Introduction to Fully Homomorphic Encryption

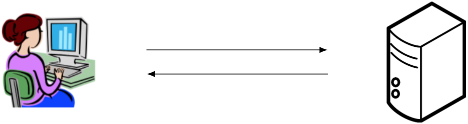
Part 1: basic techniques

Jean-Sébastien Coron

University of Luxembourg

Overview

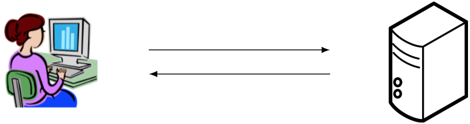
- What is Fully Homomorphic Encryption (FHE) ?
  - Basic properties
  - Cloud computing on encrypted data: the server should process the data without learning the data.



- 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

Overview

- 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.

$$\begin{aligned} \mathrm{AES}_K(m_1) &= \texttt{0x3c7317c6bc5634a4ad8479c64714f4f8} \\ \mathrm{AES}_K(m_2) &= \texttt{0x7619884e1961b051be1aa407da6cac2c} \\ \mathrm{AES}_K(m_1 \oplus m_2) &= \; ? \end{aligned}$$

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

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

Homomorphic Encryption

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

$$
\begin{aligned}
\mathrm{AES}_K(m_1) &= \texttt{0x3c7317c6bc5634a4ad8479c64714f4f8} \\
\mathrm{AES}_K(m_2) &= \texttt{0x7619884e1961b051be1aa407da6cac2c} \\
\mathrm{AES}_K(m_1 \oplus m_2) &= \ ?
\end{aligned}
$$

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

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

Homomorphic Encryption with RSA

- Multiplicative property of RSA.

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

- 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 $\delta(x) = x^d \bmod N$

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

$$\begin{array}{lccc}
\text{Ciphertexts} & \mathbb{Z}/N\mathbb{Z} \times \mathbb{Z}/N\mathbb{Z} & \xrightarrow{\;\times\;} & \mathbb{Z}/N\mathbb{Z} \\
 & \downarrow{\scriptstyle \delta,\delta} & & \downarrow{\scriptstyle \delta} \\
\text{Plaintexts} & \mathbb{Z}/N\mathbb{Z} \times \mathbb{Z}/N\mathbb{Z} & \xrightarrow{\;\times\;} & \mathbb{Z}/N\mathbb{Z}
\end{array}$$

Paillier Cryptosystem

- Additively homomorphic: Paillier cryptosystem [P99]

$$\left.\begin{array}{l} c_1 = g^{m_1} \bmod N^2 \\ c_2 = g^{m_2} \bmod N^2 \end{array}\right. \Rightarrow c_1 \cdot c_2 = g^{m_1+m_2\,[N]} \bmod N^2$$

$$\begin{array}{lccc}
\text{Ciphertexts} & \mathbb{Z}/N^2\mathbb{Z} \times \mathbb{Z}/N^2\mathbb{Z} & \xrightarrow{\times} & \mathbb{Z}/N^2\mathbb{Z} \\
 & \big\downarrow \delta,\delta & & \big\downarrow \delta \\
\text{Plaintexts} & \mathbb{Z}/N\mathbb{Z} \times \mathbb{Z}/N\mathbb{Z} & \xrightarrow{+} & \mathbb{Z}/N\mathbb{Z}
\end{array}$$

Application of Paillier Cryptosystem

- Additively homomorphic: Paillier cryptosystem

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

- 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.

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

- Additively homomorphic: Paillier

$$\left.\begin{aligned} c_1 &= g^{m_1} \bmod N^2 \\ c_2 &= g^{m_2} \bmod N^2 \end{aligned}\right. \Rightarrow c_1 \cdot c_2 = g^{m_1+m_2\,[N]} \bmod N^2$$

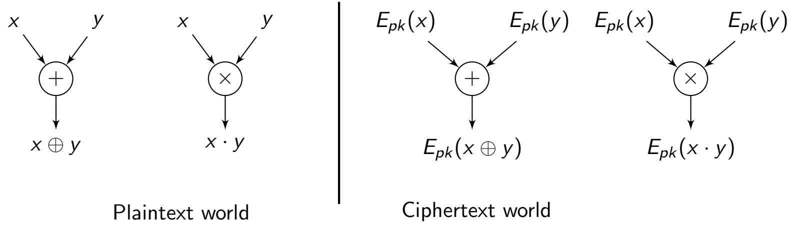
- Fully homomorphic: homomorphic for both addition and multiplication
  - Open problem until Gentry's breakthrough in 2009.

Fully homomorphic public-key encryption

- We restrict ourselves to public-key encryption of a single bit:
  - $0 \xrightarrow{E_{pk}} 203\text{ef}6124\ldots23\text{ab}87_{16}$, $1 \xrightarrow{E_{pk}} \text{b}327653\text{c}1\ldots\text{db}3265_{16}$
  - 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.

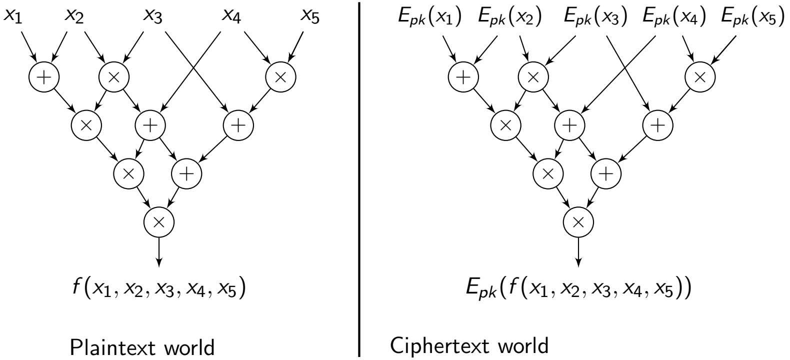
Fully homomorphic public-key encryption

- We restrict ourselves to public-key encryption of a single bit:
  - $0 \xrightarrow{E_{pk}} 203ef6124\ldots23ab87_{16}$, $1 \xrightarrow{E_{pk}} b327653c1\ldots db3265_{16}$
  - 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.

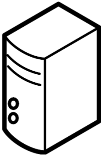


Evaluation of any function

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



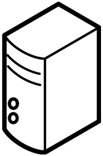
Outsourcing computation (1)



- 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)

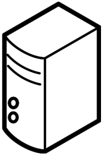
$$c_i = E_{pk}(x_i)$$



- 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)

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

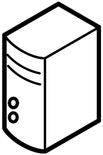


- The server homomorphically evaluates $f(x)$
  - by writing $f(x) = f(x_1, \ldots, x_n)$ as a boolean circuit.
  - Given $E_{pk}(x_i)$, the server eventually obtains $c = E_{pk}(f(x))$
- Finally Alice decrypts c into $y = f(x)$
  - The server does not learn x.
  - Only Alice can decrypt to recover $f(x)$.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

Outsourcing computation (2)

$c_i = E_{pk}(x_i)$ →

← $c = E_{pk}(f(x))$

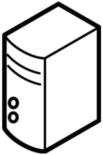


- The server homomorphically evaluates $f(x)$
  - by writing $f(x) = f(x_1, \ldots, x_n)$ as a boolean circuit.
  - Given $E_{pk}(x_i)$, the server eventually obtains $c = E_{pk}(f(x))$
- Finally Alice decrypts c into $y = f(x)$
  - The server does not learn x.
  - Only Alice can decrypt to recover $f(x)$.
  - 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 $f(x) = f(x_1, \ldots, x_n)$ 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 $m \in \{0,1\}$:

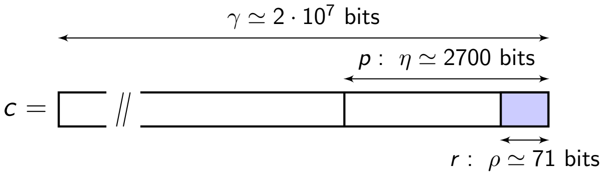
$$c = q \cdot p + 2r + m$$

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

- Decryption:

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

- Parameters:



Homomorphic Properties of DGHV

- Addition:

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

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

- Multiplication:

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

  with

$$r'' = 2r_1 r_2 + r_1 m_2 + r_2 m_1$$

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

Homomorphic Properties of DGHV

- Addition:

$$\left.\begin{array}{l} c_1 = q_1 \cdot p + 2r_1 + m_1 \\ c_2 = q_2 \cdot p + 2r_2 + m_2 \end{array}\right. \Rightarrow c_1 + c_2 = q' \cdot p + 2r' + m_1 + m_2$$

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

- Multiplication:

$$\left.\begin{array}{l} c_1 = q_1 \cdot p + 2r_1 + m_1 \\ c_2 = q_2 \cdot p + 2r_2 + m_2 \end{array}\right. \Rightarrow c_1 \cdot c_2 = q'' \cdot p + 2r'' + m_1 \cdot m_2$$

  with

$$r'' = 2r_1 r_2 + r_1 m_2 + r_2 m_1$$

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

- DGHV ciphertext:

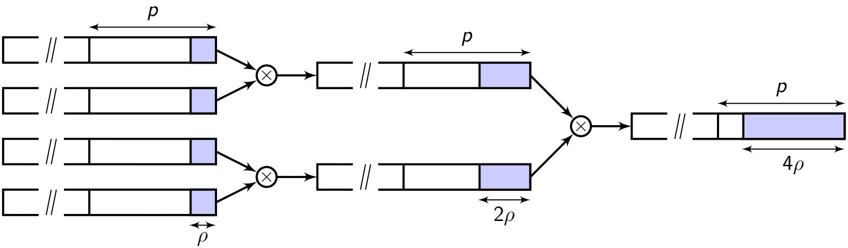
$$c = q \cdot p + 2r + m$$

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

$$\begin{array}{lccc}
\text{Ciphertexts} & \mathbb{Z} \times \mathbb{Z} & \xrightarrow{+,\times} & \mathbb{Z} \\
 & \downarrow{\scriptstyle \delta,\delta} & & \downarrow{\scriptstyle \delta} \\
\text{Plaintexts} & \mathbb{Z}_2 \times \mathbb{Z}_2 & \xrightarrow{\oplus,\times} & \mathbb{Z}_2
\end{array}$$

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 τ 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 τ 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 $\mathbb{Z}$

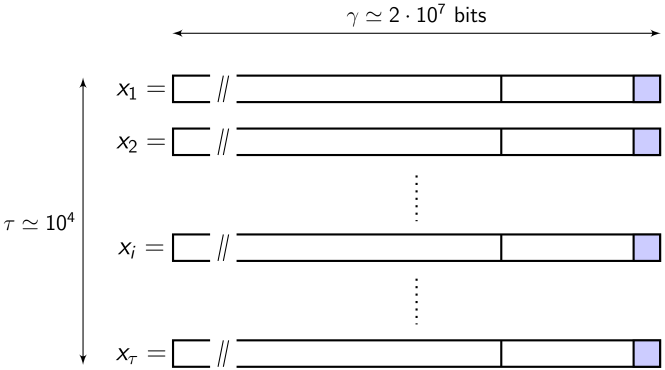
$$\left.\begin{aligned} c_1 &= q_1 \cdot p + 2r_1 + m_1 \\ c_2 &= q_2 \cdot p + 2r_2 + m_2 \end{aligned}\right. \Rightarrow c_1 \cdot c_2 = q' \cdot p + 2r' + m_1 \cdot m_2$$

  - 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

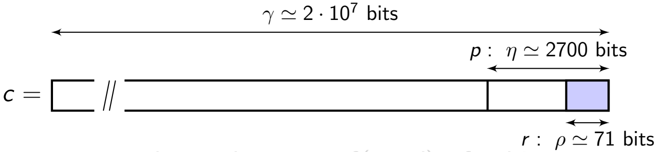
Public-key size



- Public-key size:
  - $\tau \cdot \gamma = 2 \cdot 10^{11}$ bits $= 25$ GB !

DGHV Ciphertext Compression

- Ciphertext: $c = q \cdot p + 2r + m$

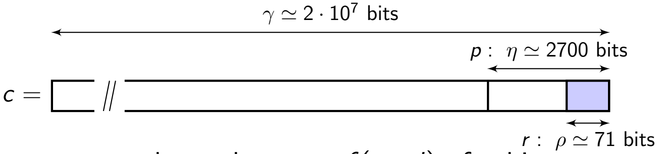


- Compute a pseudo-random $\chi = f(seed)$ of γ bits.

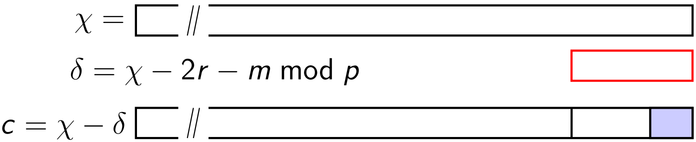
- Only store *seed* and the small correction δ.
- Storage: $\simeq 2\,700$ bits instead of $2 \cdot 10^7$ bits !

DGHV Ciphertext Compression

- Ciphertext: $c = q \cdot p + 2r + m$



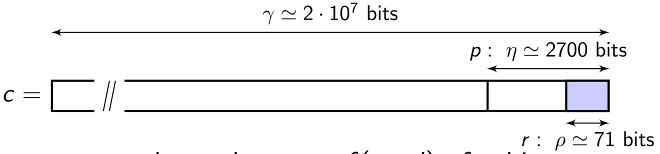
- Compute a pseudo-random $\chi = f(seed)$ of γ bits.



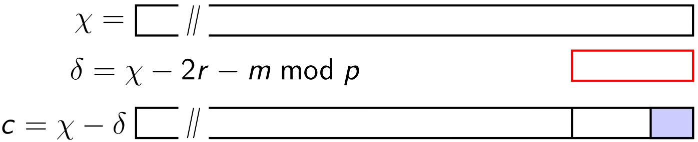
  - Only store *seed* and the small correction δ.
  - Storage: $\simeq 2\,700$ bits instead of $2 \cdot 10^7$ bits !

DGHV Ciphertext Compression

- Ciphertext: $c = q \cdot p + 2r + m$



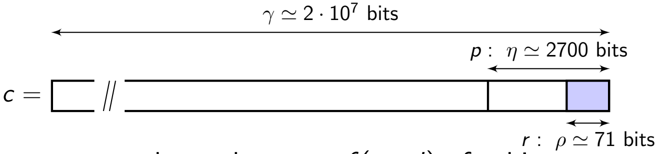
- Compute a pseudo-random $\chi = f(seed)$ of γ bits.



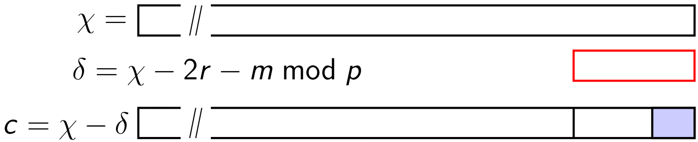
  - Only store *seed* and the small correction δ.
  - Storage: $\simeq 2\,700$ bits instead of $2 \cdot 10^7$ bits !

DGHV Ciphertext Compression

- Ciphertext: $c = q \cdot p + 2r + m$

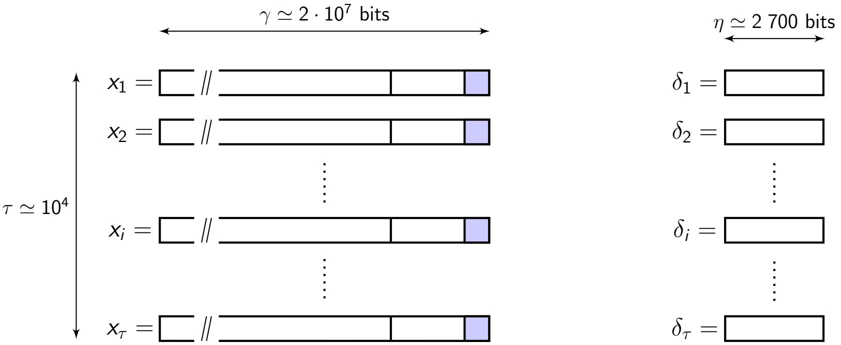


- Compute a pseudo-random $\chi = f(seed)$ of γ bits.



  - Only store *seed* and the small correction δ.
  - Storage: $\simeq 2\,700$ bits instead of $2 \cdot 10^7$ bits !

Compressed Public Key



Old *pk*: 25 GB

New *pk*: 3.4 MB !

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.
  - This remains the case with the compressed public-key, under the random oracle model.

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^ρ gcd computation
  - Countermeasure: take a sufficiently large ρ

Improved attack against PACD [CN12]

- Given $x_0 = p \cdot q_0$ and many $x_i = p \cdot q_i + r_i$, find p.
- Improved attack in $\tilde{\mathcal{O}}(2^{\rho/2})$ [CN12]

$$
\begin{aligned}
p &= \gcd\left(x_0, \prod_{i=0}^{2^\rho-1} (x_1 - i) \bmod x_0\right) \\
&= \gcd\left(x_0, \prod_{a=0}^{m-1} \prod_{b=0}^{m-1} (x_1 - b - m \cdot a) \bmod x_0\right), \text{ where } m = 2^{\rho/2} \\
&= \gcd\left(x_0, \prod_{a=0}^{m-1} f(a) \bmod x_0\right)
\end{aligned}
$$

- $f(y) := \prod_{b=0}^{m-1} (x_1 - b - m \cdot y) \bmod x_0$
- Evaluate the polynomial $f(y)$ at m points in time $\tilde{\mathcal{O}}(m) = \tilde{\mathcal{O}}(2^{\rho/2})$

Approximate GCD attack

- Consider t integers: $x_i = p \cdot q_i + r_i$ and $x_0 = p \cdot q_0$.
  - Consider a vector $\vec{u}$ orthogonal to the x_i's:

$$\sum_{i=1}^{t} u_i \cdot x_i = 0 \mod x_0$$

  - This gives $\sum_{i=1}^{t} u_i \cdot r_i = 0 \mod p$.
- If the u_i's are sufficiently small, since the r_i's are small this equality will hold over $\mathbb{Z}$.
  - Such vector $\vec{u}$ can be found using LLL.
- By collecting many orthogonal vectors one can recover $\vec{r}$ and eventually the secret key p
- Countermeasure
  - The size γ of the x_i's must be sufficiently large.

The DGHV scheme (simplified)

- Key generation:
  - Generate a set of τ public integers:

$$x_i = p \cdot q_i + r_i, \quad 1 \leq i \leq \tau$$

    and $x_0 = p \cdot q_0$, where p is a secret prime.
  - Size of p is η. Size of x_i is γ. Size of r_i is ρ.
- Encryption of a message $m \in \{0,1\}$:
  - Generate random $\varepsilon_i \leftarrow \{0,1\}$ and a random integer r in $(-2^{\rho'}, 2^{\rho'})$, and output the ciphertext:

$$c = m + 2r + 2\sum_{i=1}^{\tau} \varepsilon_i \cdot x_i \bmod x_0$$

- Decryption:

$$c \equiv m + 2r + 2\sum_{i=1}^{\tau} \varepsilon_i \cdot r_i \pmod p$$

  - Output $m \leftarrow (c \bmod p) \bmod 2$

The DGHV scheme (simplified)

- Key generation:
  - Generate a set of τ public integers:

$$x_i = p \cdot q_i + r_i, \quad 1 \leq i \leq \tau$$

    and $x_0 = p \cdot q_0$, where p is a secret prime.
  - Size of p is η. Size of x_i is γ. Size of r_i is ρ.
- Encryption of a message $m \in \{0, 1\}$:
  - Generate random $\varepsilon_i \leftarrow \{0, 1\}$ and a random integer r in $(-2^{\rho'}, 2^{\rho'})$, and output the ciphertext:

$$c = m + 2r + 2 \sum_{i=1}^{\tau} \varepsilon_i \cdot x_i \bmod x_0$$

- Decryption:

$$c \equiv m + 2r + 2 \sum_{i=1}^{\tau} \varepsilon_i \cdot r_i \pmod{p}$$

  - Output $m \leftarrow (c \bmod p) \bmod 2$

The DGHV scheme (simplified)

- Key generation:
  - Generate a set of τ public integers:

$$x_i = p \cdot q_i + r_i, \quad 1 \leq i \leq \tau$$

    and $x_0 = p \cdot q_0$, where p is a secret prime.
  - Size of p is η. Size of x_i is γ. Size of r_i is ρ.
- Encryption of a message $m \in \{0, 1\}$:
  - Generate random $\varepsilon_i \leftarrow \{0, 1\}$ and a random integer r in $(-2^{\rho'}, 2^{\rho'})$, and output the ciphertext:

$$c = m + 2r + 2\sum_{i=1}^{\tau} \varepsilon_i \cdot x_i \bmod x_0$$

- Decryption:

$$c \equiv m + 2r + 2\sum_{i=1}^{\tau} \varepsilon_i \cdot r_i \pmod{p}$$

  - Output $m \leftarrow (c \bmod p) \bmod 2$

The DGHV scheme (contd.)

- Noise in ciphertext:
  - $c = m + 2 \cdot r' \mod p$ where $r' = r + \sum_{i=1}^{\tau} \varepsilon_i \cdot r_i$
  - r' is the noise in the ciphertext.
  - It must remain $< p$ for correct decryption.
- Homomorphic addition: $c_3 \leftarrow c_1 + c_2 \mod x_0$
  - $c_1 + c_2 = m_1 + m_2 + 2(r'_1 + r'_2) \mod p$
  - Works if noise $r'_1 + r'_2$ still less than p.
- Homomorphic multiplication: $c_3 \leftarrow c_1 \cdot c_2 \mod x_0$
  - $c_1 \cdot c_2 = m_1 \cdot m_2 + 2(m_1 \cdot r'_2 + m_2 \cdot r'_1 + 2r'_1 \cdot r'_2) \mod p$
  - Works if noise $r'_1 \cdot r'_2$ remains less than p.
- Somewhat homomorphic scheme
  - Noise grows with every homomorphic addition or multiplication.
  - This limits the degree of the polynomial that can be applied on ciphertexts.

The DGHV scheme (contd.)

- Noise in ciphertext:
  - $c = m + 2 \cdot r' \mod p$ where $r' = r + \sum_{i=1}^{\tau} \varepsilon_i \cdot r_i$
  - r' is the noise in the ciphertext.
  - It must remain $< p$ for correct decryption.
- Homomorphic addition: $c_3 \leftarrow c_1 + c_2 \mod x_0$
  - $c_1 + c_2 = m_1 + m_2 + 2(r'_1 + r'_2) \mod p$
  - Works if noise $r'_1 + r'_2$ still less than p.
- Homomorphic multiplication: $c_3 \leftarrow c_1 \cdot c_2 \mod x_0$
  - $c_1 \cdot c_2 = m_1 \cdot m_2 + 2(m_1 \cdot r'_2 + m_2 \cdot r'_1 + 2r'_1 \cdot r'_2) \mod p$
  - Works if noise $r'_1 \cdot r'_2$ remains less than p.
- Somewhat homomorphic scheme
  - Noise grows with every homomorphic addition or multiplication.
  - This limits the degree of the polynomial that can be applied on ciphertexts.

The DGHV scheme (contd.)

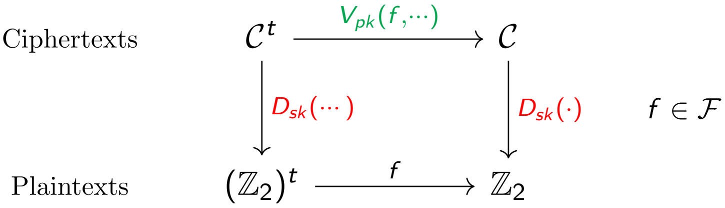
- Noise in ciphertext:
  - $c = m + 2 \cdot r' \mod p$ where $r' = r + \sum_{i=1}^{\tau} \varepsilon_i \cdot r_i$
  - r' is the noise in the ciphertext.
  - It must remain $< p$ for correct decryption.
- Homomorphic addition: $c_3 \leftarrow c_1 + c_2 \mod x_0$
  - $c_1 + c_2 = m_1 + m_2 + 2(r'_1 + r'_2) \mod p$
  - Works if noise $r'_1 + r'_2$ still less than p.
- Homomorphic multiplication: $c_3 \leftarrow c_1 \cdot c_2 \mod x_0$
  - $c_1 \cdot c_2 = m_1 \cdot m_2 + 2(m_1 \cdot r'_2 + m_2 \cdot r'_1 + 2r'_1 \cdot r'_2) \mod p$
  - Works if noise $r'_1 \cdot r'_2$ remains less than p.
- Somewhat homomorphic scheme
  - Noise grows with every homomorphic addition or multiplication.
  - This limits the degree of the polynomial that can be applied on ciphertexts.

The DGHV scheme (contd.)

- Noise in ciphertext:
  - $c = m + 2 \cdot r' \mod p$ where $r' = r + \sum_{i=1}^{\tau} \varepsilon_i \cdot r_i$
  - r' is the noise in the ciphertext.
  - It must remain $< p$ for correct decryption.
- Homomorphic addition: $c_3 \leftarrow c_1 + c_2 \mod x_0$
  - $c_1 + c_2 = m_1 + m_2 + 2(r'_1 + r'_2) \mod p$
  - Works if noise $r'_1 + r'_2$ still less than p.
- Homomorphic multiplication: $c_3 \leftarrow c_1 \cdot c_2 \mod x_0$
  - $c_1 \cdot c_2 = m_1 \cdot m_2 + 2(m_1 \cdot r'_2 + m_2 \cdot r'_1 + 2r'_1 \cdot r'_2) \mod p$
  - Works if noise $r'_1 \cdot r'_2$ remains less than p.
- Somewhat homomorphic scheme
  - Noise grows with every homomorphic addition or multiplication.
  - This limits the degree of the polynomial that can be applied on ciphertexts.

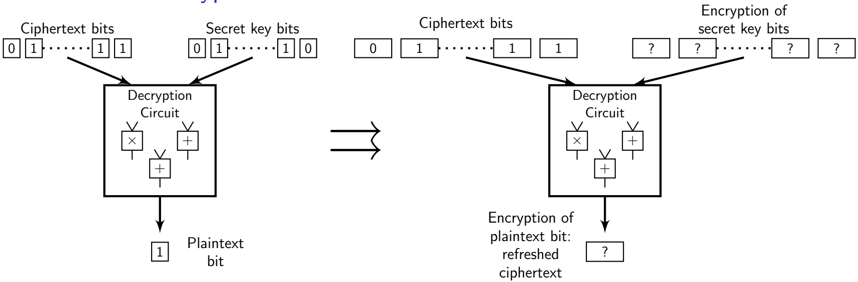
Gentry's technique to get fully homomorphic encryption

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



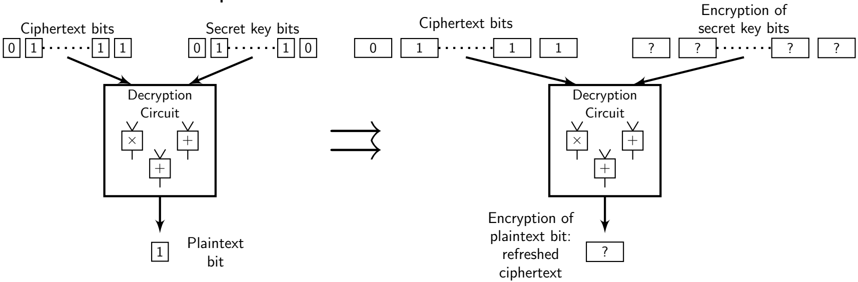
Ciphertext refresh: bootstrapping

- 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

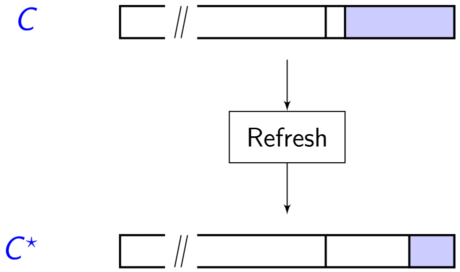
- Gentry's breakthrough idea: refresh the ciphertext using the decryption circuit homomorphically.
  - Instead of recovering the bit plaintext m, one gets an encryption of this bit plaintext, *i.e.* yet another ciphertext for the same plaintext.



- will be explained in next lecture.

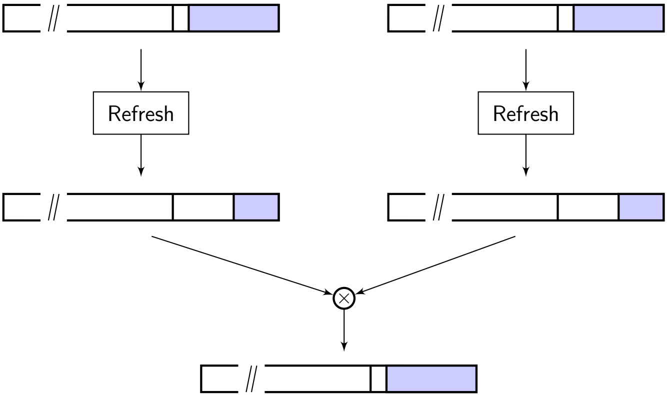
Ciphertext refresh

- 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.

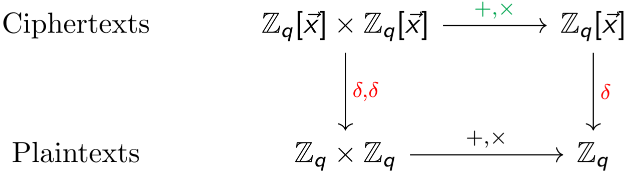


Four generations of FHE

- 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: $\delta : \mathbb{Z}_q[\vec{x}] \to \mathbb{Z}_q[x]$ given by evaluation at secret $\vec{s} = (s_1, \ldots, s_n)$



- 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

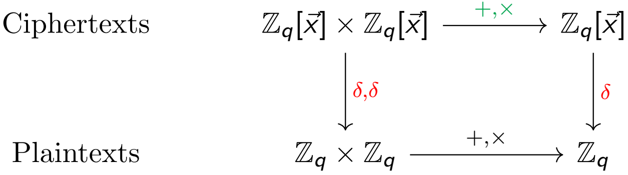
- Homomorphic encryption based on polynomial evaluation
  - Homomorphism: $\delta : \mathbb{Z}_q[\vec{x}] \to \mathbb{Z}_q[x]$ given by evaluation at secret $\vec{s} = (s_1, \ldots, s_n)$

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

- 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 $\vec{s} = (s_1, \ldots, s_n)$



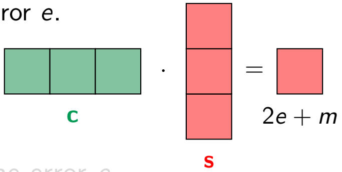
- One must add some noise, otherwise broken by linear algebra.
  - $f(\vec{s}) = 2e + m \bmod q$, for some small noise $e \in \mathbb{Z}_q$
- LWE assumption [R05]
  - Linear polynomials $f_i(\vec{x})$ with $|f_i(\vec{s}) \bmod q| \ll q$ are comp. indist. from random $f_i(\vec{x})$ modulo q.

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.



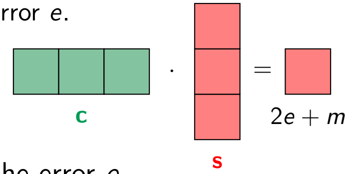
- Distribution of the error e
  - One can take the centered binomial distribution χ with parameter κ.
  - Let $e = h(u) - h(v)$ where $u, v \leftarrow \{0,1\}^\kappa$, where h is the Hamming weight function.
- Decryption
  - Compute $m = (\mathbf{c} \cdot \mathbf{s} \bmod q) \bmod 2$
  - Decryption works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

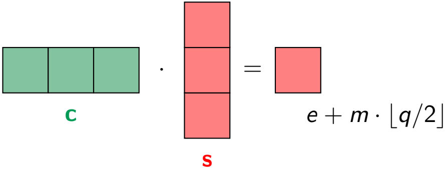


- Distribution of the error e
  - One can take the centered binomial distribution χ with parameter κ.
  - Let $e = h(u) - h(v)$ where $u, v \leftarrow \{0, 1\}^\kappa$, where h is the Hamming weight function.
- Decryption
  - Compute $m = (\mathbf{c} \cdot \mathbf{s} \bmod q) \bmod 2$
  - Decryption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

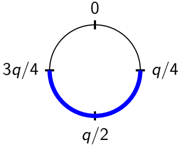
LWE-based encryption: alternative encoding

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

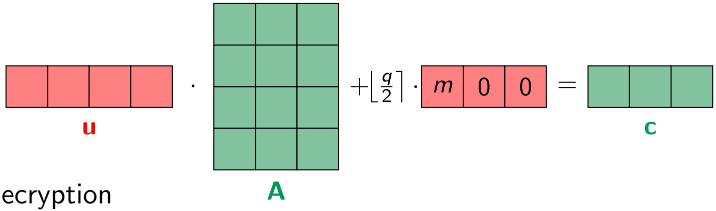


LWE-based public-key encryption

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

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

  - for a small $\mathbf{u}$



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

RLWE-based schemes

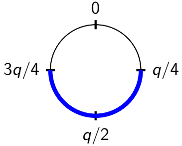
- RLWE-based scheme
  - We replace $\mathbb{Z}_q$ by the polynomial ring $R_q = \mathbb{Z}_q[x]/ < x^\ell + 1 >$, where ℓ is a power of 2.
  - Addition and multiplication of polynomials are performed modulo $x^\ell + 1$ and prime q.
  - We can take $m \in R_2 = \mathbb{Z}_2[x]/ < x^\ell + 1 >$ instead of $\{0, 1\}$: more bandwidth.
- Ring Learning with Error (RLWE) assumption
  - $t = a \cdot s + e$ for small s, $e \leftarrow R$
  - Given t, a, it is difficult to recover s.

RLWE-based public-key encryption

- Key generation
  - $t = a \cdot s + e$ for random $a \leftarrow R_q$ and small s, $e \leftarrow R$.
- Public-key encryption of $m \in R_2$
  - $c = (a \cdot r + e_1,\ t \cdot r + e_2 + \lfloor q/2 \rceil m)$, for small e_1, e_2 and r.
- Decryption of $c = (u, v)$
  - Compute $m = \text{th}(v - s \cdot u)$

$$\begin{aligned} v - s \cdot u &= t \cdot r + e_2 + \lfloor q/2 \rceil m - s \cdot (a \cdot r + e_1) \\ &= (t - a \cdot s) \cdot r + e_2 + \lfloor q/2 \rceil m - s \cdot e_1 \\ &= \lfloor q/2 \rceil m + \underbrace{e \cdot r + e_2 - s \cdot e_1}_{\text{small}} \end{aligned}$$

  - $m \in R_2 = \mathbb{Z}_2[x]/ < x^\ell + 1 >$: more bandwidth.



Homomorphic addition

- LWE ciphertexts can be added
  - with a small increase in the noise

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

Homomorphic multiplication

- Homomorphic multiplication of two ciphertexts is more complex, with 3 steps:
  - 1) Tensor product
    - We obtain a ciphertext in $\mathbb{Z}_q^{n^2}$, under a new key $\mathbf{s} \times \mathbf{s}$.
  - 2) Binary decomposition
    - We obtain a binary ciphertext in $\{0,1\}^{n^2 \cdot n_q}$, under a new key $\mathbf{s}' = \mathrm{PowerOfTwo}(\mathbf{s} \times \mathbf{s})$, with $n_q = \lceil \log_2 q \rceil$
  - 3) Key switching
    - We switch the key from $\mathbf{s}'$ back to the original key $\mathbf{s}$.

Tensor product

- LWE ciphertexts can be multiplied by tensor product.

$$2\langle \mathbf{c}_1, \mathbf{s}\rangle \cdot \langle \mathbf{c}_2, \mathbf{s}\rangle = 2\left(\sum_{i=1}^{n} c_{1,i}s_i\right)\left(\sum_{i=1}^{n} c_{2,i}s_i\right)$$
$$= 2(e_1 + (q+1)/2 \cdot m_1) \cdot (e_2 + (q+1)/2 \cdot m_2)$$

- This gives

$$\sum_{i=1}^{n}\sum_{j=1}^{n} 2c_{1,i}c_{2,j} \cdot s_i s_j = e + m_1 m_2 \cdot (q+1)/2 \pmod{q}$$

  - for a new eroor $e = 2e_1e_2 + m_1e_2 + m_2e_1$
- Therefore $\mathbf{c}' = (2c_{1,i} \cdot c_{2,j})_{i,j} \in \mathbb{Z}_q^{n^2}$ is a new LWE ciphertext
  - for the secret-key $\mathbf{s}' = (s_i \cdot s_j)_{i,j} \in \mathbb{Z}_q^{n^2}$

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

- The bitsize of the noise has roughly doubled.
  - We get a ciphertext with n^2 components instead of n.

Binary decomposition

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

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

  - $\mathrm{BitDecomp}(a) = (a_0, \ldots, a_{n_q - 1})$ and $\mathrm{PowerOf2}(b) = (b, 2^1 b, \ldots, 2^{n_q - 1} b)$.
  - We extend BitDecomp and 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 $\mathbf{c}' = \mathrm{BitDecomp}(\mathbf{c})$, and $\mathbf{s}' = \mathrm{PowerOf2}(\mathbf{s})$

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

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

Key switching

- How to switch keys ?
  - Start with a binary ciphertext $\mathbf{c} \in \{0,1\}^m$ under key $\mathbf{s} \in \mathbb{Z}_q^m$.
  - We write $u = \langle \mathbf{c}, \mathbf{s} \rangle = \sum_{i=1}^{m} c_i \cdot s_i \pmod q$
  - Let $\mathbf{s}' \in \mathbb{Z}_q^n$ be another key.
  - We consider LWE pseudo-encryptions $\mathbf{t}_i$ of each s_i under the new key $\mathbf{s}'$, with $\langle \mathbf{t}_i, \mathbf{s}' \rangle = f_i + s_i \pmod q$ for small errors f_i.
- Generating the new ciphertext under $\mathbf{s}'$
  - We can write:

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

  - We can define a new ciphertext $\mathbf{c}' = \sum_{i=1}^{m} c_i \mathbf{t}_i \pmod q$ and we get for a small error f:

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

  - $\Rightarrow$ the two ciphertexts encrypt the same message

Summary of homomorphic multiplication

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

Conclusion

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