Homomorphic Encryption

Ben LeVeque

April 13, 2013

Contents

Chapter 0

Acknowledgements

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 developed to 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 (INSERT ADDRESS).

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 in this work 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* and sometimes denoted $\mathcal{M}$, and the ciphertexts lie in a (possibly) different space, referred to as the *ciphertext space* and sometimes denoted $\mathcal{C}$, so we have

$$e : \mathcal{M} \longrightarrow \mathcal{C}$$
$$d : \mathcal{C} \longrightarrow \mathcal{M}$$

A cryptosystem is then given by the 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 the definition. 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:

$$\begin{tikzcd}
\text{encryptions} \arrow[r, "\text{operations}"] \arrow[dr, phantom, "\circlearrowright"] & \text{operations applied to encryptions} \arrow[d, "\text{decryption}"] \\
\text{messages} \arrow[u, "\text{encryption}"] \arrow[r, "\text{operations}"] & \text{operations applied to messages}
\end{tikzcd}$$

Homomorphic encryption ensures that we achieve the same result by traversing this diagram in either direction starting from our messages. 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 were possible, 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 the 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 a homomorphism in most circumstances, 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 large circuits might exceed. For motivation, consider the following toy example.

First, we will construct a somewhat homomorphic encryption scheme and then 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

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

However, if $m \geq p$, then $d(e(m))$ will no longer equal m, but will be the reduction of m modulo p. One way to attempt to avoid this problem is to encrypt values only less than p. However, once we start considering circuits applied to encryptions, this idea shows its flaws; the sum of two encryptions of $p-1$, for example, would really be an encryption of $2p-2$, but it would decrypt to $-2 \equiv p-2 \pmod{p}$. 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 to for messages to live in $\mathbb{Z}/p\mathbb{Z}$, since when we compute, for example, 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. Suppose we were asked to perform many such encryptions. We would end up with a collection of multiples of p. With this information, it would be very fast to 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. This attack and others are discussed in the appendix.

Chapter 2

Schemes under investigation

2.1 Introduction to present work

In his 2009 dissertation CITATION, Craig Gentry introduced a fully homomorphic encryption scheme based on the theory of ideal lattices (ideals in polynomial quotient rings).

This thesis will present two encryption schemes and explore their security and efficiency.

In this project, we formulate two schemes that show promise as either somewhat or fully homomorphic encryption schemes. 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 (CITATION – from GENTRY AS WELL?)—have been formulated 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.

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 $M \geq 1$ 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 $((K+1)P)^M < \prod p_i$
6. Pick primes $\{q_i\}_{i=1}^N$
7. Return private key $(N, K, P, \{p_i\}, \{q_i\})$ and public key $(N, \{p_i q_i\})$

- Encryption (e)

1. Pick a message $m \in \mathcal{M}$
2. Pick random integers $\{a_i\}_{i=1}^N$
3. Pick a random 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 the $\{p_i\}$ satisfy the constraints in step 4 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. This is a trade-off between efficiency and security that will be analyzed in more depth later on, but for now, we note that it is also possible to create a larger message space by increasing the size of N as well. 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 or add noise 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 multiplication operations to be performed. 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:

$$\begin{aligned} e(0) &= (a_{1,1}p_1 + b_{1,1}p_1q_1, \ldots, a_{1,N}p_N + b_{1,N}p_Nq_N) \\ &\vdots \\ e(0) &= (a_{r,1}p_1br, 1p_1q_1, \ldots, a_{r,N}p_Nb_{r,N}p_Nq_N) \end{aligned}$$

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 $M \geq 1$), 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 multiplications. Note that we measure the level of "homomorphicity" using multiplications rather than additions because mulitplications have a larger effect on the size of ciphertexts, and will more quickly cause our numbers to exceed $\prod p_i$. 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

$$\begin{aligned} e(m_1)+e(m_2) &= (m_1+k_1P+a_{1,i}p_i \pmod{p_iq_i}) + (m_2+k_2P+a_{2,i}p_i \pmod{p_iq_i}) \\ &= (m_1+m_2+(k_1+k_2)P+(a_{1,i}+a_{2,i})p_i \pmod{p_iq_i}) \end{aligned}$$

Let us assume for the moment that $m_1+m_1+(k_1+k_2)P < \prod p_i$. 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 and 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 $e(m_2)$:

$$\begin{aligned} e(m_1)e(m_2) &= (m_1+k_1P+a_{1,i}p_i \pmod{p_iq_i}) \cdot (m_2+k_2P+a_{2,i}p_i \pmod{p_iq_i}) \\ &= (m_1m_2+(m_1k_2+m_2k_1+k_1k_2P)P+ \\ &\qquad ((m_1+k_1P)a_{2,i}+(m_2+k_2P)a_{1,i}+a_{1,i}a_{2,i}p_i)p_i \pmod{p_iq_i}) \end{aligned}$$

In this case, as long as

$$m_1m_2+(m_1k_2+m_2k_1+k_1k_2P)P < \prod p_i$$

we see that $e(m_1)e(m_2)$ will decrypt as m_1m_2 by applying the decryption function. Therefore, $e(m_1)e(m_2)$ is a valid encryption of m_1m_2, so $e(m_1)e(m_2) = e(m_1m_2)$. 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 $((K_1)P)^M$, 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

$$\begin{aligned} \{p_i\} &= \{263, 251\} \\ \{q_i\} &= \{223, 263\} \end{aligned}$$

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

$$\begin{aligned} \{p_i\} &= \{97, 67, 89\} \\ \{q_i\} &= \{107, 79, 127\} \end{aligned}$$

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

$$\begin{aligned} e(m_1) &= (8097, 649, 3072) \\ e(m_2) &= (8293, 4805, 7791) \\ e(m_3) &= (4515, 1728, 5037) \end{aligned}$$

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

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

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 $\mathbb{Z}/p_iq_i\mathbb{Z}$ 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 4.

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 $m < \sqrt[M]{P}$ 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 $\mathcal{M} = \mathbb{Z}_{< \sqrt[M]{P}}$.

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 $|m| = \sqrt[M]{P}$, while a ciphertext has size

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

Note that if we did not reduce each component modulo p_iq_i, our ciphertexts could grow quite rapidly. Also, since we're reducing modulo p_iq_i, we do not experience any ciphertext expansion when we apply circuits to ciphertexts. [[[[[According to (CITATION OF P.C.R), this scheme is called *compact*, meaning that there is a fixed polynomial bound on the size of]]]]]

2.3 Using multivariate polynomial rings

Another approach to homomorphic encryption is to use the properties of arithmetic in polynomial rings. The ring structure essentially guarantees that if our encryption involves only arithmetic operations, it will be homomorphic. 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:

$$e(m_1) + e(m_2) = m_1 + m_2 + (a_1 + a_2)f = e(m_1 + m_2)$$

and

$$e(m_1)e(m_2) = m_1m_2 + (m_2a_1 + m_1a_2 + a_1a_2)f = e(m_1m_2)$$

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 I 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

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

Now, if Eve takes the greatest common divisor of the elements in this collection (note that this is very similar 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 is a basic case of a more general problem:

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

If the structure of R is simple, then this problem is correspondingly easy, as we saw above in the case of our set of generators being multiples of a single polynomial $f \in \mathbb{Z}[x]$. The ease of solution here comes from the fact that greatest common divisors are fairly easy to compute over $\mathbb{Z}[x]$; with the greatest common divisor of our generators available, we can easily reduce further encryptions by the common divisor to reveal their corresponding plaintexts. 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. GIVE EXAMPLE OF THE NECESSITY OF ORDERING MONOMIALS. It is with this motivation that we formulate the following scheme:

2.3.1 Scheme

The scheme goes as follows:

- Key generation (KG)
  1. Pick a general degree bound D and a coefficient bound B
  2. Pick a random number $z_0 < B$
  3. Pick a random polynomial f with total degree less than or equal to D and coefficients less than B
  4. Pick a random polynomial g' of total degree less than or equal to $D - 1$
  5. Set $g := (y - z_0)g'$
  6. Return private key (f, g, z_0)
- Encryption (e)
  1. Pick a message $m \in \mathcal{M} := \mathbb{Z}$
  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 an encryption at z_0
  2. Reduce the result modulo $f(x, z_0)$, a single-variable polynomial, and return the result
- Addition of ciphertexts ($+$)
- Multiplication of ciphertexts ($\cdot$)

2.3.2 Explanation

We can use the results below to choose our degree and coefficient bounds appropriately to assure us of security. The idea behind our scheme is to give an encryption which is a multivariate polynomial (to increase the complexity of the Groebner basis computation), but which can be easily reduced during the decryption stage. This is accomplished by constructing our secret key g such that it vanishes at the line $y = z_0$. We can then plug in (x, z_0) to eliminate g, and we are left with $m + (af)(x, z_0)$. Reduction modulo single variable polynomials is easy to accomplish, so we can then reduce modulo $f(x, z_0)$ to retrieve m. Since reducing modulo a polynomial does not put any restrictions on the size of m (as reducing modulo an integer would), m can be arbitrarily large and still go through encryption and decryption in a well-defined manner.

2.3.3 Correctness of encryption/decryption

We have broached this subject above, and now we will prove it

Claim: If $m \in \mathbb{Z}$, then $d(e(m)) = m$

Proof of Claim:

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 order to decrypt, we first evaluate at (x, z_0), which gives

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

This is a single variable polynomial which can be reduced modulo the polynomial $f(x, z_0)$ (which is known to the decryptor, since f and z_0 are both components of the secret key) to give m. 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.

Claim: If $m_1, m_2 \in \mathbb{Z}$, then $e(m_1) + e(m_2) = e(m_1 + m_2)$

Proof of Claim:

Let us recall first that the equality above is a coset equality, since encryptions are equivalent if they ar

2.3.5 Practical analysis of homomorphicity

2.3.6 Generalizing

... One benefit to this scheme is that it can be easily generalized to n variables – i.e. we can consider an analogous scheme over $\mathbb{Z}[x_1, \ldots, x_n]$. This is done by choosing g such that $g(x_1, \ldots, x_n) = g_1(x_1)g_2(x_2)\cdots g_n(x_n)$ and $z_2, \ldots, z_n$ such that z_i is a root of g_i. Then encryption proceeds as in the two-variable case:

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

and to decrypt, we first evaluate at $(x_1, z_2, \ldots, z_n)$ and then reduce modulo $f(x_1, z_2, \ldots, z_n) \in \mathbb{Z}[x_1]$. Further tests would be necessary to see if increasing the number of variables is an effective way to increase the computational complexity of Groebner basis computation.

Chapter 3

Applications

Beyond theoretical formulation, it is important to see how these schemes might perform in practice. As mentioned in the introduction, the efficiency of homomorphic schemes has been a major obstacle to their dissemination, and

3.1 Application to medical needs

3.2 Application to finance

EXAMPLE OF STD-DEV

A significant drawback of using our choice-based encryption scheme for the manipulation of statistics is made apparent when we consider taking the standard deviation of many numbers. This requires being able to make a large number of well-defined multiplications, which may be impossible, especially if the numbers in question are large. Problems like this may be more suited to our multivariate scheme, which can make arbitrary numbers of well-defined multiplications. We may lose efficiency, but in computations that require correctness over

Chapter 4

Implementation

Chapter 5

Appendix

5.1 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, $\mathbb{Z}[x_1, \dots, x_n]$
- 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 product by 2:

$$\begin{tikzcd}
(2x, 2y) \arrow[r, "f"] & 2x + 2y \arrow[d, "h"] \\
(x, y) \arrow[u, "g"] \arrow[r, "f"] \arrow[ur, phantom, "\circlearrowleft"] & x + y
\end{tikzcd}$$

5.2 Common Attacks

This section is a compilation of and reference for some of the types of attacks considered in the duration of this project.

- gcd attacks
- G.B. attacks (type of gcd)
- using known encryptions of zero to identify encryptions of the same value
-

5.3 An intro to Gröbner bases

5.4 Failed Attempts

5.5 Notes

Cite Donald Knuth, Art of Computer Programming, Addison-Wesley Publishing Company, Reading, Massachusetts, second addition, pp 273-4 for representation of multivariate polynomials using linked lists.

Cite Gentry, Fellows and Koblitz

Page 38 of HPS