Functional Bootstrapping: the FHEW approach to Homomorphic Encryption

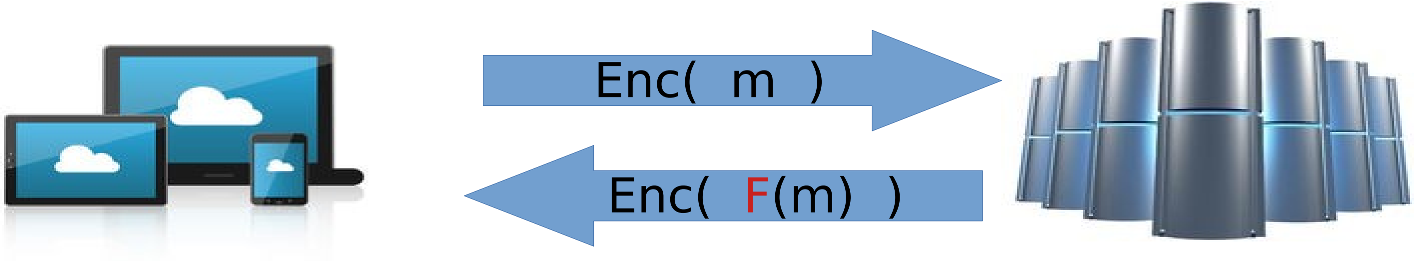
Daniele Micciancio
(UC San Diego)

[May 2024]

Fully Homomorphic Encryption

- Encryption: used to protect data at rest or in transit

- Fully Homomorphic Encryption: supports arbitrary computations (F) on encrypted data



FHE Timeline

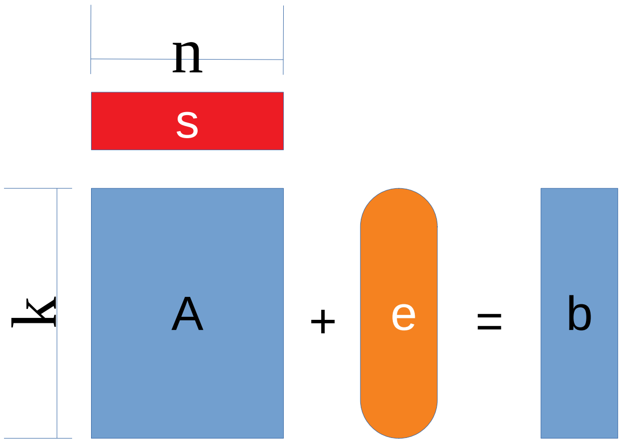
- Bootstrapping: [G'09]
- FHE from (R)LWE
  - [BV'11],[B/FV'12],[BGV'12],[GHS'12],…
- FHE from (R)LWE with polynomial modulus
  - [GSW'13],[BV'14],[AP'14],…
- Functional bootstrapping
  - [DM'15] (FHEW), [CGGI'16],…,[LW'23],[DKM'24]
- Approximate FHE: [CKKS'17],…

This Talk

- High level
  - focus on conceptual ideas
  - very few technical details (read the papers!)
- Describe
  - current state of the art
  - how we got here
  - open problems / research directions

Lattice-based Encryption

- secret key: $sk = s \in Z^n$
- $Enc_s(m) = (A,\ b = As + m\Delta + e) \bmod q$
  - $A \leftarrow$ random matrix (mod q)
  - $e \leftarrow$ random noise ($|e| < \Delta/2$)
- $Dec_s(A,b)$:
  - $c = b - As = \Delta\, m + e$
  - output round(c / Δ) = m
  - Notation: $[[\, c\,]] = \text{round}(\, c / \Delta\,)$

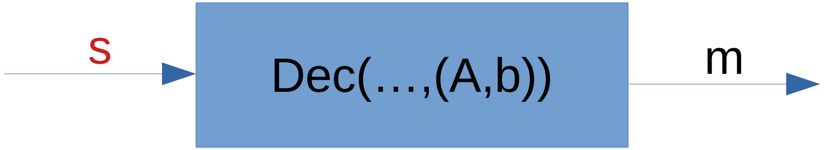


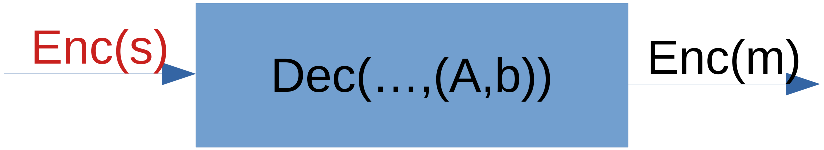
Homomorphic Operations

- $C_0 = Enc_s(m_0) = (A_0, A_0 s + m_0 \Delta + e_0)$
- $C_1 = Enc_s(m_1) = (A_1, A_1 s + m_1 \Delta + e_1)$
- $C_0 + C_1 = ((A_0 + A_1), (A_0 + A_1)s + (m_0 + m_1)\Delta + (e_0 + e_1))$
- Similarly for (tensor) product … but trickier
- If $|e_i| < \Delta/4$, then
  - $|e_0 + e_1| < \Delta/2$
  - $C_0 + C_1$ is a valid encryption of $m_0 + m_1$ (mod $p = q/\Delta$)
- ... but $C_0 + C_1 + C_2$ may not decrypt correctly

Bootstrapping [Gentry'09]

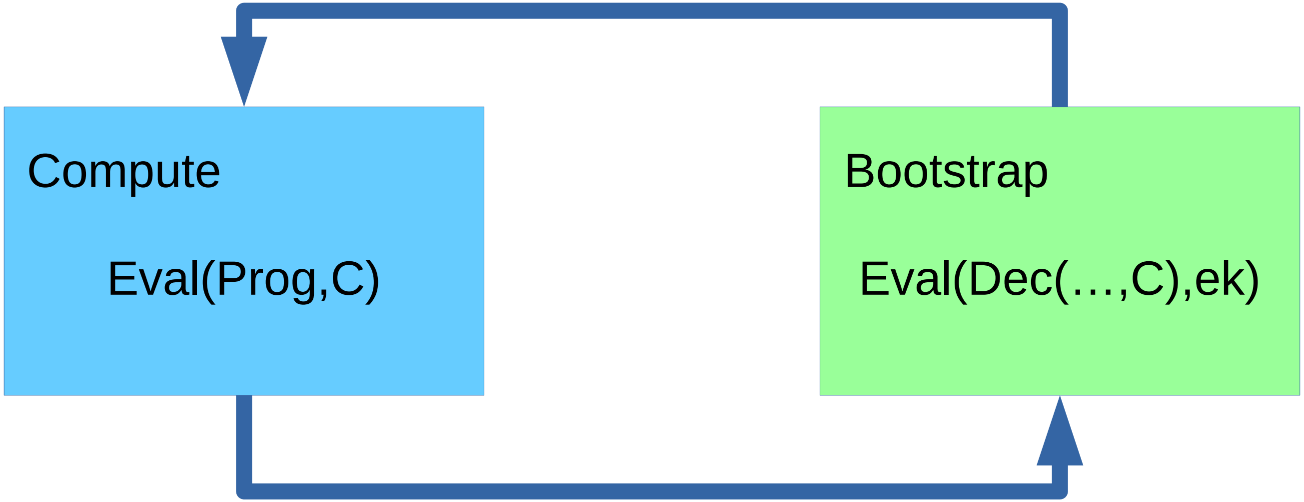
- Decryption removes noise from $(A,b) = \mathrm{Enc}(m)$
  - $\mathrm{Dec}(s,(A,b)) = [[\, b - As \,]] = [[\, \Delta m + e \,]] = m$
  - But Dec requires knowing s!
- Bootstrapping: publish $ek = \mathrm{Enc}(s)$
  - Compute Dec(...,(A,b)) homomorphically on $ek = \mathrm{Enc}(s)$
  - $\mathrm{Boot}((A,b)) = \mathrm{Eval}(\mathrm{Dec}(\ldots,(A,b)), ek)$





Noise growth and FHE

- M = max noise for C to decrypt correctly
- Compute homomorphically until Noise(C) = M
- Apply Boot(C) to reduce noise



Roles of Eval in FHE

- Actual computation on encrypted data
  - application specific, directly exposed to the user
  - Outer Encryption: Enc(.)
- Evaluate Decryption function/bootstrapping
  - More of a bookkeeping task
  - Technically necessary, but unrelated to application
  - Inner encryption: Enc(.)

Traditional approach

- Used by [Gentry,BV,BGV,BFV, etc.]
- Basic operations: {+,*}
  - Enough to describe arbitrary computations
  - Boolean circuits: {xor, /\} = {+, *} mod 2
- Bootstrapping:
  - Express Dec as a arithmetic/boolean circuit
  - Evaluate using the same Enc ≈ Enc
  - Enc, Enc may use different parameters as an optimization

Functional Bootstrapping approach

- Introduced in [DM’15] FHEW, and further developed in FHEW-like schemes like TFHE
- Boot[f]: Enc(m) → Enc(f(m))
  - Boot = Boot[id] : Enc(m) → Enc(m)
- Does Boot[f] already give FHE?
  - No: f is a unary function on fixed message space
  - need at least one binary operation to combine inputs
- Typically enough for Enc to support {+}

Functional Bootstrapping

- New computational model: { +, f }
- Boot[f]: Essentially the same cost as Boot[id]
- Outer scheme: enough for Enc to support {+}
  - Very simple Enc,Dec
  - Very fast Boot[f] = Eval(f(Dec(...)))

Example: FHEW "NAND" circuits

- Encode bits {0,1} as subset of Z_3={0,1,2}
- Addition: {0,1} x {0,1} → {0,1,2}
  - regular addition, does not use reduction mod p
- Let f be the function
  - f(0)=1, f(1)=1, f(2)=0
- NAND(x,y) = f(x+y)
  - x=y=1 iff x+y=2 iff f(x+y)=0
- [DM'15]: FHE bootstrapping in a fraction of a second

Other functions

- majority(x,y,z):
  - p=4, greaterThanOne(x+y+z)
- symmetric boolean function:
  - $f(x_1+\ldots+x_k)$ using p=k
- arbitrary functions:
  - $f(x_1+2x_2+4x_3+8x_4+\ldots)$ using $p=2^k$
  - note: p exponential in k

Putting FHEW into context

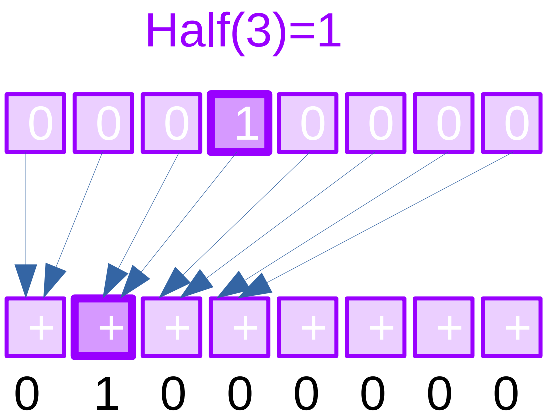
- [GSW’13] new homomorphic multiplication with “asymmetric” error growth
- [BV’14] bootstrapping
  - Express GSW.Dec as a branching program
  - Use GSW to evaluate branching program
- [AlperinSheriff,Peikert’14]
  - Idea: use different Enc for normal Eval and Boot
  - Implement Boot using a scheme optimized for Dec
  - Outer (Enc,Eval) still need to support {+,*}

Homomorphic Decryption

- Constant value: (A,b) mod q
- Encrypted input: ek = Enc(s mod q)
- Computation: s → [[b – As]]
  - **x** = (b-As) linear operation modulo q
  - [[**x**]] non-linear operation: $Z_q \rightarrow Z_p$

AP’14 bootstrapping (basic)

- Let Enc(b) be a “bit” encryption scheme supporting scalar products
- Encrypt (x mod q) as q ciphertexts
  - E[v] = [Enc(0),…,Enc(0),Enc(1),Enc(0),…,Enc(0)]
  - E[(v+w) mod q] = E[v]★E[w]   (convolution)
- Bootstrap using ek[c,i] = E[c*s_i]
  - Initialize Acc := E[b]
  - Iterate: Acc[v] := Acc[v] ★ ek[-A_i,i] = Acc[v - $A_i s_i$]
- Rounding: map Acc[b-As] → Enc([[b – As]])
  - [Enc(?),Enc(?),Enc(?),Enc(?),Enc(?),Enc(?),Enc(?),Enc(?)]



Polynomial Rings (simplified)

- Cyclic $R[n] = Z[X] / (X^n - 1) \approx Z^n$
- BGV/BFV: use $R[n]$ to perform n operations in parallel
- Here: use elements of $R[n]$ to encode Z_n
  - $[0,\ldots,0,1,0,\ldots,0] \rightarrow X^v$
  - $[v+w \bmod n] = [v]\star[w] \rightarrow X^{v+w \bmod n} = X^v X^w$
- $Acc[v] = RLWE(X^v)$, $ek[v] = RGSW[X^v]$
- extract: $Acc[X^v] \rightarrow LWE(f(v))$
- For security use cyclotomic R instead of cyclic

FHEW [Ducas,M.’15]

- Efficient (Ring-based) version of [AP’14]
  - Instead of q LWE ciphertexts [Enc(0),…,Enc(1),…,Enc(0)]
  - Use a single ring ciphertext RLWE(v)=(a,as+e+ΔX^v)
  - coeff(X^v) = (0,…,0,1,0,…,0)
  - Ring dimension n = q
- Functional bootstrapping
  - Instead of rounding [[x]], use an arbitrary function f(x)
  - map Acc[b-as] = LWE(f(b-as))
- [DM’15] (Functional) bootstrapping in fraction of a second

Developments and State of the Art

- Different bootstrapping algorithms
  - [CKKS’16],[LMK+’23]
- Amortized bootstrapping
  - [MS’18,GPvL’23,DKMS’24]
- Expanding the message space
  - [BDF’18],[LW’23] (Note: F.H.Liu, H.Wang)
- Hybrid bootstrapping:
  - [LW’23] (different paper/people: Z.Liu,Y.Wang !)

Bootstrapping Algorithms/Keys

- FHEW/DM: ring version of [AP'14]
  - $ek[i,j]=E(2^i s_j)$ for **{i<lg q, j<n}**
  - Write $a_j = \sum 2^i a_j$
- TFHE/CKKS: variant based on [GINX'16]
  - Write $s_j=\sum 2^i s_{ij}$ with $s_{ij} \in \{0,1\}$, **{i<lg q, j<n}**
  - $ek[i,j]=E(s_{ij})$
  - Better than FHEW/DM when $s_i \in \{0,1\}$ (see ePrint 2020/086)
- [LMK+23]: Best performance using automorphisms

Amortized FHEW bootstrapping

- FHEW: uses $R=Z^n$ to encode Z_n for n=q
  - Very inefficient encoding
  - Sequential bootstrapping: one ciphertext at a time
- [Micciancio,Sorrell'18]:
  - Alternative method to "parallelize" bootstrapping
  - Combine n LWE ciphertexts into a single RLWE
  - Decryption: $b - a \star s$ where a,b,s are in $R=Z^n$
  - $s \in R=Z^n$ is still encrypted as $E(s_i)$ for i=1…n

Amortized bootstrapping: efficiency

- Homomorphic computation:
  - Still operates on scalar values b, a_i, s_i mod q, encrypted as ring elements RGSW(X^c)
  - Improvement is from smaller operation count
- Cost of bootstrapping n ciphertexts
  - Sequential: n scalar products a*s, for total n^2 ops
  - Amortized: 1 convolution a★s, for total O(n log n) ops (potential)
  - [MS'18]: $O(n^{1+\varepsilon})$ ops, for constant ε
  - Amortized cost per input ciphertext: $O(n) \rightarrow O(n^{\varepsilon})$

Amortized bootstrapping in practice

- [MS'18] asymptotic amortized cost $O(n^{\varepsilon}) = c\, n^{\varepsilon}$
  - Only addition in the exponent: $X^{v+w} = X^v X^w$
  - Large hidden constant $c=\exp(-\varepsilon)$
  - Far from practical due to Nussbaumer transform
- [GPvL'23],[DKMS'24] reduce c to $1/\varepsilon$
  - Use automorphisms for twiddle factors: much closer practical
  - Require non-power-of-2 cyclotomic rings
  - Limited improvement over sequential bootstrapping

Expanding the message space

- [DM’15,CKKS’16, etc.]: E(v mod q) using R[q]
  - LWE ciphertext modulus q=Δp
  - E[q] ring dimension q=Δp is linear in p
  - Exponential in |v|=log p
- Increasing p is very inefficient!
- [AP’14]: better encoding for large q

Cycle products

- if $q = p\Delta = p_1 p_2 ... p_k$ use CRT:
  - Instead of $R[q] = R[p_1 p_2 ... p_k]$
  - Use $R[p_1], R[p_2], \ldots, R[p_k]$
  - Final “interpolation” step of computation is still linear in q
- [Bonnoron,Ducas,Fillinger’18]: practical variant
  - Use just the product of two cycles $q=p_1 p_2$,
  - Increases FHEW message from 1-bit to 6-bit words

[Liu,Wang'23 (a)]

- Input LWE: q can be as small as $\sqrt{n}$
- Use ring product R[pq] = (R[q] ★ R[p])
  - $p \approx \sqrt{n}$, ring size $\sqrt{n}*\sqrt{n} = n$
  - each ring element packs $p\approx\sqrt{n}$ values mod q
- Homomorphic product:
  - this is a tensor product, contains unwanted "cross terms"
- Solution: use R[pqr] = R[q]★R[p]★R[r] where $p,r \approx n^{1/4}$
  - Use Homomorphic Trace computation to cancel "cross terms"
  - drawback: can make effective use of only $n^{1/4}$ slots

[Liu,Wang‘23 (b)]

- [LW’23a]: packs $n^{1/4}$ slots in 1 FHEW ciphertext
  - Amortized complexity of bootstrapping: $n^{0.75}$
  - Better than FHEW (n), but worse than [MS’18] n^{ε}
- [LW’23b]: combines [LW’23a]+[MS’18]
  - amortized complexity: polylog(n)!
  - inherits Nussbaumer transform from [MS’18]
  - great in theory, but far from practical

Practical considerations

- [BDF’18],[LW’23ab],[MS’18/GPvL’23/DKM’24]
  - improve FHEW in interesting directions
  - mostly theoretical, poor performance in practice
  - require rings R[q] for q other than 2^n
- Arithmetic in non-powers-of-two rings
  - not well supported by libraries
  - seems 10x slower or worse in practice

Hybrid approach

- [Liu,Wang'23] Use BFV to bootstrap FHEW
  - express FHEW bootstrap as a degree q polynomial
  - evaluate polynomial using BFV SIMD {+,*} operations
- Performance
  - n parallel bootstrapping thanks to BFV SIMD
  - supports FHEW functional bootstrapping
  - 7ms per bootstrapping!

Conclusion

- Functional bootstrapping
  - powerful model of homomorphic encryption
  - Many interesting theoretical developments
  - Best performance: hybrid with BGV/BFV (superpoly)
  - Explored also in pure BGV/BFV/CKKS setting (superpoly)
- Research directions
  - improve practicality of FHEW-like functional boostrapping with poly(n) modulus
  - needed: practical support of arbitrary cyclotomic rings