More Than Error Correction: Cryptography from Codes

by

Yao Chen

A thesis
presented to the University of Waterloo
in fulfillment of the
thesis requirement for the degree of
Master of Applied Science
in
Electrical and Computer Engineering

Waterloo, Ontario, Canada, 2014

Abstract

The first code-based cryptosystem, McEliece, was invented in the very early development of public-key cryptography, yet code-based cryptosystems received little attention for decades due to their relatively large key-sizes. But recently they are re-discovered for their potentials to provide efficient post-quantum cryptographic tools and homomorphic encryption schemes, and the development of large storage and fast Internet have made these schemes closer to practice than ever.

Through our review of the revolution of code-based cryptography, we will demonstrate the usage of codes in cryptographic applicaitons. We will follow the path of the development, from the design, analysis, and implementation of McEliece cryptosystem and the quantum attack resistance to the latest fully homomorphic encryption scheme based on Learning with Errors, a code-related problem, designed by Brakerski *et al.* We will also cover algebraic manipulation detection codes, a newly proposed extension of error-correcting codes and a lightweight alternative to MACs as an authentication component embedded in security protocols.

Acknowledgements

I would like to thank my supervisor, Prof. Guang Gong, for her guidance, encouragement, and care in both my academics and life. She led me to this fascinating area with her vision and experience and guided me through difficulties. She also provided me with the opportunity to join the CryptoWorks21 training program so that I am able to gain better understanding on quantum computation and touch the state of the art.

I also feel grateful to Dr. Yin Tan, who organized the reading group on coding theory, discussed a lot of ideas with me, and corrected me on mathematical reasonings. Without his help, it would be very difficult for me to explore the world of codes.

Special thanks to Teng Wu, who delivered the idea of AMD codes and non-malleable codes in his seminar and thus enabled me to discover more about them.

Finally, I would also like to thank my other colleagues and friends for their support throughout my master's program.

Dedication

To my parents, who always support me to chase my dream.

Table of Contents

List of Tables

List of Figures

Chapter 1

Introduction

Code-based cryptosystem was born during the earliest discoveries of public-key cryptography in the 1970s, marked by its representative McEliece cryptosystem [McE78]. Niederreiter soon developed a variant [Nie86], which can be adapted into a digital signature scheme. These cryptosystems garble the coding matrices, making it indistinguishable from a random code. Their security is hence based on the difficulty of decoding random codes (or "general decoding"), which is an **NP**-hard problem [BMVT78].

The property that McEliece and related cryptosystems can be based on **NP**-hard problems are favourable, as the exact hardness of other problems such as factoring and discrete logarithm is not yet known. Unfortunately, their relatively large key sizes have become the major disadvantage compared with other public-key cryptosystem candidates. While people widely deploy RSA and discrete-logarithm-based cryptosystems, code-based ones have gained little popularity. Therefore, a number of McEliece variants have been designed to reduce the key size. The main technology is to replace the Goppa code underlying McEliece and Niederreiter by other codes. Most of such variants have been broken [FOPT10, UL10, CGGU$^+$13], but Goppa-code-based ones have remained secure.

Recently, we started to see more research on code-based cryptography for three reasons. Firstly, the disadvantage brought by large key size is no longer fatal thanks to more advanced storage and transmission technology. Secondly, the study on quantum computation shows that the widely deployed public-key cryptosystems can all be efficiently broken by quantum computers [Sho94], but code-based ones are immune to known quantum attacks. They become a candidate of post-quantum cryptography since then. Last but not least, codes naturally support some homomorphic operations. The emergence of the BGV scheme [BGV12] has proven that codes are very promising in fully homomorphic encryption

(FHE) schemes.

The main approach of using codes in the design of FHE schemes is through the Learning with Errors (LWE) problem. This problem can be seen as a variant of general decoding over the ring of integers [Reg09]. LWE itself also has a few extensions such as Ring-LWE (RLWE). LWE and its extensions have been shown to be at least as hard as hard lattice problems, another source of post-quantum cryptosystem and FHE schemes, through quantum and classical reductions [Reg09, Pei09, BLP+13]. Being simpler and faster in applications [GSW13], LWE has seen its rise in popularity among designs in encryption schemes with various properties. The LWE-based BGV scheme, already capable of evaluating complex functions such as AES encryption [GHS12], has pushed FHE closer to its practical application than ever.

In addition to directly applying the concept of error-correcting codes in cryptography, researchers also attempted to adapt this concept to suit a situation with adversaries. Proposed by Cramer *et al.*, AMD code is an example of such new primitives [CDF+08]. Though its first appearance was in 2008, it already has many applications. For example, it can be added to linear secret-sharing schemes as a means of authentication; it is also the basis of some designs of secure storage device [WK11] and non-malleable codes [DPW10].

Contribution and organization In this thesis, we will summarize the latest development as well as review the history of code-based cryptography. In particular, some state-of-the-art research such as the BGV encryption scheme is demonstrated in a more accessible way. We compare the performance of RSA, McEliece, and BGV schemes by simulation data, giving an intuitive understanding of their differences. We also provide a new bound and related constructions for a class of AMD codes.

This thesis is organized following the timeline of code-based cryptography. In Chapter 2, we review McEliece cryptosystem, implement it with Goppa code, and compare its performance against RSA using the current parameters. Chapter 3 explains quantum Fourier transform (QFT), Shor's algorithm, and the resistance of code-based schems against quantum attacks. Fully homomorphic encryption (FHE) is discussed in Chapter 4, with emphasis in BGV scheme based on Learning with Errors (LWE). Chapter 5 summarizes known results and applications of a recently proposed cryptographic primitive, AMD code, and gives our bound and constructions. Chapter 6 concludes the thesis and provides some directions for future research along this line.

Chapter 2

McEliece Cryptosystem: Design and Implementation

2.1 Background

Code-based cryptosystems can be dated back to McEliece's invention in 1978, roughly the same time RSA were announced. Usually based on an error-correcting code with long length n and large dimension k , McEliece cryptosystem has a huge public key size, due to its public key being a generator matrix with dimension $k \times n$. As a result, McEliece cryptosystem was not popular in its early years.

But recently McEliece cryptosystem regains some attention for two reasons:

1. As storage space becomes cheaper, large public key size is no longer a serious restriction. Meanwhile, the advancement of attacks against RSA has forced people to employ larger RSA keys to maintain the same security level. The simplicity of encoding and decoding will eventually make McEliece surpass RSA on the performance at a higher security level.

2. The general decoding problem, which McEliece cryptosystem relies on, is an **NP**-hard problem [BMVT78] without known quantum attacks.

In this chapter we will review the current development of McEliece cryptosystem, and then demonstrate an implementation and its performance.

2.2 The McEliece cryptosystem

In this section we describe the algorithms of McEliece cryptosystem [McE78]. We assume that an underlying code is given along with its encoding and decoding algorithms. Essentially any error-correcting code can be chosen to construct a McEliece cryptosystem, but the choices are usually restricted to a small class due to security concerns. We will cover the security issues in Section 2.3.

`KeyGen` `KeyGen` will generate a garbled generator matrix as the public key:

1. Choose a binary linear code C, specify the parameters (n, k, t) in particular, according to the security parameter l. Obtain the generator matrix G.
2. Randomly choose a (dense) invertible matrix $S \in \mathbb{Z}_2^{k\times k}$ and a permutation matrix $P \in \mathbb{Z}_2^{n\times n}$. Compute $G' = SGP$.
3. Set the public key $pk = G'$ and private key $sk = (S, G, P)$.

`Enc` Encryption is simply encoding with the public key (garbled generator matrix G') with t errors added:

1. Generate a random error vector $\mathbf{e}$ with $\mathrm{wt}(\mathbf{e}) = t$.
2. Encrypt by $\mathbf{c} = \mathbf{m}G' + \mathbf{e}$.

`Dec` In order to use the decoding algorithm, we need to recover the structure of the code. Therefore the decryption steps are

1. Undo the permutation: $\mathbf{c}' = \mathbf{c}P^{-1}$.
2. Decode $\mathbf{c}'$ to obtain $\mathbf{m}'$.
3. $\mathbf{m} = \mathbf{m}'S^{-1}$.

It is easy to verify the correctness. Notice that P^{-1} preserves the weight of the error vector, $\mathbf{c}'$ can therefore be decoded into $\mathbf{m}' = \mathbf{m}S$. Then we obtain $\mathbf{m}$ by $\mathbf{m}'S$.

Example 2.2.1 (A toy example of McEliece algorithms)**.** Below is a generator matrix of an $[8, 3]$-Goppa code with $n = 8, k = 3$, and $t = 2$.

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

- `KeyGen`: we generate random non-singular matrix

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

  and random permutation matrix

  $$P = \begin{bmatrix} 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 \end{bmatrix}.$$

  Then the public key is

  $$pk = G' = SGP = \begin{bmatrix} 1 & 0 & 0 & 1 & 1 & 1 & 1 & 0 \\ 1 & 1 & 1 & 0 & 0 & 1 & 1 & 1 \end{bmatrix},$$

  and the secret key is composed of the matrices S, G, and P.

- `Enc`: suppose the plaintext message is

  $$\mathbf{m} = \begin{bmatrix} 0 & 1 \end{bmatrix},$$

  and we generate random error vector (with weight $t = 2$)

  $$\mathbf{r} = \begin{bmatrix} 0 & 1 & 0 & 1 & 0 & 0 & 0 & 0 \end{bmatrix},$$

  then the ciphertext is given by

  $$\mathbf{c} = \mathbf{m}G' + \mathbf{e} = \begin{bmatrix} 1 & 0 & 1 & 1 & 0 & 1 & 1 & 1 \end{bmatrix}.$$

- `Dec`: first the ciphertext goes through the inverse permutation:

  $$\mathbf{c}' = \mathbf{c}P^{-1} = \begin{bmatrix}1 & 1 & 0 & 1 & 1 & 1 & 0 & 1\end{bmatrix}.$$

  The decoding of $\mathbf{c}'$ gives

  $$\mathbf{m}' = \begin{bmatrix}1 & 1\end{bmatrix}$$

  . After applying S^{-1}, we successfully obtain

  $$m = \begin{bmatrix}0 & 1\end{bmatrix}.$$

2.3 Security analysis

There are two ways to break McEliece cryptosystem:

1. Without structural information, decoding a ciphertext as the codeword of a random code. The goal of such attacks is to search for the plaintext.

2. Trying to recover the structure from the garbled underlying code. The secret key can potentially be revealed in this case.

The most successful method in the first category is a class of *information set decoding* algorithms. They apply to all McEliece instances. For McEliece cryptosystem based on Goppa codes, these algorithms are still the best known attacks so far.

Attacks in the second category have been discovered for certain McEliece cryptosystem designed to reduce the public key size. See [FOPT10] and [UL10] for attacks against quasi-cyclic and dyadic McEliece cryptosystem, and [CGGU+13] for McEliece cryptosystem based on generalized Reed-Solomon codes. These attacks are all able to recover the secret key.

2.3.1 General decoding attacks

Information set decoding was introduced by Prange in 1962 [Pra62] and is currently the most effective method to decode random codes. Informally speaking, an "information set" is a set of coordinates that can uniquely determine a codeword of linear code C. The components in a codeword can be divided into information bits (whose indices are in the information set) and redundancy bits (whose indices are out of the information set).

Generally, information set decoding algorithms solve the following low-weight codeword problem (Definition 2.3.1). Some algorithms accepts generator matrices as input, others work on parity-check matrices, but the ideas are similar. We will explain it below.

Definition 2.3.1 (Low-weight codeword problem)**.** Given a binary linear code C by its generator matrix G (or parity-check matrix H) and a weight w, the low-weight codeword problem is to find a codeword $\mathbf{c} \in C$ with $\mathrm{wt}(c) \leq w$.

The plaintext-recovery attack against McEliece cryptosystems can be formalized as the decoding of random codes (Definition 2.3.2).

Definition 2.3.2 (General decoding problem)**.** Given a binary linear code $C \subset \mathbb{Z}_2^n$ by its generator matrix G (or parity-check matrix H) and a vector $\mathbf{v} \in \mathbb{Z}_2^n$, the general decoding problem is to find a codeword $\mathbf{c} \in C$ such that the distance between $\mathbf{v}$ and $\mathbf{c}$ is minimum, i.e.,

$$\mathbf{c} = \arg\min_{\mathbf{c}' \in C} \mathrm{wt}(\mathbf{v} - \mathbf{c}').$$

The low-weight codeword problem is slightly different from the general decoding problem, but the latter can be easily reduced to the former by adding the vector $\mathbf{v}$ as a codeword, i.e., modifying the generator matrix G into

$$G' = \begin{bmatrix} G \\ \hline \mathbf{v} \end{bmatrix}.$$

Note that we can obtain G from H and vice versa, but the input must match the algorithm.

Information set decoding algorithms

We first demonstrate the idea behind information set decoding algorithms with a simple example.

Consider a generator matrix of a binary linear code C in its systematic form

$$G_{sys} = \left[I_k \mid A_{k \times (n-k)} \right].$$

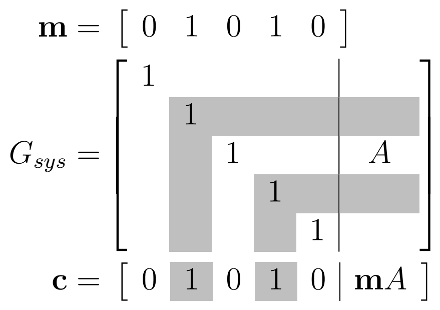
For any codeword $\mathbf{c} \in C$, we have

$$\mathbf{c} = \mathbf{m} G_{sys} = \left[\mathbf{m} \mid \mathbf{m}A \right].$$

Therefore $\mathrm{wt}(\mathbf{c}) = \mathrm{wt}(\mathbf{m}) + \mathrm{wt}(\mathbf{m}A)$, and $\mathbf{m}A$ is exactly the sum of A's rows corresponding to the positions of 1's in $\mathbf{m}$ (see Figure 2.1 for an illustration). Since we want to bound $\mathrm{wt}(\mathbf{c})$ by w, one strategy can be as follows:

1. Let $w = x + y$
2. Search for combinations of at most x rows in A such that the sum of them has weight no more than y
3. If the search successfully returns a set of rows, the desired codeword can be constructed accordingly.

This strategy is probabilistic, i.e., the search can be unsuccessful. However, once the search returns a combination, it guarantees that $\mathrm{wt}(\mathbf{m}) \leq x$ and $\mathrm{wt}(\mathbf{m}A) \leq y$, and we can conclude that $\mathrm{wt}(\mathbf{c}) \leq x + y = w$.



$\mathbf{m} = [\,0\ 1\ 0\ 1\ 0\,]$
$G_{sys} =$
1
1
1
1
1
A
$\mathbf{c} = [\,0\ 1\ 0\ 1\ 0 \mid \mathbf{m}A\,]$

Figure 2.1: Correspondance of A's rows to positions of 1's in $\mathbf{m}$

Based on the above idea, we give a framework of information set decoding algorithms using generator matrix G in Algorithm 1. Line 5 to 6 (highlighted) are the flexible part of this framework. Different algorithms adopt different strategy to select x and search for candidate combinations to achieve an ideal trade-off between the searching complexity and succeeding probability.

Similarly, with parity-check matrix as input, we can first reduce it into a "systematic form" (it can be compared to the usual definition of systematic parity-check matrix in coding theory, but here the positions of the two components are switched)

$$H_{sys} = \left[I_{n-k} \mid (A_{k\times(n-k)})^\top \right].$$

By the definition of parity-check matrix and codeword,

$$\mathbf{c}H_{sys}^\top = \mathbf{c}\begin{bmatrix} I_{n-k} \\ \hline A_{k\times(n-k)} \end{bmatrix} = [\mathbf{c}_0 \mid \mathbf{c}_1]\begin{bmatrix} I \\ \hline A \end{bmatrix} = \mathbf{c}_0 I + \mathbf{c}_1 A = \mathbf{0},$$

Algorithm 1: Information set decoding with generator matrix

```
  Input: Generator matrix $G$, weight upper bound $w$
  Output: A codeword $\mathbf{c} \in C$ such that $\mathrm{wt}(\mathbf{c}) \leq w$, or an indication that the search
          is unsuccessful
  // Randomly permute columns of $G$
1 $P \leftarrow$ random permutation matrix;
2 $G' \leftarrow GP$;
3 if Gaussian elimination transforms $G'$ into a systematic form then
4     $\left[ I \mid A \right] = G_{sys} \leftarrow SG'$, where $S$ represents the row transformations;
5     Select $x$ and let $y \leftarrow w - x$;
6     Search for combinations of $x$ rows in $A$, and let candidate combinations form a
      set $T$;
7     foreach combination in $T$ do
8         Construct codeword $\mathbf{c}$;
9         if $\mathrm{wt}(\mathbf{c}) \leq w$ then
              // recover the order of bits
10            return $\mathbf{c}P^{-1}$;
11        end
12    end
13    return "Search unsuccessful.";
14 else
15    Go back to Line 1;
16 end
```

where $\mathbf{c}_0 = \left[c_0, c_1, \ldots, c_{n-k-1}\right]$ and $\mathbf{c}_1 = \left[c_{n-k}, c_{n-k+1}, \ldots, c_{n-1}\right]$. Then we have the relationship $\mathrm{wt}(\mathbf{c}_1 A) = \mathrm{wt}(\mathbf{c}_0 I) = \mathrm{wt}(\mathbf{c}_0)$ and $\mathrm{wt}(\mathbf{c}) = \mathrm{wt}(\mathbf{c}_0) + \mathrm{wt}(\mathbf{c}_1) \leq w$, and we can apply the strategy of splitting w into $x + y$ and search for x rows in A whose sum have a weight y. Now the same strategy applies.

We briefly mention the attack proposed by Bernstein *et al.* below. For other algorithms, Chabaud made a very concise summary of those proposed before 1995 in [Cha95]; the most up-to-date generic decoding algorithm for binary linear codes can be found in [BJMM12].

The attack of Bernstein *et al.* and parameter selection

Based on Stern's algorithm [Ste89], Bernstein *et al.* propose the best known attack against McEliece cryptosystem [BLP08].

Stern's algorithm adopts a meet-in-the-middle approach. It randomly divides the columns of $A^\top$ (rows of A) into two subsets X and Y, and also chooses a set Z of l rows:

$$H_{sys} = \left[\begin{array}{c|c} \quad I \quad & \begin{array}{c|c} X & Y \end{array} \\ & \overline{\qquad Z \qquad} \end{array}\right]$$

search all p column combinations in both X and Y, and check if any pair of combination each from X and Y has a sum of weight $w - 2p$. Collisions are found by examining l bits in the Z set.

Example 2.3.1 (A toy example of Stern's algorithm)**.** H is a parity-check matrix of $[7, 4]$-Hamming code:

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

We set the parameters $w = 3$, $p = 1$, and $l = 1$. Stern's algorithm starts by obtaining a systematic form of H with random permutation of columns and Gaussian elimination:

$$\begin{array}{cc} & \begin{array}{ccccccc} & & & X & & Y & \\ & & & \uparrow & & \uparrow & \end{array} \\ H_{sys} = & \left[\begin{array}{ccc|cc|cc} 1 & 0 & 0 & 1 & 1 & 0 & 1 \\ 0 & 1 & 0 & 1 & 0 & 1 & 1 \\ \hline 0 & 0 & 1 & 0 & 1 & 1 & 1 \end{array}\right] \rightarrow Z \end{array}$$

List the sums of all p-column combinations in set X:

$$\mathbf{x}_0 = \begin{bmatrix}1\\1\\0\end{bmatrix}, \quad \mathbf{x}_1 = \begin{bmatrix}1\\0\\1\end{bmatrix}.$$

Similarly for set Y

$$\mathbf{y}_0 = \begin{bmatrix}0\\1\\1\end{bmatrix}, \quad \mathbf{y}_1 = \begin{bmatrix}1\\1\\1\end{bmatrix}.$$

For each pair $(\mathbf{x}_i, \mathbf{y}_j)$, examine their third components (rows indexed by set Z). A collision is found if they are identical. We can find two collisions $(\mathbf{x}_i, \mathbf{y}_j)$ in this example and further calculate the sum $\mathbf{x}_i + \mathbf{y}_j$:

$$\mathbf{s}_0 = \mathbf{x}_1 + \mathbf{y}_0 = \begin{bmatrix}1\\1\\0\end{bmatrix}, \quad \mathbf{s}_1 = \mathbf{x}_1 + \mathbf{y}_1 = \begin{bmatrix}0\\1\\0\end{bmatrix}.$$

The weight of $\mathbf{s}_1$ is 1, which is exactly $w - 2p$. Therefore we have the following codeword $\mathbf{c}$ of weight $w = 3$, formed by the sum $\mathbf{s}_1^\top$ as the first component and 1's corresponding to columns sum up to $\mathbf{s}_1$ in the rest of it.

$$\mathbf{c} = \left[\begin{array}{ccc|cc|cc} 0 & 1 & 0 & 0 & 1 & 0 & 1 \end{array}\right]$$

$$H_{sys} = \left[\begin{array}{ccc|cc|cc} 1 & 0 & 0 & 1 & 1 & 0 & 1 \\ 0 & 1 & 0 & 1 & 0 & 1 & 1 \\ \hline 0 & 0 & 1 & 0 & 1 & 1 & 1 \end{array}\right] \rightarrow \mathbf{s}_1 = \begin{bmatrix}0\\1\\0\end{bmatrix}$$

Bernstein *et al.* found a few improvements on the efficiency of Stern's algorithm by reducing redundant computations, most of which lie in the Gaussian elimination step (see their paper [BLP08, Section 4] for a complete list). Based on their theoretical analysis and numerical experiments, taking into consideration list decoding algorithm of Goppa codes, they suggested codes of rate approximately 0.75 to best resist their attack and particularly parameters in Table 2.1.

2.4 Implementation

2.4.1 McEliece cryptosystem

This implementation is based on binary Goppa codes, described in Section 2.4.2. In this section we only describe the algorithms for McEliece cryptosystem supposing that

Security (bits)	n	k	t	No. of errors	Notes
80	1632	1269	33	34	
	2048	1751	27	27	Without list decoding
128	2960	2288	56	57	
256	6624	5129	115	117	
84.88	1744	1359	35	36	Key size bounded by powers of 2; k was not given in their paper
107.41	2480	1940	45	46	
147.94	3408	2604	67	68	
191.18	4624	3389	95	97	
266.94	6960	5413	119	121	

Table 2.1: Security parameters suggested by Bernstein *et al.* [BLP08]

encoding and decoding algorithms of a code are given.

Function RandomPermutationMatrix(n) [Knu97]

```
for $i \leftarrow 0, \dots, n-1$ do
    $a_i \leftarrow i$;
end
for $i \leftarrow 0, \dots, n-2$ do
    $j \leftarrow_R \{i, \dots, n-1\}$;
    Swap $a_i$ and $a_j$;
end
$P \leftarrow n \times n$ zero matrix $\mathbf{0}$;
for $i \leftarrow 0, \dots, n-1$ do
    $P_{i,a_i} \leftarrow 1$;
end
return $P$;
```

Lemma 2.4.1. *The function RandomInvertiblematrix is expected to terminate within 4 rounds.*

Proof. Note that in every round, a matrix is uniformly sampled from $\mathbb{F}_2^{n \times n}$. The probability for a uniformly random $n \times n$ matrix over $\mathbb{F}_2$ to be invertible is

$$p_n \triangleq \Pr[\det(A_{n\times n}) = 1] = \prod_{k=1}^{n}(1 - 2^{-k}) \text{ [Mor06]}. \tag{2.1}$$

Function RandomInvertiblematrix(n)

```
repeat
    for $i \leftarrow 0, \dots, n-1$ do
        for $j \leftarrow 0, \dots, n-1$ do
            $A_{i,j} \leftarrow_R \{0, 1\}$;
        end
    end
until $A$ is invertible;
return $A$;
```

The sequence $\{p_i\}_{i=1}^{n}$ monotonically decreases as i increases and has a limit of approximately 0.289. The number of runs is distributed according to a geometric distribution with $p \gtrapprox 0.288$, which has an expected value $1/p \lessapprox 3.472$. □

Algorithm 2: KeyGen

```
Input: $\mathbb{1}^l$, the unitary representation of the security parameter $l$
Output: Public key $pk$, private key $sk$
// See Section 16
$(n, k, t) \leftarrow$ choice of parameters according to the underlying code;
Generate the generator matrix $G$ for the select code;
$S \leftarrow$ RandomInvertiblematrix($k$);
$P \leftarrow$ RandomPermutationMatrix($n$);
$pk \leftarrow SGP$;
$sk \leftarrow (S^{-1}, P^{-1})$;
```

2.4.2 Goppa codes

Goppa codes were introduced by Goppa in his paper *A New Class of Linear Correcting Codes* [Gop70]. However, the original publication was in Russian, therefore we will refer to Berlekamp's summary [Ber73] for the definition of Goppa codes.

Definition

A Goppa code is defined in terms of a *Goppa polynomial* $g(x) \in \mathbb{F}_{q^m}[x]$. Let L be the subset of $\mathbb{F}_{q^m}$ consisting of the elements that are not roots of $g(x)$. Suppose $|L| = n$ and we can number the elements of L as $\gamma_0, \gamma_1, \ldots, \gamma_{n-1}$, then the codewords of this Goppa code $\mathbf{c} = [c_0, c_1, \ldots, c_{n-1}]$ satisfies the equation

$$\sum_{i=0}^{n-1} \frac{c_i}{x - \gamma_i} \equiv 0 \pmod{g(x)}. \tag{2.2}$$

Lemma 2.4.2 establishes a lower bound for the minimum distance of Goppa codes.

Lemma 2.4.2. *If a Goppa code is defined by $g(x)$ with degree t, then the minimum distance is at least $t + 1$ [SKHN75].*

Binary Goppa codes are simply Goppa codes defined over $\mathbb{F}_{2^m}$, i.e., the special case where $q = 2$. Binary Goppa codes have a favourable property that efficient decoding algorithms are known to correct up to t errors (i.e., the minimum distance is at least $2t + 1$), where $t = \deg(g)$, as stated in Lemma 2.4.3.

Lemma 2.4.3. *If $g(x) \in F_{2^m}[x]$ with degree t and no repeated irreducible factors, then the algebraic decoding algorithm (described in Section 2.4.2) can correct up to t errors for the binary Goppa code defined by $g(x)$ [Ber73].*

For simplicity, $g(x)$ is often chosen to be an irreducible polynomial over $\mathbb{F}_{2^m}$ so that all the elements of $\mathbb{F}_{2^m}$ can be used to construct the Goppa code, and in this case the length of the code is $n = 2^m$.

Algebraic decoding of binary Goppa codes

According to (2.2) in the definition of Goppa codes, the syndrome polynomial $S(x)$ is affected only by errors:

$$S(x) = \sum_{i=0}^{n-1} \frac{r_i}{x - \gamma_i} \tag{2.3}$$

$$= \sum_{i=0}^{n-1} \frac{c_i + e_i}{x - \gamma_i} \tag{2.4}$$

$$\equiv \sum_{i=0}^{n-1} \frac{e_i}{x - \gamma_i} \pmod{g} \tag{2.5}$$

Define error-locator polynomial $\sigma(x)$ as the polynomial with roots corresponding to error locations exactly,

$$\sigma(x) = \prod_{i=0}^{n-1} (x - \gamma_i)^{e_i}. \tag{2.6}$$

It is easy to see that

$$\sigma' \equiv S\sigma \pmod{g}. \tag{2.7}$$

(2.7) is often referred to as the *key equation* of Goppa code. The error-locator polynomial $\sigma(x)$ can be obtained by solving the key equation.

If we write

$$\sigma(x) = \alpha^2(x) + x\beta^2(x), \tag{2.8}$$

then

$$\sigma'(x) = \beta^2(x), \tag{2.9}$$

since the derivative of squares are eliminated in the polynomial ring over the binary field $\mathbb{F}_{2^m}$. Therefore we have

$$\beta^2(x) \equiv (\alpha^2(x) + x\beta^2(x))S(x) \pmod{g(x)}, \tag{2.10}$$

$$\beta^2(x)(S^{-1}(x) + x) \equiv \alpha^2(x) \pmod{g(x)}. \tag{2.11}$$

Let $T^2(x) \equiv S^{-1}(x) + x \pmod{g(x)}$, then

$$\beta T \equiv \alpha \pmod{g}. \tag{2.12}$$

If we can find a solution (α, β) to (2.12) satisfying $\deg(\alpha) \leq \lfloor r/2 \rfloor$ and $\deg(\beta) \leq \lfloor (r-1)/2 \rfloor$, then the error-locator polynomial σ can be constructed using (2.8).

Algorithms

In this section we give pseudocode of Goppa code algorithms.

Since Goppa codes are defined by parity-check matrix H, we will generate H first, and then obtain generator matrix G by transforms of H.

Given parity-check matrix H, we can transform it into its *systematic form* $H' = \left[A^\top | I_{n-k}\right]$, and then the corresponding generator matrix G can be obtained by $G = \left[I_k | A\right]$ (note that we are dealing with binary codes. For q-ary codes it should be $G = \left[I_k | -A\right]$). We obtain the systematic parity-check matrix H' purely for the creation of generator matrix G; specifically, any column swap during the transform of H to H' must be undone on G, because the order of $\gamma_i \in \mathbb{F}_{2^m}$ is crucial in the algebraic decoding of Goppa codes.

Next we give the decoding algorithm, along with a few auxiliary functions.

Algorithm 3: Generating the parity-check matrix H

```
Input: $m, t$
Output: Goppa polynomial $g$, parity-check matrix $H$
1 $g \leftarrow$ random irreducible polynomial of degree $t$ over $\mathbb{F}_{2^m}$;
2 $n \leftarrow 2^m$;
3 for $i \leftarrow 0, \dots, n-1$ do
4     $\gamma_i \leftarrow$ $i$-th element of $\mathbb{F}_{2^m}$;
5     $\mathbf{H}'_i \leftarrow$ coefficients of $f(x)$, where $f(x)(x - \gamma_i) \equiv 1 \pmod{g(x)}$;
      // $\mathbf{H}'_i \in \mathbb{F}_{2^m}^t$
6     $\mathbf{H}_i \leftarrow$ components of $\mathbf{H}'_i$ written as vectors over $\mathbb{F}_2^m$;
      // $\mathbf{H}_i \in \mathbb{F}_2^{mt}$
7 end
8 $H \leftarrow \left[\mathbf{H}_0^\top, \dots, \mathbf{H}_{n-1}^\top\right]$;
```

Lemma 2.4.4. *Let $g \in \mathbb{F}_{2^m}[x]$ be an irreducible polynomial of degree t. Then for any $p \in \mathbb{F}_{2^m}[x]$, $p^{2^{mt-1}} \equiv \sqrt{p} \pmod{g}$.*

Proof. Irreducible polynomial g generates the field $\mathbb{F}_{2^{mt}}$. The result follows by viewing polynomial (p mod g) as an element of $\mathbb{F}_{2^{mt}}$. □

Lemma 2.4.5. *Function FindAlphaBeta finds a solution to $\beta T \equiv \alpha \pmod{g}$ satisfying both degree constraints*

1. $\deg(\alpha) \leq \lfloor t/2 \rfloor$, *and*
2. $\deg(\beta) \leq \lfloor (t-1)/2 \rfloor$.

Proof. Given a, b and d, the function DegreeConstraintXGCD returns a pair (u, v) such that $au + bv$ is a common divisor of a and b with degree no more than d.

Let $g = Tq_0 + r_0, \deg(r_0) < \deg(T)$. Since

$$\begin{aligned} 1 \cdot T &\equiv T \pmod{g}, \\ q_0 \cdot T &\equiv r_0 \pmod{g}, \end{aligned}$$

clearly $\alpha = uT + vr_0, \beta = u + vq_0$ is a solution to (2.12). It remains to show that the algorithm eventually gives $\beta = u + vq_0$ with $\deg(\beta) \leq \lfloor (t-1)/2 \rfloor$.

Algorithm 4: Generating the generator matrix G

```
   Input: $H$
   Output: Generator matrix $G$
   // Based on Gaussian elimination
1  $P \leftarrow I_{n-k}$, the identity matrix;
2  for $i \leftarrow 0, \ldots, n-k-1$ do
3      if $H_{i,i+k} = 0$ then
4          Find $p$ such that $H_{p,i+k} \neq 0$;
5          Swap $i$-th and $p$-th columns of $H$;
6          Swap $i$-th and $p$-th rows of $P$;
7      end
8      for $j \leftarrow i+1, \ldots, n-k-1$ do
9          if $H_{j,i+k} = 1$ then
10             Add $i$-th row to $j$-th row in $H$;
11         end
12     end
13     for $j \leftarrow 0, \ldots, i-1$ do
14         if $H_{j,i+k} = 1$ then
15             Add $i$-th row to $j$-th row in $H$;
16         end
17     end
18 end
   // Now $H$ becomes its systematic form $\left[A^\top | I_{n-k}\right]$
19 $G \leftarrow \left[I_k | A\right] (P^\top)^{-1}$;
```

Function PolynomialSquareRoot(p, g)

```
   // $p \in \mathbb{F}_{2^m}[x]/(g(x)) \cong \mathbb{F}_{2^{mt}}$, i.e., $p^{2^{mt-1}} \equiv \sqrt{p} \pmod{g}$
1  $r \leftarrow p$;
2  for $i \leftarrow 1, \ldots, mt-1$ do
3      $r \leftarrow r^2 \bmod g$;
4  end
5  return $r$;
```

Function DegreeConstraintXGCD(a, b, d, g)

```
if $\deg(a) < \deg(b)$ then
    $(u', v') \leftarrow$ DegreeConstraintXGCD$(b, a, d, g)$;
    $u \leftarrow v', v \leftarrow u'$;
    return $(u, v)$;
end
if $\deg(a) \leq d$ then
    return $(1, 0)$;
else
    Write $a$ as $bq + r$, where $\deg(r) < \deg(b)$;
    $(u', v') \leftarrow$ DegreeConstraintXGCD$(b, r, d, g)$;
    $u \leftarrow v' \bmod g$;
    $v \leftarrow (u' - v'q) \bmod g$;
    return $(u, v)$;
end
```

Function FindAlphaBeta(g, T)

```
Write $g$ as $Tq + r$, where $\deg(r) < \deg(T)$;
$(u, v) \leftarrow$ DegreeConstraintXGCD$(T, r, \lfloor \deg(g)/2 \rfloor, g)$;
$\alpha \leftarrow (uT + vr) \bmod g$;
$\beta \leftarrow (u + vq) \bmod g$;
return $(\alpha, \beta)$;
```

Algorithm 5: Decoding of Goppa codes

```
Input: Parity-check matrix $H$, received codeword $\mathbf{r}$, Goppa polynomial $g$
Output: Message $\mathbf{m}$
// Construct syndrome polynomial $S$
1  $s' \leftarrow \mathbf{r}H^\top$;
2  for $i \leftarrow 0, \ldots, t-1$ do
3      $s_i \leftarrow \left[s'_{mi}, s'_{mi+1}, \ldots, s'_{mi+t-1}\right]$, treated as an element of $\mathbb{F}_{2^m}$;
4  end
5  $S \leftarrow \sum_{i=0}^{t-1} s_i x^i$;
// Solve the key equation to find error-locator polynomial $\sigma$
6  $T \leftarrow$ PolynomialSquareRoot($S + x$);
7  $(\alpha, \beta) \leftarrow$ FindAlphaBeta($g, T$);
8  $\sigma \leftarrow \alpha^2 + x\beta^2$;
9  $\mathbf{e} \leftarrow \mathbf{0}$;
10 for $i \leftarrow 0, \ldots, 2^m - 1$ do
11     $\gamma_i \leftarrow$ $i$-th element of $\mathbb{F}_{2^m}$;
12     if $\sigma(\gamma_i) = 0$ then
13         $e_i \leftarrow 1$;
14     end
15 end
16 $\mathbf{c} = \mathbf{r} + \mathbf{e}$;
17 Solve $\mathbf{c} = \mathbf{m}G$ to get $\mathbf{m}$;
```

Let (α_i, β_i) be the corresponding solution (α, β) if the algorithm stops at i-th iteration.

Case 1: $i = 0$. In this case $(\alpha_0, \beta_0) = (T, 1)$, and $\deg(\beta_0) = 0$.

Case 2: $i \geq 1$. We show that for all $i \geq 1$, $\deg(\alpha_{i-1}) + \deg(\beta_i) = \deg(g)$ always holds by induction.

1. Basis:

$$\begin{aligned}\deg(\alpha_0) &= \deg(T) \\ \deg(\beta_1) &= \deg(q_0) = \deg(g) - \deg(T) \\ &= \deg(g) - \deg(\alpha_0)\end{aligned}$$

2. For $i \geq 2$. Let $\alpha_{i-2} = \alpha_{i-1}q_i + r_i$, then (α_i, β_i) can be given by

$$\begin{aligned}\alpha_i &= r_i, \\ \beta_i &= \beta_{i-2} + \beta_{i-1}q_i.\end{aligned}$$

   Therefore

$$\begin{aligned}\deg(\alpha_i) &= \deg(r_i) < \deg(\alpha_{i-1}) & (2.13) \\ \deg(\beta_i) &= \max\{\deg(\beta_{i-2}), \deg(\beta_{i-1}) + \deg(q_i)\} \\ &= \max\{\deg(\beta_{i-2}), \deg(\beta_{i-1}) + \deg(\alpha_{i-2}) - \deg(\alpha_{i-1})\} & (2.14)\end{aligned}$$

   Assume that for $k \geq 1$, $\deg(\alpha_{k-1}) + \deg(\beta_k) = \deg(g)$, then

$$\begin{aligned}\deg(\beta_{k+1}) &= \max\{\deg(\beta_{k-1}), \deg(\beta_k) + \deg(\alpha_{k-1}) - \deg(\alpha_k)\} \\ &= \max\{\deg(\beta_{k-1}), \deg(g) - \deg(\alpha_k)\} \\ &= \max\{\deg(g) - \deg(\alpha_{k-1}), \deg(g) - \deg(\alpha_k)\} \\ &= \deg(g) - \deg(\alpha_k), & (2.15)\end{aligned}$$

   according to (2.13) and (2.14).

Since α_i are monotonically decreasing, there must exist some k such that $\deg(\alpha_{k-1}) > \lfloor t/2 \rfloor$ and $\deg(\alpha_k) \leq \lfloor \deg(g)/2 \rfloor$. The algorithm will stop at Iteration k with $\deg(\beta_k) = \deg(g) - \deg(\alpha_{k-1}) < t - \lfloor t/2 \rfloor \leq \lfloor (t-1)/2 \rfloor$. □

Through the proof we can see that the (α, β) finding procedure can be implemented in an iterative way (IterativeFindAlphaBeta).

Function IterativeFindAlphaBeta(g, T)

```
1  Write $g$ as $Tq + r$;
2  $\alpha_0 \leftarrow T$;
3  $\beta_0 \leftarrow 1$;
4  $\alpha_1 \leftarrow r$;
5  $\beta_1 \leftarrow q$;
6  while $\deg(\alpha_0) > \lfloor \deg(g)/2 \rfloor$) do
7      Write $\alpha_0$ as $\alpha_1 q + r$;
8      $\alpha_0 \leftarrow \alpha_1$;
9      $\alpha_1 \leftarrow r$;
10     $\beta_{tmp} = \beta_0 + \beta_1 q$;
11     $\beta_0 = \beta_1$;
12     $\beta_1 = \beta_{tmp}$;
13 end
14 return $(\alpha_0, \beta_0)$;
```

2.4.3 Performance

We implemented the algorithms naïvely in C++ with Shoup's NTL library. The implementation was compared with RSA benchmark of OpenSSL 1.1.0. Running time is measured on a machine with Intel i5-3210M quadcore CPU and 4 GB memory. Security levels of RSA are found in [SP800-57]; security level of McEliece is determined by the cryptanalysis by Bernstein *et al.* [BLP08]

Table 2.2 shows the comparison of key sizes. Table 2.3, 2.4 and 2.5 contain running time of `KeyGen`, `Enc` and `Dec` respectively.

From Table 2.2, we can see that McEliece keys are much larger than that of RSA. Even for just 80 bits of security, we need around one megabyte of space to store the keys.

As for running time of `KeyGen`, Table 2.3 shows that as key size increases, the time that RSA takes to generate a key pair grows dramatically, while McEliece only takes mildly more time. Note that the data in Table 2.3 are averages. In a single run, McEliece takes roughly the same time as the average, but RSA tends to vary greatly, ranging from tens of seconds to a few minutes. We also note that RSA keys need to be generated with carefully designed pseudorandom number generators, otherwise they can be easily recovered through a precomputed table.

Table 2.4 and 2.5 show the comparison of between RSA and McEliece in the running

Security	RSA			McEliece		
	k	pk (bits)	sk (bits)	(n, k, t)	pk (MB)	sk (MB)
80	1024	≈ 1024	≈ 2048	$(2048, 1751, 27)$	0.4483	1.356
112	2048	≈ 2048	≈ 4096			
128	3072	≈ 3072	≈ 6144	$(2960, 2288, 56)$	0.8466	2.597
192	7680	≈ 7680	≈ 15360			
256	15360	≈ 15360	≈ 30720	$(6624, 5129, 115)$	4.247	13.020

Table 2.2: Comparison of key sizes. Value k is the size of the modulus of RSA. (n, k, t) is the parameter of underlying code. For RSA, key sizes vary and values shown are estimated under the assumption that $e = 65537$ is used.

Security	RSA		McEliece	
	k	Time (s)	(n, k, t)	Time (s)
80	1024	0.0323	$(2048, 1751, 27)$	0.968
112	2048	0.0837		
128	3072	0.320	$(2960, 2288, 56)$	3.376
192	7680	10.926		
256	15360	216.294	$(6624, 5129, 115)$	40.737

Table 2.3: Running time of `KeyGen`. RSA data are given by OpenSSL tests. All time values are obtained by averaging over three runs and are in seconds.

Security	RSA		McEliece		
	k	Time (ms)	(n, k, t)	Time (ms)	Normalized
80	1024	0.0107	$(2048, 1751, 27)$	0.0590	0.0345
112	2048	0.0353			
128	3072	0.0741	$(2960, 2288, 56)$	0.101	0.136
192	7680	0.456			
256	15360	1.782	$(6624, 5129, 115)$	0.386	1.156

Table 2.4: Running time of `Enc`. RSA data are given by OpenSSL tests averaging over runs in 10 s. McEliece data are obtained by averaging over three runs. McEliece running times are also normalized to reflect the difference in plaintext size. All time values are in milliseconds.

Security	RSA		McEliece		
	k	Time (ms)	(n, k, t)	Time (ms)	Normalized
80	1024	0.162	$(2048, 1751, 27)$	23.728	13.863
112	2048	1.179			
128	3072	3.526	$(2960, 2288, 56)$	210.525	282.571
192	7680	74.925			
256	15360	411.200	$(6624, 5129, 115)$	848.982	2542.311

Table 2.5: Running time of `Dec`. RSA data are given by OpenSSL tests averaging over runs in 10 s. McEliece data are obtained by averaging over three runs. McEliece running times are also normalized to reflect the difference in plaintext size. All time values are in milliseconds.

time of `Enc` and `Dec`. Although RSA outperforms McEliece in relatively low security levels, McEliece catches up quickly. If we compare the time taken by RSA and McEliece in 128-bit and 256-bit security, we can see that when we double the security parameter, RSA will be $\approx$ 20 times slower in encryption and $\approx$ 100 times slower in decryption; for McEliece the two numbers are only around 10. Moreover, McEliece is already faster than RSA in encryption in 256-bit security level.

The running-time data show that the naïve implementation is only slower than RSA in decryption, but it is also obvious that the complexity of McEliece is lower than that of RSA. Considering that OpenSSL is a well-maintained toolkit, with proper optimization McEliece could become faster than RSA in decryption at 256-bit security level.

Chapter 3

Quantum Computation and Quantum Cryptanalysis

3.1 Background

The interest of building a quantum computer, which is essentially different from a classical one, largely arose from the demand to simulate a quantum system, as Feynman [Fey82] asked for in 1982. Ten years later, Shor found that quantum computer can efficiently solve the factoring and discrete log problems [Sho94] and thus break many widely used public-key cryptosystems. Grover also found an algorithm searching random database with quadratic speedup [Gro96].

Fortunately, some cryptosystem designs are based on problems without known efficient quantum algorithms. Two representatives of such problems are coding problems and lattice problems.

3.2 The Quantum Computation Model

The first attempt to model such a computer was the quantum Turing machines proposed by Deutsch [Deu85] and refined by Bernstein *et al.* [BV93], as a natural extension of classical Turing machine to the quantum setting. However, quantum circuits, developed by Yao [Yao93] as an extension of boolean circuits and an equivalent to quantum Turing machines, are more convenient and widely used to describe quantum algorithms.

This section provides an overview of the model of quantum computation. Section 3.2.1 describes the framework that governs quantum computation. and also reviews the necessary linear algebra background and introduces the standard bra-ket notation for vectors. Section 3.2.2 introduces the components of a quantum circuit and defines the complexity of a quantum algorithm.

More thorough documentation of quantum computing can be found in [NC10] and [KLM07].

3.2.1 Physical restrictions of quantum computing

In the classical setting, we have been describing an algorithms by an input, an output, and a series of operations to turn the input into the output. However, not every operation is efficient or even possible due to the physical reality. In order to comply with this limit of the real world, we have to restrict ourselves on a few very basic "atomic" operations. That becomes the model of classical computation.

The description of a quantum algorithm is very similar to its classical counterpart, just that the information is stored in a *quantum system*, manipulated by *quantum operations*, and extracted by a *measurement*, which produces a probabilistic output. In each stage of the computation, the data can be represented by a configuration known as the quantum state.

Data as quantum states and their measurements

Classical computers store data in *bits*, each of which can only take a value of either 0 or 1. Quantum mechanics allows a particle to stay "somewhere in between", in the sense that the particle, when observed, can sometimes be 0 and sometimes be 1 with certain probability.

Quantum states A quantum state is usually written in Dirac notation (or "Bra-ket" notation), but it can also be interpreted as a vector.

Definition 3.2.1 (Dirac notation)**.** $|\psi\rangle$ is called a *ket* and denotes a (column) vector corresponding to ψ; $\langle\psi|$ is called a *bra* and denotes the conjugate transpose of $|\psi\rangle$, i.e.,

$$\langle\psi| = (|\psi\rangle^*)^\top.$$

The inner product of two vectors $|\varphi\rangle$ and $|\psi\rangle$ is written as $\langle\varphi|\psi\rangle$.

Definition 3.2.2 (Hilbert space)**.** A *Hilbert space* $\mathcal{H}$ is a vector space over complex numbers with an inner product $\langle\cdot|\cdot\rangle$.

Postulate 1 (State space)**.** *The state of an n-qubit system can be described by a unit vector in a 2^n-dimensional Hilbert space.*

Example 3.2.1. The state of one qubit can be represented by a vector

$$|\psi\rangle = \alpha_0|0\rangle + \alpha_1|1\rangle,$$

in the two-dimensional Hilbert space $\mathcal{H} = \mathbb{C}^2$, where α_0 and α_1 are arbitrary complex numbers satisfying the normalization constraint $|\alpha_0|^2 + |\alpha_1|^2 = 1$ (in order to be a unit vector). States $|0\rangle$ and $|1\rangle$ are frequently used in quantum computation and are called the *computational basis*, which have vector representation

$$|0\rangle = \begin{bmatrix} 1 \\ 0 \end{bmatrix}, \quad |1\rangle = \begin{bmatrix} 0 \\ 1 \end{bmatrix}.$$

Composite states The state of n qubits can be represented by a vector in the 2^n-dimensional Hilbert space.

Definition 3.2.3 (Tensor product)**.** The *tensor product* of an a-dimensional Hilbert space $\mathcal{H}_A$ and a b-dimensional Hilbert space $\mathcal{H}_B$, denoted as $\mathcal{H}_A \otimes \mathcal{H}_B$, is a Hilbert space $\mathcal{H}$ with dimension $\dim(\mathcal{H}) = ab$, where $|\psi_A\rangle = \begin{bmatrix}\alpha_0, \alpha_1, \ldots, \alpha_{2^a-1}\end{bmatrix}^\top \in \mathcal{H}_A$ and $|\psi_B\rangle = \begin{bmatrix}\beta_0, \beta_1, \ldots, \beta_{2^b-1}\end{bmatrix}^\top \in \mathcal{H}_B$ are combined as

$$|\psi\rangle = |\psi_A\rangle \otimes |\psi_B\rangle = \begin{bmatrix} \alpha_0\beta_0 \\ \alpha_0\beta_1 \\ \vdots \\ \alpha_0\beta_{2^b-1} \\ \alpha_1\beta_0 \\ \vdots \\ \alpha_{2^a-1}\beta_{2^b-1} \end{bmatrix} = \begin{bmatrix} \alpha_0|\psi_B\rangle \\ \alpha_1|\psi_B\rangle \\ \vdots \\ \alpha_{2^a-1}|\psi_B\rangle \end{bmatrix}$$

(Kronecker product). The tensor product of operators A on $\mathcal{H}_A$ and B on $\mathcal{H}_B$ is defined by

$$(A \otimes B)(|\psi_A\rangle \otimes |\psi_B\rangle) = (A|\psi_A\rangle) \otimes (B|\psi_B\rangle).$$

The tensor product of $|\psi_A\rangle$ and $|\psi_B\rangle$ is often written as $|\psi_A\rangle|\psi_B\rangle$ and $|\psi_A\psi_B\rangle$ for short.

Postulate 2 (Composite systems)**.** *A system composed by two subsystems associated with state spaces $\mathcal{H}_A$ and $\mathcal{H}_B$ has a state space $\mathcal{H} = \mathcal{H}_A \otimes \mathcal{H}_B$.*

Example 3.2.2. If a quantum system consists of two qubits, with the state of each of them being $|\psi_1\rangle$ and $|\psi_2\rangle$, respectively, then the state of the whole system $|\psi\rangle$ can be described by

$$|\psi\rangle = |\psi_1\rangle \otimes |\psi_2\rangle.$$

However, most of the possible states cannot be written as the tensor product decomposition of single-qubit states. A well-known example is the Bell state

$$\frac{1}{\sqrt{2}}\left(|0\rangle|0\rangle + |1\rangle|1\rangle\right).$$

In such cases, the states are called *entangled*.

Measurement In order to extract some information out of an quantum state $|\psi\rangle$, we have to measure it in an *orthonormal basis*.

Postulate 3 (Measurement)**.** *Assume a 2^n-dimensional Hilbert space $\mathcal{H}$ has orthonormal basis $\{\,|\psi_i\rangle\,\}_{i=0}^{2^n-1}$, and a quantum state $|\psi\rangle \in \mathcal{H}$ can be written as*

$$|\psi\rangle = \sum_{i=0}^{2^n-1} \alpha_i |\psi_i\rangle,$$

then a measurement yields $|\psi_i\rangle$ with probability $|\alpha_i|^2$, and leave the system in $|\psi_i\rangle$.

Note that after the measurement, the qubit will stay in the basis state we just observed rather than the original state. This means that we can obtain no more information about the computation. More precisely, there is Holevo's theorem claiming that we cannot extract more than n classical bits of information from an n-qubit state.

Measurement can also be done to a subsystem. For a state

$$|\psi\rangle = \sum_{i=0}^{2^n-1} \alpha_i |\psi_i\rangle \otimes |\phi_i\rangle \in \mathcal{H}_A \otimes \mathcal{H}_B$$

where $\{\,|\psi_i\rangle\,\}_{i=0}^{2^n-1}$ are an orthonormal basis of $\mathcal{H}_A$, a measurement of the subsystem with state space $\mathcal{H}_A$ will yield $|\psi_i\rangle \otimes |\phi_i\rangle$ with probability α_i and leave the whole system in the state $|\psi_i\rangle \otimes |\phi_i\rangle$.

Example 3.2.3. The outcome of measuring the single-qubit state $\frac{1}{\sqrt{2}}(|0\rangle + |1\rangle)$ in computational basis will be either $|0\rangle$ or $|1\rangle$, both with probability 1/2. We can also measure it in the Hadamard basis

$$\{\, |+\rangle, |-\rangle \,\} = \left\{ \frac{1}{\sqrt{2}} \begin{bmatrix} 1 \\ 1 \end{bmatrix}, \frac{1}{\sqrt{2}} \begin{bmatrix} 1 \\ -1 \end{bmatrix} \right\},$$

and this time we will always obtain $|+\rangle$.

Since the probability of each measurement outcome is only determined by the magnitude of each component state, we cannot distinguish two states $|\psi\rangle$ and $|\psi'\rangle$ if they only differ by a *global phase factor* $\mathrm{e}^{\mathrm{i}\varphi}$

$$|\psi\rangle = \mathrm{e}^{\mathrm{i}\varphi}|\psi'\rangle, \quad \varphi \in [0, 2\pi).$$

Therefore an arbitrary qubit

$$|\psi\rangle = r_0 \mathrm{e}^{\mathrm{i}\varphi_0}|0\rangle + r_1 \mathrm{e}^{\mathrm{i}\varphi_1}|1\rangle$$

can be rewritten as an equivalent $|\psi'\rangle$

$$|\psi'\rangle = r_0|0\rangle + r_1 \mathrm{e}^{\mathrm{i}(\varphi_1 - \varphi_0)}|1\rangle$$

with non-negative real r_0.

Operations on quantum states

Definition 3.2.4 (Linear operator)**.** A *linear operator* A on a vector space $\mathcal{H}$ is a linear transformation from $\mathcal{H}$ to itself. Linear operator A can be written as a square matrix.

Definition 3.2.5 (Unitary operator)**.** A *unitary operator* U on a vector space $\mathcal{H}$ is a linear operator satisfying $U^\dagger U = I$, where $U^\dagger$ is the conjugate transpose of U in matrix representation and I is the identity matrix.

Postulate 4 (Evolution)**.** *The evolution of the state of a closed system from time t_0 to t_1 can be described by a unitary operator U, i.e.,*

$$|\psi(t_1)\rangle = U|\psi(t_0)\rangle.$$

This postulate states that every operation in quantum computation must be unitary. It also implies that the computation process must be invertible.

Example 3.2.4. To obtain a Hadamard state $|+\rangle$, we can first prepare the state $|0\rangle$ and then apply the Hadamard operator H (represented by a 2×2 Hadamard matrix) on it:

$$|+\rangle = H|0\rangle,$$

or in matrix representation,

$$\frac{1}{\sqrt{2}}\begin{bmatrix}1\\1\end{bmatrix} = \frac{1}{\sqrt{2}}\begin{bmatrix}1 & 1\\1 & -1\end{bmatrix}\begin{bmatrix}1\\0\end{bmatrix}.$$

The postulate also leads to some unintuitive restrictions on quantum computing, for example the no-cloning theorem.

Theorem 3.2.1 (No-cloning theorem)**.** *There exists no unitary operator U such that for arbitrary state $|\psi\rangle$*

$$U(|\psi\rangle \otimes |a\rangle) = |\psi\rangle \otimes |\psi\rangle,$$

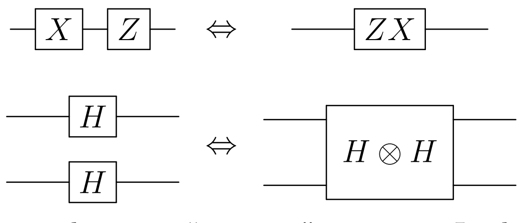
where $|a\rangle$ is some fixed ancilla state.

3.2.2 Quantum circuits

Quantum circuits consists of gates that are quantum operations (unitary operators) and wires, each of which carries a single qubit.

Table 3.1 shows some basic quantum gates. Similar to boolean circuits, we can pick some gates to form a universal set that can *approximate arbitrary unitary operations efficiently*, e.g., $\{ H, R_{\pi/4}, \text{CNOT} \}$.

Example 3.2.5. From the perspective of matrix representation, the combination of two gates have the following effect:



Example 3.2.6. Figure 3.1 shows a "copying" circuit. It does not violate no-cloning theorem. Instead, it often creates entangled states, for example

$$|+\rangle = \frac{1}{\sqrt{2}}(|0\rangle + |1\rangle) \qquad \rightarrow \qquad \frac{1}{\sqrt{2}}(|00\rangle + |11\rangle).$$

However, this circuit is useful in subroutine calls, where we need to store the output before the uncomputing step (for the necessity of uncomputing, see [BBBV97]).

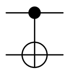
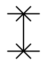
Gate name	Label	Matrix	Gate name	Label	Matrix
Hadamard	H	$\frac{1}{\sqrt{2}}\begin{bmatrix}1 & 1\\ 1 & -1\end{bmatrix}$	Pauli-X (NOT)	X	$\begin{bmatrix}0 & 1\\ 1 & 0\end{bmatrix}$
Pauli-Y	Y	$\begin{bmatrix}0 & -i\\ i & 0\end{bmatrix}$	Pauli-Z	Z	$\begin{bmatrix}1 & 0\\ 0 & -1\end{bmatrix}$
Phase-shift	R_θ	$\begin{bmatrix}1 & 0\\ 0 & \mathrm{e}^{\mathrm{i}\theta}\end{bmatrix}$			
CNOT		$\begin{bmatrix}1 & & & \\ & 1 & & \\ & & & 1\\ & & 1 & \end{bmatrix}$	Swap		$\begin{bmatrix}1 & 0 & 0 & 0\\ 0 & 0 & 1 & 0\\ 0 & 1 & 0 & 0\\ 0 & 0 & 0 & 1\end{bmatrix}$

Table 3.1: Basic quantum gates that frequently appear. H, X, Y, Z and R_θ are 1-qubit gates; CNOT (controlled NOT) and Swap are 2-qubit gates.

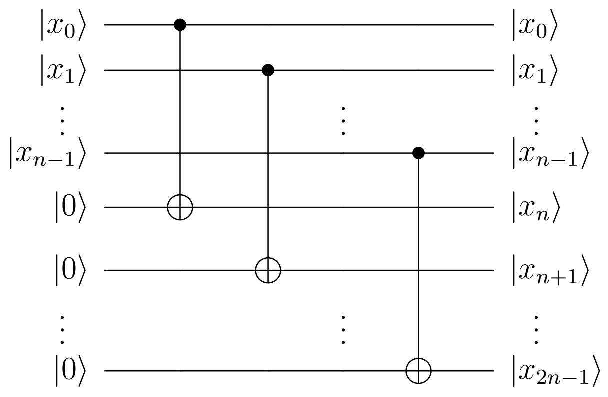


Figure 3.1: Quantum “copying” circuit

Quantum complexity

When deciding the complexity of an algorithm described by a circuit model, we have to be careful because circuit models are unlike Turing machines, which we are more familiar with.

1. A single Turing machine works for every input, while circuits must be enlarged for a longer input, yielding a family of circuits. As a result, we also need to ensure that the family of circuits can be efficiently generated by a Turing machine (see Definition 3.2.7).
2. The time complexity of a Turing machine is measured by computation steps taken before halting, while the complexity of a circuit is measured by its size (the total number of gates) and depth (the number of gates along the longest path).

Example 3.2.7. The "copying" process in Figure 3.1 has complexity $O(n)$.

The complexity of circuits is more precisely characterized in Definition 3.2.7.

Here we define the class of polynomial-time randomized algorithms **BPP** and its quantum counterpart, **BQP**.

Definition 3.2.6 (**BPP**)**.** A language L is in **BPP** if and only if there exists a polynomial-time deterministic Turing machine M that takes two inputs, x and y with $|y| \le p(|x|)$ for some polynomial p, such that

1. For all $x \in L$, $\frac{\left|\left\{ y \in \{0,1\}^{p(|x|)} \,\middle|\, M(x,y)=1 \right\}\right|}{2^{p(|x|)}} \ge 2/3$;
2. For all $x \notin L$, $\frac{\left|\left\{ y \in \{0,1\}^{p(|x|)} \,\middle|\, M(x,y)=1 \right\}\right|}{2^{p(|x|)}} < 1/3$.

Definition 3.2.7 (Polynomial-time uniform family)**.** A family of circuits $\{ C_n : n \in \mathbb{N} \}$ is *polynomial-time uniform* if and only if there exists a polynomial-time deterministic Turing machine M, wuch that

$$M(\mathbb{1}^n) = \texttt{Enc}(C_n),$$

where $\texttt{Enc}(C_n)$ is the encoding of C_n.

Definition 3.2.8 (**BQP**)**.** A language L is in **BQP** if and only if there exists a polynomial-time uniform family of quantum circuits $\{ Q_n : n \in \mathbb{N} \}$ with n-qubit input and 1-qubit output, such that

1. For all $x \in L$, $\Pr\left[Q_{|x|}(x) = 1 \right] \ge 2/3$;
2. For all $x \notin L$, $\Pr\left[Q_{|x|}(x) = 1 \right] < 1/3$.

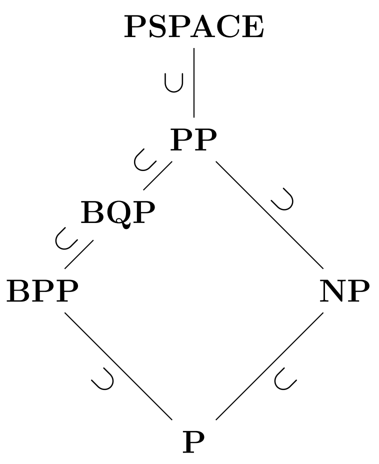


Figure 3.2: The inclusion diagram of a few complexity classes. This diagram is also part of [Kup].

3.3 The Hidden Subgroup Problem (HSP) Family and Quantum Fourier Transform

Since the invention of the famous Shor's algorithm [Sho94], quantum computers have been shown to efficiently solve many number theory problems underlying popular public-key cryptosystems, including RSA and ElGamal. Such problems have also been attracting interest from theoretical computer scientists because they can be an hint to the answer of a number of classical open questions (e.g., **P** vs. **PSPACE**) in addition to providing new quantum complexity classes. Surprisingly, almost all these problems can be captured within a unified framework, namely the *hidden subgroup problem* (HSP). The algorithms solving them hence share a lot of similarities, in particular a core technique for most of them known as *quantum order-finding*.

Definition 3.3.1 (HSP)**.** Given a set of generators of a group G, a finite image set X and a function $f : G \to X$ such that f hides a subgroup $H < G$ in the sense that $f(x) = f(y)$ if and only if x and y are in the same coset of H, the HSP problem is to find a set of generators of H.

Not all HSP instances are known to have efficient quantum algorithm. This section will review Simon's problem, boolean hidden shift problem, and factoring problem, which are efficiently solvable HSP instances on a quantum computer. We will come to difficult HSP instances in Section 3.3.3.

3.3.1 Simon's problem and boolean functions

Simon's problem

Simon's problem is deliberately designed to allow quantum computers to outperform classical computers. It shows that $\mathbf{BPP}^O \subsetneq \mathbf{BQP}^O$ for an oracle O.

Definition 3.3.2 (Simon's problem)**.** Given oracle O_f of a function $f : \mathbb{Z}_2^n \to \mathbb{Z}_2^n$ such that $f(\mathbf{x}) = f(\mathbf{y})$ if and only if $\mathbf{x} \oplus \mathbf{y} = \mathbf{0}$ or $\mathbf{s}$, where $\oplus$ is bitwise exclusive-or and $\mathbf{s}$ is an unknonwn fixed nonzero vector, Simon's problem is to find the hidden $\mathbf{s}$, with queries to the oracle O_f.

It's clear that Simon's problem is a special case of HSP where $G = \mathbb{Z}_2^n, H = \mathbf{s}^\perp$.

The quantum algorithm that efficiently solves Simon's problem is shown in Figure 3.3. The purpose of the first group of Hadamard gates is to prepare a state with every possible x with equal magnitudes

$$|\psi_1\rangle = \frac{1}{2^{n/2}} \sum_{\mathbf{x} \in \mathbb{Z}_2^n} |\mathbf{x}\rangle|0\rangle.$$

After the O_f computation, the state will be

$$|\psi_2\rangle = \frac{1}{2^{n/2}} \sum_{\mathbf{x} \in \mathbb{Z}_2^n} |\mathbf{x}\rangle|f(\mathbf{x})\rangle.$$

Since $f(\mathbf{x}) = f(\mathbf{y})$ if and only if $\mathbf{x} \oplus \mathbf{y} = \mathbf{0}$ or $\mathbf{s}$, there are two cases.

- Case 1: $\mathbf{s} = \mathbf{0}$. In this case f is a permutation, and the first register of $\psi_{\mathbf{2}}$ still contains every x with equal magnitudes. After going through the second group of Hadamard gates, the first register will contain $|\mathbf{0}\rangle$.
- Case 2: $\mathbf{s} \neq \mathbf{0}$. In this case we have

$$|\psi_2\rangle = \frac{1}{2^{n/2}} \sum_{x \in \mathbf{s}^\perp} (|\mathbf{x}\rangle + |\mathbf{x} \oplus \mathbf{s}\rangle)|f(\mathbf{x})\rangle.$$

After going through the second group of Hadamard gates, the state will be

$$\begin{aligned}|\psi_3\rangle &= \frac{1}{2^n}\sum_{\mathbf{y}\in\mathbb{Z}_2^n}\sum_{\mathbf{x}\in\mathbf{s}^\perp}\left((-1)^{\mathbf{x}\cdot\mathbf{y}}+(-1)^{(\mathbf{x}\oplus\mathbf{s})\cdot\mathbf{y}}\right)|\mathbf{y}\rangle|f(\mathbf{x})\rangle\\ &= \frac{1}{2^n}\sum_{\mathbf{y}\in\mathbb{Z}_2^n}\sum_{\mathbf{x}\in\mathbf{s}^\perp}(-1)^{\mathbf{s}\cdot\mathbf{y}}|\mathbf{y}\rangle|\mathbf{f}(\mathbf{x})\rangle\\ &= \frac{1}{2^n}\sum_{\mathbf{y}\in\mathbb{Z}_2^n}(-1)^{\mathbf{s}\cdot\mathbf{y}}|\mathbf{y}\rangle\sum_{\mathbf{x}\in\mathbf{s}^\perp}|f(\mathbf{x})\rangle.\end{aligned}$$

Now if we measure the first register, in Case 1, the measurement always yields $\mathbf{0}$; in Case 2, the measurement will output some $\mathbf{a}$ such that $\mathbf{s}\cdot\mathbf{a}=0$. Therefore we can execute the algorithm multiple ($\gtrapprox n$) times. If we keep obtaining $\mathbf{0}$'s from the measurement, we claim that $\mathbf{s}=0$, otherwise it is highly likely that the algorithm returns n linearly independent equations of $\mathbf{s}$ and $\mathbf{s}$ can be solved for.

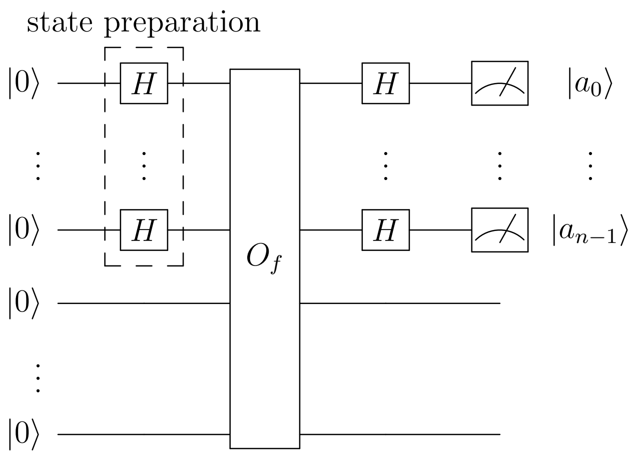


Figure 3.3: Simon's algorithm

Boolean hidden shift problem (BHSP)

Definition 3.3.3 (BHSP)**.** Given oracle O_f of a boolean function $f:\mathbb{Z}_2^n\to\mathbb{Z}_2$ and oracle O_g of function g that is f shifted by a fixed $\mathbf{s}$

$$g(\mathbf{x})=f(\mathbf{x}+\mathbf{s}),\qquad \forall\mathbf{x}\in\mathbb{Z}_2^n,$$

the BHSP problem is to find the hidden s, with queries to the oracles O_f and O_g.

Note that f must be given through oracle access. In fact, if the construction of f is known, after recovering $\mathbf{s}$, the construction of g is also known. This version of BHSP is equivalent to the exact membership learning problem, for which the numbers of samples required for quantum and classical learning algorithms have a polynomial relationship [SG01].

It turns out that BHSP is easy for *bent functions* on a quantum computer [Röt10].

Definition 3.3.4 (Bent function)**.** A boolean function $f : \mathbb{Z}_2^n \to \mathbb{Z}_2$ is *bent* if and only if for all $\mathbf{w}$, the Walsh spectrum

$$\hat{f}(\mathbf{w}) = \frac{1}{2^n} \sum_{\mathbf{x} \in \mathbb{Z}_2^n} (-1)^{\mathbf{w}\cdot\mathbf{x}+f(\mathbf{x})} = \pm\frac{1}{2^{n/2}}. \tag{3.1}$$

Formula (3.1) is called Walsh-Hadamard transform. Its matrix representation is $H^{\otimes n}$, which means it can be efficiently implmemented in a quantum circuit with n parallel Hadamard gates. Bent functions have perfectly flat Walsh spectra.

Given an instance of BHSP, we can define $F(\mathbf{x}), G(\mathbf{x})$ and $H_f(b, \mathbf{x})$ as

$$H_f(b,\mathbf{x}) = \begin{cases} F(\mathbf{x}) = \sum_{\mathbf{y}\in\mathbb{Z}_2^n} (-1)^{f(\mathbf{x}+\mathbf{y})} |\mathbf{y}\rangle, & b = 0, \\ G(\mathbf{x}) = \sum_{\mathbf{y}\in\mathbb{Z}_2^n} (-1)^{g(\mathbf{x}+\mathbf{y})} |\mathbf{y}\rangle, & b = 1. \end{cases}$$

If f is a bent function, then it is easy to verify that $F(\mathbf{x}), G(\mathbf{x})$ and $H_f(b, \mathbf{x})$ have the following properties

1. $F(\mathbf{x}) = G(\mathbf{x} + \mathbf{s})$ and $G(\mathbf{x}) = F(\mathbf{x} + \mathbf{s})$ for all $\mathbf{x} \in \mathbb{Z}_2^n$.
2. F and G are "injective" in the sense that given two different $\mathbf{x}$'s, different quantum states are output.
3. (Corollary of the above) $H_f(b_0, \mathbf{x}_0) = H_f(b_1, \mathbf{x}_1)$ if and only if $[\, b_1 - b_0 \,|\, \mathbf{x}_1 - \mathbf{x}_0 \,] \in \{\, [\, 0 \,|\, \mathbf{0} \,], [\, 1 \,|\, \mathbf{s} \,] \,\}$.
4. H_f can be implemented with oracles O_f and O_g and other basic quantum gates, as shown in Figure 3.5.

Property 3 shows that a BHSP instance with f being a bent function can be reduced to Simon's problem.

Therefore we have 35 3.3.1 [Röt10].

Theorem 3.3.1. *Given oracles O_f of a bent function $f : \mathbb{Z}_2^n \to \mathbb{Z}_2$ and O_g of function g that is f shifted by a fixed $\mathbf{s}$, then there exists a polynomial-time quantum algorithm that makes $O(n)$ queries to O_f and O_g and computes $\mathbf{s}$ with high probability.*

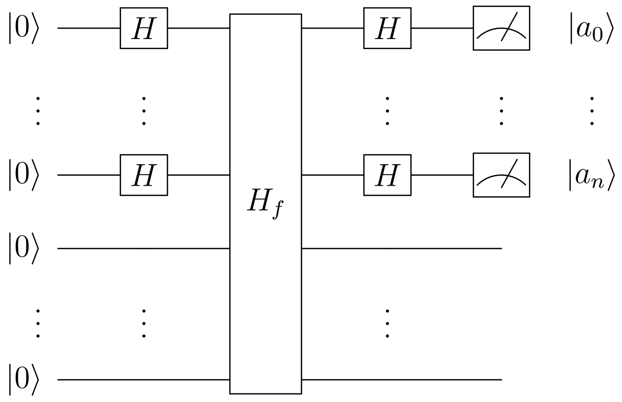


Figure 3.4: Quantum algorithm solving the BHSP problem

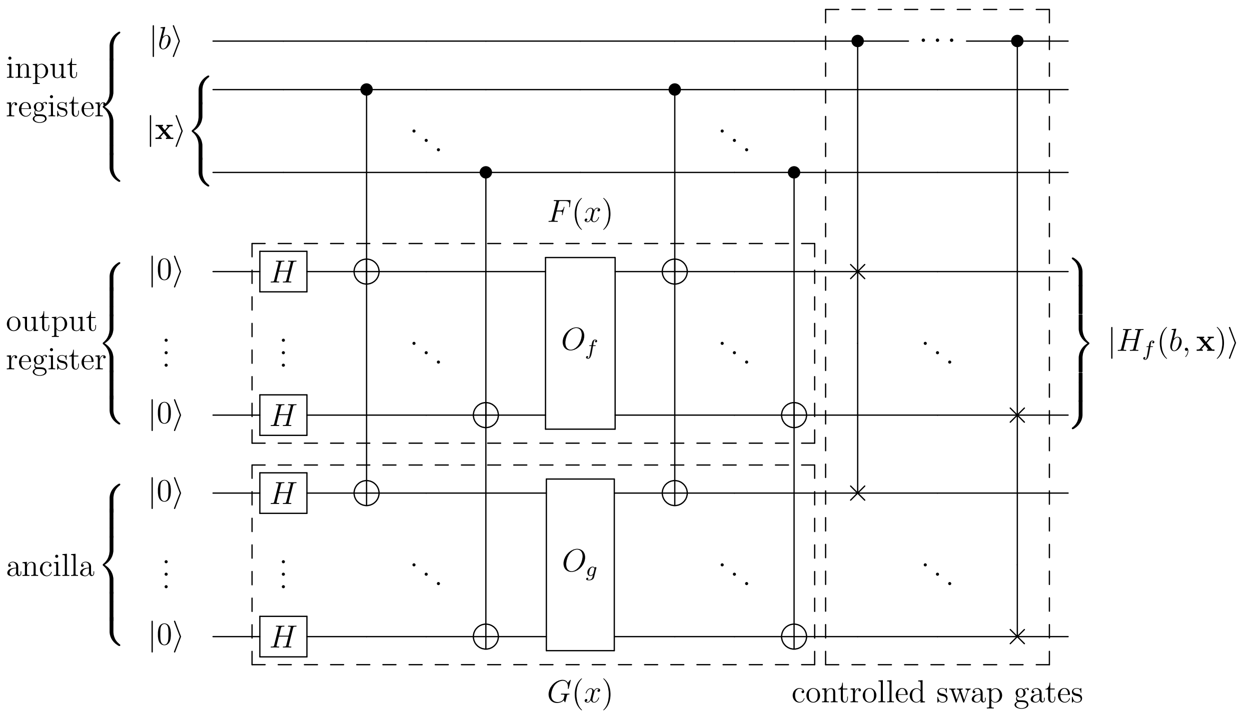


Figure 3.5: Implementation of H_f. The idea is to compute both $F(\mathbf{x})$ and $G(\mathbf{x})$ with queries to the oracles, and swap the value stored in the ancilla register to the output register if $b = 1$. In order to compute $F(\mathbf{x})$ and $G(\mathbf{x})$, we need to add $|\mathbf{x}\rangle$ to both the output and ancilla registers.

The separation of quantum and classical computation powers Certain BHSP instances are hard for classical computers, for example BHSP with f being a Maiorana-McFarland bent function [Röt10].

Definition 3.3.5 (Maiorana-McFarland bent function)**.** Let $n = 2k$. The *Maiorana-McFarland construction of bent functions* has the form

$$f(x, y) = x \cdot \pi(y) + \sigma(y),$$

where π is a permutation over $\mathbb{Z}_2^k$ and σ is an arbitrary boolean function from $\mathbb{Z}_2^k$ to $\mathbb{Z}_2$.

Theorem 3.3.2. *Given oracles O_f of a Maiorana-McFarland bent function f and O_g of function g that is f shifted by a fixed $\mathbf{s}$, then a classical computer requires $\Theta(2^{n/2})$ queries to compute $\mathbf{s}$.*

Though the BHSP problem with a bent function can be reduced to Simon's problem and hence HSP. However, a general reduction from BHSP to HSP is unknown, so it may lead to interesting new results. Another observation is that this algorithm works for other highly nonlinear boolean functions whose Walsh spectra are all nonzeros. Based on this adaptation, the authors were able to show a general exponential separation between $\mathbf{BPP}^O$ and $\mathbf{BQP}^O$ [GRR11].

3.3.2 Shor's algorithm

Based on Miller's recognition of the periodic nature underlying the factoring problem [Mil76], Shor realized that a similar technique to Simon's algorithm can be used to factorize integers on a quantum computer.

In this section we will review Shor's algorithm. Section 3.3.2 shows the reduction of the factoring problem to an order-finding problem. Section 3.3.2 introduces the quantum Fourier transform algorithm that serves as a foundation of the order-finding process. Details about the extraction of information from the output of QFT will be discussed in Section 3.3.2.

The factoring problem

For simplicity, the factoring problem is usually defined as in Definition 3.3.6, which poses restriction on the choice of N. However, solving this class of factoring problem is equivalent to solving the general factoring problem, because the other cases are either trivial (e.g., N is even or prime power) or can be tackled using the same technique (e.g., N has more than two distinct prime factors). It is worth notice that Definition 3.3.6 entirely covers the factorization of RSA moduli.

Definition 3.3.6 (Factoring)**.** Given a positive integer N which is the product of two distinct odd primes p and q, the factoring problem is to find either p or q.

Similar to many other factoring algorithms (e.g., number field sieve [LLJMP93]), Shor's algorithm factorizes integers by finding a nontrivial square root modulo N. Shor made the following observation: since $N = pq$, for any $a \in \mathbb{Z}_N^*$, we have $\mathrm{ord}(a) \mid \phi(N) = (p-1)(q-1)$, which means that certain a's can have an even order. If we have $\mathrm{ord}(a) = 2k$ and $a^k \not\equiv -1 \pmod{N}$, a^k will be a nontrivial square root of 1 modulo N, and we can factor N by computing $\gcd(a \pm 1, N)$. Therefore the factoring problem is reduced to an order-finding problem [Sho94].

Definition 3.3.7 (Factoring as an order-finding problem)**.** Given a function

$$\begin{aligned} f : \mathbb{Z}_{\phi(N)} &\to \mathbb{Z}_N^* \\ x &\mapsto a^x \end{aligned}$$

for some $a \in Z_N^*$ (i.e., a coprime with N), the order-finding problem is to find the smallest t such that $f(x + t) = f(x)$.

The factoring problem can be also viewed as a special case of HSP, where $G = \mathbb{Z}_{\phi(N)} = \mathbb{Z}_{(p-1)(q-1)}$ (additive group of the exponents), and $H = \langle t \rangle$.

Figure 3.6 shows the quantum circuit that implements Shor's algorithm. The implementation of f is shown in Figure 3.7. The first half of Shor's algorithm is almost identical to Simon's algorithm, staring with a state preparation stage that creates a state that is the superposition of every possible $|x\rangle$ with equal magnitudes. The second register, however, is $|1\rangle$ because of the construction of f. After the computation of f, the state will be

$$\begin{aligned} |\psi_1\rangle &= \frac{1}{2^{n/2}} \sum_{x=0}^{2^n-1} |x\rangle |a^x \bmod N\rangle \\ &= \frac{1}{2^{n/2}} \sum_{r=0}^{t-1} \sum_{q=0}^{\lceil (2^n-r)/t \rceil - 1} |tq + r\rangle |a^r \bmod N\rangle. \end{aligned}$$

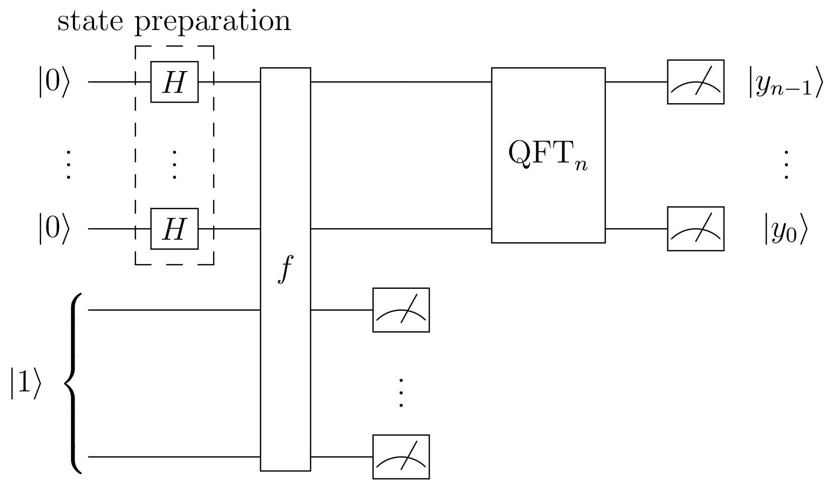


Figure 3.6: Shor's algorithm

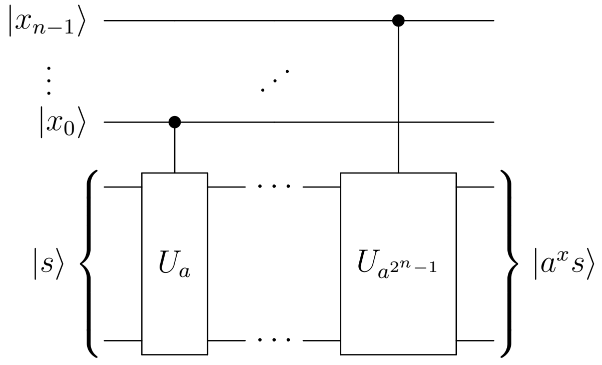


Figure 3.7: The implementation of exponential function $f : x \mapsto a^x$. Each $U_{a^{2^i}}$ maps the input $|t\rangle$ to $\left|a^{2^i} t\right\rangle$.

With the early measurement in the second register, we will settle the first register on some r:

$$|\psi_2\rangle = \frac{1}{\sqrt{\lceil \frac{2^n - r}{t} \rceil}} \sum_{q=0}^{\lceil (2^n - r)/t \rceil - 1} |tq + r\rangle |a^r \bmod N\rangle.$$

The superposed values in the first register roughly have a period of t, and we can use Fourier transform to find out this period. However, we must be careful because the period is not exact unless $t \mid 2^n$. We will discuss it in Section 3.3.2.

Quantum Fourier transform (QFT)

Before we come to the quantum order-finding algorithm, we will review quantum Fourier transform because it is the core of order-finding.

Fast Fourier transform Let $N = 2^n$, the discrete Fourier transform (DFT) of a sequence $\{a_t\}_{t=0}^{N-1}$ with period N is defined by

$$A_k = \frac{1}{\sqrt{N}} \sum_{t=0}^{N-1} a_t \omega_N^{tk}, \quad \text{for } k = 0, \ldots, N-1, \tag{3.2}$$

where ω_N is a N-th primitive root of unity (usually we let $\omega_N = \mathrm{e}^{2\pi \mathrm{i}/N}$) and A_k are sometimes called the frequency spectrum of the sequence. It can also be written in the form of matrix multiplication, as in Equation (3.3).

$$\begin{bmatrix} A_0 \\ A_1 \\ \vdots \\ A_{N-1} \end{bmatrix} = \frac{1}{\sqrt{N}} F_N \begin{bmatrix} a_0 \\ a_1 \\ \vdots \\ a_{N-1} \end{bmatrix}, \tag{3.3}$$

where the DFT matrix F_N is the Vandermonde matrix

$$\begin{bmatrix} 1 & 1 & \cdots & 1 \\ 1 & \omega_N & \cdots & \omega_N^{N-1} \\ \vdots & \vdots & \ddots & \vdots \\ 1 & \omega_N^{N-1} & \cdots & \omega_N^{(N-1)^2} \end{bmatrix} \tag{3.4}$$

If we rearrange the columns in F_N by separating even and odd columns, and let $\omega_{N/2} = \omega_N^2$, we can obtain a recursive relationship between matrices to speed up the computation.

$$F_N = \left[\begin{array}{cccc|cccc} 1 & 1 & \cdots & 1 & 1 & 1 & \cdots & 1 \\ 1 & \omega_{N/2} & \cdots & \omega_{N/2}^{N/2-1} & \omega_N & \omega_N^3 & \cdots & \omega_N^{N-1} \\ \vdots & \vdots & \ddots & \vdots & \vdots & \vdots & \ddots & \vdots \\ 1 & \omega_{N/2}^{N/2-1} & \cdots & \omega_{N/2}^{(N/2-1)^2} & \omega_N^{N/2-1} & \omega_N^{3(N/2-1)} & \cdots & \omega_N^{(N-1)(N/2-1)} \\ \hline 1 & 1 & \cdots & 1 & -1 & -1 & \cdots & -1 \\ 1 & \omega_{N/2} & \cdots & \omega_{N/2}^{N/2-1} & -\omega_N & -\omega_N^3 & \cdots & -\omega_N^{N-1} \\ \vdots & \vdots & \ddots & \vdots & \vdots & \vdots & \ddots & \vdots \\ 1 & \omega_{N/2}^{N/2-1} & \cdots & \omega_{N/2}^{(N/2-1)^2} & -\omega_N^{N/2-1} & -\omega_N^{3(N/2-1)} & \cdots & -\omega_N^{(N-1)(N/2-1)} \end{array}\right]$$
$$= \left[\begin{array}{c|c} F_{N/2} & G_{N/2} \\ \hline F_{N/2} & -G_{N/2}. \end{array}\right]$$

Here $F_{N/2}$ is the DFT matrix of size $2^{n-1} \times 2^{n-1}$, and $G_{N/2} = P_{N/2}F_{N/2}$, where

$$P_{N/2} = \begin{bmatrix} 1 & & & & \\ & \omega_N & & & \\ & & \omega_N^2 & & \\ & & & \ddots & \\ & & & & \omega_N^{N/2-1} \end{bmatrix}.$$

Quantum Fourier transform The idea behind the quantum Fourier transform (QFT) is very similar to that of the fast fourier transform (FFT) algorithm. The QFT algorithm exploits the same recursive relationship as that of FFT, but with the benefit of quantum superposition, it's exponentially faster than FFT.

Recall that we separate the even and odd columns of F_N in FFT. In quantum circuits, the input to Fourier transform can be a single superposition state

$$\sum_{t=0}^{N-1} a_t|t\rangle,$$

or if we write down the binary expansion of each $t = 2^{n-1}t_{n-1} + 2^{n-2}t_{n-2} + \cdots + 2t_1 + t_0$,

$$\sum_{t=0}^{N-1} a_t|t_{n-1}\rangle|t_{n-2}\rangle\cdots|t_1\rangle|t_0\rangle.$$

The separation of even and odd terms can be done by just looking at the last qubit $|t_0\rangle$, and $F_{N/2}$ will be applied on $|t_{n-1}\rangle|t_{n-2}\rangle\cdots|t_1\rangle$. Next we will rewrite the FFT matrix so that we can express it by basic gates of quantum circuits:

$$\begin{aligned}
F_N &= \left[\begin{array}{c|c} F_{N/2} & G_{N/2} \\ \hline F_{N/2} & -G_{N/2} \end{array}\right] = \left[\begin{array}{c|c} F_{N/2} & P_{N/2}F_{N/2} \\ \hline F_{N/2} & -P_{N/2}F_{N/2} \end{array}\right] \\
&= \left[\begin{array}{c|c} I_{N/2} & I_{N/2} \\ \hline I_{N/2} & -I_{N/2} \end{array}\right] \left[\begin{array}{c|c} I_{N/2} & \\ \hline & P_{N/2} \end{array}\right] \left[\begin{array}{c|c} F_{N/2} & \\ \hline & F_{N/2} \end{array}\right] \\
&= (H \otimes I_{N/2}) \left[\begin{array}{c|c} I_{N/2} & \\ \hline & P_{N/2} \end{array}\right] (I_2 \otimes F_{N/2}),
\end{aligned}$$

and also notice that if $\omega_N = \mathrm{e}^{2\pi\mathrm{i}/N}$, then $P_{N/2}$ can be written as the tensor product of phase-shift matrices

$$\begin{aligned}
P_{N/2} &= \begin{bmatrix} 1 & \\ & \omega_N \end{bmatrix} \otimes \begin{bmatrix} 1 & \\ & \omega_N^2 \end{bmatrix} \otimes \cdots \otimes \begin{bmatrix} 1 & \\ & \omega_N^{N/4} \end{bmatrix} \\
&= R_{2\pi/N} \otimes R_{4\pi/N} \otimes \cdots \otimes R_{\pi/2},
\end{aligned}$$

therefore we can obtain the QFT circuit for F_N $(N > 2)$ in Figure 3.8. Notice that phase-shift gates will only be applied to odd terms, i.e., controlled phase-shift gates are used in the actual circuit. For $N = 2$ the QFT circuit is simply a Hadamard gate H. Note

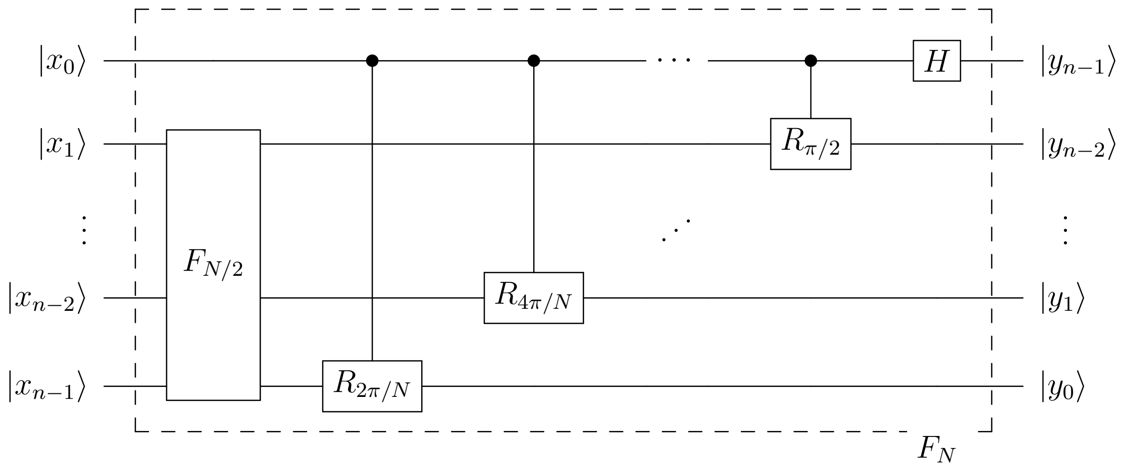


Figure 3.8: The QFT circuit

that the order of bits $|y_i\rangle$ output by the QFT circuit is reversed. This is the result of the column rearrangement we applied to the Fourier transform matrix F_N. The separation of even and odd columns corresponds to moving the least significant bit of input $|x_0\rangle$ to the most significant bit, as can be seen in the example below. The overall effect on $|\mathbf{x}\rangle$ is that the input bits are reversed.

Example 3.3.1. Table 3.2 shows how the order of input bits is reversed through the decomposition of F_N into $F_{N/2}$ and other basic quantum gates.

Stage	F_8	F_4	F_2
Separation	000	000	000
	001	010	100
	010	100	010
	011	110	110
	100	001	001
	101	011	101
	110	101	011
	111	111	111
Bit order	$x_2x_1x_0$	$x_0x_2x_1$	$x_0x_1x_2$

Table 3.2: 3-qubit example of the bit order reversal in QFT

Remark. The complexity of QFT in terms of N is $O(\log^2 N)$, exponentially smaller than the $O(N \log N)$ complexity of FFT. But QFT also has its limitation, the most serious problem probably being that we cannot obtain the spectrum directly through its output. Given input $|\mathbf{a}\rangle = \sum_{t=0}^{N-1} a_t|t\rangle$, the QFT circuit produces the state

$$\sum_{k=0}^{N-1} A_k|k\rangle,$$

which, when measured, gives $|k\rangle$ with probability $|A_k|^2$. Unless we compute A_k with classical Fourier transform or estimate A_k by the frequency of $|k\rangle$ through repeated experiments (in this case we may need number of runs proportional to N to obtain reliable results), we are unable to determine the spectrum. Therefore, the use of QFT is largely restricted to order-finding in problems that are known to have some periodic property.

Quantum order-finding

Now we come back to the order-finding problem. The QFT algorithm gives a precise answer only when the input has a period dividing 2^n, but we need to deal with the case where the period r has other factors. It turns out that if n is moderately large, the error will be small enough for us to deal with.

Recall that in Shor's algorithm (Figure 3.6), after the measurement of the second register, the first register is left in the state

$$|\phi_1\rangle = \frac{1}{\sqrt{m}} \sum_{q=0}^{m-1} |qt + r\rangle,$$

where $m = \lceil (2^n - r)/t \rceil$. The QFT procedure returns

$$\begin{aligned} |\phi_2\rangle &= \frac{1}{\sqrt{2^n m}} \sum_{y=0}^{2^n-1} \sum_{q=0}^{m-1} \omega^{(qt+r)y} |y\rangle \\ &= \frac{1}{\sqrt{2^n m}} \sum_{y=0}^{2^n-1} \left(\omega^{ry} \sum_{q=0}^{m-1} (\omega^{ty})^q \right) |y\rangle. \end{aligned}$$

It is obvious that the magnitude of $|y\rangle$ is determined by the sum $\sum_{q=0}^{m-1} (\omega^{ty})^q$. Since $\omega = \mathrm{e}^{2\pi \mathrm{i}/2^n}$, we have $\omega^{ty} = \mathrm{e}^{2\pi \mathrm{i}(ty/2^n)}$. The closer $ty/2^n$ is to an integer (i.e., $y/2^n$ close to k/t for some integer k), the larger the magnitude is for $|y\rangle$. Therefore a measurement is likely to yield $y \approx 2^n k/t$. Using continued fraction we can approximate t or at least a factor of t with a high probability.

3.3.3 Hard HSP instances and post-quantum cryptosystem

The study of HSP has extended Shor's algorithm to all HSP instances with abelian groups [Kit95] and even some semi-direct products of abelian groups [BCvD05]. However, for HSP over general non-abelian groups, this approach is not sufficient. Some of such HSP instances form the basis of most post-quantum cryptosystems.

Among these instances, dihedral group HSP is associated with the unique shortest vector problem (uSVP) in lattices [Reg04b], which relates to the security of some lattice-based cryptosystems such as Regev's design in [Reg04a]; the security of certain McEliece and McEliece-like cryptosystems can be reduced to non-abelian group HSP as well [DMR11].

The reduction of the lattice problem to the dihedral HSP is done by constructing a "discrete Gaussian state", which is a superposition of lattice points with magnitude of each point proportional to the value assigned to the point by a continuous Gaussian distribution. More precisely, a measurement of such a state yields $\mathbf{x} \in L$ with probability

$e^{-\pi\|\mathbf{x}/r\|^2}$. Suppose we can construct such states for arbitrary point $\mathbf{x} \in \mathbb{R}^n$ and radius r of the discrete Gaussian ball, then by controlling r, we can obtain a lattice point close to any given $\mathbf{x}$. Thus it solves the closest vector problem (CVP). Regev also shows that a solution to the learning with errors (LWE, see Section 4.3) problem implies that such a quantum state can be constructed [Reg09].

Finally, it is worth mentioning that even if these problems prove to be intractable for quantum computers, a general quadratic speedup for brute-force attacks is available through Grover's algorithm [Gro96]. If practical quantum computers can be made, we still need to increase parameters of these cryptosystems to provide an appropriate level of security.

Chapter 4

Fully Homomorphic Encryption and Learning with Errors

4.1 Background

As early as 1978, Rivest *et al.* foresaw the potential application of cloud computing and described a “privacy homomorphism” to support computation over ciphertexts [RAD78]. Since then, the search for a fully homomorphic encryption (FHE) scheme has been around for more than three decades, but only until 2009 did Gentry announce the first plausible design [Gen09].

The first construction is very complicated. Fortunately, as new FHE designs emerge, FHE schemes become much simpler and more efficient and require less security assumptions at the same time. The most notable new design is probably those based on the LWE problem, an equivalent to hard coding problems. The theoretical improvement of performance is also verified by the HElib implementation [GHS12] that is capable to evaluate AES encryption, which involves a lot of operations.

4.2 The Framework of FHE

We will review Gentry’s design in Section 4.2.1. Here we briefly discuss the definition of FHE and give an overview of the currently existing designs.

Definition 4.2.1 (FHE scheme)**.** A public-key encryption scheme ($\texttt{Enc}, \texttt{Dec}$) is a *fully homomorphic encryption (FHE) scheme* if for some universal set of operations U, there exists an evaluating function $\texttt{Eval}$ such that if we let f be a function consisting of operations from the set U, $\mathbf{m} = \left[m_0, m_1, \ldots, m_k\right]$, and $\mathbf{c} = \left[\texttt{Enc}(pk, m_0), \texttt{Enc}(pk, m_1), \ldots, \texttt{Enc}(pk, m_k)\right]$, then

1. (Correctness) Evaluation over ciphertext is equivalent to evaluation over plaintext, i.e.,
$$\texttt{Dec}(sk, \texttt{Eval}(pk, f, \mathbf{c})) = f(\mathbf{m}).$$

2. (Performance) There exists a polynomial $p(x)$ such that for any security parameter λ,
$$|\texttt{Eval}(pk, f, \mathbf{c})| \leq p(\lambda).$$

The purpose of the performance requirement in Definition 4.2.1 is to rule out trivial schemes which simply concatenate the function definition f and parameters $\mathbf{c}$. In other words, it forces the schemes to actually evaluate the function under the mask of encryption.

4.2.1 Gentry's roadmap

Figure 4.1 shows the process Gentry proposed to construct an FHE scheme. Gentry adopted circuits as the computation model, and chose addition and multiplication as the universal set.

The first step is to pick a *somewhat homomorphic encryption (SWHE) scheme* that supports both addition and multiplication. The SWHE scheme usually achieves security by adding a small noise during encryption and eliminating it through decryption. Such approach causes two problems:

1. The magnitude of noise will grow with the evaluation of functions over ciphertexts, and eventually the noise will be large enough to result in an decryption error. The growth of noise poses a limit on the number of operations in a function that we can homomorphically evaluate.

2. The amount of noise increased by each operation differs. It often means that we can evaluate one operation many times more than we can evaluate another.

To understand what functions can be evaluated by the SWHE scheme and to evaluate functions we are interested in, we need to estimate the noise level and choose suitable parameters. In the language of circuits, by doing so we can evaluate all functions with circuits up to a certain depth, hence we have a *leveled fully homomorphic encryption (leveled FHE) scheme*.

To turn a leveled FHE scheme into an FHE scheme, Gentry proposed a "bootstrapping" method that we can construct a leveled FHE scheme capable of homomorphically evaluating its own decryption function. Since the noise is eliminated after decryption, the resulted ciphertext may contain less noise and allow us to continue performing operations on it. However, to achieve this goal, we need to address two problems:

1. Decryption complexity usually grows with the increase of homomorphic evaluation capability. In Gentry's design, the complexity decryption function is decreased by adding "hint" about the secret key to alleviate the decryption computation. Later designs based on LWE have extremely simple decryption function (inner product and modular arithmetic) and do not need this extra step any more.

2. To evaluate the decryption function, an encryption of the secret key must be provided. Hence the security of a FHE scheme following this framework will not only depend on the hardness of the underlying problem, but also depend on assumption on key-dependent message (KDM) security.

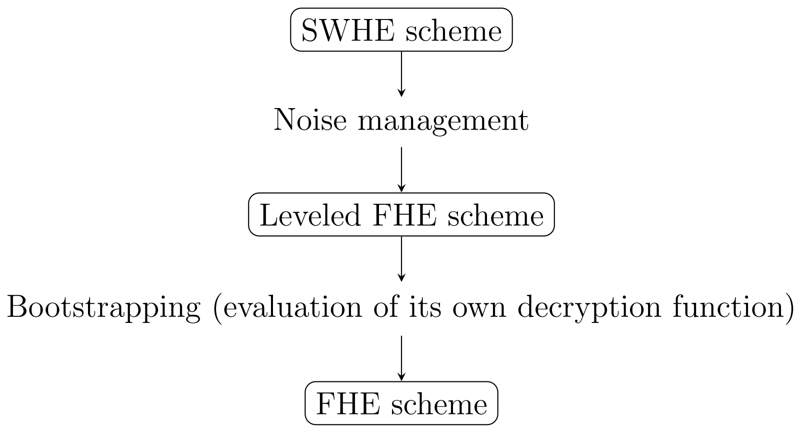


Figure 4.1: A flowchart of Gentry's roadmap

Basically all the existing FHE scheme designs follow this framework, though they are based on different problems. In particular, they stick to the universal set of operations that Gentry selected for his first FHE scheme (addition and multiplication) and operate on a ring. An interesting deviation from this framework is the attempt by Goldwasser *et al.* to use Turing machines instead of circuits as the computation model [GKP+13], yet there has not been much more investigation into this direction.

4.2.2 Recent development

The first FHE scheme, which Gentry proposed, is based on ideal lattices (lattices defined by $\mathbb{Z}[x]/(f)$ for some irreducible polynomial $f \in \mathbb{Z}[x]$) [Gen09]. Messages are encrypted Since the lattice is itself a ring, it natually supports addition and multiplication. But making it fully homomorphic is very difficult, because the original decryption function of this scheme is too complicated to be evaluated by itself. Gentry introduced into the public key some information about the secret key to alleviate the decryption complexity ("squashing" the decryption circuit), but this approach requires additional assumptions on the intractability of the sparse subset sum problem.

After the publication of the first construction, various new designs were proposed. Some base the security on different hardness assumptions such as approximate GCD [vDGHV10, KLYC13], others are improvements on the first construction such as the elimination of the demand for a squashing step [GH11a].

Recently Brakerski *et al.* introduced learning with errors (LWE) into the construction of new FHE schemes and showed that it has several advantages over existing schemes including

1. Low evaluation complexity (faster evaluation),
2. Low decryption complexity (squashing no longer needed), and
3. Hardness of LWE equivalent to worst-case lattice problems.

We are going to see LWE and LWE-based schemes in the following sections.

4.3 Learning with Errors (LWE)

In this section, we introduce the learning with errors (LWE) problems, giving definitions and summarizing some hardness results. FHE constructions based on LWE will present in Section 4.4.2.

Definition 4.3.1 ((Search) LWE)**.** Given integers $n = n(\lambda)$ and $q = q(\lambda) \geq 2$, and distribution $\chi = \chi(\lambda)$ over $\mathbb{Z}$. Fix secret $\mathbf{s} \leftarrow_U \mathbb{Z}_q^n$, obtain N samples $[\,\mathbf{a}_i \,|\, b_i\,], 0 \leq i \leq N-1$ by letting $\mathbf{a}_i \leftarrow_U \mathbb{Z}_q^n$, $e_i \leftarrow_\chi \mathbb{Z}_q$, and computing $b_i = \mathbf{a}_i\mathbf{s}^\top + e_i$. The samples form an $N \times (n+1)$ matrix. The (search) LWE problem is to find the secret $\mathbf{s}$.

An algorithm $\mathcal{A}$ is said to (N, t, ε)-solve the (search) LWE problem if it runs for at most time t, takes N samples and outputs the correct $\mathbf{s}$ with probability ε.

Note that the running time is at least $O(N)$ (input size), which means an upper bound of running time implies an upper bound of the number of samples. Specifically, an algorithm that efficiently solves LWE problem, which runs in polynomial time, must take only polynomially many samples. Thus the number of samples is often omitted in some descriptions of LWE. Here we use both interchangeably.

The belief that LWE is intractable can be established by two observations. First of all, the search LWE problem is equivalent to the general decoding problem (Definition 2.3.2). Suppose there are N samples $[\,\mathbf{a}_0 \,|\, b_0\,], \ldots, [\,\mathbf{a}_{N-1} \,|\, b_{N-1}\,]$, we have the relationship

$$\begin{bmatrix} b_0 \\ \vdots \\ b_{N-1} \end{bmatrix} = \begin{bmatrix} \mathbf{a}_0 \\ \vdots \\ \mathbf{a}_{N-1} \end{bmatrix} \mathbf{s}^\top + \begin{bmatrix} e_0 \\ \vdots \\ e_{N-1} \end{bmatrix} = A\mathbf{s}^\top + \mathbf{e}^\top.$$

If we view A as the generator matrix, $\mathbf{s}$ as the message, and $\mathbf{e}$ as the error vector, then clearly an algorithm that decodes a random linear code could obtain $\mathbf{s}$ and hence solve (search) LWE.

The second observation is the reduction of LWE to worst-case lattice problems, e.g., GapSVP and SIVP. The first such reduction was contributed by Regev but it involves quantum algorithm (the reduction works by constructing a suitable initial state described in Section 3.3.3) [Reg09]; Peikert came up with a classical one that only works for large q (exponential of n) [Pei09]; an average-case classical reduction was finally done by Brakerski *et al.* in 2013 [BLP+13].

Definition 4.3.2 (Decisional LWE)**.** Given integers $n = n(\lambda)$ and $q = q(\lambda) \geq 2$, and distribution $\chi = \chi(\lambda)$ over $\mathbb{Z}$, the decisional LWE problem is to distinguish the following two distributions of $[\,\mathbf{a}_i \,|\, b_i\,]$:

1. Uniform distribution: $[\,\mathbf{a}_i \,|\, b_i\,] \leftarrow_U \mathbb{Z}_q^{n+1}$

2. Fix secret $\mathbf{s} \leftarrow_U \mathbb{Z}_q^n$, compute $[\, \mathbf{a}_i \,|\, b_i \,]$ by

$$\begin{array}{lcl} \mathbf{a}_i & \leftarrow_U & \mathbb{Z}_q^n, \\ e_i & \leftarrow_\chi & \mathbb{Z}_q, \\ b_i & = & \mathbf{a}_i \mathbf{s}^\top + e_i \end{array}$$

It has been shown that the search and decisional versions of LWE are equivalent for $q = p^e$, where p is a prime, in the sense that search LWE can be reduced to decisional LWE with only polynomial overhead [ACPS09]. We will not distinguish between them in later sections.

4.3.1 Extensions of the LWE problem

The LWE problem has two extensions. For completeness, they are defined below. The GLWE problem is a generalization of LWE and RLWE problems, i.e., GLWE problems with parameter $d = 1, n > 1$ are LWE problems; GLWE problems with parameter $d > 1, n = 1$ are RLWE problems, yet little research has been done for other cases.

Definition 4.3.3 (RLWE)**.** Given $d = d(\lambda)$ that is some power of 2, integer $q = q(\lambda) \geq 2$, ring $R = \mathbb{Z}[x]/(x^d+1)$, and distribution $\chi = \chi(\lambda)$ over R. Let $R_q = R/qR = \mathbb{Z}_q[x]/(x^d+1)$, the RLWE problem is to distinguish the following two distributions of $[\, a_i \,|\, b_i \,]$:

1. Uniform distribution: $[\, a_i \,|\, b_i \,] \leftarrow_U R_q^2$;

2. Fix secret $s \leftarrow_U R_q$, compute $[\, a_i \,|\, b_i \,]$ by

$$\begin{array}{lcl} a_i & \leftarrow_U & R_q \\ e_i & \leftarrow_\chi & R_q \\ b_i & = & a_i \cdot s + e_i \end{array} .$$

Definition 4.3.4 (GLWE)**.** Given integer dimension $n = n(\lambda)$, polynomial $f(x) = x^d + 1$ where $d = d(\lambda)$ is a power of 2, prime $q = q(\lambda)$ and distribution $\chi = \chi(\lambda)$ over $R = \mathbb{Z}[x]/f(x)$. Let $R_q = R/qR = \mathbb{Z}_q[x]/f(x)$, the GLWE problem is to distinguish the following two distributions of $[\, \mathbf{a}_i \,|\, b_i \,]$:

1. Uniform distribution: $[\, \mathbf{a}_i \,|\, b_i \,] \leftarrow_U R_q^{n+1}$;

2. Fix secret $\mathbf{s} \leftarrow_U R_q^n$, compute $[\, \mathbf{a}_i \,|\, b_i \,]$ by

$$\begin{array}{lcl} \mathbf{a}_i & \leftarrow_U & R_q^n, \\ e_i & \leftarrow_\chi & R_q, \\ b_i & = & \mathbf{a}_i \mathbf{s}^\top + e_i \end{array} .$$

Definition 4.3.5 (LPN)**.** Given integer dimension $n = n(\lambda)$, $\chi = \chi(\lambda)$ being a Beroulli distribution. The LPN problem is to distinguish the following two distributions of $[\,\mathbf{a}_i \,|\, b_i\,]$:

1. Uniform distribution: $[\,\mathbf{a}_i \,|\, b_i\,] \leftarrow_U \mathbb{Z}_2^{n+1}$;
2. Fix secret $\mathbf{s} \leftarrow_U \mathbb{Z}_2^n$, compute $[\,\mathbf{a}_i \,|\, b_i\,]$ by

$$\begin{array}{lcl} \mathbf{a}_i & \leftarrow_U & Z_2^n, \\ e_i & \leftarrow_\chi & Z_2, \\ b_i & = & \mathbf{a}_i\mathbf{s}^\top + e_i \end{array} .$$

The LPN problem is a special case of the LWE problem with $p = 2$ (actually, LWE was originallyu obtained by extending LPN to arbitrary modulus). Therefore, properties of LWE, such as the equivalence between decisional and search versions, also apply to LPN [KS06]. Some researchers suspect that LPN is easier than LWE. However, it remains an open question.

4.4 Design of FHE Schemes

4.4.1 Overview

Security

Table 4.1 shows a summary of known security levels of SWHE schemes. Obviously, homomorphic encryption schemes cannot be IND-CCA2 secure. Actually, if the adversary is allowed to choose ciphertext after receiving the challenge as in the CCA2 model, he/she can almost arbitrarily decrypt or fabricate messages due to the homomorphism.

Given the impossibility of CCA2 secure FHE schemes, the highest security level they can reach is IND-CCA1. It seems difficult to construct IND-CCA1 secure FHE schemes. In fact, all schemes in the IND-CPA column of Table 4.1 are vulnerable to CCA1 key-recovery attacks [LMSV10, ZPS12].

Implementations and performance

The first working implementation of fully homomorphic encryption is presented in [GH11b]. Its success is based on a lot of optimizations on Gentry's scheme. However, theoretical analysis of LWE-based schemes proves them to be more efficient than other

Security / Problem	Insecure	IND-CPA	IND-CCA1
Lattice		Gentry [Gen09]	ccSHE [LMSV12]
AGCD		vDGHV [vDGHV10]	
LWE		BV [BV14] BGV [BGV12]	
LPN	BL [BL11] [Bra13]		

Table 4.1: Security of some SWHE schemes

existing schemes ($\tilde{O}(\lambda^{3.5})$ versus $\tilde{O}(\lambda^2)$ [BGV12]). Soon after the proposal of the construction, Gentry *et al.* implemented the BGV scheme and successfully completed a homomorphic evaluation of AES encryption [GHS12]. Their code is published online at `https://github.com/shaih/HElib`.

4.4.2 BGV — the FHE scheme from LWE

In this section we summarize the BGV scheme [BGV12], which is based on GLWE. For simplicity, we describe the scheme with LWE, but our proofs also apply to the other two.

From the perspective of the evaluation circuit, the BGV scheme can be viewed as basic schemes, with decreasing moduli and corresponding keys, interconnected by the "key switching" and "modulus switching" procedures. Unlike previous schemes, whose noise grows with evaluations to a certain limit, the modulus switching procedure keeps the noise level almost constant among levels but slowly decrease the modulus. Figure 4.2 is an illustration of this process.

Below we will introduce the basic scheme and the homomorphic operations. We describe the algorithms with ciphertext space $\mathbb{Z}_q^{n+1}$ (LWE) for simplicity. They can be easily generalized to operate over R_q^{n+1} (GLWE), where R_q denotes the quotient ring $\mathbb{Z}_q[x]/(f(x))$. Note that the plaintext space is always $\mathbb{Z}_2$, regardless of the ciphertext space.

We will use the same notation — security parameter λ, dimension n, modulus q, and error distribution χ — from the decisional LWE problem (Definition 4.3.2). We also use N to denote the number of samples.

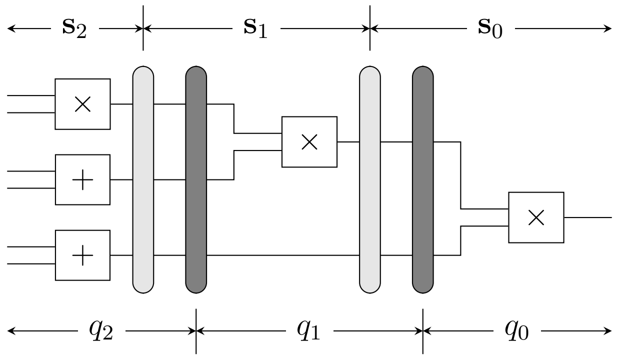


Figure 4.2: An illustration of the execution of BGV scheme. Integers q_i are moduli and $\mathbf{s}_i$ are corresponding keys. Light gray and dark gray bars are key switching and modulus switching procedures, respectively.

The basic scheme

The basic scheme is a normal public-key encryption scheme ($\mathtt{KeyGen}, \mathtt{Enc}, \mathtt{Dec}$), with algorithms specified below.

- $\mathtt{KeyGen}(\mathbb{1}^\lambda)$

  Randomly generate a matrix $A' \leftarrow_U \mathbb{Z}_q^{N\times n}$ and a vector $\mathbf{s}' \leftarrow_U \mathbb{Z}_q^n$. Obtain $\mathbf{e}$ by sampling $e_i \leftarrow_\chi \mathbb{Z}_q, 0 \le i \le N-1$. Let $\mathbf{b} = A'\mathbf{s}'^\top + 2\mathbf{e}$ and $A = [\,\mathbf{b}^\top \,|\, -A'\,]$.

  Public key: $pk = A$.

  Private key: $sk = \mathbf{s} = [\,1\,|\,\mathbf{s}'\,]$.

- $\mathtt{Enc}(pk, m)$

  Expand plaintext $m \in \mathbb{Z}_2$ to an $(n+1)$-dimensional vector by adding zeros: $\mathbf{m}' = [\,m\,|\,\mathbf{0}\,]$. Randomly select $\mathbf{r} \in \mathbb{Z}_q^N$, then $\mathbf{c} = \mathtt{Enc}(pk, m) = \mathbf{r}A + \mathbf{m}'$.

- $\mathtt{Dec}(sk, \mathbf{c})$

  Decryption is done by $m = \mathtt{Dec}(sk, \mathbf{c}) = (\mathbf{c}\mathbf{s}^\top \bmod q) \bmod 2$, where $x \bmod q$ falls into the interval $(-q/2, q/2]$.

It is straightforward to verify the correctness of this basic scheme. Below we give a security proof.

Definition 4.4.1 (Decisional LWE (DLWE) assumption)**.** The decisional LWE (DLWE) assumption is that there exists no algorithm $\mathcal{A}$ that (N, t, ε)-solves decisional LWE problem with $t = O(\text{poly}(n))$ and non-negligible ε.

Lemma 4.4.1. *Under the* LWE *assumption, the public key A of the basic BGV scheme is indistinguishable from $B \leftarrow_U \mathbb{Z}_q^{N\times(n+1)}$.*

Lemma 4.4.2. *If $A \in \mathbb{Z}_q^{N\times(n+1)}$ is indistinguishable from $B \in \mathbb{Z}_q^{N\times(n+1)}$, then $\mathbf{r}A$ is indistinguishable from $\mathbf{r}B$ for $\mathbf{r} \leftarrow_U \mathbb{Z}_q^N$.*

Lemma 4.4.3. *If $\mathbf{c}_0, \mathbf{c}_1 \in \mathbb{Z}_q^{n+1}$ are both indistinguishable from $\mathbf{r} \in \mathbb{Z}_q^{n+1}$, then $\mathbf{c}_0$ is indistinguishable from $\mathbf{c}_1$.*

Theorem 4.4.4. *Under the* LWE *assumption, the basic BGV scheme is IND-CPA secure.*

Proof. For encryptions of message $m = 0$: by Lemma 4.4.1, the public key A is indistinguishable from random matrix $B \leftarrow_U \mathbb{Z}_q^{N\times(n+1)}$. By Lemma 4.4.2, $\mathbf{r}A$ (the encryption of 0) is indistinguishable from a random vector $\mathbf{r}' \leftarrow_U \mathbb{Z}_q^N$.

For encryptions of message $m = 1$: similar to the case where $m = 0$, with the only difference being that the first component of $\mathbf{c}$ is incremented by 1, which does not affect the indistinguishability because it is a bijective mapping on $\mathbb{Z}_q$. Therefore $\mathbf{r}A + [\,1\,|\,\mathbf{0}\,]$ (the encryption of 1) is indistinguishable from a random vector $\mathbf{r}' \leftarrow_U \mathbb{Z}_q^N$.

Finally, by Lemma 4.4.3, encryptions of 0 are indistinguishable from encryptions of 1, i.e., the basic scheme is IND-CPA secure. □

We should note that the BGV scheme still suffers from a CCA1 key-recovery attack [LMSV10].

Homomorphic operations

Evaluation The two operations that BGV scheme can homomorphically evaluated are still addition and multiplication. Since its decryption function is a modular inner product, it can be viewed as a linear polynomial of the secret key $\mathbf{s}$ with coefficients from the ciphertext $\mathbf{c}$. Hence we have plaintext corresponding to the point-value representation and ciphertext corresponding to the coefficient representation of the same polynomial, and additions and multiplications on the plaintext are equivalent to polynomial additions and multiplications. With coefficient representation, addition is trivial, but multiplication will result in a polynomial of the tensor product $\mathbf{s} \otimes \mathbf{s}$ of the secret key.

Key Switching Key switching eliminates the growth of key size and ciphertext size by transform the ciphertext $\mathbf{c}_1$ under $\mathbf{s}_1 \otimes \mathbf{s}_1$ to $\mathbf{c}_2$ under $\mathbf{s}_2$, preserving the inner product. During the transform, we have two sets of parameters. We will denote them by subscripts corresponding to $\mathbf{s}_1$ and $\mathbf{s}_2$.

Here we demonstrate the idea of key switching with a more straightforward construction first. Note that for a secret key $\mathbf{s}$, we can sample as many times as we want in the generation of public key. If we set $N_2 = n_1^2$ and let $A_2 = [\, \mathbf{b}_2^\top \mid -A_2' \,]$ be the generated public key, then apparently

$$B = [\, \mathbf{b}_2^\top + (\mathbf{s}_1 \otimes \mathbf{s}_1)^\top \mid -A_2' \,]$$

satisfies $(\mathbf{c}_1 B \mathbf{s}_2^\top \bmod q) \bmod 2 = (\mathbf{c}_1(\mathbf{s}_1 \otimes \mathbf{s}_1)^\top \bmod q) \bmod 2$. Therefore we can let $\mathbf{c}_2$ be $\mathbf{c}_1 B$.

In the above construction, the dimension of the matrix B is $N_2 \times (n_2+1) = n_1^2 \times (n_2+1)$. However, Brakerski *et al.* introduced two different representations of $\mathbf{c}_1$ and $\mathbf{s}_1 \otimes \mathbf{s}_1$ into the key-switching process, namely the outputs of BitDecomp and Powersof2 transforms. Their construction has the dimension of B being $n_1^2 \lceil \log q_1 \rceil \times (n_2 + 1)$.

Modulus switching Modulus switching is the core of noise management in the BGV scheme. Based on Lemma 4.4.5 [BGV12, Wei13] quoted below, we can reduce the magnitude of noise by switching from a larger modulus to a smaller one.

Here d is the degree of the polynomial $f(x)$ defining $R = \mathbb{Z}[x]/(f(x))$ in RLWE and GLWE; $\|a\|$ is the Euclidean norm of polynomial coefficients of $a \in R$; γ_R is a parameter defined by $\gamma_R = \max\left\{\frac{\|ab\|}{\|a\|\|b\|} : a, b \in R\right\}$; $\|\mathbf{s}\|_1^{(R)}$ is the l_1 norm defined as $\sum_{i=0}^{n-1} \|s_i\|$; $\mathsf{Scale}(\mathbf{c}, q, p, r)$ returns the vector $\mathbf{c}' \in R^n$ closest to $(p/q) \cdot \mathbf{c}$ such that $\mathbf{c} \equiv \mathbf{c}' \pmod r$.

Lemma 4.4.5. *Let $q > p > r$ be positive integers such that $q \equiv p \equiv 1 \pmod r$ and $\mathbf{c}' = \mathsf{Scale}(\mathbf{c}, q, p, r)$. Then for any $\mathbf{s} \in R^n$ such that $\left\|\mathbf{c}\mathbf{s}^\top \bmod q\right\| < q/2 - (q/p) \cdot \gamma_R \cdot (r/2) \cdot \sqrt{d} \cdot \|\mathbf{s}\|_1^{(R)}$, we have*

$$(\mathbf{c}'\mathbf{s}^\top \bmod p) \equiv (\mathbf{c}\mathbf{s}^\top \bmod q) \pmod r$$

and

$$\left\|\mathbf{c}'\mathbf{s}^\top \bmod p\right\| \leq (p/q) \cdot \left\|\mathbf{c}\mathbf{s}^\top \bmod q\right\| + \gamma_R \cdot (r/2) \cdot \sqrt{d} \cdot \|\mathbf{s}\|_1^{(R)}.$$

In particular, in LWE cases where $r = 2$,

$$\left\|\mathbf{c}'\mathbf{s}^\top \bmod p\right\| \leq (p/q) \cdot \left\|\mathbf{c}\mathbf{s}^\top \bmod q\right\| + \|\mathbf{s}\|_1^{(R)}.$$

Parameter selection

We must take into consideration two aspects during parameter selection:

1. Security: since the security is based on the basic scheme and hence GLWE problems, it suffices to choose appropriate n, q, d and χ for the basic scheme. See [GHS12, Appendix C] for an example of analysis.

2. Correctness and performance as a FHE scheme (homomorphic evaluation capability): correctness is directly related to the modulus switching procedure, which is responsible for reducing the magnitude of noise. Therefore, parameter selection for correctness is mostly about the relationship between adjacent moduli. This issue is addressed by Theorem 4.4.6 [BGV12].

Theorem 4.4.6. *There exists some some $\mu = \Theta(\log \lambda + \log L)$, for a BGV scheme that has security parameter λ and L moduli, with each modulus q_i being a $(\mu(j+1))$-bit integer, the BGV scheme correctly evaluates circuits of depth up to L with addition and multiplication over R_2.*

Performance

We compare the efficiency of the basic BGV scheme based on LWE and McEliece by public key size in Table 4.2 and ciphertext size in Table 4.3.

Security	McEliece			BGV		
	(n, k, t)	*pk* (MB)	*sk* (MB)	$(n, \log q, N)$	*pk* (MB)	*sk* (KB)
80	$(2048, 1751, 27)$	0.4483	1.356	$(354, 16, 11329)$	7.671	0.6934
128	$(2960, 2288, 56)$	0.8466	2.597	$(460, 16, 14721)$	12.962	0.9004
256	$(6624, 5129, 115)$	4.247	13.020	$(791, 17, 26895)$	43.113	1.641

Table 4.2: Comparison of key sizes. The parameter n and $\log q$ of basic BGV scheme is the minimal pairs taken from [Wei13], where LWE is considered and χ is fixed to be a normal distribution with standard deviation $\sigma = 1$, N is the minimal value satisfying $N > 2n \log q$ [BGV12]. Note that BGV private key size is displayed in KB.

We can see that on the same security level, the basic BGV scheme has a much larger public key but a smaller private key. This observation aligns with the simple decryption function of BGV scheme. We also note that in order to support homomorphic operations,

Security	McEliece			BGV		
	$\lvert\mathbf{m}\rvert$ (bits)	$\lvert\mathbf{c}\rvert$ (bits)	Expansion	$\lvert m\rvert$ (bits)	$\lvert\mathbf{c}\rvert$ (bits)	Expansion
80	1751	2048	1.170	1	5680	5680
128	2288	2960	1.294	1	7376	7376
256	5129	6624	1.291	1	13464	13464

Table 4.3: Ciphertext size compared with plaintext size. Both McEliece and basic BGV schemes use parameters from Table 4.2.

key-switching matrices must be added to the public key, which will significantly increase the public key size (we omit the comparison here). Generally, these increases are a trade-off to gain the fully homomorphic property of BGV.

McEliece and BGV schemes have similar encryption functions, which consist of a matrix-vector multiplication and a vector addition. Therefore we can expect the running time of `Enc` to be proportional to the public key size. In contrast, the running time of `Dec` of basic BGV scheme will be much less that that of McEliece, due to its simplicity. The exact time depends on the particular implementation. For example, the HElib implementation [HS14] uses a variant of BGV scheme to enable SIMD operations and it also optimizes the performance in a few operations such as key switching by a new algorithm [GHS12].

As a demonstration, in Table 4.4 we give a few timing data of HElib under the same testing environment (Intel i5-3210M quadcore with 4 GB memory). HElib operates on the ring $\mathbb{Z}_q[x]/\Phi_u(x)$, where $\Phi_u(x)$ is the u-th cyclotomic polynomial whose degree is $\phi(u)$. Values in the table are select by HElib using the following parameters:

- Security parameter $l = 80, 128, 192$, and 256;
- The number of levels of the circuit it can evaluate $L = 1$ (i.e., only the basic scheme);
- Hamming weight of secret key $w = 64$;
- Ciphertext space $\mathbb{Z}_2$;
- Other parameters such as the number of columns in key-switching matrices and the minimum number of plaintext slots (for SIMD operations) are set to 0.

Note that HElib is based on RLWE and more efficient than the LWE-based version of BGV.

Security (l)	u	Time of `KeyGen` (s)	Time of `Enc` (ms)	Time of `Dec` (ms)
80	3133	0.18	3.18	3.58
128	4051	0.26	3.67	3.46
192	4859	0.24	5.34	7.78
256	5461	0.16	5.64	16.19

Table 4.4: Timing of HElib

The time of `KeyGen` relies on the speed of arithmetic operations over $\mathbb{Z}_q[x]/(\Phi_u(x))$. It varies since for certain u's the polynomials may be easier to evaluate.

Another observation in this table is that decryption takes more time than encryption. This is due to the encoding for SIMD support. If we exclude the encoding and decoding processes, `Dec` will take less time than `Enc`, which agrees with the fact that decryption has fewer operations than encryption in the scheme. However, the exact cause needs to be further investigated.

Chapter 5

AMD Codes

5.1 Background

First introduced in [CDF+08], algebraic manipulation detection (AMD) codes are a cryptographic primitive very similar to error detecting codes and message authentication codes (MACs). Error detecting codes are designed for random errors (bit flips) in storage and transmission and thus may not prevent an adversary flipping more bits than the error-detection capability, while MACs usually needs a pre-shared key. In this case, AMD codes, aiming to the detection of malicious manipulations without keys, will be more helpful.

In the model of AMD codes, there is a storage device that is leakage-proof but temper-prone, i.e., an adversary can manipulate the data stored in it, though not knowing any information about the data. Precisely speaking, the storage device is capable of storing a single element α of some finite abelian group $\mathcal{G}$. Any manipulation of the data can be written additively as $\alpha + \delta$, for some $0 \neq \delta \in \mathcal{G}$, called an algebraic manipulation. AMD codes guarantee that for any δ, such manipulation will be detected with high probabilities.

This model seems to be ridiculous at the first sight. However, a mask of encryption (even a linear encryption) is sufficient to create such a leakage-proof but temper-prone device. Therefore, AMD codes are often used in conjunction with other cryptographic primitives. Their applications include robust secret-sharing, fuzzy extractors, non-malleable codes and secure memories.

In this chapter, we first review the definitions, known bounds, and applications of AMD codes in Section 5.2; then we obtain our new bounds for weak systematic AMD codes by investigating the nonlinearity of functions in Section 5.3, and show that they are better than known bounds with constructions.

5.2 Known Results

5.2.1 Definitions

Definition 5.2.1 ((Strong) (m, n, ε)-AMD codes)**.** Let $\mathcal{S}$ be a set of size $m > 1$ and $\mathcal{G}$ be an abelian group of order n. Consider a pair $(\mathtt{Enc}, \mathtt{Dec})$ formed by a probabilistic encoding map $\mathtt{Enc} : \mathcal{S} \to \mathcal{G}$ and a deterministic decoding map $\mathtt{Dec} : \mathcal{G} \to \mathcal{S} \cap \{ \perp \}$ such that $\mathtt{Dec}(\mathtt{Enc}(s)) = s$ with probability 1 for every $s \in \mathcal{S}$. The pair $(\mathtt{Enc}, \mathtt{Dec})$ is a (strong) (m, n, ε)-AMD code if for every $s \in \mathcal{S}$ and for every $\delta \in \mathcal{G}$, the probability

$$\Pr [\mathtt{Dec}(\mathtt{Enc}(s) + \delta) \notin \{ s, \perp \} \mid s, \delta] \tag{5.1}$$

is at most ε.

There is also a weak version of AMD codes defined below, where the probability (5.1) for certain values of s may exceed ε:

Definition 5.2.2 (Weak (m, n, ε)-AMD codes)**.** Let $\mathcal{S}$, $\mathcal{G}$, $\mathtt{Enc}$ and $\mathtt{Dec}$ be defined as in Definition 5.2.1. The pair $(\mathtt{Enc}, \mathtt{Dec})$ is an (m, n, ε)-AMD code if for s uniformly sampled over $\mathcal{S}$ and for every $\delta \in G$, the probability

$$\Pr [\mathtt{Dec}(\mathtt{Enc}(s) + \delta) \notin \{ s, \perp \} \mid \delta] \tag{5.2}$$

is at most ε.

Let p_s denote the probability that s is chosen when sampling. Since s is uniformly sampled, the probability (5.2) can be computed as

$$\begin{aligned}(5.2) &= \sum_{s \in \mathcal{S}} \Pr [\mathtt{Dec}(\mathtt{Enc}(s) + \delta) \notin \{ s, \perp \} \mid s, \delta] \cdot p_s \\ &= \frac{1}{|\mathcal{S}|} \sum_{s \in \mathcal{S}} \Pr [\mathtt{Dec}(\mathtt{Enc}(s) + \delta) \notin \{ s, \perp \} \mid s, \delta] .\end{aligned}$$

Definition 5.2.3 (Systematic AMD codes)**.** An AMD code is systematic if the source set $\mathcal{S}$ is a group and the encoding is of the form

$$\begin{aligned}\mathtt{Enc} : \mathcal{S} &\to \mathcal{G} = \mathcal{S} \times \mathcal{G}_1 \times \mathcal{G}_2 \\ s &\mapsto (s, x, f(x, s)),\end{aligned}$$

for some function $f : \mathcal{G}_1 \times \mathcal{S} \to \mathcal{G}_2$, where $\mathcal{G}_1$ and $\mathcal{G}_2$ are groups and x is taken uniformly at random from $\mathcal{G}_1$. The decoding function of a systematic AMD code is naturally given by

$$\texttt{Dec}(s, x, e) = \begin{cases} s, & \text{if } e = f(x, s), \\ \perp, & \text{otherwise.} \end{cases}$$

5.2.2 Known bounds

Cramer *et al.* gave a few bounds for both strong and weak versions of (m, n, ε)-AMD codes [CFP13]. We summarize them here.

Theorem 5.2.1 (Lower bound of n in (strong) AMD codes)**.** *Every (strong) (m, n, ε)-AMD code satisfies that*

$$n \geq \frac{m-1}{\varepsilon^2} + 1.$$

Theorem 5.2.2 (Lower bound of n in weak AMD codes)**.** *Every weak (m, n, ε)-AMD code satisfies that*

$$n \geq \frac{m-1}{\varepsilon} + 1.$$

Theorem 5.2.3 (Lower bound of ε in (strong) systematic AMD codes)**.** *Every (strong) systematic $(m, mn_1n_2, \varepsilon)$-AMD code with $\varepsilon < 1$ satisfies that*

$$\varepsilon \geq \frac{\log m}{n_1 \log n_2}.$$

All three bounds can be attained, as shown by detailed constructions in their paper.

We notice that Cramer *et al.* mentioned the equivalence between weak systematic AMD codes and robust codes given by Karpovsky *et al.* [KT04], but the latter was mostly concerned about specific construction, and a general bound was not established.

5.2.3 Applications

AMD codes can be used to construct or enhance a variety of security schemes. Here we list two of them: robust secret-sharing schemes and non-malleable codes.

Robust secret-sharing scheme A secret-sharing scheme is a pair of functions (`Share`, `Recover`), such that the `Share` function distributes information about the secret among a set of entities, and the `Recover` function takes in a subset of the distributed information and reconstruct the secret. Some secret-sharing schemes, such as Shamir's scheme based on polynomial interpolation [Sha79], are linear in the sense that the $\texttt{Recover}(S + \Delta) = \texttt{Recover}(S) + \texttt{Recover}(\Delta)$. Such schemes ensure that only a qualified subset of the entities, by gathering their pieces of information, are able to correctly reconstruct the secret. However, the adversary can alter the reconstructed secret by controlling as few as one entities in that set. In this case, the secret can be seen as being stored in a leakage-proof but temper-prone device, and we can embed AMD codes into the secret-sharing scheme to make it robust, i.e., difficult for an adversary controlling only an unqualified set of entities to alter the reconstructed secret. The embedding is easily done by letting $\texttt{Share}^*(s) = \texttt{Share}(\texttt{Enc}(s))$, $\texttt{Recover}^*(S) = \texttt{Dec}(\texttt{Recover}(S))$, and the improved scheme be $(\texttt{Share}^*, \texttt{Recover}^*)$.

Non-malleable codes As an extension of error-detecting and correcting codes, non-malleable codes relaxes the requirement of identifying errors. Instead, they only guarantee that a tempered codeword either can be decoded to the original message, or has a distribution independent of the original message (i.e., no leakage of any information about it). This feature makes it useful in the resistance of leakage attacks, including various side-channel attacks. In their paper [DPW10], Dziembowski *et al.* constructed a class of non-malleable codes against bit-wise independent temperings using linear error-correcting secret-sharing (LECSS) schemes (see their paper for details) and AMD codes, combined in a very similar way to that of the robust secret-sharing scheme construction.

5.3 New Bounds and Constructions

In this section, we present a new lower bound on the value of ε (Theorem 5.3.2) and $|\mathcal{S}|$ (Proposition 5.3.4), and then give constructions attaining these bounds.

In the following discussions, we will let $\mathcal{S}, \mathcal{G}_1, \mathcal{G}_2$ be finite abelian groups, $\mathcal{G} = \mathcal{S} \times \mathcal{G}_1 \times G_2$, and f be a mapping from $\mathcal{S} \times \mathcal{G}_1$ to $\mathcal{G}_2$. Without loss of generality, we may write the operations in $\mathcal{S}, \mathcal{G}_1, \mathcal{G}_2$ additively. We denote $|\mathcal{S}|$ by m, $|\mathcal{G}_1|$ by n_1, $|\mathcal{G}_2|$ by n_2, and $|\mathcal{G}|$ by $n = mn_1n_2$, where $|X|$ denotes the size of a finite set X.

The `Enc` and `Dec` mappings are defined by f as follows:

$$\texttt{Enc}(s) = (s, x, f(x, s)), \tag{5.3}$$

where x is taken uniformly at random from $\mathcal{G}_1$;

$$\texttt{Dec}(s,x,e) = \begin{cases} s, & \text{if } e = f(x,s), \\ \perp, & \text{otherwise.} \end{cases} \tag{5.4}$$

For each $\alpha = (\Delta_s, \Delta_k, \Delta_y) \in \mathcal{G}$, we define

$$\delta_\alpha = \left|\{\, (s,k) \in \mathcal{S} \times \mathcal{G}_1 \mid f(s+\Delta_s, k+\Delta_k) - f(s,k) = \Delta_y \,\}\right|, \tag{5.5}$$

then the nonlinearity of f can be characterized by differential uniformity as in Definition 5.3.1.

Definition 5.3.1 (Differential uniformity)**.** Let

$$\delta = \max_{\alpha \in (\mathcal{G}\setminus\{\,0\,\})\times\{\,0\,\}\times\mathcal{G}_2} \delta_\alpha,$$

then the *differential uniformity* of f is δ, or f is a *differentially δ-uniform* function.

To obtain our bound on ε, we first prove the following lemma.

Lemma 5.3.1. *Let* `Enc` *and* `Dec` *be the functions in (5.3) and (5.4). Then* (`Enc`, `Dec`) *is a weak systematic (m,n,ε)-AMD code defined in Definition 5.2.3 if and only if*

$$\begin{array}{rcl} \sum_{\substack{\Delta_k \in \mathcal{G}_1\setminus\{\,0\,\} \\ \Delta_y \in \mathcal{G}_2}} \delta_{(0,\Delta_k,\Delta_y)} & \geq & (mn_1)^2 - mn_1 - \varepsilon(m-1)mn_1^2 n_2, \text{ and} \\ \delta_{(\Delta_s,\Delta_k,\Delta_y)} & \leq & \varepsilon mn_1, \text{ if } \Delta_s \neq 0, \end{array} \tag{5.6}$$

where $\delta_{(\Delta_s,\Delta_k,\Delta_y)}$ is defined in (5.5)*.*

Proof. Denote the subgroup $\mathcal{S} \times \mathcal{G}_1 \times \{\,0\,\}$ of $\mathcal{G}$ by $\mathcal{G}'$. For every $\alpha \in \mathcal{G}$, let d_α denote the number of pairs $(x_1, x_2) \in \mathcal{G}' \times \mathcal{G}'$ such that $x_1 - x_2 = \alpha$, then it is easy to verify that $d_\alpha = \delta_\alpha$.

Below we determine the value or bound of δ_α for $\alpha = (\Delta_s, \Delta_k, \Delta_y)$ in three cases.

1. $\Delta_s = \Delta_k = \Delta_y = 0$. In this case it is clear to see that $\delta_\alpha = |\mathcal{S}|\,|\mathcal{G}_1| = mn_1$;
2. $\Delta_s = \Delta_k = 0$ and $\Delta_y \neq 0$. Since f is a function of (s,k), $\delta_\alpha = 0$;

3. $\Delta_s \neq 0$. By the definition of weak (m, n, ε)-AMD code, we have that for every $\alpha = (\Delta_s, \Delta_k, \Delta_y) \in \mathcal{G}$ with $\Delta_s \neq 0$ and s sampled uniformly from $\mathcal{S}$,

$$\begin{aligned}
&\Pr\left[\,\texttt{Dec}(\texttt{Enc}(s) + \alpha) \notin \{\, s, \perp \,\} \mid \alpha\,\right] \\
=&\sum_{s\in\mathcal{S}} \Pr\left[\,\texttt{Dec}(\texttt{Enc}(s) + \alpha) \notin \{\, s, \perp \,\} \mid s, \alpha\,\right] \cdot p_s \\
=&\frac{1}{|\mathcal{S}|}\sum_{s\in\mathcal{S}} \Pr\left[\,\texttt{Dec}(\texttt{Enc}(s) + \alpha) \notin \{\, s, \perp \,\} \mid s, \alpha\,\right] \\
=&\frac{1}{|\mathcal{S}|}\sum_{s\in\mathcal{S}} \frac{|\{\, k \in \mathcal{G}_1 \mid f(k+\Delta_k, s+\Delta_s) - f(k,s) = \Delta_y \,\}|}{|\mathcal{G}_1|} \\
=&\frac{|\{\, (s,k) \in \mathcal{S}\times\mathcal{G}_1 \mid f(k+\Delta_k, s+\Delta_s) - f(k,s) = \Delta_y \,\}|}{|\mathcal{S}|\,|\mathcal{G}_1|} \\
=&\frac{\delta_\alpha}{|\mathcal{S}|\,|\mathcal{G}_1|} \leq \varepsilon,
\end{aligned}$$

which gives $\delta_\alpha \leq \varepsilon m n_1$.

Since the sum of δ_α's is exactly $|\mathcal{G}' \times \mathcal{G}'|$, and $|\mathcal{G}'| = mn_1$,

$$(mn_1)^2 = \sum_{\alpha\in\mathcal{G}} \delta_\alpha.$$

Substituting the values of δ_α into the above equation, we obtain the following inequality:

$$(mn_1)^2 \leq n_1(m-1)n_2 \cdot \varepsilon n_1 m + mn_1 + \sum_{\substack{\Delta_k\in\mathcal{G}_1\setminus\{\,0\,\}\\ \Delta_y\in\mathcal{G}_2}} \delta_{(0,\Delta_k,\Delta_y)}$$

Rearranging its terms gives us the inequality (5.6).

Conversely, let f be a function satisfying the property in (5.6), we need to show that $(\texttt{Enc}, \texttt{Dec})$ is an (m, n, ε)-AMD code. In particular, we need to show that for every $\alpha = (\Delta_s, \Delta_k, \Delta_y) \in \mathcal{G}$ and s sampled uniformly from $\mathcal{S}$,

$$\Pr\left[\,\texttt{Dec}(\texttt{Enc}(s) + \alpha) \notin \{\, s, \perp \,\} \mid \alpha\,\right] \leq \varepsilon.$$

We discuss two cases of α:

1. $\Delta_s \neq 0$: Similar to Case 3 above, we have

$$\begin{aligned}
&\Pr[\,\texttt{Dec}(\texttt{Enc}(s)+\alpha) \notin \{\, s, \perp \,\} \mid \alpha\,] \\
=&\frac{1}{size\mathcal{S}} \sum_{s\in\mathcal{S}} \Pr[\,\texttt{Dec}(\texttt{Enc}(s)+\alpha) \mid s, \alpha\,] \\
=&\frac{1}{|\mathcal{S}|} \sum_{s\in\mathcal{S}} \frac{|\{\, (s,k) \in \mathcal{S}\times\mathcal{G}_1 \mid f(k+\Delta_k, s+\Delta_s) - f(k,s) = \Delta_y \,\}|}{|\mathcal{S}|\,|\mathcal{G}_1|} \\
=&\frac{\delta_\alpha}{|\mathcal{S}|\,|\mathcal{G}_1|} \leq \frac{\varepsilon m n_1}{m n_1} = \varepsilon;
\end{aligned}$$

2. $\Delta_s = 0$: According to the definition of $\texttt{Dec}$, $\texttt{Dec}(\texttt{Enc}(s)+\alpha)$ always yields s or $\perp$ in this case. Therefore the probability

$$\Pr[\,\texttt{Dec}(\texttt{Enc}(s)+\alpha) \notin \{\, s, \perp \,\} \mid \alpha\,] = 0.$$

We complete the proof. □

Lemma 5.3.1 provides a method to prove that $(\texttt{Enc}, \texttt{Dec})$ is a weak systematic (m, n, ε)-AMD code.

Theorem 5.3.2. *Let f be a function from $\mathcal{S}\times\mathcal{G}_1$ to $\mathcal{G}_2$, and $(\texttt{Enc}, \texttt{Dec})$ be the systematic AMD code derived from f. Then*

$$\varepsilon \geq \max\left\{ \frac{(mn_1)^2 - mn_1 - \sum_{\Delta_k\in\mathcal{G}_1^*, \Delta_y\in\mathcal{G}_2} \delta_{(0,\Delta_k,\Delta_y)}}{(m-1)mn_1^2 n_2}, \max_{\substack{(\Delta_s,\Delta_k,\Delta_y)\\ \Delta_s\neq 0}} \frac{\delta_{(\Delta_s,\Delta_k,\Delta_y)}}{mn_1} \right\}.$$

Particularly, assume the differential uniformity of f is d. Then

$$\varepsilon \geq \frac{(mn_1)^2 - mn_1 - d(n_1-1)n_2}{(m-1)mn_1^2 n_2},$$

Proof. By Lemma 5.3.1, we have both

$$\sum_{\substack{\Delta_k\in\mathcal{G}_1\setminus\{\,0\,\}\\ \Delta_y\in\mathcal{G}_2}} \delta_{(0,\Delta_k,\Delta_y)} \geq (mn_1)^2 - mn_1 - \varepsilon(m-1)mn_1^2 n_2 \tag{5.7}$$

and

$$\delta_{(\Delta_s,\Delta_k,\Delta_y)} \leq \varepsilon m n_1, \text{ for } \Delta_s \neq 0. \tag{5.8}$$

From (5.7), we have

$$\varepsilon \geq \frac{\sum_{\Delta_k \in \mathcal{G}_1^*, \Delta_y \in \mathcal{G}_2} \delta_{(0,\Delta_k,\Delta_y)} - (mn_1)^2 + mn_1}{(m-1)mn_1^2 n_2}.$$

From (5.8) we have

$$\varepsilon \geq \max_{\substack{(\Delta_s,\Delta_k,\Delta_y)\\ \Delta_s \neq 0}} \frac{\delta_{(\Delta_s,\Delta_k,\Delta_y)}}{mn_1}.$$

If the differential uniformity of f is d, then $\delta_\alpha \leq d$ for every $\alpha = (\Delta_s, \Delta_k, \Delta_y)$ where either $\Delta_s \neq 0$ or $\Delta_k \neq 0$, therefore

$$\varepsilon \geq \frac{(mn_1)^2 - mn_1 - \sum\limits_{\Delta_k \in \mathcal{G}_1^*, \Delta_y \in \mathcal{G}_2} d}{(m-1)mn_1^2 n_2} = \frac{(mn_1)^2 - mn_1 - d(n_1-1)n_2}{(m-1)mn_1^2 n_2}.$$

□

Corollary 5.3.3. *Let $\mathcal{S}, \mathcal{G}_1$ and $\mathcal{G}_2$ be abelian groups with $|\mathcal{S}| = m, |\mathcal{G}_1| = n_1$ and $|\mathcal{G}_2| = n_2$. Let f be a function $f : \mathcal{G}_1 \times \mathcal{S} \to \mathcal{G}_2$. If f generates a weak systematic (m, n, ε)-AMD code as in Definition 5.2.3 and Definition 5.2.2, then a lower bound for the maximum differential uniformity of $f_s(k) \triangleq f(k, s)$ is*

$$\frac{mn_1^2 - n_1 - \varepsilon(m-1)n_1^2 n_2}{(n_1-1)n_2}.$$

Proof. Let d_s be the differential uniformity of $f_s(k)$, then

$$\begin{aligned} \sum_{\substack{s \in \mathcal{S}\\ \Delta_k \in \mathcal{G}_1 \setminus \{0\}\\ \Delta_y \in \mathcal{G}_2}} d_s &\geq \sum_{\substack{\Delta_k \in \mathcal{G}_1 \setminus \{0\}\\ \Delta_y \in \mathcal{G}_2}} \delta_{(0,\Delta_k,\Delta_y)} \\ &\geq (mn_1)^2 - mn_1 - \varepsilon m(m-1)n_1^2 n_2, \end{aligned}$$

according to Lemma 5.3.1. Therefore

$$\max_{s \in \mathcal{S}} \{ d_s \} \geq \frac{(mn_1)^2 - mn_1 - \varepsilon m(m-1)n_1^2 n_2}{m(n_1-1)n_2} = \frac{mn_1^2 - n_1 - \varepsilon(m-1)n_1^2 n_2}{(n_1-1)n_2}.$$

□

In Proposition 5.3.4, we make a slight improvement to Cramer *et al.*'s bound on general weak AMD codes (Theorem 5.2.2) by taking into consideration $n_1 = |\mathcal{G}_1|$.

Proposition 5.3.4. *Let $\mathcal{S}, \mathcal{G}_1$ and $\mathcal{G}_2$ be abelian groups with $|\mathcal{S}| = m, |\mathcal{G}_1| = n_1$ and $|\mathcal{G}_2| = n_2$. Let $(\mathtt{Enc}, \mathtt{Dec})$ be a weak systematic (m, n, ε)-AMD code where $n = mn_1n_2$, and encoding map $\mathtt{Enc}$ is from $\mathcal{S}$ to $\mathcal{S} \times \mathcal{G}_1 \times \mathcal{G}_2 \triangleq \mathcal{G}$ and decoding map $\mathtt{Dec}$ is from $\mathcal{G}$ to $\mathcal{S} \cup \{\perp\}$, respectively. Then $(\mathtt{Enc}, \mathtt{Dec})$ satisfies*

$$n \geq \frac{(m-1)n_1}{\varepsilon} + 1.$$

Proof. Let p_δ denote the probability that δ is chosen by the adversary. Since probability (5.2) is at most ε for every δ, the probability

$$\begin{aligned} &\Pr\left[\mathtt{Dec}(\mathtt{Enc}(s) + \delta) \notin \{s, \perp\}\right] && (5.9)\\ =&\sum_{\delta \in \mathcal{G}} \Pr\left[\mathtt{Dec}(\mathtt{Enc}(s) + \delta) \notin \{s, \perp\} \mid \delta\right] \cdot p_\delta \\ \leq&\sum_{\delta \in \mathcal{G}} \varepsilon \cdot p_\delta = \varepsilon. && (5.10)\end{aligned}$$

Now suppose δ is uniformly sampled from $\mathcal{G}$. Let $\mathtt{Dec}^{-1}(s)$ denote the set $\{g \in \mathcal{G} \mid \mathtt{Dec}(g) = s\}$. Since s and δ are independently chosen, for any s fixed,

$$\begin{aligned} &\Pr\left[\mathtt{Dec}(\mathtt{Enc}(s) + \delta) \notin \{s, \perp\} \mid s\right] \\ =&\frac{\left|\bigcup_{s' \in \mathcal{S} \setminus \{s\}} \mathtt{Dec}^{-1}(s')\right|}{|\mathcal{G}| - 1} \\ =&\frac{\sum_{s' \in \mathcal{S} \setminus \{s\}} |\mathtt{Dec}^{-1}(s')|}{|\mathcal{G}| - 1},\end{aligned}$$

therefore probability (5.9) can be computed in a second way as

$$\begin{aligned}(5.9) &= \frac{1}{|\mathcal{S}|} \sum_{s \in \mathcal{S}} \Pr\left[\mathtt{Dec}(\mathtt{Enc}(s) + \delta) \notin \{s, \perp\} \mid s\right] \\ &= \frac{1}{size \mathcal{S}} \cdot \frac{(|\mathcal{S}| - 1) \sum_{s \in \mathcal{S}} |\mathtt{Dec}^{-1}(s)|}{|\mathcal{G}| - 1} \\ &= \frac{|\mathcal{S}| - 1}{|\mathcal{G}| - 1} \cdot \frac{\sum_{s \in \mathcal{S}} |\mathcal{G}_1|}{|\mathcal{S}|} \\ &= \frac{|\mathcal{S}| - 1}{|\mathcal{G}| - 1} \cdot |\mathcal{G}_1| = \frac{m-1}{n-1} \cdot n_1. && (5.11)\end{aligned}$$

Combining (5.10) and (5.11) gives us

$$\varepsilon \geq \frac{(m-1)n_1}{n-1},$$

and hence

$$n \geq \frac{(m-1)n_1}{\varepsilon} + 1.$$

□

Below we give two constructions to show that our bound is tight (i.e., can be achieved). Also note that our bound can be a guidance in designing optimal and near-optimal AMD codes — AMD codes derived from highly nonlinear functions often attain or get very close to our bound. Combined with the abundance of highly nonlinear functions, it also means that there can be a large number of good AMD code constructions.

Example 5.3.1. Let $\mathcal{S} = \mathbb{F}_{q^k}, \mathcal{G}_1 = \mathcal{G}_2 = \mathbb{F}_q$. Let f be the function

$$\begin{aligned} f : \mathbb{F}_{q^k} \times \mathbb{F}_q &\to \mathbb{F}_q \\ (s,k) &\mapsto k\,\mathrm{Tr}(s). \end{aligned}$$

Then f generates a weak systematic $(q^k, q^{k+2}, 1/q)$-AMD code. (To see it is indeed a weak AMD code, consider the case $s = 0$.) Also note that f has a differential uniformity of q^k. It can be easily verified that this AMD code attains the lower bound given by Theorem 5.3.2. In contrast, the improved bound (Proposition 5.3.4)

$$\varepsilon \geq \frac{q(q^k-1)}{q^{k+2}-1}$$

can only be asymptotically attained.

Example 5.3.2. Let $\mathcal{S} = \mathbb{F}_{2^k}$, $\mathcal{G}_1 = \mathbb{F}_2$, and $\mathcal{G}_2 = \mathbb{F}_{2^{k+1}}$. Let f be the function

$$\begin{aligned} f : \mathbb{F}_{2^{k+1}} &\to \mathbb{F}_{2^{k+1}} \\ x &\mapsto x^3. \end{aligned}$$

If we view $x \in \mathbb{F}_{2^{k+1}}$ as $(s,k) \in \mathbb{F}_{2^k} \times \mathbb{F}_2$ (taking binary fields $\mathbb{F}_{2^{k+1}}$ and $\mathbb{F}_{2^k}$ as vector space over $\mathbb{F}_2$), then f generates a weak systematic $(2^k, 2^{2k+2}, 1/2^{k+1})$-AMD code. This code asymptotically attains our bound. Note that f is an almost perfect nonlinear (APN) function on $\mathbb{F}_{2^{k+1}}$.

Chapter 6

Conclusion and Future Work

Through the development of code-based cryptography, we can see the huge potential of applying codes to the design of encryption schemes and other security protocols. Dispite the disadvantage of large key sizes, code-based encryption schemes enjoy good security and simple implementations. Though with the current parameters, McEliece cannot compete with RSA in performance until we reach 256 bits of security, RSA is asymptotically slower than McEliece, while McEliece still has plenty of space for optimiazation. Moreover, all traditional public-key cryptosystems based on factoring, discrete logarithm, and elliptic curve discrete logarithm all suffer from quantum attacks. In contrast, code-based cryptosystems are resistant to currently known quantum attacks.

Apart from the known benefits of quantum attack resistance, recent studies on LWE problems and FHE designs based on them further reveal the versatility of codes and their connection with the more thoroughly studied lattice problems. AMD codes also show the possibility to extend error-correcting codes to obtain other useful cryptographic primitives.

However, error-correcting codes are initially designed to correct random errors. Early coding theory research focused on efficiency and were largely restricted on small codes. On the contrary, for security reasons, code-based security solutions require the use of large codes, which has not yet been fully explored. The selection of codes is another important problem in the design of code-based cryptosystems. The possible impact of quantum computers also needs further examination. For example:

1. For McEliece cryptosystem, it remains a question whether there are choices other than Goppa codes for a secure design; even for those based on Goppa code, we cannot guarantee that a more efficient attack that recovers structure from the garbled generator matrix does not exist;

2. For FHE schemes, random codes are used in the LWE-based BGV scheme, but we still need to carefully tune the many parameters to achieve good security, homomorphic evaluation capability, and performance;
3. For quantum computers, we may investigate whether the algorithms of encryption schemes can be sped up in order to resist more efficient attacks.

Another demand of code-based cryptography lies in implementation. Unlike traditional public-key cryptosystems such as RSA, code-based cryptosystems have a lot of variants but relatively few available libraries. As for FHE, currently HElib is the only implementation available to the public. With the ongoing research on FHE schemes, a practical implementation may eventually appear.

References

[ACPS09] Benny Applebaum, David Cash, Chris Peikert, and Amit Sahai. Fast cryptographic primitives and circular-secure encryption based on hard learning problems. In *Advances in Cryptology-CRYPTO 2009*, pages 595–618. Springer, 2009.

[BBBV97] Charles H Bennett, Ethan Bernstein, Gilles Brassard, and Umesh Vazirani. Strengths and weaknesses of quantum computing. *SIAM journal on Computing*, 26(5):1510–1523, 1997.

[BCvD05] Dave Bacon, Andrew M Childs, and Wim van Dam. From optimal measurement to efficient quantum algorithms for the hidden subgroup problem over semidirect product groups. In *Foundations of Computer Science, 2005. FOCS 2005. 46th Annual IEEE Symposium on*, pages 469–478. IEEE, 2005.

[Ber73] Elwyn Berlekamp. Goppa codes. *Information Theory, IEEE Transactions on*, 19(5):590–592, 1973.

[BGV12] Zvika Brakerski, Craig Gentry, and Vinod Vaikuntanathan. (leveled) fully homomorphic encryption without bootstrapping. In *Proceedings of the 3rd Innovations in Theoretical Computer Science Conference*, pages 309–325. ACM, 2012.

[BJMM12] Anja Becker, Antoine Joux, Alexander May, and Alexander Meurer. Decoding random binary linear codes in 2 n/20: how 1+1=0 improves information set decoding. In *Advances in Cryptology–EUROCRYPT 2012*, pages 520–536. Springer, 2012.

[BL11] Andrej Bogdanov and Chin Ho Lee. Homomorphic encryption from codes. *arXiv preprint arXiv:1111.4301*, 2011.

[BLP08] Daniel J Bernstein, Tanja Lange, and Christiane Peters. Attacking and defending the mceliece cryptosystem. In *Post-Quantum Cryptography*, pages 31–46. Springer, 2008.

[BLP$^+$13] Zvika Brakerski, Adeline Langlois, Chris Peikert, Oded Regev, and Damien Stehlé. Classical hardness of learning with errors. In *Proceedings of the forty-fifth annual ACM symposium on Theory of computing*, pages 575–584. ACM, 2013.

[BMVT78] Elwyn R Berlekamp, Robert J McEliece, and Henk CA Van Tilborg. On the inherent intractability of certain coding problems. *IEEE Transactions on Information Theory*, 24(3):384–386, 1978.

[Bra13] Zvika Brakerski. When homomorphism becomes a liability. In *Theory of Cryptography*, pages 143–161. Springer, 2013.

[BV93] Ethan Bernstein and Umesh Vazirani. Quantum complexity theory. In *Proceedings of the twenty-fifth annual ACM symposium on Theory of computing*, pages 11–20. ACM, 1993.

[BV14] Zvika Brakerski and Vinod Vaikuntanathan. Efficient fully homomorphic encryption from (standard) lwe. *SIAM Journal on Computing*, 43(2):831–871, 2014.

[CDF$^+$08] Ronald Cramer, Yevgeniy Dodis, Serge Fehr, Carles Padró, and Daniel Wichs. Detection of algebraic manipulation with applications to robust secret sharing and fuzzy extractors. In *Advances in Cryptology–EUROCRYPT 2008*, pages 471–488. Springer, 2008.

[CFP13] Ronald Cramer, Serge Fehr, and Carles Padró. Algebraic manipulation detection codes. *Science China Mathematics*, 56(7):1349–1358, 2013.

[CGGU$^+$13] Alain Couvreur, Philippe Gaborit, Valérie Gauthier-Umaña, Ayoub Otmani, and Jean-Pierre Tillich. Distinguisher-based attacks on public-key cryptosystems using reed–solomon codes. *Designs, Codes and Cryptography*, pages 1–26, 2013.

[Cha95] Florent Chabaud. On the security of some cryptosystems based on error-correcting codes. In *Advances in CryptologyEUROCRYPT'94*, pages 131–139. Springer, 1995.

[Deu85] David Deutsch. Quantum theory, the church-turing principle and the universal quantum computer. *Proceedings of the Royal Society of London. A. Mathematical and Physical Sciences*, 400(1818):97–117, 1985.

[DMR11] Hang Dinh, Cristopher Moore, and Alexander Russell. Mceliece and niederreiter cryptosystems that resist quantum fourier sampling attacks. In *Advances in Cryptology–Crypto 2011*, pages 761–779. Springer, 2011.

[DPW10] Stefan Dziembowski, Krzysztof Pietrzak, and Daniel Wichs. Non-malleable codes. In *ICS*, pages 434–452, 2010.

[Fey82] Richard P Feynman. Simulating physics with computers. *International journal of theoretical physics*, 21(6):467–488, 1982.

[FOPT10] Jean-Charles Faugere, Ayoub Otmani, Ludovic Perret, and Jean-Pierre Tillich. Algebraic cryptanalysis of mceliece variants with compact keys. In *Advances in Cryptology–EUROCRYPT 2010*, pages 279–298. Springer, 2010.

[Gen09] Craig Gentry. Fully homomorphic encryption using ideal lattices. In *STOC*, volume 9, pages 169–178, 2009.

[GH11a] Craig Gentry and Shai Halevi. Fully homomorphic encryption without squashing using depth-3 arithmetic circuits. In *Foundations of Computer Science (FOCS), 2011 IEEE 52nd Annual Symposium on*, pages 107–109. IEEE, 2011.

[GH11b] Craig Gentry and Shai Halevi. Implementing gentrys fully-homomorphic encryption scheme. In *Advances in Cryptology–EUROCRYPT 2011*, pages 129–148. Springer, 2011.

[GHS12] Craig Gentry, Shai Halevi, and Nigel P Smart. Homomorphic evaluation of the aes circuit. Cryptology ePrint Archive, Report 2012/099, 2012. `http://eprint.iacr.org/`.

[GKP$^+$13] Shafi Goldwasser, Yael Tauman Kalai, Raluca Ada Popa, Vinod Vaikuntanathan, and Nickolai Zeldovich. How to run turing machines on encrypted data. In *Advances in Cryptology–CRYPTO 2013*, pages 536–553. Springer, 2013.

[Gop70] Valerii Denisovich Goppa. A new class of linear correcting codes. *Problemy Peredachi Informatsii*, 6(3):24–30, 1970.

[Gro96] Lov K Grover. A fast quantum mechanical algorithm for database search. In *Proceedings of the twenty-eighth annual ACM symposium on Theory of computing*, pages 212–219. ACM, 1996.

[GRR11] Dmitry Gavinsky, Martin Roetteler, and Jérémie Roland. Quantum algorithm for the boolean hidden shift problem. In *Computing and Combinatorics*, pages 158–167. Springer, 2011.

[GSW13] Craig Gentry, Amit Sahai, and Brent Waters. Homomorphic encryption from learning with errors: Conceptually-simpler, asymptotically-faster, attribute-based. In *Advances in Cryptology–CRYPTO 2013*, pages 75–92. Springer, 2013.

[HS14] Shai Halevi and Victor Shoup. Algorithms in helib. Cryptology ePrint Archive, Report 2014/106, 2014. http://eprint.iacr.org/.

[Kit95] A Yu Kitaev. Quantum measurements and the abelian stabilizer problem. *arXiv preprint quant-ph/9511026*, 1995.

[KLM07] Phillip Kaye, Raymond Laflamme, and Michele Mosca. An introduction to quantum computing, 2007.

[KLYC13] Jinsu Kim, Moon Sung Lee, Aaram Yun, and Jung Hee Cheon. Crt-based fully homomorphic encryption over the integers. Cryptology ePrint Archive, Report 2013/057, 2013. http://eprint.iacr.org/.

[Knu97] Donald E Knuth. *Art of Computer Programming, Volume 1: Fundamental Algorithms, 3rd. Ed., The*. Addison-Wesley Professional, 1997.

[KS06] Jonathan Katz and Ji Sun Shin. Parallel and concurrent security of the hb and hb+ protocols. In *Advances in Cryptology-EUROCRYPT 2006*, pages 73–87. Springer, 2006.

[KT04] Mark Karpovsky and Alexander Taubin. New class of nonlinear systematic error detecting codes. *Information Theory, IEEE Transactions on*, 50(8):1818–1819, 2004.

[Kup] Greg Kuperberg. Complexity zoology: Active inclusion diagram. https://www.math.ucdavis.edu/~greg/zoology/diagram.xml. Accessed: August 12, 2014.

[LLJMP93] Arjen K Lenstra, Hendrik W Lenstra Jr, Mark S Manasse, and John M Pollard. *The number field sieve*. Springer, 1993.

[LMSV10] Jake Loftus, Alexander May, Nigel P Smart, and Frederik Vercauteren. On cca-secure fully homomorphic encryption. *IACR Cryptology ePrint Archive*, 2010:560, 2010.

[LMSV12] Jake Loftus, Alexander May, Nigel P Smart, and Frederik Vercauteren. On cca-secure somewhat homomorphic encryption. In *Selected Areas in Cryptography*, pages 55–72. Springer, 2012.

[McE78] Robert J McEliece. A public-key cryptosystem based on algebraic coding theory. *DSN progress report*, 42(44):114–116, 1978.

[Mil76] Gary L Miller. Riemann's hypothesis and tests for primality. *Journal of computer and system sciences*, 13(3):300–317, 1976.

[Mor06] Kent Morrison. Integer sequences and matrices over finite fields. *J. Integer Seq*, 9(2):06–2, 2006.

[NC10] Michael A Nielsen and Isaac L Chuang. *Quantum computation and quantum information*. Cambridge university press, 2010.

[Nie86] Harald Niederreiter. Knapsack-type cryptosystems and algebraic coding theory. *Problems of Control and Information Theory — Problemy Upravleniya i Teorii Informatsii*, 15(2):159–166, 1986.

[Pei09] Chris Peikert. Public-key cryptosystems from the worst-case shortest vector problem. In *Proceedings of the forty-first annual ACM symposium on Theory of computing*, pages 333–342. ACM, 2009.

[Pra62] Eugene Prange. The use of information sets in decoding cyclic codes. *Information Theory, IRE Transactions on*, 8(5):5–9, 1962.

[RAD78] Ronald L Rivest, Len Adleman, and Michael L Dertouzos. On data banks and privacy homomorphisms. *Foundations of secure computation*, 4(11):169–180, 1978.

[Reg04a] Oded Regev. New lattice-based cryptographic constructions. *Journal of the ACM (JACM)*, 51(6):899–942, 2004.

[Reg04b] Oded Regev. Quantum computation and lattice problems. *SIAM Journal on Computing*, 33(3):738–760, 2004.

[Reg09] Oded Regev. On lattices, learning with errors, random linear codes, and cryptography. *Journal of the ACM (JACM)*, 56(6):34, 2009.

[Röt10] Martin Rötteler. Quantum algorithms for highly non-linear boolean functions. In *Proceedings of the Twenty-First Annual ACM-SIAM Symposium on Discrete Algorithms*, pages 448–457. Society for Industrial and Applied Mathematics, 2010.

[SG01] Rocco A Servedio and Steven J Gortler. Quantum versus classical learnability. In *Computational Complexity, 16th Annual IEEE Conference on, 2001.*, pages 138–148. IEEE, 2001.

[Sha79] Adi Shamir. How to share a secret. *Communications of the ACM*, 22(11):612–613, 1979.

[Sho94] Peter W Shor. Algorithms for quantum computation: discrete logarithms and factoring. In *Foundations of Computer Science, 1994 Proceedings., 35th Annual Symposium on*, pages 124–134. IEEE, 1994.

[SKHN75] Yasuo Sugiyama, Masao Kasahara, Shigeichi Hirasawa, and Toshihiko Namekawa. A method for solving key equation for decoding goppa codes. *Information and Control*, 27(1):87–99, 1975.

[SP800-57] Elaine Barker, William Barker, William Burr, William Polk, and Miles Smid. Recommendation for key management – part 1: General (revision 3). July 2012.

[Ste89] Jacques Stern. A method for finding codewords of small weight. In *Coding theory and applications*, pages 106–113. Springer, 1989.

[UL10] V Gauthier Umana and Gregor Leander. Practical key recovery attacks on two mceliece variants. In *International Conference on Symbolic Computation and Cryptography–SCC*, volume 2010, page 62, 2010.

[vDGHV10] Marten van Dijk, Craig Gentry, Shai Halevi, and Vinod Vaikuntanathan. Fully homomorphic encryption over the integers. In *Advances in Cryptology–EUROCRYPT 2010*, pages 24–43. Springer, 2010.

[Wei13] Brandon Weir. Homomorphic encryption. Master's thesis, University of Waterloo, 2013.

[WK11] Zhen Wang and Mark Karpovsky. Algebraic manipulation detection codes and their applications for design of secure cryptographic devices. In *On-Line Testing Symposium (IOLTS), 2011 IEEE 17th International*, pages 234–239. IEEE, 2011.

[Yao93] Andrew Chi-Chih Yao. Quantum circuit complexity. In *Foundations of Computer Science, 1993. Proceedings., 34th Annual Symposium on*, pages 352–361. IEEE, 1993.

[ZPS12] Zhenfei Zhang, Thomas Plantard, and Willy Susilo. On the cca-1 security of somewhat homomorphic encryption over the integers. In *Information Security Practice and Experience*, pages 353–368. Springer, 2012.