2011

Integer-based fully homomorphic encryption

Michael Snook

Rochester Institute of Technology
College of Science
School of Mathematical Sciences

Integer-Based Fully Homomorphic Encryption

by

Michael Snook

Thesis submitted in partial fulfillment of the requirements for the degree of
Master of Science
in
Applied and Computational Mathematics

17 June 2011

Committee Signatures

Dr. Anurag Agarwal

Dr. Stanisław Radziszowski

Prof. David Barth-Hart

Abstract

The concept of fully homomorphic encryption has been considered the "holy grail" of cryptography since the discovery of secure public key cryptography in the 1970s. Fully homomorphic encryption allows arbitrary computation on encrypted data to be performed securely. Craig Gentry's new method of bootstrapping introduced in 2009 provides a technique for constructing fully homomorphic cryptosystems.

In this paper we explore one such bootstrappable system based on simple integer arithmetic in a manner that someone without a high level of experience in homomorphic encryption can readily understand. Further, we present an implementation of the system as well as a lattice-based attack. We present performance results of our implementation under various parameter choices and the resistance of the system to the lattice-based attack under those parameters. Unfortunately, while the system is very interesting from a theoretical point of view, the results show that it is still not feasible for use.

Contents

Chapter 1

Background

The notion of a "privacy homomorphism" was first proposed in 1978 by Rivest, Adleman and Dertouzos in [18]; it is highly likely that the question was raised partly due to the multiplicative homomorphism of the RSA cryptosystem published earlier that same year in [19]. The concept of a privacy homomorphism is what is known today as a *fully homomorphic* cryptosystem. This refers to a cryptosystem that preserves both addition and multiplication, or the algebraic structure of an abstract ring, across the encryption function. In such a system, arbitrary algebraic functions can be performed on a collection of ciphertexts in a way that corresponds to performing the same operations on the plaintexts.

1.1 Requirements

In order to discuss any homomorphic cryptosystem, we need an understanding of what makes a homomorphic system different from a generic cryptosystem. Any cryptosystem has three algorithms defined:

1. Key generation, which takes a desired security level and outputs a

private/public key pair $(\mathrm{sk}, \mathrm{pk})$ for an asymmetric cryptosystem, or a secret key sk for a symmetric cryptosystem.

2. Encryption, which takes a plaintext m and a (public) key pk and outputs a ciphertext c. We designate this by $c = \mathcal{E}(m)$.

3. Decryption, which takes a ciphertext c and secret key sk, and outputs the corresponding plaintext m. We designate this with $m = \mathcal{D}(c)$.

For any system, decryption must invert encryption; that is, $\mathcal{D}(\mathcal{E}(m))$ must be the same plaintext as m. However, multiple unique ciphertexts may be returned by $\mathcal{E}$ for a given plaintext. A cryptosystem that does so is called *probabilistic*, and only such a system can be *semantically secure*, as proposed by Goldwasser and Micali in [11]. Semantic security is equivalent to ciphertext indistinguishability, which states that an attacker, if given two plaintexts a ciphertext obtained by encrypting one of them and the public key used to perform the encryption, cannot determine with probability greater than one-half (the probability of a random guess) which plaintext corresponds to the ciphertext.

A homomorphic cryptosystem has these three algorithms, and adds to them a fourth: the Evaluate algorithm. This takes in a t-tuple of ciphertexts $\mathbf{c} = \langle c_1, \ldots, c_t \rangle$, the public key pk that the resulting ciphertext is to be encrypted under and a function or circuit C that takes t inputs. The Evaluation algorithm outputs a ciphertext corresponding to the application of C to the ciphertexts $\mathbf{c}$. The strength of homomorphic encryption comes from the fact that the Evaluate function transfers operations in the plaintext space to corresponding operations in the ciphertext space.

Definition 1 (Correct Homomorphic Decryption)**.** A cryptosystem with encryption function $\mathcal{E}$ and decryption function $\mathcal{D}$ is *correct* for a given t-input circuit C if the following holds: For any t plaintext bits $\mathbf{m} = \langle m_1, \ldots, m_t \rangle$ and corresponding ciphertexts $\mathbf{c} = \langle c_1 \ldots, c_t \rangle$, where $c_i = \mathcal{E}(m_i)$, $\mathcal{D}c = C(m_1, \ldots, m_t)$ where c is the output of the Evaluate algorithm given the

ciphertexts $\mathbf{c}$ and circuit C.

A system that is correct for every circuit in some class $\mathcal{C}$ of circuits is said to be homomorphic for the class $\mathcal{C}$. A system that is correct for all Boolean circuits is called *fully homomorphic*.

This definition for what makes a cryptosystem homomorphic is largely useful, but it has one important weakness; it is satisfied by trivial systems where the Evaluate algorithm simply appends a description of the circuit C to the collection of ciphertexts, and leaves the actual computation to the decrypting party to perform on the decrypted plaintexts. Since this defeats the purpose of a homomorphic cryptosystem, we want an additional requirement to exclude these trivial solutions. The way we do this is by requiring that the size of the output of the Evaluate algorithm is bounded by some polynomial $f(\lambda)$, independent of the size of the circuit C.

1.1.1 Bootstrapping

The major breakthrough for fully homomorphic encryption was the notion of bootstrapping, presented by Craig Gentry in his 2009 PhD thesis,[8]. The bootstrapping technique transforms a somewhat homomorphic system into a fully homomorphic one as long as the decryption process can be expressed as a Boolean circuit small enough to to be evaluated homomorphically by the system.

More formally, take a cryptosystem with decryption function $\mathcal{D}$ expressed as a Boolean circuit. Then for any Boolean gate g, we form a new circuit:
Definition 2 (Augmented Decryption Circuit)**.** The g-augmented decryption circuit for a Boolean gate g with n inputs is the circuit formed by joining n copies of the decryption circuit $\mathcal{D}$ with the gate g; the output of each copy of $\mathcal{D}$ leads to a separate input of g.

The act of decrypting a ciphertext homomorphically in essence "recrypts" the ciphertext. That is, it provides a new ciphertext $\hat{c}$ for any given ciphertext c, such that $\mathcal{D}(c) = \mathcal{D}(\hat{c}) = m$, with the advantage that the new ciphertext $\hat{c}$ is *fresh*, as if it has had no operations applied to it previously.

From this, we have what it means to be a bootstrappable system. A cryptosystem is bootstrappable with respect to some set Γ of gates if it correctly evaluates the g-augmented decryption circuit for every gate $g \in \Gamma$. Then any circuit C made up of gates in Γ, no matter how large, can be evaluated by replacing every gate in C with the corresponding augmented decryption circuit and evaluating the resulting circuit. Therefore, a cryptosystem which is bootstrappable with respect to some functionally complete set of operations (e.g. logical AND, XOR and constant TRUE) would be fully homomorphic.

One key point to make about bootstrapping is that performing the bootstrapping process itself involves an encrypted version of the private key. Thus, one can set up a chain of n keys, with each private key being encrypted by the next public key in the chain, and use these to evaluate a circuit requiring at most n levels of bootstrapping. If a secret key remains secure after being encrypted with its corresponding public key — that is, if knowing that a given ciphertext corresponds to a secret key sk using the corresponding public key pk offers no additional information about sk — then arbitrarily many bootstrapping procedures may be made using only the one key. This property is known as *circular security* for the key sk, and is generally taken to be true though formally proving such tends to be difficult.

1.2 Applications

Since the original proposal of the concept of fully homomorphic encryption by Rivest, Adleman and Dertouzos, a number of uses for fully homomorphic systems have been proposed. The general concept behind all of them

is fairly straightforward: some sort of data/information is encrypted for privacy/security reasons, while still allowing third parties to manipulate the encrypted data in a meaningful way. One general case is the outsourcing of processing sensitive data to a third party. The third party, usually one with extensive computing resources, has to be able to properly process the data, but should not be able to tell what it has processed. Some applications only require homomorphism over either addition or multiplication; others require both: a fully homomorphic system.

1.2.1 Multi-Party Communication

The problem of secure multi-party communication results when two or more parties wish to know the value of some function where each party has one input to the function without revealing their own data to the others. Put more simply, in a group of n users, each user i knows piece of information m_i. The secure multi-party problem arises when they want to combine their knowledge in some way, $f(m_1, \ldots, m_n)$ while keeping each individual m_i private.

Millionaire Problem

One of the most prominent examples of such a problem is the millionaire problem proposed by Andrew Yao in [23]. Two millionaires, Alice and Bob, wish to know which of them is richer without revealing their actual worths to each other. Yao proposed a solution with the problem statement, but that solution takes exponential time and space. A number of protocols to improve this have been proposed using additively homomorphic cryptosystems as a base; the protocol described in [16] in particular can use either an additively or multiplicatively homomorphic system, where the multiplicative systems incur a somewhat lower computation cost.

The millionaire problem can be generalized to comparing a number of different values to find their order, the maximum value and so forth. This leads readily to the concept of a private auction. If n bidders make bids, then solving $n-1$ instances of the millionaire problem reveals who the highest bidder is, while protecting the privacy of all those involved by keeping their actual bets secret.

E-Voting

A very large-scale example of an application of homomorphic encryption is the idea of electronic voting. An ideal voting scheme allows for quick computation of the winner(s) of a vote, while preserving each individual voter's anonymity. It should also provide some mechanism for verifying that the count is correctly computed, that each vote is counted as the individual voter intended, and that no tampering of the votes has occurred. Of course, not every implementation of the concept will necessarily have all of these attributes.

The concept of a voting system being *receipt free* is related to prevent a voter selling his vote to a third party or being coerced into voting in a particular way. A receipt free system prevents a voter from proving whom they voted for to any third party. Thus, an individual voter's choice must be kept secret; the system enforces this. In [12], Martin Hirt and Kazue Sako present a receipt free system for 1-out-of-N elections, in which each voter chooses one of N options to vote for. Hirt and Sako's system relies on having an underlying encryption scheme that is homomorphic over addition.

Various authors have attempted to improve existing e-voting systems in a number of ways. For example, [1] explores the use of homomorphic encryption for elections in which each voter gives an ordered list of preferences, rather than a simple 1-out-of-N election. They show how homomorphic en-

cryption can be used for preferential elections, but find that its efficiency is too low to be used practically, and instead turn to a technique known as mix-networks. On the other hand, [17], which has some of the same authors, proposes an e-voting system based off of multiplicatively homomorphic cryptosystems rather than the additively homomorphic systems more commonly studied. The goal of using multiplicatively homomorphic systems is to improve efficiency for 1-out-of-N elections.

More Examples

Consider three employees Alice, Bob and Charlie. The three want to know their average salary, but each wishes to keep their own salary private. That is, if their salaries are a, b, c respectively, then fully homomorphic encryption can be used to find $(a + b + c)/3$ without revealing the values of a, b or c. Alice can of course tell whether her own salary is above or below average, but cannot discern more about Bob's salary or Charlie's salary than knowing the average would tell her anyway. This concept can again be generalized to any number of employees greater than two — the system still works with two employees, but knowing the average salary would reveal the other employee's salary in that case.

Another example related to homomorphic encryption is the *dining cryptographers problem*, a method for anonymous public broadcast among a group of peers. The dining cryptographers problem is typically presented as a group of cryptographers seated around a table at a restaurant. They are informed by a waiter that their bill has been paid in advance, though they are not told who paid the bill. The cryptographers wish to know if one of them paid the bill or whether a third party did so, while respecting each individual's right to make a payment anonymously. The typical solution, appropriately called the *dining cryptographers protocol*, uses techniques similar to those found in (additively) homomorphic encryption to achieve similar goals. Note

however, that the dining cryptographers protocol uses only one operation, so fully homomorphic encryption is more powerful than the protocol requires.

1.2.2 Zero-Knowledge Proofs

One useful idea in cryptography is the notion of a zero-knowledge proof. A zero-knowledge proof is some method where Peggy, the *prover*, wishes to prove some statement to Victor, the *verifier* of Peggy's claim, without revealing any other information to Victor. Specifically, a zero-knowledge proof must:

- always convince Victor of Peggy's claim when it it is true
- only convince Victor with some small probability (which can usually be made arbitrarily small) when Peggy tries to cheat the system
- reveal nothing to Victor other than the fact that Peggy is attempting to prove

As a related example to the millionaire problem above, suppose Peggy is a millionaire with m dollars and wishes to prove to Victor that $m > 1000000$, that she is in fact a millionaire. Simply revealing to Victor the value of m would indeed prove to his satisfaction that $m > 1000000$, but a zero-knowledge proof would be one in which Victor learns only that m is greater than one million and nothing more. A solution to the millionaire problem becomes a simple proof in this case: if Victor pretends to be a millionaire with exactly one million dollars, then solving the millionaire problem will reveal that Peggy has more money than Victor without revealing her actual worth m.

Just as Yao published a solution to his millionaire problem without use of homomorphic encryption, zero-knowledge proofs can be implemented without homomorphic encryption; the use of homomorphic encryption is not to

enable zero-knowledge proofs but to improve the efficiency with which they can be performed.

1.2.3 Security in Cloud Computing

The true power of fully homomorphic encryption is that it can be used to securely compute any Boolean function; some applications can be achieved with only additive or multiplicative homomorphism, but an arbitrary computation engine must have access to both (or some equivalent ability). This allows one party, the client, to offload *any* computation they want done on encrypted data to an untrusted third party, a very useful thing to be able to do.

The act of processing large databases in some manner is a common problem in the modern, digital era. Companies have customer histories and financial records, hospitals and other medical facilities have patients' medical records, and so on. One growing trend is to outsource the storage and/or processing of data to external providers in order to save money on the purchasing and maintenance of large computer systems. Private individuals are increasingly using so-called cloud computing for the ease of universal access and participating in large-scale distributed computation projects such as the Great Internet Mersenne Prime Search (GIMPS) or the Folding@home project for understanding protein-related diseases.

One issue with doing so is that the data to be processed may be confidential or otherwise sensitive. For example, hospitals are legally required to preserve the confidentiality of their patients' medical records; even without being required to keep the data secret, companies may wish to do so for their own reasons. The application of a fully homomorphic cryptosystem to this problem allows companies, hospitals and members of the general public to offload the processing of sensitive information to untrusted parties.

Related Challenges

This outsourcing of computations has a number of important challenges related to it, and a general system must be capable of solving all of them to the satisfaction of the client. Fully homomorphic encryption is a large piece of the puzzle, but it only protects one thing: the privacy of the data being operated on. Other aspects of the client's privacy must be dealt with in other ways.

Recall that fully homomorphic encryption allows the third party to compute $\mathcal{E}(f(m_1, \ldots, m_n))$ given Boolean circuit C corresponding to f and the ciphertexts $\mathcal{E}(m_1), \ldots, \mathcal{E}(m_n)$. However, the company requesting the data processing may not wish to reveal even the processing being done; they with to keep f secret. This concept is known as *circuit privacy*. Andrew Yao — the same one to pose the millionaire problem — proposed the idea of garbled circuits in 1986 [24] as a tool for secure multi-party communication (see section 1.2.1). The garbled circuit construction provides a rudimentary level of circuit privacy; Gentry, Halevi and Vaikuntanathan created a *re-randomizable* variant to deal with the problem of evaluating nested functions homomorphically, that is computing $\mathcal{E}(g(f(x)))$ from $\mathcal{E}(f(x))$, called i-hop homomorphic encryption[10].

The second major issue to consider is the honesty of the computing party. A client outsourcing their computations can use fully homomorphic encryption to enable the computing party to evaluate $c = \mathcal{E}(f(m_1, \ldots, m_k))$ without compromising the security of $\{m_i\}$. However, this does not guarantee that the computing party will, in fact evaluate c. A dishonest server could perform an unrelated computation and return the result or even return a random number to the client. Under a paid outsourcing agreement, this effectively cheats the client out of the agreed-on fee. Distributed computing systems run into similar issues with individuals trying to claim credit for discoveries they don't actually perform. The most common protection they implement is to

withhold credit until independent verification; this technique is vulnerable to collusion, and wastes resources by performing redundant calculations. A combination of fully homomorphic encryption and garbled circuits is presented in [7] to introduce a formal concept of *verifiable computing* as well as provide an initial solution to the issue.

1.3 A Brief History

1978 RSA published, with unintentional side effect of preserving multiplication across encryption [19]. Rivest, Adleman and Dertouzos propose idea of privacy homomorphisms [18], most likely due to accidental homomorphism of RSA.

1996 Josep Domingo-Ferrer publishes a privacy homomorphism [5].

2000 Brahm Cohen posts a public key system similar to the later van Dijk system to the internet [4], though this system is not homomorphic as-is. Hirt and Sako propose their receipt-free voting scheme [12].

2002 Ferrer Publishes another, different privacy homomorphism system [6].

2003 Ferrer's system from 2002 is shown to be insecure and breakable [22]. Australia based team investigates use of homomorphic encryption for preferential elections [1].

2004 A second Australian team, consisting of many of the same members as the preferential elections paper, proposes a voting scheme based off of multiplication rather than addition [17].

2005 Dan Boneh, Eu-Jin Goh and Kobbi Nissim publish a partially homomorphic cryptosystem [2]. Their system is homomorphic over addition, as well as a single multiplication.

2006 Ferrer's original 1996 system also shown to be insecure [3].

2008 Levieil and Naccache publish a system based on an additively homomorphic cryptosystem to prevent cheating on exams [15].

2009 Craig Gentry publishes his PhD. thesis, in which he details the technique of bootstrapping to turn partially homomorphic systems into fully homomorphic systems [8]. The thesis also includes a lattice-based system to demonstrate the bootstrapping ideas.

2010 Gentry's techniques are used by van Dijk et al. to create a fully homomorphic cryptosystem based off of integer arithmetic [21]. This system has the advantage of being conceptually easier than Gentry's lattice system. Gentry's lattice system is refined by Smart and Vercauteren [20] and implemented with bootstrapping by Gentry and Halevi [9].

Chapter 2

Underlying Homomorphic System

Shortly after Gentry's breakthrough paper on bootstrapping, a team consisting of Marten van Dijk of MIT and Gentry, Shai Halevi and Vinod Vaikuntanathan from IBM Research published a fully homomorphic encryption scheme based on simple modular arithmetic[21]. Their system has the advantage of being conceptually easier than the lattice-based system Gentry advanced in [8]. Their system is similar to a (non-homomorphic) system proposed in 2000 by Bram Cohen[4] and a system from 2008 from Levieil and Naccache[15] which uses additive homomorphism to prevent students from cheating on exams.

2.1 Description

The system proposed by van Dijk et al. encrypts a single bit plaintext as a large integer using a relatively simple arithmetic equation. Let p be a randomly chosen η bit long odd integer; this is the secret key. Let r be a

random integer ρ bits long, and let q be a random integer such that qp is γ bits long. The variables q and r are ephemeral keys, and need not be retained. For a plaintext $m \in \{0, 1\}$, set c according to

$$c = \mathcal{E}(m) = pq + 2r + m \tag{2.1}$$

then c is a γ bit ciphertext corresponding to m.

Decryption can be achieved by reversing this process.

$$m = \mathcal{D}(c) = (c \bmod p) \bmod 2 \tag{2.2}$$

Also, since $(c \bmod p) = c - p\lfloor c/p \rfloor$, the decryption formula can be alternatively expressed as $m = (c \bmod 2) \oplus (\lfloor c/p \rfloor \bmod 2)$. Note that in their original paper, van Dijk's team use the *closest integer* function, $\lfloor c/p \rceil$, which requires interpreting modulo as $-\lfloor n/2 \rfloor < x \bmod n \leq \lfloor n/2 \rfloor$. For this paper, we use the *greatest integer* function instead, and interpret modulo as $0 \leq x \bmod n < n$. Doing so eases implementation of the system, while incurring no significant penalty.

2.1.1 Correctness

Theorem 1. *Let m_1 and m_2 be two one-bit plaintexts, and let $c_1 = \mathcal{E}(m_1) = q_1 p + 2r_1 + m_1$ and $c_2 = \mathcal{E}(m_2) = q_2 p + 2r_2 + m_2$. Then $c_1 + c_2$ is a valid encryption of $m_1 \oplus m_2$ and $c_1 c_2$ is a valid encryption of $m_1 m_2$, subject to $(2r_1 + m_1)(2r_2 + m_2) = 4r_1 r_2 + 2r_1 m_2 + 2r_2 m_1 + m_1 m_2 < p$.*

Proof. Expand the sum and product:

$$\begin{aligned} c_1 + c_2 &= pq_1 + 2r_1 + m_1 + pq_2 + 2r_2 + m_2 \\ &= p(q_1 + q_2) + 2(r_1 + r_2) + (m_1 + m_2) \\ &\equiv 2(r_1 + r_2) + (m_1 + m_2) \pmod{p} \end{aligned}$$

and

$$\begin{aligned} c_1 \times c_2 &= (pq_1 + 2r_1 + m_1)(pq_2 + 2r_2 + m_2) \\ &= p^2 q_1 q_2 + 2pq_1 r_2 + pq_1 m_2 \\ &\quad + 2pq_2 r_1 + 4r_1 r_2 + 2r_1 m_2 \\ &\quad + pq_2 m_1 + 2r_2 m_1 + m_1 m_2 \\ &\equiv (2r_1 + m_1)(2r_2 + m_2) \pmod{p} \end{aligned}$$

If $(2r_1 + m_1)(2r_2 + m_2) < p$, then reducing it mod p has no effect, so further reducing mod 2 retrieves $m_1 m_2$. Likewise, if $2(r_1 + r_2) + (m_1 + m_2) < p$, reducing mod 2 retrieves $m_1 + m_2$. However, if either of these is greater than or equal to p, reducing modulo p can change the parity, so that the reduction mod 2 gives a different value. Thus, the system can evaluate addition correctly as long as $2(r_1 + r_2) + (m_1 + m_2) < p$, and it can evaluate multiplication correctly as long as $(2r_1 + m_1)(2r_2 + m_2) < p$. Since $2(r_1 + r_2) + (m_1 + m_2) < (2r_1 + m_1)(2r_2 + m_2)$, it suffices to say that both are correctly evaluated if $(2r_1 + m_1)(2r_2 + m_2) < p$. □

For the purposes of this thesis, all operations are implicitly assumed to remain within these bounds when the size of the function being evaluated is not directly relevant.

Bit Lengths

As seen above, the somewhat homomorphic scheme correctly evaluates both addition and multiplication, provided that the total noise of the computation remains less than p. In order to simplify analysis of the system, we use the length in bits of both the noise and p. The bit length of a number n is equal to $\lfloor \log n \rfloor + 1$ (all logarithms in this paper are assumed to be base 2 unless explicitly specified). As long as the noise is shorter than p, it is also smaller than p, so the system remains homomorphic.

For a given set of operations to perform on a number of inputs, the effect of the operations on the length of the noise is easily computed. A single addition of two ciphertexts has noise equal to the sum of the noises of the summands. This is equal to the length of the longer noise, plus possibly a single carry bit. A single multiplication produces noise with length equal to the sum of the lengths of the noises of the factors.

2.1.2 Completeness

Section 2.1.1 shows that as long as the number of operations performed remains bounded, the system can properly evaluate any sequence of multiplications and additions on a collection of ciphertexts, and the result will be a valid encryption for the same sequence of multiplications and additions carried out on the corresponding plaintexts. The only difference is that the plaintexts are considered as elements of $\mathbb{Z}_2$, and so plaintext addition and multiplication are considered modulo 2. The next thing to consider is what types of functions the system can then evaluate.

The primary limitation on what can be evaluated is the limit on function size seen above. Gentry's bootstrapping technique can be used to remove this limitation, provided the decryption algorithm for the system can be expressed

as a function small enough to be correctly evaluated. The application of bootstrapping to this system will be explored in section 2.3.

Besides the limitation on function size, this system can evaluate any function composed of multiplication and addition. When considered as operations over $\mathbb{Z}_2$, these correspond to logical AND and XOR, respectively. Further, the number 1 is a valid, if insecure, encryption of plaintext 1. Thus, the set of Boolean functions that can be evaluated by this system are those that can be expressed by logical AND, XOR and constant TRUE. Since these form a functionally complete set of operations, any Boolean function can be correctly evaluated by this system. The expression of an arbitrary Boolean function in terms of AND and XOR may be somewhat larger than expressing the same function with some other set of gates, though.

2.2 Public Key Version

The system as described above is a symmetric encryption scheme: p is used for encryption as well as decryption. In order to turn it into a public key system, we take advantage of the homomorphic nature of the system. Given a number of ciphertexts $c_1, \ldots, c_k$ all of which encrypt the plaintext 0, the sum $c_1 + \ldots + c_k$ is also a valid encryption of zero. Since zero and one are also valid encryptions of themselves, the sum $c_1 + \ldots + c_k + m$ is a valid encryption of the plaintext m. This idea becomes the basis for the public key version of the system.

The private key for the new system remains the integer p, which is odd and η bits long. The public key consists of a set K of τ integers that are near-multiples of p. The integers are of the form $x_i = q_i p + r_i$, where r is a random noise factor ρ bits long and x_i itself is γ bits long. Encryption of a plaintext m is then performed by selecting a random subset S of K, a random noise

ρ' bits long, and outputting the ciphertext c according to

$$c = 2\sum_{x \in S} x + 2r + m$$

Each x in the public key is of the form $q_i p + r_i$. The multiplication by 2 turns this into $(2q_i)p + 2r_i$, which is a valid encryption of zero. This can be done all during encryption rather than making each x_i a valid encryption of zero at key generation time. Note that the decryption process can remain unchanged.

The system described by van Dijk et al. provides an additional space saving optimization. The public key also includes the number $x_0 = q_0 p + r_0$, generated the same way as the other x_i, with the restrictions that x_0 is the largest element of the public key as well as odd, r_0 is even and x_0 is never chosen as an element of S during encryption. Rather, the ciphertext is reduced modulo x_0 during the encryption process, making the actual encryption equation

$$c = \left(2\sum_{x \in S} x + 2r + m\right) \bmod x_0 \tag{2.3}$$

This additional modification reduces the size of a ciphertext output by the encryption algorithm without impacting the other aspects of the system.

2.2.1 Security

The security of the new public key version of the system relies on the difficulty of two number theoretic problems. The first is the subset sum problem. Given a set $X = \{x_i\}$ and a sum y, is there a subset $S \subseteq X$ such that $\sum_{x \in S} x = y$. An attacker that can solve this problem quickly and knows the value of r in the encryption $c = 2\sum x_i + 2r + m$ can use the solution of the subset sum problem to determine if some subset of the public key K sums to $(c - 2r)/2$.

Name	Meaning	Bound	Suggested Value
λ	security parameter		
ρ	length of noise in public key	$\rho = \omega(\log \lambda)$	λ
ρ'	length of noise during encryption	$\rho' = \omega(\log \lambda)$	2λ
η	length of secret key	$\eta \geq \rho \cdot \Theta(\lambda \log^2 \lambda)$	$\tilde{\mathcal{O}}(\lambda^2)$
γ	length of integers in public key	$\gamma = \omega(\eta^2 \log \lambda)$	$\tilde{\mathcal{O}}(\lambda^5)$
τ	number of integers in public key	$\tau \geq \gamma + \omega(\log \lambda)$	$\gamma + \lambda$
	Parameters Used in Bootstrapping		
κ	precision of elements of $\mathbf{y}$		$\gamma\eta/\rho'$
Θ	number of elements in $\mathbf{y}$		$\omega(\kappa \cdot \log \lambda)$
θ	size of sparse subset $S \subset \mathbf{y}$		λ

Table 2.1: Summary of System Parameters

If it does, then m is equal to zero. Fortunately, the recommended setting of $\rho' = 2\lambda$ makes a brute force search of the noise take exponential time. Further, the subset sum problem is NP-complete, so that the best known algorithm for solving it itself takes exponential time.

The second problem is the approximate gcd problem. Given a set of near multiples of an integer p, that is a set of numbers of the form $qp + r$, the problem asks to recover the integer p. A solution to the approximate gcd problem then recovers the private key to the system from the public key. As it turns out, provided that $\tau \geq \gamma + \omega(\log \lambda)$, this is the easiest attack against the system. Conversely, decreasing τ appears to make the problem itself harder. A more detailed analysis of this problem can be found in section 2.4.

2.2.2 Complexity

In any computational venture, the required resources are an important consideration. Thus, we turn our attention to the complexity of the system.

Space consumption is more straightforward to evaluate, so we look that first before examining time complexity.

There are two major size concerns: key size and ciphertext size. By definition, the size of a ciphertext corresponding to a single bit of plaintext is γ bits; the size of the public key is $\mathcal{O}(\gamma\tau)$ bits. Using the suggested parameters, this results in a ciphertext of size $\tilde{\mathcal{O}}(\lambda^5)$ bits, and public key of size $\tilde{\mathcal{O}}(\lambda^{10})$ bits. Even at trivial values of λ, these both grow rather rapidly. In chapter 4 we examine how large these get in an actual implementation. We also examine how reducing the value of γ improves them. Furthermore, if bootstrapping is applied to the system, it increases the size of both ciphertexts and the public key. Section 6.2 describes the results of our bootstrapping implementation.

Time Complexity

The time complexity of the system is also important to take into consideration. First is key generation. Generating a single integer for the public key involves generating a ρ bit integer r, a $\gamma - \eta$ bit integer q, multiplying q by the η bit secret key p, and adding r to the product. This is dominated by the multiplication. Using the GMP library like we did, for high values of γ this can be accomplished in $\mathcal{O}(\gamma^{1.4})$ time. For the full value of γ suggested by [21] this becomes $\tilde{\mathcal{O}}(\lambda^7)$, while under our relaxed settings it becomes $\tilde{\mathcal{O}}(\lambda^{4.2})$. Now, this must be repeated τ times to generate the entire key, for $\mathcal{O}(\gamma^{2.4})$ time taken total. Again, this expands to $\tilde{\mathcal{O}}(\lambda^{12})$ under the full value of γ and $\tilde{\mathcal{O}}(\lambda^{6.2})$ under our relaxation. Fortunately, this process only needs to happen once for each key.

The second concern is for the time taken to encrypt a plaintext. Encryption involves adding a number of randomly chosen integers from the plaintext, adding in a noise factor ρ' bits long, and taking the modulus with respect to x_0. This is the same as performing modular addition of those public key

integers chosen, which is linear in the size γ of the summands. Then if k integers from the public key are chosen, the overall time to encrypt a single bit plaintext is $\mathcal{O}(k\gamma)$. Now, each addition has the possible effect of increasing the noise by one bit. Since the noise in each key integer is ρ and the noise desired during encryption is ρ', then this becomes $k = \rho' - \rho$. Since $\rho = \lambda$ and $\rho' = 2\lambda$, $k = \lambda$. Therefore, the encryption takes $\mathcal{O}(\lambda\gamma)$ time: $\tilde{\mathcal{O}}(\lambda^6)$ under the original system parameters and $\tilde{\mathcal{O}}(\lambda^4)$ under our relaxed parameters.

2.3 Bootstrapping

Gentry's technique of bootstrapping involves homomorphically evaluating the decryption algorithm. Unfortunately, as described above, the smallest function that decrypts ciphertexts is a constant factor too large to be homomorphically evaluated. Gentry gives in [8] a technique for "squashing" the decryption circuit of his lattice system to a small enough size to be evaluated; the integer-based system uses a similar idea. By embedding additional information about the secret key in the public key, we provide a way to "post process" ciphertexts in a way that makes decryption more efficient. Doing so has the cost of increasing the size of a ciphertext, as well as introducing the additional hardness assumption that the extra information embedded in the public key does not help an attacker break the system.

The bootstrappable construction appends three more parameters: κ, θ and Θ. To the public key, add a set of Θ rational numbers each in the interval $[0, 2)$ with κ bits of precision. Call this set $\mathbf{y} = \langle y_1, \ldots, y_\Theta \rangle$. The set $\mathbf{y}$ must also be chosen so that it contains a sparse subset S of size θ so that $\sum_{y \in S} y \cong 1/p \pmod 2$. The secret key is replaced with the subset S.

The bootstrappable system requires some modification to the encryption algorithm, as well as those for evaluating the addition and multiplication and the decryption algorithm. For the encryption algorithm, generate a ciphertext

c from plaintext m according to equation (2.3) as before. In addition, for every $i \in \{1, \ldots, \Theta\}$ use y_i to create $z_i = (c \cdot y_i) \bmod 2$. Keep only $n = \lceil \log \theta \rceil + 3$ bits of precision after the binary point for each one. Then the ciphertext is composed of c together with $\mathbf{z} = \langle z_1, \ldots, z_\Theta \rangle$. Create the same vector $\mathbf{z}$ for the resultant ciphertext when adding and multiplying ciphertexts.

The decryption algorithm remains largely the same. The original decryption equation could be expressed as

$$m = (c \bmod 2) \oplus (\lfloor c/p \rfloor \bmod 2)$$

Under the bootstrappable system, this can be modified to

$$m = (c \bmod 2) \oplus \left\lfloor \sum_{y_i \in S} z_i \right\rceil \tag{2.4}$$

That is, the sum is taken over those elements z_i with indices matching those elements y_i of $\mathbf{y}$ that are in the subset S.

2.4 Approximate GCD

At the heart of the cryptosystem's security is the *approximate gcd problem*. The problem itself is to recover an integer p from a number of integers $x_1, \ldots, x_n$ all of the form $x_i = q_i + r_i$ for some integers q_i and r_i. This problem is exactly the problem of recovering the secret key p from the public key $\{x_0, x_1, \ldots, x_\tau\}$. Furthermore, if $\tau \geq \gamma + \omega(\log \lambda)$, then any attack on the system can be converted to a solution to the approximate gcd problem in polynomial time; the means the approximate gcd problem is, in one sense, the easiest attack against the cryptosystem.

The first issue in accurately defining the approximate gcd problem is what it means to be an approximate divisor. The simplified statement asks to recover

an integer p from a set of integers known to have the form $x_i = q_i p + r_i$. Of course, any integer at all can be written as $x = qp + r$ for some integer q and some $0 \leq r < p$ by the well-known division algorithm. The difference is that we also require each r_i to be bounded: r_i is a ρ-bit integer, so that $0 \leq r_i < 2^\rho$.

Once the noise is bounded, there is still some ambiguity as to the solution to the approximate gcd problem. After all, any number x is at most 2 above a multiple of 3, at most 4 above a multiple of 5, and so forth. These small approximate divisors are not solutions, and so we need a way to specify which is the number p. The size of p is also assumed to be known, at η bits, as is the fact that p is odd in the given cryptosystem.

2.4.1 Example

Let $p = 15$, so that $\eta = 4$, and let $\rho = 3$. These values for ρ and η are too small to be secure, but they work well for illustrative purposes. With these, let the public key be the three values $37 = 2 \cdot 15 + 7$, $65 = 4 \cdot 15 + 5$ and $107 = 7 \cdot 15 + 2$. The approximate gcd problem in this case is to find an odd integer p' such that $k_1 p' \leq 37 < k_1 p' + 8$, $k_2 p' \leq 65 < k_2 p' + 8$ and $k_3 p' \leq 107 < k_3 p' + 8$. That is, each of the three public values is less than 8 above a multiple of p'. The actual value of $p = 15$ satisfies this; the numbers 3 and 5 both almost do as well, being factors of 15, but are excluded by the size requirement. In fact, any factor of p is shorter than p and would be so excluded, no matter the value of p.

Attempting to brute force every possible noise values for any pair and then using a fast technique to find the gcd will always return p, but this takes exponential time in ρ, and may not return p as a *unique* answer. Likewise, attempting to brute force every η-bit possibility for p is exponential in η, so the public key is resistant to both types of brute force.

Chapter 3

Integer Lattices

Lattices provide a method for combining the power of linear algebra with the discrete settings of number theory, and can provide tools for working with long lists of integers at once. We study them briefly here because lattice-based attacks are among the most promising methods of breaking the approximate gcd problem on which the security of the private key rests.

3.1 Properties

Definition 3 (Lattice)**.** A lattice is a discrete subgroup of $\mathbb{R}^n$, usually one which spans $\mathbb{R}^n$. That is, a lattice is the set of all integral linear combinations of a set $\mathcal{B}$ of linearly independent basis vectors in $\mathbb{R}^n$, similar to how a real vector space is the set of all real linear combinations of a set of basis vectors. The number of elements in the basis for a lattice L is called the *dimension* of L.

Conceptually, a lattice reduces the real vector space $\mathbb{R}^n$ into an integer-based analogue. A lattice is defined in a manner very similar to a vector space; the only difference is that only integer scalar multiples of vectors are allowed,

rather than arbitrary real number scalar multiples. Readers with more experience with algebra will note that a lattice is an instance of a $\mathbb{Z}$-module.

Note that the definition of a lattice restricts the *scalars* that lattice vectors may be multiplied by to the integers; the individual lattice elements are themselves not constrained to having integer coordinates, that is, being elements of $\mathbb{Z}^n$. The lattice itself will be a subset of $\mathbb{Z}^n$ if and only if every basis vector is an element of $\mathbb{Z}^n$.

One of the simplest lattices is the one spanned by the standard basis vectors $e_1 = \left[\begin{smallmatrix} 1 \\ 0 \end{smallmatrix}\right]$ and $e_2 = \left[\begin{smallmatrix} 0 \\ 1 \end{smallmatrix}\right]$, which turns out to be a square lattice, as seen in figure 3.1.

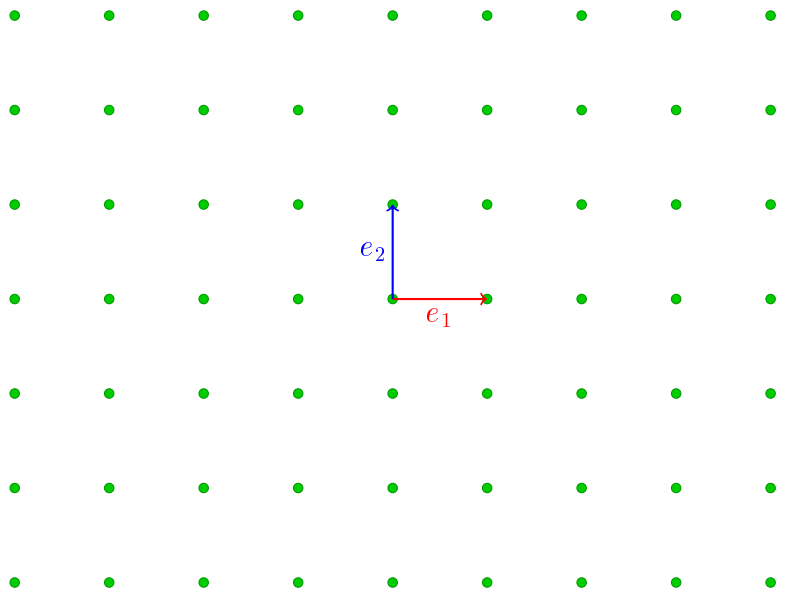


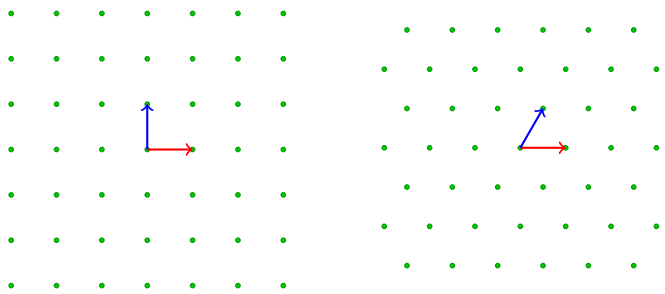
Figure 3.1: Square Lattice Spanned by e_1 and e_2

3.1.1 Different Bases

As described above, an n-dimensional lattice is generated from a basis of n vectors in a manner similar to generating an n-dimensional vector space.

However, the choice of basis is far more important than generating a real vector space. For elements of $\mathbb{R}^n$, there is only one n-dimensional vector space that can be generated; any n linearly independent vectors form a basis for the space $\mathbb{R}^n$. The only difference between bases are their convenience for use.

Lattices, on the other hand, are very dependent on the basis used. Figure 3.2 shows how a simple rotation of one basis vector can dramatically alter the nature of the lattice. However, there is no single, unique basis for a lattice. Many different bases of vectors generate the same lattice. Figure 3.3 exhibits two very different bases that both generate the same lattice.



(a) Square Lattice (b) Hexagonal Lattice

Figure 3.2: Lattices from Different Bases

In a real vector space V, any basis of V can be used to find an orthogonal basis of V, using a Gram-Schmidt or similar process. These basis vectors can all be scaled to unit length to create an orthonormal basis. With lattices this is not entirely possible: most lattices have no orthogonal basis to find, and basis vectors cannot be arbitrarily scaled. Instead, it is desirable to have a basis for a given lattice whose elements are short and nearly orthogonal. Section 3.3 describes one technique for finding such bases and its use in attacking the somewhat homomorphic scheme described in chapter 2.

The process of *lattice reduction* attempts not to create an orthonormal basis, but instead the next best thing: a basis whose elements are as short and

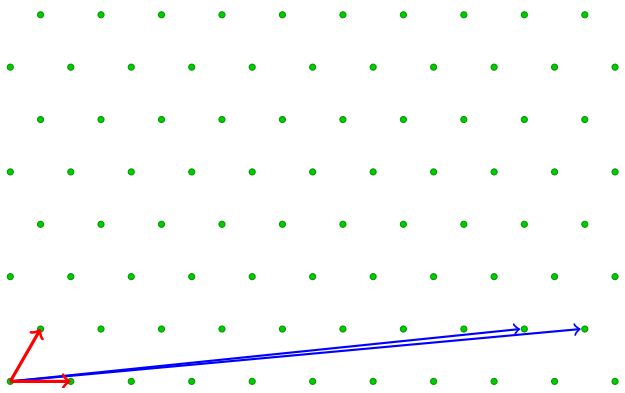


Figure 3.3: Two bases for a Lattice

as close to being orthogonal as possible. Finding the basis closest to being orthogonal is an NP-complete problem; approximate solutions can be found in polynomial time using a number of techniques, such as the LLL lattice reduction algorithm described in section 3.3.

Defining the shortest or most orthogonal basis is itself an interesting problem. The shortest can be taken to be the basis $\mathcal{B}$ that minimizes $\sum_{\mathbf{v}\in\mathcal{B}}\|\mathbf{v}\|$, or it can be the basis that minimizes $\min_{\mathbf{v}\in\mathcal{B}}\|\mathbf{v}\|$. Orthogonality becomes a second condition. For any basis $\mathcal{B} = \{\mathbf{v}_1, \ldots, \mathbf{v}_n\}$ of a lattice L, the n-dimensional volume of the parallelepiped formed by the basis vectors is a constant $d(L)$ that depends only on the lattice L itself. The product of the lengths of the basis vectors is always greater than or equal to $d(L)$, and is equal to $d(L)$ only if the basis is orthogonal. The ratio

$$\delta(\mathcal{B}) = \frac{\prod_{\mathbf{v}\in\mathcal{B}}\|\mathbf{v}\|}{d(L)},$$

known as the orthogonality defect, measures how far from being orthogonal the basis is, where $\delta = 1$ indicates the basis is orthogonal.

3.2 Shortest Vector Problem

Lattices present a number of interesting and useful problems which are interesting in their own right and provide solutions to number-theoretic problems. One lattice problem of particular importance is the *shortest vector problem* (SVP).

Definition 4 (Shortest Vector Problem)**.** Given a lattice L, find a nonzero vector $\mathbf{v} \in L$ such that for every nonzero vector $\mathbf{u} \in L$, $\|\mathbf{v}\| \leq \|\mathbf{u}\|$.

Note that the solution $\mathbf{v}$ to the SVP is not unique. In particular, if $\mathbf{v}$ is a solution to the SVP for a given lattice, then so is $-\mathbf{v}$; in the square lattice shown in figure 3.1, the vectors $e_1, -e_1, e_2, -e_2$ are all valid solutions to the SVP.

The shortest vector problem itself is known to be NP-hard. However, approximate solutions can be found using a number of polynomial time algorithms. Lattice reduction techniques exist for finding short, nearly orthogonal bases in polynomial time. These methods can find vectors within a fixed constant C of being the shortest vector in a lattice L, where C depends only on the dimension of the lattice.

The shortest vector problem has serious implications for the security of the integer-based cryptosystem. There are two basic steps for doing so. First, for any integer x_i in the public key, $\lfloor x_i/q_i \rfloor = p$. Thus, recovering any q_i would allow an attacker to recover p. Next, note that the ratio $y_i = x_i/x_0$ is very close to the ratio q_i/q_0. Thus, the sequence of numbers $\{y_1, y_2, \ldots, y_t\}$ can be approximated by the sequence $\{q_1/q_0, q_2/q_0, \ldots, q_t/q_0\}$. This problem of approximating one sequence of numbers by a sequence of rational numbers, all with the same bounded denominator $q \leq N$, is called *simultaneous Diophantine approximation* (SDA). J. C. Lagarias presents in [13] a method of solving instances of SDA using the LLL algorithm to find a short vector of a specific lattice. We apply his method to the sequence $\{y_1, y_2, \ldots, y_t\}$ as an

attempt to recover the denominator q_0, which we then use to recover p. Section 3.3 describes the LLL algorithm in more detail and section 3.4 describes its use in the SDA problem; our implementation and its results are discussed in chapter 5.

3.3 LLL Reduction

In 1982, Arjen Lenstra, Hendrik Lenstra and László Lovász introduced a method of lattice reduction which runs in polynomial time [14]. This algorithm is known as the LLL algorithm after the creators' initials. As with all lattice reduction algorithms, LLL takes as input a basis for a lattice, typically as given by a matrix whose rows are the basis elements. It transforms this basis into a new basis for the same lattice, where the resulting basis is both shorter and more orthogonal than the original. The act of finding the *most* optimal basis is NP-hard; the LLL algorithm is an approximation algorithm, and so becomes polynomial time by only finding a basis within some known distance of the optimal basis.

Consider the lattice L with the basis $\mathcal{B} = \{\mathbf{b}_1, \mathbf{b}_2, \ldots, \mathbf{b}_n\}$. The first step in LLL basis reduction to form the orthogonal basis $\mathcal{B}^* = \{\mathbf{b}_1^*, \mathbf{b}_2^*, \ldots, \mathbf{b}_n^*\}$ using the Gram-Schmidt process. Then, define a set of scalars $\mu_{i,j} = \dfrac{\mathbf{b}_i \cdot \mathbf{b}_j^*}{\mathbf{b}_j^* \cdot \mathbf{b}_j^*}$. A basis is said to be *LLL reduced* if two conditions are met:

Size Condition

$$|\mu_{i,j}| = \frac{|\mathbf{v}_i \cdot \mathbf{v}_j^*|}{\|\mathbf{v}_j^*\|^2} \leq \frac{1}{2} \quad \text{for all } 1 \leq j < i \leq n.$$

Lovász Condition

$$\|\mathbf{v}_i^*\|^2 \geq \left(\delta - \lambda_{i,i-1}^2\right) \|\mathbf{v}_{i-1}^*\|^2 \quad \text{for all } \ \leq i \leq n.$$

The size condition ensures that the reduced basis is small, while the Lovász condition ensures the basis is close to being orthogonal. The parameter δ is typically taken to be $3/4$, though it can be take any value such that $1/4 < \delta < 1$; if $\delta = 1$ then the LLL algorithm is still defined, though at this point it no longer runs in polynomial time. In general, higher values of δ take longer to run but produce more accurate results. The LLL algorithm produces a lattice basis that satisfies both conditions.

After the Gram-Schmidt basis is determined, we work on one vector $\mathbf{b}_k$ of the basis $\mathcal{B}$ at a time, starting with $\mathbf{b}_2$. The size of $\mathbf{b}_k$ is reduced by subtracting close integer multiples of $\mathbf{b}_j$ for each $j < k$, that is, the Gram-Schmidt vectors corresponding to those lattice vectors that have already been reduced. Then if $\mathbf{b}_k$ satisfies the Lovász condition, move on to the next vector, if not then swap it with the previous vector and move back to vector $\mathbf{b}_{k-1}$. Note that $\mathbf{b}_{k-1}$ is now the vector that was just reduced as $\mathbf{b}_k$. Note that if $\mathbf{b}_2$ is the current vector, so that the vectors $\mathbf{b}_1$ and $\mathbf{b}_2$ were swapped, the reduction is resumed with the new vector $\mathbf{b}_2$ rather than the new $\mathbf{b}_1$. Algorithm 1 presents the LLL algorithm as well, in an easier manner to implement it from than this prose description. Note that the order of the basis vectors $\mathbf{b}_1, \ldots, \mathbf{b}_n$ affects the algorithm, so reordering the basis $\mathcal{B}$ can change the resulting basis.

3.3.1 Example

The LLL algorithm is fairly abstract, and so may not be easily understood at first. Therefore, a concrete example for illustrative purposes presents itself as a good idea. We present the example using the usual value of $\delta = 3/4$.

Input : A basis $\mathcal{B} = \{\mathbf{b}_1, \ldots, \mathbf{b}_n\}$ for a lattice L.
Output: A basis $\mathcal{B} = \{\mathbf{b}_1, \ldots, \mathbf{b}_n\}$ for the lattice L satisfying the Size and Lovász conditions

Set $k = 2$
Set $\mathbf{b}_1^* = \mathbf{b}_1$
while $k \leq n$ **do**
    Recalculate $\mathbf{b}_1^*, \ldots, \mathbf{b}_k^*$ and $\mu_{i,j}$ by using Gram-Schmidt on $\mathbf{b}_1, \ldots, \mathbf{b}_k$
    for $j = 1, 2, \ldots, k-1$ **do**
        `// Size Reduction`
        Set $\mathbf{b}_k = \mathbf{b}_k - \lfloor \mu_{k,j} \rceil \mathbf{v}_j$
    end
    `// Lovász Condition`
    if $\|\mathbf{b}_k^*\|^2 \geq \left(\delta - \mu_{k,k-1}^2\right) \|\mathbf{b}_{k-1}^*\|^2$ **then**
        Set $k = k + 1$
    else
        `// Swap Step`
        Swap $\mathbf{b}_{k-1}$ and $\mathbf{b}_k$
        Set $k = \max(k-1, 2)$
    end
end
Return LLL reduced basis $\{\mathbf{b}_1, \ldots, \mathbf{b}_n\}$

Algorithm 1: LLL Lattice Basis Reduction

Consider the lattice L spanned by the rows of the matrix

$$B = \begin{bmatrix} 1 & 1 & 1 \\ 0 & -2 & 4 \\ 3 & 4 & -2 \end{bmatrix}.$$

Applying Gram-Schmidt gives the orthogonal basis given by the rows of

$$B^* = \begin{bmatrix} 1 & 1 & 1 \\ -2/3 & -8/3 & 10/3 \\ 11/30 & -23/15 & 7/6 \end{bmatrix}$$

with the values $\mu_{2,1} = 2/3, \mu_{3,1} = 5/3, \mu_{3,2} = -29/20$.

We start the reduction with $k = 2$. We take $\mathbf{b}_2 = \begin{bmatrix} 0 & -2 & 4 \end{bmatrix}$ and replace it with $\mathbf{b}_2 - \lfloor \mu_{2,1} \rceil \mathbf{b}_1$. Since $\mu_{2,1}$ rounds to 1, $\mathbf{b}_2$ becomes $\begin{bmatrix} -1 & -3 & 3 \end{bmatrix}$. Then we recalculate $\mathbf{b}_1^*, \mathbf{b}_2^*$ and $\mu_{2,1}$, and compare $\|\mathbf{b}_2^*\|$ to $\|\mathbf{b}_1^*\|$. Since $\|\mathbf{b}_2^*\|^2 \geq (3/4 - \mu_{2,1}^2)\|\mathbf{b}_1^*\|^2$, we continue to $k = 3$.

Now we replace $\mathbf{b}_3$ with $\mathbf{b}_3 - \lfloor \mu_{3,1} \rceil \mathbf{b}_1^* - \lfloor \mu_{3,2} \rceil \mathbf{b}_2^*$. That is, $\mathbf{b}_3 - 2\mathbf{b}_1 + \mathbf{b}_2$. This turns $\mathbf{b}_3$ into $\begin{bmatrix} 1 & 0 & 0 \end{bmatrix}$. Again, we compare $\|\mathbf{b}_3^*\|$ to $\|\mathbf{b}_2^*\|$. Then we must reapply Gram-Schmidt to our current version of B once again. At this point,

$$B = \begin{bmatrix} 1 & 1 & 1 \\ -1 & -3 & 3 \\ 1 & 0 & 0 \end{bmatrix}$$

In this case, the row $\begin{bmatrix} 1 & 0 & 0 \end{bmatrix}$ will fail the Lovász condition and be swapped with the second row. Then we turn our attention to the row $\begin{bmatrix} 1 & 0 & 0 \end{bmatrix}$ as the new second row. It will again fail the Lovász condition and be swapped with

the first row. At this point, we now have

$$B = \begin{bmatrix} 1 & 0 & 0 \\ 1 & 1 & 1 \\ -1 & -3 & 3 \end{bmatrix}.$$

Then size reducing $\mathbf{b}_2$ once more turns it into $\begin{bmatrix} 0 & 1 & 1 \end{bmatrix}$, which passes the Lovász condition. Thus, we pass on to the last row. We apply one last size reduction, which turns the basis matrix into

$$B = \begin{bmatrix} 1 & 0 & 0 \\ 0 & 1 & 1 \\ 0 & -3 & 3 \end{bmatrix}.$$

At this point, the final row passes the Lovász condition, so we terminate the algorithm.

3.4 LLL Attack

The LLL attack uses has at its heart the LLL algorithm for lattice reduction. Lagarias' method for solving an instance of simultaneous Diophantine approximation involves forming a lattice based on the numbers to be approximated and the bound N on the approximation denominator. Then the LLL algorithm is applied to the lattice to find a new basis of short vectors, with the shortest basis vector being taken as an approximate shortest vector. The entries of this vector are related to the approximations $\{q_1/q_0, \ldots, q_t/q_0\}$. Note that an attacker can disregard integers from the public key other than x_0; this has the effect of attacking a smaller, but still valid, public key. Attacking a smaller key is faster, but more sensitive to noise and thus less likely to correctly retrieve the private key.

Lagarias builds his lattice to be reduced as follows: Let $\{a_1/b_1, a_2/b_2, \ldots, a_t/b_t\}$ be the list of numbers to approximate. The approximations are a list of rational numbers all with denominator q. Let $B = b_1 b_2 \cdots$; using the Lagarias technique as an attack on the integer cryptosystem has $a_i = x_i$ and $b_i = x_0$ for every $1 \leq i \leq t$ so that $B = x_0^t$. Let N be the desired upper bound on the denominator q. In the attack, the attacker wishes to recover $q = q_0$. Since p is η bits long and $x_0 \approx q_0 p$ is γ bits long, q_0 is bounded above by $2^{\gamma-\eta}$, so let $N = 2^{\gamma-\eta}$. Then take the lattice spanned by the rows of the matrix

$$
\begin{aligned}
M' &= \begin{bmatrix}
NB & 0 & 0 & \cdots & 0 \\
0 & NB & 0 & \cdots & 0 \\
0 & 0 & NB & \cdots & 0 \\
\vdots & \vdots & \vdots & \ddots & \vdots \\
0 & 0 & \cdots & NB & 0 \\
NB\frac{a_1}{b_1} & NB\frac{a_2}{b_2} & \cdots & NB\frac{a_t}{b_t} & 2^{\gamma+\rho-\eta}
\end{bmatrix} \\
&= \begin{bmatrix}
2^{\gamma-\eta}x_0^t & 0 & 0 & \cdots & 0 \\
0 & 2^{\gamma-\eta}x_0^t & 0 & \cdots & 0 \\
0 & 0 & 2^{\gamma-\eta}x_0^t & \cdots & 0 \\
\vdots & \vdots & \vdots & \ddots & \vdots \\
0 & 0 & \cdots & 2^{\gamma-\eta}x_0^t & 0 \\
2^{\gamma-\eta}x_0^{t-1}x_1 & 2^{\gamma-\eta}x_0^{t-1}x_2 & \cdots & 2^{\gamma-\eta}x_0^{t-1}x_t & 2^{\gamma+\rho-\eta}
\end{bmatrix}.
\end{aligned}
$$

The algorithm as described by Lagarias uses 2^j and iterates over j; however, in our case the length of the desired solution q_0 is already known to be $\gamma - \eta$. Specifically, we have $x_i/x_0 = (q_i + s_i)/q_0$ where the error term $s_i \sim 2^\rho$. That is, $q_0 x_i = q_i x_0 + s_i x_0$. The new error term $s_i x_0$ is approximately $2^{\gamma+\rho-\eta}$; the appearance of this value in M' is not completely arbitrary.

Reducing every row of M by the constant $2^{\gamma-\eta}x_0^{t-1}$ gives a new lattice containing the lattice spanned by the rows of M', so the target solution still exists in the new lattice. Then, rearranging the basis vectors and reordering

the dimensions of the lattice have no effect on the lattice itself, and can be expressed by permuting the rows and columns of the basis matrix. Lastly, any basis element can be replaced with its negation without effecting the lattice, so the LLL reduction can therefore be performed on the lattice corresponding to the new matrix

$$M = \begin{bmatrix} 2^\rho & x_1 & x_2 & \cdots & x_t \\ & -x_0 & & & \\ & & -x_0 & & \\ & & & \ddots & \\ & & & & -x_0 \end{bmatrix}.$$

The target solution to the SDA of $q_1/q_0, q_2/q_0, \ldots, q_t/q_0$ corresponds to the vector $\mathbf{v} = \langle q_0, q_1, \ldots, q_t \rangle \cdot M = \langle q_0 2^\rho, q_0 x_1 - q_1 x_0, \ldots, q_0 x_t - q_t x_0 \rangle$, which has length approximately $2^{\gamma+\rho-\eta}\sqrt{t+1}$. The smaller the value of t, the more efficient the method, though the accuracy decreases as well; small values of t increase the probability that vectors smaller than the target $\mathbf{v}$, obscuring $\mathbf{v}$. Conversely, as t grows, the probability that $\mathbf{v}$ is the vector found grows, as does the computation time required to find it.

Chapter 4

Cryptosystem Implementation

The theoretical background above describes the creation of a fully homomorphic encryption scheme. Now we take a look at the issue of implementing the system, including performance results. Currently, the performance of the system is the largest obstacle to its success; we examine some parameter adjustments to improve this situation.

4.1 Implementation Details

All programs used to implement and test the fully homomorphic encryption system for this paper were compiled and run on a single laptop computer. The programs had approximately 3GiB of memory available and ran on a 64-bit processor; no effort was made to use multiple cores.

The implementation programs were written in C++ and compiled using gcc version 4.5. The code may be downloaded from the RIT Digital Media Library. You may use and modify according to the terms of the GNU General Public License version 2, available with the code, or any later version at your

option. Compiling and running the code yourself requires the GNU Multiprecision (GMP) library and Number Theory Library (NTL) for the numeric computations as well as the GNU `longopts` library.

The core of the implementation is the file `homomorphic.cpp` with its associated header file `homomorphic.h`. The files `keygen.cpp`, `encrypt.cpp` and `decrypt.cpp` provide simple command-line programs for using the functionality that `homomorphic.cpp` provides. The file `compare-png.cpp` homomorphically evaluates whether an encrypted input is a PNG format image; this serves as a simple reference for the system's performance and demonstrates its correctness. Lastly, the file `lll.cpp` implements the LLL-based attack described in chapter 5.

4.2 Suggested Parameters

Consider the various complexities of the integer based system relative to the various definable parameters. For now, we shall keep things simpler by deferring discussion of the complexity bootstrapping adds to section 6.2. A single freshly encrypted ciphertext, corresponding to a single bit plaintext, is γ bits long. A single element of the public key is as well; since there are $\tau + 1$ such elements, the public key is $\mathcal{O}(\gamma\tau)$ bits in size. Thus, the values of τ and especially γ dominate the size complexity of the system.

Recall as seen in table 2.1 that the suggested values of γ and τ are $\gamma = \tilde{\mathcal{O}}(\lambda^5)$ and $\tau = \gamma + \lambda$. This gives that a single bit plaintext is encrypted by a ciphertext of size $\tilde{\mathcal{O}}(\lambda^5)$ bits and the public key has size $\tilde{\mathcal{O}}(\lambda^{10})$ bits. For a moderately small security level $\lambda = 64 = 2^6$, this gives ciphertexts on the order of 128MiB and a public key of 128PiB. These are far too large for everyday use, and attempting to store the public key itself requires around one million dollars worth of hard drives when using consumer-grade products.

Even a trivially weak setting of $\lambda = 8$ gives ciphertexts of 4KiB and a public key of 128MiB. This is at least computationally manageable, though completely insecure and still larger than desired. A summary of key generation tests involving the default parameters is provided in table 4.1. Note that an attempt to raise λ to 16 with these settings wrote approximately 15GiB to disk after one hour, but was unable to finish key generation. Every test with a larger value for λ crashed after running out of memory.

λ	Time Taken (s)	Public Key Size (MiB)
2	2	0.15
4	0.4	4.7
8	60	474

Table 4.1: Key generation with default γ and τ

4.3 Relaxed Parameters

The size of the integers involved is the primary issue with the efficiency of the partially homomorphic encryption scheme. Decreasing the values of some of the parameters given in table 2.1 can lower the work required to use the system; this reduction must be made carefully so as to preserve the security of the system. The size of the public key is $\mathcal{O}(\gamma\tau)$ and the size of a ciphertext corresponding to a one-bit plaintext is $\mathcal{O}(\gamma)$. Thus, we focus primarily on the value of γ.

4.3.1 Relaxing γ

The values suggested by van Dijk et al. give $\gamma = \tilde{\mathcal{O}}(\lambda^5)$ and $\tau = \gamma + \lambda$, so that the message expansion is $\tilde{\mathcal{O}}(\lambda^5)$, and the size of the public key is $\tilde{\mathcal{O}}(\lambda^{10})$ bits. As an extreme attempt at reduction, we set $\gamma = \lambda^{2.5}$. Note that η must be

at least $\tilde{\mathcal{O}}(\lambda^2)$ to allow bootstrapping, and each $x_i = q_i p + r_i$ must be longer than p, a value of $\gamma = \lambda^{2.5}$ approaches the limits of correct computation of the system. Tables 4.2 and 4.3 summarize the results of this relaxation on performance.

λ	γ	τ	Time Taken (s)	Public Key (MiB)
16	1024	1040	0.364	4.2
64	32768	32832	86.795	386
256	1048576	2048	707.162	625

Table 4.2: Relaxed γ key generation

As table 4.2 shows, the reduction of γ to $\lambda^{2.5}$ has a dramatic effect on the complexity of the cryptosystem. Compared to the case $\lambda = 8$ with the suggested value of $\gamma = \lambda^5$, the relaxed parameters achieve a smaller key size in only double the time, while nominally multiplying the security parameter by eight. For the case where $\lambda = 256$, the key size was still prohibitively large, so τ was decreased further than the reduction in γ alone would bring it. Of course, increasing the value of λ is meaningless if the actual security of the system is decreased by relaxing γ.

λ	Plaintext Size (B)	Ciphertext Size (KiB)	Encryption Time (s)	Decryption Time (ms)
16	29	72.891	0.057	18.5
16	40	99.617	0.072	24.0
16	238	580.70	0.346	70.0
64	29	2312.8	0.390	258.9
64	40	3160.8	21.344	348.0
64	238	18425.2	134.869	1879.6

Table 4.3: Relaxed γ encryption and decryption

The performance of both the encryption and decryption algorithms is reasonably fast. Both are roughly linear with respect to the input size, taking

approximately half of a second to encrypt each byte at security $\lambda = 64$, and less than a tenth of a second to decrypt the corresponding ciphertexts. Note that while table 4.2 lists the results of individual key generations, the results in table 4.3 are the averages over eight trials for each plaintext at each security level.

Unfortunately, the results of the encryption are not entirely good ones. Though the time taken is fairly small, the resulting ciphertexts are not. At the level of $\lambda = 64$, each ciphertext requires approximately 80KiB for every byte of plaintext; the expansion ratio is around 80000. Part of this is due to the implementation. For portability and code simplicity, a ciphertext c is output in ASCII using the base 10 representation of c. That is, a single digit of c takes a byte of output. On average, a decimal digit encodes $\log_2 10 = 3.3$ bits, so 3.3 bits are encoded by 8 bits. Thus, the most efficient encoding of the ciphertext c would be approximately half the size it currently is. However, a ciphertext c is a γ bit number before encoding, so the expansion ratio for the cryptosystem is at a minimum γ.

As seen before, the value $\gamma = \lambda^{2.5}$ is approaching the minimum possible value of γ, so ciphertexts cannot be made much smaller without some way to bypass either the bound of γ on expansion or the current value of η, which would allow further reduction of γ. As a small side topic, [21] mentions a method of compressing the ciphertexts output by this system in a way that they can still be correctly decrypted. However, doing so renders the compressed ciphertexts useless for the fully homomorphic encryption, so that it is only practical for reducing transmission costs of an encrypted computation result. Furthermore, the technique for bootstrapping increases the size of the resulting ciphertexts even more. Above the small size of the plaintexts tested, the storage requirements to use the integer based system become prohibitively large.

4.3.2 Relaxing τ

Relaxing γ in section 4.3.1 leads to impressive gains in performance; unfortunately, it is not necessarily secure. The high value suggested for γ is designed to make the system resistant to lattice-based attacks on the public key. Next we relax τ, in the hope that the possible security implications are less severe than those of relaxing γ, though at the cost of less performance gains. Recall that the suggested value of τ depends on the value of γ, so that reducing the value of γ will reduce the numerical value of τ. This is not considered a reduction of τ. Note that γ and τ may be relaxed simultaneously for even higher performance gains if the resulting security implications are considered acceptable.

The first security implication of relaxing τ is on the subset sum problem. Since breaking a known ciphertext can be reduced to an instance of the subset sum problem with τ elements, requiring exponential resources to break the system requires that $\tau \geq \Omega(\lambda)$. This provides a simple lower bound on τ, but does not guarantee complete security on its own.

The other aspect to consider is the role of τ in security of the approximate gcd problem. Reducing τ cannot reduce an attacker's resources required to attack the approximate gcd: even though reducing the dimension of the lattice used in the LLL attack reduces the time taken, the attacker is always able to ignore elements of the public key anyway. Thus, reducing τ in no way improves the attacker's ability to do so; if anything, it forces the attacker to settle for the lower accuracy of the faster attack instead of being able to choose the balance between speed and accuracy.

The other concern for τ and the approximate gcd is not the difficulty of the approximate gcd itself. Rather, it is the relative difficulty of attacking the system using the approximate gcd. As a consequence of the leftover hash lemma, van Dijk et al. show that the approximate gcd is the "easiest" attack

against the integer based system, in the sense that any attack against the system can be used to find a fast solution to the approximate gcd problem. This result holds when $\tau \geq \gamma + \omega(\log \lambda)$. Reducing τ below this simply invalidates the proof of this result. Even if the approximate gcd is no longer the easiest attack, there is no alternative attack vector suggested. Thus, reducing τ only provides a theoretical possibility that a new attack exists; this new attack is not known at this time, nor is its existence even guaranteed.

As seen above, reducing τ appears to be relatively safe. Using $\tau = \lambda$ as a lower bound imposed by the subset sum problem, there are no glaring security holes from reducing τ below the suggested $\gamma + \lambda$. Thus, we take $\tau = \lambda$ as an extreme case for the relaxation of τ. Table 4.4 lists the results of key generation using the relaxed τ. Compare these to the results for reducing γ in table 4.2. In the small case, $\lambda = 16$, the time taken is around ten times that taken for the γ relaxation, though this remains quite small. The public key also increases slightly, though we expect that using $\gamma = \lambda^3$ instead of $\gamma - \lambda^{2.5}$ would have keys the same size as the $\tau = \lambda$ reduction.

λ	γ	τ	Time Taken (s)	Public Key (MiB)
16	1383604	16	2.454	6.9
16	1383604	16	3.646	6.9
16	1383604	16	16.801	6.9
16	1383604	16	2.456	6.9
16	1383604	16	3.699	6.9
16	1383604	16	1.280	6.9
16	1383604	16	8.474	6.9

Table 4.4: Relaxed τ key generation

Unfortunately, as the security parameter grows past $\lambda = 64$, the time required to generate a key is very long. In our tests, the full value of $\gamma = \lambda^5 = 2^{30}$ required slightly over half an hour to generate each integer of the public key; generating even the reduced number $\tau = 64$ integers would require over one

full day. Unfortunately, while relaxing τ does indeed speed up key generation slightly, it does not do so well enough to scale to secure levels.

From this the next step is to examine how keys constructed using a relaxed γ value withstand attacks. We have shown that relaxing γ can bring the time required by the system to manageable levels, though further improvement is of course welcome; relaxing τ would appear to have fewer security concerns for the system, but is unable to provide sufficient performance gains to be viable. Even with the size reduction afforded by relaxing γ, the size of both keys and ciphertexts is very large; if the system retains security under the relaxation, however, it may be possible to modify the system somehow to obtain viable ciphertext sizes.

Chapter 5

LLL-Based Attack

Here we describe our implementation of the LLL based attack described in section 3.4. The attack implementation was carried out under the same hardware and operating environment as the implementation of the system itself — see the beginning of chapter 4. Additionally, the Number Theory Library (NTL) was used for the implementation of the LLL algorithm itself; the GNU Multiprecision Library was again used as the backend for handling the individual arithmetic operations.

5.1 Attack Success

There are two main aspects to the success of using the LLL algorithm to attack this system: whether the attacks works or doesn't, and the time it takes in doing so. For the case of attacking the cryptosystem, these two goals are in opposition to a degree. The LLL attack attempts to retrieve the value of the private key p from a collection $\{x_i\}_{i=0}^{t}$ of the public key integers. Increasing t uses more of the public key integers, meaning more information is used to extract p; this results in a more accurate attack, but at the cost of

increased time taken.

Consider the time complexity of using the LLL algorithm in attacking the given cryptosystem. Suppose that we have a d-dimensional lattice embedded in n-dimensional space, and the longest of these vectors has length B under the standard Euclidean norm. Then the running time of the LLL algorithm is $\mathcal{O}(d^5 n \log^3 B)$. Recall that in using LLL reduction to attack the public key, we construct the $(t+1) \times (t+1)$ matrix

$$M = \begin{bmatrix} 2^\rho & x_1 & x_2 & \cdots & x_t \\ & -x_0 & & & \\ & & -x_0 & & \\ & & & \ddots & \\ & & & & -x_0 \end{bmatrix}$$

Then the lattice spanned by the rows of M is spanned by $t+1$ vectors, each with $t+1$ elements. Because of this, $d = n = t+1$. The longest vector in the given basis is the first row, $\mathbf{b_0} = \langle 2^\rho, x_1, x_2, \ldots, x_t \rangle$. Each x_i is approximately 2^γ, so that $\|\mathbf{b_0}\| \approx \sqrt{2^{2\rho} + t2^{2\gamma}} \approx 2^\gamma \sqrt{t}$. Thus, the running time of the LLL lattice reduction against the cryptosystem is $\mathcal{O}(\gamma^3 t^6 \log^3 t)$. This is polynomial time, but even for the case of $\gamma = \lambda^3$ the complexity is $\mathcal{O}(\lambda^9)$ for any fixed t.

5.1.1 Relaxed γ

The recommended value for γ is at least $\tilde{\mathcal{O}}(\lambda^5)$. At this level, the attack takes approximately 11 hours to run on the relatively small values $\lambda = 16$ and $t = 5$. This low value of t severely impacts the accuracy of the attack, resulting in the "key" recovered being about 800 times longer than the actual secret key. At this level of γ, improving the accuracy by increasing t quickly consumes prohibitively large amounts of memory; section 5.2 explores these memory

usage issues in greater detail. The issue at hand is whether or not reducing the value of γ to around λ^3 is enough to make the attack practical. Since it allows an attacker to use a larger value for t while using comprable time and space, the attacker should be able to increase their accuracy somewhat.

Relaxing the value of γ does indeed speed up the attack, as expected. Even when increasing λ up to 64, the overall value of γ is reduced by a factor of 4 over the full strength test described above, with $\lambda = 16$ and $\gamma = \lambda^5$. Retaining the small value of $t = 5$ cut the running time immensely, from around 11 hours to less than 10 minutes; this would seem to make an attacker's job easier. However, the accuracy of the attack is important as well.

In the case where $\lambda = 16$ and $\gamma = \lambda^5$, the LLL algorithm recovers a number much larger than the private key. With the reduced γ, this attack still recovers an overly large key, though the recovered key is only about 7 times as long rather than the massive disparity seen with the unrelaxed key. However, an attacker is free to use a higher value of t; the low initial times with low t values means that raising t can be somewhat practical. Of course, the running time is still $\mathcal{O}(t^6 \log^3 t)$ for any fixed value of γ so the time taken by using more vectors in the attack grows fairly rapidly. The time complexity is subexponential, but a casual attacker will likely be put off by the time required by the LLL algorithm. Increasing t from 5 to 8, a small change on its own, already brings the running time of the attack to over an hour, with only slight improvement in accuracy.

5.2 Memory Usage

The LLL-based attack has a secondary cost, other than the time taken to run. The amount of memory the attack uses is large as well. Again, the attack

works by reducing the lattice basis defined by the rows of the matrix

$$M = \begin{bmatrix} 2^{\rho} & x_1 & x_2 & \cdots & x_t \\ & -x_0 & & & \\ & & -x_0 & & \\ & & & \ddots & \\ & & & & -x_0 \end{bmatrix}$$

and each x_i is γ bits long. Then, the individual entries of the matrix require a minimum of $2\gamma t + \rho$ bits to store; the use of signed integers increases this requirement, as does the overhead of storing the elements as a matrix. For the parameters we study most, $\lambda = 64$ and $\gamma = \lambda^3 = 2^{18}$, this puts the memory requirements around $(64t)$KiB, just to store the matrix M. For the full value of $\gamma = \lambda^5$, storing M requires about t GiB, though reducing λ to 16 reduces this to approximately the same level as $\lambda = 64$ under the relaxed γ condition.

There is more to the memory usage than simply storing M like this. One issue is that the large values of the individual matrix elements typically require a degree of overhead to store, inflating the size requirements. More importantly, the original matrix M is very sparse, while performing the basis reduction does not necessarily preserve this sparseness. The memory required to store M is linear in t; the memory required to store a full matrix of the same size is quadratic in t. Lastly, these calculations only apply to the storage of a single matrix. The actual running of the LLL algorithm requires more space of the individual operations used, which pushes the costs up even farther. At the modest level of $\lambda = 64$, testing $t = 16$ used about 10MiB. This is small usage on modern machines, but is ten times the memory required to simply store M at that level, and the 16 integers used from the public key represent a tiny portion of the 262208 available in the same key. Thus, the memory usage provides a strong limit on how high an attacker can make t, trading

off speed for accuracy; the number τ of integers in the public key offers only a much weaker bound on t.

Chapter 6

Homomorphic Computation on Ciphertexts

The preceding chapters discussed the general practicality of the integer-based somewhat homomorphic cryptosystem. However, since this system is being used to create a fully homomorphic system, its performance for homomorphically evaluating functions should be examined as well. This falls under two broad headings: the system's performance when evaluating a small Boolean circuit, and the system's performance when performing a single bootstrapping cycle. Since an arbitrary circuit is made by joining smaller circuits using bootstrapping, both have an effect on the overall performance of the system.

6.1 Evaluation

The running time of evaluation of a Boolean circuit C depends on the circuit itself, as well as the size of the ciphertexts involved. Recall that the result of adding two numbers is as long as the longer of the two summands, with a possible carry bit; multiplying two numbers results in a product whose length

is the sum of the lengths of the factors. Thus, the output of each addition or multiplication gate in C is $\mathcal{O}(\gamma)$ bits long, as the input to each one is $\mathcal{O}(\gamma)$ bits long.

The GMP library we used for computation achieves addition of two numbers in linear time, while a variety of multiplication algorithms are used by the library: larger inputs use more asymptotically efficient algorithms as the input reaches certain thresholds. As γ grows very large, the Toom 4-way algorithm used approaches $\mathcal{O}(\gamma^{1.404})$. From this, the time taken to evaluate is dominated by the multiplication gates in C, for an overall runtime of $\mathcal{O}(M\gamma^{1.404})$, where M is the number of multiplication gates in the circuit C.

6.1.1 Bitstring Comparison

We measure the performance of the evaluation with a straightforward test circuit. Files in the PNG format for images start with a well-defined constant header 64 bits long. Comparing the first 64 bits of a file to this header has a simple conversion to a Boolean circuit expressed using our allowed operations of AND and XOR. Let $m_1, m_2, \ldots, m_{64}$ be the first 64 bits of a plaintext file, and let $h_1, h_2, \ldots, h_{64}$ be the 64 bits of the PNG header. Then the file has the PNG header if and only if $m_i = h_i$ for every $1 \leq i \leq 64$.

We can express this condition using the allowed Boolean operations as follows: $m_i = h_i$ if $m_i \oplus h_i = 0$. Equivalently, $m_i = h_i$ if $m_i \oplus h_i \oplus 1 = 1$. Since the h_i are public and constant, we can negate them and input those negations into the comparison circuit; then $m_i = h_i$ if $m_i \oplus \overline{h_i} = 1$. Since matching the header occurs when this holds for every index i, we have

$$\bigwedge_{i=1}^{64} \left(m_i \oplus \overline{h_i}\right) = \begin{cases} 1 & \text{PNG header present,} \\ 0 & \text{PNG header absent.} \end{cases} \tag{6.1}$$

Since this equation uses only the operations of AND and XOR, it can be directly converted into a corresponding equation involving ciphertexts:

$$\prod_{i=1}^{64} \left[\mathcal{E}\left(m_i\right) + \overline{h_i}\right] = \begin{cases} \mathcal{E}\left(1\right) & \text{PNG header present,} \\ \mathcal{E}\left(0\right) & \text{PNG header absent.} \end{cases} \tag{6.2}$$

Note that there is no need to encrypt the value of $\overline{h_i}$ before evaluating equation (6.2). Since the values of h_i are publicly documented, an attacker would already know them. Encrypting them adds no security, and a plaintext works as an encryption of itself for purposes of evaluation. Therefore we forego the extra overhead for encrypting them and operate directly on the plaintext h_i. We tested this 63 multiplication circuit on ciphertexts corresponding encrypted using $\lambda = 64, \gamma = \lambda^3$. The entire circuit was correctly evaluated without bootstrapping in approximately 15 minutes.

6.2 Bootstrapping

Bootstrapping is Craig Gentry's very powerful contribution to fully homomorphic encryption. The bootstrapping technique allows us to convert a somewhat homomorphic system capable of only a bounded number of operations, such as the integer based system given by van Dijk et al, into a fully homomorphic system. Since the purpose of this somewhat homomorphic system is to be used as a bootstrappable system, a full investigation of the system's efficiency requires bootstrapping the system be looked at.

6.2.1 Key Modification

Bootstrapping the integer based system first requires supplementing the public key with a "hint" to the private key. Without this hint, the decryption

function winds up being a constant multiple too large to evaluate homomorphically; adding the hint effectively "squashes" the decryption routine to a small enough size to be correctly evaluated. A bootstrappable public key adds to a normal public key $\{x_i\}_{i=0}^{\tau}$ a set of Θ rational numbers $\{y_i\}_{i=1}^{\Theta}$, each in the interval $[0, 2)$ and with κ bits of precision. The generation of these y_i is done in such a way that a sparse subset of cardinality θ exists whose sum is related to the original public key p. See table 2.1 for a summary of these new parameters in relation to the system parameters not used in bootstrapping.

The sparse subset of $\{y_i\}$ can be used as a private key, and can be encoded in a way that makes it easily encrypted. Overall, the public key has its size increased by $\mathcal{O}(\kappa\Theta)$ bits; with the relaxation $\gamma = \lambda^3$ this becomes $\mathcal{O}(\lambda^9)$, so the public key increases in size from $\mathcal{O}(\lambda^6)$ to $\mathcal{O}(\lambda^9)$. The new private key is represented as a vector of Θ bits, where the ith bit is 1 if y_i is in the sparse subset. Then the bootstrappable private key is $\mathcal{O}(\Theta) = \mathcal{O}(\lambda^5)$ bits long.

The key augmentation itself takes a nontrivial time to complete. At the level $\lambda = 16$, the process takes slightly under 5 minutes. Unfortunately, this security level is far too low for our purposes: for the purposes of this test, we desire a security parameter of at least $\lambda = 64$. At this level, bootstrapping is fairly unusable for members of the general public. Six hours worth of computation was not enough to create even 2% of the set $\{y_i\}$. At this rate, creating a full bootstrappable key takes between one and two *weeks* of computation time.

6.2.2 Recryption

The heart of bootstrapping is the ability to *recrypt* ciphertexts: by homomorphically evaluating the decryption algorithm, a ciphertext is transformed into a different ciphertext corresponding to the same plaintext. The result of recryption is a reduction of the noise in a ciphertext so that it may be further

used in a Boolean circuit. Unfortunately, the process is far from efficient.

Martin van Dijk and his team adapted a technique from Gentry's work in [8] and used it to prove an upper bound on the complexity of decryption. Specifically, they show that using the information generated for bootstrapping allows decryption by a polynomial of degree at most $128\lambda \log^2 \lambda$. With the values $\lambda - 64, \gamma = \lambda^3$ as explored above, this involves almost 300 thousand multiplications. Recall that the PNG header comparison has 63 multiplications, and took approximately 15 minutes at this security level. If the time taken is directly proportional to the polynomial degree, then this takes over 1.5 months worth of computation time.

Of course, this is not feasible to carry out for a John Doe using consumer grade hardware; even for a dedicated cluster or supercomputer this represents a significant drain on resources. Recall that this time represents only a single bootstrapping operation. The choice of η is such that a bootstrapping cycle must be done for nearly every basic operation in a circuit C to be computed, if C is not small enough to be evaluated on its own. Increasing η would allow for more operations to be done in each bootstrapping cycle, as well as allowing for larger circuits to be evaluated without bootstrapping, but would require increasing γ as well. A circuit large enough to require bootstrapping could take thousands of years worth of computing time. Without a massively parallel computer system, this cryptosystem cannot be efficiently bootstrapped.

Chapter 7

Conclusions

7.1 Results Summary

The results of our test were mixed. The integer-based system on its own takes too much time and space to be used at any significant level of security. This was the primary motivation for attempting to relax γ. Relaxing γ did reduce the time taken for the system to generate keys and to encrypt data down to viable levels. It even did so while retaining decent security; the sizes involved kept the LLL-based attack from compromising the system without massive amounts of processing time.

Unfortunately, the rest of the results were rather poor. While key generation and encryption had their times reduced to manageable levels by relaxing γ, the size of the resulting keys and ciphertexts is still excessively high. Furthermore, the time taken to homomorphically evaluate functions on ciphertexts is also high. Very simple circuits can be evaluated quickly, but as the function to evaluate grows in complexity, it becomes infeasible; circuits large enough to require bootstrapping become intractable.

7.2 Future Direction

While the results we obtained were less than promising, the somewhat homomorphic system is not entirely worthless. At the very least, it provides a simple introduction to the concept of fully homomorphic encryption and bootstrapping for classroom settings. Some of the issues are possibly addressable as well. Implementing some form of block encryption could possibly reduce the size of ciphertexts and the complexity of homomorphic evaluation. Also, improving the efficiency of homomorphic decryption, that is, refreshing ciphertexts, provides another possibility for improving the overall system. Since the value of η depends on the complexity of the homomorphic decryption algorithm, improving that complexity allows for a reduction of η, allowing γ to be further reduced.

Improving the underlying system in one of these ways is likely to be difficult. A different approach is to work on a different underlying system to bootstrap. Gentry's original lattice-based system shows more promise in this regard. As presented, Gentry's lattice system is already more efficient than the integer system.

7.3 Final Remarks

The integer-based somewhat homomorphic cryptosystem presents itself as a good proof of concept and educational tool. At this point it fails to be a viable method of fully homomorphic encryption. Our results show that it can be improved, but we fail to improve it to the point of practicality. It may be possible to improve the system further; even if such improvements are not found, the system remains interesting from a theoretical point of view.

References

[1] Riza Aditya, Colin Boyd, Ed Dawson, and Kapali Viswanathan, *Secure e-voting for preferential elections*, Electronic Government, Lecture Notes in Computer Science, vol. 2739, Springer Berlin / Heidelberg, 2003, pp. 1064–1064.

[2] Dan Boneh, Eu-Jin Goh, and Kobbi Nissim, *Evaluating 2-dnf formulas on ciphertexts*, Theory of Cryptography (Joe Kilian, ed.), Lecture Notes in Computer Science, vol. 3378, Springer Berlin / Heidelberg, 2005, pp. 325–341.

[3] Jung Hee Cheon, Woo-Hwan Kim, and Hyun Soo Nam, *Known-plaintext cryptanalysis of the Domingo-Ferrer algebraic privacy homomorphism scheme*, Information Processing Letters vol. 97 (2006), no. 3, 118 – 123.

[4] Bram Cohen, *Bram's public key encryption algorithm*, Web document, 2000, `http://bramcohen.com/simple_public_key.html`.

[5] Josep Domingo-Ferrer, *A new privacy homomorphism and applications*, Information Processing Letters vol. 60 (1996), no. 5, 277 – 282.

[6] Josep Domingo-Ferrer, *A provably secure additive and multiplicative privacy homomorphism*, Information Security (Agnes Chan and Virgil Gligor, eds.), Lecture Notes in Computer Science, vol. 2433, Springer Berlin / Heidelberg, 2002, pp. 471–483.

[7] Rosario Gennaro, Craig Gentry, and Bryan Parno, *Non-interactive verifiable computing: Outsourcing computation to untrusted workers*, Advances in Cryptology – CRYPTO 2010 (Tal Rabin, ed.), Lecture Notes in Computer Science, vol. 6223, Springer Berlin / Heidelberg, 2010, pp. 465–482.

[8] Craig Gentry, *A fully homomorphic encryption scheme*, Ph.D. thesis, Stanford University, 2009, `http://crypto.stanford.edu/craig`.

[9] Craig Gentry and Shai Halevi, *Implementing gentry's fully-homomorphic encryption scheme*, Cryptology ePrint Archive, Report 2010/520, 2010, `http://eprint.iacr.org/`.

[10] Craig Gentry, Shai Halevi, and Vinod Vaikuntanathan, *i-hop homomorphic encryption and rerandomizable Yao circuits*, Advances in Cryptology – CRYPTO 2010 (Tal Rabin, ed.), Lecture Notes in Computer Science, vol. 6223, Springer Berlin / Heidelberg, 2010, pp. 155–172.

[11] Shafi Goldwasser and Silvio Micali, *Probabilistic encryption & how to play mental poker keeping secret all partial information*, Proceedings of the fourteenth annual ACM symposium on Theory of computing (New York, NY, USA), STOC '82, ACM, 1982, pp. 365–377.

[12] Martin Hirt and Kazue Sako, *Efficient receipt-free voting based on homomorphic encryption*, Advances in Cryptology — EUROCRYPT 2000 (Bart Preneel, ed.), Lecture Notes in Computer Science, vol. 1807, Springer Berlin / Heidelberg, 2000, pp. 539–556.

[13] Jeffery C. Lagarias, *The computational complexity of simultaneous diophantine approximation problems*, Annual IEEE Symposium on Foundations of Computer Science (1982), 32–39.

[14] Arjen K. Lenstra, Hendrik W. Lenstra, and László Lovász, *Factoring*

polynomials with rational coefficients, Mathematische Annalen vol. 261 (1982), 515–534, 10.1007/BF01457454.

[15] Eric Levieil and David Naccache, *Cryptographic test correction*, Public Key Cryptography – PKC 2008 (Ronald Cramer, ed.), Lecture Notes in Computer Science, vol. 4939, Springer Berlin / Heidelberg, 2008, pp. 85–100.

[16] Hsiao-Ying Lin and Wen-Guey Tzeng, *An efficient solution to the millionaires' problem based on homomorphic encryption*, Applied Cryptography and Network Security (John Ioannidis, Angelos Keromytis, and Moti Yung, eds.), Lecture Notes in Computer Science, vol. 3531, Springer Berlin / Heidelberg, 2005, pp. 97–134.

[17] Kun Peng, Riza Aditya, Colin Boyd, Ed Dawson, and Byoungcheon Lee, *Multiplicative homomorphic e-voting*, Progress in Cryptology - INDOCRYPT 2004 (Anne Canteaut and Kapaleeswaran Viswanathan, eds.), Lecture Notes in Computer Science, vol. 3348, Springer Berlin / Heidelberg, 2005, pp. 1403–1418.

[18] Ronald Rivest, Leonard Adleman, and Michael Dertouzos, *On data banks and privacy homomorphisms*, Foundations of Secure Computation, Academic Press, 1978, pp. 169–177.

[19] Ronald L. Rivest, Adi Shamir, and Leonard Adleman, *A method for obtaining digital signatures and public-key cryptosystems*, Commun. ACM vol. 21 (1978), 120–126.

[20] Nigel P. Smart and Frederik Vercauteren, *Fully homomorphic encryption with relatively small key and ciphertext sizes*, Public Key Cryptography – PKC 2010 (Phong Nguyen and David Pointcheval, eds.), Lecture Notes in Computer Science, vol. 6056, Springer Berlin / Heidelberg, 2010, pp. 420–443.

[21] Marten van Dijk, Craig Gentry, Shai Halevi, and Vinod Vaikuntanathan, *Fully homomorphic encryption over the integers*, Advances in Cryptology – EUROCRYPT 2010 (Henri Gilbert, ed.), Lecture Notes in Computer Science, vol. 6110, Springer Berlin / Heidelberg, 2010, pp. 24–43.

[22] David Wagner, *Cryptanalysis of an algebraic privacy homomorphism*, Information Security, Lecture Notes in Computer Science, vol. 2851, Springer Berlin / Heidelberg, 2003, pp. 234–239.

[23] Andrew C. Yao, *Protocols for secure computations*, Annual IEEE Symposium on Foundations of Computer Science (1982), 160–164.

[24] Andrew C. Yao, *Protocols for secure computations*, Annual IEEE Symposium on Foundations of Computer Science (1982), 160–164.