Homomorphic Encryption
Tutorial

Shai Halevi — IBM
CRYPTO 2011

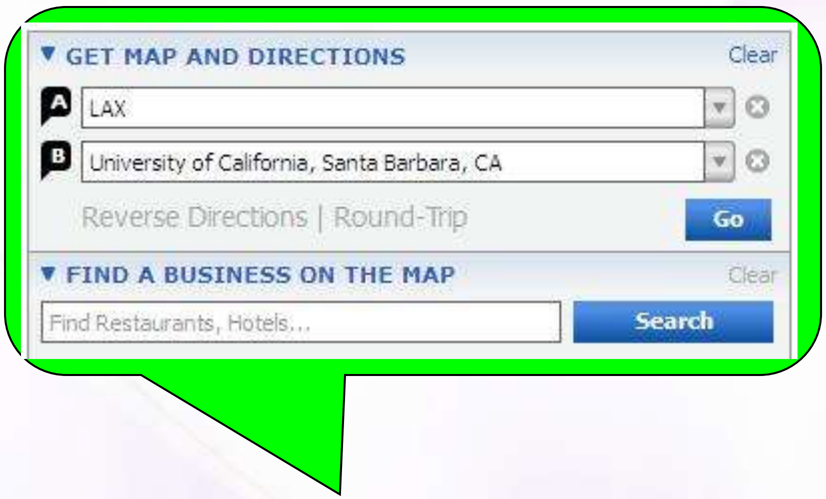
Computing on Encrypted Data

- Wouldn't it be nice to be able to…
  - Encrypt my data before sending to the cloud
  - While still allowing the cloud to search/sort/edit/… this data on my behalf
  - Keeping the data in the cloud in encrypted form
    - Without needing to ship it back and forth to be decrypted

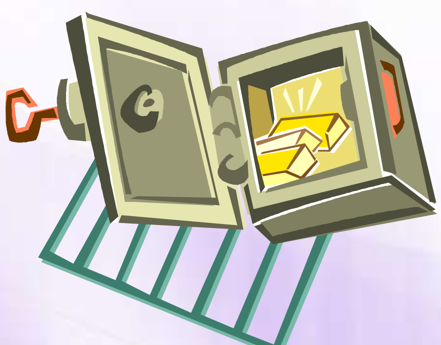
Computing on Encrypted Data

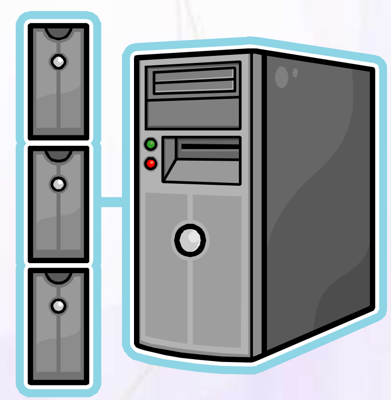
- Wouldn’t it be nice to be able to…
  - Encrypt my queries to the cloud
  - While still allowing the cloud to process them
  - Cloud returns encrypted answers
    - that I can decrypt

Computing on Encrypted Data

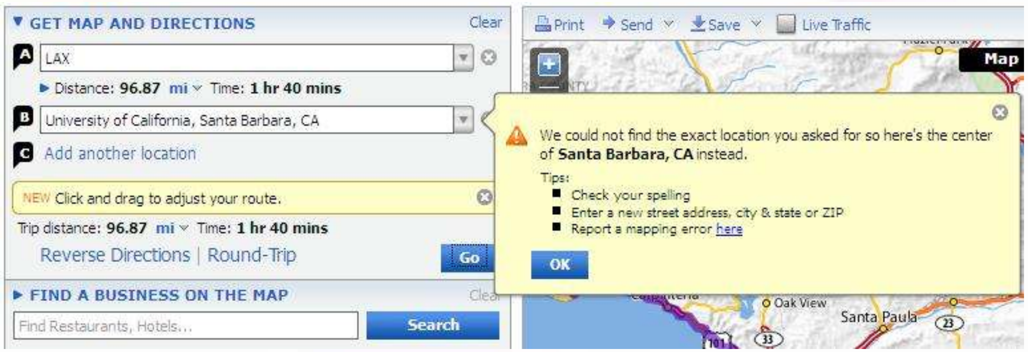


$skj#hS28ksytA@ ...

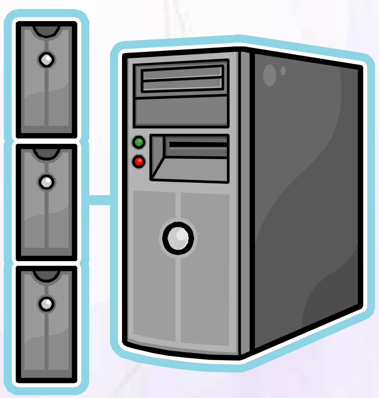




Computing on Encrypted Data



$kjh9*mslt@na0
&maXxjq02bflx
m^00a2nm5,A4.
pE.abxp3m58bsa
(3saM%w,snanba
nq~mD=3akm2,A
Z,ltnhde83|3mz{n
dewiunb4]gnbTa*
kjew^bwJ^mdns0



Organization of the Tutorial

- Two parts with a 10-minute break in between
- First part quite high-level
  - Lots of pictures/animations on the slides
  - Covers the [Gentry 2009] blueprint
- Second part more algebraic
  - Lots of formulas on the slides
  - Covers newer constructions [GH’11,BV’11,BGV’11] (April 2011 and on)

Some Notations

- An encryption scheme: (KeyGen, Enc, Dec)
  - Plaintext-space = {0,1}
  - $(pk,sk) \leftarrow$ KeyGen(\$), $c \leftarrow \mathrm{Enc}_{pk}(b)$, $b \leftarrow \mathrm{Dec}_{sk}(c)$
- Semantic security [GM'84]:

$$(pk, \mathrm{Enc}_{pk}(0)) \approx (pk, \mathrm{Enc}_{pk}(1))$$

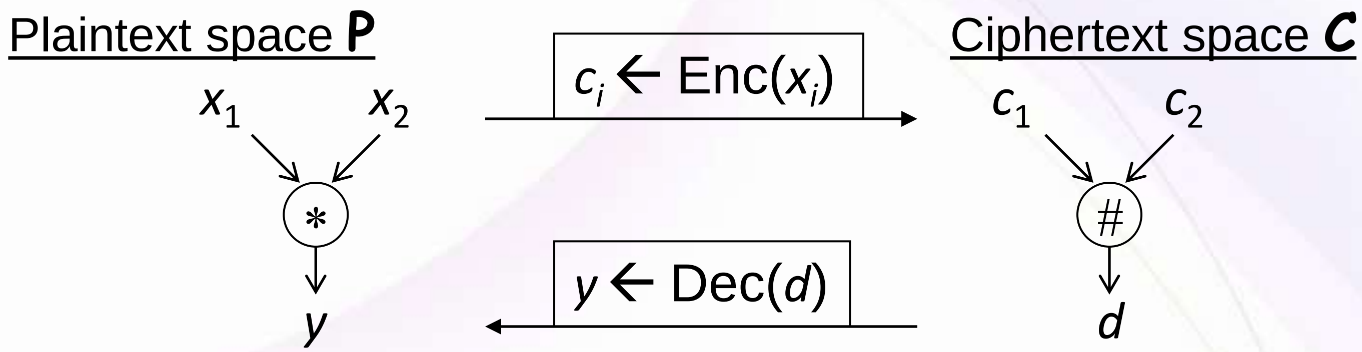
$\approx$ means indistinguishable by efficient algorithms

Homomorphic Encryption (FHE)

- H = {KeyGen, Enc, Dec, Eval}

  $c^* \leftarrow \mathrm{Eval}_{pk}(f, \mathbf{c})$
- Homomorphic: $\mathrm{Dec}_{sk}(\underbrace{\mathrm{Eval}_{pk}(f, \mathrm{Enc}_{pk}(x))}_{c^*}) = f(x)$
  - c^* may not look like a "fresh" ciphertext
  - As long as it decrypts to $f(x)$
- Compact: Decrypting c^* easier than computing f
  - Otherwise we could use $\mathrm{Eval}_{pk}(f, \mathbf{c}) = (f, \mathbf{c})$ and $\mathrm{Dec}_{sk}(f, \mathbf{c}) = f(\mathrm{Dec}_{sk}(\mathbf{c}))$
  - $|c^*|$ independent of the complexity of f

Privacy Homomorphisms [RAD78]



Some examples:

- "Raw RSA": $c \leftarrow x^e \bmod N$ ($x \leftarrow c^d \bmod N$)
  - $x_1^e \times x_2^e = (x_1 \times x_2)^e \bmod N$
- GM84: $\mathrm{Enc}(0) \in_R \mathrm{QR}$, $\mathrm{Enc}(1) \in_R \mathrm{QNR}$ (in Z_N^*)
  - $\mathrm{Enc}(b_1) \times \mathrm{Enc}(b_2) = \mathrm{Enc}(b_1 \oplus b_2) \bmod N$

More Privacy Homomorphisms

- Mult-mod-p [ElGamal’84]
- Add-mod-N [Pallier’98]
- Quadratic-polys mod p [BGN’06]
- Branching programs [IP’07]
- A different type of solution for any circuit [Yao’82,...]
  - Not compact, ciphertext grows with circuit complexity
  - Also NC1 circuits [SYY’00]

(x,+)-Homomorphic Encryption

It will be really nice to have…

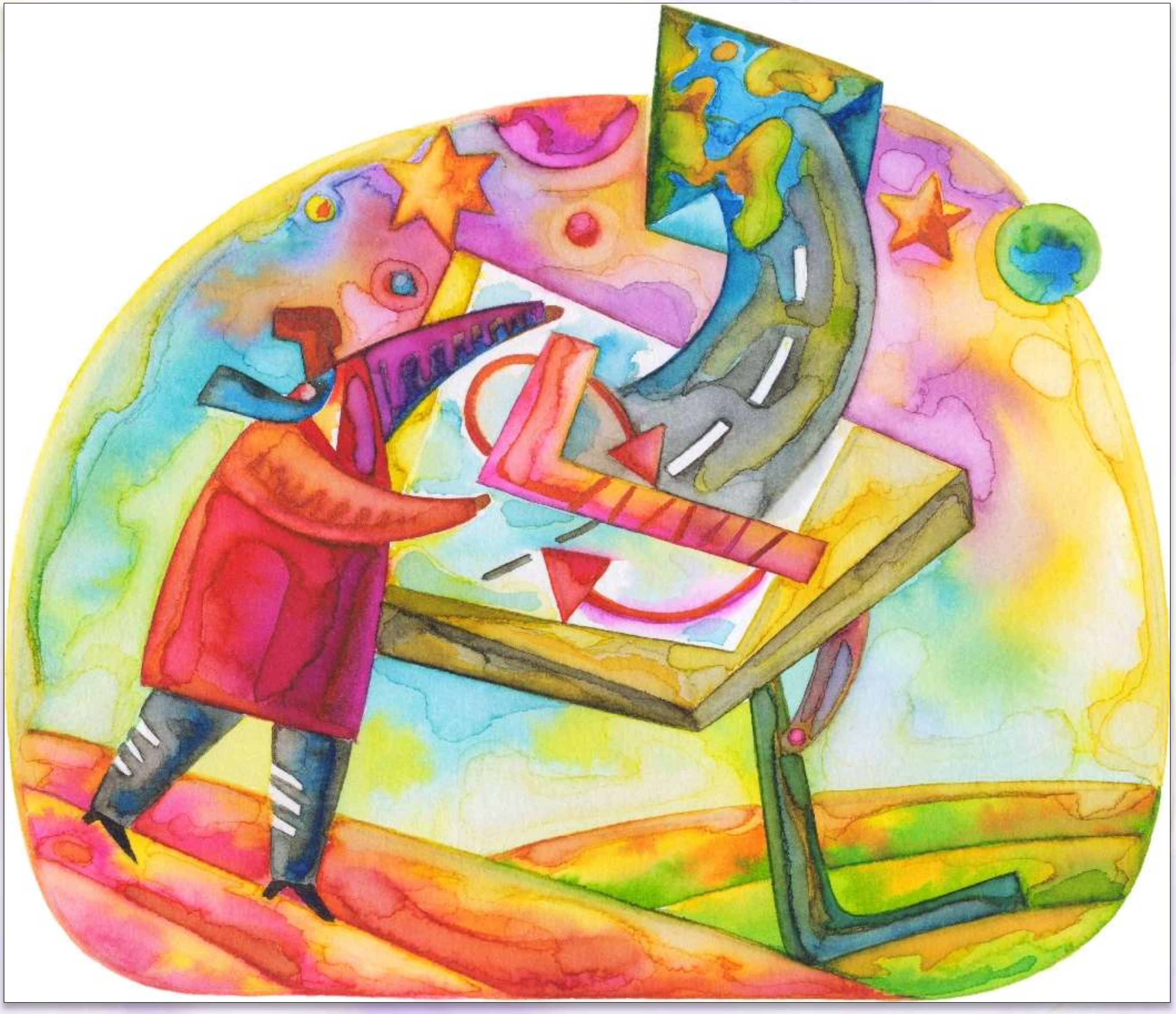
- Plaintext space Z_2 (w/ ops +,x)
- Ciphertexts live in algebraic ring R (w/ ops +,x)
- Homomorphic for both + and x
  - Enc(b_1) + Enc(b_2) in R = Enc(b_1+ b_2 mod 2)
  - Enc(b_1) x Enc(b_2) in R = Enc(b_1 x b_2 mod 2)

➔Can compute any function on the encryptions

- Since every binary function is a polynomial

- Won't get exactly this, but it's a good motivation

The [Gentry 2009] Blueprint



The [Gentry 2009] blueprint

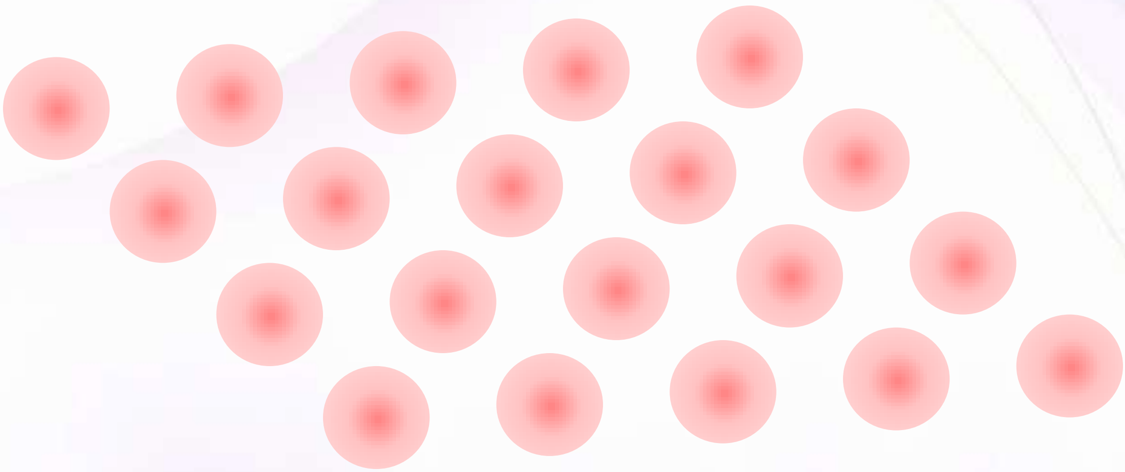
Evaluate any function in four "easy" steps

- Step 1: Encryption from linear ECCs
  - Additive homomorphism
- Step 2: ECC lives inside a ring
  - Also multiplicative homomorphism
  - But only for a few operations (low-degree poly's)
- Step 3: Bootstrapping
  - Few ops (but not too few) → any number of ops
- Step 4: Everything else
  - "Squashing" and other fun activities

Error-Correcting Codes
(not Elliptic-Curve Cryptography)

Step 1: Encryption from Linear ECCs

- For “random looking” codes, hard to distinguish close/far from code

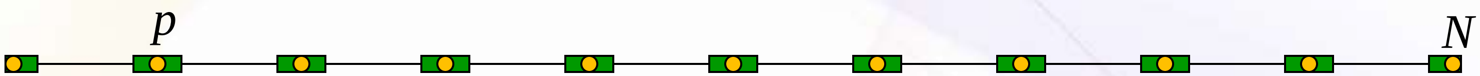


- Many cryptosystems built on this hardness
  - E.g., [McEliece’78, AD’97, GGH’97, R’03,…]

Step 1: Encryption from Linear ECCs

- KeyGen: choose a "random" *Code*
  - Secret key: "good representation" of *Code*
    - Allows correction of "large" errors
  - Public key: "bad representation" of *Code*
    - Can generate "random code-words"
    - Hard to distinguish close/far from the code
- Enc(0): a word close to *Code*
- Enc(1): a random word
  - Far from *Code* (with high probability)

Example: Integers mod p [vDGHV 2010]

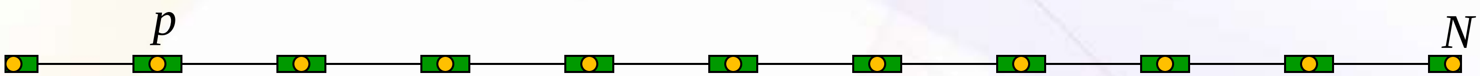


- Code determined by a secret integer p
  - Codewords: multiples of p
- Good representation: p itself
- Bad representation:
  - $N = pq$, and also many $x_i = pq_i + r_i$

$r_i << p$

- Enc(0): subset-sum(x_i's)+r mod N
  - r is new noise, chosen by encryptor
- Enc(1): random integer mod N

A Different Input Encoding



$x_i = pq_i + r_i$

- Both Enc(0), Enc(1) close to the code
  - Enc(0): distance to code is even
  - Enc(1): distance to code is odd
  - Security unaffected when p is odd

Plaintext encoded in the "noise"

- In our example of integers mod p:
  - $\text{Enc}(b) = 2(r + \text{subset-sum}(x_i\text{'s})) + b \bmod N$
    $= \kappa p + \boxed{2(r + \text{subset-sum}(r_i\text{'s})) + b}$

much smaller than p/2

  - $\text{Dec}(c) = (c \bmod p) \bmod 2$

Additive Homomorphism

- $c_1 + c_2 = (\text{codeword}_1 + \text{codeword}_2)$
  $+ \; (2r_1 + b_1) + (2r_2 + b_2)$
  - $\text{codeword}_1 + \text{codeword}_2 \in$ *Code*
  - $(2r_1 + b_1) + (2r_2 + b_2) = 2(r_1 + r_2) + b_1 + b_2$ is still small
- If $2(r_1 + r_2) + b_1 + b_2$ < min-dist/2, then
  dist($c_1 + c_2$, *Code*) = $2(r_1 + r_2) + b_1 + b_2$
  ➔ dist($c_1 + c_2$, *Code*) $\equiv b_1 + b_2$ (mod 2)
- Additively-homomorphic while close to *Code*

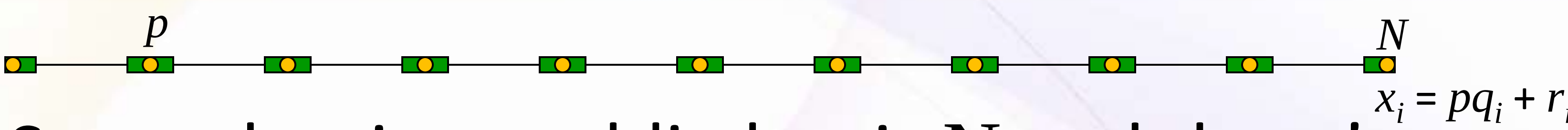
Step 2: *Code* Lives in a *Ring*

- What happens when multiplying in *Ring*:
  - $c_1 \cdot c_2$ = (codeword$_1$+$2r_1$+b_1)·(codeword$_2$+$2r_2$+b_2)
    = codeword$_1 \cdot X$ + $Y \cdot$codeword$_2$
    + $(2r_1+b_1)\cdot(2r_2+b_2)$
- If:
  - codeword$_1 \cdot X$ + $Y \cdot$codeword$_2$ $\in$ *Code*
  - $(2r_1+b_1)\cdot(2r_2+b_2) < \text{min-dist}/2$
- Then
  - dist($c_1 c_2$, *Code*) = $(2r_1+b_1)\cdot(2r_2+b_2) = b_1 \cdot b_2 \bmod 2$

Code is an *ideal*

Product in *Ring* of small elements is small

Example: Integers mod p [vDGHV 2010]



- Secret-key is p, public-key is N and the x_i's
- $c_i = \text{Enc}_{pk}(b_i) = 2(r+\text{subset-sum}(x_i\text{'s})) + b \bmod N$
  $= k_i p + \boxed{2r_i+b_i}$
  - $\text{Dec}_{sk}(c_i) = (c_i \bmod p) \bmod 2$
- $c_1+c_2 \bmod N = (k_1p+2r_1+b_1)+(k_2p+2r_2+b_2) - kqp$
  $= k'p + \boxed{2(r_1+r_2) + (b_1+b_2)}$
- $c_1c_2 \bmod N = (k_1p+2r_1+b_1)(k_2p+2r_2+b_2) - kqp$
  $= k'p + \boxed{2(2r_1r_2+r_1b_2+r_2b_1)+b_1b_2}$
- Additive, multiplicative homomorphism
  - As long as noise $< p/2$

Summary Up To Now

- We need a linear error-correcting code $\mathcal{C}$
  - With “good” and “bad” representations
  - $\mathcal{C}$ lives inside an algebraic ring R
  - $\mathcal{C}$ is an ideal in R
  - Sum, product of small elements in R is still small
- Can find such codes in Euclidean space
  - Often associated with lattices
- Then we get a “somewhat homomorphic” encryption, supporting low-degree polynomials
  - Homomorphism while close to the code

Instantiations

- [G 2009] Polynomial Rings
  - Security based on hardness of “Bounded-Distance Decoding” in ideal lattices
- [vDGHV 2010] Integer Ring
  - Security based on hardness of the “approximate-GCD” problem
- [GHV 2010] Matrix Rings (only partial solution)
  - Only qudratic polynomials, security based on hardness of “Learning with Errors” (LWE)
- [BV 2011a] Polynomial Rings
  - Security based on “ring LWE”

Step 3: Bootstrapping

- So far, can evaluate low-degree polynomials

x_1

x_2

…

x_t

P(x_1, x_2 ,…, x_t)

Step 3: Bootstrapping

- So far, can evaluate low-degree polynomials

x_1
x_2
…
x_t

P

$P(x_1, x_2, \ldots, x_t)$

- Can eval $y = P(x_1, x_2 \ldots, x_n)$ when x_i's are "fresh"
- But y is an "evaluated ciphertext"
  - Can still be decrypted
  - But eval $Q(y)$ will increase noise too much
- "Somewhat Homomorphic" encryption (SWHE)

Step 3: Bootstrapping

- So far, can evaluate low-degree polynomials

x_1
x_2
…
x_t

$P(x_1, x_2, \ldots, x_t)$

- Bootstrapping to handle higher degrees
  - We have a noisy evaluated ciphertext y
  - Want to get another y with less noise

Step 3: Bootstrapping

- For ciphertext c, consider $\mathbf{D}_c(sk)$ = Dec$_{sk}(c)$
  - Hope: $D_c(*)$ is a low-degree polynomial in sk
- Include in the public key also Enc$_{pk}(sk)$

Requires "circular security"

c y

sk_1
sk_2
…
sk_n

c' Dc(sk)
= Dec$_{sk}(c)$ = y

- Homomorphic computation applied only to the "fresh" encryption of sk

Step 3: Bootstrapping

- Similarly define $\mathbf{M}_{c_1,c_2}(sk) = \mathrm{Dec}_{sk}(c_1)\cdot\mathrm{Dec}_{sk}(c_1)$

y_1 y_2

c_1 c_2

sk_1

sk_2

…

sk_n

c'

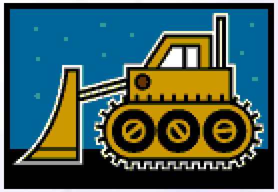
$\mathrm{M}_{c_1,c_2}(sk)$

$= \mathrm{Dec}_{sk}(c_1) \times \mathrm{Dec}_{sk}(c_2) = y_1 \times y_2$

- Homomorphic computation applied only to the “fresh” encryption of *sk*

Step 4: Everything Else

- Cryptosystems from [G'09, vDGHV'10, BG'11a] cannot handle their own decryption
- Tricks to "squash" the decryption procedure, making it low-degree
  - Nontrivial, requires putting more information about the secret key in the public key
  - Requires yet another assumption, namely hardness of the Sparse-Subset-Sum Problem (SSSP)
  - I will not talk about squashing here

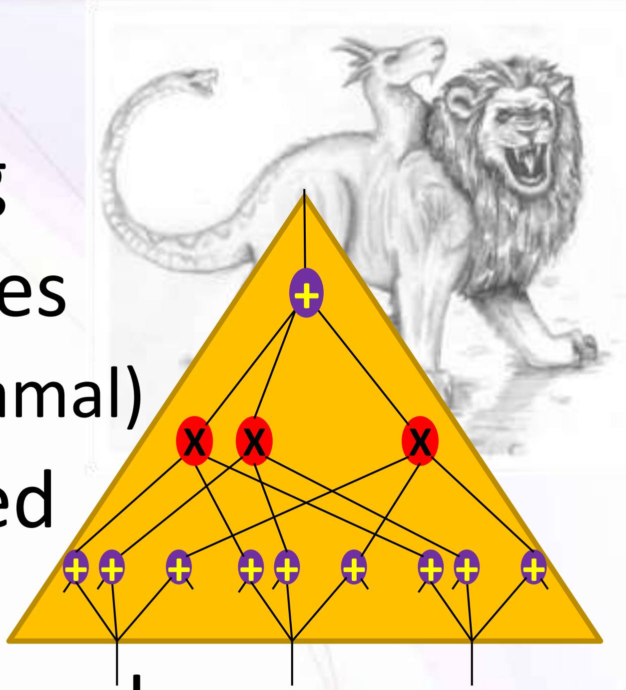


Performance of Blueprint

- SWHE schemes may be reasonable
- But bootstrapping is inherently inefficient
  - Homomorphic decryption for each multiplication
  - Asymptotically, overhead of at least $\tilde{O}(\lambda^{3.5})$
- Best implementation so far is [GH 2011a]
  - Implemented a variant of [Gentry 2009]
  - Public key size ~ 2GB
  - Bootstrapping takes 3-30 minutes
- Similar performance also in [CMNT 2011]
  - Implemented the scheme from [vDGHV'10]

Beyond the Blueprint

Chimeric HE [GH 2011b]



- Bootstrapping without squashing
- Hybrid of SWHE and MHE schemes
  - MHE = Multiplicative HE (e.g., Elgamal)
- Express decryption as a "restricted depth-3" $\Sigma\Pi\Sigma$ arithmetic circuit
- Switch to MHE for the middle Π level
  - All necessary MHE ciphertexts found in public key
- Translate back to SWHE for the top Σ level
  - SWHE evaluates MHE decryption, not own decryption
- No need for squashing, SSSP

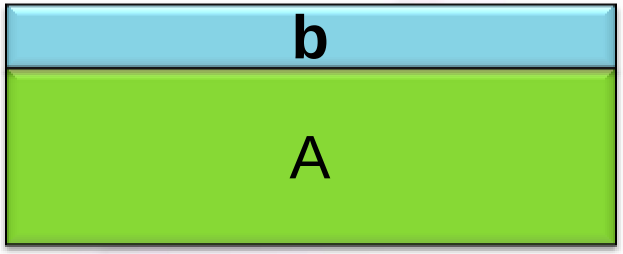
[Brakerski-Vaikuntanathan 2011b]

- FHE without squashing, security based on Learning-with-Errors (LWE), or ring-LWE
- Main innovation: multiplicative homomorphism without a ring structure
- A host of new techniques/tricks, can be used for further improvements

Learning with Errors (LWE) [Regev 2005]

Hard to solve linear equations with noise

- Given:



$\in \mathbf{Z}_q{}^m$

$\in_R \mathbf{Z}_q{}^{nxm}$

decide if

  - **b** is a random vector in $\mathbf{Z}_q{}^m$, or
  - **b** is close to the row-space of A (distance < βq)
    - $\boldsymbol{b} = \boldsymbol{s}\mathbf{A} + \boldsymbol{e}$ for random $\boldsymbol{s} \in \mathbf{Z}_q{}^n$ and random short $\boldsymbol{e} \in \mathbf{Z}_q{}^m$

- Parameters: n (dim), $q \geq \text{poly}(n)$ (modulus), $\beta \leq 1/\text{poly}(n)$ (noise magnitude), $m = \text{poly}(n)$

[Regev'05, Peikert'09]: As hard as some worst-case lattice problems in dim n (for certain range of params)

The [BV'11b] Construction

- Bit-by-bit encryption, plaintext is a bit b
- Think of it as symmetric encryption for now
- Secret-key $\boldsymbol{s}$, ciphertext $\boldsymbol{c}$, are vectors in $\mathbf{Z}_q{}^n$
  - Simplifying convention: $\boldsymbol{s}_1 = 1$, i.e., $\boldsymbol{s} = (1|\boldsymbol{t})$
- Decryption is **b=(<$\boldsymbol{s}$,$\boldsymbol{c}$> mod q) mod 2**
  - (<$\boldsymbol{s}$,$\boldsymbol{c}$> mod q) is small, absolute value $\le \beta q$
- In other words:

mod q maps to [-q/2, q/2]

  - Ciphertexts are "close" to space orthogonal to $\boldsymbol{s}$
  - Plaintext encoded in parity of "distance"
    - "distance" is the size of (<$\boldsymbol{s}$,$\boldsymbol{c}$> mod q)

Homomorphism

- This is an instance of encryption from linear ECCs, additive homomorphism is "for free"
  - As long as things remain close to the code
- But how to multiply?
  - Ciphertexts are vectors, not ring elements
- Tensor product (??) $\boxed{M = \boldsymbol{u}\otimes\boldsymbol{v},\ M_{ij} = u_i \cdot v_j \bmod q}$
  - Can decrypt M(!), $\boldsymbol{s}(\boldsymbol{u}\otimes\boldsymbol{v})\boldsymbol{s}^t = \langle\boldsymbol{s},\boldsymbol{u}\rangle\cdot\langle\boldsymbol{s},\boldsymbol{v}\rangle \pmod q$
  - If no wraparound then $(\boldsymbol{s}(\boldsymbol{u}\otimes\boldsymbol{v})\boldsymbol{s}^t \bmod q) = (\langle\boldsymbol{s},\boldsymbol{u}\rangle \bmod q)\cdot(\langle\boldsymbol{s},\boldsymbol{v}\rangle \bmod q)$
  - So $(\boldsymbol{s}(\boldsymbol{u}\otimes\boldsymbol{v})\boldsymbol{s}^t \bmod q) \bmod 2 = \mathrm{Dec}_{\boldsymbol{s}}(u)\cdot\mathrm{Dec}_{\boldsymbol{s}}(v)$

Multiplying More than Once?

- $\boldsymbol{s}(\boldsymbol{u}\otimes\boldsymbol{v})\boldsymbol{s}^t$ is a bilinear form in $\boldsymbol{s}$, so linear in $\boldsymbol{s}\otimes\boldsymbol{s}$
  - Opening $\boldsymbol{u}\otimes\boldsymbol{v}$, $\boldsymbol{s}\otimes\boldsymbol{s}$ into vectors, we get $\boldsymbol{s}(\boldsymbol{u}\otimes\boldsymbol{v})\boldsymbol{s}^t = \langle \mathrm{vec}(\boldsymbol{s}\otimes\boldsymbol{s}), \mathrm{vec}(\boldsymbol{u}\otimes\boldsymbol{v})\rangle$
- Denote $\boldsymbol{s}^*=\mathrm{vec}(\boldsymbol{s}\otimes\boldsymbol{s})$, $\boldsymbol{c}^*=\mathrm{vec}(\boldsymbol{u}\otimes\boldsymbol{v})$, then:
  - $\mathrm{Dec}_{\boldsymbol{s}^*}(\boldsymbol{c}^*) = (\langle \boldsymbol{s}^*, \boldsymbol{c}^*\rangle \bmod q) \bmod 2$
  - $\langle \boldsymbol{s}^*, \boldsymbol{c}^*\rangle \bmod q$ is still quite small, $\leq (\beta q)^2 \ll q$
- We can repeat the process
  - But dimension is squared, $n \rightarrow n^2 \rightarrow n^4 \rightarrow n^8 \ldots$ so can repeat only a constant number of times

Reducing the Dimension

- We have an "extended ciphertext" $\boldsymbol{c}^*$ with respect to "extended secret key" $\boldsymbol{s}^*=\text{vec}(\boldsymbol{s}\otimes\boldsymbol{s})$
- Want a low-dimension ciphertext $\boldsymbol{c}'$ with respect to a "standard secret key" $\boldsymbol{s}'$
  - Maybe $\boldsymbol{s}'=\boldsymbol{s}$, maybe not
- Key idea: publish "an encryption" of $\boldsymbol{s}^*$ under $\boldsymbol{s}'$ to enable the translation
  - Hopefully just a matrix $\text{M}(\boldsymbol{s}^*\rightarrow\boldsymbol{s}')\in\mathbf{Z}_q^{\dim(\boldsymbol{s}')\text{x}\dim(\boldsymbol{s}^*)}$, so that $\boldsymbol{c}' = \text{M}\cdot\boldsymbol{c}^*\in\mathbf{Z}_q^{\dim(\boldsymbol{s}')}$

An Attempt that Almost Works

$$M=\begin{bmatrix} \mathbf{b} \\ A \end{bmatrix} \begin{array}{l} = -t'A+(2e+s^*) \bmod q \\ \in_R \mathbf{Z}_q^{\dim(t') \text{ x } \dim(s^*)} \end{array}$$

e is short

- Recall $s'=(1|t')$, so $s'M = t'A+b = 2e+s^*$
- Let $c' = M \cdot c^* \in \mathbf{Z}_q^{\dim(s')}$, then mod q we have:

  $<s',c'> \equiv s'Mc^* \equiv <2e+s^*,c^*> \equiv <s^*,c^*>+2<e,c^*>$
- *If only c^* was short*, then $2<e,c^*>$ was small, so $(<2e+s^*,c^*> \bmod q) = (<s^*,c^*> \bmod q)+2<e,c^*>$
  - Hence $(<s',c'> \bmod q) \equiv (<s^*,c^*> \bmod q) \pmod 2$
  - So $\mathrm{Dec}_{s'}(c') = \mathrm{Dec}_{s^*}(c^*)$

Can we Make $\boldsymbol{c}^*$ Short?

- Want to "represent" the long vector $\boldsymbol{c}^*$ by some short vector $\boldsymbol{c}'$, perhaps in higher dimension
- Example: $\boldsymbol{c}^*$ =(76329, 31692, 43870)
  - l_2-norm ~ 90000

represented by $\boldsymbol{c}'$=(7,6,3,2,9, 3,1,6,9,2, 4,3,8,7,0)
  - l_2-norm only ~ 21
- Later we will use binary rather than decimal
- Note that we have a "linear relation":

$$\boldsymbol{c}^* = 10^4 \cdot \boldsymbol{c}'_{1,6,11} + \cdots + 10 \cdot \boldsymbol{c}'_{4,9,14} + \boldsymbol{c}'_{5,10,15}$$

Can we Make c^* Short?

- Denote $\boldsymbol{c}^*=(c_1^*, \ldots, c_k^*)$, i.e., c_i^* is the i'th entry
- Let $c_{il}^* \ldots c_{i0}^*$ be binary representation of c_i^*
  - $c_i^* = \sum_{j=0}^{l} 2^j c_{ij}^*$
- Let $\boldsymbol{b}_j$ be the vector of j'th bits $\boldsymbol{b}_j=(c_{1j}^*, \ldots, c_{kj}^*)$
  - so $\boldsymbol{c}^*= \sum_{j=0}^{l} 2^j \boldsymbol{b}_j$, and $\langle \boldsymbol{s}^*,\boldsymbol{c}^* \rangle= \sum_{j=0}^{l} 2^j \langle \boldsymbol{s}^*, \boldsymbol{b}_j \rangle$
- Let $\boldsymbol{s}^{**}=\text{PowersOf2}_q(\boldsymbol{s}^*)= (\boldsymbol{s}^*|2\boldsymbol{s}^*|4\boldsymbol{s}^*|\ldots|2^l\boldsymbol{s}^*) \bmod q$, and $\boldsymbol{c}^{**}=\text{BitDecomp}(\boldsymbol{c}^*) = (\boldsymbol{b}_0|\boldsymbol{b}_1|\boldsymbol{b}_2|\ldots|\boldsymbol{b}_l)$
- Then $\langle \boldsymbol{s}^{**},\boldsymbol{c}^{**} \rangle \equiv \langle \boldsymbol{s}^*,\boldsymbol{c}^* \rangle \pmod q$
- $\boldsymbol{c}^{**}$ is short (in l_2-norm), it is a 0-1 vector

Dimension Reduction (Key-Switching)

- Publish the matrix $M(\boldsymbol{s}^{**} \rightarrow \boldsymbol{s}') \in \mathbf{Z}_q^{\dim(\boldsymbol{s}') \times \dim(\boldsymbol{s}^{**})}$
- Given the expanded ciphertext $\boldsymbol{c}^*$
  - Compute the "doubly expanded" $\boldsymbol{c}^{**}$
  - Set $\boldsymbol{c}' = M \cdot \boldsymbol{c}^{**} \bmod q$
- We know that $\langle \boldsymbol{s}^{**}, \boldsymbol{c}^{**} \rangle \equiv \langle \boldsymbol{s}^*, \boldsymbol{c}^* \rangle \pmod{q}$
- Also $\langle \boldsymbol{s}', \boldsymbol{c}' \rangle \equiv \langle \boldsymbol{s}^{**}, \boldsymbol{c}^{**} \rangle + 2\langle \boldsymbol{e}, \boldsymbol{c}^{**} \rangle \pmod{q}$
- ($\langle \boldsymbol{s}^*, \boldsymbol{c}^* \rangle$ mod q) is small and so is $2\langle \boldsymbol{e}, \boldsymbol{c}^{**} \rangle$ hence $(\langle \boldsymbol{s}', \boldsymbol{c}' \rangle \bmod q) = (\langle \boldsymbol{s}^*, \boldsymbol{c}^* \rangle + 2\langle \boldsymbol{e}, \boldsymbol{c}^{**} \rangle \bmod q)$
  - Last equality is over the integers

$\rightarrow$ $\mathrm{Dec}_{\boldsymbol{s}'}(\boldsymbol{c}') = \mathrm{Dec}_{\boldsymbol{s}^*}(\boldsymbol{c}^*)$

Security

$$M(s^* \to s') = \begin{bmatrix} -\boldsymbol{t'}\mathrm{A}+2\boldsymbol{e}+\boldsymbol{s}^* \\ \mathrm{A} \end{bmatrix}$$

- Under LWE, cannot tell $M(\boldsymbol{s}^* \to \boldsymbol{s}')$ from random
  - Even if you know $\boldsymbol{s}^*$ (but not $\boldsymbol{s}'$)
  - Assuming q is odd

Pf: if $(\mathrm{A}, \boldsymbol{r}) \approx (\mathrm{A}, \boldsymbol{t}\mathrm{A}+\boldsymbol{e})$ then $(2\mathrm{A}, 2\mathrm{r}) \approx (2\mathrm{A}, 2\boldsymbol{t}\mathrm{A}+2\boldsymbol{e})$

- For odd q: $(2\mathrm{A}, 2\boldsymbol{r}) \equiv (\mathrm{A}, \boldsymbol{r})$,
  $(2\mathrm{A}, 2\boldsymbol{t}\mathrm{A}+2\boldsymbol{e}) \equiv (\mathrm{A}, \boldsymbol{t}\mathrm{A}+2\boldsymbol{e})$
  - $\equiv$ means that these distributions are identical
- We get $(\mathrm{A}, \boldsymbol{r}) \approx (\mathrm{A}, \boldsymbol{t}\mathrm{A}+2\boldsymbol{e})$
- It follows that $(\mathrm{A}, \boldsymbol{r}) \equiv (\mathrm{A}, \boldsymbol{r}+\boldsymbol{s}^*) \approx (\mathrm{A}, \boldsymbol{t}\mathrm{A}+2\boldsymbol{e}+\boldsymbol{s}^*)$

The [BV'11b] "Leveled SWHE"

(Key-size ≥linear in depth of circuits to evaluate)

- KeyGen: choose random $\boldsymbol{s}_0, \boldsymbol{s}_1, \ldots, \boldsymbol{s}_d \in \mathbf{Z}_q^n$
  - First entry in each $\boldsymbol{s}_i$ is 1
  - Public key has matrices $M_0 = M(0 \rightarrow \boldsymbol{s}_0)$ and $M_{i+1} = M(\boldsymbol{s}_i^{**} \rightarrow \boldsymbol{s}_{i+1})$ for $i = 0, 1, \ldots, d-1$
    - Then $\boldsymbol{s}_0 M_0 = 2\boldsymbol{e}_0$, and $\boldsymbol{s}_i M_i = 2\boldsymbol{e}_i + \boldsymbol{s}_{i-1}^{**}$
- Enc(b): $\boldsymbol{r} \in_R \{0,1\}^m$, $\boldsymbol{c} \leftarrow M_0 \boldsymbol{r} + [b, 0, \ldots, 0]$, output $(\boldsymbol{c}, 0)$
- Dec($\boldsymbol{c}, i$): Recover $b \leftarrow (\langle \boldsymbol{s}_i, \boldsymbol{c} \rangle \bmod q) \bmod 2$
  - For level-0: $\langle \boldsymbol{s}_o, \boldsymbol{c} \rangle = \boldsymbol{s}_0 M_0 \boldsymbol{r} + b = 2\langle \boldsymbol{e}_0, \boldsymbol{r} \rangle + b$
  - $\boldsymbol{e}_0, \boldsymbol{r}$ are short so $2\langle \boldsymbol{e}_0, \boldsymbol{r} \rangle \ll q$, hence no wraparound

The [BV'11b] "Leveled SWHE"

- Ciphertexts in same level can be added directly
- To multiply two level-i ciphertexts $(\boldsymbol{c}_1,i),(\boldsymbol{c}_2,i)$
  - Compute the extended $\boldsymbol{c}^*$=vec$(\boldsymbol{c}_1 \otimes \boldsymbol{c}_2)$, the "doubly extended" $\boldsymbol{c}^{**}$, and set $\boldsymbol{c}' \leftarrow \mathrm{M}_i \boldsymbol{c}^{**}$
  - $(\boldsymbol{c}', i+1)$ is a level-$(i+1)$ ciphertext
- Semantic-security follows because:
  - Under LWE, the M_i's are pseudo-random
  - If they were random then ciphertexts would have no information about the encrypted plaintexts
    - By leftover hash lemma

From SWHE to FHE

- The "noise" in a ciphertext $(\mathbf{c},i)$ is $<\mathbf{s}_i,\mathbf{c}>$ mod q
  - Noise magnitude roughly doubles on addition, get squared on multiplication
  - Can only evaluate log-depth circuits before the noise magnitude exceeds q
- How to evaluate deeper circuits?
  - Squash & bootstrap,
  - Chimeric & bootstrap,
  - or an altogether new technique…

Modulus Switching

- Converting $\boldsymbol{c},\boldsymbol{s}$ into $\boldsymbol{c}',\boldsymbol{s}'$ s.t. for some $p < q$
  $(\langle \boldsymbol{s}',\boldsymbol{c}'\rangle \bmod p) \equiv (\langle \boldsymbol{s},\boldsymbol{c}\rangle \bmod q) \pmod 2$
- [BV'11b]: Use with $p \ll q$ to reduce decryption complexity, can bootstrap without squashing
  - Modulus-switching & key-switching combined
  - The resulting c' can be decrypted, but cannot participate in any more homomorphic operations
- [BGV'11] Use with $p < q$ to reduce the noise, can compute deeper circuits w/o bootstrapping
  - Roughly just by scaling, $\boldsymbol{c}' \leftarrow \text{round}(p/q \cdot \boldsymbol{c})$
  - Rounding "appropriately"

Modulus Switching – Main Lemma

- Let $p<q$ be odd integers, $\mathbf{c},\mathbf{s}\in\mathbf{Z}_q^{\,n}$ such that $|\langle\mathbf{s},\mathbf{c}\rangle \bmod q| < q/2 - q/p\cdot\|\mathbf{s}\|_1$
  - $\|\mathbf{s}\|_1$ is the l_1 norm of $\mathbf{s}$

s must be short

- Let $\mathbf{c}'=\mathrm{rnd}_c(p/q\cdot\mathbf{c})$, where $\mathrm{rnd}_c(\cdot)$ rounds each entry up or down so that $\mathbf{c}'\equiv\mathbf{c} \pmod 2$

- Then (i) $(\langle\mathbf{s},\mathbf{c}'\rangle \bmod p) \equiv (\langle\mathbf{s},\mathbf{c}\rangle \bmod q) \pmod 2$
  and (ii) $|\langle\mathbf{s},\mathbf{c}'\rangle \bmod p| \leq \frac{p}{q}\cdot|\langle\mathbf{s},\mathbf{c}\rangle \bmod q| + \|\mathbf{s}\|_1$

Modulus Switching – Main Lemma

Proof:

- For some κ, $\langle \mathbf{s},\mathbf{c}\rangle \bmod q = \langle \mathbf{s},\mathbf{c}\rangle - \kappa q \in [\frac{-q}{2}, \frac{q}{2}]$
  - Actually in a smaller interval
    $\langle \mathbf{s},\mathbf{c}\rangle - \kappa q \in [\frac{-q}{2}+\frac{q}{p}\|\mathbf{s}\|_1, \frac{q}{2}-\frac{q}{p}\|\mathbf{s}\|_1]$
  - Multiplying by p/q we get
    $\langle \mathbf{s},\frac{p}{q}\mathbf{c}\rangle - \kappa p \in [\frac{-p}{2}+\|\mathbf{s}\|_1, \frac{p}{2}-\|\mathbf{s}\|_1]$
- Replacing $\frac{p}{q}\mathbf{c}$ by $\mathbf{c}'=\mathrm{rnd}_c(\frac{p}{q}\mathbf{c})$, adds error $\le\|\mathbf{s}\|_1$:
  $\langle \mathbf{s},\mathbf{c}'\rangle - \kappa p \in [\frac{-p}{2}, \frac{p}{2}]$, so $\langle \mathbf{s},\mathbf{c}'\rangle - \kappa p = \langle \mathbf{s},\mathbf{c}'\rangle \bmod p$
- This also proves Part (ii)

Modulus Switching – Main Lemma

Proof:

- We know that $\langle \mathbf{s},\mathbf{c}\rangle \bmod q = \langle \mathbf{s},\mathbf{c}\rangle - \kappa q$ and $\langle \mathbf{s},\mathbf{c}'\rangle \bmod p = \langle \mathbf{s},\mathbf{c}'\rangle - \kappa p$ for the same κ
- Since p,q are odd then $\kappa p \equiv \kappa q \pmod 2$
- Since $\mathbf{c}' \equiv \mathbf{c} \pmod 2$ then $\langle \mathbf{s},\mathbf{c}'\rangle \equiv \langle \mathbf{s},\mathbf{c}\rangle \pmod 2$
- $$\begin{aligned}(\langle \mathbf{s},\mathbf{c}'\rangle \bmod p) &= \langle \mathbf{s},\mathbf{c}'\rangle - \kappa p \\ &\equiv \langle \mathbf{s},\mathbf{c}\rangle - \kappa q \pmod 2 \\ &= (\langle \mathbf{s},\mathbf{c}\rangle \bmod q)\end{aligned}$$
- This proves part (i) ∎

Making s Small

- If $\boldsymbol{s}$ is random in $\mathbf{Z}_q{}^n$ then $\|\boldsymbol{s}\|_1 > q$
- Luckily [ACPS 2009] proved that LWE is hard even when $\boldsymbol{s}$ is a random short vector
  - chosen from the same distribution as the noise $\boldsymbol{e}$
- So we use this distribution for the secret keys
- Alternatively, we could have used the trick with BitDecomp() and PowersOf2()

Modulus Switching

- Example: q=127, p=29, $\boldsymbol{c}$=(175,212), $\boldsymbol{s}$=(2,3)
- $\langle \boldsymbol{s},\boldsymbol{c}\rangle \bmod q = 986 - 8 \times 127 = -30$
- $p/q \cdot \boldsymbol{c} \approx (39.9, 48.4)$
  - To get $\boldsymbol{c}' \equiv \boldsymbol{c}$ (mod 2) we round down both entries
  - $\boldsymbol{c}'$=(39,48)
- $\langle \boldsymbol{s},\boldsymbol{c}'\rangle \bmod p = 222 - 8 \times 29 = -10$
- Indeed κ=8 in both cases, $-10 \equiv -30$ (mod 2)
- The noise magnitude decreased from 30 to 10
  - But the relative magnitude increased, $\frac{10}{29} > \frac{30}{127}$

How Does Modulus-Switching Help?

- Start with large modulus q_0, small noise $\eta \ll q_0$
- After 1st multiplication, noise grows to $\approx \eta^2$
- Switch the modulus to $q_1 \approx q_0/\eta$,
  - Noise similarly reduced to $\approx \eta^2/\eta = \eta$
- After next multiplication layer, noise again grows to $\approx \eta^2$, switch to $q_2 \approx q_1/\eta$ to reduce it back to η
- Keep switching moduli after each layer
  - Setting $q_{i+1} \approx q_i/\eta$
  - Until last modulus is too small, $q_d/2 \leq \eta$

How Does Modulus-Switching Help?

- Example: $q_0 \approx \eta^5$

	Using mod-switching		Without mod-switching	
	Noise	Modulus	Noise	Modulus
Fresh ciphertexts	η	η^5	η	η^5
Level-1, degree=2	η	η^4	η^2	η^5
Level-2, degree=4	η	η^3	η^4	η^5
Level-3, degree=8	η	η^2	η^8	η^5
Level-4, degree=16	η	η		

decryption errors

Putting It All Together

- Use tensor-product for multiplication
- Then reduce the dimension with M($\boldsymbol{s} \rightarrow \boldsymbol{s}'$)
  - First need to use PowersOf2/BitDecomp
- Then reduce the noise by switching modulus
  - This works if the secret key $\boldsymbol{s}$ is short
- Repeat until modulus is too small

The [BGV'11] "Leveled FHE"

- d-level circuits, initial noise η
  - Also $\tau \triangleq \eta \cdot \mathrm{poly}(n)$ is another parameter
- Set odd moduli $q_0,\ldots,q_d$ s.t. $q_i \approx \tau^{d-i+1}$

<u>Key generation:</u>

- Choose short secret $\boldsymbol{s}_i \in \mathbf{Z}_{q_i}{}^n$, i=0,…,d, first entry=1
  - Set $\boldsymbol{s}_i{}^* = \mathrm{vec}(\boldsymbol{s}_i \otimes \boldsymbol{s}_i) \in \mathbf{Z}_{q_i}{}^{n^2}$; $\boldsymbol{s}_i^{**} = \mathrm{PowersOf2}_{q_i}(\boldsymbol{s}_i{}^*) \in \mathbf{Z}_{q_i}{}^{t_i}$
- Public key has $\mathrm{M}_0 = \mathrm{M}(0 \rightarrow \boldsymbol{s}_0) \in \mathbf{Z}_{q_0}{}^{n \times t_0}$ and $\mathrm{M}_i = \mathrm{M}(\boldsymbol{s}_{i-1}^{**} \rightarrow \boldsymbol{s}_i) \in \mathbf{Z}_{q_{i-1}}{}^{n \times t_{i-1}}$
  - The "short error vector" in M_i is $\boldsymbol{e}_i \in \mathbf{Z}_{q_{i-1}}{}^{t_{i-1}}$
  - Then $\boldsymbol{s}_0 \mathrm{M}_0 = 2\boldsymbol{e}_0 \bmod q_0$ and $\boldsymbol{s}_i \mathrm{M}_i = 2\boldsymbol{e}_i + \boldsymbol{s}_{i-1}^{**} \bmod q_{i-1}$

$t_0 = 3n\log(q_0)$ and
$t_i = n^2\log(q_i)$

The [G'11] "Leveled FHE"

- Enc, Dec, and homomorphic addition are the same as in the leveled SWHE
  - Level-i ciphertexts are modulo q_i
- To multiply two level-i ciphertexts, $\boldsymbol{c}_1,\boldsymbol{c}_2$:
  - $\boldsymbol{c}^* \leftarrow \mathrm{vec}(\boldsymbol{c}_1 \otimes \boldsymbol{c}_2) \in \mathbf{Z}_{q_i}^{n^2}$, $\quad (\langle \boldsymbol{s}_i^*, \boldsymbol{c}^* \rangle \bmod q_i) \equiv b_1 b_2 \pmod 2$
  - $\boldsymbol{c}^{**} \leftarrow \mathrm{BitDecom}(\boldsymbol{c}^*)$, $\quad (\langle \boldsymbol{s}_i^{**}, \boldsymbol{c}^{**} \rangle \bmod q_i) \equiv b_1 b_2 \pmod 2$
  - $\boldsymbol{c}' \leftarrow \mathrm{M}_{i+1} \boldsymbol{c}^{**} \bmod q_i$ $\quad (\langle \boldsymbol{s}_{i+1}, \boldsymbol{c}' \rangle \bmod q_i) \equiv b_1 b_2 \pmod 2$
  - $\boldsymbol{c} \leftarrow \mathrm{rnd}_{\boldsymbol{c}'}(q_{i+1}/q_i \cdot \boldsymbol{c}')$, $\quad (\langle \boldsymbol{s}_{i+1}, \boldsymbol{c} \rangle \bmod q_{i+1}) \equiv b_1 b_2 \pmod 2$
- Noise in $\boldsymbol{c}$ is bounded by $(\eta^2+\text{stuff})/\tau \le \eta$

What We Have So Far

- A leveled-FHE:
  - Public-key size at least linear in circuit depth
  - Can handle circuits of arbitrary polynomial depth
- Security based on LWE
  - $\frac{1}{\beta} \approx \frac{\text{modulus}}{\text{noise}} = (\text{poly}(n))^{\text{depth}}$
  - For “interesting” circuits this is more that poly(n)
- Modulus gets smaller as we go up the circuit
  - Lower levels somewhat more expensive

Variants and Optimizations

- Use bootstrapping to recover large modulus
  - Size of largest modulus depends on decryption circuit, not the circuits that we evaluate
  - Can be made into “pure” FHE (non-leveled), need to assume circular security
- Base security on ring-LWE
  - LWE over a ring other than $\mathbf{Z}_q$ (e.g., $\mathbf{R}=\mathbf{Z}_q[x]/f(x)$)
  - Can use smaller dimension (e.g., dim=2)
- Large plaintext space (not just $\mathbf{Z}_2$)
  - Must tweak the modulus-switching technique

Variants and Optimizations

- Batching: pack many bits into each ciphertext
  - E.g., using the Chinese Remainders Theorem
  - An operation (+,x) on ciphertext acts separately on each the packed bits
- Combining these optimizations, can reduce the overhead to $\tilde{O}(\lambda)$
  - Compare to $\tilde{O}(\lambda^{3.5})$ for the original blueprint

Current Status of HE constructions

- Many new ideas are at the table now
  - Still figuring out what works and what doesn't
  - Looking at recent history, we can expect more new ideas in the next few months/years
- Implementation efforts are underway
  - Goal: get usable FHE
  - At least for some applications
  - My personal guess: almost at hand, perhaps only 2-3 years away
- Many open problems remain