Lattice based cryptography

Boaz Barak

**Disclaimer:** The MS-Word version of this text has significant formatting issues compared to the PDF and HTML versions.

# Lattice based cryptography

Lattice based public key encryption (and its cousins known as knapsack and coding based encryption) have almost as long a history as discrete logarithm and factoring based schemes. Already in 1976, right after the Diffie-Hellman key exchange was discovered (and before RSA), Ralph Merkle was working on building public key encryption from the NP hard *knapsack* problem (see [Diffie’s recollection](http://cr.yp.to/bib/1988/diffie.pdf)). This can be thought of as the task of solving a linear equation of the form (where is a given matrix, is a given vector, and the unknown are ) over the real numbers but with the additional constraint that must be either or . His proposal evolved into the Merkle-Hellman system proposed in 1978 (which was broken in 1984).

McEliece proposed in 1978 a system based on the difficulty of the decoding problem for general linear codes. This is the task of solving *noisy linear equations* where one is given and such that for a “small” error vector , and needs to recover . Crucially, here we work in a finite field, such as working modulo for some prime (that can even be ) rather than over the reals or rationals. There are special matrices for which we know how to solve this problem efficiently: these are known as efficiently decodable [error correcting codes](https://goo.gl/vM7Pvv). McEliece suggested a scheme where the key generator lets be a “scrambled” version of a special (based on the [Goppa algebraic geometric code](https://goo.gl/Vd4yye)). So, someone that knows the scrambling could solve the problem, but (hopefully) someone that doesn’t know it wouldn’t. McEliece’s system has so far not been broken.

In a 1996 breakthrough, Ajtai showed a *private key* scheme based on integer lattices that had a very curious property- its security could be based on the assumption that certain problems were only hard in the *worst case*, and moreover variants of these problems were known to be NP hard. This re-ignited the hope that we could perhaps realize the old dream of basing crypto on the mere assumption that . Alas, we now understand that there are fundamental barriers to this approach.

Nevertheless, Ajtai’s work attracted significant interest, and within a year both Ajtai and Dwork, as well as Goldreich, Goldwasser and Halevi came up with lattice based constructions for *public key* encryption (the former based also on *worst case* assumptions). At about the same time, Hoffstein, Pipher, and Silverman came up with their NTRU public key system which is based on stronger assumptions but offers better performance, and they started a company around it together with Daniel Lieman.

You may note that I haven’t yet said what *lattices* are; we will do so later, but for now if you simply think of questions involving linear equations modulo some prime , you will get enough of the intuition that you need. (The lattice viewpoint is more geometric, and we’ll discuss it more below; it was first used to *attack* cryptosystems and in particular break the Merkle-Hellman knapsack scheme and many of its variants.)

Lattice based cryptography has captured a lot of attention recently from both theory and practice. In the theory side, many cool new constructions are now based on lattice based cryptography, and chief among them fully homomorphic encryption, as well as indistinguishability obfuscation (though the latter’s security’s foundations are still far less solid). On the applied side, the steady advances in the technology of quantum computers have finally gotten practitioners worried about RSA, Diffie Hellman and Elliptic Curves. While current constructions for quantum computers are nowhere near being able to, say, factor larger numbers that can be done classically (or even than can be done by hand), given that it takes many years to develop new standards and get them deployed, many believe the effort to transition away from these factoring/dlog based schemes should start today (or perhaps should have started several years ago). The NSA has [suggested](https://www.nsa.gov/ia/programs/suiteb_cryptography/index.shtml) that it plans to initiate the process to “transition to quantum resistant algorithms in the not too distant future”; see also this [very interesting FAQ](https://cryptome.org/2016/01/CNSA-Suite-and-Quantum-Computing-FAQ.pdf) on this topic.

Cryptography has the peculiar/unfortunate feature that if a machine is built that can factor large integers in 20 years, it can still be used to break the communication we transmit *today*, provided this communication was recorded. So, if you have some data that you expect you’d want still kept secret in 20 years (as many government and commercial entities do), you might have reasons to worry. Currently lattice based cryptography is the only real “game in town” for potentially quantum-resistant public key encryption schemes.

Lattice based cryptography is a huge area, and in this lecture and this course we only touch on few aspects of it. I highly recommend [Chris Peikert’s Survey](https://web.eecs.umich.edu/~cpeikert/pubs/lattice-survey.pdf) for a much more in depth treatment of this area.

### Quick linear algebra recap

A *field* is a set that supports the operations and contains the numbers and (more formally the additive identity and multiplicative identity) with the usual properties that the real numbers have. (That is associative, commutative, and distributive law, the fact that for every there is an element such that and that if there is an element such that .) Apart from the real numbers, the main field we will be interested in this section is the field of the numbers with addition and multiplication done modulo , where is a prime number.[[1]](#footnote-28)

You should be comfortable with the following notions:

* A vector and a *matrix* . An matrix has rows and columns. We think of vectors as *column vectors* and so we can think of a vector as an matrix. We write the -the coordinate of as and the -th coordinate of as (i.e. the coordinate in the -th row and the -th column.) We often write a vector as but we still mean that it’s a column vector unless we say otherwise.
* If is a *scalar* (i.e., a number) and is a vector then is the vector . If are dimensional vectors then is the vector .
* A *linear subspace* is a non-empty set of vectors such that for every vectors and , . In particular this means that contains the all zero vector (can you see why?). A subset is *linearly independent* if there is no collection and scalars such that . It is known (and not hard to prove) that if is linearly independent then . It is known that for every such linear subspace there is a linearly independent set of vectors, with , such that for every there exist such that . Such a set is known as a *basis* for . A subspace has many bases, but all of them have the same size which is known as the *dimension* of . An *affine subspace* is a set of the form where is a linear subspace. We can also write as . We denote the dimension of as the dimension of in such a case.
* The inner product (also known as “dot product”) between two vectors of the same dimension that is defined as (addition done in the field ).[[2]](#footnote-29)
* The *matrix product* of an and a matrix, that results in an matrix. If we think of the rows of as the vectors and the columns of as , then the -th coordinate of is . Matrix product is associative and satisfies the distributive law but is *not commutative*: there are pairs of square matrices such that .
* The *transpose* of an matrix is the matrix such that .
* The *inverse* of a square matrix is the matrix such that where is the *identity matrix* such that if and otherwise.
* The *rank* of an matrix is the minimum number such that we can write as where and . We can think of the ’s as the columns of an matrix and the ’s as the rows of an matrix , and hence the rank of is the minimum such that where is and is . It can be shown that an matrix is full rank if and only if it has an inverse.
* Solving *linear equations* can be thought of as the task of given an matrix and -dimensional vector , finding the -dimensional vector such that . If the rank of is at least (which in particular means that ) then it means that by dropping rows of and coordinates of we can obtain the equation where is an matrix that has an inverse. In this case a solution (if it exists) will be equal to . If for a set of equations we have and we can find two such matrices such that then we say it is *over determined* and in such a case it has no solutions. If a set of equations has more variables than equations we say it’s *under-determined*. In such a case it either has no solutions or the solutions form an affinte subspace of dimension at least .
* The *gaussian elimination* algorithm can be used to obtain, given a set of equations a solution to if such exists or a certification that no solution exists. It can be executed in time polynomial in the dimensions and the bit complexity of the numbers involved. This algorithm can also be used to obtain an inverse of a given matrix , if such an inverse exists.

Throughout this chapter, and while working in lattice based cryptography in general, it is crucial to keep track of the dimensions. Whenever you see a symbol such as ask yourself:

* Is it a *scalar*, a *vector* or a *matrix*?
* If it is a vector or a matrix, what are its dimensions?
* If it’s a matrix, is it “square” (i.e., ), “short and fat” (i.e., ) or “tall and skinny”? ()?

## A world without Gaussian elimination

The general approach people used to get a public key encryption is to obtain a hard computational problem with some mathematical *structure*. We’ve seen this in the *discrete logarithm* problem, where the task is to invert the map , and the integer factoring problem, where the task is to invert the map . Perhaps the simplest structure to consider is the task of solving linear equations.

Pretend that we didn’t know of Gaussian elimination,[[3]](#footnote-32) and that if we picked a “generic” matrix then the map would be hard to invert. (Here and elsewhere, our default interpretation of a vector is as a *column* vector, and hence if is dimensional and is then is dimensional. We use to denote the row vector obtained by *transposing* .) Could we use that to get a public key encryption scheme?

Here is a concrete approach. Let us fix some prime (think of it as polynomial size, e.g., is smaller than or so, though people can and sometimes do consider of exponential size), and all computation below will be done modulo . The secret key is a vector , and the public key is where is a random matrix with entries in and . Under our assumption, it is hard to recover the secret key from the public key, but how do we use the public key to encrypt?

The crucial observation is that even if we don’t know how to solve linear equations, we can still combine several equations to get new ones. To keep things simple, let’s consider the case of encrypting a single bit.

# 

If you have a CPA secure public key encryption scheme for single bit messages then you can extend it to a CPA secure encryption scheme for messages of any length. Can you see why?

If are the rows of , we can think of the public key as the set of equations in the unknown variables . The idea is that to encrypt the value we will generate a new *correct* equation on , while to encrypt the value we will generate an *incorrect* equation. To decrypt a ciphertext , we think of it as an equation of the form and output if and only if the equation is correct.

How does the encrypting algorithm, that does not know , get a correct or incorrect equation on demand? One way would be to simply take two equations and and add them together to get the equation . This equation is correct and so one can use it to encrypt , while to encrypt we simply add some fixed nonzero number to the right hand side to get the incorrect equation . However, even if it’s hard to solve for given the equations, an attacker (who also knows the public key ) can try itself all pairs of equations and do the same thing.

Our solution for this is simple- just add more equations! If the encryptor adds a random subset of equations then there are possibilities for that, and an attacker can’t guess them all. That is, if the rows of are , then we can pick a vector at random, and consider the equation where and . In other words, we can think of this as the equation (note that and so we can think of this as the equation that we obtain from by multiplying both sides on the right by the row vector ).

Thus, at least intuitively, the following encryption scheme would be “secure” in the Gaussian-elimination free world of attackers that haven’t taken freshman linear algebra:

**Scheme “LwoE-ENC”:** Public key encryption under the hardness of “learning linear equations without errors”.

* *Key generation*: Pick random matrix over , and , the secret key is and the public key is where .
* *Encryption*: To encrypt a message , pick and output for some fixed nonzero .
* *Decryption:* To decrypt a ciphertext , output iff .

# 

Please stop here and make sure that you see why this is a valid encryption (not in the sense that it is secure - it’s not - but in the sense that decryption of an encryption of returns the bit ), and this description corresponds to the previous one; as usual all calculations are done modulo .

## Security in the real world.

Like it or not (and cryptographers typically don’t) Gaussian elimination *is* possible in the real world and the scheme above is completely insecure. However, the Gaussian elimination algorithm is extremely *brittle*.  
Errors tend to be amplified when you combine equations. This is usually thought of as a bad thing, and numerical analysis is much about dealing with issue. However, from the cryptographic point of view, these errors can be our saving grace and enable us to salvage the security of the ridiculous scheme above.

To see why Gaussian elimination is brittle, let us recall how it works. Think of for simplicity. Given equations in the unknown variables , the goal of Gaussian elimination is to transform them into the equations where is the identity matrix (and hence the solution is simply ). Recall how we do it: by rearranging and scaling, we can assume that the top left corner of is equal to , and then we add the first equation to the other equations (scaled appropriately) to zero out the first entry in all the other rows of (i.e., make the first column of equal to ) and continue onwards to the second column and so on and so forth.

Now, suppose that the equations were *noisy*, in the sense that we added to a vector such that for every .[[4]](#footnote-37) Even ignoring the effect of the scaling step, simply adding the first equation to the rest of the equations would typically tend to increase the relative error of equations from to . Now, when we repeat the process, we increase the error of equations from to , and we see that by the time we’re done dealing with about variables, the remaining equations have error level roughly . So, unless was truly tiny (and truly big, in which case the difference between working in and simply working with integers or rationals disappears), the resulting equations have the form where is so big that we get no information on .

The *Learning With Errors (LWE)* conjecture is that this is *inherent*:

**Conjecture (Learning with Errors, Regev 2005):** Let and be some functions. The *Learning with Error (LWE) conjecture with respect to* ,denoted as , is the following conjecture: for every polynomial and polynomial-time adversary ,

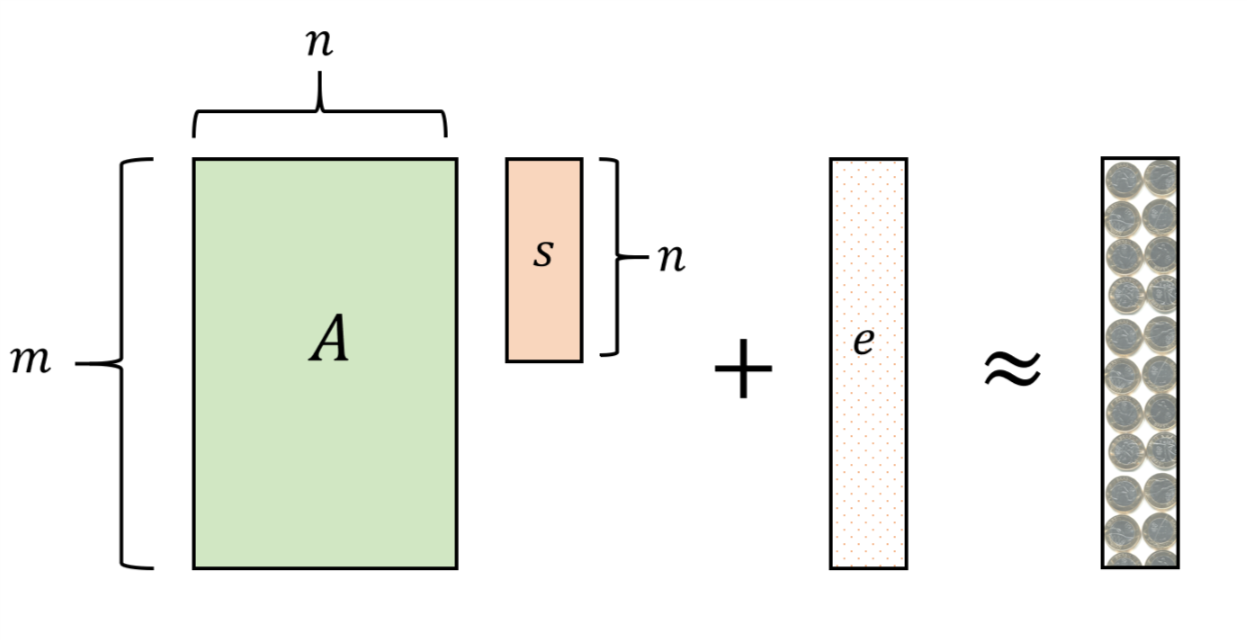
where for and , this probability is taken over a random matrix in , a random vector in and a random “noise vector” in where for every .[[5]](#footnote-38)

The *LWE conjecture* (without any parameters) is that there is some absolute constant such that for every polynomial there, if then LWE holds with respect to and .[[6]](#footnote-39)

It is important to note the order of quantifiers in the learning with error conjecture. If we want to handle a noise of low enough magnitude (say ) then we need to choose the modulos to be large enough (for example it is believed that will be good enough for this case) and then the adversary can choose to be as big a polynomial as they like, and of course run in time which is an arbitrary polynomial in . Therefore we can think of such an adversary as getting access to a “magic box” that they can use number of times to get “noisy equations on ” of the form with , where ).

## Search to decision

It turns out that if the LWE is hard, then it is even hard to distinguish between random equations and nearly correct ones:



The search to decision reduction (LWEsearchtodecthm) implies that under the LWE conjecture, for every , if we choose and fix a random matrix over , the distribution is indistinguishable from a random vector in , where is a random vector in and is a random “short” vector in . The two distributions are indistinguishable even to an adversary that knows .

If the LWE conjecture is true then for every and and , the following two distributions are computationally indistinguishable:

* where is random matrix in , is random in and is random noise vector of magnitude .
* where is random matrix in and is random in .

Suppose that we had a decisional adversary that succeeds in distinguishing the two distributions above with bias . For example, suppose that outputs with probability on inputs from the first distribution, and outputs with probability on inputs from the second distribution.

We will show how we can use this to obtain a polynomial-time algorithm that on input noisy equations on and a value , will learn with high probability whether or not the first coordinate of equals . Clearly, we can repeat this for all the possible values of to learn the first coordinate exactly, and then continue in this way to learn all coordinates.

Our algorithm gets as input the pair where and we need to decide whether . Now consider the instance , where is a random vector in and the matrix is simply the matrix with first column equal to and all other columns equal to . If is random then is random as well. Now note that and hence if then we still have an input of the same form .

In contrast, we claim that if if then the distribution where and is identical to the uniform distribution over a random uniformly chosen matrix and a random and independent uniformly chosen vector . Indeed, we can write this distribution as where is chosen uniformly at random, and where is a random and independent vector. (Can you see why?) Since , this amounts to adding a random and independent vector to , which means that the distribution is uniform and independent.

Hence if we send the input to our the decision algorithm , then we would get with probability if and an output of with probability otherwise.

Now the crucial observation is that if our decision algorithm requires equations to succeed with bias , we can use equations (which is still polynomial) to invoke it times. This allows us to distinguish with probability between the case that outputs with probability and the case that it outputs with probability (this follows from the Chernoff bound; can you see why?). Hence by using polynomially more samples than the decision algorithm , we get a search algorithm that can actually recover .

## An LWE based encryption scheme

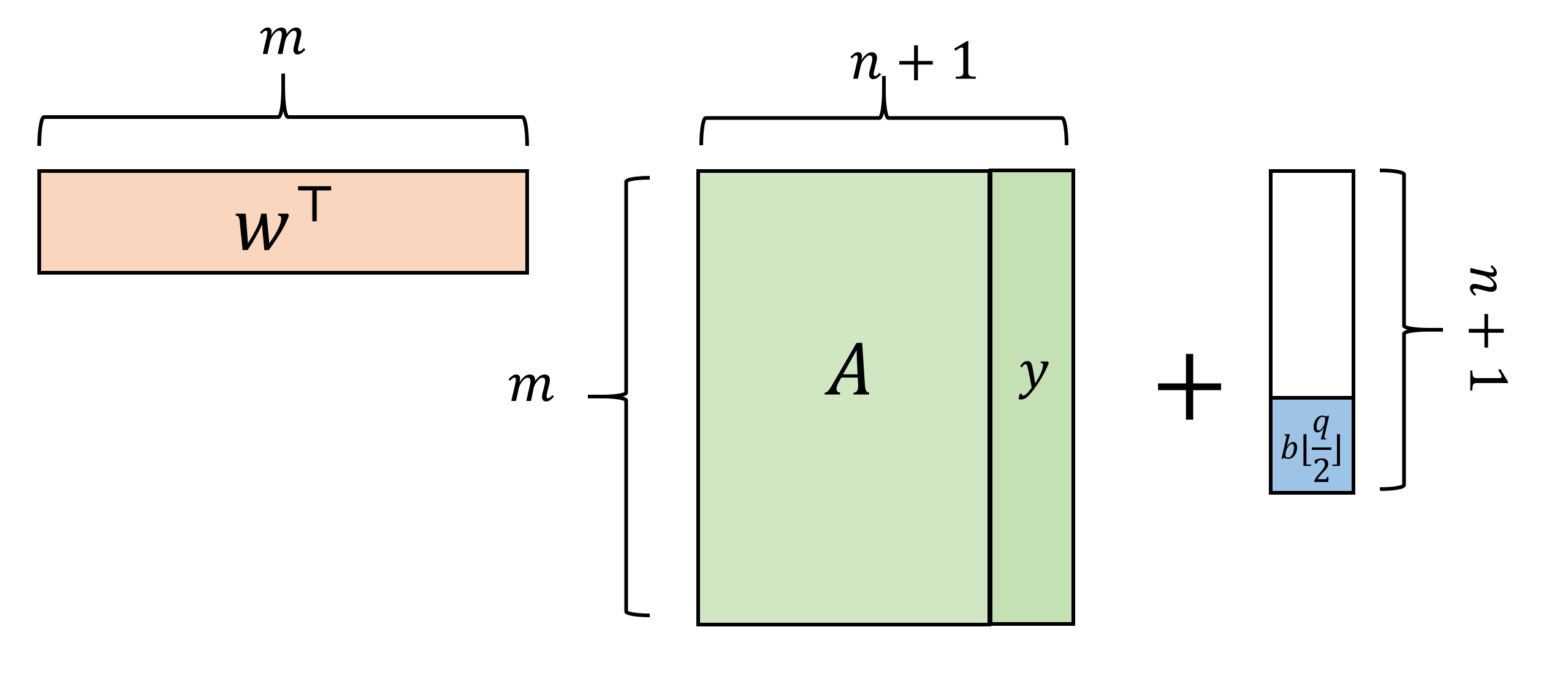
We can now show the secure variant of our original encryption scheme:

**LWE-based encryption LWEENC:** \* *Parameters:* Let and let be a prime such that LWE holds w.r.t. . We let .

* *Key generation:* Pick . The private key is and the public key is with with a -noise vector and a random matrix.
* *Encrypt:* To encrypt given the key , pick and output (all computations are done in ).
* *Decrypt:* To decrypt , output iff .

# 

The scheme LWEENC is also described in lweencdescfig with slightly different notation. I highly recommend you stop and verify you understand why the two descriptions are equivalent.



In the encryption scheme LWEENC, the public key is a matrix , where and is the secret key. To encrypt a bit we choose a random , and output . We decrypt to zero with key iff where the inner product is done modulo .

Unlike our typical schemes, here it is not immediately clear that this encryption is valid, in the sense that the decrypting an encryption of returns the value . But this is the case:

# 

With high probability, the decryption of the encryption of equals .

. Hence, if then . But since every coordinate of is either or , for our choice of parameters.[[7]](#footnote-50) So, we get that if and then which will be smaller than iff .

We now prove security of the LWE based encryption:

# 

If the LWE conjecture is true then LWEENC is CPA secure.

For a public key encryption scheme with messages that are just bits, CPA security means that an encryption of is indistinguishable from an encryption of , even given the public key. Thus LWEENCthm will follow from the following lemma:

Let be set as in LWEENC. Then, assuming the LWE conjecture, the following distributions are computationally indistinguishable:

* : The distribution over four-tuples of the form where is uniform in , is uniform in , is chosen with , , and is uniform in .
* : The distribution over four-tuples where all entries are uniform: is uniform in , is uniform in , is uniform in and is uniform in .

# 

You should stop here and verify that **(i)** You understand the statement of LWEENClem and **(ii)** you understand why this lemma implies LWEENCthm. The idea is that LWEENClem shows that the concatenation of the public key and encryption of is indistinguishable from something that is completely random. You can then use it to show that the concatenation of the public key and encryption of is indistinguishable from the same thing, and then finish using the hybrid argument.

We now prove LWEENClem, which will complete the proof of LWEENCthm.

Define to be the distribution as in the lemma’s statement (i.e., for some , chosen as above). Define to be the distribution where is chosen uniformly in .

We claim that is computationally indistinguishable from under the LWE conjecture. Indeed by LWEsearchtodecthm (search to decision reduction) this conjecture implies that the distribution over pairs with is indistinguishable from the distribution over pairs where is uniform. But if there was some polynomial-time algorithm distinguishing from then we can design a randomized polynomial-time algorithm distinguishing from with the same advantage by setting for random .

We will finish the proof by showing that the distribution is *statistically indistinguishable* (i.e., has negligible total variation distance) from . This follows from the following claim:

**CLAIM:** Suppose that . If is a random matrix in , then with probability at least over the choice of , the distribution over which is obtained by choosing at random in and outputting has at most statistical distance from the uniform distribution over .

Note that the randomness used for the distribution is only obtained by the choice of , and *not* by the choice of that is fixed. (This passes a basic “sanity check” since has random bits, while the uniform distribution over requires random bits, and hence at least has a “fighting chance” in being statistically close to it.) Another way to state the same claim is that the pair is statistically indistinguishable from the uniform distribution where is a vector chosen independently at random from .

The claim completes the proof of the theorem, since letting be the matrix and , we see that the distribution , as the form where is a uniformly random matrix and is sampled from (i.e., where is uniformly chosen in ). Hence this means that the statistical distance of from (where all elements are uniform) is . (Please make sure you understand this reasoning!)

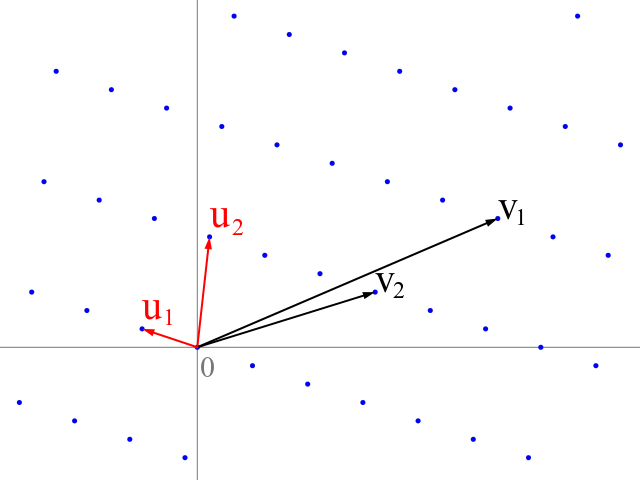
We will not do the whole proof of the claim (which uses the mod version of the [leftover hash lemma](https://goo.gl/KXpccP) which we mentioned before and is also “Wikipedia-able” - the **scribe writers** for this lecture should add those details thogh!) but the idea is simple. For every matrix over , define to be the map . This collection can be shown to be a “good” hash function collection in some specific technical sense, which in particular implies that for every distribution with much more than bits of min-entropy, with all but negligible probability over the choice of , is statistically indistinguishable from the uniform distribution. Now when we choose at random in , it is coming from a distribution with bits of entropy. If , then because the output of this function is so much smaller than , we expect it to be completely uniform, and this is what’s shown by the leftover hash lemma.

# 

The proof of LWEENCthm is quite subtle and requires some re-reading and thought. To read more about this, you can look at the survey of Oded Regev, [“On the Learning with Error Problem”](http://www.cims.nyu.edu/~regev/papers/lwesurvey.pdf) Sections 3 and 4.

## But what are lattices?

You can think of a lattice as a discrete version of a subspace. A lattice is simply a discrete subset of such that if and are integers then .[[8]](#footnote-58) A lattice is given by a basis which simply a matrix such that every vector is obtained as for some vector of integers . It can be shown that we can assume without loss of generality that is full dimensional and hence it’s an by invertible matrix. Note that given a basis we can generate vectors in , as well as test whether a vector is in by testing if is an integer vector. There can be many different bases for the same lattice, and some of them are easier to work with than others (see latticebasesfig).



A *lattice* is a discrete subspace that is closed under *integer* combinations. A *basis* for the lattice is a minimal set (typically ) such that every is an integer combination of . The same lattice can have different bases. In this figure the lattice is a set of points in , and the black vectors and the ref vectors are two alternative bases for it. Generally we consider the basis “better” since the vectors are shorter and it is less “skewed”.

Some classical computational questions on lattices are:

* *Shortest vector problem:* Given a basis for , find the nonzero vector with smallest norm in .
* *Closest vector problem:* Given a basis for and a vector that is *not* in , find the closest vector to in .
* *Bounded distance decoding:* Given a basis for and a vector of the form where is in , and is a particularly short “error” vector (so in particular no other vector in the lattice is within distance to ), recover . Note that this is a special case of the closest vector problem.

In particular, if is a linear subspace of , we can think of it also as a lattice of where we simply say that that a vector is in if all of ’s coordinates are integers and if we let then . The learning with error task of recovering from can then be thought of as an instance of the bounded distance decoding problem for .

A natural algorithm to try to solve the *closest vector* and *bounded distance decoding* problems is that to take the vector , express it in the basis by computing , and then round all the coordinates of to obtain an integer vector and let be a vector in the lattice. If we have an extremely good basis for the lattice then may indeed be the closest vector in the lattice, but in other more “skewed” bases it can be extremely far from it.

## Ring based lattices

One of the biggest issues with lattice based cryptosystem is the key size. In particular, the scheme above uses an matrix where each entry takes bits to describe. (It also encrypts a single bit using a whole vector, but more efficient “multi-bit” variants are known.) Schemes using *ideal lattices* are an attempt to get more practical variants. These have very similar structure except that the matrix chosen is not completely random but rather can be described by a single vector. One common variant is the following: we fix some polynomial over with degree and then treat vectors in as the coefficients of degree polynomials and always work modulo this polynomial . (By this I mean that for every polynomial of degree at least we write as where is the polynomial above, is some polynomial and is the “remainder” polynomial of degree ; then .) Now for every fixed polynomial , the operation which is defined as is a linear operation mapping polynomials of degree at most to polynomials of degree at most , or put another way, it is a linear map over . However, the map can be described using the coefficients of as opposed to the description of a matrix. It also turns out that by using the Fast Fourier Transform we can evaluate this operation in roughly steps as opposed to . The ideal lattice based cryptosystem use matrices of this form to save on key size and computation time. It is still unclear if this structure can be used for attacks; recent papers attacking principal ideal lattices have shown that one needs to be careful about this.

One ideal-lattice based system is the [“New Hope” cryptosystem](https://newhopecrypto.org/) (see also [paper](https://eprint.iacr.org/2015/1092.pdf)) that has been experimented with by Google. People have also made highly optimized general (non ideal) lattice based constructions, see in particular the [“Frodo” system](https://frodokem.org/) (paper [here](https://eprint.iacr.org/2016/659), can you guess what’s behind the name?). Both New Hope and Frodo have been submitted to the [NIST competition](https://csrc.nist.gov/Projects/Post-Quantum-Cryptography) to select a “post quantum” public key encryption standard.

1. While this won’t be of interest for us in this chapter, one can also define finite fields whose size is a *prime power* of the form where is a prime and is an integer; this is sometimes useful and in particular fields of size are sometimes used in practice. In such fields we usually think of the elements as *vector* with addition done component-wise but multiplication is not defined component-wise (since otherwise a vector with a single coordinate zero would not have an inverse) but in a different way, via interpreting these vectors as coefficients of a degree polynomial. [↑](#footnote-ref-28)
2. There is a much more general notion of inner product typically defined, and in particular over fields such as the complex numbers we would define the inner product as where for , denotes the *complex conjugate* of . However, we stick to the simple case above for this chapter. [↑](#footnote-ref-29)
3. Despite the name, [Gaussian elimination](https://goo.gl/3HNb5U) has been known to Chinese mathematicians since 150BC or so, and was popularized in the west through the 1670 notes of Isaac Newton. [↑](#footnote-ref-32)
4. Over , we can think of also as the number , and so on. Thus if , we define to be the minimum of and . This ensures the absolute value satisfies the natural property of . [↑](#footnote-ref-37)
5. One can think of as chosen by simply letting every coordinate be chosen at random in . For technical reasons, we sometimes consider other distributions and in particular the *discrete Gaussian* distribution which is obtained by letting every coordinate of be an independent Gaussian random variable with standard deviation , conditioned on it being an integer. (A closely related distribution is obtained by picking such a Gaussian random variable and then rounding it to the nearest integer.) [↑](#footnote-ref-38)
6. People sometimes also consider variants where both and can be as large as exponential. [↑](#footnote-ref-39)
7. In fact, due to the fact that the *signs* of the error vector’s entries are different, we expect the errors to have significant cancellations and hence we would expect to only be roughly of magnitude , but this is not crucial for our discussions. [↑](#footnote-ref-50)
8. By discrete we mean that points in are isolated. One formal way to define it is that there is some such that every distinct are of distance at least from one another. [↑](#footnote-ref-58)