Lattice Based Cryptography and Learning With Errors

Ani Nadiga

2019

1 Why Lattice Based Cryptography?

At a high level, all cryptographic schemes use a problem that is "difficult" to solve in order to manipulate information. Specifically, an encryption scheme encrypts a message in such a way that decrypting the message requires solving a difficult instance of a problem. RSA, the most common form of encryption, uses the difficulty of factoring large integers to encrypt information. Lattice based cryptographic schemes base their security on the difficulty of certain problems regarding lattices. Lattice based schemes offer three major advantages over currently used schemes:

1. They are secure against attacks from quantum adversaries
2. They are just as secure in the worst and average case
3. The only known fully homomorphic encryption schemes are lattice based

Each of these properties is more rigorously defined and discussed below.

Post Quantum Cryptography The solution to a general class of problems will be an algorithm that will find an answer to each of the problems in that class. The measure of how long an algorithm takes is called its complexity, and if its complexity is a polynomial function of its inputs we call it efficient. If all of the known algorithms to solve a class of problems are inefficient, then we say that the problem is difficult or computationally infeasible.

Algorithms can be either classical or quantum. As the names imply, a classical algorithm is one that could be executed using a classical computer, and a quantum algorithm would require a quantum computer to be executed. Classical computers, which store information using 0-1 bits, are ubiquitous and fairly well understood. Quantum computers, on the other hand, store information using quantum-bits that can be any value between 0 and 1. There are certain problems that are difficult to solve using classical computers, meaning that any known classical algorithm for that problem takes a long (super-polynomial) amount of time (or resources) to complete. There appear to be some problems that are

difficult for classical algorithms but are not for quantum ones. Unfortunately, this includes almost all of the problems that are used in current cryptography. (For example, Shor's quantum algorithm efficiently factors integers.) Thus, if a viable quantum computer was ever built it would be able to crack all of the cryptography that we rely on. This raises the challange of building a classical encryption scheme (i.e. one that classical computers can use) that is resistant to attacks from quantum adversaries.

Theoretically, building a large quantum computer is possible. Fortunately for cryptographers (and the population at large), there seem to be a number of engineering obstacles to constructing large scale quantum computers. While these computers are being developed, it will be important to discover and implement new cryptographic schemes that are based on problems that are difficult for quantum computers as well as classical computers.

The most promising such schemes are latticed based, meaning that they base their security on the "quantum-difficulty" certain lattice problems. These lattice problems are generally conjectured to be difficult even for quantum algorithms. Thus, cryptosystems that use certain lattice problems to manipulate or encrypt data are secure against attacks from both classical and quantum computers.

Average/Worst Case Reductions For many problems, the difficulty of solving certain instances of the problem is very different from the difficulty of solving an "average" instance of that problem. For example, factoring integers that are the product of two large primes of about the same size is much more difficult than factoring some random integer. This is why in RSA encryption the integer that is used to encrypt information is always the product of two large primes. This variability in difficulty introduces a problem: All cryptosystems in the real world are based on some specific instance of a problem. If an algorithm that can efficiently solve this specific instance is discovered, then the security of the scheme is compromised. Thus the constant worry is that some new result in mathematics may compromise a specific scheme.

For many classes of lattice problems, it has been shown that the difficulty of solving the average case of the problem is just as difficult as solving the worst case of the problem. This is done by first assuming that an algorithm that efficiently solves the average case of the problem exists, and then showing that this algorithm can be used as a subroutine in another algorithm to efficiently solve the worst case of that problem. Thus, if a lattice based scheme is based on a problem that falls into one of these classes, the worst case hardness of the problem will guarantee the security of implementation of the scheme, even if these implementations use an average case version of the problem. This releases cryptographers from the perpetual worry that their specific instance of a scheme may be compromised.

This property of lattice problems also lends some credibility to the conjecture that they are computationally unfeasible even for quantum algorithms. Over the course of decades, there has not been significant progress in solving even the average case of these problems, let alone the worst case.

Fully Homomorphic Encryption Currently, data can be stored in an encrypted format, but must be decrypted before performing any manipulations. This means that only trusted computers can process sensitive data, which can becomes a problem if the computational power needed is large. A fully homomorphic encryption scheme would allow data to be encrypted in such a way that it can be processed without being decrypted. This would enable you to encrypt both your data and some function that you want to be performed on the data, and then give both to an untrusted computer. The untrusted computer would return the encrypted result of the function performed on the data, which only you could decrypt. The computer, however, would not know anything about the data, the function it effectively performed, or the result of the function on the data! More concretely, suppose that two numbers a and b have homomorphic encryptions $\text{Enc}(a)$ and $\text{Enc}(b)$. Then given only $\text{Enc}(a)$ and $\text{Enc}(b)$ one can find $\text{Enc}(a+b)$ and $\text{Enc}(a \cdot b)$ (with out knowing either a or b). This magical sounding idea has only been implemented using certain lattice based schemes.

Outline

In this paper we will discuss the merits and short comings of lattice based cryptography. We begin with some background on lattices and lattice problems. Then we will discuss the Learning With Errors (LWE) problem. The LWE problem has been shown to be as hard as certain lattice problems. Most "lattice based" schemes actually use LWE or some variant of LWE instead of lattices directly. However, since breaking the cryptosystem would require solving the LWE problem efficiently, the fact that LWE is as hard as certain lattice problems will assure the security of the scheme. In the next section we will present simple a public key cryptosystems based on LWE. Finally we will present a fully homomorphic encryption scheme based on LWE. We finish with a discussion of the state of lattice based cryptography as a whole.

2 Lattices and Lattice Problems

2.1 What is a Lattice

The idea of a lattice is very intuitive. The mental image that you may have of a repeating pattern of points is an accurate depiction of a 2-dimensional lattice. More formally, a lattice is a discrete additive subgroup of $\mathbb{R}^n$. A basis of a lattice is a set of vectors in the lattice such that any lattice vector can be written as a linear combination of the basis vectors.

Obviously, we can only visualize lattices up to 3 dimensions, but the concept of a lattice generalizes to higher dimensions. For the lattice problems that are discussed below, the dimension of the lattice will be large, often in the hundreds.

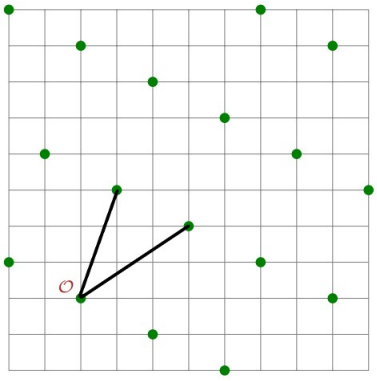

Figure 1: A 2-dimensional lattice and a set of two vectors that form a basis

2.2 The Many Lattice Problems

Although there are many lattice problems, almost all of the ones relevant to cryptography have something to do with the shortest vector of that lattice. The simplest one is the Shortest Vector Problem (SVP). As its name implies, the SVP is to find the shortest vector of a lattice given some arbitrary basis of the lattice. This may at first glance seem simple. However, the fact that we are given an arbitrary basis for the lattice and not a "nice" one will make this problem difficult. A good basis would be one with short basis vectors that are "as orthogonal as possible". On the other hand, a "bad" basis may have long vectors that point in almost the same direction.

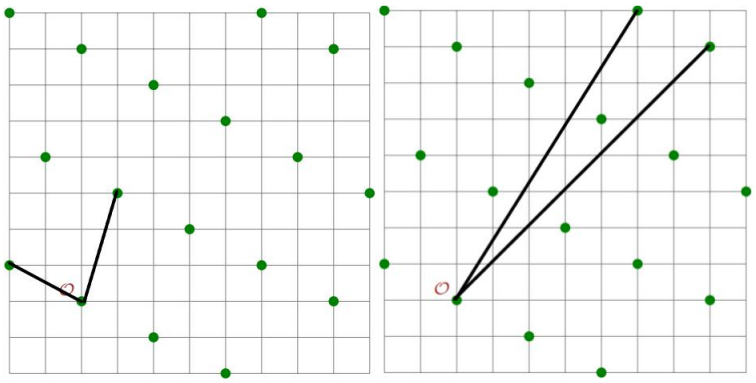

Figure 2: A "good" basis and a "bad" basis

If we tried to solve the Shortest Vector Problem using the basis given on the left, it seems easy. If we imagine being given the basis on the right, this

problems seems much more difficult. We can note that there are few "good" bases, while there are (infinitely) many bad ones. Thus, if we are given an arbitrary basis, with overwhelming probability the shortest vector will not only not be a basis vector, it will also not even be close to a basis vector! In general, the SVP seems difficult in high dimensions with arbitrary basis.

A variant of this problem is called GapSVP$_\beta$. Informally, the GapSVP$_\beta$ problem is to decide if the shortest vector of a lattice is "long" or "short" (relative to β). More formally, an algorithm must decide, given a basis of a lattice, if the shortest vector in the lattice has norm less than 1 or greater than β. In more symbolic terms, if we have a lattice L with shortest vector of length $\lambda(L)$, then given a basis B of L, the GapSVP$_\beta$ problem is to decide if $\lambda(L) \leq 1$ or if $\lambda(L) > \beta$. If neither case is true then the algorithm is allowed to return anything. Like with SVP, the "quality" of the basis will clearly affect the difficulty of solving the GapSVP$_\beta$ problem.

2.3 The Quantum Computational Complexity of Lattice Problems

Obviously, one would hope that there is a proof lattice problems are difficult for both classical and quantum algorithms. However, as Regev states, the best evidence for the quantum difficulty of these lattice problems is "that there are no known quantum algorithms for lattice problems that outperform classical algorithms, even though this is probably one of the most important open questions in the field of quantum computing." [1]This may initially seem like an unconvincing argument. However, current cryptography is based on similar assumptions about integer factoring; there are no proofs about the existence or non-existence of an algorithm to efficiently factor arbitrary integers. However, the fact that the efforts of decades of mathematics has yielded no such algorithm would seemingly indicate that no such algorithm exists.

3 The Learning With Errors Problem

Before discussing the Learning With Errors problem, we will describe a much simpler problem that is analogous. If you are given the numbers 20,500,790,40,70 and asked what they all have in common, the answer seems simple; they are all multiples of ten. On the other hand, if you are given 11,40,651,49,560, then the pattern may be less obvious. The answer here is that they are all multiples of 10 with either -1,0, or 1 added. At a high level, the LWE problem is similar; there is a secret vector that is multiplied (through the dot product) by many other vectors and then some small error is added. The problem is to find the the secret vector.

3.1 Problem Definition

The Learning With Errors problem is parameterized by some positive integer modulus q and some error distribution χ over the ring $\mathbb{Z}_q^n$. Suppose that there is some "secret vector" $\mathbf{s} \in \mathbb{Z}_q^n$ that is sampled uniformly randomly from $\mathbb{Z}_q^n$. Then pick many $\mathbf{a}_i$ uniformly randomly from $\mathbb{Z}_q^n$ and $e_i \leftarrow \chi$. For each $\mathbf{a}_i$ and e_i, define $b_i = \mathbf{s} \cdot \mathbf{a}_i + e_i$. Then make the pairs $(\mathbf{a}_i, b_i)$. The $\text{LWE}_{q,\chi}$ problem is to figure out $\mathbf{s}$ given the many pairs.

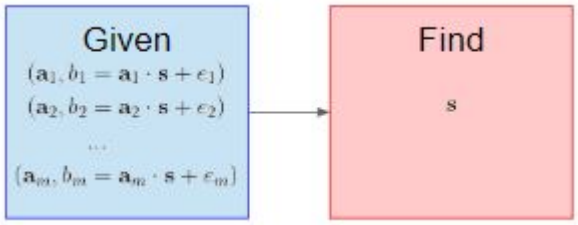

Figure 3: A depiction of the LWE problem (note that one does not have access to the information on the right side of the equations, they only have the value of b_i

Technically, this is called the search LWE problem, since the problem is to "search" for the secret given the many pairs. There is different variant of this problem called the decision LWE problem. The decision LWE problem is to decide, given many pairs, if they are of the LWE "form" or if they are just totally random. More formally, if one is given many pairs $((a)_i, b_i)$ and an error distribution χ over $\mathbb{Z}_q^n$, the decision LWE problem is to decide if there exists some $\mathbf{s} \in \mathbb{Z}_q^n$ such that every $b_i = \mathbf{a}_i \cdot \mathbf{s} + e_i$ for $e_i \leftarrow \chi$.
[1]

3.2 Lattice Problem to LWE reduction

Not all instances of the search LWE problem are computationally infeasable. For instance, if χ is always 0, then one can simply find $\mathbf{s}$ through Gaussian elimination. However, it has been shown that for certain constraints on the parameters, the seach LWE problem is quantum and clasically "difficult".

In his paper, Regev establishes the computational infeasiblity of the search LWE problem by showing that if there exists and algorithm to efficiently solve the search LWE problem, then that algorithm can be used to solve the GapSVP problem. More precicely, he shows a quantum algorithm to solve GapSVP that contains an algorithm that solves search LWE as a subroutine. The complexity of GapSVP algorithm depends on the complexity of the LWE algorithm that it

[1] The fact that the search LWE problem is computationally infeasible for quantum computers may be surprising for those who are familiar with the Bernstien-Vazarani Problem, and the associated quantum algorithm that can efficiently solve it. If we set $q = 2$ then LWE samples are essentially samples of the Berstien-Vazarani type but with a small amount of error.

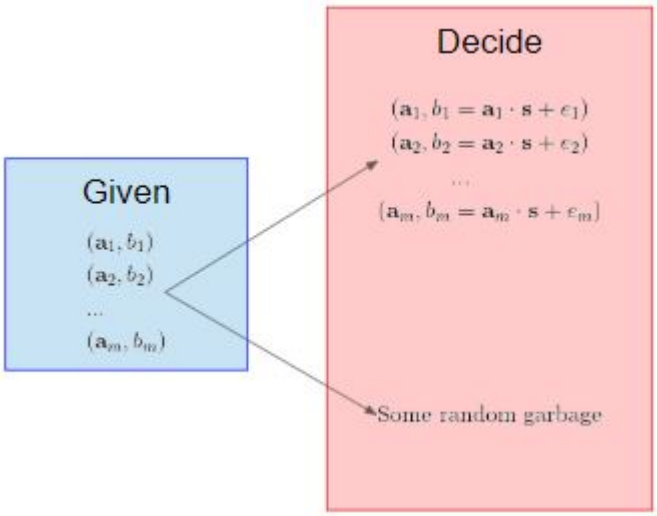

Figure 4: A deptiction of the decision LWE problem

uses. If the LWE algorithm is efficient then the GapSVP algorithm will be as well. Thus, if GapSVP is quantum computationally infeasible, then there must not exist an efficient algorithm (quantum or classical) for solving the search LWE problem.

In the paper, the reduction from the lattice problem to the search LWE problem is not simple. First, Regev shows that if one can efficiently solve the search LWE problem, then one can efficiently sample the *discrete Gaussian* distribution with certain parameters. The n-dimensional discrete Gaussian distribution is the discrete analogue of the continuous n-dimensional Gaussian. This reduction makes use of a quantum algorithm. Then Regev shows that if one can efficiently sample the discrete Guassian distribution, then one can solve the GapSVP problem efficiently. Thus, being able to efficiently solve the search LWE problem would allow one to efficiently solve the GapSVP problem.

3.3 Search Decision Equivalence

The decision LWE problem is more= useful for cryptographic applications. Therefore, we are more interested in the computational infeasablility of the decision LWE problem than that of the search LWE problem. An interesting property of these problems, however, is that the decision problem is at least as hard as the search problem. Regev shows this be showing that if one has an algorithm that can solve the decision LWE problem, then that algorithm can be used to solve the search LWE problem (which was proved to be computationally infeasible) [1].

He starts by assuming that there is some algorithm that can efficiently dif-

ferentiate between LWE type samples and uniformly random samples (i.e. an algorithm that can efficiently love the decision LWE problem). He uses this algorithm to find $\mathbf{s}$, coordinate by coordinate. To find the first coordinate s_1, there are q possible choices (because $\mathbf{s} \in \mathbb{Z}_q^n$). So one can run through each of these possible values. Let us "guess" that $s_1 = k$. Now, given any LWE sample $(\mathbf{a}, b = \mathbf{a} \cdot \mathbf{s} + e)$, we can note that if we add l to the first coordinate of $\mathbf{a}$ then b will increase by $\mathbf{s}_1 \cdot l$. If our guess k is correct, then $(a + (\overset{l}{0}\overset{}{\underset{0}{\vdots}}), b + k \cdot l)$ will be a "proper" LWE sample. If our guess for k is wrong then it will not be. We can now pass this modified sample to the efficient decision LWE algorithm to learn if it is a "proper" LWE sample or not. Thus we can find the value of s_1 by running through the possible values of k, and we can repeat this for each coordinate of $\mathbf{s}$

3.4 Average Case/Worst Case Reduction

Many times, proofs of security will rely on worst case assumptions, rather than average case ones. This means that if the worst case is actually very unlikely and the average case is much easier, the security of the cryptosystem can be compromised. However, for the LWE problems. Regev shows that with in certain constraints on the parameters, the worst case difficulty of the problem is no more than that of the average case. Essentially, he shows an algorithm that, given an algorithm that can solve the average case LWE problems efficiently, can "amplify the success" to efficiently solve LWE problems in the worst case. This method essentially takes the average of many runs of the average case algorithm to guess solutions to the worst case LWE problem.

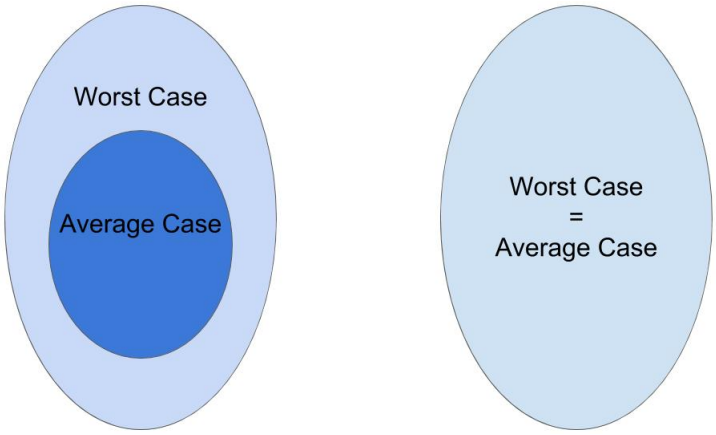

Figure 5: A visualization of the parameters a problem is difficult. On the left is the situation for which the parameters that make the average case difficult are different from the parameters for the worst case. On the right, the computational difficulty of the average and worst case are the same.

3.5 Why LWE is good for cryptography

There are a number of properties of the decision LWE problem that make it particularly useful for cryptographic applications. Each of these properties have already been discussed, but not in the context of cryptography. The first property is that the LWE problems seem to be computationally infeasable for even quantum computers. This means that a cryptosystem based on LWE could be run on classical computers, but could be safe from attacks from quantum adversaries.

The second property is the average/worst case computational complexity of the LWE problems. For the problems that are used for current cryptography, the average and worset case instances of a problem are not equally difficult. Thus, the perpetual worry is that some way will be discovered to solve the specific instance of the problem that is currently being used. On the other hand, with in certain constraints, the LWE problems are as hard in the average case as they are in the worst case. This means that if one finds a way to solve certain intsances of the problem then they can solve all of them. Thus, given that LWE problems generally seem to be difficult, one can confidently pick a specific instance of an LWE problem to base a crytosystem on.

The final benefit of the LWE problems is that they have some kind of natural algebraic structure. We can componenent-wise add LWE samples or we can take their tensor products. This will be useful when trying to implement fully homomorphic encryption. Recall that a fully homomorphic encryption scheme is one such that given the encryptions of two numbers, $\text{Enc}(a)$ and $\text{Enc}(b)$, one can find $\text{Enc}(a+b)$ and $\text{Enc}(a \cdot b)$ (with out knowing a or b). We will see later that fully homomorphic encryption schemes based on LWE problems "inherit" some kind of algebraic structure from the space of LWE samples.

4 LWE based Cryptosystems

In this section we present an LWE based public key cryptosystem. A public key cryptosystem has a secret key and a public key. Any one can use the public key to encrypt a message, but only someone with the secret key can decrypt a message.

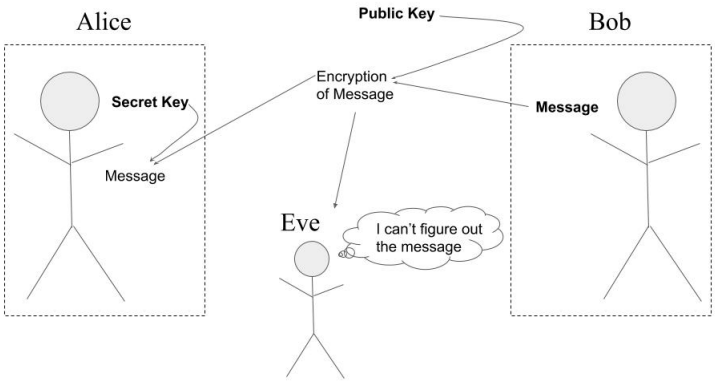

Figure 6: A public key cryptosystem: Alice decides on a secret key and keeps it to herself. Then she uses the secret key to generate a public key. Bob uses this public key to encrypt his message. Alice can use her secret key to decrypt the message. The adversary Eve can not figure out what the message is using publicly available knowledge (the public key and the encryption of Bob's message).

4.1 Review of LWE

LWE problems are parameterized by the error distribution χ and the secret vector $\mathbf{s}$ (since $\mathbf{s} \in \mathbb{Z}_q^n$ this implicitly sets n and q). We define the LWE distribution $A_{\mathbf{s},\chi}$ as the distribution of samples $(\mathbf{a}, b = \mathbf{a} \cdot \mathbf{s} + e)$, where $\mathbf{a}$ is a uniformly random vector of $\mathbb{Z}_q^n$ and e is sampled from χ.

4.2 Building up to the Cryptosystem

Note that $b - \mathbf{a} \cdot \mathbf{s} = e$. This leads to a possible way to encrypt a one bit message x. Let Alice's secret key be $\mathbf{s}$ and the public key be some sample from $A_{\mathbf{s},\chi}$, $(\mathbf{a}, b)$. Then, to encrypt $x \in \{0, 1\}$, Bob modifies the public key to $(\mathbf{a}', b') = (\mathbf{a}, b + \lfloor \frac{q}{2} \rfloor x)$. To decrypt the message, Alice calculates $b' - \mathbf{a}' \cdot \mathbf{s} = e + \lfloor \frac{q}{2} \rfloor x$. If this value is closer to 0 than to $\frac{q}{2}$, then she concludes that $x = 0$, and otherwise she concludes that $x = 1$. As long as the error term e is small enough ($< q/4$) then this will clearly work. This can be guaranteed with high probability by choosing a χ with a low standard deviation.

Now we pose two questions. The first is whether, given publicly available information, one can find the secret key. The public key is simply an LWE sample, and the secret key is the secret vector of the sample. Thus the task of finding the secret key given the public key is simply the search LWE problem. Since the search LWE problem is computationally infeasible, we conclude that this is impossible.

The second question is whether an adversary can observe the encryption of some messages and then figure out the messages. Unfortunately, for this scheme the answer is yes. If Eve intercepts an encryption that Bob sends out then she can simply compare it to the public key. If the encryption is the same as the public key, then clearly Bob's message is a 0, and if it is different then it must be a 1.

To deal with this, we could imagine a slightly different scheme. Alice's secret key would still be some secret vector $\mathbf{s}$. The public key would now be a large set of m LWE samples with secret $\mathbf{s}$. Then to encrypt, Bob simply picks one of the samples at random and then encrypts as before. However, Eve can still break this system easily. She can simply see if an encryption that Bob releases is in the public key or not. If it is in the public key the the message is most likely 0, and if it is not then the message is most likely 1. For a final solution we want many possible ciphertexts which do not "pair up" with the elements of the public key.

4.3 The Scheme

The final scheme will be as before, except Bob will use some random subset of the LWE samples in the public key. Let the m LWE samples in the public key be indexed by $(\mathbf{a}_i, b_i)$. The way in which Bob will use random subsets of the m LWE samples in the public key is as follows. He will pick a random bit string $S \in \{0,1\}^m$, and create the new "sample" $(\mathbf{a}, b) = (\sum_{i \in S} \mathbf{a}_i, \sum_{i \in S} b_i)$. This new "sample" is simply a subset sum of the m original samples. Then Bob uses this new sample to encrypt his message the same way he did in the first iteration of the scheme (he outputs the ciphertext $(\mathbf{a}, \mathbf{b} + \lfloor \frac{q}{2} \rfloor)x)$. Alice decrypts as before, but now $b - \mathbf{c} \cdot \mathbf{s} = \sum_{i \in S} e_i + \lfloor \frac{p}{2} \rfloor x$. Thus, the condition for correct decryption is that the sum of many e_i must be small.

Informally, it makes sense that the scheme is secure because there is no clear way to "match" a given encryption to an element of the ciphertext. A more rigorous discussion of security that relies on properies of subset sums will follow later.

The scheme is described more formally below:

Let n be the security parameter of the cryptosystem. This means that the higher n is, the more computationally difficult it will be to crack the cryptosystem. We pick q, the integer modulus, to be some prime number between n^2 and $2n^2$. We pick m, the number of LWE samples in the public key, to be $(1+\epsilon)(n+1)\log q$, where $\epsilon > 0$ is some arbitrary constant. The error distribution χ is taken to be a discrete Gaussian distribution over $\mathbb{Z}_q^n$ whose standard deviation is $\alpha(n) = o(1/(\sqrt{n}logn)$. All of the operations that follow are performed

mod q.

- **Private Key:** Choose some uniformly random vector $\mathbf{s} \in \mathbb{Z}_q^n$ for the secret key.
- **Public Key:** For $i = 1, ..., m$ pick $\mathbf{a}_i$ uniformly randomly from $\mathbb{Z}_q^n$ and e_i from χ. Then we let $b_i = \mathbf{a}_i \cdot \mathbf{s} + e_i$. The public key is the set of LWE samples $(\mathbf{a}_i, b_i)_{i=1}^m$
- **Encryption:** Pick a random $S \in \{0,1\}^n$. Then let $(\mathbf{a}, b) = (\sum_{i \in S} \mathbf{a}_i, \sum_{i \in S} b_i)$. To encrypt a bit $x \in \{0,1\}$, we let the encryption be $(\mathbf{a}', b') = (\mathbf{a}, b + \lfloor \frac{q}{2} \rfloor x)$
- **Decryption:** To decrypt a pair $(\mathbf{a}', b')$, consider $b' - \mathbf{a}' \cdot \mathbf{s}$. If this value is closer to 0 than $\lfloor \frac{q}{2} \rfloor$, the decryption is 0, otherwise it is 1.

4.4 Correctness of the Scheme

Obviously, we would like for the decyption of a ciphertext to be the actual message that was sent. Recall that to decrypt we "round" the quantity $b - \mathbf{a} \cdot \mathbf{s} = \sum_{i \in S} e_i + \lfloor q/2 \rfloor$. In order for this rounding to work correctly, we need the quantity $\sum_{i \in S} e_i$ to be very small. Regev shows that for our choice of parameters, the probability that the sum of errors drawn from χ is greater than $q/2$ negligible. Thus, the probability that the subset sum of the errors will change the way that the rounding works is negligible as well, and the scheme will almost always work correctly.

4.5 Security of the Scheme

In order for the scheme to be secure, an adversary should not be able to distinguish encryptions of 0 and encryptions of 1. Regev shows that if there is some polynomial time algorithm that can differentiate between encryptions of 0 and 1, then there is some algorithm that can distinguish between LWE samples and uniformly random samples with a significant chance of success. Since we assume that decision LWE is computationally infeasible, this would imply that there does not exist a polynomial time algorithm that distinguishes between encryptions of 0 and 1.

4.6 Key Size

In order for a cryptosystem to be viable in the real world, the key sizes must be. We see that the size of the public key is $O(mn \log q)$. We define $\tilde{O}$ to be the complexity ignoring all logarithmic factors. Thus, given the definition of m, the size of the public key is $\tilde{O}(n^2)$. This is a fairly large key size, as the security factor will often be in the many hundreds.

One idea to reduce the key size is to distribute to first half of the public key, $\mathbf{a}_1, ..., \mathbf{a}_m$, before hand. Then if any one wanted to set up a public key system they would simply need to pick a secret and generate the b_i that is

associated with each $\mathbf{a}_i$. In a real world setting we could imagine that the vectors $\mathbf{a}_i$ are included in the security software that is distributed to users, and then users simply have to generate the last half of the public key in order to set up a public key cryptosystem. This reduces the key size to $\tilde{O}(n)$, which is a substantial improvement.

5 Fully Homomorphic Encryption from LWE

Recall that a fully homomorphic encryption scheme is a scheme such that given the encryption of two numbers a and b, one can find the encryption of $a + b$ or ab, with out knowing a or b. The methods that define a fully homomorphic encryption scheme are the same as those that define a public key system (SecretKeyGen, PublicKeyGen, Enc, Dec), in addition to the methods Add and Mult, which describe how to homomorphically add and multiply ciphertexts.

We will first present the methods SecretKeyGen, PublicKeyGen, and Enc. Then we will describe Add and Mult. Finally, we will describe Dec.

In order to make computation and implementation easier, the scheme is presenting using slightly different notations from the public key cryptosystem presented above. Specifically, it will allow us to describe the methods in terms of matrix and vector operations. However, it works in much the same way. One of the key differences, however, is that the secret is drawn from the error distribution χ rather than uniformly randomly. Another distinction is that the error that is added is always even. This means that it does not change the parity of what it is added to. Decryption in this scheme works by looking at the parity of something related to the ciphertext rather than the magnitude of it, like was done in the public key system.

- SecretKeyGen: Draw a secret vector $\mathbf{s}'$ from χ^n. Then make the secret key $\mathbf{s} = (1, \mathbf{s}') = (1, \mathbf{s}'[1], ..., \mathbf{s}'[n]$.

- PublicKeyGen: Uniformly randomly sample a matrix $\mathbf{A}'$ from $\mathbb{Z}_q^{N \times n}$. The N rows of this matrix are analogous to the $\mathbf{a}_i$ that were used in the public key system. Then we sample a vector $\mathbf{e} \leftarrow \chi^N$, and calculate $\mathbf{b} = \mathbf{A}'\mathbf{s}' + 2\mathbf{e}$. This vector is analogous to the b_i from the previous scheme. The public key is the matrix $\mathbf{A}$ whose first column is $\mathbf{b}$ and whose subsequent columns are the columns pf $-\mathbf{A}'$. Thus, each row of $\mathbf{A}$ is an LWE sample with an even error.

- Enc: To encrypt a message $m \in \{0, 1\}$, we first set $\mathbf{m} = (m, 0, ..., 0) \in \mathbb{Z}_q^{n+1}$. Then we pick a random bit string $\mathbf{r} \in \mathbb{Z}_2^N$. This is like the S that was chosen in the public key system. We output the ciphertext $\mathbf{c} = \mathbf{m} + \mathbf{A}^T\mathbf{r}$. Just like in the public key system, this is like taking a random subset sum of the LWE samples in the public key and adding the message to the b part of the sum.

- Dec: Output $[[\langle \mathbf{c}, \mathbf{s} \rangle]_q]_2$. Just like in the previous scheme, we first take

$\langle \mathbf{c}, \mathbf{s} \langle$. This will leave us with the sum of some random even errors plus the message

5.0.1 Homomorphic Addition

If we have two ciphertexts $\mathbf{c}_1$, $\mathbf{c}_2$ which encrypt messages m_1, m_2, then the sum $\mathbf{c}_1 + \mathbf{c}_2$ will be a valid encryption of $m_1 + m_2$

5.0.2 Homomorphic Multiplication

Given two ciphertexts $\mathbf{c}_1$, $\mathbf{c}_2$ which encrypt messages m_1, m_2 under the secret key $\mathbf{s}$, we want to find a ciphertext for $m_1 m_2$. We can do this by taking the tensor product of the two ciphertexts, $\mathbf{c}_1 \otimes \mathbf{c}_2$. However, the secret for this ciphertext will no longer be $\mathbf{s}$, it will be $\mathbf{s} \otimes \mathbf{s}$. This means that in order to decrypt the ciphertext we will compute $[[\langle \mathbf{c}_1 \otimes \mathbf{c}_2, \mathbf{s} \otimes \mathbf{s} \rangle]_q]_2$.

5.1 Problems with the Scheme

5.1.1 Key and Ciphertext Growth

This ciphertext has two coordinates. Informally, we can view the first as some padding of a message (that makes it impossible to read), and the second as some information which, in combination with the secret, allows us to "unpad" the message.

When we multiply two ciphertexts they go from being decryptable under the secret $\mathbf{s}$ to being decryptable under $\mathbf{s} \otimes \mathbf{s}$. Intuitively, what is happening is that when we tensor the two ciphertexts, the amount of information that is needed for decryption increases.

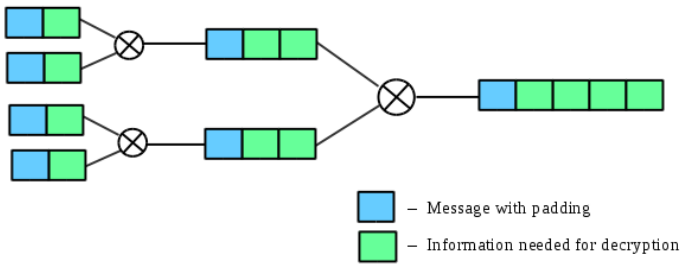

Figure 7: Caption

In this scheme, homomorphically multiplying two ciphertexts that are encrypted under a secret $\mathbf{s}$ of dimension n , will result in a ciphertext that is encrypted under $\mathbf{s} \otimes \mathbf{s}$, which has dimension n^2. Then, as we homomorphically multiply this chiphertext with other ciphertexts, the size of the ciphertext and the size of the secret key grow exponentially. This makes the scheme as it his impractical for real world use.

5.1.2 Noise Growth

The noise associated with a ciphertext $\mathbf{c}$ (under the secret key $\mathbf{s}$) is $|\langle \mathbf{c}, \mathbf{s} \rangle|$. If this value exceeds $q/2$ then there will be decryption error.*

If two ciphertexts have noise at most B, their sum will have noise at most $2B$. This means that the noise grows about linearly under addition.

On the other hand, the product of the ciphertexts will have noise B^2 (under the new secret key $\mathbf{s} \otimes \mathbf{s}$). This means that the noise grows exponentially as a function of the number of multiplications. This means that we can do at most SOME ASYMPTOTIC NOTATION BULLSHIT HERE.

5.2 Solution to Problems

In this section we will present an algorithm that will allow us to switch the secret key that is needed to decrypt ciphertexts which will function with out actually decrypting the ciphertext. More formally, if we have a ciphertext $\mathbf{c}$ that encrypts a message m under the secret key $\mathbf{s}$, we will define a transformation that would allow us to transform $\mathbf{c}$ to $\mathbf{c}'$, which encrypts m under a new secret key $\mathbf{s}'$.

Remember that if we take the product of ciphertexts, we get some new ciphertext $\mathbf{c}_2 \otimes \mathbf{c}_1$ that encrypts $m_1 \cdot m_2$ under the key $\mathbf{s} \otimes \mathbf{s}$. With the algorithm described above, we can transform this to a new ciphertext that is an encryption of $m_1 \cdot m_2$ under the original secret key $\mathbf{s}$. This addresses the issue of growing key size.

5.2.1 Key Switching

The algorithm that is described below can effectively switch between any two secret keys. However, we will describe it in the context of switching between $\mathbf{s} \otimes \mathbf{s}$ and $\mathbf{s}$, which will allow us to deal with the issue of key length growth. The cost will be a small increase in the noise of the cipertext that is output. The key switching will happen in two parts. First a matrix will be generated using $\mathbf{s} \otimes \mathbf{s}$ and $\mathbf{s}$. Then we wil simple multiply the old ciphertext by this matrix and it will give us the new ciphertext. We will see that the cost of this operation will be a small additive increase in the noise of the ciphertext.

5.3 Circular Security

Although we have proved that it is safe to encrypt a random message using the cryptosytem, there is one potential issue with the key switching algorithm. We can essentially view the matrix that was generated to enable key switching an an encryption of $\mathbf{s} \otimes \mathbf{s}$ under the secret key $\mathbf{s}$. However, we only know that the cryptosystem is secure for random messages. It has not been proven that it is possible to safely encrypt messages that relate to the secret. The ability to encrypt the secret of a cryptosytem securely is called circular security.

For this cryptosystem, there is no guarantee of circular security. On the other hand, there is no proof that it is impossible to encrypt the secret safely.

So we are left with a choice: we can either assume circular security in which case the scheme as described is completely functional, or we can choose to not assume circular security. If we do not assume circular security then we need to find a new way to address the key size growth.

What if instead of switching from $\mathbf{s} \otimes \mathbf{s}$ to $\mathbf{s}$, we intead switched to $\mathbf{s}'$ for some different $\mathbf{s}'$? Then we would avoid the issue of circular security. In fact, we could make a full sequence of secret keys $\mathbf{s}_1, ..., \mathbf{s}_k$. Then, after the first homomorphic encryption is carried out, the secret key will be $\mathbf{s}_1 \otimes \mathbf{s}_1$. We can key switch from this to $\mathbf{s}_2$ securely. Thus we can continue down the "ladder" of secrets until we have reached the last one. The size of this ladder will be determined by the amount that the noise grows with each multiplication and by the noise growth of each key switch. Specifically, we must ensure that the total noise is low enough to allow for proper decryption.

References

[1] Oded Regev. On lattices, learning with errors, random linear codes, and cryptography. *J. ACM*, 56(6):34:1–34:40, September 2009.

[2] Vadim Lyubashevsky, Chris Peikert, and Oded Regev. On ideal lattices and learning with errors over rings. *J. ACM*, 60(6):43:1–43:35, November 2013.

[3] Zvika Brakerski, Craig Gentry, and Vinod Vaikuntanathan. (leveled) fully homomorphic encryption without bootstrapping. *ACM Trans. Comput. Theory*, 6(3):13:1–13:36, July 2014.