Public-key encryption from LWE & Implementing lattice-based cryptosystems

Notes by Henry Corrigan-Gibbs and Yael Kalai

MIT - 6.5610

Lecture 8 (February 26, 2024)

> ***Warning:*** This document is a rough draft, so it may contain bugs. Please feel free to email me with corrections.

Outline

- Reminder: Learning-with-Errors assumption
- Reminder: Definition of public-key encryption
- Regev's public-key encryption scheme
- Implementing lattice-based cryptosystems

But first: a knock-knock joke.

Learning-with-errors (LWE) assumption

We saw Regev's LWE assumption [2] a few lectures ago. To refresh your memory, the LWE assumption, loosely speaking, asserts that it is hard to solve noisy linear equations modulo integers of a certain type.

I tend to use $\overset{c}{\approx}$ to denote computational indistinguishability of two ensembles of distributions. I say *distribution ensembles* because to talk about computational indistinguishability of distributions, we must actually work with infinite families of distributions parameterized by the security parameter λ. So when we write $\mathcal{D}_0 \overset{c}{\approx} \mathcal{D}_1$, we really mean that the ensembles $\mathcal{D}_0 = \{\mathcal{D}_{0,\lambda}\}_{\lambda=1}^{\infty}$ and $\mathcal{D}_1 = \{\mathcal{D}_{1,\lambda}\}_{\lambda=1}^{\infty}$ are computationally indistinguishable.

You can see why we prefer the shorthand notation.

The LWE assumption is a family of assumptions associated with integral parameters $n = n(\lambda)$, $m = m(\lambda)$, $q = q(\lambda)$ and an error distribution $\chi = \chi(\lambda)$ over $\mathbb{Z}_q$.

Definition 1. The $\text{LWE}_{n,m,q,\chi}$ assumption asserts that

$$(\mathbf{A}, \mathbf{As} + \mathbf{e}) \overset{c}{\approx} (\mathbf{A}, \mathbf{U})$$

where $\mathbf{A} \overset{\text{R}}{\leftarrow} \mathbb{Z}_q^{m \times n}$ $\mathbf{s} \overset{\text{R}}{\leftarrow} \mathbb{Z}_q^n$, $\mathbf{e} \leftarrow \chi^m$.

For the LWE assumption to be useful for building cryptosystems, we need $m \gg n \log q$. In theory, we typically let the error distribution χ be the "discrete Gaussian" distribution, where we restrict the outputs to be in $[-B, B]$, where $-B = q - B$.

When χ is the discrete-Gaussian distribution, appropriately parameterized, we have the special consequence that if there exists a p.p.t. adversary that breaks $\text{LWE}_{n,m,q,\chi}$ then one can use this adversary to break worse-case lattice problems (such as finding a good approximation to the shortest vector in a lattice). We will not elaborate on this, and for those who are interested, Vinod Vaikuntanathan has fantastic lecture notes on this topic here.

As we will see, in practice, it is convenient to choose χ as some other distribution. One surprise about LWE is that the assumption is extremely robust: many many variants of LWE problem are essentially as hard as the standard LWE problem.

For us, all we will use is that χ takes values in $[-B, B]$ where B is a small error parameter (significantly smaller than q).

Review: Symmetric Encryption Scheme from LWE

We review Regev's symmetric-key encryption scheme, since we will need it for the public-key version. I am going to write the decryption algorithm here slightly differently than we did last week. This alternative formulation ends up being more convenient when we use Regev encryption as a building block in more powerful cryptosystems.

Construction. The message space is $\{0,1\}$. The secret key is $\mathbf{s} \xleftarrow{R} \mathbb{Z}_q^n$. To encrypt a bit $b \in \{0,1\}$:

$$\mathsf{Enc}(\mathbf{s}, b) = (\mathbf{a}, \mathbf{a}^\mathsf{T}\mathbf{s} + e + b \cdot \frac{q}{2}) \in \mathbb{Z}_q^{n+1},$$

where $\mathbf{a} \xleftarrow{R} \mathbb{Z}_q^n$ and $\mathbf{e} \xleftarrow{R} \chi$. To decrypt a ciphertext $\mathbf{c} \in \mathbb{Z}_q^{n+1}$:

For simplicity, I am assuming here that q is even. If not, replace $q/2$ everywhere with $\lfloor q/2 \rfloor$.

$$\mathsf{Dec}(\mathbf{s}, \mathbf{c}) : \text{ output } 0 \text{ iff } |\mathbf{c}^\mathsf{T} \cdot (-\mathbf{s}|1)| \le \frac{q}{4}.$$

Correctness holds because the expression inside the absolute-value bars is:

The notation $(-\mathbf{s}|1)$ just indicates that we are concatenating the element "1" onto the end of the vector $\mathbf{s}$.

$$\begin{aligned}(\mathbf{a}, \mathbf{a}^\mathsf{T}\mathbf{s} + e + b(\frac{q}{2}))^\mathsf{T} \cdot (-\mathbf{s}|1) &= -\mathbf{a}^\mathsf{T}\mathbf{s} + \mathbf{a}^\mathsf{T}\mathbf{s} + e + b(\frac{q}{2}) \\ &= e + b(\frac{q}{2}).\end{aligned}$$

Since $e \le B \ll q/4$, the decryption algorithm will always return the bit b.

Linear homomorphism. Recall from the first PIR lecture that this encryption scheme is *linearly homomorphic*. Namely, for every $b_1, b_2, \dots, b_m \in \{0,1\}$ it holds that

$$\mathsf{Dec}(\mathbf{s}, \mathsf{Enc}(\mathbf{s}, b_1) + \mathsf{Enc}(\mathbf{s}, b_2) + \cdots + \mathsf{Enc}(\mathbf{s}, b_m)) = b_1 \oplus b_2 \oplus \cdots b_m.$$

The error grows a little with each homomorphic addition. Namely, if the original error of each ciphertext had B-bounded error, the ciphertext after m additions has (Bm)-bounded error. As long as $Bm \ll q/4$, we maintain correctness.

Review: CPA-secure public-key encryption

Last time we defined the notion of a CPA-secure symmetric key encryption and showed how to construct it from any PRF family.

There is a stronger definition of security, which is the "gold standard" for security of encryption schemes. The stronger notion is *security against chosen ciphertext attacks (or CCA security)*. In the CCA-security game, the attacker may ask the challenger for decryptions of ciphertexts of its choice, in addition to seeing encryptions of messages of its choice.

Definition 2. A public-key encryption scheme over a message space $\mathcal{M} = \{\mathcal{M}_\lambda\}_{\lambda \in \mathbb{N}}$ consists of three efficient (p.p.t.) algorithms $(\mathsf{Gen}, \mathsf{Enc}, \mathsf{Dec})$, where:

- Gen takes as input the security parameter 1^λ (in unary) and outputs a key pair $(\mathsf{pk}, \mathsf{sk})$.
- Enc takes as input a public key pk and a message $m \in \mathcal{M}_\lambda$ and outputs a ciphertext $\mathsf{Enc}(\mathsf{pk}, m)$.
- Dec takes as input a secret key sk and a ciphertext ct and outputs a message m. Typically, Dec is deterministic.

The correctness guarantee is that for every $\lambda \in \mathbb{N}$, for every $m \in \mathcal{M}_\lambda$ and for every $(\mathsf{sk}, \mathsf{pk})$ that $\mathsf{Gen}(1^\lambda)$ may output,

$$\Pr[\mathsf{Dec}(\mathsf{sk}, \mathsf{Enc}(\mathsf{pk}, m)) = m] = 1,$$

where the probability is over $(\mathsf{pk}, \mathsf{sk}) \leftarrow \mathsf{Gen}(1^\lambda)$ and over the randomness used by Enc.

Sometimes we relax the definition of perfect completeness and allow a negligible probability of error over $(\mathsf{pk}, \mathsf{sk}) \leftarrow \mathsf{Gen}(1^\lambda)$ and over the randomness used by Enc.

Definition 3 (Semantic security for public-key encryption – "Weak" security)**.** A public-key encryption scheme $(\mathsf{Gen}, \mathsf{Enc}, \mathsf{Dec})$ is said to be *semantically secure* (i.e., CPA secure, i.e., secure against chosen-plaintext attack) if for every $\lambda \in \mathbb{N}$ and every $m_0, m_1 \in \mathcal{M}_\lambda$ it holds that

$$(\mathsf{pk}, \mathsf{Enc}(\mathsf{pk}, m_0)) \overset{c}{\approx} (\mathsf{pk}, \mathsf{Enc}(\mathsf{pk}, m_1)),$$

where the distributions are over $(\mathsf{pk}, \mathsf{sk}) \leftarrow \mathsf{Gen}(1^\lambda)$ and over the randomness of Enc.

Semantically security is the same as CPA security, and different names are used for the symmetric setting and the public-key setting, though sometimes CPA security is used for the public-key setting as well.

Are we assuming here that the adversary is given only a single ciphertext? No! Notice that in the public key setting ciphertexts can be computed efficiently given pk, and pk is given in both distributions above.

As we mentioned above, in practice we demand that our encryption schemes satisfy the stronger notion of security against adaptive chosen-ciphertext attacks ("CCA security"). There are generic ways to "lift" a CPA-secure encryption system to a CCA-secure encryption system, as long as you are fine using a cryptographic hash function that you model as a random oracle.

As far as I know, the question of whether it is possible to lift *any* CPA-secure encryption scheme into a CCA-secure *without* using random oracles is still open.

Public-Key Encryption from LWE

The first public-key encryption schemes that most students learn are ElGamal and RSA encryption. Computing discrete logs is enough to break ElGamal cryptosystem. Factoring integers is enough to break the RSA cryptosystem. A large-enough quantum computer—but still polynomially large—can compute discrete logs and can factor

RSA was developed here at MIT by Rivest, Shamir and Adleman. Ron Rivest will give a guest lecture in our class later this semester!

integers. So a quantum computer can break both of these encryption schemes in polynomial time.

We will focus on constructions that are plausibly post-quantum secure. Typically, breaking such constructions requires solving some variant of the learning-with-errors (LWE) problem, which we saw earlier in the course. Since LWE is a very different type of computational problem than discrete log or factoring, and since we have no good algorithms for solving LWE on a quantum computer, we expect these construction to withstand quantum attacks.

- $\mathsf{Gen}(1^\lambda)$:
  1. Let $n = \lambda$, $q = \text{poly}(\lambda)$ and $m = \Theta(n \cdot \log q)$. Let χ be a B-bounded distribution such that $B \cdot m \ll q/4$.
  2. Generate $\mathbf{s} \xleftarrow{R} \mathbb{Z}_q^n$, $\mathbf{A} \xleftarrow{R} \mathbb{Z}_q^{m\times n}$, and $\mathbf{e} \leftarrow \chi^m$.
  3. Output $\mathsf{pk} = (\mathbf{A}, \mathbf{As} + \mathbf{e})$ and $\mathsf{sk} = \mathbf{s}$.

  A B-bounded distribution is just one that takes on values in the range $\{-B, \ldots, B\} \in \mathbb{Z}_q$. As usual, we slightly abuse notation and write $-x \in \mathbb{Z}_q$ to mean $q - x \in \mathbb{Z}_q$.

  Denote by $\mathbf{B} \in \mathbb{Z}^{m\times(n+1)}$ the matrix consisting of $\mathbf{A}$ with the vector $\mathbf{As} + \mathbf{e}$ appended as an extra column. Thus, we can think of $\mathsf{pk} = \mathbf{B}$.

  Intuition. What is going on here? You can think of the public key as containing m symmetric-key encryptions of the bit "0"—one per row—all encrypted under secret key $\mathbf{s}$.

  Note that we can decrypt all of these m ciphertexts at once by multiplying by $(-\mathbf{s}|1)$ on the right, just like running m copies of the symmetric-key decryption routine:

$$\mathbf{B} \cdot (-\mathbf{s}|1) = (\mathbf{A}|\mathbf{As} + \mathbf{e}) \cdot (\mathbf{s}|1) = (\mathbf{As} + \mathbf{e}) - \mathbf{As} = \mathbf{e}.$$

- $\mathsf{Enc}(\mathsf{pk}, b)$ chooses a random $\mathbf{r} \xleftarrow{R} \{0,1\}^m \subseteq \mathbb{Z}_q^m$ and outputs

$$\mathbf{r}^\mathsf{T}\mathbf{B} + b \cdot (0, \ldots, 0, \frac{q}{2}).$$

  Intuition. Since we think of the public key $\mathbf{B}$ as containing m symmetric-key encryptions of "0," the public-key encryption algorithm here is just taking a random subset of these m encryptions and "adding them up" using the homomorphic property of the encryption scheme. This gives us a new symmetric-key encryption of "0." The public-key encryption algorithm then flips the sign of this encrypted message if the bit $b = 1$ by adding $q/2$ to the encrypted value.

- $\mathsf{Dec}(\mathsf{sk}, \mathbf{c})$ outputs "0" iff $|\mathbf{c}^\mathsf{T}(-\mathbf{s}|1)| \leq q/4$. Decryption here works exactly as in the secret-key setting!

Correctness. The correctness of this public-key scheme follows directly from the correctness of the linearly homomorphic secret-key scheme. The output ciphertext here is the sum of at most m encryptions of "0," plus $q/2$ if the message bit is "1."

If the error in each original ciphertext is B-bounded, then the output of the Enc algorithm will be a ciphertext with (Bm)-bounded error. As long as $Bm \ll q/4$, decryption will work.

Is this scheme secure?

Security follows from the LWE assumption on parameters (n, m, q, χ), but proving this is requires a bit more work than in the symmetric key setting.

To prove security we will argue that for every $b \in \{0,1\}$, the following ensembles of probability distributions are computationally indistinguishable,

Again the notation $\mathcal{D}_0 \overset{c}{\approx} \mathcal{D}_1$ denotes that two ensembles of probability distributions are computationally indistinguishable.

$$(\mathsf{pk}, \mathsf{Enc}(\mathsf{pk}, b)) \overset{c}{\approx} (\mathsf{pk}, \mathbf{u}), \tag{1}$$

where $(\mathsf{sk}, \mathsf{pk}) \leftarrow \mathsf{Gen}(1^\lambda)$ and $\mathbf{u}$ is uniform over $\mathbb{Z}_q^{n+1}$. Proving Eq. (1) is enough to prove CPA security, since we can apply Eq. (1) twice to show that for all messages $m_0, m_1 \in \mathcal{M}$:

$$(\mathsf{pk}, \mathsf{Enc}(\mathsf{pk}, m_0)) \overset{c}{\approx} (\mathsf{pk}, \mathbf{u}) \overset{c}{\approx} (\mathsf{pk}, \mathsf{Enc}(\mathsf{pk}, m_1))$$

If we let $\mathbf{B}$ denote the public key, and we let $\mathbf{b} = (0, 0, \ldots, 0, q/2) \in \mathbb{Z}_q^{n+1}$, we have

Here, we use $\equiv$ to denote that two distribution ensembles are identical.

$$(\mathsf{pk}, \mathsf{Enc}(\mathsf{pk}, b)) \equiv (\mathbf{B}, \mathbf{r}^\mathsf{T}\mathbf{B} + \mathbf{b}). \tag{2}$$

The security argument works in two steps:

1. *Replace the public key with a uniform random matrix.* First, we appeal to the LWE assumption to argue that the public key pk is computationally indistinguishable from being uniform in $\mathbb{Z}_q^{m \times (n+1)}$. So, under the LWE assumption, we can replace the matrix $\mathbf{B}$ with a uniform random matrix $\mathbf{U} \overset{\mathrm{R}}{\leftarrow} \mathbb{Z}_q^{m \times (n+1)}$ and no efficient algorithm will be able to distinguish:

$$(\mathbf{B}, \mathbf{r}^\mathsf{T}\mathbf{B} + \mathbf{b}) \overset{c}{\approx} (\mathbf{U}, \mathbf{r}^\mathsf{T}\mathbf{U} + \mathbf{b}). \tag{3}$$

2. *Argue information-theoretically that the ciphertext is statistically close to a uniform random vector.* Our last task is to show that:

$$(\mathbf{U}, \mathbf{r}^\mathsf{T}\mathbf{U} + \mathbf{b}) \overset{s}{\approx} (\mathbf{U}, \mathbf{v}), \tag{4}$$

for $\mathbf{U} \overset{\mathrm{R}}{\leftarrow} \mathbb{Z}_q^{m \times (n+1)}$ and $\mathbf{v} \overset{\mathrm{R}}{\leftarrow} \mathbb{Z}_q^m$.

To prove Eq. (4), we appeal to a special case of the "Leftover Hash Lemma," which is used all over the place in cryptography. Look at

The $\overset{s}{\approx}$ notation indicates that two distribution ensembles are *statistically close*. If two distribution ensembles are statistically close, then no efficient algorithm can distinguish them. This is an unconditional statement—it requires no computational assumptions. Distributions that are statistically close are aaaaalmost identical, but not quite.

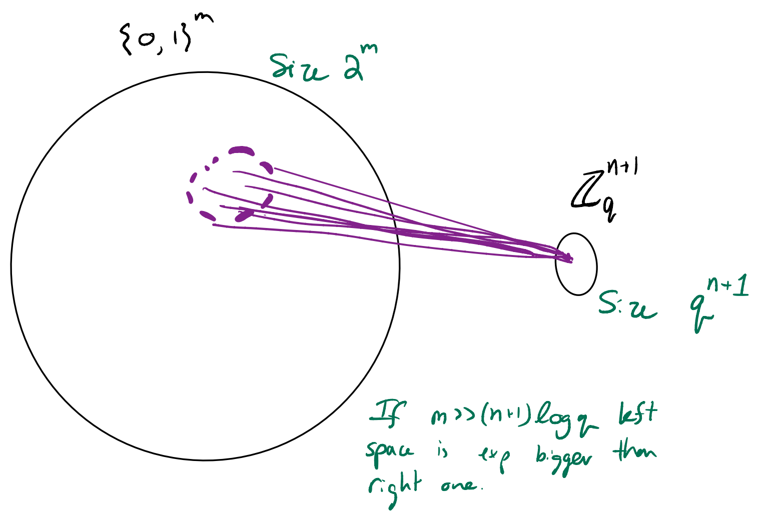
$\{0,1\}^m$
Size 2^m
$\mathbb{Z}_q^{n+1}$
Size q^{n+1}
If $m \gg (n+1)\log q$ left space is exp bigger than right one.

Figure 1: My attempt to explain the Leftover Hash Lemma with a picture.

the left side of Eq. (4). The term $\mathbf{r}^\top \mathbf{B}$ consists of a vector of $(n+1)$ elements of $\mathbb{Z}_q$.

We will not prove the lemma here. One way to get some intuition: If I give you $(\mathbf{U}, \boldsymbol{\alpha} = \mathbf{r}^\top \mathbf{U})$, where $\mathbf{U} \xleftarrow{R} \mathbb{Z}_q^{m\times(n+1)}$ and $\mathbf{r} \xleftarrow{R} \mathbb{Z}_q^m$, I'm really just giving you a system of $n+1$ linear equations, defined by $\mathbf{r}^\top \mathbf{U} = \boldsymbol{\alpha}$ with $m \gg n+1$ unknowns (the values of $\mathbf{r}$) modulo q. As long as $m \gg (n+1)\log q$, for each possible value of $\boldsymbol{\alpha}$, there are an exponential number of possible solutions $\mathbf{r}$, all about equally likely. So the value $\mathbf{r}^\top \mathbf{U}$ cannot reveal much about $\mathbf{r}$ and in fact is statistically close to uniform random, even given $\mathbf{U}$. Then the value $\mathbf{r}^\top \mathbf{U}$ is essentially a one-time-pad encryption of $\mathbf{b}$ and that gives Eq. (4).

See Regev's 2005 paper for a clean statement of the Leftover Hash Lemma as needed here.

The last step is to put Eq. (2), Eq. (3) and Eq. (4) together, which gives the statement that we wanted to prove, Eq. (1).

Implementing lattice-based cryptography

Let us discuss a few implementation considerations with Regev's public-key encryption system. Practical lattice-based cryptosystems at their core often just use a variant Regev's encryption system, though with many performance optimizations and tweaks. (Some of these require stronger cryptographic assumptions than plain LWE.)

Choosing parameters

In RSA, there is essentially only one parameter to pick: the length of the modulus. In lattice-based cryptosystems, we have many more:

- n – the secret dimension,
- m – the number of LWE samples revealed,
- q – the modulus, and
- χ – the B-bounded error distribution.

The particular application imposes some restrictions on the relationship of these parameters to each other as well. For example, for correctness in Regev public-key encryption, we need $m > 2n \log q$ and also that $Bm < q/4$. Other applications have even more constraints. For public-key encryption, we can fix $m = 2n \log q$, so we just need to worry about (n, q, χ).

Often, we start by picking the modulus q to be something convenient. The choices $q = 2^{16}$ or $q = 2^{32}$ are nice since mod-q operations with these moduli are just native machine operations.

Now, to set n and χ, we have to understand, for a given choice of (q, m, n, χ), what the running time of the best known LWE algorithm is. We want this running time to be greater than 2^{128} or so. Unfortunately, there is no clean closed-form expression for the running time of the best attack for a given set of parameters. Instead, to choose the parameters, we typically resort to the "lattice estimator," a script that approximates the running time of the best lattice attack for a given parameter set.

In contrast, for discrete log on certain elliptic-curve groups of order q, the best attack runs in time roughly $2^{q/2}$. This makes it easy to pick the group order. (Picking the parameters of the curve, in contrast, is very tricky business.)

Increasing the secret dimension n makes LWE harder. For a given secret dimension n, the "modulus-to-noise" ratio $q/\text{stddev}(\chi)$ roughly determines how easy or hard LWE is:

- When this ratio is close to one, χ adds a gigantic amount of noise and LWE is relatively hard. In this case, we can set the secret dimension n to be relatively small.
- When this ratio is very large, χ adds very little noise relative to the modulus. In this case, we must set the secret dimension n to be relatively large—otherwise LWE is not hard enough.

Packing more bits into each ciphertext

The basic Regev scheme we have seen encrypts a single bit into $n + 1$ $\mathbb{Z}_q$ elements. Since $n \approx 2^{10}$ and $q \approx 2^{16}$, this is almost a 64,000$\times$ overhead in bitlength. We can squeeze a little more juice out of Regev encryption by jamming a few more bits into each ciphertext. In particular we can choose a plaintext modulus $p > 2$ and encrypt $\mathbb{Z}_p$

elements.

To do so, we encode $b \in \mathbb{Z}_p$ as $b(q/p)$. Then when we decrypt, we round the answer to the nearest multiple of q/p. As long as the noise has magnitude less than $q/(2p)$, we will always decrypt correctly.

Even so, Regev still imposes a large overhead—at least $1000\times$, typically.

Large keys

The public key in Regev's scheme is a matrix of dimension roughly $m \times n$, where n is the secret dimension and $m \geq 2n \log q$. Taking $q = 2^{16}$ is a common choice. Then if $\text{stdev}(\chi) = 16$, we need roughly $n = 2^9 = 512$ for 128-bit security and $m = 2n \log q = 2^{10} \cdot 2^4 = 2^{14}$.

The public-key matrix is then roughly $mn = 2^{23}$ $\mathbb{Z}_q$ elements, each is two bytes each. So the public key is around $2^{24} = 16$ MB! Compare this with the 32-byte public keys that standard discrete-log-based cryptosystems use!

The standard trick here is to use a cryptographic hash function $H(\cdot,\cdot)$ in counter mode to generate the LWE matrix $\mathbf{A}$ from a small λ-bit seed, rather than choosing $\mathbf{A}$ truly at random from $\mathbb{Z}_q^{m\times n}$. That is, on seed σ, we would generate the entries of $\mathbf{A}$ as $H(\sigma, 0), H(\sigma, 1), H(\sigma, 2), \ldots$ and so on.

When using this Regev variant, the public key consists of a (1) a λ-bit key for a pseudorandom function and (2) a vector in $\mathbb{Z}_q^m$. The total size here is now more like $2^{16} = 64$ KB.

The catch is that, to prove security, we have to model the function H as a random oracle. But that is typically good enough in practice.

Ring LWE: Faster computation and *less communication*

The last trick we will discuss solves two problems at once: it reduces the encryption and decryption time by a factor of roughly $n \approx 1000$. It also reduces the ciphertext blowup by a factor of roughly $n \approx 1000$. Does it sound too good to be true? The major caveat is that we have to make a stronger assumption than the LWE assumption to get this performance boost: the Ring LWE assumption, of Lyubashevsky, Peikert, and Regev [1].

Recall that the LWE assumption on parameters (n, m, q, χ) says that:

$$(\mathbf{A}, \mathbf{A}\mathbf{s} + \mathbf{e}) \overset{c}{\approx} (\mathbf{A}, \mathbf{u}) \qquad \text{for} \qquad \begin{matrix} \mathbf{A} \overset{\text{R}}{\leftarrow} \mathbb{Z}_q^{m\times n} \\ \mathbf{s} \overset{\text{R}}{\leftarrow} \mathbb{Z}_q^{n} \\ \mathbf{e} \overset{\text{R}}{\leftarrow} \chi^{m} \\ \mathbf{u} \overset{\text{R}}{\leftarrow} \mathbb{Z}_q^{m} \end{matrix} .$$

Observe that $\mathbf{A}$ is a uniform random, and totally unstructured, matrix. For my simple brain, the easiest way to think about ring LWE is that we replace the unstructured matrix $\mathbf{A}$ with a *structured* one. In particular, for a vector $\mathbf{a} = (a_1, a_2, \dots, a_n) \in \mathbb{Z}_q^n$, define the $n \times n$ "negacyclic" matrix $\mathsf{Nega}(\mathbf{a})$ as:

The more principled way to think about ring LWE is that we are replacing the $\mathbb{Z}_q$ elements in LWE with polynomials with coefficients in $\mathbb{Z}_q$ taken modulo some polynomial $f(x)$. We then have a ring of polynomials $\mathcal{R} = \mathbb{Z}_q[x]/f(x)$. We can now define the LWE problem over $\mathcal{R}$ instead of over $\mathbb{Z}_q$. The assumption is that $a \cdot s + e \overset{c}{\approx} u \in \mathcal{R}$, where $a, s, u \xleftarrow{R} \mathcal{R}$, and e is a polynomial in $\mathcal{R}$ with "small" coefficients.

$$\mathsf{Nega}(\mathbf{a}) := \begin{pmatrix} a_1 & a_2 & a_3 & \cdots & a_n \\ -a_n & a_1 & a_2 & \cdots & a_{n-1} \\ -a_{n-1} & -a_n & a_1 & \ddots & a_{n-2} \\ \vdots & \vdots & \vdots & \cdots & \vdots \\ -a_2 & -a_3 & -a_4 & \cdots & a_1 \end{pmatrix} \in \mathbb{Z}_q^{n \times n}.$$

Two points about $\mathsf{Nega}(\mathbf{a})$ are:

1. this matrix has dimension $n \times n$ but only requires n $\mathbb{Z}_q$ elements to represent (i.e., the vector $\mathbf{a}$), and
2. there is a near-linear-time matrix-vector multiplication algorithm for computing the product $\mathsf{Nega}(\mathbf{a}) \cdot \mathbf{v}$, provided that the modulus q has some special structure.

This algorithm uses the Fast Fourier Transform and requires only $O(n \log n)$ operations in $\mathbb{Z}_q$.

Now we can construct the $\mathbf{A}_{\mathsf{Nega}}$ matrix for Regev's public-key encryption scheme as:

$$\mathbf{A}_{\mathsf{Nega}} = \begin{pmatrix} \mathsf{Nega}(\mathbf{a}_1) \\ \mathsf{Nega}(\mathbf{a}_2) \\ \cdots \\ \mathsf{Nega}(\mathbf{a}_\ell) \end{pmatrix} \in \mathbb{Z}_q^{\ell n \times n}.$$

Then we have $m = \ell n$.

The most expensive step in computing a Regev encryption is the product $\mathbf{r}^\mathsf{T}\mathbf{A}$, which we can do in time now $O(m \log n) \ll O(mn)$. The n-to-$\log n$ reduction gives a $\frac{n}{\log n}\times$ speedup, or maybe $100\times$ for realistic parameter settings.

But is it secure? The question now is whether the ring LWE problem (or the LWE problem with a structured $\mathbf{A}$ matrix) is still hard. Ring LWE is not harder than LWE, but it could potentially be much easier. For instance, we know that if you choose $\mathbf{A}$ to be a *cyclic* matrix instead of a negacyclic one, ring LWE is easy. What exact sort of structure you impose on the matrix has a major impact on the hardness of the resulting LWE problem.

At the same time, we know of no better attacks on ring LWE than just attacking LWE itself. Given the enormous performance benefits of using ring LWE, many lattice-based cryptosystems rely on it in spite of the potential of better-than-LWE attacks in the ring setting.

References

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

[2] Oded Regev. On lattices, learning with errors, random linear codes, and cryptography. In Harold N. Gabow and Ronald Fagin, editors, *Proceedings of the 37th Annual ACM Symposium on Theory of Computing, Baltimore, MD, USA, May 22-24, 2005*, pages 84–93. ACM, 2005.