Math 259 Class Notes
Spring 2019

Christelle Vincent

April 3, 2019

Contents

Chapter 1

Introduction

These notes will be updated as the course progresses; you can always see if you have the most up-to-date version of the notes by checking the date on the first page. I will also try as much as possible to keep a changelog so you can see how your version differs from the latest version (so you don't miss any updates).

These notes will **not** be complete. Whenever a topic is covered adequately in the literature, I will give references rather than write anything down. In particular, I will refer to our two recommended course books:

- Simon Rubinstein-Salzedo's *Cryptography*, which I will call SRS's *Cryptography* for short, and
- Wade Trappe and Lawrence C. Washington's *Introduction to Cryptography with Coding Theory*, which I will call TW's *Coding Theory* for short and contrast with the other book.

1.1 Important note

Throughout the course, we will simply assume that all messages we want to send are numbers (where here by "number" we mean an element of $\mathbb{Z}/N\mathbb{Z}$ for some N, or a finite field, or...) Depending on the source, some authors make a big deal of how to translate English/letters into numbers. While this is of considerable interest to some, this will not come up in this class. In the kind of cryptography we will be interested in, we can assume that there is one publicly known way to translate between English and any number system we are using. This assumption does not compromise the security of the cryptography we are using.

Chapter 2

Classical public-key cryptography

In this Chapter we will cover only RSA and DLP over finite fields.

2.1 RSA

RSA stands for Rivest-Shamir-Adleman, who are the people who came up with it. The references to learn about RSA are:

- SRS's *Cryptography*, Section 12.1
- TW's *Coding Theory*, Section 6.1
- Wikipedia's article on RSA

2.1.1 Required skills/knowledge

1. Definition of the Euler-totient function, although in the case we care about (and this does require for some assumptions to be satisfied!) we have that $\varphi(pq) = (p-1)(q-1)$.
2. Euler's Theorem, which says that if a and N are two integers with $\gcd(a, N) = 1$, then
$$a^{\varphi(N)} \equiv 1 \pmod{N}.$$
3. Given two integers a, b with $\gcd(a, b) = 1$, be able to compute $c \equiv a^{-1} \pmod{b}$. When we do this by hand we would usually use the Extended Euclidean Algorithm. For this class, it will be sufficient to be able to do this with a computer (for your homework) or be able to look it up in a table (for the quiz).

For graduate students:

4. The Chinese Remainder Theorem

2.1.2 Hard and easy things

For the purposes of problems, you may assume that the following operations are "easy" and can be performed quickly and efficiently by all:

- Addition, subtraction, multiplication modulo N for any N
- Exponentiation modulo N for any N, i.e. computing $a^e \pmod N$ for any integers a and e
- Computing the modular inverse $e^{-1} \pmod n$ for any pair (e, n) such that $\gcd(e, n) = 1$
- Using the Chinese Remainder Theorem
- Computing the greatest common divisor (gcd) of any two integers

However, the following operations are "hard" and given numbers that are *suitably chosen*, they cannot be performed quickly or efficiently:

- Factoring, and by extension computing $\varphi(N)$

2.1.3 Notation we will use

Alice chooses two primes p and q, and obtains a *modulus* $N = pq$. The Euler-totient function is either $\phi(N)$ or $\varphi(N)$, depending on taste (I personally like $\varphi(N)$ so that is what I will use but there shouldn't be confusion). She also chooses a positive integer e which is relatively prime to $\varphi(N)$ and which is called her *encryption exponent*. Then $d \equiv e^{-1} \pmod{\varphi(N)}$ is her *decryption exponent*.

The public key is (N, e) and $\varphi(N)$ and d are kept secret.

2.2 DLP

DLP stands for "discrete log problem," because the security of this scheme has to do with how difficult it is to compute logarithms in discrete groups. Another name for this scheme is ElGamal with underlying group $(\mathbb{Z}/p\mathbb{Z})^\times$ (ElGamal considers a general group, but we only consider this specific group).

The references to learn about DLP are:

- SRS's *Cryptography*, Chapters 11.2 and 11.3
- TW's *Coding Theory*, Chapters 7.1 and 7.5

2.2.1 Required skills/knowledge

1. Roughly what a primitive root is (mostly just that g should be a primitive root modulo p, but you do not need to know how to find a primitive root modulo p)

2. We will quickly practice how to compute with discrete logarithms

For graduate students:

3. What a primitive root modulo N is

2.2.2 Hard and easy things

For the purposes of problems, you may assume that the following operations are "easy" and can be performed quickly and efficiently by all:

- Addition, subtraction, multiplication modulo N for any N
- Exponentiation modulo N for any N, i.e. computing $a^e \pmod N$ for any integers a and e
- Computing the modular inverse $e^{-1} \pmod n$ for any pair (e, n) such that $\gcd(e, n) = 1$

However, the following operations are "hard" and given numbers that are *suitably chosen*, they cannot be performed quickly or efficiently:

- Computing the discrete logarithm, i.e. given $h \equiv g^a \pmod p$, with both h and g known, computing the exponent a

2.2.3 Notation we will use

Alice chooses a prime p and a primitive root g modulo p. She also chooses a *secret exponent* a, and computes and publishes $h \equiv g^a \pmod p$. To encrypt his message, Bob will also need a *random secret exponent* b, and he will compute and send the *ciphertext pair* (c_1, c_2), where

$$c_1 \equiv g^b \pmod p \quad \text{and} \quad c_2 \equiv mh^b \pmod p, \tag{2.1}$$

where m is his secret message.

The public key is (p, g, h), and both exponents a and b must be kept secret. The ciphertext pair c_1 and c_2 is public.

Chapter 3

Shor's algorithm

In this part of the class we will study how one can factor an integer N is (probabilistic) polynomial time in $\log N$ using a quantum computer. What this means in practice is that with a quantum computer, the "hard"/time-consuming problem of factoring is in fact an "easy"/quick-to-solve problem.

If a quantum computer could be built at scale, this would thus render RSA unacceptably insecure according to the current standards to which we hold cryptographic algorithms. Although we will only study the algorithm that factors numbers, a variation of Shor's algorithm can also be used to solve DLP (for any group, including ECDLP) in probabilistic polynomial time.

3.1 Quantum computing basics

Coming soon (?)

3.2 Discrete Fourier Transform

3.2.1 Required skills/knowledge

3.2.2 Roots of unity

We have the following:

Proposition 3.2.1. *Let $n \geq 1$ be an integer. Then there are n distinct nth roots of unity, given by*

$$1, e^{2\pi i \frac{1}{n}}, e^{2\pi i \frac{2}{n}}, e^{2\pi i \frac{3}{n}}, \ldots, e^{2\pi i \frac{n-2}{n}}, e^{2\pi i \frac{n-1}{n}}. \tag{3.1}$$

Furthermore, there are $\varphi(n)$ distinct primitive *nth roots of unity, which are of the form*

$$e^{2\pi i \frac{k}{n}} \tag{3.2}$$

above, but with $\gcd(k, n) = 1$.

About the primitive nth roots of unity we have the following result:

Proposition 3.2.2. *Let $\zeta_n \neq 1$ be any other nth root of unity. Then*

$$\sum_{k=0}^{n-1} \zeta_n^k = 1 + \zeta_n + \zeta_n^2 + \cdots + \zeta_n^{n-2} + \zeta_n^{n-1} = 0.$$

3.2.3 The Discrete Fourier Transform

Definition 3.2.3. Let $a_0, a_1, a_2, \ldots, a_{m-1}$ be a sequence of complex numbers of length m. Then the *Discrete Fourier Transform* (DFT) of this sequence is another sequence $b_0, b_1, \ldots, b_{m-1}$, also of length m, such that the jth term of the sequence is

$$b_j = \frac{1}{\sqrt{m}} \sum_{k=0}^{m-1} a_k e^{\frac{2\pi i}{m} jk}. \tag{3.3}$$

Note that in fact any other primitive mth root of unity could be used in place of $e^{\frac{2\pi i}{m}}$, but we choose this one to keep things concrete.

To compute the DFT of a sequence in Sage, you can use the following code. First enter

```
J = list(range(9))
A = [1,2,3,1,2,3,1,2,3]
s = IndexedSequence(A,J)
```

In the first line, you should replace 9 with the length of your own sequence, and then in the second line you can put any sequence you want for A. Then s is the object that Sage can actually interact with; you must tell Sage that the elements of the sequence A are indexed by the elements of the list J.

Once you have an indexed sequence, you can ask Sage to compute the DFT:

```
DFT = s.dft()
```

That's it!

Now you might want to see the DFT you just computed. To see the terms one-by-one as exact terms, with roots of unity, you can type

```
for a in DFT.list():
    print a
```

To print floating point approximations of the terms, in the form `x + y * I`:

```
for a in DFT.list():
    print CC(a)
```

And finally, to print the absolute values of the terms, to see which terms are large and which terms are small:

```
for a in DFT.list():
    print CC(a).abs()
```

Let's define one more term:

Definition 3.2.4. Let $\{a_k\}$ be a periodic sequence with period p and length m. Then the *frequency* of this sequence is the expression

$$f = \frac{m}{p}.$$

Note that the frequency need not in general be an integer.

Then after computing some examples, you might notice the following:

Proposition 3.2.5. *Let $\{a_k\}$ be a periodic sequence of complex numbers, and suppose that its frequency is an integer. In that case, if the jth term of the DFT, b_j, is nonzero, then j is a multiple of the frequency of the sequence $\{a_k\}$.*

Another way to say this is that if j is *not* a multiple of the frequency of the sequence $\{a_k\}$, then $b_j = 0$. (This is the converse of the statement, which is always true!) Note that the converse in not true in general in this case: It is possible for j to be a multiple of the frequency but b_j to be zero anyway (although certainly it requires some work to arrange the sequence $\{a_k\}$ for that to happen).

3.2.4 Examples when the frequency is an integer

To illustrate and understand Proposition 3.2.5, we work out some examples:

Example 3.2.6. Consider the sequence

$$c_0, c_1, c_0, c_1, c_0, c_1.$$

It has length 6, period 2 and frequency 3. According to (3.3), the jth term of the DFT of this sequence is

$$b_j = \frac{1}{\sqrt{6}}\left(c_0 e^{\frac{2\pi i}{6}j\cdot 0} + c_1 e^{\frac{2\pi i}{6}j\cdot 1} + c_0 e^{\frac{2\pi i}{6}j\cdot 2} + c_1 e^{\frac{2\pi i}{6}j\cdot 3} + c_0 e^{\frac{2\pi i}{6}j\cdot 4} + c_1 e^{\frac{2\pi i}{6}j\cdot 5}\right).$$

To see what is going on, let's first rearrange the terms so all of the c_0s are together and all of the c_1s are together:

$$b_j = \frac{1}{\sqrt{6}}\left(c_0(e^{\frac{2\pi i}{6}j\cdot 0} + e^{\frac{2\pi i}{6}j\cdot 2} + e^{\frac{2\pi i}{6}j\cdot 4}) + c_1(e^{\frac{2\pi i}{6}j\cdot 1} + e^{\frac{2\pi i}{6}j\cdot 3} + e^{\frac{2\pi i}{6}j\cdot 5})\right).$$

In the sum with the c_1, I can factor out $e^{\frac{2\pi ij}{6}}$ from each term. Working out each exponential beforehand, this is what I get:

$$e^{\frac{2\pi i}{6}j\cdot 1} = (e^{\frac{2\pi ij}{6}})^1 = e^{\frac{2\pi ij}{6}} \cdot (e^{\frac{2\pi ij}{6}})^0 = e^{\frac{2\pi ij}{6}} \cdot e^{\frac{2\pi i}{6}j\cdot 0}$$

(this looks a bit weird, I know, but I want it to look like the other term with c_0!)

$$e^{\frac{2\pi i}{6}j\cdot 3} = (e^{\frac{2\pi ij}{6}})^3 = e^{\frac{2\pi ij}{6}} \cdot (e^{\frac{2\pi ij}{6}})^2 = e^{\frac{2\pi ij}{6}} \cdot e^{\frac{2\pi i}{6}j\cdot 2}$$

and

$$e^{\frac{2\pi i}{6}j\cdot 5} = (e^{\frac{2\pi ij}{6}})^5 = e^{\frac{2\pi ij}{6}} \cdot (e^{\frac{2\pi ij}{6}})^4 = e^{\frac{2\pi ij}{6}} \cdot e^{\frac{2\pi i}{6}j\cdot 4}.$$

So now I have

$$b_j = \frac{1}{\sqrt{6}}\left(c_0(e^{\frac{2\pi i}{6}j\cdot 0} + e^{\frac{2\pi i}{6}j\cdot 2} + e^{\frac{2\pi i}{6}j\cdot 4}) + c_1 e^{\frac{2\pi ij}{6}}(e^{\frac{2\pi i}{6}j\cdot 0} + e^{\frac{2\pi i}{6}j\cdot 2} + e^{\frac{2\pi i}{6}j\cdot 4})\right).$$

Now I am going to make another decision, and let $z_j = e^{2\pi i\frac{2j}{6}}$. Let's work out each exponential beforehand again:

$$e^{\frac{2\pi i}{6}j\cdot 0} = e^{2\pi i\frac{j\cdot 0}{6}} = e^{2\pi i\frac{2j\cdot 0}{6}} = (e^{2\pi i\frac{2j}{6}})^0 = z_j^0$$

for the first exponential,

$$e^{\frac{2\pi i}{6}j\cdot 2} = e^{2\pi i\frac{j\cdot 2}{6}} = e^{2\pi i\frac{2j\cdot 1}{6}} = (e^{2\pi i\frac{2j}{6}})^1 = z_j$$

for the second exponential, and

$$e^{\frac{2\pi i}{6}j\cdot 4} = e^{2\pi i\frac{j\cdot 4}{6}} = e^{2\pi i\frac{2j\cdot 2}{6}} = (e^{2\pi i\frac{2j}{6}})^2 = z_j^2$$

for the last exponential. Note also that $z_j = e^{2\pi i\frac{2j}{6}} = e^{2\pi i\frac{j}{3}}$ is a third root of unity, no matter what j is.

Now my jth term of the DFT looks like this:

$$b_j = \frac{1}{\sqrt{6}}\left(c_0(z_j^0 + z_j + z_j^2) + c_1 e^{\frac{2\pi ij}{6}}(z_j^0 + z_j + z_j^2)\right),$$

with z_j some third root of unity which depends on j. This doesn't look too bad!

At this point, to see what happens to the value of b_j, I can apply Proposition 3.2.2: If $z_j \neq 1$, then the sum

$$z_j^0 + z_j + z_j^2$$

will be zero. This will make $b_j = 0$ automatically. So when is $z_j = e^{2\pi i\frac{j}{3}} \neq 1$? Exactly when j is *not* a multiple of 3.

What about when $z_j = 1$ (or in other words, when $\frac{j}{3}$ is an integer)? In that case, the sum $z_j^0 + z_j + z_j^2 = 3$ then, which is relatively large! (But exactly what happens depends on the values of c_0 and c_1, which could still show some cancelation.)

Let's turn our attention to another example where the frequency is 3 to make sure that this wasn't a coincidence.

Example 3.2.7. Consider the sequence

$$c_0, c_1, c_2, c_3, c_0, c_1, c_2, c_3, c_0, c_1, c_2, c_3.$$

It has length 12, period 4 and frequency 3. According to (3.3), the jth term of the DFT of this sequence is

$$\begin{aligned} b_j = \frac{1}{\sqrt{12}} \Big(& c_0 e^{\frac{2\pi i}{12} j \cdot 0} + c_1 e^{\frac{2\pi i}{12} j \cdot 1} + c_2 e^{\frac{2\pi i}{12} j \cdot 2} + c_3 e^{\frac{2\pi i}{12} j \cdot 3} + \\ & c_0 e^{\frac{2\pi i}{12} j \cdot 4} + c_1 e^{\frac{2\pi i}{12} j \cdot 5} + c_2 e^{\frac{2\pi i}{12} j \cdot 6} + c_3 e^{\frac{2\pi i}{12} j \cdot 7} + \\ & c_0 e^{\frac{2\pi i}{12} j \cdot 8} + c_1 e^{\frac{2\pi i}{12} j \cdot 9} + c_2 e^{\frac{2\pi i}{12} j \cdot 10} + c_3 e^{\frac{2\pi i}{12} j \cdot 11} \Big). \end{aligned}$$

We do the same manipulations again, this time a bit faster:

$$\begin{aligned} b_j = \frac{1}{\sqrt{12}} \Big(& c_0 (e^{\frac{2\pi i}{12} j \cdot 0} + e^{\frac{2\pi i}{12} j \cdot 4} + e^{\frac{2\pi i}{12} j \cdot 8}) + \\ & c_1 e^{\frac{2\pi i j}{12}} (e^{\frac{2\pi i}{12} j \cdot 0} + e^{\frac{2\pi i}{12} j \cdot 4} + e^{\frac{2\pi i}{12} j \cdot 8}) + \\ & c_2 e^{\frac{4\pi i j}{12}} (e^{\frac{2\pi i}{12} j \cdot 0} + e^{\frac{2\pi i}{12} j \cdot 4} + e^{\frac{2\pi i}{12} j \cdot 8}) + \\ & c_3 e^{\frac{6\pi i j}{12}} (e^{\frac{2\pi i}{12} j \cdot 0} + e^{\frac{2\pi i}{12} j \cdot 4} + e^{\frac{2\pi i}{12} j \cdot 8}) \Big) \end{aligned}$$

This time we let $z_j = e^{\frac{2\pi i}{12} j \cdot 4}$. But this is $z_j = e^{2\pi i \frac{j}{3}}$ again, just like in the previous example! So the same thing will happen again:

$$\begin{aligned} b_j = \frac{1}{\sqrt{12}} \Big(& c_0 (z_j^0 + z_j + z_j^2) + c_1 e^{\frac{2\pi i j}{12}} (z_j^0 + z_j + z_j^2) + \\ & c_2 e^{\frac{4\pi i j}{12}} (z_j^0 + z_j + z_j^2) + c_3 e^{\frac{6\pi i j}{12}} (z_j^0 + z_j + z_j^2) \Big), \end{aligned}$$

and $b_j = 0$ when $z_j \neq 1$ (i.e. if j is not a multiple of 3). When j is a multiple of 3 (i.e. when $\frac{j}{3}$ is an integer), the sum $z_j^0 + z_j + z_j^2 = 3$ again, but once again exactly what happens to b_j depends on the values of c_0, c_1, c_2 and c_3, which could still show some cancelation.

We could continue working out more examples with frequency $f = 3$, and each time we would see the same thing happen: For each coefficient c_k, there would be a sum $z_j^0 + z_j + z_j^2$. Furthermore we can show that in each case $z_j = e^{2\pi i \frac{j}{3}}$, no matter what the length/period is! Indeed suppose that the sequence is of length m and period p, so that $\frac{m}{p} = 3$ (remember that we assume that the frequency is 3 right now). Then we will have, as part of the sum defining b_j, the expression

$$c_0 (e^{\frac{2\pi i}{m} j \cdot 0} + e^{\frac{2\pi i}{m} j \cdot p} + e^{\frac{2\pi i}{m} j \cdot 2p}),$$

since c_0 will appear again as the pth term of the sequence. Then $z_j = e^{\frac{2\pi i}{m} j \cdot p} = e^{2\pi i \frac{j}{3}}$.

This gives us a good idea of what will happen in general, so let's test our idea by working out an example where the frequency is 6.

Example 3.2.8. Consider the sequence

$$c_0, c_1, c_0, c_1, c_0, c_1, c_0, c_1, c_0, c_1, c_0, c_1.$$

It has length 12, period 2 and frequency 6. According to (3.3), the jth term of the DFT of this sequence is

$$\begin{aligned} b_j = \frac{1}{\sqrt{12}} \Big(& c_0 e^{\frac{2\pi i}{12} j\cdot 0} + c_1 e^{\frac{2\pi i}{12} j\cdot 1} + c_0 e^{\frac{2\pi i}{12} j\cdot 2} + c_1 e^{\frac{2\pi i}{12} j\cdot 3} + \\ & c_0 e^{\frac{2\pi i}{12} j\cdot 4} + c_1 e^{\frac{2\pi i}{12} j\cdot 5} + c_0 e^{\frac{2\pi i}{12} j\cdot 6} + c_1 e^{\frac{2\pi i}{12} j\cdot 7} + \\ & c_0 e^{\frac{2\pi i}{12} j\cdot 8} + c_1 e^{\frac{2\pi i}{12} j\cdot 9} + c_0 e^{\frac{2\pi i}{12} j\cdot 10} + c_1 e^{\frac{2\pi i}{12} j\cdot 11} \Big). \end{aligned}$$

We do the same manipulations again, this time it looks a bit different of course, but it's the same idea:

$$\begin{aligned} b_j = \frac{1}{\sqrt{12}} \Big(& c_0 (e^{\frac{2\pi i}{12} j\cdot 0} + e^{\frac{2\pi i}{12} j\cdot 2} + e^{\frac{2\pi i}{12} j\cdot 4} + e^{\frac{2\pi i}{12} j\cdot 6} + e^{\frac{2\pi i}{12} j\cdot 8} + e^{\frac{2\pi i}{12} j\cdot 10}) + \\ & c_1 e^{\frac{2\pi i j}{12}} (e^{\frac{2\pi i}{12} j\cdot 0} + e^{\frac{2\pi i}{12} j\cdot 2} + e^{\frac{2\pi i}{12} j\cdot 4} + e^{\frac{2\pi i}{12} j\cdot 6} + e^{\frac{2\pi i}{12} j\cdot 8} + e^{\frac{2\pi i}{12} j\cdot 10}) \Big) \end{aligned}$$

So we let $z_j = e^{\frac{2\pi i}{12} j\cdot 2}$. This simplifies to $z_j = e^{2\pi i \frac{j}{6}}$, which is a sixth root of unity, no matter what j is. Then we can write

$$b_j = \frac{1}{\sqrt{12}} \Big(c_0 (z_j^0 + z_j + z_j^2 + z_j^3 + z_j^4 + z_j^5) + c_1 e^{\frac{2\pi i j}{12}} (z_j^0 + z_j + z_j^2 + z_j^3 + z_j^4 + z_j^5) \Big)$$

and $b_j = 0$ when $z_j \neq 1$ once again, since by Proposition 3.2.2, any sixth root of unity, as long as it's not 1, when plugged into $1 + \zeta_6 + \zeta_6^2 + \zeta_6^3 + \zeta_6^4 + \zeta_6^5$, will give zero.

Again we see that when j is a multiple of 6, or equivalently when $\frac{j}{6}$ is an integer, then $z_j = 1$, and the sum $z_j^0 + z_j + z_j^2 + z_j^3 + z_j^4 + z_j^5 = 6$. The value of b_j then depends on the value of c_0 and c_1.

What we have seen here is what will happen in general: Given a sequence with frequency $f \in \mathbb{Z}$, the sum for b_j can be rearranged so that each term contains a sum

$$z_j^0 + z_j + z_j^2 + \cdots + z_j^{f-1},$$

and z_j is an fth root root of unity, $z_j = e^{2\pi i \frac{j}{f}}$. Then whenever j is not a multiple of f, we get that the sum is zero, so $b_j = 0$, and when j is a multiple of f (or $\frac{j}{f}$ is an integer, which is how we will think of this soon), then $z_j = 1$ and in general the sum b_j will be large, although it depends on the exact values of the c_ks.

3.2.5 Examples when the frequency is not an integer

Unfortunately for us, most of the time when we use the DFT the frequency will not be an integer. In that case, the results aren't quite as clear and pleasing, but we will see that the DFT can still give us quite a bit of information about the approximate frequency of the sequence.

Example 3.2.9. Consider the sequence

$$c_0, c_1, c_2, c_0, c_1, c_2, c_0.$$

It has length 7 and period 3, so its frequency is $\frac{7}{3}$.

We can still write the jth coefficient of the DFT and rearrange as we did in the previous Section:

$$\begin{aligned} b_j &= \frac{1}{\sqrt{7}}\left(c_0 e^{\frac{2\pi i}{7}j\cdot 0} + c_1 e^{\frac{2\pi i}{7}j\cdot 1} + c_2 e^{\frac{2\pi i}{7}j\cdot 2} + c_0 e^{\frac{2\pi i}{7}j\cdot 3} + c_1 e^{\frac{2\pi i}{7}j\cdot 4} + c_2 e^{\frac{2\pi i}{7}j\cdot 5} + c_0 e^{\frac{2\pi i}{7}j\cdot 6}\right) \\ &= \frac{1}{\sqrt{7}}\left(c_0(e^{\frac{2\pi i}{7}j\cdot 0} + e^{\frac{2\pi i}{7}j\cdot 3} + e^{\frac{2\pi i}{7}j\cdot 6}) + c_1 e^{\frac{2\pi i j}{7}}(e^{\frac{2\pi i}{7}j\cdot 0} + e^{\frac{2\pi i}{7}j\cdot 3}) + c_2 e^{\frac{4\pi i j}{7}}(e^{\frac{2\pi i}{7}j\cdot 0} + e^{\frac{2\pi i}{7}j\cdot 3})\right). \end{aligned}$$

This time we let $z_j = e^{\frac{2\pi i}{7}j\cdot 3} = e^{2\pi i\frac{3j}{7}}$. This is a seventh root of unity, no matter what j is. Then our jth DFT coefficient is

$$b_j = \frac{1}{\sqrt{7}}\left(c_0(z_j^0 + z_j + z_j^2) + c_1 e^{\frac{2\pi i j}{7}}(z_j^0 + z_j) + c_2 e^{\frac{4\pi i j}{7}}(z_j^0 + z_j)\right).$$

Now we have a mismatch: There is no value of j that will make z_j a third root of unity, and that will make

$$z_j^0 + z_j + z_j^2 = 0.$$

There is also no value of j that will make z_j a second root of unity, and therefore that will make

$$z_j^0 + z_j = 0.$$

Because of this, we don't see as nice of results in this case as we did earlier.

However, if we run some computations, we still see some patterns:

For the sequence

$$1, 2, 3, 1, 2, 3, 1,$$

we have

j	0	1	2	3	4	5	6
$\lvert b_j\rvert$	13	1.21	3.65	1.47	1.47	3.65	1.21

.

For the sequence

$$1, 2, -1, 1, 2, -1, 1,$$

we have

j	0	1	2	3	4	5	6
$\lvert b_j\rvert$	5	0.70	4.37	3.65	3.65	4.37	0.70

.

Finally for the sequence

$$-1, 2, 1, -1, 2, 1, -1,$$

we have

j	0	1	2	3	4	5	6
$\lvert b_j\rvert$	3	2.01	5.79	1.84	1.84	5.79	2.01

.

We see a definite tendency towards having b_0 be very large, and b_2 and b_5 be somewhat large. The values b_1, b_3, b_4 and b_6 seem smaller. (It depends a little bit on the example, the exact value of the c_ks can make more or less cancelation happen.)

Let's go back to our expression for b_j:

$$b_j = \frac{1}{\sqrt{7}}\left(c_0(z_j^0 + z_j + z_j^2) + c_1 e^{\frac{2\pi i j}{7}}(z_j^0 + z_j) + c_2 e^{\frac{4\pi i j}{7}}(z_j^0 + z_j)\right),$$

where we remember that $z_j = e^{2\pi i \frac{3j}{7}}$.

We see that b_j takes its largest value when $j = 0$. In that case, $z_0 = 1$, and $b_0 = \frac{1}{\sqrt{7}}(3c_0 + 2c_1 + 2_2)$. If there isn't much cancellation between c_0, c_1 and c_2, b_0 will be large. (This is also what we have observed when the frequency was an integer. b_0 is always really large because $z_0 = 1$.)

If $j = 2$, then $z_2 = e^{2\pi i \frac{6}{7}}$. $\frac{6}{7}$ is very close to being 1, which means that z_2 is very close to being 1 as well. This makes b_2 large in all of our examples. If $j = 5$, then $z_5 = e^{2\pi i \frac{15}{7}}$. Because $\frac{15}{7}$ is very close to being an integer (the integer 2), it means that z_5 is also very close to being 1. This makes b_5 large.

Indeed we can see that z_j being near 1 (or $\frac{3j}{7}$ being near an integer, which is the same), makes b_j large by making the sums $z_j^0 + z_j + z_j^2$ and $z_j^0 + z_j$ large.

j	$\frac{3j}{7}$	$\lvert z_j^0 + z_j + z_j^2\rvert$	$\lvert z_j^0 + z_j\rvert$
0	0	3	2
2	0.85...	2.24...	1.80...
5	2.14...	2.24...	1.80...

On the other hand, if $j = 1, 3, 4$ or 6, we have that $z_1 = e^{2\pi i \frac{3}{7}}$, $z_3 = e^{2\pi i \frac{9}{7}} = e^{2\pi i \frac{2}{7}}$, $z_4 = e^{2\pi i \frac{12}{7}} = e^{2\pi i \frac{5}{7}}$ and $z_6 = e^{2\pi i \frac{18}{7}} = e^{2\pi i \frac{4}{7}}$. The fractions $\frac{3}{7}$, $\frac{9}{7}$, $\frac{12}{7}$, and $\frac{18}{7}$ are all not very close to integers. For example, since $\frac{3}{7} \approx 0.42\ldots$, this means that z_1 is somewhere between a second and a third root of unity, and therefore it makes both $z_j^0 + z_j + z_j^2$ and $z_j^0 + z_j$ kind of small. (In fact, $|z_j^0 + z_j + z_j^2| \approx 0.80\ldots$, which is not so small but much small than 3, and $|z_j^0 + z_j| \approx 0.44\ldots$.) Arranging the relevant values in a table, we have that

j	$\frac{3j}{7}$	$\lvert z_j^0 + z_j + z_j^2\rvert$	$\lvert z_j^0 + z_j\rvert$
1	0.42...	0.80...	0.44...
3	1.28...	0.55...	1.24...
4	1.71...	0.55...	1.24...
6	2.57...	0.80...	0.44...

We now have an explanation why the coefficients b_1, b_3, b_4 and b_6 are relatively smaller: There is a lot of cancelation in the sum $z_j^0 + z_j + z_j^2$ and $z_j^0 + z_j$.

Let's do one last example quickly:

Example 3.2.10. Consider the sequence

$$c_0, c_1, c_2, c_0, c_1, c_2, c_0, c_1, c_2, c_0, c_1.$$

The length is $m = 11$, the period is 3 and so the frequency is $\frac{11}{3}$. If we write out b_j and rearrange the terms like we usually do, we get

$$b_j = \frac{1}{\sqrt{11}} \left(c_0(z_j^0 + z_j + z_j^2 + z_j^3) + c_1 e^{2\pi i \frac{j}{11}}(z_j^0 + z_j + z_j^2 + z_j^3) + c_2 e^{2\pi i \frac{2j}{11}}(z_j^0 + z_j + z_j^2) \right),$$

with $z_j = e^{2\pi i \frac{3j}{11}}$. (Make sure you can get this yourself!)

We have that $\frac{3j}{11}$ is an integer when $j = 0$, as usual. That should be the largest Fourier coefficient. After that, it is nearest an integer when $j = 4$ or $j = 7$, so we expect those to be the next largest. Then $j = 3$ and $j = 8$ make $\frac{3j}{11}$ the numbers $\frac{9}{11}$ and $\frac{24}{11} = 2\frac{2}{11}$, respectively, so b_3 and b_8 should be next largest. And so on.

Looking at it the other way, when $j = 1$ or $j = 10$, we have $\frac{3j}{11}$ be $\frac{3}{11}$ and $\frac{30}{11} = 2\frac{8}{11}$, which are close to the fractions $\frac{1}{4}$ and $2\frac{3}{4}$. This makes z_j very close to a fourth root of unity, so b_1 and b_{10} should be small. If $j = 5$ or $j = 6$, then $\frac{3j}{11}$ is respectively $\frac{15}{11} = 1\frac{4}{11}$, which is near $1\frac{1}{3}$, and $\frac{18}{11} = 1\frac{7}{11}$, which is near $1\frac{2}{3}$ Therefore z_5 and z_6 are close to being third roots of unity, and we expect b_5 and b_6 to be small too.

We work out a few sequences to show these results hold up: For the sequence

$$1, 2, 3, 1, 2, 3, 1, 2, 3, 1, 2$$

we have

j	0	1	2	3	4	5	6	7	8	9	10
$\lvert b_j \rvert$	21	1.07	1.40	2.94	4.98	1.17	1.17	4.98	2.94	1.40	1.07

.

Our predictions worked out pretty well! Let's just work out one more example:

Consider the sequence

$$-1, -2, 3, -1, -2, 3, -1, -2, 3, -1, -2,$$

then we have

j	0	1	2	3	4	5	6	7	8	9	10
$\lvert b_j \rvert$	3	3.22	4.14	8.30	12.42	1.76	1.76	12.42	8.30	4.14	3.22

.

In this particular example, because c_0, c_1 and c_2 cancel each other perfectly, b_0 is actually not very large at all. However, $|b_3|, |b_4|, |b_7|$ and $|b_8|$ are all very large, as we expect, and $|b_1|, |b_5|, |b_6|$ and $|b_{10}|$ are small.

3.3 Continued fractions

The notation we are using in class is the following: A *simple* continued fraction is an expression of the form

$$a_0 + \cfrac{1}{a_1 + \cfrac{1}{a_2 + \cfrac{1}{a_3 + \cfrac{1}{\ddots \; + \cfrac{1}{a_n + \cfrac{1}{\ddots}}}}}},$$

where $a_0 \in \mathbb{Z}$ and for $i \geq 1$, $a_i \in \mathbb{Z}$ and $a_i \geq 1$. For compactness of notation, we usually write $[\, a_0; a_1, a_2, a_3, \ldots, a_n, \ldots\,]$ for this expression.

There are many fun things to say about continued fractions, but for now we will state the two facts we will need. These notes might be expanded in the future to contain more information.

One important quantity will be the so-called *convergents* of a continued fraction:

Definition 3.3.1. Given a continued fraction $[\, a_0; a_1, a_2, a_3, \ldots, a_n, \ldots\,]$, its *kth convergent* is the number given by

$$\frac{p_k}{q_k} = [\, a_0; a_1, a_2, a_3, \ldots, a_k\,].$$

As one might expect, when $\frac{p_k}{q_k}$ is written in lowest terms, the number p_k is called the *numerator of the kth convergent* of $[\, a_0; a_1, a_2, a_3, \ldots, a_n, \ldots\,]$ and q_k is called the *denominator of the kth convergent* of $[\, a_0; a_1, a_2, a_3, \ldots, a_n, \ldots\,]$.

With this notation in place, we have

Proposition 3.3.2. *The numerator p_k and the denominator q_k of the kth convergent of $[\, a_0; a_1, a_2, a_3, \ldots, a_n, \ldots\,]$ are given recursively by the following formulae:*

$$\begin{aligned} p_{-2} &= 0, \quad p_{-1} = 1 \\ q_{-2} &= 1, \quad q_{-1} = 0, \end{aligned}$$

and for $k \geq 0$,

$$\begin{aligned} p_k &= a_k p_{k-1} + p_{k-2}, \\ q_k &= a_k q_{k-1} + q_{k-2}. \end{aligned}$$

Remark 3.3.3. Note that in the definition, the fraction $\frac{p_k}{q_k}$ is *in lowest terms*. Accordingly, the proposition gives $\frac{p_k}{q_k}$ *also* in lowest terms! This can help you spot a mistake in your calculations if you get any factors in common between p_k and q_k for some k.

The significance of convergents and continued fraction expansions, to us, is the following:

Theorem 3.3.4. *Let α be an arbitrary real number. If the rational number $\frac{a}{b}$, where $b \geq 1$ and $\gcd(a, b) = 1$ satisfies*

$$\left|\alpha - \frac{a}{b}\right| < \frac{1}{2b^2},$$

then $\frac{a}{b}$ is one of the convergents $\frac{p_k}{q_k}$ of the continued fraction expansion of α.

As a kind of converse to this theorem, one might wonder if its convergents ever get this close to α, and they do:

Proposition 3.3.5. *Let α be an irrational number. Then for any two consecutive convergents, at least one of them, which we will denote $\frac{p}{q}$, satisfies the inequality*

$$\left|\alpha - \frac{p}{q}\right| < \frac{1}{2q^2}.$$

So at least half of the convergents of an irrational number give excellent approximations to this number.

3.4 Shor's algorithm

For this section we will assume that we wish to factor a positive integer N, which is odd and is not a power of a prime. By "factor" here, explicitly we mean "to give a divisor d, $1 < d < N$, of N." The reason we do not consider even integers is that it is easy to spot an even integer, and to give in that case the nontrivial factor $d = 2$.

The reason we do not consider powers of (odd) primes here is because our algorithm will not work.

3.4.1 Classical steps

We reduce the problem of factoring N to the problem of, given a with $1 < a < N$ with $\gcd(a, N) = 1$, finding the multiplicative order of a modulo N. We do this in the following manner:

We first suppose that we can input an integer a with $1 < a < N$ with $\gcd(a, N) = 1$, and output *quickly* its multiplicative order r. One upshot of the fact that we can do this quickly is that we can further assume that we can find a with $1 < a < N$ with $\gcd(a, N) = 1$, of course, but also such that

- r is even; and
- $a^{r/2} \not\equiv -1 \pmod{N}$.

The reason we may assume these two further properties of a is that such as exist because N is the product of at least two distinct odd primes, by our assumption on N. Therefore, were we to randomly choose a *not* satisfying these two extra conditions, we can just throw it out and start over with a new random a which hopefully does satisfy the two extra conditions. Furthermore, according to Rubinstein-Salzedo's *Cryptography*, page 213, the odds of a satisfying all the conditions we want are larger than $\frac{2}{3}$, so we may keep choosing random values of a and hope to find one that works reasonably quickly.

Accordingly, here is the classical part of the algorithm:

1. Choose an integer a, $1 < a < N$ at random.
2. Compute $\gcd(a, N)$. If $\gcd(a, N) > 1$, this produces a non-trivial divisor of N and we are done. Otherwise, proceed, a satisfies now two of the four conditions we need.
3. Use the quantum computer to compute the multiplicative order r of a modulo N (see next section).
4. If r is odd, throw away a and start over with a different a. Otherwise, proceed, a satisfies now three of the four conditions we need.
5. Compute $a^{r/2} \pmod N$. If $a^{r/2} \equiv -1 \pmod N$, throw away a and start over with a different a. Otherwise, proceed, a satisfies all four conditions we need.
6. Since $a^{r/2} \not\equiv 1 \pmod N$ either (why?), we have that $x \equiv a^{r/2} \pmod N$ is a solution to $x^2 \equiv 1 \pmod N$, but $x \not\equiv \pm 1 \pmod N$.
7. Compute $\gcd(x - 1, N)$ and $\gcd(x + 1, N)$, these are two non-trivial factors of N.

The reason why step 7 works is the following: Since $x^2 \equiv 1 \pmod N$, we have

$$x^2 - 1 \equiv (x - 1)(x + 1) \equiv 0 \pmod N.$$

In this case, there are three options: Either $x - 1 \equiv 0 \pmod N$, but since $x \not\equiv 1 \pmod N$, that is not the case, or $x + 1 \equiv 0 \pmod N$, but since $x \not\equiv -1 \pmod N$, that is not the case either, *or* each of $x - 1$ and $x + 1$ share *some* factors with N, in such a way that N divides neither $x - 1$ nor $x + 1$, but N divides their product. This is the situation we must be in, and that is what the gcd computes, therefore giving us some nontrivial factors of N.

3.4.2 Quantum steps

We thus now turn our attention to the problem of computing the multiplicative order of an integer a modulo N.

We begin by fixing q such that

$$N^2 \leq 2^q < 2N^2.$$

(This always exists and can be taken to be

$$q = \lceil \log_2(N^2) \rceil .)$$

Then the quantum steps of Shor's algorithm are:

1. Start with a $2q$-qbit register $|0\rangle$. Throughout we will think of this as two q-qbit registers, one next to the other.

2. Apply the q-qbit Hadamard gate to the first q-qbits of the register:

$$\frac{1}{\sqrt{2^q}} \sum_{k=0}^{2^q-1} |\, k\ 0 \,\rangle.$$

3. In the second q-qbits of the register, for each k compute and store the value $a^k \pmod N$:

$$\frac{1}{\sqrt{2^q}} \sum_{k=0}^{2^q-1} |\, k\ a^k \,\rangle.$$

4. Apply the quantum Fourier transform to the first q-qbits, which sends

$$|k\rangle \mapsto \frac{1}{\sqrt{2^q}} \sum_{j=0}^{2^q-1} e^{\frac{2\pi i}{2^q} jk} |j\rangle.$$

   This gives us the superposition

$$\frac{1}{2^q} \sum_{k=0}^{2^q-1} \sum_{j=0}^{2^q-1} e^{\frac{2\pi i}{2^q} jk} |\, j\ a^k \,\rangle. \tag{3.4}$$

5. We observe this superposition, and record j, the value of the first q-qbits we observe.

6. With high probability, we have observed j such that

$$\frac{jr}{2^q}$$

   is very near an integer M. In other words, there is an integer M such that the value

$$\left| \frac{j}{2^q} - \frac{M}{r} \right|$$

   is very small. This value is in fact so small, compared to r, that it is likely that $\frac{M}{r}$ is a convergent of $\frac{j}{2^q}$. Therefore we compute the convergents of $\frac{j}{2^q}$ until we reach the last convergent $\frac{p_k}{q_k}$ with $q_k < N$. q_k is very likely to be r, the multiplicative order of a.

7. Verify if q_k is the multiplicative order of a by computing a^{q_k} and seeing if it is 1 modulo N. If not, try some small multiples of q_k, such as $2q_k$, $3q_k$, etc. (in case there was cancelation with M, remember that $\frac{p_k}{q_k}$ is in lowest terms!). If not, try perhaps some nearby convergents. If nothing works very quickly, repeat the algorithm, either with the same value of a or a new value of a.

3.4.3 Explanation of step 6

There are two "mathematical mysteries" in Step 6 of the quantum algorithm. The first is why we will observe j such that $\frac{jr}{2^q}$ is near an integer. This relates directly to the properties of the Quantum/Discrete Fourier transform, and hopefully should feel familiar from the examples we worked out there. The second is why computing convergents will help guess a fraction whose numerator *and* denominator we don't know.

First mystery: Why is $\frac{rj}{2^q}$ near an integer?

We first investigate the first mystery, which is that when we observe the superposition we have created in equation (3.4), we are very likely to observe a number whose first q-bits are a number j such that

$$\frac{jr}{2^q} \approx M,$$

where r is the (unknown) multiplicative order of a modulo N, M is an unknown integer, j is (part of) the output we observe, and 2^q is a known quantity. For now we leave the symbol $\approx$ to be vague and simply say that $\frac{jr}{2^q}$ is "near" the integer M; we will say more about this soon.

To see this, we need to manipulate equation (3.4) to "collect like terms." We will first do this abstractly, and then with relatively small values.

We begin by observing that we have

$$\frac{1}{2^q}\sum_{k=0}^{2^q-1}\sum_{j=0}^{2^q-1} e^{\frac{2\pi i}{2^q}jk}\big|\, j\ a^k\,\big\rangle = \frac{1}{2^q}\sum_{j=0}^{2^q-1}\sum_{k=0}^{2^q-1} e^{\frac{2\pi i}{2^q}jk}\big|\, j\ a^k\,\big\rangle.$$

This is because rather than organizing our sum by thinking of k first, fixing it and letting j range over all the values, we can think of organizing our sum by the value of j that appears.

The next step is the tricky one: As k ranges from 0 to 2^q-1, there will be several different values of k for which a^k takes the same value. We wish to collect those. We first notice that by definition of the multiplicative order of a modulo N, the expression a^k takes exactly the values

$$1, a, a^2, a^3, a^4, \ldots, a^{r-2}, a^{r-1},$$

these values are all different, and a^k takes no other values than those. In fact, the sequence of values of a^k modulo N is exactly

$$1, a, a^2, \ldots, a^{r-2}, a^{r-1}, 1, a, a^2, \ldots, a^{r-2}, a^{r-1}, 1, a, a^2, \ldots$$

This tells us furthermore that for a fixed k with $0 \le k \le r-1$, we have

$$a^k \equiv a^{k+r} \equiv a^{k+2r} \equiv a^{k+3r} \equiv \ldots \equiv a^{k+mr} \equiv \ldots \pmod{N},$$

or in other words that the sequence repeats with period r. Therefore instead of letting k go from 0 to $2^q - 1$, we can group up the values of k that give the same values of a^k and write

$$\frac{1}{2^q}\sum_{k=0}^{2^q-1}\sum_{j=0}^{2^q-1} e^{\frac{2\pi i}{2^q}jk}\big|\, j\ a^k\,\big\rangle = \frac{1}{2^q}\sum_{j=0}^{2^q-1}\sum_{k=0}^{2^q-1} e^{\frac{2\pi i}{2^q}jk}\big|\, j\ a^k\,\big\rangle$$
$$= \frac{1}{2^q}\sum_{j=0}^{2^q-1}\sum_{k=0}^{r-1}\sum_{m=0}^{??} e^{\frac{2\pi i}{2^q}j(k+mr)}\big|\, j\ a^k\,\big\rangle$$

We stop for a second to contemplate this number ?? atop the third summation. This number is simply whatever it needs to be so that the last value $k + mr \leq 2^q - 1$, which is after all where the original k sum was supposed to end. If we solve for m we get

$$m \leq \frac{2^q - k - 1}{r},$$

and since m must be an integer but $\frac{2^q-k-1}{r}$ is almost certainly not an integer, it is customary to write this with a floor function: For each $0 \leq k \leq r-1$, the largest value that m will take is

$$\left\lfloor \frac{2^q - k - 1}{r} \right\rfloor.$$

So we have

$$\frac{1}{2^q}\sum_{k=0}^{2^q-1}\sum_{j=0}^{2^q-1} e^{\frac{2\pi i}{2^q}jk}\big|\, j\ a^k\,\big\rangle = \frac{1}{2^q}\sum_{j=0}^{2^q-1}\sum_{k=0}^{2^q-1} e^{\frac{2\pi i}{2^q}jk}\big|\, j\ a^k\,\big\rangle$$
$$= \frac{1}{2^q}\sum_{j=0}^{2^q-1}\sum_{k=0}^{r-1}\sum_{m=0}^{\lfloor \frac{2^q-k-1}{r}\rfloor} e^{\frac{2\pi i}{2^q}j(k+mr)}\big|\, j\ a^k\,\big\rangle.$$

Therefore the probability of observing a certain pair $\big|\, j\ a^k\,\big\rangle$ is

$$\left|\sum_{m=0}^{\lfloor \frac{2^q-k-1}{r}\rfloor} e^{\frac{2\pi i}{2^q}j(k+mr)}\right|^2.$$

This is just a sum of roots of unity and we've looked at many of those. We will do our usual tricks:

$$\left|\sum_{m=0}^{\lfloor \frac{2^q-k-1}{r}\rfloor} e^{\frac{2\pi i}{2^q}j(k+mr)}\right|^2 = \left|\sum_{m=0}^{\lfloor \frac{2^q-k-1}{r}\rfloor} e^{\frac{2\pi i}{2^q}jk} e^{\frac{2\pi i}{2^q}jmr}\right|^2$$
$$= \left| e^{\frac{2\pi i}{2^q}jk} \sum_{m=0}^{\lfloor \frac{2^q-k-1}{r}\rfloor} \left(e^{\frac{2\pi i r j}{2^q}}\right)^m \right|^2.$$

Now we let $z_j = e^{\frac{2\pi i r j}{2^q}} = e^{2\pi i \frac{rj}{2^q}}$ as usual. We know that a sum

$$\sum z_j^m$$

is largest when z_j is closest to being 1. This happens when $\frac{rj}{2^q}$ is close to being an integer. M is just what we've been calling this integer.

Our conclusion is that we are most likely to observe an output $|\, j\ \ell \,\rangle$ such that $\frac{rj}{2^q}$ is near an integer M, because it makes the sum

$$\left| e^{\frac{2\pi i}{2^q} jk} \sum_{m=0}^{\left\lfloor \frac{2^q - k - 1}{r} \right\rfloor} \left(e^{\frac{2\pi i r j}{2^q}} \right)^m \right|^2,$$

which is the probability of observing $|\, j\ \ell \,\rangle$ if $\ell \equiv a^k \pmod N$, the largest.

Second mystery: How to recover r from j?

We say above that the fraction $\frac{rj}{2^q}$ is near an integer M. What's more is that from our experiments we also saw that the largest values happen when in fact

$$\left| \frac{rj}{2^q} - M \right| \leq \frac{1}{2^q},$$

or when we are just $\frac{1}{2^q}$ away from an integer, which is as close as we can get without actually being an integer. Dividing through by r, we get

$$\left| \frac{j}{2^q} - \frac{M}{r} \right| \leq \frac{1}{r 2^q}.$$

Now recall that $N^2 \leq 2^q$, and $N \geq 2$, so certainly $2N \leq 2^q$. Furthermore, r is the order of a modulo N, so $r \leq N$ (the sequence must repeat before it's gone through more than the total number of values that exist modulo N). Therefore, $2r \leq 2N \leq 2^q$, from which it follows that

$$\left| \frac{j}{2^q} - \frac{M}{r} \right| \leq \frac{1}{r 2^q} \leq \frac{1}{2r^2}.$$

Now, recall that 2^q is very large compared to r. By Theorem 3.3.4, the only very good approximations of $\frac{j}{2^q}$ with small denominators such as r are convergents of $\frac{j}{2^q}$. Therefore the fraction $\frac{M}{r}$ must be among the convergents of $\frac{j}{2^q}$.

The last piece of information we have about $\frac{M}{r}$ is that its denominator is less than N. So we start computing the convergents of $\frac{j}{2^q}$, which we stress here is a known number: j is the value we observed in the first q-bits of our output, and 2^q is a number we chose at the beginning of the algorithm. At some point, we will get a convergent whose denominator is greater than N. If we take the convergent right before that one, the denominator of that convergent is very likely to be r.

We might be a little bit unlucky and there might have been some canceling: Perhaps r is even and M is even too, but of course the convergents will give us $\frac{M}{r}$ in lowest terms. This is why, when we check if the denominator q_k of the convergent is r, we also check if maybe $2q_k$ or $3q_k$ is r, just in case there is a small amount of cancelation. This doesn't take much more time.

Chapter 4

Learning with Errors

4.1 Classical LWE

In his 2005 paper, Regev suggests the following cryptographic scheme, which relies on the difficult of the LWE (Learning With Errors) problem. Please see the remark below about the values belonging to the integers versus their belonging to $\mathbb{Z}/q\mathbb{Z}$, as is usually prescribed.

1. Setting up:

   - Choose a positive integer n (the size), and suitable positive integers q (the modulus) and m (the number of equations; exact specifications for q and m are in the paper, and other authors suggest different choices)
   - Choose a probability distribution χ with value in the integers from which to draw errors (the exact properties necessary for this probability distribution should become clear, but for now we will say that it should give "small" error values)
   - Generate a secret integer vector of length n $\vec{s} \in \mathbb{Z}^n$
   - Generate m integer vectors of length n $\vec{a}_i \in \mathbb{Z}^n$
   - Generate m integer error values $e_i \in \mathbb{Z}$ drawn from the probability distribution χ
   - Compute the m integers
     $$b_i = \vec{a}_i \cdot \vec{s} + e_i,$$
     where $\cdot$ denotes the usual dot product.

   Then the public key is
   $$(q, \{(\vec{a}_i, b_i)\}_{i=1}^m),$$
   i.e., consists in the prime number q and the m pairs $(\vec{a}_i, b_i)$.

2. Encryption: This scheme allows one to send a 1-bit message, consisting either of the bit 0 or the bit 1.

   - The sender first chooses a random set $T \subseteq \{1, 2, \ldots, m\}$.

- – To send the bit 0, the sender then sends the pair $(\vec{a}, b)$, where

$$\vec{a} = \sum_{i \in T} \vec{a}_i, \quad \text{and} \quad b = \sum_{i \in T} b_i.$$

  – To send the bit 1, the sender then sends the pair $(\vec{a}, b)$, where again

$$\vec{a} = \sum_{i \in T} \vec{a}_i,$$

  but this time

$$b = \left\lfloor \frac{q}{2} \right\rfloor + \sum_{i \in T} b_i,$$

  where $\lfloor \cdot \rfloor$ denotes the floor function as usual.

3. Decryption: To decrypt, the receiver computes

$$b - \vec{a} \cdot \vec{s},$$

   and computes the remainder of this quantity upon division by q. The receiver decrypts the message to be 0 if this remainder falls in the interval

$$\left[0, \frac{q}{4}\right) \cup \left(\frac{3q}{4}, q - 1\right]$$

   and the message to be 1 if the remainder falls in the interval

$$\left(\frac{q}{4}, \frac{3q}{4}\right).$$

   In other words, the message is decrypted to be 0 if the remainder is "closer to 0 than to $\left\lfloor \frac{q}{2} \right\rfloor$ modulo q," and decrypts the message to be 1 otherwise.

The rationale for the decryption algorithm is the following: We have that

$$b - \vec{a} \cdot \vec{s} = \begin{cases} \displaystyle\sum_{i \in T} e_i, & \text{if the sent bit is 0;} \\ \displaystyle\left\lfloor \frac{q}{2} \right\rfloor + \sum_{i \in T} e_i, & \text{if the sent bit is 1.} \end{cases}$$

Therefore, there will be a decryption error (the process will decrypt 0 to be 1 or vice-versa) only in the event where

$$\left| \sum_{i \in T} e_i \right| > \frac{q}{4}.$$

We can control the probability of this happening by choosing the probability distribution χ carefully. For example, if χ is a discrete normal distribution with mean 0 and standard distribution suitably small, then the probability of this sum being large can be made very small. In turn, this makes the probability of a decryption error very small.

Remark 4.1.1. When $q = 2$, the LWE problem is called the LPN problem, Learning Parity with Noise.

As promised, we now discuss the issue of working over $\mathbb{Z}$ versus working over the ring $\mathbb{Z}/q\mathbb{Z}$. it is customary to set up the algorithm with $\vec{s}, \vec{a}_i \in (\mathbb{Z}/q\mathbb{Z})^n$ and for χ to be a distribution taking values in $\mathbb{Z}/q\mathbb{Z}$ (which also results in $b_i \in \mathbb{Z}/q\mathbb{Z}$ of course). However, this leads to some mathematical issues at the decryption stage, where one would like to say that some classes in $\mathbb{Z}/q\mathbb{Z}$ are "closer" to each other than to other classes (which can be defined with some care) or worse, that some classes in $\mathbb{Z}/q\mathbb{Z}$ are "smaller" than some others (which is mathematically nonsensical without choosing representatives for these classes).

We avoid this issue here by keeping all quantities in $\mathbb{Z}$, and instead ending the algorithm with taking the remainder upon division by q, which is a well-defined unique integer in the interval $[0, q-1]$. We note first that from an implementation point of view this is certainly inefficient; carrying around the potentially large values that $\vec{a}_i \cdot \vec{s}$ can take in the integers is unnecessary when one will only want the remainder upon division by q in the end. For an implementation therefore, it makes sense to compute everything in $\mathbb{Z}/q\mathbb{Z}$, and to only consider representatives of these classes in the integers at the end, at the decryption stage. One can show that this is mathematically sound and will not lead to inconsistencies.

Secondly we note that if one will be picking representatives for the classes, it is even more advantageous to pick representatives in the interval $[-\lfloor \frac{q}{2} \rfloor, \lfloor \frac{q}{2} \rfloor]$ if q is odd and in the interval $[-\frac{q}{2} + 1, \frac{q}{2}]$ if q is even (so called "balanced" representatives). This leads to the less awkward decryption condition that the message decrypts to be 0 if the representative r of $b - \vec{a} \cdot \vec{s} \pmod{q}$ satisfies

$$-\frac{q}{4} < r < \frac{q}{4},$$

and decrypts to be 1 otherwise.

4.1.1 Sage commands

Sage already has some code which can be used to play with Regev's version of LWE (or any version, it can be customized).

First, I mentioned in class that in his paper Regev gives suggestions for what q and χ should be given a choice of parameter n. To find out his suggestion, we can ask Sage to create a Regev-LWE oracle and see what parameters Sage chooses:

```
from sage.crypto.lwe import Regev
my_lwe = Regev(7); my_lwe
```

This should prompt Sage to respond

```
LWE(7, 53, Discrete Gaussian sampler over the Integers with sigma =
1.014010 and c = 53, 'uniform', None).
```

Let's dissect what this means:

- The first parameter is n, so this tells us that $n = 7$.

- The next parameter is q, so we learn that with $n = 7$, Regev suggests we use $q = 53$. Looking at the original paper, we see that Regev suggests that q be prime and $n^2 < q < 2n^2$. Sage sensibly chooses the prime number immediately following n^2 in its implementation.

- Then we see the discrete normal distribution used for the errors; here the discrete normal has $\sigma = 1.014010$ and mean $\mu = 53$ (Sage uses c for "center" but in mathematics it's more common to use μ for the mean of a distribution). We note that Regev recommends a discrete normal with $\sigma = \frac{q}{\sqrt{2\pi n}(\ln n)^2}$, which is what 1.014010 is. We also note that while the mean should "technically" be 0, since Sage does all of the computations modulo q choosing the mean to be q does not affect the final answer.

- The next parameter is `'uniform,'` which says that the secret will be chosen uniformly at random from $\mathbb{Z}/q\mathbb{Z}^n$.

- Finally, the last parameter is `'none,'` which says that we are not putting any upper bound on m, the number of LWE pairs that can be sampled by a user.

If we want to actually see some LWE pairs generated with these parameters, we can type either

```
my_lwe()
```

to get one pair, or if we want many pairs quickly, we can type

```
my_pairs = [my_lwe() for _ in range(10)]; my_pairs
```

which will give 10 LWE pairs. One can also use a different syntax that doesn't explicitly create a LWE object:

```
from sage.crypto.lwe import samples
samples(7,10,'Regev')
```

which will give 10 LWE pairs with $n = 7$ and other parameters as specified by Regev.

Finally, if we need access to the secret $\vec{s}$ to decrypt, we can get it by typing

```
s = my_lwe._LWE__s; s
```

For example, if we want to see all of the errors from our sample, we can list them:

```
my_errors = [a.dot_product(s) - b for (a,b) in my_pairs]; my_errors
```

4.1.2 Brakerski-Vaikuntanathan (BV) LWE

We present now a slight variation on Regev's LWE cryptographic scheme. We note that the original BV scheme was for R-LWE, which we will cover later, but there doesn't seem to be any harm in introducing their ideas as an LWE scheme.

The set up is essentially the same: One chooses a size n, a modulus q, a number m of equations, and a probability distribution χ for the errors. Then one generates a secret vector $\vec{s}$ and m vectors $\vec{a}_i$. However, to complete the LWE pairs, we use the formula

$$b_i = \vec{a}_i \cdot \vec{s} + 2e_i.$$

The public key is again $(q, \{(\vec{a}_i, b_i)\}_{i=1}^m)$.

The encryption then is also slightly different: To send a 1-bit message $x \in \{0, 1\}$, the sender chooses a random set $T \subseteq \{1, 2, \ldots, m\}$, and sends the pair $(\vec{a}, b)$, where as before $\vec{a} = \sum_{i \in T} \vec{a}_i$, but this time

$$b = x + \sum_{i \in T} b_i$$

(this is instead of $b = x\lfloor \frac{q}{2} \rfloor + \sum_{i \in T} b_i$ in the classical Regev scheme).

To decrypt, it then suffices to compute

$$b - \vec{a} \cdot \vec{s} \pmod 2,$$

as this will give the value x. Indeed, we have

$$\begin{aligned} b - \vec{a} \cdot \vec{s} &= x + \sum_{i \in T} b_i - \vec{a} \cdot \vec{s} \\ &= x + \sum_{i \in T} b_i - \left(\sum_{i \in T} \vec{a}_i \right) \cdot \vec{s} \\ &= x + \sum_{i \in T} b_i - \sum_{i \in T} (\vec{a}_i \cdot \vec{s}) \\ &= x + \sum_{i \in T} (b_i - \vec{a}_i \cdot \vec{s}) \\ &= x + \sum_{i \in T} 2e_i \\ &\equiv x \pmod 2. \end{aligned}$$

4.2 Hardness of the problems

4.2.1 Lattice problems

Throughout, we will refer to a set of pairs $(\vec{a}_i, b_i)$ as a *set of LWE pairs* if there is a common vector $\vec{s}$ such that

$$b_i = \vec{a}_i \cdot \vec{s} + e_i,$$

for e_i drawn from some error distribution χ. (Here we leave the meaning of "error distribution" vague to mean any probability distribution that outputs "small values" for a definition of small that we leave to be established as needed in context.)

Throughout, we will need a notion of size for a vector. For $\vec{z} = (z_1, \dots z_n) \in \mathbb{Z}^n$, we will write

$$\|\vec{z}\| = \sqrt{z_1^2 + z_2^2 + \cdots + z_n^2},$$

for the Euclidean norm of $\vec{z}$. Note that if in fact $\vec{z} \in (\mathbb{Z}/q\mathbb{Z})^n$, then the norm is not well-defined, although $\|\vec{z}\|^2 \in \mathbb{Z}/q\mathbb{Z}$ is well-defined.

We begin with the two problems most closely related to LWE:

Problem 4.2.1 (Search-LWE, or just LWE)**.** Given a set of LWE pairs $\{(\vec{a}_i, b_i)\}$, compute the secret $\vec{s}$.

Problem 4.2.2 (Decision-LWE)**.** Given a set of pairs $\{(\vec{a}_i, b_i)\}$, decide if these pairs are LWE, or if for each i, b_i was just chosen uniformly randomly from the set $\mathbb{Z}/q\mathbb{Z}$ (and randomly lifted to an integer if we are working with $b_i \in \mathbb{Z}$ as we are).

These are related to a whole host of so-called "lattice problems." We highlight here a selection of them:

Problem 4.2.3 (Short Integer Solution (SIS))**.** Fix $\beta > 0$. Given an $n \times m$ matrix A with entries in $\mathbb{Z}/q\mathbb{Z}$ find a nonzero vector $\vec{z} \in \mathbb{Z}^m$ such that

1. $\|\vec{z}\| \leq \beta$, and
2. $A\vec{z} \equiv 0 \pmod{q}$.

Note that to guarantee that there exists such a $\vec{z}$, one can require $\beta \geq \sqrt{n \log q}$ (but at the same time, so the problem is not trivial, one should require $\beta < q$ so that $\vec{z} = (q, 0, 0, \dots, 0)$ is not a solution), and $m \geq n \log q$.

Problem 4.2.4 (Shortest Vector (SVP))**.** Given a basis $\vec{v}_1, \vec{v}_2, \dots, \vec{v}_n$ for a lattice L, give the shortest nonzero vector $\vec{v}$ belonging to the lattice. In other words, let

$$\lambda(L) = \min_{0 \neq \vec{v} \in L} \|\vec{v}\| . \tag{4.1}$$

Then give a vector $\vec{v} \in L$ with $\|\vec{v}\| = \lambda(L)$.

A variation on this problem is this:

Problem 4.2.5 (GapSVP)**.** Fix a constant $\beta > 0$. Given a basis $\vec{v}_1, \vec{v}_2, \dots, \vec{v}_n$ for a lattice L, decide if the shortest vector of L has length less than or equal to 1, or strictly greater than β. In other words, with $\lambda(L)$ as in equation (4.1), decide if $\lambda(L) \leq 1$ or $\beta < \lambda(L)$.

Note that in the GapSVP problem, we promise that the case of $1 < \lambda(L) \leq \beta$ will not be asked. Or if it is asked, we allow our oracle to give the wrong answer or refuse to answer in this case. In other words, we pretend as much as possible that only those two options ($\lambda(L) \leq 1$ or $\beta < \lambda(L)$) are possible, and simply ask to decide which of the two options is the case to consider the problem solved.

In his paper, Regev shows that there is quantum reduction (i.e. one that requires a quantum computer) from the LWE problem to the GapSVP problem for certain parameters. This means that if one has an oracle that can solve the LWE problem using only a polynomial number of LWE pairs, as long as certain requirements are satisfied by q and χ, then this same oracle can solve the GapSVP problem for a value of β which depends on q and χ. This establishes that LWE should be a hard problem. The author notes that there probably is a nonquantum reduction, but none is known currently as far as we know.

4.2.2 Decision implies search

Instead of wading into the difficult hardness proofs, we instead now present some easier reduction results to give a taste of the subject. We begin with a counter-intuitive result. It seems clear that if one can solve the Search-LWE problem, then one can solve the Decision-LWE problem. (One could apply the Search-LWE oracle to the Decision-LWE sample, see what vector $\vec{s}$ comes out and see if for each i $b_i - \vec{a}_i \cdot \vec{s}$ is distributed according to χ.) However, it turns out that the other way works too! If one can only tell apart LWE pairs from non-LWE pairs, then one can recover $\vec{s}$, as long as q is polynomial in n.

Proposition 4.2.6. *If one has an oracle that can solve the Decision-LWE problem, then one can solve the Search-LWE problem, as long as q is polynomial in n.*

Proof. Let $k \in \mathbb{Z}/q\mathbb{Z}$. We show how to determine if s_j, the jth coordinate of the vector $\vec{s}$, is equal to k. By repeating this process for different values of k for each $j = 1, \ldots, n$, since q is polynomial in n, one can recover $\vec{s}$ by applying this process a number of times that is polynomial in n.

To determine if $s_j = k$, we generate m random integers r_i and replace the pairs $\{(\vec{a}_i, b_i)\}$ with the pairs

$$\{(\vec{a}'_i = \vec{a}_i + (0, \ldots, 0, r_i, 0, \ldots 0), b'_i = b_i + r_i k)\},$$

where r_i is in the jth coordinate of the vector.

If $s_j = k$, then this new set of pairs will be an LWE pair according to the oracle. If $s_j \neq k$, then the oracle will reject this set as being random.

The reason why the new pairs are LWE if $s_j = k$ is the following. For each i, we have:

$$\begin{aligned} b'_i - \vec{a}'_i \cdot \vec{s} &= (b_i + r_i k) - (\vec{a}_i + (0, \ldots, 0, r_i, 0, \ldots 0)) \cdot \vec{s} \\ &= b_i + r_i k - \vec{a}_i \cdot \vec{s} - (0, \ldots, 0, r_i, 0, \ldots 0) \cdot \vec{s} \\ &= (b_i - \vec{a}_i \cdot \vec{s}) + r_i k - r_i s_j \\ &= e_i + r_i (k - s_j). \end{aligned}$$

We see that if $s_j = k$, then we are left only with e_i, and so the pairs are LWE. Otherwise, the number $r_i(k - s_j)$ behaves like a random number, since r_i is random, so the pairs are seen as random by the oracle. □

4.2.3 SIS implies decision

We also present another nice result: If one has an SIS oracle, then one can solve Decision-LWE:

Proposition 4.2.7. *If one has an oracle that can solve the SIS problem, then one can solve the Decision-LWE problem with high probability.*

Proof. Suppose that we have a list of m pairs $\{(\vec{a}_i, b_i)\}$ and want to determine if they are LWE. Suppose further that for any matrix we can solve the Short Integer Solution problem efficiently. Then we can solve the Decision-LWE problem as follows.

We begin by forming a large number of subsets $T_j \subset \{1, 2, \ldots, m\}$, $j = 1, 2, \ldots, N$, say. For each j, we form the matrix

$$A_j = \begin{pmatrix} \vec{a}_{k_1} \\ \vec{a}_{k_2} \\ \cdots \\ \vec{a}_{k_\ell} \end{pmatrix},$$

whose rows are the elements $\vec{a}_k$ for $k \in T_j$, and then use the SIS solver to give a short integer solution $\vec{z}_j$ to the equation $A_j^T \vec{z}_j \equiv 0 \pmod{q}$.

Then for each j we form the vector

$$\vec{B}_j = \begin{pmatrix} b_{k_1} \\ b_{k_2} \\ \cdots \\ b_{k_\ell} \end{pmatrix},$$

and compute $\vec{B}_j^T \vec{z}_j$. If the values $\vec{B}_j^T \vec{z}_j$ are "small" consistently, then we conclude that the pairs were LWE. If the values $\vec{B}_j^T \vec{z}_j$ are uniformly distributed in $\mathbb{Z}/q\mathbb{Z}$, then we conclude that the pairs were not LWE.

The reason why this works is the following: If the pairs were LWE all along, then we have that for each j

$$\vec{B}_j = A_j \vec{s} + \vec{E}_j,$$

where $\vec{E}_j$ is the vector of errors. In that case then, we have that

$$\begin{aligned} \vec{B}_j^T \vec{z}_j &= (A_j \vec{s} + \vec{E}_j)^T \vec{z}_j \\ &= (\vec{s}^T A_j^T + \vec{E}_j^T) \vec{z}_j \\ &= \vec{s}^T A_j^T \vec{z}_j + \vec{E}_j^T \vec{z}_j \\ &= \vec{E}_j^T \vec{z}_j, \end{aligned}$$

since $A_j^T \vec{z}_j \equiv 0 \pmod{q}$. Recall that we have that

$$\vec{E}_j \cdot \vec{z}_j \leq \left\| \vec{E}_j \right\| \|\vec{z}_j\|,$$

and $\left\| \vec{E}_j \right\|$ is smaller in absolute value than $\frac{q}{4}$ with high probability. Then if $\|\vec{z}_j\|$ is small enough (which we can control with the oracle), we will see a small value of $\vec{E}_j \cdot \vec{z}_j$ with high probability and therefore very often.

If the pairs are not LWE, then $\vec{B}_j^T \vec{z}_j$ is just a uniformly chosen random vector whose dot product is taken with a short vector. This will not be as short, and should sometimes be big even, so with high probability we will see larger values. □

4.3 Ring-LWE

Warning: This section is still under construction and will require some edits before being understandable. Read at your own risk!

The Ring Learning with Error problem (R-LWE) is pretty much identical to the LWE problem, except with polynomials instead of vectors. There are several set ups, we present here the Brakersky-Vaikuntanathan set up as it is one of the early homomorphic ones.

To be precise, here is the encryption/decryption scheme, including the set up:

1. Setting up:

   - Choose a prime q, a polynomial $\Phi(x)$ that is irreducible over $\mathbb{F}_q[x]$ (the ring of polynomials with coefficients in the finite field with q elements) of degree n, and a positive integer m.
   - Choose a probability distribution χ with values in the quotient ring $\mathbb{F}_q[x]/\Phi(x)$ from which to draw errors. We will not dwell on the exact properties that χ should have, but the polynomials that it outputs should have "small" coefficients.
   - Generate a secret polynomial $s(x) \in \mathbb{F}_q[x]/\Phi(x)$.
   - Generate m polynomials $a_i(x) \in \mathbb{F}_q[x]/\Phi(x)$.
   - Generate m error polynomials $e_i(x) \in \mathbb{F}_q[x]/\Phi(x)$, drawn from the distribution χ.
   - Compute the m polynomials

     $$b_i(x) = a_i(x)s(x) + 2e_i(x),$$

     where here the operations are the usual polynomial multiplication and addition in $\mathbb{F}_q[x]/\Phi(x)$.

   Then the public key is the data of the ring $\mathbb{F}_q[x]/\Phi(x)$, and the m pairs $(a_i(x), b_i(x))$.

2. Encryption: This scheme allows one to send a polynomial $y \in \mathbb{F}_q[x]/\Phi(x)$ all of whose coefficients are 0 or 1. This essentially allows one to send $\deg(\Phi(x))$ different 1-bit messages at once.

- The sender chooses a random set $T \subseteq \{1, \ldots, m\}$.
- They form, and send, the pair $(a(x), b(x))$, where

$$a(x) = \sum_{i \in T} a_i(x), \quad \text{and} \quad b(x) = y(x) + \sum_{i \in T} b_i(x).$$

3. Decryption: To decrypt, the receiver computes

$$b(x) - a(x)s(x) \pmod 2.$$

This will give the value $y(x)$. The reason for this is the following; we have

$$\begin{aligned} b(x) - a(x)s(x) &= y(x) + \sum_{i \in T} b_i(x) - \left(\sum_{i \in T} a_i(x) \right) s(x) \\ &= y(x) + \sum_{i \in T} (a_i(x)s(x) + 2e_i(x)) - \sum_{i \in T} a_i(x)s(x) \\ &= y(x) + \sum_{i \in T} (a_i(x)s(x) + 2e_i(x) - a_i(x)s(x)) \\ &= y(x) + \sum_{i \in T} 2e_i(x) \\ &\equiv y(x) \pmod 2. \end{aligned}$$

Here we used that the coefficients of $y(x)$ are all 0 or 1 to conclude that $y(x) \equiv y(x) \pmod 2$. (This is a weird statement to make, as it is trivially true mathematically, but we mean here that $y(x)$ did not have "more information" before reducing modulo 2.)

4.3.1 Fully Homomorphic Encryption

One reason to use R-LWE rather than LWE is that it requires small key sizes. Another is that it allows one to define *fully homomorphic encryption schemes*. By this we mean an encryption scheme where if m is the message and `Encr` is the encryption method, then we have that

$$\texttt{Encr}(m_1) + \texttt{Encr}(m_2) = \texttt{Encr}(m_1 + m_2)$$

and

$$\texttt{Encr}(m_1) \cdot \texttt{Encr}(m_2) = \texttt{Encr}(m_1 \cdot m_2).$$

In other words, one can perform operations on the encrypted information, and obtain an encryption of the result of the operation on the original information. This is useful for computations in the cloud, where one might want to keep the information secure (encrypted) while having a third party perform operations on the information.

Let us consider the R-LWE scheme presented here. Suppose that one has two message polynomials y_1 and y_2. Suppose further that (a, b) is a pair encrypting y_1 and (c, d) is pair

encrypting y_2. For simplicity, we will further suppose that in each case the sets T contained only one element; this is not necessary but will lighten the notation considerably. We also drop the variable x from the notation to similarly lighten the notation.

Then to be clear we have

$$b = y_1 + as + 2e_1,$$

where e_1 is an error polynomial (really a sum of them, but a sum of error polynomials should remain an error polynomial) and

$$d = y_2 + cs + 2e_2,$$

and again e_2 is an error polynomial.

We have that

$$b + d = (y_1 + y_2) + (a + c)s + 2(e_1 + e_2),$$

so the pair $(a + c, b + d)$ is an encryption of $y_1 + y_2$, at the cost of the error having grown a little bit more. (We can see this by computing

$$(b + d) - (a + c)s \equiv y_1 + y_2 \pmod 2.)$$

Therefore the additivity of the encryption scheme is easily obtained.

Let us consider now the trickier case of the multiplication. We have

$$\begin{aligned} bd &= (y_1 + as + 2e_1)(y_2 + cs + 2e_2) \\ &= y_1y_2 + (y_1c + y_2a + 2e_2a + 2e_1c)s + acs^2 + 2y_1e_2 + 2e_1y_2 + 4e_1e_2 \\ &= y_1y_2 + (y_1c + y_2a)s + acs^2 + 2e_1(cs + y_2) + 2e_2(as + y_1) + 4e_1e_2 \end{aligned}$$

If we compute

$$bd - acs \pmod 2,$$

we do not get y_1y_2. So without modification the BV scheme is not fully homomorphic.

Therefore the authors generalize their scheme in the following way:

1. Setting up: The set up remains the same, but one computes a secret vector

   $$\vec{s} = (1, s, s^2, s^3, \ldots, s^D)$$

   for some value D which will determine how many multiplications can be done on the ciphertexts.

2. Encryption remains mostly the same, except that to encrypt the polynomial $y(x)$, we will output the pair $(b, -a)$, where, as before,

   $$a(x) = \sum_{i \in T} a_i(x), \quad \text{and} \quad b(x) = y(x) + \sum_{i \in T} b_i(x).$$

   We also note that in general now a ciphertext will not necessarily be a pair, but instead a tuple $\vec{c} = (c_0, c_1, c_2, \ldots, c_D)$. The length of the tuple will depend on how many

multiplications the ciphertext has gone through: A 'freshly encrypted" message will be a pair (c_0, c_1) with $c_0 = b$ and $c_1 = -a$, but the ciphertext coming from multiplying two "fresh" ciphertexts will be a triple (c_0, c_1, c_2), for example, and in general the ciphertext coming from multiplying a ciphertext of length $D_1 + 1$ with a ciphertext of length $D_2 + 1$ will be of length $D_1 + D_2 + 1$.

3. Ciphertext addition and multiplication: This is a new category! Since the scheme is fully homomorphic, we must also provide an addition and a multiplication, not just an encryption and a decryption.

   The addition will be component-wise; if $\vec{c_1} = (c_{10}, c_{11}, c_{12}, \ldots, c_{1D})$ and $\vec{c_2} = (c_{20}, c_{21}, c_{22}, \ldots, c_{2D})$ (where we might have padded $\vec{c}_1$ and $\vec{c}_2$ with zeroes at the end to get them both to be length D) then

   $$\vec{c}_1 + \vec{c}_2 = (c_{10} + c_{20}, c_{11} + c_{21}, \ldots, c_{1D} + c_{2D}).$$

   The multiplication is slightly more complicated. If $\vec{c_1} = (c_{10}, c_{11}, c_{12}, \ldots, c_{1D_1})$ and $\vec{c_2} = (c_{20}, c_{21}, c_{22}, \ldots, c_{2D_2})$ (here there is no need to pad for both vectors to be the same length), let U be a variable and consider the two polynomials

   $$c_1(U) = c_{10} + c_{11}U + c_{12}U^2 + \cdots + c_{1D_1}U^{D_1} \quad \text{and} \quad c_2(U) = c_{20} + c_{21}U + c_{22}U^2 + \cdots + c_{2D_2}U^{D_2}.$$

   Then their product $\hat{c}(U) = c_1(U)c_2(U)$ is also a polynomial and we can just compute what it is using normal polynomial multiplication. It has some coefficients:

   $$\hat{c}(U) = \sum_{j=0}^{D_1+D_2} \hat{c}_i U^i = \hat{c}_0 + \hat{c}_1 U + \ldots + \hat{c}_{D_1+D_2} U^{D_1+D_2}.$$

   Then we define the multiplication of the two ciphertexts $\vec{c}_1\vec{c}_2$ to be

   $$\vec{c}_1\vec{c}_2 = (\hat{c}_0, \hat{c}_1, \ldots, \hat{c}_{D_1+D_2}).$$

4. The decryption algorithm for a general ciphertext $\vec{c} = (c_0, c_1, \ldots, c_D)$ is just

   $$\vec{c} \cdot \vec{s} \pmod 2.$$

We see now how the choice of D at the beginning affects how many multiplication we can perform: Every multiplication makes the ciphertext longer, but we can only decrypt vectors $\vec{c}$ that have $D + 1$ or fewer entries.

Let us now verify that this scheme is fully homomorphic. Throughout, let $\vec{c}_1$ be an encryption of y_1, by which we mean that

$$\vec{c}_1 \cdot \vec{s} \equiv y_1 \pmod 2,$$

and let let $\vec{c}_2$ be an encryption of y_2, by which we mean that

$$\vec{c}_2 \cdot \vec{s} \equiv y_2 \pmod 2.$$

To show that this new scheme preserves addition, we must then show that $\vec{c}_1 + \vec{c}_2$ is an encryption of $y_1 + y_2$, or equivalently that $(\vec{c}_1 + \vec{c}_2) \cdot \vec{s} \equiv y_1 + y_2 \pmod 2$. Indeed, we compute:

$$(\vec{c}_1 + \vec{c}_2) \cdot \vec{s} = \vec{c}_1 \cdot \vec{s} + \vec{c}_2 \cdot \vec{s} \equiv y_1 + y_2 \pmod 2.$$

Finally we get to the multiplication: For simplicity and concreteness, let us consider the multiplication of two "fresh" ciphertexts. The general proof is not much more difficult but perhaps less illuminating. To keep the notation from before, let $\vec{c}_1 = (b, -a)$, which is still an encryption of y_1, and let $\vec{c}_2 = (d, -c)$, which is still an encryption of y_2, per our earlier assumption. Then the ciphertext product of $\vec{c}_1$ and $\vec{c}_2$ is $(bd, -(ad + bc), ac)$, because of the product

because

$$(b - aU)(d - cU) = bd - (ad + bc)U + acU^2.$$

We verify that this indeed decrypts to $y_1 y_2$:

$$\begin{aligned}
(bd, -(ad + bc), ac) \cdot (1, s, s^2) &= bd - (ad + bc)s + acs^2 \\
&= (b - as)(d - cs) \\
&= ((b, -a) \cdot (1, s)) \, ((d, -c) \cdot (1, s)) \\
&= (\vec{c}_1 \cdot \vec{s}) \, (\vec{c}_2 \cdot \vec{s}) \\
&\equiv y_1 y_2 \pmod 2.
\end{aligned}$$