CSE 206A: Lattice Algorithms and Applications Fall 2017

Fully Homomorphic Encryption

Instructor: *Daniele Micciancio* UCSD CSE

In these notes we describe how to use our family of random lattices $\Lambda_q(\mathbf{A})$ to build a fully homomorphic encryption scheme, i.e., an encryption scheme that allows to perform arbitrary computations on encrypted messages, without knowing the secret decryption key. For simplicity, we consider private encryption schemes, where the encryption and decryption algorithms use the same secret key. This is already enough for the typical application of securely outsourcing computation: here a user can pick a random secret key, use it to encrypt its programs and data, and ship them to an untrusted server. The server, without knowing the secret key, can execute the program on the data, and compute the encrypted result, which can be decrypted by the user. Since encryption and decryption is performed by the same user, one can use a private key encryption scheme.

We also remark that private encryption schemes that are homomorphic, can be transformed into public key encryption schemes using general techniques: the basic idea is that one can publish many encryptions of 0 as a public key. Then, a user wishing to encrypt under it, can take a random linear combination of these public encryptions, to obtain a random encryption of 0, without knowning the private encryption/decryption key. Random encryptions of other messages are also easily obtained adding the known constant message to a random encryption of 0. So, focusing on private encryption schemes is without much loss of generality.

For simplicity, we focus on encryption schemes for 2-bit messages $m \in \{0, 1, 2, 3\}$, and use random q-ary lattices $\Lambda_q(\mathbf{A})$ for $q = n^4$, where n is a power of 2. (In particular, q is also a power of 2.) Adapting the scheme to arbitrary modulus q, and other message spaces is left as an exercise.

As a matter of notation, recall that we use $(\mathbf{A}, \mathbf{B})$ for the vertical stacking of matrices $\mathbf{A}, \mathbf{B}$, and $[\mathbf{A}, \mathbf{B}]$ for horizontal concatenation. So, $(x_1, \ldots, x_n)$ is a column vector, and $[x_1, \ldots, x_n]$ is a row vector.

1 How to encrypt with LWE

Let $\mathbf{A} \leftarrow \mathbb{Z}_q^{n \times m}$ be a randomly chosen matrix, and

$$\mathbf{b} = \mathbf{A}^T \mathbf{s} + \mathbf{e},$$

where $\mathbf{s} \in \mathbb{Z}_q^n$ is chosen uniformly at random, and $\mathbf{e} \in \mathbb{Z}_q^m$ is chosen at random from a set of "short" vectors. The problem of recovering $\mathbf{s}$ from

$$\bar{\mathbf{A}} = (\mathbf{A}, \mathbf{b}^T) \in \mathbb{Z}^{(n+1) \times m}$$

is the "Learning With Errors" (LWE) problem, an average-case variant of the bounded distance decoding (BDD) problem for a random q-ary lattice

$$\Lambda_q(\mathbf{A}) = \{\mathbf{A}^T\mathbf{x} \mid \mathbf{x} \in \mathbb{Z}_q^n\} + q\mathbb{Z}^m$$

and target vector $\mathbf{b}$. For appropriate error distributions, this problem is known to be at least as hard as approximating various lattice problems in the worst case over n-dimensional lattices. It is also known that if LWE is hard to solve, than it is also pseudorandom, i.e., it is computationally infeasible to distringuish $\bar{\mathbf{A}}$ from a uniformly random matrix in $\mathbb{Z}_q^{(n+1)\times m}$.

We will not use any specific property of the error vector distribution, other than the fact that it produces short vectors, say vectors with all coordinates bounded by $\|\mathbf{e}\|_\infty \leq \beta$ for some β much smaller than q, and that it is symmetric about the origin, i.e., χ and $-\chi$ are the same distribution. Typically $\beta = \sqrt{n}$, as this is the smallest value supported by theoretical results.

As a first step toward the construction of a fully homomorphic encryption scheme, we use the LWE problem to build a simple private key encryption scheme. The main idea is that, since $\bar{\mathbf{A}}$ is pseudorandom, we can use it as a one-time pad, and define the encryption of a message $m \in \mathbb{Z}$ as

$$\mathbf{C} = \bar{\mathbf{A}}^T + m\mathbf{G}^T = [\mathbf{A}^T, \mathbf{A}^T\mathbf{s} + \mathbf{e}] + m\mathbf{G}^T$$

where $\mathbf{G} \in \mathbb{Z}_q^{(n+1)\times m}$ is a suitably chosen matrix used to encode the message m. Security of LWE encryption (for any choice of $\mathbf{G}$) immediately follows from the pseudorandomness of the LWE matrix $\bar{\mathbf{A}} = (\mathbf{A}^T, \mathbf{b})$, because when $\bar{\mathbf{A}}$ is uniformly random, the ciphertext $\mathbf{C}$ is statistically independent of the message m.

Geometrically, one can think of $\bar{\mathbf{A}} = (\mathbf{A}, \mathbf{b}^T)$ as the description of a lattice $\Lambda_q(\bar{\mathbf{A}})$. We know (by a counting argument) that if $\bar{\mathbf{A}}$ is chosen uniformly at random, then $\Lambda_q(\bar{\mathbf{A}})$ is unlikely to contain any vector of length substantially smaller than Minkowski's bound. But, in fact, $\Lambda_q(\mathbf{A})$ contains a very short vector

$$\bar{\mathbf{A}}^T(-\mathbf{s}, 1) = [\mathbf{A}^T, \mathbf{b}](-\mathbf{s}, 1) = \mathbf{b} - \mathbf{A}^T\mathbf{s} = \mathbf{e} \pmod q.$$

We can use the knowledge of this short lattice vector, and its coordinates

$$\bar{\mathbf{s}} = (-\mathbf{s}, 1)$$

with respect to $\bar{\mathbf{A}}^T$, as a secret key. Decryption works by taking this secret linear combination of the ciphertext vectors, to obtain

$$\mathbf{C}\bar{\mathbf{s}} = (\bar{\mathbf{A}}^T + m\mathbf{G}^T)\bar{\mathbf{s}} = \mathbf{e} + m\mathbf{G}^T\bar{\mathbf{s}} \approx m\mathbf{G}^T\bar{\mathbf{s}} \pmod q.$$

The encoding matrix $\mathbf{G}$ should be chosen in such a way that the message m can be recovered from $\mathbf{e} + m\mathbf{G}^T\bar{\mathbf{s}}$. The LWE private key encryption scheme simply sets $\mathbf{G}$ to a single vector

$$\mathbf{g} = (0, \ldots, 0, q/4) \in \mathbb{Z}_q^{n+1}.$$

Similarly, $\mathbf{A} = \mathbf{a} \in \mathbb{Z}_q^n$ is just a vector, and the error $\mathbf{e} = e \in \mathbb{Z}_q$ is a scalar. This yields the following encryption scheme (specialized to message space $\{0, 1, 2, 3\}$).

Definition 1 *The LWE private key encryption scheme with message space* $\{0,1,2,3\}$ *works as follows:*

- *The secret key* $\mathbf{s} \in \mathbb{Z}_q^n$ *is chosen uniformly at random.*
- *The encryption algorithm, on input the secret key* $\mathbf{s}$ *and a message* $m \in \{0,1,2,3\}$*, picks* $\mathbf{a} \leftarrow \mathbb{Z}_q^n$ *uniformly at random and a small* $e \leftarrow \chi$*, and outputs the ciphertext*

$$\mathrm{LWE}_{\mathbf{s}}(m) = [\mathbf{a}^T, \mathbf{a}^T\mathbf{s} + e] + m\mathbf{g}^T = [\mathbf{a}^T, \mathbf{a}^T\mathbf{s} + e + m(q/4)].$$

- *The decryption algorithm, on input secret key* $\mathbf{s}$*, and ciphertext* $\mathbf{c}^T = [\mathbf{a}^T, b]$*, rounds* $\mathbf{c}^T\bar{\mathbf{s}} = \mathbf{c}^T(-\mathbf{s}, 1)$ *to the closest multiple of* $q/4$*, and outputs*

$$\mathrm{LWE}_{\mathbf{s}}^{-1}(\mathbf{c}^T) = \lfloor (4/q)((q/8) + \mathbf{c}^T\bar{\mathbf{s}}) \rfloor.$$

Recall that $q = n^4$ is a power of two, so $q/8$ is an integer, and the value $v = q/8 + \mathbf{c}^T\bar{\mathbf{s}} \in \mathbb{Z}_q$ in the decryption procedure is computed modulo q. The final output $\lfloor 4v/q \rfloor$ of the decryption function is given by the two most significant bits of v. It is easy to verify that if $|e| < q/8$, then the decryption algorithm correctly recover the message m.

Theorem 2 *If* $|e| < q/8$*, LWE ciphertexts are correctly decrypted, i.e.,*

$$\mathrm{LWE}_{\mathbf{s}}^{-1}(\mathrm{LWE}_{\mathbf{s}}(m)) = m.$$

Proof. The decryption algorithm outputs

$$\lfloor (4/q)((q/8) + \mathbf{c}^T\bar{\mathbf{s}}) \rfloor = \lfloor (4/q)((q/8) + e + mq/4) \rfloor = \lfloor (1/2) + 4e/q + m \rfloor = m$$

because $1/2 + 4e/q \in [0, 1)$. □

Notice that, unlike most other encryption methods, in lattice cryptography ciphertexts are not all equal: ciphertexts come with a natural notion of "quality", expressed by the error bound $|e| < \beta$. This bound is important for two reasons:

- On the one hand, the quality of the ciphertexts should be sufficiently high (i.e., β should be small) in order to enable correct decryption. In our example, if $\beta > q/8$ then decryption may fail!
- At the same time, errors are necessary for security: given sufficiently many noiseless ($\beta = 0$) encryptions of known messages, one can easily recover the secret key by solving a linear system of equations modulo q. In fact, $\beta \geq \sqrt{n}$ is required to prove that LWE is as hard as worst-case lattice problems, and when $\beta = o(\sqrt{n})$ LWE can be broken in subexponential time using relinearization techniques.

We will use notation $\mathrm{LWE}_{\mathbf{s}}[m, \beta]$ to denode the set of LWE encryptions of m with noise $|e| < \beta$, and assume $\beta \in [\sqrt{n}, q/8]$ for ciphertexts to be both secure and decryptable.

2 Computing on Ciphertexts

LWE encryption is (approximately) linearly homomorphic, supporting both the negation and addition of ciphertexts.

Theorem 3 *If* $\mathbf{c}_0^T \in \mathrm{LWE}_{\mathbf{s}}(m_0, \beta_0)$ *and* $\mathbf{c}_1^T \in \mathrm{LWE}_{\mathbf{s}}(m_1, \beta_1)$*, then*

- $[\mathbf{0}^T, mq/4] \in \mathrm{LWE}_{\mathbf{s}}(m, 0)$ *is a noiseless encryption of* $m \bmod 4$*.*
- $\mathbf{c}_0^T + \mathbf{c}_1^T \in \mathrm{LWE}(m_0 + m_1, \beta_0 + \beta_1)$ *is an encryption of* $(m_0 + m_1) \bmod 4$*,*
- $-\mathbf{c}_0^T \in \mathrm{LWE}(-m_0, \beta_0)$ *is an encryption of* $(-m_0) \bmod 4$*.*

Theorem 3 falls short of giving a Fully Homomorphic encryption scheme for two reasons:

- Addition only allows to express linear or affine functions. An FHE scheme should allow the computation of arbitrary functions/circuits.
- Even when limited to linear functions, the above scheme does not allow to perform an arbitrary number of additions, because, each time two ciphertexts are added, the error gets bigger.

We solve both problems at the same time by giving a refreshing procedure that on input a ciphertext $\mathbf{c}^T \in \mathrm{LWE}[m, q/8]$, outputs a encryption of

$$\textsc{half}(m) = \lfloor m/2 \rfloor \in \{0, 1\} \subseteq \{0, 1, 2, 3\}$$

with much smaller noise β, say $\beta = q/16$. Formally, we need a procedure

$$\textbf{Refresh}\colon \mathrm{LWE}_{\mathbf{s}}(m, q/8) \to \mathrm{LWE}(\textsc{half}(m), q/16).$$

It is easy to see that, together with linear functions, this operation allows to implement a NAND gate, which is complete for boolean circuits: for any $m_0, m_1 \in \{0, 1\}$,

$$\textsc{half}(2 + m_0 + m_1) = \neg(m_0 \wedge m_1).$$

So, let $\mathbf{c}^T \in \mathrm{LWE}_{\mathbf{s}}[m, q/8]$ be an encryption of $m \in \{0, 1, 2, 3\}$. We want to compute an encryption of

$$\textsc{half}(m) = \lfloor (\mathrm{LWE}_{\mathbf{s}}^{-1}(\mathbf{c}^T))/2 \rfloor = \lfloor (2/q)(q/8 + b - \mathbf{a}^T\mathbf{s}) \rfloor = \textsc{msb}(q/8 + b - \mathbf{a}^T\mathbf{s})$$

where $\mathbf{c} = (\mathbf{a}, b)$ and $\textsc{msb}$ is the most significant bit. We should think of this value as a function (defined by the ciphertext $\mathbf{c}$) mapping the secret key $\mathbf{s}$ to $\textsc{half}(m)$:

$$f_{\mathbf{c}}(\mathbf{s}) = \textsc{half}(m) = \textsc{msb}(q/8 + b - \mathbf{a}^T\mathbf{s})$$

so that, evaluating $f_{\mathbf{c}}$ homomorphically on the encryption of $\mathbf{s}$, yields the encryption of $\textsc{half}(m)$.

Let $(b_j)_{j<\log q} \in \{0,1\}^{\log q}$ be the binary representation of

$$q/8 + b = \sum_{j<\log q} 2^j b_j \pmod q.$$

Similarly, let

$$\sum_{j<\log q} 2^j a_{i,j} = -a_i \pmod q$$

be the binary representation of the coordinates of $-\mathbf{a} = -(a_1, \ldots, a_n)$, and

$$\sum_{j<\log q} 2^j s_{i,j} = s_i \pmod q$$

be the bits of the secret key. Using the bits $b_j, a_{i,j}, s_{i,j} \in \{0,1\}$, we want to compute the most significant bit of

$$\begin{aligned} q/8 + b - \mathbf{a}^T\mathbf{s} &= \sum_{j<\log q} 2^j b_j + \sum_{i=1}^{n} \Big(\sum_{j'<\log q} 2^{j'} a_{i,j'}\Big)\Big(\sum_{j''<\log q} 2^{j''} s_{i,j''}\Big) \\ &= \sum_{j<\log q} 2^j \left(b_j + \sum_{i=1}^{n} \sum_{h=0}^{j} a_{i,j-h} s_{i,h} \right). \end{aligned}$$

We compute this value using an accumulator $\mathbf{ACC}$, and the following algorithm:

1. Initialize $\mathbf{ACC} \leftarrow 0$
2. Repeat the following for $j = 0, \ldots, \log q - 1$:
   (a) $\mathbf{ACC} \leftarrow \lfloor \mathbf{ACC}/2 \rfloor$
   (b) if $b_j = 1$, then $\mathbf{ACC} \leftarrow \mathbf{ACC} + 1$
   (c) For $i = 1, \ldots, n$ and $h = 0, \ldots, j$, if $a_{i,j-h} = 1$ then
   $$\mathbf{ACC} \leftarrow \mathbf{ACC} + s_{i,h}.$$
3. Output $\mathbf{ACC} \bmod 2$

Notice that throughout the execution of the algorithm, $\mathbf{ACC}$ always takes values in the range $\{0, \ldots, N-1\}$ where $N = 2n \log q$. Of course, we cannot run this algorithm directly on the secret key because we will be given only an encryption of $s_{i,h}$. The algorithm should be evaluated homomoprhically on $E(s_{i,h})$, where E is an appropriate encryption function used to protect the secret key, and should produce the final result $\mathbf{ACC} \bmod 2 \in \{0,1\}$ encrypted under LWE with noise bound $\beta = q/16$. The accumulator data structure $\mathbf{ACC}$ and associated encryption scheme $E()$ are described in the next section.

3 The cryptographic accumulator

In order to evaluate the refreshing algorithm homomorphically on $E(s_{i,j})$, we need a cryptographic accumulator data structure $\mathbf{ACC}[v]$ holding values $v \in \{0, \ldots, 2n \log q - 1\}$, and supporting the following operations:

- INIT : $\mathbf{ACC}[0]$
- HALF : $\mathbf{ACC}[v] \to \mathbf{ACC}[\lfloor v/2 \rfloor]$
- INC : $\mathbf{ACC}[v] \to \mathbf{ACC}[v+1]$. (Here addition may be defined modulo N, but it does not matter because v will never reach N.)
- MOD2 : $\mathbf{ACC}[v] \to \mathrm{LWE}_{\mathbf{z}}[v \bmod 2, q/16]$
- ACCUM : $E_{\mathbf{z}}(s) \times \mathbf{ACC}[v] \to \mathbf{ACC}[v+s]$.

Notice that the final output of $\textsc{Mod2}(E_{\mathbf{z}}(s), \mathbf{ACC}[v])$ is encrypted under the same key used to encrypt the bits $E_{\mathbf{z}}(s)$. In practice, we will set $\mathbf{z} = \mathbf{s}$ to the same key as the initial LWE encryption scheme. But, for clarity, we describe the accumulator using a different symbol $\mathbf{z} \in \mathbb{Z}_q^n$.

We implement the accumulator as a sequence of N LWE ciphertexts, and representing the value $v \in \{0, \ldots, N-1\}$ as

$$\mathbf{ACC}[v] = \mathrm{LWE}_{\mathbf{z}}[\delta_{v,0}, \beta_0] \times \cdots \times \mathrm{LWE}_{\mathbf{z}}[\delta_{v,N-1}, \beta_{N-1}]$$

where $\delta_{v,v} = 1$ and $\delta_{v,i} = 0$ for $v \neq i$. Write $\mathbf{ACC}[v, \beta]$ for the set of accumulators holding a value v with total noise $\beta = \sum_{i<N} \beta_i$. Notice that these accumulators can be easily initialized

$$\textsc{Init} = [\mathrm{LWE}(1,0), \mathrm{LWE}(0,0), \ldots, \mathrm{LWE}(0,0)] \in \mathbf{ACC}[0,0]$$

without knowing the secret key $\mathbf{z}$. It is also easy to apply any function $f\colon \mathbb{Z}_N \to \mathbb{Z}_N$ to an accumulator without increasing the total noise.

Theorem 4 *For any function $f\colon \mathbb{Z}_N \to \mathbb{Z}_N$ there is an efficient algorithm*

$$\textsc{Apply}_f\colon \boldsymbol{ACC}[v, \beta] \to \boldsymbol{ACC}[f(v), \beta]$$

that applies f to the value stored in the accumulator without knowing the secret key $\mathbf{z}$ and without increasing the total noise.

Proof. On input $[\mathbf{c}_0, \ldots, \mathbf{c}_{N-1}] \in \mathbf{ACC}[v, \beta]$, $\textsc{Apply}_f$ uses the LWE linear homomorphic properties to compute $\mathbf{c}'_w = \mathrm{LWE}(0,0) + \sum\{\mathbf{c}_v \mid f(v) = w\}$ for all w, and outputs $[\mathbf{c}'_0, \ldots, \mathbf{c}'_{N-1}]$. □

This immediately allows to implement the functions HALF, INC and MOD2 by applying an appropriate function to the accumulator:

- $\textsc{Half}(\mathbf{ACC}) = \textsc{Apply}_f(\mathbf{ACC})$, where $f(v) = \lfloor v/2 \rfloor$.
- $\textsc{Inc}(\mathbf{ACC}) = \textsc{Apply}_f(\mathbf{ACC})$ where $f(v) = v + 1$.
- $\textsc{Mod2}(\mathbf{ACC})$ first computes $[\mathbf{c}_0, \ldots, \mathbf{c}_{N-1}] = \textsc{Apply}_f(\mathbf{ACC})$ for $f(v) = v \bmod 2$, and then outputs $\mathbf{c}_1$.

It immediately follows from the theorem that $\textsc{Half} : \mathbf{ACC}[v, \beta] \to \mathbf{ACC}[\lfloor v/2 \rfloor, \beta]$, $\textsc{Inc} : \mathbf{ACC}[v, \beta] \to \mathbf{ACC}[v + 1, \beta]$ and $\textsc{Mod2} : \mathbf{ACC}[v, \beta] \to \mathrm{LWE}_{\mathbf{z}}[v \bmod 2, \beta]$ for any noise bound β.

It remains to devise an appropriate encryption scheme $E_{\mathbf{z}}(s)$ that allows to implement $\textsc{Accum}: E(s) \times \mathbf{ACC}[v] \to \mathbf{ACC}[v + s]$. This operation will increase the noice of the accumulator, but only by an additive term. For $E_{\mathbf{z}}(s)$, we will use the GSW encryption scheme, which is a variant of the general LWE-based encryption described at the beginning of the notes, using a matrix

$$\mathbf{G} = [2^j \mathbf{I} \mid j = 0, \ldots, \log q - 1] \in \mathbb{Z}_q^{(n+1) \times (n+1) \log q}.$$

More specifically, a message $s \in \mathbb{Z}$ is encrypted as

$$E_{\mathbf{z}}(s) = [\mathbf{A}^T, \mathbf{A}^T \mathbf{z} + \mathbf{e}] + s\mathbf{G}$$

and we will write $E_{\mathbf{z}}[s, \gamma]$ as the set of GSW encryptions of s under key $\mathbf{z}$ with noise $\|\mathbf{e}\|_\infty \leq \gamma$. We will use $E_{\mathbf{z}}$ with noise bound $\gamma = \sqrt{n}$. The messa $s \in \mathbb{Z}$ can be an arbitrary integer, but we will only encrypt single bit messages $s \in \{0, 1\}$. Just like LWE, this cryptosystem is linearly homomorphic. In particular, it allows to compute

$$\overline{E(s)} = \mathbf{G}^T - E(s) = E(1 - s)$$

without increasing the noise. As before, security follows from the pseudorandomness of $\bar{\mathbf{A}}$. We will not make direct use of the decryption procedure, so we omit its explicit descryption.

This simple variation on the standard LWE encryption has the very useful property that one can multiply $\mathrm{LWE}_{\mathbf{z}}(m)$ by $E_{\mathbf{z}}(s)$.

Theorem 5 *There is an efficient algorithm that on input $\mathbf{c}^T \in \mathrm{LWE}_{\mathbf{z}}[m, \beta]$ and $\mathbf{C} \in E_{\mathbf{z}}[s, \gamma]$, computes an encryption $\mathbf{c}^T \odot \mathbf{C} \in \mathrm{LWE}_{\mathbf{z}}[sm, s\beta + \gamma(n + 1) \log q]$.*

Proof. Let $\bar{\mathbf{c}} = (\mathbf{c}_0, \ldots, \mathbf{c}_{\log q - 1}) \in \{0, 1\}^{(n+1) \log q}$ be the binary representation of

$$\mathbf{c} = \sum_{j < \log q} \mathbf{c}_j 2^j = \mathbf{G}\bar{\mathbf{c}}.$$

The product is computed as

$$\mathbf{c}^T \odot \mathbf{C} = \bar{\mathbf{c}} \cdot \mathbf{C}$$

where $\cdot$ is the standard vector-matrix multiplication. Let $\mathbf{C} = [\mathbf{A}^T, \mathbf{A}^T\mathbf{z} + \mathbf{e}] + s\mathbf{G}^T$ with $\|\mathbf{e}\|_\infty < \gamma$. Then,

$$\begin{aligned}\mathbf{c}^T \odot \mathbf{C} &= [\bar{\mathbf{c}}^T\mathbf{A}^T, \bar{\mathbf{c}}^T\mathbf{A}^T\mathbf{z} + \bar{\mathbf{c}}^T\mathbf{e}] + s\bar{\mathbf{c}}^T\mathbf{G}^T \\ &= [\mathbf{a}^T, \mathbf{a}^T\mathbf{z} + e'] + s\mathbf{c}^T \\ &\in \mathrm{LWE}_{\mathbf{z}}[0, |e'|] + s\mathrm{LWE}_{\mathbf{z}}[m, \beta] = \mathrm{LWE}_{\mathbf{z}}[sm, |e'| + s\beta]\end{aligned}$$

where $\mathbf{a} = \mathbf{A}\bar{\mathbf{c}}$ and $e' = \bar{\mathbf{c}}^T\mathbf{e}$. Since $\bar{\mathbf{c}}$ is a binary vector with $(n+1)\log q$ coordinates, we have $|e'| \leq (n+1)\log q\|\mathbf{e}\|_\infty = \gamma(n+1)\log q$. □

Using this product, we can implement the last operation required by the refreshing algorithm:

$$\begin{aligned}\textsc{Accum}(\mathbf{C} \in E_{\mathbf{z}}[s], \mathbf{ACC}[v]) &= \mathbf{ACC}[v] \odot \overline{\mathbf{C}} + \textsc{Inc}(\mathbf{ACC}[v]) \odot \mathbf{C} \\ &\in \mathbf{ACC}[v(1-s) + (v+1)s] = \mathbf{ACC}[v+s]\end{aligned}$$

where accumulators are multiplied by $\odot$ componentwise. If the inputs are in $E_{\mathbf{z}}[s, \gamma]$ and $\mathbf{ACC}[v, \beta]$, then the output $\mathbf{ACC}[v+s, \beta']$ has noise at most

$$\beta' \leq ((1-s)\beta + N\gamma(n+1)\log q) + (s\beta + N\gamma(n+1)\log q) = \beta + 2N\gamma(n+1)\log q.$$

So, the Accum operation increases the noise of the accumulator by an additive term $2N\gamma(n+1)\log q$, i.e.,

$$\textsc{Accum}\colon E[s, \gamma] \times \mathbf{ACC}[v, \beta] \to \mathbf{ACC}[v+s, \beta + 2N\gamma(n+1)\log q].$$

4 Conclusion

We gave a refreshing procedure that computes a LWE encryption of $\textsc{half}(m)$ by initializing the accumulator to 0, and then performing at most $n\log q(\log q + 1)/2$ Accum operations (as well as a number of other operations that do not increase the noise of the accumulator.) It follows that the output of the refreshing procedure $\mathrm{LWE}_{\mathbf{z}}(\textsc{msb}(v), \beta')$ has noise at most

$$\beta' \leq n\log q(\log q + 1)N\gamma(n+1)\log q = 2n^{2.5}(n+1)\log^3 q(\log q + 1) = O(n^{3.5}\log^4 n),$$

which, for n large enough, is less than $q/16 = \Omega(n^4)$, as desired.

In order to evalute arbitrary circuits, one needs to publish the encryption $E_{\mathbf{s}}(s_{i,j})$ of all the bits of the secret key. This necessitate a "circular security" assumption, since under the standard LWE assumption, $E_{\mathbf{s}}(m)$ is pseudorandom only when encrypting messages m that do not depend on the secret key. One way to avoid this circular security assumption is to use a different key $\mathbf{z} = \mathbf{s}^{(l)}$ at every level of the circuit, and then publish the encryptions $E_{\mathbf{s}^{(l+1)}}(s_{i,j}^{(l)})$ of each key under the secret key of the next level. This is provably secure under the standard LWE assumption, but it results of large evaluation keys, as well as a maximum bound on the depth of the circuits supported by the FHE scheme. Proving the circular security of $E_{\mathbf{s}}(s_{i,j})$ is an important open problem.