

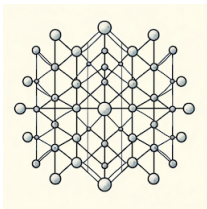
# Introduction to Fully Homomorphic Encryption

## Part 2: leveled FHE and bootstrapping

Jean-Sébastien Coron

University of Luxembourg

- Previous lecture: basic techniques for fully homomorphic encryption
  - First generation of FHE, the DGHV scheme
  - Overview of bootstrapping
- This lecture: leveled FHE and bootstrapping
  - LWE-based encryption, security against lattice attacks
  - Relinearization for ciphertext multiplication
  - Modulus switching, leveled FHE
  - Bootstrapping



# Four generations of FHE

- First generation: bootstrapping, slow
  - Breakthrough scheme of Gentry [G09], based on ideal lattices.
  - FHE over the integers: [DGHV10]
- Second generation: [BV11], [BGV11]
  - More efficient, (R)LWE based. Relinearization, depth-linear construction with modulus switching.
- Third generation [GSW13]
  - No modulus switching, slow noise growth
  - Improved bootstrapping: [BV14], [AP14]
- Fourth gen: [CKKS17]
  - Approximate floating point arithmetic

# LWE-based encryption [R05]

- Key generation
  - Secret-key:  $\mathbf{s} \in (\mathbb{Z}_q)^n$
- Encryption of  $m \in \{0, 1\}$ 
  - A vector  $\mathbf{c} \in \mathbb{F}_q$  such that

$$\langle \mathbf{c}, \mathbf{s} \rangle = 2e + m \pmod{q}$$

- for a small error  $e$ .

The diagram shows a dot product operation. On the left, a horizontal row of three green boxes is labeled  $\mathbf{c}$ . To its right is a vertical column of three red boxes labeled  $\mathbf{s}$ . A dot operator  $\cdot$  is placed between the two vectors. To the right of the dot is an equals sign, followed by a single red box labeled  $2e + m$ .

- Distribution of the error  $e$ 
  - One can take the centered binomial distribution  $\chi$  with parameter  $\kappa$ .
  - Let  $e = h(u) - h(v)$  where  $u, v \leftarrow \{0, 1\}^\kappa$ , where  $h$  is the Hamming weight function.
- Decryption
  - Compute  $m = (\mathbf{c} \cdot \mathbf{s} \bmod q) \bmod 2$
  - Decryption works if  $|e| < q/4$

# LWE-based encryption [R05]

- Key generation
  - Secret-key:  $\mathbf{s} \in (\mathbb{Z}_q)^n$
- Encryption of  $m \in \{0, 1\}$ 
  - A vector  $\mathbf{c} \in \mathbb{F}_q$  such that

$$\langle \mathbf{c}, \mathbf{s} \rangle = 2e + m \pmod{q}$$

- for a small error  $e$ .

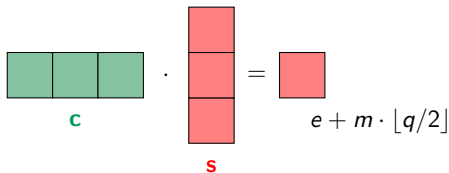
The diagram shows a dot product operation. On the left, a horizontal row of three green boxes is labeled  $\mathbf{c}$  below it. To its right is a vertical column of three red boxes labeled  $\mathbf{s}$  below it. A dot operator  $\cdot$  is placed between the two vectors. To the right of the dot is an equals sign  $=$ , followed by a single red box labeled  $2e + m$  below it.

- Distribution of the error  $e$ 
  - One can take the centered binomial distribution  $\chi$  with parameter  $\kappa$ .
  - Let  $e = h(u) - h(v)$  where  $u, v \leftarrow \{0, 1\}^\kappa$ , where  $h$  is the Hamming weight function.
- Decryption
  - Compute  $m = (\mathbf{c} \cdot \mathbf{s} \bmod q) \bmod 2$
  - Decryption works if  $|e| < q/4$

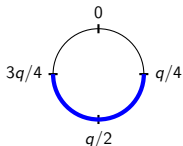
# LWE-based encryption: alternative encoding

- The message  $m$  can also be encoded in the MSB.
- Encryption of  $m \in \{0, 1\}$ 
  - A vector  $\mathbf{c} \in \mathbb{F}_q$  such that

$$\langle \mathbf{c}, \mathbf{s} \rangle = e + m \cdot \lfloor q/2 \rfloor \pmod{q}$$



- Decryption
  - Compute  $m = \text{th}(\langle \mathbf{c}, \mathbf{s} \rangle \bmod q)$
  - where  $\text{th}(x) = 1$  if  $x \in (q/4, 3q/4)$ , and 0 otherwise.



# LWE-based public-key encryption

- Key generation
  - Secret-key:  $\mathbf{s} \in (\mathbb{Z}_q)^n$ , with  $s_1 = 1$ .
  - Public-key:  $\mathbf{A}$  such that  $\mathbf{A} \cdot \mathbf{s} = \mathbf{e}$  for small  $\mathbf{e}$ 
    - Every row of  $\mathbf{A}$  is an LWE encryption of 0.

- Encryption of  $m \in \{0, 1\}$

$$\mathbf{c} = \mathbf{u} \cdot \mathbf{A} + (m \cdot \lfloor q/2 \rfloor, 0, \dots, 0)$$

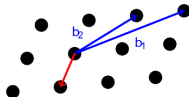
- for a small  $\mathbf{u}$

The diagram illustrates the encryption process. On the left, a red vector  $\mathbf{u}$  (represented by four red boxes) is multiplied by a green matrix  $\mathbf{A}$  (represented by a 4x3 grid of green boxes). The result is added to a red vector  $(\lfloor \frac{q}{2} \rfloor \cdot m, 0, 0)$  (represented by three red boxes). The final result is a green vector  $\mathbf{c}$  (represented by three green boxes).

- Decryption
  - Compute  $m = \text{th}(\langle \mathbf{c}, \mathbf{s} \rangle \bmod q)$

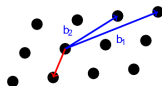
# Security of LWE-based encryption

- The previous scheme is semantically secure under the LWE assumption:
  - Given polynomially samples, it is hard to distinguish  $(\mathbf{a}, \langle \mathbf{a}, \mathbf{s} \rangle + e)$  from  $(\mathbf{a}, b)$ , for random  $\mathbf{a} \leftarrow \mathbb{Z}_q^n$ ,  $b \leftarrow \mathbb{Z}_q$  and  $e \leftarrow \chi$
- Security against lattice attacks
  - The LWE problem can be solved using the LLL and BKZ algorithms, which compute an approximation of the shortest vector problem in a lattice.



- Lattice definition

- A lattice  $L \subset \mathbb{Z}^n$  is the set of integer linear combinations of  $n$  linearly-independent vectors  $\mathbf{b}_1, \dots, \mathbf{b}_n$ .

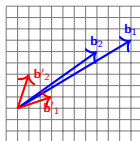


- Let  $\lambda_1(L)$  the norm of a shortest non-zero vector of the lattice. Gaussian heuristic: for a “random” lattice:

$$\lambda_1(L) \simeq \sqrt{n/(2\pi e)} \cdot \det(L)^{1/n}$$

- Lattice reduction

- Takes as input a lattice basis and outputs a reduced basis of shorter vectors
- Lattice reduction algorithms: LLL and BKZ
- Quality of the basis characterized by Hermite factor  $\delta^n$ :



$$\|\mathbf{b}_1\| \simeq \delta^n \det(L)^{1/n}$$

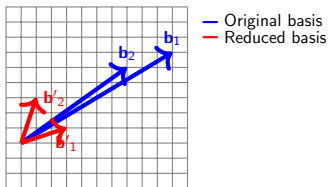
# The LLL lattice reduction algorithm

- The LLL algorithm [LLL82]
  - Takes as input a basis  $(\mathbf{b}_1, \dots, \mathbf{b}_n)$  and outputs a reduced basis, in polynomial time.

## Lemma (LLL-reduced basis)

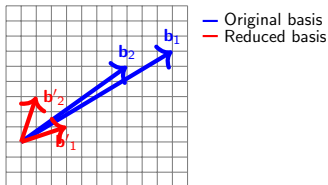
Let  $\mathbf{b}_1, \dots, \mathbf{b}_n$  an LLL-reduced basis of a lattice  $L$ . Then  $\|\mathbf{b}_1\| \leq 2^{(n-1)/2} \lambda_1(L)$  and  $\|\mathbf{b}_1\| \leq 2^{(n-1)/4} (\det L)^{1/n}$ .

- Performances
  - Heuristic running time of fast variants:  $\mathcal{O}(n^3 \log^2 B)$ .
  - Practical Hermite factor:  $\delta^n$  with  $\delta \simeq 1.0219$ .



# The BKZ algorithm

- The BKZ algorithm [S87]
  - Works by solving SVP on blocks of size  $k < n$ . For LLL:  $k = 2$ .
  - Achieves a better Hermite factor than LLL, but with computation time exponential in the block-size  $k$ .
- Performances:
  - Approximate Hermite factor:  $\delta = k^{1/2k}$ , or even  $\delta = 2^{1/k}$ .
  - Running time:  $2^{0.292 \cdot k + o(k)}$  using sieving for implementing the SVP oracle.



# Primal lattice attack against LWE

- LWE instance
  - $\mathbf{A} \in \mathbb{Z}_q^{m \times n}$  such that  $\mathbf{A}\mathbf{s} = \mathbf{e} \bmod q$  for small  $\mathbf{s} \in \mathbb{Z}^n$ ,  $\mathbf{e} \in \mathbb{Z}^m$ .
  - Goal: recover  $\mathbf{s}$ .
- Primal lattice attack [APS15]
  - Consider the lattice of dimension  $d = m + n$

$$L = \{\mathbf{x} \in \mathbb{Z}^d : (\mathbf{A} \mid \mathbf{I}_m) \cdot \mathbf{x} = \mathbf{0} \bmod q\}$$

- Lattice has a short vector  $\mathbf{v} = (\mathbf{s}, -\mathbf{e})$  of norm  $\lambda \simeq \zeta \sqrt{n+m}$ , where  $\zeta$  is standard deviation of  $\mathbf{x}$  and  $\mathbf{e}$ .
- Writing  $\mathbf{a}_1, \dots, \mathbf{a}_n \in \mathbb{Z}^m$  the column vectors of  $\mathbf{A}$ , a basis matrix of  $L$  is given by

$$L = \begin{bmatrix} 1 & & & -\mathbf{a}_1^T \\ & \ddots & & \\ & & 1 & -\mathbf{a}_n^T \\ & & & q\mathbf{I}_m \end{bmatrix}$$

# Analysis of primal lattice attack

- Simple analysis [GN08]
  - Gaussian heuristics: short vectors of the lattice have norm  $\simeq \sqrt{d/(2\pi e)} \cdot \det(L)^{1/d}$
  - $\mathbf{v} = (\mathbf{s}, -\mathbf{e})$  is an unusually short lattice vector.
  - Lattice reduction with Hermite factor  $\delta$  can recover  $\mathbf{v}$  if there is a large enough gap between the norm of  $\mathbf{v}$  times the Hermite factor, and the norm of the other short vectors:

$$\|\mathbf{v}\| \cdot \delta^d < \sqrt{\frac{d}{2\pi e}} \cdot q^{m/d}$$

- Asymptotically, one must have  $n = \Omega(\log q)$ .
- Applying the attack
  - Optimize for the number of vectors  $m$  in  $d = m + n$ .
  - Apply LLL and BKZ for increasing large block-sizes
  - Recover  $\mathbf{s}$ .

# Homomorphic addition

- LWE ciphertexts can be added

$$\langle \mathbf{c}_1, \mathbf{s} \rangle = e_1 + m_1 \cdot (q + 1)/2 \pmod{q}$$

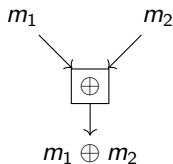
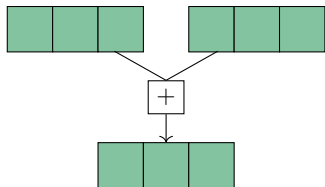
$$\langle \mathbf{c}_2, \mathbf{s} \rangle = e_2 + m_2 \cdot (q + 1)/2 \pmod{q}$$

$$\langle \mathbf{c}_1 + \mathbf{c}_2, \mathbf{s} \rangle = e_1 + e_2 + (m_1 + m_2) \cdot (q + 1)/2 \pmod{q}$$

- write  $m_1 + m_2 = 2u + (m_1 \oplus m_2)$

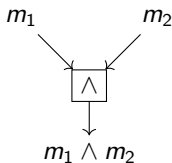
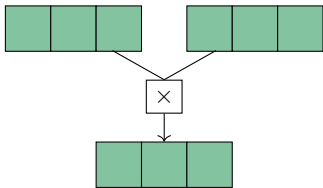
$$\langle \mathbf{c}_1 + \mathbf{c}_2, \mathbf{s} \rangle = e_1 + e_2 + u + (m_1 \oplus m_2) \cdot (q + 1)/2 \pmod{q}$$

- Therefore  $\mathbf{c}_1 + \mathbf{c}_2$  is an encryption of  $m_1 \oplus m_2$ , with a small increase in the noise.



# Homomorphic multiplication of ciphertexts

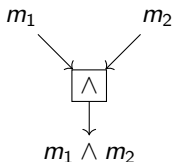
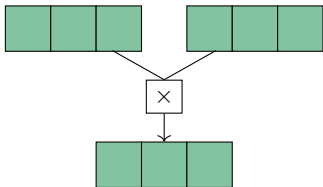
- We would like to perform homomorphic multiplication of ciphertexts



- but homomorphic multiplication is more complex, with 2 steps
- First step: tensor product
  - We obtain a ciphertext  $\mathbf{c}_1 \times \mathbf{c}_2 \in \mathbb{Z}_q^{n^2}$ , under a new key  $\mathbf{s} \times \mathbf{s}$ .
- Second step: key switching
  - We switch the key  $\mathbf{s} \times \mathbf{s}$  back to the original key  $\mathbf{s}$ .
  - Only works for binary ciphertext, so we first compute a binary decomposition of the ciphertext, under a new key.

# Homomorphic multiplication of ciphertexts

- We would like to perform homomorphic multiplication of ciphertexts



- but homomorphic multiplication is more complex, with 2 steps
- First step: tensor product
  - We obtain a ciphertext  $\mathbf{c}_1 \times \mathbf{c}_2 \in \mathbb{Z}_q^{n^2}$ , under a new key  $\mathbf{s} \times \mathbf{s}$ .
- Second step: key switching
  - We switch the key  $\mathbf{s} \times \mathbf{s}$  back to the original key  $\mathbf{s}$ .
  - Only works for binary ciphertext, so we first compute a binary decomposition of the ciphertext, under a new key.

# Product of ciphertexts

- We want to compute the product of  $\langle \mathbf{c}_1, \mathbf{s} \rangle$  and  $\langle \mathbf{c}_2, \mathbf{s} \rangle$

$$\langle \mathbf{c}_1, \mathbf{s} \rangle = e_1 + m_1 \cdot (q + 1)/2 \pmod{q}$$

$$\langle \mathbf{c}_2, \mathbf{s} \rangle = e_2 + m_2 \cdot (q + 1)/2 \pmod{q}$$

$$\begin{aligned} 2\langle \mathbf{c}_1, \mathbf{s} \rangle \cdot \langle \mathbf{c}_2, \mathbf{s} \rangle &= 2(e_1 + (q + 1)/2 \cdot m_1) \cdot (e_2 + (q + 1)/2 \cdot m_2) \\ &= e + m_1 m_2 \cdot (q + 1)/2 \pmod{q} \end{aligned}$$

- This corresponds to an encryption of  $m_1 \cdot m_2$  for a new error  $e = 2e_1 e_2 + m_1 e_2 + m_2 e_1$ .
  - The bitlength of the error has roughly doubled.
- Goal: compute a new ciphertext  $\mathbf{c}'$  and a new key  $\mathbf{s}'$  such that

$$\langle \mathbf{c}', \mathbf{s}' \rangle = e + m_1 m_2 \cdot (q + 1)/2 \pmod{q}$$

- Tool: tensor product

# Product of ciphertexts

- We want to compute the product of  $\langle \mathbf{c}_1, \mathbf{s} \rangle$  and  $\langle \mathbf{c}_2, \mathbf{s} \rangle$

$$\langle \mathbf{c}_1, \mathbf{s} \rangle = e_1 + m_1 \cdot (q + 1)/2 \pmod{q}$$

$$\langle \mathbf{c}_2, \mathbf{s} \rangle = e_2 + m_2 \cdot (q + 1)/2 \pmod{q}$$

$$\begin{aligned} 2\langle \mathbf{c}_1, \mathbf{s} \rangle \cdot \langle \mathbf{c}_2, \mathbf{s} \rangle &= 2(e_1 + (q + 1)/2 \cdot m_1) \cdot (e_2 + (q + 1)/2 \cdot m_2) \\ &= e + m_1 m_2 \cdot (q + 1)/2 \pmod{q} \end{aligned}$$

- This corresponds to an encryption of  $m_1 \cdot m_2$  for a new error  $e = 2e_1 e_2 + m_1 e_2 + m_2 e_1$ .
  - The bitlength of the error has roughly doubled.
- Goal: compute a new ciphertext  $\mathbf{c}'$  and a new key  $\mathbf{s}'$  such that

$$\langle \mathbf{c}', \mathbf{s}' \rangle = e + m_1 m_2 \cdot (q + 1)/2 \pmod{q}$$

- Tool: tensor product

# Tensor product of ciphertexts

- LWE ciphertexts can be multiplied by tensor product.

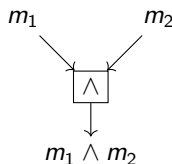
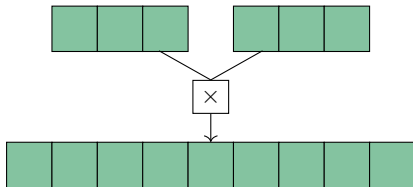
$$2\langle \mathbf{c}_1, \mathbf{s} \rangle \langle \mathbf{c}_2, \mathbf{s} \rangle = 2 \left( \sum_{i=1}^n c_{1,i} s_i \right) \left( \sum_{i=1}^n c_{2,i} s_i \right) = \sum_{i=1}^n \sum_{j=1}^n 2c_{1,i} c_{2,j} \cdot s_i s_j$$

- We can write

$$2\langle \mathbf{c}_1, \mathbf{s} \rangle \cdot \langle \mathbf{c}_2, \mathbf{s} \rangle = \langle \mathbf{c}', \mathbf{s}' \rangle$$

- for new LWE ciphertext:  $\mathbf{c}' = (2c_{1,i}c_{2,j})_{i,j} \in \mathbb{Z}_q^{n^2}$
- and new key:  $\mathbf{s}' = (s_i s_j)_{i,j} \in \mathbb{Z}_q^{n^2}$

$$\langle \mathbf{c}', \mathbf{s}' \rangle = e + m_1 m_2 \cdot (q + 1)/2 \pmod{q}$$



- Initial ciphertext under key  $\mathbf{s}$ 
  - Start with a binary ciphertext  $\mathbf{c} \in \{0, 1\}^m$  under key  $\mathbf{s} \in \mathbb{Z}_q^m$ .
  - We write

$$\langle \mathbf{c}, \mathbf{s} \rangle = \sum_{i=1}^m c_i \cdot s_i \pmod{q}$$

- New key  $\mathbf{s}'$ 
  - To switch to  $\mathbf{s}'$ , we consider LWE pseudo-encryptions  $\mathbf{t}_i$  of each  $s_i$ , under the new key  $\mathbf{s}'$

$$\langle \mathbf{t}_i, \mathbf{s}' \rangle = f_i + s_i \pmod{q}$$

- for small errors  $f_i$ .
- Computing a new ciphertext under  $\mathbf{s}'$ 
  - We inject  $s_i$  from the second equation to the first equation.

- Generating the new ciphertext under  $\mathbf{s}'$

$$\begin{aligned}\langle \mathbf{c}, \mathbf{s} \rangle &= \sum_{i=1}^m c_i \cdot s_i = \sum_{i=1}^m c_i (\langle \mathbf{t}_i, \mathbf{s}' \rangle - f_i) \\ &= \left\langle \sum_{i=1}^m c_i \mathbf{t}_i, \mathbf{s}' \right\rangle - \sum_{i=1}^m c_i \cdot f_i \pmod{q}\end{aligned}$$

- We can define a new ciphertext  $\mathbf{c}' = \sum_{i=1}^m c_i \mathbf{t}_i \pmod{q}$ :

$$\langle \mathbf{c}, \mathbf{s} \rangle = \langle \mathbf{c}', \mathbf{s}' \rangle + f \pmod{q}$$

- for an additional error  $f = -\sum_{i=1}^m c_i \cdot f_i \pmod{q}$
- We want  $f$  to be small, so that  $\langle \mathbf{c}, \mathbf{s} \rangle \simeq \langle \mathbf{c}, \mathbf{s} \rangle$ 
  - But the  $c_i$ 's are full size modulo  $q$
  - Tool: binary decomposition of ciphertext

# Binary decomposition

- We want to have a ciphertext with binary components only.
  - For any  $0 \leq a, b < q$ , we have, using  $n_q = \lceil \log_2 q \rceil$ :

$$a \cdot b = \sum_{i=0}^{n_q-1} a_i \cdot 2^i b = \langle \text{BitDecomp}(a), \text{PowerOf2}(b) \rangle$$

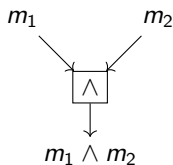
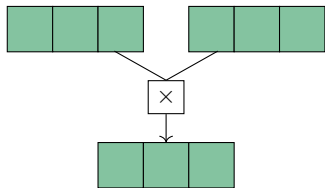
- $\text{BitDecomp}(a) = (a_0, \dots, a_{n_q-1})$
  - $\text{PowerOf2}(b) = (b, 2^1 b, \dots, 2^{n_q-1} b)$ .
  - We extend  $\text{BitDecomp}$  and  $\text{PowerOf2}$  to vectors, by concatenation
- New binary ciphertext from  $\mathbf{c} \in \mathbb{Z}_q^m$  and  $\mathbf{s} \in \mathbb{Z}_q^m$ 
  - Let  $\mathbf{c}' = \text{BitDecomp}(\mathbf{c})$ , and  $\mathbf{s}' = \text{PowerOf2}(\mathbf{s})$

$$\langle \mathbf{c}', \mathbf{s}' \rangle = \langle \text{BitDecomp}(\mathbf{c}), \text{PowerOf2}(\mathbf{s}) \rangle = \langle \mathbf{c}, \mathbf{s} \rangle$$

- The new binary ciphertext  $\mathbf{c}'$  encrypts the same message under the new secret-key  $\mathbf{s}'$ .
  - One can then perform the key switching.

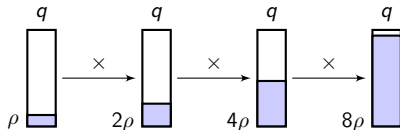
# Summary of homomorphic multiplication

- 1) Tensor product
  - We obtain a ciphertext  $\mathbf{c}' = \mathbf{c}_1 \times \mathbf{c}_2 \in \mathbb{Z}_q^{n^2}$ , under a new key  $\mathbf{s}' = \mathbf{s} \times \mathbf{s}$ .
- 2) Binary decomposition
  - We obtain a binary ciphertext  $\mathbf{c}'' \in \{0, 1\}^{n^2 \cdot n_q}$ , under a new key  $\mathbf{s}'' = \text{PowerOfTwo}(\mathbf{s}')$ , with  $n_q = \lceil \log_2 q \rceil$
- 3) Key switching
  - We switch the key from  $\mathbf{s}''$  back to the original key  $\mathbf{s}$ .



# Somewhat homomorphic encryption

- Problem: exponential growth of noise



- One can only accommodate a limited number of multiplications
  - For  $L$  multiplicative layers, one needs  $\log q = \Omega(2^L)$ , with dimension  $n = \Omega(\log q) = \Omega(2^L)$ .
  - Can only accommodate logarithmic depth.
- Solutions
    - Leveled FHE via modulus switching [BGV11]
    - Bootstrapping [G09]

# Modulus switching

- Consider a ciphertext modulo  $q$

$$\begin{aligned}\langle \mathbf{c}, \mathbf{s} \rangle &= \lfloor q/2 \rfloor \cdot m + e \pmod{q} \\ &= q/2 \cdot m + \varepsilon + e + \lambda \cdot q\end{aligned}$$

- for  $|\varepsilon| \leq 1/2$  and  $\lambda \in \mathbb{Z}$
- Switching to a ciphertext modulo  $p < q$

$$\langle \mathbf{c} \cdot \frac{p}{q}, \mathbf{s} \rangle = p/2 \cdot m + \varepsilon \cdot \frac{p}{q} + e \cdot \frac{p}{q} + \lambda \cdot p$$

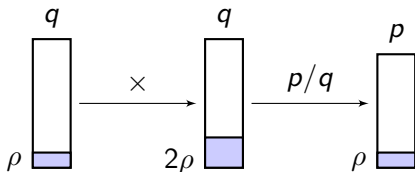
- Write  $\mathbf{c}' = \lfloor \mathbf{c} \cdot p/q \rfloor = \mathbf{c} \cdot p/q + \mathbf{u}$  where  $\|\mathbf{u}\|_\infty \leq 1/2$ . Then

$$\langle \mathbf{c}', \mathbf{s} \rangle = \lfloor p/2 \rfloor \cdot m + e' \pmod{p}$$

- where  $|e'| \leq e \cdot p/q + 1 + \frac{1}{2} \cdot \|\mathbf{s}\|_1$
- We get a new ciphertext  $\mathbf{c}'$  modulo  $p$  encrypting the same  $m$ 
  - with scaled error  $e' \simeq e \cdot p/q$ .

# The BGV scheme: modulus switching [BGV11]

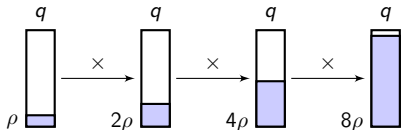
- Modulus switching from  $\mathbf{c}$  modulo  $q$  to  $\mathbf{c}'$  modulo  $p < q$ 
  - Encrypts the same message  $m$ , but with error scaled by  $p/q$
- Application: reducing noise growth. Assume  $p/q = 2^{-\rho}$ .



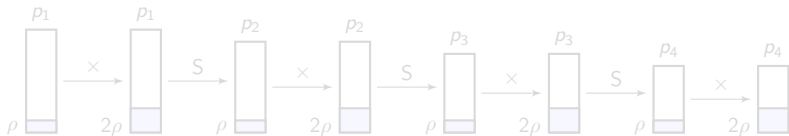
- Noise reduction without bootstrapping !

# Leveled fully homomorphic encryption

- Previous model: exponential growth of noise



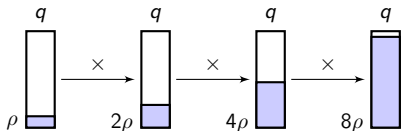
- Only bootstrapping can give FHE
- New model: modulus switching after each multiplication layer
  - with a ladder of moduli  $p_i$  such that  $p_{i+1}/p_i = 2^{-\rho}$



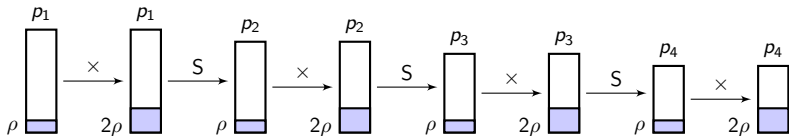
- Leveled FHE
  - Size of  $p_1$  linear in the circuit depth
  - Parameters depend on the depth
  - Can accommodate polynomial depth

# Leveled fully homomorphic encryption

- Previous model: exponential growth of noise



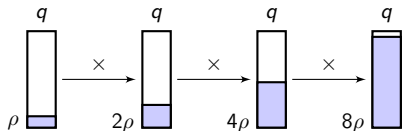
- Only bootstrapping can give FHE
- New model: modulus switching after each multiplication layer
  - with a ladder of moduli  $p_i$  such that  $p_{i+1}/p_i = 2^{-\rho}$



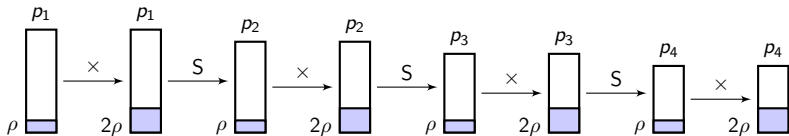
- Leveled FHE
  - Size of  $p_1$  linear in the circuit depth
  - Parameters depend on the depth
  - Can accommodate polynomial depth

# Leveled fully homomorphic encryption

- Previous model: exponential growth of noise



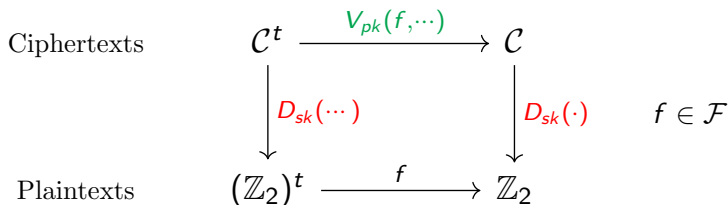
- Only bootstrapping can give FHE
- New model: modulus switching after each multiplication layer
  - with a ladder of moduli  $p_i$  such that  $p_{i+1}/p_i = 2^{-\rho}$



- Leveled FHE
  - Size of  $p_1$  linear in the circuit depth
  - Parameters depend on the depth
  - Can accommodate polynomial depth

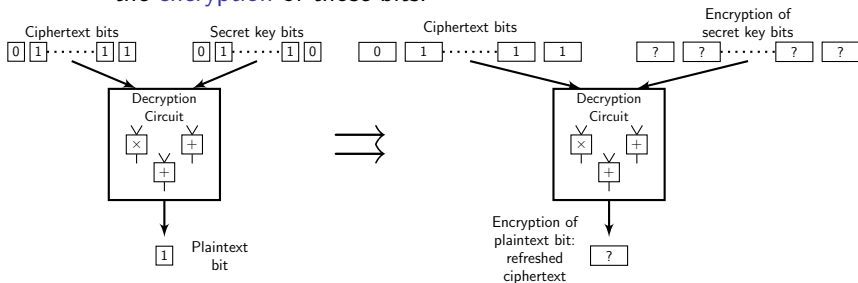
# Gentry's technique to get fully homomorphic encryption

- To build a FHE scheme, start from the **somewhat homomorphic** scheme, that is:
  - Only a polynomial  $f$  of small degree can be computed homomorphically, for  $\mathcal{F} = \{f(b_1, \dots, b_t) : \deg f \leq d\}$
  - $V_{pk}(f, E_{pk}(b_1), \dots, E_{pk}(b_t)) \rightarrow E_{pk}(f(b_1, \dots, b_t))$



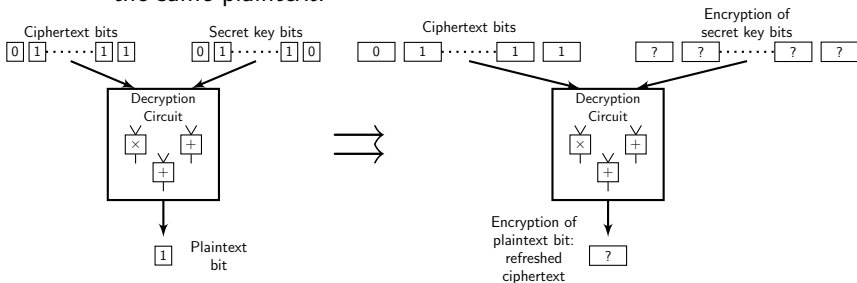
# Ciphertext refresh: bootstrapping

- Gentry's breakthrough idea: refresh the ciphertext using the decryption circuit homomorphically.
  - Evaluate the decryption polynomial not on the bits of the ciphertext  $c$  and the secret key  $sk$ , but homomorphically on the **encryption** of those bits.



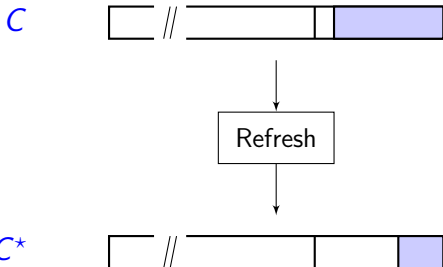
# Ciphertext refresh: bootstrapping

- Gentry's breakthrough idea: refresh the ciphertext using the decryption circuit homomorphically.
  - Instead of recovering the bit plaintext  $m$ , one gets an encryption of this bit plaintext, *i.e.* yet another ciphertext for the same plaintext.



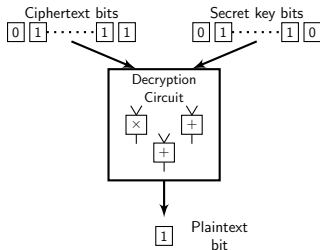
# Ciphertext refresh

- Refreshed ciphertext:
  - If the degree of the decryption polynomial  $D(\cdot, \cdot)$  is small enough, the resulting noise in the new ciphertext can be smaller than in the original ciphertext.



# Bootstrapping LWE ciphertexts

- Building the decryption circuit
  - Takes as input the bits of the ciphertext, and the bits of the secret-key.
  - Outputs the decrypted message  $m \in \{0, 1\}$



- Easier to switch to encryption modulo  $2^k$ , instead of  $q$ 
  - We perform a modulus switching to modulo  $2^k$  using previous technique.

# Building the decryption circuit

- First step: modulus switching to modulo  $2^k$ 
  - Let  $\mathbf{c} \in \mathbb{Z}_q^n$  such that

$$\langle \mathbf{c}, \mathbf{s} \rangle = e + m \cdot (q + 1)/2 \pmod{q}$$

- From the previous modulus switching technique, we get

$$\langle \mathbf{c}', \mathbf{s} \rangle = 2^{k-1} \cdot m + e' \pmod{2^k}$$

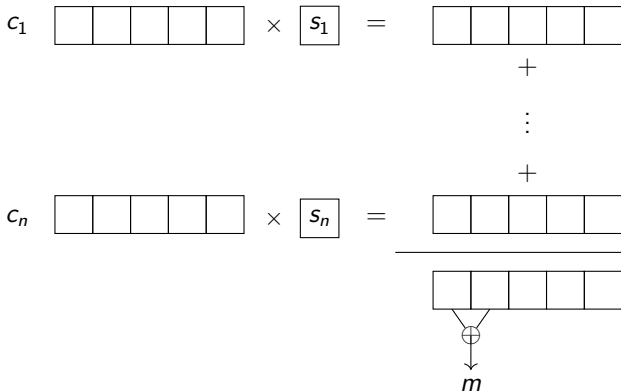
- where  $|e'| \leq e \cdot 2^k/q + 1 + n/2$ .
- For correct decryption, we should have  $|e'| \leq 2^{k-2}$ .
- Therefore we can take  $k = \mathcal{O}(\log n)$ .
- Second step: write the decryption circuit
  - Using only Xor and And gates

# Building the decryption circuit (2)

- We now have a ciphertext  $\mathbf{c} \in \mathbb{Z}_{2^k}^n$  such that:

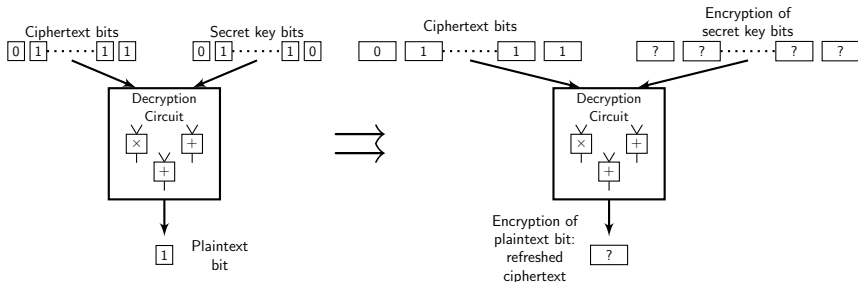
$$\langle \mathbf{c}, \mathbf{s} \rangle = \sum_{i=1}^n c_i \cdot s_i = 2^{k-1} \cdot m + e \pmod{2^k}$$

- We want to write this operation with Xor and And gates only.
- Decryption circuit



# Bootstrapping achieved

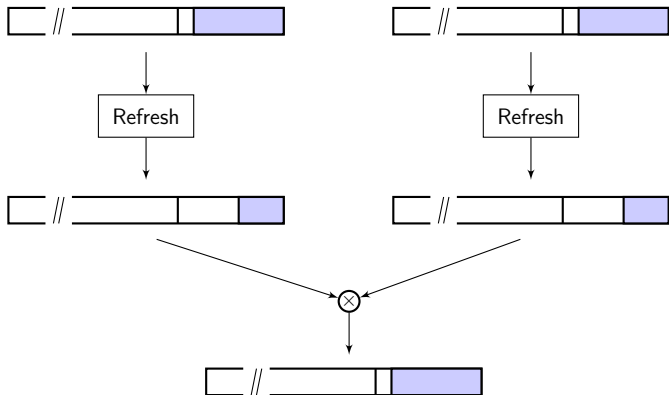
- Bootstrapping
  - We perform the same operations as above, but homomorphically
  - Using an encryption of the secret-key bits



- Refreshed ciphertext  $c'$ 
  - The noise of  $c'$  only depends on the depth of the decryption circuit, not on the initial noise of  $c$ .

# Fully homomorphic encryption

- Fully homomorphic encryption
  - Using this “ciphertext refresh” procedure, the number of homomorphic operations becomes unlimited
  - We get a fully homomorphic encryption scheme.



- Leveled FHE
  - LWE-based encryption.
  - Security against lattice attacks
  - Relinearization for ciphertext multiplication
  - Modulus switching, leveled FHE for polynomial depth circuits
- Bootstrapping for FHE
  - Building the decryption circuit.
  - Homomorphic evaluation of the decryption circuit.
- Next lecture
  - FHE for approximate arithmetic with CKKS.

- BGV11** Zvika Brakerski, Craig Gentry, Vinod Vaikuntanathan. Fully Homomorphic Encryption without Bootstrapping. *Electron. Colloquium Comput. Complex.* 18: 111 (2011)
- Gen09** Craig Gentry. Fully homomorphic encryption using ideal lattices. *STOC 2009*: 169-178
- R05** Oded Regev. On lattices, learning with errors, random linear codes, and cryptography. *STOC 2005*: 84-93