CSE 206A: Lattice Algorithms and Applications Fall 2019

Somewhat Homomorphic Encryption (10/31, 11/5, 11/7)

Instructor: *Daniele Micciancio* Scribe: *Rex Lei and Aaron Geelon So*

In this lecture, we will use a variant of the Learning with Errors (LWE) problem to build a leveled homomorphic encryption scheme. By *homomorphic*, we mean that it is possible to perform computation on encrypted messages without ever decrypting them in the process. For example, if m_1 and m_2 are two messages, then an additively homomorphic encryption scheme provides an operation $\oplus$ that acts on ciphertexts, so that decrypting

$$\texttt{Enc}(m_1) \oplus \texttt{Enc}(m_2)$$

yields the sum $m_1 + m_2$.

Homomorphic encryption is motivated by applications where data must be kept private, but where the party that owns the data cannot run the computation on the data directly. One use case is the computation-as-a-service model, where computation may be offloaded to the cloud—users encrypt their data and the cloud homomorphically performs the computation on the encrypted data. As another use case, hospitals could benefit from performing statistical analyses over the pooled patient data; however, they are also legally required to protect patient privacy. Homomorphic encryption could enable collaboration while preserving patient privacy.

To this end, we describe a series of homomorphic encryption schemes with increasing complexity and flexibility. Our goal is (i) a symmetric encryption scheme, (ii) an additively homomorphic encryption scheme, and (iii) a multiplicatively homomorphic encryption scheme. These encryption schemes are not yet truly additive or multiplicative because the number of addition and multiplication operations that may be performed while preserving correctness is bounded. But, by making use of *bootstrapping*, it is possible to build a fully homomorphic encryption scheme, which has the proper closure under encrypted addition and multiplication. Let us first review **LWE** and one variant.

1 LWE and RingLWE

Let X be a collection of "short vectors" in $\mathbb{Z}^m$. For example, let $X = \{x \in \mathbb{Z}^m : \|x\|_2 \leq \beta\}$ for some parameter β, or let $X = \{0,1\}^m$ (when x is a binary vector, then $\|x\|_2 \leq \sqrt{n}$).

Definition 1 *The* Learning with Errors *problem* **LWE***, on an input matrix* $\mathbf{A} \in \mathbb{Z}_q^{n \times m}$ *and a vector* $\mathbf{b}$*, asks to find a vector* $\mathbf{s} \in \mathbb{Z}_q^n$ *such that* $\mathbf{b} - \mathbf{A}^t\mathbf{s} \in X$*.*

We can use the **LWE** problem to generate a public and secret key: choose $\mathbf{A} \in \mathbb{Z}_q^{n \times m}$, $\mathbf{s} \in \mathbb{Z}_q^n$ and $\mathbf{e} \in \mathbb{Z}_q^m$ at random. Compute $\mathbf{b} \in \mathbb{Z}_q^m$ as follows:

$$\mathbf{b}^t = \mathbf{s}^t\mathbf{A} + \mathbf{e}^t.$$

The pair $(\mathbf{A}, \mathbf{b})$ forms the public key and $\mathbf{s}$ is the secret key. For example, $\mathbf{s}$ could be chosen uniform at random from $\mathbb{Z}_q^n$ or from a discrete Gaussian $\mathcal{D}_{\mathbb{Z},\sigma}^n$. Likewise, to sample a short vector $\mathbf{e}$, we can draw from the uniform distribution over m-dimensional binary vectors or from a discrete Gaussian distribution.

Definition 2 *The discrete* 1*-dimensional Gaussian distribution* $\mathcal{D}_{\mathbb{Z},\sigma}$ *with parameter* σ *is a distribution over* $\mathbb{Z}$ *where:*

$$\mathcal{D}_{\mathbb{Z},\sigma}(n) \propto \exp\left(\frac{-\pi n^2}{\sigma^2}\right).$$

The discrete Gaussian distribution is preferred over the uniform distribution over $\{0,1\}^m$ because the distribution is spherically symmetric and the coordinates remain independent.

As discussed in the previous class, the Ring Learning with Erros **RingLWE** can be more efficiently implemented than **LWE** via anti-circulant matrices and the discrete Fourier transform, so practical implementations of FHE use **RingLWE**. Recall the ring of polynomials $R_q = \mathbb{Z}_q[X]/(X^n+1)$ is ring homomorphic to the ring of anti-circulant $n \times n$ matrices. For convenience, $n = 2^j$ for some $j \in \mathbb{Z}$.

We choose $a, s, e \in R_q$ randomly, and compute $b = s \cdot a + e$. Again, e is drawn from a small error distribution. Then, it is hard to distinguish between the distributions over pairs $(a,b) \in R_q^2$ constructed in this fashion and those that are drawn uniformly at random. More generally, it is hard to distinguish between $m = \text{poly}(n)$ copies of the problem: choose random $\mathbf{a}, \mathbf{e} \in R_q^m$, and let $\mathbf{b} = s \cdot \mathbf{a} + \mathbf{e}$ (where s acts on $\mathbf{a}$ as scalar multiplication). The distribution over $(\mathbf{a}, \mathbf{b}) \in R_q^{2m}$ is pseudorandom. We'll assume this that computing s from $\mathbf{a}$ and $\mathbf{b}$ is computationally difficult for now; we will explore this later in the course. In summary, we have the two problems:

LWE(n, m, q): Generate
matrix $A \in_R \mathbb{Z}_q^{n \times m}$, and let
$b^t = s^t A + e^t$, where
$s \leftarrow \mathbb{Z}_q^n$ (or $s \leftarrow \mathcal{D}_{\mathbb{Z},\sigma}^n$) and
$e \leftarrow \mathcal{D}_{\mathbb{Z},\sigma}^m$.
Problem: Given A and b, compute s.

RingLWE(n, m, q): Generate
vector $\mathbf{a} \in R_q^m \subseteq \mathbb{Z}_q^{(n \times n)m}$, and let
$\mathbf{b} = s \cdot \mathbf{a} + \mathbf{e}$, where
$s \leftarrow R_q$ (or $s \leftarrow \mathcal{D}_{\mathbb{Z},\sigma}^n$) and
$\mathbf{e} \leftarrow \mathcal{D}_{\mathbb{Z},\sigma}^{nm}$.
Problem: Given $\mathbf{a}$ and $\mathbf{b}$, compute s.

2 Symmetric Encryption

Definition 3 *A* symmetric encryption scheme *is given by three protocols* $(\texttt{KeyGen}, \texttt{Enc}, \texttt{Dec})$*,*

$$k \leftarrow \texttt{KeyGen}(n) \qquad c \leftarrow \texttt{Enc}_k(m; r) \qquad m \leftarrow \texttt{Dec}_k(c),$$

where n *is a security parameter,* k *is the secret key,* m *is a message,* r *is a source of randomness, and* c *is the ciphertext.*

The `KeyGen` operation generates the secret key k. `Enc` encrypts messages using the secret key and some randomness, while `Dec` decrypts ciphertexts using the secret key. Note that the correctness property follows because $\texttt{Dec}_k(\texttt{Enc}_k(m;r)) = m$. Further, it should be hard for an adversary to decode c without access to k.

We can use the hardness of **RingLWE** to design an encryption scheme with this security property. Because $(a, b) \in R_q^2$ as drawn from the **RingLWE** distribution is pseudorandom (where $b = s \cdot a + e$), then so is $(a, b + m)$ for any fixed message $m \in R_q$. On the other hand, if we knew the secret key s, then we could *approximately* decrypt the ciphertext $(a, b + m)$ up to the error term e by subtracting $s \cdot a$ from $b + m$:

$$m + e = (b + m) - s \cdot a.$$

The hardness assumption about **RingLWE** implies a security property about this approximate encryption scheme. Since the ciphertext $(a, b + m)$ is pseudorandom—to an adversary without the key, it looks like the uniform distribution on R_q^2—it follows from transitivity that the distributions for two ciphertexts $\texttt{Enc}_k(m_1; r_1)$ and $\texttt{Enc}_k(m_2; r_2)$ are also indistinguishable, as they are both indistinguishable from random.

Using the full space R_q enables us to encrypt messages m with $n \lg q$ bits: R_q is isomorphic to $\mathbb{Z}_q^n$ (as a $\mathbb{Z}_q$-module). However, if we wish to be able to exactly decrypt the ciphertext, then we can make our message space sparser and build an error-correcting mechanism into the encryption scheme. In particular, we might consider encrypting only binary strings with encoding $\widetilde{m} \in \{0, \frac{q}{2}\}^n$. That way, if the error $\|e\|_\infty$ is small relative to $q/2$, then we could simply round the approximate decrypted ciphertext to retrieve the original message. In the following, let $m \in \{0, 1\}^n$ be a message and $\widetilde{m} \in \{0, \frac{q}{2}\}^n$ its encoding:

$\texttt{KeyGen}(n)$:	$\texttt{Enc}_s(m \in \{0,1\}^n)$:	$\texttt{Dec}_s(a, c)$:
$s \leftarrow R_q$	$a \leftarrow R_q$	$\widehat{m} = c - s \cdot a$ (i.e. $\frac{q}{2}m + e$)
return s	$e \leftarrow \mathcal{D}_\sigma^n$	return $\left\lfloor \frac{2}{q} \cdot \widehat{m} \right\rceil_2$
	$b = s \cdot a + e$	
	return $(a, b + \widetilde{m})$	

Here, $\lfloor v \rceil_2$ denotes the vector produced when each component of v is rounded to the nearest integer mod 2. Assuming the error $\|e\|_\infty < q/4$ is small, the decryption procedure will correctly round the input to the original message.

We defer the technical discussion of security until later in the course.

3 Additive homomorphism

Next, we show that the sum of the encoding of two messages is equal to the encoding of the sum of the messages, though at the expense of a (possibly) larger error term. Denote by $\texttt{Enc}_s(m; a, e)$ the encryption of $m \in \mathbb{Z}_2^n$, where $\texttt{Enc}_s(m; a, e) = (a, s \cdot a + e + \widetilde{m})$. Then:

$$\begin{aligned}\texttt{Enc}_s(m_0; a_0, e_0) + \texttt{Enc}_s(m_1; a_1, e_1) &= \big(a_0, s \cdot a_0 + e_0 + \widetilde{m_0}\big) + \big(a_1, s \cdot a_1 + e_1 + \widetilde{m}_1\big) \\ &= \big(a_0 + a_1, s \cdot (a_0 + a_1) + (e_0 + e_1) + (\widetilde{m}_0 + \widetilde{m}_1)\big) \\ &= \texttt{Enc}_s(m_0 + m_1; a_0 + a_1, e_0 + e_1),\end{aligned}$$

where $m_0 + m_1$ is performed componentwise modulo 2 (i.e. over the vector space $\mathbb{Z}_2^n$).

In addition to addition, we can also perform scalar multiplication homomorphically, where given some $c \in \mathbb{Z}_2$, then:

$$\begin{aligned}c \cdot \texttt{Enc}_s(m; a, e) &= c\dot{(}a, s \cdot a + e + \widetilde{m}) \\ &= (ca, s \cdot (ca) + (ce) + c\widetilde{m}) = \texttt{Enc}_s(cm; ca, ce).\end{aligned}$$

Of course, our current message space is $\mathbb{Z}_2^n$, so multiplication by a scalar $c \in \{0, 1\}$, and so cm is either $\mathbf{0}^n$ or m. However, scalar multiplication remains possible even for a larger message space $\mathbb{Z}_r^n$ where $\mathbb{Z}_r$ is encoded as $\{\frac{q}{r} \cdot j : j \in \mathbb{Z}_r\}$.

3.1 Least significant bit encoding

So far, we've encoded the message $m \in \mathbb{Z}_2^n$ into the most significant bits (MSB) of a string $\widetilde{m} \in \mathbb{Z}_q^n$. That way, if we add bounded noise e, the most significant bits of $\widetilde{m} + e$ remain unchanged. But we could have also encoded the message into the least significant bits (LSB).

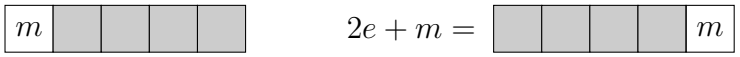

Figure 1: Let $n = 1$, so that $m \in \mathbb{Z}_2$. We can encode m into $\mathbb{Z}_q$ by using the most significant bit (left) or the least significant bit (right). Here, each box represents one of the $\lceil \lg q \rceil$ bits in the binary expansion of an element in $\mathbb{Z}_q$. The grayed boxes correspond to the error term.

Assume that q is odd, so that $2 \in \mathbb{Z}_q^\times$ is invertible in $\mathbb{Z}_q$. The (**R**)**LWE** assumption states that $(a, s \cdot a + e)$ is pseudorandom in $\mathbb{Z}_q^2$. Letting $a' = 2a$, we see that $(a', s \cdot a' + 2e)$ is also pseudorandom. This motivates this slightly different encryption/decryption scheme:

$$\texttt{Enc}_s(m) = (a, s \cdot a + 2e + m) \qquad \texttt{Dec}_s(a, b) = (b - s \cdot a) \mod 2.$$

Notice that upon subtracting $s \cdot a$ from b, we're left with an encoded message $2e + m$, where the message is encoded into the least significant bit. The error can be masked out, leaving the original message. It is straightforward to check that this encryption scheme is homomorphic under addition and scalar multiplication, so long as $(2e + m) \ll q$ is sufficiently small.

We can also perform multiplication homomorphically with this encoding.

4 Multiplicative homomorphism

There are two broad approaches to build a multiplicatively homomorphic encryption scheme on top of (**R**)**LWE**: the first is to ensure that the error can be masked out by a decoder, while the second is through *relinearization* and *key-switching*.

Consider two encrypted messages (a_0, b_0) and (a_1, b_1). For now, we neglect the error so b_i is only approximately $s \cdot a_i + m_i$. Notice that if we knew the secret key s, then:

$$m_0 m_1 \approx (b_0 - s \cdot a_0)(b_1 - s \cdot a_1) = s^2 \cdot (a_0 a_1) + s \cdot (-a_0 b_1 - b_0 a_1) + b_0 b_1.$$

But the terms $(a_0 a_1)$, $(a_0 b_1 + b_0 a_1)$ and $(b_0 b_1)$ can be computed without knowledge of s, so a decryption algorithm just needs to compute a polynomial expression in s.

In other words, the decryption expressions $b - s \cdot a$ can be interpreted as symbolic polynomials; multiplication can be performed by multiplying the decryption expressions as polynomials. But notice that the resulting expression is linear in (s^2, s^1, s^0). This is the perspective we'll take in the multiplication by relinearization approach. And since a linear function can be computed homomorphically, in the key-switching step, the tuple (s^2, s^1, s^0) can be encrypted under a new secret key t then published. Decryption is performed homomorphically with respect to t (the result remains encrypted under t).

4.1 Multiplication by error masking

As a brief sketch of masking out the error, note that the product of two noisy messages of the form $(m_i + 2e_i)$ is also of the form:

$$(m_0 + 2e_0)(m_1 + 2e_1) = m_0 m_1 + 2(e_0 m_1 + m_0 e_1 + 2e_0 e_1) = m_0 m_1 + 2e'.$$

If the new error term e' is sufficiently small, modding out by 2 recovers the product $m_0 m_1$. In other words, to decrypt the product of two ciphertexts, we need to compute the following quadratic with coefficients $(a_0 a_1, -a_0 b_1 - b_0 a_1, b_0 b_1)$:

$$m_0 m_1 = \left\lfloor s^2 \cdot (a_0 a_1) + s \cdot (-a_0 b_1 - b_0 a_1) + b_0 b_1 \right\rceil_2.$$

As we multiply more ciphertexts together, the decryption algorithm entails computing a polynomial of greater degree. But notice also that the error term e' can grow rather quickly from just a few multiplication steps.

To deal with this, we'll combine the error masking idea with the most significant bit encoding. Assume that $q/2$ is odd, and that we can recover an approximate encoded message $\widetilde{m}_i + 2e_i$, where as before $\widetilde{m}_i \in \{0, \frac{q}{2}\}$. Since $2\widetilde{m}_i \mod q = 0$, we have:

$$\begin{aligned}(\widetilde{m}_0 + 2e)(\widetilde{m}_1 + 2e) &\equiv \widetilde{m}_0 \widetilde{m}_1 + 2(e_0 \widetilde{m}_1 + \widetilde{m}_0 e_1 + 2e_0 e_1) \mod q \\ &\equiv \widetilde{m}_0 \widetilde{m}_1 + 4e_0 e_1 \mod q\end{aligned}$$

We claim that if $m = m_0 m_1$, then $\widetilde{m} = \widetilde{m}_0 \widetilde{m}_1$. In particular, for the case $m_0 = m_1 = 1$, we just need that $(q/2)^2 \equiv (q/2) \mod q$. But then, this shows that as long as $4e_0 e_1$ remains small, this procedure allows us to recover the product $m_0 m_1$ from the values $\texttt{Enc}_s(m_0; a_0, e_0)$ and $\texttt{Enc}_s(m_1; a_1, e_1)$. We leave this an exercise:

Exercise 1 *Let $q/2$ be odd. Show that $(q/2)^2 \mod q \equiv q/2$. (Hint: apply the Chinese remainder theorem).*

Note that formulating multiplication in this way is easier under the ring setting; in the matrix setting, we would need the tensor product. (As an exercise, work out the details if you're familiar with the tensor product). It would also be much less efficient as a result.

4.2 Multiplication by relinearization and key switching

One way to understand the multiplicatively homomorphic property is as follows. Given an encryption $c = (a, b) \in R_q^2$ of some message m under some random a (i.e. $b = sa + e + m$), we can let (a, b) represent the symbolic polynomial $b - sa \in R_q$. While we normally think of encryption and decryption protocols as functions of some message or ciphertext, we can also view them as functions of the secret key, when the message or ciphertext is fixed.

Letting $\odot$ to be multiplication in $R_q[s]$, we obtain the quadratic polynomial in s,

$$(b_0 - sa_0) \odot (b_1 - sa_1) = s^2 \cdot c_2 + s \cdot c_1 + c_0,$$

where $c_0, c_1, c_2 \in R_q$ are constants. With knowledge of s and s^2, encrypted under some different key z, it is possible to map the quadratic polynomials in s to linear polynomials $c_0' + z \cdot c_1'$. That is, this new linear polynomial is the encryption of a new message under the key z. This technique, known as "key switching", is the other main approach to achieving the multiplicatively homomorphic property.

Before we describe in detail relinearization and key switching, we'll discuss an encryption scheme built on **LWE** and show that it can compute linear transformations, which will allow us to show how to perform key switching.

5 Generalization to arbitrary matrices

5.1 Matrix LWE

First, we introduce a matrix version of **LWE**(n, m, q, k): like before, we'll draw matrices $A \in \mathbb{Z}_q^{n \times m}$, $S \in \mathbb{Z}_q^{k \times n}$, and $E \in \mathbb{Z}_q^{k \times m}$ to compute $B = SA + E$. The adversary aims to recover S from the matrix $(A, B) \in \mathbb{Z}_q^{(n+k) \times m}$. The security assumption inherited from **LWE** is that the distributions of $(A, SA + E)$ and $\mathbb{Z}_q^{(n+k) \times m}$ are indistinguishable.

LWE(n, m, q, k): Generate matrix
$A \in_R \mathbb{Z}_q^{n \times m}$
$S \leftarrow \mathbb{Z}_q^{k \times n}$, $E \leftarrow \mathcal{D}_{\mathbb{Z},\sigma}^{k \times m}$
$B = SA + E$
return $(A, B) := \begin{bmatrix} A \\ SA + E \end{bmatrix}$

As before, we can use this instantiation of the **LWE** problem to construct an encryption scheme. Suppose that we have a pair $(\textsc{encode}, \textsc{decode})$ such that with high probability, for all error matrices E produced in the **LWE** problem, the following holds:

$$\textsc{decode}(\textsc{encode}(M) + E) = M.$$

In other words, we have a valid error-correcting code. We'll define:

$$\texttt{Enc}_S(M) := \widetilde{\texttt{Enc}}_S(\textsc{encode}(M)) \qquad \texttt{Dec}_S(A, B) := \textsc{decode}(\widetilde{\texttt{Dec}}_S(A, B)),$$

where, denoting $\textsc{encode}(M)$ by $\widetilde{M}$, we also define:

$\texttt{KeyGen}(n)$:	$\widetilde{\texttt{Enc}}_S(\widetilde{M})$:	$\widetilde{\texttt{Dec}}_S(A, B)$:
$S \leftarrow \mathbb{Z}_q^{k \times n}$	$B = SA + E$	$\widehat{M} = B - SA$ (i.e. $\widetilde{M} + E$)
	return $(A, B + \widetilde{M})$	return $\widehat{M}$

We'll assume that $q = 2^w$ for some integer $w \in \mathbb{N}$. This will be the *ciphertext modulus*. Let $m \in \mathbb{Z}_t$ for some $t = 2^h$ with $h < w$ be the message. As before, we want to perform addition and multiplication operations homomorphically, without the error growing too large.

5.2 Linear homomorphic properties

Assuming that we have a suitable encoding and decoding function, we will show that this scheme has linear homomorphic properties, as before. For clarity, let $\widetilde{\texttt{Enc}}_S(M; A, E)$ be the ciphertext $(A, SA + E + M)$.

5.2.1 Addition

Just like before, the encryption function $\widetilde{\texttt{Enc}}_S$ respects addition, with the resulting encryption having the sum of the original errors. Let $A := A_0 + A_1$.

$$\begin{aligned}
\widetilde{\texttt{Enc}}_S(M_0; A_0, E_0) + \widetilde{\texttt{Enc}}_S(M_1; A_1, E_1) &= (A_0, SA_0 + E_0 + M_0) + (A_1, SA_1 + E_1 + M_0) \\
&= (A, SA + E_0 + E_1 + M_0 + M_1) \\
&= \widetilde{\texttt{Enc}}_S(M_0 + M_1; A, E_0 + E_1)
\end{aligned}$$

5.2.2 Multiplication by a (small) scalar

Multiplication by a small constant c is also permissible. The error E scales linearly with c, so c cannot be too large.

$$\begin{aligned}
c \cdot \widetilde{\texttt{Enc}}_S(M; A, E) &= c \cdot (A, SA + E + M) \\
&= (cA, cSA + cE + cM) \\
&= \widetilde{\texttt{Enc}}_S(cM; cA, cE)
\end{aligned}$$

5.2.3 Linear transformations

More generally, one can apply full rank linear transformations $T \in \mathbb{Z}_q^{m \times m}$ to ciphertexts:

$$\widetilde{\texttt{Enc}}_s(M; A, E) \cdot T = \begin{bmatrix} A \\ SA + E + M \end{bmatrix} \cdot T = \begin{bmatrix} AT \\ SAT + ET + MT \end{bmatrix} = \widetilde{\texttt{Enc}}_S(MT; AT, ET)$$

When T is an isomorphism, then T takes a uniformly random A to another uniformly random element A'.

Exercise 2 *What happens when $T \in \mathbb{Z}_q^{n \times n}$ is not full rank?*

5.2.4 Concatentation

We can also join ciphertexts by concatenation. Let $[X, Y]$ denote the side-by-side concatenation of matrices X, Y which share the same number of rows.

$$\begin{aligned} \left[\widetilde{\texttt{Enc}}_S(M_0; E_0), \widetilde{\texttt{Enc}}_S(M_1, E_1)\right] &= \begin{bmatrix} A_0 & A_1 \\ SA_0 + E_0 + M_0 & SA_1 + E_1 + M_1 \end{bmatrix} \\ &= \begin{bmatrix} [A_0, A_1] \\ S[A_0, A_1] + [E_0, E_1] + [M_0, M_1] \end{bmatrix} \\ &= \widetilde{\texttt{Enc}}_S\big([M_0, M_1]; [A_0, A_1], [E_0, E_1]\big) \end{aligned}$$

5.3 A new way to encode (and decode)

Our previous MSB encoding was limited to messages $m \in \{0, 1\}$, defined by:

$$\textsc{encode}(m) = m \cdot \frac{q}{2} \qquad \textsc{decode}(x) = \left\lfloor \frac{2x}{q} \right\rceil_2,$$

as long as $|e| < q/4$. Here, we encoded only $0, 1 \in \mathbb{Z}_q$, and made the encoding error tolerant by shifting the relevant bit (in this case, the least significant bit) to the most significant bit. But if we're given an element $r \in \mathbb{Z}_q$ expanded in binary, we can shift any of its bits to the most significant bit, ensuring that it is recoverable after perturbation. In particular, if we pay a $\lg q$ factor, we can encode arbitrary elements of $\mathbb{Z}_q$ into $\mathbb{Z}_q^{\lg q}$.

More generally, we can apply this encoding to arbitrary matrices $M \in \mathbb{Z}_q^{k \times n}$. Define:

$$\begin{aligned} \widetilde{M} = \textsc{encode}'(M) &= M \otimes [1, 2, 2^2, \ldots, q/2] \\ &= \left[M, 2M, 4M, \ldots, \frac{q}{2}M\right] \mod q, \end{aligned}$$

where $\otimes$ is the tensor product (and recall that $q = 2^w$) and $\widetilde{M} \in \mathbb{Z}_q^{k \times (n \lg q)}$. For the decoding function, assume that we're given a noisy encoding $\widehat{M}$ of M with error E such that $|e_{ij}| < q/4$,

$$\widehat{M} = \left[\widehat{M}_{w-1}, \widehat{M}_{w-1}, \ldots, \widehat{M}_0\right] = \widetilde{M} + [E_{w-1}, \ldots, E_0].$$

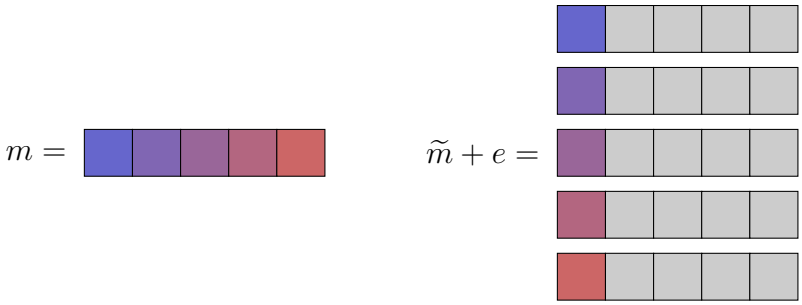

Figure 2: Let $m \in \mathbb{Z}_q$ where $q = 2^w$. Let $m_{w-1} \dots m_0$ be the binary encoding of m (left). Then, we can shift m_i to the most significant bit by multiplying by 2^{w-1-i}. Applying this to each bit leaves us with a representation that costs a factor of $\lg q$ more (right). However, because each of these terms are tolerant to noise up to $q/4$, the error will not affect any of the most significant bits.

We can perform $\textsc{decode}'(\widehat{M})$ iteratively by recovering the LSB and working our way back up to the MSB. In particular, from $\widehat{M}_0$, we can recover the LSB M_0 of M by:

$$M_0 = \left\lfloor \frac{2}{q} \cdot \widehat{M}_0 \right\rceil_2,$$

where the rounding to the nearest integer is performed component-wise. Inductively, once we've recovered the first i bits $M^{(i)} = \sum_{j=0}^{i-1} 2^j \cdot M_j$, we can recover the $(i+1)$th bit:

$$M_{(i+1)} = \left\lfloor \frac{2}{q} \left(\widehat{M}_{i+1} - 2^{(w-1)-(i+1)} M^{(i)} \right) \right\rceil_2$$

In this fashion, we can recover the original $M = \sum_{j=0}^{w-1} 2^j \cdot M_j$.

5.4 Linear transformations revisited

With this new encoding function, lets return to linear transformations. Recall that:

$$\mathtt{Enc}'_S(M) = \widetilde{\mathtt{Enc}}_S\left(\widetilde{M}\right) = \widetilde{\mathtt{Enc}}_S\left(\left[M, 2M, 4M, \dots, \frac{q}{2}M\right]\right).$$

We can also decompose the linear transformation $T \in \mathbb{Z}_q^{m \times m}$ bitwise into $\lg q$-many bit matrices: $T = \sum_{j=0}^{w-1} 2^j T_j$, where $T_j \in \{0,1\}^{m \times m}$. Define the multiplication by a linear

transformation using a new operation $\odot$:

$$\begin{aligned}
\mathtt{Enc}'_S(M;E) \odot T &:= \widetilde{\mathtt{Enc}}_S\left([M, 2M, \ldots, \frac{q}{2}M]; [E_0, E_1, \ldots, E_w]\right) \cdot \begin{bmatrix} T_0 \\ T_1 \\ \vdots \\ T_{w-1} \end{bmatrix} \\
&= \begin{bmatrix} A_0 & A_1 & \cdots & A_{w-1} \\ SA_0 + E_0 + M & SA_1 + E_1 + 2M & \cdots & SA_{w-1} + E_{w-1} + 2^{w-1}M \end{bmatrix} \begin{bmatrix} T_0 \\ T_1 \\ \vdots \\ T_{w-1} \end{bmatrix} \\
&= \widetilde{\mathtt{Enc}}_S\left(\sum_i 2^i M T_i; \sum_i E_i T_i\right) = \widetilde{\mathtt{Enc}}_S\left(MT; \sum_i E_i T_i\right)
\end{aligned}$$

As the T_i's are matrices with entries in $\{0,1\}$, the error increases by at most a $\lg q$ factor.

Remark 4 *Importantly, we at this point know how to perform addition and linear transformations homomorphically. Notice that even without knowledge of S, we can construct encryptions of M, $-S$, and $S \cdot T$ (though without security). This is useful if we need to perform homomorphic computations that make use of these values. They are simply:*

$$\mathtt{Enc}'_S(M,0) = \begin{bmatrix} 0 \\ M \end{bmatrix} \qquad \mathtt{Enc}'_S(-S,0) = \begin{bmatrix} I \\ 0 \end{bmatrix} \qquad \mathtt{Enc}'_S(ST,0) = \begin{bmatrix} -T \\ 0 \end{bmatrix}.$$

We'll make use of these in the next section on key switching.

5.5 Key Switching

Now, we will show how to use an encryption of our key S under a separate key K to decrypt some encryption (A, B) which implicitly contains error E. Let the *switching key* be $\mathtt{Enc}'_K(S) := S_K$. We will compute the decryption circuit homomorphically:

$$\textsc{eval}_K\left(\mathtt{Dec}'_{(\cdot)}(A,B), S_K\right) \equiv \mathtt{Enc}'_K\left(\mathtt{Dec}'_S(A,B)\right) \equiv \mathtt{Enc}'_K(B - SA).$$

Suppose that we are given S_K and the encrypted ciphertext (A, B). Then, we can homomorphically compute the decryption circuit $B - SA$ under the secret key K. By our previous remark, we can compute $\mathtt{Enc}'_K(B;0)$ without knowledge of K, and we can compute $\mathtt{Enc}'_K(S) \odot A$. Together:

$$\begin{aligned}
\mathtt{Enc}'_K(B;0) - \mathtt{Enc}'_K(S) \odot A &= \mathtt{Enc}'_K(B;0) - \mathtt{Enc}'_K(SA;E') \\
&= \widetilde{\mathtt{Enc}}_K(\widetilde{M} + E; E') \\
&= \widetilde{\mathtt{Enc}}_K(\widetilde{M}; E + E') = \mathtt{Enc}'_K(M; E + E').
\end{aligned}$$

Using key K, we can decrypt this value, as long as $E + E'$ is still small. Note that the extra error E' is needed to ensure that the key switching is secure.

5.6 A new multiplication operation $\odot'$

The previous operation $\odot$ defines multiplication via linear transformations, but notice that the operation $\mathtt{Enc}'_S(M) \odot T$ results in the encryption $\widetilde{\mathtt{Enc}}_S(MT)$ and not $\mathtt{Enc}'_S(MT)$. That is, while began with an encoding that consists of $\lg q$ matrices $[M, 2M, \ldots, \frac{q}{2}M]$, we obtain an encrypted result consisting of a single unencoded matrix MT. This multiplication procedure can be computed once, but it is not necessarily clear how, without knowing the cleartext of MT how to compute additional multiplication operations.

We will define a new operation $\odot'$, allowing multiplication of $\mathtt{Enc}'_S(M)$ by a constant matrix T, resulting in $\mathtt{Enc}'_S(MT)$.

$$\begin{aligned}
\mathtt{Enc}'_S(M) \odot' T &:= \mathtt{Enc}'_S(M) \odot \textsc{encode}'(T) \\
&= \mathtt{Enc}'_S(M) \odot [T, 2T, 4T, \ldots, \frac{q}{2}T] \\
&= \left[\mathtt{Enc}'_S(M) \odot T, \mathtt{Enc}'_S(M) \odot 2T, \ldots, \mathtt{Enc}'_S(M) \odot \frac{q}{2}T\right] \\
&= \left[\widetilde{\mathtt{Enc}}_S(MT), \widetilde{\mathtt{Enc}}_S(2MT), \ldots, \widetilde{\mathtt{Enc}}_S\left(\frac{q}{2}MT\right)\right] = \mathtt{Enc}'_S(MT)
\end{aligned}$$

5.7 Multiplication!

Before we show how to multiply two ciphertexts, let's recap our current encryption scheme and basic properties we have seen so far. Let $q = 2^w$. We have:

- Let $G = [I, 2I, \ldots, 2^{w-1}I]$. Then, the *encoding function* is:

$$\textsc{encode}'(M) = [M, 2M, \ldots, 2^{w-1}M] = MG.$$

- The *encryption function* is $\mathtt{Enc}'_S(M) = \widetilde{\mathtt{Enc}}_S(MG)$, where:

$$\widetilde{\mathtt{Enc}}_S(\widetilde{M}) = (A, SA + E + \widetilde{M}) \qquad \widetilde{\mathtt{Dec}}_S(A, B) = B - SA.$$

- We can perform *addition and linear transformations* homomorphically:

$$\mathtt{Enc}'_S(M_0) + \mathtt{Enc}'_S(M_1) = \mathtt{Enc}'_S(M_0 + M_1) \qquad \mathtt{Enc}'_S(M) \odot T = \widetilde{\mathtt{Enc}}_S(MT).$$

Additionally, we saw that to enable more than a single instance of homomorphic linear transformation, we needed to also encode the linear transformation T:

$$\mathtt{Enc}'_S(M) \odot' T = \mathtt{Enc}'_S(M) \odot TG.$$

We can view applying a linear transformation as the same thing as multiplying by a known or constant matrix (no security provided for T). If we wish to enable homomorphic multiplication, we'll need to incorporate the decryption circuit into the encoding function. Notice that to decrypt a message, we need to compute the left linear transformation:

$$M + E = \begin{bmatrix} -S & I \end{bmatrix} \begin{bmatrix} A \\ SA + E + M \end{bmatrix}.$$

This motivates our final encoding function:

$$\textsc{encode}^\dagger(M) := M \cdot [-S, I]G,$$

along with our final encryption function $\mathtt{Enc}^\dagger_S(M)$,

$$\mathtt{Enc}^\dagger_S(M) := \widetilde{\mathtt{Enc}}_S(\textsc{encode}^\dagger(M)).$$

First, let's show that under this new encryption function, we can still apply linear transformations on the right, but now with encryption. Because we can perform linear transformations homomorphically with $\mathtt{Enc}'_S$, we have:

$$\begin{aligned}\mathtt{Enc}^\dagger_S(M; E_M) \odot \widetilde{\mathtt{Enc}}_S(T; E_T) &= \mathtt{Enc}'_S(M \cdot [-S, I]) \odot \begin{bmatrix} A \\ SA + E_T + T \end{bmatrix} \\ &= \widetilde{\mathtt{Enc}}_S\left(M[-S, I] \begin{bmatrix} A \\ SA + E_T + T \end{bmatrix}; E_{MT}\right),\end{aligned}$$

where T has a binary decomposition $\sum 2^i T_i$ and $E_{MT} = E_M \sum T_i$. But now, the decryption circuit contained in the encoding of M decrypts T and we obtain:

$$\begin{aligned}\mathtt{Enc}^\dagger_S(M; E_M) \odot \widetilde{\mathtt{Enc}}_S(T; E_T) &= \widetilde{\mathtt{Enc}}_S\left(M(T + E_T); E_{MT}\right) \\ &= \widetilde{\mathtt{Enc}}_S(MT; E_{MT} + ME_T).\end{aligned}$$

This implies that we can multiply encrypted matrices—replacing T with $N \cdot [-S, I]G$, we immediately obtain:

$$\mathtt{Enc}^\dagger_S(M) \odot \mathtt{Enc}^\dagger_S(N) = \widetilde{\mathtt{Enc}}_S(MN \cdot [-S, I]G) = \mathtt{Enc}^\dagger_S(MN).$$

As a final note, if the errors for the encryption of M and N are E_M and E_N, respectively, then the error for their product is $E_M \sum_i N_i + ME_N$. While the first term is bounded by $\lg q \cdot E_M$, the second term may be large if M is large. If this is the case, we could first compute the binary decomposition of M before encrypting it. That is, when $M = \sum 2^i M_i$,

$$MN = \sum_{i=0}^{w-1} 2^i M_i N = \begin{bmatrix} M_0 & \cdots & M_{w-1} \end{bmatrix} \begin{bmatrix} 2^0 N \\ \vdots \\ 2^{w-1} N \end{bmatrix}.$$

In particular, the binary decomposition of M allows us to compute:

$$\mathtt{Enc}^\dagger_S(MN) = \mathtt{Enc}^\dagger_S\big([M_0, \cdots, M_{w-1}]\big) \odot \mathtt{Enc}^\dagger_S\big((2^0 N, \cdots, 2^{w-1} N)\big).$$

The binary decomposition of M has only $\{0, 1\}$ entries, so the contribution in error growth by both terms are bounded by at most $\lg q \cdot (E_M + E_N)$.

Remark 5 *We now have a somewhat homomorphic encryption scheme: we can only compute so many operations before the error becomes too large. In order to transform this into a fully homomorphic encryption scheme, we can make use of* bootstrapping. *See other class notes on this procedure.*