POPPING BUBBLES:
CRYPTANALYSIS OF HOMOMORPHIC ENCRYPTION

by

Corre Steele

A Master's Project

Submitted to the Faculty

of the

WORCESTER POLYTECHNIC INSTITUTE

in partial fulfillment of the requirements for the

Degree of Master of Science

in

Industrial Mathematics

by

April 2016

APPROVED:

Dr. William Martin, Major Adviser

Dr. Luca Capogna, Department Head

Abstract

Imagine an encryption scheme where it is possible to add and multiply numbers without any knowledge of the numbers. Instead one could manipulate encryptions of the numbers and then the decryption of the result would give the result of the arithmetic on the original numbers. Encryption algorithms with this property are called homomorphic and have various applications in cloud computing. Homomorphic encryption schemes exist but are generally so inefficient that they are not practical. This report introduces a toy cryptosystem called Bubbles: a somewhat homomorphic encryption scheme created by Professor Martin and Professor Sunar at Worcester Polytechnic Institute. We will show that the original scheme is insecure and may be efficiently "popped". We will then examine two variations of the scheme that introduce noise to increase security and show that Bubbles is still vulnerable except when parameters are carefully chosen. However these safe parameter choices make Bubbles more inefficient than other recent homomorphic schemes.

Contents

List of Algorithms

List of Figures

List of Tables

Chapter 1

Introduction

Cloud computing is a powerful new technology that empowers many people and companies to utilize computation power and storage without the expense of buying and maintaining shelves of servers. Large companies such as Netflix, Pinterest and Apple, along with many smaller companies and individuals, all use cloud services to outsource hardware [1]. Cloud services are currently a multi-billion dollar industry and the International Data Corporation estimates that over $127 billion will be spent on cloud services in 2018 [2].

Cloud services allow companies to not worry about housing, maintaining and updating racks of servers. However this great convenience comes at a cost. Introducing data to the Internet always produces security concerns. Now data is sent over the Internet to a cloud provider instead of being stored on site. This makes it significantly easier for an attacker to access the data by either intercepting it as it is being sent over the Internet or by infiltrating the cloud provider.

These concerns can be addressed by encrypting the data before sending it to the cloud provider. As long as the encryption method is secure, the data may be sent, stored and retrieved without a breach. This solution works for data storage but does not allow for data processing. In the vast majority of encryption schemes such as AES or 3DES, data could be encrypted when sent to the cloud but for any processing it must be decrypted, processed and then encrypted again. This process is shown in Table 1.1 for a case where Bob wants to add two numbers p_1 and p_2. These numbers are called plaintext and their encryptions are called ciphertext.

Bob encrypts plaintext	$c_1 = \mathrm{Enc}(p_1)$, $c_2 = \mathrm{Enc}(p_2)$
Bob sends ciphertext to cloud	
Cloud provider decrypts ciphertext	$p_1 = \mathrm{Dec}(c_1)$, $p_2 = \mathrm{Dec}(c_2)$
Cloud provider processes plaintext	$p = p_1 + p_2$
Cloud provider encrypts result	$c = \mathrm{Enc}(p)$
Bob retrieves and decrypts ciphertext	$p = \mathrm{Dec}(c)$

Table 1.1: Process for cloud to process data

For many applications this is acceptable as the cloud provider can be mostly trusted and the data is not overly sensitive. However for very sensitive data, such as medical or financial data, it is not acceptable for the cloud provider to ever have access to the decrypted data. Consider the situation where Bob works at a hospital and wants to process medical data on the cloud. The processing could be as simple as taking statistics of vital signs or processing the raw results of an MRI to produce an image for a doctor to look at. In this case Bob would like there to be a way for a cloud provider to manipulate the encrypted data without ever seeing the actual numbers. This is what homomorphic encryption allows.

In a homomorphic encryption scheme there are efficient operations $\oplus$ and $\otimes$ such that

$$\begin{array}{c} c_1 = \mathrm{Enc}(p_1) \\ c_2 = \mathrm{Enc}(p_2) \end{array} \implies \begin{array}{c} p_1 + p_2 = \mathrm{Dec}(c_1 \oplus c_2) \\ p_1 \cdot p_2 = \mathrm{Dec}(c_1 \otimes c_2). \end{array}$$

The operation $\oplus$ allows the cloud provider to add plaintexts together without ever knowing what they are. If Bob uses homomorphic encryption, he can change the process for a cloud provider to add the two numbers p_1 and p_2 to look like that described in Table 1.2.

Bob encrypts plaintext	$c_1 = \mathrm{Enc}(p_1)$, $c_2 = \mathrm{Enc}(p_2)$
Bob sends ciphertext to cloud	
Cloud provider processes ciphertext	$c = c_1 \oplus c_2$
Bob retrieves and decrypts ciphertext	$p = \mathrm{Dec}(c)$ where $p = p_1 + p_2$

Table 1.2: Process for cloud to process data using homomorphic encryption

The idea for homomorphic encryption schemes was first proposed by Rivest et al. in 1978 [10]. Just a few months earlier the same Rivest along with Shamir and Adleman had introduced the public key cryptosystem RSA. Then in [10] they showed that RSA is multiplicatively homomorphic i.e. there is an operation $\otimes$ but not $\oplus$. Many homomorphic cryptosystems have been proposed since then and a variety of applications have been suggested [7].

Until recently every scheme has only allowed so many additions or multiplications to be performed on the ciphertexts. To talk about how many additions or multiplications a scheme can support we look at arithmetic circuits. An additive circuit of depth d takes 2^d ciphertexts and computes their sum. This is illustrated in Figure 1.1. Similarly, a multiplicative circuit of depth d computes the product of 2^d ciphertexts.

A cryptographic scheme that can correctly decrypt the output of arbitrarily large additive and multiplicative circuits is called *fully homomorphic.* Many proposed homomorphic schemes cannot do this. Most homomorphic schemes get security from adding small amounts of noise. Someone with the secret encryption key can decrypt in spite of the noise as long as it is small enough, but the noise obscures the plaintext to someone without the key. However combining too many ciphertexts in an arithmetic circuit will increase the noise above the threshold where someone with the key can still decrypt correctly. Homomorphic cryptosystems that only work with limited depth additive or multiplicative circuits are called *somewhat homomorphic.*

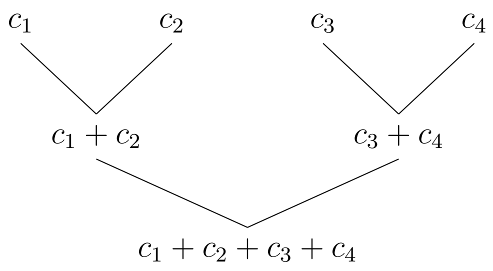

Figure 1.1: Additive circuit of depth 2

Until 2009 all proposed schemes were somewhat homomorphic. Then in 2009 Gentry proposed the first fully homomorphic encryption scheme [6]. While Gentry showed that a fully homomorphic cryptosystem exists, the system he came up with is too inefficient to be practical. Since Gentry's breakthrough paper, multiple fully homomorphic schemes have been proposed and suggestions made on how to make them more practical [7]. However no improvement has made fully homomorphic encryption efficient enough to implement.

Even somewhat homomorphic encryption tends to be impractical. Some systems such as Paillier's scheme and NTRU have been implemented but have yet to see widespread use [9, 3, 7]. There is still a while to go before somewhat homomorphic encryption, never mind fully homomorphic encryption, is practical to be widely implemented. From now on we will refer to partially homomorphic schemes as homomorphic.

In this paper we will look at Bubbles, a toy cryptosystem made by Bill Martin and Berk Sunar at Worcester Polytechnic Institute [unpublished]. Bubbles is a partially homomorphic encryption scheme based on Shamir's secret sharing scheme [8, p. 526]. All knowledge of the basic workings of this cryptosystem was via personal communication. Since we will only be discussing Bubbles in this report, we will refer to partially homomorphic schemes as homomorphic.

In its most basic form Bubbles is easily broken (the bubbles are popped!) but by adding noise it can be made more secure. In Chapter 3, we will introduce the most basic version of Bubbles and show why it is not secure. In Chapter 4, we vary the original scheme to make it more secure by adding a type of noise we call chaff. We will then show how this revision is still easily breakable using the same idea behind the algorithm to break basic Bubbles. In Chapter 5, we introduce a different type of noise which is significantly harder to break. In fact, some parameter choices make the system secure against all algorithms we will introduce. However these parameter choices make Bubbles more inefficient than the most recently proposed homomorphic encryption schemes.

Chapter 2

Mathematical Preliminaries

In this section we introduce some notation, definitions, lemmas and propositions that will be needed later. It is recommended that the reader glace through the next few paragraphs to understand the notation we will use, but treat the rest of the chapter as a reference.

All work will be in $\mathbb{F}_q$, a finite field of order q. We denote the ring of polynomials of a single variable over this field by $\mathbb{F}_q[x]$. The set $\mathbb{F}_q[x]_k$ is a subset of $\mathbb{F}_q[x]$ that contains only polynomials of degree less than k.

It is well-known that any polynomial of degree less than k can be reconstructed from knowledge of any k or more values using Lagrange interpolation [5]. Let $f(x) \in \mathbb{F}_q[x]_k$ and $f(x_i) = y_i$ for all $i = 1, ..., k$. Then Lagrange interpolation gives

$$f(x) = \sum_{i=1}^{k} y_i \prod_{j \neq i} \frac{x - x_j}{x_i - x_j}.$$

To see this note that the polynomial

$$g_i(x) = \prod_{j \neq i} \frac{x - x_j}{x_i - x_j}$$

has degree $k - 1$ and $g_i(x_j) = \delta_{ij}$.

We use $\mathbb{1}$ to denote the all one's vector and 0 to denote the zero matrix. The *Hamming weight* of a vector $\mathbf{v} \in \mathbb{F}_q^n$, denoted $\mathrm{w_H}(\mathbf{v})$, is the number of non-zero entries of $\mathbf{v}$ i.e. $\mathrm{w_H}(\mathbf{v}) = |\{i \mid v_i \neq 0\}|$.

For a set S we write $x \leftarrow_u S$ to denote that x is chosen uniformly at random from the elements in S. We denote the power set of S by $\mathcal{P}(S)$ and the set of all subsets of S of size k by $\mathcal{P}_k(S)$. Frequently we will want a subset of the integers between 1 and n and so we will abbreviate $\mathcal{P}(\{1, ..., n\})$ to $\mathcal{P}(n)$.

For $A \in \mathbb{F}_q^{n \times m}$, $\alpha \in \mathcal{P}(n)$ and $\beta \in \mathcal{P}(m)$ we define A_α (resp. A^β) as the matrix derived from A by deleting all rows (columns) other than those indexed by elements in α (β). Similarly, for $\mathbf{a} \in \mathbb{F}_q^m$ we define $\mathbf{a}_\beta$ to be the vector derived from $\mathbf{a}$ by deleting all entries other than those indexed by elements in β.

Example 2.1. For $\alpha = \{1,4\}$, $\beta = \{3,4\}$, and

$$A = \begin{pmatrix} 8 & 3 & 5 & 2 \\ 3 & 7 & 4 & 5 \\ 4 & 8 & 7 & 2 \\ 6 & 5 & 1 & 9 \\ 2 & 4 & 1 & 4 \end{pmatrix}$$

then

$$A_\alpha = \begin{pmatrix} 8 & 3 & 5 & 2 \\ 6 & 5 & 1 & 9 \end{pmatrix},\ A^\beta = \begin{pmatrix} 5 & 2 \\ 4 & 5 \\ 7 & 2 \\ 1 & 9 \\ 1 & 4 \end{pmatrix},\ A_\alpha^\beta = \begin{pmatrix} 5 & 2 \\ 1 & 9 \end{pmatrix}.$$

In Lemmas 2.2, 2.3 and 2.4 we refer to the matrix

$$V_0 \coloneqq \begin{pmatrix} x_1 & x_2 & \dots & x_n \\ x_1^2 & x_2^2 & & x_n^2 \\ \vdots & & \ddots & \vdots \\ x_1^{k-1} & x_2^{k-1} & \dots & x_n^{k-1} \end{pmatrix}$$

where $k-1 \leq n$ and the field elements $x_i \in \mathbb{F}_q$ are distinct and non-zero.

Lemma 2.2. $\text{rank}(V_0) = k-1$

Proof. If $\text{rank}(V_0) \neq k-1$ then the rows of V_0 are linearly dependent. Therefore there exists $\mathbf{f} = (f_1, ..., f_{k-1}) \in \mathbb{F}_q^{k-1}$ such that $\mathbf{f}V_0 = 0$. This is equivalent to saying that the polynomial $f(x) \coloneqq f_1 x + ... + f_{f-1}x^{k-1} = x(f_1 + ... + f_{k-1}x^{k-2})$ has roots $x_1, ..., x_n$. However since $f(x)$ has degree at most $k-1 \leq n$ and has 0 as a root, it cannot have n distinct non-zero roots. □

Lemma 2.3. *For $A \in \mathbb{F}_q^{m\times(k-1)}$,* $\text{rank}(AV_0) = k-1$ *if and only if* $\text{rank}(A) = k-1$.

Proof. If $\text{rank}(A) \neq k-1$ then $\text{rank}(A) < k-1$ because the rank of A may not exceed the number of columns of A. Since $\text{rank}(AV_0) \leq \text{rank}(A)$ then $\text{rank}(AV_0) < k-1$.

If $\text{rank}(A) = k-1$ then A has $k-1$ linearly independent rows. Consider the submatrix A' of A that consists of $k-1$ linearly independent rows. Then A' is is invertible and so $\text{rank}(A'V_0) = \text{rank}(V_0)$. By Lemma 2.2 $\text{rank}(V_0) = k-1$ so $\text{rank}(A'V_0) = k-1$. Thus $\text{rank}(AV_0) \geq k-1$ because adding on rows to a matrix cannot decrease its rank. Since $\text{rank}(AV_0) \leq \text{rank}(V_0) = k-1$ we have $\text{rank}(AV_0) = k-1$. □

Lemma 2.4. *Let $C \in \mathbb{F}_q^{k\times n}$ be of the form $C = FV_0 + E$. Let $\rho = \{i \leq n \mid \bar{E}^{\{i\}} = 0\}$ and let β be the set of indices of the columns of C that end in 0. If there is a set $\rho' \subset \rho \cap \beta$ with $|\rho'| \geq k-1$ then $\rho \subset \beta$.*

Proof. We will show that colsp(C^ρ) is a subspace of colsp(C^β). Since every column indexed in β ends in 0 this will mean that every column indexed in ρ must also end in 0 and so $\rho \subset \beta$.

Since $E^\rho = 0$ we have $C^\rho = (FV_0)^\rho$. Thus the column space of C^ρ is a subspace of colsp($(FV_0)^\rho$) which is a subspace of colsp(FV_0).

We know that rank($FV_0^{\rho'}$) = rank(F) by Lemma 2.3 since $\rho' \geq k-1$. Lemma 2.3 also tells us that rank(FV_0) = rank(F) and so rank($FV_0^{\rho'}$) = rank(FV_0). Clearly colsp($FV_0^{\rho'}$) is a subspace of colsp(FV_0) and since these spaces have equal dimension colsp(FV_0) = colsp($FV_0^{\rho'}$).

We also know that $FV_0^{\rho'} = (FV_0)^{\rho'} = C^{\rho'}$ and the column space of $C^{\rho'}$ is a subspace of the column space of C^β because $\rho' \subset \beta$. Therefore colsp($FV_0^{\rho'}$) is a subspace of colsp(C^β).

Putting everything together gives

$$\text{colsp}(C^\rho) \subset \text{colsp}(FV_0) = \text{colsp}(FV_0^{\rho'}) \subset \text{colsp}(C^\beta).$$

□

Corollary 2.5. *Let $C \in \mathbb{F}_q^{k\times n}$ be of the form $C = FV_0 + E$. Let $\rho = \{i \leq n \mid \bar{E}^{\{i\}} = 0\}$ and let β be the set of indices of the columns of C that end two 0's. If there is a set $\rho' \subset \rho \cap \beta$ with $|\rho'| \geq k-1$ then $\rho \subset \beta$.*

Proof. Let C' be the matrix C with the last row removed and define F' and E' similarly. Now β is the set of indices of the columns of C' that end in 0 and so the result follows from Lemma 2.4. □

Definition 2.6. Let $\gamma \in \mathcal{P}(n)$ be a subset of the indices of the columns of $C \in \mathbb{F}_q^{m\times n}$. Then to *row reduce C with respect to the columns indexed in γ* we

1. Construct an $n \times n$ permutation matrix P whose first $|\gamma|$ columns have the form $(0, ..., 0, 1, 0, ..., 0)^T$ where 1 is the i^{th} entry for $i \in \gamma$.

2. Apply Gaussian elimination to CP.

3. Multiply the result by P^{-1} on the right.

Lemma 2.7. *If $\bar{C}$ is the result of row reducing C with respect to the columns indexed in γ then there is an invertible matrix M such that $\bar{C} = MC$.*

Proof. Let P be the permutation matrix prescribed in Definition 2.6. There is an invertible matrix M such that MCP is the result of applying Gaussian elimination to CP (cf. [5, chap. 3]). Then $\bar{C} = MCPP^{-1} = MC$. □

Lemma 2.8. *For $A \leftarrow_u \mathbb{F}_q^{m\times n}$ where $m \geq n$*

$$Pr[\text{rank}(A) = n] = \prod_{i=0}^{n-1} 1 - q^{i-m}$$

Proof. Let A_i denote the submatrix of A formed by its first i columns. We proceed by induction.

$$Pr[\text{rank}(A_1) = 1] = \frac{q^m - 1}{q^m}$$

since we must rule out the possibility of the first column being all zero.

For A_i to be full rank, A_{i-1} must be full rank and the i^{th} column of A must not be one of the q^{i-1} linear combinations of the first $i - 1$ columns of A.

$$Pr[\text{rank}(A_i) = i] = Pr[\text{rank}(A_{i-1}) = i - 1] \cdot \frac{q^m - q^{i-1}}{q^m}$$

Thus

$$\begin{aligned} Pr[\text{rank}(A) = n] &= \prod_{i=0}^{n-1} \frac{q^m - q^i}{q^m} \\ &= \prod_{i=0}^{n-1} 1 - q^{i-m}. \end{aligned}$$

□

Chapter 3

Basic Bubbles

To set up an encryption session we choose a finite field $\mathbb{F}_q$ of order q and integer parameters $k \leq n < q$. These are all parameters that we assume an adversary knows or could easily find out. Each ciphertext will encrypt a single field element in a vector of length n. Usually the parameter $k \ll n$ for reasons that will soon be apparent. The key is a vector of distinct nonzero field elements $\mathbf{x} = (x_1, ..., x_n) \in (\mathbb{F}_q \backslash \{0\})^n$ where $x_i \neq x_j$ for $i \neq j$.

To encrypt a plaintext $p \in \mathbb{F}_q$ we generate a random polynomial $f(x) \leftarrow_u \mathbb{F}_q[x]_{k-1}$ and set $\bar{f}(x) = p + xf(x)$. We call $\bar{f}(x)$ the *encrypting polynomial.* Then the encryption of p is

$$\mathrm{Enc}_{\mathbf{x}}(p) = (\bar{f}(x_1), \bar{f}(x_2), ..., \bar{f}(x_n)).$$

Sometimes we will want to specify a specific polynomial g instead of having one generated at random. We denote this by $\mathrm{Enc}_{\mathbf{x},g}(p) = (\bar{g}(x_1), \bar{g}(x_2), ..., \bar{g}(x_n))$ where $\bar{g}(x) = p + xg(x)$ as before. If a polynomial is not specified, i.e. $\mathrm{Enc}_{\mathbf{x}}(p)$, then it is understood one should be selected uniformly at random from $\mathbb{F}_q[x]_{k-1}$.

A ciphertext can be decrypted using Lagrange interpolation to reconstruct $\bar{f}(x)$ and then it is easy to find the plaintext by computing $p = \bar{f}(0)$.

The encryption system is summarized in Algorithm 3.1.

Example 3.1.
Public parameters: $\mathbb{F}_q = \mathbb{Z}_{11}$, $n = 4$ and $k = 3$
Key: $\mathbf{x} = (3, 5, 2, 10)$
Plaintext: $p = 7$
Random polynomial: $f(x) = 4 + 0x \in \mathbb{Z}_{11}[x]_{k-1}$

To encrypt p we set the encrypting polynomial $\bar{f} = p + xf(x) = 7 + 4x$ and so

$$\mathrm{Enc}_{\mathbf{x},f}(7) = (8, 5, 4, 3).$$

To decrypt with Lagrange interpolation we only need to use three x values because

```
Setup:
    1. Public parameters: finite field $\mathbb{F}_q$, integers $k \leq n < q$
    2. Key: $\mathbf{x} \in (\mathbb{F}_q \backslash \{0\})^n$ with $x_i \neq x_j$ for $i \neq j$

Encryption of plaintext $p \in \mathbb{F}_q$:
    1. Generate random polynomial $f(x) \leftarrow_u \mathbb{F}_q[x]_{k-1}$
    2. Set encrypting polynomial $\bar{f}(x) = p + xf(x)$
    3. Ciphertext $\mathbf{c} = \text{Enc}_{\mathbf{x},f}(p) = (\bar{f}(x_1), \bar{f}(x_2), ..., \bar{f}(x_n))$

Decryption of ciphertext $\mathbf{c}$:
    1. Use Lagrange interpolation to reconstruct $\bar{f}$ from $\mathbf{x}$ and $\mathbf{c}$
    2. Calculate $p = \bar{f}(0)$
```

Algorithm 3.1: Basic Bubbles encryption and decryption

$\deg(f) < k = 3$. For a general ciphertext $\mathbf{c}$, f is

$$\begin{aligned} f(x) &= c_1 \cdot \frac{(x-5)(x-2)}{(3-5)(3-2)} + c_2 \cdot \frac{(x-3)(x-2)}{(5-3)(5-2)} + c_3 \cdot \frac{(x-3)(x-5)}{(2-3)(2-5)} \\ &= c_1 \cdot (x^2 + 4x + 10) \cdot 5 + c_2 \cdot (x^2 + 6x + 6) \cdot 2 + c_3 \cdot (x^2 + 3x + 4) \cdot 4 \\ &= x^2(5c_1 + 2c_2 + 4c_3) + x(9c_1 + c_2 + c_3) + (6c_1 + c_2 + 5c_3). \end{aligned}$$

Then $f(0) = 6c_1 + c_2 + 5c_3$. Note we went through substantially more computation than is necessary. We do not need to reconstruct the entire polynomial $f(x)$ since we only need $f(0)$. The alternative computation is

$$\begin{aligned} f(0) &= c_1 \cdot \frac{5 \cdot 2}{(3-5)(3-2)} + c_2 \cdot \frac{3 \cdot 2}{(5-3)(5-2)} + c_3 \cdot \frac{3 \cdot 5}{(2-3)(2-5)} \\ &= 6c_1 + c_2 + 5c_3 \end{aligned}$$

which is the same result. Using this we can decrypt the ciphertext $(8, 5, 4, 3)$.

$$\text{Dec}_{\mathbf{x}}((8, 5, 4, 3)) = 6 \cdot 8 + 5 + 5 \cdot 4 = 7$$

This is the original plaintext p that we encrypted.

Bubbles gets its name because we can visualize the ciphertext as bubbles floating above the x-axis as in Figure 3.1. Anyone with the key knows where the bubbles are "tied down" and can use Lagrange interpolation to reconstruct $f(0)$. Everyone else just sees the bubbles floating. They know the height of each one but not the x coordinate.

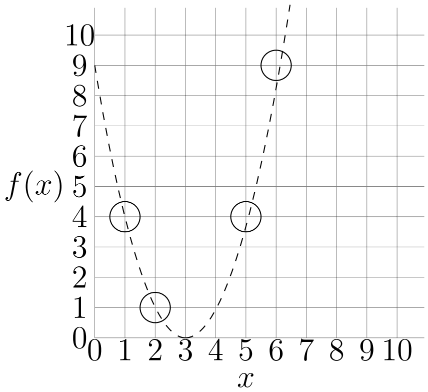

Figure 3.1: Ciphertext as bubbles: $\mathbf{x} = (1, 5, 2, 6)$, $\bar{f}(x) = x^2 - 6x + 9$, $\mathbf{c} = (4, 4, 1, 9)$

3.1 Homomorphic

This system has redundancy since n (the length of ciphertexts and the key) only needs to be equal to k (bound on the degree of polynomials) in order to decrypt. However we typically set $n \gg k$. This is so that the system is multiplicatively homomorphic.

Recall that a system is homomorphic if

$$\begin{matrix} c_1 = \text{Enc}_{\mathbf{x}}(p_1) \\ c_2 = \text{Enc}_{\mathbf{x}}(p_2) \end{matrix} \implies \begin{matrix} p_1 + p_2 = \text{Dec}_{\mathbf{x}}(c_1 \oplus c_2) \\ p_1 \cdot p_2 = \text{Dec}_{\mathbf{x}}(c_1 \otimes c_2) \end{matrix}$$

for some efficient operations $\oplus$ and $\otimes$. In this case $\oplus$ is entrywise addition and $\otimes$ is entrywise multiplication i.e. $(1, 2, 3, 4) \oplus (2, 1, 0, 1) = (3, 3, 3, 5)$ and $(1, 2, 3, 4) \otimes (2, 1, 0, 1) = (2, 2, 0, 1)$. Proposition 3.2 says that if we add ciphertexts together we get an encryption of the sum of their plaintexts and that $c_1 \otimes c_2$ is an encryption of the product of their plaintexts

Proposition 3.2. *If* $\mathbf{c_1} = \text{Enc}_{\mathbf{x},f_1}(p_1)$ *and* $\mathbf{c_2} = \text{Enc}_{\mathbf{x},f_2}(p_2)$ *then* $\mathbf{c_1} \oplus \mathbf{c_2} = \text{Enc}_{\mathbf{x},f_1+f_2}(p_1 + p_2)$ *and* $\mathbf{c_1} \otimes \mathbf{c_2} = \text{Enc}_{\mathbf{x},f_1 \cdot f_2}(p_1 p_2)$.

Proof.

$$\mathbf{c_1} = \text{Enc}_{\mathbf{x},f_1}(p_1) = (\bar{f_1}(x_1), \bar{f_1}(x_2), ..., \bar{f_1}(x_n))$$
$$\mathbf{c_2} = \text{Enc}_{\mathbf{x},f_2}(p_2) = (\bar{f_2}(x_1), \bar{f_2}(x_2), ..., \bar{f_2}(x_n))$$

$$\begin{aligned} \mathbf{c_1} \oplus \mathbf{c_2} &= (\bar{f_1}(x_1) + \bar{f_2}(x_1), \bar{f_1}(x_2) + \bar{f_2}(x_2), ..., \bar{f_1}(x_n) + \bar{f_2}(x_n)) \\ &= (\overline{f_1 + f_2}(x_1), \overline{f_1 + f_2}(x_2), ..., \overline{f_1 + f_2}(x_n)) \\ &= \text{Enc}_{\mathbf{x},f_1+f_2}(p_1 + p_2) \end{aligned}$$

because

$$\begin{aligned}\bar{f}_1(x) + \bar{f}_2(x) &= p_1 + xf_1(x) + p_2 + xf_2(x) \\ &= p_1 + p_2 + x(f_1(x) + f_2(x)) \\ &= \overline{f_1 + f_2}(x)\end{aligned}$$

The proof for multiplication is similar. □

We can decrypt $c_1 \oplus c_2$ and $c_1 \otimes c_2$ by reconstructing $f_1 + f_2$ and $f_1 \cdot f_2$ respectively and evaluating at 0. This can be done without a hitch for $f_1 + f_2$ since $\deg(f_1 + f_2) < k \leq n$. However $\deg(f_1 \cdot f_2) = \deg(f_1) + \deg(f_2)$ so if $\deg(f_1) = \deg(f_2) = k$ then $\deg(f_1 \cdot f_2) = 2k$ which can only be recovered if $n > 2k$. Thus the size of n limits the number of multiplications that can be performed on the ciphertext.

Recall that in a multiplicative circuit of depth d, 2^d ciphertexts are multiplied together. This means that the encrypting polynomial of the product ciphertext is a product of 2^d polynomials. Figure 3.2 shows a circuit of depth 2 that shows the degree of the encrypting polynomial of the ciphertext at each stage.

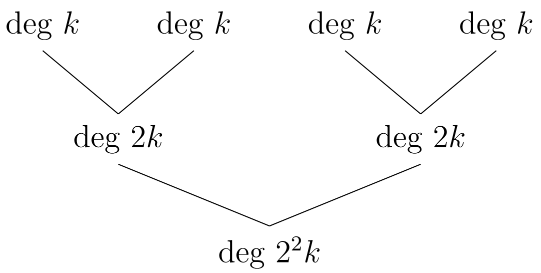

Figure 3.2: Multiplicative circuit of depth 2

To decrypt ciphertexts that are output from a circuit of depth d we must make

$$n \geq 2^d(k-1) + 1. \tag{3.1}$$

(Recall the encrypting polynomials have degree less than k.) Thus $n \gg k$ if we want to retain the ability to decrypt after ciphertexts have been multiplied with each other many times. Table 3.1 shows the maximum possible depth of multiplicative circuits for various values of n and k.

Throughout this paper we will only deal with cryptanalysis of ciphertexts that are the output of the encryption function as opposed to the product of an arithmetic circuit. We call such ciphertexts *fresh*. Being able to decrypt fresh ciphertexts constitutes a substantial attack even though we may not be able to decrypt the result of an arithmetic circuit. For instance consider the scenario where Bob is sending data to the cloud to be processed and then retrieving the result. The fresh and processed ciphertexts are equally vulnerable and an attacker could read any sensitive data as it was sent to the cloud.

		k					
		2	10	50	100	1000	$10,000$
	10	3	0				
	50	5	2	0			
	100	6	3	1	0		
n	500	8	5	3	2		
	1000	9	6	4	3	0	
	10^6	19	16	14	13	9	6
	10^9	29	26	24	23	19	16

Table 3.1: Maximum depth of multiplicative circuit

3.2 Linear Algebra Formulation

An alternative way of formulating the encryption process of Bubbles is through linear algebra. It turns out that each ciphertext is a linear combination of rows of a Vandermonde matrix and the space of all ciphertexts is the row space of a submatrix of the Vandermonde matrix. We will use this fact to perform a known plaintext attack. While breaking the scheme we will almost exclusively use the linear algebra formulation.

We assume an encryption session has been set up to work over a finite field $\mathbb{F}_q$, has integer parameters $k \leq n < q$, and has secret key $\mathbf{x} = (x_1, ..., x_n) \in (\mathbb{F}_q \backslash \{0\})^n$ where $x_i \neq x_j$ for $i \neq j$.

Construct the first k rows of a Vandermonde matrix

$$V = \begin{pmatrix} 1 & 1 & & 1 \\ x_1 & x_2 & \dots & x_n \\ x_1^2 & x_2^2 & & x_n^2 \\ \vdots & & \ddots & \vdots \\ x_1^{k-1} & x_2^{k-1} & \dots & x_n^{k-1} \end{pmatrix}.$$

Then to encrypt plaintext p we construct a vector

$$\mathbf{f} = (p, f_1, f_2, ..., f_{k-1}) \text{ where } f_1, ..., f_{k-1} \leftarrow_u \mathbb{F}_q.$$

Note that this vector also defines the polynomial $\bar{f}(x) = p + f_1 x + ... + f_{k-1}x^{k-1}$ and that $(\bar{f}(x_1), ..., \bar{f}(x_n)) = \mathbf{f}V$. So we can write

$$\mathrm{Enc}_{\mathbf{x}}(p) = \mathbf{f}V.$$

As an example we rework Example 3.1 in terms of linear algebra.

Example 3.3 (Reworking of Example 3.1 in linear algebra terms)**.**
Public parameters: $\mathbb{F}_q = \mathbb{Z}_{11}$, $n = 4$ and $k = 3$
Key: $\mathbf{x} = (3, 5, 2, 10)$

Plaintext: $p = 7$
Random polynomial: $f(x) = 4 + 0x \in \mathbb{Z}_{11}[x]_{k-1}$

We set

$$\mathbf{f} = (7, 4, 0) \quad \text{and} \quad V = \begin{pmatrix} 1 & 1 & 1 & 1 \\ 3 & 5 & 2 & 10 \\ 9 & 3 & 4 & 1 \end{pmatrix}.$$

This gives

$$\mathrm{Enc}_{\mathbf{x},f}(7) = \mathbf{f}V = (8, 5, 4, 3)$$

which is exactly the ciphertext we got before.

This formulation makes it abundantly clear that the space of all possible ciphertexts is a k dimensional subspace of $\mathbb{F}_q^n$. The space of all possible encryptions of 0 is a $k-1$ dimensional subspace of $\mathbb{F}_q^n$ equal to the row space of

$$V_0 = \begin{pmatrix} x_1 & x_2 & \dots & x_n \\ x_1^2 & x_2^2 & & x_n^2 \\ \vdots & & \ddots & \vdots \\ x_1^{k-1} & x_2^{k-1} & \dots & x_n^{k-1} \end{pmatrix}.$$

3.3 Cryptanalysis

The fact that the set of all ciphertexts is a subspace of $\mathbb{F}_q^n$ makes bubbles wide open to a known-plaintext attack. The general idea is that if we collect enough plaintext-ciphertext pairs we can find a basis for all polynomials f with $\deg(f) < k$ and $f(0) = 0$. Then we reduce any new ciphertext modulo this basis and end up with a constant polynomial which is easy to decode.

Now we outline the cryptanalysis process including explanation about what the steps do. Algorithm 3.2 gives the same method much more concisely. We assume we have the plaintexts $p_1, ..., p_m$, their encryptions $\mathbf{c}_1, ..., \mathbf{c}_m$ and a ciphertext $\mathbf{c}$ that we wish to decrypt.

1. Construct a matrix $C \in \mathbb{F}_q^{m \times n}$ where the i^{th} row is $\mathbf{c}_i - p_i \mathbb{1}$.

   Each row of C is an encryption of 0. Using the linear algebra formulation $C = FV_0$ for some $F \in \mathbb{F}_q^{m \times (k-1)}$. Assuming m is large enough and F is sufficiently random, the row space of C is the space of all possible ciphertexts encrypting 0.

2. Row reduce C and take all non-zero rows as a basis B' of all possible ciphertexts encrypting 0.

   Here we can easily check that the row space of C is the space of all possible ciphertexts encrypting 0 since if it is $\mathrm{rank}(C) = |B'| = k - 1$.

3. Create the matrix B whose last row is the all ones vector and the rest of the rows are the vectors in B'.

The row space of B is the space of all possible ciphertexts. This means that there are some $f_1, ..., f_{k-1}$ such that $(f_1, ..., f_{k-1}, p)B = \mathbf{c}$.

4. Solve the equation $B\mathbf{a} = (0, ..., 0, 1)^T$.

5. The plaintext is $\mathbf{ca}$.

   This is because $\mathbf{ca} = (f_1, ..., f_{k-1}, p)B\mathbf{a} = (f_1, ..., f_{k-1}, p)(0, ..., 0, 1)^T = p$

Note that all but the last step can be done without $\mathbf{c}$ and are essentially precomputation.

We will find it convenient to define a function ω on $(k-1) \times k$ matrices where if the rows of $C \in \mathbb{F}_q^{(k-1)\times k}$ are a basis for all encryptions of 0 then $\omega(C)$ is the precomputation of the algorithm to break Bubbles i.e. steps $2 - 4$.

Definition 3.4. To compute $\omega(C)$ augment the matrix C by adding a row of all ones on the bottom of C and call this matrix C'. If C' is not invertible then $\omega(C) = 0$. Otherwise $\omega(C)$ is the last column of C'^{-1}.

We can show that if the rows of C are linearly independent encryptions of 0, then the product of a ciphertext with $\omega(C)$ is the plaintext.

Proposition 3.5. *Let $F \in \mathbb{F}_q^{k-1,k-1}$ be full rank. Then $\mathrm{Enc}_{\mathbf{x}}(p)\omega(FV_0) = p$ where the x_i in V_0 are the entries of $\mathbf{x}$.*

Proof. Set C equal to the matrix FV_0 augmented with a row of ones on the bottom and set F' equal to the matrix F augmented as shown

$$F' = \begin{pmatrix} 0 & & & \\ \vdots & & F & \\ 0 & & & \\ 1 & 0 & ... & 0 \end{pmatrix}.$$

Then $C = F'V$ and F' is invertible. What's more the last column of F'^{-1} is $(1, 0, ..., 0)^T$. Since Vandermonde matrices are invertible and the product of invertible matrices is invertible, C is invertible and so

$$\begin{aligned} \omega(FV_0) &= \text{last column of } C^{-1} \\ &= \text{last column of } V^{-1}F'^{-1} \\ &= V^{-1}(1, 0, ..., 0)^T. \end{aligned}$$

Next note that $\mathrm{Enc}_{\mathbf{x}}(p) = (p, f_1, ..., f_{k-1})V$ for some $f_1, ..., f_{k-1} \in \mathbb{F}_q$ and so

$$\begin{aligned} \mathrm{Enc}_{\mathbf{x}}(p)\omega(FV_0) &= (p, f_1, ..., f_{k-1})VV^{-1}(1, 0, ..., 0)^T \\ &= (p, f_1, ..., f_{k-1})(1, 0, ..., 0)^T \\ &= p. \end{aligned}$$

□

```
Precomputation:
    1. Collect $m$ plaintext-ciphertext pairs $(p_1, \mathbf{c}_1), ..., (p_m, \mathbf{c}_m)$ and set $C$ as the matrix whose $i^{th}$ row is $\mathbf{c}_i - p_i\mathbb{1}$.
    2. Row reduce $C$ and let $\alpha = \{i \mid C_{\{i\}} \neq 0\}$ be the set of indices of all non-zero rows.
    3. If $|\alpha| < k - 1$ then go back to (1). Otherwise continue to (3).
    4. Compute $\mathbf{a} = \omega(C_\alpha^\beta)$ where $\beta = \{1, ..., k\}$.

Decryption of ciphertext $\mathbf{c}$:
    1. Calculate $p = \mathbf{ca}$
```

Algorithm 3.2: Cryptanalysis of basic Bubbles

Using this function ω, we write the cryptanalysis method formally in Algorithm 3.2.

It is interesting to note that this attack can work even if no plaintext is known as long as multiple ciphertexts encrypt each plaintext i.e. instead of plaintext-ciphertext pairs you have ciphertext-ciphertext pairs where both ciphertexts are encryptions of the same plaintext. In this case we just alter step 1. Instead of making the i^{th} row of C $\mathbf{c}_i - p_i\mathbb{1}$, we make it $\mathbf{c}_i - \mathbf{c}'_i$. Each row is still an encryption of 0.

Chapter 4

Adding Chaff

The method of popping bubbles relies on the ciphertexts all living in a k dimensional subspace of $\mathbb{F}_q^n$. The next two chapters describe different ways to take the ciphertexts out of this subspace in the hope of making it secure. However we will see that an attacker can usually put them back in the subspace and so these versions of bubbles are insecure except for when the parameters are chosen well.

4.1 Encryption and Decryption

As a first attempt to take the ciphertexts out of their k dimensional subspace we expand each ciphertext by inserting random entries called chaff in predetermined positions. These positions are part of the key and kept secret. Now the ciphertexts are not restrained to a k dimensional subspace of $\mathbb{F}_q^n$ so the cryptanalysis method from Chapter 3 will no longer work. However anyone with the key knows to ignore those positions when decrypting. This new encryption and decryption scheme is shown in Algorithm 4.1 and an example is worked out in Example 4.1.

Example 4.1 (Expansion of Example 3.1)**.**
Public parameters: $\mathbb{F}_q = \mathbb{Z}_{11}$, $n = 4$, $k = 3$, $n' = 7$
Key: $\mathbf{x} = (3, 5, 2, 10)$ $\mathbf{y} = (1, 3, 7)$
Plaintext: $p = 7$

As in Example 3.1 we randomly choose $f(x) = 4 + 0x$ so that our encrypting polynomial is $\bar{f}(x) = 7 + 4x$. For each of the three coordinates specified in $\mathbf{y}$ we uniformly choose a value from $\mathbb{Z}_{11}$, namely 4, 10 and 2. Then

$$\mathrm{Enc}(7) = (4, 8, 10, 5, 4, 3, 2).$$

```
Setup:
    1. Public parameters: finite field $\mathbb{F}_q$, integers $k \leq n < q$ and $n' \geq n$
    2. Key: $\mathbf{x} \in (\mathbb{F}_q \backslash \{0\})^n$ with $x_i \neq x_j$ for $i \neq j$ and $\mathbf{y} \in \mathcal{P}_{n'-n}(n')$

Encryption of plaintext $p \in \mathbb{F}_q$:
    1. Generate random polynomial $f(x) \leftarrow_u \mathbb{F}_q[x]_{k-1}$
    2. Set encrypting polynomial $\bar{f}(x) = p + xf(x)$
    3. Set $\mathbf{c} = \mathrm{Enc}_{\mathbf{x},f}(p) = (\bar{f}(x_1), \bar{f}(x_2), ..., \bar{f}(x_n))$
    4. Generate random $r_i \leftarrow_u \mathbb{F}_q$ for $i \in \mathbf{y}$.
    5. Ciphertext is $\mathbf{c}$ with $r_i$ inserted into the $i^{th}$ coordinate for $i \in \mathbf{y}$.

Decryption of ciphertext $\mathbf{c}$:
    1. Remove the entries of $\mathbf{c}$ indexed by values in $\mathbf{y}$.
    2. Use Lagrange interpolation to reconstruct $p = \bar{f}(0)$ from $\mathbf{x}$ and $\mathbf{c}$
```

Algorithm 4.1: Bubbles with chaff encryption and decryption

4.2 Cryptanalysis

To break this scheme all we have to do is figure out which locations are chaff. To do this we construct a matrix $C \in \mathbb{F}_q^{m \times n'}$ as before where the i^{th} row is an encryption of 0. When there is no chaff, i.e. $n' = n$, then C has rank k. Adding chaff adds a column to C that is most likely linearly independent of the current columns and so raises the rank to $k+1$. For now we assume that this is always true and later will analyze the probability of it being false. Because of this assumption we can also assume that any column is a linear combination of other columns if and only if it is not a chaff column.

If $2k - 1 \leq n$ we can find k non-chaff columns simply by row reducing C and taking all of the non-pivot columns. These are guaranteed to be non-chaff since they are linear combinations of other columns. There will be at least k of them because the space spanned by the non-chaff columns is $k-1$ dimensional and so only $k-1$ non-chaff columns will be pivot columns. This leaves $n-(k-1) \geq k$ columns as non-pivot columns. Now any ciphertext can be decrypted as in the previous section by only considering the entries corresponding to the selected non-chaff columns.

If $2k - 1 > n$ then the above method will not supply enough non-chaff columns. Instead we can winnow out the chaff columns one by one by comparing $\operatorname{rank}(C)$ with $\operatorname{rank}(C_i)$ where C_i is the matrix C without the i^{th} column. Then $\operatorname{rank}(C_i) = \operatorname{rank}(C)$ if column i is not chaff and $\operatorname{rank}(C_i) = \operatorname{rank}(C) - 1$ if column i is chaff. This method requires more computation but is guaranteed to find all non-chaff columns.

These methods depend on (1) n being larger than k and (2) the chaff columns being linearly independent from the non-chaff columns and each other. If $n = k$ then the set of non-chaff columns are linearly independent. If (2) also holds, every single column will be weeded out as chaff leaving no information that can be used to decrypt a ciphertext. This is a success for Bob. However if $n = k$ then Bob cannot decrypt the output of any multiplicative circuit. He is restricted to additive circuits. Thus Bubbles with chaff may be secure if it will only be used with additive circuits.

If we assume that Bob has designed the system so that he can decrypt the output of a multiplicative circuit of depth at least 1 then we know that (1) holds. Then the probability of (2) can be made as close to 1 as we want by increasing m, the number of encryptions of zero, or q, the order of the field. The cryptanalyst does not have control over q but we assume he can have as many known plaintext-ciphertext pairs as he wants and so he can control the likelihood of success through m.

4.2.1 Probability of Success

We now calculate the probability of a successful decryption assuming that (1) holds i.e. we calculate the probability of (2). Consider the probability that a vector of length m is linearly independent of the columns of a matrix with m rows and rank k over a field of order q. We denote this probability as

$$Pr_k^{m,q} := Pr[\mathbf{v} \notin \operatorname{colsp}(C) |\, \operatorname{rank}(C) = k, \mathbf{v} \underset{u}{\leftarrow} \mathbb{F}_q^m].$$

If a vector $\mathbf{v}$ and the columns of C are linearly dependent then the vector is one of the q^k vectors in the column space of C. Since there are q^m possible values for $\mathbf{v}$ the probability $\mathbf{v}$ is dependent is q^{k-m}. Therefore

$$Pr_k^{m,q} = 1 - q^{k-m}.$$

If there are $s < m - k$ chaff columns then the probability they are all linearly independent w.r.t. each other and the non-chaff columns is

$$\prod_{i=0}^{s-1} Pr_{k+i}^{m,q} = \prod_{i=0}^{s-1} 1 - q^{k+i-m} > \left(1 - q^{k+s-1-m}\right)^s$$

If we want to be able to separate the "wheat" from the chaff with probability τ then we need at least

$$m = k + s - 1 - \log_q\left(1 - \sqrt[s]{\tau}\right) < n - \log_q\left(1 - \sqrt[s]{\tau}\right)$$

ciphertext plaintext pairs.

Chapter 5

Adding Error

Bubbles is easily broken in its original form and adding chaff does not increase its security. Bubbles with chaff was easily broken because all spurious entries were always in the same positions. What would happen if random values were inserted at random places in the ciphertext? As an attacker collected more and more encryptions of 0 (plaintext-ciphertext pairs), more and more columns of the matrix C from the chaff decryption algorithm would contain errors. In a well designed system, close to every column of C would have an error by the time an attacker had enough encryptions of 0. This renders the attacker's decryption algorithm in Chapter 4 useless.

However introducing errors in random locations also prevents Bob (someone who has the key) from decrypting using any algorithm introduced so far. Bob needs a way to recognize which entries are noise and recover the original ciphertext. This is the study of error correcting codes. Fortunately for Bob, all ciphertexts encrypted using a single key are precisely generalized Reed-Solomon code words [11].

Reed-Solomon codes are error correcting codes used extensively in CD's, DVD's, barcodes, RAID storage architecture etc. They have the property that if a limited number of entries of a codeword are corrupted (i.e. changed to a different value) then the original codeword can be recovered efficiently. The interested reader may consult [11] for a thorough explanation of Reed-Solomon decoding. We will only need the following proposition which we state without proof.

Proposition 5.1. *Let* $\mathbf{c} = \mathrm{Enc}_{\mathbf{x},f,\epsilon}(p)$, $n = \mathrm{length}(\mathbf{x}) = \mathrm{length}(\epsilon)$ *and* $k = \deg(f) + 1$. *Then Reed-Solomon decoding can reconstruct* $\bar{f}$ *from* $\mathbf{x}$ *and* $\mathbf{c}$ *if* $w_H(\epsilon) < \frac{n-k}{2}$.

The encryption function $\mathrm{Enc}_{\mathbf{x},f,\epsilon}(p)$ will be defined soon. For now it suffices to know that the vector $\epsilon \in \mathbb{F}_q^n$ is a vector of errors we add to the ciphertext.

Anyone with the key can use Reed-Solomon decoding to decrypt the ciphertext but the cryptanalysis methods of the previous sections no longer work. However an algorithm based on the same principles as in the previous sections will break Bubbles with errors for some parameter choices. While some parameter choices leave Bubbles with errors resistant to this attack, we will see that those parameter choices make Bubbles very inefficient and impractical. At this time virtually all homomorphic encryption systems are impractical so this may not seem like a big deal. However these safe parameter choices make Bubbles less efficient than the most recent inefficient schemes.

5.1 Bob's Perspective

First we look at the system from the perspective of Bob wanting to encrypt and decrypt. The set up is the same as before except we also choose a parameter $e < \frac{n-k}{2}$. This is the number of errors per ciphertext. For each encryption we select a polynomial $f(x) \leftarrow_u \mathbb{F}_q[x]_{k-1}$ as before. We also select an error vector $\epsilon = (\epsilon_1, ..., \epsilon_n)$ uniformly at random from the set of all vectors in $\mathbb{F}_q^n$ with Hamming weight e. We denote the ciphertext resulting from these choices f and ϵ by

$$\mathrm{Enc}_{\mathbf{x},f,\epsilon}(p) := (\bar{f}(x_1) + \epsilon_1, \bar{f}(x_2) + \epsilon_2, ..., \bar{f}(x_n) + \epsilon_n).$$

If f and ϵ are not specified we write $\mathrm{Enc}_{\mathbf{x}}(p)$ and assume f and ϵ are chosen uniformly at random.

To decrypt a ciphertext $\mathbf{c} = (\bar{f}(x_1) + \epsilon_1, ..., \bar{f}(x_n) + \epsilon_n)$, Bob can use generalized Reed Solomon decoding to reconstruct $\bar{f}(x)$ and then compute $p = \bar{f}(0)$. By Proposition 5.1 this works as long as $\mathbf{x}$ is known and $\mathrm{w_H}(\epsilon) < \frac{n-\deg(f)-1}{2}$.

These encryption and decryption algorithms are summarized in Algorithm 5.1.

Setup:

1. Choose public parameters: finite field $\mathbb{F}_q$, integers $k \leq n < q$ and $e < \frac{n-k}{2}$
2. Choose secret key: $\mathbf{x} \in (\mathbb{F}_q \backslash \{0\})^n$ with $x_i \neq x_j$ for $i \neq j$

Encryption of plaintext $p \in \mathbb{F}_q$:

1. Generate random polynomial $f(x) \leftarrow_u \mathbb{F}_q[x]_{k-1}$
2. Set $\bar{f}(x) = p + xf(x)$
3. Generate random vector $\epsilon \leftarrow_u \{\epsilon \in \mathbb{F}_q^n \mid w_H(\epsilon) = e\}$
4. Ciphertext $\mathbf{c} = \mathrm{Enc}_{\mathbf{x},f,\epsilon}(p) = (\bar{f}(x_1) + \epsilon_1, \bar{f}(x_2) + \epsilon_2, ..., \bar{f}(x_n) + \epsilon_n)$

Decryption of ciphertext $\mathbf{c}$:

1. Use Reed-Solomon decoding to reconstruct $\bar{f}(x)$ from $\mathbf{x}$ and $\mathbf{c}$
2. Calculate $p = \bar{f}(0)$

Algorithm 5.1: Bubbles with errors encryption and decryption

Bob can decrypt any fresh ciphertexts. However he runs into issues whenever dealing with additive or multiplicative circuits of depth greater than 0. The non-zero ϵ are not restricted to the same positions in different ciphertexts like the chaff as described in Chapter 4. Let c and c' be ciphertexts with $w_H(\epsilon) = e$ and $w_H(\epsilon') = e'$. Their sum and product have up to $e + e'$ values corrupted. If multiple ciphertexts with e errors are input to an additive or multiplicative circuit of depth d then the resulting ciphertext has $2^d e$ errors. Therefore to ensure correct decryption of the output of an additive circuit of depth d, e must be bounded by

$$e \leq \frac{n-k}{2^{d+1}}. \tag{5.1}$$

In a multiplicative circuit we must also factor in how the degrees of the encrypting polynomials f grow when the ciphertexts are multiplied. Bob must be able to decrypt a message encrypted with a polynomial of degree up to $2^d(k-1)$. This means

$$e \leq \frac{n - 2^d(k-1) - 1}{2^{d+1}} \tag{5.2}$$

to ensure Reed-Solomon decryption will correctly decode the message.

5.2 Cryptanalysis

The upper bounds on e in (5.1) and (5.2) will allow an adversary to break the scheme. We will find that for any choices of n, k, and q and for all $\tau \in (0, 1)$ there is a value D such that an adversary can perform a known plaintext attack with probability of success greater than τ if $d \geq D$. Note that this is trivially true if D is set to be the maximum multiplicative circuit depth possible for a given n and k. In this case $e = 0$ and so the methods of Chapter 3 guarantee success. Our goal is to make D as small as possible. Loosely we will find that D is about half of the maximum multiplicative circuit depth.

5.2.1 Algorithms

The basic idea of the method is that to decrypt a fresh ciphertext we just need k entries in the ciphertext that are error free as well as $k-1$ linearly independent encryptions of 0 with the same k entries being error free. If we have this we can decrypt using the method from Chapter 3.

However we do not know which entries are error free. Instead of proceeding with certainty we must guess. If we guess many times we will get multiple potential plaintexts. Some will be wrong because errors messed up our computation but most likely the correct plaintext will also show up. In fact the correct plaintext will most likely show up more frequently than any other value if $d \geq D$. By guessing enough times we can control the probability that the most frequently occurring value is the correct plaintext.

```
Precomputation Method 1:
  1. Collect $m$ plaintext-ciphertext pairs $(p_1, \mathbf{c}_1), ..., (p_m, \mathbf{c}_m)$ and set $C$ as the matrix whose $i^{th}$ row is $\mathbf{c}_i - p_i$.
  2. For $i = 1, ..., \lambda$
     (a) $(\alpha_i, \beta_i) \leftarrow_u \mathcal{P}_{k-1}(m) \times \mathcal{P}_k(n)$
     (b) $\mathbf{a}_i = \omega(C_{\alpha_i}^{\beta_i})$
     (c) Store $\mathbf{a}_i$ and $\beta_i$

Precomputation Method 2 (if $ke < \frac{n-k+1}{2}$):
  1. Collect $k$ plaintext-ciphertext pairs $(p_1, \mathbf{c}_1), ..., (p_k, \mathbf{c}_k)$ and set $C$ as the matrix with $k$ rows whose $i^{th}$ row is $\mathbf{c}_i - p_i$.
  2. For $\gamma \in \mathcal{P}_{k-1}(n)$
     (a) Row reduce $C$ with respect to the columns indexed in $\gamma$ and call the resulting matrix $\bar{C}$.
     (b) If over $\frac{n+k-1}{2}$ of the columns end in two 0's go to step 1.
     (c) If over $\frac{n+k-1}{2}$ of the columns end in one 0
         i. Set $\beta$ to be the set of indices of columns of $\bar{C}$ that end in 0.
        ii. Break to step 3.
  3. For $i = 1, ..., \lambda$
     (a) $\beta_i \leftarrow_u \mathcal{P}_k(\beta)$, $\alpha = \{1, ..., k-1\}$
     (b) $\mathbf{a}_i = \omega(\bar{C}_{\alpha}^{\beta_i})$
     (c) Store $\mathbf{a}_i$ and $\beta_i$

Decrypting ciphertext $\mathbf{c}$:
  1. Calculate the product $p_i = \mathbf{c}_{\beta_i} \mathbf{a}_i$ for $i = 1, ..., \lambda$.
  2. The plaintext is the value $p$ which maximizes $|\{i : p_i = p\}|$.
```

Algorithm 5.2: Two methods for breaking Bubbles with errors

Figure 5.2 contains two algorithms to break Bubbles with errors. Each algorithm performs different precomputation on encryptions of 0 but process a ciphertext in the same way. Precomputation Method 1 exactly follows the method sketched above. Precomputation Method 2 spends time and computation weeding out many of the corrupted terms similarly to how we winnowed out the chaff. However these extra steps only make sense if $ke < \frac{n-k+1}{2}$. Thus Method 2 can only be applied in some cases and has the cost of more computation but is more likely than Method 1 to decrypt correctly.

5.2.2 Probability of Successful Decryption Using Method 1

We now calculate τ_1 the probability of a successful decryption using Precomputation Method 1. Suppose $\mathbf{c}$ is an encryption of p^*. If over half of the p_i equal p^* then we are guaranteed successful decryption and so

$$\tau > \Pr[|\{i : p_i = p^*\}| > \lambda/2].$$

We define π_1 as the probabilities that $p_i = p^*$ for $i \leftarrow_u \{1, ..., \lambda\}$ when Precomputation Method 1 is used. In all future probabilities we shall assume that $i \leftarrow_u \{1, ..., \lambda\}$. Since the choices of (α_i, β_i) are independent events, so is the event that $p_i = p^*$. This means we can write

$$\Pr[|\{i : p_i = p^*\}| = r] = \pi^r \cdot (1 - \pi)^{\lambda - r} \binom{\lambda}{r}$$

which gives

$$\tau_1 > \Pr\left[|\{i : p_i = p^*\}| > \lambda/2\right] = \sum_{r=\lfloor \frac{\lambda}{2} \rfloor + 1}^{\lambda} \pi_1^r \cdot (1 - \pi_1)^{\lambda - r} \binom{\lambda}{r} \tag{5.3}$$

If $\pi_1 > 0.5$ we can force τ_1 to be within epsilon of 1 by increasing λ. So it is possible to break the scheme if π_1 is larger than 0.5. To give an idea of the relationship between these numbers Table 5.1 shows the lower bound on τ_1 given by equation (5.3) for a few values of π_1 and λ.

			λ	
		10	30	50
π_1	.3	0.047349	0.0063703	0.00093318
	.4	0.16624	0.097057	0.057344
	.5	0.37695	0.42777	0.44386
	.6	0.6331	0.82463	0.90219
	.7	0.84973	0.98306	0.99763
	.8	0.96721	0.99977	1.00000
	.9	0.99837	1.00000	1.00000

Table 5.1: Lower bound on τ_1 given by equation (5.3)

Derivation of π_1

We now calculate π_1. We can write

$$\underset{1\times n}{\mathbf{c}} = [p^* \,\vdots\, \underset{1\times(k-1)}{\mathbf{f}} \quad]\cdot \underset{k\times n}{V} + \underset{1\times n}{\mathbf{e}}$$

and

$$\underset{m\times n}{C} = \underset{m\times(k-1)}{F} \cdot \underset{(k-1)\times n}{V_0} + \underset{m\times n}{E}.$$

In the encryption process $\mathbf{f}$ and the rows of F were chosen uniformly at random from $\mathbb{F}_q^{k-1}$ and $\mathbf{e}$ and the rows of E were chosen uniformly at random from the vectors in $\mathbb{F}_q^n$ with Hamming weight e. The matrices V_0 and V have the form

$$V := \begin{pmatrix} 1 & 1 & & 1 \\ x_1 & x_2 & \dots & x_n \\ x_1^2 & x_2^2 & & x_n^2 \\ \vdots & & \ddots & \vdots \\ x_1^{k-1} & x_2^{k-1} & \dots & x_n^{k-1} \end{pmatrix} \quad \text{and} \quad V_0 := \begin{pmatrix} x_1 & x_2 & \dots & x_n \\ x_1^2 & x_2^2 & & x_n^2 \\ \vdots & & \ddots & \vdots \\ x_1^{k-1} & x_2^{k-1} & \dots & x_n^{k-1} \end{pmatrix}. \tag{5.4}$$

where the x_i are the entries of the secret key used to encrypt all ciphertexts.

By Proposition 3.5, $p_i = \mathbf{c}_{\beta_i}\omega(C^{\beta_i}) = p^*$ if $\mathbf{e}_{\beta_i} = 0$, $E_{\alpha_i}^{\beta_i} = 0$ and $\text{rank}(F_{\alpha_i}) = k-1$. These are all independent events so

$$\pi_1 \geq \Pr[\mathbf{e}_{\beta_i} = 0] \cdot \Pr[E_{\alpha_i}^{\beta_i} = 0] \cdot \Pr[\text{rank}(F_{\alpha_i}) = k-1]. \tag{5.5}$$

Since $\mathbf{e}$ has length n and Hamming weight e

$$\begin{aligned} \Pr[\mathbf{e}_{\beta_i} = 0] &= \frac{\binom{n-e}{k}}{\binom{n}{k}} \\ &> \left(\frac{n-e-k+1}{n-k+1}\right)^k \\ &= \left(1 - \frac{e}{n-k+1}\right)^k. \end{aligned} \tag{5.6}$$

Using similar logic and knowing that the rows of E are independent of each other gives

$$\Pr[E_{\alpha_i}^{\beta_i} = 0] = \left(\frac{\binom{n-e}{k}}{\binom{n}{k}}\right)^{k-1} > \left(1 - \frac{e}{n-k+1}\right)^{k(k-1)}. \tag{5.7}$$

By Lemma 2.8 since F_{α_i} is $(k-1)\times(k-1)$ we know that

$$\Pr[\text{rank}(F_{\alpha_i}) = k-1] = \prod_{i=0}^{k-2} 1 - q^{i-k+1} > \left(1 - \frac{1}{q}\right)^{k-1}. \tag{5.8}$$

Substituting equations (5.6) - (5.8) into (5.5) gives

$$\pi_1 > \left(1 - \frac{e}{n-k+1}\right)^{k^2} \left(1 - \frac{1}{q}\right)^{k-1}.$$

Recall that e is bounded above to ensure that someone with the key can correctly decrypt the output from an additive or multiplicative circuit of depth d. See (5.1) for the bound for an additive circuit and (5.2) for the bound in multiplicative circuits. Also, $q > n$ since the key $\mathbf{x}$ consists of n distinct $x_i \in \mathbb{F}_q$. We can use these bounds to bound π_1 below in terms of just n, d and k. If the ciphertexts are designed to work in a multiplicative circuit of depth d then

$$\pi_1 > \pi_1^{\times}(n,d,k) := \left(1 - \frac{\lfloor \frac{n-2^d(k-1)-1}{2^{d+1}} \rfloor}{(n-k+1)}\right)^{k^2} \left(1 - \frac{1}{n}\right)^{k-1}. \tag{5.9}$$

If they can be decrypted after an additive circuit of depth d then

$$\pi_1 > \pi_1^{+}(n,d,k) := \left(1 - \frac{\lfloor \frac{n-k}{2^{d+1}} \rfloor}{(n-k+1)}\right)^{k^2} \left(1 - \frac{1}{n}\right)^{k-1}. \tag{5.10}$$

Table 5.2 (resp. Table 5.3) shows the worst case values for $\pi_1^{\times}$ (resp. π_1^{+}) for various values of n and $d < \log_2 n$ (see (3.1)). For each n, d pair software was used to find the k that minimizes $\pi_1^{\times}$ (resp. π_1^{+}). Then k was set to the integer nearest this value. This k was used to calculate $\pi_1^{\times}$ (resp. π_1^{+}) so that Table 5.2 (resp. Table 5.3) represents a worst case scenario from the perspective of an attacker. The tables in Appendix A show these choices for k and the corresponding maximum value for e. In all tables, any number smaller than 10^{-100} is written as 0.000000.

This imitates the parameter choice of someone setting up an encryption session. From the perspective of someone encrypting perhaps the most important parameters are n and d since n is the overhead and d the utility of the system. Each ciphertext is n times the size of the plaintext and d is the maximum depth of a multiplicative circuit that can process the ciphertexts. The parameter k affects the amount of computation time to encrypt and decrypt.

Recall that once $\pi_1 > 0.5$ we can increase the algorithm's probability of success τ_1 by increasing the number of samples λ. Hence Table 5.2 shows that for $n = 100$ ciphertexts designed for an additive circuit of depth 4 or more can be decrypted by this algorithm with probability $\tau < 1$. Thus someone encrypting is restricted to circuits of depth at most 3. Similarly, if $n = 1000$ then the greatest circuit depth ciphertexts can be designed for is 5. If $n = 10^6$ then the max circuit depth is 12. Finally, if $n = 10^9$ then the max depth is 18.

It is reasonable to suppose that the scheme may fare better if we just consider additive circuits since when ciphertexts are added the degree of the encrypting polynomial does not increase. This means that there can be more error terms which makes it harder to break. However Table 5.3 shows us that for each of the given n an additive circuit can only have depth one greater than the maximum safe depth of a multiplicative circuit. This is not a significant increase so just considering additive circuits does not increase the usability of Bubbles with error.

		n			
		100	1000	10^6	10^9
	1	$1.043314 \cdot 10^{-64}$	0.000000	0.000000	0.000000
	2	$3.748847 \cdot 10^{-05}$	0.000000	0.000000	0.000000
	3	0.155564	$9.864651 \cdot 10^{-68}$	0.000000	0.000000
	4	0.739350	$3.593685 \cdot 10^{-9}$	0.000000	0.000000
	5	0.970299	$8.214621 \cdot 10^{-2}$	0.000000	0.000000
	6	0.990000	0.775151	0.000000	0.000000
	7		0.971304	0.000000	0.000000
	8		0.998001	0.000000	0.000000
	9		0.999000	0.000000	0.000000
	10			$1.216064 \cdot 10^{-30}$	0.000000
	11			$1.778075 \cdot 10^{-04}$	0.000000
	12			0.340336	0.000000
	13			0.872445	0.000000
	14			0.982473	0.000000
d	15			0.998044	0.000000
	16			0.999691	0.000000
	17			0.999971	$5.139106 \cdot 10^{-15}$
	18			0.999998	$1.642708 \cdot 10^{-02}$
	19			0.999999	0.597796
	20				0.937945
	21				0.992045
	22				0.998989
	23				0.999875
	24				0.999984
	25				0.999998
	26				0.999999
	27				1.000000
	28				1.000000
	29				1.000000

Table 5.2: Lower bound on π_1 given by equation (5.9) (d is depth of multiplicative circuit)

		n			
		100	1000	10^6	10^9
	1	0.000000	0.000000	0.000000	0.000000
	2	$4.825043 \cdot 10^{-35}$	0.000000	0.000000	0.000000
	3	$4.510746 \cdot 10^{-5}$	0.000000	0.000000	0.000000
	4	0.328208	$6.919062 \cdot 10^{-55}$	0.000000	0.000000
	5	0.822048	$1.118197 \cdot 10^{-7}$	0.000000	0.000000
	6	0.990000	0.158691	0.000000	0.000000
	7		0.818199	0.000000	0.000000
	8		0.981122	0.000000	0.000000
	9		0.999000	0.000000	0.000000
	10			0.000000	0.000000
	11			$4.417561 \cdot 10^{-26}$	0.000000
	12			$6.585117 \cdot 10^{-04}$	0.000000
	13			0.397272	0.000000
	14			0.891018	0.000000
d	15			0.985658	0.000000
	16			0.998194	0.000000
	17			0.999801	$3.703848 \cdot 10^{-97}$
	18			0.999981	$8.833603 \cdot 10^{-13}$
	19			0.999999	$3.113657 \cdot 10^{-2}$
	20				0.648420
	21				0.947287
	22				0.993225
	23				0.999150
	24				0.999895
	25				0.999987
	26				0.999998
	27				0.999999
	28				1.000000
	29				1.000000

Table 5.3: Lower bound on π_1 given by equation (5.10) (d is depth of additive circuit)

5.2.3 Probability of Successful Decryption Using Method 2

We will now calculate a lower bound for τ_2, the probability of decrypting correctly using Precomputation Method 2.

As in Method 1 let us write

$$\underset{k\times n}{C} = \underset{k\times(k-1)}{F} \cdot \underset{(k-1)\times n}{V_0} + \underset{k\times n}{E}$$

for the matrix C generated in step 1 of the algorithm. By Lemma 2.7 there is a matrix $M \in \mathbb{F}_q^{k-1\times k}$ such that $\bar{C} = MC$. We set $\bar{F} = MF$ and $\bar{E} = ME$ so that $\bar{C} = \bar{F}V_0 + \bar{E}$.

Similarly to Method 1, $p_i = p$ if $\mathbf{e}_{\beta_i} = 0$, $\bar{E}_\alpha^{\beta_i} = 0$ and $\operatorname{rank}(\bar{F}_\alpha) = k-1$. The bound for $\Pr[\mathbf{e}_{\beta_i} = 0]$ in (5.6) still holds. However the other two probabilities have changed.

First consider $\operatorname{rank}(\bar{F}_\alpha)$. This is constant over i instead of changing over i like it did in Method 1. This means we will want to calculate $\Pr[p_i = p^* \mid \operatorname{rank}(\bar{F}_\alpha) = k-1]$ instead of $\Pr[p_i = p^*]$. Therefore we write

$$\begin{aligned} \tau_2 &> \Pr[|\{i : p_i = p^*\}| > \lambda/2] \\ &= \Pr[|\{i : p_i = p^*\}| > \lambda/2 \mid \operatorname{rank}(\bar{F}_\alpha) = k-1] \cdot \Pr[\operatorname{rank}(\bar{F}_\alpha) = k-1] \end{aligned}$$

and define $\pi_2 = \Pr[p_i = p^* \mid \operatorname{rank}(\bar{F}_\alpha) = k-1]$.

We will later show that $\Pr[\operatorname{rank}(\bar{F}_\alpha) = k-1] > (1-q^{-2})^{k-1}$. Note that $(1-q^{-2})^{k-1} = 1 + \mathcal{O}(q^{-2})$. For large q this is very close to 1 in which case Table 5.1, which gives lower bounds for τ_1 for different values of π_1, also applies to τ_2 and π_2 for all practical purposes. Thus our goal is to make $\pi_2 > 0.5$.

We set

$$\begin{aligned} \rho &:= \{i \le n \mid \bar{E}^{\{i\}} = 0\}, \\ P &:= |\rho|, \\ P' &:= |\{i \in \beta \mid \bar{E}^{\{i\}} = 0\}|, \\ L &:= |\{i \le n \mid \bar{E}^{\{i\}} \ne 0\}| \text{ and} \\ L' &:= |\{i \in \beta \mid \bar{E}^{\{i\}} \ne 0\}|. \end{aligned}$$

Thus ρ is the set of indices of "pure" columns of $\bar{C}$ which can be used to correctly decrypt a ciphertext and L is the number of corrupted columns which may "lie" about what the plaintext is. We note that $P + L = n$, $P' + L' = |\beta|$, $P' \le P$, $L' \le L$ and $e \le L \le ke$.

Remember that we only consider using Method 2 if $ke < \frac{n-k+1}{2}$. We will show that this guarantees that $P' = P > \frac{n+k-1}{2}$.

Since $L \le ke < \frac{n-k+1}{2}$, and $P = n - L$ we have that $P > \frac{n+k-1}{2}$. Now we just need that $P' = P$.

We have that $|\rho| = P > \frac{n+k-1}{2}$, $|\beta| > \frac{n+k-1}{2}$, and $\rho, \beta \subset \mathcal{P}(n)$. Therefore $|\rho \cap \beta| > k-1$ and by Lemma 2.4 $\rho \subset \beta$. This means that $P' = P$. In other words β contains the indices of all pure columns.

Derivation of $\Pr[\mathrm{rank}(\bar{F}_\alpha) = k-1]$

We can now calculate a lower bound for the probability that $\mathrm{rank}(\bar{F}_\alpha) = k-1$.

Lemma 5.2. $\mathrm{rank}(\bar{F}_\alpha) = \mathrm{rank}(F)$

Proof. The last row of $\bar{C}^\rho$ is 0 since $\rho \subset \beta$. Because $\bar{E}^\rho = 0$ we have that $\bar{C}^\rho = \bar{F}V_0^\rho$ and so the last row of $\bar{F}V_0^\rho$ is 0. But the rows of V_0^ρ are linearly independent by Lemma 2.2 since $|\rho| > k-1$. This means that the last row of $\bar{F}$ is all zeros. Since $\bar{F}_\alpha$ is $\bar{F}$ without its last row we have $\mathrm{rank}(\bar{F}_\alpha) = \mathrm{rank}(\bar{F})$.

Since $\bar{F} = MF$ and M is invertible $\mathrm{rank}(F) = \mathrm{rank}(\bar{F})$. Therefore $\mathrm{rank}(\bar{F}_\alpha) = \mathrm{rank}(F)$ □

By Lemma 2.8 the probability that F is full rank is greater than $(1-q^{-2})^{k-1}$. Therefore Lemma 5.2 gives that

$$\Pr[\mathrm{rank}(\bar{F}_\alpha) = k-1] > (1-q^{-2})^{k-1}.$$

Derivation of π_2

We now calculate a lower bound for π_2, the probability that $p_i = p^*$ for $i \leftarrow_u \{1, ..., \lambda\}$ when Precomputation Method 2 is used given that $\mathrm{rank}(\bar{F}_\alpha) = k-1$. So far we have

$$\begin{aligned}\pi_2 &\geq \Pr[\mathbf{e}_{\beta_i} = 0] \cdot \Pr[E_\alpha^{\beta_i} = 0] \\ &> \left(1 - \frac{e}{n-k+1}\right)^k \cdot \Pr\left[\bar{E}_\alpha^{\beta_i} = 0\right]. \end{aligned} \tag{5.11}$$

Note that if $L' = 0$ then $\bar{E}_\alpha^{\beta_i} = 0$ so

$$\Pr\left[\bar{E}_\alpha^{\beta_i} = 0\right] \geq \Pr\left[L' = 0\right].$$

There is a q^{-1} probability that a corrupted column lies in the $(k-1)$-dimensional space spanned by the columns of $\bar{C}^\beta$ and so

$$\Pr[L' = 0] = \left(1 - \frac{1}{q}\right)^L.$$

Since $n < q$ and $L \leq ke$ we have

$$\Pr[L' = 0] > \left(1 - \frac{1}{n}\right)^{ke}.$$

Therefore

$$\pi_2 > \left(1 - \frac{e}{n-k+1}\right)^k \left(1 - \frac{1}{n}\right)^{ke}. \tag{5.12}$$

Like we did when calculating π_1 we can write e in terms of k and d where d is the depth of a multiplicative or additive circuit. In this way we define

$$\pi_2^{\times}(n,d,k) = \left(1 - \frac{\lfloor\frac{n-2^d(k-1)-1}{2^{d+1}}\rfloor}{n-k+1}\right)^k \left(1-\frac{1}{n}\right)^{k\lfloor\frac{n-2^d(k-1)-1}{2^{d+1}}\rfloor} \tag{5.13}$$

and

$$\pi_2^{+}(n,d,k) = \left(1 - \frac{\lfloor\frac{n-k}{2^{d+1}}\rfloor}{n-k+1}\right)^k \left(1-\frac{1}{n}\right)^{k\lfloor\frac{n-k}{2^{d+1}}\rfloor}. \tag{5.14}$$

Again Tables 5.2 and 5.3 contain the lowest bound for $\pi_2^{\times}$ and π_2^{+}, respectively. For each n and d combination, computer software was used to minimize $\pi_2^{\times}$ and π_2^{+} with respect to k and then that k was used to calculate the value of $\pi_2^{\times}$ and π_2^{+}. The tables in Appendix B contain these values for k along the values values for e.

We can compare the probabilities of success for Methods 1 and 2 by comparing the π_1 tables (Tables 5.2 and 5.3) with the π_2 tables (Tables 5.4 and 5.5). Whenever Method 2 can be used (when $ke < \frac{n-k+1}{2}$) then $\pi_2 \geq \pi_1$. This is as expected since Method 2 introduces more precomputation to increase the probability of success. Also note that whenever we cannot use Method 2, Method 1 is very unlikely to work so Method 2 is our best bet at successfully decrypting a ciphertext.

These results are perhaps not quite as satisfactory as the results from the previous chapters because there are parameter choices that leave Bubbles with errors resistant to our attacks. However these parameters make Bubbles have a larger blow up factor than some of the most recent homomorphic cryptosystems. There is a homomorphic cryptosystem that can handle an arithmetic circuit of depth 20 when the ciphertext is $26.4 \cdot 10^6$ times the size of the plaintext [4]. This is a smaller ciphertext blowup than Bubbles with errors requires. To find the minimum ciphertext blowup Bubbles will have while supporting a depth 20 arithmetic circuit, we minimize n with the constraint that $\pi_2^{\times}(n,20,k) < 0.5$. Computer software gives $n \approx 10^{13}$.

			n		
		100	1000	10^6	10^9
	1	–	–	–	–
	2	–	–	–	–
	3	0.645258	–	–	–
	4	0.902480	–	–	–
	5	1.000000	0.775452	–	–
	6	1.000000	0.947174	–	–
	7		0.990025	–	–
	8		1.000000	–	–
	9		1.000000	0.384769	–
	10			0.788044	–
	11			0.941963	–
	12			0.985110	–
	13			0.996236	–
	14			0.999048	0.394015
d	15			0.999776	0.792297
	16			0.999936	0.943459
	17			0.999990	0.985551
	18			1.000000	0.996370
	19			1.000000	0.999090
	20				0.999773
	21				0.999943
	22				0.999985
	23				0.999996
	24				0.999999
	25				0.999999
	26				1.000000
	27				1.000000
	28				1.000000
	29				1.000000

Table 5.4: Minimum value for $\pi_2^\times$ given by equation (5.13) for given ciphertext size n and multiplicative circuit depth d.

		n			
		100	1000	10^6	10^9
	1	–	–	–	–
	2	–	–	–	–
	3	–	–	–	–
	4	0.781712	–	–	–
	5	0.921592	0.375510	–	–
	6	1.000000	0.797584	–	–
	7		0.952926	–	–
	8		0.992016	–	–
	9		1.000000	–	–
	10			0.385899	–
	11			0.787647	–
	12			0.941963	–
	13			0.985104	–
	14			0.996286	–
d	15			0.999070	0.394039
	16			0.999776	0.792293
	17			0.999952	0.943459
	18			0.999992	0.985554
	19			1.000000	0.996370
	20				0.999092
	21				0.999773
	22				0.999943
	23				0.999985
	24				0.999996
	25				0.999999
	26				0.999999
	27				1.000000
	28				1.000000
	29				1.000000

Table 5.5: Minimum value for π_2^+ given by equation (5.14) for given ciphertext size n and additive circuit depth d.

Appendix A

Lower bounds for π_1

These tables contain the values for k and e that minimize $\pi_1^\times$ and π_1^+ for a given circuit depth d. Any number smaller than 10^{-100} is written as 0.000000.

d	k	e	$\pi_1^\times$
1	34	8	$1.043314 \cdot 10^{-64}$
2	20	2	$3.748847 \cdot 10^{-5}$
3	9	2	0.155564
4	5	1	0.739350
5	4	0	0.970299
6	2	0	0.990000

d	k	e	π_1^+
1	48	13	0.000000
2	25	9	$4.825043 \cdot 10^{-35}$
3	13	5	$4.510746 \cdot 10^{-5}$
4	7	2	0.328208
5	4	1	0.822048
6	2	0	0.990000

Table A.1: $n = 100$

d	k	e	$\pi_1^\times$
1	493	3	0.000000
2	191	29	0.000000
3	88	18	$9.864651 \cdot 10^{-68}$
4	43	10	$3.593685 \cdot 10^{-9}$
5	22	5	$8.214621 \cdot 10^{-2}$
6	11	2	0.775151
7	5	1	0.971304
8	3	0	0.998001
9	2	0	0.999000

d	k	e	π_1^+
1	500	125	0.000000
2	250	93	0.000000
3	111	55	0.000000
4	63	29	$6.919062 \cdot 10^{-55}$
5	32	15	$1.118197 \cdot 10^{-7}$
6	16	7	0.158691
7	8	3	0.818199
8	4	1	0.981122
9	2	0	0.999000

Table A.2: $n = 1000$

d	k	e	$\pi_1^\times$
1	499997	1	0.000000
2	249998	1	0.000000
3	124998	1	0.000000
4	62497	1	0.000000
5	31248	1	0.000000
6	15614	5	0.000000
7	7749	32	0.000000
8	3689	109	0.000000
9	1278	338	0.000000
10	642	167	$1.216064 \cdot 10^{-30}$
11	337	76	$1.778075 \cdot 10^{-4}$
12	173	36	0.340336
13	77	23	0.872445
14	42	10	0.982473
15	22	4	0.998044
16	10	3	0.999691
17	5	1	0.999971
18	3	0	0.999998
19	2	0	0.999999

d	k	e	π_1^+
1	500000	125000	0.000000
2	250000	93750	0.000000
3	125000	54687	0.000000
4	62500	29296	0.000000
5	31250	15136	0.000000
6	15625	7690	0.000000
7	7813	3875	0.000000
8	3907	1945	0.000000
9	1208	975	0.000000
10	977	487	0.000000
11	489	244	$4.417561 \cdot 10^{-26}$
12	245	122	$6.585117 \cdot 10^{-4}$
13	123	61	0.397272
14	62	30	0.891018
15	31	15	0.985658
16	16	7	0.998194
17	8	3	0.999801
18	4	1	0.999981
19	2	0	0.999999

Table A.3: $n = 10^6$

d	k	e	$\pi_1^{\times}$	d	k	e	π_1^{+}
1	499999992	4	0.000000	1	499999992	125000002	0.000000
2	249999996	2	0.000000	2	249999996	93750000	0.000000
3	124999998	1	0.000000	3	124999998	54687500	0.000000
4	62499998	1	0.000000	4	62499999	29296875	0.000000
5	31249998	1	0.000000	5	31250000	15136718	0.000000
6	15624998	1	0.000000	6	15625000	7690429	0.000000
7	7812498	1	0.000000	7	7812500	3875732	0.000000
8	3906248	1	0.000000	8	3906250	1945495	0.000000
9	1953123	1	0.000000	9	1953125	974655	0.000000
10	976559	2	0.000000	10	976563	487804	0.000000
11	488262	10	0.000000	11	488282	244021	0.000000
12	244099	21	0.000000	12	244141	122040	0.000000
13	121837	117	0.000000	13	122071	61027	0.000000
14	60233	401	0.000000	14	61036	30515	0.000000
15	26066	2226	0.000000	15	30518	15258	0.000000
16	10146	2556	0.000000	16	10544	7629	0.000000
17	5066	1282	$5.1391 \cdot 10^{-15}$	17	7630	3814	$3.7038 \cdot 10^{-97}$
18	2564	625	$1.6427 \cdot 10^{-2}$	18	3815	1907	$8.8336 \cdot 10^{-13}$
19	1268	320	0.597796	19	1908	953	$3.1137 \cdot 10^{-2}$
20	625	164	0.937945	20	954	476	0.648420
21	314	81	0.992045	21	477	238	0.947287
22	157	41	0.998989	22	239	119	0.993225
23	72	24	0.999875	23	120	59	0.999150
24	41	9	0.999984	24	60	29	0.999895
25	21	4	0.999998	25	30	14	0.999987
26	10	2	0.999999	26	15	7	0.999998
27	4	2	1.000000	27	8	3	0.999999
28	4	0	1.000000	28	4	1	1.000000
29	2	0	1.000000	29	2	0	1.000000

Table A.4: $n = 10^9$

Appendix B

Lower bounds for π_2

We only consider using Precomputation Method 2 when $ke < \frac{n-k+1}{2}$. If there are k and e values that violate this inequality, $\pi_2^{\times}$ and π_2^{+} are undefined.

d	k	e	$\pi_2^{\times}$
1	47	1	–
2	21	2	–
3	7	3	0.645258
4	5	1	0.902480
5	4	0	1.000000
6	2	0	1.000000

d	k	e	π_2^{+}
1	50	12	–
2	25	9	–
3	10	5	–
4	4	3	0.781712
5	4	1	0.921592
6	2	0	1.000000

Table B.1: $n = 100$

d	k	e	$\pi_2^{\times}$
1	497	1	–
2	245	2	–
3	107	9	–
4	29	17	–
5	14	9	0.775452
6	9	3	0.947174
7	5	1	0.990025
8	3	0	1.000000
9	2	0	1.000000

d	k	e	π_2^{+}
1	500	125	–
2	250	93	–
3	125	54	–
4	63	29	–
5	32	15	0.375510
6	16	7	0.797584
7	8	3	0.952926
8	4	1	0.992016
9	2	0	1.000000

Table B.2: $n = 1000$

d	k	e	$\pi_2^\times$
1	499997	1	–
2	249994	3	–
3	124987	6	–
4	62472	14	–
5	31190	30	–
6	15498	63	–
7	7544	134	–
8	3337	285	–
9	978	488	0.384769
10	492	242	0.788044
11	245	122	0.941963
12	125	60	0.985110
13	65	29	0.996236
14	34	14	0.999048
15	14	8	0.999776
16	8	4	0.999936
17	5	1	0.999990
18	3	0	1.000000
19	2	0	1.000000

d	k	e	π_2^+
1	500000	125000	–
2	250000	93750	–
3	125000	54687	–
4	62500	29296	–
5	31250	15136	–
6	15625	7690	–
7	7813	3875	–
8	3907	1945	–
9	1954	974	–
10	977	487	0.385899
11	489	244	0.787647
12	245	122	0.941963
13	123	61	0.985104
14	62	30	0.996286
15	31	15	0.999070
16	16	7	0.999776
17	8	3	0.999952
18	4	1	0.999992
19	2	0	1.000000

Table B.3: $n = 10^6$

d	k	e	$\pi_2^\times$
1	499999992	4	–
2	249999996	2	–
3	124999986	7	–
4	62499967	16	–
5	31249929	35	–
6	15624850	75	–
7	7812181	159	–
8	3905832	209	–
9	1952240	442	–
10	974689	937	–
11	484312	1985	–
12	235732	4204	–
13	104261	8905	–
14	30554	15241	0.394015
15	15333	7592	0.792297
16	7644	3807	0.943459
17	3820	1905	0.985551
18	1908	953	0.996370
19	954	477	0.999090
20	483	235	0.999773
21	242	117	0.999943
22	119	60	0.999985
23	62	29	0.999996
24	31	14	0.999999
25	13	8	0.999999
26	8	3	1.000000
27	4	2	1.000000
28	4	0	1.000000
29	2	0	1.000000

d	k	e	π_2^+
1	499999992	125000002	–
2	249999996	93750000	–
3	124999998	54687500	–
4	62499999	29296875	–
5	31250000	15136718	–
6	15625000	7690429	–
7	7812500	3875732	–
8	3906250	1945495	–
9	1953125	974655	–
10	976563	487804	–
11	488282	244021	–
12	244141	122040	–
13	122071	61027	–
14	61036	30515	–
15	30518	15258	0.394039
16	15259	7629	0.792293
17	7630	3814	0.943459
18	3815	1907	0.985554
19	1908	953	0.996370
20	954	476	0.999092
21	477	238	0.999773
22	239	119	0.999943
23	120	59	0.999985
24	60	29	0.999996
25	30	14	0.999999
26	15	7	0.999999
27	8	3	1.000000
28	4	1	1.000000
29	2	0	1.000000

Table B.4: $n = 10^9$

Bibliography

[1] G. Allouche. *7 Well-Known Companies Who Have Moved to the Cloud*. Sept. 2013. URL: `http://www.smartdatacollective.com/gilallouche/145341/7-well-known-companies-have-moved-cloud` (visited on 04/25/2016).

[2] International Data Corporation. *IDC Forecasts Public IT Cloud Services Spending Will Reach $127 billion in 2018 as the Market Enters a Critical Innovation Stage.* Nov. 2014. URL: `http://www.idc.com/getdoc.jsp?containerId=prUS25219014` (visited on 04/25/2016).

[3] Y. Doröz, Y. Hu, and B. Sunar. "Homomorphic AES evaluation using the modified LTV scheme". In: *Designs, Codes and Cryptography* (2015), pp. 1–26.

[4] Y. Doröz and B. Sunar. *Flattening NTRU for Evaluation Key Free Homomorphic Encryption.* Cryptology ePrint Archive, Report 2016/315. 2016.

[5] S.H. Friedberg, A.J. Insel, and L.E. Spence. *Linear Algebra.* Featured Titles for Linear Algebra (Advanced) Series. Pearson Education, 2003.

[6] C. Gentry. "Fully homomorphic encryption using ideal lattices". In: *STOC*. Ed. by Michael Mitzenmacher. ACM, 2009, pp. 169–178.

[7] K. Lauter, M. Naehrig, and V. Vaikuntanathan. *Can Homomorphic Encryption be Practical?* Tech. rep. MSR-TR-2011-61. May 2011.

[8] A. Menezes, S. Vanstone, and P. Van Oorschot. *Handbook of Applied Cryptography.* Boca Raton, FL, USA: CRC Press, Inc., 1996.

[9] P. Paillier. In: *Advances in Cryptology — EUROCRYPT '99: International Conference on the Theory and Application of Cryptographic Techniques Prague, Czech Republic, May 2–6, 1999 Proceedings.* Ed. by J. Stern. Berlin, Heidelberg: Springer Berlin Heidelberg, 1999. Chap. Public-Key Cryptosystems Based on Composite Degree Residuosity Classes, pp. 223–238.

[10] R.L. Rivest, L. Adleman, and M.L. Dertouzos. "On Data Banks and Privacy Homomorphisms". In: *Foundations of Secure Computation, Academia Press* (1978), pp. 169–179.

[11] S.A. Vanstone and P.C. van Oorschot. In: *An Introduction to Error Correcting Codes with Applications*. Vol. 71. The Springer International Series in Engineering and Computer Science. Springer US, 1989. Chap. Error Correction Techniques and Digital Audio Recording, pp. 237–266.