Section 8

FHE!!

Bootstrapping

- Given (1-hop) (`Gen`,`Enc`,`Dec`,`Eval`) supporting functions

  $f_{c,c'}$`(sk) = Dec(sk,c) nand Dec(sk,c')`

- Define (multi-hop) FHE scheme with `Func = { nand }`

```
Gen'() = (sk,pk) ← Gen()
   ek ← Enc(pk,sk)
   return (sk,(pk,ek))

Enc'((pk,ek),m) = Enc(pk,m)

Eval'((pk,ek),nand,c,c')
   = EvalC(pk,f_{c,c'},ek)
```

LWE Homomorphic Encryption

- Goal: homomorphic evaluation of

  $f_{c,c'}$`(sk) = Dec(sk,c)` **`nand`** `Dec(sk,c')`

- LWE-based cryptosystem
  - Supports bounded depth addition and multiplication
  - Bit operations: `x` **`nand`** `y = 1 - (1-x)\cdot (1-y)`

- Key Switching

```
ek[i] = Enc'(sk',sk[i])
KeySwitch(ek,(a[],b)) =
  Const(b) - ∑i CMul(a[i],ek[i])
```

- Homomorphic evaluation of `Dec'(a,b)= b - Sa`

Not enough

- Key switching only computes the linear part of `Dec`
- We also need to round the result to `decode(b - Sa)`
- Is this really needed?
  - Yes, `b - Sa = (q/p)m + e`
  - Key switching gives a noisy encryption of `(q/p)+e`
  - Without rounding, noise keeps getting bigger

Questions

- Can we express rounding as a polynomial function (mod q)?
- What is the degree of the polynomial?

Error growth and bounded computation

We have seen two methods to multiply ciphertexts:

- Tensor products
  - error growth $\sim \beta \to \beta\sigma$
  - can evaluate arbirary circuits with **multiplicative depth** L
  - even for $L = \log n$, requires superpolynomial modulus $q > \sigma^L \approx n^{O(\log n)}$
- Nested Encryption / Homomorphic Decryption
  - asymmetric error growth: $(m_0, e_0) \times (m_1, e_1) \to m_0 e_1 + e_0 \beta$
  - can evaluate arbitrary multiplication **chains** of L **fresh** encryptions of **binary** messages
  - even for large L, polynomial modulus $q \approx L\beta^2$ is enough

Roadmap

For each multiplication method

1. Describe/analyze a bootstrapping algorithm
2. Homomorphically evaluate the algorithm using an appropriate cryptographic data structure (encrypted accumulator)
3. Implement the cryptographic data structure using LWE

Cryptographic accumulators

- Cryptographic Data Structure `ACC[v]`
  - Holds a value $v \in V$ in encrypted form
  - Input Encryption scheme: `Enc'`
  - Output Encryption scheme: `Enc''`
- Operations on `ACC[v]`
  - Given `Enc'(x)`, update `ACC[v]` $\rightarrow$`ACC[f(v,x)]`
  - Given `ACC[v]`, output `Enc''(f(v))`
- Bootstrapping:
  - Bootstrapping key: `Enc'(s)`
  - Final output: `Enc''(m)`

Boostrapping problem

- Assume $p = 2, m \in \{0, 1\}$
- Decryption Algorithm:
  - Input: $a[1..n] \in \mathbb{Z}_q^n$, $b \in \mathbb{Z}_q$
  - Secret key: $s[1..n] \in \mathbb{Z}^n$
  - Compute $d = b - \sum_i a[i]s[i] + (q/4) \pmod q$
  - Round d to $MSB(d) = \lfloor 2d/q \rfloor$
- Homomorphic Computation:
  - Given Enc`(s[i])`
  - Compute Enc`(MSB(d))`
- Simplifying assumption:
  - $s[i] \in \{0, 1\}$
  - without loss of generality using $(a, 2a, 4a, \ldots)$

Ripple-carry addition

- Standard schoolbook method
  - using binary digits
  - add n numbers at a time
  - *carry* in $\{0, \ldots, n\}$
- Input digits are encrypted

Ripple-carry accumulator

- Parameters:
- Message space `V = {v',...,v''}`
- Input: `Enc'(x)= Enc`$^{\#}$`(x)`
- Output: `Enc''(x)= LWE(x)`
- `ACC[x] = (Enc''("x=v"): v` $\in$ `V)`
    - `Init(v)= ACC[v]`
    - Function application: `f(ACC[v])= ACC[f(v)]`
    - Selection:
      `Enc'(b)? ACC[v0] : ACC[v1] = ACC[b?v0:v1]`
    - Output: `p(ACC[v])= Enc''(p(b))`

Bootstrapping algorithm

```
b+q/4 = ∑_j 2^j b[j]
a[i] = ∑_j 2^j a[i,j]
ACC  ← ACC[0]
for h = 0..k-1
  ACC[x] ← f(ACC[x]) where f(x) = (x/2) + b[h]
  forall i,j
    if (a[i,j] = 1)
      ACC[x + s[i]] ← Enc'(s[i]) ? ACC[x] :
            ACC[x+1]
return (even(ACC[x])) = Enc''(even(x))
```

Carry-save accumulator

- Parameters: bit length `k`
- `ACC[x] = (x0,x1)`
  - `x = x0+x1` (modulo 2^k)
  - `x0[0,..,k-1]` and `x1[0,..,k-1]`
  - redundant representation
- Operations:
  - add `y` to `ACC`
  - compute `MSB(ACC)`

Carry-save addition


```
Add(ACC(x0,x1),y):
  x0'[i]   = (x0[i] + x1[i] + y[i]) mod 2
  x1'[i+1] = (x0[i] + x1[i] + y[i] > 1)
  return ACC(x0',x1')
```

MSB computation

- Standard MSB computation
  - addition `x0+x1` with carry propagation
  - `O(log(k))` depth circuit where `k=log(q)`
- Can also add in `log(k)` depth
  - Compute both `MSB(ACC)` and `MSB(ACC+1)`
  - `ACC[k]`: k-bit accumulator
  - Recursive algorithm: split `ACC[k] = (HiACC[k/2],LoACC[k/2])`

```
MSBs(ACC=(HiACC,LoACC)):
  parallel:
    hi[0,1] = MSBs(HiACC)
    lo[0,1] = MSBs(LoACC)
  out[0] = hi[lo[0]]
  out[1] = hi[lo[1]]
  return out
```

Bootstrapping algorithm

```
ACC[0] = b+q/4
for i=1..n
  ACC[i] = s[i]*a[i]
ACC = Sum(ACC[0],...,ACC[n])
return MSB(ACC)

Sum(ACC[0..n])
  if n=0
    then return ACC[0]
    else h=n/2
         ACC0 = Sum(ACC[0..h-1])
         ACC1 = Sum(ACC[h..n])
         (x0,x1) = ACC1
         return (ACC0 + x0) + x1
```

Summary

Bootstrapping functions can be computed by

1. $O(n \log q)$-long sequence of multiplications, or
2. $\log(n) + \log\log(q)$-depth arithmetic circuits

Error growth:

1. Using LWE $\odot$: final error $\approx O(n) \cdot \beta$
2. Using LWE $\otimes$: final error $\approx \sigma^{\log n + \log\log q} = \sigma^{O(\log n)}$

Parameters $\beta(n), \sigma(n)$: fixed polynomials in n

Modulus:

1. polynomial modulus $q(n) \approx O(n)\beta = n^{O(1)}$
2. quasipolynomial $q(n) = n^{O(\log n)}$

Summary (security)

- Hardness of lattice problems within factor $\gamma \approx q/\beta$
  1. LWE $\odot$: polynomial $\gamma = n^{O(1)}$
  2. LWE $\otimes$: quasipolynomial $\gamma = n^{O(\log n)}$
- Circular security assumption
  - Needed by tensor product multiplication / keyswitching
  - Needed to apply bootstrapping
  - Not needed for leveled homomorphic encryption

Summary (security)

- Hardness of lattice problems within factor $\gamma \approx q/\beta$
  1. LWE $\odot$: polynomial $\gamma = n^{O(1)}$
  2. LWE $\otimes$: quasipolynomial $\gamma = n^{O(\log n)}$
- Circular security assumption
  - Needed by tensor product multiplication / keyswitching
  - Needed to apply bootstrapping
  - Not needed for leveled homomorphic encryption

Question

Remove circular security assumption:

- *Can you build (unbounded) FHE from standard LWE?*
- *Can you build (unbounded) linearly homomorphic HE?*

Efficiency

- Main security parameter $n > 100$ (typically, $n \approx 1000$)
- Modulus $q(n) < 2^n$ has bitsize $\log q < n$
- Assume 1GHz, arithmetic operations modulo q
- Bootstrapping: homomorphically evaluate decryption algorithm (once or twice per gate)

Question

Can you estimate the cost of a single FHE operation?