Homomorphic Encryption from Learning with Errors Conceptually-Simpler,Asymptotically-Faster, Attribute-Based

LATTICE

Shanghai Jiao Tong University

May 21, 2019

Content

Content

Motivation

- In this work, we propose a new technique for building FHE schemes that we call the approximate eigenvector method.
- In our scheme, homomorphic addition and multiplication are just matrix addition and multiplication. This makes our scheme both asymptotically faster and easier to understand.
- In previous schemes, the homomorphic evaluator needs to obtain the user's evaluation key, which consists of a chain of encrypted secret keys. Our scheme has no evaluation key.

Content

Homomorphic Operations(1)

For some modulus q and dimension parameter N, a ciphertext C is an $N \times N$ matrix over $\mathbb{Z}_q$, with "small" entries (much smaller than q) and the secret key $\mathbf{v}$ is an N-dimensional vector over $\mathbb{Z}_q$ with at least one "big" coefficient v_i. We restrict the message μ to be a "small" integer. We say C encrypts μ when

$$C \cdot \mathbf{v} = \mu \cdot \mathbf{v} + \mathbf{e}, \text{ where } \mathbf{e} \text{ is a "small" error vector.}$$

To decrypt, we extract the i-th row C_i from C, compute $x \leftarrow \langle C_i, \mathbf{v} \rangle = \mu \cdot v_i + e_i$, and output $\mu = \lfloor x/v_i \rceil$. In a nutshell, the essence of our scheme is that the secret key $\mathbf{v}$ is an approximate eigenvector of the ciphertext matrix C, and the message μ is the eigenvalue.

Homomorphic Operations(2)

Suppose C_1, C_2 encrypt μ_1, μ_2 in that $C_i \cdot \mathbf{v} = \mu_i \cdot \mathbf{v} + \mathbf{e}_i$ for small $\mathbf{e}_i$.

- Matrix Addition
  $C^+ \cdot \mathbf{v} = (\mu_1 + \mu_2) \cdot \mathbf{v} + (\mathbf{e}_1 + \mathbf{e}_2)$, where the error has grown a little.

- Matrix Multiplication

$$\begin{aligned} C^\times \cdot \mathbf{v} &= C_1 \cdot (\mu_2 \cdot \mathbf{v} + \mathbf{e}_2) \\ &= \mu_2 \cdot (\mu_1 \cdot \mathbf{v} + \mathbf{e}_1) + C_1 \cdot \mathbf{e}_2 \\ &= \mu_1 \cdot \mu_2 \cdot \mathbf{v} + \mu_2 \cdot \mathbf{e}_1 + C_1 \cdot \mathbf{e}_2 \\ &= \mu_1 \cdot \mu_2 \cdot \mathbf{v} + \mathit{small}, \end{aligned}$$

  where the final error vector is hopefully "small", since μ_2, C_1, $\mathbf{e}_1$, and $\mathbf{e}_2$ are all small.

(Decisional) Learning with Errors (LWE) Problem

Let λ be a security parameter. For parameters $n = n(\lambda)$, $q = q(\lambda) \geq 2$, and a distribution $\chi = \chi(\lambda)$ over $\mathbb{Z}$, the $LWE_{n,q,\chi}$ problem is to distinguish the following distributions:

- ***Distribution 0**: The i-th sample $(\boldsymbol{a_i}, b_i) \in \mathbb{Z}_q^{n+1}$ is computed by uniformly sampling $\boldsymbol{a_i} \xleftarrow{\$} \mathbb{Z}_q^n$ and $b_i \xleftarrow{\$} \mathbb{Z}_q$.*
- ***Distribution 1**: Generate uniform vector $\boldsymbol{s} \xleftarrow{\$} \mathbb{Z}_q^n$. The i-th sample $(\boldsymbol{a_i}, b_i) \in \mathbb{Z}_q^{n+1}$ is computed by uniformly sampling $\boldsymbol{a_i} \xleftarrow{\$} \mathbb{Z}_q^n$, sampling an error value $e_i \xleftarrow{\$} \chi$ and computing $b_i \leftarrow \langle \boldsymbol{a_i}, \boldsymbol{s} \rangle + e_i$.*

B-bounded distributions and $GapSVP_\gamma$

Definition (B-bounded distributions)

A distribution ensemble $\{D_n\}_{n\in\mathbb{N}}$, supported over the integers, is called B-bounded if $Pr_{e\xleftarrow{\$}D_n}[|e| > B] = negl(n)$.

Definition ($GapSVP_\gamma$)

Let n be a lattice dimension, and let d be a real number. Then $GapSVP_\gamma$ is the problem of deciding whether an n-dimensional lattice has a nonzero vector shorter than d or no nonzero vector shorter than $\gamma(n) \cdot d$.

Reduction from LWE to $GapSVP_\gamma$

Theorem (Reg05,Pei09,MM11,MP12, Bra12)

Let $q = q(n) \in \mathbb{N}$ be either a prime power or a product of small (poly(n)) distinct primes, and let $B \geq \omega(\log n) \cdot \sqrt{n}$. Then there exists an efficient sampleable B-bounded distribution such that if there is an efficient algorithm that solves the average-case $LWE_{n,q,\chi}$ problem, then:

- *There is an efficient quantum algorithm that solves $GapSVP_{\tilde{O}(nq/B)}$ on any n-dimensional lattice.*
- *If $q > \tilde{O}(2^{n/2})$, then there is an efficient classical algorithm for $GapSVP_{\tilde{O}(nq/B)}$ on any n-dimensional lattice.*

B-boundedness

Definition (B-boundedness)

Let $B < q$ be an integer. Let $\mathbf{C}$ be a ciphertext matrix that encrypts μ. Let $\mathbf{v}$ be a secret key vector such that $\mathbf{C} \cdot \mathbf{v} = \mu \cdot \mathbf{v} + \mathbf{e}$. Then $\mathbf{C}$ is said to be B-bounded (with respect to $\mathbf{v}$) if the magnitude of μ is at most B, the magnitude of all the entries of $\mathbf{C}$ is at most B, and $\|\|\mathbf{e}\|\|_\infty \leq B$.

B-boundedness

Suppose C_1 and C_2 are B-bounded ciphertexts, then C^+ is $2B$-bounded, and $C^\times$ is $(N+1)B^2$-bounded. In short, the error level grows worse than B^{2^L}, doubly exponentially with the multiplicative depth L of the circuit being evaluated.

$$
\begin{aligned}
C^+ \cdot \mathbf{v} &= (\mu_1 + \mu_2) \cdot \mathbf{v} + (\mathbf{e}_1 + \mathbf{e}_2) \\
C^\times \cdot \mathbf{v} &= C_1 \cdot (\mu_2 \cdot \mathbf{v} + \mathbf{e}_2) \\
&= \mu_2 \cdot (\mu_1 \cdot \mathbf{v} + \mathbf{e}_1) + C_1 \cdot \mathbf{e}_2 \\
&= \mu_1 \cdot \mu_2 \cdot \mathbf{v} + \mu_2 \cdot \mathbf{e}_1 + C_1 \cdot \mathbf{e}_2 \\
&= \mu_1 \cdot \mu_2 \cdot \mathbf{v} + \mathit{small},
\end{aligned}
$$

The modulus q can be chosen to bound the error, but we must be careful to ensure that the ratio q/B is at most subexponential in N to guarantee security.

B-strong-boundedness

Definition (B-strong-boundedness)

Let $B < q$ be an integer. Let $\mathbf{C}$ be a ciphertext matrix that encrypts μ. Let $\mathbf{v}$ be a secret key vector such that $\mathbf{C} \cdot \mathbf{v} = \mu \cdot \mathbf{v} + \mathbf{e}$. Then $\mathbf{C}$ is said to be B-bounded (with respect to $\mathbf{v}$) if the magnitude of μ is at most 1, the magnitude of all the entries of $\mathbf{C}$ is at most 1, and $\|\mathbf{e}\|_\infty \leq B$.

NAND Gate

Let $\mathbf{C}_1$ and $\mathbf{C}_2$ be ciphertext matrices that encrypt $\mu_1, \mu_2 \in \{0, 1\}$ respectively. A NAND gate can be evaluated on two ciphertexts $\mathbf{C}_1$ and $\mathbf{C}_2$ as follows:

$$\mathbf{C}_3 = \mathbf{I}_N - \mathbf{C}_1 \cdot \mathbf{C}_2, \text{ where } \mathbf{I}_N \text{ is the } N \times N \text{ identity matrix.}$$

Now if $\mathbf{C}_1$ and $\mathbf{C}_2$ are B-strongly-bounded, then the coefficients of $\mathbf{C}_3$'s error vector have magnitude at most $(N+1)B$, which is in contrast to $(N+1)B^2$ above where $\mathbf{C}_1$ and $\mathbf{C}_2$ were just B-bounded.

Suppose there were some way to preserve strong-boundedness in $\mathbf{C}_3$ (i.e. to ensure the magnitude of its entries remained at most 1). Then it would be the case that $\mathbf{C}_3$ is $(N+1)B$-strongly bounded. As a result, the error level would grow to at most $(N+1)^L B$ when evaluating a circuit of NAND gates of depth L.

Basic Operations(1)

Basic Operations Let $\ell_q = \lfloor \lg q \rfloor + 1$. Let $\boldsymbol{v} \in \mathbb{Z}_q^{m'}$ be a vector of some dimension m' over $\mathbb{Z}_q$. Let $N = m' \cdot \ell_q$.

- **BitDecomp**$(\boldsymbol{v})$: We define an algorithm BitDecomp that takes as input a vector $\boldsymbol{v} \in \mathbb{Z}_q^{m'}$ and outputs an N-dimensional vector $(v_{1,0}, \ldots, v_{1,\ell_q-1}, \ldots, v_{k,0}, \ldots, v_{k,\ell_q-1})$ where $v_{i,j}$ is the j-th bit in v_i's binary representation (ordered from least significant to most significant).
- **BitDecomp**$^{-1}(\boldsymbol{v'})$: We define an "inverse" algorithm BitDecomp^{-1} that takes an N-dimensional vector $\boldsymbol{v'} = (v'_{1,0}, \ldots, v'_{1,\ell_q-1}, \ldots, v'_{k,0}, \ldots, v'_{k,\ell_q-1})$, and outputs a m'-dimensional vector $(\sum_{j=0}^{\ell_q-1} 2^j \cdot v'_{1,j}, \ldots, \sum_{j=0}^{\ell_q-1} 2^j \cdot v'_{k,j})$. Note that the input vector $\boldsymbol{v'}$ need not be binary, the algorithm is well-defined for any input vector in $\mathbb{Z}_q^N$.
- **Flatten**$(\boldsymbol{v'})$: The algorithm Flatten takes as input an N-dimensional vector $\boldsymbol{v'} \in \mathbb{Z}_q^N$ and outputs an N-dimensional binary vector BitDecomp(BitDecomp$^{-1}(\boldsymbol{v'}) \in \{0,1\}^N$.
- **Powersof2**$(\boldsymbol{v})$: The algorithm Powersof2 takes a m'-dimensional vector $\boldsymbol{v} \in \mathbb{Z}_q^{m'}$ and outputs an N-dimensional vector $(v_1, 2v_1, \ldots, 2^{\ell_q-1}v_1, \ldots, v_k, 2v_k, \ldots, 2^{\ell_q-1}v_k)$.

Basic Operations(2)

We also define *BitDecomp*, $BitDecomp^{-1}$ and *Flatten* for matrix inputs. In this case, the respective algorithm is applied to each row independently.

Let $\mathbf{a}, \mathbf{b} \in \mathbb{Z}_q^{m'}$ be m'-dimensional vectors, and let $\mathbf{a}' \in \mathbb{Z}_q^N$ be an N-dimensional vector, we have:

- $\langle \text{BitDecomp}(\boldsymbol{a}), \text{Powersof2}(\boldsymbol{b}) \rangle = \langle \boldsymbol{a}, \boldsymbol{b} \rangle$.
- $\langle \boldsymbol{a'}, \text{Powersof2}(\boldsymbol{b}) \rangle = \langle \text{BitDecomp}^{-1}(\boldsymbol{a'}), \boldsymbol{b} \rangle = \langle \text{Flatten}(\boldsymbol{a'}), \text{Powersof2}(\boldsymbol{b}) \rangle$.

Basic Operations(3)

With the help of the above techniques, we can tackle the problem of preserving strong-boundedness after a NAND operation. In order to make the coefficients of $\mathbf{C}_3$ above have magnitude at most 1, we apply *Flatten* to the matrix $\mathbf{C}_3$.

Thus, we compute $C^{NAND} \leftarrow Flatten(\mathbf{C}_3)$ to produce the output ciphertext of the NAND gate. The vector $\mathbf{v}$ must have a special form, more precisely, $\mathbf{v}$ is computed as $Powersof2(\mathbf{s}) \in \mathbb{Z}_q^N$ for some secret key vector $\mathbf{s} \in \mathbb{Z}_q^{m'}$ for some m'.

With this form of secret key vector $\mathbf{v}$, it holds that $Flatten(\mathbf{C}) \cdot \mathbf{v} = \mathbf{C} \cdot \mathbf{v}$ for any $N \times N$ matrix $\mathbf{C}$. So C^{NAND} will have entries in $\{0, 1\}$ and thus be strongly-bounded.

Content

Basic Encryption Scheme(1)

- $\mathsf{Setup}(1^\lambda, 1^L)$: Choose a modulus q of $\kappa = \kappa(\lambda, L)$ bits, lattice dimension parameter $n = n(\lambda, L)$, and error distribution $\chi = \chi(\lambda, L)$ appropriately for LWE that achieves at least 2^λ security against known attacks. Also, choose parameter $m = m(\lambda, L) = O(n \log q)$. Let $params = (n, q, \chi, m)$. Let $\ell = \lfloor \log q \rfloor + 1$ and $N = (n+1) \cdot \ell$.
- $\mathsf{SecretKeyGen}(params)$: Sample $\boldsymbol{t} \leftarrow \mathbb{Z}_q^n$. Output $sk = \boldsymbol{s} \leftarrow (1, -t_1, \ldots, -t_n) \in \mathbb{Z}_q^{n+1}$. Let $\boldsymbol{v} = \mathsf{Powersof2}(\boldsymbol{s})$.
- $\mathsf{PublicKeyGen}(params, sk)$: Generate a matrix $B \leftarrow \mathbb{Z}_q^{m \times n}$ uniformly and a vector $\boldsymbol{e} \leftarrow \chi^m$. Set $\boldsymbol{b} = B \cdot \boldsymbol{t} + \boldsymbol{e}$. Set A to be the $(n+1)$-column matrix consisting of $\boldsymbol{b}$ followed by the n columns of B. Set the public key $pk = A$. (*Remark:* Observe that $A \cdot \boldsymbol{s} = \boldsymbol{e}$.)

Basic Encryption Scheme(2)

– $\mathsf{Enc}(params, pk, \mu)$: To encrypt a message $\mu \in \mathbb{Z}_q$, sample a uniform matrix $R \in \{0,1\}^{N \times m}$ and output the ciphertext C given below.

$$C = \mathsf{Flatten}\big(\mu \cdot I_N + \mathsf{BitDecomp}(R \cdot A)\big) \in \mathbb{Z}_q^{N \times N}.$$

– $\mathsf{Dec}(params, sk, C)$: Observe that the first ℓ coefficients of $\boldsymbol{v}$ are $1, 2, \ldots, 2^{\ell-1}$. Among these coefficients, let $v_i = 2^i$ be in $(q/4, q/2]$. Let C_i be the i-th row of C. Compute $x_i \leftarrow \langle C_i, \boldsymbol{v} \rangle$. Output $\mu' = \lfloor x_i / v_i \rceil$.

Basic Encryption Scheme(3)

If C is properly generated, then by the elementary properties of *BitDecomp* and *Powersof*2, we have

$$\begin{aligned} C \cdot v &= \mu \cdot v + BitDecomp(R \cdot A) \cdot v \\ &= \mu \cdot v + BitDecomp(R \cdot A) \cdot Powerof2(s) \\ &= \mu \cdot v + R \cdot A \cdot s \\ &= \mu \cdot v + R \cdot e \end{aligned}$$

Dec only uses the i-th coefficient of the above expression, which is $x_i = \mu \cdot v_i + \langle R_i, e_i \rangle$. The error $\langle R_i, e_i \rangle$ has magnitude at most $\|e\|_1$.

In general, if $x_i = \mu \cdot v_i + e'$ for some error e' of magnitude at most $q/8$, and if $v_i \in (q/4, q/2]$, then x_i/v_i differs from μ by at most $(q/8)/v_i < 1/2$, and *Dec* uses rounding to output the correct value of μ.

Homomorphic Operations(1)

We provide additional homomorphic operations MultConst, Add, Mult, NAND as follows.

- $\mathsf{MultConst}(C, \alpha)$: To multiply a ciphertext $C \in \mathbb{Z}_q^{N \times N}$ by known constant $\alpha \in \mathbb{Z}_q$, set $M_\alpha \leftarrow \mathsf{Flatten}(\alpha \cdot I_N)$ and output $\mathsf{Flatten}(M_\alpha \cdot C)$. Observe that:

$$\begin{aligned}\mathsf{MultConst}(C, \alpha) \cdot \boldsymbol{v} &= M_\alpha \cdot C \cdot \boldsymbol{v} = M_\alpha \cdot (\mu \cdot \boldsymbol{v} + \boldsymbol{e}) = \mu \cdot (M_\alpha \cdot \boldsymbol{v}) + M_\alpha \cdot \boldsymbol{e} \\ &= \alpha \cdot \mu \cdot \boldsymbol{v} + M_\alpha \cdot \boldsymbol{e}\end{aligned}$$

- $\mathsf{Add}(C_1, C_2)$: To add ciphertexts $C_1, C_2 \in \mathbb{Z}_q^{N \times N}$, output $\mathsf{Flatten}(C_1 + C_2)$. The correctness of this operation is immediate. Note that the addition of messages is over the full base ring $\mathbb{Z}_q$.

Homomorphic Operations(2)

– $\mathsf{Mult}(C_1, C_2)$: To multiply ciphertexts $C_1, C_2 \in \mathbb{Z}_q^{N\times N}$, output $\mathsf{Flatten}(C_1 \cdot C_2)$. Observe that:

$$\begin{aligned}\mathsf{Mult}(C_1, C_2) \cdot \boldsymbol{v} = C_1 \cdot C_2 \cdot \boldsymbol{v} &= C_1 \cdot (\mu_2 \cdot \boldsymbol{v} + \boldsymbol{e}_2) + \mu_2 \cdot (\mu_1 \boldsymbol{v} + \boldsymbol{e}_1) + C_1 \cdot \boldsymbol{e}_2 \\ &= \mu_1 \cdot \mu_2 \cdot \boldsymbol{v} + \mu_2 \cdot \boldsymbol{e}_1 + C_1 \cdot \boldsymbol{e}_2\end{aligned}$$

– $\mathsf{NAND}(C_1, C_2)$: To NAND ciphertexts $C_1, C_2 \in \mathbb{Z}_q^{N\times N}$ that are known to encrypt messages $\mu_1, \mu_2 \in \{0, 1\}$, output $\mathsf{Flatten}(I_N - C_1 \cdot C_2)$. Observe that:

$$\mathsf{NAND}(C_1, C_2) \cdot \boldsymbol{v} = (I_N - C_1 \cdot C_2) \cdot \boldsymbol{v} = (1 - \mu_1 \cdot \mu_2) \cdot \boldsymbol{v} - \mu_2 \cdot \boldsymbol{e}_1 - C_1 \cdot \boldsymbol{e}_2$$

Homomorphic Operations(3)

MultConst increases the error by a factor of at most N, regardless of what element $\alpha \in \mathbb{Z}_q$ is used for multiplication. Mult and NAND homomorphic operation increases the error by a factor of at most $N + 1$.

The circuit can be converted to use only NAND gates, the final ciphertext's error will be bounded by $(N + 1)^L \cdot B$, where L is the NAND-depth of the circuit, and B is the original bound on the error of a fresh encryption of $\{0, 1\}$.

As long as $q/B > 8(N + 1)^L$, we can evaluate a depth-L circuit of NANDs over B-bounded ciphertexts to obtain a $q/8$-bounded ciphertext, which Dec will decrypt correctly.

References

Craig Gentry, Amit Sahai, and Brent Waters (2013)

Homomorphic Encryption from Learning with Errors: Conceptually-Simpler,Asymptotically-Faster,Attribute-Based

Advances in Cryptology-CRYPTO 2013 pp.75-92.

Michael Clear and Ciaran McGoldrick (2015)

Multi-identity and multi-key leveled FHE from learning with errors.

In Advances in Cryptology - CRYPTO 2015,Proceedings, Part II pp. 630-656.

Pratyay Mukherjee and Daniel Wichs (2016)

Two Round Multiparty Computation via Multi-key FHE

Advances in Cryptology-EUROCRYPT 2016 , Proceedings, Part II, pp.735-763.