

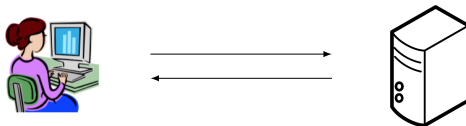
# Introduction to Fully Homomorphic Encryption

## Part 1: basic techniques

Jean-Sébastien Coron

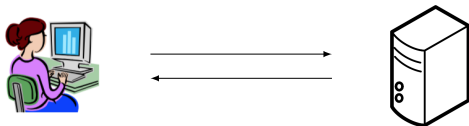
University of Luxembourg

- What is Fully Homomorphic Encryption (FHE) ?
  - Basic properties
  - Cloud computing on encrypted data: the server should process the data without learning the data.



- 4 generations of FHE:
  - 1st gen: [Gen09], [DGHV10]: bootstrapping, slow
  - 2nd gen: [BGV11]: more efficient, (R)LWE based, depth-linear construction (modulus switching).
  - 3rd gen: [GSW13]: no modulus switching, slow noise growth
  - 4th gen: [CKKS17]: approximate computation

- What is Fully Homomorphic Encryption (FHE) ?
  - Basic properties
  - Cloud computing on encrypted data: the server should process the data without learning the data.



- 4 generations of FHE:
  - 1st gen: [Gen09], [DGHV10]: bootstrapping, slow
  - 2nd gen: [BGV11]: more efficient, (R)LWE based, depth-linear construction (modulus switching).
  - 3rd gen: [GSW13]: no modulus switching, slow noise growth
  - 4th gen: [CKKS17]: approximate computation

# Homomorphic Encryption

- Homomorphic encryption: perform operations on plaintexts while manipulating only ciphertexts.
  - Normally, this is not possible.

$$\text{AES}_K(m_1) = 0x3c7317c6bc5634a4ad8479c64714f4f8$$

$$\text{AES}_K(m_2) = 0x7619884e1961b051be1aa407da6cac2c$$

$$\text{AES}_K(m_1 \oplus m_2) = ?$$

- For some cryptosystems with algebraic structure, this is possible. For example RSA:

$$\begin{aligned} c_1 &= m_1^e \bmod N \\ c_2 &= m_2^e \bmod N \end{aligned} \Rightarrow c_1 \cdot c_2 = (m_1 \cdot m_2)^e \bmod N$$

# Homomorphic Encryption

- Homomorphic encryption: perform operations on plaintexts while manipulating only ciphertexts.
  - Normally, this is not possible.

$$\text{AES}_K(m_1) = 0x3c7317c6bc5634a4ad8479c64714f4f8$$

$$\text{AES}_K(m_2) = 0x7619884e1961b051be1aa407da6cac2c$$

$$\text{AES}_K(m_1 \oplus m_2) = ?$$

- For some cryptosystems with algebraic structure, this is possible. For example RSA:

$$\begin{aligned} c_1 &= m_1^e \bmod N \\ c_2 &= m_2^e \bmod N \end{aligned} \Rightarrow c_1 \cdot c_2 = (m_1 \cdot m_2)^e \bmod N$$

# Homomorphic Encryption with RSA

- Multiplicative property of RSA.

$$\begin{aligned}c_1 &= m_1^e \bmod N \\c_2 &= m_2^e \bmod N\end{aligned} \Rightarrow c = c_1 \cdot c_2 = (m_1 \cdot m_2)^e \bmod N$$

- Homomorphic encryption: given  $c_1$  and  $c_2$ , we can compute the ciphertext  $c$  for  $m_1 \cdot m_2 \bmod N$ 
  - using only the public-key
  - without knowing the plaintexts  $m_1$  and  $m_2$ .

# Homomorphism of RSA

- RSA homomorphism: decryption function  $\delta(x) = x^d \bmod N$

$$\delta(c_1 \times c_2) = \delta(c_1) \times \delta(c_2) \pmod{N}$$

Ciphertexts	$\mathbb{Z}/N\mathbb{Z} \times \mathbb{Z}/N\mathbb{Z}$	$\xrightarrow{\times}$	$\mathbb{Z}/N\mathbb{Z}$
	$\downarrow \delta, \delta$		$\downarrow \delta$
Plaintexts	$\mathbb{Z}/N\mathbb{Z} \times \mathbb{Z}/N\mathbb{Z}$	$\xrightarrow{\times}$	$\mathbb{Z}/N\mathbb{Z}$

- Additively homomorphic: Paillier cryptosystem [P99]

$$\begin{aligned}c_1 &= g^{m_1} \bmod N^2 \\c_2 &= g^{m_2} \bmod N^2\end{aligned} \Rightarrow c_1 \cdot c_2 = g^{m_1+m_2} \bmod N^2$$

Ciphertexts

$$\mathbb{Z}/N^2\mathbb{Z} \times \mathbb{Z}/N^2\mathbb{Z} \xrightarrow{\times} \mathbb{Z}/N^2\mathbb{Z}$$

$$\downarrow \delta, \delta$$

Plaintexts

$$\mathbb{Z}/N\mathbb{Z} \times \mathbb{Z}/N\mathbb{Z} \xrightarrow{+} \mathbb{Z}/N\mathbb{Z}$$

# Application of Paillier Cryptosystem

- Additively homomorphic: Paillier cryptosystem

$$\begin{aligned}c_1 &= g^{m_1} \bmod N^2 \\c_2 &= g^{m_2} \bmod N^2\end{aligned} \Rightarrow c_1 \cdot c_2 = g^{m_1+m_2} \bmod N^2$$

- Application: e-voting.

- Voter  $i$  encrypts his vote  $m_i \in \{0, 1\}$  into:

$$c_i = g^{m_i} \cdot z_i^N \bmod N^2$$

- Votes can be aggregated using only the public-key:

$$c = \prod_i c_i = g^{\sum_i m_i} \cdot z \bmod N^2$$

- $c$  is eventually decrypted to recover

$$m = \sum_i m_i$$

# Fully homomorphic encryption

- Multiplicatively homomorphic: RSA.

$$\begin{aligned}c_1 &= m_1^e \bmod N \\c_2 &= m_2^e \bmod N\end{aligned} \Rightarrow c_1 \cdot c_2 = (m_1 \cdot m_2)^e \bmod N$$

- Additively homomorphic: Paillier

$$\begin{aligned}c_1 &= g^{m_1} \bmod N^2 \\c_2 &= g^{m_2} \bmod N^2\end{aligned} \Rightarrow c_1 \cdot c_2 = g^{m_1+m_2} \bmod N^2$$

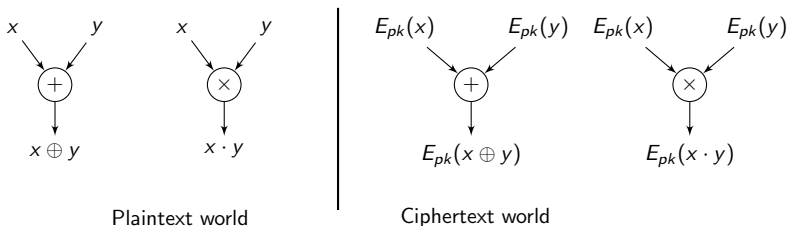
- Fully homomorphic: homomorphic for both addition and multiplication
  - Open problem until Gentry's breakthrough in 2009.

# Fully homomorphic public-key encryption

- We restrict ourselves to public-key encryption of a single bit:
  - $0 \xrightarrow{E_{pk}} 203ef6124 \dots 23ab87_{16}$ ,  $1 \xrightarrow{E_{pk}} b327653c1 \dots db3265_{16}$
  - Encryption must be probabilistic.
- Fully homomorphic property
  - Given  $E_{pk}(x)$  and  $E_{pk}(y)$ , one can compute  $E_{pk}(x \oplus y)$  and  $E_{pk}(x \cdot y)$  without knowing the private-key.

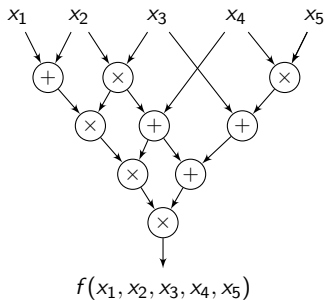
# Fully homomorphic public-key encryption

- We restrict ourselves to public-key encryption of a single bit:
  - $0 \xrightarrow{E_{pk}} 203ef6124 \dots 23ab87_{16}$ ,  $1 \xrightarrow{E_{pk}} b327653c1 \dots db3265_{16}$
  - Encryption must be probabilistic.
- Fully homomorphic property
  - Given  $E_{pk}(x)$  and  $E_{pk}(y)$ , one can compute  $E_{pk}(x \oplus y)$  and  $E_{pk}(x \cdot y)$  without knowing the private-key.

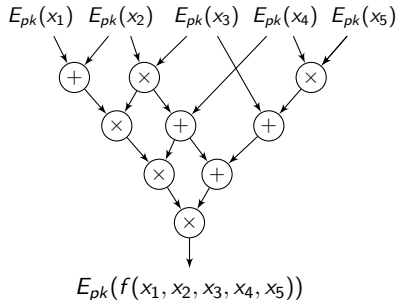


# Evaluation of any function

- Universality
  - We can evaluate homomorphically any boolean computable function  $f : \{0, 1\}^n \rightarrow \{0, 1\}$

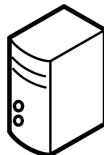


Plaintext world



Ciphertext world

# Outsourcing computation (1)

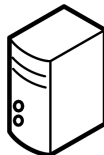
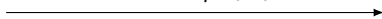


- Alice wants to outsource the computation of  $f(x)$ 
  - but she wants to keep  $x$  private
- She encrypts the bits  $x_i$  of  $x$  into  $c_i = E_{pk}(x_i)$  for her  $pk$ 
  - and she sends the  $c_i$ 's to the server

# Outsourcing computation (1)



$$c_i = E_{pk}(x_i)$$

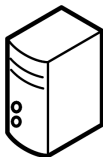


- Alice wants to outsource the computation of  $f(x)$ 
  - but she wants to keep  $x$  private
- She encrypts the bits  $x_i$  of  $x$  into  $c_i = E_{pk}(x_i)$  for her  $pk$ 
  - and she sends the  $c_i$ 's to the server

# Outsourcing computation (2)



$$c_i = E_{pk}(x_i)$$

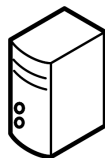


- The server homomorphically evaluates  $f(x)$ 
  - by writing  $f(x) = f(x_1, \dots, x_n)$  as a boolean circuit.
  - Given  $E_{pk}(x_i)$ , the server eventually obtains  $c = E_{pk}(f(x))$
- Finally Alice decrypts  $c$  into  $y = f(x)$ 
  - The server does not learn  $x$ .
  - Only Alice can decrypt to recover  $f(x)$ .
  - Alice could also keep  $f$  private.

# Outsourcing computation (2)



$$\begin{array}{c} \xrightarrow{c_i = E_{pk}(x_i)} \\ \xleftarrow{c = E_{pk}(f(x))} \end{array}$$



- The server homomorphically evaluates  $f(x)$ 
  - by writing  $f(x) = f(x_1, \dots, x_n)$  as a boolean circuit.
  - Given  $E_{pk}(x_i)$ , the server eventually obtains  $c = E_{pk}(f(x))$
- Finally Alice decrypts  $c$  into  $y = f(x)$ 
  - The server does not learn  $x$ .
  - Only Alice can decrypt to recover  $f(x)$ .
  - Alice could also keep  $f$  private.

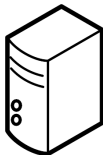
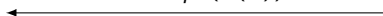
# Outsourcing computation (2)



$$c_i = E_{pk}(x_i)$$



$$c = E_{pk}(f(x))$$



$$y = D_{sk}(c) = f(x)$$

- The server homomorphically evaluates  $f(x)$ 
  - by writing  $f(x) = f(x_1, \dots, x_n)$  as a boolean circuit.
  - Given  $E_{pk}(x_i)$ , the server eventually obtains  $c = E_{pk}(f(x))$
- Finally Alice decrypts  $c$  into  $y = f(x)$ 
  - The server does not learn  $x$ .
  - Only Alice can decrypt to recover  $f(x)$ .
  - Alice could also keep  $f$  private.

# Fully Homomorphic Encryption: first generation

- 1. Breakthrough scheme of Gentry [G09], based on ideal lattices. Some optimizations by [SV10].
  - Implementation [GH11]: PK size: 2.3 GB, recrypt: 30 min.
- 2. van Dijk, Gentry, Halevi and Vaikuntanathan's scheme over the integers [DGHV10].
  - Implementation [CMNT11]: PK size: 1 GB, recrypt: 15 min.
  - Public-key compression [CNT12]
  - Batch and homomorphic evaluation of AES [CCKLLTY13].

# Fully Homomorphic Encryption: first generation

- 1. Breakthrough scheme of Gentry [G09], based on ideal lattices. Some optimizations by [SV10].
  - Implementation [GH11]: PK size: 2.3 GB, recrypt: 30 min.
- 2. van Dijk, Gentry, Halevi and Vaikuntanathan's scheme over the integers [DGHV10].
  - Implementation [CMNT11]: PK size: 1 GB, recrypt: 15 min.
  - Public-key compression [CNT12]
  - Batch and homomorphic evaluation of AES [CCKLLTY13].

# The DGHV Scheme

- Ciphertext for  $m \in \{0, 1\}$ :

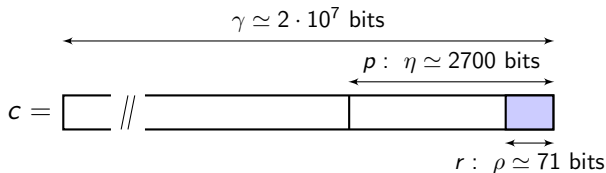
$$c = q \cdot p + 2r + m$$

where  $p$  is the secret-key,  $q$  and  $r$  are randoms.

- Decryption:

$$(c \bmod p) \bmod 2 = m$$

- Parameters:



# Homomorphic Properties of DGHV

- Addition:

$$\begin{aligned}c_1 &= q_1 \cdot p + 2r_1 + m_1 \\c_2 &= q_2 \cdot p + 2r_2 + m_2\end{aligned} \Rightarrow c_1 + c_2 = q' \cdot p + 2r' + m_1 + m_2$$

- $c_1 + c_2$  is an encryption of  $m_1 + m_2 \bmod 2 = m_1 \oplus m_2$

- Multiplication:

$$\begin{aligned}c_1 &= q_1 \cdot p + 2r_1 + m_1 \\c_2 &= q_2 \cdot p + 2r_2 + m_2\end{aligned} \Rightarrow c_1 \cdot c_2 = q'' \cdot p + 2r'' + m_1 \cdot m_2$$

with

$$r'' = 2r_1r_2 + r_1m_2 + r_2m_1$$

- $c_1 \cdot c_2$  is an encryption of  $m_1 \cdot m_2$
- Noise becomes twice larger.

# Homomorphic Properties of DGHV

- Addition:

$$\begin{aligned}c_1 &= q_1 \cdot p + 2r_1 + m_1 \\c_2 &= q_2 \cdot p + 2r_2 + m_2\end{aligned} \Rightarrow c_1 + c_2 = q' \cdot p + 2r' + m_1 + m_2$$

- $c_1 + c_2$  is an encryption of  $m_1 + m_2 \bmod 2 = m_1 \oplus m_2$

- Multiplication:

$$\begin{aligned}c_1 &= q_1 \cdot p + 2r_1 + m_1 \\c_2 &= q_2 \cdot p + 2r_2 + m_2\end{aligned} \Rightarrow c_1 \cdot c_2 = q'' \cdot p + 2r'' + m_1 \cdot m_2$$

with

$$r'' = 2r_1r_2 + r_1m_2 + r_2m_1$$

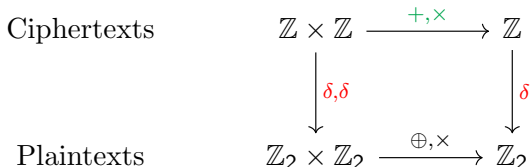
- $c_1 \cdot c_2$  is an encryption of  $m_1 \cdot m_2$
- Noise becomes twice larger.

# Homomorphism of DGHV

- DGHV ciphertext:

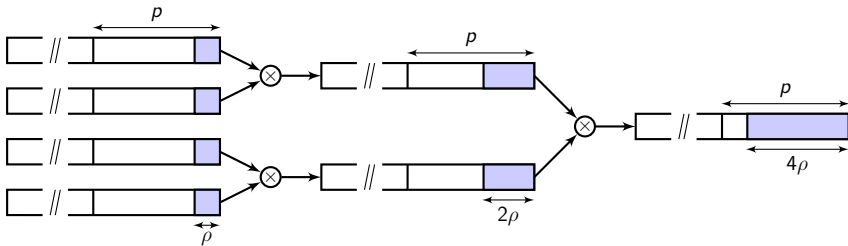
$$c = q \cdot p + 2r + m$$

- Homomorphism:  $\delta(x) = (x \bmod p) \bmod 2$ 
  - only works if noise  $r$  is smaller than  $p$



# Somewhat homomorphic scheme

- The number of multiplications is limited.
  - Noise grows with the number of multiplications.
  - Noise must remain  $< p$  for correct decryption.



# Public-key Encryption with DGHV

- For now, encryption requires the knowledge of the secret  $p$ :

$$c = q \cdot p + 2r + m$$

- We can actually turn it into a public-key encryption scheme
  - Using the additively homomorphic property
- Public-key: a set of  $\tau$  encryptions of 0's.

$$x_i = q_i \cdot p + 2r_i$$

- Public-key encryption:

$$c = m + 2r + \sum_{i=1}^{\tau} \varepsilon_i \cdot x_i$$

for random  $\varepsilon_i \in \{0, 1\}$ .

# Public-key Encryption with DGHV

- For now, encryption requires the knowledge of the secret  $p$ :

$$c = q \cdot p + 2r + m$$

- We can actually turn it into a public-key encryption scheme
  - Using the additively homomorphic property
- Public-key: a set of  $\tau$  encryptions of 0's.

$$x_i = q_i \cdot p + 2r_i$$

- Public-key encryption:

$$c = m + 2r + \sum_{i=1}^{\tau} \varepsilon_i \cdot x_i$$

for random  $\varepsilon_i \in \{0, 1\}$ .

# Bounding ciphertext size

- DGHV multiplication over  $\mathbb{Z}$

$$\begin{aligned}c_1 &= q_1 \cdot p + 2r_1 + m_1 \\c_2 &= q_2 \cdot p + 2r_2 + m_2\end{aligned} \Rightarrow c_1 \cdot c_2 = q' \cdot p + 2r' + m_1 \cdot m_2$$

- Problem: ciphertext size has doubled.
- Constant ciphertext size
  - We publish an encryption of 0 without noise  $x_0 = q_0 \cdot p$
  - We reduce the product modulo  $x_0$

$$\begin{aligned}c_3 &= c_1 \cdot c_2 \bmod x_0 \\&= q'' \cdot p + 2r' + m_1 \cdot m_2\end{aligned}$$

- Ciphertext size remains constant

# Semantic security of DGHV

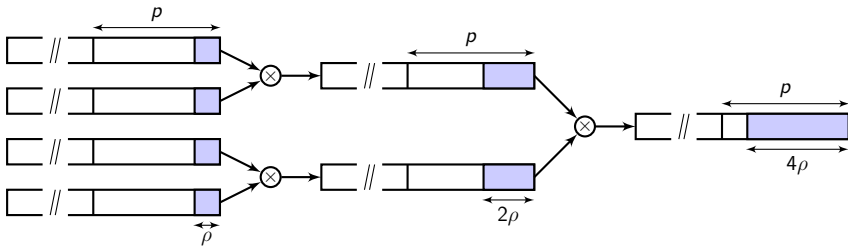
- Semantic security [GM82] for  $m \in \{0, 1\}$ :
  - Knowing  $pk$ , the distributions  $E_{pk}(0)$  and  $E_{pk}(1)$  are computationally hard to distinguish.
- The DGHV scheme is semantically secure, under the approximate-gcd assumption.
  - Approximate-gcd problem: given a set of  $x_i = q_i \cdot p + r_i$ , recover  $p$ .

# The approximate GCD assumption

- Efficient DGHV variant: secure under the **Partial Approximate Common Divisor** (PACD) assumption.
  - Given  $x_0 = p \cdot q_0$  and polynomially many  $x_i = p \cdot q_i + r_i$ , find  $p$ .
- Brute force attack on the noise
  - Given  $x_0 = q_0 \cdot p$  and  $x_1 = q_1 \cdot p + r_1$  with  $|r_1| < 2^\rho$ , guess  $r_1$  and compute  $\gcd(x_0, x_1 - r_1)$  to recover  $p$ .
  - Requires  $2^\rho$  gcd computation
  - Improved attack against PACD [CN12], with complexity  $\tilde{O}(2^{\rho/2})$
  - Countermeasure: take a sufficiently large  $\rho$
- Orthogonal lattice attack with LLL
  - Construct the lattice of vectors orthogonal to the  $x_i$ 's.
  - Apply LLL to recover the secret  $p$ .
  - Countermeasure: take a sufficiently large size  $\gamma$  of the  $x_i$ 's.

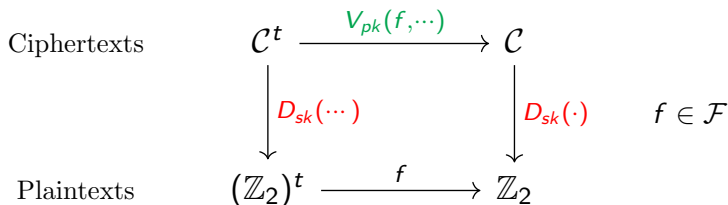
# Somewhat homomorphic scheme

- The number of multiplications is limited.
  - Noise grows with the number of multiplications.
  - Noise must remain  $< p$  for correct decryption.



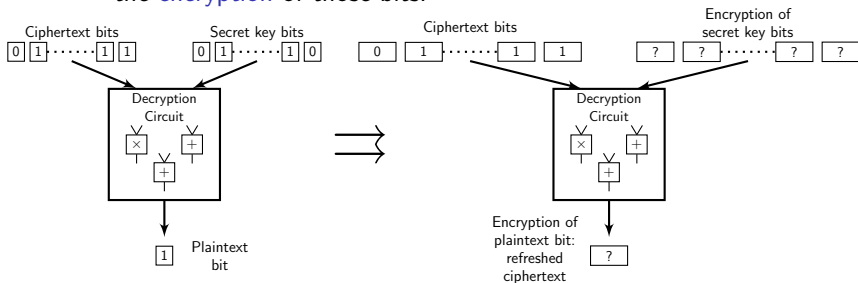
# Gentry's technique to get fully homomorphic encryption

- 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  $\mathcal{F} = \{f(b_1, \dots, b_t) : \deg f \leq d\}$
  - $V_{pk}(f, E_{pk}(b_1), \dots, E_{pk}(b_t)) \rightarrow E_{pk}(f(b_1, \dots, b_t))$



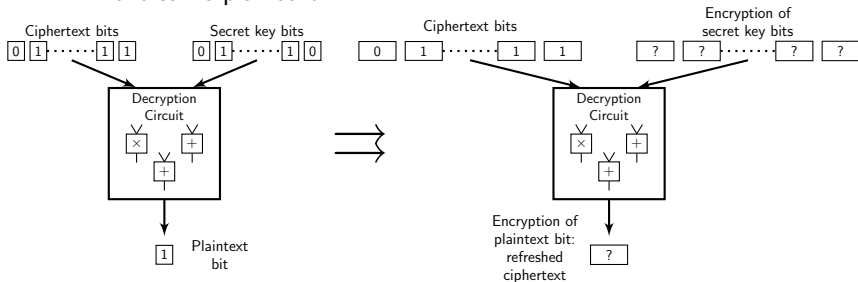
# Ciphertext refresh: bootstrapping

- 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

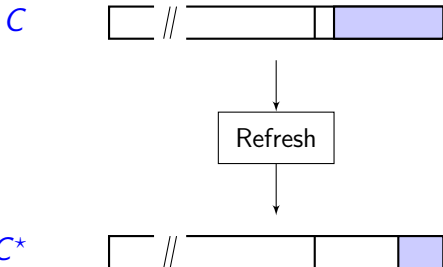
- Gentry's breakthrough idea: refresh the ciphertext using the decryption circuit homomorphically.
  - Instead of recovering the bit plaintext  $m$ , one gets an encryption of this bit plaintext, *i.e.* yet another ciphertext for the same plaintext.



- will be explained in next lecture.

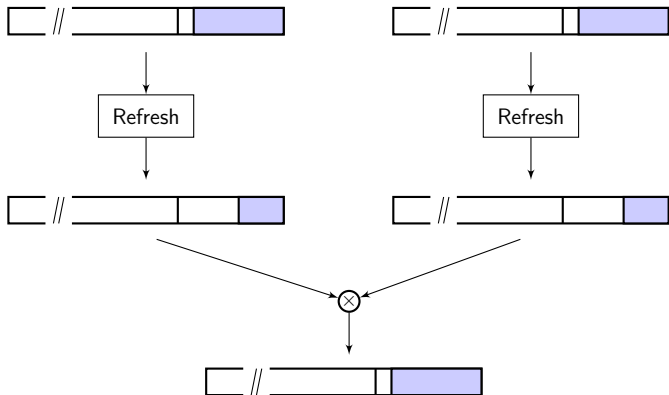
# Ciphertext refresh

- 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.



# 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.



# 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

# Second generation: LWE-based encryption

- Homomorphic encryption based on polynomial evaluation
  - Homomorphism:  $\delta : \mathbb{Z}_q[\vec{x}] \rightarrow \mathbb{Z}_q[x]$  given by evaluation at secret  $\vec{s} = (s_1, \dots, s_n)$

$$\begin{array}{ccc} \text{Ciphertexts} & \mathbb{Z}_q[\vec{x}] \times \mathbb{Z}_q[\vec{x}] & \xrightarrow{+, \times} \mathbb{Z}_q[\vec{x}] \\ & \downarrow \delta, \delta & \downarrow \delta \\ \text{Plaintexts} & \mathbb{Z}_q \times \mathbb{Z}_q & \xrightarrow{+, \times} \mathbb{Z}_q \end{array}$$

- One must add some noise, otherwise broken by linear algebra.
  - $f(\vec{s}) = 2e + m \bmod q$ , for some small noise  $e \in \mathbb{Z}_q$
- LWE assumption [R05]
  - Linear polynomials  $f_i(\vec{x})$  with  $|f_i(\vec{s}) \bmod q| \ll q$  are comp. indist. from random  $f_i(\vec{x})$  modulo  $q$ .

# Second generation: LWE-based encryption

- Homomorphic encryption based on polynomial evaluation
  - Homomorphism:  $\delta : \mathbb{Z}_q[\vec{x}] \rightarrow \mathbb{Z}_q[x]$  given by evaluation at secret  $\vec{s} = (s_1, \dots, s_n)$

$$\begin{array}{ccc} \text{Ciphertexts} & \mathbb{Z}_q[\vec{x}] \times \mathbb{Z}_q[\vec{x}] & \xrightarrow{+, \times} \mathbb{Z}_q[\vec{x}] \\ & \downarrow \delta, \delta & \downarrow \delta \\ \text{Plaintexts} & \mathbb{Z}_q \times \mathbb{Z}_q & \xrightarrow{+, \times} \mathbb{Z}_q \end{array}$$

- One must add some noise, otherwise broken by linear algebra.
  - $f(\vec{s}) = 2e + m \bmod q$ , for some small noise  $e \in \mathbb{Z}_q$
- LWE assumption [R05]
  - Linear polynomials  $f_i(\vec{x})$  with  $|f_i(\vec{s}) \bmod q| \ll q$  are comp. indist. from random  $f_i(\vec{x})$  modulo  $q$ .

# Second generation: LWE-based encryption

- Homomorphic encryption based on polynomial evaluation
  - Homomorphism:  $\delta : \mathbb{Z}_q[\vec{x}] \rightarrow \mathbb{Z}_q[x]$  given by evaluation at secret  $\vec{s} = (s_1, \dots, s_n)$

$$\begin{array}{ccc} \text{Ciphertexts} & \mathbb{Z}_q[\vec{x}] \times \mathbb{Z}_q[\vec{x}] & \xrightarrow{+, \times} \mathbb{Z}_q[\vec{x}] \\ & \downarrow \delta, \delta & \downarrow \delta \\ \text{Plaintexts} & \mathbb{Z}_q \times \mathbb{Z}_q & \xrightarrow{+, \times} \mathbb{Z}_q \end{array}$$

- One must add some noise, otherwise broken by linear algebra.
  - $f(\vec{s}) = 2e + m \bmod q$ , for some small noise  $e \in \mathbb{Z}_q$
- LWE assumption [R05]
  - Linear polynomials  $f_i(\vec{x})$  with  $|f_i(\vec{s}) \bmod q| \ll q$  are comp. indist. from random  $f_i(\vec{x})$  modulo  $q$ .

# Third generation of FHE: ciphertext matrices

- Homomorphic encryption with matrices [GSW13]
  - Ciphertexts are square matrices instead of vectors
  - Homomorphism:  $\delta(C, \mathbf{v}) = \mu$  where  $\mu$  is eigenvalue for secret eigenvector  $\mathbf{v}$
  - Homomorphically add and multiply ciphertext using (roughly) matrix addition and multiplication

$$\begin{array}{ccc} \text{Ciphertexts} & \mathbb{Z}^{N \times N} \times \mathbb{Z}^{N \times N} & \xrightarrow{+, \times} \mathbb{Z}^{N \times N} \\ & \downarrow \delta, \delta & \downarrow \delta \\ \text{Plaintexts} & \mathbb{Z} \times \mathbb{Z} & \xrightarrow{+, \times} \mathbb{Z} \end{array}$$

- One must add some noise, otherwise broken by linear algebra
  - $C \cdot \mathbf{v} = \mu \cdot \mathbf{v} + \mathbf{e} \pmod{q}$
  - for message  $\mu \in \mathbb{Z}$ , for some small noise  $\mathbf{e}$ .
  - Security based on LWE problem.

# Third generation of FHE: ciphertext matrices

- Homomorphic encryption with matrices [GSW13]
  - Ciphertexts are square matrices instead of vectors
  - Homomorphism:  $\delta(C, \mathbf{v}) = \mu$  where  $\mu$  is eigenvalue for secret eigenvector  $\mathbf{v}$
  - Homomorphically add and multiply ciphertext using (roughly) matrix addition and multiplication

$$\begin{array}{ccc} \text{Ciphertexts} & \mathbb{Z}^{N \times N} \times \mathbb{Z}^{N \times N} & \xrightarrow{+, \times} \mathbb{Z}^{N \times N} \\ & \downarrow \delta, \delta & \downarrow \delta \\ \text{Plaintexts} & \mathbb{Z} \times \mathbb{Z} & \xrightarrow{+, \times} \mathbb{Z} \end{array}$$

- One must add some noise, otherwise broken by linear algebra
  - $C \cdot \mathbf{v} = \mu \cdot \mathbf{v} + \mathbf{e} \pmod{q}$
  - for message  $\mu \in \mathbb{Z}$ , for some small noise  $\mathbf{e}$ .
  - Security based on LWE problem.

# Fourth generation: homomorphic encryption for approximate numbers

- Homomorphic encryption for real numbers [CKKS17]
  - Floating point arithmetic, instead of exact arithmetic.
  - Starting point: Regev's scheme.
  - Homomorphism:  $\delta : \mathbb{Z}_q[\mathbf{x}] \rightarrow \mathbb{Z}_q$  given by evaluation at  $\mathbf{s}$

$$\begin{array}{ccc} \text{Ciphertexts} & \mathbb{Z}_q[\mathbf{x}] \times \mathbb{Z}_q[\mathbf{x}] & \xrightarrow{+, \times} \mathbb{Z}_q[\mathbf{x}] \\ & \downarrow \delta, \delta & \downarrow \delta \\ \text{Plaintexts} & \mathbb{Z}_q \times \mathbb{Z}_q & \xrightarrow{+, \times} \mathbb{Z}_q \end{array}$$

- One must add some noise, otherwise broken by linear algebra.
  - $f(\mathbf{s}) = m + e \pmod q$ , for small  $e \in \mathbb{Z}_q$
  - Noise only affects the low-order bits of  $m$ : approximate computation, as in floating point arithmetic.
  - Application: neural networks.

# Fourth generation: homomorphic encryption for approximate numbers

- Homomorphic encryption for real numbers [CKKS17]
  - Floating point arithmetic, instead of exact arithmetic.
  - Starting point: Regev's scheme.
  - Homomorphism:  $\delta : \mathbb{Z}_q[\mathbf{x}] \rightarrow \mathbb{Z}_q$  given by evaluation at  $\mathbf{s}$

$$\begin{array}{ccc} \text{Ciphertexts} & \mathbb{Z}_q[\mathbf{x}] \times \mathbb{Z}_q[\mathbf{x}] & \xrightarrow{+, \times} \mathbb{Z}_q[\mathbf{x}] \\ & \downarrow \delta, \delta & \downarrow \delta \\ \text{Plaintexts} & \mathbb{Z}_q \times \mathbb{Z}_q & \xrightarrow{+, \times} \mathbb{Z}_q \end{array}$$

- One must add some noise, otherwise broken by linear algebra.
  - $f(\mathbf{s}) = m + e \pmod q$ , for small  $e \in \mathbb{Z}_q$
  - Noise only affects the low-order bits of  $m$ : approximate computation, as in floating point arithmetic.
  - Application: neural networks.

- First generation of fully homomorphic encryption
  - The DGHV scheme
  - Overview of bootstrapping
- Next lecture
  - (R)LWE-based encryption
  - Ciphertext multiplication: relinearization
  - Bootstrapping explained

# References

- CN12 Yuanmi Chen, Phong Q. Nguyen. Faster Algorithms for Approximate Common Divisors: Breaking Fully-Homomorphic-Encryption Challenges over the Integers. EUROCRYPT 2012: 502-519
- CMNT11 Jean-Sébastien Coron, Avradip Mandal, David Naccache, Mehdi Tibouchi: Fully Homomorphic Encryption over the Integers with Shorter Public Keys. CRYPTO 2011: 487-504
- CNT12 Jean-Sébastien Coron, David Naccache, Mehdi Tibouchi. Public Key Compression and Modulus Switching for Fully Homomorphic Encryption over the Integers. EUROCRYPT 2012: 446-464
- DGHV10 Marten van Dijk, Craig Gentry, Shai Halevi, Vinod Vaikuntanathan. Fully Homomorphic Encryption over the Integers. EUROCRYPT 2010: 24-43
- Gen09 Craig Gentry. Fully homomorphic encryption using ideal lattices. STOC 2009: 169-178
- P99 Pascal Paillier. Public-Key Cryptosystems Based on Composite Degree Residuosity Classes. EUROCRYPT 1999: 223-238
- R05 Oded Regev. On lattices, learning with errors, random linear codes, and cryptography. STOC 2005: 84-93