Summary

This project is mainly focused on whether the Primal-Dual approach can be generalized to a lager field beyond $GF(2)$. And how is the approach generalized.
In coding theory, a linear code is usually denoted by $[n,\ k,\ d]$. n is the length of the code, k is the dimension of the code and d is the Hamming distance. For a linear code, its Hamming distance is also called the primal distance.
A dual code is a linear code that has n - k dimension and it is orthogonal to the code $[n,\ k,\ d]$. So a dual code could be denoted by $[n,\ n-k,\ d^{\perp}]$. $d^{\perp}$ is the Hamming distance of the code. It is also called the dual distance.
The goal of this project is to find several upper and lower bounds for a linear code with given the primal and dual distances. In other words the aim of the project is to find the minimum length of a linear code with given the minimum Hamming distance and the minimum dual distance.
In this project, it is denoted by $N(d,d^{\perp})$. $N(d,d^{\perp}) \geq x$ means the lower bounds found and $N(d,d^{\perp}) \leq x$ means the upper bounds found. To solve the problem, we derive some new inequalities from the existent approach to represent upper and lower bounds in $GF(4)$. And implement these algorithms using *C#.* The exhaustive search is used to find the bound with given small input $(d,d^{\perp})$. Table 10 shows the complete results.
This problem is directly related to the security of Boolean function. And Boolean function's security has respect to the security of the S-Box. S-Box is a very important concept in cryptography especially important for symmetric key algorithms. Therefore, this project in not only a theoretical one but also has some practical meanings.

The list of the main contributions:

1. A theoretical security problem of Boolean function is converted into a practical mathematical problem. See pages 8-11.
2. Three famous bounds which have been proved by some other experts are found. See pages 21-27.
3. Six new inequalities to represent the upper bounds and the lower bounds in $GF(4)$ are derived. See pages 27-35.
4. A comparison of the bounds in the different situations(in $GF(2)$ and $GF(4)$) is provided. See pages 36-37.
5. Some algorithms are created to realize all the bounds with given some small input $(d,d^{\perp})$ to make a test of the new bounds. See pages 38-47.
6. A discussion of the results is given. See pages 48-49.

Abstract

In coding theory, a linear code is always seemed as error-correcting code which is applied in improving the quality of information's transmitting. In computing area, linear code also has special importance in cryptograph. A linear code of length n is a subspace of the finite field F^n over the field F Let a linear $[n, k, d]_q$ code denote a linear code which is a k-dimensional subspace. With a length of n and d is the Hamming distance. In coding theory, we define the dual code is the code in which the n - k dimensional subspace orthogonal to the linear code. A dual code also has a Hamming distance usually denoted by $d^{\perp}$. In the past several years, people have investigated in primal-dual bounds on d or $d^{\perp}$ independently. More interesting, people have recently found taking them together can lead to new insights based on $GF(2)$. Therefore, the aim of my project is to see whether this primal-dual approach can be generalized to larger fields like 4.
This paper demonstrates on what I have researched on the project and gives a brief discussion about what I have discovered.

Acknowledgments

I would thank to my supervisor Martijn Stam who gives me sincerely help, grateful advices and continuous support for this research project
I would thank to Nigel Smart and Dan Page who give invaluable suggestions and feedback in the demonstration presentation.
I would thank to my family who give me the chance to go to the fantastic university.

Contents

Introduction

In modern society, computer is not only used for production but also used for lighting people's life. How to keep the information's security and integrity during its transmission process becomes an important problem. Linear code which is used as a error-correcting code provides a smart method to recover the original data. For cryptograph area, linear code could provide more efficient algorithms for encoding and decoding. Additionally, applying linear code to cryptograph could combine error correcting with system encryption. This combination will heavily reduce the system's working fact.

Suppose a linear code *C* $[n, k, d]$, n is the length of code *C*. k means this code has k dimensions and d is the Hamming distance. Hamming distance here means the total number of different symbols in the corresponding position between two strings which have the same length. More information of the Hamming distance is introduced in *Section 4.3*. More specifically, if *C* is a linear code [10, 4, 5] then it means the whole information length of code *C* is 10 and the actual codeword is 4. Therefore, the number 6(n - k = 10 - 4 = 6) is again the redundancy of *C*. A linear code of length n is also a subspace of the finite field F^n[1] over the field F as it usually works with finite field. People have gained rich experience on linear code and finite field in these years. Primal-dual approach on a finite field is one of the major achievements. Let's look back to the linear code *C*. We say the $n - k$ dimensional subspace orthogonal to code *C* is the dual code of *C*. We denote it as $C^{\perp}$. And it's Hamming distance is $d^{\perp}$. Fortunately, some lower and upper bounds have been found on d or $d^{\perp}$ separately, like famous Gilbert-Varshamov Bound [2], Singleton bound [3] and so on. (These existent bounds are introduced specifically in the Existing Approaches part later.). More interesting, it has been showed that looking d and $d^{\perp}$ together can even bring new ideas in the case of $GF(2)$ from recent researches.

Therefore, the aim of my project is to find out the minimum length n of a linear code *C* which satisfies the minimum Hamming distance d and $d^{\perp}$. For shortly, I try to find $N(d,d^{\perp})$ over the larger field like $GF(4)$. $N(d,d^{\perp})$ here means the minimum length of a linear code with given the minimum Hamming distance d and the dual minimum Hamming distance $d^{\perp}$.

More importantly, the approach of studying the minimum length of a linear code with given the minimum Hamming distance and the minimum dual Hamming distance could have a new insight of the security of Boolean functions. Furthermore, it has a profound effect on the security of S-Box. This is another reason why we study the project.

A good knowledge of coding theory like linear code and having the base of programming such as *C* language are the basic preliminaries of the project. I have grabbed a good knowledge of primal-dual bounds and finite field such as $GF(n)$ through COMS30124—Introduction to Cryptography and COMSM2004—Advanced Cryptography. And beyond the knowledge of lectures I have also prepared some

other source materials. *C* programming language is suitable for exhaustive search to find the bounds. Beyond this I have also found the *C* tutorial[5] which gives me powerful support.

Background of project

The main purpose of the project is to get new lower and higher bounds in $GF(4)$ by taking the minimum Hamming distance d and the minimum dual Hamming distance $d^{\perp}$ together. In other words we want to find the minimum length of a linear code with given a minimum Hamming distance d and a Hamming distance $d^{\perp}$ of the dual code. More specifically, some advanced restrictions will be found out at the end the project on the base of existing approaches. So the key point to the project is to derive new bounds from the existent bounds by some intelligent mathematic skills in the large fields like $GF(4)$.

In general terms, this project is mainly a Type III project. It is built on some famous theories which have validated now. For my project, some basic preliminaries are needed to know before doing the project. For example, some background of linear code, finite field and Hamming weight are indispensable. Comparing with some practical projects, my project's requirement is not clear enough. It gives me a big direction on a scope like 'Primal-Dual bounds beyond $GF(2)$', rather than gives me a specific implementation requirement. As an MSc project I think it is more important to extend the existed achievements to some new domains. So I try to apply the Primal-Dual bounds to new large fields like $GF(4)$. And for my project, it requires more advanced intelligent mathematic skills. So I think it is generally a theoretical project. Some bounds need to be realized so I implement the bounds using *C* programming language [5].

As the project being a theoretical one, so the main methodology is mathematic inferences and some other theoretical discussing. After the mathematic model is finished to build, I use some programming language to implement it like *C*. According to my estimate, at least 80% part of the project is theoretical part. And the programming part takes at most 20% of the whole project.

A large amount of relative materials are the keys to the successful. So I prepared some literatures for my project. Firstly, I searched on Wikipedia to find out some basic concepts. Though Wikipedia can't be used as a reference, I think it is still a very useful tool to give me a superficial understanding of my project. For example, what is a linear code, what is finite field and so on? However, Wikipedia can't provide enough support for me for successfully finishing the project. I collect some academic papers which I think is related to my project closely. For example, one of them is 'Primal-dual distance bounds of linear codes with application to cryptography' [4] which published by Ryutaroh Matsumoto,etd. It gives a novel idea about the upper bounds and lower bounds on the minimum length n of a linear code with a given Hamming distance and a dual Hamming distance in the case of $GF(2)$. This literature is the most important part of all the literatures. All the literatures I prepared are

provided in references.

Why do we need to study this project?

Linear codes as an efficiency error-correction codes play an important role in many scopes especially in computing security. We focus on the relationship between the linear codes and the computing security in this part. In other word it shows why we study this project.
As we have mentioned in the beginning of the report I need to find the minimum length of a given linear code which also has a Hamming distance and the Hamming distance of its dual code over a larger size finite field like $GF(4)$. This problem is related to the security of cryptographic Boolean functions closely. So I think this project is not only a task for an MSc student but also gives some contribution on the security of cryptography.

3.1 AES algorithm and S-box

3.1.1 AES algorithm

Advanced Encryption Standard [6] algorithm which is usually known as AES is a clever symmetric key encryption algorithm with a high level of security. National Institute of Standards and Technology(NIST) of the United States of America finally chose the AES encryption algorithm as a substitute of DES algorithm in 2000. It is still to be seen as a security algorithm until now.
AES algorithm is roughly based on the Substitution and Permutation. It has three different fixed block ciphers AES-128, AES-192 and AES-256 input and the corresponding keys also have three different fixed sizes 128, 192 and 256. So, the longer block cipher means more security of the encryption. These ciphers can be implemented in software using logical and arithmetic operations. These operations usually include *SubBytes*, *ShiftRows*, *MixColumns* and *AddRoundKey*. These four operations are really important for AES algorithm. The algorithm's security is directly related to them.
In the processing of encryption, there are several rounds implemented. Each round contains the four steps except the last round excluding the operation of *MixColumns*.
SubBytes is an operation that each byte is replaced by another through the lookup table which is known as well as S-box.
ShiftRows is easy to understand from its name. This operation shifts each row according to some fixed numbers
MixColumns is a mixing operation which operates on each column with combining the four bytes in each column.
AddRoundKey is an operation which adds the subkey to each round.
In the whole four operations, *SubBytes* is an operation based on Substitution while

ShiftRows and *MixColumns* are two Permutation operations. So we can conclude that AES algorithm is an encryption method based on the Substitution and Permutation. The Substitution operation is also based on the Substitution-Box which is known as well as S-Box in AES algorithm. Therefore, we can say the security of S-Box is closely related to the security of AES. In fact, many other symmetric key algorithms are based on S-Box. The detail of S-Box is introduced in the following part.

3.1.2 Substitution-Box

Substitution-Box as a basic component of an encryption algorithm which performs the substitution operation is usually used to obscure the relationship. Obviously it is a non-linear operation in most encryption algorithms, because the box is determined previously. Actually in AES, S-Box is only one non-linear component so its security determines the algorithm's security in this level. The most interesting point is the construction of S-Box.

Definition[7]: A function of the form S: $GF(2)^n \rightarrow GF(2)^m$ is called an $n \times m$ *S-Box,* which takes n bits as the input and output m bits. If each output bit is called the n-variable Boolean function f_i, then $S(x) = (f_1(x), \dots, f_m(x))$ where $x \in GF(2)^n$. ($GF(2)^n$ here means the finite field see *Section4.2.1*)

From the definition we can see clearly that S-Box operates on some bits as input and output some other bit. For example input n bits and output m bits. This property is extremely like a Boolean function.

Definition[7]: A *Boolean function:* $GF(2)^n \rightarrow GF(2)$ is the function which has the input all of the possible n tuples $x = (x_1, .., x_n)$ of $GF(2)$ and produces an output of one bit. The set of all n-variables Boolean function are denoted by V_n.

Comparing the two definitions, it is much clearer to see that S-Box could be seen as a set of Boolean functions. Boolean function makes n bits input and 1 bit output while S-Box makes n bits input and m bits output. So S-box could be treated as a set of m Boolean functions. Therefore, the security of S-Box is closely related to the security of Boolean function. Moreover, S-Box's security roughly equals the security of AES algorithm. Then we can deduce the security of Boolean function determines the security of AES encryption algorithm in some level. Actually, most symmetric key encryption algorithms are related to the Boolean function not only the AES algorithm. The aim of my project is to find the minimum length of a linear code with given the minimum Hamming distance and the minimum Dual Hamming distance. The minimum length of a linear code has special relationship with the security of Boolean function (This is explained in *Section3.2*).Therefore, it has important meanings for the security of some symmetric key encryption algorithms like AES. This is why I do the project

AES encryption algorithm and S-Box are two very clever encryption methods but it is not the theme of this project. So I don't go far more beyond this. However, they are good open questions.

3.2 The minimum length and Security

In cryptographic security, Boolean function [8] plays a critical role especially for a symmetry key algorithm like AES.[6] In cryptography theory, a Boolean function is function like $f\colon B^k \to B$. The value of B is in the domain {0,1} and k is a non-negative integer . In the last part we have known a symmetry key algorithm's security is directly related to the S-Box's security. And S-Box could be considered as a set of Boolean functions. Therefore, we can roughly believe that Boolean function's security equals to the algorithm's security.

It is obviously that a Boolean function usually has a truth table which contains 0 and 1 entries. If the number of 0 entry equals the number of 1 entry then we define the Boolean function is balanced. For any positive integers n and l, we say a Boolean function satisfies propagation criterion [9] $PC(l)$ if for any nonzero word e, the Boolean function $f(x) + f(x+1)$ is balanced, and $1 \leq \omega t(e) \leq l$. $\omega t(e)$ here denotes the Hamming weight of e. So we can conclude that no matter what different e input the output of the Boolean function is balanced by the same number of entry 0 and entry 1 if it satisfies the propagation criterion. In other words if a Boolean function satisfies $PC(l)$ then $f(x) + f(x+1)$ is uniformly distributed for any e. So this criterion is extremely important for defending the differential attacks, because e could be seen as the input difference and $f(x) + f(x+1)$ is the difference of output.

Another famous criterion is called SAC (strict avalanche criterion). 'An S-box satisfies the strict avalanche criterion (SAC), if and only if for any single input bit of the S-box, the inversion of it changes each output bit with probability one half' [10] It is obviously a generalized version of avalanche effect. It means that if each bit changes as input then each bit of the output will change with a probability at least 50%. So we can clearly know that SAC is directly related to the S-Box's security from the definition. This criterion is also used to against the difference attacks so we can treat it as equal as the propagation criterion in the opinion of S-Box's security.

After providing the propagation criterion Bart Preneel[24] introduced another stronger notion. That is the extended propagation criteria $EPC(l)$ [9]. We say if $f(x)$ satisfies $PC(l)$ even if any k bits of $x = (x_1, x_2 \ldots, x_{n-1}, x_n)$ are fixed to any constant then $f(x)$ satisfies the $EPC(l)$ of order k. Kurosawa et al.[11] were first to try to give the construction method of $EPC(l)$ on the foundation of Miorana-McFarland construction. They creatively found that if a linear code C exists with a given minimum Hamming distance d and a given minimum Hamming distance $d^{\perp}$ of $C^{\perp}$. There must be an $EPC(l)$ of order k and the Hamming distance $d = k + 1$, $d^{\perp} = l + 1$.

In other words, if we can find the minimum length n of a linear code C which has the minimum Hamming distance d and $d^{\perp}$, we can prove that there must be a

extended propagation criterion $EPC(l)$ somewhere. And the Boolean function can be roughly believed security. Then we can deduce the S-Box is security.
We convert a theoretical problem into a mathematical problem successfully. Through the project we want to explain a cryptographic security question from the point of the view of mathematical.

What is a linear code?

4.1 Forward error correction

Firstly, before we introduce what linear code is another important definition Forward Error Correction should be explained.
In the data transmission it is unavoidable to produce some errors during the. So how to control errors for data transmission has been a hard challenge both in information theory and in telecommunication. One efficient solution is called forward error correction (FEC). Richard Hamming [12] an American mathematician firstly focused on this point and created the first forward error correction code in 1949. The essential part of this technique is to add redundant data to its message before the sending. Senders add redundancy to the original data which will be transmitted according to some relevant predetermined algorithm. After transmitting the receivers could detect and correct the data errors using the redundancy. For example, a code (0, 0) may be changed to (0, 0, 1) by simply being added a redundant bit. The redundancy of a code is transmitted at the same time with the transmitting of original code. Actually, they are transmitted as a whole body. Therefore, it allowed the receivers to find and correct the errors without the need to ask the senders some additional information. In other words, it doesn't need the second communication between the sender and receiver. So it is a one way method to correct errors.
It can be seen easily that FEC work by adding redundancy to the transmitted information according to some predetermined algorithm. So using different algorithms may come into being different error correcting codes. Block code is one of the common types of error correcting codes. It is used widely to lower the bit error rate (BER).
In coding theory, block code is one of the famous code type. During the process of transmission of information each message is divided into several message blocks. Each message block has the same length, therefore the whole message are averagely transformed into several blocks. These blocks are called block codes. Each block contains a fixed number of information and change the information which may be represented by symbols over an alphabet set $\sum$ into a fixed length sequence of n encoding symbols. A block code is usually denoted by [n ,k]. For example, a code C [n ,k] means this block has k information and the k information is encoded into a new code of n length. The number n - k is again the redundancy of code C.
When k and n - k has a linear relationship we called this code is a linear code.

We can see this transformation on geometry. It produces some distances between some points through the expansion of the space like figure 1

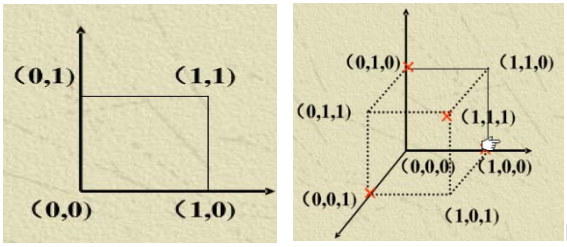

Figure 1

4.2 Linear code

4.2.1 Galois Field

In abstract algebra, the Galois Field[1] is also called the finite field. It is easy to understand what this kind of field is from its name. If a field that contains a finite number of elements we say this is a finite field. Galois Field is a very important concept in coding theory especially for linear codes. Differing from other fields like rational number field, Galois Field is classified by its size. A finite field is defined as p^n, where p is a prime number and n is a positive integer. Therefore, the total number of elements of the finite field is p^n. We also call this is the order of the field. Let another integer $q = p^n$. We denote the Galois Field as $GF(q)$. For instance, if p = 2 and n = 1, the field has p^n = 2 elements. We denote this field as GF(*2*). If p = 2 and n = 2, the field has p^n = 4 elements. We denote this field as $GF(4)$ which is closely related with our article. We will discuss this later. For every prime number p and positive integer n there must be a finite field. If two finite fields which have the same elements or have the same order, the two fields are isomorphic. Therefore, if a field's order is given, the finite field is fixed uniquely. Galois Field is a special field so the operations are also different from other fields. For a finite field which contains *q* elements, given any element q_1 and the positive integer n, then q_1 mod n. Modular arithmetic is a operation that get the reminder. For example, 7 mod 2 = 1, because 7 – (2*3)=1. Apart from the modular arithmetic there are many other arithmetic like Field polynomial, Reduction and so on. In finite field, there are two main operations. They are Addition and Multiplication. Differing from on other fields, addition in Galois Field is simply XOR operation. It means the difference. For instance, 0 + 0 = 0, 0 + 1 = 1, 1 + 0 = 1 and 1 +1 = 0. The Multiplication operation is a little complicated. It can be seen as the ordinary binary multiplication. However if the result is not in the given field another reduction will be applied.
From the definition we can know that for a finite field. The number of its elements must be some prime's power. This prime is the characteristic of the finite field. In coding theory $GF(2^n)$ is widely used and has important significance. For my project, $GF(2)$ and $GF(4)$ are two important concepts. When p = 2 and n = 1 the finite field is denoted by $GF(2)$. This finite field has two elements and it is the smallest

finite field. This is easy to understand because p = 2 is the smallest prime and n = 1 is the smallest positive integer. In generally the two elements are always set by 0 and 1. A linear code which on $GF(2)$ is called a binary code. When p = 2 and n = 2 the finite field is a little larger. It has four elements so we call it $GF(4)$. It can be defined as an extension of a binary field. It has four elements so differing from $GF(2)$ they may be represented by {0,1,2,3}.

4.2.2 Linear code

In coding theory a linear code as a block code is usually used to control errors of data transmission. The length of a linear code is the whole information this code contains. And dimension k means the real information which is always the length of codewords. So, $n - k$ is the redundancy codewords. The property of redundancy can be used to correct errors efficiently. We will discuss this point specifically later. Obviously, it is on the base of abstract algebra because linear code is usually connected with Galois Field. From *Section 4.1* we can see that a linear code is a special block code. Therefore, a linear code C could also be represented as $[n, k]$. We usually denote a linear code by $[n, k]_q$. This means this code is a linear code of length n over the field F_q. So in other words, linear codes are special sets of words of the length n over an alphabet {0,1,…, *q* -1}, where *q* is a power of some prime. The alphabet set {0,1,…, *q* -1} is called the Galois Field. From now on, we can regard a linear code $[n, k]_q$ as a vector space $V(n, q)$. More specifically, a linear code with length n and rank k is a linear subspace of the vector space $V(n, q)$ with k dimension. This linear code is a k dimension subspace of vector space $V(n, q)$, so it has q^k codewords. For example if a $[n, k]_q$ linear code *C* exists, we can say it is a linear subspace *C* with dimension k of the vector space F_q^n where F_q is the finite field with *q* elements. We also say this linear code C is a *q*-ary code. For example, if *q* = 2, code *C* is called a binary code. If *q* = 3, *C* is a ternary code.
Until now, we have introduced some basic definitions about linear codes and then several important properties of linear codes will be brought in.

Property 1: Any two components' summation is another component in the same vector space. This definition may be a little obscure. We can transform it into a mathematic formula as following:
If *C* is a linear code it
(1) $C \subseteq V(n, q)$
(2) $a + b \in C$ for all $a \in C, b \in C$
We can see a simple instance to show the property.
C_0 = {00, 01, 10, 11} is obviously a linear code in $GF(2)$. It can be seen that 00 + 01 = 01, 00 + 10 = 10, 01 + 11 = 10, …. It satisfied *Property 1.*
C_1 = {01, 10} is not a linear code. Because 01 + 10 =11 which is not in the code C_1.

Property 2:

If *C* is a linear code it must have the zero word. This property is so easy to understand.

Property 3:
In linear code, the minimum Hamming distance of different codes is as the same as Hamming weight of non-zero codes.
Hamming weight is the number of non-zero entries of code *c*. It is denoted by $\omega t(c)$.
Hamming distance and Hamming weight are two extremely important concepts in linear code and they are especially important for my project. More information about them is explained in the *Section 4.3*.
Therefore, a linear code is a code which converts a codeword of length k into a codeword of length n by adding $n - k$ check codewords in essence.

4.2.3 Dual code

Since now no, we know a linear code $[n, k]_q$ is a linear subspace of the vector space F_q^n with dimension k. The $n - k$ dimensional subspace which is orthogonal to the linear code $[n, k]_q$ is known as the dual code in coding theory. It is another important approach in coding theory. This definition can be converted into a math definition like following:
(1) $C \subseteq V(n,q)$ is a linear code $[n, k]_q$
(2) The dual code of code *C* as

$$C^{\perp} = \{ a \in V \mid a \cdot b = 0 \text{ for every } b \in C \}$$

From the definition we know the dual code $C^{\perp}$ is also a linear code but which is special in having $n - k$ dimension. So, it also can be denoted as $C^{\perp}$: $[n, n-k]_q$.

This property is really interesting and plays an important role in the formula's derivations which we discuss later.

As $k + n - k = n$ so we can get another property of dual code $C^{\perp}$

Property : $dim(C) + dim(C^{\perp}) = n$

If code *C* has Hamming distance d(see *Section 4.3*) then the dual code $C^{\perp}$ has dual Hamming distance $d^{\perp}$ correspondingly. This project is mainly on linear codes, so it needs to firstly introduce what linear codes and dual codes are.

4.3 Hamming weight and Hamming distance

In this section some important concepts which are related to Hamming weight and Hamming distance are introduced. They are directly related to the ability of error detecting and error correcting for a linear code.
For a linear code *V* the Hamming weight of *V* is the number of non-zero codeword in *V*, denoted by $\omega t(v)$ [13]. For example, if code *V* = (010101) then $\omega t(v) = 3$. The minimum non-zero codeword is called the minimum Hamming weight of *V*, usually denoted by $\omega t_{\min}(v)$.

$$\omega t_{\min}(v) = \min\{\ \omega t(x)\ \mid\ x \in V, x \neq 0\ \}$$

Comparing with Hamming weight Hamming distance plays a more important role in coding theory. In information theory, a Hamming distance between two strings which the two strings have the same length is the number of different symbols in the corresponding position. This definition may be obscure. Giving some examples may be much clearer. For example in the *figure 2*, the Hamming distance between two strings '111000' and '000111' is 6. For binary strings Hamming distance is in a Hamming cube. For example, the following Hamming distance between '100' and '000' is the path form '100' to '000' in the cube.

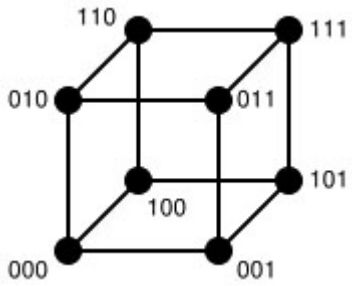

Figure 2
Similarly, the Hamming distance of a linear code could be defined as follows:
(1) $P \in V, \quad Q \in V$
(2) $d(P,Q) = \sum_{i=0}^{n-1}(p_i \oplus q_i)$
As we know a codeword of a linear code could be considered as a point of vector space. So the Hamming distance is the distance between two corresponding points in the space. Hamming distance as a special distance it also satisfies some basic axioms:
(1) $d(P,Q) \geq 0$
(2) $d(P,Q) = d(Q,P)$
In all codeowrds of linear code *V*, the minimum distance between any two codeword is the minimum Hamming distance of *V*. The minimum distance could be defined as below:

$$d_{\min} = \min\{\ d(C_i, C_j)\}, \quad i, j\ = 0,1,\ldots,\ 2^k - 1,\ i \neq j$$

From *Property 3* in *Section 4.2.2* we know for linear codes the minimum Hamming distance = the minimum Hamming weight over a field. Therefore, the minimum

Hamming distance of the linear code *V* is the minimum weight of the code *V*.
Proof : For a linear code *V* $[n, k]_q$, $P \in V$, $Q \in V$, and $Z = P - Q$, so $Z \in V$. Because linear code is a closed space. So we can get Z as the third vector must in the space *V* from the vector arithmetic. Therefore,

$$d(Q, P) = d(Q - P) = \omega t(Z).$$

From this equation it could infer the minimum Hamming distance equals the minimum Hamming weight. This property is very important for this project especially in formulas' derivations.
As we have mentioned above the minimum Hamming distance is an important parameter for ability for detecting and correcting the errors. Normally if the minimum Hamming distance is larger the ability of error detecting and correcting of the linear code is more powerful. For a simpler understanding a linear code's minimum distance larger means the differences between two codewords in this linear code is bigger. So the ability of error correcting and detecting is more powerful.
Through studying it has been proved that if there are l errors being detected that the minimum Hamming distance

$$d_{\min} \geq l + 1.[14]$$

If there are l errors being corrected then the minimum Hamming distance

$$d_{\min} \geq 2l + 1.$$

If there are t errors being corrected and l errors being detected at the same time, then the minimum Hamming distance

$$d_{\min} \geq l + t + 1.$$

Generator Matrix and Parity Check Matrix

5.1 Generator Matrix

This section introduces the Generator Matrix. In linear algebra, a family of vectors is linearly independent[15] if any one vector of this correction is can't be represented by any other vectors in this correction. It seems that this definition is a litter complex. We can give a simple example to show what linearly independent is. R3 is a vector space in 3-dimension Euclid space[16]. A = (1,0,0), B = (0,1,0) and C = (0,0,1) are three vectors in R3. We say these three vectors are linearly independent (A $\neq$ B + C, B $\neq$ A + C and C $\neq$ A + B). However, another three vectors A' = (2,1,1), B' = (1,0,1) and C' = (3,1,2) are not linearly independent. Because we can see C' = A' + B'. In other words vector C' could be represented by vector A' and vector B'.
From *Section 4.2.2* we know a linear code $[n,\ k,\ d]$ is linear vector space with k dimension, so there must exist k codewords which are linearly independent in this space. In geometry this means there must be k linearly independent vectors which could be expended to the whole vector space. So any codeword of the linear code could be represented by this group of linearly independent vectors.

We denote $y_1, y_2, \cdots, y_{\mathrm{k}}$ are k linearly independent vectors of space *V*(n, q). Y is any codeword in code $[n,\ k,\ d]$. So

$$Y = x_1 y_1 + x_2 y_2 + \cdots + x_{\mathrm{k}} y_{\mathrm{k}} = (x_1, x_2, \cdots, x_{\mathrm{k}}) \begin{pmatrix} y_1 \\ y_2 \\ \vdots \\ y_{\mathrm{k}} \end{pmatrix} = X\ G$$

X here is the message and G is called the generator matrix of linear code $[n,\ k,\ d]$ which has a special position in linear codes theory. X is a message so the generator matrix G produce the codeword which is corresponding to the particular message X. In the formula above we know $G = \begin{pmatrix} y_1 \\ y_2 \\ \vdots \\ y_{\mathrm{k}} \end{pmatrix}$, actually it should be written as

$$G = \begin{pmatrix} y_1 \\ y_2 \\ \vdots \\ y_{\mathrm{k}} \end{pmatrix} = \begin{pmatrix} y_{11} & y_{12} & \cdots & \cdots & \cdots & \cdots & y_{1n} \\ y_{21} & y_{22} & \cdots & \cdots & \cdots & \cdots & y_{2n} \\ \cdots & \cdots & \cdots & \cdots & \cdots & \cdots & \cdots \\ y_{k1} & y_{k2} & \cdots & \cdots & \cdots & \cdots & y_{kn} \end{pmatrix}.$$

Each row in the genitor matrix $y_i = (y_{i1}, y_{i2}, \ldots, y_{in})$ represents a codeword. A

Generator Matrix is a basis for a linear code. So any possible codewords could be generated by it. Each codeword of the linear code $[n, k]$ is a linear combination of some row vectors of the genitor matrix, so the total 2^k codwords construct the subspace of vector space *V* with dimension k which is expended by the rows of the genitor matrix. In other words it explains the definition of linear codes in another way. In a more general term, a generator matrix can be defined like:

$$G = [I_k \mid P]$$

In this formula, I_k is a $k \times k$ identity matrix. In general I_k =

$$\begin{bmatrix} 1 & 0 & \cdots & 0 \\ 0 & 1 & \cdots & 0 \\ \vdots & \vdots & \ddots & \vdots \\ 0 & 0 & \cdots & 1 \end{bmatrix}$$

There are k '1' in the diagonal.
And the other part of the formula '*P*' is of dimension $k \times (n - k)$.
For example, we have known in the above instance:

$$G = \begin{pmatrix} y_1 \\ y_2 \\ \vdots \\ y_k \end{pmatrix} = \begin{pmatrix} y_{11} & y_{12} & \cdots & \cdots & \cdots & \cdots & y_{1n} \\ y_{21} & y_{22} & \cdots & \cdots & \cdots & \cdots & y_{2n} \\ \cdots & \cdots & \cdots & \cdots & \cdots & \cdots & \cdots \\ y_{k1} & y_{k2} & \cdots & \cdots & \cdots & \cdots & y_{kn} \end{pmatrix}.$$

This is not a standardized generator matrix. Through primary permutation G has been a generator which has a form like $[I_k \mid P]$:

$$G = \begin{pmatrix} y_1 \\ y_2 \\ \vdots \\ y_k \end{pmatrix} = \begin{pmatrix} y_{11} & y_{12} & \cdots & \cdots & \cdots & \cdots & y_{1n} \\ y_{21} & y_{22} & \cdots & \cdots & \cdots & \cdots & y_{2n} \\ \cdots & \cdots & \cdots & \cdots & \cdots & \cdots & \cdots \\ y_{k1} & y_{k2} & \cdots & \cdots & \cdots & \cdots & y_{kn} \end{pmatrix} =$$

$$\begin{pmatrix} 1 & 0 & \cdots & 0 & y_{11} & \cdots & y_{1(n-k)} \\ 0 & 1 & \cdots & 0 & y_{21} & \cdots & y_{2(n-k)} \\ \cdots & \cdots & \cdots & \cdots & \cdots & \cdots & \cdots \\ 0 & 0 & \cdots & 1 & y_{k1} & \cdots & y_{k(n-k)} \end{pmatrix} = [I_k \mid P]$$

It may be still a little confused about how a matrix can generate a linear code. Let's see a specific example. The generator matrix of linear code [7, 4] is as follows:

$$G = \begin{pmatrix} 1 & 0 & \dots & 0 & y_{11} & \dots & y_{1(n-k)} \\ 0 & 1 & \dots & 0 & y_{21} & .. & y_{2(n-k)} \\ \dots & \dots & \dots & \dots & \dots & \dots & \dots \\ 0 & 0 & \dots & 1 & y_{k1} & \dots & y_{k(n-k)} \end{pmatrix} = \begin{pmatrix} 1 & 0 & 0 & 0 & 1 & 0 & 1 \\ 0 & 1 & 0 & 0 & 1 & 1 & 1 \\ 0 & 0 & 1 & 0 & 1 & 1 & 0 \\ 0 & 0 & 0 & 1 & 0 & 1 & 1 \end{pmatrix}$$

We have known $Y = X\ G$.

If X = (1010), then

$$Y = [1010] \begin{pmatrix} 1 & 0 & 0 & 0 & 1 & 0 & 1 \\ 0 & 1 & 0 & 0 & 1 & 1 & 1 \\ 0 & 0 & 1 & 0 & 1 & 1 & 0 \\ 0 & 0 & 0 & 1 & 0 & 1 & 1 \end{pmatrix} = (1010011).$$

5.2 Parity Check Matrix

Comparing with the generator matrix of linear code C, its dual code $C^{\perp}$' generator matrix is called a parity check matrix. So we can derive the parity matrix for a given code from its generator matrix easily.
If a generator matrix of a linear code is given:

$$G = [I_k \mid P]$$

Then the parity check matrix is also known:

$$H = [-P^T \mid I_{n-k}]$$

From these two defenitions we can know there may be some relationships between the generator matrix and the Parity check matrix, because it looks like that I and P change the position in the two formulas. Actually, in linear codes, parity check matrix can be transformed into generator matrix directly. The generator matrix also can be transformed into the parity check matrix.

Proof: From the definition above it can be got that $G_{k\times n}H^T_{n\times(n-k)} = 0_{k\times(n-k)}$. Let's denote:

$G = [I_k \;\; Q_{k\times(n-k)}]$, and $H = [P_{k\times(n-k)} \;\; I_{n-k}]$

So $G \times H^T = [\mathrm{I_k}\ \mathrm{Q_{k\times(n-k)}} \mid \mathrm{P_{k\times(n-k)}}\ \mathrm{I_{n-k}}]^T = [\mathrm{I_k}\ \mathrm{Q_{k\times(n-k)}} \mid [{}^{(P_{k\times(n-k)})^T}_{I_{n-k}}]] =$

$(P_{k\times(n-k)})^T + \mathrm{Q_{k\times(n-k)}} = 0_{k\times(n-k)}$.

Therefore,

$\mathrm{Q_{k\times(n-k)}} = (P_{k\times(n-k)})^T$.

We can get $\begin{cases} G = [\mathrm{I_k}\ (P_{k\times(n-k)})^T] \\ H = [P_{k\times(n-k)}\ I_{n-k}] \end{cases}$

Let's see a simple example. If the parity check matrix of the code $[6,\ 3]$ is

$$H = \begin{pmatrix} 1 & 0 & 0 & 1 & 1 & 0 \\ 0 & 1 & 0 & 1 & 0 & 1 \\ 0 & 0 & 1 & 0 & 1 & 1 \end{pmatrix}$$

then we can get the generator matrix of code$[6,\ 3]$ directly through the property:

$$G = \begin{pmatrix} 1 & 1 & 0 & 1 & 0 & 0 \\ 1 & 0 & 1 & 0 & 1 & 0 \\ 0 & 1 & 1 & 0 & 0 & 1 \end{pmatrix}$$

Several Bounds of Linear codes

Linear codes as an important concept both in coding theory and Internet security has being studied by many specialists in past decades. One of the fundamental problems is to find the minimum length of linear codes with given the minimum Hamming distance and the number of codewords. In this section three famous bounds are introduced. Firstly, the Singleton bound is introduced. And then we have a briefly explanation about Gilbert-Varshamov bound. Finally, the famous bound Hamming bound is brought in. These bounds are foundations of our project because we get the minimum length of a code with given the minimum Hamming distance and the minimum Dual Hamming distance on the base of these bounds' derivatives.

6.1 Singleton bounds

Singleton bound [17] which is named after Richard Collom Singleton (1964), is a very simple bound in representation. However, it is very important in coding theory, especially for linear codes.
The famous Singleton bound can be described as following:

$$A_q(n,d) \leq q^{n-d+1}$$

In this formula $A_q(n,d)$ means the maximum number of possible codewords in a q-ary linear code which has a length of n. d here represents the minimum Hamming distance as we explained in *Section 4.3*. As it showing $A_q(n,d)$ means the maximum number of possible codewords so this expression can be roughly seen as $A_q(n,d) = q^k$. So we can get another expression of Singleton bound:

$$q^k \leq q^{n-d+1}$$

Proof: We have mentioned in *Section 4.2.2* that a linear code $[n, k, d]_q$ there are q^k many q-ary words of length n.
Let C be a code of length n and having a minimum distance d over an alphabet of size q. It can be seen clearly that all codewords $c \in C$ are different. Then $|C| \leq q^{n-d+1}$. To prove this point we firstly suppose that $|C| > q^{n-d+1}$. Then we want to refute the assumption by reducing it to absurdity. So we prove Theorem 1 indirectly. According to the pigeonhole principle, if n items are put into m pigeonholes which $n > k$, the only method is that put several items into one pigeonhole. In other words, there must be at least one hole that holds more than one item. And for this problem if the codewords are treated as the 'items' there must be two codewords c', $c'' \in C$ that agree on the first $n - d + 1$location.
Because we suppose $|C| > q^{n-d+1}$, it is like the situation $n > k$. Then we can get:

$$\Delta(c', c'') \leq d - 1 < d$$

It is contradictory to the truth that d is the minimum distance of code C. So the

assumption $|C| > q^{n-d+1}$ in the beginning is absurd.
The codes are called Maximum distance separable (MDS) codes when they achieve the equality in Singleton bound.
Every linear code $[n, k, d]$ satisfies $k + d \leq n + 1$, and when $k + d = n + 1$ is called MDS.
Now we get the Singleton bound as follows:

$$q^k \leq q^{n-d+1}$$

We can simplify the expression as:

$$k \leq n - d + 1$$

If we change the position of k and n we can get:

$$n \geq k + d - 1$$

For example, code C given a rank $k = 3$, $d = 3$ we can get the minimum length n of the code is at least 5 from the Singleton bound.
The Singleton bound is a relatively crude bound. However, the bound indeed exists. For this project it has advanced meanings to get the lower bound with given the minimum Hamming distance and the minimum dual distance.
If we change the position of d and k we can get:

$$d \leq n - k + 1$$

Therefore, the maximum distance is $n - k + 1$. If a linear code with the length n is 5 and k is 3 then the Hamming distance is 3. Form this expression, we can get the maximum distance. This means we can get the maximum ability of error detecting and error correcting of the linear code. This may be a little far away from our project. But it is a good open question to discuss.

6.2 Gilbert-Varshamov bound

The Gilbert-Varshamov bound[2] which is named after Edgar Gilbert and Rom Varshamov is another bound in coding theory. Comparing with the Singleton bound it is a little more complicated. Just as the same as the Singleton bound Let $A_q(n,d)$ denote the maximum possible size of a q-ary code C with length n and the minimum Hamming weight d (a q-ary code is a code over the field F_q of q elements).
Then:

$$A_q(n,d) \geq \frac{q^n}{\sum_{j=0}^{d-1}\binom{n}{j}(q-1)^j}$$

For the prime power case, a new sublimated bound has been proposed. We say the original Gilbert-Varshamov bound can be improved to the new bound:

$$A_q(n,d) \geq q^k.$$

In the inequality k is the greatest integer in which

$$q^k < \frac{q^n}{\sum_{j=0}^{d-2}\binom{n-1}{j}(q-1)^j}$$

Proof: As the same as the Singleton bound, the maximal size of a linear code C is

$$|C| = A_q(n,d).$$

Then for all code $c \in F_q^n$(a linear code could be seen as a linear subspace of F_q^n) there must exist at least one codeword that $C_c \in C$ and the Hamming distance $d(c,C_c)$ satisfies:

$$d(c,C_c) \leq d - 1$$

This is easily correct because if not there is a contradiction on the size of $|C|$ as we could add c to the code but still keeping the linear code's minimum Hamming distance unchanged. Therefore, the whole of F_q^n is included in the union of all the balls whose radius $d - 1$ having their centre at some $c' \in C$. This can expressed as:

$$F_q^n = \cup_{c' \in C} B(c', d - 1)$$

Remark: the ball is entirely different from the ball's concept in people's daily life. From the knowledge if preview section we can see F_q^n could be seen a n-dimensional space. So here it is treated as a ball. The space of the ball is the F_q^n. As we know

$$d(c,C_c) \leq d - 1$$

So we can image the whole of F_q^n is contained in the union of all the balls which has the radius of $d - 1$. $B(c', d - 1)$ here means a ball which centered at a point c' and has a radius $d - 1$.

Now each ball has size as:

$$\sum_{j=0}^{d-1} \binom{n}{j} (q-1)^j$$

In geometry, if two balls are totally independent, then the union of these two balls is equal the summation of the two balls. If these two balls are overlapped then the union of these two balls is less than the two balls' summation. So from this we can deduce:

$$\left|\cup_{c' \in C} B(c', d - 1)\right| \leq \sum_{c' \in C} \left|B(c', d - 1)\right|$$

So:

$$\left|F_q^n\right| = \left|\cup_{c' \in C} B(c', d - 1)\right| \leq \sum_{c' \in C} \left|B(c', d - 1)\right|$$

$$\left|F_q^n\right| \leq \sum_{c' \in C} \left|B(c', d - 1)\right|$$

Each ball has a size of:

$$\sum_{j=0}^{d-1} \binom{n}{j} (q-1)^j$$

Then:

$$\sum_{c' \in C} \left|B(c', d - 1)\right| = |C| \sum_{j=0}^{d-1} \binom{n}{j} (q-1)^j$$

Remark: $|C|$ means the cardinality of set C.

So:

$$\left|F_q^n\right| = \left|\cup_{c' \in C} B(c', d - 1)\right| \leq |C| \sum_{j=0}^{d-1} \binom{n}{j} (q-1)^j,$$

$$\left|F_q^n\right| \leq |C| \sum_{j=0}^{d-1} \binom{n}{j} (q-1)^j$$

$\sum_{j=0}^{d-1} \binom{n}{j} (q-1)^j$ as the size of a ball is certainly greater than zero, so it can't be a negative number. Therefore,

$$|C| \geq \frac{\left|F_q^n\right|}{\sum_{j=0}^{d-1} \binom{n}{j} (q-1)^j} = \frac{q^n}{\sum_{j=0}^{d-1} \binom{n}{j} (q-1)^j}$$

So:

$$A_q(n,d) \geq \frac{q^n}{\sum_{j=0}^{d-1} \binom{n}{j} (q-1)^j}$$

6.3 Hamming bound

In coding theory the Hamming bound[18] is another important bound which is also known as the sphere-packing bound or volume bound. From its definition we can guess Hamming bound may be closely relationship with some balls in geometry. In this section we will introduce the Hamming bound and prove it in geometry.
The Hamming bound can be represented by the following:

$$A_q(n,d) \leq \frac{q^n}{\sum_{i=0}^{\lfloor d-1/2 \rfloor}\binom{n}{i}(q-1)^i} \text{ [19]}$$

In this inequality, integers n, d have a such relationship $1 \leq d \leq n$. And integer $q > 1$. To prove this inequality, we firstly denote $C = \{c_1,\ldots,c_k\}$ to be an optimal (n, k, d) code over A, where $|A|$= q. And $A_q(n,d)$ means the maximum possible size of a q-ary block code C which the length is n and has a minimum Hamming distance d. We denote $e = \left\lfloor \frac{d-1}{2} \right\rfloor$. This function is called floor function. It means the maximum integer which is smaller than the value of $\frac{d-1}{2}$.

To prove the Hamming bound we bring in a new concept—Sphere packing. Sphere packing is always used in geometry. Each sphere is supposed a containing space which contains the same amount of codewords.
In *Section 5.3* we have discussed that if there are l errors being wanted to correct then the minimum Hamming distance

$$d_{\min} \geq 2l + 1.$$

So we can get the inequality below just by some simple changes through the previous inequality.

$$l \leq \frac{d_{\min}-1}{2}$$

This means if the minimum distance is $d_{\min}$ then $\left\lfloor \frac{d_{\min}-1}{2} \right\rfloor$ errors are allowed. We denote $e = l$ then we get

$$e = \left\lfloor \frac{d-1}{2} \right\rfloor$$

Given a code $c_i \in C$, consider the ball which centered with c_i and have a radius of e. We have expounded that $e = \left\lfloor \frac{d-1}{2} \right\rfloor$ is the errors being allowed. Therefore, each ball is non-intersecting, otherwise, the minimum hamming distance $d_{\min} \geq 2l + 1$ is not the minimum distance. Just as the same as the Gilbert-Varshmov bound we denote $B(c_i,e)$ as a ball which is centered by c_i and has a radius of e. Because the packing spheres $B(c_i,e)$ are disjoint we can add all the codewords that each sphere contains. So we can get:

$$\bigcup_{i=1}^{M} B(c_i, e) \subseteq A^n$$

From this inequality we can get the following inequality：

$$M \cdot V_q^n(e) \leq q^n$$

which derives the following expression:

$$A_q(n, d) = M \leq \frac{q^n}{V_q^n(e)} = \frac{q^n}{V_q^n(\lfloor d-1/2 \rfloor)}$$

$V_q^n(e)$ means the volume of the ball which in other words the total codewords the ball containing. e is the errors that a code could be corrected with the minimum Hamming distance $d_{\min}$. Therefore, e are the maximum components of the n components of a codeoword that we allow to deviate from the centre of the ball to one of the $(q-1)$ possible values. So we can define $V_q^n(e)$ as following:

$$V_q^n(e) = \sum_{i=0}^{e} \binom{n}{i} (q-1)^i = \sum_{i=0}^{\lfloor d-1/2 \rfloor} \binom{n}{i} (q-1)^i$$

Therefore, the complete expression of Hamming bound is as:

$$A_q(n, d) \leq \frac{q^n}{\sum_{i=0}^{\lfloor d-1/2 \rfloor} \binom{n}{i} (q-1)^i}$$

Linear codes are special block codes so it also satisfies the Hamming bound.

Primal-dual distance bounds of linear codes on $GF(4)$

Given any two parameters of a linear code $[n,\ k,\ d]$ to find the third parameter's bounds has been studied by many experts. The Hamming bounds, Gilbert-Varshmov bound and some other famous bounds introduced in *Section 6* are all examples in this aspect. In this paper we show another novel approach to find the minimum length of a linear code with given the minimum Hamming distance and the minimum dual Hamming distance. Luckily, some reputable experts like Matsumoto have investigated in this scope and found looking them together could lead new insights. Several lower bounds and upper bounds were finally found in the case of $GF(2)$. And now we extend them to find new bounds on $GF(4)$

7.1 Upper bounds on $GF(4)$

7.1.1 The bijective function

Firstly we introduce some preliminaries which are used to serve our deduction processes. Bijective is a mathematic concept. It is a function that it satisfies both the injective and surjective at the same time. We have a function f and a set A.
For any two different elements a_1 and a_2 belong to the set A, if $f(a_1) \neq f(a_2)$ we say the function f is injective.
Suppose a function f is like that $f(A) \rightarrow B$ and any element of the set B is the image of some element of set A we say the function f is surjective.
If the function f has both of the two properties we say the function f is bijective. So we can conclude that a bijective function is a one-to-one correspondence between two sets. The bijective function is an important concept in math. For example it could be used in isomorphism. For more clearly explanation we give a example as the *Figure 3*

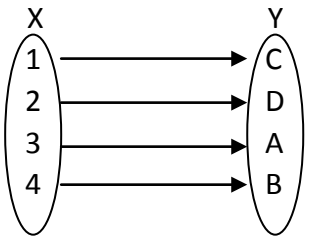

Figure 3

This function is certainly a injective and a surjective function so it a bijective function.

7.1.2 Linear map

Another concept is the linear map. In mathematics, a linear map is also a function that preserves the vector addition and scalar multiplication operations between two vector spaces.
Let two vector spaces V and W over the same field K. A function $f: V \to W$ is defined a linear map if for any two vectors $v \in V$ and $w \in V$ and any scalar $\alpha \in K$ it satisfies the two following conditions:

$f(\vec{v} + \vec{w}) = f(\vec{v}) + f(\vec{w})$
$f(\alpha\vec{v}) = \alpha f(\vec{v})$

It is easy to see that the first condition is the vector addition and the second condition is the scalar multiplication.

7.1.3 Upper bounds based on Gilbert-Varshamov bound on $GF(4)$

In this section some upper bounds on $N(d,d^{\perp})$ are provided. The first upper bound is built on the base of the famous Gilbert-Varshamov bound that means we deduce the new bounds given the minimum distance and the dual minimum distance on the base of it.
We firstly denote that C is an $[n, k]$ linear code. And we set

$S_{n,k} = \{ C \mid C \text{ is a } [n, k] \text{ linear code}\}$, $S_{n,k}(v) = \{ C \in S_{n,k} \mid C \ni v \}$ and

$S_{n,k}^{\perp}(v) = \{ C \in S_{n,k} \mid C^{\perp} \ni v \}$.

Definition 1:

$$S_{n,k} = \{ C \mid C \text{ is a } [n, k] \text{ linear code}\}$$

$$S_{n,k}(v) = \{ C \in S_{n,k} \mid C \ni v \}$$

$$S_{n,k}^{\perp}(v) = \{ C \in S_{n,k} \mid C^{\perp} \ni v \}$$

So we have the following lemma if the 'v' is a nonzero vector and 'v' belongs to $GF(4)^n$.

Lemma 1:

$$\frac{|S_{n,k}(v)|}{|S_{n,k}|} = \frac{4^k-1}{4^n-1} \qquad (1)$$

$$\frac{\left|S_{n,k}^{\perp}(v)\right|}{\left|S_{n,k}\right|} = \frac{4^{n-k}-1}{4^{n}-1} \qquad (2)$$

Proof:

Suppose u and v are both nonzero vector on $GF(4)^n$. It can be easily get the following equations((3), (4), (5)):

$$\left|S_{n,k}(u)\right| = \left|S_{n,k}(v)\right| \qquad (3)$$

$$\left|S_{n,k}^{\perp}(u)\right| = \left|S_{n,n-k}(u)\right| \qquad (4)$$

$$\left|S_{n,k}^{\perp}(u)\right| = \left|S_{n,k}^{\perp}(v)\right| \qquad (5)$$

To prove these 3 equations, we need to firstly define the groupGL_n. It means that the group is the set of bijective linear maps f on $GF(4)^n$. So the group GL_n is bijective and also linear maps. And g is also a bijective linear map over $GF(4)^n$. But the map g is fixed on $g(v) = u$.

C_1 is a fixed linear code and $S_{n,k} \ni C_1$. These preliminaries are very helpful for people to understand the three equations above. We then get the following inferences:

$$\begin{aligned}
&\left|S_{n,k}(u)\right| \\
&= \left|\{C \in S_{n,k} | C \in u\}\right| \\
&= |\{f(C_1) | f(C_1) \ni u, f \in GL_n\}| \\
&= |\{f(C_1) | f(C_1) \ni g(v), f \in GL_n\}| \\
&= |\{g^{-1} \circ f(C_1) | g^{-1} \circ f(C_1) \ni v, f \in GL_n\}| \\
&= |\{f(C_1) | f(C_1) \ni v, f \in GL_n\}| \\
&= \left|S_{n,k}(v)\right|
\end{aligned}$$

From the continuous equations we can prove the equation (3). A dual code is code orthogonal to a linear code. So we can treat a dual code as a special linear code which has $n-k$ dimensions. According to this definition, we can get:

$$\begin{aligned}
&\left|S_{n,k}^{\perp}(u)\right| \\
&= \left|\{C \in S_{n,k} | C^{\perp} \ni u\}\right| \\
&= \left|\{C \in S_{n,n-k} | C \ni u\}\right| \\
&= \left|S_{n,n-k}(u)\right|
\end{aligned}$$

Therefore, the equation (4) proved. From the equation (3) and the equation (4) we can easily prove equation (5) is also correct in the case of $GF(4)$. Obtaining the three equations is not our final target. These three equations are used to prove equation (1) and (2). We denote A as the set of a pair of vector u and $C \in S_{n,k}$, such that $u \in C$. Then we have $|A| = (4^k - 1)|S_{n,k}|$, because there are $(4^k - 1)$ vectors u that $u \in C$ for every $C \in S_{n,k}$ in $GF(4)$. That is different from $GF(2)$, because there are only $(2^k - 1)$ vectors u that $u \in C$ for every $C \in S_{n,k}$ in $GF(2)$.

Obviously for every vector u there are $|S_{n,k}(u)|$ linear codes C that $u \in C$.

Then we have

$$|A| = \sum_{0 \neq u \in GF(4)^n} |S_{n,k}(u)| = (4^n - 1)\,|S_{n,k}(v)|$$

So until now we have known:

$$|A| = (4^k - 1)|S_{n,k}|$$

$$|A| = (4^n - 1)|S_{n,k}(v)|.$$

This leads to $(4^k - 1)|S_{n,k}| = (4^n - 1)|S_{n,k}(v)|$. We change the position of $|S_{n,k}|$ and $(4^n - 1)$ we get the equation (1). So the equation (1) proved. If we connect equation (1) with equation (4) we can get equation (2).

If a code $[n, k, d]$ has the dual distance $d^{\perp}$, and it satisfies the condition below, we say it has the upper bound in the case of $GF(4)$.

Theorem 1: there one linear code $[n, k, d]_4$ which is upper bounded by

$$\frac{4^k - 1}{4^n - 1} \sum_{i=1}^{d-1} \binom{n}{i} + \frac{4^{n-k} - 1}{4^n - 1} \sum_{i=1}^{d^{\perp}-1} \binom{n}{i} < 1$$

Proof:

Through the proving process we have got:

$$\frac{|S_{n,k}(v)|}{|S_{n,k}|} = \frac{4^k - 1}{4^n - 1} \quad (1)$$

$$\frac{|S_{n,k}^{\perp}(v)|}{|S_{n,k}|} = \frac{4^{n-k} - 1}{4^n - 1} \quad (2)$$

$|S_{n,k}|$ is always greater than zero. So from equation (1) and equation (2) we get:

$$|S_{n,k}(v)| = \frac{4^k - 1}{4^n - 1} |S_{n,k}|$$

$$\left|S_{n,k}^{\perp}(v)\right| = \frac{4^{n-k}-1}{4^n-1}\left|S_{n,k}\right|$$

According to Gilbert-Varshamov bound[23, pp 557-558], the number of vectors v of Hamming weight less than d is

$$\sum_{i=1}^{d-1}\binom{n}{i}.$$

As we supposed the code exists iff d is the minimum Hamming distance and $d^{\perp}$ is the minimum dual Hamming distance, so

$$S_{n,k} = \bigcup_{d\le \omega t(v)} S_{n,k}(v) \cup \bigcup_{d^{\perp}\le \omega t(v)} S_{n,k}^{\perp}(v)$$

Therefore, we can get:

$$\begin{aligned} &\left(\sum_{1\le \omega t(v)\le d-1}\left|S_{n,k}(v)\right| + \sum_{1\le \omega t(v)\le d^{\perp}-1}\left|S_{n,k}^{\perp}(v)\right|\right) \\ \le &\left(\frac{4^k-1}{4^n-1}\left|S_{n,k}\right|\sum_{i=1}^{d-1}\binom{n}{i} + \frac{4^{n-k}-1}{4^n-1}\left|S_{n,k}\right|\sum_{i=1}^{d^{\perp}-1}\binom{n}{i}\right) \\ &\Leftrightarrow \\ &\left(\sum_{1\le \omega t(v)\le d-1}\left|S_{n,k}(v)\right| + \sum_{1\le \omega t(v)\le d^{\perp}-1}\left|S_{n,k}^{\perp}(v)\right|\right) \\ \le &\left(\frac{4^k-1}{4^n-1}\sum_{i=1}^{d-1}\binom{n}{i} + \frac{4^{n-k}-1}{4^n-1}\sum_{i=1}^{d^{\perp}-1}\binom{n}{i}\right)\left|S_{n,k}\right| \end{aligned}$$

So the bound is found

$$\frac{4^k-1}{4^n-1}\sum_{i=1}^{d-1}\binom{n}{i} + \frac{4^{n-k}-1}{4^n-1}\sum_{i=1}^{d^{\perp}-1}\binom{n}{i} < 1$$

7.1.4 Other two upper bounds on punctured code

We denote C^* is the punctured code [20] of linear code C. So we can derive the minimum Hamming distance of C^* is at least $d-1$ and the dual distance is at least $d^{\perp}$. A punctured code is a linear code that could get by deleting the same coordinate p in each codeword from the code C. In most situations a minimum Hamming weight of code C has a nonzero pth coordinate, so the minimum Hamming distance of C^* is $d-1$. And the dual distance is still $d^{\perp}$. Therefore, we can get another upper bound:

$$N(d-1, d^{\perp}) \le N(d, d^{\perp}) - 1 \text{ (for } d \ge 2)$$

We consider the punctured code of the dual code of code C in the same logic we can get another interesting bound:

$$N(d, d^{\perp} - 1) \leq N(d, d^{\perp}) - 1 \text{ (for } d^{\perp} \geq 2)$$

7.2 Lower bounds

In this section, three lower bounds are showed which have been investigated on $N(d,d^{\perp})$ over $GF(4)$. The first lower bound is on the base of the Hamming bound [18]. And the second is based on Griesmer bound. [21] The third bound is related to the Singleton bound. [22]

7.2.1 Bound on the base of the Hamming bound on *GF*(*4*)

First, we define the function $\ell(n,d)$ as following. n and d stand for any positive integers.

$$\ell(n,d) = \begin{cases} \sum_{i=0}^{(d-1)/2} \binom{n}{i} & \text{for odd } d \\ \sum_{i=0}^{d/2-1} \binom{n}{i} + \binom{n-1}{d/2-1} & \text{for even } d \end{cases}$$

'And we have several discrete variables $X_1, X_2, \dots, X_n$ which are selected randomly. We say they are d-wise independent if

$$\Pr\left[X_{i_1} = x_{i_1}, \dots, X_{i_d} = x_{i_d}\right] = \prod_{j=1}^{d} \Pr\left[X_{i_j} = x_{i_j}\right]$$

for all d-tuples of indices ($i_1, i_2, \dots, i_d$) and all realizations ($X_{i_1}, X_{i_2}, \dots, X_{i_d}$) of random variables.'[4] And now we have already had the definition of X-wise independent. Therefore, there are some non-constant variables $X_1, X_2 \dots, X_n$ being selected randomly and they are $(d-1)$-wise independent. If they map the space Ω to {0 , 1}. Then we get
$|\Omega| \geq \ell(n,d)$.

Theorem 3: We have $4^{n-k} \geq \ell(n,d)$ for a $[n, k, d]$ linear code.

This theorem could be derived by the lemma which we have just mentioned above. because $|\Omega| \geq \ell(n,d)$, so $4^{n-k} = |\Omega| \geq \ell(n,d)$. Obviously, this *theorem 3* is an improvement of the Hamming bound when d is even.
Now let's to see another corollary:

$$N(d,d^{\perp}) \geq \min\{n | n \geq log_4\ell(n,d) + log_4\ell(n,d^{\perp})\}$$

Proof: From *theorem A* we can get the following theory:

$$4^{n-k} \cdot 4^{k} \mid \geq \ell(n,d) \cdot \ell(n,d^{\perp})$$

$$\Leftrightarrow$$

$$n \geq log_4\ell(n,d) + log_4\ell(n,d^{\perp})$$

For a $[n, k, d]$ linear code with a given dual Hamming distance $d^{\perp}$, we have the above inequality. As we have denoted $N(d,d^{\perp})$ in the beginning of the report. There exist a linear code of the minimum length n and given a minimum Hamming distance d and the dual distance $d^{\perp}$. $N(d,d^{\perp})$ has a lower bound as it is explained by the above inequality.

7.2.2 The lower bound related to Griesmer bound on $GF(4)$

According to the famous Griesmer bound we have discussed in early:

$$n \geq d + \sum_{i=1}^{k-1} \left\lceil \frac{d}{4^i} \right\rceil$$

So if a linear code $[n,\ k,\ d]$ has a dual distance $d^{\perp}$, then we can also get

$$n \geq d^{\perp} + \sum_{i=1}^{n-k-1} \left\lceil \frac{d^{\perp}}{4^i} \right\rceil$$

This can be got by a simple inference from the Griesmer bound. We just replace the position of C with $C^{\perp}$. And if we add with two inequalities we can get the following inequality:

$$2n \geq d + d^{\perp} + \sum_{i=1}^{k-1} \left\lceil \frac{d}{4^i} \right\rceil + \sum_{i=1}^{n-k-1} \left\lceil \frac{d^{\perp}}{4^i} \right\rceil$$

So we have a new theorem:

$$N(d,d^{\perp}) \geq \min\left[n: 2n \geq d + d^{\perp} + min_{k=1,\ldots,n-1} \left\{\sum_{i=1}^{k-1} \left\lceil \frac{d}{4^i} \right\rceil + \sum_{i=1}^{n-k-1} \left\lceil \frac{d^{\perp}}{4^i} \right\rceil\right\}\right]$$

We say $2N(d,d^{\perp})$ is lower bounded under the condition of possible n and k. Because $N(d,d^{\perp})$ is the minimum n such that a linear code of length n exists with a minimum distance d and the dual distance $d^{\perp}$.

7.2.3 The lower bound on Singleton bound on $GF(4)$

This section shows the lower bound with given the minimum Hamming distance and the minimum dual Hamming distance which is deduced on the base of famous Singleton bound which we have explained specifically in *Section 6.1*.
From *Section 6.1* we have the Singleton bound could be expressed as following:

$$k \leq n - d + 1 \qquad (7)$$

Here, n is the length of a linear code. k is the codeword and d is the Hamming distance. We denote $C^{\perp}$ is the dual code of code C $[n, k, d]$. So $C^{\perp}$ could be stated like $[n, n-k, d^{\perp}]$. Therefore we could see the dual code as a special linear code. We apply the Singleton bound's expression to the dual code, we could get:

$$n - k \leq n - d^{\perp} + 1 \quad (8)$$

We add the inequality (7) with the inequality (8). A new formula is produced.

$$\begin{gathered} n - k + k \leq n - d^{\perp} + 1 + n - d + 1 \\ \Leftrightarrow \\ n \leq 2n - d - d^{\perp} + 2 \\ \Leftrightarrow \\ n \geq d + d^{\perp} - 2 \end{gathered}$$

Thus we get a new lower bound which is based on the Singleton bound.

Remark: All the theorems and formulas of *section 7* are deduced on the bounds of several existed bounds and also deduced on the base of 'Primal-dual distance bounds of linear codes with application to cryptography' Ryutaroh Matsumoto, *Member, IEEE,* Kaoru Kurosawa, *Member, IEEE,* Toshiya Itoh, *Nonmember,* Toshimitsu Konno, *Nonmember,* Tomohiko Uyematsu, *Member, IEEE*, which is focused on Primal-dual approach on $GF(2)$.

7.3 Summary

This section gives a briefly conclusion about the Upper and Lower bounds with Primal-dual approach on $GF(4)$. And it gives an interesting contrast of each corresponding bound in $GF(2)$ and $GF(4)$. We mainly focus on each bound's differences in the different finite fields and discuss what cause the differences.

7.3.1 Upper bounds

Ryutaroh Matsumoto and some other professors in Japan find the Upper bound on $GF(2)$ which is also derived from the Gilbert-Varshamov bound.[4] Therefore, this bound has great values in comparing. Let's to see what they have got.

$$\frac{2^k-1}{2^n-1}\sum_{i=1}^{d-1}\binom{n}{i}+\frac{2^{n-k}-1}{2^n-1}\sum_{i=1}^{d^\perp-1}\binom{n}{i}<1 \quad (10)$$

It looks as the same as what we have got. But actually they have different meanings. In the inequality (10) the following formula could be abstracted.

$$\frac{2^k-1}{2^n-1}(X)+\frac{2^{n-k}-1}{2^n-1}(X)<1$$

However, in $GF(4)$ the bound could be represented as

$$\frac{4^k-1}{4^n-1}(X)+\frac{4^{n-k}-1}{4^n-1}(X)<1$$

After simplifying, it could be seen clearly that the same bound in different finite fields is different.

The root cause for the differences is obscure. In *Section 4.2.2*, we know that if a linear code C $[n, k, d]$ is a code on $GF(q)$ then the possible number of the codewords of this linear code is q^k. If code C is on $GF(2)$, then it has a maximum number of codewords of 2^k. If C is on $GF(4)$, then the maximum possible number is 4^k. This directly leads to the differences in the equation (1) and equation (2) in the deduction process. Moreover, this leads to the upper bounds from the Gilbert-Varshamov bound in different finite fields being different.

One of the aims for this project's is to try to find the Upper and Lower bounds on $GF(4)$. So in this project, the minimum length is upper bounded by

$$\frac{4^k-1}{4^n-1}\sum_{i=1}^{d-1}\binom{n}{i}+\frac{4^{n-k}-1}{4^n-1}\sum_{i=1}^{d^\perp-1}\binom{n}{i}<1$$

As a result, the bounds we obtained are different from the bounds Ryutaroh Matsumoto got. The specific results are shown in Table 10.

The other two bounds are inferences of this upper bound. So the result is also different.

7.3.2 Lower bounds

In previous section, a comparison for upper bounds is given. In this section, we give a summary about the lower bounds.
The lower bound which is deduced from the singleton bound is the simplest one. It is totally the same as it in $GF(2)$. From the deduction process we can see that no matter what 'q' is the final bound is the same. So there's no difference in $GF(2)$ or in $GF(4)$.

For the lower bound which is related to the Hamming bound, we get as:

$$N(d,d^{\perp}) \geq \min\{n | n \geq log_4 \ell(n,d) + log_4 \ell(n,d^{\perp})\}$$

In $GF(2)$ the same bound is represented as:

$$N(d,d^{\perp}) \geq \min\{n | n \geq log_2 \ell(n,d) + log_2 \ell(n,d^{\perp})\}$$

Form this two formulas it shows the differences in $GF(2)$ and $GF(4)$ clearly.
The famous Griesmer bound could be expressed as:

$$n \geq d + \sum_{i=1}^{k-1} \left\lceil \frac{d}{q^i} \right\rceil$$

This inequality has been proved by many experts. We don't repeat it again here.
When a linear code is on $GF(4)$ then $q = 4$. So the inequality also could be represented as:

$$n \geq d + \sum_{i=1}^{k-1} \left\lceil \frac{d}{4^i} \right\rceil$$

For a dual code:

$$n \geq d^{\perp} + \sum_{i=1}^{n-k-1} \left\lceil \frac{d^{\perp}}{4^i} \right\rceil$$

This equals:

$$2n \geq d + d^{\perp} + \sum_{i=1}^{k-1} \left\lceil \frac{d}{4^i} \right\rceil + \sum_{i=1}^{n-k-1} \left\lceil \frac{d^{\perp}}{4^i} \right\rceil$$

The same logic is applied for $GF(2)$, when the code is on $GF(2)$ then $q = 2$ so

$$2n \geq d + d^{\perp} + \sum_{i=1}^{k-1} \left\lceil \frac{d}{2^i} \right\rceil + \sum_{i=1}^{n-k-1} \left\lceil \frac{d^{\perp}}{2^i} \right\rceil$$

Now, we can see that the same bound for different finite fields is not the same. And the reason is that $GF(2)$ and $GF(4)$ has different orders. The order of $GF(2)$ is 2 while the order of $GF(4)$ is 4.
The differences in the formula also lead to the results different. We discuss this later.

Realization of the upper and lower bounds with small $(d, d^{\perp})$

This section is mainly to introduce something about how to implement these bounds which have been found with some small $(d, d^{\perp})$.
As a project for MSc students in Computer Science, it is not enough to just find some mathematical models in theoretical. I implement some small programs to realize the bounds.

8.1 Exhaustive Search

The exhaustive search[24] method is introduced in this section. It is the main method I have taken to find the upper and lower bounds.
Exhaustive search is also known as the brute-force search in computer science. It is a useful technical to find a solution for a problem. The core of this method is systematically enumerating all possible candidate solutions if a problem has and then checking every candidate to see whether it satisfies the condition of the problem. If it satisfies one solution is found. We pick it up.
For a simple example, find the largest number of n which $n \geq 10$. The solution for the problem seems impossible if there is no other restricts. So another condition of the range of n (n is any positive integer $10 \leq n \leq 100$) is added. Exhaustive search in this situation is compare each integer from 10 to 100 to the problem $n \geq 10$ and find the optimal solution 100.
The instance above is so simple that it happens nearly impossible in real problems' solving. It is more complicated than the example above in this project. For different situations I add different conditions for a problem. For example, for an upper bound, the condition of the maximum number is added to make the computer pick the largest number up after the exhaustive search. And for lower bounds the smallest number is chosen.
Exhaustive search as a fully-fledged algorithm is very simple, but on the other hand it is very reliable. For a group of candidates, the correct solution for the problem is always found if it is in the group. This exhaustive search's property is very appropriate for my project. For the upper bounds, the formula has been deduced successfully. However, it is still not clear enough to classify the upper bound. Each pair of (n, k) is tried until the solution being found. More details about how to find the upper bound are explained in the *Section 8.3*
Exhaustive search as a basic algorithm is low-level intelligential so exhaustive search has a major flaw. It costs so much. It flows through all the candidates each time. When the queue of the candidates is so long, the time of searching may be so long that we can't accept. One possible way to speed up is using binary searching[25]. It searches from the middle of the queue each time. So it reduces the searching space heavily.

8.2 Programming language

This section is focused on the programming language used in the project.
C programming language is applied in the project to implement the upper and lower bounds. *C* programming language could be defined as middle-level language. It lies in the middle between the Assembly Language and High-level language such as JAVA. Therefore, it inherits the practicability and also has the same basic structure as High-Level Language. Differing from some Object-oriented language like JAVA, *C* programming language has the ability to handle with some basic byte and address. For my project, I just implement some basic formulas. So some high-level languages like JAVA are not appropriate. In the beginning I planned to use Python to realize my project. Finally it is given up. Because *C* programming language could give me enough support to finish my project. And Python is totally new to me.

8.3 Realization of Upper bounds

This section is about the Realization of the upper bounds. First part introduces the realization about the upper bounds which is related on the Gilbert-Varshamov bound. And then the other two bounds which are deduced from the first upper bounds are shown.

8.3.1 Upper bounds related on the Gilbert-Varshamov bound

From *Section 7.1.3* we have got the upper bound as following:

$$\frac{4^k-1}{4^n-1}\sum_{i=1}^{d-1}\binom{n}{i}+\frac{4^{n-k}-1}{4^n-1}\sum_{i=1}^{d^\perp-1}\binom{n}{i}<1$$

It is clearly known that k is always less than n from the definition of a linear code. Therefore, some assumptions are needed. Firstly, some small pairs of the minimum distance d and the dual minimum distance $d^\perp$ are given as the input. Table 1 shows the pairs d and $d^\perp$. We choose some small integers for d and $d^\perp$ so that it is easy to calculator for computers. The data in Table 1 are not only used for the upper bounds, but also as the input for other lower bounds.

d	3	4	4	5	5	5	6	6	6	6	7	7	8	8
$d^\perp$	3	3	4	3	4	5	3	4	5	6	3	4	3	4

Table 1

Secondly, we suppose the minimum length n is in the range of 0-500. And as we have discussed $k \leq n$ form a linear code's definition, so the range of k is also 0-500. According to our estimations 500 is large enough for the range of the minimum length n with given some small $(d, d^\perp)$.
After that we tried every pair of n and k to see whether the inequality above

satisfied. Then we pick up the largest one of all the 'n' which satisfies the inequality above.

8.3.1.1 Some codes for the Upper bound related on the Gilbert-Varshamov bound

```
int main()
{
    int d1,d2;
    double n,m;
    double N,Nmax=0;
    printf("Please input d and d'.\n");
    scanf("%d %d",&d1,&d2);

    for(n=1.0;n<=500.0;n++)
    {
        for(m=1.0;m<=n-1.0;m++)
        {

    N=(pow(4.0,m)-1.0)/(pow(4.0,n)-1.0)*Sum(d1-1,n)+(pow(4.0,n-m)-1.0)/(pow(4.0
,n)-1.0)*Sum(d2-1,n);
            if(N<=1 && n>Nmax)
                Nmax=n;
        }
    }
    printf("%f.\n",Nmax);
    return 1;
```

In the table above shows some important codes for this bound.
The Main Function: d_1 and d_2 are two integers as the input as $(d, d^{\perp})$. And the code of 'N = (pow(4.0,m)-1.0)/(pow(4.0,n)-1.0) * Sum(d1-1,n) + (pow(4.0,n-m)-1.0) / (pow(4.0,n)-1.0) * Sum(d2-1,n)' realizes the expression below.

$$\frac{4^k-1}{4^n-1}\sum_{i=1}^{d-1}\binom{n}{i}+\frac{4^{n-k}-1}{4^n-1}\sum_{i=1}^{d^{\perp}-1}\binom{n}{i}$$

The Sum(*) Function: represents the continuously summation.

The Factorial Function and the Division Function: The binomial $\binom{n}{i}$ could be represented by $\frac{n!}{i!(n-i)!}$. So it is realized by the union of the two functions.

Therefore, this upper bound is realized by a small program with four functions.
The complete codes is provided in Table 3 in the Appendix

8.3.1.2 Results for the Upper bound

After the realization some results are captured. The results are showed in Table 2.

d	3	4	4	5	5	5	6	6	6	6	7	7	8	8
$d^{\perp}$	3	3	4	3	4	5	3	4	5	6	3	4	3	4
Upper bound	36	40	45	44	50	55	49	54	58	60	51	57	53	59

Table 2

8.3.2 Upper bounds related on Punctured code

Through the deduction of the punctured code, other two upper bounds are derived from the upper bound above. They are expressed as follows:

$$N(d-1, d^{\perp}) \leq N(d, d^{\perp}) - 1 \text{ (for } d \geq 2)$$
$$N(d, d^{\perp}-1) \leq N(d, d^{\perp}) - 1 \text{ (for } d^{\perp} \geq 2)$$

They are closely related to the upper bound in *Section 8.3.1.* In other words these two bounds are realized on the base of the first one existed.

When d = *4* and $d^{\perp}$ = *3* then $N(3,3) \leq N(4,3) - 1 = 39$

When d = *4* and $d^{\perp}$ = *4* then $N(3,4) \leq N(4,4) - 1 = 44$

The logic operation is the same for other pairs of d and $d^{\perp}$. The input is the same data of pairs of $(d, d^{\perp})$ in Table 1

d	3	4	4	5	5	5	6	6	6	6	7	7	8	8
$d^{\perp}$	3	3	4	3	4	5	3	4	5	6	3	4	3	4

Table 1

After implementation some results are obtained. They are shown in Table 4.

d	3	4	4	5	5	5	6	6	6	6	7	7	8	8
$d^{\perp}$	3	3	4	3	4	5	3	4	5	6	3	4	3	4
Upper bound(2)	39	43	49	48	53	57	50	56	59	60	52	58	55	59
Upper bound(3)	42	44	50	49	54	58	53	57	59	62	56	59	58	60

Table 4

8.4 Realization of Lower bounds

This section is about the realization of the lower bounds which have been obtained. First part introduces the realization about the lower bounds which is related on the improvement of Hamming bound. And then the bound based on the Griesmer bound is realized. Finally the lower bound which is deduced from the famous Singleton bound is shown.

8.4.1 Lower bound related on improvement of Hamming bound

In *Section 7.2.1* we have a lower bound on $GF(4)$:

$$N(d,d^{\perp}) \geq \min\{n | n \geq log_4 \ell(n,d) + log_4 \ell(n,d^{\perp})\}$$

$\ell(n,d)$ is a function defined as:

$$\ell(n,d) = \begin{cases} \sum_{i=0}^{(d-1)/2} \binom{n}{i} & \text{for odd } d \\ \sum_{i=0}^{d/2-1} \binom{n}{i} + \binom{n-1}{d/2-1} & \text{for even } d \end{cases}$$

For realization this bound, we also firstly set up some small pairs of $(d, d^{\perp})$ like in Table 1.

d	3	4	4	5	5	5	6	6	6	6	7	7	8	8
$d^{\perp}$	3	3	4	3	4	5	3	4	5	6	3	4	3	4

Table 1

After input each pair of $(d, d^{\perp})$ we tried each value of n. The smallest n is selected from all the $'n'$ which satisfy the inequality above. The range of the values of n is also the same 0-500.

8.4.1.1 Some important codes for this Lower bound

```
    int main()
{
    int d1,d2;
    int n;
    double N;
    double Nmin;
    printf("Please input d1 and d2.\n");
    scanf("%d %d",&d1,&d2);
     N = LOG(4.0,L(1,d1))+LOG(4.0,L(1,d2));
        Nmin=N;
    printf("%f.\n",Nmin);
    for(n=2;n<500;n++)
    {
        N = LOG(4.0,L(n,d1))+LOG(4.0,L(n,d2));
        if(N<Nmin && N>0)
            Nmin = N;
    }
    Nmin=ceil(Nmin);
    printf("%f\n",Nmin);
}
```

We give the main function of the code for implementing the lower bound. In the main function we can see that the difficulty of this problem is in the implementation of the function of $\ell(n,d)$. This part isn't showed here because the complete code is too long but it is provided in Table 6 in the Appendix.
In *Section 8.3.1.1* we have showed how to solve the continuously summation problem and the binomial problem. So the same methods are applied in this situation.

8.4.1.2 Results for the Lower bound

The results obtained are shown in Table 5:

d	3	4	4	5	5	5	6	6	6	6	7	7	8	8
$d^{\perp}$	3	3	4	3	4	5	3	4	5	6	3	4	3	4
Lower bound on Hamming bound	1	1	1	1	1	1	1	1	1	1	1	1	1	1

Table 5

8.4.1.3 Test for the results for the Lower bound

In the beginning of this phase my first feeling is that something is wrong. All the bounds of different pairs of (d,$d^{\perp}$) are the same as '1'. This strange phenomenon really confuses me. I date back to the deduction process to see whether the formula of this bound is deduced correctly. On the other hand I check the code of the program carefully. I find each step runs in order. I finally decide to test it manually to find out the problem.

I first test it with the input $(d, d^{\perp}) = (3, 3)$.

So

$$n \geq log_4 \ell(n, d) + log_4 \ell(n, d^{\perp})$$

$$\Leftrightarrow$$

$$n \geq log_4 \ell(n, 3) + log_4 \ell(n, 3)$$

3 is an odd integer, so

$$\ell(n, 3) = \sum_{i=0}^{(d-1)/2} \binom{n}{i} = \sum_{i=0}^{(3-1)/2} \binom{n}{i}$$

$$= \sum_{i=0}^{1} \binom{n}{i} = \binom{n}{0} + \binom{n}{1}$$

$$= 1 + \frac{n!}{1!(n-1)!}$$

$$= 1 + n$$

Put it into the inequality:

$$n \geq log_4(1 + n) + log_4(1 + n)$$

n has a range that $0 < n \leq 500$. Therefore, the minimum length of n is obviously '1'.

I also try another pair of $(d, d^{\perp}) = (6, 4)$, the result is still '1'. I find that if the minimum n is very large than the function $\ell(n, d)$ will be really large. This is because $log_4 \ell(n, d)$ is a logarithm function. This means $\ell(n, d)$ increases rapidly. At this time I realize that the results obtained may be correct. To get the answer I ask for help for my supervisor. He confirms my suspect. As a lower bound it is entirely probable to have a very small bound.

8.4.2 Lower bounds based on the Griesmer bound

A lower bound of a linear code $[n,\ k,\ d]$ which is given the minimum distance and the dual minimum distance has been deduced from the Griesmer bound in *Section 7.2.2*. The bound could be represented by a formula:

$$N(d,d^{\perp})\ \geq\ \min\left[n: 2n\ \geq d+\ d^{\perp}+\ min_{k=1,\ldots,n-1}\ \left\{\sum_{i=1}^{k-1}\left\lceil\frac{d}{4^i}\right\rceil+\ \sum_{i=1}^{n-k-1}\left\lceil\frac{d^{\perp}}{4^i}\right\rceil\right\}\right]$$

For simply and conveniently understanding, the formula is simplified:

$$2n\ \geq\ d+\ d^{\perp}+\ \sum_{i=1}^{k-1}\left\lceil\frac{d}{4^i}\right\rceil+\ \sum_{i=1}^{n-k-1}\left\lceil\frac{d^{\perp}}{4^i}\right\rceil$$

When realizing the formula, firstly some small pairs of $(d,d^{\perp})$ are given as the input as usual. Table 1 shows the data of $(d,d^{\perp})$. Secondly we restrict the range of n. According to the $(d,d^{\perp})$ we set the range of n is 0-500. k is from 1 to $n-1$. Then we tried each pair of (n,k) for every $(d,d^{\perp})$. The smallest integer is picked up as the minimum length for the corresponding $(d,d^{\perp})$ from all the 'n' which satisfies the inequality.

8.4.2.1 Some codes for the lower bound

```
    int sum(int upper_limit, int d);

void main()
{
    int d1,d2,m,n;
    int N,Nmin;
    printf("Please input d1 and d2.\n");
    scanf("%d %d",&d1,&d2);

     Nmin=d1+d2+sum(1,d1)+sum(3-1-1,d2);
    //printf("%d\n",Nmin);
    for(n=3;n<=500;n++)
    {
     for(m=1;m<=n-1;m++)
    {
        N=d1+d2+sum(m-1,d1)+sum(n-m-1,d2);
        if(N<Nmin)
            Nmin=N;
    }
    }
    printf("%d\n",Nmin);
```

The main function: given d_1 and d_2 as the input $(d, d^{\perp})$. As a main function of the whole program, it has two 'for' loops to realize the algorithm.

$$2n \geq d + d^{\perp} + \sum_{i=1}^{k-1} \left\lceil \frac{d}{4^i} \right\rceil + \sum_{i=1}^{n-k-1} \left\lceil \frac{d^{\perp}}{4^i} \right\rceil$$

The Sum function: As the same as in the other bounds, it means the summation operation. But this program has one point that others don't have. The *ceil function* appears firstly. In the math library of *C#*, it has clearly definition. So I implement it directly.
The complete code is provided in Table 8 in Appendix.

8.4.2.2 Results for the Lower bound

The results we obtained are presented in Table 7:

d	3	4	4	5	5	5	6	6	6	6	7	7	8	8
$d^{\perp}$	3	3	4	3	4	5	3	4	5	6	3	4	3	4
Lower bound on Griesmer bound	7	8	9	9	10	12	10	11	13	14	11	12	12	13

Table 7

8.4.3 Lower bounds based on the Singleton bound

Comparing with other two lower bounds the lower bound which is deduced from the Singleton bound is relatively simple. It could be expressed as:

$$n \geq d + d^{\perp} - 2$$

Given some small pairs of $(d, d^{\perp})$ the bound is easily obtained by the summation of d and $d^{\perp}$ minus 2.
The data of $(d, d^{\perp})$ is shown in Table 1. The results are presented in Table 9

d	3	4	4	5	5	5	6	6	6	6	7	7	8	8
$d^{\perp}$	3	3	4	3	4	5	3	4	5	6	3	4	3	4
Lower bound on Singleton	4	5	6	6	7	8	7	8	9	10	8	9	9	10

Table 9

8.5 The complete results

After a few minutes to unscramble the results of all the bounds, a table of complete results is given. In table 12 all the results are showed together. It is good for readers to compare each bound with given some corresponding $(d, d^{\perp})$

d	$d^{\perp}$	Lower bounds			Upper bounds		
		Griesmer bound	Singleton bound	Hamming bound	Gilbert – Varshmov bound	Improvement 1	Improvement 2
3	3	7	4	1	36	39	42
4	3	8	5	1	40	43	44
4	4	9	6	1	45	49	50
5	3	9	6	1	44	48	49
5	4	10	7	1	50	53	54
5	5	12	8	1	55	57	58
6	3	10	7	1	49	50	53
6	4	11	8	1	54	56	57
6	5	13	9	1	58	59	59
6	6	14	10	1	60	61	62
7	3	11	8	1	51	52	56
7	4	12	9	1	57	58	59
8	3	12	9	1	53	55	58
8	4	13	10	1	59	59	60

Table 10

Summary

In this section, we give some summaries about the results. A comparison of the derived bounds in Table 10 is showed.

9.1 A comparison of the results

9.1.1 A Horizontal comparison of the results

Table 10 shows the complete results we obtained from the algorithms in *Section 7*. An entry x in Table 10 means that $N(d, d^{\perp}) \geq x$ for the lower bounds and $N(d, d^{\perp}) \leq x$ for the upper bounds.
From Table 10 we can get the following conclusion. The lower bounds are decreasing from the Griesmer bound to the improvement of the Hamming bound. The lower bounds which are derived from the Griesmer bound and the Singleton bound are relatively close. And the bound related to improvement of Hamming bound is rather smaller than the other two bounds. More interesting, this bound constantly keeps the same of the value of '1' even with given different pairs of $(d, d^{\perp})$. The other two bounds maybe also have the same values with some different pairs of $(d, d^{\perp})$. But this situation appears in only few pairs of $(d, d^{\perp})$. They vary with the different pairs of $(d, d^{\perp})$ in the mass. This situation appearing may be because of the inner property of the lower bound. We can see that this lower bound which related to Hamming bound could be represented by some logarithm functions. And a logarithm function will have very small answers when the antilogarithm is small. However, the first two bounds don't have this feature so they vary frequently.

And for Upper bounds it seems that it increases from the first upper bound to the third bound from Table 10. Although there is no enough powerful proof to prove this increasing trend, it is still worth studying the phenomenon for us. The first upper bound which is based on the Gilbert-Varshamov bound could be seen as the predecessor of the other two upper bounds. For example, it is obviously that $N(5,3) = N(6,3) - 1 = 48$ according to the upper bound of the improvement 1. And we get $N(5,3) = 44$ in the first upper bound through the computing. So it can be seen clearly the size relationship between the two bounds. For the second upper bound and the third bound, it also has an increasing trend. This is may be caused by the difference of the minimum distance and the minimum dual distance. However, in my opinion this phenomenon is just a coincidence. There is no inevitable connection between them.

9.1.2 A Vertical comparison of the results

After the Horizontal comparison between each bound we now give a vertical comparison. This means we treat each bound independently. For lower bounds, we can make the following observation. The value of the first lower bound is becoming bigger with the $(d, d^{\perp})$ being bigger. For example, when $(d, d^{\perp}) = (3,3)$ the lower bound related to Griesmer bound is '7'. And when $(d, d^{\perp}) = (8,4)$, it becomes '13'. The same rule is applied for the second lower bound. For the third bound we have just discussed that it keeps the same value of '1'. This rule is applied only when d and $d^{\perp}$ are both increasing. If one of them increases and the other one decreases, the lower bound maybe decreases. For example, when $(d, d^{\perp}) = (6,6)$, the lower bound based on Griesmer bound is '14'. When d increases to 7 and $d^{\perp}$ decreases to '3', the value of the bound decreases to '11'.

For the first Upper bounds it can be seen clearly that the minimum bound is $N(5,3) = 36$. And the maximum bound is $N(6,6) = 60$. The difference between them is '25'. However, the maximum distance between two different $(d, d^{\perp})$ in the first lower bound is just '6'. This shows that the upper bound is in Table 10 is very loose for some small pairs of $(d, d^{\perp})$. The same situation is also appeared in the other two upper bounds. The looseness strongly shows that the exhaustive search method used in the program is not good enough. Some other smart search methods should be applied.
Another important observation for the upper bound is that the bound is not monotonically increased. Even in the same bound, the number is large or small at sometime.

Evaluation

The purpose of this section is to evaluate whether the main aims of the project have been reached and whether all the requirements have been realized. Beyond this, a discussion about the aims and the requirements of the project is given in this part. Some strengths and weaknesses are also presented. An evaluation as a quality checking is very important for a successful project, because it gives a valuable summary for the whole project.

10.1 How to evaluate the project

It can be seen clearly that the aim is the heart of the whole project. All the measures are taken in aim-orientation. So, the first rule I used to evaluate the system is to check whether the project's aim is met.
The requirement is a main outline of the project. It gives instructions about what the specific requirements finally like. Therefore the second evaluative criterion I use is to check whether the project confirms to each requirement. Of course, everything can't be exactly the same as the initial requirement. Because my understanding of the project is developed with my relevant knowledge accumulated. So the requirements are revised upon the base of my understanding of the project.
Thirdly, the feedback of my supervisor is really worth to consider. Martijn Stam is a senior professor in Cryptography and coding theory. So his suggestions have clear directions for this project.

10.1.1 Whether the project's aim met

One of the main aims of this project is to find the minimum length of a linear code with given the minimum distance and the minimum dual distance on $GF(4)$. To find the minimum length of a linear code means finding the lower and upper bounds of this code. It can be seen clearly that this objective has been reached successfully. The results from Table 12 shows that some lower and upper bounds have been found with given some pairs of $(d, d^{\perp})$.
The second object of this project is to point out the internal connection between the minimum length of a linear code and the security of Boolean function. A Boolean function's security is very important in cryptography. It has direct relationship with the security of S-Box. S-box is one of the key components of a symmetric algorithm like AES. In the project, we have successfully turned the Boolean function's security into a mathematical problem of finding the minimum length of linear code with given $(d, d^{\perp})$. So it could be defined that this objective of the project is met from this point.

10.1.2 Whether the project's requirements met

The requirements as the specific extensions of the project's objectives are also used to evaluate a project. Differing from the aims, requirements include some more specific criterions. In this section, an explanation is presented about whether the requirements are reached or not.

1. The first requirement is that some basic famous bounds which have been existed should be found. In *Section 6* some bounds like Singleton bound, Hamming bound, etc are introduced. These bounds are always given two parameters of a linear code $[n,\ k,\ d]$ and try to restrict another one. So they don't have a direct relationship with my project. But, they are the project's foundations. All the bounds I found with given $(d, d^{\perp})$ on $GF(4)$ are deduced from them. The Singleton bound, Hamming bound and Gilbert-Varshamov bound are introduced in *Section 6* with their own proving procedures.
2. The second requirement is that some new bounds should be found on $GF(4)$ with given the minimum distance and the minimum dual distance. This requirement is the most important one. This is actually what the project required. In *Section 7* three Upper bounds and three Lower bounds are provided. The first Upper bound is got from the Gilbert-Varshamov bound. And the other two bounds are deduced from it with some knowledge of punctured code. For the three lower bounds, they are related to the Hamming bound, Griesmer bound and Singleton bound respectively. In the whole deduction procedures, the key factor is applying the mathematical skills flexibly.
3. The last requirement of the project is to realize all the bounds which have been found on $GF(4)$. In *Section 8* each upper and lower bound are implemented by some programs which are written in *C* programming language. Each bound is represented by some inequalities. Then these algorithms are implemented by *C* programs. Some small pairs of $(d, d^{\perp})$ are set as the inputs and the outputs are the different bounds according to the different algorithms. The complete results obtained are provided in Table 10

These three requirements are the extensions of the aims of the project. If the project is well finished then all the three requirements should be satisfied. Through the explanation above, it can be seen clearly that the requirements are basically met. In other words in my opinion the project has been carried out successfully.

10.2 Strength and Weakness

In this section, the strength and the weakness of the project are introduced.

10.2.1 Strength

In the whole project, the main strength is finding some basic bounds like Hamming bound and then having a deducting process to extend the bounds to new insights on $GF(4)$. In the whole project, this step is key element. All the bounds could be realized under this condition.

The second strength is that all these equalities are implemented successfully. After we get the equalities we can see that some bounds like the lower bounds based on the Singleton bound is easy to carry out. However, it is nearly impossible to calculator other bounds without computer programs. If the data is large, it is more difficult to carry out. So some programs are implemented using *C* programming language to make computers help us to carry the bounds out.

Another significant strength is that we have successfully turned the Boolean function's security into a mathematical problem of finding the minimum length of linear code with given $(d, d^{\perp})$. This is also why the project being researched.

10.2.2 Weakness

Comparing with what we achieved, we worry more about what we missed. The weaknesses of the project have more instructional meanings for the future work.

One of the major weaknesses is that the structure of the all the bounds is not reasonable. This point is reflected by the results. It can be seen that the lower bounds are not precise enough from the results. For example, with given some small different input $(d, d^{\perp})$, the results are all '1'. And for Upper bounds, the results are much looser, some other better bounds should be found in the future.

Another weakness of the project is in programming. When the upper bound is realized, exhaustive search is used as the main method. To make the program more efficient, a method of binary search is planned to apply. But it doesn't have a success, the program is continuously calculating and no results obtained. For some input which have large values, binary search will play a more important role that the ordinary exhaustive search.

Therefore, these weaknesses are the focal points that in the future work.

11. Bibliography

[1] Sebastian T.J. Fenn, Mohammed Benaissa, and David Taylor 'GF(2") Multiplication and Division Over the Dual Basis' *IEEE TRANSACTIONS ON COMPUTERS, VOL. 45, NO. 3, MARCH 1996*

[2] Venkatesan Guruswami 'Introduction to Coding Theory CMU: Spring 2010 Notes 2: Gilbert-Varshamov bound' January 2010

[3] R. W. Yeung and N. Cai, '.Network error correction, part I: basic concepts and upper bounds,' *Communications in Information and Systems, vol. 6, no. 1, pp*. 19 . 36, 2006.

[4] Ryutaroh Matsumoto, Member, IEEE, Kaoru Kurosawa, Member, IEEE, Toshiya Itoh, Nonmember, Toshimitsu Konno, Nonmember, Tomohiko Uyematsu, Member, IEEE 'Primal-dual distance bounds of linear codes with application to cryptography' *IEEE Trans. Inform. Theory, vol. 52, no. 9, pp. 4251-4256, Sept. 2006*

[5] This is an online programming tutorial of *C* programming language (http://einstein.drexel.edu/courses/Comp_Phys/General/C_basics/)

[6] Kean Hong Boey, Hodgers, P., Yingxi Lu, O'Neill, M, Woods, R. 'Security of AES Sbox designs to power analysis' *Electronics, Circuits, and Systems (ICECS), 2010 17th IEEE International Conference on*, 12-15 Dec. 2010

[7] Senay Yildiz, 'Construction of Substitution Boxes Depending on Linear Block Codes', The Middle East Technical University.

[8] Claude Carlet 'On Cryptographic Propagation Criteria for Boolean Functions' Information and Computation 151, 32_56 (1999),

[8] The Dual Code and Parity Check Matrices available at: http://msor.victoria.ac.nz/twiki/pub/Courses/MATH324_2009T2/WebHome/notes5.pdf

[9] Lars Eirik Danielsen, T. Aaron Gulliver, Matthew G. Parker 'Aperiodic Propagation Criteria for Boolean Functions' January 30, 2006

[10] Phyu Phyu Mar, Khin Maung Latt 'New Analysis Methods on Strict Avalanche Criterion of SBoxes' *World Academy of Science, Engineering and Technology 48 2008*

[11] K. Kurosawa and T. Satoh, "Design of SAC/PC(l) of order k Boolean functions and three other cryptographic criteria," in Advances in Cryptology – EUROCRYPTO'97, ser. Lecture Notes in Computer Science, vol. 1233. Springer-Verlag, 1997, pp. 434–449.

[12] Clark, George C., Jr., Cain, J.Bibb, '*Error-correction Coding for Digital Communications*' New York, Plenum Press. 1981

[13] Chapter 3 Linear codes---Michigan State University available at: http://www.math.msu.edu/~jhall/classes/codenotes/Linear.pdf

[14] Hamming,Richard W., '*Error detecting and eror correcting codes*' Bell System Technical Journal 29(2): 147-160

[15] Falkowski, B.J; Lozano, C.C; Luba, T., '*New Fastest Linearly Independent Tranforms over GF(3)*' 2007, International System.

[16] Kelly, John L. *'General Topology'* Springer-Verlag 1975

[17] Venkatesan Guruswami, 'Notes 4: Elementary bounds on codes' *Introduction to Coding Theory*, CMU: Spring 2010

[18] Raymond Hill, '*A First Course In Coding Theory*' Oxford University Press.

[19] This article is a online reading available at: http://www.scribd.com/doc/51426879/27/Hamming-bound

[20] Ivana Maric, Roy Yates, 'Performance of Repetition Codes and Punctured Codes for Accumulative Broadcast' WINLAB, Rutgers University,

[21] L. Storme, 'Linear codes meeting the Griesmer bound' Algebra, Krijgslaan 281-S22, 9000 Ghent, Belgium

[22] R.C. Singleton. '*Maximum distance q-nary codes*'. *IEEE Trans. Inf. Theory* 10: 116-118.

[23] F.J.MacWilliams and N.J.A.Sloane, *The Theory of Error-Correcting Codes*. Amterdam

[24] Christof Paar, Jan Pelzl, Bart Preneel, *Understanding Cryptography: A Teextbook for Students and Practitioners*. Springer.p.7

[25] Wong,C.K. Shi-Kuo Chang, '*Parallel Generation of Binary Search Trees*'. *IEEE Trans. Page 268-271*

[26] D. Bertsimas and J. N. Tsitsiklis, *Introduction to Linear Optimization*. Nashua, NH, USA: Athena Scientific, 1997.\

[27] K.Kurosawa and T.Satoh, '*Design of SAC/PC(l) of order k Boolean functions and three other cryptographic criteria*,' in *Advances in Cryptology- EUROYPTO'97*. Ser. Lecture Notes in Computer Science, vol. 1233. Springer-Verlag,1997,pp, 434-449.

12. Appendix

In this section the complete source codes are provided.

Source codes for the Upper bound

```
#include <stdio.h>
#include <math.h>

int Factorial(double n);
double C(double n, int k);
double Sum(int Uplimit,double n);

int main()
{
	int d1,d2;
	double n,m;
	double N,Nmax=0;
	printf("Please input d and d'.\n");
	scanf("%d %d",&d1,&d2);

	for(n=1.0;n<=500.0;n++)
	{
		for(m=1.0;m<=n-1.0;m++)
		{

	N=(pow(2.0,m)-1.0)/(pow(2.0,n)-1.0)*Sum(d1-1,n)+(pow(2.0,n-m)-1.0)/(pow(2.0,n)-1.0
)*Sum(d2-1,n);
			if(N<=1 && n>Nmax)
				Nmax=n;
		}
	}
	printf("%f.\n",Nmax);
	return 1;
}

int Factorial(double n)
{
	int F=1,i;
	for(i=1;i<=n;i++)
	{
		F=F*i;
	}
	return F;
```

```
}

double C(double n, int k)
{
	double m;
	if(k==0)
		m = 1.0;
	else if(n==0)
		m = 0.0;
	else if(k>=n)
		m = 1.0;
	else
		m = double(Factorial(n))/(double(Factorial(k))*double(Factorial(n-k)));
	return m;
}

double Sum(int upperlimit,double n)
{
	int i;
	double S=0;
	for(i=0;i<=upperlimit;i++)
	{
		S=S+C(n,i);
	}
	return S;
}
```

Table 3

Source code for the Lower bound related to Hamming bound

```
#include <stdio.h>
#include <math.h>

double LOG(double base, double x);//logy(x)    y is base
int Factorial(int n);
double C(int n, int k);
int OE(int p);
double L(int n,int d);

int main()
{
	int d1,d2;
	int n;
	double N;
	double Nmin;
```

```
	printf("Please input d1 and d2.\n");
	scanf("%d %d",&d1,&d2);

	N = LOG(4.0,L(1,d1))+LOG(4.0,L(1,d2));

		Nmin=N;
	printf("%f.\n",Nmin);
	for(n=2;n<500;n++)
	{
		N = LOG(4.0,L(n,d1))+LOG(4.0,L(n,d2));
		if(N<Nmin && N>0)
			Nmin = N;
	}
	Nmin=ceil(Nmin);
	printf("%f\n",Nmin);
}

double LOG(double base,double x)
{
	double logarithm;
	logarithm = log(x)/log(base);
	return logarithm;
}

int Factorial(int n)
{
	int F=1,i;
	for(i=1;i<=n;i++)
	{
		F=F*i;
	}
	return F;
}

double C(int n, int k)
{
	double m;
	if(k==0)
		m = 1.0;
	else if(n==0)
		m = 0.0;
	else if(k>=n)
		m = 1.0;
	else
```

```
            m = double(Factorial(n))/(double(Factorial(k))*double(Factorial(n-k)));
      return m;
}

int OE(int p)
{
      //int i;
      if(p%2 == 0)
            return 0;//p is even
      else
            return 1;//p is odd
}

double L(int n, int d)
{
      double l=0;
      int i;
      if(OE(d)==0)
      {
            for(i=0;i<=d/2-1;i++)
            {
                  l=l+C(n,i)+C(n-1,d/2-1);
            }
      }
      else if (OE(d)==1)
      {
            for(i=0;i<=(d-1)/2;i++)
            {
                  l=l+C(n,i);
            }
      }
      return l;
}
```

Table 6

Source codes for the lower bound which is based on Griesmer bound

```
#include <stdio.h>
#include <math.h>

int sum(int upper_limit, int d);

void main()
{
      int d1,d2,m,n;
```

```
    int N,Nmin;
    printf("Please input d1 and d2.\n");
    scanf("%d %d",&d1,&d2);

    Nmin=d1+d2+sum(1,d1)+sum(3-1-1,d2);
    //printf("%d\n",Nmin);
    for(n=3;n<=500;n++)
    {
    for(m=1;m<=n-1;m++)
    {
        N=d1+d2+sum(m-1,d1)+sum(n-m-1,d2);
        if(N<Nmin)
            Nmin=N;
    }
    }
    printf("%d\n",Nmin);
}

int sum(int upper_limit, int d)
{
    int s=0;
    int i;
    for(i=1;i<=upper_limit;i++)
        s=s+ceil(d/pow(4.0,i));
    return s;
}
```

Table 8