Approaches to Homomorphic Encryption Using Polynomial Rings and the Chinese Remainder Theorem

Benjamin T. LeVeque

May 3, 2013

Contents

Chapter 0

Acknowledgements

The writing of this thesis has been a deeply rewarding and exciting experience. I am deeply indebted to my advisor, Prof. Jeffrey Hoffstein, for giving me considerable freedom to explore these concepts while always being able to point me in the right direction. The chance to work with him the past year has been so much fun, and I am very appreciative of his patience and humor as well as his instruction. I am very grateful for the help of Prof. Jill Pipher and Prof. Joseph Silverman, who offered helpful pointers throughout the project (and co-authored the book that introduced me to this subject). I would also like to thank Prof. Mike Stillman for a very instructive conversation on Gröbner bases and for pointing me in the direction of the Macaulay2 computer algebra system. I am very thankful also for the constant support from my family and friends, whose encouragement made the writing of this thesis much more manageable and who I can always count on to listen to my ideas, even when they make no sense. Finally, I would like to acknowledge the staff, faculty, and my fellow students in the math department for making the past four years so enjoyable.

Chapter 1

Introduction and background

1.1 Introduction

In an age of ubiquitous computing, digital security is an important aspect of everyday life. Cryptography is already a pervasive concept, providing privacy for everything from email communication to financial transactions. As computations involving large data sets become more expensive and time-consuming, and as the need for data storage increases, cloud computing has rapidly become a platform to which people and institutions alike turn for their computational needs. With this rise in popularity comes the need for a new class of security protocols that allow data to be stored in encrypted form yet manipulated in a meaningful way by third parties. In this thesis, we consider two such cryptographic schemes. We discuss the motivation behind them, analyze their security, and present data regarding both their security and efficiency. We also provide implementations of the systems in question in C++ and explain the code's design.

Two goals in the very close periphery throughout the development of this project have been reproducibility and accessibility. In this vein, the implementations produced and the data generated are posted in a public repository on Github at https://github.com/bleveque/HomEnc, and a website accompanying the project can be found at http://btleveque.org/homenc.php.

1.2 Introduction to cryptographic concepts and terminology

Before explaining and analyzing specific schemes, we give a brief background on some common cryptographic concepts. A *cryptosystem* is a collection of procedures that allow a user to securely map a *message* (also called a *plaintext*) m to a *cipertext* c such that the reverse map can only be easily computed given some private information, called a *secret key*. The map taking a message to a ciphertext is referred to as an *encryption function* and will be denoted by e, while the reverse map is referred to as a *decryption function* and will be denoted by d. The process of determining the secret key is known as *key generation*, and the key generation function will be denoted by KG. The messages lie in some space, referred to as the *message space* ($\mathcal{M}$), and the ciphertexts lie in a (possibly) different space, referred to as the *ciphertext space* ($\mathcal{C}$), so we have

$$\begin{aligned} e &: \mathcal{M} \longrightarrow \mathcal{C} \\ d &: \mathcal{C} \longrightarrow \mathcal{M} \\ (d \circ e) &= id_{\mathcal{M}} \end{aligned}$$

A cryptosystem is then given by a collection

$$\{\mathcal{M}, \mathcal{C}, KG, e, d\}$$

This information together tells us exactly where our messages are coming from and how to encrypt and decrypt them.

To illustrate the construction above with a brief example, consider the cryptosystem defined by the following elements:

$$\begin{aligned}
&\mathcal{M} = \mathbb{Z} \\
&\mathcal{C} = \mathbb{Z} \\
&KG \text{ gives a random integer } k > 1 \\
&e : m \mapsto km \\
&d : c \mapsto c/k
\end{aligned}$$

This simple scheme encrypts an integer message m by taking its product with the secret key k. The resulting ciphertext can be decrypted by dividing by k:

$$d(e(m)) = d(km) = m$$

It is therefore well-defined to encrypt and decrypt a message. This scheme is not very secure, since the value of k can be easily discovered by examining several message/ciphertext pairs, but it is illustrative of the basic ideas.

A lingering question might be "my message is a regular sentence! how can the message space $\mathbb{Z}$ be of any use to me?" The answer comes in the form of an invertible *encoding function*. An encoding function is not meant to provide any extra security to our scheme; its sole purpose is to transform real messages into a format we can use in our encryption function. For example, we could encode a word by sending each letter to a number:

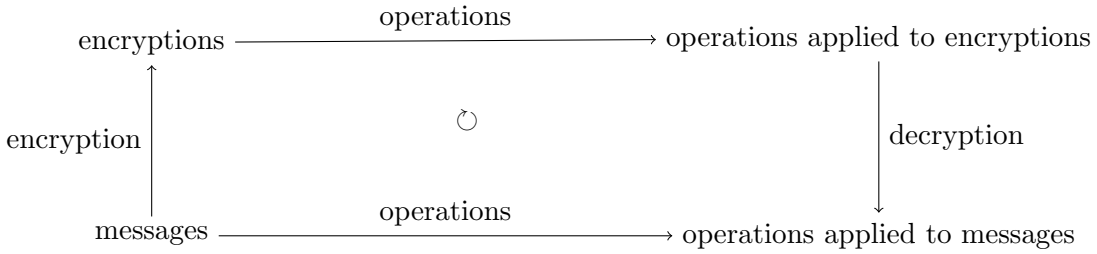
$$\begin{aligned}
&a \mapsto 1 \\
&b \mapsto 2 \\
&\vdots \\
&z \mapsto 26
\end{aligned}$$

so the word "crypto" would be encoded as the sequence of numbers 3-18-25-16-20-15. We could then use our encryption function on each of these numbers. When we decrypt the result, we will get the sequence of numbers back, and then we must perform the inverse of the encoding function to recover our word as a sequence of letters c-r-y-p-t-o.

1.3 Introduction to homomorphic encryption

The idea of a homomorphic cryptosystem extends the ideas presented above by adding an additional requirement to our definitions. To motivate this requirement, imagine you have a large collection of data points and want to compute their mean or standard deviation. You have have two options:

you can either do this computation yourself at a potentially high computational cost, or you can delegate this work to another party and risk losing privacy by exposing your data. The goal of fully homomorphic encryption is to minimize both potential costs by allowing secure delegation. In practice, this might mean being able to store data on the Cloud in an encrypted form such that a third party could still manipulate the data in a meaningful way. By meaningful, we simply mean that after they operate on our data, we can decrypt the result and obtain exactly the value their operations would have achieved on our original plaintext data. The following diagram illustrates this process:



Homomorphic encryption ensures that we achieve the same result by traversing this diagram in either direction starting from our messages (i.e. the diagram is *commutative*). More technically, by "meaningful" we mean that any combination of sums and products of encryptions decrypt to the corresponding sums and products of the original, unencrypted data (we call such a combination an *arithmetic circuit*, or just a *circuit*). If this condition holds, data could be encrypted, operated upon, and decrypted in a completely well-defined manner. What we are looking for, then, is an encryption function that is a *ring homomorphism* from the message space $\mathcal{M}$ to the ciphertext space $\mathcal{C}$. Recall that a ring homomorphism is a map $\sigma : \mathcal{M} \to \mathcal{C}$ such that for any $x, y \in \mathcal{M}$,

$$\sigma(x) + \sigma(y) = \sigma(x + y)$$

$$\sigma(x)\sigma(y) = \sigma(xy)$$

If our encryption function e, then, is a ring homomorphism, we could add or multiply two encryptions and the result would be an encryption of the sum or product of the corresponding plaintexts. A cryptosystem employing such an encryption function is said to give *fully homomorphic encryption* (FHE). If the encryption function in question is homomorphic on most inputs, but is not guaranteed to give well-defined decryption after the application of every circuit, the scheme is said to provide *somewhat homomorphic encryption*. Such a scheme might arise if the decryption function requires its input to be of a certain size, which the application of large circuits might exceed. For motivation, consider the following toy example.

1.3.1 Example

First, we will construct a somewhat homomorphic encryption scheme, and then we will attempt to extend it to be fully homomorphic. Consider the message space $\mathcal{M} = \mathbb{Z}_{\geq 0}$ and choose as a secret key some prime number p. We will encrypt a message $m \in \mathcal{M}$ by setting $e(m) = m + ap$ for some

random integer a. Decryption is then performed by reducing modulo p. As long as the message m is less than p, this scheme is perfectly well defined, since

$$\begin{aligned} d(e(m)) &= m + ap \pmod{p} \\ &= m \end{aligned}$$

The scheme is additively homomorphic for messages m_1 and m_2 if $m_1 + m_2 < p$ because we have

$$\begin{aligned} d(e(m_1) + e(m_2)) &= m_1 + m_2 + (a_1 + a_2)p \pmod{p} \\ &= m_1 + m_2 \end{aligned}$$

and multiplicatively homomorphic if $m_1 m_2 < p$, since

$$\begin{aligned} d(e(m_1)e(m_2)) &= m_1 m_2 + (m_1 a_2 + m_2 a_1 + a_1 a_2 p)p \pmod{p} \\ &= m_1 m_2 \end{aligned}$$

However, if any of the messages above satisfy $m \geq p$, then $d(e(m))$ will no longer equal m, but will be the reduction of m modulo p. We can attempt to avoid this problem by encrypting only messages less than p, but once we start considering circuits applied to encryptions, this idea shows its flaws. For example, we might want the sum of two encryptions of $p - 1$ to decrypt to $2p - 2$, but it would decrypt instead to

$$2p - 2 \pmod{p} = p - 2$$

Another solution is to recognize the limitations of the system and include as a part of the system's protocol a bound on the number of additions and multiplications that can be performed on encryptions before the result should be decrypted.

To make this scheme fully homomorphic, we could set $\mathcal{M} = \mathbb{Z}/p\mathbb{Z}$, since in this case, reducing modulo p during decryption is consistent with the structure of the message space. However, it is perhaps unnatural for messages to live in $\mathbb{Z}/p\mathbb{Z}$, since when we compute circuits such as the computation of standard deviations, we don't want to reduce modulo p at any point during the computation. In this case, it may be best to accept the limitations of the somewhat homomorphic scheme above and work within its framework.

It should be mentioned, as well, that the scheme above is just a "toy" example because as in the case of the example on page 4, it is very insecure. Suppose an eavesdropper asked us to encrypt the message $m = 0$ using this system. The result would be a multiple of p. If we were asked to perform many such encryptions, the eavesdropper would end up with a collection of multiples of p. With this information, he could very quickly compute the greatest common divisor of these values using the Euclidean Algorithm, and the result would likely be p itself. Using p, the eavesdropper could then decrypt any message he pleased by simply reducing it modulo p. This type of attack—using encryptions of the message 0 to reveal an essential feature of the structure of the scheme itself—will appear again, and is a necessary threat to consider when formulating encryption schemes.

Chapter 2

Schemes under investigation

2.1 Introduction to present work

In his 2009 dissertation [8], Craig Gentry presented the first secure fully homomorphic encryption scheme. His construction was based on the theory of ideal lattices (ideals in quotient polynomial rings), and it represented a major breakthrough as a proof-of-concept and started a rapid-pace movement towards creating an *efficient* fully homomorphic scheme. This thesis will present two cryptosystems that show promise as either somewhat or fully homomorphic encryption schemes and explore their security and efficiency.

The first, which is formulated entirely over the integers (and quotient rings thereof), relies on the hardness of making N consecutive correct choices, where each choice is between two values. We will refer to this as "choice-based encryption" (CBE). The second, which is formulated over multivariate polynomial rings, relies on the hardness of the ideal membership problem (IMP) in this setting. Similar schemes in the past—sometimes referred to as *Polly Cracker* schemes after their introduction by Koblitz and Fellows in [7]—have been formulated, often over finite fields. The cryptosystem presented in section 2.3 below can be formulated over general rings, though we focus on an implementation over the integers. Translating this to an implementation over $\mathbb{Z}/p\mathbb{Z}$ is a matter of reducing coefficients, so while it is easy to reformulate in this setting, the security of the scheme may be compromised, as we will see.

Finally, we note that both schemes are presented as *symmetric* schemes, meaning that the secret key information is shared by the encryptor and the decryptor. While such schemes do not enjoy the same flexibility as *asymmetric* schemes (in which anyone can encrypt a message meant to be decrypted by a certain recipient, so the knowledge of keys is asymmetric), they are sufficient for many purposes, including some of the natural applications of homomorphic encryption. For example, symmetric encryption is well-suited to the situation in which the encryptor has a collection of data she wants to store on the cloud and later retrieve, since only one person needs to know the secret information necessary to both encrypt and decrypt.

2.2 Choice-based encryption

2.2.1 Scheme

We will first describe the scheme in question and subsequently analyze the motivation behind its construction and its cryptographic properties. The choice-based cryptosystem is defined using the following procedures:

- Key generation (KG)
  1. Pick a prime P; the message space is $\mathcal{M} := \mathbb{Z}/P\mathbb{Z}$
  2. Pick a lower bound $M \geq 0$ on the number of operations that our scheme will allow
  3. Pick an integer K; this is the number of ways we will be able to mask a given message
  4. Pick $N \in \mathbb{Z}$ so that 2^N is sufficiently large
  5. Pick primes $\{p_i\}_{i=1}^N$ such that $((K+1)P)^{M+1} < \prod p_i$ and $p_i \neq P$ for all i
  6. Pick primes $\{q_i\}_{i=1}^N$ such that $q_i \neq P$ for all i
  7. Return private key $(N, P, M, K, \{p_i\}, \{q_i\})$ and public key $(N, M, \{p_i q_i\})$
- Encryption (e)
  1. Pick a message $m \in \mathcal{M}$
  2. Pick random integers $\{a_i\}_{i=1}^N$
  3. Pick a random positive integer $k < K$
  4. Return $e(m) := (m + kP + a_i p_i \pmod{p_i q_i})_{i=1}^N$; the ciphertext space is $\mathcal{C} := \prod_{i=1}^N (\mathbb{Z}/p_i q_i \mathbb{Z})$
- Decryption (d)
  1. For all $i = 1, \ldots, N$, reduce the ith component of the encryption modulo p_i
  2. Run the Chinese Remainder Theorem with moduli $\{p_i\}$ on the resulting components, which gives $m + kP \pmod{\prod p_i}$
  3. Reduce the result modulo P to retrieve m
- Addition of ciphertexts ($+$)
  1. The sum of two ciphertext vectors is their component-wise sum in $\prod(\mathbb{Z}/p_i q_i \mathbb{Z})$
- Multiplication of ciphertexts ($\cdot$)
  1. The product of two ciphertext vectors is their component-wise product in $\prod(\mathbb{Z}/p_i q_i \mathbb{Z})$

2.2.2 Motivation

As mentioned above, this scheme is formulated entirely over quotient rings of the integers: the message space is $\mathbb{Z}/P\mathbb{Z}$, where P is secret, the ciphertext space is the product of the quotient rings $\mathbb{Z}/p_i q_i \mathbb{Z}$, and the Chinese Remainder Theorem in step two of the decryption function returns a result in $\mathbb{Z}/\prod p_i \mathbb{Z}$. Since we are working strictly with integers (and quotient rings of integers), operations on ciphertexts and messages are easy to describe and implement in code. Simplicity

of design is one of the motivations behind the construction, and indeed, it is possible to take p_i and q_i to be quite small as long as our security parameter N is large enough. The underlying hardness assumption is also quite easy to state: given N choices, each between two options, can an eavesdropper identify all N correct options? With a brute-force algorithm, this will take 2^N trials, which is computationally infeasible for large N. Changing N, therefore, affects the hardness of the underlying problem, but increasing its value also increases the size of ciphertexts (by adding components to the encryption vectors). It should be noted, too, that choosing even $N-1$ correct values and a single wrong value for the $\{p_i\}$ (say we choose q_j instead of p_j) can give a drastically wrong result, since we would not be completely removing the multiple of p_j in step 1 of decryption, and step 2 of decryption would run the Chinese Remainder Theorem on the wrong collection of primes.

Before proving the correctness and analyzing the characteristics of this system, we will also give some explanation for the other components of the scheme, beginning with the public and private keys. The prime P is the size of our message space. The primes $\{p_i\}$ and $\{q_i\}$ can be chosen arbitrarily as long as they satisfy the constraints in steps 5 and 6 of key generation above. The sizes of the primes p_i and q_i are up to the user; larger primes allow for a larger message space, but they also lead to larger ciphertexts. We note, too, that it is also possible to create a larger message space by increasing the size of N. This will also affect the size of a cipher text, but it has the potential benefit of keeping the values in each component relatively smaller.

The integers M and K together represent the number of ways we can mask (i.e. add *noise* to) a message with multiples of P before we exceed $\prod p_i$. In other words, we are allowing up to K multiples of P to be added as noise to each component of an encryption, and we are allowing up to M operations to be performed on ciphertexts. We will discuss the implications of these numbers in section 2.2.6 below. Note that if we did not allow adding of noise, then it would be easy for an eavesdropper to determine the primes p_i from encryptions of zero. For example, consider a collection of r encryptions of zero (recall that they are not the same, since we choose the values a_i randomly in step 2 of encryption):

$$\begin{aligned} e(0) &= (a_{1,1}p_1 \pmod{p_1q_1}, \ldots, a_{1,N}p_N \pmod{p_Nq_N}) \\ &\vdots \\ e(0) &= (a_{r,1}p_1 \pmod{p_1q_1}, \ldots, a_{r,N}p_N \pmod{p_Nq_N}) \end{aligned}$$

Since reducing an integer l modulo M is done by adding an appropriate multiple of M to l, we can express the above as:

$$\begin{aligned} e(0) &= (a_{1,1}p_1 + b_{1,1}p_1q_1, \ldots, a_{1,N}p_N + b_{1,N}p_Nq_N) \\ &\vdots \\ e(0) &= (a_{r,1}p_1 + br, 1p_1q_1, \ldots, a_{r,N}p_N + b_{r,N}p_Nq_N) \end{aligned}$$

where the $b_{i,j}$ are the appropriate factors necessary to perform reduction in each component. Now, the ith component of each jth encryption is a multiple of p_i, namely $(a_{j,i} + b_{j,i}q_i)p_i$, so the component-wise GCD of the r encryptions above will likely give us all of the p_i, and the security of the scheme (which is based on the difficulty of identifying the p_i) will be compromised.

Finally, we note that by setting the message space to $\mathbb{Z}/P\mathbb{Z}$ rather than $\mathbb{Z}$, we ensure that even if a sum or product of messages is greater than P, reducing modulo P will give the well-defined sum

or product in the message space. In some applications, we may want to ensure that this reduction never occurs (for example, if we are computing statistical quantities over the integers), but it is at least well-defined if it does.

2.2.3 Correctness of encryption/decryption

In order for the choice-based scheme above to be a valid cryptosystem, we need to prove that the composition of encryption with decryption is the identity function on the message space. Formally, we want:

$$d \circ e \equiv id_{\mathbb{Z}/P\mathbb{Z}}$$

We prove this below:

Theorem 1 (Correctness of encryption/decryption for CBE)**.** *If $m \in \mathbb{Z}/P\mathbb{Z}$, then $d(e(m)) = m$.*

Proof. Let $(N, \{p_i\}, \{q_i\}, K, P)$ be our secret key. Then

$$e(m) = (m + kP + a_i p_i \pmod{p_i q_i})_{i=1}^{N}$$

for some random a_i and a random $k < K$. Step 1 of decryption performs reduction of each component modulo the corresponding p_i, giving

$$\{m + kP \pmod{p_i}\}_{i=1}^{N}$$

Now, using the Chinese Remainder Theorem in step 2 gives

$$m + kP \pmod{\prod p_i}$$

Since $m < P$ (here, we are abusing notation slightly and identifying the equivalence class $\bar{m} \in \mathbb{Z}/P\mathbb{Z}$ with its minimal representative) and $k < K$, and P was defined such that $(K+1)P < \prod p_i$ (since $M \geq 0$), we have that $m + kP < \prod p_i$, so

$$m + kP \pmod{\prod p_i} = m + kP$$

Now, reducing modulo P in step 3 gives m, since $m \in \mathbb{Z}/P\mathbb{Z}$. □

2.2.4 Correctness of homomorphic operations

We now prove that the scheme is somewhat homomorphic up to the desired M operations. Let us first make the following definitions:

Definition 1 (Primitive encryption)**.** *A* primitive encryption $e(m) \in \mathcal{C}$ *is an encryption on which no operations have been performed.*

Definition 2 (n-circuit)**.** *An* n-circuit *is a sequence of additions and multiplications such that when all multiplications are distributed out, there no more than n total operations in the expression.*

Intuitively, we can think of a primitive encryption as one for which $< K$ multiples of P have been added to each component as noise, as is the case when a message is first encrypted. Now, we will prove correctness:

Theorem 2 (Correctness of homomorphic operations for CBE)**.** *If C is an M-circuit which acts on $n \leq M$ values, $C(m_1, \ldots, m_n) = d(C(e(m_1), \ldots, e(m_n)))$.*

Proof. Consider two encryptions

$$e(m_1) = (m_1 + k_1 P + a_{1,i} p_i \pmod{p_i q_i})_{i=1}^{N}$$
$$e(m_2) = (m_2 + k_2 P + a_{2,i} p_i \pmod{p_i q_i})_{i=1}^{N}$$

Our procedure for addition of ciphertexts tells us that we should perform addition component-wise, so we get

$$\begin{aligned} e(m_1) + e(m_2) &= (m_1 + k_1 P + a_{1,i} p_i \pmod{p_i q_i}) + (m_2 + k_2 P + a_{2,i} p_i \pmod{p_i q_i}) \\ &= (m_1 + m_2 + (k_1 + k_2) P + (a_{1,i} + a_{2,i}) p_i \pmod{p_i q_i}) \end{aligned}$$

Let us assume for the moment that $m_1 + m_2 + (k_1 + k_2)P < \prod p_i$. Then decrypting $e(m_1) + e(m_2)$ gives $m_1 + m_2$, so we have that $e(m_1) + e(m_2) = e(m_1 + m_2)$ (where equality here really means that $e(m_1) + e(m_2)$ is *an* encryption of $m_1 + m_2$, since random numbers are involved in each encryption, so it unlikely that an arbitrary encryption of m_1 plus an arbitrary encryption of m_2 will equal an arbitrary encryption of $m_1 + m_2$). We will return to the question of the size of $m_1 + m_2 + (k_1 + k_2)P$ shortly.

Now consider the product of $e(m_1)$ and $e(m_2)$:

$$\begin{aligned} e(m_1)e(m_2) &= (m_1 + k_1 P + a_{1,i} p_i \pmod{p_i q_i}) \cdot (m_2 + k_2 P + a_{2,i} p_i \pmod{p_i q_i}) \\ &= (m_1 m_2 + (m_1 k_2 + m_2 k_1 + k_1 k_2 P) P + \\ &\qquad ((m_1 + k_1 P) a_{2,i} + (m_2 + k_2 P) a_{1,i} + a_{1,i} a_{2,i} p_i) p_i \pmod{p_i q_i}) \end{aligned}$$

In this case, as long as

$$m_1 m_2 + (m_1 k_2 + m_2 k_1 + k_1 k_2 P) P < \prod p_i$$

we see that $e(m_1)e(m_2)$ will decrypt as $m_1 m_2$ by applying the decryption function. Therefore, $e(m_1)e(m_2)$ is a valid encryption of $m_1 m_2$, so $e(m_1)e(m_2) = e(m_1 m_2)$. Together with our result for addition, this proves that our scheme is somewhat homomorphic, up to the point that the combination of m and P in each component is less than $\prod p_i$.

We now show that this is the case for any M-circuit. Consider the growth of the combination of m and P as we add and multiply encryptions. Primitive encryptions have combinations less than $(K+1)P$, so a sum of two primitive encryptions will have a combination less than $2(K+1)P$ and a product of encryptions will have a combination less than $((K+1)P)^2$. An M-circuit with exclusively multiplications will then have a combination less than $((K+1)P)^{M+1}$, which by our constraints on the system is less than $\prod p_i$. Since multiplications have a greater effect on the size of the combination than additions, we have shown that our scheme is somewhat homomorphic for all M-circuits.

□

2.2.5 Example of choice-based encryption

To illustrate the concepts above, we will carry out a numerical example of encryption, application of circuits, and decryption. For this purpose, we will take our keys to be quite small, letting $N = 2$,

$M = 3$, $K = 3$, and $P = 7$. Running the key generation function gives us

$$\{p_i\} = \{263, 251\}$$
$$\{q_i\} = \{223, 263\}$$

Note that

$$\begin{aligned}((K+1)P)^M &= 21952 \\ &< 66013 \\ &= \prod p_i\end{aligned}$$

Now, suppose our message is $m = 4$. Running the encryption procedure with random values $\{a_i\} = \{11, 13\}$ and $k = 2$ gives

$$\begin{aligned}e(4) &= (4 + kP + a_1 p_1 \pmod{p_1 q_1}, 4 + kP + a_2 p_2 \pmod{p_1 q_1}) \\ &= (2911 \pmod{58649}, 3281 \pmod{66013}) \\ &= (2911, 3281)\end{aligned}$$

Now, running our decryption circuit, we first reduce each ith component by p_i, which gives

$$(2911 \pmod{263}, 3281 \pmod{251}) = (18, 18)$$

We next use the Chinese Remainder Theorem, which trivially gives us

$$m + kP \equiv 18 \pmod{66013}$$

Finally, we reduce modulo $P = 7$ to get

$$18 \equiv 4 \pmod{7}$$

which matches our original message $m = 4$.

Now, we'll give an example of applying a circuit to encryptions. Consider the keys $N = 3$, $M = 3$, $K = 4$, and $P = 11$; the messages $m_1 = 2$, $m_2 = 4$, $m_3 = 9$; and the circuit $C(x_1, x_2, x_3) = x_1 x_2 + x_3$. Since $M = 3$, and this circuit has three operations, these operations should be well-defined. Running key generation gives

$$\{p_i\} = \{97, 67, 89\}$$
$$\{q_i\} = \{107, 79, 127\}$$

Encrypting our three messages (computations omitted for brevity) gives:

$$e(m_1) = (8097, 649, 3072)$$
$$e(m_2) = (8293, 4805, 7791)$$
$$e(m_3) = (4515, 1728, 5037)$$

Now, applying our circuit to the encryptions gives

$$\begin{aligned}C(e(m_1), e(m_2), e(m_3)) &= e(m_1) * e(m_2) + e(m_3) \\ &= (8097 \cdot 8293 + 4515, 649 \cdot 4805 + 1728, 3072 \cdot 7791 + 5037) \\ &= (806, 2596, 10538)\end{aligned}$$

Decrypting this gives

$$d(C(e(m_1), e(m_2), e(m_3))) = 6$$

Applying our circuit to our unencrypted messages likewise gives

$$\begin{aligned} C(m_1, m_2, m_3) &= m_1 m_2 + m_3 \\ &= 6 \end{aligned}$$

(recall that our messages are elements of $\mathbb{Z}/P\mathbb{Z}$). This shows that the diagram on page 5 is indeed commutative for our example.

2.2.6 Analysis of homomorphicity in practice

In practice, we may want to know the most efficient way to increase the number of homomorphic operations supported by our system. Since the parameters of the scheme tell us that

$$\#ops \geq \log_{(K+1)P} \left(\prod_{i=1}^{N} p_i \right) - 1$$

we have several options for parameters to alter. We can decrease K, decrease P, increase N, or increase the values of the p_i. Decreasing P or K affects the security of the system, since we would have either a smaller message space or fewer ways to mask encryptions. On the other hand, increasing the size of each prime p_i will increase the size of the ring $\mathbb{Z}/p_i q_i \mathbb{Z}$ in each component of an encryption, and will therefore cause our arithmetic on encryptions to run more slowly. Increasing N (and maintaining the size of the p_i and q_i) has a similar effect; now we have more components in each encryption, so arithmetic will be less efficient. However, increasing N has the added benefit of increasing the complexity of a brute-force search through all 2^N combinations of p_i and q_i.

Which option should we take in practice? In increasing the number of operations allowed, we certainly do not want to negatively impact the security or flexibility of our scheme, so decreasing K or P is not ideal. It would seem, then, that increasing N might be provide the greatest benefit: increasing the number of multiplications (by increasing the product $\prod p_i$) while also increasing the complexity of brute-force search. Analyzing this problem further would be an interesting application of the implementations outlined in chapter 3.

An additional concern is that our message space may not realistically resemble $\mathbb{Z}/p\mathbb{Z}$, for example if the information we want to encode is a collection of integer values, and we want to compute the product of these integers, we expect the result of performing the encrypted operation to be the true product, even if it exceeds P. As soon as the results of computations exceed P, the scheme fails to return the correct value to the user, so practically speaking, we want our messages to be much smaller than P, which is in turn much smaller than $\prod p_i$. Precisely, this means that requiring that any message m satisfy $m < \sqrt[M+1]{P}$ guarantees that products of messages will always decrypt properly, since M is the number of allowed multiplications from above. In practice, this means that if we want our operations to remain in the integers, we should set $\mathcal{M} = \mathbb{Z}_{< \sqrt[M+1]{P}}$.

A desirable aspect of any cryptosystem is that the size of any ciphertext is not excessively large relative to the size of its corresponding plaintext. In this system, including the considerations in

the paragraph above, a message m has worst-case size $|m| = \sqrt[M+1]{P}$, while a ciphertext has size

$$\sum_{i=1}^{N} |(\mathbb{Z}/p_i q_i \mathbb{Z})| = \sum_{i=1}^{N} p_i q_i$$

Note that if we did not reduce each component modulo $p_i q_i$, our ciphertexts could grow quite rapidly. Also, since we're reducing modulo $p_i q_i$, we do not experience any ciphertext expansion when we apply circuits to ciphertexts. This is a major benefit of this cryptosystem: despite the limitation on the number of operations we can perform on ciphertexts, we can perform all of these operations without worrying about ciphertext growth.

2.2.7 Security

We now address the security of this scheme. As we mentioned above, masking our messages with multiples of P in each component avoids a greatest common divisor attack on encryptions of $m = 0$. This is because encryptions of $m = 0$ have the form

$$e(0) = (kP + a_i p_i \pmod{p_i q_i})_{i=1}^{N}$$

Since P is coprime to $p_i q_i$ (it is a prime which is not equal to either p_i or q_i), the components of this encryption should have no particular form; in other words, P is a unit in each component, so its multiples fill each $\mathbb{Z}/p_i q_i \mathbb{Z}$. Note that if $K < p_i q_i$, then the possible multiples kP for $k < K$ in each component may not completely fill the corresponding $\mathbb{Z}/p_i q_i \mathbb{Z}$, but the multiples that do appear will be indistinguishable from multiples of any other unit u. This means that encryptions of $m = 0$ are indistinguishable from other encryptions.

Intuitively speaking, the fact that we have incorporated *noise* into our encryption function ensures that it is difficult to identify the underlying structure of our scheme. Breaking the scheme, then, should depend on being able to solve some sort of approximation problems (for an example, called the *approximate greatest common divisor problem*, see [18]). However, the fact that our noise is not required to be small, per se (since, for example, a large N allows us to choose p_i relatively small compared to P), means that we are not in an approximate greatest common divisor situation.

The security of this scheme then seems primarily based on the difficulty of choosing each p_i in sequence. The difficulty of this task is quantifiable to an average and worst case of $\mathcal{O}(2^N)$.

2.3 Using multivariate polynomial rings

Another approach to homomorphic encryption is to use the properties of arithmetic in polynomial rings. Perhaps the most natural thing to try in this setting is the following algorithm: choose a principal ideal $(f) \subset \mathbb{Z}[x]$ and encrypt an integer message m by adding a random element $af \in (f)$, so $e(m) = m + af$. Then to decrypt, we simply reduce modulo the polynomial f. This process is homomorphic because ideals are closed under addition and multiplication:

$$\begin{aligned} e(m_1) + e(m_2) &= m_1 + m_2 + (a_1 + a_2)f \\ &= e(m_1 + m_2) \end{aligned}$$

and

$$\begin{aligned} e(m_1)e(m_2) &= m_1m_2 + (m_2a_1 + m_1a_2 + a_1a_2)f \\ &= e(m_1m_2) \end{aligned}$$

Note that our definition of equality above is somewhat loose; since $e(m_1+m_2)$ encrypts by choosing a *random* multiple of f, it may not be exactly $(a_1+a_2)f$ (and likewise for the case of multiplication). The notion of equality we adopt, then, is a notion of coset equality: $e(m_1) + e(m_2)$ is in the same coset of (f) as $e(m_1 + m_2)$, so they will decrypt identically, and the same is true for $e(m_1)e(m_2)$ and $e(m_1m_2)$.

While this scheme illustrates the ideas we will use later, it is not very secure on its own. For example, suppose an eavesdropper, Eve, asks us to encrypt $m = 0$ several times. The resulting ciphertexts would be a collection

$$\{a_1f_1, a_2f_2, \ldots, a_nf_n\}$$

Now, if Eve takes the greatest common divisor of the elements in this collection (note that this is very similar to the attack employed to break the toy example in section 1.3), there is a very high chance that the result will be f itself or a small multiple of f. With f in hand, Eve could then decrypt any message she pleased, and the scheme would be compromised.

The scheme above was easily broken because it relied on a fairly easy problem: given a collection of polynomials in a principal ideal, find a generator for the ideal. This is easily found because we can compute fairly efficiently the greatest common divisor of two polynomials in $\mathbb{Z}[x]$. This allows us to easily determine, in particular, further encryptions of $m = 0$, since we can reduce any encryption modulo the generator f we found and check if the result is 0. This is a basic case of a more general problem.

Definition 3 (Ideal Membership Problem (IMP))**.** *Given a ring R, a set of elements $\{r_1, \ldots, r_n\} \in R$, and an element $f \in R$, determine whether $f \in (r_1, \ldots, r_n) \subset R$.*

If the structure of R is simple, then this problem is correspondingly easy, as we saw above in the case of our generators being multiples of a single polynomial $f \in \mathbb{Z}[x]$. However, if we look instead at multivariate polynomial rings such as $\mathbb{Z}[x, y]$ or $\mathbb{Z}/p\mathbb{Z}[x, y]$, the problem might not be so simple. To give an indication of the increased complexity of multivariate polynomial rings, consider the seemingly simple problem of reduction modulo a polynomial $f \in \mathbb{Z}[x, y]$, which is essentially an attempt to reduce the degree of a polynomial by subtracting multiples of f. However, which degree do we want to reduce? We could mean either the degree of x or that of y, and choosing one over the other turns out to affect the result of reduction. Consider, for example

$$\begin{aligned} g &= x + 4y^2 \\ f &= x - 2y \end{aligned}$$

If we want to reduce g modulo f and choose the *ordering* $x > y$ (i.e. we prioritize reducing the degree of x), then

$$\begin{aligned} g \pmod f &= g - f \\ &= 4y^2 + 2y \end{aligned}$$

which has x-degree 0. On the other hand, with the ordering $y > x$, we have

$$\begin{aligned} g \pmod f &= f - g^2 + 2xg \\ &= x^2 + x \end{aligned}$$

which has y-degree 0. Where reduction was straightforward in the single-variable case, it is now convoluted by the structure of the multivariate case. Using multivariate polynomial rings as the foundation of a cryptosystem might provide increased security, then, since the complexity of the underlying ring has increased. It is with this motivation that we formulate the following scheme.

2.3.1 Scheme

The scheme goes as follows:

- Key generation (KG)
  1. Set $\mathcal{M} := \mathbb{Z}$ and $\mathcal{C} := \mathbb{Z}[x, y]$
  2. Pick a degree bound D and a coefficient bound B
  3. Pick a random number $z_0 < B$
  4. Pick a random polynomial f with total degree less than or equal to D and coefficients less than B
  5. Pick a random polynomial g' of total degree less than or equal to $D - 1$ and coefficients less than B
  6. Set $g := (y - z_0)g'$
  7. Return private key (f, g, z_0)
- Encryption (e)
  1. Pick a message $m \in \mathbb{Z}$
  2. Pick random polynomials a and b which respect the degree and coefficient bounds
  3. Return $e(m) := m + af + bg$
- Decryption (d)
  1. Evaluate the encryption at z_0
  2. Reduce the result modulo $f(x, z_0)$, a single-variable polynomial, and return the result
- Addition of ciphertexts ($+$)
  1. Given two ciphertexts in $\mathbb{Z}[x, y]$, their sum is simply their sum as multivariate polynomials
- Multiplication of ciphertexts ($\cdot$)
  1. Given two ciphertexts in $\mathbb{Z}[x, y]$, their product is simply their product as multivariate polynomials

2.3.2 Explanation

Before analyzing the security properties of this scheme, we give a brief summary of its procedures. Key generation uses degree and coefficient bounds to create a secret root z_0 and secret polynomials f and g, where g vanishes on the line $y = z_0$. The fact that g vanishes at (x, z_0) for all $x \in \mathbb{Z}$ allows us to evaluate an encryption at (x, z_0) to eliminate the bg term completely. We are then in a single-variable setting (evaluating at (x, z_0) is essentially a map $\mathbb{Z}[x, y] \to \mathbb{Z}[x]$), and can easily reduce modulo $f(x, z_0)$. It is therefore straightforward to perform encryption and decryption, since we are only performing simple polynomial arithmetic, evaluating polynomials, and reducing modulo one-variable polynomials. We prove below that this gives a well-defined encryption scheme. We note also that the degree and coefficient bounds can be chosen as to make the underlying problem (discussed in section 2.3.7 below) as complex as desired (though larger parameters will be a detriment to the efficiency of the scheme). In chapter 3, we discuss the problem of choosing appropriate parameters.

One benefit of using polynomials to encrypt integer messages, as we will prove below, is that since our message is zero-degree, there is no limit on the number of additions and multiplications we can perform.

2.3.3 Correctness of encryption/decryption

We will now prove the well-definition of encryption and decryption.

Theorem 3 (Correctness of encryption/decryption for multivariate encryption)**.** *If $m \in \mathbb{Z}$, then $d(e(m)) = m$.*

Proof. First, encryption gives us

$$e(m) = m + af + bg$$

for some a and b, where we know that $g(x, z_0) = 0$ by construction. In step 1 of decryption, we evaluate at (x, z_0), giving

$$e(m)(x, z_0) = m + a(x, z_0)f(x, z_0)$$

This is a single variable polynomial which can be reduced modulo the polynomial $f(x, z_0)$ (this is known to the decryptor, since f and z_0 are both components of the secret key) to give m, since m is simply a constant value and will not be affected by reduction by any non-constant polynomial. This proves correctness. □

2.3.4 Correctness of homomorphic operations

Now we prove that sums and products of encryptions are valid encryptions of the sums and products of the corresponding plaintexts. Unlike the choice-based scheme, there is no limit on the number of operations that can be performed on ciphertexts (i.e. we can apply an n-circuit for any n in a well-defined manner).

Theorem 4 (Correctness of homomorphic operations for multivariate encryption)**.** *If $m_1, m_2 \in \mathbb{Z}$, then $e(m_1) + e(m_2) = e(m_1 + m_2)$ and $e(m_1)e(m_2) = e(m_1 m_2)$.*

Proof. Note first that two ciphertexts are encryptions of the same value if and only if they lie in the same coset of $(f, g) \subset \mathbb{Z}[x, y]$. The encryptions $e(m_1)$ and $e(m_2)$ are of the form

$$e(m_1) = m_1 + a_1 f + b_1 g$$
$$e(m_2) = m_2 + a_2 f + b_2 g$$

for some $a_i, b_i \in \mathbb{Z}[x, y]$. Then

$$e(m_1) + e(m_2) = m_1 + m_2 + (a_1 + a_2)f + (b_1 + b_2)g$$

This is in the coset $m_1 + m_2$ of (f, g), so it is indeed an encryption of $m_1 + m_2$. This proves that the scheme is additively homomorphic. Consider the product now:

$$e(m_1)e(m_2) = m_1m_2 + (m_1a_2 + m_2a_1)f + ((m_1 + a_1f)b_2 + (m_2 + a_2f)b_1)g$$

This is in the coset m_1m_2 of (f, g), so it is an encryption of m_1m_2. This proves that the scheme is multiplicatively homomorphic as well. □

At no point in the proof above does the size of either message affect the ability of the scheme to encrypt and decrypt properly; the scheme is fully homomorphic in the sense that arbitrary-depth circuits can by applied homomorphically.

2.3.5 Example of multivariate encryption

We will now work out an example using the multivariate encryption scheme above. Let us choose $D = 2$ and $B = 10$. Running the key generation step, we get

$$\begin{aligned} z_0 &= 6 \\ f &= 4xy + 6y + 1 \\ g &= y^2 + 3y - 54 \end{aligned}$$

Note that it is coincidence only that g is a polynomial in $\mathbb{Z}[y]$. Also, its constant coefficient is larger than our coefficient bound! The reason for this transgression is that g is generated as a product of a random polynomial (which does adhere to the degree bound) and the polynomial $(y - z_0)$, so g itself may have coefficients as large as B^2.

Now, suppose we want to encrypt $m = 1024$. Running our encryption procedure with random polynomials

$$\begin{aligned} a &= 5xy + x + y + 5 \\ b &= 3xy + 8x + 3y + 1 \end{aligned}$$

we get

$$\begin{aligned} e(m) &= m + af + bg \\ &= 20x^2y^2 + 3xy^3 + 4x^2y + 75xy^2 + 3y^3 - 107xy + 52y^2 - 431x - 122y + 975 \end{aligned}$$

Now, we run our decryption circuit by first plugging in $z_0 = 6$, which gives

$$\begin{aligned} e(m)(x, z_0) &= 20x^2(6^2) + 3x(6^3) + 4x^2(6) + 75x(6^2) + 3(6^3) - 107x(6) + 52(6^2) - 431x - 122(6) + 975 \\ &= 744x^2 + 2275x + 2763 \end{aligned}$$

Finally, we reduce modulo $f(x, z_0)$ to get

$$\begin{aligned} e(m)(x, z_0) \pmod{f(x, z_0)} &= (744x^2 + 2275x + 2763) \pmod{24x + 37} \\ &= 1024 \end{aligned}$$

This is indeed our original message, which shows that encryption/decryption is well-defined for this example.

We now give an example of a circuit applied to three encryptions (in fact, the same circuit as we used in section 2.2.5), namely $C(x_1, x_2, x_3) = x_1x_2 + x_3$. Suppose we want to encrypt $m_1 = 123$, $m_2 = 234$, and $m_3 = 345$ and apply the circuit C to the encryptions. We show this process below (omitting some computations for brevity). First, running key generation gives

$$\begin{aligned} f &= 7xy + 5x + 6y + 5 \\ g &= 2xy - 14x + 3y - 21 \\ z0 &= 7 \end{aligned}$$

Now, encrypting our messages gives:

$$\begin{aligned} e(m_1) &= 42x^2y^2 - 42x^2y - 36x^2 + 45xy^2 - 42xy - 137x + 51y + 1 \\ e(m_2) &= 24x^2y^2 - 60x^2y + 34xy^2 - 44xy + 2x + 6y^2 + 47y + 222 \\ e(m_3) &= 42x^2y^2 - 15x^2y + 45x^2 + 62xy^2 - 78xy + 57x + 21y^2 - 46y + 343 \end{aligned}$$

Applying the circuit C to these encryptions gives

$$\begin{aligned} C(e(m_1), e(m_2), e(m_3)) &= e(m_1)e(m_2) + e(m_3) \\ &= 1008x^4y^4 - 3528x^4y^3 + 1656x^4y^2 + 2160x^4y + 2508x^3y^4 - 6984x^3y^3 - \\ &\quad 60x^3y^2 + 9720x^3y - 72x^3 + 1782x^2y^4 - 462x^2y^3 + 1420x^2y^2 - \\ &\quad 5147x^2y - 8221x^2 + 270xy^4 + 3597xy^3 + 5046xy^2 - 15783xy - \\ &\quad 30355x + 306y^3 + 2424y^2 + 11323y + 565 \end{aligned}$$

As a brief aside, note that in this scheme, ciphertexts grow quite quickly as we apply circuits. We will discuss this in more depth in section 2.3.6 below. Now, decrypting the result above gives

$$\begin{aligned} d(C(e(m_1), e(m_2), e(m_3))) &= C(e(m_1), e(m_2), e(m_3))(x, z_0) \pmod{f(x, z_0)} \\ &= 29127 \end{aligned}$$

Now, applying C to the original messages gives

$$\begin{aligned} C(m_1, m_2, m_3) &= m_1m_2 + m_3 \\ &= 29127 \end{aligned}$$

This shows that applying a circuit is well-defined for this example.

2.3.6 Practical analysis of homomorphicity

Unlike the choice-based scheme above, this scheme as presented suffers from significant ciphertext expansion as circuits with a large number of multiplications are applied. Sums of ciphertexts do not grow in size too quickly, because when two ciphertexts are added, the degree of the result will not have grown. The coefficients roughly double in an addition, but this is negligible when compared

to the result of a multiplication. When two ciphertexts (of comparable degree) are multiplied, the total degree of the result is roughly twice the total degree of the initial ciphertexts. Let d be the maximum of the maximum x-degree and y-degree of the product. Since we are working with two variables, there are roughly d^2 possible terms, each having coefficients larger than those in the initial ciphertexts. For example, the coefficient of the highest degree term will be the product of the corresponding highest-degree coefficients of the original ciphertexts, and the coefficients of smaller degree terms will be sums of such products, leading to very fast coefficient growth.

There are then two issues with homomorphic operations in this scheme: coefficient growth and degree growth. In order to combat coefficient growth, we could imagine setting up our scheme with $\mathcal{C} = \mathbb{Z}/p\mathbb{Z}[x, y]$ for some prime p rather than $\mathcal{C} = \mathbb{Z}[x, y]$. We will discuss the feasibility of this possibility in more depth in section 2.3.7 below, but doing so negatively affects the security of the scheme. To attempt to deal with degree growth, we could consider making an encryption of $m = 0$ public knowledge and then reducing modulo this encryption whenever we want to reduce the degree of a ciphertext (with the caveat that the leading coefficient of the public encryption must divide the leading coefficient of the ciphertext). In general, reducing modulo a polynomial may not reduce both the x- and y-degree of the polynomial it is reducing, but suppose we choose one encryption of zero (i.e. element of (f, g)) which lies entirely in $\mathbb{Z}[x]$ (of degree n) and one which lies entirely in $\mathbb{Z}[y]$ (of degree m). If this were possible, and if its leading coefficient divided any leading coefficient of an encryption, we could reduce the x- and y-degrees of our ciphertext independently, so that the maximum x-degree we would ever encounter would be n, and similarly for the y-degree and m.

Given certain restrictions on f and g, we know that it is possible to find such elements of (f, g) by the theory of resultants, which we now discuss briefly:

Definition 4 (Resultant)**.** *If $h, h' \in \mathbb{Z}[x]$ are given by*

$$h = a_n x^n + \ldots + a_1 x + a_0$$
$$h' = b_m x^m + \ldots + b_1 x + b_0$$

then we define the resultant *$Res(h, h')$ of h and h' to be the determinant of the following $(m+n) \times (m+n)$ matrix:*

$$Res(h, h') = \det \begin{pmatrix} a_n & a_{n-1} & \ldots & \ldots & a_0 & & \\ & a_n & a_{n-1} & \ldots & \ldots & a_0 & \\ & & & \ddots & & & \\ & & & a_n & a_{n-1} & \ldots & \ldots & a_0 \\ b_m & b_{m-1} & \ldots & \ldots & b_0 & & \\ & b_m & b_{m-1} & \ldots & \ldots & b_0 & \\ & & & \ddots & & & \\ & & & b_m & b_{m-1} & \ldots & \ldots & b_0 \end{pmatrix} \begin{matrix} \left.\vphantom{\begin{matrix} a \\ a \\ a \\ a \end{matrix}}\right\} m \\ \left.\vphantom{\begin{matrix} a \\ a \\ a \\ a \end{matrix}}\right\} n \end{matrix}$$

From the definition, it is clear that if h and h' are integer polynomials in one variable, $Res(h, h') \in \mathbb{Z}$. From [13], we also have that there exist polynomials $l, l' \in \mathbb{Z}[x]$ such that

$$hl + h'l' = Res(h, h')$$

Now, consider our secret polynomials f and g. By the above, we know that

$$Res(f(x, 1), g(x, 1)) =: R_x \in \mathbb{Z}$$
$$Res(f(1, y), g(1, y)) =: R_y \in \mathbb{Z}$$

and that there exist $l_{f,x}, l_{g,x} \in \mathbb{Z}[x]$ and $l_{f,y}, l_{g,y} \in \mathbb{Z}[y]$ such that

$$l_{f,x}f(x,1) + l_{g,x}g(x,1) = R_x$$
$$l_{f,y}f(1,y) + l_{g,y}g(1,y) = R_y$$

By replacing the variables y and x, respectively, we can see that

$$l_{f,x}f + l_{g,x}g \in \mathbb{Z}[y]$$
$$l_{f,y}f + l_{g,y}g \in \mathbb{Z}[x]$$

Furthermore, both polynomials above (we will call them *resultant polynomials*) are elements of (f, g). In general, these resultant polynomials will not have leading coefficients which will allow reduction of arbitrary encryptions, so we need to restrict the form of f and g slightly to allow this. Experimentation with different f and g leads to the following:

Lemma 1. *Suppose the leading term of f is x^{d_1}, and the leading term of g is y^{d_2} for some d_1, d_2, and $d_1 > \deg_x(g)$, $d_2 > \deg_y(f)$. Then each resultant polynomial of f and g will have leading coefficient* 1.

Proof. Without loss of generality, we will prove this just for the y-resultant polynomial (that is, the resultant polynomial in $\mathbb{Z}[y]$). Note that with the notation in the definition above, $d_1 = n$ and $d_2 = m$. Consider the determinant of the matrix $M = \{z_{i,j}\}_{i=1,j=1}^{n+m,n+m}$ in the definition, where we now consider the entries to be elements of $\mathbb{Z}[y]$. If the strictly highest-degree term of f is x^{d_1}, then all of the a_n entries on the diagonal will be 1, since the leading term has coefficient $1 \in \mathbb{Z}[y]$, and since the strictly highest-degree term of g is y^{d_2}, the entries b_0 will all be monic d_2-degree polynomials in y, since considering g as a polynomial in $\mathbb{Z}[x][y]$, y^{d_2} is in the constant term. Therefore, the term in the determinant obtained by taking the product of the diagonal elements will have leading term $y^{d_1 d_2}$.

Now, we want to show that deviating from the diagonal gives a collection of terms whose product has smaller y-degree than our diagonal product — since terms in the determinant of M are products of entries in M such that no two share a row or a column, we can look at general collections of such entries. We know that the collective deviation (or vertical "distance") of an entry in the upper triangle of our matrix (i.e. the difference between their row and column indices) is the negative of the deviation of those in the lower triangle. This is because there must be one entry in our product for each row and column of M, so the average deviation must be 0 along the rows and the columns. Since vertical deviation is the additive inverse of horizontal deviation (we subtract the indices in the opposite order), we know that the horizontal deviation of all entries below the diagonal is equal to the vertical deviation of all entries above the diagonal. This tells us that any terms in our determinant come from a product of entries whose collective horizontal deviation is equal to its collective vertical deviation. In terms of the resultant matrix above, horizontal deviation to the left is equivalent to reducing y-degree by at least one, since y^{d_2} being the leading term tells us that the y-degree in the x^1 term must be at most $d_2 - 2$ (so in particular, the first time we move left from the diagonal, we reduce the y-degree by 2), and this reduction continues as the x-degree grows. Vertical deviation corresponds to increasing the y-degree by at most one, since the y-degree can grow as the x-degree in f shrinks. We know by the constraint $d_2 > \deg_y(f)$ that the y-degree of b_0 is strictly larger than that of a_0, so as the y-degree increases with vertical deviation, it still never reaches d_2. Therefore, since the sums of the deviations are equal, and as we deviate horizontally,

we start by subtracting two from the y-degree, it is impossible for a product of any other collection of entries to have y-degree as large as the product along the diagonal. This proves that $y^{d_1 d_2}$ is the leading term of $\det(M) = Res(f, g)$, and it is indeed monic. □

Therefore, if we follow the restriction above on f and g of degree $d_1 d_2$, we will be able to reduce encryptions modulo degree $d_1 d_2$ monic resultant polynomials and achieve the desired degree reduction.

Below, we show an example of computing resultants using Sage [16]:

```
sage: S.<x,y>=ZZ[]
sage: f=0; g=0
sage: for i in range(3):
    for j in range(3):
        f+=floor(random()*10)*x^i*y^(3-j)
        g+=floor(random()*10)*x^(3-i)*y^j
....:
sage: f
9*x^2*y^3 + 8*x^2*y^2 + 8*x*y^3 + 9*x^2*y + 3*x*y^2 + 8*y^3 + 3*x*y + 2*y^2 + 5*y
sage: g
9*x^3*y^2 + 8*x^3*y + 3*x^2*y^2 + 6*x^3 + 8*x*y^2 + x^2 + 5*x*y + 8*x
sage: f+=x^6; g+=y^6
sage: f.resultant(g,x)
y^36 + 5778*y^31 + .... + 11788353*y^2 + 1310720*y
sage: f.resultant(g,y)
x^36 + 8406*x^31 + .... - 3046439*x^2 + 125000*x
```

2.3.7 Security

We now look at the security characteristics of this system. We have seen that the scheme is constructed to rely on the hardness of the Ideal Membership Problem for multivariate polynomial rings. In particular, encryptions of $m = 0$ lie in $(f, g) \subset \mathbb{Z}[x, y]$, so if there is a procedure to efficiently decide whether an element of $\mathbb{Z}[x, y]$ is in the ideal (f, g), we would be able to test whether arbitrary encryptions are encryptions of 0. The ability to do so would be disastrous for the the security of our system; beyond simply being able to test if we are sending the message $m = 0$ (which we could always avoid doing if need be), it allows us to *compare* encryptions. We know that any two encryptions of a message $m \in \mathcal{M}$ are in the same coset of (f, g), so their difference will be in the coset $\bar{0}$ (namely, it will be in (f, g)). Therefore, if an eavesdropper collects several plaintext/ciphertext pairs, she essentially has a table of encryptions to check any future encryptions against. For example, if our message space is small, an eavesdropper could perform a chosen-plaintext attack and ask our encryption scheme to encrypt each of the possible messages, and then when she intercepts any message, she could easily determine its value based on her knowledge and ability to recognize encryptions of zero.

One tool that helps solve the IMP in the setting of polynomial rings is the theory of *Gröbner bases*. Given a set of generators for an ideal, a the idea of a Gröbner basis is to provide a set of generators for the same ideal, but which have other properties that make them ideally suited for computation. We define two separate (but similar) notions of a Gröbner basis below, since both appear in the literature. Note that the definitions below are given in the context of $\mathbb{Z}[x, y]$, but

they are applicable for polynomial rings in arbitrary numbers of variables over general rings. We include the following two definitions to clarify our usage, as there does not seem to be a standard usage in the literature (see [4] for example, where the following two definitions are switched):

Definition 5 (Monomial)**.** *A* monomial *in the variables $x_1, \ldots, x_n$ is an expression of the form $m = x_1^{d_1} \cdots x_n^{d_n}$. The degree of m is then $d_1 + \ldots + d_n$.*

Definition 6 (Term)**.** *A* term *in the variables $x_1, \ldots, x_n$ is an expression of the form $t = cx_1^{d_1} \cdots x_n^{d_n}$, where $c \in \mathbb{Z}$ (in the case of general rings R, $c \in R$).*

Definition 7 (Leading term)**.** *The* leading term *of a polynomial f, denoted $LT(f)$, is its term with largest total degree.*

Definition 8 (Ideal of leading terms)**.** *Given an ideal $I \subset \mathbb{Z}[x, y]$, its ideal $LT(I)$ of leading terms is defined as*

$$LT(I) = \langle \{LT(f) : f \in I\} \rangle$$

Definition 9 (Strong Gröbner basis)**.** *A* strong Gröbner basis *for the ideal $I \subset R[x, y]$ is a collection of polynomials $G = \{g_1, \ldots, g_n\}$ that satisfy the following two conditions*

1. *$\langle \{g_i\} \rangle = I$*
2. *If $f \in I$, then there exists $g \in G$ such that $LT(g) \mid LT(f)$*

Definition 10 (Weak Gröbner basis)**.** *A* weak Gröbner basis *for the ideal $I \subset R[x, y]$ is a collection of polynomials $G = \{g_1, \ldots, g_n\}$ that satisfy the following two conditions*

1. *$\langle \{g_i\} \rangle = I$*
2. *If $f \in I$, then $LT(f) \in \langle \{LT(g) : g \in G\} \rangle$; in other words, $\langle \{LT(g) : g \in G\} \rangle = LT(I)$*

By *Gröbner basis*, we refer generally to the notion of a *strong* Gröbner basis. It is worth noting that there are benefits to both types of bases; since strong Gröbner bases have stricter divisibility conditions, they are generally larger than weak Gröbner bases (and therefore require more memory to store). On the other hand, strong Gröbner bases are easier to compute [17]. From the definitions, strong Gröbner bases form a subset of weak Gröbner bases, so when we discuss ease of computation, we are referring to computing small weak Gröbner bases. Intuitively, we can imagine that this discrepancy comes from the need to discover and eliminate redundant polynomials in order to make our weak Gröbner basis as small as possible. It is worth noting that every ideal in $\mathbb{Z}[x, y]$ does indeed have a Gröbner basis, which may be non-obvious [4]. An algorithm known as the *Buchberger algorithm* was among the first methods for finding Gröbner bases over fields, and adaptations work over the integers as well (see Tables 5.4 and 10.1 in [4] for the original and adapted algorithms).

Now, imagine an eavesdropper had access to a Gröbner basis for the ideal (f, g) in our multivariate encryption scheme, and she has intercepted a new ciphertext. She could iterate through her table of known plaintext/ciphertext pairs, checking each ciphertext against the intercepted encryption. Each check is comprised of taking a difference between the known ciphertext and the new encryption. If the leading term of the resulting polynomial is not divisible by any leading term in the Gröbner basis, we already know that the difference is not an element of (f, g), so the known ciphertext cannot be an encryption of the same message as the intercepted encryption. If there

does exist a leading term in the basis dividing the leading term of the difference, then Theorem 10.23 in [4] tells us that we can algorithmically determine whether our new encryption is in (f, g).

Barkee et al [2] describe another potential threat to the security of this scheme, namely an attack which reduces the complexity of the ideal membership problem. The applicability of their threat is not explicit from their exposition, since they work entirely over a field k rather than over the integers, but it is worthwhile to mention that regardless of applicability, our scheme can (with appropriate constraints) withstand it. Their attack relies on a theorem of Dickenstein et al [5], which says that given an ideal $I = (g_1, \ldots, g_l) \in k[x_1, \ldots, x_n]$, where each g_i is public and of degree less than or equal to some integer d, and an element $h = M + \sum_{i=1}^{l} p_i g_i$ for $M = h \mod I$ and some p_i of degree less than or equal to some $r \in \mathbb{Z}$, then we can reduce h to find M by using only the polynomials output by a simplified version of the Buchberger algorithm mentioned above. Namely, the original algorithm tests each *pair* of polynomials in the input collection $\{g_i\}$ to see if a certain reduction of a combination of this pair should be included in our Gröbner basis, but the simplification allows us to ignore each pair (g_i, g_j) such that

$$\deg(g_i, g_j) := \max\left\{\deg\left(\frac{\mathrm{lcm}(LM(g_i), LM(g_j))}{LM(g_i)} g_i\right), \deg\left(\frac{\mathrm{lcm}(LM(g_i), LM(g_j))}{LM(g_j)} g_j\right)\right\} > d + r$$

In the context of our scheme, we can consider the g_i here to be encryptions $e_i(0)$ of zero which generate (f, g), which for the purposes of security analysis we can consider to be public. We use subscripts in $e_i(0)$ to denote different encryptions rather than different encryption functions. The h above is then an arbitrary encryption $e(m)$, and the p_i are the polynomials necessary to express h as

$$h = e(m) = m + \sum_{i=1}^{l} p_i e_i(0)$$

The theorem then tells us that in order to reduce h modulo (f, g), which would recover m in our scheme, it is only necessary to run the Buchberger algorithm on the pairs of polynomials $(e_i(0), e_j(0))$ such that $\deg(e_i(0), e_j(0))$ is less than or equal to the degree of the maximal-degree encryption plus the degree of the maximal-degree p_i. If we can show that all pairs satisfy this constraint, then the complexity of the Buchberger algorithm will not have been reduced (note that the Buchberger algorithm is not the fastest Gröbner basis computation algorithm, so reducing our problem to the complexity of the Buchberger algorithm does not help an eavesdropper). However, we can ensure that this is the case by the construction of our algorithm. For example, in our key generation step we can ensure that the random polynomials a_i and b_i are chosen such that the leading monomials of all encryptions are the same, then

$$\deg(g_i, g_j) = \max\left\{\deg\left(\frac{\mathrm{lcm}(LM(e_i(0)), LM(e_j(0)))}{LM(e_i(0))} e_i(0)\right), \deg\left(\frac{\mathrm{lcm}(LM(e_i(0)), LM(e_j(0)))}{LM(e_j(0))} e_j(0)\right)\right\}$$

$$= \max\left\{\deg\left(\frac{LM(e_i(0))}{LM(e_i(0))} e_i(0)\right), \deg\left(\frac{LM(e_j(0))}{LM(e_j(0))} e_j(0)\right)\right\}$$

$$= \deg(e_i(0)) = \deg(e_j(0))$$

and since all encryptions have the same leading monomial, they all have the same degree d, and $\deg(e_i(0)) = d \leq d + r$ from above, and we cannot eliminate any pairs of encryptions from our

execution of the Buchberger algorithm. Making the leading monomials of all encryptions the same should not tell us any additional information about f and g, since it is the a_i and b_i which create these equal degrees. We could also generalize this approach to allow a_i and b_i that make all of our encryptions have the same x-degree. In this case, the least common multiple of the leading monomials of each pair of encryptions would be one of the encryptions back again, so $\deg(e_i(0), e_j(0))$ would be exactly $\max\{\deg(e_i(0)), \deg(e_j(0))\} \leq d$, where d again is the upper bound on the degrees of the encryptions we are considering. In light of these possibilities, we will consider our scheme to rely on the hardness of Gröbner basis computation.

Using this scheme, then, relies on choosing our keys in such a way that the process of finding a Gröbner basis for (f, g) given a list of encryptions of zero is hard (for security against a *chosen plaintext attack*, which assumes that an eavesdropper has as access to as many plaintext/ciphertext pairs as she desires [11]). We address this question in more depth in chapter 3 below, where we discuss the implementation of these cryptosystems and related data generation. For now, we mention that the degree of the polynomials f and g appears to play a large role in the complexity of the computation (as might be expected, since increasing the total degree of a two-variable polynomial increases the number of terms quadratically, and any computation becomes more time-consuming). Though we only produce data for the two-variable case, the number of variables also plays a role in this complexity (see, for example, [3]).

2.3.8 Generalizing

As mentioned in the preceding section, the complexity of Gröbner basis computation depends in part on the number of variables involved. One potential benefit of the scheme presented is that it generalizes very easily and naturally to polynomial rings in arbitrary numbers of variables while maintaining ease of encryption and decryption. Since the scheme remains largely unchanged from that presented above, we list only the main features:

- Our secret polynomials f, g are now elements of $\mathbb{Z}[x_1, \ldots, x_n]$
- The polynomial g is now a product of a random polynomial $g' \in \mathbb{Z}[x_1, \ldots, x_n]$ and $n-1$ polynomials g_i, where $g_i = x_i - z_i$ for some random integers z_i (for $i = 2, \ldots, n$)
- Encryption is still $e(m) = m + af + bg$ for some random polynomials $a, b \in \mathbb{Z}[x_1, \ldots, x_n]$
- To decrypt, we first evaluate an encryption at $(x, z_2, \ldots, z_n)$ and then reduce modulo $f(x, z_2, \ldots, z_n) \in \mathbb{Z}[x]$

The proof of the correctness of encryption and decryption is identical to that presented in section 2.3.1, and it is thus merely sketched. In the encryption step, we add to $m \in \mathbb{Z}$ a random multiple of f and a random multiple of $g = g' \cdot (x_2 - z_2) \cdots (x_n - z_n)$. Evaluating this encryption at $(x, z_2, \ldots, z_n)$ eliminates the term involving g, and leaves us with $m + a(x, z_2, \ldots, z_n) f(x, z_2, \ldots, z_n)$. Finally, reducing modulo $f(x, z_2, \ldots, z_n)$ gives us the message, as desired. The proof of the well-definition of homomorphic operations is omitted, for it is identical to that presented above as well.

When we increase the number of variables in our scheme, we also lose efficiency, since general polynomials in more variables of the same total degree contain a greater number of terms. This becomes a significant problem when we consider multiplying ciphertexts. For this reason, it may not be practical to consider using many variables, but the possibility is there.

It is worth noting that if we attempted to generalize schemes such as those presented on pages 12 and 23 of [1], the efficiency of the decryption step would rely on reducing modulo a basis in many variables. Our scheme avoids such messy computations by allowing decryption via simple evaluations and reduction by a single-variable polynomial.

Chapter 3

Implementation and Results

In this chapter, we explain our implementation of the two encryption schemes presented above. We implement both in C++ to take advantage of the efficiency of a compiled language and the flexibility and power of an object-oriented language. In particular, classes are used to model polynomials and vectors. Furthermore, we require manipulation of integers of arbitrary size. For this purpose, we use the GNU Multiple Precision Arithmetic Library (GMP) [9]. The files referred to in the sections below can be found at https://github.com/bleveque/HomEnc, and more extensive runtime results can be found at http://btleveque.org/homenc.php.

Beyond theoretical formulation, it is important to see how these schemes might perform in practice. As such, we include runtime results and some preliminary tests of Gröbner basis computation times using the computer algebra systems Macaulay2 [10] and Maple [14] [15] (in particular, Jean-Charles Faugère's Gröbner basis package FGb [6], which is available through Maple), as well as runtime results for our schemes.

3.1 Code design

Choice-based scheme implementation

Our choice-based encryption scheme requires an implementation of vectors of size N, where N is part of the scheme's protocol. Operations on ciphertexts require the implementation of component-wise addition and multiplication of such vectors. To accomplish this, we define a vector class in the files *vec.h* and *vec.cpp*, and provide for it all of the necessary arithmetic methods. Since N is a static parameter (applying circuits does not change N), the implementation simply wraps the built-in array functionality in C++. Since the key generation function sets each element in our encryption vector before any elements are accessed, we do not need to worry about initializing the values in our vector, which helps the efficiency of our implementation.

Our implementation of the Chinese Remainder Theorem is straightforward, since we are working strictly over the integers. It makes use of GMP's built-in `mpz_gcdext` function, which for inputs $a, b \in \mathbb{Z}$ produces $c, d \in \mathbb{Z}$ such that

$$ac + bd = \gcd(a, b)$$

Multivariate scheme implementation

Following the implementation of multivariate polynomials in [12], we use linked lists in *llist.h* and *llist.cpp* to model multivariate polynomials. Each node in a list has three components: a coefficient (again represented using GMP), an x-degree, and a y-degree. This approach has several benefits over using standard arrays. Most importantly, if the polynomials in question are of large degree (say n) but are very sparse (say with only t non-zero terms), using standard arrays will necessitate $\mathcal{O}(n^2)$ storage to accommodate each of the possible terms, while the linked list will only require $\mathcal{O}(t)$ memory. Additionally, it is very straightforward to add and multiply using this construction, provided we have a well-formulated algorithm for adding nodes to our list. This insertion algorithm is outlined below:

```
 1: insert(list, node):
 2:     set term <-- list.head
 3:     while term.xdeg > node.xdeg
 4:         set term to be the next term
 5:     check if term.xdeg < node.xdeg
 6:         if so insert node before term, return
 7:     while term.xdeg == node.xdeg and term.ydeg > node.ydeg
 8:         set term to be the next term
 9:     check if term.xdeg < node.xdeg or term.ydeg < node.ydeg
10:         if so, insert node before term, return
11:     add node.coeff to term.coeff
```

This algorithm maintains the loop invariant that the input node's x-degree is always greater than or equal to that of the current term, and if they are equal, then the node's y-degree is also greater than or equal to that of the current term. If this is ever untrue (we check in lines 5 and 9), we insert the node before the current term and return (lines 6 and 10). Since the list is in descending order of degree with the variable ordering $x > y$, this successfully places our node. Note that if we reach a term with x- and y-degree both equal to those of the input node, then we combine like terms (line 11).

As mentioned above, this routine makes it straightforward to implement addition and multiplication, since like-term combination is take care of. Addition can then be accomplished by simply creating a new list and inserting each node from the two input polynomials into this list. Multiplication can be accomplished by inserting each product of pairs of terms from the two inputs.

Finally, we implement reduction in $\mathbb{Z}[x]$ by using polynomial long division. Reduction by a degree n polynomial $f \in \mathbb{Z}[x]$ is not in general guaranteed to produce a polynomial of degree less than n, since the leading coefficient of f may not divide the intermediate leading coefficients of the dividend as we perform long division. However, since we only reduce polynomials $e(m)(x, z_0)$ (which are constants added to multiples of $f(x, z_0)$) modulo $f(x, z_0)$, we never run into this problem.

3.1.1 Running times

Here we give a brief tabulation of some running times of the cryptosystems described above.

Choice-based encryption runtimes

For choice-based encryption, we include parameters for the size of the prime P, the number of ways K that we can mask messages with multiples of P, the number of operations M we are allowing, and the encryption vector length N. A small sample of running times with various parameters is given below:

Runtimes for choice-based cryptosystem				
P	K	M (#ops)	N	time (sec)
1031	10	0	256	0.5611786
1031	10	40	256	0.536617
1031	10	0	512	2.499592
1031	30	40	512	3.43153
1073741827	10	0	256	0.8517384
1073741827	10	40	256	0.569316
1073741827	30	0	512	3.998316
1073741827	30	20	512	3.807948
1073741827	30	40	512	3.56772

Further testing is necessary to establish the soundness of these results, since the average times above were taken from fairly small sample sets (five values), but the range of time values for each set of parameters was less than 0.7 seconds (and most often substantially smaller). One oddity about the data above is that applying circuits does not seem to have an effect on the time it takes to encrypt and decrypt. On the contrary, it often seems to reduce the time. Further testing would help to isolate the cause of this inconsistency. More extensive data is available at http://btleveque.org/homenc.php.

Multivariate encryption runtimes

As expected, our multivariate scheme exhibits much larger ciphertext expansion than did our previous scheme, due to the fact that upon multiplications, the polynomials roughly double in degree, which corresponds to a quadratic increase in the number of terms. As such, runtimes quickly slow down. We have not implemented reduction modulo resultant polynomials, which could help this issue. The parameters we test with are the degree of f and g, the bound on the coefficient size of f, g, and the random polynomials a and b, and the number of operations we are allowing.

Runtimes for multivariate cryptosystem			
Degree of f, g	2-log of coefficient bound	#ops	time (sec)
2	2	0	0.0002388
2	2	4	0.0024646
2	2	8	0.0108714
2	10	0	0.0001832
10	10	0	0.0078788
10	10	4	17.3912
10	10	8	290.8336

We see indeed that as we start applying circuits, our scheme slows down considerably. It would be interesting further research to see how resultant reduction would impact this situation.

3.1.2 Gröbner basis computation times

The main tool that could be used to break our multivariate encryption scheme above is the theory of *Gröbner bases*. We defined and discussed this notion in section 2.3.7, and here we give some sample results for current Gröbner basis generation software using a variety of parameters. We use Jean-Charles Faugère's FGb package through Maple and the Macaulay2 function `gb`. In Macaulay2, we generate Gröbner bases over both the integers and $\mathbb{Z}/p\mathbb{Z}$ for several choices of p. The results were generated using the scripts *grob.sh* (calling Macaulay2 over the integers), *ZpGrob.sh* (calling Macaulay2 over $\mathbb{Z}/p\mathbb{Z}$), and *MapleScript.sh* (calling FGb over the integers). We did *not* verify the correctness of the bases output by these methods. The calls to FGb sometimes resulted in errors, and we have done our best to remove the corresponding runtimes manually from the resulting data, since an error causes the function to return relatively quickly with the incorrect result. More complete results may be found on the website accompanying this thesis (located at http://btleveque.org/homenc.php).

Results from running Macaulay2 over the integers were promising, with calls to `gb` taking several minutes for fairly small-degree (i.e. degree 6) secret polynomials f and g. Note that then encryptions would have roughly twice that degree, since f and g are multiplied by random polynomials a and b in the encryption process.

Computed in Macaulay2 over Z			
degree	log coeff bd.	num. encryptions	time (sec)
3	3	2	0.00694
3	6	4	0.01548
3	10	4	0.02332
4	3	2	0.052821
4	6	4	0.23544
4	10	4	1.28991
5	10	4	10.3009
6	10	4	153.14296

Computed in Macaulay2 over $\mathbb{Z}/p\mathbb{Z}$				
degree	log coeff bd.	num. encryptions	p	time (sec)
3	10	4	11	0.00151
4	10	4	11	0.00348
5	10	4	11	0.0079
6	10	4	11	0.0178
10	10	4	11	0.1442
3	10	4	32749	0.00157
4	10	4	32749	0.004929
5	10	4	32749	0.01223
6	10	4	32749	0.02294
10	10	4	32749	0.24629
30	16	2	32749	16.9890
30	16	10	32749	50.30124

Computed using FGb over Z			
degree	log coeff bd.	num. encryptions	time (sec)
3	3	2	0.02967
3	6	4	0.025
3	10	4	0.02487
4	3	2	0.0368
4	6	4	0.0262
4	10	4	0.027
5	10	4	0.12
6	10	4	0.1232
16	20	10	43.7112
16	60	10	115.4898

We see that while all three implementations run very quickly for small degrees, the runtime starts growing rapidly as the degree increases. The FGb implementation, seems to perform best over the integers, at least for the parameters we tested. We see also that Gröbner basis computation is much faster over $\mathbb{Z}/p\mathbb{Z}$ than over the integers. Note that we do not alter the coefficient bounds in the second table above too much, since we are working over a finite field. The results above certainly show that an implementation of the multivariate encryption scheme presented in the preceding chapters would need to use fairly large-degree polynomials, which would impede efficiency.

Chapter 4

Appendix

4.1 Common notation and terminology

We recall in this section some commonly used notation used throughout the preceding pages:

- $\mathbb{Z}$ — the ring of integers, i.e. $\{\dots, -2, -1, 0, 1, 2, \dots\}$
- $\mathbb{Z}/p\mathbb{Z}$ — the ring of integers modulo p, i.e. the set of equivalence classes $\{\bar{0}, \bar{1}, \dots, (\bar{p-1})\}$
- $R[x_1, \dots, x_n]$ — the ring of polynomials in the variables $x_1, \dots, x_n$ with coefficients in the ring R; used most often above is the ring of polynomials with integer coefficients, $\mathbb{Z}[x_1, \dots, x_n]$
- If $\{r_1, \dots, r_n\} \subset R$ is a collection of ring elements, we define the *ideal generated by* $\{r_1, \dots, r_n\}$ as the collection of all finite
$$\langle r_1, \dots, r_n \rangle = \left\{ \sum_{i=1}^{n} a_i r_i \right\}$$
where $a_i \in R$. In particular, the ideal above is *finitely generated*, since the collection $\{r_i\}$ is finite. The definition is analogous for infinite generating sets; we still consider finite sums in this case.
- If f is a multivariate polynomial, we mean by its *degree* or *total degree* the largest sum of degrees of variables in any term. For example, the degree of $4x^7y + 4x^3y^3$ is 8.
- $LM(f)$ — given a polynomial f, $LM(f)$ is the highest-degree monomial term of f (i.e. not including its coefficient).
- $LC(f)$ — given a polynomial f, $LC(f)$ is the coefficient of the highest-degree term of f.
- $LT(f)$ — the highest degree term in f.
- A *commutative diagram* is a visual representation of a collection of maps which can be applied

in multiple ways to get the same result. For example, if we have the maps

$$\begin{aligned} f : \mathbb{R}^2 &\longrightarrow \mathbb{R} \\ (x, y) &\mapsto x + y \end{aligned}$$

$$\begin{aligned} g : \mathbb{R}^2 &\longrightarrow \mathbb{R}^2 \\ (x, y) &\mapsto (2x, 2y) \end{aligned}$$

$$\begin{aligned} h : \mathbb{R}^2 &\longrightarrow \mathbb{R}^2 \\ (x, y) &\mapsto (\frac{x}{2}, \frac{y}{2}) \end{aligned}$$

the following commutative diagram represents the fact that adding two numbers is the same as first multiplying both by 2, adding them, and then dividing the sum by 2:

$$\begin{array}{ccc} (2x, 2y) & \xrightarrow{\quad f \quad} & 2x + 2y \\ \uparrow g & \circlearrowleft & \downarrow h \\ (x, y) & \xrightarrow{\quad f \quad} & x + y \end{array}$$

Bibliography

[1] Martin R. Albrecht, Pooya Farshim, Jean-Charles Faugère, and Ludovic Perret. Polly Cracker, revisited. In *Advances in cryptology—ASIACRYPT 2011*, volume 7073 of *Lecture Notes in Comput. Sci.*, pages 179–196. Springer, Heidelberg, 2011.

[2] Boo Barkee, Deh Cac Can, Julia Ecks, Theo Moriarty, and R. F. Ree. Why you cannot even hope to use Gröbner bases in public key cryptography: an open letter to a scientist who failed and a challenge to those who have not yet failed. *J. Symbolic Comput.*, 18(6):497–501, 1994.

[3] David Bayer and Michael Stillman. On the complexity of computing syzygies. *J. Symbolic Comput.*, 6(2-3):135–147, 1988. Computational aspects of commutative algebra.

[4] Thomas Becker and Volker Weispfenning. *Gröbner Bases: A Computational Approach to Commutative Algebra*, volume 141 of *Graduate Texts in Mathematics*. Springer-Verlag, New York, 1993. In Cooperation with Heinz Kredel.

[5] Alicia Dickenstein, Noaï Fitchas, Marc Giusti, and Carmen Sessa. The membership problem for unmixed polynomial ideals is solvable in single exponential time. *Discrete Applied Mathematics*, 33(13):73 – 94, 1991.

[6] Jean-Charles Faugère. FGb: A Library for Computing Grbner Bases. In Komei Fukuda, Joris Hoeven, Michael Joswig, and Nobuki Takayama, editors, *Mathematical Software - ICMS 2010*, volume 6327 of *Lecture Notes in Computer Science*, pages 84–87, Berlin, Heidelberg, September 2010. Springer Berlin / Heidelberg.

[7] Michael Fellows and Neal Koblitz. Combinatorial cryptosystems galore! In *Finite fields: theory, applications, and algorithms (Las Vegas, NV, 1993)*, volume 168 of *Contemp. Math.*, pages 51–61. Amer. Math. Soc., Providence, RI, 1994.

[8] Craig Gentry. *A fully homomorphic encryption scheme*. PhD thesis, Stanford University, 2009. crypto.stanford.edu/craig.

[9] Torbjörn Granlund and the GMP development team. *GNU MP: The GNU Multiple Precision Arithmetic Library*, 5.0.5 edition, 2012. http://gmplib.org/manual/.

[10] Daniel R. Grayson and Michael E. Stillman. Macaulay2, a software system for research in algebraic geometry. Available at http://www.math.uiuc.edu/Macaulay2/.

[11] Jeffrey Hoffstein, Jill Pipher, and Joseph H. Silverman. *An introduction to mathematical cryptography*. Undergraduate Texts in Mathematics. Springer, New York, 2008.

[12] Donald E. Knuth. *The Art of Computer Programming, Volume I: Fundamental Algorithms, 2nd Edition*. Addison-Wesley, 1973.

[13] Serge Lang. *Algebra*, volume 211 of *Graduate Texts in Mathematics*. Springer-Verlag, New York, third edition, 2002.

[14] a division of Waterloo Maple Inc. Maplesoft. *Maple User Manual*. Toronto, 2005-2013. http://www.maplesoft.com/support/help/Maple/view.aspx?path=author.

[15] a division of Waterloo Maple Inc. Maplesoft. *Maple Programming Guide*. Toronto, 2013. http://www.maplesoft.com/support/help/Maple/view.aspx?path=author.

[16] W. A. Stein et al. *Sage Mathematics Software (Version 4.7.2)*. The Sage Development Team, 2011. `http://www.sagemath.org`.

[17] Mike Stillman. Personal communication, 2013.

[18] Marten van Dijk, Craig Gentry, Shai Halevi, and Vinod Vaikuntanathan. Fully homomorphic encryption over the integers. In *Advances in cryptology—EUROCRYPT 2010*, volume 6110 of *Lecture Notes in Comput. Sci.*, pages 24–43. Springer, Berlin, 2010.