Lecture 19: Fully Homomorphic Encryption (FHE)

Pt. 1

Plan

Recap: LWE

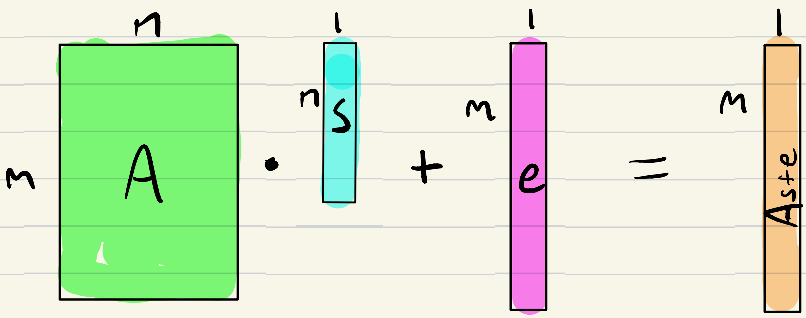
Fully Homomorphic Encryption

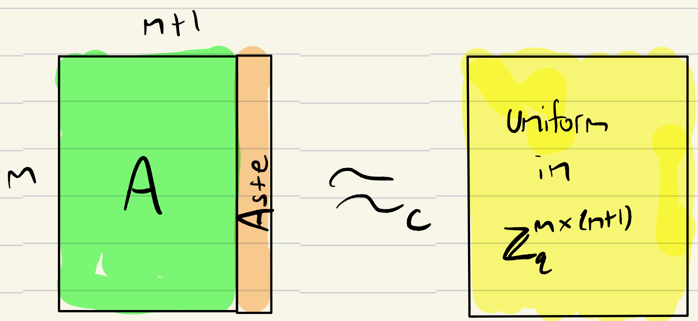
- Introduction & history
- Syntax & security
- Building <u>leveled</u> FHE

Recap: Learning with Errors

$LWE(n, m, q, \chi_B)$

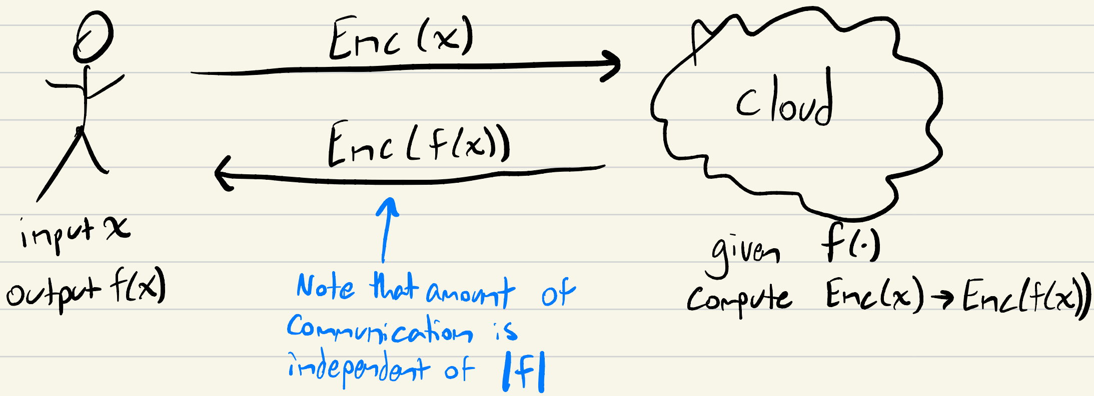
$$\left\{ (A, A\vec{s}+\vec{e}) : \begin{array}{l} A \xleftarrow{R} \mathbb{Z}_q^{m\times n} \\ \vec{s} \xleftarrow{R} \mathbb{Z}_q^{n} \\ \vec{e} \xleftarrow{R} \chi_B^{m} \end{array} \right\} \approx_c \left\{ (A, u) : \begin{array}{l} A \xleftarrow{R} \mathbb{Z}_q^{m\times n} \\ u \xleftarrow{R} \mathbb{Z}_q^{m} \end{array} \right\}$$





Fully Homomorphic Encryption

Idea: outsource computation without revealing inputs!



Examples:

- PIR: input i, output $f(i) = DB[i]$
- private ML: training, inference
- whatever you want to outsource!

Brief history

Context: Diffie & Hellman introduced PK crypto in 1976

- 1978 – Rivest, Adleman, Dertouzos introduced a notion of FHE

↓ no unbroken candidates for many years

- 2009 – Craig Gentry (Stanford PhD student)

↖ CS355 TA fall '07

gives first construction!
  - from new (non-standard) assumption
  - introduces "bootstrapping" idea (see next lecture)

- 2011 – Brakerski, Vaikuntanathan

FHE based on LWE

- 2013 – Gentry, Sahai, Waters

"3rd gen" FHE ← today's topic

FHE Syntax

$\mathrm{KeyGen}(1^\lambda) \to sk$ — We will look at symmetric-key FHE. There is actually a generic transformation to get PK FHE too

$\mathrm{Enc}(sk, \mu) \to ct$

$\mathrm{Dec}(sk, ct) \to \mu$

$\mathrm{Eval}(F, \underbrace{ct_1, \ldots, ct_\lambda}_{\text{encryptions of inputs}}) \to \tilde{ct}$ — encryption of circuit output

function to evaluate, represented as boolean circuit

FHE Properties

1) Correctness: $\forall F: \{0,1\}^\ell \to \{0,1\}$, $\mu_1, \ldots, \mu_\ell \in \{0,1\}$

$sk \leftarrow KeyGen(1^\lambda)$, then with probability 1:

$$Dec(sk, Eval(F, Enc(sk, \mu_1), \ldots, Enc(sk, \mu_\ell))) = F(\mu_1, \ldots, \mu_\ell)$$

+ usual encryption correctness

2) Semantic security

$\forall \mu_0, \mu_1 \in \{0,1\}$ $\quad \{Enc(sk, \mu_0)\} \approx_c \{Enc(sk, \mu_1)\}$

3) Compactness

$\forall F, sk$ $\quad ct_i \leftarrow Enc(sk, \mu_i)$

if $\tilde{ct} \leftarrow Eval(F, ct_1, \ldots, ct_\ell)$

then $|\tilde{ct}| = poly(\lambda)$ ← $\tilde{ct}$ size is independent of $|F|$, ℓ

Note: without compactness, any encryption scheme is also fully homomorphic!

$$\begin{cases} Eval(F, \{ct_i\}) \to (F, \{ct_i\}) \\ Dec(sk, (F, \{ct_i\})) \to F(Dec(sk, ct_1), \ldots, Dec(sk, ct_\ell)) \end{cases}$$

Eval just writes down F and all the inputs,
Dec evaluates F after decrypting!

Constructing FHE

Today: construct leveled FHE,

Can only evaluate low-depth circuits

Reason: Ciphertexts have noise that grows with each gate in circuit. Eventually, the noise overwhelms the msg.

Next time: use bootstrapping to remove restriction on circuit depl

Idea: refresh ciphertexts to clear accumulated noise.

Attempt 1 (insecure)

Secret Key is a vector $\vec{s}$

$\vec{s}$ is eigenvector of C w/ eigenvalue μ

Enc$(\vec{s}, \mu) \to$ matrix C s.t. $\underbrace{C \cdot \vec{s} = \mu \cdot \vec{s}}$

Dec$(\vec{s}, C) \to$ Compute $C \cdot \vec{s} = \mu \cdot \vec{s}$ and find μ

Homomorphism: if C_1, C_2 are encryptions of μ_1, μ_2

Addition: $\tilde{C} = C_1 + C_2$
Eval("+", C_1, C_2)

$$(C_1 + C_2)\vec{s} = C_1\vec{s} + C_2\vec{s} \underset{\substack{\uparrow \\ \text{by def} \\ \text{of Enc}}}{=} \mu_1\vec{s} + \mu_2\vec{s} = (\mu_1 + \mu_2)\vec{s}$$

multiplication: $\tilde{C} = C_1 \cdot C_2$
Eval("·", C_1, C_2)

$$(C_1 \cdot C_2)\vec{s} = C_1 \cdot (C_2\vec{s}) \underset{\substack{\uparrow \\ \text{by def of Enc}}}{=} C_1 \mu_2 \vec{s} = \mu_2 \cdot (C_1 \cdot \vec{s}) \underset{\substack{\uparrow \\ \text{by def of Enc}}}{=} \mu_2 \mu_1 \vec{s} = \mu_1 \mu_2 \vec{s}$$

Can eval $+, \cdot$ $\to$ fully homomorphic!

Problem: Given C, it's easy to find $\vec{s}$ using Gaussian elimination :(

idea: We can make Gaussian elimination hard by adding noise!

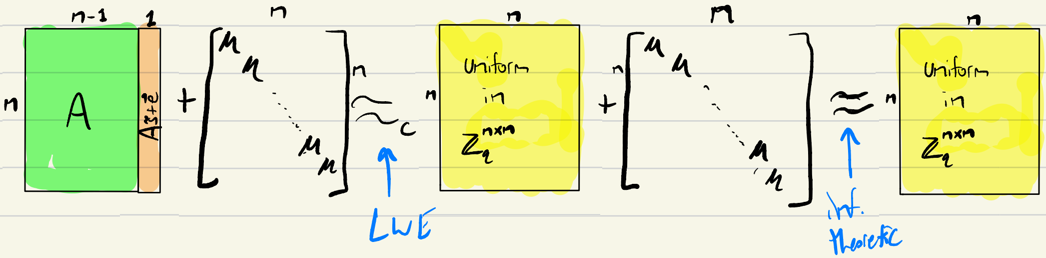
$$C \cdot \vec{s} = \mu \cdot \vec{s} + \underset{\substack{\uparrow \\ \text{Small noise}}}{\vec{e}}$$

Attempt 2: Secret-key variant of Regev encryption

$\text{KeyGen}(1^n)$: $\tilde{s} \xleftarrow{R} \mathbb{Z}_q^{n-1}$, $\vec{s} \leftarrow \begin{pmatrix} \tilde{s} \\ -1 \end{pmatrix} \in \mathbb{Z}_q^n$

$\text{Enc}(\vec{s}, \mu)$: $A \xleftarrow{R} \mathbb{Z}_q^{n \times (n-1)}$ $\vec{e} \xleftarrow{R} \chi_B^n$

$$\text{output } C = \underbrace{(A, A\tilde{s} + \vec{e})}_{\text{pseudorandom by LWE}} + \mu \cdot I_n \in \mathbb{Z}_q^{n \times n}$$



$\text{Dec}(\vec{s}, C)$: Compute $C \cdot \vec{s}$, output $\begin{cases} 0 & \text{if } \|C \cdot \vec{s}\|_\infty \text{ small} \\ 1 & \text{otherwise} \end{cases}$

$$C \cdot \vec{s} = (A, A\tilde{s} + \vec{e}) \begin{pmatrix} \tilde{s} \\ -1 \end{pmatrix} + \mu I_n \vec{s}$$

$$= \cancel{A\tilde{s}} - \cancel{A\tilde{s}} - \vec{e} + \mu\vec{s} = \mu \vec{s} + \text{noise} \rightarrow \begin{cases} \text{small}: & \mu = 0 \\ \text{large}: & \mu = 1 \end{cases}$$

$\vec{s}$ is "approximate eigenvector" of C with approx. eigenvalue μ.

Homomorphism:

Addition: $\tilde{C} \leftarrow C_1 + C_2$

$$(C_1 + C_2)\vec{s} = C_1\vec{s} + C_2\vec{s} = \mu_1\vec{s} + \vec{e}_1 + \mu_2\vec{s} + \vec{e}_2 = (\mu_1 + \mu_2)\vec{s} + \underbrace{(\vec{e}_1 + \vec{e}_2)}_{\text{noise still reasonably small}}$$

Noise doesn't grow too much, so we have additive homomorphism

(Would need to adjust Dec for $\mu \notin \{0,1\}$)

Multiplication: Can we do $\tilde{C} \leftarrow C_1 \cdot C_2$?

$$(C_1 \cdot C_2)\vec{s} = C_1(\mu_2\vec{s} + \vec{e}_2) = \mu_2 C_1\vec{s} + C_1 \cdot \vec{e}_2$$

$$= \mu_2(\mu_1\vec{s} + \vec{e}_1) + C_1 \cdot \vec{e}_2$$

$$= \mu_1 \cdot \mu_2 \cdot \vec{s} + \underbrace{\mu_2 \cdot \vec{e}_1}_{\substack{\text{still reasonably small} \\ \text{for } \mu_2 \in \{0,1\}}} + \underbrace{C_1 \cdot \vec{e}_2}_{\text{Can be large :(}}$$

So we're still unable to multiply b/c noise grows with $\|C_1\|_\infty$, which can be large.

Need a way to make ciphertext matrices have small norm.

Idea: represent number $x \in \mathbb{Z}_q$ as a small-norm vector via binary decomposition!

Binary Decomposition

For $x \in \mathbb{Z}_q$,

Define $\hat{x} = (x_0, x_1, \dots, x_{\log q - 1})$ s.t. $x = \sum_{i=0}^{\log q - 1} x_i \cdot 2^i$

inverse operation $\hat{x} \to x$ is linear!

let G be the vector that recovers x from $\hat{x}$: $\hat{x} \cdot \vec{G} = x$

$$\hat{x} \quad \cdot \quad \vec{G} \quad = x$$

$$1\ \underbrace{(x_0, \dots, x_{\log q - 1})}_{\log q} \cdot \underbrace{\left.\begin{pmatrix} 2^0 \\ 2^1 \\ \vdots \\ 2^{\log q - 1} \end{pmatrix}\right\} \log q}_{1} = x$$

Note: we call it $\vec{G}$ for "gadget"

Can extend $(\hat{\cdot})$ operation to vectors

$$\vec{x} \in \mathbb{Z}_q \longrightarrow \hat{x} = (x_{0,0}, x_{0,1}, \ldots, x_{0,\log q-1}, \ldots, x_{n,0}, x_{n,1}, \ldots, x_{n,\log q-1})$$

$$\in \{0,1\}^{n \log q}$$

$$\vec{x} = \underbrace{\hat{x}}_{1 \times n\log q} \cdot \underbrace{G}_{n\log q \times n} \quad \text{is a linear transformation}$$

where G is the matrix that recovers $\vec{x}$ from $\hat{x}$: $\hat{x} \cdot G = \vec{x}$

$$G = n\log q \left\{ \underbrace{\begin{bmatrix} 1 & & & \\ 2 & & & \\ \vdots & & & \\ \log q - 1 & & & \\ & 1 & & \\ & 2 & & \\ & \vdots & & \\ & \log q - 1 & & \\ & & \ddots & \\ & & & 1 \\ & & & 2 \\ & & & \vdots \\ & & & \log q - 1 \end{bmatrix}}_{n} \right. \in \mathbb{Z}_q^{n\log q \times n}$$

Finally, can also extend $(\hat{\cdot})$ to matrices

$$C = \underbrace{\begin{pmatrix} \vec{c}_1 \\ \vdots \\ \vec{c}_m \end{pmatrix}}_{m \times n} \rightarrow \hat{C} = \underbrace{\begin{pmatrix} \hat{c}_1 \\ \vdots \\ \hat{c}_m \end{pmatrix}}_{m \times n \log q}$$

And $C = \hat{C} \cdot G$ is still a linear transformation

(with gadget matrix G same as above)

Note: Some sources refer to G as G^{-1} because it inverts bit decomposition

Now, let's get back to FHE!

3^{rd} (and final) attempt: the GSW scheme

KeyGen(1^n): $\tilde{s} \xleftarrow{R} \mathbb{Z}_q^{n-1}$ $\quad \vec{s} \leftarrow \begin{pmatrix} \tilde{s} \\ -1 \end{pmatrix} \in \mathbb{Z}_q^n$

Enc$(\vec{s}, \mu)$: $A \xleftarrow{R} \mathbb{Z}_q^{m \times (n-1)}$ $\quad$ for $m = n \log q$

$\vec{e} \xleftarrow{R} \chi_B^m$

$$C = \underbrace{(A,\ A\tilde{s} + \vec{e}) + \mu G}_{m \times n}$$

output $\underbrace{\hat{C}}_{m \times m \,=\, m \times n\log q}$

Observe that $ct = \hat{C}$ has low norm since it is a $\{0,1\}$-matrix!

Dec$(\vec{s}, \hat{C})$: Compute $\underbrace{\hat{C} \cdot G}_{C} \cdot \vec{s} = C \cdot \vec{s}$

$$= (A, A\tilde{s} + \vec{e}) \begin{pmatrix} \tilde{s} \\ -1 \end{pmatrix} + \mu \cdot G \cdot \vec{s}$$

$$= \mu \cdot G \cdot \vec{s} - \vec{e}$$

if first element is small, output $\mu = 0$. Else, output $\mu = 1$.

why first element? $(\mu \cdot G \cdot \vec{s})_1 = \mu \; (1 \; 0 \ldots 0) \begin{pmatrix} s_1 \\ s_2 \\ \vdots \\ s_{n-1} \\ -1 \end{pmatrix}$

$= \mu \cdot |s_1|$ which is $\Theta(q)$ whp. iff $\mu = 1$

Let's see how this solves our multiplication problem:

Eval("·", $\hat{C}_1$, $\hat{C}_2$): output $\hat{C} = \hat{C}_1 \cdot \hat{C}_2$ ← all $m \times m$ matrices

Need to check $\mathrm{Dec}(\vec{s}, \hat{C}) \stackrel{?}{=} \mu_1 \cdot \mu_2$

Proof:

$$\hat{C}_1 \cdot \underbrace{\hat{C}_2 \cdot G}_{C_2} \cdot \vec{s} = \hat{C}_1 \cdot (C_2 \cdot \vec{s})$$

$$= \hat{C}_1 \cdot (\mu_2 \cdot G \cdot \vec{s} + \vec{e}_2)$$

$$= \mu_2 \cdot \underbrace{\hat{C}_1 \cdot G}_{C_1} \cdot \vec{s} + \hat{C}_1 \cdot \vec{e}_2$$

$$= \mu_2 (\mu_1 \cdot G \cdot \vec{s} + \vec{e}_1) + \hat{C}_1 \cdot \vec{e}_2$$

$$= \mu_1 \mu_2 G \vec{s} + \mu_2 \cdot \vec{e}_1 + \hat{C}_1 \cdot \vec{e}_2$$

Small if $\mu_2 \in \{0,1\}$ — Small since $\|\hat{C}_1\|_\infty = \text{Small}$

What about addition? Could that make noise bigger?

Turns out it's sufficient to support the universal NAND gate:

$$\mathrm{NAND}(a,b) = \mathrm{NOT}(\mathrm{AND}(a,b))$$

Using NAND for other gates:

$\mathrm{NOT}(a) = \mathrm{NAND}(a,a)$

$\mathrm{AND}(a,b) = \mathrm{NOT}(\mathrm{NAND}(a,b))$

$\mathrm{OR}(a,b) = \mathrm{NAND}(\mathrm{NOT}(a), \mathrm{NOT}(b))$

So how to build NAND?

$$\mathrm{Eval}(\mathrm{NAND}, \hat{C}_1, \hat{C}_2): \underbrace{I_{m\times m}}_{\substack{\text{ct has } \mu \text{ added} \\ \text{along its diagonal,} \\ \text{so } 1-\mu \text{ is NOT}}} - \underbrace{\hat{C}_1 \cdot \hat{C}_2}_{\substack{\text{mult over } \{0,1\} \\ \text{is AND}}}$$

Yay! We have constructed an encryption scheme that can compute a universal gate over ciphertexts!

But this is a <u>leveled</u> FHE, so we're not done yet.

Next time we'll see why this is not quite an FHE yet (noise growth), as well as a technique called Bootstrapping that will allow us to get a full FHE scheme.