^m}$ and the result is the Boolean function given by $Tr(f(x))$ [49]. Then it verifies if the function is near-bent by iterating over the spectrum to check if it meets the definition. Trace properties of these functions can be exploited to find equal functions. For example, for $f : \mathbb{F}_{2^3} \to \mathbb{F}_{2^3}$, $f(x) = Tr(x^3)x^3 + x^6 + x^{12} = x^6 + x^{12} + x^3 = Tr(x^{12})$. There is also the concept of equivalence, which we consider in Section 4. This concept is important as the enumeration of bent and near-bent functions is an open problem [15]. Classifying these functions in equivalent classes is one way to determine this.

Chapter 2: Results

3. Error-Correcting Code Construction and Almost-Bent/APN Exponents

3.1. **Error-Correcting Codes and Minimum Distance Computations.** In [67], we constructed an algorithm that builds cyclic codes in two roots considering all the bent, near-bent and APN exponents studied. The length of these codes ranged from $n = 2^4 - 1$ to $n = 2^{12} - 1$. We utilized this construction (and slight modifications to it) to study the properties of cyclic-codes in two roots. One of the main properties we studied was the weight distribution of these codes. However, note that since these codes have minimum dimension $2^m - 1 - 2m$, then as m increases, the dimension (and hence the total number of codewords) increases much faster, quickly approaching n. For example, in six variables, the corresponding cyclic code has $2^{2^6-1-2(6)} = 2^{51}$ codewords at minimum. Thus, it is not viable to compute the weight distribution of these codes directly. Instead, the weight distribution of their dual codes is computed. These will have at most 2^{2m} codewords, which for $m = 12$ is 2^{24} variables, a much smaller number than the total codewords of the code for six variables. The weight distribution of the dual codes can be related to the weight distribution of the original code by the MacWilliams identity [42]. For cyclic codes of length $2^m - 1$, the weight distribution of a code can be represented as the vector:

$$(A_0, A_1, \ldots, A_n)$$

Where A_w is the number of codewords of weight w. The weight enumerator polynomial, as defined in [42], is given by:

Definition 12 (Justesen [42] Weight Enumerator Polynomial)**.** The weight enumerator polynomial $A(x)$ of a linear code C is defined as:

$$A(x) = \sum_{w=0}^{2^m-1}(A_w x^w)$$

where the weight distribution of the code is represented by the vector $(A_0, A_1, \ldots, A_n)$

The MacWilliams identity is given by:

Theorem 4 (Justesen [42] MacWilliams Identity)**.** If $A(x)$ is the weight enumerator polynomial of a binary $(2^m - 1, k)$ code C, then the weight enumerator polynomial $B(x)$ of the dual code $C^\perp$ is given by :

$$B(x) = 2^{-k}(1 + x)^{2^m-1}A(\tfrac{1-x}{1+x}).$$

Thus, studying the weight distribution of the dual code is equivalent to studying the weight distribution of the original code. The algorithm used to construct these codes follows:

Algorithm 2. [67] {`def Ccode2r(`m`):`

```
N = []
I = []
D = []
W = []
n = 2^m - 1
L = []
```

```
    SLLL = []
    for i in range(0,len(SecondRoot[m])):
      C = codes.CyclicCode(field = GF(2), length = n, D = [1,SecondRoot[m][i][0]])
      L.append(["Root is", SecondRoot[m][i][0]])
      h = C.check_polynomial()
      DC = codes.CyclicCode(generator_pol = h.reverse(), length = n)
      sd = DC.spectrum(algorithm = "binary")
      a = [y for y in sd if y != 0]
      g = 0
      c = 0
      for w in range(0,n+1):
        g = g + sd[w]x^w
      for j in range(0,len(SLLL)):
        if g == SLLL[j][0]:
          SLLL[j].append([1,SecondRoot[m][i][0]])
          c = c + 1
          break
      if c == 0:
        SLLL.append([g,[1,SecondRoot[m][i][0]]])
      print("\n", C , ",","Distance is", C.minimum_distance(algorithm = "guava"),
" roots are:", 1, SecondRoot[m][i][0])
      if set(SecondRoot[m][i]) & set(G) != set():
        print("is Gold")
        L.append(", Is Gold,")
      if set(SecondRoot[m][i]) & set(K) != set():
        print("is Kasami-Welch")
        L.append(", Is Kasami-Welch,")
      if m%2 == 1:
        I.append(2^m - 2)
        W.append(2^((m - 1)/2 + 3))
        if set(SecondRoot[m][i]) & set(I) != set():
          print("is Inverse")
          L.append(", Is Inverse,")
        if set(SecondRoot[m][i]) & set(W) != set():
          print("is Welch")
          L.append(", Is Welch,")
        if m%4 == 1:
          N.append(2^((m-1)/2) + 2^((m-1)/4) - 1)
          if set(SecondRoot[m][i]) & set(N) != set():
            print("is Niho even case")
            L.append(", Is Niho Even Case,")
        if m%4 == 3:
          N.append(2^((m - 1)/2) + 2^((3 * m - 1)/4) - 1)
          if set(SecondRoot[m][i]) & set(N) != set():
```

```
            print("is Niho odd case")
            L.append(", Is Niho Odd Case,")
          if m% 5 ==0:
            D.append(2^((4 * m)/5) + 2^((3 * m)/5) + 2^((2 * m)/5) + 2^((m)/5) - 1)
            if set(SecondRoot[m][i]) & set(D) != set():
              print("is Dobbertin")
              L.append(", Is Dobbertin,")
        L.append(["Number of nonzero weights is", len(a) - 1])
        print("Weight Enumerator polynomial of the Dual Code is:",g, ", Coefficients
are:",a, ", Number of nonzero codeword weights is", len(a) - 1)
    print("\n", SLLL)
```

The algorithms take as an input "m" the number of variables considered. The corresponding code will be of length $n = 2^m - 1$. The first four lines define empty lists in which we store the non-Gold and Kasami-Welch exponents considered. This is done as these exponents depend directly on the value of m. The Gold and Kasami-Welch exponents are assigned to the lists G,K that were pre-generated. Next, the algorithm defines the length of the code n, and two additional empty lists, "L" and "SLLL." The list L will store the second root of the defining set, to which APN exponent it corresponds and the number of nonzero weights of the corresponding cyclic code. The list SLLL will store defining sets of cyclic codes with the generator polynomial of the respective dual code. This is such that all defining sets that lead to cyclic codes with the same weight distribution will be grouped together. Line 8 of the algorithm begins a for loop that iterated over a pre-generated list of exponents (called "SecondRoot"), which are used to construct the cyclic code. SecondRoot is a list that contains the roots and their cyclotomic cosets as a sublist. SecondRoot[m] selects the cosets modulo, SecondRoot[m][i] the specific cyclotomic coset and SecondRoot[m][i][0] is the representative of that coset. The roots are appended to the list L; the check polynomial is computed and used to construct the dual code. Then, the weight distribution (or spectrum) of the dual code is computed. A list "a" is constructed such that it takes all nonzero entries in the weight distribution of the code. A variable "g" is used to store the weight enumerator polynomial and a counter variable "c" is used to help store the weight enumerator polynomial in the following loop. Line 17 begins a for loop that stores in g the weight enumerator polynomial of the dual code according to Definition 12. Lines 19-25 include the for loop used to construct the list SLLL. First, the algorithm verifies if the value stored in g matches the first entry in SLLL, if it does, then it will append the defining set of the current code as it shares a weight enumerator polynomial with the others in the list SLLL. If it does not, then we check the second weight enumerator polynomial entry in the list SLLL and so on. Line 24 verifies if the counter variable is 0, if it is, then g is not in SLLL and thus is added (with its corresponding defining set) to the list. Next, the algorithm prints the code parameters and their defining set. This is followed by a set of if statements that verify if the current exponent considered is in the cyclotomic coset of one of the APN/near-bent/bent exponents considered in this work. If it is, then the exponent type is printed and appended to the list L. If not, we move on to the next exponent. Then, the algorithm appends the number of nonzero-weights to the list "L." The output of the algorithm will then be the list L, which prints the considered

exponent, its type, and the number of nonzero weights of the corresponding dual code, and the list SLLL, which contains a list of weight enumerator polynomials with the defining sets of cyclic codes that lead to the polynomial. Finally, it outputs the parameters of the cyclic code, including the weight distribution of its dual, number of nonzero weights, minimum distance, and other properties that we modify the algorithm to display.

We utilized Algorithm 2 to construct cyclic codes with defining set of size two and compute their minimum distance and weight distribution. We analyzed the weight distribution of these codes theorems/conjectures in subsection 6.1. We further expanded this analysis. For minimum distance computations, we utilized the GAP package "Guava" via SAGE in the virtual online workspace for calculations, research, collaboration, and authoring documents (COCALC). Without this package, computations of the minimum distance for codes of lengths above 255 were not done in a reasonable time. However, even with the Guava package, we limited our computations up to $m = 11$. We construct Table 10 based on Algorithm 2. We note that there are 66 distance five codes from this construction. There are several functions related to these codes that are not directly identified with APN or AB exponents. In particular, the exponent of the form $2^{m-3} - 1$ (dubbed as the "pre-Inverse") case is interesting, as, for an odd integer m, it is a distinct exponent that leads to distance five codes. Equivalence analysis of the functions was performed to verify if some correspond through some criteria to known APN functions.

It is important to note that while we consider the cyclotomic coset representative criteria (as discussed at the beginning of this chapter), there are still other equivalence criteria to consider. CCZ-equivalence (named after Carlet, Charpin, and Zinoviev) is the most general equivalence criteria for functions that preserve the APN property [11]. In [20], it is shown that two APN power functions are CCZ-equivalent if and only if they are cyclotomic-equivalent. We state this definition from [5] in Definition 9. The first condition states that l and k are in the same cyclotomic coset. We have already eliminated this possibility by taking the cyclotomic cosets representatives of the APN and AB exponents and verifying that they are distinct. The second condition states that k and the inverse of l (or vice versa) share a cyclotomic coset. We construct Algorithm 3 to verify this equivalence and apply it to construct a table that verifies if the observed "new" functions that lead to two-error-correcting codes are APN or not. Since the functions are cyclotomic-equivalent, the corresponding cyclic codes are as well. The results are observed in Table 13. Since we choose exponents such that they are the cyclotomic cosets representatives of cosets that contain an APN/AB/bent exponent, then the first condition of cyclotomic equivalence is not met. We thus only focus on the second condition.

Algorithm 3. [67]

```
def cycloequiv(m):
  f = FindModNoncube(m)
  R.<x> = GF(2^m,'a', modulus = f)[]
  k.<a> = GF(2**m, modulus = f)
  CRL = RepCyclo(m), n = 2^m - 1, Cl = Cyclo(1,n)
  for i in range(len(CRL)):
    if w in range(len(CRL)):
```

```
        if (CRL[i]*CRL[w])%(n) in Cl and gcd(CRL[i],n) == 1:
          print(",f(x) = $x^{", CRL[i],"}$ is cyclotomic-equivalent to g(x) =
$x^{", CRL[w],"}$")
```

This algorithm verifies the second condition for cyclotomic equivalence between power functions over $\mathbb{F}_{2^m}$ as in Definition 9 and in [5]. The algorithm takes as an input the number of variables, then finds an irreducible polynomial of degree m (over $\mathbb{F}_{2^m}$) and then constructs a Boolean polynomial Ring over $\mathbb{F}_{2^m}$ such that it assigns a as the congruence class $[x]$ modulo f (the irreducible polynomial from the previous algorithm). This is done via the "FindMod-NonCube" pre-defined algorithm, which uses Conway polynomials to find the corresponding irreducible polynomial. Then, it constructs a finite field as powers of "a" where a is a primitive element in the field. It then assigns a list of all the cyclotomic coset representatives $(\text{mod } 2^m-1)$ to the variable CRL, assigns $n = 2^m-1$ and the cyclotomic coset of $1 \pmod n$ to the list CL. Then, it iterates over the list CRL and verifies for every pair of cyclotomic coset representatives if the second condition of the cyclotomic equivalence definition is met. The results from this algorithm were organized, and the exponents corresponding to the list of functions that lead to two-error-correcting codes (obtained from Algorithm 2) were compared and identified as equivalent to an APN function or not.

Note that the "pre-Inverse" case that we observed before is APN only when it is equivalent to a Gold function (five and seven variables). From [41] we know that if an exponent d leads to a two-error-correcting codes, then $f(x) = x^d$ has to be APN. Thus, we establish that the minimum-distance commands from SAGE have some errors. We propose a new minimum distance algorithm using MacWilliams identity and derivatives of the weight enumerator polynomial for codes with a low number of weights on their respective dual code.

Algorithm 4. [67] `def MinimumDistance2r(`m`):`

```
    n = 2^m − 1
    print("Code - Minimum Distance - Defining Set - Weight Coefficients - nonzero
Weights")
    for i in range(0,len(CrepNB[m])):
      cr = CrepNB[m][i],D = [1, cr]
      C = codes.CyclicCode(field=GF(2), length=n, D=[1, cr])
      h = C.check_polynomial()
      DC = codes.CyclicCode(generator_pol = h.reverse(), length = n)
      sd = DC.spectrum(algorithm = "binary")
      a = [y for y in sd if y != 0]
      g2 = 2^(-C.dimension())*(1+x)^(n),az = 0
      for w2 in range(0, n+1):
        az = az + sd[w2]*((1-x)/(1+x))^(w2)
      g2 = g2*az
      for i2 in range(1, degree(g2)):
        g2 = sym.diff(g2)
        if g2.subs(x:0) != 0:
          d = i2
          break
```

```
print(C, "-",d, "-", D, "-", len(a) - 1)
```

The algorithm takes as input the number of variables considered, assigns the length of the corresponding codes to n, and prints the table headings. Then, it begins iterating over the list of cyclotomic coset representatives of the near-bent and APN exponents and constructs the cyclic code C with defining set $D = \{1, cr\}$. The generator polynomial for the dual code $C^{\perp}$ is obtained by taking the check polynomial of C and reversing it. The spectrum (weight distribution) of this code is computed; we take the nonzero weights (which correspond to the indices in the distribution) and begin construction of the weight enumerator polynomial of C by applying the MacWilliams identity. Once this polynomial is obtained, we iterate over the derivatives of the polynomial and evaluate them at $x = 0$, with the first nonzero result meaning that the lowest (nonzero) degree term has been reduced to a constant, this degree corresponding to the minimum weight (which is the same as the minimum distance of a linear code [42]).

The results from this algorithm are seen in Table 11 and are verified and complemented by the APN conjecture and its relation to two-error-correcting codes. We observe that the weight distribution of the dual codes contains mostly three nonzero weights. Furthermore, the dimension of the dual codes is at most $2m$ while the dimension of the original code is at least $2^m - 1 - 2m$, which is a vastly greater number as m increases. We apply Theorem 4 to the weight enumerator polynomial of the dual code but do not expand the resulting polynomial. We then compute the successive derivatives of the resulting polynomial and evaluate at $x = 0$. The first nonzero result obtained will determine the minimum distance of the code. Thus, a smaller number of codewords to consider, fewer computations by not expanding these polynomials, and results that match the theory show that this algorithm improves the current implementation in SAGE.

4. Cyclotomic Coset Analysis of the Gold and Kasami-Welch Exponents

In this section, we analyze the Gold and Kasami-Welch exponent distribution in cyclotomic cosets $\pmod{2^m - 1}$. Some of the proofs in this section were results from our Springer PROMS article [67]. Some of the proofs have been improved (see Subsubsection 4.2.1), while others, like the Kasami-Welch case, have been completed. As mentioned before, for the trace functions in m variables of the form $Tr(x^d)$ will be equal for all d such that they are in the same cyclotomic coset $\pmod{2^m - 1}$. Similarly, for the construction of cyclic codes, as seen in Subsection 1.3, if you exchange entries of the defining set for another in the same cyclotomic coset, then the resulting codes are equivalent. Finding the distribution of Gold and Kasami-Welch exponents in these cosets would then allow further characterization of these functions and codes. Our computational results led us to state the theorems that follow in the subsequent subsections. We present these as a new proof for results on known CCZ-equivalence between Gold and Kasami-Welch functions, as shown previously by Budaghyan [4] and Yoshiara [71]. Consider the Gold and Kasami-Welch functions $g_l = x^{2^l+1}, k_l = x^{2^{2l}-2^l+1}$, $0 < l < m, (l, m) = 1$. Budaghyan has showed that g_{l_1} and g_{l_2} are CCZ-equivalent if and only if $l_1 = l_2$ or $l_1 + l_2 = m$ (and thus $\frac{\phi(m)}{2}$ equivalence classes). Yoshiara has a proposition for the CCZ-equivalence between Gold, Kasami-Welch, Niho, and Welch exponents. Our results are related to the first and fifth points of Proposition 2 in [71]:

Proposition 3 (Yoshiara [71])**.** For the Gold and Kasami-Welch functions $g_{s_i} = x^{2^{s_i}+1}, k_{r_i} = x^{2^{2r_i}-2^{r_i}+1}$, $0 < s_i < \frac{m}{2}, 1 < r_i < \frac{m}{2}, (s_i, m) = 1, (2, r_i) = 1$, then:

(1) The Gold functions g_{l_1} and g_{l_2} defined on $\mathbb{F}_{2^m}$ are CCZ-equivalent if and only if $l_1 = l_2$.

(2) The Kasami-Welch functions k_{l_1} and k_{l_2} defined on $\mathbb{F}_{2^m}$ are CCZ-equivalent if and only if $l_1 = l_2$.

4.1. **CCZ and Cyclotomic equivalence of Boolean Functions.** We focus on key cryptographic properties of these functions as we want good codes associated with the final selection of functions. One such property is the differential uniformity, which defines APN functions. The most general form of equivalence that preserves this property is CCZ-equivalence [11]. Extended Affine-equivalence (EA-equivalence) is a less general form of equivalence that also preserves properties like the algebraic degree [10, 64]. EA-equivalence coincides with CCZ-equivalence for Boolean functions and vectorial Boolean functions [6, 7]. As such, we focus on CCZ-equivalence in this investigation. These equivalences are defined as follows:

Definition 13 (Canteaut [11] CCZ-equivalence)**.** Two functions $F : \mathbb{F}_{2^m} \to \mathbb{F}_{2^k}$ and $G : \mathbb{F}_{2^m} \to \mathbb{F}_{2^k}$ are CCZ-equivalent if there exists an affine permutation A of $\mathbb{F}_{2^m} X \mathbb{F}_{2^m}$ such that:

$\{(x, F(x)), x \in \mathbb{F}_{2^m}\} = A(\{(x, G(x)), x \in \mathbb{F}_{2^m}\})$

Definition 14 (Canteaut [11] EA-equivalence)**.** Two functions $F : \mathbb{F}_{2^m} \to \mathbb{F}_{2^k}$ and $G : \mathbb{F}_{2^m} \to \mathbb{F}_{2^k}$ are EA-equivalent if there exist two affine permutations $A : \mathbb{F}_{2^m} \to \mathbb{F}_{2^m}$, $B : \mathbb{F}_{2^k} \to \mathbb{F}_{2^k}$ and an affine function $C : \mathbb{F}_{2^m} \to \mathbb{F}_{2^k}$ such that:

$F(x) = (BoGoA)(x) + C(x))$

We studied several references to obtain a proper list of important properties that are preserved by this type of equivalence. The importance of this is that if we have multiple EA or CCZ-equivalent functions in our list, we can reduce computational time for our algorithms by focusing on one of these equivalent functions. The properties preserved are: Differential Spectrum [11] (Both) , Walsh Spectrum [11] (Both), Walsh spectrum preserved means that the bent and near-bent properties are preserved [11, 15] (Both). Differential uniformity [3, 11] (Both) and differential uniformity $= 2 \implies$ the function is APN, as such, the APN property is also preserved [2, 5, 11] (Both). Extended Walsh spectrum [11] (Both) , Algebraic Degree [3, 5, 11] (EA) , Nonlinearity [3, 64] (Both), Plateaued property (with single amplitude and the value of the amplitude for Both) [64] and Minimum degree [64] (EA) .

As stated previously, for the Boolean case, EA and CCZ-equivalence coincide. The concept of cyclotomic equivalence was proven to be equivalent to CCZ-equivalence for Boolean functions by Dempwolf and Yoshiara [20, 71]. In 2016, Yoshiara proved a result on CCZ-equivalence of Gold and Kasami-Welch functions of the form $g_s(x) = x^{2^s+1}, k_r(x) = x^{2^{2r}-2^r+1}$ in m variables $(s, m) = 1, (r, 2) = 2$ with $0 < s < \frac{m}{2}, 1 < r < \frac{m}{2}$.

In Subsection 4.3, we prove a result on the multiplicative inverse of the Gold and Kasami-Welch exponents in cyclotomic cosets that contain them. These results, in conjunction with the results from Subsection 4.2, lead to a proof of conditions of cyclotomic equivalence (and hence CCZ-equivalence) between Gold (and respectively, Kasami-Welch) functions. In our case, we provide a new proof for Yoshiara's result by taking the restriction $0 < i, x < m, (m, l) = 1$ for $g_i g_x, k_i, k_x$ and $m > 3$ in the Gold case, $m > 5$ in the Kasami-Welch case.

CCZ-equivalence is the most general form of equivalence between Boolean functions that preserves the APN property [10]. Thus, the CCZ-equivalence (and cyclotomic equivalence) properties of these functions are important to classify these functions under distinct classes of APN functions.

Going back to Definition 9 the first condition of cyclotomic equivalence states that the two exponents share a cyclotomic coset. For Gold and Kasami-Welch exponents of the form $2^i+1, 2^j+1$ and $2^{2i}-2^i+1, 2^{2j}-2^j+1$ respectively, we have showed that they share a cyclotomic coset only when $i+j=m$ (given that $(i,m)=(j,m)=1$. The second condition, $kl \equiv 2^a \pmod{2^m-1} \rightarrow 2^{m-a}kl \equiv 1 \pmod{2^m-1} \rightarrow (2^{m-a}k)l \equiv 1 \pmod{2^m-1} \rightarrow (2^{m-a}l)k \equiv 1 \pmod{2^m-1}$, means that, for k, l corresponding to two Gold or Kasami-Welch exponents in different cyclotomic cosets, then they share a cyclotomic coset with the multiplicative inverse of the other exponent. We prove in Subsection 4.3 that this condition cannot be met, and as such, if $i+j \neq m$ then the respective functions cannot be cyclotomic (and thus, CCZ) equivalent.

4.2. **Distribution of the Gold and Kasami-Welch Exponents in a Cyclotomic Coset.** Numeric experimentation was done to identify patterns in the distribution of the Gold and Kasami-Welch exponents. We constructed Tables 8 and 17, for m an even integer. We observe that the exponents are distributed such that there are two per cyclotomic coset in the majority of cyclotomic cosets that contain them. The exceptions occur when the exponents are of the form $2^{\frac{m}{2}}+1, 2^{2\frac{m}{2}}-2^{\frac{m}{2}}+1$. For the Gold case, the size of the cyclotomic coset for which only one exponent is found is $\frac{m}{2}$, whereas, for the Kasami-Welch case, all cyclotomic cosets are of size m. For the odd case, in Tables 16 and 18, all cyclotomic cosets observed contain a pair of distinct exponents. Furthermore, all the cosets that contained the exponents were of size m. The following lemmas are used in the proof of our results which are derived from these observations.

Lemma 2. The sum $\sum_{i=0}^{n-1} a_i 2^i \leq 2^n-1$, with coefficients from $\{0,1\}$. With equality if and only if all the coefficients are 1.

An important consequence of this lemma that we utilize in the proofs is that if you have two sums of distinct powers of 2, if they differ in at least one term, then they sum to different numbers. We state and prove this as follows:

Lemma 3. For two sums of distinct powers of two, if they differ in at least one term, then they sum to different numbers.

Proof: By the unique 2-adic representation of integers, if the two sums are distinct, they represent a different integer.

□

Since we are considering sums $\pmod{2^m-1}$, the applications of these lemmas must consider this fact. For example, $2^x+2^4+2 \equiv 2^4+2+1 \pmod{2^5-1}$ is true for $x=5$. In fact, it is true for x being any multiple of 5. In these cases, if we can show that 2^x is not equivalent $\pmod{2^5-1}$ to all powers of 2 on the right side of the equivalence, then we can apply Lemma 3 to assert that the two sums are equivalent $\pmod{2^5-1}$ to distinct integers less than 2^5-1. For more unknown powers, say: $2^a+2^b+2^c \equiv 2^d+2^e+2^f \pmod{2^m-1}$, our

approach is to verify that the sums on the left and right side of the equivalence are between non-equivalent powers of 2. Then, we verify that the equivalence does not hold if their sum is equivalent $\pmod{2^m-1}$ to distinct integers less than 2^m-1. It is important to note that if both sides of the equivalence are sums of less than m distinct powers of 2, then the equivalence turns to equality. This is because, by Lemma 2, if we have the sum of less than m distinct powers of 2, then they cannot sum to an integer greater than 2^m-1. For the proofs we show, all powers of 2 shall be substituted by their equivalence class representative such that we can substitute equivalence for equality in our analysis. The equivalence class representatives of a power of 2 is necessarily another power of 2 (less than 2^m). This is because for $c > m, 0 < c_2 < m, c \equiv c_2 \pmod{m}$ and an integer $0 < b < 2^m-1$, $2^c \equiv b \pmod{2^m-1} \rightarrow 2^{-c}b \equiv 1 \pmod{2^m-1} \rightarrow 2^{-c_2}b \equiv 1 \pmod{2^m-1} \rightarrow b \equiv 2^{c_2} \pmod{2^m-1}$. In our proofs, we consider the "equivalence class representative" of an integer "a" as the smallest non-negative integer that is less than 2^m-1 which is in the same equivalence class.

Theorem 5 (Velazquez, Janwa [67] The Number of Gold Exponents in The Cyclotomic Cosets $\pmod{2^m-1}$)**.** Let 2^l+1 be a Gold exponent. Gold exponents occur in pairs of the form $2^i+1, 2^j+1$ in a cyclotomic coset of size a $\pmod{2^m-1}$ where $j+i=m$, and $m > 2$.

Proof: Let i,j,w,k,a,t and $m \in \mathbb{Z}$

Case 1: m is an odd prime.

We know from that the size of the cyclotomic cosets $\pmod{2^m-1}$ is a non-trivial divisor of m, and since m is prime, all the cyclotomic cosets (aside from the one composed of only the 0 element) are of size m. Say we have the cyclotomic coset containing 2^i+1: $C(2^i+1) = \{2^i+1, 2*(2^i+1), \ldots 2^{m-1}*(2^i+1)\}$. Consider j such that $j+i=m$,$0 < j, i < m$ (if j = i then the pair of 2^i+1 is itself) then we have $2^j(2^i+1) \in C(2^i+1)$ and

$$2^j(2^i+1) \equiv 2^{i+j}+2^j \pmod{2^m-1} \equiv 2^m+2^j \pmod{2^m-1} \equiv 1+2^j \pmod{2^m-1}. \quad (8)$$

Thus, we found a "pair" of Gold exponents in the same cyclotomic coset. Now we show that no other Gold exponent can be found in the same coset.

Consider $k \neq j, i$, $0 < k, w < m$, $w \neq j$ such that: $2^w(2^i+1) \equiv 2^k+1 \pmod{2^m-1}$. Note that $w \neq k$ or else we have $(2^{i+k}+2^k) \equiv 2^k+1 \pmod{2^m-1} \rightarrow 2^{i+k} \equiv 1 \pmod{2^m-1} \rightarrow k = j$ (since both i and k are less than m). Furthermore, $w+i \neq m$ (forces w to equal j) nor $w+i = k$ as this would mean: $2^w(2^i+1) \equiv 2^{w+i}+2^w \pmod{2^m-1} \equiv 2^k+2^w \pmod{2^m-1} \equiv 2^k+1 \pmod{2^m+1} \rightarrow 2^w \equiv 1 \pmod{2^m-1} \rightarrow w = m$ or $w = 0$ which is false.

With the conditions given above, we have:

$$2^w(2^i+1) \equiv 2^{w+i}+2^w \pmod{2^m-1} \equiv 2^k+1 \pmod{2^m-1}. \quad (9)$$

Note that if w + i $> m$, say w+i = e + m, then $2^{w+i} \equiv 2^e \pmod{2^m-1}$ with 2^e being a power of 2 less than 2^m. And if $w+i = e < m$ we have the same situation. Thus, on both sides we have the sum of two distinct powers of 2 that are less than 2^m and, since m is a prime odd integer with a minimum value of three, both sums are less than 2^m-1 (from lemma 2). The left and right expressions are representatives of distinct congruence classes $\pmod{2^m-1}$ and, as such, must be distinct. Thus, no such k and w exist, and we get our result.

Case 2: m is an even integer.

As discussed previously, the sizes of the cyclotomic cosets divide m, so in this case, we could have Gold exponents in a coset of length a, where $a|m$. For the case where the size of the coset is m, we can apply the same argument as the previous case to find precisely two Gold exponents in the coset whenever $m > 2$.

Consider the size of the coset to be $0 < a < m$ that is:

$$C(2^i + 1) = \{2^i + 1, 2 * (2^i + 1) + \ldots 2^{a-1}(2^i + 1)\} \tag{10}$$

such that $a|m$. The biggest value "a" can take is $\frac{m}{2}$. Without loss of generality, we can assume that $0 < i < j < m$ with $i < \frac{m}{2}$ and $\frac{m}{2} \leq j$. First, if $\frac{m}{2} < j$, then $a < j$, say $j = a + e < m$ for some $0 < e < m$. Then we have that $2^j(2^i + 1) \equiv 2^{a+e}(2^i + 1) \pmod{2^m - 1} \equiv 2^e(2^i + 1) \pmod{2^m - 1}$. Thus, we have :

$$2^{e+i} + 2^e \equiv 2^j + 1 \pmod{2^m - 1} \tag{11}$$

Since $i < \frac{m}{2}$ and $e < j < m$, then $2^{e+i} < 2^m \pmod{2^m - 1}$. Furthermore $e + i \neq e$ (as otherwise $i = 0$, which is false). This means that $2^{e+i} + 2^e$ sum to an even integer less than $2^m - 1$. On the right side of Equivalence 11, we have that $j < m \rightarrow 2^j + 1 < 2^m - 1 \rightarrow 2^j + 1$ sum to an odd integer less than $2^m - 1$. Thus, Equivalence 11 turns into an equality between an odd and an even integer, which is a contradiction.

The next case is when $j = a = \frac{m}{2}$. Since $i + j = m$, then $i = \frac{m}{2}$. Thus, such a pair can exist in a cyclotomic coset of size $a = \frac{m}{2}$. To show that his pair is unique, suppose there exists a third Gold exponent, say $2^w + 1$ $(0 < w < m)$ such that for some integer $0 < k < a$ we have $2^k(2^i + 1) \equiv 2^w + 1 \pmod{2^m - 1}$. Then:

$$2^{k+i} + 2^k \equiv 2^w + 1 \pmod{2^m - 1}. \tag{12}$$

Since $k < \frac{m}{2}$, then $2^{k+i} < 2^m$ and $k \neq k + i \rightarrow 2^{k+i} + 2^k$ sums to an even integer less than $2^m - 1$. On the other hand, $2^w + 1$ sums to an odd integer less than $2^m - 1$. Thus we reach a contradiction as we reach an equality between an odd and an even integer.

Finally, if $j = \frac{m}{2}$ and $a < j$, say $j = a + e$ for some integer $e < m$, we have:

$$2^{e+i} + 2^e \equiv 2^j + 1 \pmod{2^m - 1} \tag{13}$$

We reach a contradiction by the same argument as the case $\frac{m}{2} < j$.

Case 3: m a non-prime odd integer.

Once more, we appeal to the argument about the size of the cyclotomic cosets. In this case the size will either be m or some odd integer a that divides m. In case 1, we have shown that if the coset is of size m, then there are precisely two Gold exponents if one is found in the coset; in case 2, we have shown that if the size is $a < m$, then if we find a Gold exponent in the cyclotomic coset, the size must be $\frac{m}{2}$, but since m is odd, no such case can happen, thus if we find a Gold exponent in a coset, there are precisely two of them. □

Corollary 1 (Velazquez, Janwa [67] Number of Gold Near-Bent Functions Over $\mathbb{F}_{2^m}$)**.** Let the Gold Boolean near-bent function in m variables $(m > 2)$ be of the form $Tr(x^{2^l+1})$ such

that $1 \leq l \leq m-1$ and $(l,m)=1$. Then, the number of non-equal Gold near-bent functions is precisely $\frac{\Phi(m)}{2}$.

Proof: The result follows from Theorem 5. The (l,m) = 1 condition follows from [15]. For m an odd integer we have precisely two Gold exponents in each cyclotomic coset that contains them. The possible values of l are $1,2,3\ldots m-1$ and only those such that $(l,m)=1$ lead to near-bent functions. The number of such values of l is given by Euler's Φ function, and we divide this total by 2. Thus, $\frac{\Phi(m)}{2}$ will be the total number of Gold near-bent functions generated.

□

Theorem 6 (Cyclotomic Cosets $\pmod{2^m-1}$ that contain Kasami-Welch Exponents for m odd, $(l,m)=1$)**.** Let $2^{2l}-2^l+1$ be a Kasami-Welch exponent and $m>3$ an odd integer. Then, the Kasami-Welch exponents occur in a unique pair of the form $2^{2i}-2^i+1, 2^{2j}-2^j+1$, $j+i=m$, $(i,m)=(j,m)=1$ and the cyclotomic coset must be of size m.

Proof: Consider: $C(2^{2i}-2^i+1)=\{2^{2i}-2^i+1, 2(2^{2i}-2^i+1)+\cdots+2^{m-1}(2^{2i}-2^i+1)\}$ as the cyclotomic coset of size m containing a Kasami-Welch exponent. Define $i,j\in\mathbb{Z}^+$ such that $0<i,j<m, i+j=m$. Now, consider:

$$2^{2j}(2^{2i}-2^i+1)=2^{2(i+j)}-2^{j+(j+i)}+2^{2j}=2^{2m}-2^{j+m}+2^{2j}\equiv 1-2^j+2^{2j} \pmod{2^m-1}, \quad (14)$$

Thus, we have found a "pair" of Kasami-Welch exponents that belong to the same cyclotomic coset.

We show that this pair is unique in the cyclotomic coset by contradiction. Suppose there is a third Kasami-Welch exponent in the cyclotomic coset, say $2^{2x}-2^x+1$, with $x\neq i$, $x\neq j$. By the argument in the previous paragraph, we can find some positive integer y such that $y+x=m\rightarrow 2^{2y}(2^{2x}-2^x+1)\equiv 2^{2y}-2^y+1 \pmod{2^m-1}$ and $2^{2x}(2^{2y}-2^y+1)\equiv 2^{2x}-2^x+1 \pmod{2^m-1}$ with $y\neq j$, $y\neq i$. Rewrite these pairs as follows, $i=\frac{m+1}{2}-e_1, j=\frac{m+1}{2}+e_1-1, x=\frac{m+1}{2}-e_2, y=\frac{m+1}{2}+e_2-1$, with $0<e_1,e_2<\frac{m+1}{2}$. Without loss of generality, we shall assume that $x<i$ (i.e, $e_1<e_2$). We note that $2y=m+2e_2-1\equiv 2e_2-1 \pmod{m}$, $2j=m+2e_1-1\equiv 2e_1-1 \pmod{m}$, $2x=m+1-2e_2\equiv 1-2e_2 \pmod{m}$ and $2i=m+1-2e_1\equiv 1-2e_1 \pmod{m}$. For these four distinct exponents to exist in the same coset, the size of the coset must be greater than three, as such $m>3$. Since $2^{2x}-2^x+1$ is in the same cyclotomic coset as $2^{2i}-2^i+1$, then there exists some integer q such that $2^q(2^{2x}-2^x+1)\equiv 2^{2i}-2^i+1 \pmod{2^m-1}$, with $0<q<m$. We will show that such q cannot exists, and thus $2^{2x}-2^x+1$ cannot be in the cyclotomic coset. We can separate all the possible values of q into three ranges. These are, $0<q<2e_2-1$, $q=2e_2-1$ and $2e_2-1<q<m$.

First Case: $0<q<2e_2-1$

$$2^q(2^{2x}-2^x+1)=2^{2x+q}-2^{x+q}+2^q=2^{m+1-2e_2+q}-2^{\frac{m+1}{2}-e_2+q}+2^q\rightarrow$$
$$2^{m+1-2e_2+q}-2^{\frac{m+1}{2}-e_2+q}+2^q\equiv 2^{2i}-2^i+1 \pmod{2^m-1}.$$

Since $q < 2e_2 - 1$, then $q + 1 < 2e_2 \rightarrow m + 1 - 2e_2 + q < m$. Furthermore, since $e_2 < \frac{m+1}{2}$ then $0 < q < \frac{m+1}{2} - e_2 + q < m + 1 - 2e_2 + q < m \rightarrow 2^{m+1-2e_2+q} > 2^{\frac{m+1}{2}-e_2+q} \rightarrow 2^{m+1-2e_2+q} - 2^{\frac{m+1}{2}-e_2+q} + 2^q$ sums to an even integer less than $2^m - 1$. On the other hand, $2i = m+1-2e_1$, since $1 \leq e_1 < \frac{m+1}{2}$, then $2 \leq 2e_1 < m+1 \rightarrow 0 < i < 2i < m \rightarrow 2^{2i} - 2^i + 1$ is an odd integer less than $2^m - 1$. Since both sides sum to integers less than $2^m - 1$, we can replace the equivalence by equality, and we have an equality between an even and an odd integer, which is a contradiction.

Second Case: $q = 2e_2 - 1$

$2^q(2^{2x}-2^x+1) = 2^{2x+q}-2^{x+q}+2^q = 2^{m+1-2e_2+q}-2^{\frac{m+1}{2}-e_2+q}+2^q = 2^m-2^{\frac{m+1}{2}+e_2-1}+2^{2e_2-1} \equiv 1 - 2^y + 2^{2y} \pmod{2^m - 1}$

This implies that:

$$2^{2y} - 2^y \equiv 2^{2i} - 2^i \pmod{2^m - 1} \tag{15}$$

Since $e_2 < \frac{m+1}{2}$ then $2e_2 - 1 < \frac{m+1}{2} + e_2 - 1 < m$. This means that $(2^y - 2^{2y})$ is equivalent $\pmod{2^m - 1}$ to an even integer less than $2^m - 1$, and thus we have $2^{2y} - 2^y = -(2^y - 2^{2y}) \equiv 2^m - 1 - (2^y - 2^{2y}) \pmod{2^m - 1}$ which is an odd integer less than $2^m - 1$. The right side of Equivalence 15 sums to an even integer less than $2^m - 1$ (as discussed in the previous case) and thus we have a contradiction as we reach an equality between an even and an odd integer.

Third Case: $2e_2 - 1 < q < m$

Let us call $q = 2e_2 - 1 + k$, where $2 \leq 2e_2 \leq m - 1, 0 < q < m \rightarrow 0 < k < m - 1$.

$2^q(2^{2x} - 2^x + 1) = 2^{2x+q} - 2^{x+q} + 2^q = 2^{m+1-2e_2+q} - 2^{\frac{m+1}{2}-e_2+q} + 2^q = 2^{m+k} - 2^{\frac{m+1}{2}+e_2-1+k} + 2^{2e_2-1+k} \equiv 2^k - 2^{y+k} + 2^{2y+k} \pmod{2^m - 1} \rightarrow 2^k - 2^{y+k} + 2^{2y+k} \equiv 2^{2i} - 2^i + 1 \pmod{2^m - 1}$

This implies that:

$$2^{2y+k} + 2^i + 2^k \equiv 2^{2i} + 2^{y+k} + 1 \pmod{2^m - 1} \tag{16}$$

Our approach is to show that the left and right sides of Equivalence 16 are distinct sums of distinct powers of 2, and thus by Lemmas 2 and 3 the equivalence cannot hold.

Left Subcase 1: $2y + k \equiv i \pmod{m}$

$\rightarrow q \equiv i \pmod{m}$ and $\rightarrow y + k \equiv i + x \pmod{m} \rightarrow$

$$2^{i+1} + 2^k \equiv 2^{2i} + 2^{i+x} + 1 \pmod{2^m - 1}. \tag{17}$$

Left Side of Equivalence 17 Sub-subcase 1: $i + 1 \equiv k \pmod{m}$

$\rightarrow q + 1 \equiv k \pmod{m} \rightarrow 2e_2 - 1 + k + 1 \equiv k \pmod{m} \rightarrow 2e_2 \equiv 0 \pmod{m}$ which is a contradiction as $(2, m) = 1$.

Right Side of Equivalence 17 Sub-subcase 1: $2i \equiv i + x \pmod{m}$

$\rightarrow i \equiv x \pmod{m} \rightarrow x = i$ which is a contradiction.

Right Side of Equivalence 17 Sub-subcase 2: $2i \equiv 0 \pmod{m}$

This is a contradiction as $(2, m) = 1$.

Right Side of Equivalence 17 Sub-subcase 3: $i + x \equiv 0 \pmod{m}$

$\rightarrow i \equiv -x \pmod{m} \rightarrow i \equiv y \pmod{m} \rightarrow i = y$ which is a contradiction.

Thus, for Equivalence 17, the left side of the equivalence are two distinct powers of 2, all of which are not equivalent to 1 $\pmod{2^m - 1}$ (as both $(0 < i, k < m - 1)$, thus summing to an even integer less than 2^m. For the right side of the equivalence, we have three distinct powers of 2, with only one of them equivalent to $2^m \pmod{2^m - 1}$, thus they sum to an odd integer. Substituting each term for their equivalence class representative, if necessary, we can substitute the equivalence for equality and reach that an even integer is equal to an odd integer, which is a contradiction.

Left Subcase 2: $2y + k \equiv k \pmod{m}$

$\rightarrow 2y \equiv 0 \pmod{m}$ which is a contradiction as $(2, m) = 1$.

Left Subcase 3: $i \equiv k \pmod{m}$

$$\rightarrow 2^{2y+i} + 2^{i+1} \equiv 2^{2i} + 2^{y+i} + 1 \tag{18}$$

Left Side of Equivalence 18 Sub-subcase 1: $2y + i \equiv i + 1 \pmod{m}$

$\rightarrow 2y \equiv 1 \pmod{m} \rightarrow 2e_2 - 1 \equiv 1 \pmod{m} \rightarrow 2e_2 \equiv 2 \pmod{m} \rightarrow e_2 \equiv 1 \pmod{m}$. Since $e_2 < \frac{m+1}{2}$ then $e_2 = 1$. This is a contradiction as we have assumed that $e_1 < e_2$, and $0 < e_1 < e_2 < \frac{m+1}{2}$ then e_1 is at minimum one, and thus e_2 is at minimum 2.

Right Side of Equivalence 18 Sub-subcase 1: $2i \equiv y + i \pmod{m}$

$\rightarrow y \equiv i \pmod{m} \rightarrow y = i$. This is a contradiction.

Right Side of Equivalence 18 Sub-subcase 2: $2i \equiv 0 \pmod{m}$

This is a contradiction as $(2, m) = 1$.

Right Side of Equivalence 18 Sub-subcase 3: $y + i \equiv 0 \pmod{m}$

$\rightarrow y \equiv -i \pmod{m} \rightarrow y \equiv j \pmod{m} \rightarrow y = j$ which is a contradiction.

Thus, the left side of Equivalence 18 is the sum of two distinct powers of 2, both of which are not equivalent to 2^m and thus, once substituted for the equivalence class representatives, they sum to an even integer less than $2^m - 1$. For the right side of the equivalence, there are three non-equivalent powers of 2, with only one of them being equivalent to $2^m \pmod{2^m - 1}$, and thus when replaced by their equivalent class representative, they sum to an odd integer less than $2^m - 1$. Thus, Equivalence 18 turns into an equality between an even and an odd integer which is a contradiction.

Now we move to the right side of Equivalence 16.

Right Subcase 1: $2i \equiv y + k \pmod{m}$.

$$\rightarrow 2^{2y+k} + 2^i + 2^k \equiv 2^{2i+1} + 1 \tag{19}$$

Left Side of Equivalence 19 Sub-subcase 1: $2y + k \equiv i \pmod{m}$

This is a contradiction by the same argument as in left subcase 1.

Left Side of Equivalence 19 Sub-subcase 2: $2y + k \equiv k \pmod{m}$

This is a contradiction by the same argument as left subcase 2.

Left Side of Equivalence 19 Sub-subcase 3: $i \equiv k \pmod{m}$

$\rightarrow 2i \equiv y + i \pmod{m} \rightarrow i \equiv y \pmod{m} \rightarrow i = y$ which is a contradiction.

Right Side of Equivalence 19 Sub-subcase 1: $2i + 1 \equiv 0 \pmod{m}$

$\rightarrow 2^{2i+1} + 1 \equiv 2 \pmod{2^m - 1}$

Thus, on the left side of Equivalence 19 we have the sum of three non-equivalent powers of 2, all of which are non-equivalent to 2^m (as $0 < q, k, i < m$) and thus sum to an even integer less than 2^m but greater than 2. On the right side we have that either the sum is equivalent to $2 \pmod{2^m - 1}$ or an odd integer less than $2^m - 1$, both cases leading to a contradiction.

Right Subcase 2: $2i \equiv 0 \pmod{m}$.

This is a contradiction as $(2, m) = 1$.

Right Subcase 3: $y + k \equiv 0 \pmod{m}$.

$\rightarrow k \equiv -y \pmod{m} \rightarrow k \equiv x \pmod{m} \rightarrow$

$$2^{2y+x} + 2^i + 2^x \equiv 2^{2i} + 2 \tag{20}$$

From our previous analysis, we know that the left side of the equivalence results in the sum of three non-equivalent powers of 2 which when replaced by their equivalence class representative sum to an even integer less than 2^m. For the right side of the equivalence, we have the following:

Right Side of Equivalence 20 Sub-subcase 1: $2i \equiv 1 \pmod{m}$.

Since $0 < i < \frac{m+1}{2}$, the only possibility for $2i \equiv 1 \pmod{m}$ is that $2i = m + 1$. However, $i < \frac{m+1}{2} \rightarrow 2i < m + 1$. Thus, we reach a contradiction.

With these subcases, we have shown that the left side of Equivalence 16 is the sum of three non-equivalent powers of 2, none of which are equivalent to $2^m \pmod{2^m - 1}$. The right side of the equivalence is also the sum of three non-equivalent powers of 2, with one of them being 1. Thus, by substituting each term by their equivalence class representative, we have equality between an even and an odd integer which is a contradiction. Thus, no such value of q can exist such that $2^{2x} - 2^x + 1$ and $2^{2i} - 2^i + 1$ share a cyclotomic coset, and thus exactly two Kasami-Welch exponents can share the same cyclotomic coset.

For the size of the cyclotomic cosets that contain these exponents, assume that there exists one such cyclotomic coset whose size is a where $a|m$. The largest possible divisor of m (aside from itself) is $\frac{m}{3}$ as m is an odd integer. As such, we shall consider $0 < a \leq \frac{m}{3}$. Let us say that the pair of exponents $2^{2i} - 2^i + 1, 2^{2j} - 2^j + 1$ are in this cyclotomic coset, with i, j defined as before. Since the size of the cyclotomic coset is a, we have that $2^a(2^{2i} - 2^i + 1) \equiv 2^{2i} - 2^i + 1 \pmod{2^m - 1}$ and the following equivalence must be true:

$$2^{2i+a} - 2^{i+a} + 2^a \equiv 2^{2i} - 2^i + 1 \pmod{2^m - 1} \tag{21}$$

First Case: $0 < a < 2e_1 - 1$ and $a < \frac{m}{3}$

$2i = m + 1 - 2e_1 \rightarrow 2i + 2e_2 - 1 = m + 1 - 2e_1 + 2e_1 - 1 = m$. Since $a < 2e_1 - 1$, then $2i + a < m$ and $0 < i + a < m$. Thus, $0 < a < i + a < 2i + a < m$. On the other hand, $2i = m + 1 - 2e_1$, since $1 \leq e_1 < \frac{m+1}{2}$, then $2 \leq 2e_1 < m + 1 \rightarrow 0 < i < 2i < m \rightarrow 2^{2i} - 2^i + 1$ is an odd integer less than $2^m - 1$.

Thus, the left side of Equivalence 21 is the sum of three non-equivalent powers of 2 less than 2^m which result in an even integer less than $2^m - 1$. The right side of the equivalence is composed of three non-equivalent powers of 2 less than 2^m, one of which is 1. Thus, they sum to an odd integer less than $2^m - 1$. The equivalence can be replaced for equality, and

we would have equality between an odd and an even integer, which is a contradiction.

Second Case: $a = 2e_1 - 1 < \frac{m}{3}$

$\rightarrow 2^{2i+a} - 2^{i+a} + 2^a \equiv 2^{m+1-2e_1+2e_1-1} - 2^{\frac{m+1}{2}-e_1+2e_1-1} + 2^{2e_1-1} \pmod{2^m-1} \rightarrow 2^m - 2^{\frac{m+1}{2}+e_1-1} + 2^{2e_1-1} \equiv 2^{2i} - 2^i + 1 \pmod{2^m-1} \rightarrow 1 - 2^j + 2^{2j} \equiv 2^{2i} - 2^i + 1 \pmod{2^m-1}$ which implies that:

$$2^{2j} - 2^j \equiv 2^{2i} - 2^i \pmod{2^m-1} \tag{22}$$

Since $e_1 < \frac{m+1}{2}$ then $2e_1 - 1 < \frac{m+1}{2} + e_1 - 1 < m$. This means that $(2^j - 2^{2j})$ is equivalent $\pmod{2^m-1}$ to an even integer less than $2^m - 1$, and thus we have $2^{2j} - 2^j = -(2^j - 2^{2j}) \equiv 2^m - 1 - (2^j - 2^{2j}) \pmod{2^m-1}$ which is an odd integer less than $2^m - 1$. The right side of Equivalence 22 sums to an even integer less than $2^m - 1$ (as discussed in the previous case) and thus we have a contradiction as we reach an equality between an even and an odd integer.

Third Case: $2e_1 - 1 < a \leq \frac{m}{3}$.

Let us define $a = 2e_1 - 1 + k \leq \frac{m}{3}$ which implies that $0 < k < \frac{m}{3}$. We have $2^{2i+a} - 2^{i+a} + 2^a \equiv 2^{m+1-2e_1+2e_1-1+k} - 2^{\frac{m+1}{2}-e_1+2e_1-1+k} + 2^{2e_1-1+k} \pmod{2^m-1} \rightarrow 2^{m+k} - 2^{\frac{m+1}{2}+e_1-1+k} + 2^{2e_1-1+k} \equiv 2^{2i} - 2^i + 1 \pmod{2^m-1} \rightarrow 2^{2j+k} - 2^{j+k} + 2^k \equiv 2^{2i} - 2^i + 1 \pmod{2^m-1} \rightarrow$

$$2^{2j+k} + 2^i + 2^k \equiv 2^{2i} + 2^{j+k} + 1 \pmod{2^m-1} \tag{23}$$

We shall show that the left side of the equivalence is equivalent to an even integer less than $2^m - 1$ while the right side is equivalent to an odd integer less than $2^m - 1$. Thus, reaching a contradiction as we would have equality between an odd and an even integer.

Left Subcase 1: $2j + k \equiv i \pmod{m}$

$\rightarrow q \equiv i \pmod{m}$. Furthermore, $2j + k \equiv i \pmod{m} \rightarrow j + k \equiv 2i \pmod{m}$

$$2^{i+1} + 2^k \equiv 2^{2i+1} + 1 \pmod{2^m-1} \tag{24}$$

For the left side of Equivalence 24, we know that $k < \frac{m}{3}$ and $i + 1 = \frac{m+1}{2} - e_1 + 1 < m$ for $m > 1$. Thus, $2^{i+1} + 2^k$ sums to an even integer less than $2^m - 1$ and greater than 2. For the right side of the equivalence we have two cases, either they sum to an odd integer less than $2^m - 1$, or $2i + 1 \equiv m \pmod{m} \rightarrow 2^{2i+1} + 1 \equiv 2 \pmod{2^m-1}$. Both cases lead to a contradiction, as the left side is neither one, nor equivalent to 2.

Left Subcase 2: $2j + k \equiv k \pmod{m}$

$\rightarrow 2j \equiv 0 \pmod{m}$. This is a contradiction as $(2, m) = 1$.

Left Subcase 3: $i \equiv k \pmod{m}$

$\rightarrow i = k \rightarrow a = 2e_1 - 1 + i = 2e_1 - 1 + \frac{m+1}{2} - e_1 = \frac{m+1}{2} + e_1 - 1 > \frac{m}{3}$ but $a \leq \frac{m}{3}$.

Right Subcase1: $2i \equiv j + k \pmod{m}$

This is the same argument as Left Subcase 1.

Right Subcase 2: $2i \equiv 0 \pmod{m}$

This is a contradiction as $(2, m) = 1$.

Right Subcase 3: $j + k \equiv 0 \pmod{m}$

$\rightarrow k \equiv -j \pmod{m} \rightarrow k \equiv i \pmod{m}$. This is a contradiction by the same argument as Left Subcase 3.

Thus, both the left and right sides of equivalence 23 are distinct sums of non-equivalent powers of 2. When you replace each term with its equivalence class representative, we have equality between an even integer on the left side, and an odd integer on the right side. This is a contradiction, and thus the only possible way that the equivalence can hold is if $a = m$. $\square$

Theorem 7 (Cyclotomic Cosets $\pmod{2^m - 1}$ that contain Kasami-Welch Exponents for m even, $(l, m) = 1$)**.** Let $2^{2l} - 2^l + 1$ be a Kasami-Welch exponent and $m > 2, (l, m) = 1$ an even integer. Then, the Kasami-Welch exponents occur only in a unique pair of the form $2^{2i} - 2^i + 1, 2^{2j} - 2^j + 1$, $j + i = m$, $(i, m) = (j, m) = 1$ and the cyclotomic coset must be of size m.

Proof: Consider: $C(2^{2i} - 2^i + 1) = \{2^{2i} - 2^i + 1, 2(2^{2i} - 2^i + 1) + \cdots + 2^{m-1}(2^{2i} - 2^i + 1)\}$ as the cyclotomic coset of size m containing a Kasami-Welch exponent. Define $i, j \in \mathbb{Z}^+$ such that $0 < i, j < m, i + j = m$. Now, consider:

$$2^{2j}(2^{2i} - 2^i + 1) = 2^{2(i+j)} - 2^{j+(j+i)} + 2^{2j} = 2^{2m} - 2^{j+m} + 2^{2j} \equiv 1 - 2^j + 2^{2j} \pmod{2^m - 1}, \quad (25)$$

Thus, we have found a "pair" of Kasami-Welch exponents that belong to the same cyclotomic coset.

We show that this pair is unique in the cyclotomic coset by contradiction. Suppose there is a third Kasami-Welch exponent in the cyclotomic coset, say $2^{2x} - 2^x + 1$, with $x \neq i$, $x \neq j$. By the argument in the previous paragraph, we can find some positive integer y such that $y + x = m \rightarrow 2^{2y}(2^{2x} - 2^x + 1) \equiv 2^{2y} - 2^y + 1 \pmod{2^m - 1}$ and $2^{2x}(2^{2y} - 2^y + 1) \equiv 2^{2x} - 2^x + 1 \pmod{2^m - 1}$ with $y \neq j$, $y \neq i$. Rewrite these pairs as follows, $i = \frac{m}{2} - e_1, j = \frac{m}{2} + e_1, x = \frac{m}{2} - e_2, y = \frac{m}{2} + e_2$, with $0 < e_1, e_2 < \frac{m}{2}$. Without loss of generality, we shall assume that $x < i$ (i.e, $e_1 < e_2$). We note that $2y = m + 2e_2 \equiv 2e_2 \pmod{m}$,

$2j = m + 2e_1 \equiv 2e_1 \pmod{m}$, $2x = m - 2e_2 \equiv -2e_2 \pmod{m}$ and $2i = m - 2e_1 \equiv -2e_1 \pmod{m}$. Since $2^{2x} - 2^x + 1$ is in the same cyclotomic coset as $2^{2i} - 2^i + 1$, then there exists some integer q such that $2^q(2^{2x} - 2^x + 1) \equiv 2^{2i} - 2^i + 1 \pmod{2^m - 1}$, with $0 < q < m$. We will show that such q cannot exists, and thus $2^{2x} - 2^x + 1$ cannot be in the cyclotomic coset. We can separate all the possible values of q into three ranges. These are, $0 < q < 2e_2$, $q = 2e_2$ and $2e_2 < q < m$.

First Case: $0 < q < 2e_2$

$$2^q(2^{2x} - 2^x + 1) = 2^{2x+q} - 2^{x+q} + 2^q = 2^{m-2e_2+q} - 2^{\frac{m}{2}-e_2+q} + 2^q \rightarrow$$
$$2^{m-2e_2+q} - 2^{\frac{m}{2}-e_2+q} + 2^q \equiv 2^{2i} - 2^i + 1 \pmod{2^m - 1}.$$

Since $q < 2e_2$, then $2x + q = m - 2e_2 + q < m$. Furthermore, since $q < x + q < 2x + q < m$ then $q < \frac{m}{2} - e_2 + q < m - 2e_2 + q < m$, and $1 < 2^q < 2^m$. This means that $2^{m-2e_2+q} - 2^{\frac{m}{2}-e_2+q} + 2^q$ is an even integer less than $2^m - 1$. On the other hand, $2^{2i} < 2^m$ as $2i = m - 2e_1 < m$ and obviously $0 < 2^i < 2^{2i}$. Thus, $2^{2i} - 2^i + 1$ is an odd integer less than $2^m - 1$. We can thus substitute the equivalence for the equality:

$$2^{m-2e_2+q} - 2^{\frac{m}{2}-e_2+q} + 2^q = 2^{2i} - 2^i + 1.$$

Which cannot be true as the left side sums to an even integer while the right side sums to an odd integer.

Second Case: $q = 2e_2$

$2^q(2^{2x} - 2^x + 1) = 2^{2x+q} - 2^{x+q} + 2^q = 2^{m-2e_2+q} - 2^{\frac{m}{2}-e_2+q} + 2^q = 2^m - 2^{\frac{m}{2}+e_2} + 2^{2e_2} \equiv 1 - 2^{\frac{m}{2}+e_2} + 2^{2e_2} \pmod{2^m - 1} \rightarrow 2^{2e_2} - 2^{\frac{m}{2}+e_2} + 1 \equiv 2^{2i} - 2^i + 1 \pmod{2^m - 1}$

This implies that:

$$2^{2e_2} - 2^{\frac{m}{2}+e_2} \equiv 2^{2i} - 2^i \pmod{2^m - 1} \tag{26}$$

From the argument in the previous case, we know that $2^{2i} - 2^i$ is an even integer less than $2^m - 1$. On the other hand, since $e_2 < \frac{m}{2}$, then $2e_2 < \frac{m}{2} + e_2 < m$. Thus, $2^{2e_2} < 2^{\frac{m}{2}+e_2} < 2^m \rightarrow 2^{\frac{m}{2}+e_2} - 2^{2e_2}$ is a positive even integer less than $2^m - 1$. We then have that $2^{2e_2} - 2^{\frac{m}{2}+e_2} = -(2^{\frac{m}{2}+e_2} - 2^{2e_2})$. Applying equivalence modulo $2^m - 1$ gives us that $-(2^{\frac{m}{2}+e_2} - 2^{2e_2}) \equiv (2^m - 1) - (2^{\frac{m}{2}+e_2} - 2^{2e_2}) \pmod{2^m - 1}$. Note that $(2^m - 1) - (2^{\frac{m}{2}+e_2} - 2^{2e_2})$ is a positive odd integer less than $2^m - 1$ as it is an odd integer minus an even integer. Thus, Equivalence 26 can be written as the equality:

$$(2^m - 1) - (2^{\frac{m}{2}+e_2} - 2^{2e_2}) = 2^{2i} - 2^i$$

This implies an odd integer is equal to an even integer, which is a contradiction.

Third Case: $2e_2 < q < m$

Say $q = 2e_2 + k$, with $0 < k < m$ then, $2^q(2^{2x} - 2^x + 1) = 2^{2x+q} - 2^{x+q} + 2^q = 2^{m-2e_2+q} - 2^{\frac{m}{2}-e_2+q} + 2^q = 2^{m+k} - 2^{\frac{m}{2}+e_2+k} + 2^{2e_2+k} \equiv 2^k - 2^{\frac{m}{2}+e_2+k} + 2^{2e_2+k} \pmod{2^m - 1} \rightarrow 2^k - 2^{\frac{m}{2}+e_2+k} + 2^{2e_2+k} \equiv 2^{2i} - 2^i + 1 \pmod{2^m - 1}$.

We can rewrite this equivalence to obtain the following equivalence:

$$2^{2e_2+k} + 2^i + 2^k \equiv 2^{2i} + 2^{\frac{m}{2}+e_2+k} + 1 \pmod{2^m - 1} \tag{27}$$

Our approach shall be to show that this is equivalent to distinct sums of three distinct powers of 2 less than $2^m - 1$, and thus by Lemmas 2 and 3, the equivalence should not hold for such value of q. The first observation is that since $m \geq 4$, the sum of three distinct powers of 2 less than 2^m is less than $2^m - 1$. Now, we show that the terms on the left side of the equivalence are all non-equivalent powers of 2.

Left Subcase 1: $2e_2 + k \equiv i \pmod{m}$

$2e_2 + k \equiv i \pmod{m} \rightarrow q \equiv i \pmod{m}$. With this, Equivalence 27 turns into:

$$2^{i+1} + 2^k \equiv 2^{2i} + 2^{\frac{m}{2}+e_2+k} + 1 \pmod{2^m - 1} \tag{28}$$

Left Side of Equivalence 28 Sub-subcase 1: $i + 1 \equiv k \pmod{m}$

We have that $i + 1 \equiv k \pmod{m} \rightarrow q + 1 \equiv k \pmod{m} \rightarrow q + 1 \equiv q - 2e_2 \pmod{m} \rightarrow 1 \equiv -2e_2 \pmod{m} \rightarrow 2$ has a multiplicative inverse in Z_m^* which is a contradiction as $(2, m) = 2$. Finally, it is clear that $i + 1, k$ are both not equivalent $\pmod{m}$ to m as $0 < i < \frac{m}{2} \rightarrow i + 1 \leq \frac{m}{2} < m$ and $0 < k < m$.

Right Side of Equivalence 28 Sub-subcase 1: $2i \equiv \frac{m}{2} + e_2 + k \pmod{m}$

If $2i \equiv \frac{m}{2} + e_2 + k \pmod{m}$ then, $2q \equiv \frac{m}{2} + e_2 + (q - 2e_2) \pmod{m} \rightarrow q \equiv \frac{m}{2} - e_2 \pmod{m} \rightarrow q \equiv x \pmod{m} \rightarrow i \equiv x \pmod{m} \rightarrow i = x$ which is a contradiction.

Right Side of Equivalence 28 Sub-subcase 2: $2i \equiv 0 \pmod{m}$

If $2i \equiv 0 \pmod{m} \rightarrow$ i is a zero divisor, but this cannot be the case as $(i, m) = 1$.

Right Side of Equivalence 28 Sub-subcase 3: $\frac{m}{2} + e_2 + k \equiv 0 \pmod{m}$

If $\frac{m}{2} + e_2 + k \equiv 0 \pmod{m}$ then, $\frac{m}{2} + e_2 + k = \frac{m}{2} + e_2 + (q - 2e_2) = \frac{m}{2} - e_2 + q \equiv 0 \pmod{m} \rightarrow x + q \equiv 0 \pmod{m} \rightarrow x \equiv -i \pmod{m}$. However, $j \equiv -i \pmod{m} \rightarrow x \equiv j \pmod{m}$ but this is a contradiction as $x \neq j$ and both of them are less than m.

Since $0 < i + 1, k, 2i < m$ then all the corresponding powers of 2 are less than 2^m and we can replace $2^{\frac{m}{2}+e_2+k}$ by the equivalence class representative (say 2^c, for $0 < c < m$). Then Equivalence 28 turns into the equality:

$$2^{i+1} + 2^k = 2^{2i} + 2^c + 1$$

This is a contradiction as we have an even integer on the left side of the equality and an odd integer on the right.

Left Subcase 2: $2e_2 + k \equiv k \pmod{m}$

If $2e_2 + k \equiv k \pmod{m}$, then $2e_2 \equiv 0 \pmod{m} \rightarrow 2e_2 = 0$ (as $0 < e_2 < \frac{m}{2}$) which is a contradiction.

Left Subcase 3: $i \equiv k \pmod{m}$

$i \equiv k \pmod{m}$ transforms Equivalence 27 to $2^{2e_2+i} + 2^i + 2^i \equiv 2^{2i} + 2^{\frac{m}{2}+e_2+i} + 1 \pmod{2^m - 1}$ which gives the equivalence:

$$2^{2e_2+i} + 2^{i+1} \equiv 2^{2i} + 2^{\frac{m}{2}+e_2+i} + 1 \pmod{2^m - 1} \tag{29}$$

Right Side of Equivalence 29 Sub-subcase 1: $2e_2 + i \equiv i + 1 \pmod{m}$

$2e_2 + i \equiv i + 1 \pmod{m} \rightarrow 2e_2 \equiv 1 \pmod{m} \rightarrow 2$ has a multiplicative inverse in Z_m^* which is not possible as $(2, m) = 2$.

Right Side of Equivalence 29 Sub-subcase 2: $2e_2 + i \equiv 0 \pmod{m}$

$2e_2 + i \equiv 0 \pmod{m} \rightarrow q \equiv 0 \pmod{m}$ which is a contradiction as $e_2 < q < m$

Right Side of Equivalence 29 Sub-subcase 3: $i + 1 \equiv 0 \pmod{m}$

If $i + 1 \equiv 0 \pmod{m}$ then $i + 1 = 0$ (as $0 < i < \frac{m}{2} \rightarrow 1 < i + 1 \leq \frac{m}{2} < m$) which is a contradiction as i is a positive integer.

Left Side of Equivalence 29 Sub-subcase 1: $2i \equiv \frac{m}{2} + e_2 + i \pmod{m}$

If $2i \equiv \frac{m}{2} + e_2 + i \pmod{m}$ then $i \equiv \frac{m}{2} + e_2 \pmod{m} \rightarrow i \equiv y \pmod{m} \rightarrow i = y$ as both of these integers are less than m. This is a contradiction as $i \neq y$.

Left Side of Equivalence 29 Sub-subcase 2: $2i \equiv 0 \pmod{m}$

If $2i \equiv 0 \pmod{m}$ then we have a contradiction as $(i, m) = 1$ and thus i is a unit in Z_m^*.

Left Side of Equivalence 29 Sub-subcase 3: $\frac{m}{2} + e_2 + i \equiv 0 \pmod{m}$

If $\frac{m}{2} + e_2 + i \equiv 0 \pmod{m}$ then we have $\frac{m}{2} + e_2 \equiv -i \pmod{m} \rightarrow y \equiv -i \pmod{m} \rightarrow y \equiv j \pmod{m} \rightarrow y = j$, which is a contradiction as $j \neq y$.

If we substitute 2^{2e_2+i} and $2^{\frac{m}{2}+e_2+i}$ for $2^c, 2^{c_2}$ respectively $(0 < c, c_2 < m)$ i.e their equivalence class representative, then Equivalence 29 becomes the equality:

$$2^c + 2^{i+1} = 2^{2i} + 2^{c_2} + 1,$$

Which is a contradiction as we have an even integer being equal to an odd integer.

Now, we move on to the right side of the Equivalence 27.

Right Subcase 1: $2i \equiv \frac{m}{2} + e_2 + k \pmod{m}$

If $2i \equiv \frac{m}{2} + e_2 + k \pmod{m}$, then $2i \equiv y + k \pmod{m}$. Equivalence 27 changes to $2^{2e_2+k} + 2^i + 2^k \equiv 2^{2i} + 2^{2i} + 1$ which implies that:

$$2^{2e_2+2i-y} + 2^i + 2^{2i-y} \equiv 2^{2i+1} + 1 \pmod{2^m - 1} \tag{30}$$

Left Side of Equivalence 30 Sub-subcase 1: $2e_2 + 2i - y \equiv i \pmod{m}$

If $2e_2 + 2i - y \equiv i \pmod{m}$ then $q \equiv i \pmod{m}$ (as $2e_2 + 2i - y = e_2 + k = q$) which we showed in Left Subcase 1 of Equivalence 27 leads to a contradiction.

Left Side of Equivalence 30 Sub-subcase 2: $2e_2 + 2i - y \equiv 2i - y \pmod{m}$

If $2e_2 + 2i - y \equiv 2i - y \pmod{m}$ then $2e_2 \equiv 0 \pmod{m} \rightarrow 2e_2 = 0$ which is a contradiction as $0 < e_2 < \frac{m}{2}$.

Left Side of Equivalence 30 Sub-subcase 3: $i \equiv 2i - y \pmod{m}$

If $i \equiv 2i - y \pmod{m}$ then $0 \equiv i - y \pmod{m} \rightarrow i \equiv y \pmod{m} \rightarrow i = y$ which is a contradiction as $0 < i < \frac{m}{2} < y < m$.

Right Side of Equivalence 30 Sub-subcase 1: $2i + 1 \equiv 0 \pmod{m}$

If $2i + 1 \equiv 0 \pmod{m}$ then $-2i \equiv 1 \pmod{2^m - 1} \rightarrow 2$ has a multiplicative inverse in Z_m^* which cannot happen as $(2, m) = 2$.

We have that on the left side of the equivalence, the powers of 2 are less than 2^m as $2e_2 + 2i - y = q < m, i < m$ and $k < m$. For the right side, since $0 < i < \frac{m}{2}$ then $0 < 2i \leq \frac{m}{2} < m \rightarrow 0 < 2i < m - 1 \rightarrow 0 < 2i + 1 \leq m - 1 < m$. Thus, Equivalence 30 becomes the equality:

$$2^{2e_2+2i-y} + 2^i + 2^{2i-y} = 2^{2i+1} + 1,$$

This cannot be true as we have an even integer on the left of the equality and an odd integer on the right.

Right Subcase 2: $2i \equiv 0 \pmod{m}$

$2i \equiv 0 \pmod{m}$ is a contradiction as $(i, m) = 1$.

Right Subcase 3: $\frac{m}{2} + e_2 + k \equiv 0 \pmod{m}$

If $\frac{m}{2} + e_2 + k \equiv 0 \pmod{m}$ then $\frac{m}{2} + e_2 + (q - 2e_2) \equiv 0 \pmod{m} \rightarrow \frac{m}{2} - e_2 + q \equiv 0 \pmod{m} \rightarrow x + q \equiv 0 \pmod{m} \rightarrow q \equiv -x \pmod{m} \rightarrow q \equiv y \pmod{m} \rightarrow q = y$ (as both y and q are less than m). Substituting this into Equivalence 27 we have that $2^{2e_2+k} + 2^i + 2^k = 2^y + 2^i + 2^{y-2e_2} \equiv 2^{2i} + 2^0 + 1 \pmod{2^m - 1}$ and from this we deduce the equivalence:

$$2^y + 2^i + 2^{y-2e_2} \equiv 2^{2i} + 2 \pmod{2^m - 1} \tag{31}$$

Left Side of Equivalence 31 Sub-subcase 1: $y \equiv i \pmod{m}$

If $y \equiv i \pmod{m}$ then $i = y$ as both of them are less than m which is a contradiction.

Left Side of Equivalence 31 Sub-subcase 2: $y \equiv y - 2e_2 \pmod{m}$

If $y \equiv y - 2e_2 \pmod{m}$ then $0 \equiv -2e_2 \pmod{m} \rightarrow 0 = -2e_2$ which is a contradiction as $0 < e_2 < \frac{m}{2}$.

Left Side of Equivalence 31 Sub-subcase 3: $i \equiv y - 2e_2 \pmod{m}$

If $i \equiv y - 2e_2 \pmod{m}$ then $i \equiv \frac{m}{2} + e_2 - 2e_2 \pmod{m} \rightarrow i \equiv \frac{m}{2} - e_2 \pmod{m} \rightarrow i \equiv x \pmod{m} \rightarrow i = x$ which is a contradiction.

Right Side of Equivalence 31 Sub-subcase 1: $2i \equiv 1 \pmod{m}$

If $2i \equiv 1 \pmod{m} \rightarrow 2$ has a multiplicative inverse in Z_m^* which is not possible as $(2, m) = 2$.

We have that on the left side of Equivalence 31, y, i and $y - 2e_2$ are all less than m and for the right side $2i \not\equiv 0 \pmod{m}$ as $0 < i < \frac{m}{2}$. Thus, the equivalence can be changed for the equality:

$$2^y + 2^i + 2^{y-2e_2} = 2^{2i} + 2$$

This is a contradiction as we have two distinct 2-adic representations of an integer, which by Lemma 3 is not possible.

With the subcases for the left and right sides of Equivalence 27 all leading to contradictions, we conclude that Equivalence 27 does not hold. On the left side, we have that $2^i < 2^m$, $2^k < 2^m$ and $2^{2e_2+k} < 2^m$ as $i < m$ $k < m$ and $2e_2 + k = q < m$. On the right side we have that $2^{2i} < 2^m$, $1 < 2^m$ as $0 < 1 < 2i < m$ and $2^{\frac{m}{2}+e_2+k} \not\equiv 2^m \pmod{2^m - 1}$ as per Right Subcase 3. We substitute $2^{\frac{m}{2}+e_2+k}$ for its equivalence class representative, say 2^c for $0 < c < m$. Thus, we can turn Equivalence 27 into the equality:

$$2^{2e_2+k} + 2^i + 2^k = 2^{2i} + 2^c + 1$$

Which is a contradiction as we have an even integer on the left and an odd integer on the right. Thus, no integer q exists such that $2^q(2^{2x} - 2^x + 1) \equiv 2^{2i} - 2^i + 1$. This means that $2^{2x} - 2^x + 1$ and $2^{2i} - 2^i + 1$ do not share a cyclotomic coset, and thus neither do $2^{2y} - 2^y + 1$ and $2^{2i} - 2^i + 1$ as $2^{2x} - 2^x + 1$ and $2^{2y} - 2^y + 1$ must share a cyclotomic coset.

For the size of the cyclotomic cosets that contain these exponents, assume that there exists one such cyclotomic coset whose size is a where $a|m$. The largest possible divisor of m (aside from itself) is $\frac{m}{2}$, and as such, we shall consider $0 < a \leq \frac{m}{2}$. Let us say that the pair of exponents $2^{2i} - 2^i + 1, 2^{2j} - 2^j + 1$ are in this cyclotomic coset, with i, j defined as before. Since the size of the cyclotomic coset is a, we have that $2^a(2^{2i} - 2^i + 1) \equiv 2^{2i} - 2^i + 1 \pmod{2^m - 1}$ and the following equivalence must be true:

$$2^{2i+a} - 2^{i+a} + 2^a \equiv 2^{2i} - 2^i + 1 \pmod{2^m - 1} \tag{32}$$

We show that no value of $0 < a \leq \frac{m}{2}$ can satisfy this equivalence, and thus an exponent of the form $2^{2i} - 2^i + 1$ cannot be found in a cyclotomic coset of size less than m. We add that, since $i = \frac{m}{2} - e_1, j = \frac{m}{2} + e_1$ then $0 < e_1 < \frac{m}{2}$ and we have that $2i \equiv -2e_1 \pmod{m}$ and $2j \equiv 2e_1 \pmod{m}$. We can divide all possible values that a can take into three cases, $0 < a < 2e_1, a = 2e_1$ and $2e_1 < a \leq \frac{m}{2}$.

First Case: $0 < a < 2e_1$

Since $a < 2e_1$, then $2i + a = m - 2e_1 + a < m$. Furthermore, $a < i + a < 2i + a < m$. Thus, the left side of Equivalence 32 sums to an even integer less than $2^m - 1$. For the right side of the equivalence, we have that $0 < i < 2i < m$ and thus the sum of the powers of 2 result in an odd integer less than $2^m - 1$. We can then change Equivalence 32 for the equality:

$$2^{2i+a} - 2^{i+a} + 2^a = 2^{2i} - 2^i + 1,$$

which is a contradiction as we have an odd integer equal to an even integer.

Second Case: $a = 2e_1$

Substituting into Equivalence 32 we have $2^{2i+a} - 2^{i+a} + 2^a = 2^{m-2e_1+2e_1} - 2^{\frac{m}{2}-e_1+2e_1} + 2^{2e_1} \equiv 2^0 - 2^{\frac{m}{2}+e_1} + 2^{2j} \pmod{2^m - 1} \rightarrow 2^{2j} - 2^j + 1 \equiv 2^{2i} - 2^i + 1 \pmod{2^m - 1}$ which gives us that:

$$2^{2j} - 2^j \equiv 2^{2i} - 2^i \pmod{2^m - 1} \tag{33}$$

Note that, $2j \equiv 2e_1 \pmod{m}$ and $j = \frac{m}{2} + e_1$. Since $e_1 < \frac{m}{2}$ then $2e_1 < \frac{m}{2} + e_1 < m \rightarrow 2^{2e_1} < 2^{\frac{m}{2}+e_1} < 2^m$. We have that, $2^{2j} - 2^j \equiv 2^{2e_1} - 2^{\frac{m}{2}+e_1} \pmod{2^m - 1} \rightarrow 2^{2e_1} - 2^{\frac{m}{2}+e_1} \equiv 2^{2i} - 2^i \pmod{2^m - 1}$. Note that, $2^{2e_1} - 2^{\frac{m}{2}+e_1} = -(2^{\frac{m}{2}+e_1} - 2^{2e_1}) \equiv (2^m - 1) - (2^{\frac{m}{2}+e_1} - 2^{2e_1}) \pmod{2^m - 1}$, with $(2^m - 1) - (2^{\frac{m}{2}+e_1} - 2^{2e_1})$ being an odd integer less than $2^m - 1$. We substitute Equivalence 33 for the equality:

$$(2^m - 1) - (2^{\frac{m}{2}+e_1} - 2^{2e_1}) = 2^{2i} - 2^i$$

which is a contradiction as the left side of the equation is an odd integer while the right side is an even integer.

Third Case: $2e_1 < a \leq \frac{m}{2}$

Let us say that $a = 2e_1 + k$ for some integer k such that $1 < k < \frac{m}{2} - 1$. We have these bounds for k as e_1 is at minimum one and thus $2e_1$ is at minimum 2. On the other hand, since $a \leq \frac{m}{2}$ then k can at most be $\frac{m}{2} - 2$ as $2e_1$ is at the minimum 2. We substitute this value on Equivalence 32 and we have $2^{2i+a} - 2^{i+a} + 2^a = 2^{m-2e_1+2e_1+k} - 2^{\frac{m}{2}-e_1+2e_1+k} + 2^{2e_1+k} = 2^{m+k} - 2^{\frac{m}{2}+e_1+k} + 2^{2e_1+k} \equiv 2^k - 2^{j+k} + 2^{2e_1+k} \pmod{2^m - 1} \rightarrow 2^{2e_1+k} - 2^{j+k} + 2^k \equiv 2^{2i} - 2^i + 1 \pmod{2^m - 1}$. We can rewrite this equivalence as:

$$2^{2e_1+k} + 2^i + 2^k \equiv 2^{2i} + 2^{j+k} + 1 \pmod{2^m - 1} \tag{34}$$

We will show that this equivalence cannot be true. First, we show that all the powers of 2 on the left side are non-equivalent $\pmod{2^m - 1}$ to each other, then we do the same for the right side. We substitute the equivalence by equality by using the equivalence class representatives of the terms (and if they are all distinct powers of 2, then they will sum to less than 2^m) and reach a contradiction.

Left Subcase 1: $2e_1 + k \equiv i \pmod{m}$

$2e_1 + k \equiv i \pmod{m} \rightarrow 2j + k \equiv i \pmod{m}$. Since $j + i = m$, then $j \equiv -i \pmod{m}$. Thus, we have $2j + k \equiv i \pmod{m} \rightarrow k \equiv 3i \pmod{m}$. Substituting into Equivalence 34 results in $2^i + 2^i + 2^{3i} \equiv 2^{2i} + 2^{j+3i} + 1 \pmod{2^m - 1} \rightarrow 2^{i+1} + 2^{3i} \equiv 2^{2i} + 2^{2i} + 1 \pmod{2^m - 1}$ This is reduced to the equivalence:

$$2^{i+1} + 2^{3i} \equiv 2^{2i+1} + 1 \pmod{2^m - 1}. \tag{35}$$

Left Side of Equivalence 35 Sub-subcase 1: $i + 1 \equiv 3i \pmod{m}$

We have that $i + 1 \equiv 3i \pmod{m} \rightarrow 1 \equiv 2i \pmod{m}$ which implies 2 has a multiplicative inverse in Z_m^* which is not possible as $(2, m) = 0$.

Right Side of Equivalence 35 Sub-subcase 1: $2i + 1 \equiv 0 \pmod{m}$

We have that $2i + 1 \equiv 0 \pmod{m} \rightarrow -2i \equiv 1 \pmod{m}$ which is the same contradiction as the previous subcase.

Finally, $0 < i + 1 \leq \frac{m}{2}$ and thus $i + 1 \not\equiv 0 \pmod{m}$ and $3i \not\equiv 0 \pmod{m}$ as otherwise it would imply that i is a zero divisor in Z_m^* which it cannot be as $(i, m) = 1$. This means that either $2^{3i} < 2^m$ or it is equivalent $\pmod{2^m - 1}$ to some power of 2 less than $2^m - 1$. Select 2^c, $0 < c < m$ as the equivalence class representative of 2^{3i}. Furthermore $0 < i < \frac{m}{2} \rightarrow 0 <$

$2i < m \to 0 < 2i < m-1 \to 1 < 2i+1 \leq m-1 < m$. Thus, Equivalence 35 can be changed for the equality:

$$2^{i+1} + 2^c = 2^{2i+1} + 1$$

This is a contradiction as we have an even integer on the left side of the equivalence and an odd integer on the right.

Left Subcase 2: $2e_1 + k \equiv k \pmod{m}$

$2e_1 + k \equiv k \pmod{m} \to 2e_1 \equiv 0 \pmod{m}$ which is a contradiction as $0 < e_1 < \frac{m}{2} \to 0 < 2e_1 < m$.

Left Subcase 3: $i \equiv k \pmod{m}$

$i \equiv k \pmod{m} \to k = i$ (as both of them are less than m) which implies that $a = 2e_1 + \frac{m}{2} - e_1 = \frac{m}{2} + e_1 = j$ but this is a contradiction as $a \leq \frac{m}{2}$.

Now for the right side of Equivalence 34.

Right Subcase 1: $2i \equiv j + k \pmod{m}$

$2i \equiv j + k \pmod{m} \to 3i \equiv k \pmod{m}$ which is the same contradiction as in Left Subcase 1.

Right Subcase 2: $2i \equiv 0 \pmod{m}$

$2i \equiv 0 \pmod{m} \to i$ is a zero divisor on Z_m^* which is a contradiction as $(i, m) = 1$.

Right Subcase 3: $j + k \equiv 0 \pmod{m}$

$j + k \equiv 0 \pmod{m} \to k \equiv i \pmod{m}$ which is the same contradiction as in Left Subcase 3.

Since $0 < a, i, k < m$, then all the terms on the left side of the equivalence are less than 2^m. For the right side, $0 < 2i < m$, and we can substitute 2^{j+k} for its equivalence class representative (say 2^c, for $0 < c < m$). With this, we can change Equivalence 34 for the equality:

$$2^{2e_1+k} + 2^i + 2^k = 2^{2i} + 2^c + 1$$

Which is a contradiction as we have an odd integer equal to an even integer. Thus Equivalence 34 is not true. Therefore, no such a such that $0 < a \leq \frac{m}{2}$ can exists such that $2^a(2^{2i} - 2^i + 1) \equiv 2^{2i} - 2^i + 1 \pmod{2^m - 1}$. This means that $\frac{m}{2} < a$ with $a|m$ and thus $a = m$. □

Corollary 2 (Exact number of Kasami-Welch Near-Bent Functions Over $\mathbb{F}_{2^m}$)**.** Let the Kasami-Welch Boolean near-bent function in m variables be of the form $Tr(x^{2^{2l}-2^l+1})$ such that $1 \leq l \leq m-1$ and $(l,m) = 1$. Then, the number of non-equal near-bent functions generated is exactly $\frac{\Phi(m)}{2}$.

Proof: Let $Tr(x^{2^{2l}-2^l+1})$ be the Kasami-Welch near bent functions in m variables with $(l,m) = 1$. By Definition 5, $Tr(x^{2^{2l}-2^l+1}) = x^{2^{2l}-2^l+1} + x^{2(2^{2l}-2^l+1)} + \cdots + x^{2^{m-1}(2^{2l}-2^l+1)}$. The exponents in the terms of the sum are the elements in the cyclotomic coset that contains $2^{2l}-2^l+1$, i.e $C(2^{2i}-2^i+1)$. It is clear that for any integer i, $Tr(x^{2^{2l}-2^l+1}) = Tr(x^{2^i(2^{2l}-2^l+1)})$ as $2^i(2^{2l}-2^l+1)$ is also in $C(2^{2l}-2^l+1)$, and as such $Tr(x^{2^i(2^{2l}-2^l+1)}) = x^{2^i(2^{2l}-2^l+1)} + x^{2^{i+1}(2^{2l}-2^l+1)} + \cdots + x^{2^{i+m-1}(2^{2l}-2^l+1)}$ with all of the exponents of the terms forming $C(2^{2i}-2^i+1)$. For an integer m, we have precisely two Kasami-Welch exponents in each cyclotomic coset that contains them. The possible values of l are $1, 2, 3 \ldots, m-1$, and only those such that $(l,m) = 1$ lead to near-bent functions. The number of such values is given by Euler's ϕ function, and the total is divided by 2, as the exponents appear in pairs in the coset and these pairs lead to the same Boolean function. Thus, $\frac{\Phi(m)}{2}$ will be the total number of non-equal Kasami-Welch near-bent functions generated.

□

4.2.1. *Improved Proofs.*

Theorem 8 (The Number of Gold Exponents in The Cyclotomic Cosets $\pmod{2^m-1}$)**.** Let 2^l+1 be a Gold exponent. Gold exponents occur in pairs of the form $2^i+1, 2^j+1$ in a cyclotomic coset of size a $\pmod{2^m-1}$ where $j+i=m$. Furthermore, the size of the cyclotomic coset is either m or $\frac{m}{2}$.

Proof: First, we shall assume a cyclotomic coset of size m. Say we have the cyclotomic coset containing 2^i+1: C$(2^i+1) = \{2^i+1, 2*(2^i+1), \ldots 2^{m-1}*(2^i+1)\}$. Consider j such that $j+i=m$,$0 < j, i < m$ (if j = i then the pair of 2^i+1 is itself) then we have $2^j(2^i+1) \in C(2^i+1)$ and

$$2^j(2^i+1) \equiv 2^{i+j}+2^j \pmod{2^m-1} \equiv 2^m+2^j \pmod{2^m-1} \equiv 1+2^j \pmod{2^m-1}. \quad (36)$$

Thus, we found a "pair" of Gold exponents in the same cyclotomic coset. Now we show that no other Gold exponent can be found. Suppose there exists another pair of Gold exponents, $2^x+1, 2^y+1$, such that $x+y=m$, $x \neq i, x \neq j, y \neq i, y \neq j$ for $0 < x < y < m$. Since this requires that we have four non-equal Gold exponents in the same cyclotomic coset, then $m \geq 4$. We will prove that this second pair cannot exist in the cyclotomic coset. Since all these exponents share a cyclotomic coset, then there must exist some integer $0 < q < m$ such that $2^q(2^x+1) \equiv 2^i+1 \pmod{2^m-1}$. Clearly $q \neq y$ or else $2^y+1 \equiv 2^i+1 \pmod{2^m-1} \rightarrow 2^y \equiv 2^i \pmod{2^m-1} \rightarrow y \equiv i \pmod{m} \rightarrow y = i$ which is a contradiction. We have the following equivalence:

$$2^{x+q}+2^q \equiv 2^i+1 \pmod{2^m-1} \quad (37)$$

We can divide the possible values that q takes into two cases, $0 < q < y, y < q < m$

Case 1: $0 < q < y$

$\rightarrow q + x < m$ and $q + x \neq q \rightarrow 2^{x+q} + 2^q < 2^m - 1$ (by Lemma 2). This means that $2^{x+q} + 2^q$ sums to an even integer less than $2^m - 1$. For the right side of Equivalence 37, $i < m \rightarrow 2^i < 2^m - 1 \rightarrow 2^i + 1$ sums to an odd integer less than $2^m - 1$. Thus, we can substitute the equivalence by equality and obtain an equality between an odd and an even integer which is a contradiction.

Case 2: $y < q < m$

Since $x + y = m$, then $m < x + q$. Let us say that $y + x = e + m$ for some integer $e < m$. Then Equivalence 37 changes to:

$$2^e + 2^q \equiv 2^i + 1 \pmod{2^m - 1} \tag{38}$$

The left side of the equivalence sums to an even integer less than $2^m - 1$ as both e, q are less than m. The right side of the equivalence sums to an odd integer less than $2^m - 1$ by the same argument as the previous case. Thus, we reach a contradiction as we reach an equality between an even and an odd integer.

Now we consider the size of the cyclotomic coset. The size of the cyclotomic coset must be a divisor of m. Since we are assuming it is not m, the size of the cyclotomic coset shall be an integer a such that $0 < a \leq \frac{m}{2}$, where a is a divisor of m. Thus, $2^a(2^i + 1) \equiv 2^i + 1 \pmod{2^m - 1}$. We shall consider the following equivalence:

$$2^{a+i} + 2^a \equiv 2^i + 1 \pmod{2^m - 1} \tag{39}$$

We shall consider four cases, 1)$i = \frac{m}{2}, a < \frac{m}{2}$, 2) $i = \frac{m}{2}, a = \frac{m}{2}$, 3) $i < \frac{m}{2}, a < \frac{m}{2}$ and 4) $i < \frac{m}{2}, a = \frac{m}{2}$. We note that $\frac{m+1}{2} - 1 < \frac{m}{2}$ for all m. Furthermore, for m odd the maximum possible value of a is $\frac{m}{3}$ Thus, we consider $i, a < \frac{m}{2}$ which are cases that are satisfied when $i \leq \frac{m+1}{2} - e_1$ and $a \leq \frac{m}{3}$ (corresponding to m being odd).

Case 1: $i = \frac{m}{2}, a < \frac{m}{2}$

$\rightarrow a + i < m, \rightarrow 2^{a+i} + 2^a$ is an even integer less than $2^m - 1$, while for the right side of Equivalence 39 we have that $2^i + 1$ is an odd integer less than $2^m - 1$. Hence, we have an equality between an odd and an even integer which is a contradiction.

Case 2: $i = \frac{m}{2}, a = \frac{m}{2}$

$\rightarrow a+i=m \rightarrow 2^{a+i} \equiv 2^0 \pmod{2^m-1} \rightarrow 1+2^a \equiv 2^i+1 \pmod{2^m-1} \rightarrow 2^a \equiv 2^i \pmod{2^m-1}$. This case holds true.

Case 3: $i < \frac{m}{2}, a < \frac{m}{2}$

As in Case 1, $2^{a+i} < 2^m$ and thus we will have that $2^{a+i}+2^a$ sums to an even integer less than 2^m-1 while 2^i+1 sums to an odd integer less than 2^m-1. This is an equality between an even and an odd integer which is a contradiction.

Case 4: $i < \frac{m}{2}, a = \frac{m}{2}$

As in cases 1 and 3, $2^{a+i} < 2^m$ and thus we will have that $2^{a+i}+2^a$ sums to an even integer less than 2^m-1 while 2^i+1 sums to an odd integer less than 2^m-1. This is an equality between an even and an odd integer which is a contradiction.

The only case where we had a cyclotomic coset of size distinct from m is Case 2. This case can be turned into an if and only if, that is, $a=\frac{m}{2}$ if and only if $i=\frac{m}{2}$. First, if $a=\frac{m}{2}$ then we see from cases 2 and 4 that we reach a contradiction unless $i=\frac{m}{2}$. For the other direction, if $i=\frac{m}{2}$ from cases 1 and 2 we know that we get a contradiction unless $a=\frac{m}{2}$.

With these results, we thus have that the size of the cyclotomic coset containing a pair of Gold exponents must be m unless the pair of exponents are of the form $2^{\frac{m}{2}}+1$, in which case the size of the cyclotomic coset is $a=\frac{m}{2}$.

□

4.3. **New proof for Yoshiara's result on the CCZ-equivalence of Gold and Kasami-Welch Near-Bent Functions.** In the previous subsection, we have finished proving that Gold and Kasami-Welch exponents occur in unique pairs on a cyclotomic coset. This result is one part of the condition of cyclotomic equivalence for (m,m) Boolean power functions of the form $f(x)=x^d$. In this subsection, we prove that the Gold and Kasami-Welch exponents, as defined previously, do not meet the second criteria for cyclotomic equivalence.

Theorem 9 (Inverse of a Gold Exponent in a Cyclotomic Coset)**.** Consider the Gold exponents 2^i+1, 2^x+1 such that $2^q(2^i+1) \not\equiv 2^x+1 \pmod{2^m-1}, 0<i,x<m$ for any integer q and $(i,m)=(x,m)=1, m>3$. Then, there does not exist an integer $0 \leq a < m$ such that $(2^i+1)*(2^x+1) \equiv 2^a \pmod{2^m-1}$.

Proof: For i,x there exists some integer j,y respectively such that $i+j=m, x+y=m$. Since the exponents do not share a cyclotomic coset, then $i \neq x$ and $j \neq y$. Since $(i,m)=(x,m)=1$, and by Theorem 5, $(2^i+1, 2^j+1)$, $(2^x+1, 2^y+1)$ are unique Gold exponents pairs in their respective cyclotomic cosets. We can assume, without loss of generality, that i,x are integers such that $0<i,x<\frac{m}{2}$ (with $\frac{m}{2}<j,y<m$). Note that $(2^i+1)*(2^x+1) \equiv 2^a \pmod{2^m-1} \rightarrow 2^{m-a}(2^i+1)*(2^x+1) \equiv 1 \pmod{2^m-1} \rightarrow$ the Gold exponents considered have an inverse $\pmod{2^m-1}$. We have two main cases, m even and m odd.

Case 1: m even

For m even, we have that $2^m \equiv 1 \pmod 3$, and as such, $2^m - 1 \equiv 0 \pmod 3$. On the other hand, since i, x must be relatively prime to m then they must be odd integers. However, for r an odd integer, $2^r \equiv 2 \pmod 3 \rightarrow 2^r + 1 \equiv 0 \pmod 3$. This means that $(2^x + 1, 2^m - 1) \geq 3, (2^i + 1, 2^m - 1) \geq 3 \rightarrow$ the exponents do not have a multiplicative inverse and thus $(2^i + 1) * (2^x + 1) \equiv 2^a \pmod{2^m - 1}$ cannot be true for any integer a.

Case 2: m odd

We note that if $m = 3$, there are only two Gold exponents defined (for $i = 1, j = 2$). Thus, we cannot define such x for which the multiplicative inverse shares a cyclotomic coset with another Gold exponent, but not the exponent itself. Because of this, we consider $m > 3$. We have that $(2^i + 1) * (2^x + 1) \equiv 2^{x+i} + 2^x + 2^i + 1 \pmod{2^m - 1}$ This means that:

$$2^{x+i} + 2^x + 2^i + 1 \equiv 2^a \pmod{2^m - 1}. \tag{40}$$

We shall prove that the left side of the equivalence is equivalent to an odd integer less than $2^m - 1$ but greater than one, while the right-hand side must be equivalent to an even integer less than $2^m - 1$ or 1. We verify that all the terms on the left-hand side are not equivalent $\pmod{2^m - 1}$ to each other.

Left Side of Equivalence 40 Subcase 1: $x + i \equiv x \pmod m$

$x + i \equiv x \pmod m \rightarrow i \equiv 0 \pmod m$ which is a contradiction as $0 < i < m$.

Left Side of Equivalence 40 Subcase 2: $x + i \equiv i \pmod m$

$x + i \equiv i \pmod m \rightarrow x \equiv 0 \pmod m$ which is a contradiction as $0 < x < m$.

Left Side of Equivalence 40 Subcase 3: $x + i \equiv 0 \pmod m$

$x + i \equiv 0 \pmod m \rightarrow x \equiv -i \pmod m \rightarrow x \equiv j \pmod m \rightarrow x = j$ which is a contradiction as $0 < x < \frac{m}{2}$ while $\frac{m}{2} < j$.

Left Side of Equivalence 40 Subcase 4: $x \equiv i \pmod m \pmod m$

$x \equiv i \pmod m \rightarrow x = i$ is a contradiction as $x \neq i$.

Left Side of Equivalence 40 Subcase 5: $x \equiv 0 \pmod m$

$x \equiv 0 \pmod m$ is a contradiction as $0 < x < m$.

Left Side of Equivalence 40 Subcase 6: $i \equiv 0 \pmod m$

$i \equiv 0 \pmod{m}$ is a contradiction as $0 < i < m$.

We note that since $x, i < \frac{m}{2}$ then $x + i < m$ the left side of the equation is the sum of four non-equivalent powers of 2. Since $m > 3$ is odd, we have that the terms on the right sum to a number less than 2^m while the right side will be less than 2^m as $0 \leq a < m$. We obtain the equality:

$$2^{x+i} + 2^x + 2^i + 1 = 2^a$$

This leads to a contradiction as the left side is an odd integer greater than one, and the right side is either 1 or an even integer. Therefore, there does not exist such an integer a for which $(2^i + 1) * (2^x + 1) \equiv 2^a \pmod{2^m - 1}$ for $m > 3$.

□

CCZ-equivalence between two Gold functions (and the number of non-equivalent functions being $\frac{\Phi(m)}{2}$) has been shown by Budaghyan in [4]. We arrive at a proof for this by combining Theorem 5 with Theorem 9

Theorem 10 (Cyclotomic-Equivalent Gold Functions)**.** Let $f(x) = x^{2^i+1}, g(x) = x^{2^x+1}$ be vectorial Gold Boolean functions $\mathbb{F}_{2^m} \to \mathbb{F}_{2^m}$, with $m > 3$. If $i \neq x, 0 < i, x < m$ and $(i, m) = (x, m) = 1$, then, these functions are cyclotomic-equivalent if and only if $i + x = m$.

Proof: For the forwards direction, if the functions are cyclotomic-equivalent, then either $(2^x + 1)(2^i + 1) \equiv 2^a \pmod{2^m - 1}$ for some integer $0 \leq a < m$ or the exponents are in the same cyclotomic coset. We have proven in Theorem 9 that no such a can exist for $m > 3$. The exponents must then share a cyclotomic coset which is only possible if $i + x = m$.

For the backwards direction, if $i + x = m$, then $2^i + 1, 2^x + 1$ share a cyclotomic coset as per Theorem 5. Thus, the functions are cyclotomic-equivalent by definition. Thus, we have our result.

□

The total number of Gold exponents $2^l + 1$ that lead to APN functions over $\mathbb{F}_{2^m}$ is given by the number of integers $0 < l < m$ such that $(l, m) = 1$. This number is given by $\Phi(m)$. Since the exponents occur in pairs $2^i + 1, 2^x + 1$ such that $x + i = m$ in a cyclotomic coset, then the total number of non CCZ-equivalent Gold exponent functions is given by $\frac{\Phi(m)}{2}$. This is a re-statement of the result from Theorem 2.1 in [4]. We obtain a similar result for the Kasami-Welch case.

Theorem 11 (Inverse of a Kasami-Welch exponent in a Cyclotomic Coset)**.** Let $f, g : \mathbb{F}_{2^m} \to \mathbb{F}_{2^m}, f(x) = x^{2^{2i}-2^i+1}, g(x) = x^{2^{2x}-2^x+1}$ be vectorial Kasami-Welch Boolean functions such that $2^q(2^{2i} - 2^i + 1) \not\equiv 2^{2x} - 2^x + 1 \pmod{2^m - 1}, 0 < i, x < m$ for some integer q and $(i, m) = (x, m) = 1, m > 5$. Then, there does not exist an integer $0 \leq a < m$ such that $(2^{2i} - 2^i + 1) * (2^{2x} - 2^x + 1) \equiv 2^a \pmod{2^m - 1}$.

Proof: For i, x, there exists some integer j, y respectively such that $i + j = m, x + y = m$. Since the exponents do not share a cyclotomic coset, then $i \neq x$ and $j \neq y$. Since $(i, m) = (x, m) = 1$, and by Theorems 6 and 7, $(2^{2i} - 2^i + 1, 2^{2j} - 2^j + 1)$, $(2^{2x} - 2^x + 1, 2^{2y} - 2^y + 1)$ are unique Kasami-Welch exponents pairs in their respective cyclotomic cosets, then we

can assume, without loss of generality, that i, x are integers such that $0 < i, x < \frac{m}{2}$ (with $\frac{m}{2} < j, y < m$). Note that $(2^{2i} - 2^i + 1) * (2^{2x} - 2^x + 1) \equiv 2^a \pmod{2^m - 1} \rightarrow 2^{m-a}(2^{2i} - 2^i + 1) * (2^{2x} - 2^x + 1) \equiv 1 \pmod{2^m - 1}$ this implies that the Kasami-Welch exponents considered have a multiplicative inverse in $\mathbb{Z}_{2^m-1}$. We have two main cases, m even and m odd.

Case 1: m even

For m even, we have that $2^m \equiv 1 \pmod 3$, and as such, $2^m - 1 \equiv 0 \pmod 3$. On the other hand, since i, x must be relatively prime to m then they must be odd integers. However, for r an odd integer, $2^r \equiv 2 \pmod 3 \rightarrow 2^{2r} - 2^r + 1 \equiv 1 - 2 + 1 \pmod 3 \rightarrow 2^{2r} - 2^r + 1 \equiv 0 \pmod 3$. This means that $(2^{2i} - 2^i + 1, 2^m - 1) \geq 3, (2^{2x} - 2^x + 1, 2^m - 1) \geq 3 \rightarrow$ the exponents do not have a multiplicative inverse and thus $(2^{2i} - 2^i + 1) * (2^{2x} - 2^x + 1) \equiv 2^a \pmod{2^m - 1}$ cannot be true for any integer a.

Case 2: m odd

We note that if $m = 3$, there are only two Kasami-Welch exponents defined (for $i = 1, j = 2$). Thus, we cannot define such x for which the multiplicative inverse shares a cyclotomic coset with another Kasami-Welch exponent, but not the exponent itself. We have that $(2^{2i} - 2^i + 1) * (2^{2x} - 2^x + 1) \equiv 2^a \pmod{2^m - 1} \rightarrow 2^{2(x+i)} - 2^{2i+x} + 2^{2i} - 2^{2x+i} + 2^{x+i} - 2^i + 2^{2x} - 2^x + 1 \equiv 2^a \pmod{2^m - 1}$. From this equivalence, we deduce that the following equivalence has to be met for $(2^{2i} - 2^i + 1) * (2^{2x} - 2^x + 1) \equiv 2^a \pmod{2^m - 1}$ to be true:

$$2^{2(x+i)} + 2^{2x} + 2^{x+i} + 2^{2i} + 1 \equiv 2^a + 2^{2x+i} + 2^{2i+x} + 2^x + 2^i \pmod{2^m - 1} \qquad (41)$$

To show that this equivalence does not hold, we shall show that the right and left sides of the equivalence are equivalent to distinct integers $\pmod{2^m - 1}$. First, we show that all the terms on the left side of the equivalence are not equivalent to each other.

Left Side of Equivalence 41 Subcase 1: $2(x + i) \equiv 2x \pmod m$

$2(x + i) \equiv 2x \pmod m \rightarrow 2i \equiv 0 \pmod m$. This is a contradiction as $(i, m) = 1$ and thus is not a zero divisor in $\mathbb{Z}_m^*$.

Left Side of Equivalence 41 Subcase 2: $2(x + i) \equiv x + i \pmod m$

$2(x+i) \equiv x+i \pmod m \rightarrow x+i \equiv 0 \pmod m \rightarrow x \equiv -i \pmod m \rightarrow x \equiv j \pmod m$ which is a contradiction as it would mean that $2^{2x} - 2^x + 1$ and $2^{2i} + 2^i + 1$ share a cyclotomic coset which is not the case.

Left Side of Equivalence 41 Subcase 3: $2(x + i) \equiv 2i \pmod m$

$2(x+i) \equiv 2i \pmod{m} \rightarrow 2x \equiv 0 \pmod{m}$ which is a contradiction as $(x, m) = 1$ which means x is not a zero divisor in $\mathbb{Z}_m^*$.

Left Side of Equivalence 41 Subcase 4: $2(x + i) \equiv 0 \pmod{m}$

$2(x + i) \equiv 0 \pmod{m} \rightarrow 2x \equiv -2i \pmod{m} \rightarrow x \equiv j \pmod{m}$ which is a contradiction by the same argument as Subcase 2.

Left Side of Equivalence 41 Subcase 5: $2x \equiv x + i \pmod{m}$

$2x \equiv x + i \pmod{m} \rightarrow x \equiv i \pmod{m}$ which is a contradiction as both of these are distinct integers less than m.

Left Side of Equivalence 41 Subcase 6: $2x \equiv 2i \pmod{m}$

$2x \equiv 2i \pmod{m} \rightarrow x \equiv i$ which is a contradiction by the same argument as Subcase 5.

Left Side of Equivalence 41 Subcase 7: $2x \equiv 0 \pmod{m}$

$2x \equiv 0 \pmod{m}$ is a contradiction by the same argument as Subcase 3.

Left Side of Equivalence 41 Subcase 8: $x + i \equiv 2i \pmod{m}$

$x + i \equiv 2i \pmod{m} \rightarrow i \equiv x \pmod{m}$ which is a contradiction by the same argument as Subcase 5.

Left Side of Equivalence 41 Subcase 9: $x + i \equiv 0 \pmod{m}$

$x + i \equiv 0 \pmod{m} \rightarrow x \equiv -i \pmod{m} \rightarrow x \equiv j \pmod{m}$ which is a contradiction by the same argument as Subcase 2.

Left Side of Equivalence 41 Subcase 10: $2i \equiv 0 \pmod{m}$

$2i \equiv 0 \pmod{m}$ which is a contradiction by the same argument as Subcase 1.

Thus, all the terms in the left side of the equivalence are equivalent to distinct powers of 2 $\pmod{2^m - 1}$. If we substitute each term by their equivalence class representative then by Lemma 2, this means that they sum to 0 $\pmod{2^m - 1}$ if $m = 5$ and an odd integer less than $2^m - 1$ if $m > 5$, as it is the sum of five distinct powers of 2. Now, for the right side of the equivalence, we show that the last four terms are not equivalent $\pmod{2^m - 1}$ to each other. We do not compare 2^a to the other terms because the only restriction on a is that $0 \leq a < m$, so we cannot reach a contradiction via direct comparison. The cases follow:

Right Side of Equivalence 41 Subcase 1: $2x + i \equiv 2i + x \pmod{m}$

$2x + i \equiv 2i + x \pmod{m} \rightarrow x \equiv i \pmod{m}$ which is a contradiction as x, i are distinct integers less than m.

Right Side of Equivalence 41 Subcase 2: $2x + i \equiv x \pmod{m}$

$2x + i \equiv x \pmod{m} \rightarrow x + i \equiv 0 \pmod{m} \rightarrow x \equiv -i \pmod{m} \rightarrow x \equiv j \pmod{m}$ which is a contradiction as it would mean that $2^{2x} - 2^x + 1$ and $2^{2i} + 2^i + 1$ share a cyclotomic coset which is not the case.

Right Side of Equivalence 41 Subcase 3: $2x + i \equiv i \pmod{m}$

$2x + i \equiv i \pmod{m} \rightarrow 2x \equiv 0 \pmod{m}$ which is a contradiction as $(x, m) = 1$ and thus x cannot be a zero divisor in $\mathbb{Z}_m^*$.

Right Side of Equivalence 41 Subcase 4: $2i + x \equiv x \pmod{m}$

$2i + x \equiv x \pmod{m} \rightarrow 2i \equiv 0 \pmod{m}$ which is a contradiction as $(i, m) = 1$ and thus i is not a zero divisor in $\mathbb{Z}_m^*$.

Right Side of Equivalence 41 Subcase 5: $2i + x \equiv i \pmod{m}$

$2i + x \equiv i \pmod{m} \rightarrow i + x \equiv 0 \pmod{m} \rightarrow x \equiv -i \pmod{m} \rightarrow x \equiv j \pmod{m}$ which is a contradiction by the same argument as Subcase 2.

Right Side of Equivalence 41 Subcase 6: $2i + x \equiv i \pmod{m}$

$x \equiv i \pmod{m}$ is a contradiction by the same argument as subcase 1.

We have showed that the last four terms of the right side of Equivalence 41 are not equivalent to each other $\pmod{2^m - 1}$. For $m = 5$, this means that if we take a to be the remaining integer that is less than m, then both sides of the equivalence will be sums of five non-equivalent powers of 2 $\pmod{2^m - 1}$, which, by Lemma 2, satisfies the equivalence. Thus, we consider $m > 5$. For $m > 5$, the possibility that the two sums differ in at least one term exists, and thus by Lemma 3, they would be two distinct 2-adic representations of the same integer, which is a contradiction. We can also show that $a \neq 0$, as otherwise, Equivalence 41 transforms into $2^{2(x+i)} + 2^{2x} + 2^{x+i} + 2^{2i} + 1 \equiv 1 + 2^{2x+i} + 2^{2i+x} + 2^x + 2^i \pmod{2^m - 1}$ which gives us:

$$2^{2(x+i)} + 2^{2x} + 2^{x+i} + 2^{2i} \equiv 2^{2x+i} + 2^{2i+x} + 2^x + 2^i \pmod{2^m - 1}. \tag{42}$$

This is the sum of non-equivalent powers of 2 on the left and right sides of the equivalence. None of the elements in the left side of the equivalence are equivalent to $2^m \pmod{2^m -}$

1), and thus their sum is equivalent $\pmod{2^m-1}$ to some even integer less than 2^m. For the right side of the equivalence, we have two possibilities:

1) If one of the terms is equivalent to $2^m \pmod{2^m-1}$, then no other term is (as they are all non-equivalent to each other). Thus, the right side of the equivalence will be the sum of a term equivalent to $1 \pmod{2^m-1}$ and three other terms that are equivalent $\pmod{2^m-1}$ to distinct powers of 2 less than 2^m. As such, their sum will be equivalent $\pmod{2^m-1}$ to some odd integer less than 2^m-1. Then, Equivalence 42 cannot be true.

2) If none of the terms on the right are equivalent to $2^m \pmod{2^m-1}$, then it suffices to show that the two sums differ in at least one term, and thus after substituting the equivalence class representatives of each term we can reach a contradiction by Lemma 3.

Subcase of Equivalence 42 2.1: $2x+i \equiv 2(x+i) \pmod{m}$

$2x+i \equiv 2(x+i) \pmod{m} \rightarrow 0 \equiv i \pmod{m}$ which is a contradiction as $0 < i < m$.

Subcase of Equivalence 42 2.2: $2x+i \equiv 2x \pmod{m}$

$2x+i \equiv 2x \pmod{m} \rightarrow i \equiv 0 \pmod{m}$ which is a contradiction by the same argument as Subcase 2.1.

Subcase of Equivalence 42 2.3: $2x+i \equiv x+i \pmod{m}$

$2x+i \equiv x+i \pmod{m} \rightarrow x \equiv 0 \pmod{m}$ which is a contradiction as $0 < x < m$.

Subcase of Equivalence 42 2.4: $2x+i \equiv 2i \pmod{m}$

$2x+i \equiv 2i \pmod{m} \rightarrow 2x \equiv i \pmod{m}$ which is the only possibility.

Subcase of Equivalence 42 2.5: $2i+x \equiv 2(x+i) \pmod{m}$

$2i+x \equiv 2(x+i) \pmod{m} \rightarrow 0 \equiv x \pmod{m}$ which is by the same argument as Subcase 2.3.

Subcase of Equivalence 42 2.6: $2i+x \equiv 2x \pmod{m}$

$2i+x \equiv 2x \pmod{m} \rightarrow 2i \equiv x \pmod{m}$ which is a possibility.

Subcase of Equivalence 42 2.7: $2i+x \equiv x+i \pmod{m}$

$2i+x \equiv x+i \pmod{m} \rightarrow i \equiv 0 \pmod{m}$ which is a contradiction as $0 < i < m$.

Subcase of Equivalence 42 2.8: $2i + x \equiv 2i \pmod{m}$

$2i + x \equiv 2i \pmod{m} \rightarrow 0 \equiv x \pmod{m}$ which is a contradiction by the same argument as Subcase 2.3.

Subcase of Equivalence 42 2.9: $x \equiv 2(x + i) \pmod{m}$

$x \equiv 2(x+i) \pmod{m} \rightarrow 0 \equiv x+2i \pmod{m} \rightarrow x \equiv -2i \pmod{m} \rightarrow x \equiv 2j \pmod{m}$ which is a possibility.

Subcase of Equivalence 42 2.10: $x \equiv 2x \pmod{m}$

$x \equiv 2x \pmod{m} \rightarrow 0 \equiv x \pmod{m}$ which is a contradiction by the same argument as Subcase 2.3.

Subcase of Equivalence 42 2.11: $x \equiv x + i \pmod{m}$

$x \equiv x + i \pmod{m} \rightarrow i \equiv 0 \pmod{m}$ which is a contradiction by the same argument as subcase 2.1.

Subcase of Equivalence 42 2.12: $x \equiv 2i \pmod{m}$

$x \equiv 2i \pmod{m} \rightarrow 0 \equiv x \pmod{m}$ which is a possibility.

Subcase of Equivalence 42 2.13: $i \equiv 2(x + i) \pmod{m}$

$i \equiv 2(x+i) \pmod{m} \rightarrow 0 \equiv 2x+i \pmod{m} \rightarrow 2x \equiv -i \pmod{m} \rightarrow 2x \equiv j \pmod{m}$ which is a possibility.

Subcase of Equivalence 42 2.14: $i \equiv 2x \pmod{m}$

$i \equiv 2x \pmod{m}$ is a possibility.

Subcase of Equivalence 42 2.15: $i \equiv x + i \pmod{m}$

$i \equiv x + i \pmod{m} \rightarrow 0 \equiv x \pmod{m}$ which is a contradiction by the same argument as Subcase 2.3.

Subcase of Equivalence 42 2.16: $i \equiv 2i \pmod{m}$

$i \equiv 2i \pmod{m} \rightarrow 0 \equiv i \pmod{m}$ which is a contradiction by the same argument as Subcase 2.1.

If the two sums in Equivalence 42 do not differ in any term, then each term in the right side of the equivalence must correspond to a unique term on the left side of the equivalence, with no repetitions (as any two terms on the right are not equivalent $\pmod{2^m-1}$ to each other). 2^{2x+i} could only possibly be equivalent to 2^{2i} and 2^{2i+x} could only be equivalent to 2^{2x}. On the other hand, 2^x could be equivalent to $2^{2(x+i)}$ or 2^{2i}, and 2^i could be equivalent to $2^{2(x+i)}$ or 2^{2x}. Note that since 2^{2x+i} could only possibly be equivalent to 2^{2i} and 2^{2i+x} could only be equivalent to 2^{2x} then it forces both $2^i, 2^x$ to be equivalent to $2^{2(x+i)} \pmod{2^m-1}$, which is a contradiction as $2^x \not\equiv 2^i \pmod{2^m-1}$. We conclude that $a \neq 0$ as Equivalence 42 cannot be true.

We now show that the last four terms of the right side of Equivalence 41 are all not equivalent to $2^m \pmod{2^m-1}$.

Right Side of Equivalence 41 Subcase 7: $2x+i \equiv 0 \pmod{m}$

$2x+i \equiv 0 \pmod{2^m-1} \to 2x \equiv -i \pmod{m} \to 2x \equiv j \pmod{m}$. This transforms Equivalence 41 into: $2^{j+2i}+2^{2x}+2^{x+i}+2^{2i}+1 \equiv 2^a+2^{j+i}+2^{2i+x}+2^x+2^i \pmod{2^m-1} \to 2^{m+i}+2^{2x}+2^{x+i}+2^{2i}+1 \equiv 2^a+2^m+2^{2i+x}+2^x+2^i \pmod{2^m-1} \to 2^i+2^{2x}+2^{x+i}+2^{2i}+1 \equiv 2^a+1+2^{2i+x}+2^x+2^i \pmod{2^m-1}$. This gives us the equivalence

$$\to 2^{2x}+2^{x+i}+2^{2i} \equiv 2^a+2^{2i+x}+2^x \pmod{2^m-1} \tag{43}$$

We have proven already that all the terms on the left side are non-equivalent to each other, and the last two terms of the right side are non-equivalent to each other. Furthermore, none of the terms on the left are equivalent to $2^m \pmod{2^m-1}$, while for the right side, $0 < a < m$, $x \not\equiv 0 \pmod{m}$ and $2i+x \equiv 0 \pmod{m} \to x \equiv -2i \pmod{m} \to x \equiv 2j \pmod{m}$. This would mean that $2x-x \equiv j-2j \pmod{m} \to x \equiv -j \pmod{m} \to x \equiv i \pmod{m} \to x = i$ which is a contradiction. We show that these two sums differ in at least one term, and thus by Lemma 3 cannot be equivalent $\pmod{2^m-1}$.

Equivalence 43 Subcase 1: $x \equiv 2x \pmod{m}$

$x \equiv 2x \pmod{m} \to x \equiv 0 \pmod{m}$ which is a contradiction as $0 < x < m$.

Equivalence 43 Subcase 2: $x \equiv x+i \pmod{m}$
$x \equiv x+i \pmod{m} \to i \equiv 0 \pmod{m}$ which is a contradiction as $0 < i < m$.

Equivalence 43 Subcase 3: $x \equiv 2i \pmod{m}$

If $x \equiv 2i \pmod{m}$ then $x+4x \equiv 2i+2j \pmod{m} \to 5x \equiv 2m \pmod{m} \to 5x \equiv 0 \pmod{m}$ which is false unless $m = 5$ as $(x, m) = 1$.

There is the possibility that $a = m-1$, and 2^a equivalent $\pmod{2^m-1}$ to one of the terms on the right, but in such case, the sum of both 2^{m-1} terms is equivalent to $2^m \pmod{2^m-1}$, in which case the sum of the terms on the right would be equivalent to some

odd integer $\pmod{2^m-1}$ less than 2^m while the sum on the left would be equivalent to some even integer $\pmod{2^m-1}$ less than 2^m (as all these terms are not equivalent to 2^m or each other $\pmod{2^m-1}$). Thus, these two sums cannot be equivalent $\pmod{2^m-1}$ to each other.

Right Side of Equivalence 41 Subcase 8: $2i+x \equiv 0 \pmod{m}$

Since the exponent distribution of the powers of 2 is symmetric on x and i, proving that $2i+x \equiv 0 \pmod{2^m-1}$ is a contradiction also proves that $2i+x \equiv 0 \pmod{m}$ is a contradiction. That is, it is the same argument as Subcase 7 except inverting x with i, and y with j.

Right Side of Equivalence 41 Subcase 9 : $x \equiv 0 \pmod{m}$

$\rightarrow x = 0$ which is a contradiction as $0 < x < m$.

Right Side of Equivalence 41 Subcase 10 : $i \equiv 0 \pmod{m}$

$\rightarrow i = 0$ which is a contradiction as $0 < i < m$.

There is one last subcase to consider, can $a = m-1$ be equivalent $\pmod{2^m-1}$ to another term on the right and thus their sum be equivalent to $2^m \pmod{2^m-1}$?

Right Side of Equivalence 41 Subcase 11 : $a \equiv m-1 \pmod{m}$ **and** $a \equiv 2x+i \pmod{2^m-1}$

We have that Equivalence 41 transforms into $2^{2(x+i)} + 2^{2x} + 2^{x+i} + 2^{2i} + 1 \equiv 2^{m-1} + 2^{m-1} + 2^{2i+x} + 2^x + 2^i \pmod{2^m-1} \rightarrow 2^{2(x+i)} + 2^{2x} + 2^{x+i} + 2^{2i} + 1 \equiv 2^m + 2^{2i+x} + 2^x + 2^i \pmod{2^m-1} \rightarrow 2^{2(x+i)} + 2^{2x} + 2^{x+i} + 2^{2i} + 1 \equiv 1 + 2^{2i+x} + 2^x + 2^i \pmod{2^m-1} \rightarrow 2^{2(x+i)} + 2^{2x} + 2^{x+i} + 2^{2i} \equiv 2^{2i+x} + 2^x + 2^i \pmod{2^m-1}$. The terms on the left are not equivalent $\pmod{2^m-1}$ to each other, and neither are the terms on the right side of the equivalence. We substitute the equivalence class representatives and change the equivalence to an equality:

$$2^a + 2^b + 2^c + 2^d = 2^{a_2} + 2^{b_2} + 2^{c_2}$$

With $0 < a, b, c, d, a_2, b_2, c_2 < m$, a, b, c, d all distinct from each other and a_2, b_2, c_2 all distinct from each other. This is a contradiction by Lemma 3 as it would mean that an integer has two different 2-adic representations. This same contradiction is reached if we had assumed that $a = m-1$ was equivalent to any of the remaining three terms on the right.

With these results, we go back to Equivalence 41. We have shown that all the terms on the left side of the equivalence are not equivalent $\pmod{2^m-1}$ to each other, and that the last four terms of the right side of the equivalence are not equivalent $\pmod{2^m-1}$ to each other or 2^m and that $0 < a < m-1$. We can substitute each of the terms of the

left side for their equivalence class representatives, which sum to an odd integer less than $2^m - 1$. On the right side, if we substitute each of the terms for their equivalence class representatives, we know that the last four terms are distinct and thus sum to an even integer. Since $0 < a < m-1$ then the right side will sum to an even integer less than $2^m - 1$. Hence, we have equality between an odd integer on the left and an even integer on the right, which is a contradiction. Thus, Equivalence 41 does not hold for $m > 5$ and there does not exist an integer $0 < a < m$ such that $(2^{2i} - 2^i + 1) * (2^{2x} - 2^x + 1) \equiv 2^a \pmod{2^m - 1}$.

□

Finally, we use Theorem 11 to state and prove the cyclotomic equivalence result for Kasami-Welch Boolean functions:

Theorem 12 (Cyclotomic-Equivalent Kasami-Welch Functions)**.** Let $f(x) = x^{2^{2i}-2^i+1}, g(x) = x^{2^{2x}-2^x+1}$ be vectorial Kasami-Welch Boolean functions $\mathbb{F}_{2^m} \to \mathbb{F}_{2^m}, m > 5$. If $i \neq x, 0 < i, x < m$ and $(i, m) = (x, m) = 1$, then, these functions are cyclotomic-equivalent if and only if $i + x = m$.

Proof: For the forwards direction, if the functions are cyclotomic-equivalent, then it means that either $(2^{2x} - 2^x + 1)(2^{2i} - 2^i + 1) \equiv 2^a \pmod{2^m - 1}$ for some integer $0 \leq a < m$ or the exponents are in the same cyclotomic coset. We have proven in Theorem 11 that no such a can exist for $m > 5$. The exponents must then share a cyclotomic coset which is only possible if $i + x = m$.

For the backwards direction, if $i+x = m$, then $2^{2i}-2^i+1, 2^{2x}-2^x+1$ share a cyclotomic coset as per Theorem 6 and Theorem 7. Thus, the functions are cyclotomic-equivalent by definition. Thus, we have our result.

□

From these results, we state a theorem on the number of non CCZ-equivalent Kasami-Welch functions.

Theorem 13 (Number of non CCZ-Equivalent Kasami-Welch Functions)**.** Let $f(x) = x^{2^{2l}-2^l+1}$ be the vectorial Kasami-Welch Boolean function $\mathbb{F}_{2^m} \to \mathbb{F}_{2^m}, m > 5, (l, m) = 1$. Then there are $\frac{\Phi(m)}{2}$ non CCZ-equivalent Kasami-Welch functions

Proof: From Theorem 12, we know that two Kasami-Welch functions $f(x) = x^{2^{2i}-2^i+1}, g(x) = x^{2^{2l}-2^l+1}, i \neq l, (i, m) = (l, m) = 1$ over $\mathbb{F}_{2^m}$ are cyclotomic-equivalent (and hence CCZ-equivalent) if and only if $i + l = m$. This only happens whenever the two exponents are in the same cyclotomic coset $\pmod{2^m - 1}$. From Theorems 6 and 7 we know that the exponents occur in these unique pairs in the cyclotomic coset. The total number of integers $0 < l < m$ such that $(l, m) = 1$ is given by $\Phi(m)$. Since they occur in pairs in the cyclotomic cosets, then the total number of non CCZ-equivalent Kasami-Welch functions will be $\frac{\Phi(m)}{2}$.

□

5. Improvement on Dillon and Dobbertin's Theorem on the Construction of Bent Gold and Kasami-Welch Functions

5.1. **Gold Bent Function Construction.** In this subsection, we consider the Boolean functions in m variables of the form $Tr(\alpha^i x^d)$. This subsection references details of the results from our Springer PROMS article [67]. In Algorithm 7, we define a list containing the Gold exponents of the form 2^l+1 for $0 < l < m$ and iterate over all the exponents in the list. For $\alpha \in \mathbb{F}_{2^m}^*$, a primitive element, we iterate over $0 \leq i < 2^m - 1$ (covering all nonzero elements in the field). We utilize SAGE online programming software to construct algorithms that generate these functions. The main components of the The three main algorithms are as follows a) select an irreducible polynomial of degree m to construct the finite field and polynomial ring, b) verify if a Boolean function is bent; c) constructs these functions, d) and verifies bent-ness. The irreducible polynomials are the Conway polynomials as described in [58]. This step is carried out to ensure that every time we construct the field, we use the same modulus. The construction of the univariate Gold bent functions is known. Dillon in 1974 deduced that the Gold function $Tr(\lambda x^{2^l-1}), x \in F_{2^m}, \lambda \in F_{2^m}^*$ is bent if and only if $Tr(\lambda x^{2^l-1})$ was restricted to U (the multiplicative subgroup U of $F_{2^m}^*$) has Hamming weight $2^l - 1$. The following construction is given in [15]:

The function $Tr(\lambda x^{2^j+1})$ where $\lambda \in F_{2^m} \backslash \{x^{2^j+1}; x \in F_{2^m}\}, \frac{m}{(j,m)} =$ even, is bent

Dillon and Dobbertin in [22] state proposition 2 for the conditions under which Gold Boolean functions are bent. In this section, we formulate a conjecture of a different construction by analyzing computational results from functions obtained via algorithms we developed via SAGE. Our results show that the (l,m) = 1 criteria and conditions over λ can be changed to produce Gold bent functions.

5.1.1. *Algorithms.* We refer to the SAGE Boolean function page for Boolean function commands [49]. Three algorithms are constructed [67]:

Algorithm 5. [67] `def FindModNonCube(`m`):`

```
L =[]
P.<a> = PolynomialRing(GF(2))
k.<a> = GF(2**m, modulus = 'conway')
r = k.modulus()
L.append(r)
return L
```

This algorithm constructs a polynomial ring over $\mathbb{F}_2$, uses the Conway polynomial method to construct a finite field $\mathbb{F}_{2^m}$, and stores the modulus on a list L. The algorithm returns the modulus, which is used to construct polynomial rings and finite fields in other algorithms.

Algorithm 6. [67]

```
def Bentness(f):
  dim = f.nvariables()
  w = f.walsh_hadamard_transform()
  for i in range(2^dim):
```

```
  if abs(w[i]) != 2^(dim/2)
    return ‘‘Not Bent’’
  else:
    i = pass
return ‘‘Bent’’
```

This algorithm takes as input a Boolean function “f,” finds in how many variables this function is defined, and determines its Walsh-Hadamard transform spectrum. There are many ways to input a Boolean function, we use the "BooleanFunction(f(x))" command, where $f(x)$ is a polynomial over $\mathbb{F}_{2^m}$ and the result is the Boolean function given by $Tr(f(x))$ [49]. Then it verifies if the function is bent by iterating over the spectrum to check if it meets Definition 1. The performance of this algorithm is comparable to the inbuilt “is_bent()” command in sage. We compare these in Tables 2 and 3.

Variables	Bent Functions Detected	Time (s)	Max Memory Allocation
4	10	0.108	27703
6	42	1.864	68556
8	170	33.754	125285
10	682	706.255	175985

Table 2: Computation time and memory allocation of is_bent() command.

Variables	Bent Functions Detected	Time (s)	Max Memory Allocation
4	10	0.106	25090
6	42	1.843	44221
8	170	35.895	124533
10	682	710.432	158697

Table 3: Computation time and memory allocation of Bentness algorithm.

We compared these commands for up to 10 variables. We utilized Python memory allocation and time tracking packages. Boolean functions of the form $Tr(\alpha^i x^3)$ iterating for $0 \leq i \leq 2^m - 1$ were computed. We observe that the computational time was similar for both methods, while the maximum memory allocation was slightly lower for the Bentness algorithm.

Algorithm 7. [67]

```
def BentTraceIterationGold(m):
  f = FindModNonCube(m)
  R.<x> = GF(2^m,’a’, modulus = f)[]
  k.<a> = GF(2**m, modulus = f)
  print("Variables, Exponent, Power of the element, Is it Bent?")
  for j in range(1,m):
```

```
for i in range(0,2^m - 1):
  GB = BooleanFunction((a^i)*x^(G[j]))
  print(m,",", G[j], ", α^",i," ,",Bent-ness(GB))
```

The algorithm takes as input the number of variables "m" and then constructs a Boolean polynomial Ring over $\mathbb{F}_{2^m}$ such that it assigns a as the congruence class $[x]$ modulo f (the irreducible polynomial from Algorithm 5). Then it constructs a finite field as powers of "a" where a is a primitive element in the field. Next, it iterates over the possible Gold exponents (which are stored in a list "G") and all the power of "a" to go over all the possible Gold functions. Finally, it verifies if the function is bent and prints the results.

5.1.2. *Explicit Families of Gold Bent Functions Not Obtained by Dillon and Dobbertin:* We observed a pattern in the generation of the bent and non-bent functions. If $Tr(\alpha^i x^d)$ was non-bent for i some multiple of an integer k, then it was not bent for all the multiples of k. Furthermore, we noted that k took on values as divisors of the Gold exponent under consideration. We denoted these exceptions by "Mk" meaning "multiples of k." Tables with all the functions were computed for up to 10 variables. We established some observations based on the resulting tables and then obtained results for 12-18 variables only considering powers of i that correspond to multiples of divisors of d. We had the following observations for up to 14 variables:

1) If (l,m) = 1, then the exceptions are the multiples of 3.
2) Let $i, i_2, t, t_2, j \in \mathbb{Z}$, $m \neq 2^j$, and say $m = 2^i * t$, then if $l = 2^{i_2} * t_2$ for $i \leq i_2$, then all cases are exceptions.
3) For all other cases, the exceptions are multiples of some divisor of d. Special case when m = 2 x l. Then the exceptions are multiples of d.

We list the obtained tables in section 8 (see Tables 4-5) noting for what multiples of k the function was not bent. We computed partial results for up to 24 variables (see Table 6) and verified that the deduced conditions apply to these cases. Computational limitations resulted in Table 6 not being completed for 20-24 variables. However, the pattern is still present in the observed results. Note that for 24 variables, when $m = 2 * l, 4 * l, 6 * l, 8 * l$, and $12 * l$, the exceptions are Md. Interestingly, $m = 3 * l$ does not lead to this result as it meets the second case (perhaps suggesting priority for that condition). For 22 variables, when $m = 2 * l$ the exceptions are for Md but for $m = 11 * l$ we see that case 2 applies and we get $M1$ as the exceptions. For 20 variables when $m = 2 * l, 4 * l, 10 * l$ the exceptions are for Md but for $m = 5 * l$ we see that case 2 applies and we get $M1$ as the exceptions. This pattern is observed for 16 and 18 variables. For the remaining cases, we note that the exceptions coincide with the smallest Gold exponent in the set of divisors of d. The exception to this is the $M9$ cases that fall under case 4 as if 9 divides the Gold exponent, so does 3. It seems that $M3$ cases only occur under condition 1) , so perhaps this could be an if and only if condition, although we do not have strong enough evidence to state this yet. These observations lead to the final set of conditions deduced from the computational results:

1) If (l,m) = 1, then the functions are bent whenever $i \notin M3$.

2) Let $i, i_2, t, t_2, j \in \mathbb{Z}^+$, $m \neq 2^j$, and say $m = 2^i * t$, then if $l = 2^{i_2} * t_2$ for $i \leq i_2$, then all cases are exceptions.

3) Given that case 2 is not met, then if $m = t * l$ for an integer t , then the functions are bent whenever $i \notin Md$.

4) If the previous cases are not met, then the functions are bent for $i \notin Mk$, where k is the smallest Gold exponent such that $k|d$, k > 3.

Based on these conditions, we state the following conjecture on the Gold Boolean bent functions:

5.1.3. *Conjectures For The Construction of Boolean Bent Gold Functions.*

Conjecture 1 (Velazquez, Janwa [67] Constructed Families of Gold Boolean Bent Functions)**.** Consider $m, l, i, k \in \mathbb{Z}$ and the finite field $\mathbb{F}_{2^m}$, where m is an even integer. Let $d = 2^l + 1$ be a Gold exponent, and $\alpha \in \mathbb{F}_{2^m}$ a primitive element in the field. Let $Tr(\alpha^i x^d)$ be a Gold Boolean function and Mk the set of all the multiple of k. Then:

1) If (l,m) = 1, $Tr(\alpha^i x^d)$ is bent if $i \notin M3$.

2) Let $i, i_2, t, t_2, j \in \mathbb{Z}^+$, $m \neq 2^j$, and say $m = 2^i * t$, then if $l = 2^{i_2} * t_2$ for $i \leq i_2$ then $Tr(\alpha^i x^d)$ is not bent.

3) Given that case 2 is not met, then if $m = t * l$ for t $\in \mathbb{Z}^+$, then $Tr(\alpha^i x^d)$ is bent if $i \notin Md$.

4) If the previous cases are not met, then $Tr(\alpha^i x^d)$ is bent if $i \notin Mk$ for k the smallest Gold exponent such that $k|d$, k > 3.

The first case follows from Dillon and Dobbertin's results.

5.2. **Further Results and Conjectures for the Gold Case.**

5.2.1. *Algorithms.* We continued our research on Gold Boolean bent functions by computationally testing conditions for equality. We develop an algorithm to generate the Gold Boolean functions in m (even) variables of the form $f(x) = Tr(\alpha^i x^{2^l+1})$. We compare these functions in four variables to observe patterns to state and prove our following theorems.

Algorithm 8. `def TestEquality(`m`):`

```
f = FindModNonCube(m)
R.< x > = GF(2^m,'a', modulus = f)[]
k.< a > = GF(2**m, modulus = f)
BANF = []
BF = []
BentL = []
c = 0
for i in range(1,m):
  for j in range(1,2^m - 1):
    BG = BooleanFunction(a^j * x^(2^i + 1))
    BentL.append(Bentness(BG))
```

```
    BF.append("Tr(a^{" + str(j) + "}x^{" + str(2^i + 1) + "})")
    BANF.append(BG.algebraic_normal_form())
for i in range(1,m/2 + 1):
  for j in range(1,2^m - 1):
    BG = BooleanFunction(a^j * x^(2^i + 1))
    ANF = BG.algebraic_normal_form()
    BF2 = ("Tr(a^{" + str(j) + "}x^{" + str(2^i + 1) + "})")
    c2 = [i for i, n in enumerate(BANF) if n == ANF]
    n = len(c2)
    print(str(BF[c]), end = "")
    c3 = c2
    c3.remove(BF.index(BF2))
    for i2 in c3:
      print(" = ", BF[i2], end = "")
    print(" = " + str(n), " = ", BentL[c])
    c = c + 1
```

The algorithm takes as inputs the number of variables considered. Then, it utilizes Algorithm 5 to find an irreducible polynomial of degree m. This polynomial (stored in the variable f) is used as the modulus of the Boolean polynomial Ring over $\mathbb{F}_{2^m}$ such that it assigns a as the congruence class $[x]$ modulo f and to construct a finite field as powers of "a" where a is a primitive element in the field. Then, three empty lists are created (BANF,BF and BentL) to store the algebraic normal forms, the string representations, and the bent-ness property of the functions, respectively. These lists are created such that the algebraic normal form in the position "i" on the list BANF corresponds to the string representation of the functions on position "i" of the list BF, and the same for BentL. A counting variable is defined to count the number of functions that are compared. Then, two for loops are used to iterate over the constructed functions' exponents of α and x. The first double loop is used to generate all the functions of the form $Tr(\alpha^i x^{2^j+1})$. The string representation of this function, and its algebraic normal and bent-ness form are stored in their respective lists. The bent property is determined via an algorithm we constructed in. Then, we iterate over the exponents again, going over all exponents of α but only up to $x^{2^{\frac{m}{2}}+1}$. This is to avoid repeating the comparison between functions. The functions are generated, their algebraic normal form found, and their string representation as a trace assigned to the variable "BF2." Next, a list of indices of the functions that share the same algebraic normal form as the current generated function is assigned to "c2" and its length is determined. The second list of indices is constructed so that the current function is removed from the list to avoid repeated results. Then, the algorithm outputs the current function and all the functions that share the same algebraic normal form (and thus are equal). It also outputs the number of such functions and whether they are bent or not. The results of the algorithm can be observed in Table 13.

Algorithm 9. `def CycloCosTestBentIteration(`m`):`

```
    f = FindModnoncube
    R.< x > = GF(2^m,'a', modulus = f)[]
```

```
k.< a > = GF(2**m, modulus = f)
l = cyclocosets(m)
print("Variables - CycloRep - Exponent - Exception - Function")
for i in l:
  for e in list(set(i)):
    st = []
    w = 0
    for y in range(2^m - 1):
      GB = BooleanFunction((a^y)*(x^e))
      B = Bentness(GB)
      if B == "Bent":
        if y == o:
          break
          pass
      else:
        w = w + 1
        st.append(y)
        if w == 3:
          st.remove(0)
          print(m, "-",CycloRep(e, 2 ^m - 1), "-", e, "- M" + str(gcd(st)),
" - Tr(α^{ i } x ^ {" + str(e) + ""}))
```

This algorithm tests if Boolean functions of the form $Tr(\alpha^i x^{C(d)})$ meet the same conditions over the value of i to be bent. The input is the number of variables considered and the first three lines find a monic irreducible polynomial in m variables to be used as the modulus for the finite field and polynomial ring constructed ($k. < a >$ and $R. < x >$). Then it assigns to "l" a list of cyclotomic cosets $\pmod{2^m - 1}$. Next, the algorithm iterates over the list of cyclotomic cosets and the elements inside them. It defines a list "st" that stores all the exponents of x that lead to non-bent functions for all values of i and a counter variable "w" that determines when we find the third non-bent function. Then, it iterates over all the possible values of i, defines a Boolean function, and tests if it is bent. If it is bent, it checks if $\alpha = 0$. If it is, then we skip over to the next cyclotomic coset. This is because the case $\mathrm{Tr}(x^d)$ is a special case we do not consider in this work. If it is not bent, we add the exponent to the list "st" and a one to the counter variable. If it is the third non-bent function found (counter = 3), then we print the results, with the exceptions being Mk where k is the greatest common divisor of the values in st.

5.2.2. *Results for Gold Boolean Functions.* The basis of this analysis is to identify a one to one correspondence between functions in m variables of the form $Tr(\alpha^a x^{2^i+1})$ and $Tr(\alpha^b x^{2^j+1})$. In Section 4, for the near-bent case, we had identified the condition that the Boolean functions in m variables $Tr(x^{2^i+1})$, $Tr(x^{2^j+1})$ and $Tr(x^{2^{2i}-2^i+1})$, $Tr(x^{2^{2j}-2^j+1})$ were equal if $j + i = m$. The proof of this is deduced from our cyclotomic coset analysis for these exponents. From the cyclic property of the definition of the trace, we know that $Tr(x) = Tr(x^{2^j})$ for some integer j. The presence of the α term should make the previous analysis not possible. We analyze the four and six variable case to observe if there are

equivalences between $Tr(\alpha^i x^d)$ and $Tr(\alpha^i x^{2^j d})$ (where d is the Gold exponent). First, we note that:

$$\mathrm{Tr}(\alpha^i x^d) = \alpha^i x^d + \alpha^{2i} x^{2d} + \alpha^{4i} x^{4d} + \ldots\ \alpha^{2^m-1 i} x^{2^m-1 d}$$
$$\mathrm{Tr}(\alpha^i x^{2d}) = \alpha^i x^{2d} + \alpha^{2i} x^{4d} + \alpha^{4i} x^{8d} + \ldots\ \alpha^{2^m-1 i} x^{d}$$

It shifts the powers of x, however, it does not indicate anything about equality or equivalence. To determine equality, we utilized Algorithm 9 to test and observe results in Table 14. We observe that for d in a particular cyclotomic coset, the conditions for bent-ness are the same. For the equality analysis, we did not find any equal functions by taking d in the same cyclotomic coset. For the Gold case in particular, we know from Theorem 5 that two Gold exponents $2^i+1, 2^j+1$, are in the same cyclotomic coset $\pmod{2^m-1}$ given that $i+j=m$. These results seem to indicate that this property is crucial for preserving the bent-ness property of these functions and potentially for equality of these functions. We observe that the Gold Boolean functions shared the same conditions on the exponent of α for them to be bent given that $i+j=m$. This is seen in Tables 4,5, 6, where we note that the values of l for each case can be paired to sum to m. However, the functions $Tr(\alpha^a x^d)$ and $Tr(\alpha^a x^{2^k(d)})$ are not necessarily equal, as for example, in four variables $Tr(\alpha^1 x^3) \neq Tr(\alpha^1 x^6)$.

Next, we focus on the case where we cycle through the exponents of both α and x. For this analysis, we apply Definition 5 to the Gold Boolean functions in m variables:

$$Tr(\alpha^a x^{2^i+1}) = (\alpha^a x^{2^i+1}) + (\alpha^a x^{2^i+1})^2 + (\alpha^a x^{2^i+1})^4 + \cdots + (\alpha^a x^{2^i+1})^{2^{m-1}} \tag{44}$$

$$= (\alpha^a x^{2^i+1}) + (\alpha^{2a} x^{2(2^i+1)}) + (\alpha^{4a} x^{4(2^i+1)}) + \cdots + (\alpha^{2^{m-1}a} x^{2^{m-1}(2^i+1)}) \tag{45}$$

$$= (\alpha^a x^{2^i+1}) + (\alpha^{2a} x^{(2^{i+1}+2)}) + (\alpha^{4a} x^{(2^{i+2}+4)}) + \cdots + (\alpha^{2^{m-1}a} x^{(2^{i+m-1}+2^{m-1})}) \tag{46}$$

We note that the exponents of α and x cycle through elements in a cyclotomic coset $\pmod{2^m-1}$. That is, the exponents of α and x in the trace sum elements will be $C(a)$ and $C(x)$ respectively. In fact, note that, for integers i, k, e, q, $0 < e, q < m, e+q = m$, say $i = mk+e$, then:

$$Tr((\alpha^a x^{2^i+1})^{2^i}) = (\alpha^a x^{2^i+1})^{2^i} + (\alpha^a x^{2^i+1})^{2^{i+1}} + (\alpha^a x^{2^i+1})^{2^{i+2}} + \cdots + (\alpha^a x^{2^i+1})^{2^{i+q}} + \cdots +$$
$$(\alpha^a x^{2^i+1})^{2^{i+m-1}} = (\alpha^a x^{2^i+1})^{2^{mk+e}} + (\alpha^a x^{2^i+1})^{2^{mk+e+1}} + (\alpha^a x^{2^i+1})^{2^{mk+e+2}} + \cdots +$$
$$(\alpha^a x^{2^i+1})^{2^{mk+e+q}} + \cdots + (\alpha^a x^{2^i+1})^{2^{mk+e+m-1}}.$$

Since $\alpha, x \in \mathbb{F}_{2^m}$, their exponents can be taken $\pmod{2^m-1}$, which gives us:

$$Tr((\alpha^a x^{2^i+1})^{2^i}) = (\alpha^a x^{2^i+1})^{2^{mk+e}} + (\alpha^a x^{2^i+1})^{2^{mk+e+1}} + (\alpha^a x^{2^i+1})^{2^{mk+e+2}} + \cdots +$$
$$(\alpha^a x^{2^i+1})^{2^{mk+e+q}} + \cdots + (\alpha^a x^{2^i+1})^{2^{mk+e+m-1}} \rightarrow Tr((\alpha^a x^{2^i+1})^{2^i}) =$$
$$(\alpha^a x^{2^i+1})^{2^e} + (\alpha^a x^{2^i+1})^{2^{e+1}} + (\alpha^a x^{2^i+1})^{2^{e+2}} + \cdots + (\alpha^a x^{2^i+1})^{2^0} + \cdots + (\alpha^a x^{2^i+1})^{2^{e-1}}.$$

Note that the exponents of the terms in the sum all form $C(1)$. Hence, applying a power of 2 to the term of the trace results in an equal function. That is, $Tr((\alpha^a x^{2^i+1})^1) = Tr((\alpha^a x^{2^i+1})^{2^i})$ for any integer i. In Table 13 we observe patterns on the equality of these

functions. First, each function with a Gold exponent of the form 2^i+1 is equal to a function with Gold exponent 2^j+1, with $i+j=4$. Second, we had previously established Conjecture 1 on the powers of α that lead to bent functions, and the powers which did not. In four variables, for the Gold exponents 3,5 and 9, the cases where the function was not bent were when the power was a multiple of 3, 5 and 3, respectively. In this table, we see that this property is preserved for the non-bent functions. For the Gold exponent 3, the functions are sent to: $Tr(\alpha^3x^3)\rightarrow Tr(\alpha^9x^9)$, $Tr(\alpha^6x^3)\rightarrow Tr(\alpha^3x^9)$, $Tr(\alpha^9x^3)\rightarrow Tr(\alpha^{12}x^9)$ and $Tr(\alpha^{12}x^3)\rightarrow Tr(\alpha^6x^9)$. For the exponent 5, we have that $Tr(\alpha^5x^5)\rightarrow Tr(\alpha^{10}x^5)$ and $Tr(\alpha^{10}x^5)\rightarrow Tr(\alpha^5x^5)$. We use Lemma 1 to prove the following:

Theorem 14 (Equal Gold Boolean functions of the form $Tr(\alpha^ax^{2^i+1})$)**.** Let $\alpha\in\mathbb{F}_{2^m}^*$ be a primitive element, and a,b,i,j integers such that $0<a,b<2^m-1$, $0<i,j<m$ and $(i,m)=(j,m)=1$. Then, the Boolean functions in m variables $Tr(\alpha^ax^{2^i+1}), Tr(\alpha^bx^{2^j+1})$ are equal if $2^j*(2^i+1)\equiv 2^j+1 \pmod{2^m-1}$ and $b=2^j(a)$, $j+i=m$.

Proof: As per Definition 5 we know that $Tr(\alpha^ax^{2^i+1})=(\alpha^ax^{2^i+1})^{2^0}+(\alpha^ax^{2^i+1})^{2^1}+\cdots+(\alpha^ax^{2^i+1})^{2^{m-1}}$. Since $0<j<m$ and $j+i=m$, one of the terms of the sum will be:

$$(\alpha^ax^{2^i+1})^{2^j}=(\alpha^{2^j(a)}x^{2^{j+i}+2^j})=(\alpha^{2^j(a)}x^{2^m+2^j}).$$

With $2^m\equiv 1 \pmod{2^m-1}\rightarrow(\alpha^{2^j(a)}x^{2^m+2^j})=(\alpha^{2^j(a)}x^{1+2^j})$
Similarly, we have that $Tr(\alpha^bx^{2^j+1})=(\alpha^bx^{2^j+1})^{2^0}+(\alpha^bx^{2^j+1})^{2^1}+\ldots(\alpha^bx^{2^j+1})^{2^{m-1}}$, since $0<i<m$, then one of the terms of the sum will be:

$$(\alpha^bx^{2^j+1})^{2^i}=(\alpha^{2^i(b)}x^{1+2^i})=(\alpha^{2^i(2^j(a))}x^{1+2^i})=(\alpha^ax^{2^i+1}).$$

As stated in previous discussions, $Tr(x)=Tr(x^{2^t})$ for any integer t. Thus, $Tr(\alpha^ax^{2^i+1})=Tr((\alpha^ax^{2^i+1})^{2^j})=Tr(\alpha^bx^{2^j+1})=Tr((\alpha^bx^{2^j+1})^{2^i})$.

□

With these theorems combined with our previous observations from subsection 5.1.3, we state the following conjecture.

Conjecture 2 (Further results on the Gold bent function cases)**.** Let $\alpha\in\mathbb{F}_{2^m}^*$ be a primitive element. For i,j,a,b integers such that $0<i,j<m$, $i+j=m$ and $0<a,b<2^m-1$ then the Gold Boolean functions in m variables of the form $Tr(\alpha^ax^{2^i+1}), Tr(\alpha^bx^{2^j+1})$ and $b=2^j(a)$ are bent for the same powers of α.

The proof for this depends on Conjecture 1. In particular, we gave various conditions over the exponent a for the Gold Boolean functions of the form $Tr(\alpha^ax^{2^l+1})$ to be bent in the previous subsection. Specifically, based on shared factors of l,m, the exponents of α such that the function was not bent varied. The most well-known case is when $(l,m)=1$ as showed by Dillon and Dobbertin [22] (although they stipulated that m could not be divisible by 6, which we do not consider in our results in subsection 5.1.3). For this case, Whenever α^a is a cube element in the field, then the function is not bent. This corresponds to saying that a is a multiple of three as per Lemma 1. For this case, it is clear that the conjecture is true as the pair of Gold functions of the form $Tr(\alpha^ax^{2^i+1}), Tr(\alpha^bx^{2^j+1}), j+i=m, (i,m)=(j,m)=1$ and $b=2^j(a)$ will both share the same condition for the function being not bent, that is, a,b

not being multiples of 3. However, for the general case, say $Tr(\alpha^a x^{2^i+1})$ being not bent for a a multiple of some integer $0 < k < 2^m - 1$. If $k|(2^m - 1)$, then we have that the multiples of k repeat themselves with a period of $\frac{2^m-1}{k} \pmod{2^m-1}$, i.e $k, 2k, 3k, \ldots, k(\frac{(2^m-1)}{k}), k(\frac{(2^m-1)}{k} + 1), \ldots$ which taken $\pmod{2^m-1}$ is $k, 2k, 3k, \ldots, 0, k(1), \ldots$. This means that for some positive integer q , if $kq > 2^m - 1$, then it is equivalent to a multiple of k when you take the modulo. Consider the pair of Gold functions of the form $Tr(\alpha^a x^{2^i+1}), Tr(\alpha^b x^{2^j+1}), j + i = m$ and $b = 2^j(a)$. If a is a multiple of k, then so is b by the previous discussion. We know that $Tr(\alpha^a x^{2^i+1})$ is not bent only for $a \in Mk$ (where Mk is the set of multiples of k). Consider any b such that $Tr(\alpha^b x^{2^j+1})$ is not bent. Then, we know from Theorem 14 that there exists some integer a such that $b = 2^j(a)$ with $Tr(\alpha^b x^{2^j+1}) = Tr(\alpha^a x^{2^i+1})$. For $Tr(\alpha^a x^{2^i+1})$ to be non-bent, a must be a multiple of k, say $a = kq$, for some positive integer q. Then, $b = 2^j(a) = 2^j(kq) = k(2^j q)$. Thus, if $Tr(\alpha^b x^{2^j+1})$ is not bent, then it must meet the same conditions as $Tr(\alpha^a x^{2^i+1})$. The final proof for our Conjecture 2 depends on proving the results from Conjecture 1 and showing that the possible values of k always divide $2^m - 1$.

The implication of this conjecture is that the Gold Boolean bent functions of the form $Tr(\alpha^i x^{2^l+1})$ are generated for $l < \frac{m}{2}$, as the other values of l will generate the same bent functions. This is consistent with Table 1 in Carlet's paper [15], as they define the Gold bent exponents associated to AB functions as $2^l + 1$ with $(i, m) = 1$ and $1 \leq l < \frac{m}{2}$. The results of our Table 13 imply that these pairs of functions are the only such equal pair of Gold bent functions for $(l, m) = 1$. This result may be related to our previous results on the size of the cyclotomic cosets that contain the Gold exponents. We use our computational results to state and prove the following theorem.

Theorem 15 (Upper Bound For The Number of Non-Equal Gold Bent Functions of The Form $Tr(\alpha^i x^{2^l+1})$)**.** Let $\alpha \in \mathbb{F}_{2^m}^*$ be a primitive element. For $0 < i < 2^m - 1$, $0 < l < m$, $i + j = m$, $(i, m) = (j, m) = 1$ and $0 < a < 2^m - 1$ define $Tr(\alpha^i x^{2^l+1})$ as the Gold Boolean bent functions in m variables. Then, there are at most $((2^m - 2) - (\frac{2^m-1}{3} - 1)) * (\frac{\Phi(m)}{2})$ non-equal Gold Boolean bent functions, where Φ is Euler's phi function.

Proof: For a Gold Boolean bent function $Tr(\alpha^i x^{2^l+1})$ in m variables, we can iterate over $2^m - 2$ different exponents of α (as we do not consider α^0). Out of these exponents, the ones that are multiples of 3 do not lead to bent functions when $(i, m) = 1$ [15, 22]. The formula for computing the number of multiples of three less than $2^m - 1$ is given by $\frac{2^m-1}{3} - 1$. We know that $3|(2^m - 1)$ as m is even and $2^m \equiv 1 \pmod 3$ for m an even integer. The total number of Gold exponents that satisfy $(l, m) = 1$ is given by $\Phi(m)$. Since these exponents appear in the trace functions in pairs of the form $(\alpha^a x^{2^i+1}), (\alpha^{2^j a} x^{2^j+1})$, by Theorem 14 at maximum the total number of non-equal Boolean bent functions is $\frac{\Phi(m)}{2}$. Thus, the total number of functions whose power of α is not a multiple of three is given by $(2^m - 2) - (\frac{2^m-1}{3} - 1)$ and the total number of Gold exponents that do not lead to equal functions is at most $\frac{\Phi(m)}{2}$. So, the total number of functions whose power of α is not a multiple of three and its power of x is a valid Gold exponent is upper bounded by $((2^m - 2) - (\frac{2^m-1}{3} - 1)) * (\frac{\Phi(m)}{2})$.

□

Based on all the results have so far, we postulate the following conjecture:

Conjecture 3 (Exact Number of Equal Gold Bent Functions of The Form $Tr(\alpha^i x^{2^l+1})$)**.** Let $\alpha \in \mathbb{F}_{2^m}^*$ be a primitive element. For $0 < i < 2^m - 1$, $0 < l < m$, $i + j = m$, $(i, m) = (j, m) = 1$ and $0 < a, b < 2^m - 1$ define $Tr(\alpha^i x^{2^l+1})$ as the Gold Boolean bent functions in m variables. Then, there are $((2^m - 2) - (\frac{2^m-1}{3} - 1)) * (\frac{\Phi(m)}{2})$ non-equal Gold Boolean bent functions, where Φ is Euler's phi function.

To prove this it remains to show that no other Gold Boolean bent function is equal to the pair $Tr(\alpha^a x^{2^i+1}), Tr(\alpha^{2^j a} x^{2^j+1})$ as described in Theorem 14.

5.3. **Kasami-Welch Bent Function Construction.** In this section, we consider the Boolean functions in m variables of the form $Tr(\alpha^i x^d)$. In this subsection we reference results from our Springer PROMS article [67]. We define a list of Kasami-Welch exponent: $2^{2l} - 2^l + 1$ for $0 < l < m$ and repeat a similar methodology as the Gold case. For the following subsection, we develop an additional algorithm to construct the Kasami-Welch bent functions. Dillon and Dobbertin studied and obtained results on the construction of the Kasami-Welch bent functions [22]. One of their main results is on the conditions under which these functions are bent (see Theorem
refTh:KWDill). In this section, we construct algorithms that generate Kasami-Welch Boolean functions and determine when they are bent. We obtain computational results that show that there exist Kasami-Welch bent functions in m number of variables divisible by 6, thus removing these criteria from Dillon and Dobbertin's theorem. We conjecture a generalized form of their result on the bent-ness of the functions based on these results.

5.3.1. *Algorithms.* We construct an algorithm to determine if the Kasami-Welch Boolean function is bent or not [67].

Algorithm 10. [67]
```
def BentTraceIterationKas(m):
    f = FindModNonCube(m)
    R.<x> = GF(2^m,'a', modulus = f)[]
    k.<a> = GF(2**m, modulus = f)
    print(''Variables, Exponent, Power of the element, Is it Bent?'')
    for j in range(1,m):
      for i in range(0,2^m - 1):
        KB = BooleanFunction((a^i)*x^(K[j]))
        print(m,",", K[j], ", α^",i," ,",Bent-ness(KB))
```

The algorithm repeats the same process as the Gold case algorithm. The key difference is that we pre-generated and used a list of Kasami-Welch exponents.

5.3.2. *Explicit Families of Kasami-Welch Bent Functions Not Obtained by Dillon and Dobbertin:* We generated the functions of the form $Tr(\alpha^i x^d)$, $d = 2^{2l} - 2^l + 1$ for up to 12 variables and for all values of i such that $0 \leq i \leq 2^m - 2$. We note that for (l,m) = 1 our computational results show that Theorem 2 is met as the exception found are when i is a multiple of 3. However, for six and 12 variables, we find bent functions. These exist for (l,m) = 1. We show the results for the six and 12 variable cases in Table 7 in Appendix 8.

The results show that the non-divisibility by six criteria given in [22] can be removed and still generate some bent functions. The conditions on λ and (l,m) = 1, however, are still satisfied for these cases. We propose a generalization of the bent function result of their theorem as follows:

5.3.3. *Conjectures For The Construction of Kasami-Welch Boolean Bent Functions.*

Conjecture 4 (Velazquez, Janwa [67] Generalized Kasami-Welch Boolean Bent Functions)**.** Let $L = \mathbb{F}_{2^m}$, where m is an even integer, let α be a primitive element in $\mathbb{F}_{2^m}$ and the Kasami-Welch exponent $d = 2^{2l} - 2^l + 1$, (l,m) = 1 . Let $Tr(\alpha^i x^d)$ be the Kasami-Welch Boolean functions and Mk the set of multiples of the integer k. We have:

$$s_d^{\alpha^i}(x) = Tr(\alpha^i x^d) \text{ and let } \rho_d^{\lambda} = (-1)^{s_d^{\lambda}}$$

then

$$\text{If } i \notin M3, \text{then } s_d^{\alpha^i}(x) \text{ is bent i.e } \hat{\rho}_d^{\alpha^i}(\beta) = \pm 1, \forall \beta \in L \tag{47}$$

6. Weight Distribution of Dual Cyclic Codes and Some Conjectures

6.1. **Symmetric Weight Distribution of Generated Codes.** The weight distribution of cyclic codes is directly related to cross-correlation values of binary m-sequences. For α as a primitive element in the field $\mathbb{F}_{2^m}$, with $m_i(x)$ being the minimal polynomial of α^i over $GF(2)$, then the cross-correlation values of two m-sequences with characteristic polynomials $m_1(x), m_i(x)$ give information on the weight distribution of the dual code of the cyclic code with defining set $\{1, i\}$ [51]. In 1976, Helleseth conjectured that two binary m-sequences of length $2^m - 1$ (m a power of 2) cannot have a 3-valued cross-correlation [38]. McGuire in [51] considered that if -1 was a cross-correlation value, then the values must have the form $-1, -1 + A, -1 + B$ for some A, B. If $A = B$ then it completes part of Helleseth's conjecture as one must then prove that the cross-correlation values do not have that specific form. This is equivalent to stating that the corresponding dual code must have a weight distribution of $[2^{m-1} - a, 2^{m-1}, 2^{m-1} + a]$. McGuire in 2004 studied the weight distribution of the dual code of the binary cyclic code of length $2^m - 1$ with two roots [52]. According to McGuire, while there are various conjectures on the instances in which the dual code is a three-weight code, a complete classification is difficult [52]. It is conjectured that the weights of these codes are symmetric over 2^{m-1}. For the construction and analysis of three-weight dual codes, we also consider the "Cusick exponents" which are known to lead to three-valued cross-correlation functions [18, 23].

Conjecture 5 (On the weights of three nonzero weight codes [52])**.** Let C be a cyclic code over $\mathbb{F}_{2^m}$ and $C^{\perp}$ its dual code. If $C^{\perp}$ has three weights of the form $w_1 = 2^{m-1} - a$, $w_2 = 2^{m-1}$, $w_3 = 2^{m-1} + b$ then $a = b$.

We have verified this conjecture for the codes we generated in Section 3. We utilized the results on the spectrum of the codes from Algorithm 2. We considered all possible values of the second element of the defining set of the codes. According to Ding and Carlet [23, 12], many of the codes with three nonzero weights are considered optimal, and as such

we focus on their construction. Using Algorithm 2 we construct Table 19. We note that all the codes satisfied the conjecture. In the following subsection we compare these codes to known theorems compiled by Ding in [23] about the weight distribution of dual codes of cyclic codes in two roots.

6.2. **Ding Weight Distribution Tables.**

Theorem 16 (Calderbank, Goethals, Gold and Kasami [8, 30, 31, 43] Three Weight Codes First Theorem)**.** Let m ≥ 4 and the defining set of the code binary cyclic code "C" of length $n = 2^m - 1$ is $\{1, 2^e + 1\}$ for some integer e $\leq \frac{m}{2}$. Then C is a three weight code if and only if either m is odd and $(2^e + 1, n) = 1$ or m is even and e $= \frac{m}{2}$.

The distribution for the first case the nonzero weights will be: $(2^{m-1} - 2^{m-1-l}, 2^{m-1}, 2^{m-1} + 2^{m-1-l})$ with $l = \frac{(m-gcd(m,e))}{2}$. For the second case, the nonzero weights are: $(2^{m-1} - 2^{\frac{m-2}{2}}, 2^{m-1}, 2^{m-1} + 2^{\frac{m-2}{2}})$.

Theorem 17 (Ding [23] Three Weight Codes Second Theorem)**.** The binary cyclic code "C" of length $n = 2^m - 1$, that has dimension $2m$ and defining set $\{1, v\}$ has the nonzero weights of the first case of the previous theorem with $l = \frac{m-1}{2}$ and $l = \frac{m-2}{2}$ (for m odd and even respectively) given that v is equal to:

1) $2^{\frac{m+2}{2}+3}$, $m \equiv 2 \pmod 4$
2) $2^{\frac{m}{2}} + 2^{\frac{m+2}{4}} + 1$, $m \equiv 2 \pmod 4$
3) Is the Kasami-Welch exponent with $\frac{m}{(l,m)}$ odd
4) Is the Welch exponent with m odd
5) Is the Niho exponent (even and odd case)

The dual codes of the codes obtained by Theorems 17 and 16 (when the exponent is $2^d + 1$ and $(d, m) = 1$) are known to be optimal [23]. With the conditions given, we begin the construction of an algorithm to identify which of the constructed codes fall under these theorems.

Algorithm 11. {`def DistIdent(`m`,CR):`

```
    C1 = []
    C2 = []
    N = []
    I = []
    D = []
    W = []
    n = 2^m - 1
    if m%4 == 2:
      C1.append(2^(m/2) + 2^((m + 2)/4) + 1)
      C2.append(2^(m/2 + 1) + 3)
    if m%2 == 1:
      I.append(2^m - 2)
      W.append(2^((m-1)/2) + 3)
```

```
      if m%4 == 1:
        N.append(2^((m - 1)/2) + 2^((m-1)/4) - 1)
      if m%4 == 3:
        N.append(2^((m - 1)/2) + 2^((3m-1)/4) - 1)
      if m%5 == 0:
        D.append(2^(4m/5) + 2^(3m/5) +2^(2m/5) +2^(m/5) - 1)
    if set(Cyclo(CR,n)) & set(K) != set([]) or set(Cyclo(CR,n)) & set(W) !=
set([]) or set(Cyclo(CR,n)) & set(N) != set([]) or set(Cyclo(CR,n)) & set(C1)
!= set([]) or set(Cyclo(CR,n)) & set(C2) != set([]):
      if m%2 == 1:
        l = (m - 1)/2
        a = 2^((m-1)/2) - 2^(m-1-l)
        b = 2^(m-1)
        c = 2^((m-1)/2) + 2^(m-1-l)
        return "Distribution type 1, is" a,b,c
      else:
        l = (m - 2)/2
        a = 2^((m-1)/2) - 2^(m-1-l)
        b = 2^(m-1)
        c = 2^((m-1)/2) + 2^(m-1-l)
        return "Distribution type 1, is" a,b,c
    if set(Cyclo(CR,n)) & set([G]) != set([]) and (m ≥ 4 and CR ≤ (2^(m/2)+
1)):
      if gcd(n,CR) == 1:
        e = log(CR - 1,2)
        l = (m - gcd(m,e))/2
        a = 2^((m-1)/2) - 2^(m-1-l)
        b = 2^(m-1)
        c = 2^((m-1)/2) + 2^(m-1-l)
        return "Distribution type 1, is" a,b,c
      if m%2 == 0 and CR ==(2^(m/2) + 1):
        a = 2^(m-1) - 2^((m-2)/2)
        b = 2^(m-1)
        a = 2^(m-1) + 2^((m-2)/2)
        return "Distribution type 2, is" a,b,c
  }
```

The input of the code will be the list of cyclotomic coset representatives of the three nonzero weight cyclic codes generated previously and m, where the length of the code is $n = 2^m - 1$. Lines 2-7 define the lists where we will store the exponents that are used in the theorems (Gold, Kasami-Welch, Welch, inverse, Cusick, Niho and Dobbertin) and define the length of the code. Lines 8-19 construct the exponents based on the length of the code (Gold and Kasami-Welch exponents are pre-generated since they do not depend on the number of variables m). Line 20 verifies if the second element in the defining set of the code belongs to

the set of exponents defined in Theorem 17. For this, we use the "Cyclo" function that gives a list of the elements of the cyclotomic coset that contains the second element of the defining set under consideration. This is done to account for cyclotomic equivalence. The following lines verify the respective conditions needed for the distribution given that the element satisfied Theorem 17 and the other lines in the middle give the weight distribution. Next, the algorithm verifies the conditions needed to apply Theorem 16 (that the element is a Gold exponent with $l \leq \frac{m}{2}$). Then the following lines verify the conditions for the distributions and return the distribution.

With this algorithm, we were able to classify some of our 95 codes under the distribution types (based on the tables in [23]) The result was Table 8. These codes were obtained by using Algorithm 2 to identify three nonzero weight dual codes after iterating over all possible cyclotomic coset representatives. This was done for cyclic codes of length $2^m - 1$, for $m = 4$ to $m = 13$. Several codes did not meet any of the theorems (yet still presented a symmetric weight distribution). It is possible that some of the codes not meeting the distribution conditions are equivalent to some that do. We construct a table considering equivalent codes. The analysis for "equivalent codes" was done as follows. Given that the defining set of the cyclic code of length $n = 2^m - 1$ is $\{1, d\}$ we identify "z" such that $d * z \equiv 1 \pmod{n}$. The new "equivalent" code will have defining set $\{1, z\}$. The condition that m must possess a multiplicative inverse must be considered, as otherwise, there is no "equivalent" code based on this construction. This construction is validated by Niho in his 1972 report [54] where they discuss these "inverse pair relationship" of codes based on the non one entry in their defining set, and the context of this work is on cross-correlation and d-decimation functions which are directly related to the weight distribution of the dual codes [47]. This notion of cyclic code equivalence also coincides with the cyclotomic equivalence concept defined in Definition 9. As seen in Table 9. Once the inverse element was identified (given its existence) we used the cyclotomic coset representative of the element in our tables. For example, the multiplicative inverse of 43 modulo 2047 is 1809, but we utilized 143 as 143 is the cyclotomic coset representative of 1809. All the codes that did not meet the theorems directly were equivalent to some codes that did (See Table 9.

6.3. **Weight Distribution Classes of Cyclic Codes.** In this subsection, we construct cyclic codes in two, three, and four roots for four, five, six and seven variables and observe their weight distribution classes. To accomplish this, we generate the corresponding trace functions and list some of their cryptographic properties. The codes are generated by taking the cyclotomic coset representatives of all the exponents considered (see Table 1). We also identified to which exponent type the defining set elements belonged to. The methodology of this section is motivated by results obtained by Janwa and Wilson in [41] and Zeng in [73]. Janwa and Wilson proved the necessity of APN exponents for the construction of two-error-correcting cyclic codes. Close to 20 years later, Zeng studied the three-error-correcting case and analyzed some combinations of known APN exponents that led to codes with the same weight distributions as the three-error-correcting BCH [1,3,5] code. This is a cyclic code with defining set $\{1, 3, 5\}$.

Due to the connections between the construction of bent and near-bent trace functions and the construction of APN functions/two-error-correcting codes, part of our analysis is focused on other cryptographic properties of these functions and how they may relate to the corresponding codes. These properties follow:

Definition 15 (Forrester [27] Correlation Immunity)**.** We say that a Boolean function is correlation immune (or resilient) of order m if the output is statistically independent of any m entries.

Definition 16 (Forrester [27] Algebraic degree)**.** The algebraic degree of a Boolean function is the degree of its algebraic normal (polynomial) form.

Definition 17 (Taranikov [65] Autocorrelation)**.** Let f be a Boolean function on $\mathbb{F}_2^m$. For each $u \in F_2^m$ the autocorrelation coefficient of the function f at the vector u is defined as:

$$\hat{f}(u) = \sum_{x \in F_2^m} (-1)^{f(x)+f(x+u)}$$

Definition 18 (Han [35] Algebraic Immunity)**.** Algebraic immunity is a measure of the resistance of a Boolean function f to algebraic attacks. It is the smallest degree of a non-trivial annihilator of f or f+1 .

6.3.1. *Algorithms.* The first algorithm constructed is Algorithm 14. The input of the algorithm is the number of variables "m." The algorithm outputs all the trace Boolean functions of the form $Tr(x^d)$ where d corresponds to an APN/near-bent/bent exponent defined over $\mathbb{F}_{2^m}$ along with the properties considered. First, we utilize Algorithm 5 which finds an irreducible polynomial of degree m that is used as the modulus to construct the finite field and polynomial ring used. Then we define the length of the code "n" and the value $t = \frac{m-1}{2}$. For the Gold and Kasami-Welch cases, we print the headers of the table, then begin a for loop where the exponents are defined, and the corresponding functions constructed. We utilize a "CycloRep" algorithm which finds the cyclotomic coset representative $\pmod{2^m - 1}$. This is done to avoid considering multiple instances of equal functions. Then the functions printed with their cyclotomic cosets representative form and the Boolean function properties considered. For the non-Gold, Dobbertin, and Kasami-Welch case, the algorithm first verifies if the number of variables is odd. Then it goes case by case depending on the specific sub-conditions, defines the corresponding function, and prints them with their properties. For the Dobbertin case, it checks if the number of variables is divisible by five (as it is the corresponding condition for those functions to be defined).

The second algorithm constructed is Algorithm 15. The algorithm takes as an input the number of variables "m" and the size of the defining sets considered "sD." Empty lists are defined to store the Gold and Kasami-Welch exponents as well as the list containing the defining set and a list that will store the weight distribution of the codes. The values "n,t" are defined as in the previous algorithm. An empty list "EL" is defined with the purpose of storing all the considered exponents. The Gold and Kasami-Welch exponents are generated and appended into their respective lists. We note that for this algorithm we use the cyclotomic coset representative of the exponents to only consider defining sets with elements corresponding to distinct minimal polynomials. After the list of Gold and Kasami-Welch exponents is defined, the elements are appended to the list "EL." The other exponents of

interests are defined and appended to "EL" in a similar manner, verifying that the conditions for the exponents to be defined are met. Once all the exponents have been added to the list "EL," we take the set of elements in this list that are distinct and turn it back into a list (see line 26). Then all the combinations of the elements in EL of size sD are constructed such that the first entry is a one and stored as a list of lists "cyclo2" (lines 27-32). Then we begin a for loop that goes over all the possible defining sets of size sD with one as the first entry and constructs the corresponding codes. The algorithm finds the check polynomial of the cyclic code, whose reverse corresponds to the generator polynomial of the dual cyclic code. Said code is constructed and its weight distribution is assigned to the list "sd." The way the spectrum command outputs the weight distribution is by assigning the number of codewords with weight "i" to the "i-th" position in the list. Thus, we obtain the list of positions in the list that are nonzero (corresponding to the weight numbers that the code has) and then eliminate the 0-th position as we are only interested in the nonzero weights. Next, we append the defining sets and weight distributions to the lists DSL and WLL, respectively. A copy of the list of weight distributions is made, and a while loop begins under the condition that the copy of the list of weight distributions is not empty. The loop goes over all the elements that are equal in the list of weight distributions and the corresponding defining set. Then it eliminates from the list all the defining sets covered in the previous iteration of the loop and loops again.

Finally, we designed Algorithm 16 which takes as an input the number of variables "m" and a specific defining set whose first entry must be 1. The output will be a list that classifies each entry in the defining set as belonging to a specific APN/AB/bent exponent family. The first four lines of the algorithm define empty lists to store the Gold and Kasami-Welch exponents as well as defining "n,t" as in the previous algorithms. Next, it defines and assigns the exponents considered for the construction of the defining sets while changing them to their cyclotomic coset representative. Then, the following lines loop over the elements in the defining set that were used as an input and the variable j2 is used to find the corresponding cyclotomic coset representative. The algorithm verifies if the cyclotomic coset representative of the element in the defining set matches the cyclotomic coset representative of one of the APN/AB/bent exponents. If it is the case, then it classifies it as such, if not, it moves to the next candidate. This loop is repeated until we go over all the elements not equal to one in the defining set.

6.3.2. *Algorithm Results.* The results obtained from Algorithm 14 are observed in Tables 20,21,22 and 23. For the four variable case, we know that the maximum nonlinearity of Boolean functions is given by $2^{4-1} = 2^{\frac{4}{2}-1} = 8 - 2 = 6$. That is, none of the displayed trace Boolean functions reach maximum nonlinearity (i.e. they are not bent). We have not found any known constructions of bent functions of the form $Tr(x^d)$, although Table 10 shows that for the specific case of 8 variables $Tr(x^{15}), Tr(x^{45})$ are bent. For $l = 2$ we observe that the nonlinearity is 0. This is because $Tr(x^5) = x^5 + x^{10} + x^{20} + x^{40} = x^5 + x^{10} + x^5 + x^{10} = 0$ over $\mathbb{F}_{2^4}$. In terms of algebraic degrees, the ones that differ are the Gold and Kasami-Welch exponents that correspond to $l = 2$. For the Gold case, we have shown that the function is the 0 function. For the Kasami-Welch case, we have a function of algebraic degree 3. Similarly, the algebraic immunity of the functions is uniform except for when $l = 2$. This matches up with

the knowledge of the conditions for these exponents to lead to high nonlinear functions as $(l, m) = 1$. However, their nonlinearity is equal, yet other important cryptographic properties do vary. The autocorrelation, correlation immunity, and algebraic immunity are important in the Global Avalanche Criterion for good cryptographic functions as described by Zhang in 1996 [74]. The cases for $l = 2$ are also the only ones that present a distinct autocorrelation spectrum for the functions. Since for five and seven variables, all possible values of l are relatively prime, we continue these observations for the six variable case. The maximum nonlinearity for Boolean functions in six variables is $2^5 - 2^2 = 28$. We observe that the maximum nonlinearity achieved by these functions is 24. The Gold cases only had one set of functions that differed in the respective properties, this being $Tr(x^9)$, corresponding to $l = 3$. For the Kasami-Welch case, the nonlinearity and correlation immunity values do not change (and coincide with the $l \neq 3$ Gold cases). However, the algebraic degrees, algebraic immunity, and autocorrelation values do change. In fact, they are paired for values of l that sum to the number of variables. For example, for $l = 1, 5, l = 2, 4$ and $l = 3$ we have "pairs" of defined functions that share the properties. For five variables, The maximum nonlinearity of Boolean functions is $2^{5-1} - 2^{\frac{5-1}{2}} = 16 - 4 = 12$ which all the functions except the Dobbertin and inverse functions achieve. The Gold functions share the properties of the same functions considered. The Kasami-Welch case presents two distinct classes of properties. One for $l = 1, 4$ and the other for $l = 2, 3$. The first cases correspond to the Gold case, indicating that for those Kasami-Welch exponents, the application of these functions is equivalent in some way to the Gold cases. The Welch function presents the same properties as the Kasami-Welch case, while the Niho case presents the same properties as the Gold case. This indicates that for the near-bent exponents in five variables, we have two types of functions. The inverse and Dobbertin functions are APN but not AB. Thus they do not achieve maximum nonlinearity. The inverse and Dobbertin functions differ with both the Gold and Kasami-Welch case in algebraic degrees, but do share correlation immunity, algebraic immunity, and autocorrelation values with the Kasami-Welch case. Thus, if we only consider these last three properties, we can separate these functions into two main classes, the Gold class, and the Kasami-Welch class. For the seven variable cases, once again we observe that all the Gold functions share the same properties, while the Kasami-Welch functions are paired for values of l that sum to m. For $l \neq 1, 6$, the Kasami-Welch functions share the same last three properties. The Welch function had similar properties to the Kasami-Welch case, while the Niho function differed in the algebraic degree but shared all other properties. It is important to note that while EA-equivalence between these functions preserves algebraic degree, CCZ-equivalence does not [11]. Thus, it is still possible to classify this function as belonging to the Kasami-Welch class for this analysis. The inverse function belongs to its own class, as it has unique property values for every property except the correlation immunity.

Results from Algorithms 15 and 16 are observed in Figures 24-33. We proceed to analyze these tables and establish conjectures based on the results.

6.3.3. *Two-root Cyclic Code Observations.* We observe that the two root case follows expected patterns. The defining sets of the form $\{1, APN\}$ correspond to cyclic codes that are two-error-correcting. In four variables, when $l = 2$, no minimum distance, five codes are found as per Janwa and Wilson's results [41]. For the five variables case, the inverse and Dobbertin

exponents coincided in the cyclotomic coset (hence the codes were the same). The weight distributions of the dual codes corresponding to these exponents have more weights than for the Gold and Kasami-Welch cases. These are also the only ones that are not near-bent exponents, only APN. In six variables Gold for $l = 1, 5$ and Kasami Welch for $l = 1, 5$ lead to two-error-correcting codes. For $l = 2, 4$ Gold and Kasami-Welch have the same distribution and were not two-error-correcting (as expected since $(l, m) \neq 1$). For $l = 3$ the Gold and Kasami-Welch exponents had a unique weight distribution. In seven variables, only the inverse exponent had a different weight distribution, although it was still two-error-correcting. As per our trace function properties tables, we can deduce that the two-error-correcting capability of the codes corresponding to these functions is not directly dependent on a strict nonlinearity value but rather on the nonlinearity being above a certain threshold. Of course, in general, these functions are described as APN for two-error-correcting purposes, although some "Perfect Nonlinear" functions do fall into this category (as in, their nonlinearity is the maximum possible for the number of variables considered). We note that in five variables, the inverse and Dobbertin functions share their last three properties with the Kasami-Welch case, yet their weight distributions are unique. This implies that the algebraic degree and nonlinearity properties may be more important for determining weight distributions than correlation/algebraic immunity and autocorrelation of the corresponding functions.

As covered in a previous subsection, it is conjectured that the weight distribution is symmetric for three-weight codes, and these are constructed via APN functions. This is mainly for the odd case however, as McGuire in [52] states that for $n = 2^m - 1, \mathbb{F} = \mathbb{F}_{2^m}$ and $\alpha \in F$ a primitive root, then the cyclic code with two zeroes α, α^d (i.e defining set $\{1, d\}$), then the code could not have a three-weight code if m is even and $d \equiv 0 \pmod 3$. For the Gold and Kasami-Welch exponents, this happens whenever $(l, m) = 1$ as it would mean l is odd and thus $2^l + 1$ is divisible by three, and so is $2^(2i) - 2^i + 1$. This is what we observe in our results. In fact, for the odd case, McGuire proves that if the minimum distance is greater than three and the weight distribution of nonzero codewords is $2^{m-1} - a, 2^{m-1}, 2^{m-1} + a$ then m must be odd, the minimum distance is five, $a = 2^{\frac{m-1}{2}}$ and the weight distribution is that of the two-error-correcting BCH code. Our tables coincide with this result, which provides validation to our methodology.

6.3.4. *Three-root Cyclic Code Observations.* For the three-root cases, in four variables, the first classification is the one corresponding to the BCH [1,3,5] (cyclic code with defining set $\{1, 3, 5\}$) three-error-correcting code. All the codes in this class have the second entry in the defining set being a Gold exponent. As per the second class seen, double Kasami-Welch exponents do not lead to three-error-correction. Even though 5 corresponds to $l = 2$ which is not relatively prime to $m = 4$, a defining set did lead to three-error-correction. The second class has exponents Gold $l = 1$ - Kasami-Welch $l = 2$. For the third class, we have Gold $l = 2$ – Kasami-Welch $l = 2$, both of which have l as a divisor of m. This class could only correct one error. It is as if they are divided into three, two and one error-correcting classes. In both the second and first classes, we have combinations of APN and non-APN elements in the sets. Our initial observation was that perhaps if we have one APN exponent in the set, then we can guarantee two-error-correction, with the following element not necessarily being APN. What condition does this following element have to meet? We note that while five

corresponds to the Gold exponent with $l = 2$ (and hence not relatively prime to m, meaning the corresponding function is not APN), it is a three-error-correcting BCH code defining set element.

For five variables, we had multiple classes of weight distributions. We have one weight distribution class corresponding to three-error-correction and minimum distance six codes. This first-class corresponds to the BCH [1,3,5] code. All the defining sets in this first-class are pairs of APN exponents. We note that there are no inverse/Dobbertin exponents in the three-error-correcting class.

For six variables, there are two classes with minimum distance of three (one error-correcting). For the first one we find Gold $l = 2$ - Kasami-Welch $l = 2$ exponents. The second class has Gold and Kasami-Welch exponents that correspond to $l = 3$ in the same defining set and is unique to them.

For the seven variable case, the first class corresponds to the BCH [1,3,5] code. All the second and third elements of the defining set are APN (as seven is relatively prime to all possible values of $0 < l < 7$) and the other elements correspond to APN functions for m odd. There is a second class of codes that correct three errors. This class contains the defining set with Gold $l = 1$ 1 – inverse exponents. However, its number of weights is 20 vs. the five of the BCH class. This is an increase of four times the total distinct weights of the dual codes. As discussed before, the inverse and Dobbertin functions are APN yet do not achieve maximum nonlinearity. Based on our observations, we conjecture the following:

Conjecture 6. Defining Sets of Size 3 for Triple error-correcting Cyclic codes

Define C as a $[n, n-r, d]$ cyclic code with defining set of size 3. Then, at least one element in the defining set must correspond to a non-inverse/Dobbertin APN function for the code to be three-error-correcting.

Conjecture 7. Weight Distribution of Dual Cyclic Codes with Defining Set of Size ≤ 3

Define C as a $[n, n-r, d]$ cyclic code, $n = 2^m - 1$, $2^m - 2$, the inverse exponent for m an odd integer and $2^{\frac{4m}{5}} + 2^{\frac{3m}{5}} + 2^{\frac{2m}{5}} + 2^{\frac{m}{5}} - 1$ with $5|m$. Then, if these exponents are contained in the defining set of a cyclic code, the corresponding dual code will not have minimum number of nonzero weights.

6.3.5. *Four-root Cyclic Code Observations.* For the four variable case, the largest minimum distance observed is 15 This is because the resulting codes (corresponding to the weight distribution of the BCH [1,3,5,7] BCH code) which is just the trivial code of all 1s and all 0s codewords. As seen in Table 24, this is the only class of weight distributions constructed from non-equivalent APN/near-bent/bent exponents.

For the five variable case, the largest minimum distance is 11 (five error-correcting, BCH code case). Once again, when looking at the cyclotomic coset representative case, we observe only the class of minimum distance 11, implying that the other codes are products of repeated roots. An important observation for this case is that, based on the pattern of expected errors corrected, a minimum distance of 9 is what we expected (hence four-error-correcting for the defining set of size 4), yet in this case, we had a five-error-correcting code.

For the six variable case, the highest minimum distance observed was 7. The BCH code weight distribution class cannot be obtained from combinations of the considered exponents. At best we have previous BCH code plus some extra root which is either redundant or

does not increase the minimum distance. Table 31 shows that we have obtained six distinct classes of weight distributions. Since the number of variables is $m = 6$, it means that the BCH [1,3,5,7] code would only have one element in the defining set that corresponds to a highly nonlinear function (only three, as five, corresponds to $l = 2$ and seven does not correspond to any of the functions considered). This means that, for this case, high nonlinearity is not a strict requirement to obtain high error-correcting codes. However, it is still possible to optimize for the small number of nonzero weights. We observe that while there are four classes of codes with minimum distance of 7, the lowest number of nonzero weights is 7, while the largest is 18.

For the seven variable case, the highest minimum distance observed is 8. In Table 33 we observe that there are four classes of weight distributions for codes with minimum distance of 8. Once again, the corresponding BCH code cannot be found with this construction. There were 13 classes of codes of minimum distance 7. We note that one code of minimum distance seven is found that has less nonzero weights than a minimum distance seven code. In particular, the code with defining set $\{1, 3, 5, 9\}$ leads to a dual code with seven nonzero weights, while the code with defining set $\{1, 3, 63\}$ has 20 nonzero weights (see Table 32). However, the $\{1, 3, 5\}$ code has five nonzero weights on its dual code. This means that some exponents considered can decay the performance or properties of codes to below the properties of codes with a greater number of roots even for the same minimum distance. In this case, we observe that it is the inverse exponent that seems to lead to a code whose number of nonzero weights is significantly higher than for other exponent combinations. It is also present in 12 out of the 13 cyclic codes with more than 20 nonzero weights in Table 33.

7. LDPC Code Algorithms and Next-Generation NASA Code Construction Through Bent and Near-Bent Functions

The goal of this section is to use cyclic codes with two roots corresponding to known APN, bent, and near-bent exponents and consider them under the LDPC code context. We use previously constructed cyclic codes in two roots, and construct their parity check matrix. This matrix is then used in algorithms developed by Neal [53] that simulate the transmission of information over an AWGN channel. We compare the performance of these codes in terms of coding gain against other constructions to verify if they are competitive. We aim to propose a construction that satisfies next-generation NASA coding schemes criteria. LDPC codes have fast linear-time decoding through Bayesian Belief propagation. The codes we construct have very high code rates, which should be ideal for high data rates applications (like Mars/Lunar missions such as links between probes sent to Mars and the Moon). We focus on obtaining BER vs. SNR graphs for our codes by encoding messages and simulating deep-space transmission. We compare the performance of these codes to other constructions to determine a relative coding gain. We note that the codes constructed may not have ideal sparseness, for which the decoding complexity could suffer. The two conditions we examine in this work are the spectral efficiency (which, with the very high rates of our codes, allow for very strict bandwidth implementations) and coding gain.

7.1. **Algorithms used.** Neal's algorithm is used to measure the performance of codes over an AWGN channel [53]. The algorithm used considers C a $[n, k, d]$ linear code with $k = n - r$, r the rank of the parity check matrix. Consider "$C^{\perp}$" a $[n, r, d]$ dual code of C. The algorithm assumes that a codeword can be divided into r check bits, c, followed by k message bits, s. That is, the message bits are at the end. This means that the kxn generator matrix must have an kxk identity matrix as its last k columns. With these assumptions and proper matrix operations, the parity check matrix H can be divided into an rxr matrix A occupying the first r columns of H and an rxk matrix B occupying the remaining columns of H. For a codeword, it must satisfy that when multiplied by a parity check matrix, the result is the 0 vector. This is written as:

$$Ac + Bs = 0$$

Where, given that A is non-singular:

$$c = A^{-1}Bs$$

For example, for the $[7, 4, 3]$ Hamming code a corresponding parity check matrix:

$$H = \begin{bmatrix} 1 & 0 & 0 & 1 & 1 & 1 & 0 \\ 0 & 1 & 0 & 1 & 1 & 0 & 1 \\ 0 & 0 & 1 & 0 & 1 & 1 & 1 \end{bmatrix}, A = \begin{bmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \end{bmatrix}, B = \begin{bmatrix} 1 & 1 & 1 & 0 \\ 1 & 1 & 0 & 1 \\ 0 & 1 & 1 & 1 \end{bmatrix}$$

Given that the rows of H are linearly independent, A will always be non-singular. This reasoning is used in the storing of the parity check and generator matrix files in Neal's algorithm. We note that each of the generator matrix representation commands includes a specification for how the columns of the parity check matrix should be re-ordered so that the message bits come last. We specifically use the "dense" format for the generator matrices. This format stores $A^{-1}B$ in a dense format which represents the matrix by a pointer to a structure of type "mod2dense." It records the number of rows and columns in the matrix, with an array of pointers where the bits of each column are stored. We note that this is for the representation of the generator matrix. However the file containing the parity check matrix constructed is not altered in its file.

The commands used in this work are:

1) make-pchk pchk-file n-checks n-bits row:col

pchk-file is the name of the pchk file. This is followed by the number of rows and the length of the code. Then you input the pairs i:j where each such entry corresponds to a 1 in the (i,j) position of the matrix. This will contain the data on the parity check matrix of the code.

2) make-gen pchk-file gen-file method

The first input is the parity check file of the code you want the generator matrix of. Then the name of the generator matrix file is inputted, followed by the specific representation method.

3) rand-src source-file seed n-bitsxNumber

Creates a file of random message vectors. First is the name of the file, then the random seed used to generate these values, followed by the length of the messages and the number of messages generated. The seed is a form of randomization for the generation of these values. For our purposes, we use 58 as the random seed.

4) encode pchk-file gen-file source-file encoded-file

The first inputs are the name of the corresponding parity check, generator matrix, and message files. Then the name of the encoded file you want to create.

5) transmit encoded-file received-file seed channel

This command transmits the encoded messages through the specified channel. A random noise will add errors to the information transmitted. This noise is generated pseudo-randomly based on the seed. We also select 58 as our random seed for all computations. The input is the name of the encoded file, followed by the name of the "received file." Then the seed and specific channel where transmission occurs. The channel input also considers the standard deviation of the AWGN channel. This has a direct relationship with the signal-to-noise ratio (SNR) measured for the transmission. We provide tables with the conversion between standard deviation values and SNR values of our codes. This conversion depends on the rate of the codes and, as such, varies with the length of our codes.

6) decode pchk-file received-file decoded-file channel method

This command decodes the codewords after being transmitted. The results show the total of blocks decoded, the number of blocks that lead to valid codewords, the average number of iterations of the decoding algorithm used, and the percentage of bits that were changed. The inputs are the name of the parity check, generator matrix, and decoded files. Then the channel is used to transmit the codewords and the method of decoding (in our case, belief propagation).

7) verify pchk-file decoded-file gen-file src-file

The command verifies if the blocks in the decoded file are codewords according to the parity check matrix in the pchk file. If the generator matrix file is specified, it displays the bit error rate (BER) by comparing the decoded message bits with the true message bits.

Algorithms were developed to construct these files and do the decoding and verification analysis. The Algorithms developed are Algorithms 17-23.

The first algorithm is "makepchkJose," which takes as input the parity check matrix of our codes (as constructed in Proposition 1). It also takes as inputs the roots of the cyclic code corresponding to the check matrix. The inputs are coded such that you can input a

cyclic code of up to six roots, but if you input less than six roots, the algorithm ignores the variables that are unassigned. Then the algorithm determines the number of rows of the matrix, its rank, and the length of the corresponding code. This is done by computing the number of entries in the first row of the matrix "(list(H[0]))." Then, the number of variables m is computed as $log_2(n+1)$ (since the length of the codes is $n = 2^m - 1$) and the number of roots. Then, it begins printing the command based on the entries needed as described above. The naming convention that we use for most commands is "cyclicLDPC-n-len-m-var-#rt-1-d1-d2-d3-d4-d5-d6," with the standard deviation and seed added for the transmission and source file cases. The random source file does not depend on the codes, so we simply title it "cyclicLDPC-n-len-m-var-#rt" and we only generate one such file per code length. Then it iterates over the roots that were used as an input and adds them to the name of the file. The "makegenJose" algorithm follows a similar methodology. Its inputs are the number of variables and the roots of the cyclic code. Then it verifies which roots were given as an input and uses them to print the command. The "randsrcJose" and "encJose" algorithms also use as inputs the roots and number of variables for the cyclic code. Both algorithms determine the number of roots on input and print the command. For the encoding algorithm, the command requires the names of previous files, so it is extended to construct such names based on the inputs. The transmit commands takes an additional input, that being the standard deviation of the AWGN channel (which is specified) under consideration. It determines the roots of the code and constructs the name of the files required in the commands. It adds the standard deviation value after the name of the received file. The verify command also takes as input the number of variables, the standard deviation, and the roots of the code. It determines the roots of the code and constructs the command needed, adding the standard deviation to the end of the decoded files name. Finally, the decoding command algorithm uses as inputs the number of variables, the standard deviation, the number of iterations used in the decoding, and the roots of the corresponding cyclic code. It adds the number of iterations at the end of the code. We also specify that the transmission occurred over an AWGN channel. We avoid using the minus sign before the number of iterations as it guarantees that the algorithm stops if it reaches a likely solution rather than doing all iterations.

7.2. **Methodology.** For our parameters, we use a list of known APN, bent and near-bent exponents based on known tables (see Table 1). Our standard number of iterations is 100, matching the number of iterations used by Andrews in [1]. We ran these commands from 3 to 12 variables. We observed the results and adjusted the codes/parameters based on the observations. Our initial results were obtained considering up to 250 iterations and with a standard deviation = 0.80. Observations showed that as the length of the code increased, although we expected more valid codewords, instead, the source and check errors increased. After length 127, no correction happened. For three variables, 28.7 iterations at max and 10% bit change. For five variables, 219 iterations at max and 1% bit changes; for seven variables, 250 iterations and 0% bit changes. Perhaps by setting the number of iterations this high, we discarded possible solutions, or not enough iterations were done for the larger length codes. We note that in [1], the maximum number of iterations was set to 100, although the author state that most codes decoded in five or so iterations. Thus, our starting value of 250 is well over the expected value. Our first attempt was to increase the number of iterations to 500.

While the number of iterations increased, the performance did not improve significantly. The bit change percentage remain like the 250 iterations case. For codes of length more than 127, no valid words were found and had a 0% bit change. We considered the possibility that too many iterations caused valid results to be thrown out. Results show that for a lower number of iterations, the code performance did not improve; we still observe significant performance degradation for lengths 31+ and 0% bit changes for codes of length 127. Our next step was to vary the standard deviation of the AWGN channel. Recall that these previous results are done for a standard deviation of 0.80. This corresponds to a high SNR value (for code rates of $\frac{1}{2}$, it is an SNR of 1.94). The results showed by Andrews in [1] (see Figure 1) indicate that the codeword error rate increases significantly for these codes for SNR values below 2. This means that our results are expected, or at the very least not outside the expected range of values.

According to Neal [53], the relationship between SNR and standard deviation of the AWGN channel is given by $\frac{E_b}{N_0} = \frac{1}{2Rs^2}$, where R is the code rate and s the standard deviation. This is then represented in units of dB by applying a log base 10 to this result and multiplying it by 10 [53]. Since our codes have varying code rates depending on their length, the range of standard deviation values used was different. Since the rate of our codes is much higher than the rate of the examples in Neal's page article (a rate of 0.5) we expect the values to change. As the length increases, we expect negative values to be showed, as the rate will be much higher, meaning the ratio computed $\frac{1}{2Rs^2}$ will be smaller, and thus the logarithm will be a smaller number.

We utilize this relation for the construction of Algorithms 12 and 13.

Algorithm 12. {
```
def StdtoSNR(s,R):
    s2 = s^2
    Ratio = 1/(2*R*s2)
    SNR = 10*log(Ratio,10)
    return(round(SNR,3),round(s,3))
}
```

Algorithm 13. {
```
def SNRLDPCJose(m,r,s):
    n = 2^m - 1
    K = n - r*m
    R = K/n
    return(StdtoSNR(s,R))
}
```

The first algorithm, "StdtoSNR" takes as an input the standard deviation of the channel and the rate of the code. It computes the square of the standard deviation, and then it utilizes the definition given by Neal on how to compute the signal to noise ratio via the relation $\frac{E_b}{N_0} = \frac{1}{2Rs^2}$. Then log base 10 is applied to this result and multiplied by 10. This algorithm is for general use; if you have the rate of the code and the standard deviation, you we can obtain the SNR. The second algorithm is for our construction. The input will be the number of variables, the number of roots considered, and the standard deviation of the AWGN channel. Then the length of the code, its dimension, and rate will be computed. The rate is then used on the previous algorithm to compute the corresponding SNR. In Table 34

we list the results from Algorithm 13. Since the rate of our codes is much higher than the rate of the examples in Neal's page (a rate of 0.5) we expect the values to change. As the length increases, we expect negative values to be showed, as the rate will be much higher, meaning the ratio computed $\frac{1}{2Rs^2}$ will be smaller and thus, the logarithm will be a smaller number. We utilize similar algorithms to convert from SNR to standard deviation values and create tables with these values.

To obtain our results, we defined the number of iterations, SNR values, and codes that we considered. We first considered the range of values for the SNR as all the integers from one to 19, but the data points were too sparse to compare with graphs from other articles. We then selected values in the range $[0.25, 19]$ going with steps of size 0.25. The source files used are the same for all the different codes of same length. The command was written such that the algorithm stops the first time it finds a valid codeword (thus, not always doing the maximum number of iterations). As a comparison point, we also computed the LDPC code of length 2000 and dimension 1000 in Neal's page (see Figure 7). We computed the standard deviation values that correspond to the range of SNR values we considered for our codes (see Table 34). Since the SNR depends on the rate of the code, a different set of standard deviation values corresponding to the same SNR range as in our codes was computed for the example case (see Table 35). We then encoded, transmitted, decoded, and verified the codes. We include the graph of this example with the graph of our codes. The example code used is an LDPC code of length 2000 and dimension 1000, i.e a rate of $\frac{1}{2}$. The code was made via the "evenboth" three method. That is, it produces a matrix in which the number of 1s in each column is approximately 3. The evenboth method attempts to make the number of checks per row be approximately uniform. The "no4cycle" option causes cycles of length four to be removed from the code if possible. This example code did not contain any cycles of length four, as 24 such cycles were eliminated by moving checks within a column. For the near-bent case, we constructed codes for lengths 7,31,127,511 and 2047. We considered defining sets $\{1, 3\}, \{1, 9\}, \{1, 57\}, \{1, 129\}$ and $\{1, 16257\}$ (see Figures 7, 8 and 9) . The performance of these codes is equivalent. For the Bent cases, we computed LDPC Codes for lengths 15,63,255 and 1023. We considered defining sets $\{1, 3\}, \{1, 5\}$ (corresponding to $l = 1, 2$ in the Gold case) and $\{1, 5\}, \{1, 9\}, \{1, 17\}, \{1, 33\}$ (see Figures 10, 11 and 12) for codes of length 15,63,255 and 1023 respectively.

7.3. **SNR Performance Improvement Comparisons to Other Codes.** We note that the codes constructed show a lower bit error rate for a given SNR than the example code in the range [0.25,1). It is of note that we can achieve much lower bit error rates with codes of length 31 and 511, which are significantly smaller in length than the example case. This proves promising. While the interval seems small, Thorpe in [66] shows his construction for protograph-based LDPC codes in the range [0.5,1.3]. This means that our area of improvement is relevant when compared to Thorpe's construction (see Figure 2). Smarandache, Divsalar Sah and also consider codes that perform well for values of SNR in these ranges [63],[24], [61] (see Figures 3, 5 and 4). A promising result is that, even though our codes have girth four (maximum size of shortest cycle in the graph), they can outperform girth > 4 LDPC codes. The girth property of graphs is important for LDPC code performance, as it directly affects the decoding process [62]. We created various files that saved the commands

used to construct the encoded, transmitted, decoded, and verification files. We also saved the results of the decoding steps on separate files to compare the performance of the codes in terms of valid codewords and the number of iterations needed. It seems that the performance improvements, based on the current parameters, plateaus at length 511. This has two important implications. First, it means that we can get comparable code performance with codes of shorter length. The second implication is that a lack of significant improvement for increasing lengths may indicate a limitation of the construction. We do note that the cases considered for the near-bent functions included cases where $(l, m) = 1$ and $(l, m) \neq 1$, yet both cases performance did not differ significantly. For the bent case, we considered cases where the corresponding (m, m) vectorial Boolean functions had different conditions for their component functions to be bent.

Figure 2 represents protograph and multi-edge based LDPC codes of length $n = 8192$ and rate $\frac{1}{2}$. Since the length of the codes is six times larger than our largest length codes, we expected significantly better performance when compared to our codes (as coding gain improves with the length of the code [1]). However, for an SNR value of 0.5, Thorpe's codes show BER values in-between 10^{-1} and 10^{-2}. Comparing it to our codes with defining set $\{1, 3\}$ in Figure 7, the observed BER values for lengths 7 and 31 codes are comparable to the multi-edge-type code $(2.37x10^{-1}, 1.10x10^{-1})$ on the other hand, the length 127, 511 and 2047 codes have BER values of $7.81x10^{-2}, 7.02x10^{-2}, 6.82x10^{-2}$, which is comparable to the protograph results. This is notable considering the large difference in code lengths. The performance improvement for larger SNR values falls off when compared to the protograph codes. Our codes remain competitive until an SNR of 0.75. When compared to the example LDPC code we also observe a similar behavior in the comparison, that is, after an SNR of 1.25 the improvement of our codes is slower than the example codes.

Figure 3 shows the performance of a Turbo code with rate $\frac{1}{2}$ decoded through a Maximum a Posteriori (MAP) algorithm and a four-state convolutional code of length 1024. For length 127 our codes have a BER of $8.38x10^{-2}$ for an SNR of 0.25 (and even lower BER for larger lengths). In the figure, we can observe that for the SNR range $[0.2, 0.4]$, the BER values, at the lowest, are around $8x10^{-2}$. For an SNR value of 0.5, the figure shows at best a BER of around $7x10^{-2}$. Our codes match this performance for a length of 511 and improve it for larger lengths. This is notable because 511 is less than half the code length of the code considered by Sah. However, they present a BER of under $6x10^{-2}$ for SNR of 75, while our best-performing code presents a BER of $6.27x10^{-2}$. Our codes remain competitive (or with performance) in the range [0,0.70). Once again, we note that the improvements to the BER seem to plateau and slow down when compared to other codes.

Figure 4 shows an analysis of Quasi-Cyclic LDPC codes constructed in [63] over an AWGN channel. These were constructed by assuming binary phase shift keyed (BPSK) modulation and a maximum of 100 iterations. All these codes had girth above four, with 14 being the largest girth observed. We note that the codes they use are of lengths 1,225 and 13,365. However, our codes have positive coding gains when compared to these codes. For example, Their codes reach a BER of close to $1x10^{-1}$ for an SNR of 0.5. Our length 63 codes reach a BER of $9.68x10^{-2}$ for an SNR of 0.25. This means that for codes that are over 200 times shorter have a relative coding gain of over 0.25 dB. Furthermore, our codes, for length

31, reach a BER of $1.1x10^{-1}$ for the same SNR and even lower for larger lengths. Their codes from examples 15 and 16 do have better performance than our codes for SNR of above 0.75. Once again, the range where our codes remain competitive against other constructions is about $[0, 0.7]$. We do note that these codes have over six times the length of our longest codes. On the other hand, our codes have comparable or better performance than the length 1,225 codes for up to an SNR of 1.

Figure 5 plots the BER and frame error rate (FER) vs. SNR via FPGA simulation computed by JPL's Universal Decoder for Sparse Codes [24]. These codes are constructed through protographs with node degrees of at least 3. The solid curves represent the BER vs. SNR graph. The codes by Divsalar have rates of $\frac{1}{2}, \frac{5}{8}, \frac{3}{4}$ and $\frac{7}{8}$. The rate $\frac{1}{2}$ codes perform worse than our codes for lengths greater than 31 for an SNR in the range $[0, 1]$ (presenting a BER of around $6x10^{-2}$ vs. our length 511 code with a BER of $5.93x10^{-2}$. They present a BER of $7x10^{-2}$ for an SNR of 0.75, while our codes of length 511 reach the same BER for an SNR of 0.5 (a 0.25 dB coding gain). For the larger rate codes, their performance is much worse than our codes (their rate $\frac{5}{8}$ code having a negative coding relative coding gain compared to our codes of about 1 dB compared to our length 511 code).

Finally, Figure 6 shows the performance difference for independently developed decoders applied to an accumulate repeat-4 jagged accumulate (AR4JA) LDPC code. This one has block length 1024 and rates $\frac{4}{5}$. For an SNR of 2.5, the AR4JA code has a BER of $4x10^{-2}$ while our length 255 codes reach this value for an SNR of less than 2.25. This means we provide a coding gain of over 0.25 dB. For the length 1023 codes (which compare directly in terms of length) a BER of $4x10^{-2}$ is reached for an SNR of less than two, meaning a relative coding gain of more than 0.5 dB. All these results show that our codes remain competitive (and even improve performance) of codes analyzed and proposed by other NASA researchers. This analysis lead us to Proposition 1. We conjecture the following for the multiple root case:

Conjecture 8. Codes that meet NASA criteria for next generation channel decoding

Consider the set of near-bent exponents $S := \{j | j \notin C(i) \forall i \in S, i \neq j\}$, α a primitive element in $\mathbb{F}_{2^m}$, $f_i(x) = x^{s_i}, s_i \in S, s_i \neq s_j$ for $i \neq j$. Consider the y*m x n matrix of the form:

$$\text{H'} = \begin{bmatrix} \alpha^{2^m-2} & \alpha^{2^m-3} & \cdots & \alpha^1 & \alpha^0 \\ f_1(\alpha^{2^m-2}) & f_1(\alpha^{2^m-3}) & \cdots & f_1(\alpha^1) & f_1(\alpha^0) \\ f_2(\alpha^{2^m-2}) & f_2(\alpha^{2^m-3}) & \cdots & f_2(\alpha^1) & f_2(\alpha^0) \\ . & . & \cdots & . & . \\ . & . & \cdots & . & . \\ . & . & \cdots & . & . \\ f_{y-1}(\alpha^{2^m-2}) & f_{y-1}(\alpha^{2^m-3}) & \cdots & f_{y-1}(\alpha^1) & f_{y-1}(\alpha^0) \end{bmatrix}$$

Then, the resulting code will be a length $n = 2^m - 1$, k $\geq 2^m - 1 - y * m$ and d(C) ≥ 5. The graph constructed by utilizing this matrix as its adjacency matrix will correspond to a code that meets NASA criteria for next generation channel decoding for $m \geq 7$ for small SNR values.

References

[1] Kenneth Andrews, Dariush Divsalar, Jon Hamkins, and Fabrizio Pollara. Error correcting codes for next generation spacecraft telecommand. In *2013 IEEE Aerospace Conference*, pages 1–8. IEEE, 2013.

[2] Blondeau, Anne Canteaut, and Pascale Charpin. Differential Properties of $x \to x^{2^t-1}$. *IEEE Transactions on Information Theory*, 57(12):8127–8137, 2011.

[3] Christina Boura. Block ciphers and Boolean functions. Available at `https://christinaboura.files.wordpress.com/2019/11/cryptobg2018-boura.pdf`.

[4] Lilya Budaghyan. On inequivalence between known power APN functions. In *Proc. Conference BFCA 2008, Copenhagen*, 2008.

[5] Lilya Budaghyan, Marco Calderini, and Irene Villa. On relations between CCZ-and EA-equivalences. *Cryptography and Communications*, 12(1):85–100, 2020.

[6] Lilya Budaghyan and Claude Carlet. Ccz-equivalence of single and multi output boolean functions. In *Post-proceedings of the 9-th International Conference on Finite Fields and Their Applications Fq*, volume 9, pages 43–54, 2010.

[7] Lilya Budaghyan and Claude Carlet. Ccz-equivalence of bent vectorial functions and related constructions. *Designs, Codes and Cryptography*, 59(1):69–87, 2011.

[8] AR Calderbank and JM Goethals. Three-weight codes and association schemes. *Philips J. Res*, 39(4-5):143–152, 1984.

[9] Gian Paolo Calzolari, Marco Chiani, Franco Chiaraluce, Roberto Garello, and Enrico Paolini. Channel coding for future space missions: New requirements and trends. *Proceedings of the IEEE*, 95(11):2157–2170, 2007.

[10] Anne Canteaut, Pascale Charpin, and Hans Dobbertin. A new characterization of almost bent functions. In *International Workshop on Fast Software Encryption*, pages 186–200. Springer, 1999.

[11] Anne Canteaut and Léo Perrin. On CCZ-equivalence, extended-affine equivalence, and function twisting. *Finite Fields and Their Applications*, 56:209–246, 2019.

[12] Claude Carlet, Pascale Charpin, and Victor Zinoviev. Codes, bent functions and permutations suitable for DES-like cryptosystems. *Designs, Codes and Cryptography*, 15(2):125–156, 1998.

[13] Claude Carlet and Philippe Gaborit. Hyper-bent functions and cyclic codes. *Journal of Combinatorial Theory, Series A*, 113(3):466–482, 2006.

[14] Claude Carlet and Andrew Klapper. Upper bounds on the numbers of resilient functions and of bent functions. In *Proceedings of 23rd Symposium on Information Theory in the Benelux*, 2002.

[15] Claude Carlet and Sihem Mesnager. Four decades of research on bent functions. *Designs, Codes and Cryptography*, 78(1):5–50, 2016.

[16] Ayça Çeşmelioğlu, Wilfried Meidl, and Alexander Pott. Vectorial bent functions and their duals. *Linear Algebra and its Applications*, 548:305–320, 2018.

[17] Pascale Charpin, Enes Pasalic, and Cédric Tavernier. On bent and semi-bent quadratic Boolean functions. *IEEE Transactions on Information Theory*, 51(12):4286–4298, 2005.

[18] Thomas W Cusick and Hans Dobbertin. Some new three-valued crosscorrelation functions for binary m-sequences. *IEEE Transactions on Information Theory*, 42(4):1238–1240, 1996.

[19] Moises Delgado and Heeralal Janwa. On the conjecture on APN functions. *arXiv preprint arXiv:1207.5528*, 2012.

[20] Ulrich Dempwolff. CCZ equivalence of power functions. *Designs, Codes and Cryptography*, 86(3):665–692, 2018.

[21] John F Dillon. Multiplicative difference sets via additive characters. *Designs, Codes and Cryptography*, 17(1-3):225–235, 1999.

[22] John Francis Dillon and Hans Dobbertin. New cyclic difference sets with Singer parameters. *Finite Fields and Their Applications*, 10(3):342–389, 2004.

[23] Cunsheng Ding, Chunlei Li, Nian Li, and Zhengchun Zhou. Three-weight cyclic codes and their weight distributions. *Discrete Mathematics*, 339(2):415–427, 2016.

[24] Dariush Divsalar and Christopher Jones. CTH08-4: protograph LDPC codes with node degrees at least 3. In *IEEE Globecom 2006*, pages 1–5. IEEE, 2006.

[25] Hans Dobbertin. Another proof of Kasami's theorem. *Designs, Codes and Cryptography*, 17(1-3):177–180, 1999.

[26] Eric Férard. A infinite class of Kasami functions that are not APN infinitely often. *Arithmetic, Geometry, Cryptography and Coding Theory*, 686:45, 2017.

[27] Jay Forrester. Boolean functions¶, 2015.

[28] Gary McGuire. APN Functions, APN Codes and S-Boxes. *https* : $//www.cosic.esat.kuleuven.be/nato_arw/slides_participants/McGuire_slides_nato08.pdf$.

[29] Joseph Geraci and Frank Van Bussel. A note on cyclotomic cosets, an algorithm for finding coset representatives and size, and a theorem on the quantum evaluation of weight enumerators for a certain class of cyclic codes. *CoRR*, abs/cs/0703129, 2007.

[30] J.-M. Goethals. *Association Schemes*, pages 243–283. Springer Berlin Heidelberg, Berlin, Heidelberg, 1979.

[31] Robert Gold. Maximal recursive sequences with 3-valued recursive cross-correlation functions (corresp.). *IEEE transactions on Information Theory*, 14(1):154–156, 1968.

[32] Anastasiya Gorodilova. On a remarkable property of APN Gold functions. *IACR Cryptology ePrint Archive*, 2016:286, 2016.

[33] Jon Hamkins. Method of error floor mitigation in low-density parity-check codes, February 18 2014. US Patent 8,656,245.

[34] Richard W Hamming. Error detecting and error correcting codes. *The Bell system technical journal*, 29(2):147–160, 1950.

[35] Gang Han, Yu Yu, Xiangxue Li, Qifeng Zhou, Dong Zheng, and Hui Li. 1-Resilient Boolean Functions on Even Variables with Almost Perfect Algebraic Immunity. *Security and Communication Networks*, 2017, 2017.

[36] Alaa Eldin S Hassan, Moawad Dessouky, AA Elazm, and Mona Shokair. Evaluation of complexity versus performance for turbo code and LDPC under different code rates. *Proc. SPACOMM*, pages 93–98, 2012.

[37] Chris Heegard and Stephen B Wicker. *Turbo coding*, volume 476. Springer Science & Business Media, 2013.

[38] Tor Helleseth. Some results about the cross-correlation function between two maximal linear sequences. *Discrete Mathematics*, 16(3):209–232, 1976.

[39] Fernando Hernando and Gary McGuire. Proof of a conjecture on the sequence of exceptional numbers, classifying cyclic codes and APN functions. *Journal of algebra*, 343(1):78–92, 2011.

[40] Honggang Hu and Dengguo Feng. On quadratic bent functions in polynomial forms. *IEEE transactions on information theory*, 53(7):2610–2615, 2007.

[41] Heeralal Janwa and Richard M Wilson. Hyperplane sections of Fermat varieties in P 3 in char. 2 and some applications to cyclic codes. In *International Symposium on Applied Algebra, Algebraic Algorithms, and Error-Correcting Codes*, pages 180–194. Springer, 1993.

[42] Jørn Justesen and Tom Høholdt. *A course in error-correcting codes*, volume 1. European Mathematical Society, 2004.

[43] Tadao Kasami. Weight distributions of bose-chaudhuri-hocquenghem codes. *Coordinated Science Laboratory Report no. R-317*, 1966.

[44] Tadao Kasami. The weight enumerators for several classes of subcodes of the 2nd order binary Reed-Muller codes. *Information and Control*, 18(4):369–394, 1971.

[45] Khoongming Khoo, Guang Gong, and Douglas R Stinson. A new characterization of semi-bent and bent functions on finite fields. *Designs, Codes and Cryptography*, 38(2):279–295, 2006.

[46] Gohar M Kyureghyan and Valentin Suder. On inverses of APN exponents. In *2012 IEEE International Symposium on Information Theory Proceedings*, pages 1207–1211. IEEE, 2012.

[47] Nian Li. Some Recent Progress in the Applications of Niho Exponents. In *Conference Hubei China*, pages 0–35. Faculty of Mathematics and Statistics, 2017.

[48] WenPing Ma, Moonho Lee, and Futai Zhang. A new class of bent functions. *IEICE Transactions on Fundamentals of Electronics, Communications and Computer Sciences*, 88(7):2039–2040, 2005.

[49] Rusydi Makarim and Yann Laigle-Chapui. Boolean functions. `https://doc.sagemath.org/html/en/reference/cryptography/sage/crypto/boolean_function.htm`, 2016. Accessed: 2019-12-7.

[50] Robert J McEliece and Laif Swanson. Reed-Solomon codes and the exploration of the solar system. 1994.

[51] Gary McGuire. On certain 3-weight cyclic codes having symmetric weights and a conjecture of helleseth. In *Sequences and their applications*, pages 281–295. Springer, 2002.

[52] Gary Mcguire. On three weights in cyclic codes with two zeros. *Finite Fields and Their Applications*, 10(1):97–104, 2004.

[53] Raford Neal. Software for Low Density Parity Check Codes. `https://www.cs.toronto.edu/~radford/ftp/LDPC-2006-02-08/index.html`, 2006. Accesed 2021-01-24.

[54] Yoji Niho. Multi-valued cross-correlation functions between two maximal linear recursive sequences. Technical report, UNIVERSITY OF SOUTHERN CALIFORNIA LOS ANGELES ELECTRONIC SCIENCES LAB, 1972.

[55] Kaisa Nyberg. Perfect nonlinear S-boxes. In *Workshop on the Theory and Application of of Cryptographic Techniques*, pages 378–386. Springer, 1991.

[56] Kaisa Nyberg. S-boxes and round functions with controllable linearity and differential uniformity. In *International Workshop on Fast Software Encryption*, pages 111–130. Springer, 1994.

[57] François Rodier. Borne sur le degré des polynômes presque parfaitement non-linéaires. In *Arithmetic, geometry, cryptography and coding theory*, volume 487 of *Contemp. Math.*, pages 169–181. Amer. Math. Soc., Providence, RI, 2009.

[58] David Roe, Jean-Pierre Flori, and Peter Bruin. Routines for Conway and pseudo-Conway polynomials. `https://doc.sagemath.org/html/en/reference/finite_rings/sage/rings/finite_rings/conway_polynomials.html?highlight=conway#module-sage.rings.finite_rings.conway_polynomials`, 2020. Accessed: 2020-02-10.

[59] Oscar S Rothaus. On "bent" functions. *Journal of Combinatorial Theory, Series A*, 20(3):300–305, 1976.

[60] William E Ryan et al. An introduction to LDPC codes, 2004.

[61] Dhaneshwar Sah. Iterative Decoding of Turbo Codes. *Journal of Advanced College of Engineering and Management*, 3:15–30, 2017.

[62] Amin Shokrollahi. LDPC codes: An introduction. In *Coding, cryptography and combinatorics*, pages 85–110. Springer, 2004.

[63] Roxana Smarandache and David GM Mitchell. Necessary and Sufficient Girth Conditions for Tanner Graphs of Quasi-Cyclic LDPC Codes. *arXiv preprint arXiv:2105.03462*, 2021.

[64] Bo Sun. *On Classification and Some Properties of APN Functions*. PhD thesis, Skipnes Kommunikasjon, 2018.

[65] Yuriy Tarannikov, Peter Korolev, and Anton Botev. Autocorrelation coefficients and correlation immunity of Boolean functions. In *International Conference on the Theory and Application of Cryptology and Information Security*, pages 460–479. Springer, 2001.

[66] Jeremy Thorpe. Low-density parity-check (LDPC) codes constructed from protographs. *IPN progress report*, 42(154):42–154, 2003.

[67] Jose Velazquez and Heeralal Janwa. Bent and Near-Bent Function Construction and 2-Error-Correcting Codes. *Springer Proceedings in Mathematics & Statistics*, 2021.

[68] Stephen B Wicker and Vijay K Bhargava. *Reed-Solomon codes and their applications*. John Wiley & Sons, 1999.

[69] DCK Wong. Cyclotomic Cosets, Codes and Secret Sharing. *Malaysian Journal of Mathematical Sciences*, 11:59–73, 2017.

[70] DCK Wong. Cyclotomic Cosets, Codes and Secret Sharing. 2017.

[71] Satoshi Yoshiara. Equivalences of power APN functions with power or quadratic APN functions. *Journal of Algebraic Combinatorics*, 44(3):561–585, 2016.

[72] Amr M. Youssef and Guang Gong. Hyper-bent functions. In Birgit Pfitzmann, editor, *Advances in Cryptology — EUROCRYPT 2001*, pages 406–419, Berlin, Heidelberg, 2001. Springer Berlin Heidelberg.

[73] Xiangyong Zeng, Jinyong Shan, and Lei Hu. A triple-error-correcting cyclic code from the Gold and Kasami–Welch APN power functions. *Finite Fields and Their Applications*, 18(1):70–92, 2012.

[74] Xian-Mo Zhang and Yuliang Zheng. GAC—the criterion for global avalanche characteristics of cryptographic functions. In *J. UCS The Journal of Universal Computer Science*, pages 320–337. Springer, 1996.

8. Appendix:Tables

Variables	l	Case	Prediction	Computational Results
4	1,3	1	M3,M3	M3,M3
4	2	3	M5	M5
6	1,5	1	M3,M3	M3,M3
6	2,4	2	M1,M1	M1,M1
6	3	3	M9	M9
8	1,3,5,7	1	M3,M3,M3,M3	M3,M3,M3,M3
8	2,4,6	3	M5,M17,M(divisor of 65)	M5,M17,M5
10	1,3,7,9	1	M3,M3,M3,M3	M3,M3,M3,M3
10	2,4,6,8	2	M1,M1,M1,M1	M1,M1,M1,M1
10	5	3	M33	M33

Table 4: Gold bent functions exceptions test 4-10 variables [67].

Variables	l	Case	Prediction	Computational Results
12	1,5,7,11	1	M3,M3,M3,M3	M3,M3,M3,M3
12	2,3,6,9,10	3	M5,Mi,M65,Mi,Mi	M5,M9,M65,M9,M5
12	4,8	2	M1,M1	M1,M1
14	1,3,5,9,11 13	1	M3,M3,M3,M3,M3 M3	M3,M3,M3,M3,M3 M3
14	2,4,6,8,10 12	2	M1,M1,M1,M1,M1 M1	M1,M1,M1,M1,M1 M1
14	7	3	M129	M129

Table 5: Gold bent functions exceptions test 12-14 variables. We define i as a divisor of a Gold exponent [67].

Variables	l	Case	Prediction	Computational Results
16	1,3,5,7,9 11,13,15	1	M3,M3,M3,M3,M3 M3,M3	M3,M3,M3,M3,M3 M3,M3
16	2,4,6,8,10 12,14	3	M5,M17,Mi,Mi,Mi Mi,Mi	M5,M17,M5,M257,M5 M17,M5
18	1,5,7,11,13 17	1	M3,M3,M3,M3,M3 M3	M3,M3,M3,M3,M3 M3
18	2,4,6,8,10 12,14,16	2	M1,M1,M1,M1,M1 M1,M1,M1	M1,M1,M1,M1,M1 M1,M1,M1
18	3,9,15	3	Mi,M513,Mi	M9,M513,M9
20	1,3,7,9,11 13	1	M3,M3,M3,M3,M3 M3	M3,M3,M3,M3,M3 M3

20	2,5,6,10,14	3	M5,Mi,Mi,M1025,Mi	M5,M33,M5,M1025,M5
20	4,8,12	2	M1,M1,M1	M1,M1,M1
22	1,3,5,7,9	1	M3,M3,M3,M3,M3	M3,M3,M3,M3,M3
22	2,4,6,8,10 12	2	M1,M1,M1,M1,M1 M1	M1,M1,M1,M1,M1 M1
22	11	3	M2049	M2049
24	1,5,7,11,13	1	M3,M3,M3,M3,M3	M3,M3,M3,M3,M3
24	2,3,4,6,9 10,12	3	M5,Mi,M17,Mi,Mi Mi,Mi	M5,M9,M17,M65,M9 M5,M4097
24	8	2	M1	M1

Table 6: Gold bent function exceptions partial test 16-24 variables [67].

Variables	Exponent (d)	Condition	Is it Bent?
6	3	(l,m) = 1, $i \notin M3$	Bent
6	993	(l,m) = 1, $i \notin M3$	Bent
12	3	(l,m) = 1, $i \notin M3$	Bent
12	993	(l,m) = 1, $i \notin M3$	Bent
12	16257	(l,m) = 1, $i \notin M3$	Bent
12	4192257	(l,m) = 1, $i \notin M3$	Bent

Table 7: Kasami-Welch Boolean bent function computations 6,12 variables. The functions are of the form $Tr(\alpha^i x^d)$ [67].

Length	Defining Set	Distribution Type	Distribution
15	{1, 5 }	2	6 8 10
31	{1, 3 }	1	12 16 20
31	{1, 5 }	1	12 16 20
31	{1, 7 }	1	12 16 20
31	{1, 11 }	1	12 16 20
63	{1, 5 }	1	24 32 40
63	{1, 9 }	2	28 32 36
63	{1, 13 }	1	24 32 40
127	{1, 3 }	1	56 64 72
127	{1, 5 }	1	56 64 72
127	{1, 9 }	1	56 64 72
127	{1, 11 }	1	56 64 72
127	{1, 13 }	1	56 64 72
127	{1, 15 }		

127	{1, 23 }	1	56 64 72
127	{1, 27 }		
127	{1, 29 }	1	56 64 72
127	{1, 43 }		
255	{1, 17 }	2	120 128 136
511	{1, 3 }	1	240 256 272
511	{1, 5 }	1	240 256 272
511	{1, 9 }	1	224 256 288
511	{1, 13 }	1	240 256 272
511	{1, 17 }	1	240 256 272
511	{1, 19 }	1	240 256 272
511	{1, 27 }		
511	{1, 31 }		
511	{1, 47 }	1	240 256 272
511	{1, 57 }	1	240 256 272
511	{1, 59 }		
511	{1, 87 }		
511	{1, 103 }		
511	{1, 171 }		
1023	{1, 5 }	1	480 512 544
1023	{1, 13 }	1	480 512 544
1023	{1, 17 }	1	480 512 544
1023	{1, 25 }		
1023	{1, 33 }	2	496 512 528
1023	{1, 41 }	1	480 512 544
1023	{1, 49 }	1	480 512 544
1023	{1, 79 }	1	480 512 544
1023	{1, 107 }		
1023	{1, 181 }		
1023	{1, 205 }		
2047	{1, 3 }	1	992 1024 1056
2047	{1, 5 }	1	992 1024 1056
2047	{1, 9 }	1	992 1024 1056
2047	{1, 13 }	1	992 1024 1056
2047	{1, 17 }	1	992 1024 1056
2047	{1, 33 }	1	992 1024 1056
2047	{1, 35 }	1	992 1024 1056
2047	{1, 43 }		
2047	{1, 57 }	1	992 1024 1056
2047	{1, 63 }		
2047	{1, 95 }	1	992 1024 1056

2047	{1, 107 }		
2047	{1, 117 }		
2047	{1, 143 }	1	992 1024 1056
2047	{1, 151 }		
2047	{1, 231 }		
2047	{1, 249 }	1	992 1024 1056
2047	{1, 315 }		
2047	{1, 365 }		
2047	{1, 411 }		
2047	{1, 413 }		
2047	{1, 683 }		
4095	{1, 17 }	1	1920 2048 2176
4095	{1, 65 }	2	2016 2048 2080
4095	{1, 241 }	1	1984 2048 2112
8191	{1, 3 }	1	4032 4096 4160
8191	{1, 5 }	1	4032 4096 4160
8191	{1, 9 }	1	4032 4096 4160
8191	{1, 13 }	1	4032 4096 4160
8191	{1, 17 }	1	4032 4096 4160
8191	{1, 33 }	1	4032 4096 4160
8191	{1, 57 }	1	4032 4096 4160
8191	{1, 65 }	1	4032 4096 4160
8191	{1, 67 }	1	4032 4096 4160
8191	{1, 71 }	1	4032 4096 4160
8191	{1, 127 }		
8191	{1, 171 }		
8191	{1, 191 }	1	4032 4096 4160
8191	{1, 241 }	1	4032 4096 4160
8191	{1, 287 }	1	4032 4096 4160
8191	{1, 347 }		
8191	{1, 367 }		
8191	{1, 635 }		
8191	{1, 723 }		
8191	{1, 911 }		
8191	{1, 1243 }		
8191	{1, 1245 }		
8191	{1, 1453 }		
8191	{1, 1639 }		
8191	{1, 1691 }		
8191	{1, 2731 }		

Table 8: Table with codes that had three nonzero weight dual codes classified by Li's distribution types.

Length of Code	Non theorem Cyclic Code Defining set	Equivalent Code Defining Set	Meet Theorems?	Distribution Type
127	{1, 15}	{1, 9}	Yes	1
127	{1, 27}	{1, 5}	Yes	1
127	{1, 43}	{1, 3}	Yes	1
511	{1, 27}	{1, 19}	Yes	1
511	{1, 31}	{1, 17}	Yes	1
511	{1, 59}	{1, 13}	Yes	1
511	{1, 87}	{1, 47}	Yes	1
511	{1, 103}	{1, 5}	Yes	1
511	{1, 171}	{1, 3}	Yes	1
1023	{1, 25}	{1, 41}	Yes	1
1023	{1, 107}	{1, 49}	Yes	1
1023	{1, 181}	{1, 17}	Yes	1
1023	{1, 205}	{1, 5}	Yes	1
2047	{1, 43}	{1, 143}	Yes	1
2047	{1, 63}	{1, 33}	Yes	1
2047	{1, 107}	{1, 249}	Yes	1
2047	{1, 117}	{1, 35}	Yes	1
2047	{1, 151}	{1, 95}	Yes	1
2047	{1, 231}	{1, 9}	Yes	1
2047	{1, 315}	{1, 13}	Yes	1
2047	{1, 365}	{1, 17}	Yes	1
2047	{1, 411}	{1, 5}	Yes	1
2047	{1, 413}	{1, 57}	Yes	1
2047	{1, 683}	{1, 3}	Yes	1
8191	{1, 127}	{1, 65}	Yes	1
8191	{1, 171}	{1, 241}	Yes	1
8191	{1, 347}	{1, 71}	Yes	1
8191	{1, 367}	{1, 67}	Yes	1
8191	{1, 635}	{1, 13}	Yes	1
8191	{1, 723}	{1, 57}	Yes	1
8191	{1, 911}	{1, 9}	Yes	1
8191	{1, 1243}	{1, 33}	Yes	1
8191	{1, 1245}	{1, 191}	Yes	1
8191	{1, 1453}	{1, 17}	Yes	1
8191	{1, 1639}	{1, 5}	Yes	1
8191	{1, 1691}	{1, 287}	Yes	1
8191	{1, 2731}	{1, 3}	Yes	1

TABLE 9. Table with the three nonzero weight cyclic codes which did not correspond to any of the theorems and their equivalent codes.

n	k	d(C)	Defining Set	APN Exponent		F(x)	Near-Bent?	Bent?
15	7	5	{1, 3}	Gold	Kasami-Welch	X^3		No
31	21	5	{1, 3}	Gold	Kasami-Welch	X^3	Yes	
31	21	5	{1, 5}	Gold	Niho even	X^5	Yes	
31	21	5	{1, 7}	Welch	Nyberg	X^7	Yes	
31	21	5	{1, 11}	Kasami-Welch		X^11	Yes	
31	21	5	{1, 15}	Inverse	Dobbertin	X^15	No	

63	51	5	{1, 3}	Gold	Kasami-Welch	X^3		No
127	113	5	{1, 3}	Gold	Kasami-Welch	X^3	Yes	
127	113	5	{1, 5}	Gold		X^5	Yes	
127	113	5	{1, 9}	Gold		X^9	Yes	
127	113	5	{1, 11}	Welch		X^11	Yes	
127	113	5	{1, 13}	Kasami-Welch		X^13	Yes	
127	113	5	{1, 15}			X^15	Yes	
127	113	5	{1, 23}	Kasami-Welch		X^23	Yes	
127	113	5	{1, 27}			X^27	Yes	
127	113	5	{1, 29}	Niho odd		X^29	Yes	
127	113	5	{1, 43}			X^43	Yes	
127	113	5	{1, 63}	Inverse	Nyberg	X^63	No	
255	239	5	{1, 3}	Gold	Kasami-Welch	X^3		No
255	239	5	{1, 9}	Gold		X^9		No
255	239	5	{1, 15}			X^15		Yes
255	239	5	{1, 39}	Kasami-Welch		X^39		No
255	239	5	{1, 45}			X^45		Yes
511	493	5	{1, 3}	Gold	Kasami-Welch	X^3	Yes	
511	493	5	{1, 5}	Gold		X^5	Yes	
511	493	5	{1, 13}	Kasami-Welch		X^13	Yes	
511	493	5	{1, 17}	Gold		X^17	Yes	
511	493	5	{1, 19}	Welch	Niho even	X^19	Yes	
511	493	5	{1, 27}			X^27	Yes	
511	493	5	{1, 31}			X^31	Yes	
511	493	5	{1, 47}	Kasami-Welch		X^47	Yes	
511	493	5	{1, 59}			X^59	Yes	
511	493	5	{1, 63}			X^63	No	
511	493	5	{1, 87}			X^87	Yes	
511	493	5	{1, 103}			X^103	Yes	
511	493	5	{1, 171}			X^171	Yes	
511	493	5	{1, 255}	Inverse		X^255	No	
1023	1003	5	{1, 3}	Gold	Kasami-Welch	X^3		No
1023	1003	5	{1, 9}	Gold		X^9		No
1023	1003	5	{1, 57}	Kasami-Welch		X^57		No
1023	1003	5	{1, 213}	Dobbertin		X^213		No
1023	1003	5	{1, 237}			X^237		No
2047	2025	5	{1, 3}	Gold	Kasami-Welch	X^3	Yes	
2047	2025	5	{1, 5}	Gold		X^5	Yes	
2047	2025	5	{1, 9}	Gold		X^9	Yes	
2047	2025	5	{1, 13}	Kasami-Welch		X^13	Yes	
2047	2025	5	{1, 17}	Gold		X^17	Yes	
2047	2025	5	{1, 33}	Gold		X^33	Yes	
2047	2025	5	{1, 35}	Welch		X^35	Yes	
2047	2025	5	{1, 43}			X^43	Yes	
2047	2025	5	{1, 57}	Kasami-Welch		X^57	Yes	
2047	2025	5	{1, 63}			X^63	Yes	
2047	2025	5	{1, 95}	Kasami-Welch		X^95	Yes	
2047	2025	5	{1, 107}			X^107	Yes	

2047	2025	5	{1, 117}			X^117	Yes	
2047	2025	5	{1, 143}	Kasami-Welch		X^143	Yes	
2047	2025	5	{1, 151}			X^151	Yes	
2047	2025	5	{1, 231}			X^231	Yes	
2047	2025	5	{1, 249}	Niho odd		X^249	Yes	
2047	2025	5	{1, 255}			X^255	No	
2047	2025	5	{1, 315}			X^315	Yes	
2047	2025	5	{1, 365}			X^365	Yes	
2047	2025	5	{1, 411}			X^411	Yes	
2047	2025	5	{1, 413}			X^413	Yes	
2047	2025	5	{1, 683}			X^683	Yes	
2047	2025	5	{1, 1023}	Inverse		X^1023	No	

Table 10: Cyclic-codes with minimum distance five from the 'guava' package in SAGE [67].

Code	d(C)	Defining Set	Non-zero Weights of $C^{\perp}$
{7, 1}	7	{1, 3}	3
{31, 21}	5	{1, 3}, {1, 5}, {1, 7}, {1, 11}	3
{31, 21}	5	{1, 15}	6
{127, 113}	5	{1, 3}, {1, 5}, {1, 9}, {1, 11} {1, 13}, {1, 15}, {1, 23}, {1, 29}	3
{127, 113}	5	{1, 63}	11
{511, 493}	5	{1, 3}, {1, 5},{1, 9},{1, 13} {1, 17},{1, 19},{1, 47},{1, 57}	3
{511, 493}	4	{1, 63}	15
{511, 493}	5	{1, 255}	23
{2047, 2025}	5	{1, 3},{1, 5},{1, 9},{1, 13} {1, 17},{1, 33},{1, 35},{1, 57} {1, 95},{1, 143},{1, 249}	3
{2047, 2025}	4	{1, 255}	16
{2047, 2025}	5	{1, 1023}	45

Table 11: Minimum distance computations with alternative method to SAGE.

n	k	Defining Set	$f(x)$	APN?	Equivalence
127	113	{1, 15}	X^15	Yes	f(x) = x^{15} ≡ g(x) = x^9 , Gold
127	113	{1, 27}	X^27	Yes	f(x) = x^{27} ≡ g(x) = x^5 , Gold
127	113	{1, 43}	X^43	Yes	f(x) = x^{43} ≡ g(x) = x^3 , Gold
511	493	{1, 27}	X^27	Yes	f(x) = x^{27} ≡ g(x) = x^{19} , Niho and Welch
511	493	{1, 31}	X^31	Yes	f(x) = x^{31} ≡ g(x) = x^{17} , Gold
511	493	{1, 59}	X^59	Yes	f(x) = x^{59} ≡ g(x) = x^{13} , Kasami-Welch

511	493	{1, 63}	X^63	No	No Equivalence
511	493	{1, 87}	X^87	Yes	f(x) = x^{87} ≡ g(x) = x^{47} , Kasami-Welch
511	493	{1, 103}	X^103	Yes	f(x) = x^{103} ≡ g(x) = x^{5} , Gold
511	493	{1, 171}	X^171	Yes	f(x) = x^{171} ≡ g(x) = x^{3} , Gold
2047	2025	{1, 43}	X^43	Yes	f(x) = x^{43} ≡ g(x) = x^{143} , Kasami-Welch
2047	2025	{1, 63}	X^63	Yes	f(x) = x^{63} ≡ g(x) = x^{33} , Gold
2047	2025	{1, 107}	X^107	Yes	f(x) = x^{107} ≡ g(x) = x^{249} , Niho
2047	2025	{1, 117}	X^117	Yes	f(x) = x^{117} ≡ g(x) = x^{35} , Welch
2047	2025	{1, 151}	X^151	Yes	f(x) = x^{151} ≡ g(x) = x^{95}, Kasami-Welch
2047	2025	{1, 231}	X^231	Yes	f(x) = x^{231} ≡ g(x) = x^{9} , Gold
2047	2025	{1, 255}	X^255	No	f(x) = x^{255} ≡ g(x) = x^{731} , not APN
2047	2025	{1, 315}	X^315	Yes	f(x) = x^{315} ≡ g(x) = x^{13}, Kasami-Welch
2047	2025	{1, 365}	X^365	Yes	f(x) = x^{365} ≡ g(x) = x^{17}, Gold
2047	2025	{1, 411}	X^411	Yes	f(x) = x^{411} ≡ g(x) = x^{5}, Gold
2047	2025	{1, 413}	X^413	Yes	f(x) = x^{413} ≡ g(x) = x^{57}, Kasami-Welch
2047	2025	{1, 683}	X^683	Yes	f(x) = x^{683} ≡ g(x) = x^{3}, Gold

Table 13: Gold Boolean function equality in four variables.

Equal Functions				Number of Equal Functions	Bent-ness
$Tr(a^1x^3)$	$Tr(a^8x^9)$			2	Bent
$Tr(a^2x^3)$	$Tr(a^1x^9)$			2	Bent
$Tr(a^3x^3)$	$Tr(a^9x^9)$			2	Not Bent
$Tr(a^4x^3)$	$Tr(a^2x^9)$			2	Bent
$Tr(a^5x^3)$	$Tr(a^{10}x^9)$			2	Bent
$Tr(a^6x^3)$	$Tr(a^3x^9)$			2	Not Bent
$Tr(a^7x^3)$	$Tr(a^{11}x^9)$			2	Bent
$Tr(a^8x^3)$	$Tr(a^4x^9)$			2	Bent
$Tr(a^9x^3)$	$Tr(a^{12}x^9)$			2	Not Bent
$Tr(a^{10}x^3)$	$Tr(a^5x^9)$			2	Bent
$Tr(a^{11}x^3)$	$Tr(a^{13}x^9)$			2	Bent
$Tr(a^{12}x^3)$	$Tr(a^6x^9)$			2	Not Bent
$Tr(a^{13}x^3)$	$Tr(a^{14}x^9)$			2	Bent
$Tr(a^{14}x^3)$	$Tr(a^7x^9)$			2	Bent
$Tr(a^1x^5)$	$Tr(a^2x^5)$	$Tr(a^4x^5)$	$Tr(a^8x^5)$	4	Bent
$Tr(a^2x^5)$	$Tr(a^1x^5)$	$Tr(a^4x^5)$	$Tr(a^8x^5)$	4	Bent
$Tr(a^3x^5)$	$Tr(a^{11}x^5)$	$Tr(a^{12}x^5)$	$Tr(a^{14}x^5)$	4	Bent
$Tr(a^4x^5)$	$Tr(a^1x^5)$	$Tr(a^2x^5)$	$Tr(a^8x^5)$	4	Bent
$Tr(a^5x^5)$	$Tr(a^{10}x^5)$			2	Not Bent
$Tr(a^6x^5)$	$Tr(a^7x^5)$	$Tr(a^9x^5)$	$Tr(a^{13}x^5)$	4	Bent

$Tr(a^7x^5)$	$Tr(a^6x^5)$	$Tr(a^9x^5)$	$Tr(a^{13}x^5)$	4	Bent
$Tr(a^8x^5)$	$Tr(a^1x^5)$	$Tr(a^2x^5)$	$Tr(a^4x^5)$	4	Bent
$Tr(a^9x^5)$	$Tr(a^6x^5)$	$Tr(a^7x^5)$	$Tr(a^{13}x^5)$	4	Bent
$Tr(a^{10}x^5)$	$Tr(a^5x^5)$			2	Not Bent
$Tr(a^{11}x^5)$	$Tr(a^3x^5)$	$Tr(a^{12}x^5)$	$Tr(a^{14}x^5)$	4	Bent
$Tr(a^{12}x^5)$	$Tr(a^3x^5)$	$Tr(a^{11}x^5)$	$Tr(a^{14}x^5)$	4	Bent
$Tr(a^{13}x^5)$	$Tr(a^6x^5)$	$Tr(a^7x^5)$	$Tr(a^9x^5)$	4	Bent
$Tr(a^{14}x^5)$	$Tr(a^3x^5)$	$Tr(a^{11}x^5)$	$Tr(a^{12}x^5)$	4	Bent

Table 13: Cyclotomic equivalence analysis of "new" two-error-correcting codes functions [67].

Variables	Coset Representative	Exponent	Exception	Function
4	0	0	M1	$\mathrm{Tr}(\alpha^i x^0)$
4	1	8	M1	$\mathrm{Tr}(\alpha^i x^8)$
4	1	1	M1	$\mathrm{Tr}(\alpha^i x^1)$
4	1	2	M1	$\mathrm{Tr}(\alpha^i x^2)$
4	1	4	M1	$\mathrm{Tr}(\alpha^i x^4)$
4	3	9	M3	$\mathrm{Tr}(\alpha^i x^9)$
4	3	3	M3	$\mathrm{Tr}(\alpha^i x^3)$
4	3	12	M3	$\mathrm{Tr}(\alpha^i x^{12})$
4	3	6	M3	$\mathrm{Tr}(\alpha^i x^6)$
4	5	10	M5	$\mathrm{Tr}(\alpha^i x^{10})$
4	5	5	M5	$\mathrm{Tr}(\alpha^i x^5)$
4	7	11	M1	$\mathrm{Tr}(\alpha^i x^{11})$
4	7	13	M1	$\mathrm{Tr}(\alpha^i x^{13})$
4	7	14	M1	$\mathrm{Tr}(\alpha^i x^{14})$
4	7	7	M1	$\mathrm{Tr}(\alpha^i x^7)$
6	0	0	M1	$\mathrm{Tr}(\alpha^i x^0)$
6	1	32	M1	$\mathrm{Tr}(\alpha^i x^{32})$
6	1	1	M1	$\mathrm{Tr}(\alpha^i x^1)$
6	1	2	M1	$\mathrm{Tr}(\alpha^i x^2)$
6	1	4	M1	$\mathrm{Tr}(\alpha^i x^4)$
6	1	8	M1	$\mathrm{Tr}(\alpha^i x^8)$
6	1	16	M1	$\mathrm{Tr}(\alpha^i x^{16})$
6	3	33	M3	$\mathrm{Tr}(\alpha^i x^{33})$
6	3	3	M3	$\mathrm{Tr}(\alpha^i x^3)$
6	3	6	M3	$\mathrm{Tr}(\alpha^i x^6)$
6	3	12	M3	$\mathrm{Tr}(\alpha^i x^{12})$
6	3	48	M3	$\mathrm{Tr}(\alpha^i x^{48})$
6	3	24	M3	$\mathrm{Tr}(\alpha^i x^{24})$

6	5	34	M1	$\mathrm{Tr}(\alpha^i x^{34})$
6	5	5	M1	$\mathrm{Tr}(\alpha^i x^{5})$
6	5	40	M1	$\mathrm{Tr}(\alpha^i x^{40})$
6	5	10	M1	$\mathrm{Tr}(\alpha^i x^{10})$
6	5	17	M1	$\mathrm{Tr}(\alpha^i x^{17})$
6	5	20	M1	$\mathrm{Tr}(\alpha^i x^{20})$
6	7	35	M1	$\mathrm{Tr}(\alpha^i x^{35})$
6	7	7	M1	$\mathrm{Tr}(\alpha^i x^{7})$
6	7	14	M1	$\mathrm{Tr}(\alpha^i x^{14})$
6	7	49	M1	$\mathrm{Tr}(\alpha^i x^{49})$
6	7	56	M1	$\mathrm{Tr}(\alpha^i x^{56})$
6	7	28	M1	$\mathrm{Tr}(\alpha^i x^{28})$
6	9	9	M9	$\mathrm{Tr}(\alpha^i x^{9})$
6	9	18	M9	$\mathrm{Tr}(\alpha^i x^{18})$
6	9	36	M9	$\mathrm{Tr}(\alpha^i x^{36})$
6	11	37	M1	$\mathrm{Tr}(\alpha^i x^{37})$
6	11	11	M1	$\mathrm{Tr}(\alpha^i x^{11})$
6	11	44	M1	$\mathrm{Tr}(\alpha^i x^{44})$
6	11	50	M1	$\mathrm{Tr}(\alpha^i x^{50})$
6	11	22	M1	$\mathrm{Tr}(\alpha^i x^{22})$
6	11	25	M1	$\mathrm{Tr}(\alpha^i x^{25})$
6	13	38	M1	$\mathrm{Tr}(\alpha^i x^{38})$
6	13	41	M1	$\mathrm{Tr}(\alpha^i x^{41})$
6	13	13	M1	$\mathrm{Tr}(\alpha^i x^{13})$
6	13	19	M1	$\mathrm{Tr}(\alpha^i x^{19})$
6	13	52	M1	$\mathrm{Tr}(\alpha^i x^{52})$
6	13	26	M1	$\mathrm{Tr}(\alpha^i x^{26})$
6	15	39	M1	$\mathrm{Tr}(\alpha^i x^{39})$
6	15	15	M1	$\mathrm{Tr}(\alpha^i x^{15})$
6	15	51	M1	$\mathrm{Tr}(\alpha^i x^{51})$
6	15	57	M1	$\mathrm{Tr}(\alpha^i x^{57})$
6	15	60	M1	$\mathrm{Tr}(\alpha^i x^{60})$
6	15	30	M1	$\mathrm{Tr}(\alpha^i x^{30})$
6	21	42	M1	$\mathrm{Tr}(\alpha^i x^{42})$
6	21	21	M1	$\mathrm{Tr}(\alpha^i x^{21})$
6	23	43	M1	$\mathrm{Tr}(\alpha^i x^{43})$
6	23	46	M1	$\mathrm{Tr}(\alpha^i x^{46})$
6	23	53	M1	$\mathrm{Tr}(\alpha^i x^{53})$
6	23	23	M1	$\mathrm{Tr}(\alpha^i x^{23})$
6	23	58	M1	$\mathrm{Tr}(\alpha^i x^{58})$
6	23	29	M1	$\mathrm{Tr}(\alpha^i x^{29})$

6	27	27	M1	$\mathrm{Tr}(\alpha^i x^{27})$
6	27	45	M1	$\mathrm{Tr}(\alpha^i x^{45})$
6	27	54	M1	$\mathrm{Tr}(\alpha^i x^{54})$
6	31	47	M1	$\mathrm{Tr}(\alpha^i x^{47})$
6	31	55	M1	$\mathrm{Tr}(\alpha^i x^{55})$
6	31	59	M1	$\mathrm{Tr}(\alpha^i x^{59})$
6	31	61	M1	$\mathrm{Tr}(\alpha^i x^{61})$
6	31	62	M1	$\mathrm{Tr}(\alpha^i x^{62})$
6	31	31	M1	$\mathrm{Tr}(\alpha^i x^{31})$

Table 14: Non-bent function conditions for Boolean functions of the form $Tr(\alpha^i x^{C(d)})$.

L	Exponent	Mod 15	Mod 63	Mod 255	Mod 1023
1	3	[3,6,12,9]	[3,6,12,24,48,33]	[3,6,12,24,48,96,192,129]	[3, 6, 12, 24, 48, 96, 192, 384, 513, 768]
2	5	[5,10]	[5,10,20,40,17,34]	[5,10,20,40,80,160,65,130]	[5, 10, 20, 40, 80, 160, 257, 320, 514, 640]
3	9	[3,6,12,9]	[9,18,36]	[9,18,36,72,144,33,66,132]	[9, 18, 36, 72, 129, 144, 258, 288, 516, 576]
4	17	[1,2,4,8]	[5,10,20,40,17,34]	[17,34,68,136]	[17, 34, 65, 68, 130, 136, 260, 272, 520, 544]
5	33	[3,6,12,9]	[3,6,12,24,48,33]	[9,18,36,72,144,33,66,132]	[33, 66, 132, 264, 528]
6	65	[5,10]	[1,2,4,8,16,32]	[5,10,20,40,80,160,65,130]	[17, 34, 65, 68, 130, 136, 260, 272, 520, 544]
7	129	[3,6,12,9]	[3,6,12,24,48,33]	[3,6,12,24,48,96,192,129]	[9, 18, 36, 72, 129, 144, 258, 288, 516, 576]
8	257	[1,2,4,8]	[5,10,20,40,17,34]	[1,2,4,8,16,32,64,128]	[5, 10, 20, 40, 80, 160, 257, 320, 514, 640]
9	513	[3,6,12,9]	[9,18,36]	[3,6,12,24,48,96,192,129]	[3, 6, 12, 24, 48, 96, 192, 384, 513, 768]

TABLE 15. Gold exponents cyclotomic cosets (mod $2^m - 1$) for m even.

L	Exponent	Mod 7	Mod 31	Mod 127	Mod511
1	3	[3,6,5]	[3,6,12,24,17]	[3,6,12,24,48,96,65]	[3,6,12,24,48,96,192,384,257]
2	5	[3,6,5]	[5,10,20,9,18]	[5,10,20,40,80,33,66]	[5,10,20,40,80,160,320,129,258]
3	9	[1,2,4]	[5,10,20,9,18]	[9,18,36,72,17,34,68]	[9,18,36,72,144,288,65,130,260]
4	17	[3,6,5]	[3,6,12,24,17]	[9,18,36,72,17,34,68]	[17,34,68,136,272,33,66,132,264]
5	33	[3,6,5]	[1,2,4,8,16]	[5,10,20,40,80,33,66]	[17,34,68,136,272,33,66,132,264]
6	65	[1,2,4]	[3,6,12,24,17]	[3,6,12,24,48,96,65]	[9,18,36,72,144,288,65,130,260]
7	129	[3,6,5]	[5,10,20,9,18]	[1,2,4,8,16,32,64]	[5,10,20,40,80,160,320,129,258]
8	257	[3,6,5]	[5,10,20,9,18]	[3,6,12,24,48,96,65]	[3,6,12,24,48,96,192,384,257]

TABLE 16. Gold exponents cyclotomic cosets (mod $2^m - 1$) for m odd.

L	Exponent	Mod 15	Mod 63	Mod 255	Mod 1023
1	3	[3,6,12,9]	[3,6,12,24,48,33]	[3,6,12,24,48,96,192,129]	[3,6,12,24,48,96,192,384,513,768]
2	13	[7,14,13,11]	[13,26,52,41,19,38]	[13,26,52,104,208,161,67,134]	[13,26,52,104,208,259,416,518,641,832]
3	57	[3,6,12,9]	[15,30,60,57,51,39]	[39,78,156,57,114,228,201,147]	[57,114,135,228,270,456,540,579,801,912]
4	241	[1,2,4,8]	[13,26,52,41,19,38]	[31,62,124,248,241,227,199,143]	[79,158,241,316,482,551,632,787,905,964]
5	993	[3,6,12,9]	[3,6,12,24,48,33]	[39,78,156,57,114,228,201,147]	[63,126,252,504,543,783,903,963,993,1008]
6	4033	[7,14,13,11]	[1,2,4,8,16,32]	[13,26,52,104,208,161,67,134]	[79,158,241,316,482,551,632,787,905,964]
7	16257	[3,6,12,9]	[3,6,12,24,48,33]	[3,6,12,24,48,96,192,129]	[57,114,135,228,270,456,540,579,801,912]
8	65281	[1,2,4,8]	[13,26,52,41,19,38]	[1,2,4,8,16,32,64,128]	[13,26,52,104,208,259,416,518,641,832]
9	261633	[3,6,12,9]	[15,30,60,57,51,39]	[3,6,12,24,48,96,192,129]	[3,6,12,24,48,96,192,384,513,768]

TABLE 17. Kasami-Welch exponents cyclotomic cosets $\pmod{2^m-1}$ for m even.

L	Exponent	Mod 7	Mod 31	Mod 127	Mod511
1	3	[3,6,5]	[3,6,12,24,17]	[3,6,12,24,48,96,65]	[3,6,12,24,48,96,192,384,257]
2	13	[3,6,5]	[11,22,13,26,21]	[13,26,52,104,81,35,70]	[13,26,52,104,208,416,321,131,262]
3	57	[1,2,4]	[11,22,13,26,21]	[23,46,92,57,114,101,75]	[57,114,228,456,401,291,71,142,284]
4	241	[3,6,5]	[3,6,12,24,17]	[23,46,92,57,114,101,75]	[47,94,188,376,241,482,453,395,279]
5	993	[3,6,5]	[1,2,4,8,16]	[13,26,52,104,81,35,70]	[47,94,188,376,241,482,453,395,279]
6	4033	[1,2,4]	[3,6,12,24,17]	[3,6,12,24,48,96,65]	[57,114,228,456,401,291,71,142,284]
7	16257	[3,6,5]	[11,22,13,26,21]	[1,2,4,8,16,32,64]	[13,26,52,104,208,416,321,131,262]
8	65281	[3,6,5]	[11,22,13,26,21]	[3,6,12,24,48,96,65]	[3,6,12,24,48,96,192,384,257]

TABLE 18. Kasami-Welch exponents cyclotomic cosets $\pmod{2^m-1}$ for m odd.

n	roots	Weights	a = b?
15	$\{1,5\}$	10,8,6	Yes
31	$\{1,3\}$	20,16,12	Yes
31	$\{1,5\}$	20,16,12	Yes
31	$\{1,7\}$	20,16,12	Yes
31	$\{1,11\}$	20,16,12	Yes
63	$\{1,5\}$	40,32,24	Yes
63	$\{1,9\}$	36,32,28	Yes
63	$\{1,13\}$	40,32,24	Yes
127	$\{1,3\}$	72,64,56	Yes
127	$\{1,5\}$	72,64,56	Yes
127	$\{1,9\}$	72,64,56	Yes
127	$\{1,11\}$	72,64,56	Yes
127	$\{1,13\}$	72,64,56	Yes
127	$\{1,15\}$	72,64,56	Yes
127	$\{1,23\}$	72,64,56	Yes
127	$\{1,27\}$	72,64,56	Yes
127	$\{1,29\}$	72,64,56	Yes
127	$\{1,43\}$	72,64,56	Yes
255	$\{1,17\}$	136,128,120	Yes
511	$\{1,3\}$	272,256,240	Yes

511	$\{1, 5\}$	272,256,240	Yes
511	$\{1, 9\}$	288,256,224	Yes
511	$\{1, 13\}$	272,256,240	Yes
511	$\{1, 17\}$	272,256,240	Yes
511	$\{1, 19\}$	272,256,240	Yes
511	$\{1, 27\}$	272,256,240	Yes
511	$\{1, 31\}$	272,256,240	Yes
511	$\{1, 47\}$	272,256,240	Yes
511	$\{1, 57\}$	288,256,224	Yes
511	$\{1, 59\}$	272,256,240	Yes
511	$\{1, 87\}$	272,256,240	Yes
511	$\{1, 103\}$	272,256,240	Yes
511	$\{1, 171\}$	272,256,240	Yes
1023	$\{1, 5\}$	544,512,480	Yes
1023	$\{1, 13\}$	544,512,480	Yes
1023	$\{1, 17\}$	544,512,480	Yes
1023	$\{1, 25\}$	544,512,480	Yes
1023	$\{1, 33\}$	528,512,496	Yes
1023	$\{1, 41\}$	544,512,480	Yes
1023	$\{1, 49\}$	544,512,480	Yes
1023	$\{1, 79\}$	544,512,480	Yes
1023	$\{1, 107\}$	544,512,480	Yes
1023	$\{1, 181\}$	544,512,480	Yes
1023	$\{1, 205\}$	544,512,480	Yes
2047	$\{1, 3\}$	1056,1024,992	Yes
2047	$\{1, 5\}$	1056,1024,992	Yes
2047	$\{1, 9\}$	1056,1024,992	Yes
2047	$\{1, 13\}$	1056,1024,992	Yes
2047	$\{1, 17\}$	1056,1024,992	Yes
2047	$\{1, 33\}$	1056,1024,992	Yes
2047	$\{1, 35\}$	1056,1024,992	Yes
2047	$\{1, 43\}$	1056,1024,992	Yes
2047	$\{1, 57\}$	1056,1024,992	Yes
2047	$\{1, 63\}$	1056,1024,992	Yes
2047	$\{1, 95\}$	1056,1024,992	Yes
2047	$\{1, 107\}$	1056,1024,992	Yes
2047	$\{1, 117\}$	1056,1024,992	Yes
2047	$\{1, 143\}$	1056,1024,992	Yes
2047	$\{1, 151\}$	1056,1024,992	Yes
2047	$\{1, 231\}$	1056,1024,992	Yes
2047	$\{1, 249\}$	1056,1024,992	Yes

2047	$\{1, 315\}$	1056,1024,992	Yes
2047	$\{1, 365\}$	1056,1024,992	Yes
2047	$\{1, 411\}$	1056,1024,992	Yes
2047	$\{1, 413\}$	1056,1024,992	Yes
2047	$\{1, 683\}$	1056,1024,992	Yes
4095	$\{1, 17\}$	2176,2048,1920	Yes
4095	$\{1, 65\}$	2176,2048,1920	Yes
4095	$\{1, 241\}$	2176,2048,1920	Yes
8191	$\{1, 3\}$	4160,4096,4032	Yes
8191	$\{1, 5\}$	4160,4096,4032	Yes
8191	$\{1, 9\}$	4160,4096,4032	Yes
8191	$\{1, 13\}$	4160,4096,4032	Yes
8191	$\{1, 17\}$	4160,4096,4032	Yes
8191	$\{1, 33\}$	4160,4096,4032	Yes
8191	$\{1, 57\}$	4160,4096,4032	Yes
8191	$\{1, 65\}$	4160,4096,4032	Yes
8191	$\{1, 67\}$	4160,4096,4032	Yes
8191	$\{1, 71\}$	4160,4096,4032	Yes
8191	$\{1, 127\}$	4160,4096,4032	Yes
8191	$\{1, 171\}$	4160,4096,4032	Yes
8191	$\{1, 191\}$	4160,4096,4032	Yes
8191	$\{1, 241\}$	4160,4096,4032	Yes
8191	$\{1, 287\}$	4160,4096,4032	Yes
8191	$\{1, 347\}$	4160,4096,4032	Yes
8191	$\{1, 367\}$	4160,4096,4032	Yes
8191	$\{1, 635\}$	4160,4096,4032	Yes
8191	$\{1, 723\}$	4160,4096,4032	Yes
8191	$\{1, 911\}$	4160,4096,4032	Yes
8191	$\{1, 1243\}$	4160,4096,4032	Yes
8191	$\{1, 1245\}$	4160,4096,4032	Yes
8191	$\{1, 1453\}$	4160,4096,4032	Yes
8191	$\{1, 1639\}$	4160,4096,4032	Yes
8191	$\{1, 1691\}$	4160,4096,4032	Yes
8191	$\{1, 2731\}$	4160,4096,4032	Yes

Table 19: Symmetric weight distribution verification for dual of cyclic codes generated.

Gold Function	Nonlinearity	Algebraic Degree	Correlation Immunity	Algebraic Immunity	Autocorrelation
$Tr(x^3) = Tr(x^3)$	4	2	0	1	{16: 4, 0: 12}
$Tr(x^5) = Tr(x^5)$	0	-1	3	0	{16: 16}
$Tr(x^9) = Tr(x^3)$	4	2	0	1	{16: 4, 0: 12}
Kasami-Welch Function	Nonlinearity	Algebraic Degree	Correlation Immunity	Algebraic Immunity	Autocorrelation
$Tr(x^3) = Tr(x^3)$	4	2	0	1	{16: 4, 0: 12}
$Tr(x^{13}) = Tr(x^7)$	4	3	0	2	{16: 1, 0: 9, 8: 6}
$Tr(x^{57}) = Tr(x^3)$	4	2	0	1	{16: 4, 0: 12}

TABLE 20. Properties of trace Boolean functions in four variables.

Gold Function	Nonlinearity	Algebraic Degree	Correlation Immunity	Algebraic Immunity	Autocorrelation
$Tr(x^3) = Tr(x^3)$	12	2	0	2	{32: 2, 0: 30}
$Tr(x^5) = Tr(x^5)$	12	2	0	2	{32: 2, 0: 30}
$Tr(x^9) = Tr(x^5)$	12	2	0	2	{32: 2, 0: 30}
$Tr(x^{17}) = Tr(x^3)$	12	2	0	2	{32: 2, 0: 30}
Kasami-Welch Function	Nonlinearity	Algebraic Degree	Correlation Immunity	Algebraic Immunity	Autocorrelation
$Tr(x^3) = Tr(x^3)$	12	2	0	2	{32: 2, 0: 30}
$Tr(x^{13}) = Tr(x^{11})$	12	3	0	3	{32: 1, 8: 16, 0: 15}
$Tr(x^{57}) = Tr(x^{11})$	12	3	0	3	{32: 1, 8: 16, 0: 15}
$Tr(x^{241}) = Tr(x^3)$	12	2	0	2	{32: 2, 0: 30}
Welch Function	Nonlinearity	Algebraic Degree	Correlation Immunity	Algebraic Immunity	Autocorrelation
$Tr(x^7) = Tr(x^7)$	12	3	0	3	{32: 1, 8: 16, 0: 15}
Inverse Function	Nonlinearity	Algebraic Degree	Correlation Immunity	Algebraic Immunity	Autocorrelation
$Tr(x^{30}) = Tr(x^{15})$	10	4	0	3	{32: 1, 8: 16, 0: 15}
Niho Function	Nonlinearity	Algebraic Degree	Correlation Immunity	Algebraic Immunity	Autocorrelation
$Tr(x^5) = Tr(x^5)$	12	2	0	2	{32: 2, 0: 30}
Dobbertin Function	Nonlinearity	Algebraic Degree	Correlation Immunity	Algebraic Immunity	Autocorrelation
$Tr(x^{339}) = Tr(x^{15})$	10	4	0	3	{32: 1, 8: 16, 0: 15}

TABLE 21. Properties of trace Boolean functions in five variables.

Gold Function	Nonlinearity	Algebraic Degree	Correlation Immunity	Algebraic Immunity	Autocorrelation
$Tr(x^3) = Tr(x^3)$	24	2	0	2	{64: 4, 0: 60}
$Tr(x^5) = Tr(x^5)$	24	2	0	2	{64: 4, 0: 60}
$Tr(x^9) = Tr(x^9)$	0	-1	5	0	{64: 64}
$Tr(x^{17}) = Tr(x^5)$	24	2	0	2	{64: 4, 0: 60}
$Tr(x^{33}) = Tr(x^3)$	24	2	0	2	{64: 4, 0: 60}
Kasami-Welch Function	Nonlinearity	Algebraic Degree	Correlation Immunity	Algebraic Immunity	Autocorrelation
$Tr(x^3) = Tr(x^3)$	24	2	0	2	{64: 4, 0: 60}
$Tr(x^{13}) = Tr(x^{13})$	24	3	0	3	{64: 1, 32: 4, 0: 27, 16: 32}
$Tr(x^{57}) = Tr(x^{15})$	24	4	0	2	{64: 1, 8: 39, 0: 15, 24: 9}
$Tr(x^{241}) = Tr(x^{13})$	24	3	0	3	{64: 1, 32: 4, 0: 27, 16: 32}
$Tr(x^{993}) = Tr(x^3)$	24	2	0	2	{64: 4, 0: 60}

TABLE 22. Properties of trace Boolean functions in six variables.

Gold Function	Nonlinearity	Algebraic Degree	Correlation Immunity	Algebraic Immunity	Autocorrelation
$Tr(x^3) = Tr(x^3)$	56	2	0	2	{128: 2, 0: 126}
$Tr(x^5) = Tr(x^5)$	56	2	0	2	{128: 2, 0: 126}
$Tr(x^9) = Tr(x^9)$	56	2	0	2	{128: 2, 0: 126}
$Tr(x^{17}) = Tr(x^9)$	56	2	0	2	{128: 2, 0: 126}
$Tr(x^{33}) = Tr(x^5)$	56	2	0	2	{128: 2, 0: 126}
$Tr(x^{65}) = Tr(x^3)$	56	2	0	2	{128: 2, 0: 126}
Kasami-Welch Function	Nonlinearity	Algebraic Degree	Correlation Immunity	Algebraic Immunity	Autocorrelation
$Tr(x^3) = Tr(x^3)$	56	2	0	2	{128: 2, 0: 126}
$Tr(x^{13}) = Tr(x^{13})$	56	3	0	3	{128: 1, 16: 64, 0: 63}
$Tr(x^{57}) = Tr(x^{23})$	56	4	0	3	{128: 1, 16: 64, 0: 63}
$Tr(x^{241}) = Tr(x^{23})$	56	4	0	3	{128: 1, 16: 64, 0: 63}
$Tr(x^{993}) = Tr(x^{13})$	56	3	0	3	{128: 1, 16: 64, 0: 63}
$Tr(x^{4033}) = Tr(x^3)$	56	2	0	2	{128: 2, 0: 126}
Welch Function	Nonlinearity	Algebraic Degree	Correlation Immunity	Algebraic Immunity	Autocorrelation
$Tr(x^{11}) = Tr(x^{11})$	56	3	0	3	{128: 1, 16: 64, 0: 63}
Inverse Function	Nonlinearity	Algebraic Degree	Correlation Immunity	Algebraic Immunity	Autocorrelation
$Tr(x^{126}) = Tr(x^{63})$	54	6	0	4	{128: 1, 16: 36, 8: 49, 0: 35, 24: 7}
Niho Function	Nonlinearity	Algebraic Degree	Correlation Immunity	Algebraic Immunity	Autocorrelation
$Tr(x^{39}) = Tr(x^{29})$	56	4	0	3	{128: 1, 16: 64, 0: 63}

TABLE 23. Properties of trace Boolean functions in seven variables.

Defining Set Size 2	Weight Distribution	Number of Nonzero Weights
{1, 3}	[4, 6, 8, 10, 12]	5
{1, 5}	[6, 8, 10]	3
{1, 7}	[4, 6, 8, 10]	4
Defining Set Size 3	Weight Distribution	Number of Nonzero Weights
{1, 3, 5}	[4, 6, 8, 10, 12]	5
{1, 3, 7}	[2, 4, 6, 8, 10, 12]	6
{1, 5, 7}	[2, 4, 6, 8, 10]	5
Defining Set Size 4	Weight Distribution	Number of Nonzero Weights
{1, 3, 5, 7}	[2, 4, 6, 8, 10, 12, 14]	7

TABLE 24. Weight distribution classes of cyclic codes in four variables with defining sets of size two, three and four considering cyclotomic coset representatives

Defining Set Size 2	Weight Distribution	Number of Nonzero Weights
{1, 3},{1, 5},{1, 7},{1, 11}	[12, 16, 20]	3
{1, 15}	[10, 12, 14, 16, 18, 20]	6
Defining Set Size 3	Weight Distribution	Number of Nonzero Weights
{1, 3, 5},{1, 3, 11},{1, 5, 7},{1, 7, 11}	[8, 12, 16, 20, 24]	5
{1, 3, 7},{1, 5, 15},{1, 11, 15}	[8, 10, 12, 14, 16, 18, 20, 22, 26]	9
{1, 3, 15},{1, 5, 11},{1, 7, 15}	[6, 8, 10, 12, 14, 16, 18, 20, 22]	9
Defining Set Size 4	Weight Distribution	Number of Nonzero Weights
{1, 3, 5, 7},{1, 3, 5, 11},{1, 3, 5, 15},{1, 3, 7, 11},{1, 3, 7, 15},{1, 3, 11, 15},{1, 5, 7, 11},{1, 5, 7, 15},{1, 5, 11, 15},{1, 7, 11, 15}	[6, 8, 10, 12, 14, 16, 18, 20, 22, 24, 26]	11

TABLE 25. Weight distribution classes of cyclic Codes in five variables with defining sets of size two, three and four considering cyclotomic coset representatives.

Defining Set Size 2	Weight Distribution	Number of Nonzero Weights
{1, 3}	[24, 28, 32, 36, 40]	5
{1, 5},{1, 13}	[24, 32, 40]	3
{1, 9}	[28, 32, 36]	3
{1, 15}	[24, 28, 32, 36]	4
Defining Set Size 3	Weight Distribution	Number of Nonzero Weights
{1, 3, 5},{1, 3, 9},{1, 13, 15}	[16, 24, 28, 32, 36, 40, 48]	7
{1, 3, 13},{1, 5, 15}	[16, 20, 24, 28, 32, 36, 40, 44]	8
{1, 3, 15}	[12, 18, 20, 22, 24, 26, 28, 30, 32, 34, 36, 38, 40, 42]	14
{1, 5, 9}	[24, 28, 32, 36, 40]	5
{1, 5, 13}	[16, 20, 24, 28, 32, 36, 40]	7
{1, 9, 13}	[20, 24, 28, 32, 36, 40, 48]	7
{1, 9, 15}	[12, 20, 24, 28, 32, 36]	6
Defining Set Size 4	Weight Distribution	Number of Nonzero Weights
{1, 3, 5, 9}	[16, 24, 28, 32, 36, 40, 48]	7
{1, 3, 5, 13},{1, 3, 9, 13},{1, 5, 9, 13},{1, 5, 13, 15}	[16, 20, 24, 28, 32, 36, 40, 44, 48]	9
{1, 3, 5, 15},{1, 3, 13, 15}	[12, 16, 18, 20, 22, 24, 26, 28, 30, 32, 34, 36, 38, 40, 42, 44, 46, 48]	18
{1, 3, 9, 15}	[12, 16, 18, 20, 22, 24, 26, 28, 30, 32, 34, 36, 38, 40, 42, 44, 48, 54]	18
{1, 5, 9, 15}	[12, 16, 20, 24, 28, 32, 36, 40, 44]	9
{1, 9, 13, 15}	[12, 16, 20, 24, 28, 32, 36, 40, 44, 48]	10

TABLE 26. Weight distribution classes of cyclic codes in six variables with defining sets of size two, three and four considering cyclotomic coset representatives.

Defining Set Size 2	Weight Distribution	Number of Nonzero Weights
{1, 3},{1, 5},{1, 9},{1, 11},{1, 13},{1, 23},{1, 29}	[56, 64, 72]	3
{1, 63}	[54, 56, 58, 60, 62, 64, 66, 68, 70, 72, 74]	11
Defining Set Size 3	Weight Distribution	Number of Nonzero Weights
{1, 3, 5},{1, 3, 9},{1, 3, 11},{1, 3, 13},{1, 5, 9},{1, 5, 11},{1, 9, 13},{1, 13, 29}	[48, 56, 64, 72, 80]	5
{1, 3, 23},{1, 5, 29},{1, 9, 23},{1, 11, 13}	[44, 48, 52, 56, 60, 64, 68, 72, 76, 80, 84]	11
{1, 3, 29},{1, 5, 23},{1, 9, 29},{1, 11, 23}	[36, 40, 44, 48, 52, 56, 60, 64, 68, 72, 76, 80]	12
{1, 3, 63}	[46, 48, 50, 52, 54, 56, 58, 60, 62, 64, 66, 68, 70, 72, 74, 76, 78, 80, 82, 84]	20
{1, 5, 13}	[36, 44, 48, 52, 56, 60, 64, 68, 72, 76, 80, 84]	12
{1, 5, 63},{1, 9, 63}	[44, 46, 48, 50, 52, 54, 56, 58, 60, 62, 64, 66, 68, 70, 72, 74, 76, 78, 80, 82, 84]	21
{1, 9, 11},{1, 23, 29}	[44, 48, 52, 56, 60, 64, 68, 72, 76, 80, 84, 92]	12
{1, 11, 29},{1, 13, 63},{1, 23, 63}	[46, 48, 50, 52, 54, 56, 58, 60, 62, 64, 66, 68, 70, 72, 74, 76, 78, 80, 86]	19
{1, 11, 63},{1, 13, 23},{1, 29, 63}	[44, 46, 48, 50, 52, 54, 56, 58, 60, 62, 64, 66, 68, 70, 72, 74, 76, 78, 80, 82]	20

TABLE 27. Weight distribution classes of cyclic codes in seven variables with defining sets of size two and three considering cyclotomic coset representatives.

Defining Set Size 4	Weight Distribution	Number of Nonzero Weights
{1, 3, 5, 9}	[32, 48, 56, 64, 72, 80, 96]	7
{1, 3, 5, 11},{1, 3, 5, 13},{1, 3, 5, 23},{1, 3, 5, 29},{1, 3, 9, 23},{1, 3, 9, 29},{1, 3, 13, 29},{1, 5, 9, 13},{1, 5, 9, 23},{1, 5, 9, 29},{1, 5, 11, 13},{1, 5, 11, 23},{1, 5, 13, 29},{1, 9, 13, 29}	[36, 40, 44, 48, 52, 56, 60, 64, 68, 72, 76, 80, 84, 88]	14
{1, 3, 5, 63}	[36, 40, 42, 44, 46, 48, 50, 52, 54, 56, 58, 60, 62, 64, 66, 68, 70, 72, 74, 76, 78, 80, 82, 84, 86, 88]	26
{1, 3, 9, 11},{1, 3, 9, 13},{1, 3, 11, 23},{1, 3, 23, 29},{1, 5, 9, 11},{1, 5, 23, 29},{1, 9, 11, 13},{1, 9, 11, 23},{1, 9, 23, 29}	[36, 40, 44, 48, 52, 56, 60, 64, 68, 72, 76, 80, 84, 88, 92]	15
{1, 3, 9, 63},{1, 3, 23, 63},{1, 5, 9, 63},{1, 11, 29, 63},{1, 13, 23, 63}	[38, 40, 42, 44, 46, 48, 50, 52, 54, 56, 58, 60, 62, 64, 66, 68, 70, 72, 74, 76, 78, 80, 82, 84, 86, 88, 90]	27
{1, 3, 11, 13}	[28, 40, 44, 48, 52, 56, 60, 64, 68, 72, 76, 80, 84, 88]	14
{1, 3, 11, 29},{1, 11, 23, 63},{1, 13, 29, 63}	[36, 40, 42, 44, 46, 48, 50, 52, 54, 56, 58, 60, 62, 64, 66, 68, 70, 72, 74, 76, 78, 80, 82, 84, 86, 88, 90]	27
{1, 3, 11, 63},{1, 13, 23, 29}	[36, 38, 40, 42, 44, 46, 48, 50, 52, 54, 56, 58, 60, 62, 64, 66, 68, 70, 72, 74, 76, 78, 80, 82, 84, 86, 88, 90, 92]	29
{1, 3, 13, 23},{1, 5, 11, 63}	[40, 42, 44, 46, 48, 50, 52, 54, 56, 58, 60, 62, 64, 66, 68, 70, 72, 74, 76, 78, 80, 82, 84, 86, 88, 90]	26
{1, 3, 13, 63},{1, 3, 29, 63},{1, 5, 11, 29},{1, 5, 13, 23},{1, 9, 29, 63},{1, 11, 13, 23}	[36, 38, 40, 42, 44, 46, 48, 50, 52, 54, 56, 58, 60, 62, 64, 66, 68, 70, 72, 74, 76, 78, 80, 82, 84, 86, 88]	27
{1, 5, 13, 63}	[36, 38, 40, 42, 44, 46, 48, 50, 52, 54, 56, 58, 60, 62, 64, 66, 68, 70, 72, 74, 76, 78, 80, 82, 84, 86, 90]	27
{1, 5, 23, 63},{1, 9, 11, 29},{1, 9, 23, 63},{1, 11, 23, 29}	[36, 38, 40, 42, 44, 46, 48, 50, 52, 54, 56, 58, 60, 62, 64, 66, 68, 70, 72, 74, 76, 78, 80, 82, 84, 86, 88, 92]	28
{1, 5, 29, 63},{1, 9, 13, 23}	[38, 40, 42, 44, 46, 48, 50, 52, 54, 56, 58, 60, 62, 64, 66, 68, 70, 72, 74, 76, 78, 80, 82, 84, 86, 88]	26
{1, 9, 11, 63}	[40, 42, 44, 46, 48, 50, 52, 54, 56, 58, 60, 62, 64, 66, 68, 70, 72, 74, 76, 78, 80, 82, 84, 86, 90, 92]	26
{1, 9, 13, 63}	[36, 38, 40, 42, 44, 46, 48, 50, 52, 54, 56, 58, 60, 62, 64, 66, 68, 70, 72, 74, 76, 78, 80, 82, 84, 86, 88, 90]	28
{1, 11, 13, 29}	[38, 40, 42, 44, 46, 48, 50, 52, 54, 56, 58, 60, 62, 64, 66, 68, 70, 72, 74, 76, 78, 80, 82, 84, 86, 90]	26
{1, 11, 13, 63},{1, 23, 29, 63}	[38, 40, 42, 44, 46, 48, 50, 52, 54, 56, 58, 60, 62, 64, 66, 68, 70, 72, 74, 76, 78, 80, 82, 84, 86, 88, 90, 92]	28

TABLE 28. Weight distribution classes of cyclic codes in seven variables with defining sets of size four considering cyclotomic coset representatives.

Defining Sets Size 2	Nonlinear Functions	Number of Weights	Minimum Distance
{1, 3}	{1, Gold Kasami-Welch }	5	5
{1, 5}	{1, Gold}	3	3
{1, 7}	{1, Kasami-Welch}	4	3
Defining Sets Size 3	Nonlinear Functions	Number of Weights	Minimum Distance
{1, 3, 5}	{1, Gold Kasami-Welch , Gold }	5	7
{1, 3, 7}	{1, Gold Kasami-Welch , Kasami-Welch }	6	5
{1, 5, 7}	{1, Gold , Kasami-Welch }	5	3
Defining Sets Size 4	Nonlinear Functions	Number of Weights	Minimum Distance
{1, 3, 5, 7}	{1, Gold Kasami-Welch , Gold , Kasami-Welch }	7	15

TABLE 29. Weight distribution classification of defining sets considered for cyclic codes in four variables with defining sets of size two, three and four considering cyclotomic coset representatives.

Defining Sets Size 2	Nonlinear Functions	Number of Weights	Minimum Distance
{1, 3},{1, 5},{1, 7},{1,11}	{1, Gold Kasami-Welch }, {1, Gold Niho }, {1, Welch }, {1, Kasami-Welch }	3	5
{1, 15}	{1, Inverse Dobbertin }	6	5
Defining Set Size 3	Nonlinear Functions	Number of Nonzero Weights	Minimum Distance
{1, 3, 5},{1, 3, 11},{1, 5, 7},{1, 7, 11}	{1, Gold Kasami-Welch , Gold Niho }, {1, Gold Kasami-Welch , Kasami-Welch }, {1, Gold Niho , Welch }, {1, Welch , Kasami-Welch }	5	7
{1, 3, 7},{1, 5, 15},{1, 11, 15}	{1, Gold Kasami-Welch , Welch }, {1, Gold Niho , Inverse Dobbertin }, {1, Kasami-Welch , Inverse Dobbertin }	9	6
{1, 3, 15},{1, 5, 11},{1, 7, 15}	{1, Gold Kasami-Welch , Inverse Dobbertin }, {1, Gold Niho , Kasami-Welch }, {1, Welch , Inverse Dobbertin }	9	5
Defining Set Size 4	Nonlinear Functions	Number of Nonzero Weights	Minimum Distance
{1, 3, 5, 7},{1, 3, 5, 11},{1, 3, 5, 15},{1, 3, 7, 11},{1, 3, 7, 15},{1, 3, 11, 15},{1, 5, 7, 11},{1, 5, 7, 15},{1, 5, 11, 15},{1, 7, 11, 15}	{1, Gold Kasami-Welch , Gold Niho , Welch }, {1, Gold Kasami-Welch , Gold Niho , Kasami-Welch }, {1, Gold Kasami-Welch , Gold Niho , Inverse Dobbertin }, {1, Gold Kasami-Welch , Welch , Kasami-Welch }, {1, Gold Kasami-Welch , Welch , Inverse Dobbertin }, {1, Gold Kasami-Welch , Kasami-Welch , Inverse Dobbertin }, {1, Gold Niho , Welch , Kasami-Welch }, {1, Gold Niho , Welch , Inverse Dobbertin }, {1, Gold Niho , Kasami-Welch , Inverse Dobbertin }, {1, Welch , Kasami-Welch , Inverse Dobbertin }	11	11

TABLE 30. Weight distribution classification of defining sets considered for cyclic codes in five variables with defining sets of size two, three and four considering cyclotomic coset representatives.

Defining Set Size 2	Nonlinear Functions	Number of Nonzero Weights	Minimum Distance
{1, 3}	{1, Gold Kasami-Welch }	5	5
{1, 5},{1, 13}	{1, Gold}, {1, Kasami-Welch }	3	3
{1, 9}	{1, Gold}	3	3
{1, 15}	{1, Kasami-Welch }	4	3
Defining Set Size 3	Nonlinear Functions	Number of Nonzero Weights	Minimum Distance
{1, 3, 5},{1, 3, 9},{1, 13, 15}	{1, Gold Kasami-Welch , Gold }, {1, Gold Kasami-Welch , Gold }, {1, Kasami-Welch , Kasami-Welch }	7	7
{1, 3, 13},{1, 5, 15}	{1, Gold Kasami-Welch , Kasami-Welch }, {1, Gold , Kasami-Welch }	8	5
{1, 3, 15}	{1, Gold Kasami-Welch , Kasami-Welch }	14	5
{1, 5, 9}	{1, Gold , Gold }	5	5
{1, 5, 13}	{1, Gold , Kasami-Welch }	7	3
{1, 9, 13}	{1, Gold , Kasami-Welch }	7	4
{1, 9, 15}	{1, Gold , Kasami-Welch }	6	3
Defining Set Size 4	Nonlinear Functions	Number of Nonzero Weights	Minimum Distance
{1, 3, 5, 9}	{1, Gold Kasami-Welch , Gold , Gold }	7	7
{1, 3, 5, 13},{1, 3, 9, 13},{1, 5, 9, 13},{1, 5, 13, 15}	{1, Gold Kasami-Welch , Gold , Kasami-Welch }, {1, Gold Kasami-Welch , Gold , Kasami-Welch }, {1, Gold , Gold , Kasami-Welch }, {1, Gold , Kasami-Welch , Kasami-Welch }	9	7
{1, 3, 5, 15},{1, 3, 13, 15}	{1, Gold Kasami-Welch , Gold , Kasami-Welch }, {1, Gold Kasami-Welch , Kasami-Welch , Kasami-Welch }	18	7
{1, 3, 9, 15}	{1, Gold Kasami-Welch , Gold , Kasami-Welch }	18	6
{1, 5, 9, 15}	{1, Gold , Gold , Kasami-Welch }	9	5
{1, 9, 13, 15}	{1, Gold , Kasami-Welch , Kasami-Welch }	10	7

TABLE 31. Weight distribution classification of defining sets considered for cyclic codes in six variables with defining sets of size two, three and four considering cyclotomic coset representatives.

Defining Set Size 2	Nonlinear Functions	Number of Nonzero Weights	Minimum Distance
{1, 3},{1, 5},{1, 9},{1, 11},{1, 13},{1, 23},{1, 29}	{1, Gold Kasami-Welch }, {1, Gold }, {1, Gold }, {1, Welch }, {1, Kasami-Welch }, {1, Kasami-Welch }, {1, Niho }	3	5
{1, 63}	{1, Inverse }	11	5
Defining Set Size 3	Nonlinear Functions	Number of Nonzero Weights	Minimum Distance
{1, 3, 5},{1, 3, 9},{1, 3, 11},{1, 3, 13},{1, 5, 9},{1, 5, 11},{1, 9, 13},{1, 13, 29}	{1, Gold Kasami-Welch , Gold }, {1, Gold Kasami-Welch , Gold }, {1, Gold Kasami-Welch , Welch }, {1, Gold Kasami-Welch , Kasami-Welch }, {1, Gold , Gold }, {1, Gold , Welch }, {1, Gold , Kasami-Welch }, {1, Kasami-Welch , Niho }	5	7
{1, 3, 23},{1, 5, 29},{1, 9, 23},{1, 11, 13}	{1, Gold Kasami-Welch , Kasami-Welch }, {1, Gold , Niho }, {1, Gold , Kasami-Welch }, {1, Welch , Kasami-Welch }	11	6
{1, 3, 29},{1, 5, 23},{1, 9, 29},{1, 11, 23}	{1, Gold Kasami-Welch , Niho }, {1, Gold , Kasami-Welch }, {1, Gold , Niho }, {1, Welch , Kasami-Welch }	12	5
{1, 3, 63}	{1, Gold Kasami-Welch , Inverse }	20	7
{1, 5, 13}	{1, Gold , Kasami-Welch }	12	6
{1, 5, 63},{1, 9, 63}	{1, Gold , Inverse }, {1, Gold , Inverse }	21	6
{1, 9, 11},{1, 23, 29}	{1, Gold , Welch }, {1, Kasami-Welch , Niho }	12	6
{1, 11, 29},{1, 13, 63},{1, 23, 63}	{1, Welch , Niho }, {1, Kasami-Welch , Inverse }, {1, Kasami-Welch , Inverse }	19	6
{1, 11, 63},{1, 13, 23},{1, 29, 63}	{1, Welch , Inverse }, {1, Kasami-Welch , Kasami-Welch }, {1, Niho , Inverse }	20	6

TABLE 32. Weight distribution classification of defining sets considered for cyclic codes in seven variables with defining sets of size two and three considering cyclotomic coset representatives.

Defining Set Size 4	Nonlinear Functions	Number of Nonzero Weights	Minimum Distance
{1, 3, 5, 9}	{1, Gold Kasami-Welch , Gold , Gold }	7	7
{1, 3, 5, 11},{1, 3, 5, 13},{1, 3, 5, 23},{1, 3, 5, 29},{1, 3, 9, 23},{1, 3, 9, 29},{1, 3, 13, 29},{1, 5, 9, 13},{1, 5, 9, 23},{1, 5, 9, 29},{1, 5, 11, 13},{1, 5, 11, 23},{1, 5, 13, 29},{1, 9, 13, 29}	{1, Gold Kasami-Welch , Gold , Welch }, {1, Gold Kasami-Welch , Gold , Kasami-Welch }, {1, Gold Kasami-Welch , Gold , Kasami-Welch }, {1, Gold Kasami-Welch , Gold , Niho }, {1, Gold Kasami-Welch , Gold , Kasami-Welch }, {1, Gold Kasami-Welch , Gold , Niho }, {1, Gold Kasami-Welch , Kasami-Welch , Niho }, {1, Gold , Gold , Kasami-Welch }, {1, Gold , Gold , Kasami-Welch }, {1, Gold , Gold , Niho }, {1, Gold , Welch , Kasami-Welch }, {1, Gold , Welch , Kasami-Welch }, {1, Gold , Kasami-Welch , Niho }, {1, Gold , Kasami-Welch , Niho }	14	7
{1, 3, 5, 63}	{1, Gold Kasami-Welch , Gold , Inverse }	26	7
{1, 3, 9, 11},{1, 3, 9, 13},{1, 3, 11, 23},{1, 3, 23, 29},{1, 5, 9, 11},{1, 5, 23, 29},{1, 9, 11, 13},{1, 9, 11, 23},{1, 9, 23, 29}	{1, Gold Kasami-Welch , Gold , Welch }, {1, Gold Kasami-Welch , Gold , Kasami-Welch }, {1, Gold Kasami-Welch , Welch , Kasami-Welch }, {1, Gold Kasami-Welch , Kasami-Welch , Niho }, {1, Gold , Gold , Welch }, {1, Gold , Kasami-Welch , Niho }, {1, Gold , Welch , Kasami-Welch }, {1, Gold , Welch , Kasami-Welch }, {1, Gold , Kasami-Welch , Kasami-Welch , Niho }	15	7
{1, 3, 9, 63},{1, 3, 23, 63},{1, 5, 9, 63},{1, 11, 29, 63},{1, 13, 23, 63}	{1, Gold Kasami-Welch , Gold , Inverse }, {1, Gold Kasami-Welch , Kasami-Welch , Inverse }, {1, Gold , Gold , Inverse }, {1, Welch , Niho , Inverse }, {1, Kasami-Welch , Kasami-Welch , Inverse }	27	8
{1, 3, 11, 13}	{1, Gold Kasami-Welch , Welch , Kasami-Welch }	14	7
{1, 3, 11, 29},{1, 11, 23, 63},{1, 13, 29, 63}	{1, Gold Kasami-Welch , Welch , Niho }, {1, Welch , Kasami-Welch , Inverse }, {1, Kasami-Welch , Niho , Inverse }	27	7
{1, 3, 11, 63},{1, 13, 23, 29}	{1, Gold Kasami-Welch , Welch , Inverse }, {1, Kasami-Welch , Kasami-Welch , Niho }	29	8
{1, 3, 13, 23},{1, 5, 11, 63}	{1, Gold Kasami-Welch , Kasami-Welch , Kasami-Welch }, {1, Gold , Welch , Inverse }	26	7
{1, 3, 13, 63},{1, 3, 29, 63},{1, 5, 11, 29},{1, 5, 13, 23},{1, 9, 29, 63},{1, 11, 13, 23}	{1, Gold Kasami-Welch , Kasami-Welch , Inverse }, {1, Gold Kasami-Welch , Niho , Inverse }, {1, Gold , Welch , Niho }, {1, Gold , Kasami-Welch , Kasami-Welch }, {1, Gold , Niho , Inverse }, {1, Welch , Kasami-Welch , Kasami-Welch }	27	7
{1, 5, 13, 63}	{1, Gold , Kasami-Welch , Inverse }	27	7
{1, 5, 23, 63},{1, 9, 11, 29},{1, 9, 23, 63},{1, 11, 23, 29}	{1, Gold , Kasami-Welch , Inverse }, {1, Gold , Welch , Niho }, {1, Gold , Kasami-Welch , Inverse }, {1, Welch , Kasami-Welch , Niho }	28	7
{1, 5, 29, 63},{1, 9, 13, 23}	{1, Gold , Niho , Inverse }, {1, Gold , Kasami-Welch , Kasami-Welch }	26	7
{1, 9, 11, 63}	{1, Gold , Welch , Inverse }	26	8
{1, 9, 13, 63}	{1, Gold , Kasami-Welch , Inverse }	28	7
{1, 11, 13, 29}	{1, Welch , Kasami-Welch , Niho }	26	7
{1, 11, 13, 63},{1, 23, 29, 63}	{1, Welch , Kasami-Welch , Inverse }, {1, Kasami-Welch , Niho , Inverse }	28	8

TABLE 33. Weight distribution classification of defining sets considered for cyclic codes in seven variables with defining sets of size four considering cyclotomic coset representatives.

Variables	SNR	Standard Deviation
4	26.32	0.05
4	20.3	0.1
4	16.778	0.15
4	14.279	0.2
4	12.341	0.25
4	10.757	0.3
4	9.418	0.35
4	8.258	0.4
4	7.235	0.45
4	6.32	0.5
4	5.492	0.55
4	4.737	0.6
4	4.041	0.65
4	3.398	0.7
4	2.798	0.75
4	2.238	0.8
4	1.711	0.85
4	1.215	0.9
4	0.745	0.95
5	24.702	0.05
5	18.681	0.1
5	15.159	0.15
5	12.661	0.2
5	10.722	0.25
5	9.139	0.3
5	7.8	0.35
5	6.64	0.4
5	5.617	0.45
5	4.702	0.5
5	3.874	0.55
5	3.118	0.6
5	2.423	0.65
5	1.779	0.7
5	1.18	0.75
5	0.619	0.8
5	0.093	0.85
5	-0.404	0.9
5	-0.873	0.95
6	23.928	0.05
6	17.907	0.1
6	14.386	0.15

6	11.887	0.2
6	9.949	0.25
6	8.365	0.3
6	7.026	0.35
6	5.866	0.4
6	4.843	0.45
6	3.928	0.5
6	3.1	0.55
6	2.344	0.6
6	1.649	0.65
6	1.005	0.7
6	0.406	0.75
6	-0.154	0.8
6	-0.681	0.85
6	-1.177	0.9
6	-1.647	0.95
7	23.518	0.05
7	17.497	0.1
7	13.975	0.15
7	11.476	0.2
7	9.538	0.25
7	7.955	0.3
7	6.616	0.35
7	5.456	0.4
7	4.433	0.45
7	3.518	0.5
7	2.69	0.55
7	1.934	0.6
7	1.239	0.65
7	0.595	0.7
7	-0.004	0.75
7	-0.565	0.8
7	-1.091	0.85
7	-1.588	0.9
7	-2.058	0.95
8	23.292	0.05
8	17.271	0.1
8	13.749	0.15
8	11.251	0.2
8	9.312	0.25
8	7.729	0.3

8	6.39	0.35
8	5.23	0.4
8	4.207	0.45
8	3.292	0.5
8	2.464	0.55
8	1.708	0.6
8	1.013	0.65
8	0.369	0.7
8	-0.23	0.75
8	-0.791	0.8
8	-1.317	0.85
8	-1.814	0.9
8	-2.283	0.95
9	23.166	0.05
9	17.145	0.1
9	13.624	0.15
9	11.125	0.2
9	9.187	0.25
9	7.603	0.3
9	6.264	0.35
9	5.104	0.4
9	4.081	0.45
9	3.166	0.5
9	2.338	0.55
9	1.582	0.6
9	0.887	0.65
9	0.243	0.7
9	-0.356	0.75
9	-0.916	0.8
9	-1.443	0.85
9	-1.939	0.9
9	-2.409	0.95
10	23.096	0.05
10	17.075	0.1
10	13.554	0.15
10	11.055	0.2
10	9.117	0.25
10	7.533	0.3
10	6.194	0.35
10	5.034	0.4
10	4.011	0.45

10	3.096	0.5
10	2.268	0.55
10	1.512	0.6
10	0.817	0.65
10	0.173	0.7
10	-0.426	0.75
10	-0.986	0.8
10	-1.513	0.85
10	-2.009	0.9
10	-2.479	0.95
11	23.057	0.05
11	17.037	0.1
11	13.515	0.15
11	11.016	0.2
11	9.078	0.25
11	7.494	0.3
11	6.155	0.35
11	4.995	0.4
11	3.972	0.45
11	3.057	0.5
11	2.229	0.55
11	1.474	0.6
11	0.778	0.65
11	0.135	0.7
11	-0.465	0.75
11	-1.025	0.8
11	-1.552	0.85
11	-2.048	0.9
11	-2.518	0.95
12	23.036	0.05
12	17.015	0.1
12	13.493	0.15
12	10.995	0.2
12	9.056	0.25
12	7.473	0.3
12	6.134	0.35
12	4.974	0.4
12	3.951	0.45
12	3.036	0.5
12	2.208	0.55
12	1.452	0.6

12	0.757	0.65
12	0.113	0.7
12	-0.486	0.75
12	-1.047	0.8
12	-1.573	0.85
12	-2.07	0.9
12	-2.539	0.95

Table 34: Standard deviations and corresponding SNR values for the two root codes.

SNR	Standard Deviation
0.972	0.25
0.944	0.5
0.917	0.75
0.891	1.0
0.866	1.25
0.841	1.5
0.818	1.75
0.794	2.0
0.772	2.25
0.75	2.5
0.729	2.75
0.708	3.0
0.688	3.25
0.668	3.5
0.649	3.75
0.631	4.0
0.613	4.25
0.596	4.5
0.579	4.75
0.562	5.0
0.546	5.25
0.531	5.5
0.516	5.75
0.501	6.0
0.487	6.25
0.473	6.5
0.46	6.75
0.447	7.0
0.434	7.25

0.422	7.5
0.41	7.75
0.398	8.0
0.387	8.25
0.376	8.5
0.365	8.75
0.355	9.0
0.345	9.25
0.335	9.5
0.325	9.75
0.316	10.0
0.307	10.25
0.299	10.5
0.29	10.75
0.282	11.0
0.274	11.25
0.266	11.5
0.259	11.75
0.251	12.0
0.244	12.25
0.237	12.5
0.23	12.75
0.224	13.0
0.218	13.25
0.211	13.5
0.205	13.75
0.2	14.0
0.194	14.25
0.188	14.5
0.183	14.75
0.178	15.0
0.173	15.25
0.168	15.5
0.163	15.75
0.158	16.0
0.154	16.25
0.15	16.5
0.145	16.75
0.141	17.0
0.137	17.25
0.133	17.5

0.13	17.75
0.126	18.0
0.122	18.25
0.119	18.5
0.115	18.75
0.112	19.0

Table 35: Standard deviation to SNR conversion for the rate .5 LDPC(2000,1000) code.

9. Appendix:Figures

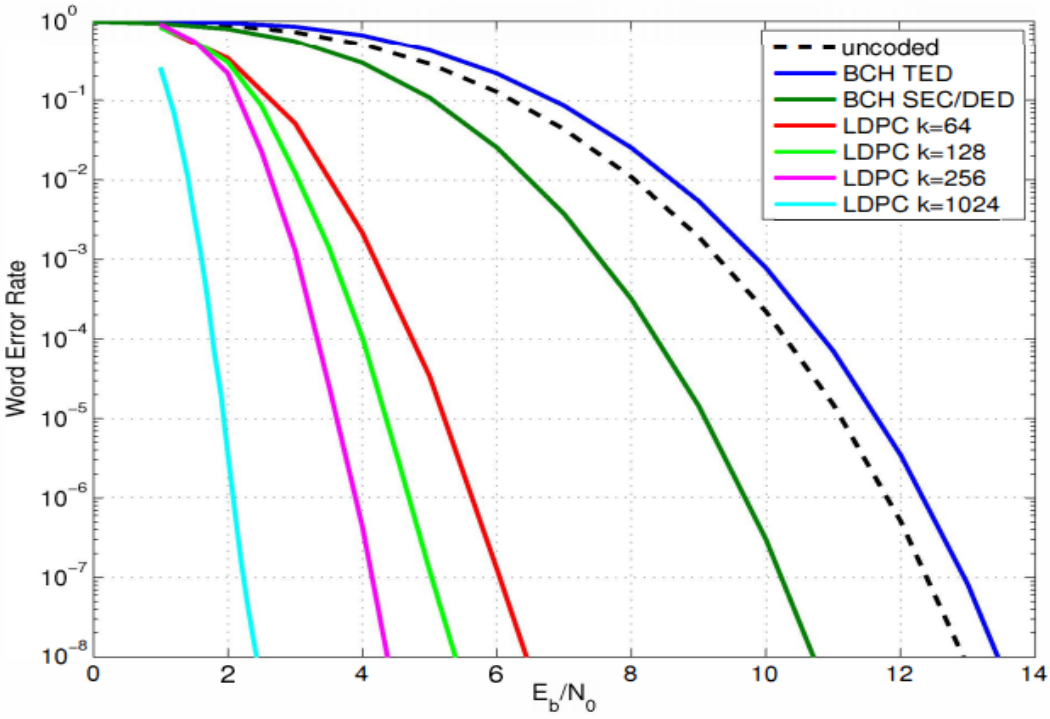


Figure 1. Andrews table of codeword error rate vs. SNR [1]

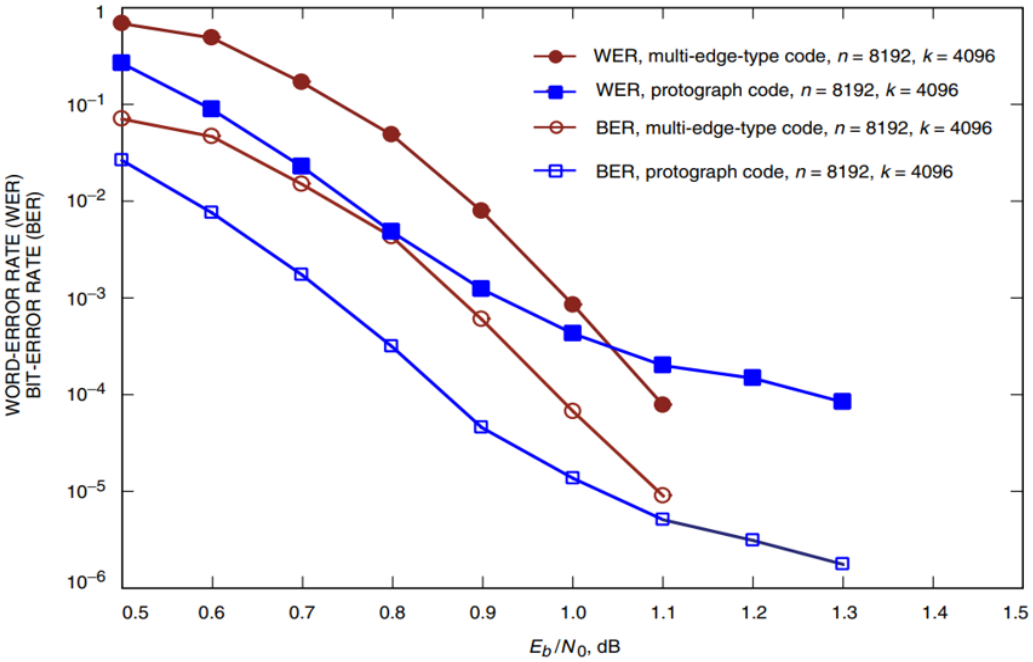
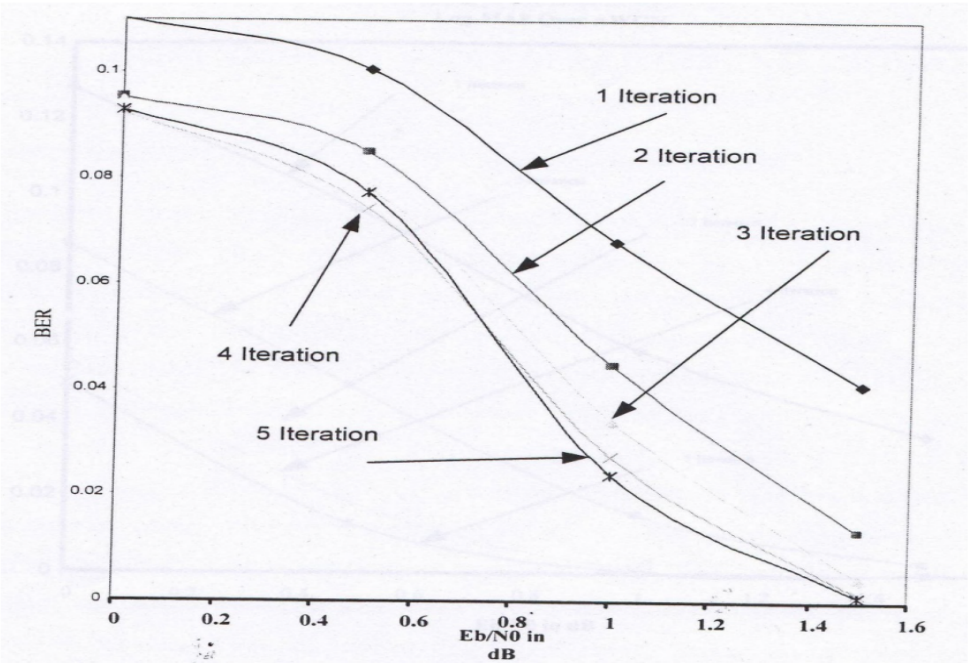


Fig. 6. Decoding performance of a large protograph code constructed from the protograph in Fig. 5.

FIGURE 2. Example graph based on Thorpe's construction taken from [66]



Performance of 1024 bit, rate 1/2 log-MAP Turbo Code versus rate 1/2 fourstate convolutional Code over AWGN.

FIGURE 3. Example graph based on Sah's construction taken from [61] where they compare the performance of rate $\frac{1}{2}$ log-MAP Turbo codes versus four state convolutional Code

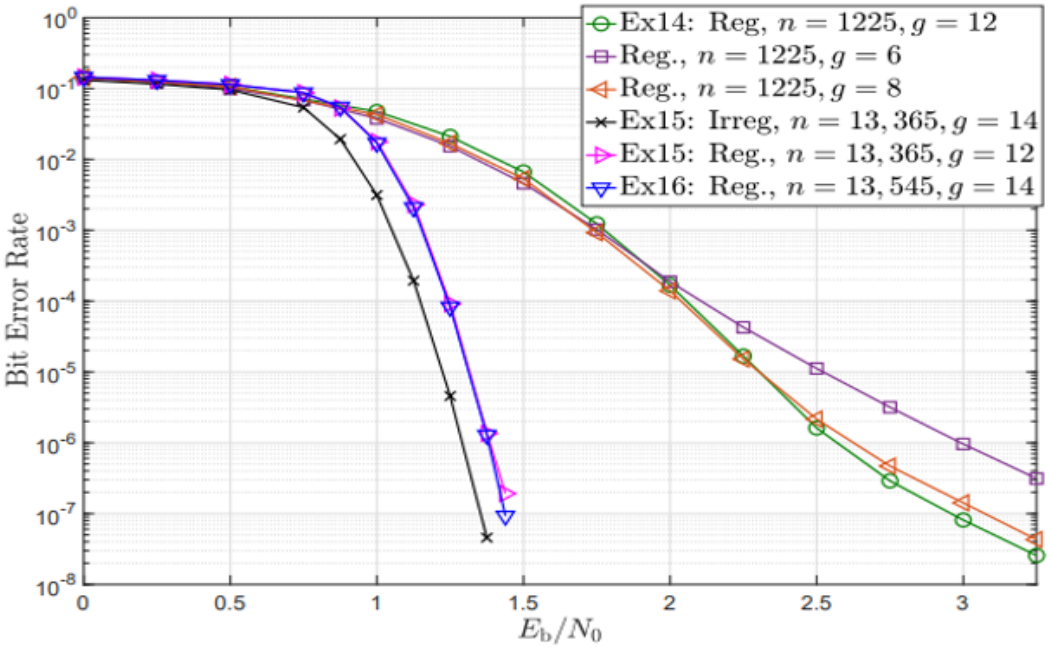


Fig. 1. Simulated decoding performance in terms of BER for the $R = 2/5$ QC-LDPC codes from Examples 14-16.

FIGURE 4. Example graph based on Smaradanche construction taken from [63] where they compare the performance of Quasi-Cyclic based LDPC codes

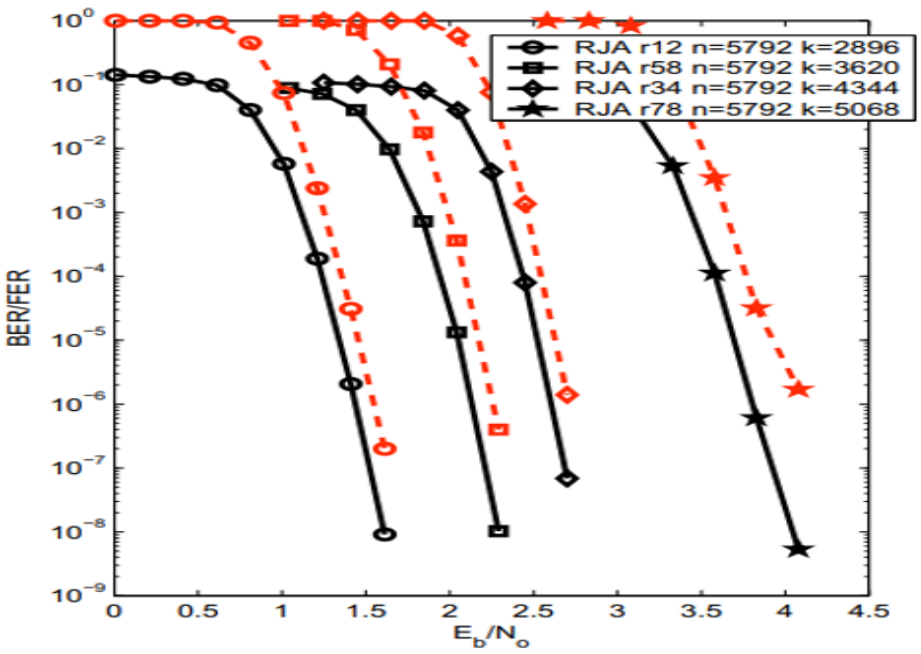


FIGURE 5. Example graph based on Divsalar's construction taken from [24] where they compare the performance of different rate codes based on their construction.

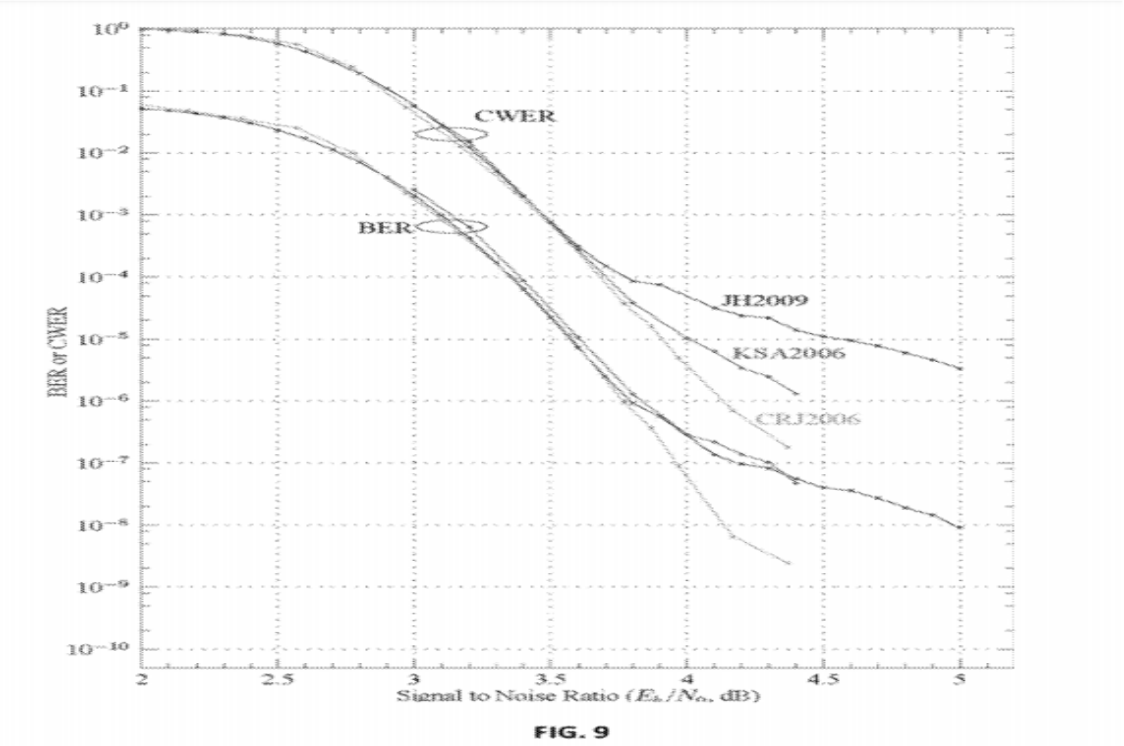


FIGURE 6. Example graph based on Hamkins construction taken from [33] where they compare the performance of selected length 1024 rate $\frac{4}{5}$ AR4JA decoders.

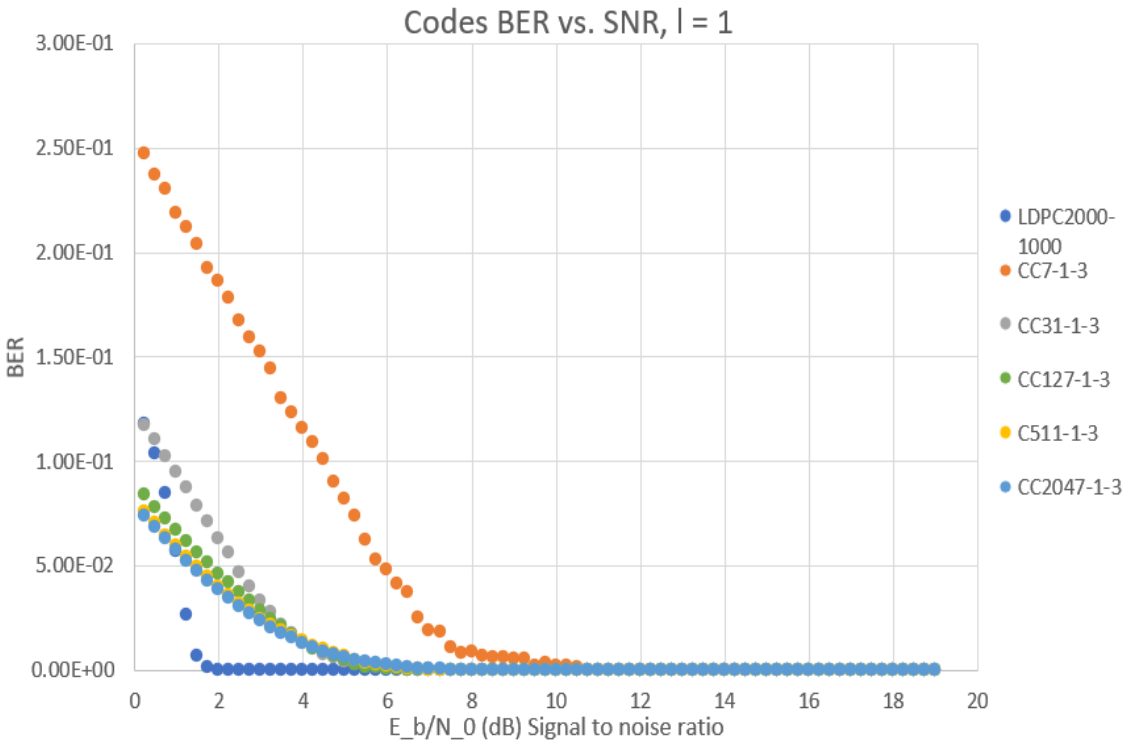


FIGURE 7. Codes with defining set $D = \{1, 3\}$ and the (2000,1000) LDPC code example of girth > 4 from Neal's page.

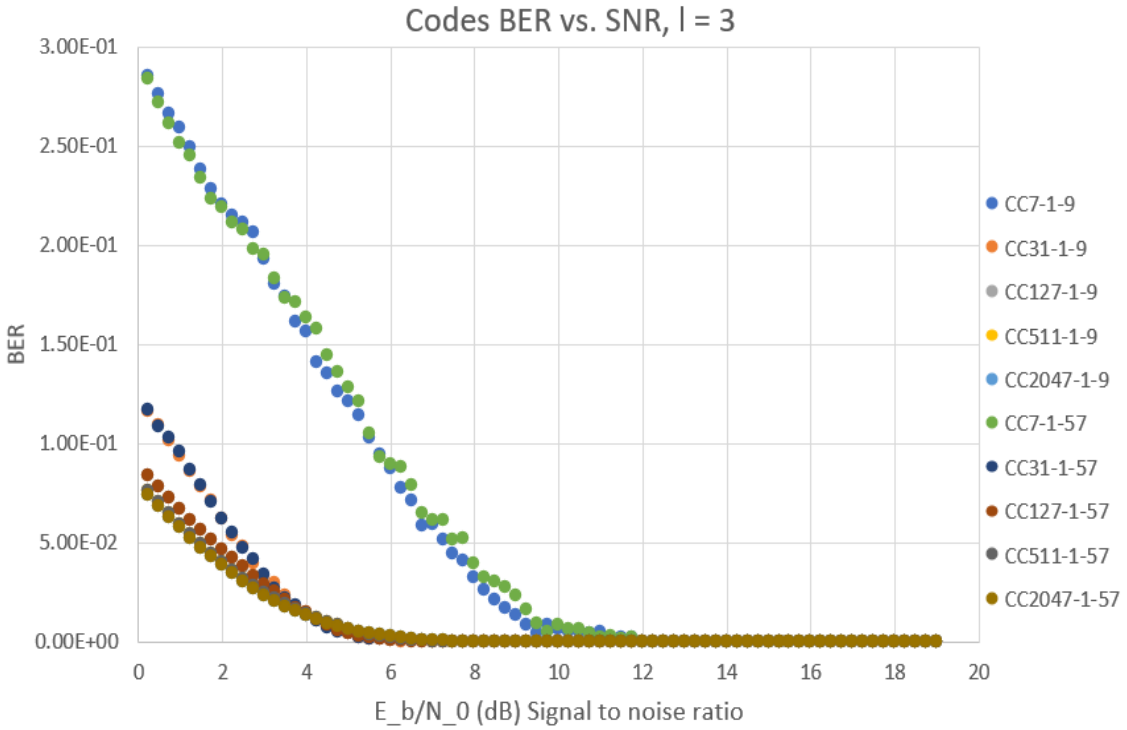


FIGURE 8. Codes from cyclic codes in odd number of variables with defining set $D = \{1, d\}$, for d a Gold or Kasami-Welch exponent $(2^l + 1, 2^{2l} - 2^l + 1)$, with $l = 3$

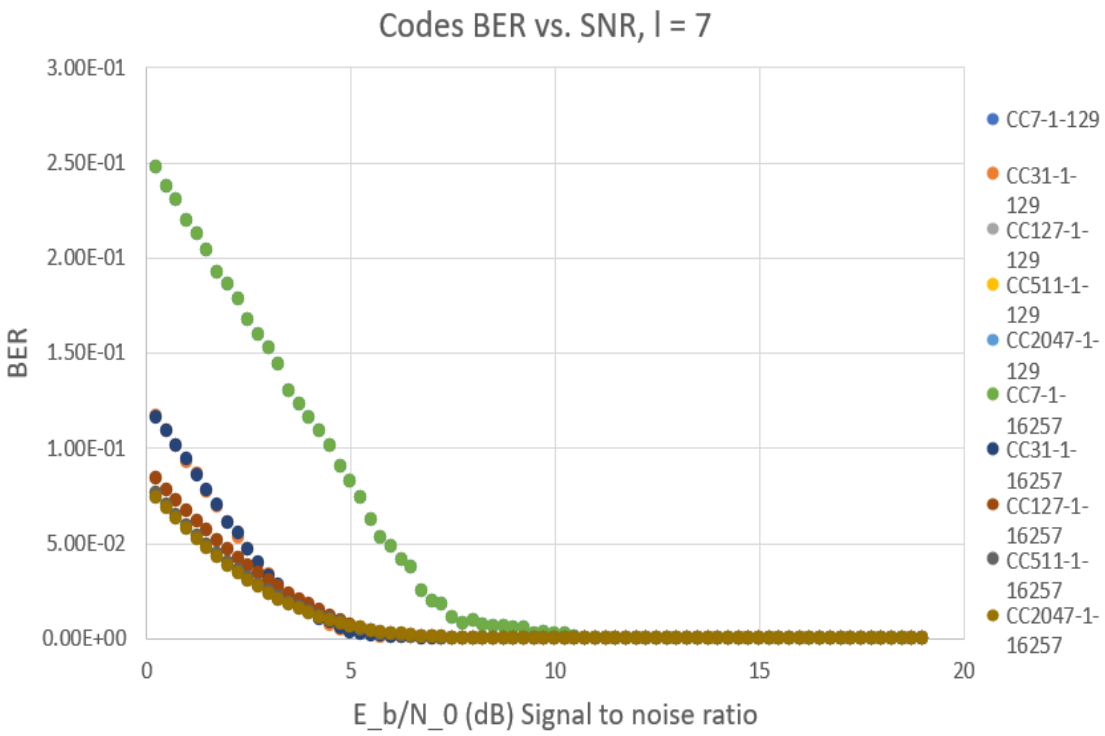


FIGURE 9. Codes from cyclic codes in odd number of variables with defining set $D = \{1, d\}$, for d a Gold or Kasami-Welch exponent $(2^l + 1, 2^{2l} - 2^l + 1)$, with $l = 7$

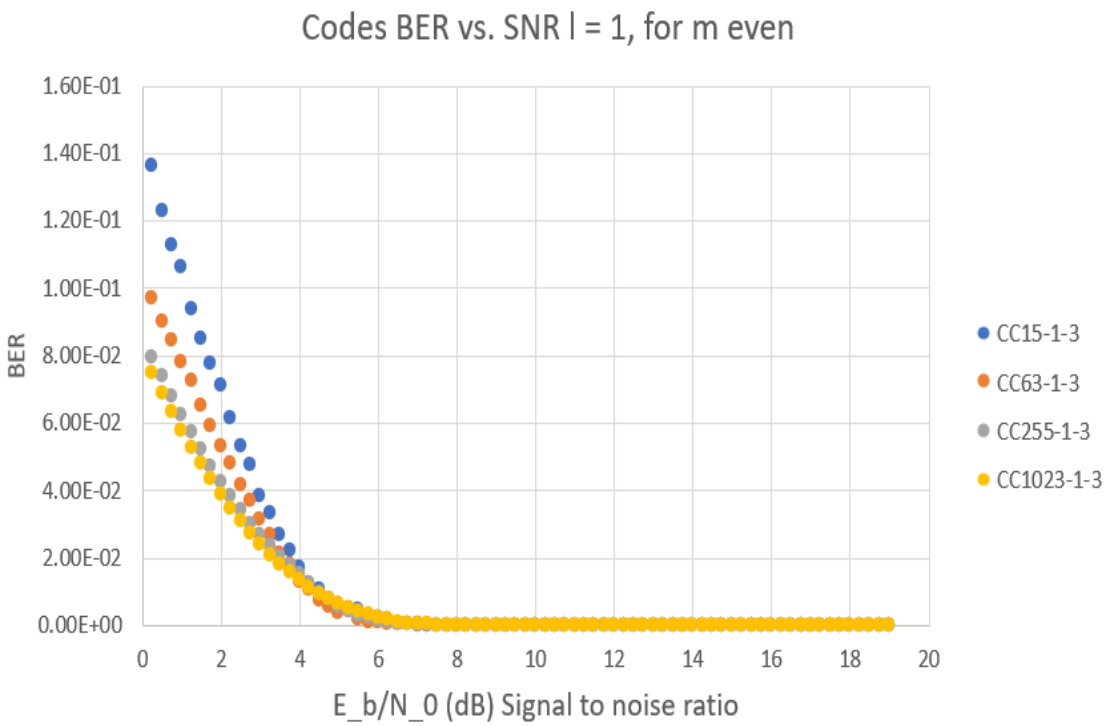


FIGURE 10. Codes from cyclic codes in even number of variables with defining set $D = \{1, d\}$, for d a Gold exponent $(2^l + 1)$, with $l = 1$

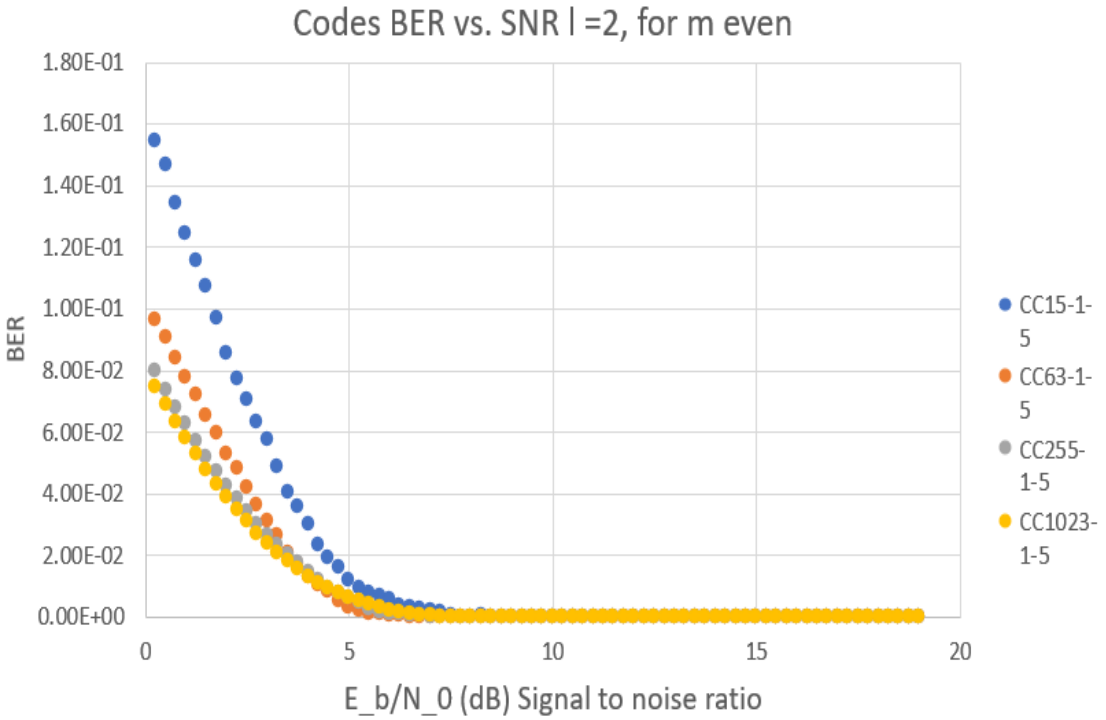


FIGURE 11. Codes from cyclic codes in even number of variables with defining set $D = \{1, d\}$, for d a Gold exponent $(2^l + 1)$, with $l = 2$

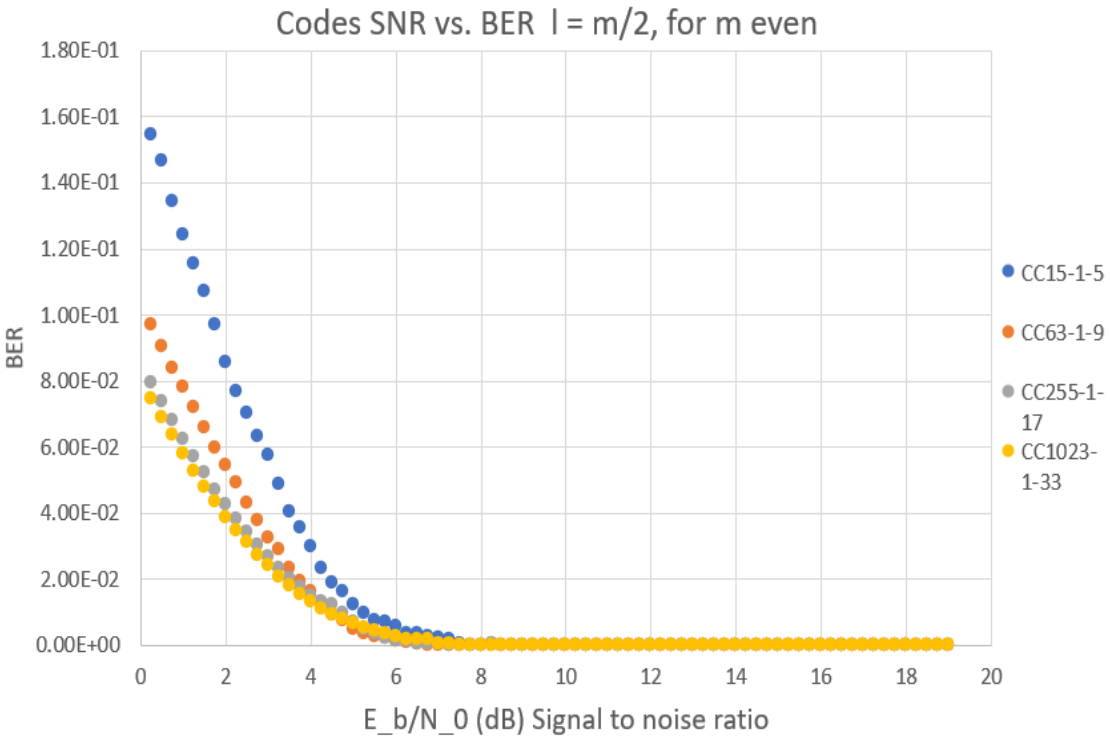


FIGURE 12. Codes from cyclic codes in even number of variables with defining set $D = \{1, d\}$, for d a Gold exponent $(2^l + 1)$, with $l = \frac{m}{2}$

10. APPENDIX:ALGORITHMS

Algorithm 14. def TraceProp(m):

```
    f = FindModNonCube(m)[0]
    R.<x> = GF(2^m,'a', modulus = f)[]
    k.<a> = GF(2**m, modulus = f)
    n = 2^m - 1
    t = (m-1)/(2)
    print("Gold Function ; Nonlinearity ; Algebraic Degree ; Correlation Immunity
; Algebraic Immunity ; Autocorrelation")
    for i in range(1,m):
      d = CycloRep((2^i + 1)%(n),n)
      Gf = BooleanFunction(x^(d))
      print( " $Tr(x^" + str(2^i + 1) + ")$ = $Tr(x^" + str(d) + ")$ ; ", Gf.nonlinearity(),
" ; " ,Gf.algebraic_degree(), " ; ", Gf.correlation_immunity(), " ; ",
Gf.algebraic_immunity()," ; ", Gf.absolute_autocorrelation())
    print("Kasami-Welch Function ; Nonlinearity ; Algebraic Degree ; Correlation
Immunity ; Algebraic Immunity ; Autocorrelation")
    for i in range(1,m):
    d = CycloRep((2^(2*i) - 2^i + 1)%(n),n)
```

```
    Kf = BooleanFunction(x^(d))
    print( "$Tr(x^" + str(2^(2*i) - 2^i + 1) + ")$ = $Tr(x^" + str(d) + ")$ ; ",
Kf.nonlinearity(), " ; " ,Kf.algebraic_degree(), " ; ", Kf.correlation_immunity(),
" ; ", Kf.algebraic_immunity()," ; ", Kf.absolute_autocorrelation())
    if m%2 == 1:
      print("Welch Function ; Nonlinearity ; Algebraic Degree ; Correlation Immunity
; Algebraic Immunity ; Autocorrelation")
      d = CycloRep((2^(t) + 3)%(n), n)
      Wf = BooleanFunction(x^(d))
      print( "$Tr(x^" + str(2^(t) + 3) + ")$ = $Tr(x^" + str(d) + ")$ ; ", Wf.nonlinearity(),
" ; " ,Wf.algebraic_degree(), " ; ", Wf.correlation_immunity(), " ; ",
Wf.algebraic_immunity()," ; ", Wf.absolute_autocorrelation())
      print("Inverse Function ; Nonlinearity ; Algebraic Degree ; Correlation
Immunity ; Algebraic Immunity ; Autocorrelation")
      d = CycloRep((2^m - 2)%(n), n)
      If = BooleanFunction(x^(d))
      print( "$Tr(x^" + str(2^m - 2) + ")$ = $Tr(x^" + str(d) + ")$ ; ", If.nonlinearity(),
" ; " ,If.algebraic_degree(), " ; ", If.correlation_immunity(), " ; ",
If.algebraic_immunity()," ; ", If.absolute_autocorrelation())
      print("Niho Function ; Nonlinearity ; Algebraic Degree ; Correlation Immunity
; Algebraic Immunity ; Autocorrelation")
      if (((m - 1))/2)%2 == 0:
        d = CycloRep((2^(t) + 2^(t/(2)) - 1)%(n),n)
        Nf = BooleanFunction(x^(d))
        print( " $Tr(x^" + str(2^(t) + 2^(t/(2)) - 1) + ")$ = $Tr(x^" + str(d) + ")$ ; ",
Nf.nonlinearity(), " ; " ,Nf.algebraic_degree(), " ; ", Nf.correlation_immunity(),
" ; ", Nf.algebraic_immunity()," ; ", Nf.absolute_autocorrelation())
      else:
        d = CycloRep((2^(t) + 2^((3*(t) + 1)/2) -1)%(n),n)
        Nf = BooleanFunction(x^(d))
        print( "$Tr(x^" + str(2^(t) + 2^((3 * (t) + 1)/2) - 1) + ")$ = $Tr(x^" + str(d) + ")$
; ", Nf.nonlinearity(), " ; " ,Nf.algebraic_degree(), " ; ",
Nf.correlation_immunity(), " ; ", Nf.algebraic_immunity()," ; ",
Nf.absolute_autocorrelation())
      if m%5 == 0:
        print("Dobbertin Function ; Nonlinearity ; Algebraic Degree ; Correlation
Immunity ; Algebraic Immunity ; Autocorrelation")
        d = CycloRep((2^(4 * t) + 2^(3 * t) + 2^(2 * t) + 2^(t) - 1)%(n), n)
        Df = BooleanFunction(x^(d))
        print( "$Tr(x^" + str(2^(4 * t) + 2^(3 * t) + 2^(2 * t) + 2^(t) - 1) + ")$ =
$Tr(x^" + str(d) + ")$ ; ", Df.nonlinearity(), " ; " ,Df.algebraic_degree(), " ;
", Df.correlation_immunity(), " ; ", Df.algebraic_immunity(), " ; ",
Df.absolute_autocorrelation())
```

Algorithm 15. def CyclicCodeClassification(m,sD):

```
    GL = [1]
    KL = []
    EL = []
    t = (m-1)/(2)
    n = 2^m - 1
    DSL = []
    WLL = []
    for i in range(1,m):
      GL.append((2^i + 1)%(2^m - 1))
      KL.append((2^(2*i) - 2^i + 1)%(2^m - 1))
    EL = EL + GL + KL
    if m%2 == 1:
      WL = [(2^(t) + 3)%(2^m - 1)]
      InL = [(2^m - 2)%(2^m - 1)]
      EL = EL + WL + InL
      if (((m - 1))/2)%2 == 0:
        NL = [(2^(t) + 2^(t/(2)) - 1)%(2^m - 1)]
        EL = EL + NL
      else:
        NL = [(2^(t) + 2^((3*(t) + 1)/2) -1)%(2^m - 1)]
        EL = EL + NL
    if m%5 == 0:
      DL = [(2^(4*t) + 2^(3*t) + 2^(2*t) + 2^(t) - 1)%(2^m - 1)]
      EL = EL + DL
    EL = list(set(EL))
    cyclo = combinations(EL, sD)
    cyclo = list(cyclo)
    cyclo2 = []
    for i in range(0,len(cyclo)):
      if cyclo[i][0] == 1:
        cyclo2.append(cyclo[i])
    for i in range(0,len(cyclo2)):
      DS = cyclo2[i]
      C = codes.CyclicCode(field=GF(2), length=n, D = DS)
      h = C.check_polynomial()
      DC = codes.CyclicCode(generator_pol = h.reverse(), length = n)
      sd = DC.spectrum(algorithm = "binary")
      indexlist = [i for i, e in enumerate(sd) if e != 0]
      indexlist = [y for y in indexlist if y != 0]
      DSL.append(DS)
      WLL.append(indexlist)
    WLL2 = WLL
    while len(WLL2) != 0:
```

```
  element = WLL2[0]
  c = [index for (index, element) in enumerate(WLL) if element == WLL2[0]]
  for i in c:
    print(list(DSL[i]), end = "")
  print(" - ",WLL2[0], " - ", len(WLL2[0]))
  WLL2 = [y for y in WLL2 if y != WLL2[0]]
```

Algorithm 16. def DsetClassification(m,DS):

```
GL = []
KL = []
t = (m-1)/(2)
n = 2^m - 1
DSL = []
for i in range(1, m):
  GL.append(CycloRep((2^i + 1)%(n),n))
  KL.append(CycloRep((2^(2*i) - 2^i + 1)%(n),n))
if m%2 == 1:
  WL = [CycloRep((2^(t) + 3)%(n),n)]
  InL = [CycloRep((2^m - 2)%(n),n)]
  if (((m - 1))/2)%2 == 0:
    NL = [CycloRep((2^(t) + 2^(t/(2)) - 1)%(n),n)]
  else:
    NL = [CycloRep((2^(t) + 2^((3*(t) + 1)/2) -1)%(n),n)]
if m%5 == 0:
  DL = [CycloRep((2^(4*t) + 2^(3*t) + 2^(2*t) + 2^(t) - 1)%(n),n)]
print("[1", end = "")
for j in DS[1:]:
  j2 = CycloRep(j,n)
  print(",", end = "")
  if j2 in GL:
    print (" Gold ",end = "")
  if j2 in KL:
    print (" Kasami-Welch ",end = "")
  if m%2 == 1:
    if j2 in WL:
      print (" Welch ",end = "")
    if j2 in InL:
      print (" Inverse ",end = "")
    if j2 in NL:
      print (" Niho ",end = "")
  if m%5 == 0:
    if j2 in DL:
      print (" Dobbertin ",end = "")
print(" ], ", end = '')
```

Algorithm 17. {
```
def makepchkJose(H,d1 = -1, d2 = -1, d3 = - 1, d4 = - 1, d5 = -1):
    l = list(H[:,0])
    k = len(l)
    r - rank(H)
    l2 = list(H[0])
    n = len(l2)
    m = log(n + 1,2)
    t = k/m
    print("make-pchk cyclicLDPC-" + str(n)+ "-len-" +str(m)+"-var-" +str(t) +
"rt", end = "")
    D = [1,d1,d2,d3,d4,d5]
    for j in range(len(D)):
      if D[j] > -1:
        print( "-"+ str(D[j]) +"",end = "" )
    print(".pchk "+ str(k) + " " + str(n) + " ", end = "")
    for i in range(k):
      for j in range(n):
        if H[i,j] == 1:
          print(str(i) + ":" + str(j) + " ", end = "")
        else:
          pass }
```

Algorithm 18. {
```
def makegenJose(m,d1 = -1,d2 = -1,d3 = -1,d4 = -1,d5 = -1):
    D = [1,d1,d2,d3,d4,d5]
    n = 2^m - 1
    for j in range(len(D)):
      if D[j] == -1:
        t = len(D[:j])
        break
    print("make-gen cyclicLDPC-" + str(n)+ "-len-" +str(m)+"-var-" +str(t) +
"rt", end = "")
    for j in range(len(D)):
      if D[j] > -1:
        print( "-"+ str(D[j]) +"",end = "" )
    print(".pchk ", end = "")
    print("cyclicLDPC-" + str(n)+ "-len-" +str(m)+"-var-" +str(t) + "rt", end
= "")
    for j in range(len(D)):
      if D[j] > -1:
        print( "-"+ str(D[j]) +"",end = "" )
    print(".gen dense ") }
```

Algorithm 19. {
```
def randsrcJose(m,d1 = -1,d2 = -1,d3 = -1,d4 = -1,d5 = -1):
    D = [1,d1,d2,d3,d4,d5]
```

```
    n = 2^m - 1
    for j in range(len(D)):
      if D[j] == -1:
        t = len(D[:j])
        break
    k = n - m*t
    print("rand-src cyclicLDPC-" + str(n)+ "-len-" +str(m)+"-var-" +str(t) +
"rt.src 58 " + str(k) + "x" + str(1000)) }
```

Algorithm 20. {def encJose(m,d1 = -1,d2 = -1,d3 = -1,d4 = -1,d5 = -1):

```
    D = [1,d1,d2,d3,d4,d5]
    n = 2^m - 1
    for j in range(len(D)):
      if D[j] == -1:
        t = len(D[:j])
        break
    print("encode cyclicLDPC-" + str(n)+ "-len-" +str(m)+"-var-" +str(t) + "rt",
end = "")
    for j in range(len(D)):
      if D[j] > -1:
        print( "-"+ str(D[j]) +"",end = "" )
    print(".pchk ", end = "")
    print("cyclicLDPC-" + str(n)+ "-len-" +str(m)+"-var-" +str(t) + "rt", end
= "")
    for j in range(len(D)):
      if D[j] > -1:
        print( "-"+ str(D[j]) +"",end = "" )
    print(".gen ", end = "")
    print("cyclicLDPC-" + str(n)+ "-len-" +str(m)+"-var-" +str(t) + "rt.src ",
end = "")
    print("cyclicLDPC-" + str(n)+ "-len-" +str(m)+"-var-" +str(t) + "rt", end
= "")
    for j in range(len(D)):
      if D[j] > -1:
        print( "-"+ str(D[j]) +"",end = "" )
    print(".enc", end = "") }
```

Algorithm 21. {def transmitJose(m,s,d1 = -1,d2 = -1,d3 = -1,d4 = -1,d5 = -1):

```
    D = [1,d1,d2,d3,d4,d5]
    n = 2^m - 1
    for j in range(len(D)):
      if D[j] == -1:
        t = len(D[:j])
        break
```

```
    print("transmit cyclicLDPC-" + str(n)+ "-len-" +str(m)+"-var-" +str(t) +
"rt", end = "")
    for j in range(len(D)):
      if D[j] > -1:
        print( "-"+ str(D[j]) +"",end = "" )
    print(".enc ", end = "")
    print("cyclicLDPC-" + str(n)+ "-len-" +str(m)+"-var-" +str(t) + "rt", end
= "")
    for j in range(len(D)):
      if D[j] > -1:
        print( "-"+ str(D[j]) +"",end = "" )
    print("-" + str(s) + "std.rec 58 awgn " + str(s), end = "") }
```

Algorithm 22. {`def decodeJose(`m`,s,it,d1 = -1,d2 = -1,d3 = -1,d4 = -1,d5 = -1):`

```
    D = [1,d1,d2,d3,d4,d5]
```

$n = 2\hat{}m - 1$

```
    for j in range(len(D)):
      if D[j] == -1:
        t = len(D[:j])
        break
    print("decode cyclicLDPC-" + str(n)+ "-len-" +str(m)+"-var-" +str(t) + "rt",
end = "")
    for j in range(len(D)):
      if D[j] > -1:
        print( "-"+ str(D[j]) +"",end = "" )
    print(".pchk ", end = "")
    print("cyclicLDPC-" + str(n)+ "-len-" +str(m)+"-var-" +str(t) + "rt", end
= "")
    for j in range(len(D)):
      if D[j] > -1:
        print( "-"+ str(D[j]) +"",end = "" )
    print("-" + str(s) + "std.rec " , end = "")
    print("cyclicLDPC-" + str(n)+ "-len-" +str(m)+"-var-" +str(t) + "rt", end
= "")
    for j in range(len(D)):
      if D[j] > -1:
        print( "-"+ str(D[j]) +"",end = "" )
    print("-" + str(s) + "std.dec awgn " +str(s) + " prprp " + str(it)) }
```

Algorithm 23. {`def verifyJose(`m`,s,d1 = -1,d2 = -1,d3 = -1,d4 = -1,d5 = -1):`

```
    D = [1,d1,d2,d3,d4,d5]
```

$n = 2\hat{}m - 1$

```
    for j in range(len(D)):
      if D[j] == -1:
        t = len(D[:j])
```

```
        break
    print("verify cyclicLDPC-" + str(n)+ "-len-" +str(m)+"-var-" +str(t) + "rt",
end = "")
    for j in range(len(D)):
      if D[j] > -1:
        print( "-"+ str(D[j]) +"",end = "" )
    print(".pchk ", end = "")
    print("cyclicLDPC-" + str(n)+ "-len-" +str(m)+"-var-" +str(t) + "rt", end
= "")
    for j in range(len(D)):
      if D[j] > -1:
        print( "-"+ str(D[j]) +"",end = "" )
    print("-" + str(s) + "std.dec " , end = "")
    print("cyclicLDPC-" + str(n)+ "-len-" +str(m)+"-var-" +str(t) + "rt", end
= "")
    for j in range(len(D)):
      if D[j] > -1:
        print( "-"+ str(D[j]) +"",end = "" )
    print(".gen ", end = "")
    print("cyclicLDPC-" + str(n)+ "-len-" +str(m)+"-var-" +str(t) + "rt.src")
  }
```