A SURVEY ON KNOWN ALGORITHMS IN SOLVING GENERALIZATION BIRTHDAY PROBLEM (K-LIST)

A THESIS SUBMITTED TO
THE GRADUATE SCHOOL OF NATURAL AND APPLIED SCIENCES
OF
MIDDLE EAST TECHNICAL UNIVERSITY

MINA NAMAZIESFANJANI

IN PARTIAL FULFILLMENT OF THE REQUIREMENTS
FOR
THE DEGREE OF MASTER OF SCIENCE
IN
MATHEMATICS

FEBRUARY 2013

Approval of the thesis:

A SURVEY ON KNOWN ALGORITHMS IN SOLVING GENERALIZATION BIRTHDAY PROBLEM (K-LIST)

submitted by **MINA NAMAZIESFANJANI** in partial fulfillment of the requirements for the degree of **Master of Science in Mathematics Department, Middle East Technical University** by,

Prof. Dr. Bülent Karasözen
Dean, Graduate School of **Natural and Applied Sciences**

Prof. Dr. Ferruh Özbudak
Head of Department, **Mathematics**

Prof. Dr. Ferruh Özbudak
Supervisor, **Department of Mathematics, METU**

Examining Committee Members:

Assist. Prof. Dr. Ömer Küçüksakallı
Department of Mathematics, METU

Prof. Dr. Ferruh Özbudak
Department of Mathematics, METU

Dr. Oğuz Yayla
Department of Cryptography, METU

Date:

Name, Last Name:	MINA NAMAZIESFANJANI
Signature :	

ABSTRACT

A SURVEY ON KNOWN ALGORITHMS IN SOLVING GENERALIZATION BIRTHDAY PROBLEM (K-LIST)

Namaziesfanjani, Mina
M.S., Department of Mathematics
Supervisor : Prof. Dr. Ferruh Özbudak

February 2013, 45 pages

A well known birthday paradox is one the most important problems in cryptographic applications. Incremental hash functions or digital signatures in public key cryptography and low-weight parity check equations of LFSRs in stream ciphers are examples of such applications which benefit from birthday problem theories to run their attacks. Wagner introduced and formulated the k-dimensional birthday problem and proposed an algorithm to solve the problem in $O(k.m^{\frac{1}{\log k}})$. The generalized birthday solutions used in some applications to break Knapsack based systems or collision finding in hash functions. The optimized birthday algorithms can solve Knapsack problems of dimension n which is believed to be NP-hard. Its equivalent problem is Subset Sum Problem finds the solution over $\mathbb{Z}/m\mathbb{Z}$. The main property for the classification of the problem is density. When density is small enough the problem reduces to shortest lattice vector problem and has a solution in polynomial time. Assigning a variable to each element of the lists, decoding them into a matrix and considering each row of the matrix as an equation lead us to have a multivariate polynomial system of equations and all solution of this type can be a solution for the k- list problem such as $F4, F5$, another strategy called eXtended Linearization (XL) and sl. We discuss the new approaches and methods proposed to reduce the complexity of the algorithms. For particular cases in over-determined systems, more equations than variables, regarding to have a single solutions Wolf and Thomea work to make a gradual decrease in the complexity of $F5$. Moreover, his group try to solve the problem by monomials of special degrees and linear equations for small lists. We observe and compare all suggested methods in this survey.

Keywords: Birthday Paradox, Cryptanalysis, Generalized Birthday Problem, K-tree Algorithms, Subset Sum Problem

ÖZ

GENEL DOĞUM GÜNÜ PROBLEMİNİ (K-LİSTE) ÇÖZEN BİLİNEN ALGORİTMALAR ÜZERİNE BİR ARAŞTIRMA

Namaziesfanjani, Mina
Yüksek Lisans, Matematik Bölümü
Tez Yöneticisi : Prof. Dr. Ferruh Özbudak

Şubat 2013 , 45 sayfa

Doğum günü paradoksu, kriptografik uygulamalardaki en önemli problemlerdendir. Doğum günü problemi, artan özet fonksiyonları, açık anahtar kriptografideki e- imzaları ve akan şifrelerdeki LFSR'ların az-ağırlıklı pariti kontrol denklemleri gibi sitemlere atak yapmada kullanılır. Wagner, k-boyutlu doğum günü problemini sunmuş ve formüle etmiştir, ayrıca problemi $O(k \cdot m^{1/\log k})$ karmaşıklıkta çözen bir algoritma tasarlamıþtır. Genel doğum günü çözümleri Knapsack temelli sistemleri kırmada veya özet fonksiyonlarına çarpıþma bulmada kullanılmıştır. NP- zor olduğuna inanılan n-boyutlu Knapsack problemi genelleştirilmiş doğum günü algoritmaları ile çözülebilir. Bunun eş-problemi, Z/mZ kümesinde tanımlanan Altküme Toplam Problemidir. Problemi sınıflandırmadaki temel özellik yoğunluktur. Yoğunluk yeteri kadar küçük olduğunda problem, en kısa latis problemine dönüşür ve problemin polinom zamanlı çözümü vardır. Listenin her elemanýna bir değişken atanmasıyla, bunların matris formunda ayrıştırılmasıyla ve matrisin her satırının bir denklem gibi düşünülmesiyle çok-değişkenli polinom denklem sistemi elde ederiz. Bu çeşit sistemlere bütün çözümler (örneğin: F4, F5, XL, sl) k-list problemine çözüm olabilir. Bu çalışmada k-list problemine çözüm algoritmaların karmaşıklığını azaltmaya yönelik yaklaşımları ve yöntemleri inceleyeceğiz. Özellikle değiþken sayısından fazla denklemin bulunduğu sistemlere Wolf ve Thomea'nın tek çözüm getiren yöntemi F5 algoritmasının karmaşıklığını önemli ölçüde düşürmüştür. Ayrıca, onların grupları, özel dereceli tek-terimlileri ve küçük listelerin doğrusal denklemlerini kullanarak problemi çözmeye çalışıyor. Bu araştırma çalışmasşnda bütün önerilen yöntemleri göstereceğiz ve karşılaştıracağız.

Anahtar Kelimeler: Doğum Günü Paradoksu, Kriptanaliz, Genel Doğum Günü Problemi, K-Ağaç Algoritmaları, Altküme Toplam Problemi

ACKNOWLEDGMENTS

I appreciate all the members of the char of Cryptography and IT-Security (www.cits.rub.de) placed in NA building of Ruhr University Bochum, Germany (RUB) for providing a pleasant time during my visit. I do specially appreciate the help of Dr. Christopher Wolf who played the main role during this research. I would like to express my gratitude to Applied Mathematics Institute at Middle east Technical University (METU) for their kind supports during my study. I would like to thank Dr. Amir Moradi (RUB) who gave me the chance being in RUB mainly by providing the fund from ECRYPT, accommodation and all he has done to support.

TABLE OF CONTENTS

LIST OF TABLES

LIST OF FIGURES

CHAPTER 1

INTRODUCTION

1.1 Birthday Paradox

Imagine there are total number of N people included you in a room. What is the probability that you have the same birth date with at least one of the other $B-1$ people of the room? If we assume that birthdays are uniformly distributed among all 365 days of the year, the above mentioned probability is the so-called birthday paradox and formulated in the following theorem [13]:

Theorem : Let $r_1, r_2, ..., r_n \in \{1, 2, ..., B\}$ be integers distributed independently and uniformly,

$$If \;\; n = 1.2 \times B^{1/2} \;\; then \;\; Pr[\exists i \neq j : r_i = r_j] \geq 1/2$$

Proof : (For uniform independent $r_1, ..., r_n$)

$Pr[\exists i \neq j : r_i = r_j] = 1 - Pr[\forall i \neq j : r_i = r_j] = 1 - (\frac{B-1}{B})(\frac{B-2}{B})...(\frac{B-n+1}{B})$

$= 1 - \prod_{i=1}^{n-1}(1 - i/B)$

Due to Taylor expansion $1 - x \leq e^{-x} = 1 - x + \frac{x^2}{2} - ...$ So we have :

$1 - \prod_{i=1}^{n-1} e^{-\frac{i}{B}} = 1 - e^{-\frac{1}{B}\sum_{i=1}^{n-1} i} \geq 1 - e^{-\frac{n^2}{2B}} = 1 - e^{-0.72} \geq 1/2.$

The reason it is called a paradox is because it is very paradoxical that the square root function grows very slowly. In particular if we try to apply the above theorem to birth dates, then lets assume that we have a number of people in a room, and lets assume that their birth dates are independent of each other and are uniform in their interval one through 365. Then what the Birthday Paradox says is that we need roughly 1.2 times the square root of 365. Which is something like 23, which says we need roughly 23 people in a room, and then with probability 1/2, two of them will actually have the same birth date [13].

Figure 1.1 shows how to graph the behavior of Birthday Paradox. So here we set $B = 10^6$, in other words we are picking random uniform samples in $[1, ..., 10^6]$. And the X axis here, is the number of samples that we are picking, and the Y axis, is the probability that we get a collision among those N samples. So we see that the proba-

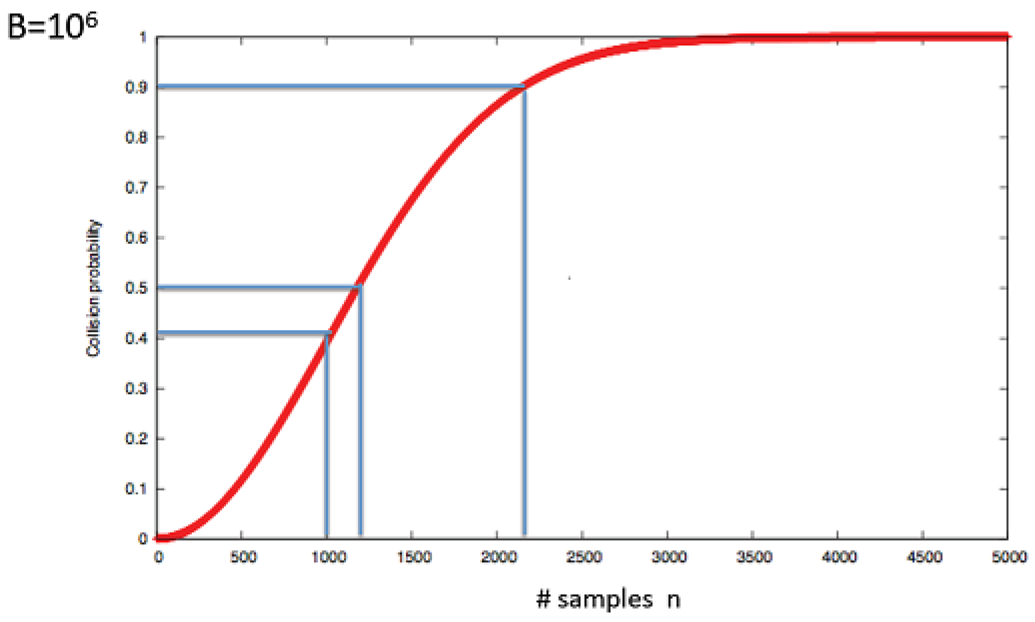

Figure 1.1: Birthday Paradox Behavior

bility of 1/2 happens around $1.2\sqrt{B}$. Roughly 12000. And if we look at exactly $\sqrt{B}$, the probability of a collisions is around 0.4 or 0.41. So we notice that the probability goes up to 1 extremely fast [13].

1.2 Cryptographic Applications

1.2.1 Birthday Attacks and Hash Functions

Weak and strong collision resistance are hash functions responsibility. we have a hash value $h(x)$ and can find ω such that $h(\omega) = h(x)$ then we can say that the hash function is broken because we changed weak collision resistance property of the function. If the hash function generates n-bit output we have to compute 2^n hashes before expecting to find such a ω, the same as brute force. If we want our hash to stay secure it should be infeasible for adversary to compute 2^n hashes. As we discussed above we need $2^{n/2}$ hashes in order to find a collision . Nowadays with parallel computing they improve birthday attacks in hash functions [12].

1.2.2 Digital Signatures

In the next five steps we conduct a birthday attack on digital signature:

1. Alice selects an evil message E to send to Bob and ask him to sign but Bob does not want to sign the message.

2. Also, Alice selects another innocent message I and she is sure that Bob is willing to sign it.

3. Then, Alice generates $2^{n/2}$ versions of the innocent messages denoted by I_i witch differs in a few bits. They are the same as I but as they change in some bits their hashes also change.

4. Similarly, she does the same for evil message. So, she has $2^{n/2}$ versions of E, they are E_i's.

5. Alice hashes all E_i's and I_i's with expectation of finding a collision $h(E_j) = h(I_k)$. If Alice has that collision, she sends I_k to Bob to sign the message. He signs and return I_k and $[h(I_k)_{Bob}]$ to Alice. Since $h(E_j) = h(I_k)$ therefore, $[h(E_j)]_{Bob} = [h(I_k)]_{Bob}$. So that, in this way by birthday paradox Alice finds Bob's signature on evil message E_j [12].

Furthermore, there are so many attacks in literature on hash functions MD4, MD5 and SHA1. As an illustration Kelsey and Kohno in a attack called Nostradamus attack [14] hired Merkle-Damgard setting to break MD4 or another one is Wang's method to attack on MD5 [15].

1.3 Generalized Birthday Problem (K-List)

K-list problem in word means: choosing x_i's from each given list $L_1, L_2, ..., L_k$,
$x_1 \in L_1,\ x \in L_2, ...,\ x \in L_k$, elements drawn uniformly and independently at random from $\{0,1\}^n$, such that $x_1 \oplus x_2 \oplus ... \oplus x_k = 0$ [1].

Dealing with k-list problem mathematically leads us to the following definition:
Definition 1(k-list problem): Let $n = kl$ for two integers $k, l \in Z^+$. Denoting $L_i = \{s_i^{(0)}, s_i^{(1)}, ..., s_i^{(j)}\}$ be k lists for $0 \leq i \leq k-1$, $0 \leq j \leq 2^{l-1}$. These elements are pairwise distinct, i.e. $\forall i \in \{0,1,...,k\}, m, n \in \{0,...,n-1\} : s_i^{(m)} = s_i^{(n)} \Leftrightarrow m = n$. Each $s_i^{(j)}$ is one element of the list consisting n-bit strings,$s_j^{(i)} = (x_{j,0}^{(i)}, ..., x_{j,n}^{(i)})$ from F_2^n drown uniformly and independently at random.
The k-list problem is defined by finding a k-tuple $(v^{(0)}, ..., v^{(k-1)}) \in L_0 \times ... \times L_{k-1}$ such that:

$$\textstyle\sum_{i=0}^{k-1} v^{(i)} = 0^n. \quad (1)$$

We call (1) the *central condition* of k-list problem in $GF(2)$. When the list elements are integers from $\mathbb{Z}/m\mathbb{Z}$ The problem has another definition and called Subset Sum problem:

Definition 2(Modular Subset Sum problem): Let $a_1, a_2, ...a_n$ and target value t are given from $\mathbb{Z}/m\mathbb{Z}$. The goal is to find a subset of a_i's sum to t in $\mathbb{Z}/m\mathbb{Z}$, i.e. finding $x_i \in \{0, 1\}$ such that

$$\sum_{i=1}^{n} a_i x_i = t \; mod \; m. \; (2)$$

We call (2) random if n, m and t are fixed parameters but a_i's are drawn uniformly at random in $\mathbb{Z}/m\mathbb{Z}$ [35].
A convenient classification was based on density defined below:

Definition 3(Density of MSS): $\frac{n}{\log^2 m}$ is called density of the modular subset sum problem instance. The problem falls into sparse range when the density is less than 1 and the other greater densities give us the dense instances of the problem.

Because there were too many algorithms and application that suggest any solution for different kinds of the problem in this survey I had to be selective and choose some well known and useful algorithms.

We have general solutions for the problem proposed by Schroeppel-Shamir [9], Wagner [1], Minder-Sinclair [3], Coron and Joux [18], Augot et, al [5], Bellare et.al [11] and Flaxman-Przydatek [20].
Dense cases started to be proposed by Chaimovich [19] in 1999, followd by Lyubashevsky (2005) [16] completed by Shallue [35] in 2007.
Sparse cases have density $d < 0.654$ and decrease the problem to fall into shortest lattice vector problem by Lagarias-Odlyzko [21] in 1985 and for all $d < 1/n$ the problem solved by polynomial time algorithms using LLL.
All the methods used in PoSSo, polynomial system solving methods, can be used to solve the k-list problem by considering the problem as a system of polynomial equations. These methods belonge to Buchberger [31] family to compute the Gröbner bases of the system such as $F4$ [25] and $F5$ [26] algorithms, linearization [33], eXtended Linearization (XL) algorithms [33] and new approaches to decrease the complexity of the mentioned algorithms to solve the polynomial equation system more efficiently.

1.4 Organization

History and general solutions of the problem is the topic of the second chapter.

In the third chapter of this survey we choose some dense and sparse cases proposed new algorithms with different time and space complexities to solve the problem.

Fourth chapter is belonged to PoSSo methods .

In the fifth chapter we talk about new approaches for solving the k-list problem as discussed in non-linear multivariate system of equations in finite fields. In non-binary fields by modelizing the k-list problem as a set of equations led us to investigates all polynomial system solving methods.

Last chapter is comparison and conclusion of the survey.

CHAPTER 2

HISTORY

2.1 First Glance

In 1973, Knuth was the first one used the priority queue in generating pairwise sum in sorted way for just 2 lists [2].

2.2 Camion and Patarin

In 1991 Paul Camion and Jacques Patarin [8] are first people in the history mentioned birthday problem in general for their applications. They were working on hash functions among a knapsack system introduced in Crypto'89. They hired an algorithm which worked the same as k-tree algorithm and broke the system in $O(2^{32})$ steps. They were successful to do a very slight optimization for this complexity and decreased it to $O(2^{30})$ [8].

2.3 Schroepple and Shamir

In 1981 Schroepple and Shamir [9] combined dynamic programming techniques with divide - and - conquer algorithm to prove their theorems in solving the problem for $k = 2$ and $k = 4$, named k-table sorted algorithm, a pairwise sum in sorted order, with less time complexity rather than brute force by the following method:

Definition : Let T_i be the table of k problem/solution with the property of having $O(2^{n/k})$ solvable problems in each. By problem P and operator $\oplus$, the procedure is based on determining k representatives $P_i \in T_i$ s.t :

$$P = P_1 \oplus P_2 \oplus ... \oplus P_k.$$

Repeatedly they divide the space into k subspace i.e. sub lists until getting k problems, each of them should be in size n/k and finish by searching all over the k problem/solution tables. The 2-table problems discussed by Knuth previously [2]. The

problem of 4-table is solved in $0(2^{n/2})$ time and $0(2^{n/4})$ space :
They suggest a 4-table balanced algorithm as follows [9] :
Let store pairs of the problem chosen from T_1 and T_2 in $Q^{'}$ and the rest in $Q^{"}$. In the second phase they sort T_2 into increasing order and T_4 into decreasing order and then, iterate in all $Q^{'}$ and $Q^{"}$ until one of them became empty. They look for pairs with the smallest sum $P_1 \oplus P_2$, called them (P_1, P_2) and the largest sum $P_3 \oplus P_4$, called them (P_3, P_4). They define S as the sum of $(P_1 \oplus P_2) \oplus (P_3 \oplus P_4)$. If this S equals to P the problem called solvable and they stop here if not, they delete pairs from Q's.

They decrease the time complexity of the algorithm from brute force $O(2^n)$ to $O(2^{n/4})$ space and $O(2^{n/2})$ time for 4- list type of the problem [9].

2.4 Chose, Joux, and Mitton Algorithm

In stream ciphers there is always a linear correlation between the stages inside the LFSR and it's nonlinear functions' output bits. This linearity always exists and should be as strong as possible because that is the target of cryptanalyses in correlation attacks [10].
As a result of doing fast correlation attack on stream ciphers, Chose and his colleague were look for a solution to parity check equations which is the all fast correlation attacks in common point. On the other word, the linear relations between register output bits x_i. Therefore, They look for all k-weighted parity check equations witch attacked D target bits of the LFSR. The equations are :

$$A(x) = x_{m_1} \oplus x_{m_2} \oplus ... \oplus x_{m_{k-1}} \oplus \sum_{j=0}^{B-1} c_{m,i}^{j} x_j$$

$A(x) = \sum_{j=0}^{L-1} a_j x_j$, $x = [x_1, \ldots, x_{L-1}]$ and the values of a_j's are constant.
$A(x)$ is used just in even cases and for odd cases it is equal to zero. The $c_{m,j}$ are binary coefficients in parity check and m_j's are all output bit's divisions. Their suggested algorithm was based on match - and - sort algorithm , solved the problem in $N^2 \log N$ time for $k = 4$.
They select k as a parameter for determining the weight of the parity check equations among all $N^{K^{'}}$ combinations, whereas the algorithm suggests to divide the space to smaller sub-spaces in order to find less restrictive collisions there, sorting and collecting the result would complete the whole task for the bigger space. They split k among for integers l_1, l_2, l_3, l_4 and solve the equation

$$l_1 + l_2 + l_3 + l_4 = k$$

and for $i = 1, ...4, l_i = \lfloor k/4 \rfloor$ or $\lceil k/4 \rceil$.
They calculated the sums of l_2 output bits in terms of the L-bit of initial state :

$$x_{j_1} \oplus ... \oplus x_{j_{l_2}} = \sum_{k=0}^{L-1} u_k x_k$$

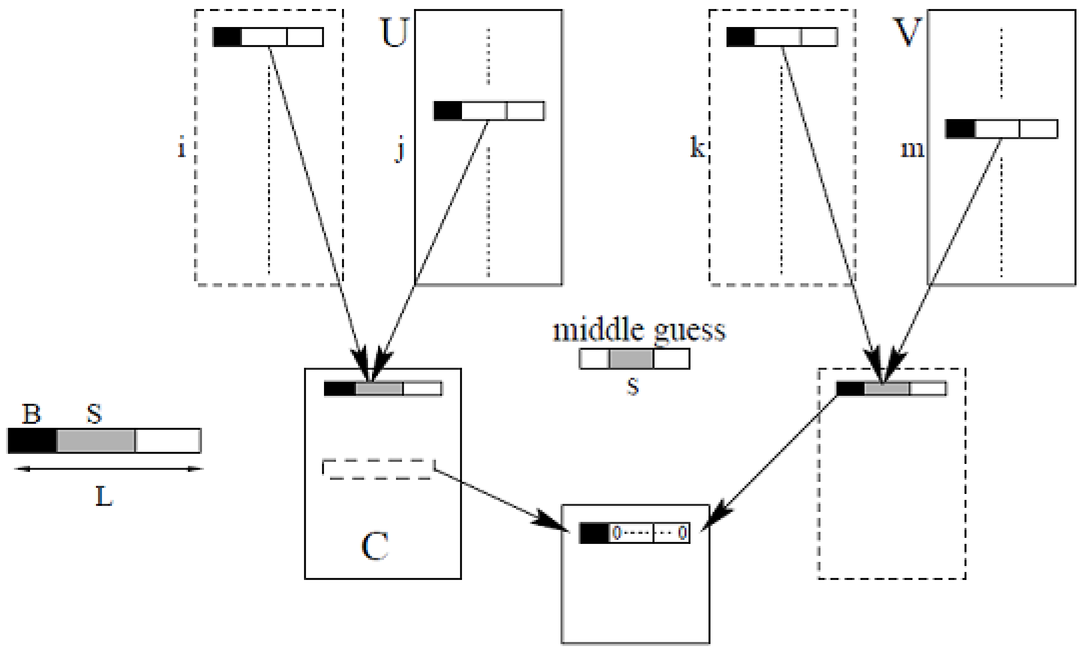

Figure 2.1: The match and sort algorithm in parity check equations finding

Such that $u = u_0 ...., u_{L-1}$. The algorithms saves all the u entries in U and construct another table with the same way for sums of l_4 output bits, stores them in V. Then, match the elements from U by sums of l_1 and V by l_3 output bits. Finding the partial collision and combining them with already found bits complete the search [10]. The match and sort algorithm in parity check equations finding can be illustrated in Figure2.1 by chose et.al [10].

Let k' be the weight of parity check equations then, the complexity of match and sort algorithm for even cases calculated in $O(N^{\lceil k'/2 \rceil} \log N)$ time and $O(N^{\lfloor (k'+1)/4 \rfloor})$ space. When k is odd $k' = k - 1$ and $A(x)$ represents x_i and the algorithm runs D times. When even cases are issude $k = k'$ and $A(x) = 0$. In terms of time complexity it is more or less the same as square root algorithm but it is very sufficient for memory complexity.

2.5 Bellare, et al., Algorithm

One of interesting points of view in k-list is what Bellare, et.al., [11] mentioned in their algorithm. They solve the problem in $GF(2^n, \oplus)$ in time complexity of $O(n^3 + kn)$ by the following methodology while working on finding collision free hash functions. Their attack was successful in breaking XHASH with introducing the idea of reducing the problem to a set of linear equations, and then, applying Gaussian elimination to peak the attack [11].
The attack is done on a hash random function from $\{0,1\}_l$ to $\{0,1\}_k$.
We have k bit string z, by applying the algorithm we are able to calculate $z = XHASH_h(x)$ where $z \in \{0,1\}_k$. To mounting the attack they reduced the prob-

lem to system of linear equations in $(F_2^n, \oplus)$. By assigning variables and values to the elements of the equations they had $n+k$ equations in $2n$ unknowns over the finite field of size 2. Then they proved a lemma which implied that the probability of existence of the solution is 1/2. they fixed $n = k + 1$ and solved the set of the equations by Gaussian elimination, then they got $n+k = 2k+1$ equations in $2n = 2k+2$ unknowns which makes the system slightly under - determined [11].

Scientists who worked on solving multivariate quadratic systems (MQ) equations, has completed the idea used by Bellare. They considered the so-called k-list problem as a system of equations and variables over finite field, try to introduce algorithms such as Gröbner based algorithms, F_4 and F_5 [24, 25], or other methods like *eXtended Linearization* (XL) strategies [33]. We are going to introduce such systems in chapter three.

David Wagner was the first one who particularly worked on formally describing the k- list problem. He had improvement to generalizing his idea for even groups with operations other than xor [1] but the algorithm failed for small values of k [3].

2.6 Wagner's Algorithm:

The standard technique for classic birthday problem is finding all match elements between two given lists. Wagner defines a joint operation $\bowtie$ [1]. The operation $S \bowtie T$ lists all common elements in both S and T. Therefore, a solution to traditional two lists birthday problem can be found just by calculating the join $L_1 \bowtie L_2$ of the two given lists L_1, L_2. Join operator has been known as an efficient way in many subjects in literature such as data base query evaluation. We have merge-join which sorts the two lists, L_1, L_2, and finds any matched pairs it detected by scanning via two sorted lists.
In a hash-join operation, it uses storing one of the lists L_1 in hash table, and by scanning through all elements of the other list L_2, checks if it is shown before in hash table or not. Hash-join table is very efficient in case of having enough memory available: It requires $\mid L_1 \mid + \mid L_2 \mid$ steps and $\min(\mid L_1 \mid, \mid L_2 \mid)$ storage units [1].

Wagner observed those above operations and came up with the result of solving birthday problem with square-root complexity. If the operation is done on n-bit values and we are free to choose the size of the lists as we desire, the above algorithms will take $O(2^{n/2})$ steps of operation. Also, there are techniques for reducing the space complexity of the mentioned algorithm in particular cases [1].

As an extension of all the above algorithms Wagner has used join operator to solve 4-sum problem. Detection all values x_1, x_2, x_3, x_4 from lists $L_1, ..., L_4$ such that $x_1 \oplus ... \oplus x_4 = 0$ is our main task. It is obvious that if they choose the lists of size at least $2^{n/4}$ they will have the solution with high probability. All previous known algorithms up to Wagner's time found the solution in at least $O(2^{n/2})$. But Wagner solved the 4-sum problem in more efficient way by defining an operator named generalized join operator $\bowtie_l$. while $low_l(x)$ defined the least significant l bit of x, so that, $L_1 \bowtie_1 L_2$ agrees in l least significant bits from all pairs in two given lists $L_1 \times L_2$. By using the following operations he suggested a new algorithm for the 4-sum problem. A disadvantage of the algorithm is that it just finds the solutions with the property that they agree with their low l bits just in zero value. Later on they suggest another approach to eliminate this restriction. I mention the basic observations in the following:
Observation 1: $low_l(x_i \oplus x_j) = 0 \Longleftrightarrow low_l(x_i) = low_l(x_j)$.

Observation 2: let L_i, L_j be our lists, all pairs like $< x_i, x_j >$ can easily generate and satisfy $x_i \in L_i, x_j \in L_j$, and by using the join operator $\bowtie_l$ we have $low_l(x_i \oplus x_j) = 0$.

Observation 3: If $x_1 \oplus x_2 = x_3 \oplus x_4$, then $x_1 \oplus x_2 \oplus x_3 \oplus x_4 = 0$.

Observation 4: If $low_l(x_1 \oplus x_2) = 0$ and $low_l(x_3 \oplus x_4) = 0$, then we have $low_l(x_1 \oplus x_2 \oplus x_3 \oplus x_4) = 0$, and the probability of the case equals to $\frac{2^l}{2^n}$.

He extends the list to have 2^l elements each. l is the height of the lists and define later. As we can see in the figure below the algorithm generates pairs from large list L_{12} using observation 2 with values $x_1 \oplus x_2$ such that $low_l(x_1 \oplus x_2) = 0$. With the same way list L_{34} is generated with values $x_3 \oplus x_4$ while their l least significant bits xor equals to zero, the algorithm searches to find the matches between L_{12} and L_{34}. Observation 3 says that any kind of such matches satisfies $x_1 \oplus x_2 \oplus x_3 \oplus x_4 = 0$ and

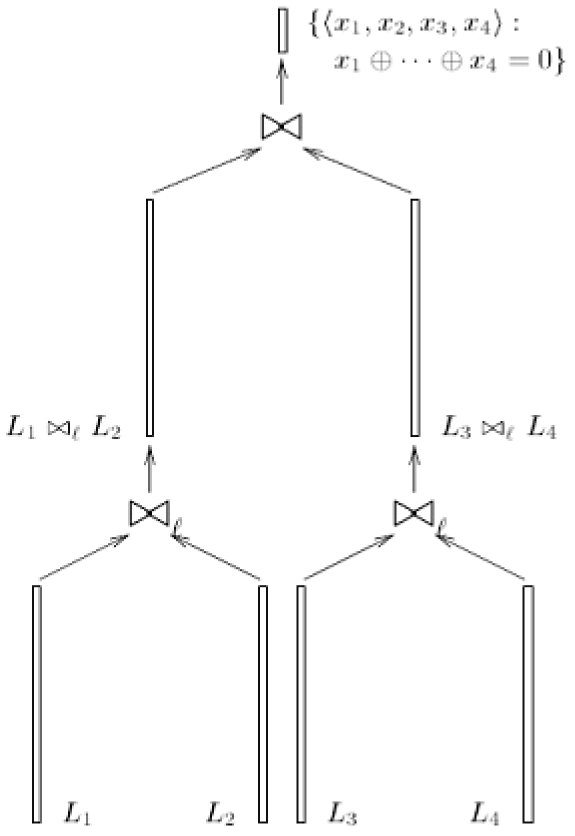

Figure 2.2: Wagner's Algorithm for $k = 4$

hence we can extract a solution for 4-sum problem [1]. Figure 2.2 shows the behavior of the algorithm for $k = 4$

The algorithm steps can be implemented in $O(2^{n/3})$ complexity. This is true because we know that $\Pr[low_l(x_1 \oplus x_2) = 0] = 1/2^l$ while x_1 and x_2 are chosen at random and uniformly so that, by birthday paradox we have :

$$E[| L_{12} |] =| L_1 | \times | L_2 | /2^l = 2^{2l}/2^l = 2^l.$$

The same is true for L_{34}. It means we expect the size of 2^l for each list. And by observation 4 we will have a match in pairs of elements from $L_{12} \times L_{34}$ with probability $2^l/2^n$. By using birthday paradox in second invocation we could find the approximate number of common elements match between two lists equals to $| L_{12} | \times | L_{34} | /2^{n-l}$. This number equals to 1 when $l > n/3$. Therefore, by setting $l = n/3$ at the first steps of the algorithm we are able to find a non-trivial solution to 4-sum problem [1].
To resolve the disadvantage of the algorithm I mentioned before (It just solves the problem with the property that the pairs xor just are zero in their least l significant bits), Wagner suggested to choose a random l-bit value namely α from the set of the solutions, and try to find pairs (x_1, x_2) and (x_3, x_4) such that their low l bits xor to α. This means we should calculate $(L_1 \bowtie (L_2 \oplus \alpha)) \bowtie (L_3 \bowtie (L_4 \oplus \alpha))$.
Moreover, the value 0 in $x_1 \oplus x_2 \oplus x_3 \oplus x_4 \oplus = 0$ can be replaced by any constant value c without loss of generality and increasing the complexity of the algorithm. To

prove this claim they replace L_k with $L_k^* = L_k \oplus c = x_k \oplus c : x_k \in L_k$, therefore, any solution to $x_1 \oplus x_2 \oplus ... \oplus x_{k-1} \oplus x_k^* = 0$ will be a solution to $x_1 \oplus x_2 \oplus ... \oplus x_k = c$ and vice versa. then we could imagine $c = 0$.
Wagner claimed that he could solve the k-sum problem even more faster for larger values of k while k is a power of two, in illustrated figure he replaced full binary tree of depth $\log k$. At internal nodes of height h, they used the join operator $\bowtie_{l_h}$ (where $l_h = hn/(1 + \log k)$), the root is an exception and uses full join operator $\bowtie$. He suggested an algorithm with time and space complexity of $O(k.2^n/(1+\log k))$ whereas he extended the lists size to be $O(2^{n/(1+\log k)})$. Each internal list's elements points back to $x^{'}$ and $x^{"}$ of the two child lists in forms of $L_{i,...,j}$, in a way of being $x = x^{'} \oplus x^{"}$. The complexity of the algorithm increases slightly by increasing k just in case k becomes very large the algorithm does not seem very sufficient [1].

Investigating on Wagner's algorithm on subset sum problem when the two list has in common elements in their last l bits other than 0. In this case the constructing L_{12} from L_1 and L_2 when the list are in interval $\{\frac{-R}{2}, \frac{R}{2})$ is after sorting two lists for $b \in L_1$ pick a random $c \in L_2$ from interval $\{-b - \frac{Rp}{2}, -b + \frac{Rp}{2})$ and then the desire new list L_{12} is $L_{12} \cup \{b + c\}$. It outputs the list in at most N elements and at most one $b + c$ is enters to the algorithm.

The complexity of the algorithm is $O(km^{1/\log k})$ [1].

2.7 Minder and Sinclair Algorithm

Wagner's algorithm fails when the length of the lists are smaller than $2^{n/(q+1)}$ [3]. The algorithm presented in this paper is an interpolation between brute force and Wagner's algorithm [3].
Furthermore, They generalized their proposed algorithm to non- binary vectors but over finite field F_r which means they are looked for solution to $\lambda_1 x_1 + \ldots + \lambda_k x_k = 0$ where $\lambda_i \in F_r^*$ and $x_i \in L_i$.

They considered the problem with $k = 2^q$ random lists number enter to the problem while q is the round number of the procedure. Each list contains n-bit vectors, $L_1, ..., L_k$, each of length m. A necessary condition for having the solution with high probability in their idea is connected to the size of the list to be grater than $2^{n/2^q}$ i.e. the expected number of solutions in this case is at least 1.

In many application we are not allowed to change the values of m, n and q as in finding a sparse feedback polynomial for a given LFSR. They fixed the value of q while it calculated the Hamming weight of the polynomial. Increasing the size of the list has effects on degree of the polynomial have to be determined. In 2009 Minder fixed m, n and q with the target of finding the answer of the problem as fast as possible with their special setting [3].

They claim to find the solution for any parameters of k with the property

$$2^{n/2^q} \leq m \leq 2^{n/(q+1)}.$$

When $m = 2^{n/(q+1)}$ it equals to the k-tree algorithm, and at the other hand when $m = 2^{n/2^q}$ it is the same as brute force algorithm.

The algorithm starts with combining two lists to have a new list, merging L_1 and L_2 to peak L'_1 in one round. So that in each round it halves the number of the list. Particularly, the list L'_i contains $x + y$ pairs such that $x \in L_{2i-1}$ and $y \in L_{2i}$ such that $x + y$ is zero on the first l_1 bits. They select l_1 as a parameter for optimization of the algorithm performance. they eliminate l_1 bits in first round. They continue the same procedure to the second round with merging $L'_1, \ldots, L'_{2^{q-1}}$ they come up with new lists of the form $L''_1, \ldots L''_{2^{q-2}}$. Sequences of l_2 bits are eliminating cause to have vectors are zero in their first $l_1 + l_2$ bits. Applying the same idea to the whole lists, at the end of the q-th round they have one list containing vectors that agree on their first $\sum_{i=1}^{q} l_i$ bits where each of them is $\sum_{i=1}^{k} x_i$ and $x_i \in L_i$. Their target is to find get sums that are zero in their all n bits, the last list contains that forms of sum lead to :

$$\sum_{i=1}^{q} l_i \geq n.$$

This can be illustrated in binary tree of height q. A list of vectors includes in each list. Level j of the tree shows the algorithm's lists after j round. Due to having random lists as inputs of the algorithm and random variables in internal nodes of the tree they choose M_j to represent the length of the list at j-th level. by values of m and L_i's the distribution of M_j can be determined. The total number of the solutions is M_q and we discuss about a parameter which is the expected number of founded solutions and write it as 2^c. The goal is satisfying the following inequality for sure :

$$E[M_q] \geq 2^c.$$

If we put $c = 0$ in expectation we will have just a single solution. The probability of failing the algorithm is connected to making c to get slightly larger. Therefore, the started point setting changes by contributing the c value in the position to be :

$$m \geq 2^{(n+c)/2^q}$$

and assuming to have the condition :

$$m \geq 2^{(n+c)/(q+1)}$$

and for technical reasons :

$$c < 2 \log m$$

while Wagner's algorithm is sufficient for all larger values of m. As they mentioned in the paper the choice of l_i fro example increasing that value affects $E[M_j]$ in contrast way and decreases it. So that, for optimization they tried to propose a way in how to choose the initial values such that the whole properties also hold. They reduce the way of finding l_i to an integer program. The mail target was finding an optimized l_i applied to the 2^c expected solution in least possible complexity by given m, n, q and c. By considering the linear programming relaxation for the integer program they succeed to prove that the suggested three phases program could find the optimal l_i with holding all defined setting as you can see in the following: $b_i = 2^i b_0$ $l_i = 0,$ $for \quad 1 \leq i < p;$
$b_p = u$ $l_p = 2^p b_0 - u;$
$b_i = u$ $l_i = u,$ $for \;\; p < i < q;$
$b_q = c$ $l_q = 2u - c$
where p is the least integer such that

$$n \leq (q - p + 1)2^p \log m - c,$$

and u is calculated by

$$u = n + c - 2^p \log m/q - p.$$

They claim that they algorithm for all values of n, q, m and $2^n/2^q \leq m \leq 2^{n/(q+1)}$ works in complexity of $O(2^{q+u^*(n,m,q)})$, where $u^*(n, m, q)$ is the optimal value of u.

They generalized the k-list problem to non-binary lists which means finding non-trivial linear combination of vectors that sum to zero. the formulation is: Let r be a prime power. We have $k = 2^q$ lists $L_1, \ldots, L_k$, each of length m, each contains independent, uniform random vectors from F_n^r, r is a prime power. They try to find $x_1 \in L_1, ..., x_k \in L_k$ and $\lambda_1, \ldots, \lambda_k \in F_r^*$ such that $\lambda_1 x_1 + \ldots + \lambda_k x_k = 0$.

They used the same merging idea over finite fields and find a solution based on linear algebra which solves the system of the equations in $\lambda x = 0$ [3].

2.8 Coron and Joux

A variant version of Wanner's algorithm [1] is available by what Coron [18] proposed an attack on provably secure hash functions of Augot et, al. [5] for special parameters which the Coron's attack is practical for their mentioned two sets of parameters based on Wanger's [1] birthday algorithm.
The hash functions is initialized as follows:
They denoted s as a length of input messages m while s has w blocks of a bits, $s_1, \dots, s_w$, each s_i has a bits. and r is the length for output message h then Let $u = 2^a$ and produces a random matrix H which divided into w sub-matrices H_i of size $r \times w$. Now each s_i should convert into an integer between $[1, \dots, u = 2^a]$. Then they choose the corresponding column in sub-matrix H_i. To have the output h which is $r$bit-string the xored the chosen columns.
To make any attack unpractical Augot et al. set three set of parameters:
1. $r = 160,\ w = 64, u = 256, n = 2^{14}$ with security level of $2^{62.3}$.
2. $r = 224, w = 96, u = 256, n = 3.2^{13}$ the security level is $2^{82.3}$.
3. $r = 288, w = 128, u = 64andn = 2^{13}$.
Coron et, al. attacked the first two set more faster based on the following idea:
The goal is to find the collision in the described hash function i. e. with two messages $m \neq m^{'}$ such that $H(m) = H(m^{'})$.
They selected two columns of per w sub-matrix H_i with u columns such that the $2w$ columns xor would be zero. With these xor values they generated a list for each sub-matrix H_i which was roughly $u^2/2$ values $x_i$to have:

$$x_1 \oplus x_2 \oplus ... \oplus x_w = 0.$$

Moreover, they define l as value with $2^l = u^2/2$. they looked among 2^l elements with their l most significant bits equals to zero. There were 2^{2l} elements $x_1 \oplus x_2,\ x_1 \in L_1$ and $x_2 \in L_2$. Created the same lists with the other L_3, L_4 in $O(2^l)$ time. With the corresponding tree and applying birthday paradox the collision can be find in $O(w.2^l)$ if:

$$r \leq (\log_2(w) + 1).l$$

where $l = 2log_2(u) - 1$.
For Generalization they worked with the all 2^{2l} elements $x_1 \oplus x_2$ and building the same tree with depth of 2^{2l-1}. The collision would be obtained in $O(w.2^{2l})$ if $r \leq (\log_2 w).l$. They break the first set in time 2^{36} and the second in time 2^{33}.

2.9 Augot,Sendrier and Finiasz [5]

Their algorithm is a following for to attack hash functions like what Coron and Joux did.
The defined the problem in lists of power 2
Theorem: Let $L_1, L_2, ..., L_{2^a}$ be our 2^a lists of r bit-strings and each list has size of $2^{\frac{r}{a+1}}$. Then the problem $x_1 \oplus x_2 \oplus ... \oplus x_{2^a}$ with $x_i \in L_i$ can be solved in $O(2^a.2^{\frac{r}{a+1}})$ time and space.
The idea is as Wagner's algorithm merging and sorting the lists but in order to extend the problem to have a general algorithm when the list's size is not exactly $2^{\frac{r}{a+1}}$ they solved the problem in two cases.
First case is $l < \frac{r}{a+1}$. The procedure is the same as Wagner proposed, zeroing l bits of the partial sums instead of zeroing $2^{\frac{r}{a+1}}$ bits. The lists size remain constant and at the end of the procedure they have $(a-1).l$ bits equal to zero. If the algorithm is not successful to find a collision chooses another pair to zeroing and starts the procedure from the beginning. The probability of finding any collision in this method is $\frac{2^{(a+1).l}}{2^a}$. The complexity of constructing the two final lists is $O(l.2^a 2^l)$ so, the total complexity is $O(l.2^{r+a-al})$ lead to have the lists of size smaller than $2^{\frac{r}{a+1}}$ to hold the complexity. Secondly dealing with the values of $l > \frac{r}{a+1}$. Here they have a precomputation step to build a new list such that they choose a parameter named α and try to make α bits equal to zero. This process makes the size of the lists equal to an average $2^{l-\alpha}$. In terms of tackling with the hash functions in this step they had to use two sub-lists and try to merge them using birthday paradox and finally benefit from Wagner's algorithm to zero the remaining $r' = r - \alpha$ bits. To have an ideal algorithm adapting with Wagner's algorithm they must have

$$l - \alpha = \frac{r'}{a+1}$$

. By solving these two equations they have

$$\alpha = \frac{l(a+1) - r}{a} \qquad r' = \frac{a+1}{a}(r-l).$$

The cost for recomputing the lists using birthday paradox is $O(2^a 2^{l/2})$ and the total complexity of Wagner's algorithm here is $O(r' 2^a 2^{\frac{r'}{a+1}})$.
Solving the equations help us to know that this is only true for $l > \frac{2r}{a+2}$. This means we are in a step that only could use $a+1$ for the attack.

CHAPTER 3

DENSE CASE ALGORITHMS

3.1 Lyubashevsky [16]

If we consider the generalized birthday not in binary lists but integers mod t and it is called subset sum (SS) problem, for a given n integers and a target number t we are supposed to find a subset of a_i's in a way that the sum equals to t.
In 2004 Lyubashevsky introduced a polynomial time algorithm algorithm for solving a version of the problem which believed to be in NP-hard, called Random Modular Subset Sum (RMSS) problem which means the subset a_i's generated in the range of $[0, M]$ and the subset sum t is in mod M in time and space complexity of $2^{O(\frac{n^\epsilon}{\log n})}$ when the parameter $M = 2^{n^\epsilon}$.
The problem is defined to pick n random $x_i \in \{0, 1\}$ and output

$$\sum_{i=1}^{n} x_i a_i (mod\ M).$$

They proved a corollary which says for the given list L consist of uniformly distributed elements in range $[0, M)$ and a target t, there is an algorithm that returns an element from each list such that their sum equals to t mod M because They did a subtraction in each list's element then they convert each integers in lists from interval $[0, M)$ to the interval $[-\frac{M}{2}, \frac{M}{2})$. this was done simply by subtracting every element in the lists from every integer greater or equal that $\frac{M}{2}$. Now as they used just one element from each list and totally $-t$ numbers used they applied their theorem which was a modified version of Wagner's algorithm. It chose just one element from each list in interval $[-\frac{M}{2}, \frac{M}{2})$ such that their sum equals to 0 with a good probability. Therefore the sum of $-t$ numbers in the interval $[0, M)$ equals to t mod M.

The following algorithm finds completes their proof.

```
Subset Sum$(a_1, ..., a_n, t, M)/HereM = 2^{n^{\epsilon}}$.
(1) Break up the $n$ numbers into $\frac{1}{2}n^{1-\epsilon}$ groups each containing $2^{n^{\epsilon}}$ numbers.
(2) For group $i = 1$ to $1/2n^{1-\epsilon}$do
(3) List $Li$ =Generate List From Group$(a_j | a_j \in groupi\}, M)$.
(4) Apply the merge and sort algorithm to $L_1, ..., L_{5n^{1-\epsilon}} t, M$.
Generate List From Group $([a_1; \ldots ; a_m]; M)$.
(5) $L = ()$, an empty list.
(6) for $i = 1$ to $n^2 2^{\frac{2n^{\epsilon}}{\log .5n^{1-\epsilon}}}$ do:
(7) Generate $m$ random $x_j \in \{0, 1\}$
(8) Add the number $\sum_{j=1}^{m} a_j x_j$ $(mod\ M)$ to list $L$.
(9) Return $L$
```

The complexity of the algorithm weakened to $O(k.m^{2/\log k})$ time and space to preserve the linear independency and uniformity from what Wagner proposed to be $O(k.m^{1/\log k})$. By assuming $m = 2^{n^{\epsilon}}$, $\epsilon < 1$ and $k = \frac{1}{2}n^{1-\epsilon}$ the complexity equals to $O(2^{\frac{2n^{\epsilon}}{(1-\epsilon)\log n}})$.

3.2 Shallue [35]

His work inspired by Lyubashevsky [16] to give an improved algorithm solving subset sum problem in a better time and space complexity.
We recall here Shallue's subset sum problem definition here with his notations:
Definition(k-set birthday problem:)
We have sets $L_1, L_2, ..., L_k$, each set contains $\alpha m^{1/\log k}$ elements from $\mathbb{Z}/m\mathbb{Z}$ that are random and uniform, distributed independently. The k-set problem is to find $b_i \in L_i$ such that $\sum_{i=1}^{k} b_i = 0 \; mod \; m$. The technical assumptions for the problem is $\alpha > max\{1024, k\}$ and $\log m > 7(\log m)(\log k)$.
For large enough randomized problems with large constant densities they use the above definition and apply the known Wagner algorithm to solve a RMMS problem.
Depending on c and k if we choose $m = 2^{cn/k}$ with $k = \log k/\log k + 4$ in order to have density greater than $k.(1+\frac{4}{\log k})$ the RMSS problem can be solved in $O(m^{1/\log k})$.

He changed some parameters of Wagner's list merging algorithm and reached an algorithm

```
ListMerge Algorithm:
Input: Lists $L_1, L_2$ of integers in interval $[-\frac{mp^\lambda}{2}, \frac{mp^\lambda}{2})$ and parameter $p < 1$
1. Sort two lists
2. For $b \in L_1$ find $c \in L_2$ in interval $[-\frac{mp^{\lambda+1}}{2}, \frac{mp^{\lambda+1}}{2})$
3. $L_{12} \cup b + c$
4. Return $L_{12}$
```

Note that by assuming to have $|L_1| = |L_2| = \frac{\alpha}{p}$ the resources would be used in $O(\frac{\alpha}{p} \log \frac{\alpha}{p})$.
Now it is time to describe the k-set sum problem of Shallue with assuming $t = 0$

```
Input: Lists of size $\alpha/p$ integers in $\mathbb{Z}/m\mathbb{Z}$, parameters $k < n, p = m^{-1/\log k}$
1. treat list elements as in interval $[-\frac{m}{2}, \frac{m}{2})$
2. For level $\lambda = 1$ to $\log k_1$ do
3. By keeping track of partial sums apply ListMerge algorithm to each pair of the list
4. After level $\log k_1$ if we have nonempty list
5. Return $(l_1, l_2, ..., l_k)$
6. If else Return "No Solution"
```

This algorithm outputs $(l_1, l_2, ..., l_k)$ such that $\sum_{i=1}^{k} l_i = 0 \; mod \; m$.

As a reason of applying the ListMerge algorithm to the pairs of the list in k-tree method after $\log k$ steps we reach just one list of integers in the interval $[-\frac{mp^{\log k}}{2}, \frac{mp^{\log k}}{2}) = [-\frac{m}{2}, \frac{m}{2})$
The algorithm applied for $2k$ lists of size at most $\alpha/p = \alpha m^{1/\log k}$ therefore the total complexity for the problem is $O(k\alpha.m^{1/\log k})$.

For proving the correctness of the algorithm it is enough to show that there exists such a $c \in L$ from interval $[-\frac{mp^\lambda}{2}, \frac{mp^\lambda}{2})$.
The application of the algorithm is to solve RMSS problem.
The idea of algorithm is as follows:
It chooses two parameters α and k, divides all a_i's into k sets and tries to generate k list of $\alpha.m^{1/\log k}$ random subset sums of that particular partition a_i. Last level is

applying the k-set birthday algorithm to find the solution.
With $m = 2^{n^\epsilon}$, $\epsilon < 1$ and $k = \frac{1}{2}n^{1-\epsilon}$ the complexity of the random modular subset sum algorithm will be equal to $O(k\alpha.m^{1/\log k}) = O(2^{\frac{n^\epsilon}{(1-\epsilon)\log n}})$ time and space.

3.3 Flaxman-Przydatek [20]

In 2005 they proposed a new algorithm which was a completness for Blum' work on solving the Subset Sum problem (SSP). The algorithm gets the elements of the lists from $\mathbb{Z}_M$ with the inputs $a_i$is and target B. They divides the problem in two sets. First sets modular are powers of 2 and the other sets have even modular and a combination of two algorithms gives a general solution for the problem in any modular. l is a parameter set to be $\log n/2$. For any molular $M = \bar{M}.M'$ where $\bar{M} = 2^m$ and M' is odd. They use the first case to reduce the problem and solve it with the second algorithm.
For the first instance of the problem they have tuple $(a_1, ..., a_n, B, M)$ is with $M = 2^m$ and target value $B \neq 0$. They transfer the information to an equivalent instance with target value $B = 0$ and $a_{n+1} = B - M$. The algorithm for zero target instance of the problem is
Search among inputs to find the maximum matching number which contains a_{n+1}.
Two input a_i, a_j is matched if their sum $a_i + a_j$ has l significant bit equals to zero i.e. $(a_i + a_j) \equiv 0 \; mod \; 2^l$ with $a_{i_s} = a_{n+1}$. Now they have matching size of $s, ((a_{i_1}, a_{j_1}), ..., (a_{i_s}, a_{j_s}))$ that can generate smaller instance of the problem by tuple $((a_{i_1}, a_{j_1})/2^l, ...(a_{i_1}, a_{j_1})/2^l, 0, 2^{m-l}$. The acceptable solution for the problem must contain the last element. By doing this algorithm recursively even with at most $(n+1)/2$ number of inputs all solution for 2^m modular is extracted.
The second case is when M is an odd modular. This time for transforming the information to other instance they add a value of $\delta := (-B/2^t)$ where $t = \lceil \log_2 M/l \rceil$. Therefore the instance problem constructed by tuple $(a_1^{'}, ..., a_n^{'}, 0, M)$ with $a_i^{'} = a_i + \delta$. They have to be sure that the odd modular algorithm returns exactly 2^t elements by selecting from the interval $[-\frac{M-1}{2}, \frac{M-1}{2}]$ and a transformation

$$a \rightarrow \begin{cases} a & \text{if } \; a \leq M - 1/2 \\ a - M & \text{otherwise} \end{cases}$$

It finds the maximum matching for their zero target instance of the problem with a_i's and odd modular.
Combining both algorithms finds the solution for the general modular problem in $O(n^{3/2})$ time and complexity when $m = 2^{O(\log n)^2})$, n is the numbers for input integers. This algorithm is a kind of dynamic algorithm in solving the problem.

CHAPTER 4

SPARSE CASES ALGORITHMS

It is believed that the general birthday problem falls into sparse instances range when it has density less than 0.64.With this range Lagarias and Odlyzko [21] proved that when n goes to infinity almost all subset sum problems reduced shortest lattice vector problem and when $m = \Omega(2^{n^2})$ it can be solved by the lattice bases reduction in polynomial time [21].
Lattice of dimention n is a set constructed from all linear combination of integer coefficient from $b_1, ..., b_n \in \mathbf{R}^d$, set of linearly independent vectors. Finding the shortest non-zero lattice vector is the main issue of the problem which has polynomial time algorithm when n is constant and NP-hard when n is varying[34].

Schnorr extended the Wagner's generalized birthday algorithm [34] first to a small sum problem (SSP) when the summation of the elements of the list has small difference instead of zero and followed in rational numbers to benefit in shortest lattice vector problem.

Definition(SSP problem):
Let $L_1, L_2, ..., L_{2^t}$ be our 2^t lists in rational numbers drown uniformly and independently at random in interval $[-\frac{1}{2}, \frac{1}{2}] \cap \mathbb{Q}$. The problem is to find $x_i \in L_i$ such that $\sum_{i=1}^{2^t} x_i \leq \frac{1}{2}2^{-m}$.

4.1 Schnorr

Schnorr first considered the case $t = 2$ to build the lists for $(4, m)$-SSP and then extend the algorithm to a general case $(2^t, m)$-SSP.
Fisrt consider the lists $L_1, ..., L_4$. Each list has $\frac{4}{3}2^{m/3}$ random elements drawn uniformly at random from rational numbers. The new list L'_i is
$L_1^{'} := \{x_1 + x_2|, |x_1 + x_2| \leq \frac{1}{2}2^{-m/3}\}$.
$L_2^{'} := \{x_3 + x_4|, |x_3 + x_4| \leq \frac{1}{2}2^{-m/3}\}$.
$L := \{x_1^{'} + x_2^{'}|, |x_1^{'} + x_2^{'}| \leq \frac{1}{2}2^{-m}$.
Applying their mentioned lemma 3 [5] with $\alpha = 2^{-m/3}$ the average size of the lists $L_1^{'}$ and $L_2^{'}$ is $|L_1^{'}| \geq |L_1|.|L_2|.2^{-m/3}\frac{3}{4} = \frac{4}{3}2^{m/3}$.

And the average list size for L by choosing $\alpha = 2^{-2m/3}$ is $\frac{4}{3}$.
The Algorithm is based on sorting and searching to build $L_1^{'}, L_1^{'}$ and L. Based on numerical values for $x_1 \in L_1, -x_2 \in -L_2$ from the interval $[-\frac{1}{2}, \frac{1}{2}]$, they sorted the two pair and then searches for close elements $x_1, -x_2$. The searching process is done by bucket search algorithm in $O(2^{m/3})$ steps by dividing the interval $[-\frac{1}{2}, \frac{1}{2}]$ into intervals with $\frac{1}{2}2^{-m/3}$ length and then spread $x_1, -x_2$ into these new intervals. The final procedure is searching for pairs $x_1, -x_2$ which fall into the same interval. All of the above process to solve $(4, m)$-SSP problem can be done in $O(\frac{4}{3}.2^{m/3})$ average time and space.
The complexity for the generalized problem $(2^t, m)$-SSP problem is $O(2^t \frac{4}{3} 2^{\frac{m}{t+1}})$. They extend the problem from rational numbers to vectors in $\mathbb{Q}^k \cap [-\frac{1}{2}, \frac{1}{2}]^k$ and find the shortest lattice vector problem for suitable m.

4.2 Blum, Kalai, and wasserman in [7]

In machine learning topics how to learn continent functions from noisy data has a solution and it is Statistical Query (SQ) model designed by Kearns. The importance of learning the process of random noise classification algorithms in SQ models are discussed in provably noise-tolerant learning algorithms. In the representation of random noise classification the class of parity functions that can not be learned in SQ models has a polynomial time algorithms. It has applications in shortest lattice vector [38], and in cryptanalyses [1].
Blum et, al. worked on extracting y in equation $y = xA$ with probability of $n < 1/2$, where A is a random $k \times n\{0, 1\}$ and x is a k-bit message, in polynomial time. By brute force among all 2^k messages they found the correct parity function in $k = O(logn)$ time. They made the algorithm work for $k = c \log n \log\log n$ for some $c > 0$. Their algorithm also can be considered as a sub-exponential time algorithm. Their final algorithm needs $2^{O(n/\log n)}$ samples.
The algorithm depends on the first $k = c \log n \log\log n$ bits of input and the remaining $n - k$ bits equal to zero. With Gaussian elimination they can write their tested example in sum of k training samples but with their own algorithms they are able to write as a sum of $O(\log n)$ examples included desired bias. By a given set of vectors the algorithm finding the smallest subset sums to the given target vector the algorithm time complexity decreases to $O(\log^2 n)$.
The idea behind the algorithm is for per length of $\log n \log\log n$ they wrote biases vectors as the sum of $O(\log n)$ samples, sum of relatively small number of samples. With their proved lemma they labeled the samples with L_i's and denoted the samples by x_i's. x_1 . $x_1 + x_2$ are their summation vectors mod 2 and $l_1 + l_2$ the labels sum mod2. The lemma says that the correct value of $(x_1 + ... + x_s).c$ for labeled examples with c, $(x_1, l_s), ..., (x_s, l_s)$, is $(l_1 + ... + l_s)$ with high probability. This lemma means the sum of labels is distinguishable from random for the constant noise. By repeating the process the yield the correct label which is the first bits of target vector c and the rest bits with high probability.
For finding the k-length parity function consider we have subspace of $\{0, 1\}^{ab}$ contains vectors which their last i bits are zero while in k- parity function problem a denotes the blocks with b bits long in each ($k = ab$). The algorithm benefits from labeled examples from $\{0, 1\}^{ab}$ and creates i-samples contains the vectors that can be written

as a sum of $2^i,\ i = 1, ..., a-1$ original sample of size s in $O(s)$ time.

CHAPTER 5

POLYNOMIAL SYSTEM SOLVING METHODS

Considering the generalized k-list problem as a system of equations all solutions proposed to solve multivariate polynomial equations can be denoted as a solution to our problem either.
If we have a polynomial system of

$$f_1(x1, ..., x_n), ..., f_m(x_1, ..., x_n).$$
$$\in K[x_1, ..., x_n]$$

Finding the common zero $\in \mathbb{K}^n$ is the main question of PoSSo a NP-complete problem. A particular form of this problem is when we work with quadratic polynomials. It is called MQ problem and still remains in the NP-complete class.

In 1965 Bruno Buchberger [31] discovered an algorithm in multivariate polynomial rings $\mathbb{F}[X]$ over finite field $\mathbb{F}$ to compute a special bases.
The inputs of the Gröbner bases algorithm are finite set of polynomials $F \subseteq \mathbb{F}[X]$ and output a Gröbner bases $G \subseteq \mathbb{F}[X]$ in a way that they both generate the same ideal I. In spite of computing the Gröbner bases was really useful, the computation part was really inefficient such that it doubled exponentially the degree of the polynomial enters to the algorithm.

Before we go through the algorithms we need notations and definitions to understand them better.

Notation:

R is the ring, $R[x] = R[x_1, x_2, ..., x_n]$ is the polynomial ring.
$T(x_1, x_2, ..., x_n)$ or simply T is the set of all terms in these variables.
$<$ is admissible ordering on terms T.

Total degree of a term $t = (x_1^{\alpha_1}, x_2^{\alpha_2}, ..., x_n^{\alpha_n}) \in T$ is $deg(t) = \sum_{i=1}^{n} \alpha_i$.

Degree of polynomial f is
$deg(f) = max\{deg(t)|t \in T(f)\}$.

Set of monomials are defined as the set
$M(f) = \{c(\alpha_1, ..., \alpha_n)x_1^{\alpha_1}, x_2^{\alpha_2}, ..., x_n^{\alpha_n}|c(\alpha_1, ..., \alpha_n) \neq 0\}$.

The head term $HT(f)$ of f is $HT(f) = max(T(f))$ the head monomial $HM(f) = max(M(f)$ and $HC(f) =$ the coefficient of $HM(f)$ and the head coefficient $HC(f)$ of f.

A critical pair of two polynomials is Pair $(f_i, f_j) := (lcm_{ij}, t_i, f_i, t_j, f_j)$ such that $lcm(Pair(f_i, f_j)) = lcm_{ij} = HT(t_i, f_i) = HT(t_j, f_j) = lcm(HT(f_i), HT(f_j))$

f reduces to r modulo p by eliminating t means
If $f, p \in \mathbb{F}[X] \setminus \{0\}$ and $t \in T(f)$ occur with coefficient c in f. If $HT(p)|t$, then $r = f - \frac{ct}{HM(p)}.p$ satisfies $t \notin T(r)$, and we say f reduces to r and write

$$f \longrightarrow_r [t]$$

. p is the reductor, f the reductee, and r the reductum.

$spol(f_i, f_j) = \frac{lcm(LM(f),LM(g))}{bLT(f)}.f - \frac{lcm(LM(f),LM(g))}{LT(g)}.g$

The S-polynomial of g_1 and g_2 is the polynomial

$$spol(g_1, g_2) = HC(g_2)u_1g_1 - HC(g_1)u_2g_2$$

where $u_j = lcm(HT(g_1), HT(g_2))HT(g_j) \;\; forj = 1, 2.$
Degree of regularity [26] of a homogeneous ideal $T = \langle p_1, ..., p_m\rangle$ is

$$d_{reg} := min\{d \geq 0 : dim(\{p \in T | deg(p) = d\}) = \left(\begin{array}{c} n+d-1 \\ d \end{array} \right)\}.$$

Furthermore, G is a Gröbner bases iff has one of the following properties according to [40]:

1. Every $f \in \langle G\rangle$ has a unique normal form with respect to $\rightarrow_G$.
2. For every $f \in \langle G\rangle$, there is a reduction $f \rightarrow_G 0$.
3. Every nonzero $f \in \langle G\rangle$ is reducible modulo G.
4. Every nonzero $f \in \langle G\rangle$ is top-reducible modulo G.
5. For every $s \in HT(\langle G\rangle)$ there exists $t \in HT(G)witht|s.$
6. $HT(\langle G\rangle) \subseteq THT(G)$
7. The polynomials $h \in matbbF[X]$ that are in normal form with respect to $\rightarrow_G$ form a system of representatives for the partition $\{f + \langle G\rangle | f \in F[X]\}$ of $F[X]$.

Extracting the solution for multivariate equation system over finite field can be modelized as follows
A solution of the system $F = (f_1, ..., f_m)$ in L^n is a sequence $(a_1, ..., a_m)$ in L^n such that $f(a_i) = 0$ for $1 = [i, ..., m]$.
Where $X = x_1, ..., x_n$ is a sequence of indeterminate, and $f_1, ..., f_m \in F[X], L$ is the extension field of F.
It is a very useful application for solving certain type of multivariate equation systems a so-called MQ problems where the total degree of the polynomial is at most 2.

Now we can describe the Buchbereger algorithm.

5.1 Buchberger Algorithm

Buchberger Algorithm: [31]
Input: $F = f_1, ..., f_m \in \mathbb{F}[X] \setminus 0$
Output: a Gröbner basis of $< F >$

```
1: $n \leftarrow m$
2: $g_i \leftarrow f_i \ for \ 1 \leq i \leq n$
3: $P \leftarrow (gi, gj)|1 \leq i < j \leq n$
4: while $P \neq 0$ ; do
5: select some $(g_i, g_j) \in P$
6: $P \leftarrow P \ (g_i, g_j)$
7: $s \leftarrow spol(g_i, g_j)$
8: $h \leftarrow$ some normal form of s modulo$\{g_1, ..., g_n\}$
9: if $h \neq 0$ then
10: $n \leftarrow n + 1$
11: $g_n \leftarrow h$
12: $P \leftarrow P[\{(g_i, g_n)|1 \leq i \leq n\}$
13: end if
14: end while
15: return $\{g_1, ..., g_n\}$.
```

This algorithm returns Gröbner bases of the function F over a finite field. Recall that P is critical pair of the problem.
The most time consuming part of the algorithm is where polynomial s reduces to a normal form h. In many examples of theses algorithm most of the time h is zero but waste a lot of effort.
The Buchberger's algorithm can have double exponential complexity 2^{2^D} where D is the degree of the

5.2 F4 Algorithm [24]

There was attempts in history to improve the algorithm discovered by Buchberger. I choose one of the primitive algorithms described by Faugère [24] to increase the speed the efficiently computation of the Gröbner bases. He improved the Buchberger algorithm in polynomial solving methods for solving algebraic systems in various fields because even very efficient implementations are not able to compute Gröbner bases for very large problems. First they described the relation between polynomial systems and matrices in linear algebra then, described their improved algorithm for computing Gröbner bases.
They represented a polynomial as a matrix and the vectors of a matrix as polynomial by doing as follows:
Let M be a matrix $s \times m$. Jth element of ith row denoted by $M_{i,j}$. And $T_M = [t1...tm]$ be an ordered set of terms, let $(\epsilon_i)_{i=1,...,m}$ be the canonical basis of R_m, They considered a linear map $\phi(T_M) : V_{T_M} \longrightarrow R^m$ (where V_{T_M} is the sub-module of $R[x]$ generated by T_M) such that $\phi_{T_M}(t_i) = \epsilon_i$. They denoted the reciprocal function by ψ_{T_M} . The vectors of R^n interpret as polynomials by ψ_{T_M}. Then (M, T_M) is a matrix with such an interpretation. Now for constructing the set of polynomials from the matrix

$$Rows(M, T_M) := \{\psi_{T_M}(Row(M, i)) | i = 1, ..., s\} \setminus \{0\}.$$

While $Row(M, i)$ is an element of R^n is the ith row of M.

Conversely, with list of polynomials l and T_l an ordered set of terms a matrix $A_{s\times m}$ can be constructed with $s = size(l)$ and $m = size(T_l)$:

$$A_{i,j} := \text{coefficient}(l[i], T_l[j]), i = 1, ..., s, j = 1, ..., m.$$

$$M = \begin{array}{c|cccc} & t_1 & t_2 & t_2 & ... \\ f_{i_1} & * & * & * & ... \\ f_{i_2} & * & * & * & ... \\ ... & ... & ... & ... & ... \\ ... & * & * & * & ... \\ ... & * & * & * & ... \\ f_{i_s} & * & * & * & ... \end{array}$$

If we have new variables such as $Y = [Y_1, Y_2, ..., Y_m]$ and matrix $M_{s\times m}$ then, with a set of equations of $F = Rows(M, Y)$ the reduced Gröbner bases of F can be computed for lexicographical ordering variables $Y_1 > ... > Y_m$. They claim to reconstruct the echelon form of matrix M with these basis.

$$M = \begin{array}{c|ccccccc} & t_1 & t_2 & & t_k & t_{k+1} & ... & t_m \\ f_{j_1} & 1 & 0 & ... & 0 & * & ... & * \\ f_{j_2} & 0 & 1 & ... & 0 & * & ... & * \\ ... & ... & ... & ... & ... & & & \\ f_{j_l} & 0 & 0 & ... & 1 & * & ... & * \\ f_{j_{l+1}} & 0 & 0 & ... & 0 & 0 & ... & 0 \\ & & & & & 0 & & 0 \\ f_{j_m} & 0 & 0 & ... & 0 & 0 & ... & 0 \end{array}$$

.
After defining the relation between polynomials and matrices and how to convert them to each other when it is necessary they formulate the F_4 algorithm based on Buchberger's historical algorithm [31] in Gröbner basis.
They had to choose a critical pair as among the list of critical pairs and a redactor among the list of redactors while reducing the polynomial to adapt themselves with Buchberger's criteria [31]. Here is the F_4 algorithm exactly with the same levels described in [24] with a very simple description and an example to see the algorithm in real scenario.

Algorithm F_4

```
Input : $F$ a finite subset of $R[x]$
$\phi$ a function List(Pairs) $\longrightarrow$ List(Pairs) such that $\phi(l) \neq 0$; if $l \neq 0$;
Output : a finite subset of $R[x]$:
$G := F; \tilde{\mathbf{F}}^{+}$
$0 := Fandd := 0$
$P := \{Pair(f; g) | f; g \in Gwithf \neq g\}$
while $P \neq 0$, do
$d := d + 1$
$P_d := \phi(P)$
$P := P/P_d$
$L_d := Left(P_d) \cup Right(P_d)$
$_d := Reduction(L_d; G)$
for $h \in \tilde{\mathbf{F}}_d^{+}$ do
$P := P \cup \{Pair(h; g) | g \in G\}$
$G := G \cup \{h\}$
return $G$
```

Faugère had a polynomial f and a list of reductor pairs $r_1, r_2, ..., r_n$. based on these elements he constructed a matrix A as I described above.
He hired a reduction function inside the algorithm which simply computes the row echelon form Ã of the matrix A. And finally read off the reduced for of the function f from Ã [39]. Because the two steps performing in parallel with respect to linear algebra software, the algorithm is very efficient.

Complexity:
Wolf and Thomea [30] define an upper and lower bound in terms of memory usage for the algorithm in terms of memory requirement in bits by assuming that they need $\lceil \log q \rceil$ bit to store one field element.

$$MemUpperF4(q, n, d_{reg}) = \lceil \log q \rceil \begin{pmatrix} n + d_{reg} - 1 \\ d_{reg} \end{pmatrix}^2$$

and the lower bound is computed by

$$MemlowerF4(q, n, d_{reg}) = \lceil \log q \rceil \begin{pmatrix} n + d_{reg} - 1 \\ d_{reg} \end{pmatrix}^{\frac{n(n+1)}{2}}$$

However estimating the amount of operation required to perform the algorithm is not an easy task for generic cases [31] .

Example of F4 : The $F4$ algorithm is equivalent to polynomial division algorithm to reduce A. In the following example we can see how F_4 performs the same computation. We will reduce $2X^2 - Y$ by $\{X-1, Y+2\}$ using the lexicographic term order where $Y \leq X$. The steps in the polynomial division algorithm are as follows:

- $(2X^2 - Y) - 2X(X-1) = 2X - Y$
- $(2X - Y) - 2(X-1) = -Y + 2$
- $(-Y+2) + (Y+2) = 4$

The same computation performs by F_4 to reduce the below matrix . Columns correspond to monomials and rows correspond to polynomials. Note that the columns are sorted in descending order to ensure that the first non-zero entry from the left in each row corresponds to the initial term of the corresponding polynomial.

$$\begin{array}{cccc} X^2 & X & Y & 1 \\ \end{array}$$
$$\begin{pmatrix} 1 & -1 & 0 & 0 \\ 0 & 1 & 0 & -1 \\ 0 & 0 & 1 & 2 \\ 2 & 0 & -1 & 0 \end{pmatrix}$$

First row corresponds to $X(X-1) = X^2 - X$
Second row corresponds to $X - 1$
Third row corresponds to $Y + 2$
Fourth row corresponds to $2X^2 - Y$

The result is the following matrix. It is the same as polynomial division algorithm in Gaussian elimination

$$\begin{pmatrix} 1 & -1 & 0 & 0 \\ 0 & 1 & 0 & -1 \\ 0 & 0 & 1 & 2 \\ 0 & 0 & 0 & 4 \end{pmatrix}$$

Note that the reduced polynomial is on the bottom row corresponds to $(0,0,0,4)$. $(X^2, X, Y, 1) = 4$, the basic way that how F_4 works.

5.3 F5 Algorithm

Faugère followed his own work to improving the $F4$ algorithm to $F5$ [25] by using the ideas of Möller, Mora, and Traversoand. The algorithm avoids extra computation in calculation of Gröbner bases and speed-up in result.
As mentioned before the more time consuming part of these kind of algorithms is computing zero. Avoiding extra computation during choosing critical pair and reductor and having powerful criteria led to remove useless critical pair during the algorithm were challenging questions and can be thought as a motivation behind publishing $F5$ algorithm in 2002.
Hybrid version of $F5$ was able to solve many kind of symmetric and asymmetric challenges [26].
For describing the algorithm we need to get familiar with some definitions:
In a nutshell for solving the system of $\mathbb{F}[X]$-linear equation in finite field modular polynomials we have a trivial solution helps to detect S-polynomials reducing to zero called *syzygy*.
In definitions *syzygy* is an element $s \in \mathbb{F}[X]^m$ with respect to sequence $f_1, \ldots, f_m \in \mathbb{F}[X]$ if $\sum_{i=0}^{m} s_i f_i = 0$ i.e. for a pair f_i, f_j we have a trivial relation $f_i f_j - f_j f_i = 0$, extended to have *syzygy* of the form $\pi ij = f_j e_i - f_j e_j$ with e_k is the k-th bases vector in $\mathbb{F}[X]^m$.

The main part of $F5$ **algorithm** is :

```
global r // array of labeled polynomials global Rules // array of simplification rules
Input: A sequence $F = (f_1, \ldots, f_m)$ of nonzero homogeneous polynomials in $F[X]$
Output: a Gröbner basis of $\rangle F \langle$

1. $m \leftarrow |F|$
2. Rules $\leftarrow (())_{j=1}^{m}$
3. $r \leftarrow ()$
4. $r_m \leftarrow (e_m, HC(f_m)^{-1} f_m)$
5. $G \leftarrow (0)_{i=1}^{m}$
6.$G_m \leftarrow m$
7. for $i \leftarrow m-1, \ldots, 1$ do
8. $G_i \leftarrow$ Algorithm $F5(f_i, i, G)$
9. if $(\exists k \in G_i)$ poly$(r_k) = 1$ then
10. return $\{1\}$
11. end if
12. end for
13. return $\{\text{poly}(r_k) | k \in G_1\}$.
```

Simply the algorithm computed the Gröbner bases as follows:
inputs the polynomials f_i,
creates S-polynomials from polynomials in $\langle F \rangle$,
divides polynomials in $\langle F \rangle$ by a constant,
subtracts polynomials in $\langle F \rangle$ from others in$\langle F \rangle$.
Complexity of the algorithm can be considered to be have the complexity of

$$O\left(m.\begin{pmatrix} n+d_{reg}-1 \\ d_{reg} \end{pmatrix}^{w}\right)$$

where $w,\ 2 < w \leq 3$ is the exponent of the matrix reduction [37].

5.4 Linearization Method

Consider the system of polynomial equations of

$$\begin{aligned} f_1(x_1,\dots,x_n) &= 0 \\ f_2(x_1,\dots,x_n) &= 0 \\ &\vdots \\ f_m(x_1,\dots,x_n) &= 0 \end{aligned}$$

over a field $\mathbb{F}$, f_i's are polynomials in a polynomial ring $\mathbb{F}[x_1,\dots,x_n]$. Linear combination of monomials $x_1^{\alpha_1},\dots,x_n^{\alpha_n} = X^\alpha$ construct each polynomial. So we have

$$f_i = \sum_{\alpha \in N'} c_\alpha^i X^\alpha$$

where c_α^i are elements of $\mathbb{F}$ and $N^{'}$ is a subset of multi-indices of N^n. For linearization we need to construct a new linear system which monomials behaves as new variables X_α. From new linearized system the matrix A_L is obtained by

$$\begin{array}{c} \\ f_1 \\ \vdots \\ f_m \end{array} \begin{array}{c} \begin{array}{ccc} X_\alpha & \dots & X_{\alpha'} \end{array} \\ \begin{pmatrix} c_\alpha^1 & & c_{\alpha'}^1 \\ \vdots & \ddots & \vdots \\ c_\alpha^m & \cdots & c_{\alpha'}^1 \end{pmatrix} \end{array}$$

Then reduce the matrix A_L to echelon form gives the answer for the original system because it computes the basis of vector subspace generated by the polynomials $f_1,\dots,f_m$.

This is a useful technique for the systems that the number of monomial are the same as the number of linearly independent polynomials [33].

For generic cases with n variables and degree d equations we have $\begin{pmatrix} n+d \\ d \end{pmatrix}$ distinct monomilas of degree at most d. In finite fields by using field operations for example in $GF(2)$ the number of distinct monomilas of degree at most 2 computed by $N = \sum_{i=0}^{d} \begin{pmatrix} n \\ d \end{pmatrix} \approx n^d$. So that the matrix A_L has m rows and N columns . This means to solve a system directly by linearization it should be highly over-defined i.e. $m \geq N - 1$ [33].

The complexity of the method is in order N^w filed operation where w is the exponent of A_L matrix reduction.

Example of linearization method

consider the polynomial ring Q[x,y,z] of polynomials in three variables over $\mathbb{Q}[X]$ by [33] Suppose we have the equation system

$$\begin{aligned}
xyz + xz + x + 2y + z - 3 &= 0\\
2xyz - 4xy + xz + yz - x + y &= 0\\
xy + xz + yz + y - z - 2 &= 0\\
2xy + y + 7 &= 0\\
2yz + x + y + z &= 0\\
xyz + x + 2z - 1 &= 0\\
2xyz - xy - 3yz - 3y &= 0.
\end{aligned}$$

This system has seven equations and seven non-constant monomials, so it is a candidate for solution by linearization. We construct the linearization matrix:

$$\begin{array}{c}
\begin{array}{cccccccc} xyz & xy & xz & yz & x & y & z & 1 \end{array}\\
\begin{pmatrix}
1 & 0 & 1 & 0 & 1 & 2 & 1 & -3\\
2 & -4 & 1 & 1 & -1 & 1 & 0 & 0\\
0 & 1 & 1 & 1 & 0 & 1 & -1 & 2\\
0 & 2 & 0 & 0 & 0 & 1 & 0 & 7\\
0 & 0 & 0 & 2 & 1 & 1 & 1 & 0\\
1 & 0 & 0 & 0 & 1 & 0 & 2 & -1\\
2 & -1 & 0 & -3 & 0 & -3 & 0 & 0
\end{pmatrix}
\end{array}$$

Applying row reduction to this matrix gives the matrix

$$\begin{array}{c}
\begin{array}{cccccccc} xyz & xy & xz & yz & x & y & z & 1 \end{array}\\
\begin{pmatrix}
1 & 0 & 0 & 0 & 0 & 0 & 0 & 6\\
0 & 1 & 0 & 0 & 0 & 0 & 0 & 3\\
0 & 0 & 1 & 0 & 0 & 0 & 0 & -6\\
0 & 0 & 0 & 1 & 0 & 0 & 0 & 2\\
0 & 0 & 0 & 0 & 1 & 0 & 0 & -3\\
0 & 0 & 0 & 0 & 0 & 1 & 0 & 1\\
0 & 0 & 0 & 0 & 0 & 0 & 1 & -2
\end{pmatrix}
\end{array}$$

This gives us the solution $x = 3, y = -1, z = 2$, which is the solution of the original polynomial equation system.

5.5 eXtended Linearization

When there are not enough linrearly independent equations the linearization method fails so that, we need to generate them [33] and the *eXtended Linearization* had been invented [26]. Basically in XL algorithm they multiply every polynomial by all non-zero monomial X_β up to a certain degree D. The degree D called solving degree and should be large enough to generate as many equations as monomilas to apply linearization and obtain the solution of the system and called the corresponding degree $D+2$ of the polynomial saturation degree [30].

XL algorithm

```
Input: Set $F = \{f_1, \dots, f_n\} \subset F[x_1, \dots, x_n]$ of polynomials of degree $d$.
Output: Set $S \subset F[x_1, \dots, x_n]$ of uni-variate linear equations corresponding to the solution of the system $f_j = 0$.
$S := 0$;
$D := d + l$;
$i := 1$;
repeat
Generate all products $P_{(\beta,j)} = X^\beta f_j$ for $f_j \in F$ and monomials $X^\beta \in$ variables $x_1, \dots, x_n$ of degree at most $D := d$;

Consider the system consisting of equations $P_{(a,j)} = 0$ and an order on the monomials such that the monomials $x_i^k$ are the lowest. Perform Gaussian reduction on the corresponding matrix, that is solve the system by linearization;
if a uni-variate $f(xi)$ is found then
Solve the uni-variate equation to get set of solutions $A_i$ in the algebraic closure of the field $F$;
Take the (unique) $a_i \in A_i$ contained in the field $F$;
Make $S := S \cup \{x - a_i\}$;
Make $P(\alpha, j)(a_i) \in F[x_{i+1}, \dots, x_n]$, that is substitute $x_i = a_i$;
Make $i := i + 1$;
else
Make $D := D + I$;
end if
until $i = n + 1$;
return $S$
```

The algorithm construct the matrix A_L by : $\begin{array}{c} \\ f_1 \\ \vdots \\ X^\beta f_1 \\ \vdots \\ X^{\beta'} f_m \end{array} \begin{array}{c} \begin{array}{ccc} X_\alpha & \cdots & X_{\alpha'} \end{array} \\ \begin{pmatrix} c^1_\alpha & \cdots & c^1_{\alpha'} \\ \vdots & \ddots & \vdots \\ c^1_{\alpha-\beta} & \cdots & c^1_{\alpha'-\beta} \\ \vdots & \ddots & \vdots \\ c^m_{\alpha-\beta'} & \cdots & c^m_{\alpha'-\beta'} \end{pmatrix} \end{array}$

And applies the linreaization. Every answer of the extended system gives a solution for the linearized system and so that, gives the solution for the original system.

The Complexity of the algorithm for generic random cases has upper and lower bound regarding to memory consumption [26]. If the algorithm generates m.

$$MemlowerXL(q,n,D) = \lceil \log q \rceil \begin{pmatrix} n+D+1 \\ D+2 \end{pmatrix}^{\frac{n(n+1)}{2}}$$

bits for saving coefficient matrix and at most

$$MemupXL(q,n,D) = \lceil \log q \rceil \begin{pmatrix} n+D+1 \\ D+2 \end{pmatrix}^{2}$$

By Thomea and Wolf [30] the XL algorithm is more efficient in computing the solution of the problem in terms of memory consumption.

There is an example to understand the algorithm performance.
Example of XL method
We consider the equation system over $GF(23)$ given by

$$x^2 + 3xy + 17 = 0 \text{ and } y^2 + 7xy + 22 = 0.$$

The XL algorithm with $D = 4$ multiplies the two polynomials by the monomials $\{x, y, x^2, xy, y^2\}$ and the corresponding matrix A_{XL} is

$$\begin{array}{ccccccccccccccc}
x^4 & x^3y & x^2y^2 & xy^3 & x^3 & x^2y & xy^2 & x^2 & xy & x & y^4 & y^3 & y^2 & y & 1 \\
\end{array}$$
$$\left(\begin{array}{ccccccccccccccc}
0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 3 & 0 & 0 & 0 & 0 & 0 & 17 \\
0 & 0 & 0 & 0 & 1 & 3 & 0 & 0 & 0 & 17 & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & 1 & 3 & 0 & 0 & 0 & 0 & 0 & 0 & 17 & 0 \\
1 & 3 & 0 & 0 & 0 & 0 & 0 & 17 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
0 & 1 & 3 & 0 & 0 & 0 & 0 & 0 & 17 & 0 & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & 1 & 3 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 17 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 7 & 0 & 0 & 0 & 1 & 0 & 22 \\
0 & 0 & 0 & 0 & 0 & 7 & 1 & 0 & 0 & 22 & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & 7 & 0 & 0 & 0 & 0 & 1 & 0 & 22 & 0 \\
0 & 7 & 1 & 0 & 0 & 0 & 0 & 22 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & 7 & 1 & 0 & 0 & 0 & 0 & 22 & 0 & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 7 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 22 & 0 & 0
\end{array}\right)$$

By doing row operations on the matrix we obtain:

$$\begin{array}{ccccccccccccccc}
x^4 & x^3y & x^2y^2 & xy^3 & x^3 & x^2y & xy^2 & x^2 & xy & x & y^4 & y^3 & y^2 & y & 1 \\
\left(\begin{matrix} 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 15 & 0 & 0 \\
0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 4 & 0 & 2 \\
0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 11 & 0 & 10 \\
0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 2 & 0 & 12 \\
0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 17 & 0 & 2 & 0 \\
0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 16 & 0 & 1 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 10 & 0 & 13 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 16 & 0 & 1 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 10 & 0 & 13 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 7 & 0 & 20 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 8 & 0 & 8 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \end{matrix}\right)
\end{array}$$

The penultimate row gives the equation $y^4+8y^2+8 = (y-3)(y-11)(y-12)(y-20) = 0$, which gives $\{(4,3),(8,11),(15,12),(19,20)\}$ It seems that among all these algorithms XL consumes less memory.

5.6 New Approach

So many efforts were done to speed up the proposed algorithm for solving such a non-linear equations over finite fields.
There exists family of XL-like algorithms, other versions of XL algorithm to speed up and improve the efficiency of it. Some important ones like FXL, XFL, MutantXL, XLF, XL$'$, WXL, and XSL, are mentioned in [32].
FXL or fixed extended linearization is based on guessing some variables beforehand.
XLF uses field operations $(x^q - x) = 0$ in F_q, works in inhomogenous case for D gets greater than $q - 2$.
MutantXL is useful just for inhomogeneous equations.The number of monomial is smaller that $D + 2$. XL follows with $D := D + 1$ and MutantXL is an step between.

Small Linearization [27] starts with Matrix-$F5$ ($F5$ version of XL) and remove all incremental computation related to Gaussian elimination in Gröbner bases and substitute the black box algorithm that a function $f_A(x)$ provided not need to access the matrix itself by computing the matrix vector product Ax to reduce the memory usage of the $F5$ algorithm whereas speed remains high. There is a new transfer algorithm which moves the solution of the matrix back to original problem space lead to reduce the solving degree and work with smaller matrices. Moreover, it is easy to parallelize.
The complexity of the suggested *sl* algorithm is

$$O\left(\binom{n + D - 1}{D}^{w} \right)$$

In terms of memory friendly algorithms there is a comparison in [30]. By the corresponding graph it is obvious that *sl* algorithm uses very less memory even in caparison with Hybrid $F5$.

Hybrid F5 [29] is a mixed algorithm between exhaustive search and Gröbner bases algorithm. The trivial solution for solving polynomial equation(PoSSo) system which is NP-hard is searching among all variables n in $O(\#\mathbb{K}^n)$ time. The idea of Hybrid approach is fixing k variables and compute the Gröbner bases for $\#\mathbb{K}^k$ sub-systems. This choice of k is a trade off which helps to lessen the complexity.
The complexity bound for hybrid approach is :

$$O\left((m.\binom{n + d_{reg} - 1}{d_{reg}})^{w} \right).$$

Where m is the number of equations and w is the algebraic constant.

Arnau in his master's thesis [38] improved the k-list problem solving complexity b working on special instance of the problem and small range of parameters.
Their system has more equations than variables so that, highly over-determined. They use a coefficient matrix to represent the polynomial system which has k rows and columns equal to the number of monomilas. The solution for this system builds a kernel. They use their own linearization method to extract the solution from from the constructed kernel. This investigation is done by brute force inside the kernel so, they have $2^{r_{ker-1}}$ combination to search. For having the valid solution they enter only one monomial per list. They use linearization method inside the kernel to reduce the rank r_{ker} and stop when it reduces enough, below a defined upper bound say B.
First they set the degree $D = 1$ and see if the initial kernel size is below than the upper bound B. If it is then stop here and return the kernel as an answer. If it does not have they generate new polynomial to extend the matrix by multiplying a degree D monomial bu all the polynomials except the zero polynomilas and add these new polynomials to the system then echelonize the new extended matrix and check again the rank. After finishing all the monomilas of degree 1 they reduce the degree by 1 and do the procedure recursively.
They do linearization when the cost of using XL is much lower than brute force unless they use exhaustive search inside the kernel.

The complexity of the suggested algorithm when n and k are close together is $n < k.\frac{1+\log k}{\log k}$ where checking the validity of the solution vector is not greater than $O(k)$.

5.7 In Progress Work

Let $L_0, ..., L_{k-1}$ be our k-lists including elements in each list defining by $s_i^{(0)}, s_i^{(1)}, ..., s_i^{(j)}$ first, second,..., jth element of the list. We have as many s as the number of elements in each given list. Doing linear algebra we need to track of the information we spread and the s labels indicate the correct row for our purpose.
By denoting x_m^n, n-th bit of m-th list, we assign a variable for each bit of the list. So for i-th list we have bits $x_i^0, ..., x_i^n$ where n is the number of the corresponding bit in every element s_i of the list L_i.
We interpret the labels of x and s of the list L_i as a matrix B, so that we have $k.n$ matrices - one for each list, and one for each bit.
We need a set of so-called *control equations* to be added to the system, these equations are

$$x_0^0 + x_1^0 + ... + x_{k-1}^0 = 0$$
$$x_0^1 + x_1^1 + ... + x_{k-1}^1 = 0$$
$$x_0^n + x_1^n + ... + x_{k-1}^n = 0$$

control equations are all possible conditions of forcing the columns to be equal to zero. At last, we allocate 1 to the existing bits and the remaining parts fill out with zeros automatically. 0 means the value of the particular bit is false. Therefore, the difference between false bits and the bits which does not exist is distinguishable.
For small values of n and k the The algorithm is based on encoding the problem in truth tables containing s and x labels. Computing the right kernel of each bit with its corresponding s label, inserting the information into a big matrix and connecting the small kernels with control equations, the right kernel of the big matrix preserves the solution for the whole problem beside some parasitic solutions. To eliminate this parasitic solutions some extra work is required. We will discuss them in section (3).
First of all we build a truth table of s's and x's. We use specific way of representing list's elements to pick our goal.
The number of variables we get is

$$k.n + 2^l.k$$

and the number of equations

$$n + k + n.k$$

As we have more variables than equations parasitic solutions are natural [33].
We have to solve the combined system variables with control equations that give us a kernel. After echolonizing the kernel we want to reduce the size of the kernel by adding more equations to have an extended matrix in order to have smaller kernel with the correct solution inside as previous work done by Arnau but with a small variation.
We know that the rows of the kernel are bases that we can construct all the space with them. By adding one row of the kernel to the original matrix we obtain smaller kernel with linear combination of the rows which have the answer inside itself.
Investigating on finding any regularity on adding which linear combination of the kernel rows to the original problem and how it affects the solution and proofs are in progress. It seems in case of obtaining reduced kernel size we can reach the solution in less complexities than the other ones.

CHAPTER 6

CONCLUSION

We investigate some of the algorithm proposed to solve the generalized birthday problem (k-list) and its equivalent problems in integer domains like subset sum problems and shortest lattice vector problem. We discussed the general solution to the problem and their complexities. As the problem has NP-complete complexity proposing the solution for the total system seems unlikely so, each algorithm suggest the solution by bounding parameters or the size of the list due to requirements of the applications. The comparison between polynomial solving methods we discussed in terms of memory complexity is in Table 6.1. It concludes that the eXtended Lireaization uses very less memory than the other ones and among new approaches *sl* algorithm is very memory friendly algorithm in Table 6.1.

Table6.1: Polynomial System Solving Algorithms Complexity

PoSSo Algorithm	complexity
F4	$\lceil \log q \rceil \binom{n+d_{reg}-1}{d_{reg}}^2$
F5	$O\left(m.\binom{n+d_{reg}-1}{d_{reg}}^2\right)$
XL algorithm	$\lceil \log q \rceil \binom{n+D+1}{D+2}^2$
Hybrid F5	$O\left((m.(\begin{smallmatrix} n+d_{reg}-1 \\ d_{reg} \end{smallmatrix}))^2\right)$
Small linearization	$O\left((\begin{smallmatrix} n+D-1 \\ D \end{smallmatrix})^2\right)$

I set the algebraic constant $w = 2$ to make a comparison between the mentioned algorithms. The XL algorithm needs less memory than upper bound for $F4$ algorithm. Among new methods *sl* algorithm is the most memory friendly so that, the fastest algorithm. In short, we can say

Small Linearization $\geq$ XL algorithm $\geq$ HYbrid F5 $\geq$ F5 Algorithm $\geq$ F4.

Due to Table 6.2 among general algorithms proposed to solve the k-list problem for large lists Wagner's algorithm stays efficient and for smaller list sizes Minder et al.

Table6.2: Complexity of General, Dense and Sparse algorithms for solving k-list

General Algorithm	k	Time	Space	List size
Schroeppel and Shamir	4	$O(2^{n/2})$	$O(2^{n/4})$	
Minder-Sinclair	2^q	$O(2^{q+u})$	$O(2^{q+u})$	$\leq 2^u$
Coron and Joux	2^a	$O(2^a.2^{n/a+1})$	$O(2^a.2^{n/a+1})$	$2^n/a+1$
Augot	2^q	$O(r.2^q.2^{\frac{r}{a+1}})$	-	$2^{\frac{r}{a+1}}$
Bellare	k	$O(n^3+kn)$	$O(n^3+kn)$	-
Wagner	2^n	$O(k.m^{1/\log k})$	$O(k.m^{1/\log k})$	$O(2^n/(1+\log k))$

Sparse cases algorithm				
Blum	$c\log n\log\log n$	$O(\log^2 n)$	$O(\log^2 n)$	
Schnorre	2^t	$O(2^t\frac{4}{3}2^{\frac{m}{t+1}})$	$O(2^t\frac{4}{3}2^{\frac{m}{t+1}})$	-

Dense case algorithm	modular	complexity		
Lyubashevsky	$m=2^{n^\epsilon}$	$O(2^{\frac{2n^\epsilon}{(1-\epsilon)\log n}})$	$O(2^{\frac{2n^\epsilon}{(1-\epsilon)\log n}})$	
Shallue	$m=2^{n^\epsilon}$	$O(2^{\frac{n^\epsilon}{(1-\epsilon)\log n}})$	$O(2^{\frac{n^\epsilon}{(1-\epsilon)\log n}})$	$m(\log m)$
Flaxman	$m=2^{O((\log n)^2)}$	$O(n^{3/2})$		

algorithm runs in less time complexity. Blum algorihm is polynomial time for sparse cases and Shallue's solution runs in less time complexity for dense cases algorithms.

REFERENCES

[1] D. Wagner, A generalized birthday problem (extended abstract). In: Advances in Cryptology CRYPTO 2002. LNCS vol. 2442 pp. 288-303. Springer, (2002)

[2] D.E. Knuth, The art of computer programming, vol3, Addison-Wesley, 1973.

[3] L. Minder, A. Sinclair, The extended k-tree algorithm, Proceeding SODA '09 Proceedings of the twentieth Annual ACM-SIAM Symposium on Discrete Algorithms Pages 586-595, 2009

[4] D.Boneh, A. Joux, PQ. Nquyen, Why Textbook ElGamal and RSA Encryption are Insecure, ASIACRYPT 2000, LNCS 1976, Springer-Verlag, pp.30-44, 2000. 2000

[5] D. Augot, M. Finiasz and Nicolas Sendrier. A family of fast syndrome based cryptographic hash functions. Proceedings of Mycrypt 2005, LNCS 3715, Springer-Verlag, 2005.

[6] A. Joux, R. Lercier, Chinese and Match, an alternative to Atkin's Match and Sort method used in the SEA algorithm, Math. Comp., 70(234) : $827-836$, AMS, 2001.

[7] A.Blum, A.Kalai, H. Wasserman, Noise-Tolerant Learning, the Parity Problem, and the Statistical Query Model, STOC 2000, ACM Press, 2000.

[8] ,P. Camion, J Patarin,The Knapsack Hash Function proposed at Crypto'89 can be broken Advances in Cryptology - EUROCRYPT '91, Workshop on the Theory and Application of of Cryptographic Techniques, Brighton, UK, April 8-11, , Proceedings, 1991

[9] R. Schroeppel, A. Shamir $AT = O(2n = 2)$ and $S = O(2n = 4)$ Algorithm for certain NP-complete problems SIAM J. Comput, 10(3):456-464, 1981.

[10] P. Chose, A. Joux and M. Mitton Fast Correlation Attacks : an Algorithmic Point of View. EUROCRYPT 2002, LNCS 2332, Springer-Verlag, 2002.

[11] M. Bellare and D. Micciancio A New Paradigm for Collision-free Hashing: Incrementality at Reduced Cost EUROCRYPT 97, LNCS 1233, Springer-Verlag, 1997.

[12] Mark Stamp,Richard-M, Breaking Ciphers in the Real World-Wiley-Low-Applied-Cryptanalysis (2007)

[13] https://class.coursera.org/crypto-2012-002/class/index

[14] J.Kelsey, T.KohnoHerding Hash Functions and Nostradamus attack, EUROCRYPT'06 Proceedings of the 24th annual international conference on The Theory and Applications of Cryptographic Techniques Pages 183-200,2006

[15] V. Klima, Finding MD5 collisions on a notebook PC using multi-message modifications at www.eprint.iacr.org.2005.102.pdf

[16] V. Lyubashevsky. On random high density subset sums. Proceedings of APPROX-RANDOM 2005, LNCS 3624, pages 378 389, Springer-Verlag, 2005.

[17] N. Courtois. Higher order correlation attacks, XL algorithm and cryptanalysis of Toyocrypt. In ICISC, volume 2587 of Lecture Notes in Computer Science, pages 182-199. Pil Joong Lee and Chae Hoon Lim, editors, Springer, 2002.

[18] J. Sebastien Coron and A.Joux, Cryptanalysis of a provably secure cryptographic hash function. Cryptology ePrint Archive Report 2004/013, 2004. (http://eprint.iacr.org/2004/013).

[19] M. Chaimovich, New algorithm for dense subset-sum problem. Astèrisque 258 pages 363-373. 1999.

[20] Flaxman, A. Przydatek, B. Solving medium-density subset sum problems in expected polynomial time. In: STACS . LNCS vol. 3404, pp. 305-314. Springer, 2005.

[21] Coster, M.J., Joux, A., LaMacchia, B.A., Odlyzko, A.M., Schnorr, C.P., Stern, J. Improved low density subset sum algorithms. Comput. Complexity 2(2) pages 111-128, 1992.

[22] J.-C. Faugère and A. Joux. Algebraic cryptanalysis of Hidden Field Equations (HFE) using Gröbner bases. In Advances in Cryptology , CRYPTO 2003, volume 2729 of Lecture Notes in Computer Science, pages 44-60. Dan Boneh, editor, Springer, 2003.

[23] J.-C. Faugère, A. Otmani, L. Perret, and J.-P. Tillich. Algebraic cryptanalysis of McEliece variants with compact keys. In EUROCRYPT, volume 6110 of Lecture Notes in Computer Science, pages 279-298. Henri Gilbert, editor, Springer, 2010. ISBN 978-3-642-13189-9.

[24] J.-C. Faugère. A new efficient algorithm for computing Gröbner bases (F4). Journal of Pure and Applied Algebra National Institute of Standards and Technology Applied Algebra, 139:61-88, June 1999.

[25] J.-C. Faugère. A new efficient algorithm for computing Gróbner bases without reduction to zero (F5). In International Symposium on Symbolic and Algebraic Computation , ISSAC 2002, pages 75-83. ACM Press, July 2002.

[26] N. T. Courtois, A. Klimov, J. Patarin, and A. Shamir. Efficient algorithms for solving over-determined systems of multivariate polynomial equations. In Advances in Cryptology . EUROCRYPT 2000, volume 1807 of Lecture Notes in Computer Science, pages 392-407. Bart Preneel, editor, Springer, 2000.

[27] E. Thomae, C. Wolf. Solving Underdetermined Systems of Multivariate Quadratic Equations revisited. In Practice and Theory in Public Key Cryptography (PKC 2012), Lecture Notes in Computer Science. Springer-Verlag, 2012.

[28] .C. Wolf and E.Thomea, Small linearization:Memory Friendly Solving of Non-Linear Equations over Finite Fields at http://eprint.iacr.org/2011/669, 2011.

[29] L. Bettale, J-C Faugère, and L. Perret. Hybrid approach: a tool for multivariate cryptography. In Proceedings of the ECRYPT Workshop on Tools for Cryptanalysis 2010. ECRYPT II, 2010.

[30] E. Thomae, C. Wolf, Solving Systems of Multivariate Quadratic Equations over Finite Fields or: From Relinearization to MutantXL, http://eprint.iacr.org/cgi-bin/print.pl, 2012

[31] B. Buchberger. A Theoretical Basis for the Reduction of Polynomials to Canonical Forms. ACM SIGSAM Bull., 10(3):19-29, 1976.

[32] W. S. A. Mohamed. Improvements for the XL Algorithm with Applications to Algebraic Cryptanalysis. PhD thesis, TU Darmstadt, Jun 2011. http://tuprints.ulb.tu-darmstadt.de/2621/.

[33] C. Cid, S. Murphy, M. Robshaw, Matt J. B. RobshawAlgebraic Aspects of the Advanced Encryption Standard, Library of Congress Control Number: 2006929676, 2006.

[34] C. Peter Schnorr, Lattice reductio by random samplying and birthday methods, In Proc. STACS 2003, Eds. H. Alt and M. Habib, LNCS 2607 pages = 145-156, 2003

[35] A. Shallue. An improved multi-set algorithm for the dense subset sum problem. Proceedings of Algorithmic Number Theory Symposium, ANTS VIII, LNCS 5011, pages 416-429, Springer-Verlag, 2008

[36] N. T. Courtois and J. Patarin. About the XL algorithm over GF(2). In CT-RSA'03: Proceedings of the 2003 RSA conference on The cryptographers' track, pages 141-157, Berlin, Heidelberg, Springer-Verlag, 2003. .

[37] L. Bettale1, J. C. Faug'ere, L. Perret, Algebraic Cryptanalyses: Solving multivariate polynomial systems over finite fields,

[38] A. Sipasseuth, A study of the k-list problem using multivariate polynomials and linear algebra , 2012.

[39] B.Hammersholt Roune, The F4 Algorithm Speeding Up Grb̈bner Basis Copmutations Using Linear Algebra,

[40] T. Stegers. Faugère's, F5 Algorithm Revisited. Thesis For The Degree Of Diplom-Mathematiker, Technische Universität Darmstadt, 2005.