## Eindhoven University of Technology

## BACHELOR

## Understanding LEDA crypt and the weak keys attack

Portegijs, Iris

Award date:
2021

Link to publication

## Disclaimer

This document contains a student thesis (bachelor's or master's), as authored by a student at Eindhoven University of Technology. Student theses are made available in the TU/e repository upon obtaining the required degree. The grade received is not published on the document as presented in the repository. The required complexity or quality of research of student theses may vary by program, and the required minimum study period may vary in duration.

## General rights

Copyright and moral rights for the publications made accessible in the public portal are retained by the authors and/or other copyright owners and it is a condition of accessing publications that users recognise and abide by the legal requirements associated with these rights.

- Users may download and print one copy of any publication from the public portal for the purpose of private study or research.
- You may not further distribute the material or use it for any profit-making activity or commercial gain


# TU/e $=$ 

Department of Mathematics and Computer Science

# Understanding LEDAcrypt and the weak keys attack 

Bachelor Final Project

Iris Portegijs

Supervisor and second corrector:
Prof. Dr. Tanja Lange
Dr. Alberto Ravagnani

Eindhoven, February 2021

## Summary

This thesis investigates the post-quantum cryptosystem LEDAcrypt and the weak keys attack against it. First some prior knowledge about cryptology, coding theory, circulant matrices and ISD algorithms is explained. The working of LEDAcrypt is described, followed by the explanation about the working of the weak keys attack. To find which keys are the weak keys, a program is developed to find data about the success rate of the attack for certain types of keys. The data is analyzed and it shows that keys with many consecutive non-zero elements or non-zero elements in regular positions can be broken by the weak key attack.

## Contents

Contents ..... 3
1 Introduction ..... 4
2 Prior knowledge ..... 6
2.1 (Quasi)-Cyclic matrices ..... 6
2.2 Support ..... 7
2.3 Cryptology ..... 7
2.4 Coding theory ..... 8
2.5 Code-based cryptology ..... 9
2.6 ISD algorithms ..... 9
2.6.1 Brute force attack ..... 9
2.6.2 Prange ..... 9
2.6.3 Lee-Brickell ..... 10
3 LEDAcrypt ..... 11
3.1 LEDAcrypt KEM ..... 12
4 Weak keys attack ..... 14
4.1 Sparse polynomials ..... 14
4.2 Properties of the support under reduction and lift ..... 15
4.3 Choosing $L^{\prime}$ ..... 16
5 Methodology ..... 17
5.1 Measuring the effectiveness of the attack ..... 17
5.2 Strategies ..... 20
5.3 Structure in the data ..... 20
5.4 Applying the methology to a larger case ..... 21
6 Results ..... 22
6.1 Alice ..... 22
6.2 Charlie ..... 22
6.3 Simon ..... 26
6.4 Ron ..... 26
6.5 Comparing the strategies ..... 29
7 Conclusion ..... 34
References ..... 35
A Checklists for the attack ..... 37
A. 1 Checklist for $p=11$ ..... 37
A. 2 Checklist for $p=13$ ..... 37

## Chapter 1

## Introduction

The topic of this thesis, LEDAcrypt, belongs to the area of applied discrete mathematics, more precisely into the areas of cryptology and coding theory.

Coding theory protects data from errors. When sending information, it may happen that some bits are changed. This means that the original message may not be equal to the received information. Although it is not possible to prevent these errors, there exist ways to detect and correct them. This often happens by adding redundancy to the message, so the receiver can check if structure in the received word is still correct or not. If this is not the case, they know that an error occurred, and can find where this error happened.

Cryptology is the combination of cryptography, the discipline that designs cryptosystems, and cryptanalysis, the discipline that tries to break those cryptosystems. Cryptosystems are used to protect information against attackers. They ensure that no-one can read a message except for the intended receiver, no-one can modify the information and the receiver can check if the message is really from the sender and not from some attacker. Thus a cryptosystem ensures confidentiality, integrity and authenticity. In cryptology, the sender, receiver and attacker are often given names, to make talking about them easier. In this thesis, they will be called $A$ (or Alice), $B$ (Bob) and $E($ Eve), respectively. Safe communication between Alice and Bob is often achieved by using keys to encrypt and decrypt information.

Many cryptosystems that are used on the internet today use two large primes as keys. When these primes are multiplied, computers cannot recover those two primes from the number that was generated in a fast way, and this makes these cryptosystems safe. Quantum computers however, can do prime factorization, which means that those cryptosystems can be broken. When the reality of quantum computers being used came into sight, cryptologists around the world were invited to help with developing new cryptosystems. NIST (The National Insitute of Standards and Technology of the United States) announced a competition to develop and standardize new cryptosystems of all sorts. LEDAcrypt is such a new cryptosystem.

LEDAcrypt stands for Low-dEnsity parity-check coDe-bAsed cryptographic systems. It uses QC-LDPC codes as described in chapter 2. It was developed for the NIST post-quantum contest in 2017 by Marco Baldi, Alessandro Barenghi, Franco Chiaraluce, Gerardo Pelosi and Paolo Santini. LEDAcrypt was one of the eighteen submissions in the category Public-key Encryption and Keyestablishment Algorithms in the second round of NIST.[10] Unfortunately, it did not make it to the third round. LEDAcrypt was designed as a safe option to send a key to another person, after which this key can be used in a symmetric-key cryptosystem. LEDAcrypt's Key Encapsulation Module (LEDAcrypt KEM) was designed so that the key cannot be read by an attacker. [4]

Unfortunately, an attack was developed against LEDAcrypt in 2020 [2]. This attack found the structure in certain weak keys that could be used to recover those keys much faster than was expected. To find out how the attack works exactly, this attack was simulated in this thesis for very small cases, with the goal to find out which keys are weak keys.

In chapter 2, mathematical topics that need to be known to understand LEDAcrypt will be described. Then LEDAcrypt will be explained in chapter 3, followed by an explanation of the
weak keys attack in chapter 4. Then the methodology of the research is introduced in chapter 5 , after which the results are analysed in chapter 6 . Both of these chapters are the author's own work.

## Chapter 2

## Prior knowledge

It is expected that the reader of this thesis knows some basic linear algebra, like matrix multiplication, inverses of a matrix and solving linear equations. It is also expected that the reader has some knowledge about fields and rings. Other topics that the reader might not have heard of or topics that need further attention are explained in this chapter.

## 2.1 (Quasi)-Cyclic matrices

Circulant matrices are matrices where shifting the first row gives the second row, shifting the second row gives the third, etc. A circulant matrix looks like this:

$$
\left(\begin{array}{ccccc}
a_{0} & a_{1} & a_{2} & \cdots & a_{p-1} \\
a_{p-1} & a_{0} & a_{1} & \cdots & a_{p-2} \\
a_{p-2} & a_{p-1} & a_{0} & \cdots & a_{p-3} \\
\vdots & \vdots & \vdots & \ddots & \vdots \\
a_{1} & a_{2} & a_{3} & \cdots & a_{0}
\end{array}\right)
$$

Under the standard addition and multiplication operations of modulo-2 matrices, the set of $p \times p$ binary circulant matrices forms an algebraic ring, where the zero element it the all-zero matrix and the identity element is the identity matrix. Addition of two binary circulant matrices $A$ and $B$ forms indeed a new circulant binary matrix, and associativity and commutativity regarding addition are inherited from the general matrix addition. Multiplication is less obvious however. We look at the multiplication of $p \times p$ circulant matrices $A$ and $B$ to see if the result $C$ is circulant as well.

$$
\left(\begin{array}{cccc}
a_{0} & a_{1} & \cdots & a_{p-1} \\
a_{p-1} & a_{0} & \cdots & a_{p-2} \\
\vdots & \vdots & \ddots & \vdots \\
a_{1} & a_{2} & \cdots & a_{0}
\end{array}\right)\left(\begin{array}{cccc}
b_{0} & b_{1} & \cdots & b_{p-1} \\
b_{p-1} & b_{0} & \cdots & b_{p-2} \\
\vdots & \vdots & \ddots & \vdots \\
b_{1} & b_{2} & \cdots & b_{0}
\end{array}\right)=\left(\begin{array}{cccc}
c_{0,0} & c_{0,1} & \cdots & c_{0, p-1} \\
c_{1,0} & c_{1,1} & \cdots & c_{1, p-1} \\
\vdots & \vdots & \ddots & \vdots \\
c_{p-1,0} & c_{p-1,1} & \cdots & c_{p-1, p-1}
\end{array}\right)
$$

where $c_{i, j}=a_{p-i} b_{j}+a_{p-i+1} b_{j-1}+\cdots+a_{p-i-1} b_{j+2}=\sum_{k=0}^{p-1} a_{j-i-k} b_{k}$. Note that all the indices are modulo $p$, the index for $a$ in this equation equals $p+j-i-k$ for $k>j-i$. Note that the value for $c_{i j}$ depends only on $j-i$ and thus $c_{(i+1)(j+1)}=c_{i j}$ for all $0 \leq i, j<p$, again with indices taken modulo $p$ thus $C$ is also circulant. Because the multiplication works as well, the binary circulant matrices form a ring. This ring is isomorphic to the polynomial ring $\mathbb{F}_{2}[x] /\left\langle x^{p}+1\right\rangle$ where the zero element is the zero polynomial and the identity element is the constant polynomial, with standard addition and multiplication modulo $x^{p}+1$ and with all coefficient in $\mathbb{F}_{2}$. More precisely,
the matrices can be mapped to a polynomial with the following map:

$$
A=\left(\begin{array}{cccc}
a_{0} & a_{1} & \cdots & a_{p-1}  \tag{2.1}\\
a_{p-1} & a_{0} & \cdots & a_{p-2} \\
\vdots & \vdots & \ddots & \vdots \\
a_{1} & a_{2} & \cdots & a_{0}
\end{array}\right) \leftrightarrow a_{0} x^{0}+a_{1} x^{1}+\ldots+a_{p} x^{p-1}=a(x)
$$

This is useful for the computation of keys and checking invertibility, since it is cheaper to perform manipulations on polynomials than it is to perform manipulations on matrices in a program.

Quasi-cyclic codes are defined in section 2.4. These codes are defined via matrices that consist of blocks of smaller circulant matrices. Because of their use for quasi-cyclic codes, we will call those matrices quasi-cyclic matrices. This is an example of a $(p+1) \times 2(p+1)$ quasi-cyclic matrix $C$ consisting of two cyclic blocks, $A$ and $B$.

$$
\begin{aligned}
C & =\left(\begin{array}{cc}
A & B
\end{array}\right) \\
& =\left(\begin{array}{cccccccc}
a_{0} & a_{1} & \cdots & a_{p} & b_{0} & b_{1} & \cdots & b_{p} \\
a_{p} & a_{0} & \cdots & a_{p-1} & b_{p} & b_{0} & \cdots & b_{p-1} \\
\vdots & \vdots & \ddots & \vdots & \vdots & \vdots & \ddots & \vdots \\
a_{1} & a_{2} & \cdots & a_{0} & b_{1} & b_{2} & \cdots & b_{0}
\end{array}\right)
\end{aligned}
$$

It is important to note that all information of a circulant or a quasi-cyclic matrix is stored in any single row. The Hamming weight $w(r)$ of a row $r$ is the number of non-zero entries. In quasi-cyclic matrices, each row has the same weight, so we refer to the row weight of $M$ as the weight of the matrix $w(M)$.

### 2.2 Support

The support of a matrix $A$ is defined as $\operatorname{supp}(A)=\left\{(i, j) \mid a_{i, j} \neq 0\right\}$. In this thesis, only circulant matrices will be considered, for which all information is captured by the first row and thus the support will be defined as $\operatorname{supp}(A)=\left\{i \mid a_{i} \neq 0\right\}$ as opposed to the standard definition. In this definition the position is more important than the value of the $a_{i}$, because for binary matrices this value will only be 0 or 1 , and in the support only the $a_{i}=1$ will be included. Note that for this definition, the support contains all the information about the circulant matrix. For the attack matrices defined over the integers (see chapter 4 for details), the values of the $a_{i}$ do not matter either, only the positions of the non-zero entries are interesting. By the isomorphism in equation 2.1, the support of a polynomial $a(x)=\sum_{i} a_{i} x^{i}$ can be defined as $\operatorname{supp}(a)=\left\{i \mid a_{i} \neq 0\right\}$. Note that here the position is again more important than the term itself. The size of the support, i.e. the number of elements in the support will be noted as $|\operatorname{supp}(A)|$ or $|\operatorname{supp}(a)|$.

### 2.3 Cryptology

In this section the basic topics of cryptology needed for the understanding of this thesis are described. This and more information can be found in [16].

There are two kinds of cryptosystems. Symmetric cryptography uses a secret key that is known to both the Alice and Bob, a so called 'shared secret'. In figure 2.1 Alice and Bob use the shared key $k$ to encrypt the message $m$ and decrypt the ciphertext $c$. Eve can intercept $c$, but as long as Eve does not know the shared secret, she cannot recover $m$ and therefore the systems is safe.

The other type of cryptosystems are the public key cryptosystems. In these cryptosystems, Alice and Bob both have a public key $\left(p k_{A}\right.$ and $p k_{B}$, respectively), that is available for everyone, and a secret key $\left(s k_{A}\right.$ and $s k_{B}$, respectively), that nobody else knows. Alice uses the public key


Figure 2.1: Symmetric Cryptosystem


Figure 2.2: Public Key Cryptosystem
of Bob to encrypt $m$, and Bob uses his own secret key to decrypt $c$. Eve can see the ciphertext again, but since she only knows $p k_{B}$ and does not know $s k_{B}$, she cannot read $m$. See figure 2.2

Well-known public key cryptosystems are Diffie-Hellman key exchange [5] and RSA[13]. Peter W. Shor found an attack using a quantum computer that breaks these cryptosystems [14]. To prepare for the post-quantom world, cryptographers are developing various types of cryptosystems that can withstand attacks with quantum computers. These types can use various mathematical structures, code-based cryptosystems are among these types [10].

### 2.4 Coding theory

This section covers the basics of coding theory and explains linear codes and Quasi-Cyclic LowDensity Parity-Check codes (QC-LDPC codes), which are used in LEDAcrypt. Information on coding theory can be found in [15] and [12].

In this thesis, only linear codes over $\mathbb{F}_{2}$ are considered. These are subspaces of the vector space $\mathbb{F}_{2}^{n}$. Such a code $C$ is denoted as $C(n, k)$ if its dimension is $k$. When two parties communicate they use $C \subset \mathbb{F}_{2}^{n}$ as the set of acceptable code words. If $c \in C$ gets sent but $x \neq c$ is received, the receiver needs to correct $x$ to $c$. This is called decoding.

An important definition in coding theory is the Hamming distance. The Hamming distance between two code words $x=\left(x_{1}, x_{2}, \ldots, x_{n}\right)$ and $y=\left(y_{1}, y_{2}, \ldots, y_{n}\right)$ is defined as follows: $d(x, y)=$ $\left|\left\{0 \leq i \leq n \mid x_{i} \neq y_{i}\right\}\right|$. The minimum distance is the minimal Hamming weight between any two code words, i.e. $\min \{d(x, y) \mid x, y \in C, x \neq y\}$. When the minimum distance is 1 , e.g. when $C=\mathbb{F}_{2}^{n}$ and all vectors of length $n$ are correct code words, an error moves from one valid code word to another and the receiver will not notice that there was an error and recover the wrong input. Error-correcting codes can detect and correct errors because they add redundancy to the message, i.e. they add bits such that $n>k$. These extra bits are given a certain structure, depending on the message. Because of this, the minimum distance is increased, and a check is put into place, that can be used to see if an error occurred, because then the structure would not be correct anymore. For linear codes, this check can be done using a parity check matrix $P$. This matrix is given, before sending someone information you agreed on how to encode/decode. Any correct code word $c$ satisfies $P c=0^{1}$, thus $C$ is defined as the kernel space of $P$. If an $x$ is found such that this equation does not hold, i.e. $P x=s \neq 0$, an error has occurred. Now we look at what happens when we check $x$, i.e. $P x=P(c+e)=P c+P e=0+P e$. We call this the syndrome $s=P e$. The parity check matrix $P$ gives all the information about the code, and can be used to describe it.

Good codes permit efficient decoding, meaning that for a limited number of errors these errors can be quickly found and corrected. In this thesis we do not consider decoding of efficient codes, but consider the general case in the section 2.6.

Low-Density Parity-Check codes (LDPC codes) are a special type of linear block codes, that have a sparse parity-check matrix $P$, meaning that $P$ has only very few non-zero elements in each

[^0]row or column. As the name suggests, a Quasi-Cyclic Low-Density Parity-Check code is an LDPC code with a quasi-cyclic parity check matrix. Such a code is used in LEDAcrypt.

### 2.5 Code-based cryptology

Code-based cryptology was introduced by McEliece in 1978 [8]. A more general version of his cryptosystem, Niederreiter's cryptosystem, will be described in chapter 3.

Code-base cryptology uses codes and hides the structure of the code by modifying the paritycheck matrix. The public key is a scrambled version $K$ of the original parity-check matrix $P$; the secret key is $P$ and the knowledge of how $K$ relates to $P$.

In the setting of code-based cryptology plaintexts are vectors $e$ of weight $t$. It is possible to encode messages into fixed-weight vectors but in chapter 3 we will see in LEDAkem that this encoding step is not necessary in modern cryptosystems. The ciphertext in Niederreiter's scheme is the syndrome $s=K e$ of $e$. Decryption means recovering $e$ from $s$, which should be hard given $K$ but easy given $P$, so $P$ describes a code with good error-correction algorithms. The goal is again to find $e$ such that $t$ is minimal.

### 2.6 ISD algorithms

This section is about ISD algorithms. The information in this section follows [6]. ISD stands for Information Set Decoding. This group of algorithms can be used to find close code words in a given code. I.e. for a received word $x \in \mathbb{F}_{2}^{n}$, it tries to find code word $c$, where $x=c+e$ and $w(e)=t$ is minimal (so $t$ errors occurred when sending $c$ ). With known $x$ and $s$, the goal is to find the $c$ closest to $x$, i.e. the $c$ for which $d(c, x)$ is minimal, since this is probably the right word that was sent. The distance is minimal for a minimal number of errors, i.e. for an $e$ with minimal weight. Thus, the ISD algorithm tries to find the $e$ with minimal weight from the known $s$. We will look at three ISD algorithms in this section: Brute force attack, Prange and Lee-Brickell. We will describe them from the point of view of an attacker on the Niederreiter scheme, thus use $K$ for the parity-check matrix.

### 2.6.1 Brute force attack

A brute force attack tries all possible outcomes, until the right one is found. In this setting, the algorithm picks $t$ random columns from $K$, adds these columns and then compares them to $s$. If it is equal then $e$ is found, if not, the steps are repeated with another combination of columns from $K$. The cost of this algorithm is $\binom{n}{t}$, the number of possible sums of $t$ columns in a matrix with $n$ columns. This takes a lot of computations and is thus not ideal, especially not for large keys.

### 2.6.2 Prange

In 1962, Prange initiated a more efficient ISD algorithm [11]. Instead of performing a brute force attack on the entire matrix $K$, some columns are not included in the brute force attack. First, the algorithm modifies $K$, i.e. $K^{\prime}=U K W$ where $W$ is a randomly chosen permutation matrix, and $U$ (which is invertible) is chosen such that $K^{\prime}$ has the form $K^{\prime}=\left(X \mid I_{n-k}\right)$. The ciphertext $s$ is updated to $U s$. The attack will succeed, if the permutation pushed all the error positions in $s$ into the last $n-k$ positions. This can be seen by inspecting the weight of $s^{\prime}$. If $w\left(s^{\prime}\right)=t$ then $e^{\prime}=\left(0 \ldots 0 \mid s^{\prime}\right)$ satisfies $s^{\prime}=K^{\prime} e^{\prime}$ and applying $W^{-1}$ to $e^{\prime}$ gives $e$. The costs of this algorithm is
$\frac{\binom{n}{t}}{\binom{n-k}{t}}$, which is a great improvement from the brute force attack.

### 2.6.3 Lee-Brickell

Lee and Brickell presented an even more efficient ISD in 1988 [7]. Similar to Prange's algorithm, $K$ is permuted and brought into the same systematic form. A difference between the two algorithms is that the Lee-Brickell algorithm does not require all the errors to be in $I_{n-k}$. Instead, it allows $b$ errors in $X$. The algorithm picks $b$ columns from $X$ and calculates their sum $x_{b}$. Then it checks the weight of $s+x_{b}$. We know that $w(e)=t$ and that Prange looks for an $s^{\prime}$ with $w\left(s^{\prime}\right)=t$. Permitting errors in the first $k$ positions means that the weight of $s^{\prime}$ does not instantly reveal success. If the $b$ positions in $X$ were indeed error positions in the permuted $e$ then $s^{\prime}+x_{b}$ is the syndrome of $e^{\prime}+f$, where $f$ has a 1 in exactly the $b$ positions chosen to get $x_{b}$, thus if $w\left(s^{\prime}+x_{b}\right)=t-b$ then $e^{\prime}=f+\left(0 \mid s^{\prime}+x_{b}\right)$.

## Chapter 3

## LEDAcrypt

A defining feature of LEDAcrypt is the use of quasi-cyclic matrices. Interesting about this is that all information in such a matrix can be stored in a single row instead of the whole matrix. Algorithm 1 shows how to generate a circulant matrix from a list $l$. This is later used in the generation of the key.

```
Algorithm 1: GenerateCirculantMatrix
    input : length of rows/columns of the matrix: \(p\),
            list with non-zero and zero elements of length \(p: l\)
    output: \(p \times p\) circulant matrix with top row \(l: M\)
    \(M=\operatorname{Matrix}()()\)
    for \(i\) in range ( \(p\) ) do
        for \(j\) in range \((p)\) do
            \(M(i)((j+i) \bmod p) \leftarrow l(j)\)
        end
    end
    // return \(M\)
```

The most important variables of the keys are the number of cyclic blocks in the keys, $n_{0}$, and the dimension of those blocks $p \times p$, where $p$ is chosen such that 2 is the generator of $\mathbb{F}_{p}^{*}$ and $d_{v}$ (odd number), the Hamming weight of the matrix $H$, where $H$ is the parity check matrix of a Low-Density Parity-Check code as described in chapter 2. $Q$ acts as the generator matrix of that code.

The LEDAcrypt key generating algorithm uses the key generating algorithm of Niederreiter [9], a variant of the key generating algorithm of McEliece [8]. This algorithm is described in algorithm 2. In this algorithm, the sparse binary matrix $Q$ distributes its weight over its cyclic blocks such that $Q$ has weight $d_{v}$, i.e. $w(Q)=\left(\begin{array}{ll}m_{0} & m_{1} \\ m_{1} & m_{0}\end{array}\right)$, where $m_{0}+m_{1}=d_{v}$. In the following chapters, we choose $m_{0}=d_{v}-m$ and $m_{1}=m$.

The reason that $p$ must be chosen such that 2 is the generator of $\mathbb{F}_{p}^{*}$, is that this, in combination with an odd weight, ensures that a matrix does not have repeated entries, thus guaranteeing that the matrix is non-singular. Therefore $H_{0}$ and $H_{1}$ are non-singular. For the cyclic blocks of $Q$ however, the weight might not be odd and this condition does not hold. Because the total weight of $Q$ is odd however, this condition of $p$ guarantees that the quasi-cyclic matrix is non-singular (by theorem 1.1.3 and theorem 1.1.4 in [4]). So both $H$ and $Q$ are non-singular as required for the decoding of LEDAcrypt.

```
Algorithm 2: Niederreiter Key Generation
    input : number of cyclic blocks in key: \(n_{0}\),
                length of blocks: \(p\),
                weight of blocks of \(H: d_{v}\),
                weight of blocks of \(Q: m\) and \(d_{v}-m\)
    output: public key: \(M\),
                secret key \((H, Q)\)
    // All matrices are objects over \(\mathbb{F}_{2}\)
    // Generating matrix H
    \(h_{0} \leftarrow\) GenerateRandomList \(\left(p, d_{v}\right) / *\) Skip this line if you have a fixed \(h_{0} * /\)
    \(H_{0} \leftarrow\) GenerateCirculantMatrix \(\left(p, h_{0}\right)\)
    \(h_{1} \leftarrow\) GenerateRandomList \(\left(p, d_{v}\right)\) /* Skip this line if you have a fixed \(h_{1} * /\)
    \(H_{1} \leftarrow\) GenerateCirculantMatrix \(\left(p, h_{1}\right)\)
    \(H \leftarrow\left(H_{0} H_{1}\right)\)
    // Generating matrix Q
    \(q_{00} \leftarrow\) GenerateRandomList \(\left(p, d_{v}-m\right) / *\) Skip this line if you have a fixed \(q_{00}\) */
    \(Q 00 \leftarrow\) GenerateCirculantMatrix \(\left(p, q_{00}\right)\)
    \(q_{01} \leftarrow\) GenerateRandomList \((p, m) / *\) Skip this line if you have a fixed \(q_{01}\) */
    \(Q 01 \leftarrow\) GenerateCirculantMatrix \(\left(p, q_{01}\right)\)
    \(q_{10} \leftarrow\) GenerateRandomList \((p, m) / *\) Skip this line if you have a fixed \(q_{10} * /\)
    \(Q 10 \leftarrow\) GenerateCirculantMatrix \(\left(p, q_{10}\right)\)
    \(q_{11} \leftarrow\) GenerateRandomList \(\left(p, d_{v}-m\right) / *\) Skip this line if you have a fixed \(q_{11}\) */
    \(Q 11 \leftarrow\) GenerateCirculantMatrix \(\left(p, q_{11}\right)\)
    \(Q \leftarrow\left(\begin{array}{cc}Q 00 & Q 01 \\ Q 10 & Q 11\end{array}\right)\)
    // The secret key is \((H, Q)\)
    // Generate public key \(M\)
    \(L \leftarrow H \cdot Q=\left(L_{0} L_{1}\right) / * L_{0}\) and \(L_{1}\) are \(p \times p\) matrices */
    \(L_{1}^{-1} \leftarrow \operatorname{Inverse}\left(L_{1}\right)\)
    \(M \leftarrow L_{1}^{-1} \cdot L / *\) with all elements modulo 2 */
    // The public key is \(M\)
```


### 3.1 LEDAcrypt KEM

LEDAcrypt can be used by Alice to sent a key to another person (Bob) safely, so that there is a shared secret that can be used in symmetric-key-cryptography. To make sure nobody but the intended receiver can read this shared secret, this information is encoded using the Key Encapsulation Module (LEDAcrypt KEM). In algorithm 4 an error vector is encrypted with Bob's public key. This $c$ is the encapsulated temporary key, i.e. the key that is shared with Bob to be used for further (symmetric) communication. Bob will decrypt it using his secret key, find $e$ and hash this using SHA-3 hash function to get $K$. Thus after this step, both Bob and Alice know $K$. The decapsulation of the KEM is not shown here, since it is straightforward what happens there.

```
Algorithm 3: GenerateRandomList
    input : length of the list: \(p\),
        weigth of the list: \(w\)
    output: list of length \(p\) with \(w\) non-zero elements: \(l\)
    \(l_{p}=\operatorname{list}()\)
    \(l_{p} \leftarrow\) RandomSample \((p, w) / *\) Picks \(w\) unique random numbers in range 0 to \(p\) */
    for \(i\) in range \(p\) do
        if \(i\) in \(l_{p}\) then
            \(l[i] \leftarrow 1\)
        else
            \(l[i] \leftarrow 0\)
        end
    end
    // return \(l\)
```

```
Algorithm 4: LEDAcrypt Key Encapsulation Module (KEM)
    input : length of blocks: \(p\),
                weight of blocks of \(H: d_{v}\),
                public key: \(M\)
    output: encapsulated temporary key: \(c\),
            temporary key: \(K\)
    \(1 e \leftarrow\) GenerateRandomList \(\left(2 p, d_{v}\right)\)
    \(2 e^{\top} \leftarrow\) Transpose (e)
    \(\mathbf{3} c \leftarrow M \cdot e^{\top}\)
    // c is the encapsulated temporary key
    \(K \leftarrow\) SHA3 \((e)\)
    // \(K\) is the temporary key
```


## Chapter 4

## Weak keys attack

The attack described in this chapter was developed in [2]. It was published in 2020 by Daniel Apon, Ray Perlner, Angela Robinson and Paolo Santini (who was one of the creators of LEDAcrypt).

The attackers found that there exist classes of weak keys in LEDAcrypt, keys that can be recovered in much less time than was assumed at first. Protecting LEDAcrypt against the attack would require a change in the cryptosystem (an alteration that has been made later) but nonetheless, LEDAcrypt was not chosen as one of the round three candidates for NIST's competition [1]. This change in structure caused a resemblance to one of the other round two candidates, BIKE [3], which has made it to the third round. In this section, the attack will be explained.

This attack uses ISD algorithms, as the ones described in chapter 2 to find a low-weight code word, after which the key can be recovered. This is possible because the structure of the key cannot be hidden completely. As we now know, ISD algorithms can be performed much faster when one knows (or assumes) that certain positions contain zeros. The attack guesses the most probable positions of the non-zero elements of the key, and then performs the ISD on the other positions, to find the actual key. This replaces the random guesses. Instead they make an attack key $L^{\prime}$, the support of which hopefully will contain the support of $L$. Then an ISD algorithm is performed on the complement of these positions and so the key can be found if the support of $L$ is indeed contained in the support of $L^{\prime}$, i.e. if $\operatorname{supp}(L) \subseteq \operatorname{supp}\left(L^{\prime}\right)$.

But how can such an $L^{\prime}$ be found? The chance that one guesses an $L^{\prime}$ that works for a secret key is very slim. Of course $L^{\prime}$ will not be guessed directly, instead $h_{0}^{\prime}, h_{1}^{\prime}, q_{00}^{\prime}, q_{01}^{\prime}, q_{10}^{\prime}$ and $q_{11}^{\prime}$ will be chosen such that a good $L^{\prime}$ will be generated. But how does one choose those polynomials correctly? The answer to this problem is to use consecutive sparse polynomials. Before this statement can be elaborated, some more thought needs to be put in the properties of consecutive sparse polynomials and their support, so it will be easier to understand the attack. The sections about these topics, section 4.1 and section 4.2 explain the thought process needed, written in my own words.

### 4.1 Sparse polynomials

A sparse polynomial is a polynomial with few non-zero terms, where few is not defined precisely. In this thesis, this definition will suffice. A consecutive sparse polynomial is a sparse polynomial, where the non-zero terms have consecutive exponents.

In this section the following notation and polynomials will be used.

$$
\begin{align*}
& f(x)=\sum_{i=0}^{k} a_{i} x^{i}  \tag{4.1}\\
& g(x)=\sum_{i=0}^{l} b_{i} x^{i}  \tag{4.2}\\
& h(x)=\sum_{i=0}^{m} c_{i} x^{i} \tag{4.3}
\end{align*}
$$

with $f, g$ and $h$ polynomials with integer coefficients (i.e. $a_{i}, b_{i}, c_{i} \in \mathbb{Z} \forall i$ ). The degrees of the polynomials $f, g$ and $h$ are $k, l$ and $m$, respectively. This means that for $f(x) k$ is the largest $i$ for which $a_{i} \neq 0$. The same holds for $l$ and $m$. The smallest $i$ for which $a_{i}, b_{i}$ and $c_{i}$ are not 0 will be 0 for all three polynomials. Note that this can be assumed without loss of generality, since if this were not the case, one could multiply the entire polynomial with $x^{-j}$, such that this smallest $i$ would be 0 and the degree would be updated to a smaller number, namely $k-j$ for $f$.

Let $h(x)=f(x) \cdot g(x)$. The number of non-zero terms of $f, g$ and $h$ will be called $f_{t}, g_{t}$ and $h_{t}$, respectively. Note that these equal the number of elements in the supports of $f, g$ and $h$. For the attack, the most interesting thing to find out about the sparse polynomials are these numbers $f_{t}, g_{t}$ and $h_{t}$ and the relations between them. This is what this section will focus on.

First, it needs to be noted that the upper bound of $h_{t}$ is $f_{t} \cdot g_{t}$. This can be deduced from looking at the number of multiplications when generating $h$ from $f$ and $g$.

$$
\begin{aligned}
h(x) & =f(x) \cdot g(x) \\
& =\underbrace{\left(a_{0} x^{0}+a_{1} x^{1}+\ldots+a_{k} x^{k}\right)}_{f_{t} \text { non-zero terms }} \cdot \underbrace{\left(b_{0} x^{0}+b_{1} x^{1}+\ldots+b_{l} x^{l}\right)}_{g_{t} \text { non-zero terms }} \\
& =\underbrace{a_{0} x^{0} \cdot\left(b_{0} x^{0}+b_{1} x^{1}+\ldots+b_{l} x^{l}\right)}_{g_{t} \text { multiplications }} \underbrace{+\ldots+}_{f_{t} \text { of such terms }} \underbrace{a_{k} x^{k} \cdot\left(b_{0} x^{0}+b_{1} x^{1}+\ldots+b_{l} x^{l}\right)}_{g_{t} \text { multiplications }}
\end{aligned}
$$

Figure 4.1: Proof for $h_{t} \leq f_{t} \cdot g_{t}$
Thus, the total number of multiplications is $f_{t} \cdot g_{t}$. These multiplications might each cause a unique term in $h$, but if there are multiple terms with the same exponents, the number of non-zero terms in $h$ is smaller than this $f_{t} \cdot g_{t}$.

In this proof, it was not considered whether $f$ and $g$ were consecutive polynomials or not. Now, the difference between those two cases can be explained. Suppose that $f$ and $g$ are not consecutive polynomials, i.e. $f_{t}<k+1$ and $g_{t}<l+1$. The multiplication of $f$ and $g$ takes on the same form as in the equations 4.1 and again $h_{t} \leq f_{t} \cdot g_{t}$. For the consecutive case however, i.e. when $f_{t}=k+1$ and $g_{t}=l+1$, a lot of terms will overlap. The term in $h$ with the smallest exponent will be the constant term, since $x^{0} \cdot x^{0}=x^{0+0}$. But since the polynomials are consecutive, the terms of $h$ will be consecutive as well. The smallest term will appear where the smallest term of $f$ and the smallest term of $g$ were multiplied, and the same goes for the largest terms. This means that $h$ will have $(k+l)+1$ terms. Thus, for $f_{t}=k+1$ and $g_{t}=l+1, h_{t}=(k+l)+1=f_{t}+g_{t}-1$. This is a relatively small number of non-zero terms and for this property, these polynomials are used in the attack.

### 4.2 Properties of the support under reduction and lift

Note that the result in the former section can be translated into a property of the support. If $|\operatorname{supp}(f)|=k+1$ and $|\operatorname{supp}(g)|=l+1$, then $|\operatorname{supp}(f \cdot g)|=k+l+1$. Now we look at another
property of the support. First a polynomial $r$ is considered. This polynomials is defined over $\mathbb{F}_{2}$ and will thus be of the form $r(x)=\sum_{i} a_{i} x^{i}$ where $a_{i} \in\{0,1\}$. Now we will lift $r$ to $\mathbb{Z}$ and call it $r^{\prime}$. Thus $r^{\prime}$ will be of the form $r^{\prime}(x)=\sum_{i} a_{i}^{\prime} x^{i}$ where $a_{i}^{\prime} \in \mathbb{Z}$. Since $0 \in \mathbb{F}_{2}$ lifts to $0 \in \mathbb{Z}$ and $1 \in \mathbb{F}_{2}$ lifts to $1 \in \mathbb{Z}$, the support of $r^{\prime}$ equals the support of $r$.

Now consider polynomial $s^{\prime}$, defined over $\mathbb{Z}$. It is of the form $s^{\prime}(x)=\sum_{i} b_{i}^{\prime} x^{i}$, with $b_{i}^{\prime} \in \mathbb{Z}$. If $s^{\prime}$ is reduced modulo 2, it becomes a polynomial $s$ of the form $s(x)=\sum_{i} b_{i} x^{i}$ where $b_{i} \in\{0,1\}$. However, since there might be some even $b_{i}^{\prime}$, not all terms of $s^{\prime}$ appear in $s$. Let's take $s^{\prime}(x)=$ $2 x+3 x^{2}+x^{5}+7 x^{6}$. Then $s(x)=x^{2}+x^{5}+x^{6}$. The supports of $s^{\prime}$ and $s$ are $\operatorname{supp}\left(s^{\prime}\right)=\{1,2,5,6\}$ and $\operatorname{supp}(s)=\{2,5,6\}$. Thus in this case $\operatorname{supp}(s) \subsetneq \operatorname{supp}\left(s^{\prime}\right)$.

Since all the polynomials (and thus the entries of the matrices as well) are defined over $\mathbb{F}_{2}$, all the entries are either 0 or 1 . If two entries are added (corresponding to adding two terms in the polynomials) a cancellation will happen and a 0 will appear, instead of a 2 . For the same polynomials and matrices defined over $\mathbb{Z}$ however, this cancellation will not happen and the 2 will just stay in its place. This means that when cancellations happen for a polynomial $p$ defined over $\mathbb{F}_{2}$, the support of the polynomial $p$ will be smaller than the support of the same computation performed on the lifted polynomials.

### 4.3 Choosing $L^{\prime}$

These properties are taken into account when choosing a useful $L^{\prime}$. The polynomials for the attack matrices $H^{\prime}=\left(\begin{array}{ll}H_{0}^{\prime} & H_{1}^{\prime}\end{array}\right)$ and $Q^{\prime}=\left(\begin{array}{ll}Q_{00}^{\prime} & Q_{01}^{\prime} \\ Q_{10}^{\prime} & Q_{11}^{\prime}\end{array}\right)$ are chosen as consecutive sparse polynomials with degree $\left\lfloor\frac{p}{4}\right\rfloor+\epsilon$ where $\epsilon$ may be zero. (In the cases that were modeled in chapters 5 and 6 , $\epsilon=1$, because the number of non-zero elements in $L^{\prime}$ was too small to have a successful attack.). The chance that all or many of the non-zero elements in $h_{0}^{\prime}, h_{1}^{\prime}, q_{00}^{\prime}, q_{01}^{\prime}, q_{10}^{\prime}$ and $q_{11}^{\prime}$ are contained in the support of these consecutive polynomials is quite big, since every non-zero entry has an at least $\frac{\frac{p}{4}}{p}=\frac{1}{4}$ chance of being in the first quarter of the polynomial. Thus it is not a very bold assumption when one assumes that all (or nearly all) of the non-zero elements are contained in the supports of the polynomials.

Then $L^{\prime}$ is generated using these polynomials of $H^{\prime}$ and $Q^{\prime}$. According to the statements in section 4.1, $\left|\operatorname{supp}\left(L_{0}^{\prime}\right)\right|=\left|\operatorname{supp}\left(H_{0}^{\prime} Q_{00}^{\prime}+H_{1}^{\prime} Q_{10}^{\prime}\right)\right|$ will equal $2\left(\frac{p}{4}+\epsilon\right)+1$, which is roughly $\frac{p}{2}$. If all non-zero elements of the polynomials of the key were indeed contained in the supports of the polynomials of the attack key, then all non-zero elements in $L$ will appear in the positions that are contained in the support of $L^{\prime}$ and thus it is expected that the other $\frac{p}{2}$ positions in the key will contain zero elements. This means that around half of the columns of the matrix can be excluded from the guessing of the ISD algorithm, i.e. the matrix $X$ as described in section 2.6 will have size $\frac{p}{2} \times p$. Therefore the ISD algorithm can be sped up quite a lot and it is relatively easy to find the key.

This would be the best case scenario for the attacker. As an user of LEDAcrypt, it is quite the opposite. But how can you choose your key such that it cannot be found by this attack? In the next chapter I consider different user strategies for choosing a key and how this affects the success probability of the attack.

## Chapter 5

## Methodology

In order to understand the LEDAcrypt system and the attack better, it is easiest to start looking at very small cases. In order to do this, different programs were developed to simulate the generation of keys, the KEM (this method was explained in algorithm 4), how to measure the safety of keys against an attack and how to gather that data. The goal of gathering the data is to be able to find the weak keys in the small cases, so some conclusions can be made that will also hold for the large cases that were suggested by the creators. This chapter presents and explains an algorithm developed to measure the effectiveness of the attack as well as the strategies we defined for choosing keys that will be tested. The last section explains the approach in how we structure gathering the data to make it easier to compare strategies.

### 5.1 Measuring the effectiveness of the attack

Chapter 4 explained that the attack succeeds if the attacker manages to correctly guess where the non-zero elements appear (and thus knows positions guaranteed to be zero). This is done by guessing possible polynomials whose supports hopefully contain the support of the actual polynomials used in the key. For the small cases with size $p=11$ and $p=13$, the polynomials that we used for the attack were $h_{0}^{\prime}=h_{1}^{\prime}=q_{00}^{\prime}=q_{01}^{\prime}=q_{10}^{\prime}=q_{11}^{\prime}=x^{0}+x^{1}+\ldots+x^{\left\lfloor\frac{p}{4}\right\rfloor+1}$. The attack key was generated using the Niederreiter Key Generation algorithm as described in algorithm 2 but with all matrices defined over $\mathbb{Z}$ instead of over $\mathbb{F}_{2}$ and with these fixed polynomials. Steps 16 and 17 will be omitted, because the inverse over $\mathbb{Z}$ cannot be computed and $M$ will not be used for this attack.

The indices of the non-zero elements of all the rows of this attack key are stored as a list of sets (every set gives the indices of one row of the attack key). This list of sets will be used to check if the support of the first row of a generated key is a subset of the checklist. If this is the case, the key can be restored using the ISD algorithm and the attack is successful. The checklists for the cases $p=11$ and $p=13$ can be found in chapter A. Note that the algorithm does not accept a few non-zero elements to be outside of the support of $L^{\prime}$, whereas the original would still be able to break the keys for which this would happen, if they used Lee-Brickell as described in 2.6.

In analyzing the effectiveness of the attack, target keys will be generated using certain input polynomials, for example for a given $h_{0}$ and $q_{00}$. The other polynomials will be randomly generated. This will be done by the function GenerateKeyFromInput as shown in algorithm 5 , which uses these polynomials to generate the key $L$ as described in chapter 3.

If the support of the first row of $L$ is a subset of a set in the checklist, it counts as a success. The code runs a number of times to try a lot of different keys and the indices of non-zero elements appearing in any of those keys (also those that do not get attacked successfully) and the number of successes are kept in the final output list $N$. So, after running this code, one can see the success rate of the attack for a certain strategy of making keys. The input polynomials and the number of runs are used as input, and the list $N$ is the output. See algorithm 6 for this code.

```
Algorithm 5: GenerateKeyFromInput
    input : input used to generate key: \(p h_{0}, p h_{1}, p q_{00}, p q_{01}, p q_{10}, p q_{11} / *\) list with correct
        length and weight or ' R ' for random */
    output: matrix \(L\) generated from the input
    // Generate \(H_{0}\)
    if \(p h_{0}=\) ' \(R\) ' then
        \(h_{0} \leftarrow\) GenerateRandomList \(\left(p, d_{v}\right)\)
        \(H_{0} \leftarrow\) GenerateCirculantMatrix \(\left(h_{0}\right)\)
    else
        \(H_{0} \leftarrow\) GenerateCirculantMatrix \(\left(p h_{0}\right)\)
    end
    if \(p h_{1}=\) ' \(R\) ' then
        \(h_{1} \leftarrow \operatorname{GenerateRandomList}\left(p, d_{v}\right)\)
        \(H_{1} \leftarrow\) GenerateCirculantMatrix \(\left(h_{1}\right)\)
    else
        \(H_{1} \leftarrow\) GenerateCirculantMatrix \(\left(p h_{1}\right)\)
    end
    // Generate \(Q_{00}\)
    if \(p q_{00}=\) ' \(R\) ' then
        \(q_{00} \leftarrow\) GenerateRandomList \(\left(p, d_{v}-m\right)\)
        \(Q_{00} \leftarrow\) GenerateCirculantMatrix \(\left(q_{00}\right)\)
    else
        \(Q_{00} \leftarrow\) GenerateCirculantMatrix ( \(p q_{00}\) )
    end
    // Generate \(Q_{01}\)
    if \(p q_{01}=' R\) ' then
        \(q_{01} \leftarrow\) GenerateRandomList \((p, m)\)
        \(Q_{01} \leftarrow\) GenerateCirculantMatrix ( \(q_{01}\) )
    else
        \(Q_{01} \leftarrow\) GenerateCirculantMatrix \(\left(p q_{01}\right)\)
    end
    // Generate \(Q_{10}\)
    if \(p q_{10}=' R\) ' then
        \(q_{10} \leftarrow\) GenerateRandomList \((p, m)\)
        \(Q_{10} \leftarrow\) GenerateCirculantMatrix \(\left(q_{10}\right)\)
    else
        \(Q_{10} \leftarrow\) GenerateCirculantMatrix \(\left(p q_{10}\right)\)
    end
    // Generate \(Q_{11}\)
    if \(p q_{11}=\) ' \(R\) ' then
        \(q_{11} \leftarrow \operatorname{GenerateRandomList}\left(p, d_{v}-m\right)\)
        \(Q_{11} \leftarrow\) GenerateCirculantMatrix \(\left(q_{11}\right)\)
    else
        \(Q_{11} \leftarrow\) GenerateCirculantMatrix \(\left(p q_{11}\right)\)
    end
    // Generate \(L\)
    \(H \leftarrow\left(H_{0} H_{1}\right)\)
    \(Q \leftarrow\left(\begin{array}{cc}Q 00 & Q 01 \\ Q 10 & Q 11\end{array}\right)\)
    \(L \leftarrow H \cdot Q / *\) with all elements modulo 2 */
```

```
Algorithm 6: MeasureAttack
    input : Number of runs: \(r\),
                            input used to generate key: \(p h_{0}, p h_{1}, p q_{00}, p q_{01}, p q_{10}, p q_{11} / *\) list with correct
        length and weight or ' R ' for random */
    output: List with number of appearances of non-zero elements per position and number
                    of successes: \(N\)
    \(s=\) false /* Boolean */
    \(N=\operatorname{list}(0) / * N\) is a list of length \(p \cdot n_{0}+1\) filled with zeros */
    \(C=\operatorname{list}(\operatorname{set}()) / *\) See appendix A for the versions of this checklist */
    for \(i\) in range ( \(r\) ) do
        \(L \leftarrow\) GenerateKeyFromInput ( \(p h_{0}, p h_{1}, p q_{00}, p q_{01}, p q_{10}, p q_{11}\) )
        \(l=\operatorname{list}(\) )
        for \(j\) in range \(\left(p \cdot n_{0}\right)\) do
            if \(L[0][j]==1\) then
            \(l \leftarrow l \mid j / *\) The position of the non-zero element is appended to \(l * /\)
            \(N[j] \leftarrow N[j]+1 / * N\) keeps track of the postions */
            end
        end
        for \(k\) in range ( \(p\) ) do
            if \(l\) SubsetOf \((C[k])\) then
                \(s \leftarrow\) True
            end
        end
        if \(s==\) True then
            \(N\left[p \cdot n_{0}\right] \leftarrow N\left[p \cdot n_{0}\right]+1 / *\) Update the number of successes */
        end
    end
```

After the testing of this code and gathering some data by hand, the code was extended to be able to gather more data in one go and to immediately save the data in a .csv file. To do this, the code took the input from a two-dimensional array $D$, which needed to be prepared beforehand. Every row of $D$ was an input line for the measuring of the effectiveness of the attack. To the function MeasureData a few lines were added to open a .csv file, to write all the elements of $N$ in that file, seperated by semicolons and to close the file. A for-loop was used to go through every row of $D$ and thus to gather a lot of data at once.

### 5.2 Strategies

When choosing polynomials for generating keys for LEDAcrypt, one can have different strategies. Obviously, the preferred key is one that can not be broken with the attack, but how to know which keys meet that demand? To try to find out which keys are 'good' and which keys are 'bad', the code described above is used to measure how well certain groups of keys perform when they are attacked. These different groups of keys correspond to possible strategies when choosing how to generate keys and to make them easily distinguishable, all strategies were given a name. They will be explained in this section.

## All-knowing Alice

Alice is familiar with a lot of cryptosystems and although she and her friend Bob have been the victims of many successful attacks of Eve in all these systems, they always manage to find a way to secure their communication. Therefore, it is assumed that Alice, as the experienced user of cryptosystems, will use a very safe strategy. She generates her keys completely at random in order to protect her bits. All the other strategies will be compared to this one, since this is probably the safest one.

## Consecutive Charlie

Charlie is not experienced at all and prefers simple keys. She will choose to put the non-zero elements of her polynomials in one consecutive block.

## Split Simon

Simon's strategy resembles the one from Charlie, but instead of using one consecutive block, Simon splits his block into two smaller blocks by putting one or more zeros in between.

## Regular Ron

Unlike Charlie and Simon, Ron does not necessarily keep his non-zero elements in blocks. Rather, he splits them up in multiple groups, or separates them all. The number of zeros in between blocks or single non-zero elements has a certain regularity to them. Either every 'hole' has the same number of zeros or some of them do. This strategy also tries out some irregular holes, Ron is secretly admiring Alice and tries to be like her sometimes.

### 5.3 Structure in the data

In order to get useful data, some structure was added to the data. For every strategy of choosing keys, the same type of structure was used so the strategies can be compared. For every strategy, the data is divided among different phases. These different phases are described below.

## Phases in the test and what they can show:

| Parameter | value for first set | Value for second set |
| :---: | :---: | :---: |
| $n_{0}$ | 2 | 2 |
| $p$ | 11 | 13 |
| $d_{v}$ | 3 | 5 |
| $m_{0}$ | 1 | 2 |
| $m_{1}$ | 2 | 3 |

Table 5.1: Parameter sets used for LEDAcrypt in this thesis

I Only fixing $h_{0}$ or $h_{1}$. This shows the influence of $h_{0}$ and $h_{1}$ and possible shifts in these polynomials

II Fixing combinations of $h_{0}$ and $h_{1}$ (how much do these affect the results?)
III Only fixing one polynomial of $Q$. This shows the influence of the polynomials of $Q$
IV Fixing combinations of two polynomials of $Q$.
V Fixing $h_{0}$ in combination with one or two polynomials of $Q$. This will tell us about the influence of combinations of polynomials.

Results and analysis of the data found will be given in the next chapter.

### 5.4 Applying the methology to a larger case

In this thesis, we consider LEDAcrypt with two sets of parameters, see table 5.1.
In practice, LEDAcrypt would not use polynomials of such small lengths, since there is not much information that can be sent with these and Eve can easily do a brute force attack, when having access to enough computational power. In the specification for LEDAcrypt [4], we find the parameters for different levels of security (different NIST categories). The following parameters are used for NIST category 1.
$\boldsymbol{n}_{\mathbf{0}}=\mathbf{2}$ In this thesis and in the attack paper, only this case was considered, but the attack and the measuring of the attack would work for the larger $n_{0}$ as well
$\boldsymbol{p}=14939$ Note that this is 1000 times larger than the cases we will consider in detail in the next chapter
$\boldsymbol{d}_{\boldsymbol{v}}=11$ This means that less than 1 on 1000 positions in the polynomial is a non-zero element. The cases that are considered in this report have almost 1 on 2
$m_{0}=4$ and $m_{1}=\mathbf{3}$ Note that the weight of $Q$ is now smaller than the weight of $H$
Since the ratio between $p$ and $d_{v}$ differs so much with the $p=11$ and $p=13$ case, we thought it would be interesting to try to attack a key of this large size. To do this, the checklist would have to be generated first. Unfortunately, we immediately ran into a memory error, my laptop was not able to do calculations with matrices of that size. Thus, instead of computing with matrices, we rewrote the checklist generating algorithm so it would work with lists of positions of non-zero elements. After a few tests, this method worked, but when performed on the $p=14939$ case, it took more than 3 days and then the file (which was more than 1GB in size) with the checklist was still not complete. Then we ran the code again, but only for the first row of the checklist. This worked perfectly, but cannot be attached as attachment, because Overleaf cannot handle that. Because of these problems, with the time and computational power on hand, we did not manage to run for the large case and gather data on that.

## Chapter 6

## Results

Using algorithm 6 data for every strategy and phase was gathered. For the $p=11$ case 135 different lines of data were gathered. For $p=13$ case 202 lines were gathered. All this data was divided and stored by strategy and phase. Then it was analyzed and visualized.

In this section the data that was gathered for the attack on keys with the parameter sets described in table 5.1 will be discussed per strategy, and the strategies will be compared at the end. Sections will first discuss the $p=11$ case, and will then discuss the $p=13$ case and compare the two. The strategies of Alice and Charlie will be described completely, so the idea about the gathering and analyzing of the data is clear. The other strategies will be compared to these two, so it is important to cover them. For the other strategies only the very remarkable results will be highlighted. A few terms that will be used in this chapter are 'shifts', 'blocks' and 'holes'. With a 'shift of one position' is meant that the positions of the elements of the polynomial are mapped one position to the right. When no shift is mentioned, the first element of the polynomial is non-zero. A 'block of size four' means four consecutive non-zero elements. A 'hole of size three' are three consecutive zero elements between non-zero elements or blocks. Note that if the program that would gather the data would run again, slightly different success rates may be found. Since the program works with randomness, results may vary around the 'true' success rate. Repeating the process a number of times will give more information about this 'true' success rate, but requires a lot of computational time.

### 6.1 Alice

For both $p=11$ and $p=13$, Alice has no fixed polynomials, but uses randomly generated polynomials. This is expected to be the best strategy. For $p=11$, the data shows that the attack will find $0.35 \%$ of all random keys (average of $0.333 \%$, i.e. the result for 1000000 tries and $0.36 \%$ ). Considering that there are $2^{22}=4194304$ possible keys, this means that about 1453327 keys can be found. Although not all of those possible keys can be generated from $H$ and $Q$, so the actual value will be lower. The result for Alice in the $p=13$ case is a bit lower, the success rate is now $0.19 \%$ (average of $0.14 \%$ and $0.23 \%$, the result for 1000000 tries). To check whether the distribution of weights in $Q$ affects the success rate, we checked Alice but with slightly different parameters. Instead of using $m_{0}=d_{v}-m$ and $m_{1}=m$, we used $m_{0}=m$ and $m_{1}=d_{v}-m$. As you can see in figure 6.1, the results are relatively close to each other. That Alice has a smaller success rate can be caused by the length. There are much more possible keys with length $13\left(2^{26}>2^{22}\right)$.

### 6.2 Charlie

As mentioned in the last chapter, Charlie likes consecutive blocks. Thus she chooses her polynomials such that all non-zero elements are in one block.


Figure 6.1: Comparing Alice for different weigth distributions of $Q$

## Phase 1

The influence of a fixed $h_{0}$ or $h_{1}$ is shown for the first time. Charlie also tries out a shift for the $p=13$ case, just to check if it has influence. As you can see in figure 6.2 , the values are still quite close to each other, although the $p=13$ case has a lower success rate again for the $h_{0}$ case, but not for $h_{1}$. The shifted $h_{0}$ does not have a very large influence, which is as expected, since the keys are cyclic and the attack checks every row of the key. Fixing just one polynomials does not affect the success rate very much, the values are still in the same range as for Alice.

Charlie phase 1


Keys

Figure 6.2: Influences of Charlie's $h_{0}$ or $h_{1}$ on the success rate

## Phase 2

In this phase the combination of a fixed $h_{0}$ and $h_{1}$ is discussed. As can be seen in figure 6.3 , this gives a much better chance for the attack to work. Around $2 \%$ of the keys is now found, which is a lot. Since there are 6 non-zero elements fixed (for the $p=11$ case) and because these are fixed in consecutive blocks, like the polynomials $h_{0}^{\prime}$ and $h_{1}^{\prime}$ have, the chance that the non-zero elements appear in the beneficial positions for the attack is quite high. Shifts of both of the polynomials or of $h_{1}$ (and thus shifts of $h_{0}$ too, since we saw in the previous section that there is not much difference between those two polynomials) do not affect the success rate very much, as expected.

Interesting is that the success rates for the $p=13$ case are much lower. This happens because fixing both $h_{0}$ and $h_{1}$ means fixing 10 non-zero elements. This would mean a higher success rate, if the attack keys had a large support that could contain these non-zero elements. The attack keys $h_{0}^{\prime}$ and $h_{1}^{\prime}$ have weight $\left\lfloor\frac{13}{4}\right\rfloor+1=4$, which means that fixing 5 non-zero elements lowers the chance of an successful attack. The shift for $h_{1}$ seems to have a small positive influence on the success rate.


Figure 6.3: Influences of Charlie's $h_{0}$ and $h_{1}$ on the success rate

## Phase 3

In phase 3 , all the polynomials for $Q$ will be considered and compared. The weights of the polynomials of $Q$ are smaller than the weights of the polynomials of $H$, but the success rates are not noticeably smaller than those for the polynomials of $H$. In figure 6.4, is is clear to see that there is not much difference between the values for $q_{00}, q_{01}$ and $q_{10}$. It also seems like the success rate for $q_{11}$ is smaller than the others (although not that much smaller for the $p=13$ case). The numbers of non-zero elements in $q_{11}$ and $q_{00}$ are the smallest, but there should be no difference between $q_{00}$ and $q_{11}$. This might just have been an unlucky set of keys, which we noticed quite late. In figure 6.5 , we can see that a shift of one position in the $p=13$ case does not cause for large differences in success rates.

## Phase 4

Now the combinations of polynomials of $Q$ are considered. For the $p=11$ case, it vaguely looks like the number of fixed non-zero elements increases the chance of success for the attack, but these differences are so small that one cannot conclude this with confidence. Interesting

Charlie phase 3


Figure 6.4: Influences of Charlie's polynomials of $Q$ on the success rate

Shifts for $p=13$ in Charlie phase 3


Figure 6.5: Influences of shifts in the polynomials of $Q$
is that the combinations of $q_{i i}$ 's cause much less of a problem for the safety of the key than the combinations of $h_{i}$ 's did. The $p=13$ case shows almost no variation in the success rates between the different combinations. For both cases, the difference between the success rates for combinations of polynomials of $Q$ and single polynomials is very small. Therefore, we can conclude that the $q_{i i}$ 's do not impact the success of the attack very much.


Figure 6.6: Combinations of polynomials of $Q$ for Charlie

## Phase 5

Now combinations of $h_{0}$ and the polynomials of $Q$ are considered. No shifts were checked, because earlier phases showed no real difference in those cases. In the figures 6.7 and 6.8 the combinations of multiple polynomials for $p=11$ and $p=13$ are shown.

This last phase seems to confirm that for $p=11$ the number of fixed non-zero elements causes the largest changes in the success rate. Combinations of $h_{0}$ and $q_{01}$ or $q_{10}$ ( 5 fixed non-zero elements) give success rates of about $0,6 \%$, whereas $h_{0}$ and $q_{00}$ or $q_{11}$ ( 4 fixed non-zero elements)
give only $0,26 \%$. Both of these values are significantly less than the values found in phase 2 ( 6 fixed non-zero elements).

Combinations of three polynomials follow the same trend with low success rates for the combination of $h_{0}, q_{00}$ and $q_{11}$ and a high success rates for the combination $h_{0}, q_{01}$ and $q_{10}$. The values for the success rates in this phase also exceed those of phase 4 , that have a maximum for the number of fixed non-zero elements at 4.

For $p=13$ however, the number of fixed non-zero elements keeps exceeding the number of non-zero elements in the attack polynomials, making it harder for the attack to succeed. A very low success rate of $0.08 \%$ is found for the combination of $h_{0}$ and $q_{10}$ ( 8 fixed non-zero elements). The combinations of three polynomials give very stable outcomes.


Figure 6.7: Charlie's combination of $h_{0}$ and a polynomials of $Q$


Figure 6.8: Influences of shifts in the polynomials of $Q$

### 6.3 Simon

Simon does not use one consecutive block, but instead splits it up into two blocks by adding zero elements. We expected these to be better keys than Charlie's. Since there are not many interesting splits to try for 1 or 2 non-zero elements, phases 3 and 4 are not very interesting for this case.

## Phase 2

The data in all the phases confirms our expectations, so we will not discuss everything. An interesting observation in figure 6.9 is the drop in the success rate for a hole of size 3 however. Apparently this causes a spread of non-zero elements, such that some are not contained in the support of the attack polynomials. The hole of size 4 however, causes a spread so wide that some elements modulo $p$ get back into the support. Thus; the size of holes also matters.

### 6.4 Ron

Ron goes a bit further than Simon and divides his non-zero elements into more than two blocks. To try to get the safest keys possible, he tries out different sizes for the holes and even uses irregular hole size sometimes, to be as much as Alice as possible.


Figure 6.9: Simon phase 2 with $h_{1}=x^{0}+x^{1}+x^{3}$

## Phase 1

When looking at the sizes of the regular holes in figure 6.10 (i.e. the holes between two non-zero elements have the same size), it can be seen that for $p=11$ there is a dip for the holes of size 2 , but with the regular holes of size 1 and 3 , the success rate is similar to that of Alice and Charlie. The values for $p=13$ somewhat resemble Charlie and Alice as well. What can explain this, is that if you have two polynomials with only non-zero elements on the even positions (so holes of size 1 ), all the non-zero elements after multiplication or addition will appear on the even positions. The odd positions are not touched, but this means that this resembles the consecutive case, but then with zero elements between the non-zero elements. More general: for a fixed size of holes $k$,

$$
\begin{equation*}
h(x)=f(x) \cdot g(x)=\sum_{i} a_{i(k+1)} x^{i(k+1)} \cdot \sum_{j} b_{j(k+1)} x^{j(k+1)}=\sum_{i, j} a_{i(k+1)} b_{j(k+1)} x^{(i+j)(k+1)} \tag{6.1}
\end{equation*}
$$

This means that all the positions $1,2, \cdots k, k+2, \cdots$ are not included in this equations and thus the support of the product remains very small. On top of that, a lot of cancellations can happen. If the holes not too large (causing wide-spread positions of non-zero elements) then these keys of Ron are quite unsafe.

Now the polynomials with non-regular sized holes are considered. In figure $6.12 h_{0}$ 's are considered where the non-zero elements are separated by blocks of size $2,1,2$ and 1 (in that order). Another polynomial has blocks (two blocks of size 2, and one single non-zero element) separated by holes of size 1 and 4 . The success rates for these non-regular polynomials are again around the $0.25 \%$. The polynomials shown in figure 6.11 all hover around the $0.34 \%$ success rate that was also seen in the other strategies, except for the last polynomials, where relatively large holes are used. Spreading the non-zero elements so far, might lead to less cancellations and, more importantly, makes it harder for the consecutive attack polynomials to catch the non-zero elements from the key in their support.

## Phase 2

Now we will consider combinations of $h_{0}$ and $h_{1}$. In figure 6.13 very large differences in success rates are shown. In this bar graph, combinations of two regular polynomials are shown, where the holes of both polynomials are equal and $h_{1}$ is shifted. The regular holes of size 1 give a success chance of a little over $1 \%$, which means the keys are not safe. As soon as the size of the holes are increased however, the success rate drops down to 10 successes for holes of size 2 and no successes for holes of size 2. Larger holes increase the chance of non-zero elements appearing outside the

Ron phase 1; regular holes in $h_{0}$


Figure 6.10: Regular holes in Ron's $h_{0}$

support of $L^{\prime}$. The shifts of the polynomials do not change the results. A similar thing happens for $p=13$, although less extreme. In figure 6.14 the difference between the holes of size 1 and the holes of size 2 is notable, but not as extreme as for the $p=11$ case. A more compact $h_{1}$ can also cause a lower success rate.

## Phase 3

In figure 6.15 a clear distinction between success rates is shown. Shifts for $h_{1}$ do not cause large success rates, but if you mirror $h_{0}=x^{0}+x^{3}+x^{5}$ and shift 1 , so $h_{1}=x^{1}+x^{3}+x^{6}$ then the success rates go up. This regularity also causes cancellations, which might be why the success rate is so high.

The other cases of Ron show more of the same, higher success rates than we would have expected at first. With a few very low success rates for polynomials with large holes like in figure
 nomials with shifts


Figure 6.15: Ron phase 3 with $h_{0}=x^{0}+x^{3}+x^{5}$
6.13. Thus these phases will not be shown anymore.

### 6.5 Comparing the strategies

In this section the four strategies will be compared to see which one is the safest. In the other section, the shifts and other variations within the strategies are analysed, in this section the most basic polynomials that fit the strategy will be considered but some references to the previous section might be made.

## Phase 1

In figure 6.16 we can see the comparison of Alice to the other strategies and it is interesting to see that Alice does not have the lowest success rate compared to the other strategies. This can be explained with the fact that Alice contains weak keys as well as good keys. Since they are randomly generated, you have a chance that a weak one is generated. Since the keys are so short, $p=11$, and the weight $d_{v}=3$ is very high in comparison to the length, the chance that the non-zero elements end up in a consecutive block is quite high. Thus, Alice also contains keys from Charlie and Ron, and this can cause the success rate to be higher than suspected. For $p=13$ we can see results that would be expected, with Alice the best. You would expect a decrease from Alice on, but the numbers are so close that it is inconclusive.


Figure 6.16: Comparing the strategies for phase 1

## Phase 2

Figure 6.17 very clearly shows that Charlie is not the right choice for your keys, there is a clear increase in the success chance. Here we can see that Ron is indeed worse than Simon (for the regular holes polynomials that were used for this graph).

## Phase 3

It is interesting to see in figure 6.18 that for $p=11$, Simon has the highest success rates. The difference between 38 and 43 is not big enough to draw conclusions. Since $w\left(q_{00}\right)=1$, one cannot split non-zero elements here. Thus there is no data for $q_{00}$ for Simon and Ron. The reason why the data for Ron for $q_{01}$ is missing is that this case would be equal to the Simon case, since there are only two non-zero elements in $q_{01}$. The same holds for $q_{00}$ in the $p=13$ case.

## Phase 4

Figure 6.19 does not really show something new here, although Simon is again a bad strategy to pick. This can be explained with the cancellations that still happen because the non-zero elements are still quite densely together. Charlie probably had a lucky run here, you would expect that success rate to be higher. For $p=11$, Ron's cases would equal Simon's cases, which is why no data was gathered for Ron here.


Figure 6.17: Comparing the strategies for phase 2


Figure 6.18: Comparing the strategies for phase 3

## Phase 5

We look at figure 6.20 and can see very clearly that the number of non-zero elements clearly influences the success rates for $p=11$, but not for $p=13$. There are very low values for $p=13$, Charlie is especially low this time, but this is probably a lucky run.

In figure 6.21 three polynomials are fixed instead of two. This gives very high values for both cases. In both figures, no data was gathered for Simon in the $p=11$ case including $q_{00}$ due to the single non-zero element in that polynomial. For Ron's data, the fact that $q_{00}$ could not be modified to his preference was ignored.


Figure 6.19: Comparing the strategies for phase 4


Figure 6.20: Comparing the strategies for phase 5


Figure 6.21: Comparing the strategies for phase 5

## Chapter 7

## Conclusion

The data in this thesis shows that the weak keys attack can indeed break certain keys. The keys of Charlie and the regular keys from Ron are weak. The keys of Simon are a bit better. The small cases with $p=11$ and $p=13$ can show us a little bit about the influence of splitsing polynomials for example, but it cannot give us a clear view on the real life cases, for example the one with $p=14939$. We saw that for both cases (but for $p=11$ in a greater extent) the number of bad keys is larger, simply because the chance that non-zero elements form consecutive blocks is large because of the ratio between the length and the weight of the polynomials. This causes the attack to have a larger success rate for Alice than was expected. For $p=13$ in particular, fixing the polynomials fixed more non-zero elements than non-zero elements in the supports of the attack polynomials. This causes a lower success rate of the attack for all strategies than were expected.

Since the data does show that the attack can break certain keys, it is not safe to use LEDAcrypt in the form that was described in this thesis.

## References

[1] G. Alagic, J. Alperin-Sherrif, D. Apon, D. Cooper, Q. Dang, J. Kelsey, Y.K. Liu, C. Miller, D. Moody, R. Peralta, R. Perlner, A. Robinson, and D. Smith-Tone. Status Report on the Second Round of the NIST Post-Quantum Cryptography Standardization Process, 2020. https://doi.org/10.6028/NIST.IR. 8309.
[2] D. Apon, R.A. Perlner, A. Robinson, and P. Santini. Cryptanalysis of LEDAcrypt. In Daniele Micciancio and Thomas Ristenpart, editors, Advances in Cryptology - CRYPTO 2020 - 40th Annual International Cryptology Conference, CRYPTO 2020, Santa Barbara, CA, USA, August 17-21, 2020, Proceedings, Part III, volume 12172 of Lecture Notes in Computer Science, pages 389-418. Springer, 2020. https://doi.org/10.1007/978-3-030-56877-1_ 14.
[3] N. Aragon, P.S.L.M. Barreto, S. Bettaieb, L. Bidoux, O. Blazy, J.C. Deneuville, P. Gaborit, S. Ghosh, S. Gueron, T. Güneysu, C.A. Melchorand, R. Misoczki, E. Persichetti, N. Sendrier, J.P. Tillich, V. Vasseur, and G. Zémor. BIKE - Bit Flipping Key Encapsulation. https: //bikesuite.org/.
[4] M. Baldi, A. Barenghi, F. Chiaraluce, G. Pelosi, and P. Santini. LEDAcrypt: Low-density parity-check code-based cryptographic systems, 30 March 2019. https://csrc.nist.gov/ projects/post-quantum-cryptography/round-2-submissions Document can be found in .zip-file of LEDAcrypt submission.
[5] W. Diffie and M.E. Hellman. New directions in cryptography. IEEE Trans. Inf. Theory, 22(6):644-654, 1976. https://doi.org/10.1109/TIT.1976.1055638.
[6] T. Lange. Overview of Code-Based Crypto Assumptions, February 2020. https:// simons.berkeley.edu/talks/overview-code-based-crypto-assumptions Talk at Quantum Cryptanalysis of Post-Quantum Cryptography.
[7] P.J. Lee and E.F. Brickell. An Observation on the Security of McEliece's Public-Key Cryptosystem. In Christoph G. Günther, editor, Advances in Cryptology - EUROCRYPT '88, Workshop on the Theory and Application of of Cryptographic Techniques, Davos, Switzerland, May 25-27, 1988, Proceedings, volume 330 of Lecture Notes in Computer Science, pages 275-280. Springer, 1988. https://doi.org/10.1007/3-540-45961-8_25.
[8] R.J. McEliece. A Public-Key Cryptosystem Based On Algebraic Coding Theory. Deep Space Network Progress Report, 42-44:114-116, jan 1978. https://ipnpr.jpl.nasa.gov/ progress_report2/42-44/44N.PDF.
[9] H. Niederreiter. Knapsack-type cryptosystems and algebraic coding theory. Problems of Control and Information Theory, 15(2):159-166, 1986.
[10] NIST. Post-Quantum Cryptography Standardization. https://csrc.nist.gov/Projects/ post-quantum-cryptography/post-quantum-cryptography-standardization.
[11] E. Prange. The use of information sets in decoding cyclic codes. IRE Trans. Inf. Theory, 8(5):5-9, 1962. https://doi.org/10.1109/TIT.1962.1057777.
[12] A. Ravagnani. Lecture notes of the "Coding Theory" course at TU/e, 2020. https://a. ravagnani.win.tue.nl/coding\ theory\ 20_21.html.
[13] R.L. Rivest, A. Shamir, and L.M. Adleman. A Method for Obtaining Digital Signatures and Public-Key Cryptosystems. Commun. ACM, 21(2):120-126, 1978. http://doi.acm.org/10. 1145/359340. 359342.
[14] P.W. Shor. Algorithms for Quantum Computation: Discrete Logarithms and Factoring. In 35th Annual Symposium on Foundations of Computer Science, Santa Fe, New Mexico, USA, 20-22 November 1994, pages 124-134. IEEE Computer Society, 1994. https://doi.org/10. 1109/SFCS. 1994.365700.
[15] H.C.A. van Tilborg. Coding theory, a first course. Eindhoven the Netherlands, 1993. https: //hyperelliptic.org/tanja/teaching/CCI11/CODING.pdf.
[16] H.C.A. Van Tilborg. Fundamentals of cryptology: a professional reference and interactive tutorial. Kluwer Academic Publishers, 2000. https://www.hyperelliptic.org/tanja/ teaching/cryptoI13/cryptodict.pdf.

## Appendix A

## Checklists for the attack

## A. 1 Checklist for $p=11$

```
check = [{0,1,2,3,4,11,12,13,14,15},
    {1,2,3,4,5,12,13,14,15,16},
    {2,3,4,5,6,13,14,15,16,17},
    {3,4,5,6,7,14,15,16,17,18},
    {4,5,6,7,8,15,16,17,18,19},
    {5,6,7,8,9,16,17,18,19, 20},
    {6,7,8,9,10,17,18,19,20,21},
    {0,7,8,9,10,11,18,19,20,21},
    {0,1,8,9,10,11,12,19,20,21},
    {0,1,2, 9, 10,11, 12,13, 20, 21}
    {0,1,2,3,10,11,12,13,14,21}]
```


## A. 2 Checklist for $p=13$

```
check13 = [{0,1, 2, 3, 4, 5, 6, 13, 14, 15,16, 17, 18, 19},
    {1,2,3,4,5,6,7,14,15,16,17,18,19, 20},
    {2,3,4,5,6,7,8,15,16,17,18,19,20, 21},
    {3,4,5,6,7,8,9,16,17,18,19,20, 21, 22},
    {4,5,6,7,8,9,10,17,18,19,20,21,22,23},
    {5,6,7,8,9,10,11, 18,19,20,21, 22, 23, 24},
    {6,7,8,9,10, 11, 12, 19, 20, 21, 22, 23, 24, 25},
    {0,7,8,9,10, 11, 12, 13, 20, 21, 22, 23,24, 25},
    {0,1,8,9,10, 11, 12, 13, 14, 21, 22, 23, 24, 25},
    {0,1,2,9,10,11,12,13,14,15,22,23,24,25},
    {0,1,2,3,10,11,12,13,14,15,16,23,24,25},
    {0,1,2,3,4,11, 12, 13, 14, 15, 16,17, 24,25},
    {0,1,2,3,4,5,12,13,14,15,16,17,18,25}]
```


[^0]:    ${ }^{1}$ vectors are seen as column vectors, but we use $w(c)$ to denote the column weight.

