國　立　交　通　大　學

資　訊　工　程　學　系

碩　士　論　文

馬可利斯公鑰密碼系統之三分法反應攻擊

A Trichotomy Reaction Attack on McEliece Public-Key Cryptosystem

研究生:　梁漢璋

指導教授:　陳榮傑 教授

中國民國　九十四　年　六　月

馬可利斯公鑰密碼系統之三分法反應攻擊

A Trichotomy Reaction Attack on McEliece Public-Key Cryptosystem

研究生: 梁漢璋 Student: Han-Chang Liang
指導教授: 陳榮傑 教授 Advisor: Prof. Rong-Jaye Chen

國 立 交 通 大 學
資 訊 工 程 學 系
碩 士 論 文

A Thesis
Submitted to Institute of Computer Science and
Information Engineering
College of Electrical Engineering and Computer Science
National Chiao Tung University
in
Partial Fulfillment of the Requirements
for the Degree of Master
in
Computer Science and Information Engineering

June 2005
Hsin-Chu, Taiwan, Republic of China

中國民國 九十四 年 六 月

誌謝

這篇論文能夠得以完成, 首先必須感謝我的指導教授陳榮傑老師, 給予我研究的方向以及資源。也感謝師母李惠慈女士爲我指出論文寫作當中的不當之處, 使我的論文得以改頭換面。更要感謝我的家人無條件地支持我, 使我得以無後顧之憂地完成碩士學業。最後要感謝實驗室的夥伴們: 鈞祥、志賢、凱群、緯凱、文鼎學長, 政愷、志彬、韋廷, 學弟葉薰以及 資科系資訊安全實驗室的冠廷。和你們一同討論, 逐步釐清思緒, 是一個很棒的經驗。另外要特別感謝實驗室已畢業的學長: 張仁俊博士, 在研究方面給了我很大的啓發。

馬可利斯公鑰密碼系統之三分法反應攻擊

研究生: 梁漢璋　　指導教授: 陳榮傑 教授

國立交通大學

資訊工程學系

摘要

馬可利斯公鑰密碼系統是第一個結合了代數編碼領域及密碼領域的公鑰密碼系統。Hall 等三位學者於 1999 年提出了第一個對於馬可利斯公鑰密碼系統的反應攻擊。相較於選擇密文攻擊而言, 反應攻擊可行性較高, 但需要較多的詢問次數。在這篇論文當中, 我們提出一個三分法反應模型。該模型假設解密者對於一個不合法的密文, 將判斷是否仍可解密同時明文正確, 據此給予兩種程度不同的警告回應, 並對於合法的密文, 給予確認回應。我們證明若存在一個演算法解決比較型偽硬幣問題, 則對於符合三分法反應模型的馬可利斯公鑰密碼系統實作, 其相對應的攻擊演算法亦存在。更進一步地, 我們提出一個有效率的演算法解決比較型偽硬幣問題。結合前述的結論, 我們得到一個三分法反應攻擊演算法, 能花費較少的詢問次數, 便得以從密文還原明文。

關鍵字: 馬可利斯公鑰密碼系統, 反應攻擊, 三分法反應攻擊, 偽硬幣問題, 比較型偽硬幣問題.

A Trichotomy Reaction Attack on McEliece Public-Key Cryptosystem

Student: Han-Chang Liang Advisor: Dr. Rong-Jaye Chen

Institute of Computer Science and Information Engineering

National Chiao Tung University

Abstract

McEliece public-key cryptosystem is the first system combining cryptography and algebraic coding theory. In 1999, Hall *et al.* introduced the reaction attack on McEliece's and two other cryptosystems. Compared with chosen-ciphertext attacks, the reaction attack has higher feasibility. However, it requires more queries. In this thesis, we propose a trichotomy reaction oracle model. In this model, the key-owner is assumed that when a illegal ciphertext is received, he determines if the ciphtext is still decryptable and the plaintext is correct, then replies two different warning responses according to the judgement. And he replies an acknowledgement response when a legal ciphertext is received. We prove that if there is an algorithm which solves the comparative counterfeit coins problem, then there is an attack algorithm on the improper implementation which matches the trichotomy reaction oracle model. Furthermore, we design an efficient algorithm to solve the comparative counterfeit coins problem. Combined with the previous conclusion, a trichotomy reaction attack algorithm with fewer queries requirement is induced.

Key Words: McEliece public-key cryptosystem, reaction attack, trichotomy reaction attack, counterfeit coins problem, comparative counterfeit coins problem.

Contents

List of Figures

List of Tables

Chapter 1

Introduction

In 1978, McEliece proposed a public-key cryptosystem which is based on a hard problem in the coding theory, that is the Nearest Codeword Problem (NCP) [4]. The NCP is defined as follows: given a generator matrix G, a received vector r, one is asked to output the vector m that minimize $weight(mG \oplus r) = t$. The decision version of NCP is proven to be an NP-complete problem by reducing the three-dimensional matching problem [16] to it, thus NCP belongs to the class of NP-hard problems. The McEliece PKC is attractive since it combines the cryptography and algebraic coding theory.

In recent thirty years, several variants of the McEliece PKC have been proposed [27] [28] [22] [18]. We can separate them into three classes by their purpose. The first type of variants aim to reduce the encryption data redundancy, the second type of variants aim to enhance the security, and the last type of variants aim to cover the above two purposes.

In the opposite direction, several attacks of the McEliece PKC have also been proposed [25] [19] [30] [20] [6] [17] [5] [29] [13]. Some attacks [25] [19] [30] [20] [6] aim to solve the underlying nearest codeword problem. Although all of these attacks requires an exponential expected time consumption, they are constructive since they inspire the research about the decoding of general linear-codes.

Under a large amount of examination, the native McEliece PKC is considered to satisfy a security notion: one-wayness against chosen-plaintext attacks (OW-CPA). OW-CPA is said to be satisfied if all the known chosen plaintext attacks cannot recover the whole plaintext of an arbitrarily given ciphertext within a practical time. Moreover, Kobara *et al* proved in the random oracle model that after applying a proper conversion on the McEliece PKC, the modified system satisfies the strongest security notion, that is, the indistinguishability of encryption against adaptively chosen-ciphertext attacks (IND-CCA2).

However, there still exist some effective attacks which break an improper implementation or a vulnerable protocol. In 1999, Hall, Goldberg, and Schneier [13] proposed the reaction attack against several public key cryptosystems based on decoding problems and lattice problems, including the McEliece [25], Hwang-Rao [14], Ajtai-Dwork [2], and Goldreich-Goldwasser-Halevi [11] cryptosystems. In their attack against the McEliece PKC, an adversary sends the key-owner a ciphertext which may contain one or more additional error bits. In common implementations, the garbled ciphertext will cause failure in decryption or an illegal plaintext checksum. The adversary then watches the reaction of the key-owner in order to determine whether or not the ciphertext is decrypted correctly. By repeatedly apply the send-and-watch action, the adversary can obtain the plaintext.

This attack is interesting since it proved that even a strongest conversion is applied [18], an improper implementation cause the whole PKC to be vulnerable.

In Chapter 4 of this thesis, we extend their work to propose a trichotomy reaction attack to break an improper implementation of the system which matches with the reichotomy reaction oracle model. Furthermore, we establish the connection between comparative counterfeit coins puzzle and the reaction attack to recover the plaintext from ciphertext in fewer send-and-watch actions.

In the rest of this thesis, we first give a review of all variation versions of the McEliece PKC. Next, we introduce all the known attacks on the native McEliece PKC, and we proposed a improved weight-checking skill, which can be applied on the general information decoding attacks. Finally, we propose a new attack which breaks an improper implementation of the decryption procedure when the behavior of the decryption procedure matches with the trichotomy reaction oracle.

Chapter 2

McEliece PKC

McEliece Public-Key Cryptosystem (MEPKC) was first introduced by R.J. McEliece in 1978 [25]. In recent thirty years, several variants of the MEPKC have been proposed. We can separate them into three classes by their purpose. The first type of variants aim to reduce the encryption data redundancy, the second type of variants aim to enhance the security, and the last type of variants aim to cover the above two purposes. In this chapter, we introduce all of these variants. We begin with an introduction to the underlying hard problem of the MEPKC.

Nearest Codeword Problem (NCP) is an NP-hard problem; it is defined as follows.

Nearest Codeword Problem:
Input: A generator matrix G, a received vector r.
Output: The m that minimize $weight(mG \oplus r)$.

If the NCP can be solved efficiently, it implies that the McEliece PKC is broken. But breaking McEliece PKC is not as hard as solving NCP since the McEliece PKC is a special case of NCP where error weight is guaranteed to be a certain value. The hard problem that an adversary really faces is a constrained version of NCP, which can be described as follows.

Constrained Nearest Codeword Problem:
Input: A generator matrix G, a nonnegative integer t, and a received vector r consists with that there exist one and only one vector m such that $weight(mG \oplus r) = t$.
Output: The m such that $weight(mG \oplus r) = t$.

The decision version of NCP [4] is proven to be an NP-complete problem by reducing the three-dimensional matching problem [16] to it, thus NCP belongs to the class of NP-hard problems. However, the Constrained NCP have not been proved as hard as NCP.

2.1 Native McEliece PKC

The native MEPKC consists of the following three algorithms:

Key Generation algorithm:
Generate four matrices G,S,P and G'

G	$k \times n$ generator matrix of a binary Goppa code with correcting ability t.
S	$k \times k$ random binary invertible matrix.
P	$n \times n$ random permutation matrix.
G'	$G' = SGP$
Secret key:	S, G, P
Public key:	G', t

Encryption algorithm:
To encrypt a $1 \times k$ message m, one has to randomly select a $1 \times n$ error vector e with Hamming weight t. Then output the corresponding ciphertext c:
Ciphertext: $c = m \cdot G' \oplus e$

Decryption algorithm:
The key-owner first multiplies c by P^{-1},i.e. $cP^{-1} = mSG \oplus eP^{-1}$. Then he applies the efficient decoding algorithm for Goppa code to eliminate the error vector eP^{-1} and obtains the vector mS, which is multiplied by S^{-1} to obtain the message m.
Plaintext: $m = mS \cdot S^{-1}$

We can see that an instance of the MEPKC can be uniquely determined by four parameters (S, G, P, t).

In the next paragraph, we give an introduction to the Goppa code [12] [3] [24], which is suggested to used in the MEPKC by the original author.

The Goppa code is first proposed by Goppa [12] in 1970. Based on the original definition, there are many equivalent definitions of it. Here we describe the definition from [23].

Definition 2.1.1. [23] Let q be a prime power, m be an integer, $g(x)$ be a polynomial with coefficients in $GF(q^m)$, L denotes a set of all elements of $GF(q^m)$ that are not roots of $g(x)$. Then there is a Goppa code with length $|L|$ and symbol field $GF(q)$. The code is defined as the set of all vectors C that consist with the condition:

$$\sum_{\gamma\in L} \frac{C_\gamma}{z-\gamma} \equiv 0 \bmod g(z).$$

For the convenience of implementation, we usually use binary Goppa codes in the MEPKC, so we describe another definition of the Goppa code from [24]. It defines the binary Goppa codes by parity check matrix.

Definition 2.1.2. [24] Let $g(x) \in GF(2^m)[x]$ be a polynomial of degree t over the field $GF(2^m)$, $L = \{\alpha_1, \alpha_2, ..., \alpha_n\}$ be a subset of elements of $GF(2^m)$ such that $g(\alpha_i) \neq 0$. We label the coordinates of the vector $a \in (GF(2^m)^n)$ with the elements of L in the following way:

$$a = (a_{\alpha_1}, a_{\alpha_2}, \ldots, a_{\alpha_n})$$

Then, the Goppa code $\Gamma(L, g)$ is the set of binary vectors $a = (a_{\alpha_1}, a_{\alpha_2}, \ldots, a_{\alpha_n})$ such that

$$a \cdot H^T = 0$$

where

$$H = \begin{bmatrix} G(\alpha_1)^{-1} & G(\alpha_2)^{-1} & \cdots & G(\alpha_n)^{-1} \\ \alpha_1 G(\alpha_1)^{-1} & \alpha_2 G(\alpha_2)^{-1} & \cdots & \alpha_n G(\alpha_n)^{-1} \\ \alpha_1^2 G(\alpha_1)^{-1} & \alpha_2^2 G(\alpha_2)^{-1} & \cdots & \alpha_n^2 G(\alpha_n)^{-1} \\ \vdots & \vdots & \ddots & \vdots \\ \alpha_1^{t-1} G(\alpha_1)^{-1} & \alpha_2^{t-1} G(\alpha_2)^{-1} & \cdots & \alpha_n^{t-1} G(\alpha_n)^{-1} \end{bmatrix}$$

A binary Goppa code $\Gamma(L, g)$ is a $[n, k, d]$-linear code with the following three properties.

1. $k \geq n - mt$.
2. $d \geq 2deg(g') + 1$, where g' is the square-free polynomial of highest degree which divides g.
3. There exist a polynomial-time decoding algorithm which corrects up to $deg(g')$ errors.

We give a construction example of the binary Goppa code with $n = 7, k = 4, d = 3$.

Example 2.1.1. Select a polynomial $f(x) = x^3 + x + 1$ which is irreducible over $GF(2)$, then we have a corresponding field $GF(2^3)$. Select a polynomial $g(x) = x$ which has only one root over $GF(2^3)$, thus we have $L = \{\alpha^1, \alpha^2, \alpha^3, \alpha^4, \alpha^5, \alpha^6, \alpha^7\}$. By definition 2.1.2, the corresponding parity-check matrix over field $GF(2^3)$ is:

$$\begin{aligned} \mathrm{H} &= \begin{bmatrix} G(\alpha^1)^{-1} & G(\alpha^2)^{-1} & G(\alpha^3)^{-1} & G(\alpha^4)^{-1} & G(\alpha^5)^{-1} & G(\alpha^6)^{-1} & G(\alpha^7)^{-1} \end{bmatrix} \\ &= \begin{bmatrix} \alpha^6 & \alpha^5 & \alpha^4 & \alpha^3 & \alpha^2 & \alpha^1 & \alpha^0 \end{bmatrix} \end{aligned}$$

We can rewrite the 1×7 parity-check matrix over $GF(2^3)$ into a 3×7 parity-check matrix over $GF(2)$ without effect of its parity-check function:

$$\mathrm{H} = \begin{bmatrix} 1 & 1 & 1 & 0 & 1 & 0 & 0 \\ 0 & 1 & 1 & 1 & 0 & 1 & 0 \\ 1 & 1 & 0 & 1 & 0 & 0 & 1 \end{bmatrix}$$

Then we have the corresponding generator matrix:

$$\mathrm{G} = \begin{bmatrix} 1 & 0 & 0 & 0 & 1 & 0 & 1 \\ 0 & 1 & 0 & 0 & 1 & 1 & 1 \\ 0 & 0 & 1 & 0 & 1 & 1 & 0 \\ 0 & 0 & 0 & 1 & 0 & 1 & 1 \end{bmatrix}$$

Example 2.1.2. Select a polynomial $f(x) = x^4 + x + 1$ which is irreducible over $GF(2)$, then we have a corresponding field $GF(2^4)$. Select a polynomial $g(x) = x^2 + x$ which has two roots over $GF(2^4)$ (0 and 1), thus we have:

$L = \{\alpha^1, \alpha^2, \alpha^3, \alpha^4, \alpha^5, \alpha^6, \alpha^7, \alpha^8, \alpha^9, \alpha^{10}, \alpha^{11}, \alpha^{12}, \alpha^{13}, \alpha^{14}\}$. By definition 2.1.2, the corresponding parity-check matrix over field $GF(2^4)$ is:

$$\mathrm{H} = \begin{bmatrix} H_1 & | & H_2 \end{bmatrix}$$

where

$$H_1 = \begin{bmatrix} \alpha^{10} & \alpha^{5} & \alpha^{13} & \alpha^{10} & \alpha^{0} & \alpha^{11} & \alpha^{14} \\ \alpha^{11} & \alpha^{7} & \alpha^{1} & \alpha^{14} & \alpha^{5} & \alpha^{2} & \alpha^{6} \end{bmatrix}$$

and

$$H_2 = \begin{bmatrix} \alpha^{5} & \alpha^{14} & \alpha^{0} & \alpha^{7} & \alpha^{7} & \alpha^{11} & \alpha^{13} \\ \alpha^{13} & \alpha^{8} & \alpha^{10} & \alpha^{3} & \alpha^{4} & \alpha^{11} & \alpha^{12} \end{bmatrix}.$$

Then we have the corresponding parity-check matrix over $GF(2)$.

$$\mathrm{H} = \begin{bmatrix} 0 & 0 & 1 & 0 & 0 & 1 & 1 & 0 & 1 & 0 & 1 & 1 & 1 & 1 \\ 1 & 1 & 1 & 1 & 0 & 1 & 0 & 1 & 0 & 0 & 0 & 0 & 1 & 1 \\ 1 & 1 & 0 & 1 & 0 & 1 & 0 & 1 & 0 & 0 & 1 & 1 & 1 & 0 \\ 1 & 0 & 1 & 1 & 1 & 0 & 1 & 0 & 1 & 1 & 1 & 1 & 0 & 1 \\ 1 & 1 & 0 & 1 & 0 & 0 & 1 & 1 & 0 & 0 & 1 & 0 & 1 & 1 \\ 1 & 0 & 0 & 0 & 1 & 1 & 1 & 1 & 1 & 1 & 0 & 0 & 1 & 1 \\ 1 & 1 & 1 & 0 & 1 & 0 & 0 & 0 & 0 & 1 & 0 & 1 & 1 & 1 \\ 0 & 1 & 0 & 1 & 0 & 0 & 0 & 1 & 1 & 1 & 0 & 1 & 0 & 1 \end{bmatrix}$$

2.2 Variants for Reducing Data Redundancy

A representative work of variants for reducing the data redundancy was proposed by Sun [27] [28]. The main idea of these variants is to construct the error vector by partial plaintext or its variations instead of selecting the error vector randomly. Because the error vector is a part of plaintext or can be converted into partial plaintext, we can proportion a part of plaintext to the error vector. In this way, more information(longer plaintext) can be carried by a fixed length ciphertext. The author proposed five variants; we describe his variant III in this section.

Variant 2.2.1:

Key Generation algorithm:
Generate four matrices G,S,P,G' and a function g.

g	An invertible function: $\{0,1\}^{\lfloor log_2\binom{n}{t}\rfloor} \leftrightarrow \{v \mid v \in \{0,1\}^n, weight(v)=t\}$.
G	$k \times n$ generator matrix of a binary Goppa code with correcting ability: t.
S	$k \times k$ random binary invertible matrix.
P	$n \times n$ random permutation matrix.
G'	$G' = SGP$
Secret key:	S, G, P
Public key:	G', g, t

Encryption algorithm:
To encrypt a message $m = m_a \| m_b$, where $|m_a| = k$, $|m_b| = \lfloor log_2\binom{n}{t}\rfloor$, one has to compute an error vector $e = g(m_b)$. Then output the corresponding ciphertext c:
Ciphertext: $c = m \cdot G' \oplus e$

Decryption algorithm:
The key-owner recover the partial plaintext m_a by applying the original decryption algorithm of MEPKC. Then, the error vector can be obtained by $e = m_aG' \oplus c$. The key owner can easily extract m_b by $m_b = g^{-1}(e)$.
Plaintext: $m = m_a \| m_b$

This variant provides additional $\lfloor log_2\binom{n}{t}\rfloor$-bits information capacity.

2.3 Variants for Enhancing Security

More variants can be found for enhancing security. For example, the variants proposed by Sun [28] [27] which resist the message-resend and related-message attack [5], the variant proposed by Loidreau [22] which resists the chosen-plaintext attacks and the Kobara *et al*'s variant [18] which resists the chosen-ciphertext attacks. We will introduce these variants in this section.

2.3.1 Sun's Variants

In the previous section, we have introduced the Sun's variant for reducing the encryption data-redundancy. Actually, the author provided four other variants in his paper [27]. His variant I and II successfully resist the message-resend attack. Moreover, his variant IV and V meet the dual purposes of reducing the encryption data-redundancy and enhancing security. The main idea of these variants is to shield the plaintext by a hash-value of the error vector. In this way, resend-messages will not cause the leak of error position information (for detail of the message-resend attack see Section 3.3). We give an introduction to variant I and IV here.

Variant 2.3.1:

Key Generation algorithm:
Generate four matrices G,S,P,G' and a function g.

h	A hash function: $\{v \vert v \in \{0,1\}^n, weight(v) = t\} \to \{0,1\}^k$.
G	$k \times n$ generator matrix of a binary Goppa code with correcting ability t.
S	$k \times k$ random binary invertible matrix.
P	$n \times n$ random permutation matrix.
G'	$G' = SGP$
Secret key:	S, G, P
Public key:	G', h, t

Encryption algorithm:
To encrypt a $1 \times k$ message m, one has to randomly select a $1 \times n$ error vector e with Hamming weight t. Then output the corresponding ciphertext c:
Ciphertext: $c = (m \oplus h(e))G' \oplus e$

Decryption algorithm:
The key-owner recover the corrupt plaintext $m' = m \oplus h(e)$ by applying the original decryption algorithm of MEPKC. Then, the error vector can be obtained by $e = m'G' \oplus c$. The key-owner can easily extract correct m by $m = m' \oplus h(e)$.
Plaintext: $m = m' \oplus h(e)$

This variant successful resist the message-resent attack. When a message first encrypted as ciphertext c_1 and re-encrypted as a different ciphertext c_2, the adversary compute $c_1 \oplus c_2$, then he will obtain $(h(e_1) \oplus h(e_2))G' \oplus e_1 \oplus e_2$ which leaks no information about e_1 and e_2 since the covering of $(h(e_1) \oplus h(e_2))G'$.

Variant 2.3.2:

Key Generation algorithm:
Generate four matrices G,S,P,G' and a function g.

q	A system parameter which denotes the length of random vector. q consists with $0 \leq q \leq \lfloor log_2\binom{n}{t} \rfloor$.
g	An invertible function: $\{0,1\}^{\lfloor log_2\binom{n}{t} \rfloor} \leftrightarrow \{v \mid v \in \{0,1\}^n, weight(v) = t\}$.
h	A hash function: $\{v \mid v \in \{0,1\}^n, weight(v) = t\} \rightarrow \{0,1\}^n$.
G	$k \times n$ generator matrix of a binary Goppa code with correcting ability: t.
S	$k \times k$ random binary invertible matrix.
P	$n \times n$ random permutation matrix.
G'	$G' = SGP$
Secret key:	S, G, P
Public key:	G', q, g, h, t

Encryption algorithm:
To encrypt a message $m = m_a \| m_b$, where $|m_a| = k$ and $|m_b| = \lfloor log_2\binom{n}{t} \rfloor - q$.
One has to randomly select a $1 \times q$ vector r, then compute an error vector $e = g(r \| m_b)$.
Output the corresponding ciphertext c:
Ciphertext: $c = (m_a \oplus h(e))G' \oplus e$

Decryption algorithm:
The key-owner recover the corrupt plaintext $m'_a = m_a \oplus h(e)$ by applying the original decryption algorithm of MEPKC. Then, the error vector can be obtained by $e = m'_a G' \oplus c$. The key-owner can easily extract correct m_a by $m_a = m'_a \oplus h(e)$.
Final, m_b can be obtained by applying the inverse function of g, that is $(r \| m_b) = g^{-1}(e)$.
Plaintext: $m = ma \| m_b$

This variant not only resists the message-resend attack, but also reduces the encryption data-redundancy. Because of the additional message m_b, this variant provides additional $\lfloor log_2\binom{n}{t} \rfloor - q$ bits information capacity.

2.3.2 Loidreau's Variant

In 2000, Loidreau proposed a modified version of the MEPKC using the modification bases on the Frobenius automorphism. The main idea of his modification is that it can enlarge the weight of the error vector to resist the chosen-plaintext attack. In the original MEPKC, the randomly-selected error vector e must consist with $weight(e) \leq t$ to make sure that the key-owner can decrypt correctly, but in the Loidreau's modification, we can select an error vector which satisfies $weight(e) > t$ if the vector e is a *t-Tower Decodable Vector*.

Consider a Goppa code $\Gamma(L, g)$ over F_{2^m} where $L = (\alpha_1, \ldots, \alpha_n)$ and g denotes the Goppa polynomial. If all the coefficients of g are in a subfield F_{2^s} of F_{2^m}, then the Frobenius automorphism function $\rho : \{0,1\}^{2^m} \rightarrow \{0,1\}^{2^m}$ is defined as follows.

$\forall c = (c_{\alpha_1}, \ldots, c_{\alpha_n}) \in \{0,1\}^{2^m}, \rho(c) = (c_{\rho'(\alpha_1)}, \ldots, c_{\rho'(\alpha_n)}) \in \{0,1\}^{2^m}$

where $\rho'(x) = x^{2^s}$.

This variant is based on the fact that the Goppa code $\gamma(L, g)$ is invariant under the Frobenius automorphism, that if c is a codeword of the Goppa code $Gamma(L, g)$, then $\rho(c)$ is also a codeword of $Gamma(L, g)$.

Example 2.3.1. Consider a Goppa code $\Gamma(L, g)$ over $GF(2^4)$, where $g = x^3 + x + 1$ and $L = (0, \alpha^0, \alpha^1, \alpha^2, \alpha^4, \alpha^8, \alpha^{12}, \alpha^3, \alpha^6, \alpha^9, \alpha^5, \alpha^{10}, \alpha^{11}, \alpha^{13}, \alpha^{14}, \alpha^7)$. We can see that all the coefficients of g are over the subfield $GF(2^1)$ of $GF(2^4)$, thus we have $s = 1 and \rho(x) = x^{2^1} = x^2$.

The generator matrix of $\Gamma(L, g)$ is:

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

All the codewords of the $\Gamma(L, g)$ are:

$$\begin{array}{rl}
c_1 = & [\ 0\ 0\ 0\ 0\ 0\ 0\ 0\ 0\ 0\ 0\ 0\ 0\ 0\ 0\ 0\ 0\ ] \\
c_2 = & [\ 0\ 0\ 0\ 1\ 0\ 1\ 1\ 1\ 0\ 0\ 0\ 1\ 0\ 1\ 0\ 1\ ] \\
c_3 = & [\ 0\ 0\ 1\ 0\ 1\ 0\ 0\ 0\ 1\ 1\ 1\ 0\ 1\ 0\ 1\ 0\ ] \\
c_4 = & [\ 0\ 0\ 1\ 1\ 1\ 1\ 1\ 1\ 1\ 1\ 1\ 1\ 1\ 1\ 1\ 1\ ] \\
c_5 = & [\ 0\ 1\ 0\ 0\ 0\ 0\ 1\ 1\ 1\ 1\ 0\ 0\ 1\ 1\ 1\ 1\ ] \\
c_6 = & [\ 0\ 1\ 0\ 1\ 0\ 1\ 0\ 0\ 1\ 1\ 0\ 1\ 1\ 0\ 1\ 0\ ] \\
c_7 = & [\ 0\ 1\ 1\ 0\ 1\ 0\ 1\ 1\ 0\ 0\ 1\ 0\ 0\ 1\ 0\ 1\ ] \\
c_8 = & [\ 0\ 1\ 1\ 1\ 1\ 1\ 0\ 0\ 0\ 0\ 1\ 1\ 0\ 0\ 0\ 0\ ] \\
c_9 = & [\ 1\ 0\ 0\ 0\ 0\ 0\ 0\ 0\ 0\ 0\ 1\ 1\ 1\ 1\ 1\ 1\ ] \\
c_{10} = & [\ 1\ 0\ 0\ 1\ 0\ 1\ 1\ 1\ 0\ 0\ 1\ 0\ 1\ 0\ 1\ 0\ ] \\
c_{11} = & [\ 1\ 0\ 1\ 0\ 1\ 0\ 0\ 0\ 1\ 1\ 0\ 1\ 0\ 1\ 0\ 1\ ] \\
c_{12} = & [\ 1\ 0\ 1\ 1\ 1\ 1\ 1\ 1\ 1\ 1\ 0\ 0\ 0\ 0\ 0\ 0\ ] \\
c_{13} = & [\ 1\ 1\ 0\ 0\ 0\ 0\ 1\ 1\ 1\ 1\ 1\ 1\ 0\ 0\ 0\ 0\ ] \\
c_{14} = & [\ 1\ 1\ 0\ 1\ 0\ 1\ 0\ 0\ 1\ 1\ 1\ 0\ 0\ 1\ 0\ 1\ ] \\
c_{15} = & [\ 1\ 1\ 1\ 0\ 1\ 0\ 1\ 1\ 0\ 0\ 0\ 1\ 1\ 0\ 1\ 0\ ] \\
c_{16} = & [\ 1\ 1\ 1\ 1\ 1\ 1\ 0\ 0\ 0\ 0\ 0\ 0\ 1\ 1\ 1\ 1\ ]
\end{array}$$

All the transformed codewords are:

$$
\begin{array}{rcccccccccccccccccc}
\rho(c_1) = & [& 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 &] \\
\rho(c_2) = & [& 0 & 0 & 1 & 0 & 1 & 0 & 0 & 0 & 1 & 1 & 1 & 0 & 1 & 0 & 1 & 0 &] \\
\rho(c_3) = & [& 0 & 0 & 0 & 1 & 0 & 1 & 1 & 1 & 0 & 0 & 0 & 1 & 0 & 1 & 0 & 1 &] \\
\rho(c_4) = & [& 0 & 0 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 &] \\
\rho(c_5) = & [& 0 & 1 & 0 & 0 & 0 & 0 & 1 & 1 & 1 & 1 & 0 & 0 & 1 & 1 & 1 & 1 &] \\
\rho(c_6) = & [& 0 & 1 & 1 & 0 & 1 & 0 & 1 & 1 & 0 & 0 & 1 & 0 & 0 & 1 & 0 & 1 &] \\
\rho(c_7) = & [& 0 & 1 & 0 & 1 & 0 & 1 & 0 & 0 & 1 & 1 & 0 & 1 & 1 & 0 & 1 & 0 &] \\
\rho(c_8) = & [& 0 & 1 & 1 & 1 & 1 & 1 & 0 & 0 & 0 & 0 & 1 & 1 & 0 & 0 & 0 & 0 &] \\
\rho(c_9) = & [& 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 1 & 1 & 1 & 1 & 1 &] \\
\rho(c_{10}) = & [& 1 & 0 & 1 & 0 & 1 & 0 & 0 & 0 & 1 & 1 & 0 & 1 & 0 & 1 & 0 & 1 &] \\
\rho(c_{11}) = & [& 1 & 0 & 0 & 1 & 0 & 1 & 1 & 1 & 0 & 0 & 1 & 0 & 1 & 0 & 1 & 0 &] \\
\rho(c_{12}) = & [& 1 & 0 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 0 & 0 & 0 & 0 & 0 & 0 &] \\
\rho(c_{13}) = & [& 1 & 1 & 0 & 0 & 0 & 0 & 1 & 1 & 1 & 1 & 1 & 1 & 0 & 0 & 0 & 0 &] \\
\rho(c_{14}) = & [& 1 & 1 & 1 & 0 & 1 & 0 & 1 & 1 & 0 & 0 & 0 & 1 & 1 & 0 & 1 & 0 &] \\
\rho(c_{15}) = & [& 1 & 1 & 0 & 1 & 0 & 1 & 0 & 0 & 1 & 1 & 1 & 0 & 0 & 1 & 0 & 1 &] \\
\rho(c_{16}) = & [& 1 & 1 & 1 & 1 & 1 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 1 & 1 & 1 &]
\end{array}
$$

Under the action of the Frobenius automorphism, several orbits in the field $GF(2^m)$ are created. That is, there are several group of bits which only change their positions with others inside the group after applying the Frobenius automorphism one or more times. The total number of orbits can be evaluated. For an extension field $F_{(2^s)^s_1}$ of F_{2^s} of prime degree s_1, the $\rho : \{0,1\}^{2^m} \rightarrow \{0,1\}^{2^m}$ operation creates $N_{s_1} = ((2^s)^{s_1} - 2^s)/s_1$ orbits of size s_1 and $N_1 = 2^s$ orbits of size 1.

Example 2.3.2. Using parameters in the previous example, the Frobenius automorphism ρ creates the following six orbits.

Size 1

$$(\underline{c_{\alpha_1}}, c_{\alpha_2}, c_{\alpha_3}, c_{\alpha_4}, c_{\alpha_5}, c_{\alpha_6}, c_{\alpha_7}, c_{\alpha_8}, c_{\alpha_9}, c_{\alpha_{10}}, c_{\alpha_{11}}, c_{\alpha_{12}}, c_{\alpha_{13}}, c_{\alpha_{14}}, c_{\alpha_{15}}, c_{\alpha_{16}})$$
$$(c_{\alpha_1}, \underline{c_{\alpha_2}}, c_{\alpha_3}, c_{\alpha_4}, c_{\alpha_5}, c_{\alpha_6}, c_{\alpha_7}, c_{\alpha_8}, c_{\alpha_9}, c_{\alpha_{10}}, c_{\alpha_{11}}, c_{\alpha_{12}}, c_{\alpha_{13}}, c_{\alpha_{14}}, c_{\alpha_{15}}, c_{\alpha_{16}})$$

Size 2

$$(c_{\alpha_1}, c_{\alpha_2}, c_{\alpha_3}, c_{\alpha_4}, c_{\alpha_5}, c_{\alpha_6}, c_{\alpha_7}, c_{\alpha_8}, c_{\alpha_9}, c_{\alpha_{10}}, \underline{c_{\alpha_{11}}, c_{\alpha_{12}}}, c_{\alpha_{13}}, c_{\alpha_{14}}, c_{\alpha_{15}}, c_{\alpha_{16}})$$

Size 4

$$(c_{\alpha_1}, c_{\alpha_2}, \underline{c_{\alpha_3}, c_{\alpha_4}, c_{\alpha_5}, c_{\alpha_6}}, c_{\alpha_7}, c_{\alpha_8}, c_{\alpha_9}, c_{\alpha_{10}}, c_{\alpha_{11}}, c_{\alpha_{12}}, c_{\alpha_{13}}, c_{\alpha_{14}}, c_{\alpha_{15}}, c_{\alpha_{16}})$$
$$(c_{\alpha_1}, c_{\alpha_2}, c_{\alpha_3}, c_{\alpha_4}, c_{\alpha_5}, c_{\alpha_6}, \underline{c_{\alpha_7}, c_{\alpha_8}, c_{\alpha_9}, c_{\alpha_{10}}}, c_{\alpha_{11}}, c_{\alpha_{12}}, c_{\alpha_{13}}, c_{\alpha_{14}}, c_{\alpha_{15}}, c_{\alpha_{16}})$$
$$(c_{\alpha_1}, c_{\alpha_2}, c_{\alpha_3}, c_{\alpha_4}, c_{\alpha_5}, c_{\alpha_6}, c_{\alpha_7}, c_{\alpha_8}, c_{\alpha_9}, c_{\alpha_{10}}, c_{\alpha_{11}}, c_{\alpha_{12}}, \underline{c_{\alpha_{13}}, c_{\alpha_{14}}, c_{\alpha_{15}}, c_{\alpha_{16}}})$$

Example 2.3.3. Using parameters in the previous example, we apply the Frobenius automorphism on a vector $(c_{\alpha_1}, c_{\alpha_2}, \cdots, c_{\alpha_{16}}) = (0, 0, 1, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0)$ for four times. Let ρ^i denote the ρ function acts for i times, that is, $\rho^i(x) = \rho(\rho^{i-1}(x))$. In the example, we focus on the first orbit of size 4 $(c_{\alpha_3}, c_{\alpha_4}, c_{\alpha_5}, c_{\alpha_6})$.

$$\begin{aligned}
(c_{\alpha_3}, c_{\alpha_4}, c_{\alpha_5}, c_{\alpha_6}) &= (1, 1, 0, 0) \\
\rho^1(c_{\alpha_3}, c_{\alpha_4}, c_{\alpha_5}, c_{\alpha_6}) &= (1, 0, 0, 1) \\
\rho^2(c_{\alpha_3}, c_{\alpha_4}, c_{\alpha_5}, c_{\alpha_6}) &= (0, 0, 1, 1) \\
\rho^3(c_{\alpha_3}, c_{\alpha_4}, c_{\alpha_5}, c_{\alpha_6}) &= (0, 1, 1, 0) \\
\rho^4(c_{\alpha_3}, c_{\alpha_4}, c_{\alpha_5}, c_{\alpha_6}) &= (1, 1, 0, 0)
\end{aligned}$$

By the definition of the Frobenius automorphism, there exist several vectors consist with following three properties. The set collecting these vectors is named t-tower decodeable vector in [22].

Definition 2.3.1. [22] t-tower decodeable vector z is a word of length n having the following properties.

Larger weight:	$wt(z) > t$.
Reducibility:	There exist a linear combination $b_0, b_1, \ldots \in \{0, 1\}$ such that $wt(z') \leq t$, where $z' = \sum_i b_i \cdot \rho^i(z)$.
Recoverability:	z is recoverable from the reduced z'.

Based on the definition of t-tower decodeable vector and its reducible property, the author provided the following modified version of the MEPKC.

Variant 2.3.3:

Key Generation algorithm:
Generate four matrices G,S ,P,G' and orbits set O

G	$k \times n$ generator matrix of a binary Goppa code $\Gamma(L, g)$ with correcting ability t.
S	$k \times k$ random binary invertible matrix.
P	$n \times n$ random permutation matrix.
G'	$G' = SGP$
O	The orbits created by Frobenius automorphism *rho*
Secret key:	S, G, P, L
Public key:	G', t, O

Encryption algorithm:
To encrypt a $1 \times k$ message m, one has to randomly select $\lfloor t/2 \rfloor$ orbits in O.
For each selected orbit, select three bits and mark them as error bits. Collect all the error bits to form an t-tower decodeable vector E.
Ciphertext: $c = m \cdot G' \oplus E$

Decryption algorithm:
The key-owner first calculate two vectors
$f_1(c) = c + \rho(c) + \rho^2(c)$ and $f_2(c) = c + \rho^2(c) + \rho^3(c)$, one of them will be decrypt correctly. Suppose a temporary message m' is obtained by decryption, Calculate $e = (m' \cdots G') \oplus c$, and obtain its corresponding enlarged vector E (It's possible since the t-tower decodeable vector E is recoverable by definition). Eliminate the error bits by $c' = c \oplus E$, then one can apply the information-set decoding to obtain the corresponding message m.
Plaintext: $m = c'_k \cdot G'^{-1}_k$

2.3.3 Kobara and Imai's Variant

KOBARA and IMAI proposed a variant of the MEPKC in 2002 [18]. Under the random oracle assumption, their variant is proved indistinguishable against the adaptive chosen-ciphertext attacks (CCA2) and the adaptive chosen-plaintext attacks.

Before describing their variant, we give an introduction to the necessary notations.

$Hw(x)$	The hamming weight of a binary string x.
$C(n,t)$	The number of combinations taking t out of n elements.
$Prep(m)$	An invertable preprocessing procedure of m.
$Hash(x)$	One way hash function maps arbitrary length binary string to a fixed length binary string.
$Conv(z')$	Bijective function maps from integer $z' \in Z_{C(n,t)}$ to $\{e \mid e \in \{0,1\}^n, weight(e) = t\}$.
$Gen(x)$	A pseudo random sequence generated by a cryptographically secure pseudo random sequence generator with input seed x.
$Msb_i(x)$	The left i bits of x.
$Const$	A public constant.
$\varepsilon^{McEliece}(m,e)$	The function that encrypts plaintext m with error vector e by native MEPKC.
$D^{McEliece}(c)$	The function that decrypts ciphertext c to obtain the plaintext m and the error vector e hidden in c.

Their modified version of the MEPKC is shown below.

Variant 2.3.4:

Encryption algorithm:

$$\begin{aligned}
r &\in_R Z_{Length(Hash(x))} \\
m' &= Prep(m) \\
y_1 &= Gen(r) \oplus (m' \| Const) \\
y_2 &= r \oplus Hash(y_1) \\
(y_5 \| y_4 \| y_3) &= y_1 \\
z' &= y_4 + (Hw(y_3) \bmod 2)\dot{2}^{\lfloor log_2 n \rfloor} \bmod C(n,t) \\
z &= Conv(z') \\
c &= y_2 \| y_5 \| \varepsilon^{McEliece}(y_3, z)
\end{aligned}$$

Decryption algorithm:

$$\begin{aligned}
(y_2 \| y_5) &= Msb_{Len(c)-n}(c) \\
z', y_3 &= D^{McEliece}(Lsb_n(c)) \\
z &= Conv^{-1}(z') \\
y_4 &= z - (Hw(y_3 \bmod 2)\dot{2}^{\lfloor log_2 n \rfloor} \bmod C(n,t) \\
&\quad \text{If } y_4 \leq 2^{\lfloor log_2 n \rfloor}, \text{ reject } c. \\
\text{Otherwise} & \\
y_1 &= (y_5 \| y_4 \| y_3) \\
r &= y_2 \oplus Hash(y_1) \\
(m' \| Const') &= y_1 \oplus Gen(r) \\
&\quad \text{If } Const' = Const \\
&\quad \text{return } m = Prep^{-1}(m') \\
\text{Otherwise} &\quad \text{reject } c
\end{aligned}$$

The detailed proof of IND-CPA and IND-CCA2 security refer to [18].

Chapter 3

Attacks to McEliece PKC

After the MEPKC was proposed, several attacks are also proposed to break the cryptosystem. However, as we mentioned in the previous chapter, the variant proposed by Kobara *et al.* is a provable public-key cryptosystem under random oracle model. When this variant is applied, all the chosen-plaintext attacks and chosen-ciphertext attacks will not work anymore. Although some of these attacks may not work on the latest variant of the MEPKC, they are still constructive since they inspire the research about the decoding of general linear-codes. In this chapter, we will review many known attacks to MEPKC.

3.1 Generalized Information-Set-Decoding Attack

This type of attacks (we call it the GISD for simplicity) try to correct the error bits in the ciphertext; once they are successful, the codeword without error bits mask $m \cdot G'$ is obtained and m can be simply recovered by information-set-decoding. The GISD problem can be formally described as input a generator matrix G of a general linear-code with error correcting ability t, and a received vector c with error-bits fewer than t, one must output the vector m such that $weight(mG' \oplus c) \leq t$ (since G' has error correcting ability t by definition, m must be unique). This type of attacks were first introduced by McEliece [25] and several improvements of it were proposed [19] [30]. In this section, we will make a review of three representative attacks of this type. We begin with an introduction to the information-set-decoding.

3.1.1 Information-set-decoding

Consider a general $[n, k]$-linear code with generator matrix G, given a received vector c without any error bits, then we can recover the corresponding message m (that is, $m \cdot G = c$) by the following procedure.

Information-Set-Decoding Algorithm

Input: A $k \times n$ generator matrix G and a received vector c with length n.
Output: The vector m such that $m \cdot G = c$.

```
1.  Select $k$ columns of $G$, such that the selected columns are linearly-independent.
2.  Collect the selected columns to form a new $k \times k$ matrix $G_k$.
3.  Collect the corresponding $k$ bits of $c$ to form a new vector $c_k$.
4.  $m = c_k \cdot G_k^{-1}$.
```

Example 3.1.1. Given a $[10, 5]$-linear code with generator matrix:

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

and a received vector $c = \begin{bmatrix} 1 & 0 & 1 & 1 & 1 & 0 & 0 & 0 & 0 & 1 \end{bmatrix}$. We select 1 - 5 columns, the selected columns are linear-independent clearly. Thus we have:

$$G_k^{-1} = G_k = \begin{bmatrix} 1 & 0 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 0 & 1 \end{bmatrix}$$

The corresponding $c_k = \begin{bmatrix} 1 & 0 & 1 & 1 & 1 \end{bmatrix}$, by $m = c_k \cdot G_k^{-1}$, we have $m = \begin{bmatrix} 1 & 0 & 1 & 1 & 1 \end{bmatrix}$.

3.1.2 McEliece's Decoding Algorithm

This attack was evoked by McEliece himself in his original paper [25]. The basic idea of this attack is to randomly select k columns in the public generator matrix G' and hope that the corresponding positions of ciphertext are error-free. Once the k error-free positions are selected luckily, the adversary can apply the information-set-decoding to recover the plaintext m. The following algorithm combines the idea of McEliece, Lee and Brickell. Lee *et al* proposed a systematic method to check whether the guess is correct or not (whether the selected k positions are error-free). They also proposed a generalized method which can recover the plaintext when there are small amount of errors in the selected k positions.

McEliece's Decoding Algorithm with parameter j

```
Input: A $k \times n$ generator matrix $G'$ and a received vector $c$ with length $n$.
Output: The vector $m$ such that $weight(m \cdot G' \oplus c) \leq t$.

1. Randomly select $k$ columns of $G'$, such that the selected columns are
   linearly-independent.
2. Collect the selected columns to form a new $k \times k$ matrix $G'_k$.
3. Collect the corresponding $k$ bits of $c$ to form a new vector $c_k$.
4. Calculate $G'^{-1}_k \cdot G'$ and $c \oplus c_k(G'^{-1}_k \cdot G')$.
5. Choose an unused $k$-bit error pattern $e_k$ with less or equal to $j$ ones.
   If $(c \oplus c_k G'^{-1}_k G') \oplus e_k(G'^{-1}_k G')$ has weight $t$ or less, then
   stop and return $(m = c_k \oplus e_k) \cdot G^{-1}_k$.
6. If still exist unused $k$-bit error pattern, goto step 5.
   Otherwise goto step 1.
```

We describe the expected running time of this algorithm by the following theorem.

Theorem 3.1.1. [19] Let W_j denote the expected value of bitwise operations required by McEliece's decoding algorithm with parameter j, then $W_j = T_j(M(k,n) + N_j \times V_{n,k,t,j})$, where

$$T_j = \frac{1}{\sum_{i=0}^{j} \binom{t}{i}\binom{n-t}{k-i} / \binom{n}{k}},$$

$$N_j = \sum_{i=0}^{j} \binom{k}{i},$$

$$V_{n,k,t,j} = j/2 \times (n-k)$$

and $M(k,n)$ represent the running time for applying the Gaussian elimination on a $k \times n$ matrix.

Proof. The T_j is the expected iterations of step 1 - 6, and the N_j is an upper bound of the iterations of the inner-loop step 5 - 6. $M(k,n)$ is an approximation of the running time of step 4 and $V_{n,k,t,j}$ is the number of bit-wise operations required by step 5. To check whether $(c \oplus c_k G_k'^{-1} G') \oplus e_k(G_k'^{-1} G')$ has weight t or less, we can calculate the last $n-k$ bits of $e_k(G_k'^{-1} G')$ and compare it with the last $n-k$ bits of $c \oplus c_k(G_k'^{-1} G')$ to see if the hamming-distance of them is less than $t - weight(e_k)$. The average number of bit-wise operation required by calculating last $n-k$ bits of $e_k(G_k'^{-1} G')$ is $j/2 \times (n-k)$ since e_k have average weight $j/2$, this gives $V_{n,k,t,j} = j/2 \times (n-k)$ (the cost of the hamming-distance checking is relatively low when comparing with the cost of vector addition, so it is ignored). □

3.1.3 Tilburg's Decoding Algorithm

Tilburg [30] proposed an improvement of decoding attack which reduces the running time of the algorithm substantially. The improved algorithm reduced the cost of the validation step (step 5 in previous algorithm) and the cost of the matrix operations (step 4 in previous algorithm). The main idea of its first improvement is to permute the selected columns to the front of the matrix, so that the validation should only make on the last $n-k$ bits. Their work is famous due to their second improvement. The main idea of the second improvement is to select k columns with only one different column between the previous selection. This selecting strategy speeds-up the Gaussian elimination when calculating G_k^{-1} in step 4. We believe that the idea comes from the simplex algorithm for linear programming problem.

The key idea of their improvement can be described by the following theorem.

Theorem 3.1.2. [30] Given a generator matrix G' and a received vector $c = mG' \oplus e$ where $weight(e) \leq t$. Consider a randomly selected $n \times n$ permutation matrix P' such that G' can be written as $G' = S'\,[I|A]\,P'^{-1}$ and cP' can be written as $cP' = mS'\,[I|A] \oplus eP'$. Then $weight(FKB(eP')) = 0$ if and only if $weight(FKB(cP')\,[I|A] \oplus cP') \leq t$, where $FKB(a_1, a_2, \ldots, a_n) =$

$(a_1, a_2, \ldots, a_k)$.

Proof.

We first prove that the left-hand-side of the statement implies the right-hand-side:

Since $weight(FKB(eP')) = 0$, we have $FKB(cP')S'^{-1} = m \cdots (1)$. By definition of c, we also know that $weight(mG'P' \oplus cP') = weight(eP') \leq t$. Substitute m by equation (1), we have:

$$weight(FKB(cP')S'^{-1}S'\,[I|A] \oplus cP') \leq t$$

$$\Rightarrow weight(FKB(cP')\,[I|A] \oplus cP') \leq t.$$

Then we show that the right-hand-side of the statement implies the left-hand-side to complete the proof:

Since $weight(FKB(cP')\,[I|A] \oplus cP') \leq t$, we have $weight(FKB(cP')S'^{-1}S'\,[I|A] \oplus cP') \leq t$. Thus:

$$weight(FKB(cP')S'^{-1}G'P' \oplus cP') \leq t$$

$$\Rightarrow FKB(cP')S'^{-1} = m$$

$$\Rightarrow FKB(cP') \text{ is error-free, } weight(FKB(eP')) = 0.$$

□

The improved algorithm is represented as follows.

Tilburg's Decoding Algorithm

Input: A $k \times n$ generator matrix G' and a received vector c with length n.
Output: The vector m such that $weight(m \cdot G' \oplus c) \leq t$.

1. Initial z=1.
2. Randomly decompose G' such that $G' = S_z\,[I|A_z]\,P_z^{-1}$, and calculate the corresponding $c_z = cP_z$.
3. Check if it holds that $weight(KFB(c_z)\,[I|A_z] \oplus c_z) \leq t$.
   This can be done by only checking the last $n-k$ bits. If it holds, goto step 7 for final recovering stage.
4. Produce a random permutation P_{z+1} that swaps one column, say i, from the selected ones (the I part of the matrix $[I|A_z]$) and for one column, say j, from the unselected ones (the A_z part of the matrix $[I|A_z]$).
5. If column j has not an **1** on the i-th row, goto step 4. Otherwise goto step 6.
6. Let P_{z+1} denote the permutation selected in previous step, decompose the matrix $[I|A_z]$ into $S_{z+1}\,[I|A_{z+1}]\,P_{z+1}$, and calculate the corresponding $c_{z+1} = c_zP_{z+1}$.
   Increase z by 1, goto step 3.
7. Now, the c_z is error-free in the first k bits, and $c_z = cP_1P_2\dots P_z$. Thus we can know the corresponding error-free bits in the original c, then the plaintext m can be obtained by a information-set-decoding.

3.1.4 Improvements of Decoding Attack

We propose two improvements in this section. We begin with an introduction to the first improvement, which aims to decrease the time comsumption of the guess-verification stage.

The structure of the McEliece's decoding algorithm can be described by two nested loops: an outer-loop and an inner-loop. The outer-loop which repeats selecting different set of k columns till the selection luckily has few error bits in the ciphertext. And the inner-loop repeats guessing then verifying the error vector hidden in the k columns selected by outer-loop, till a guess is verified to be correct or all possible k-vectors that have weight less than a threshold are examined to be wrong guesses. In the analysis of McEliece's decoding algorithm with parameter j (see Theorem 3.1.1), we use T_j to denote the expected iterations of the outer-loop and use N_j to denote the expected iterations of the inner-loop. Note that the verification should be applied in each iteration of the inner-loop and its cost is non-negligible, thus we use $V_{n,k,t,j}$ to denote the bit-wise operations of it and take it into consideration in further discussion. According to the analysis from Theorem 3.1.1, the bit-wise operations required by McEliece's decoding algorithm can be approximated by $T_j \times N_j \times V_{n,k,t,j}$. In the following sections, we propose an improvement to significantly reduce $V_{n,k,t,j}$ and slightly increase T_j, thus the total number of bit-wise operations requirement is reduced. In this chapter, we only show how to apply the improvement in the McEliece's decoding algorithm, but actually the idea of improvement can be applied to all other decoding algorithms with the same nested loops structure.

Recall the McEliece's decoding algorithm:

```
McEliece's Decoding Algorithm with parameter $j$

Input: A $k \times n$ generator matrix $G'$ and a received vector $c$ with length $n$.
Output: The vector $m$ such that $weight(m \cdot G' \oplus c) \leq t$.

1.  Randomly select $k$ columns of $G'$, such that the selected columns are
    linearly-independent.
2.  Collect the selected columns to form a new $k \times k$ matrix $G'_k$.
3.  Collect the corresponding $k$ bits of $c$ to form a new vector $c_k$.
4.  Calculate $G'^{-1}_k \cdot G'$ and $c \oplus c_k(G'^{-1}_k \cdot G')$.
5.  Choose an unused $k$-bit error pattern $e_k$ with less or equal to $j$ ones.
    If $(c \oplus c_k G'^{-1}_k G') \oplus e_k(G'^{-1}_k G')$ has weight $t$ or less, then
    stop and return $(m = c_k \oplus e_k) \cdot G^{-1}_k$.
6.  If still exist unused $k$-bit error pattern, goto step 5.
    Otherwise goto step 1.
```

The step 1-6 is the so-called outer-loop and the step 5-6 is the inner-loop. According to the analysis given by Theorem 3.1.1, the bit-wise operations required by verification in each iteration of the inner-loop is approximated to $V_{n,k,t,j} = j/2 \times (n-k)$. Our improvement introduces a probability method for verification which makes the expected number of bit-wise operations requirement be $p(j/2 \times (n-k)) + (1-p)(j/2 \times \epsilon)$, where p is a small probability and ϵ is a parameter of the algorithm.

Observe that when we make a bad selection in step 1 (that is, there exists more than j errors in the selected columns), we still have to spend so much time in the inner-loop and finally realize that the selection in step 1 is bad. If we can realize it earlier that a bad selection has made in step 1, then a large amount of redundancy check can be skipped. The main idea of our improvement

```
Modified McEliece's Decoding Algorithm with parameter $(j, \epsilon)$

Input: A $k \times n$ generator matrix $G'$ and a received vector $c$ with length $n$.
Output: The vector $m$ such that $weight(m \cdot G' \oplus c) \leq t$.

1. Randomly select $k$ columns of $G'$, such that the selected columns are
   linearly-independent.
2. Collect the selected columns to form a new $k \times k$ matrix $G'_k$.
3. Collect the corresponding $k$ bits of $c$ to form a new vector $c_k$.
4. Calculate $G'^{-1}_k \cdot G'$ and $c \oplus c_k(G'^{-1}_k \cdot G')$.
5. Choose an unused $k$-bit error pattern $e_k$ with less or equal to $j$ ones.
6. Randomly select $\epsilon$ columns in the last $n-k$ columns of $G'^{-1}_k G'$, let $C$
   denote the indexes of the selected columns.
7. Calculate $c \oplus (c_k G'^{-1}_k G') \oplus e_k(G'^{-1}_k G')$ bit-by-bit and ignore the
   unselected columns to form a vector $D_C$ with length $\epsilon$.
8. If $D_C$ is a zero vector, then re-calculate $c \oplus (c_k G'^{-1}_k G') \oplus e_k(G'^{-1}_k G')$ on
   the unselected columns to form a vector $D_{\bar{C}}$ with length $n-k-\epsilon$.
   Otherwise if still exist unused $k$-bit error pattern, goto step 5.
   Otherwise goto step 1.
9. If $weight(D_{\bar{C}}) \leq t - weight(e_k)$ then stop and returns $m = (c_k \oplus e_k)G'^{-1}_k$.
   Otherwise if still exist unused $k$-bit error pattern, goto step 5.
   Otherwise goto step 1.
```

Figure 3.1: Modified McEliece's Decoding Algorithm with parameter (j, ϵ)

is instead of checking if the whole vector consists with $weight(c \oplus c_k G'^{-1}_k G' \oplus e_k G'^{-1}_k G') \leq t$, we only calculate the vector $(c \oplus c_k G'^{-1}_k G' \oplus e_k G'^{-1}_k G')$ on some randomly selected positions to form a shorter vector D, and apply the original check if and only if D is a zero vector. The key point of this method is based on a conjecture [30]:

Conjecture 3.1.1. [30] When the selection in step 1 is bad, the weight of $c \oplus c_k G'^{-1}_k G' \oplus e_k G'^{-1}_k G'$ is not only larger than t but also has approximate weight density 0.5.

We apply the idea of the improvement in the McEliece's decoding algorithm and propose a modified version of it; see Figure 3.1. An analysis of the algorithm is given by the following theorem.

Theorem 3.1.3. If the Conjecture 3.1.1 holds, then the expected number of bit-wise operations required by the modified algorithm is different from the original requirement by a factor:

$$\left\{\left[(n-k)\times\frac{\binom{\frac{n-k}{2}}{\epsilon}}{\binom{n-k}{\epsilon}}+\epsilon\times(1-\frac{\binom{\frac{n-k}{2}}{\epsilon}}{\binom{n-k}{\epsilon}})\right]/(n-k)\right\}/\left\{1-\sum_{i=1}^{\epsilon}\frac{\binom{t}{i}\binom{n-k-t}{\epsilon-i}}{\binom{n-k}{\epsilon}}\right\}.$$

Proof. we discuss the effect caused by modification.

speed-up factor: The modification reduces the time consumption on checking the weight of $c\oplus(c_kG_k'^{-1}G')\oplus e_k(G_k'^{-1}G')$. The original requirement is $j/2\times(n-k)$, when the modification is applied, there is a conditional probability $\binom{\frac{n-k}{2}}{\epsilon}/\binom{n-k}{\epsilon}$ of the event: the selected ϵ columns results in a zero vector under the condition of bad selection in step 1. In these cases, original $j/2\times(n-k)$ operations are needed, otherwise we need only $j/2\times\epsilon$ operations. Compared with the original requirement, the left-hand-side of the equation is obtained.

slow-down factor: Actually in the modified algorithm, the expected number of outer-loop iterations will increase slightly. The outer-loop of the original algorithm halts if and only if the k-column-set with fewer than j errors in it is selected, it occurs with a probability $\sum_{i=0}^{j}\binom{t}{i}\binom{n-t}{k-i}/\binom{n}{k}$. But in the modified algorithm, the outer-loop halts if and only if the k-column-set with fewer than j errors in it is selected, and we did not make an erroneous judgement in the ”partial checking” of step 6-8 when the actual error vector hidden in selected columns is examined. There is an conditional probability upper-bound $\sum_{i=1}^{\epsilon}\frac{\binom{t}{i}\binom{n-k-t}{\epsilon-i}}{\binom{n-k}{\epsilon}}$ of the event: erroneous judgement- the selected ϵ positions have one or more errors under the condition of good selection in step 1. Thus we have a lower-bound $1-\sum_{i=1}^{\epsilon}\frac{\binom{t}{i}\binom{n-k-t}{\epsilon-i}}{\binom{n-k}{\epsilon}}$ of correct judgement, the expected number of outer-loop iterations increase by a upper-bounded factor $1/\left[1-\sum_{i=1}^{\epsilon}\frac{\binom{t}{i}\binom{n-k-t}{\epsilon-i}}{\binom{n-k}{\epsilon}}\right]$. □

Example 3.1.2. When $n=1024$, $k=524$, $t=50$ and using the parameter $\epsilon=2$. Let C_{orig} denote the expected number of bit-wise operations required by the original McEliece's decoding algorithm. The modified algorithm reduce the bit-wise operations requirement in inner-loop by a factor of 0.2525, and increase the iterations of outer-loop by a factor of 1.2348. Thus the overall bit-wise operation requirement is $0.311787\times C_{orig}$.

Next, we give an introduction of the second improvement, which aims to eliminates the duplicate guesses of the nearly error-free locations.

Recall that in the Tilburg's decoding algorithm, a random column-swap operation (in the step 4) is needed for each iteration of main-loop. Actually, randomly selecting two columns to swap may cause duplicate situation and imply redundant iterations of loop. One can easily build a hash-table to solve this problem but the size of the hash-table must be approximate to $\binom{n}{k}$ bits. It is not practical in implementation. We recommend to swap the columns according to a real-time generated Gray code for combinations, instead of selecting randomly. We define the Gray code for combinations as follows.

Definition 3.1.1. [26] A (n, k)-Gray code for combinations is a sequence of all the $\binom{n}{k}$ combinations so that successive combinations differ by only one element.

Example 3.1.3. We show a $(5, 3)$-Gray code for combinations:

$$\begin{array}{cccccccccc} 134 & \rightarrow & 234 & \rightarrow & 124 & \rightarrow & 145 & \rightarrow & 245 & \rightarrow \\ & 345 & \rightarrow & 135 & \rightarrow & 235 & \rightarrow & 125 & \rightarrow & 123 \end{array}$$

Follow Example 3.1.3, to apply the Gray code for combinations in the algorithm, we may decompose $G' = S_1 [I|A_1] P_1^{-1}$ such that P_1 permutes columns $1, 3, 4$ to columns $1, 2, 3$ in the first iteration of loop, then decompose $G' = S_2 [I|A_2] P_2^{-1}$ such that P_2 permutes columns $2, 3, 4$ to columns $1, 2, 3$ in the second iteration of loop and so on.

Clearly, if we follow the sequence of Gray code for combinations, we will traverse all possible combinations of $\binom{n}{k}$ without missing one. Moreover, we swap one column each time (since the successive combinations differ by only one element), thus the Tilburg's second improvement idea is still able to apply here to speed up the Gaussian elimination of $G'P_i$.

Several algorithms to construct the Gray code for combinations were proposed; the most classical one is the revolving door algorithm [26]. However, this algorithm requires $O(n)$ time complexity to generate the next combination. When applying in the decoding algorithm, an algorithm with constant-time requirement for generating each combination such as [10] [9] is more suitable.

3.2 Finding Low-Weight-Codeword Attack

Another approach of the attack (we call it the FLWC for simplicity) is to find a codeword in a general linear-code with a given (low) weight t. The FLWC problem can be formally described as follows: input a generator matrix G of a general linear-code, and a integer t, one must output a codeword v of G such that $weight(v) \leq t$. Actually, the GISD problem with a generator matrix G and a received vector c with error-bits fewer than the correcting ability t can be reduced to the FLWC problem. For each input instance of the GISD problem (G, c), we can construct an instance of the FLWC problem (G^*, t) as follows.

$$G^* = \begin{bmatrix} G \\ c \end{bmatrix}$$

and t is the correcting ability of G. We claim the solution of this instance of the FLWC problem is unique and it will be the error vector hidden in the received vector c.

Theorem 3.2.1. [22] If G is a generator matrix of a linear-code with error correcting ability t and c is a received vector with error bits fewer than t, then the solution of instance (G^*, t) of FLWC exists and is unique.

Proof. We prove the existence and uniqueness by the following discussion. First we show the existence of the solution. Suppose $c = mG \oplus e$ for some m, then e is a solution of instance (G^*, t) of FLWC since $e = [m|1]\,G^*$. Then we show the uniqueness of the solution. If e' is a solution of instance (G^*, t) of FLWC such that $e' \neq e$, there are three possibilities:

Case 1: $e' = [m'|0]\,G^*$, $m' \neq m$

$weight(m'G) \leq t$, but G has correcting ability t, all of its codewords should have weight be larger than $2t$, which is a contradiction.

Case 2: $e' = [m'|1]\,G^*$, $m' \neq m$

$weight(c \oplus m'G) \leq t$ and it implies that $weight(mG' \oplus m'G') \leq 2t$ holds. But G has correcting

$$G' = \left[\begin{array}{c|c|c} B & Z & I \\ D & 0 & 0 \end{array}\right].$$

Figure 3.2: The matrix after arrangement.

ability t, all the distances of its codewords should larger than $2t$, it is a contradiction.

Case 3: $e' = [m|0]\,G^*$

$weight(mG) \leq t$, but G has correcting ability t, all of its codewords should have weight be larger than $2t$, which is a contradiction.

□

3.2.1 Leon's Algorithm

The algorithm proposed by Leon [20] is a probabilistic algorithm. It introduces two parameters σ and p. The main idea of this algorithm is to randomly select $k + \sigma$ columns from G, where σ is a comparatively small value such as 2 or 3, and hope that the "solution vector" has low weight on these selected positions. Each time we randomly select the $k + \sigma$ columns, we apply a permutation P so that the selected columns are gathered together on the right-hand-side of the matrix (the solution of permuted instance can be transformed to the solution of the original one by making a multiplication of P^{-1}). Then we apply the Gaussian elimination (the solution is not effected by the Gaussian-elimination) so that the resulting matrix G' looks like the form in Figure 3.2.

Where I is a $e \times e$ identity matrix, B is an $(n - k - \sigma) \times e$ matrix, Z is a $(k + \sigma - e) \times e$ matrix and D is an $(n - k - \sigma) \times (k - e)$ matrix. After the rearrangement, an exhaustive search on linear-combinations of row of $Z|I$ is applied to find the combinations such that the resulting sum-vector v consists with $weight(v) \leq p$, the p is a threshold value in intuition. We show the precise operations by the following algorithm.

```
Leon's Algorithm with parameters $\sigma$ and $p$

Input: A $k \times n$ generator matrix $G$ and an integer $t$.
Output: The codeword $c$ of $G$ such that $weight(c) \leq t$.

1. Randomly select $k + \sigma$ columns, rearrange the matrix $G$ to the form of Figure 3.2.
2. Search for the linear combinations of $[Z||I]$ that lead to a $k + \sigma$ vector $v$ such that
   $weight(v) \leq p$. This can be achieved by considering the single matrix $Z$.
   If a vector $v$ consists with $weight(v) \leq p$ is found, goto step 3.
3. Calculate the corresponding linear combinations of $B$, if the resulting $n - k - \sigma$
   vector $b$ consists with $weight(b) + weight(v) \leq t$ then stop and returns $[b||v]\, P^{-1}$.
   Otherwise step 4.
4. Search for all combination of the rows of $D$, there will be $2^{k-e}$ combinations.
   For each combination, calculate the summation of the corresponding rows to form
   a $n - k - \sigma$ vector $d$, and check if $d$ consists with $weight(d \oplus b) + weight(z) \leq t$.
   If some $d$ consists with the inequality, then stop and returns $[b \oplus d||v]\, P^{-1}$.
   Otherwise step2.
```

We introduce an analysis of the Leon's algorithm [6] by following theorem.

Theorem 3.2.2. [20] The expected value of bitwise operations required per selection of the $k + \sigma$ columns by the Leon's algorithm is:

$$J_{(\sigma,p)}(n,k) = \sum_{e=1}^{k} \rho(\sigma,k,e) \times \left[\sum_{i=1}^{p} i \binom{p}{i} \left[(k+\sigma-e) + (n-k-\sigma) \times 2^{k-e} \times \sum_{j=0}^{p-i} \frac{\binom{k+\sigma-e}{j}}{2^{k+\sigma-e}} \right] \right]$$

where

$$\rho(\sigma,k,e) = \frac{2^{e(e-1)/2}}{2^{k+\sigma}k} \prod_{i=0}^{e-1} \frac{(2^{k+\sigma-i}-1)(2^{k-i}-1)}{(2^{i+1}-1)}.$$

And the expected value of the required column selections is:

$$N_{\sigma,p}(n,k) = 1 / \sum_{i=0}^{p} \frac{\binom{n-t}{k+\sigma-i}\binom{t}{i}}{\binom{n}{k+\sigma}},$$

Thus the expected value of the overall bitwise operations required is $N_{\sigma,p}(n,k) \times J_{(\sigma,p)}(n,k)$.

Proof. The probability of the event that the "solution vector" has fewer weight than p in the selected $k + \sigma$ columns is:

$$\sum_{i=0}^{p} \frac{\binom{n-t}{k+\sigma-i}\binom{t}{i}}{\binom{n}{k+\sigma}}$$

Thus the expected number of the column selection requirement is its inverse. According to [21], the number of $(k, k+\sigma)$ binary matrices with rank e is:

$$2^{e(e-1)/2} \prod_{i=0}^{e-1} \frac{(2^{k+\sigma-i}-1)(2^{k-i}-1)}{(2^{i+1}-1)},$$

so the probability of the event that the rearranged matrix has a specified value of e is given as $\rho(\sigma, k, e)$. The average number of bit-wise additions and weight-checking on $(k+\sigma-e)$-bit words is given by $(k+\sigma-e) \times \sum_{i=1}^{p} i\binom{e}{i}$, and for each checking, the probability of satisfying the $weight(v) \leq p$ is $\sum_{j=0}^{p-i} \frac{\binom{k+\sigma-e}{j}}{2^{k+\sigma-e}}$. In those cases, additional 2^{k-e} linear combination of D should be examined and it takes $(n-k-\sigma)2^{k-e}$ bit-wise operations. Summarizing the above discussion, the formula of $J_{(\sigma,p)}(n, k)$ is given. □

3.2.2 Stern's Algorithm

Observing the Leon's algorithm, we can see that the time consumption of step 2 is quiet large, the main purpose of the Stern's algorithm is to reduce the time consuming of step 2. Precisely speaking, they proposed a faster method to search for partial-codewords (the k-prefix of codewords) which have hamming weights fewer than a given threshold. Two parameters p and l are introduced in their algorithm.

Their algorithm is first designed to operate on the parity check matrix, actually the algorithm can be slightly modified to work on a generator matrix. We describe the algorithm of the parity check matrix version as shown in Figure 3.3.

It is interesting that, in the Stern's algorithm, the "solution vector" is expected to have more weight (precisely, $weight = t - 2p$) in the $n-k$ columns selected in step 1. In other words, we hope that the solution vector has less weight in the unselected k columns. This hope is the same as Leon's idea. The observation gives us an intuition that these two algorithms are quiet similar in large scale.

Stern's Algorithm with parameters p and l

Input: A $k \times n - k$ parity check matrix H and an integer t.
Output: The codeword c of H such that $weight(c) \leq t$.

1. Randomly select $n - k$ columns of H, apply a permutation P on H so that the selected columns are gathered together on the right-hand-side of the matrix (the solution of permuted instance can be transformed to the solution of the original one by making a multiplication of P^{-1}). Then we apply the Gaussian elimination (the solution is not effected by the Gaussian-elimination) so that the resulting matrix $H'P$ looks like the following form.

$$H'P \quad = \quad \left[\begin{array}{c|c} Q & I_{n-k} \end{array}\right].$$

2. Randomly split the columns of matrix Q into two subsets, then apply another permutation (all the permutation applied can be summarized to a overall permutation P^*, and the solution of permuted instance still can be transformed to the solution of the original one by making a multiplication of P^{*-1}) so that the matrix HP^* looks like the following form.

$$H'P^* \quad = \quad \left[\begin{array}{c|c|c} X & Y & I_{n-k} \end{array}\right].$$

3. Randomly select l rows of the matrix $[X|Y]$ and apply another permutation on rows (the permutation applied on rows does not effect the solution of the instance) so that the resulting matrix $H''P^*$ looks like the following form.

$$H''P^* \quad = \quad \left[\begin{array}{c|c|c} X_l & Y_l & \multirow{2}{*}{J_{n-k}} \\ X_{k-l} & Y_{k-l} & \end{array}\right].$$

4. Search for all combinations of p columns of X_l, compute the corresponding summation vector v_{C_X} for each combination C_X, save c_X into a hash table with key v_{C_X} for further lookup.

5. Search for all combinations of p columns of Y_l, compute the corresponding summation vector v_{C_Y} for each combination C_Y, look up the hash table built in step 4 for the entry of key v_{C_Y}.

6. Each time we have $Hash(v_{C_Y}) \neq \phi$ in previous step, then summarize the $(k - l) \times k$ matrix $[X_{k-l}|Y_{k-l}]$ on columns $Hash(v_{C_Y}) \cup C_Y$ to a $(k - l) \times 1$ vector z.

7. If z consists with $weight(z) \leq t - 2p$, then find the columns set C_J in J_{n-k} such that the summation of HP^* on columns $Hash(v_{C_Y}) \cup C_Y \cup C_J$ is a zero vector, transform these columns into a vector m' such that $m'_i = 1$ if and only if $i \in \{Hash(v_{C_Y}) \cup C_Y \cup C_J\}$. Then stop and output $m'P^{*-1}$.

8. Otherwise ($Hash(v_{C_Y}) = \phi$ for all C_Y), goto step 1.

Figure 3.3: Stern's Algorithm with parameters p and l

3.2.3 Canteaut *et al.*'s Algorithm

A. Canteaut and F. Chabaud [6] proposed an improvement method for both Leon and Stern's algorithm. The idea of their improvement is very similar with the Tilburg's second improvement (see section 3.1.3). Observe step 1 of both Leon and Stern's algorithm, we can see that a Gaussian elimination is needed for each selection ($k + \sigma$ columns in Leon's algo. and $n - k$ columns in Stern's algo.). The Canteaut *et al.* proposed a strategy: to select $k + \sigma$ (or $n - k$ for Stern's algo.) columns with only one different column between the previous selection. The new column joining the selection is randomly selected from the unselected columns and the column departs from the selection is randomly selected from previous selected columns. Furthermore, they made a precise analysis on the expected value of iterations required by their improved algorithm.

Till now, all the mentioned attacks requires an exponential expected time consumption. But actually, the native MEPKC is proven vulnerable when partial information of the plaintext is leaked even against the chosen-plaintext attack. Known attacks of this type are [17] [5]. This kind of attacks substantially reduce the time consumption of the decoding attack with the help of partial information. We will give an introduction to these attacks in the next section.

3.3 Message-Resend and Related-Message Attack

The message-resend and related-message attack was first proposed by T.A. Berson. When a sender encrypts the same message m twice and two different error vectors are used, then several error-free positions of both ciphertext are leaked with a high probability. The adversary may recover the plaintext by general information-set decoding (see section 3.1) for few tries.

When an instance of MEPKC with public key $G : k \times n$ and t is used, consider a message m which is encrypted twice as:

$$c_1 = mG' \oplus e_1$$

$$c_2 = mG' \oplus e_2.$$

The vector $c_1 \oplus c_2$ is equal to $e_1 \oplus e_2$. Let $v = c_1 \oplus c_2$, the expected value of zeros in v is given by:

$$Z = \sum_{i=0}^{t} (2t - 2i) \frac{\binom{t}{i}\binom{n-t}{t-i}}{\binom{n}{t}},$$

since for any specified i, v has weight $2t - 2i$ if and only if e_1 and e_2 have errors on exact i positions, the probability of that event is $\binom{t}{i}\binom{n-t}{t-i}/\binom{n}{t}$, thus we have the equation above. For each zero position i of v, the probability of $e_1(i) = 1$ and $e_2(i) = 1$ is $(t/n)^2$, thus the expected value of the "real error-free positions in both e_1 and e_2" in v is:

$$RZ = \lfloor (1 - (\frac{t}{n})^2) \sum_{i=0}^{t} (2t - 2i) \frac{\binom{t}{i}\binom{n-t}{t-i}}{\binom{n}{t}} \rfloor.$$

If we randomly select k positions in zero positions of v, then the probability that all the selected positions are error-free in both e_1 and e_2 is $\binom{Z-RZ}{k}/\binom{Z}{k}$, thus the expected iterations of the k-position selection is $\binom{Z}{k}/\binom{Z-RZ}{k}$. The expected iterations are small in practical instances of MEPKC, for example, when $n = 1024$, $k = 524$ and $t = 50$, then $Z \approx 930$, $RZ \approx 3$ and the expected iterations is $\binom{930}{524}/\binom{930-3}{524} \approx 12$.

Moreover, the author provided an extension of the attack, that is, if two different messages are sent and an adversary knows the bitwise XOR of these two messages, then both messages can be recovered in few tries. Consider two messages m_1 and m_2 are encrypted by using two different random vector (this occurs with a high probability $1 - \binom{n}{t}^{-1}$) and $m_1 \oplus m_2$ is leaked suddenly. Let the two ciphertext be c_1 and c_2:

$$c_1 = m_1 G' \oplus e_1$$

$$c_2 = m_2 G' \oplus e_2,$$

then the vector $c_1 \oplus c_2$ is equal to $m_1 G' \oplus m_2 G' \oplus e_1 \oplus e_2$. An adversary can eliminate the $m_1 G' \oplus m_2 G'$ term by XOR it with the known $(m_1 \oplus m_2)G'$, thus he can obtain the vector $e_1 \oplus e_2$. The same technique in the message-resend attack can be applied to find the positions which are error-free in both e_1 and e_2.

3.4 Known-Partial-Plaintext Attack

The known-partial-plaintext attack was proposed by Kobara *et al.* and it is a practical attack in some interesting cryptosystem applications. For example, consider a command exchange routine during the war, a message such like ”Today’s password is: Seattle”, the password changes everyday but the prefix string is constant. In this case, they formalize the message as $m = (m_1 \| m_2)$; m_1 is already known and let l denote its length. In this viewpoint, the ciphertext can be rewritten as:

$$c = (m_1 \| m_2) \cdot \left[\begin{array}{c} G'_{l \times n} \\ \hline G'_{(k-l) \times n} \end{array} \right] + e.$$

Thus we have:

$$c = m_1 G'_{l \times n} \oplus m_2 G'_{(k-l) \times n} \oplus$$

$$\Rightarrow c' = c \oplus m_1 G'_{l \times n} = m_2 G'_{(k-l) \times n} \oplus e.$$

The time consumption required for recovering m_2 from c' and $G'_{(k-l) \times n}$ is much lower.

3.5 Chosen-Ciphertext Attack and Malleability Attack

In 2000, Sun [29] proposed an adaptive chosen-ciphertext attack and a malleability attack to break the MEPKC. In the adaptive chosen-ciphertext attack, the key to success is that when a ciphertext c is given, we may construct another ciphertext c' corresponding to the same plaintext with a high probability. The construction of c' is to randomly select two bits in the ciphertext and flip them. The flipped ciphertext c' has the same error bit amount with c if and only if one of the flipped bits is an error bit in c and the orther is not, and this event happens with probability $\binom{t-1}{1}\binom{n-t}{1}/\binom{n}{2} = 2(t-1)(n-t)/n(n-1)$. Thus the expected iterations of selection is $n(n-1)/2(t-1)(n-t) \approx n/2t$, a quantity having polynomial relation with n, and it is an acceptable amount in practical instance of the MEPKC.

A malleability attack can be roughly described as when a ciphertext c corresponding to plaintext m is given (note that m is not given), if an adversary is able to construct another ciphertext c' which corresponds to another plaintext m' with some relation to m, then we say that the malleability attack is workable.

The malleability attack proposed by the author is able to construct another ciphertext c' corresponding to $\bar{m}$. The attack scheme is to XOR all the rows of G' with c to form c'. Actually this attack also leads us to construct ciphertext corresponding to any $m' = m \oplus v$, where v is an adversary-controllable vector.

3.6 Reaction Attack

In 1999, Hall, Goldberg, and Schneier [13] proposed the reaction attack against several public key cryptosystems based on decoding problems and lattice problems, including the McEliece, Hwang-Rao, Ajtai-Dwork, and Goldreich-Goldwasser-Halevi cryptosystems. In their attack against the MEPKC, an adversary sends the key-owner a ciphertext which may contain one or more additional error bits. In common implementations, the garbled ciphertext will cause failure in decryption or an illegal plaintext checksum. The adversary then watches the reaction of the key-owner in order to determine whether or not the ciphertext is decrypted correctly. This *send-and-watch* behavior can be modeled as querying the reaction oracle. By repeatedly querying the reaction oracle, the adversary can obtain the plaintext.

Suppose a (n, k)-linear code with error correcting ability t is used in the MEPKC, their attack takes at most $n + 2t$ queries under a reasonable reaction oracle model.

3.6.1 The Reaction Oracle Model

Compared with the chosen-ciphertext attack, the reaction attack uses a weaker assumption. They assume that the key-owner does not return the decrypted plaintext for decryption queries from the adversary, but he leaks the information about the legality of the ciphertext as a result of his reaction. Since this assumption is comparatively weak, their attack is considered more feasible than traditional CCA. Their model also depends on the following assumption.

Assumption 3.6.1. [13] If a (n, k) error-correcting code is used which can correct t or fewer errors, then a received vector with error-weight $> t$ will cause either failure in correcting stage or an illegal message checksum.

The assumption is reasonable since there are several decoders for Reed-Solomon and Goppa codes which meet this criterion. In their reaction oracle model, input a vector c, there are two types of oracle outputs.

Type 1: Return an error message due to the failure in decryption or illegal plaintext checksum.
Type 2: Return nothing or an acknowledgement message to reflect the successful decryption and legal plaintext checksum.

We can abstract the reaction oracle as follows.

Reaction Oracle in (S, G, P, t)**-MEPKC:**
Input: A vector c.
Output: Compare with the codeword $m \cdot G'$, one of the reactions is presented.
Type 1: If c corresponds to a received vector with error-weight $> t$.
Type 2: If c corresponds to a received vector with error-weight $\leq t$.

3.6.2 The Attack Algorithm

Under the reaction oracle model, they gave two algorithms to recover the corresponding plaintext m of a given ciphertext c.

```
Algorithm A
1. Let $i = 1$
2. Flip bits 1 through $i$ of $c$ to form $c'$.
3. Request the reaction oracle by $c'$.
4. If the output of the oracle is Type 1, halt the procedure and continue onto the
   next algorithm. Otherwise, increment i and goto step 2.
```

```
Algorithm B
1. Let $i = 1$
2. Flip bit $i$ of $c'$ to form $c''$.
3. Request the reaction oracle by $c''$.
4. If the output of the oracle is Type 2, then bit $i$ is in error. Record $i$, add 1 to $i$
   and goto step 2 if $i \leq n$. Otherwise halt.
```

The attack requires at most $2t + 1$ queries in Algorithm A, and at most $n - 1$ queries in Algorithm B, so the total number of queries is upper-bounded by $n + 2t$.

3.6.3 An Improved Attack Algorithm

We note that an adversary does not have to make the whole $1 \times n$ ciphertext c to be error-free. He only has to make sure some k-bits $\{a_1, a_2, \ldots, a_k\} \subset \{1, 2, \ldots, n\}$ are error-free, where the corresponding columns of G' are linearly independent. Thus we design the following attack algorithm.

Algorithm 3.6.1: Improved reaction attack algorithm

Input: Ciphertext c.
Output: Plaintext m corresponds to c.

```
1.  Select $k$ columns index by $\{a_1, a_2, \ldots, a_k\} \subset \{1, 2, \ldots, n\}$ in $G'$, such
    that the $k$ columns of $G'$ are linearly independent.
2.  Combine the $k$ columns of $G'$ to form a $k \times k$ matrix $G_k$.
3.  Combine the $k$ bits $\{c_{a_1}, c_{a_2}, \ldots, c_{a_k}\}$ of c to form a vector $c_k$.
4.  Run the original Algorithm A and B to eliminate the errors in $c_k$.
5.  $m = c'' \cdot G_k'^{-1}$
```

that the algorithm requires at most $k + 2t$ queries to recover the plaintext.

Chapter 4

Trichotomy Reaction Attack

In the reaction attack presented by Hall *et al*, it takes $n + 2t$ queries to recover the plaintext in the worst case. In this chapter, we propose a new trichotomy reaction oracle model. Under the new model, we establish connection between the reaction attack of the MEPKC and a special version of the counterfeit coins problem, which we name the *Comparative Counterfeit Coins Problem* (CCCP). Combine the connection and a greedy approach of the CCCP, we can design a new algorithm to recover the plaintext from ciphertext in at most $\lfloor k/2 \rfloor + t + 3$ queries when $4t \leq k$.

This chapter is organized as follows: In Section 1, we propose the trichotomy reaction oracle model. In Section 2, we introduce the formal definition of the counterfeit coins problem and some modified versions of it. In Section 3, we design a greedy algorithm to solve the CCCP. In Section 4, we show the connection between the attack of MEPKC and the CCCP. In Section 5, we design a new attack algorithm by applying the greedy algorithm proposed in Section 3.

4.1 Trichotomy Reaction Oracle Model

In the original reaction oracle model, when a ciphertext with less than t error bits is received, it will be treated as a valid ciphertext to decrypt, and represent the Type 2 reaction. But in the MEPKC, randomly selecting an error vector with constant weight t is suggested to avoid weak encryptions. So it is reasonable to assume that a common implementation of the MEPKC will check whether the ciphertext is corresponding to a received vector with error weight t. This check can be simply done by watching if the decrypted plaintext m consists with $weight((m \cdot G') \oplus c) = t$. To match its purpose, we call it the error-weight check.

Clearly, the check should be done with a correct plaintext m to make sense. Thus one has to apply the check after the decryption and the checksum verification procedures. When the error-weight check fails, it indicates an improper encryption. But since the ciphertext c passed the decryption and the decrypted plaintext m passed the checksum verification, the failure in the error-weight check does not hurt the correctness of m. Thus it is reasonable to assume that the error-weight fault is considered as a non-critical error and a Warning Message is returned in this case. Under the assumption, we propose the trichotomy reaction oracle model. In this reaction oracle, when we input a vector c, there are three types of oracle outputs:

Type 1:	Return an error message due to failure in decryption or illegal plaintext checksum.
Type 2:	Return nothing or an acknowledgement message to reflect the successful decryption, legal plaintext checksum and proper error-weight.
Type 3:	Return a warning message due to the failure in the error-weight check.

Our model also requires the assumption proposed in Assumption 3.6.1. We can abstract the trichotomy reaction oracle as follows.

Trichotomy Reaction Oracle in (S, G, P, t)**-MEPKC:**
Input: A vector c.
Output: Compare with the codeword $m \cdot G'$, one of the reactions is presented.

Type 1:	If c corresponds to a receive vector with error-weight $> t$.
Type 2:	If c corresponds to a receive vector with error-weight $= t$.
Type 3:	If c corresponds to a receive vector with error-weight $< t$.

We present an instance of MEPKC implementation that matches the trichotomy reaction oracle assumption.

Key Generation algorithm:
Generate four matrices G,S,P,G', a checksum function $Checksum(m)$ and its corresponding verifying function $Varify(m', C)$.

G	$k \times n$ generator matrix of a binary Goppa code with correcting ability t.
S	$k \times k$ random binary invertible matrix.
P	$n \times n$ random permutation matrix.
G'	$G' = SGP$
$Checksum(m)$	a checksum function, it outputs an fixed-length characteristic binary string C for each input string with low collision probability.
$Verify(m', C)$	the corresponding verifying function which gives a acknowledgement output if and only if $C = Checksum(m')$.
Secret key:	S, G, P
Public key:	$G', t, Checksum, Verify$

Encryption algorithm:
To encrypt a $1 \times (k - |Checksum(m)|)$ message m, one has to randomly select a $1 \times n$ error vector e with Hamming weight t. Then output the corresponding ciphertext c:
Ciphertext: $c = (m \| Checksum(m)) \cdot G' \oplus e$

Decryption algorithm:
To decrypt a ciphertext c, one has to go through the following steps.

1. Calculate $c' = c \cdot P^{-1}$, thus $c' = m'SG \oplus eP^{-1}$
2. Apply the decoding algorithm for Goppa code to eliminate the error vector.
   If the decoding procedure failed (it will happen only when a more-than-t error bits are present), output **Error**.
   Otherwise we have the vector $m'S$.
3. Calculate $m'' = m' \cdot S^{-1}$.
4. split m'' into $(m \| C) = m'$, where $|C|$ is the length of the checksum and $|m| = k - |C|$.
5. If $Verify(m, C) \neq acknowledgement$, output **Error**. Otherwise step 6.
6. If $|(m'' \cdot G') \oplus c| \neq t$, output **Warning**. Otherwise step 7.
7. Output message m and an **Acknowledgement**.

The behavior of the decryption algorithm in this implementation can be described by a codeword-ball diagram, see Figure 4.1.

Clearly, the behavior of the decryption algorithm consists with the assumption of the trichotomy reaction oracle model.

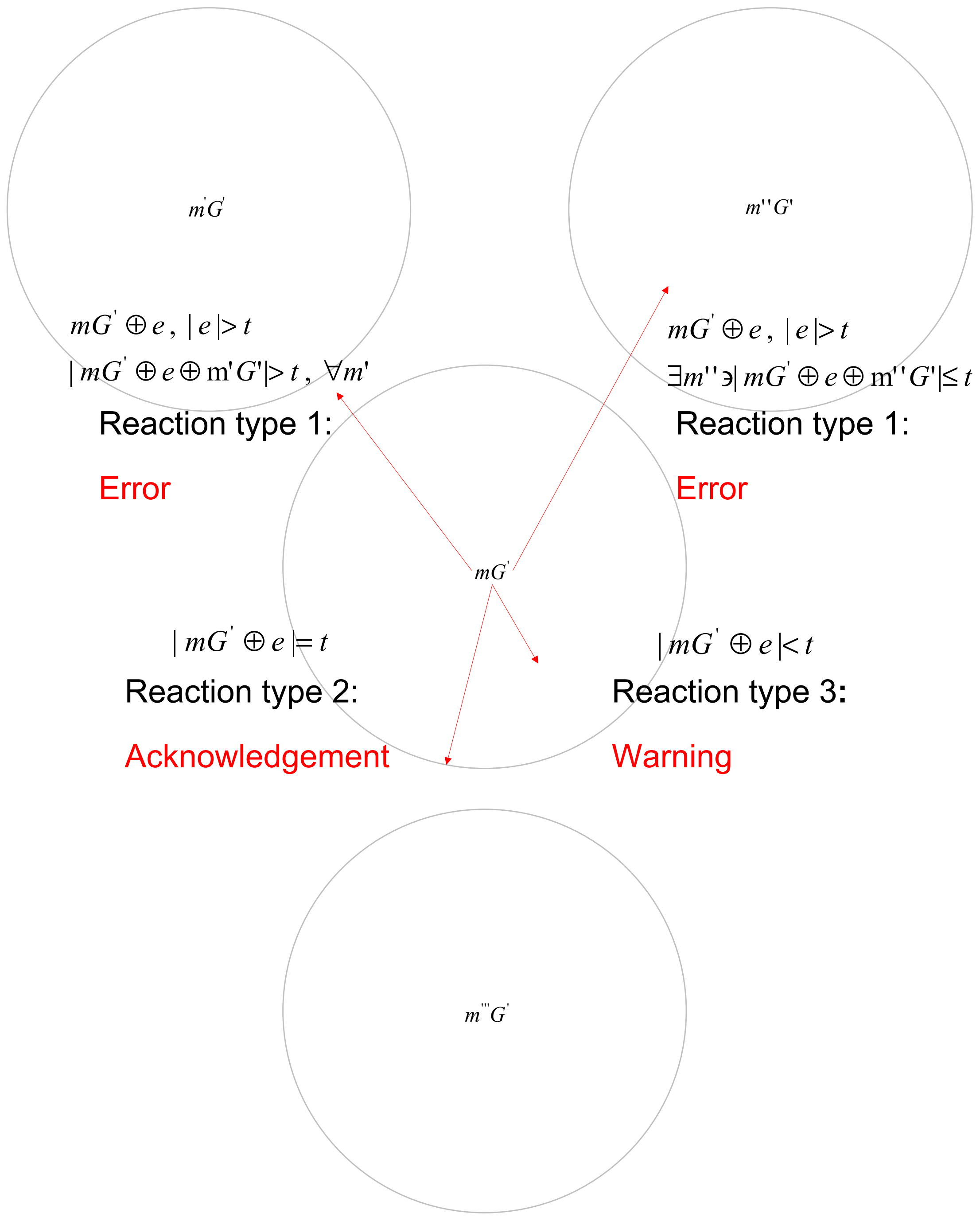

Figure 4.1: The decryption algorithm behavior that consists with trichotomy reaction oracle model.

4.2 Counterfeit Coins Problem

Counterfeit Coins Problem (CCP) is an old puzzle which has been widely discussed. It is defined as follows. Given a set of coins, there may exist some counterfeit ones whose weights are different from the normal ones. We want to identify all the counterfeit coins (if any) with as few weighings as possible by using an equal arms balance. There are several variations of this problem. A great part of them are different on the weighting oracle model. In this chapter, we will introduce two of them: Additive model and Comparative model.

4.2.1 Additive Counterfeit Coins Problem

Very similar to CCP, the Additive Counterfeit Coins Problem (ACCP) with parameters (n, d) is defined as follows. Suppose there are n coins with at most d counterfeit coins in them. We want to identify all the counterfeit coins (if any) with as few queries as possible by an *Additive Weighting Oracle* (AWO) which can tell "How many counterfeit coins in the sample." We formally define the AWO in (n, d)-ACCP as follows.

Additive Weighting Oracle in (n, d)-ACCP:
Input: A set of sample coin indexes S, $S \subseteq \{1, \ldots, n\}$.
Output: Number of counterfeit coins in the sample S.

In [7] and [15], the ACCP was solved by constructing a combinatorial object named k-selective family. Their approach requires $O(d \times log^2 n)$ queries. We use $M_{ACCP}(n, d)_A$ to denote the worst-case number of queries required by algorithm A to solve (n, d)-ACCP, and we define $M_{ACCP}(n, d) = Min_A\{M_{ACCP}(n, d)_A\}$.

Corollary 4.2.1. [8] $M_{ACCP}(n, d) \geq \ulcorner log_{d+1} \sum_{i=0}^{k} \binom{n}{i} \urcorner$

Proof. Straight from the information theory's lower bound. □

4.2.2 Comparative Counterfeit Coins Problem (CCCP)

The Comparative Counterfeit Coins Problem (CCCP) with parameters (n, d) is almost equal to the (n, d)-ACCP, but differ only on the weighting oracle model. In CCCP, the problem is defined with the *Comparative Weighting Oracle* (CWO). Let N_0 denote the number of normal coins in the sample and N_1 denote the number of counterfeit ones. The CWO tells "whether $N_0 < N_1$, $N_0 > N_1$ or $N_0 = N_1$ is true." We formally define the CWO in the (n, d)-CCCP as follows.

Comparative Weighting Oracle in (n, d)-CCCP:
Input: A set of sample coin indexes S, $S \subseteq \{1, \ldots, n\}$.
Output: $N_0 < N_1$, $N_0 > N_1$ or $N_0 = N_1$ in the sample S.

As far as we know, no solution for this problem is documented. In Section 7, we will introduce a greedy approach for the case of $4d \leq n$. Before that, we make a serial of inferences about the CCCP at the end of this section. We use $M_{CCCP}(n, d)_A$ to denote the worst-case number of weighings required by algorithm A to solve (n, d)-CCCP, and we define $M_{CCCP}(n, d) = Min_A\{M_{CCCP}(n, d)_A\}$.

Lemma 4.2.1. $M_{CCCP}(n, d) \geq M_{ACCP}(n, d)$

Proof. Assume we have an algorithm A which solves the (n, d)-CCCP. In order to solve (n, d)-ACCP, we run the algorithm A and respond to each query of CWO from A by some queries of the AWO that we have, then algorithm A will indicate the counterfeit coin set by assumption. That is, A acts like solving a (n, d)-CCCP as it is designed for, but actually it solves (n, d)-ACCP for us. The counterfeit coin set given by A is the answer to the (n, d)-ACCP that we face. Furthermore, suppose A requires at most ϵ queries of CWO and if we can respond to each query from A with a single query of the AWO, then this strategy for (n, d)-ACCP will also take at most ϵ queries. By the above discussion, we can obtain a strategy for (n, d)-ACCP from any algorithm A for (n, d)-CCCP with the same queries requirement. As we may solve (n, d)-ACCP by other designs which require less queries, this implies the statement of the lemma. We state the CWO simulation algorithm with single query of AWO as follows to complete the proof.

Algorithm 4.2.1: (n, d)**-Comparative Weighting Oracle simulation algorithm**

Input: (n, d) and a set of sample coin indexes S. Let $|S| = l$, $S \subseteq \{1, \dots, n\}$.
Output: $N_0 < N_1$, $N_0 > N_1$ or $N_0 = N_1$ in the sample S.

```
1.  Query the additive weighting oracle with $S$, let $r$ denote the output.
2.  If $2r > l$, output $N_0 < N_1$. Otherwise step 3.
3.  If $2r < l$, output $N_0 > N_1$. Otherwise step 4.
4.  If $2r = l$, output $N_0 = N_1$.
```

□

Corollary 4.2.2. $M_{CCCP}(n, d) \leq n$

Proof. There exists a trivial algorithm to find all the counterfeit coins by exact n queries. We simply query the Comparative Weighting Oracle with each single coin. The oracle returns either "$N_0 > N_1$" or "$N_0 < N_1$", thus we know whether the input coin is counterfeit or not. □

In the next section, we will propose a greedy approach of CCCP, which solves all (n, d)-CCCP with at most $\lfloor n/2 \rfloor + d + 3$ queries when $4d \leq n$.

4.3 A Greedy approach of CCCP

In this section, we focus on solving (n, d)-CCCP. The main idea of the greedy approach is to split n coins into small groups each with 4 coins, and solve each group with an optimal nonadaptive algorithm for 4 coins.

Definition 4.3.1. [8] A coin-weighting algorithm A is nonadaptive if all query samples must be specified without knowing the outcome of other queries.

Example 4.3.1. We give an example of nonadaptive algorithm to solve (3,3)-CCCP.

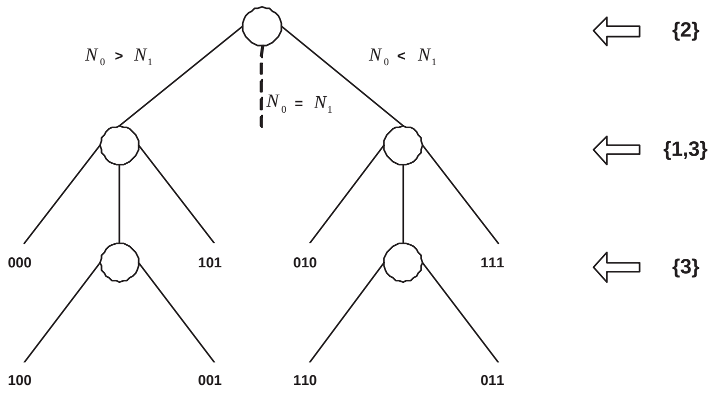

Figure 4.2: A nonadaptive algorithm solving (3,3)-CCCP

Algorithm 4.3.1: A nonadaptive algorithm solving (3,3)-CCCP

Input: Permission to query the Comparative Weighting Oracle.
Output: The set of counterfeit coin indexes $B \subseteq \{1,2,3\}$.

```
1.  Query the Comparative Weighting Oracle with {2}, let $r_1$ denote the output.
2.  Query the Comparative Weighting Oracle with {1,3}, let $r_2$ denote the output.
3.  If $r_1$="$N_0 > N_1$" and $r_2$="$N_0 > N_1$", return {}.
4.  If $r_1$="$N_0 > N_1$" and $r_2$="$N_0 < N_1$", return {1,3}.
5.  If $r_1$="$N_0 < N_1$" and $r_2$="$N_0 > N_1$", return {2}.
6.  If $r_1$="$N_0 < N_1$" and $r_2$="$N_0 < N_1$", return {1,2,3}.
7.  Query the Comparative Weighting Oracle with {3}, let $m_3$ denote the output.
8.  If $r_1$="$N_0 > N_1$" and $r_3$="$N_0 > N_1$", return {1}.
9.  If $r_1$="$N_0 > N_1$" and $r_3$="$N_0 < N_1$", return {3}.
10. If $r_1$="$N_0 < N_1$" and $r_3$="$N_0 > N_1$", return {1,2}.
11. If $r_1$="$N_0 < N_1$" and $r_3$="$N_0 < N_1$", return {2,3}.
```

It is much simpler to represent the algorithm by a decision tree diagram, see Figure 4.2. In the diagram, situation of coins are represented by binary strings, where bit 1 indicates a counterfeit coin, bit 0 indicates a normal coin. We will use this notation in the following discussion.

4.3.1 Optimal nonadaptive algorithm for 4 coins

We find optimal nonadaptive algorithms for 4 coins by an exhaustive search. We first introduce the measurement we use.

Definition 4.3.2. Suppose A_n is a nonadaptive algorithm to identify the situation of n coins. We use $Q_{A_n}(x)$ to denote the number of queries required by A_n when the actual situation of coins is x, where x is a binary string with length n.

Example 4.3.2. Let A_3 be the algorithm described in Figure 4.2, then $Q_{A_3}(000)$ corresponds to the length of path from root to leaf 000. Thus $Q_{A_3}(000) = 2$, $Q_{A_3}(010) = 2$, $Q_{A_3}(100) = 3$, $Q_{A_3}(011) = 3$.

Definition 4.3.3. Let R_{A_n} be a $1 \times (n+1)$ vector, $R_{A_n} = (R^0_{A_n}, R^1_{A_n}, \ldots, R^n_{A_n})$, where $R^i_{A_n} = \sum_{|x|=i} Q_{A_n}(x)$ for $0 \leq i \leq n$.

Example 4.3.3. Let A_3 be the algorithm described in Figure 4.2, then $R^0_{A_3} = Q_{A_3}(000) = 2$, $R^1_{A_3} = Q_{A_3}(001) + Q_{A_3}(010) + Q_{A_3}(100) = 8$, $R^2_{A_3} = Q_{A_3}(011) + Q_{A_3}(110) + Q_{A_3}(101) = 8$, $R^3_{A_3} = Q_{A_3}(111) = 2$. Thus $R_{A_3} = (2, 8, 8, 2)$.

Definition 4.3.4. Let A_n and B_n be two nonadaptive algorithms to identify the situation of n coins. We say $R_{A_n} \leq R_{B_n}$ if and only if $R^i_{A_n} \leq R^i_{B_n}$ for all $0 \leq i \leq n$.

Let OPT_4 denote the algorithm represented by figure 4.3, where the input of OPT_4 is a coin indexes set $S = \{S_1, S_2, S_3, S_4\}$ and the output is the set of counterfeit coin indexes $B \subseteq S$. We have the following theorem.

Theorem 4.3.1. *OPT_4 is an optimal nonadaptive algorithm to identify the situation of 4 coins in sense of $R_{OPT_4} \leq R_{A_4}$ for any nonadaptive algorithm A_4.*

Proof. We establish this theorem by making an exhaustive search of all candidate nonadaptive algorithms which identify the situation of 4 coins. A candidate nonadaptive algorithm corresponds to a sequence of subsets, where the subsets belong to the power-set of {1,2,3,4}. Note that any subsets-sequence longer than 4 needs not to be considered since it is even worse than the trivial method. Thus there are $\binom{16}{4} \times 4!$ candidates. We calculate the R_{A_4} vector for each candidate algorithm A_4. The algorithm represented by figure 4.3 is one of the optimal algorithms. It has $R_{OPT_4} = (2, 12, 18, 12, 2)$ □

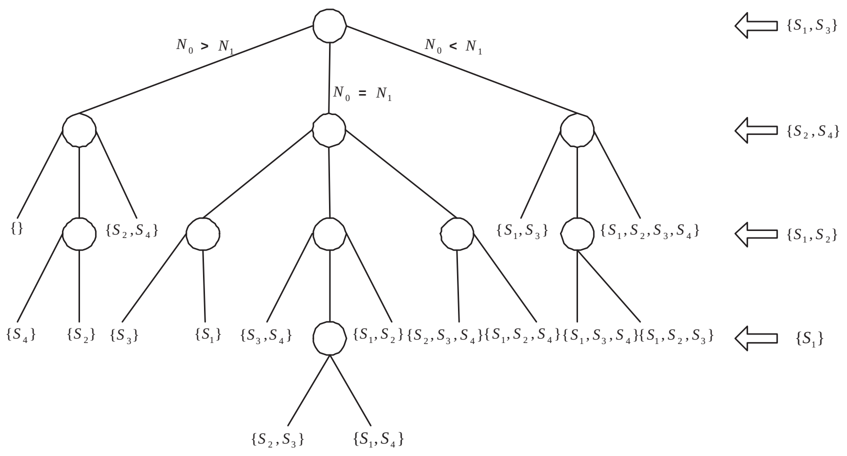

Figure 4.3: OPT_4, An optimal nonadaptive algorithm for 4 coins

By the same technique, we obtain optimal nonadaptive algorithms for 1, 2 and 3 coins. We use OPT_1, OPT_2 and OPT_3 to denote them. They are:

Algorithm 4.3.2: OPT_1

Input: Coin indexes set S, let $S = \{S_1\}$.
Output: The set of counterfeit coin indexes $B \subseteq S$.

```
1.  Query the comparative weighting oracle with $\{S_1\}$, let $r_1$ denote the output.
2.  If $r_1$="$N_0 > N_1$", return $\{\}$.
3.  If $r_1$="$N_0 < N_1$", return $\{S_1\}$.
```

Algorithm 4.3.3: OPT_2

Input: Coin indexes set S, let $S = \{S_1, S_2\}$.
Output: The set of counterfeit coin indexes $B \subseteq S$.

```
1.  Query the comparative weighting oracle with $\{S_1, S_2\}$, let $r_1$ denote the output.
2.  If $r_1$="$N_0 > N_1$", return $\{\}$.
3.  If $r_1$="$N_0 < N_1$", return $\{S_1, S_2\}$.
4.  Query the comparative weighting oracle with $\{S_1\}$, let $r_2$ denote the output.
5.  If $r_2$="$N_0 > N_1$", return $\{S_2\}$.
6.  If $r_2$="$N_0 < N_1$", return $\{S_1\}$.
```

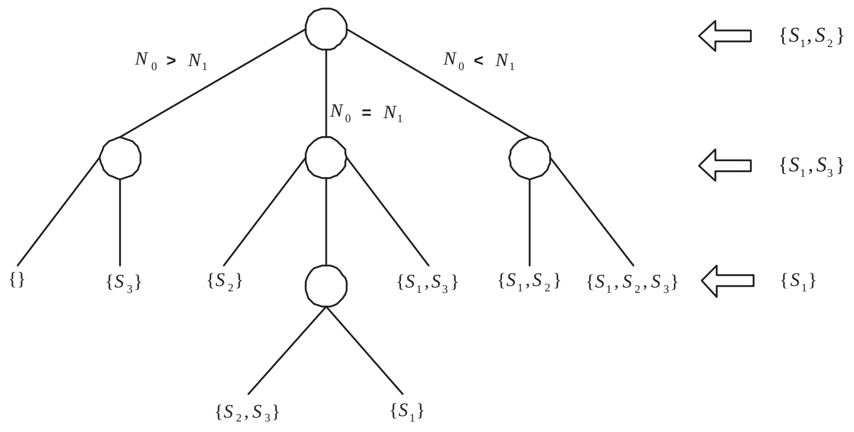

Figure 4.4: Decision tree diagram of OPT_3.

Algorithm 4.3.4: OPT_3

Input: Coin indexes set S, let $S = \{S_1, S_2, S_3\}$.
Output: The set of counterfeit coin indexes $B \subseteq S$.

```
1.  Query the comparative weighting oracle with $\{S_1, S_2\}$, let $r_1$ denote the output.
2.  Query the comparative weighting oracle with $\{S_1, S_3\}$, let $r_2$ denote the output.
3.  If $r_1$="$N_0 > N_1$" and $r_2$="$N_0 > N_1$", return $\{\}$.
4.  If $r_1$="$N_0 > N_1$" and $r_2$="$N_0 = N_1$", return $\{S_3\}$.
5.  If $r_1$="$N_0 = N_1$" and $r_2$="$N_0 > N_1$", return $\{S_2\}$.
6.  If $r_1$="$N_0 = N_1$" and $r_2$="$N_0 < N_1$", return $\{S_1, S_3\}$.
7.  If $r_1$="$N_0 < N_1$" and $r_2$="$N_0 = N_1$", return $\{S_1, S_2\}$.
8.  If $r_1$="$N_0 < N_1$" and $r_2$="$N_0 < N_1$", return $\{S_1, S_2, S_3\}$.
9.  Query the comparative weighting oracle with $\{S_1\}$, let $r_3$ denote the output.
10. If $r_3$="$N_0 > N_1$", return $\{S_2, S_3\}$.
11. If $r_3$="$N_0 < N_1$", return $\{S_1\}$.
```

We represent OPT_3 by a decision tree diagram, see Figure 4.4.

4.3.2 A greedy algorithm for (n, d)-CCCP

Based on the previous discussion, we develop the following algorithm to solve (n, d)-CCCP.

Algorithm 4.3.5: Greedy algorithm for (n, d)-CCCP

Input: (n, d) and the permission to query the comparative weighting oracle.
Output: The set of counterfeit coin indexes $B \subseteq \{1, 2, \ldots, n\}$.

```
1. Let $i = 1$.
2. If $i + 3 > n$ then step 5.
3. Call $OPT_4$ to identify the situation of the 4-coins group $(i, i + 1, i + 2, i + 3)$.
4. $i = i + 4$, goto step 2.
5. If $i = n$, call $OPT_1$ to identify the situation of the coin $(i)$.
6. If $i = n - 1$, call $OPT_2$ to identify the situation of the coins $(i, i + 1)$.
7. If $i = n - 2$, call $OPT_3$ to identify the situation of the coins $(i, i + 1, i + 2)$.
```

We analyze this algorithm with the following theorem.

Theorem 4.3.2. When $4d \leq n$, the algorithm requires at most $\lfloor n/2 \rfloor + d + 3$ queries.

Proof. We discuss this bound by four cases.

Case n mod $4 = 0$: The worst-cases happen on minimizing the all zero 4-coins groups. For example, in (12,2)-CCCP, the coins situation $\underline{0000}\ \underline{0001}\ \underline{0001}$ is a worst-case instance which totally requires 8 queries, and $\underline{0000}\ \underline{0011}\ \underline{0000}$ isn't. Since $4d \leq n$, we have at least $(n - 4d)/4$ all zero groups, each of them requires 2 queries. Each of the rest groups requires at most 3 queries, so the total queries no more than $3d + 2(n - 4d)/4 = n/2 + d$.

Case n mod $4 = 1$: By case 1, total queries no more than $(n - 1)/2 + d + 1$.

Case n mod $4 = 2$: By case 1, total queries no more than $(n - 2)/2 + d + 2$.

Case n mod $4 = 3$: By case 1, total queries no more than $(n - 3)/2 + d + 3$.

Combining the four cases, we have an upper-bound $\lfloor n/2 \rfloor + d + 3$ on query numbers.

□

Corollary 4.3.1. $M_{CCCP}(n, d) \leq \lfloor n/2 \rfloor + d + 3$ when $4d \leq n$.

4.4 Attack under Trichotomy Reaction Oracle Model

Under the trichotomy reaction oracle model combining with the idea proposed in section 3.6.3, an adversary who wants to recover the plaintext m from ciphertext c, is equivalent to facing a combinatorial problem; we call it the *Reaction Attack Problem* (RAP). The problem with parameters (c, C, t) can be stated as follows. Given a ciphertext c and a coordinate set $C = \{C_1, C_2, \ldots\} \subseteq \{1, \ldots, |c|\}$. In the coordinates C of the ciphertext c, there are at most t coordinates effected by errors. We want to identify all the effected coordinates (if any) in C with as few queries as possible by using the trichotomy reaction oracle (TRO) proposed in section 4.1.

We use $M_{RAP}(c, C, t)_A$ to denote the worst-case number of queries required by algorithm A to solve (c, C, t)-RAP. Define $M_{RAP}(c, C, t) = Min_A\{M_{RAP}(c, C, t)_A\}$, two main theorems in this chapter are present.

Theorem 4.4.1. $M_{RAP}(c, C, t) \leq M_{CCCP}(|C|, t)$

Proof. We use a similar proving technique as in Lemma 4.2.1. View the coordinates in C as $|C|$ coins. The coordinates effected by errors correspond to the counterfeit coins, and the unaffected coordinates correspond to the normal coins. Assume we have an algorithm A which solves $(|C|, t)$-CCCP. In order to solve (c, C, t)-RAP, we run algorithm A and respond to each query of CWO from A by some queries of the TRO that we have, then algorithm A will indicate the counterfeit coin set by assumption. The counterfeit coin set given by A is the answer to the (c, C, t)-RAP after translating it to the corresponding coordinate set. Furthermore, suppose A requires at most ϵ queries of CWO and if we can respond each query from A with a single query of the TRO, then this strategy for (c, C, t)-RAP will also take at most ϵ queries. The success of CWO simulation by TRO largely depends on a trick: Let T be a coordinate set. If we flip c on all coordinates in T, and send it to the TRO, the Type1 Reaction indicates there are fewer effected coordinates than unaffected ones in T before the flip, the Type2 Reaction indicates equal effected coordinates and unaffected ones, and the Type3 Reaction indicates more effected coordinates than unaffected ones. We state the CWO simulation algorithm with single query of TRO as follows to complete the proof.

Algorithm 4.4.1: **($|C|, t$)-Comparative Weighting Oracle simulation algorithm**

Input: (c, C, t) and a set of sample coin indexes $S = \{S_1, S_2, \ldots\} \subseteq \{1, \ldots, |C|\}$.
Output: $N_0 < N_1$, $N_0 > N_1$ or $N_0 = N_1$ in the sample S.

```
1. Flip all bits on coordinates $\{C_{S_1}, C_{S_2}, \ldots, C_{S_{|S|}}\}$ of ciphertext $c$ to form $c'$.
2. Query the trichotomy reaction oracle with $c'$, let $r$ denote the output reaction.
3. If $r$ = Type 1 reaction, output $N_0 > N_1$. Otherwise step 4.
4. If $r$ = Type 2 reaction, output $N_0 = N_1$. Otherwise step 5.
5. If $r$ = Type 3 reaction, output $N_0 < N_1$.
```

□

By the proof of Theorem 4.4.1, we obtain the most important theorem in this chapter.

Theorem 4.4.2. If we have an algorithm solve $(|C|, t)$-CCCP with at most ϵ queries, then we can design an algorithm to solve (c, C, t)-RAP with at most ϵ queries.

Proof. We state the design of the algorithm for (c, C, t)-RAP to complete the proof. Let A denote the algorithm solving $(|C|, t)$-CCCP.

Algorithm 4.4.2: **An algorithm for (c, C, t)-RAP**

Input: Ciphertext c, coordinate set C and effected coordinates upper-bound t.
Output: The error-effected coordinate set $B \subseteq C$.

```
1. Run Algorithm $A$.
   When the algorithm queries the CWO with sample set $S$, simulate the response
   by Algorithm 4.4.1 with parameters $(c, C, t, S)$.
2. Let $B' = \{B'_1, B'_2, \ldots\}$ denote the output.
   Translate index set $B'$ to coordinate set $B = \{C_{B'_1}, C_{B'_2}, \ldots\}$.
```

□

Corollary 4.4.1. $M_{RAP}(c, S, t) \leq |S|$

Proof. Combine Theorem 4.4.1 and Corollary 4.2.2, the corollary is obtained. □

Corollary 4.4.2. $M_{RAP}(c, C, t) \leq \lfloor |C|/2 \rfloor + t + 3$ when $4t \leq |C|$.

Proof. By Theorem 4.4.1 and Theorem 4.3.2, the corollary is obtained. □

4.5 Trichotomy Reaction Attack Algorithm

By Algorithm 4.6.1 and Algorithm 4.7.5 and the idea of Algorithm 4.3.1, we have designed a new reaction attack algorithm.

```
Algorithm 4.5.1: New Reaction Attack Algorithm

Input: Ciphertext $c$, public-key $(G', t)$ in the MEPKC.
Output: Plaintext $m$ corresponds to $c$.

1. Select $k$ linearly independent columns in $G'$, denote the indexes of these columns
   by a set $C$.
2. Combine the $k$ columns of $G'$ to form a $k \times k$ matrix $G_k$.
3. Run Algorithm 4.3.5 with parameters $(k, t)$.
   When the algorithm queries the CWO with sample set $S$, simulate the response
   by Algorithm 4.4.1 with parameters $(c, C, t, S)$.
4. Let $B$ denote the output, flip $c$ on coordinates $\{C_{B_1}, C_{B_2}, \ldots, C_{B_{|B|}}\}$.
5. Combine the $k$ coordinates of $c$ which selected in step1 to form a $1 \times k$ vector $c'$.
6. $m = c' \cdot G'^{-1}_k$
```

In practical MEPKC instances, $4t \leq k$ is hold. For example, the original parameters suggested by McEliece [25] where $n = 1024, t = 50$ and $k \geq 524$ consist with this inequality. Moreover, to maximize the expected work factor of the attack proposed by McEliece himself, Adams and Meijer [1] suggested parameters $n = 1024, t = 37$ and $k = 654$. This suggestion also consists with $4t \leq k$. When $4t \leq k$ holds, by Corollary 4.4.2, Algorithm 4.5.1 requires at most $\lfloor k/2 \rfloor + t + 3$ queries.

By connecting the reaction attack and the counterfeit coins puzzle, this chapter gives a new direction to improve the attack. That is, if someone is good at playing the puzzle, his playing strategy will induce a better reaction attack algorithm against the MEPKC.

Chapter 5

Conclusion

In this thesis, two improvements of the general-information-decoding algorithm are proposed (see Section 3.1.4). One improvement decreases the time consumption of the guess-verification stage, and the other improvement eliminates the duplicate guesses of the nearly error-free locations.

We also introduce a new connection between the trichotomy reaction attack and the comparative counterfeit coins problem (see Chapter 4). With this connection, a high-performance playing strategy of the CCCP induce a high-performance attack algorithm to break the improper implementation of MEPKC. This work extend the directions to increase the feasibility of attacks under the trichotomy reaction oracle model.

After the connection is established, we propose a greedy approach to solve the (n, d)-CCCP. The worst-case queries requirement of it is proven to be $\lfloor n/2 \rfloor + d + 3$ when $4d \leq n$. With this algorithm, a trichotomy reaction attack algorithm with worst-case queries requirement $\lfloor k/2 \rfloor + t + 3$ is induced. We show an arrangement of the queries requirement of three attacks: [29], [13], and the trichotomy reaction attack in this thesis in the following table.

	Chosen-Cipher Attack	Reaction Attack	Trichotomy Reaction Attack
Oracle Model	decryption oracle	reaction oracle	trichotomy reaction oracle
# of Queries	$\frac{n(n-1)}{2(t-1)(n-t)}$ (1)	$n + 2t$ (2)	$\lfloor \frac{k}{2} \rfloor + t + 3$ (3)

(1)	:	average case.
(2)	:	worst case.
(3)	:	worst case, when $4t \leq k$.

Table 5.1: An arrangement of queries requirement

Bibliography

[1] C. M. Adams and H. Meijer. Security-related comments regarding McEliece's public-key cryptosystem. *IEEE Transactions on Information Theory*, 35(2):454–455, 1989.

[2] M. Ajtai and C. Dwork. A public-key cryptosystem with worst-case/average-case equivalence. In *STOC '97: Proceedings of the twenty-ninth annual ACM symposium on Theory of computing*, pages 284–293, 1997.

[3] E. R. Berlekamp. Goppa codes. *IEEE Transactions on Information Theory*, 19:590–592, 1973.

[4] E.R. Berlekamp, R.J. McEliece, and H.C.A. van Tilborg. On the inherent intractability of certain coding problems. *IEEE Transactions on Information Theory*, 24:384–386, 1978.

[5] T. A. Berson. Failure of the McEliece public-key cryptosystem under message-resend and related-message attack. In *CRYPTO*, pages 213–220, 1997.

[6] A. Canteaut and N. Sendrier. Cryptoanalysis of the original McEliece cryptosystem. In *ASIACRYPT*, pages 187–199, 1998.

[7] A. E. F. Clementi, A. Monti, and R. Silvestri. Selective families, superimposed codes, and broadcasting on unknown radio networks. In *SODA*, pages 709–718, 2001.

[8] D. Z. Du and F. K. Hwang. *Combinatorial Group Testing And Its Applications 2nd Edition.* 2000.

[9] G. Ehrlich. Algoruithm 466. four combinatorial algorithms. *Commum. ACM*, pages 691–691, 1973.

[10] G. Ehrlich. Loopless algorithms for generating permutations, combinations, and other combinatorial configurations. *J. ACM*, 20(3):500–513, 1973.

[11] O. Goldreich, S. Goldwasser, and S. Halevi. Eliminating decryption errors in the ajtai-dwork cryptosystem. In *CRYPTO '97: Proceedings of the 17th Annual International Cryptology Conference on Advances in Cryptology*, pages 105–111, 1997.

[12] V. D. Goppa. A new class of linear correcting codes. *Probl. Pered. Info.*, 6(3):24–30, 1970.

[13] C. Hall, I. Goldberg, and B. Schneier. Reaction attacks against several public-key cryptosystems. In *the 2nd International Conference on Information and Communications Security (ICICS'99), LNCS 1726*, pages 2–12, 1999.

[14] T. Hwang and T. R. N. Rao. Secret error-correcting codes (secc). In *CRYPTO*, pages 540–563, 1988.

[15] P. Indyk. Explicit constructions of selectors and related combinatorial structures, with applications. In *SODA*, pages 697–704, 2002.

[16] R. M. Karp. On the computational complexity of combinatorial problems. *Networks.*, pages 45–68, 1975.

[17] K. Kobara and H. Imai. Countermesures against all the known attacks to the McEliece PKC. In *International Symposium on Information Theory and Its Applications.*, pages 661–664, 2000.

[18] K. Kobara and H. Imai. Semantically secure McEliece public-key cryptosystems-conversions for McEliece PKC. In *Public Key Cryptography*, pages 19–35, 2001.

[19] P. J. Lee and E. F. Brickell. An observation on the security of McEliece's public-key cryptosystem. In *EUROCRYPT*, pages 275–280, 1988.

[20] J. S. Leon. A probabilistic algorithm for computing minimum weights of large error-correcting codes. *IEEE Transactions on Information Theory*, 34(5):1354–, 1988.

[21] Rudolf Lidl and Harald Niederreiter. *Finite fields*, volume 20 of *Encyclopedia of Mathematics and its Applications*. Cambridge University Press, second edition, 1997.

[22] P. Loidreau. Strengthening McEliece cryptosystem. In *ASIACRYPT*, pages 585–598, 2000.

[23] P. Loidreau. Codes derived from binary Goppa codes. *Probl. Inf. Transm.*, 37(2):91–99, 2001.

[24] F. J. MacWilliams and N. J. A. Sloane. *The theory of error-correcting codes. I and II.* North-Holland Publishing Co., 1977.

[25] R.J. McEliece. A public-key cryptosystem based on algebraic coding theory. *DSN Progress Report*, pages 114–116, 1978.

[26] C. Savage. A survey of combinatorial gray codes. *SIAM Rev.*, 39(4):605–629, 1997.

[27] H. M. Sun. Improving the security of the McEliece public-key cryptosystem. In *ASIACRYPT*, pages 200–213, 1998.

[28] H. M. Sun. Enhancing the security of the McEliece public-key cryptosystem. *J. Inf. Sci. Eng.*, 16(6):799–812, 2000.

[29] H. M. Sun. Further cryptanalysis of the McEliece public-key cryptosystem. *IEEE Communications Letters.*, 4(1):18–19, 2000.

[30] J. van Tilburg. On the McEliece public-key cryptosystem. In *CRYPTO*, pages 119–131, 1988.