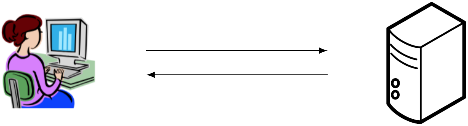
Introduction to Fully Homomorphic Encryption

Jean-Sébastien Coron

University of Luxembourg

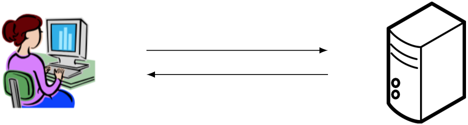
- What is Fully Homomorphic Encryption (FHE) ?
  - Basic properties
  - Cloud computing on encrypted data: the server should process the data without learning the data.



- 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

Overview

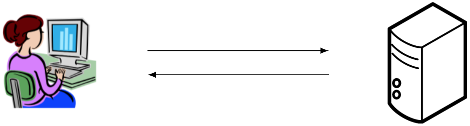
- What is Fully Homomorphic Encryption (FHE) ?
  - Basic properties
  - Cloud computing on encrypted data: the server should process the data without learning the data.



- 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.

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

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

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

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

Paillier Cryptosystem

- Additively homomorphic: Paillier cryptosystem [P99]

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

$$\begin{array}{ll} \text{Ciphertexts} & \begin{tikzcd} \mathbb{Z}/N^2\mathbb{Z} \times \mathbb{Z}/N^2\mathbb{Z} \arrow[r, "\times"] \arrow[d, "\delta{,}\delta"] & \mathbb{Z}/N^2\mathbb{Z} \arrow[d, "\delta"] \\ \mathbb{Z}/N\mathbb{Z} \times \mathbb{Z}/N\mathbb{Z} \arrow[r, "+"] & \mathbb{Z}/N\mathbb{Z} \end{tikzcd} \\ \text{Plaintexts} & \end{array}$$

Application of Paillier Cryptosystem

- Additively homomorphic: Paillier cryptosystem

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

- 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:

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

  - 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

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

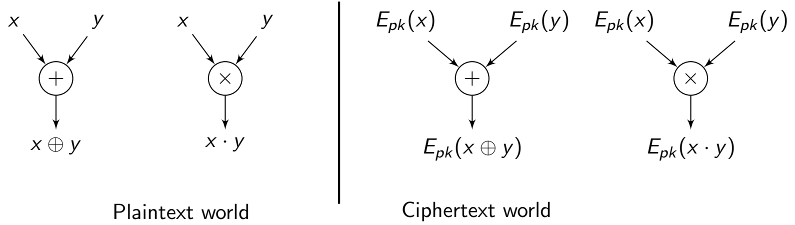
- Fully homomorphic: homomorphic for both addition and multiplication
  - Open problem until Gentry's breakthrough in 2009.

Fully homomorphic public-key encryption

- We restrict ourselves to public-key encryption of a single bit:
  - $0 \xrightarrow{E_{pk}} 203\text{ef}6124\ldots 23\text{ab}87_{16}$, $1 \xrightarrow{E_{pk}} \text{b}327653\text{c}1\ldots \text{db}3265_{16}$
  - 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.

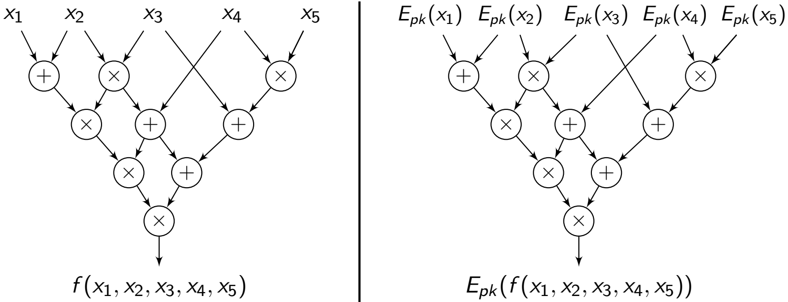
Fully homomorphic public-key encryption

- We restrict ourselves to public-key encryption of a single bit:
  - $0 \xrightarrow{E_{pk}} 203\text{ef}6124\ldots 23\text{ab}87_{16}$, $1 \xrightarrow{E_{pk}} \text{b}327653\text{c}1\ldots\text{db}3265_{16}$
  - 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.



Evaluation of any function

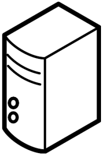
- Universality
  - We can evaluate homomorphically any boolean computable function $f : \{0,1\}^n \to \{0,1\}$



Plaintext world

Ciphertext world

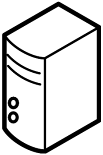
Outsourcing computation (1)



- 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)

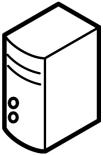
$$c_i = E_{pk}(x_i) \longrightarrow$$



- 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)

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

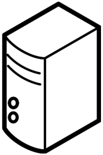


- The server homomorphically evaluates $f(x)$
  - by writing $f(x) = f(x_1, \ldots, x_n)$ as a boolean circuit.
  - Given $E_{pk}(x_i)$, the server eventually obtains $c = E_{pk}(f(x))$
- Finally Alice decrypts c into $y = f(x)$
  - The server does not learn x.
  - Only Alice can decrypt to recover $f(x)$.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

Outsourcing computation (2)

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

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

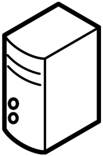


- The server homomorphically evaluates $f(x)$
  - by writing $f(x) = f(x_1, \ldots, x_n)$ as a boolean circuit.
  - Given $E_{pk}(x_i)$, the server eventually obtains $c = E_{pk}(f(x))$
- Finally Alice decrypts c into $y = f(x)$
  - The server does not learn x.
  - Only Alice can decrypt to recover $f(x)$.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

Outsourcing computation (2)

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

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



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

- 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

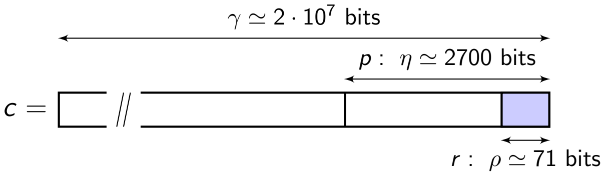
- Ciphertext for $m \in \{0, 1\}$:

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

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

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

- Parameters:



Homomorphic Properties of DGHV

- Addition:

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

  - $c_1 + c_2$ 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

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

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

Homomorphic Properties of DGHV

- Addition:

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

  - $c_1 + c_2$ 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.

Homomorphism of DGHV

- DGHV ciphertext:

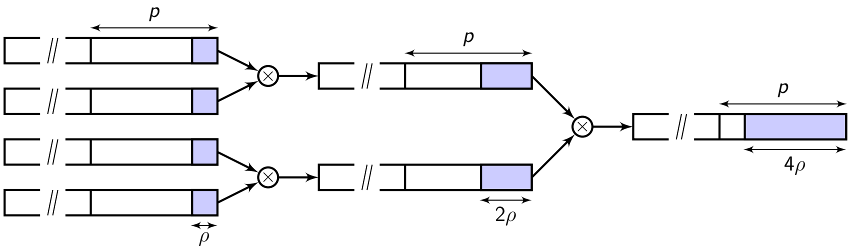
$$c = q \cdot p + 2r + m$$

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

$$\begin{tikzcd}
\text{Ciphertexts} & \mathbb{Z} \times \mathbb{Z} \arrow[r, "{+,\times}"] \arrow[d, "{\delta,\delta}"] & \mathbb{Z} \arrow[d, "\delta"] \\
\text{Plaintexts} & \mathbb{Z}_2 \times \mathbb{Z}_2 \arrow[r, "{\oplus,\times}"] & \mathbb{Z}_2
\end{tikzcd}$$

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 $\mathbb{Z}$

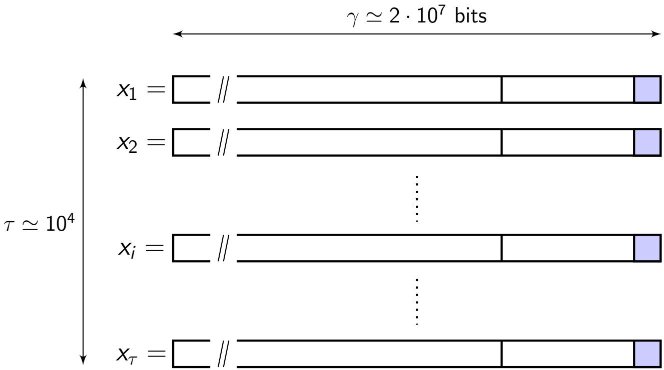
$$\begin{array}{l} c_1 = q_1 \cdot p + 2r_1 + m_1 \\ c_2 = q_2 \cdot p + 2r_2 + m_2 \end{array} \Rightarrow c_1 \cdot c_2 = q' \cdot p + 2r' + m_1 \cdot m_2$$

  - 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

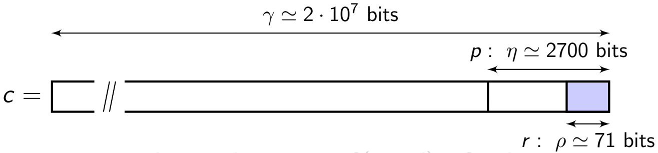
Public-key size



- 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.

$\chi =$

$\delta = \chi - 2r - m \bmod p$

$c = \chi - \delta$

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

DGHV Ciphertext Compression

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

$\gamma \simeq 2 \cdot 10^7$ bits

$p : \ \eta \simeq 2700$ bits

$c =$

$r : \ \rho \simeq 71$ bits

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

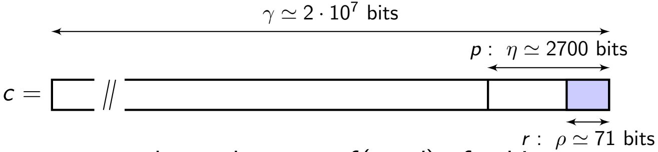
$\chi =$

$\delta = \chi - 2r - m \bmod p$

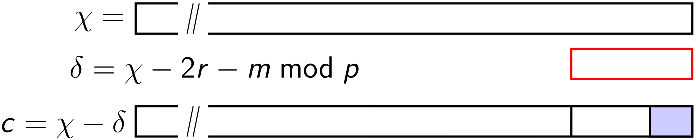
$c = \chi - \delta$

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

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



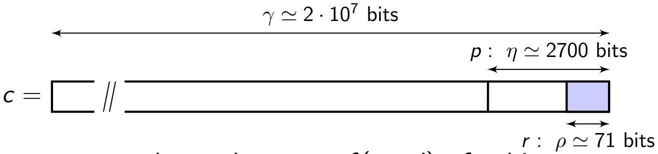
- Compute a pseudo-random $\chi = f(seed)$ of γ bits.



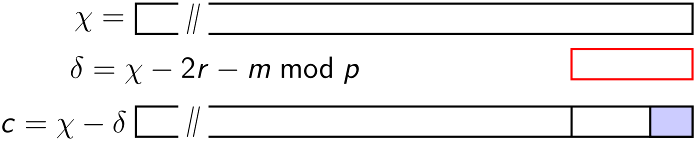
  - Only store *seed* and the small correction δ.
  - Storage: $\simeq 2\,700$ bits instead of $2 \cdot 10^7$ bits !

DGHV Ciphertext Compression

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

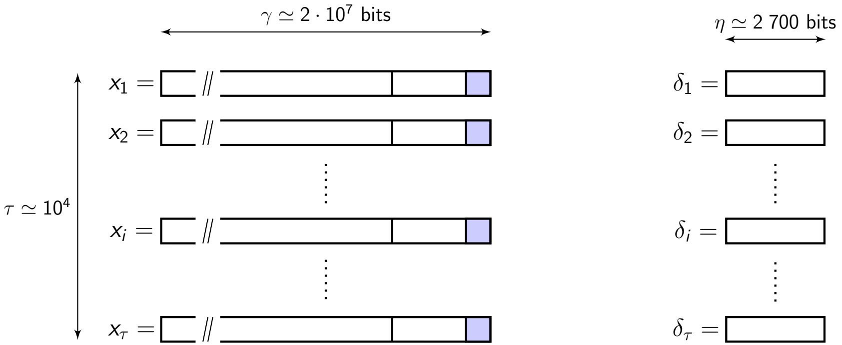


- Compute a pseudo-random $\chi = f(seed)$ of γ bits.



  - 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:

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

    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:

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

    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:

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

    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.

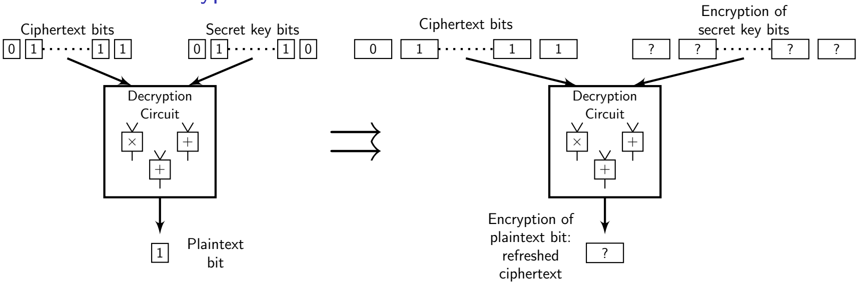
Gentry's technique to get fully homomorphic encryption

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

$$\begin{tikzcd}
\text{Ciphertexts} & \mathcal{C}^t \arrow[r, "V_{pk}(f{,}\cdots)"] \arrow[d, "D_{sk}(\cdots)"] & \mathcal{C} \arrow[d, "D_{sk}(\cdot)"] & f \in \mathcal{F} \\
\text{Plaintexts} & (\mathbb{Z}_2)^t \arrow[r, "f"] & \mathbb{Z}_2 &
\end{tikzcd}$$

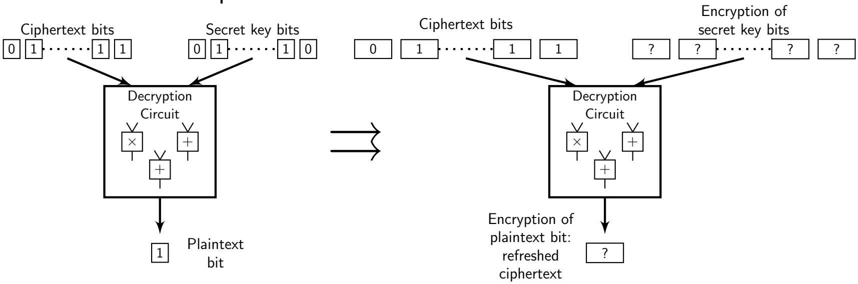
Ciphertext refresh: bootstrapping

- 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.



Bootstrapping

- Evaluating the decryption function homomorphically
  - with $f = D_{sk}(\cdot)$
  - We obtain a new ciphertext $C^\star$ with possibly less noise

$$
\begin{tikzcd}
\text{Ciphertexts} & \mathcal{C}^t \arrow[r, "V_{pk}(f,\cdots)"] \arrow[d, "D_{sk}(\cdots)"] & \mathcal{C} \arrow[d, "D_{sk}(\cdot)"] & \\
\text{Plaintexts} & (\mathbb{Z}_2)^t \arrow[r, "f"] & \mathbb{Z}_2 & f \in \mathcal{F}
\end{tikzcd}
$$

Bootstrapping

- Evaluating the decryption function homomorphically
  - with $f = D_{sk}(\cdot)$
  - We obtain a new ciphertext $C^\star$ with possibly less noise

$$(E_{pk}(c_1), \ldots, E_{pk}(c_t)) \qquad\qquad C^\star$$

$$\begin{tikzcd}
\text{Ciphertexts} & \mathcal{C}^t \arrow[r, "V_{pk}(f,\cdots)"] \arrow[d, "D_{sk}(\cdots)"] & \mathcal{C} \arrow[d, "D_{sk}(\cdot)"] & f \in \mathcal{F} \\
\text{Plaintexts} & (\mathbb{Z}_2)^t \arrow[r, "f"] & \mathbb{Z}_2 &
\end{tikzcd}$$

$$C = (c_1, \ldots, c_t) \qquad\qquad m$$

Bootstrapping (2)

- Evaluating the decryption function homomorphically
  - Actually we use $f = D(\cdot, \cdot)$
  - Using public $(E_{pk}(sk_1), \ldots, E_{pk}(sk_t))$
  - We obtain a new ciphertext $C^{\star}$ with possibly less noise

$$
\begin{tikzcd}
\text{Ciphertexts} & \mathcal{C}^{2t} \arrow[r, "V_{pk}(f{,}\cdots)"] \arrow[d, "D_{sk}(\cdots)"] & \mathcal{C} \arrow[d, "D_{sk}(\cdot)"] & f \in \mathcal{F} \\
\text{Plaintexts} & (\mathbb{Z}_2)^{2t} \arrow[r, "f"] & \mathbb{Z}_2 &
\end{tikzcd}
$$

- Evaluating the decryption function homomorphically
  - Actually we use $f = D(\cdot,\cdot)$
  - Using public $(E_{pk}(sk_1), \ldots, E_{pk}(sk_t))$
  - We obtain a new ciphertext $C^\star$ with possibly less noise

$(E_{pk}(sk_1), \ldots, E_{pk}(sk_t))$
$(E_{pk}(c_1), \ldots, E_{pk}(c_t))$ $\qquad C^\star$

$$
\begin{tikzcd}
\text{Ciphertexts} & \mathcal{C}^{2t} \arrow[r, "V_{pk}(f,\cdots)"] \arrow[d, "D_{sk}(\cdots)"] & \mathcal{C} \arrow[d, "D_{sk}(\cdot)"] & & f \in \mathcal{F} \\
\text{Plaintexts} & (\mathbb{Z}_2)^{2t} \arrow[r, "f"] & \mathbb{Z}_2
\end{tikzcd}
$$

$SK = (sk_1, \ldots, sk_t)$ $\qquad m$
$C = (c_1, \ldots, c_t)$

Squashing the decryption procedure

- Evaluating the decryption function homomorphically
  - We use $f = D(\cdot, \cdot)$.
  - We must have $f \in \mathcal{F}$: f must be a low-degree polynomial in the inputs
  - !!! This is not the case with $D(p, c) = (c \bmod p) \bmod 2$
- "Squash" the decryption procedure:
  - express the decryption function as a low degree polynomial in the bits of the ciphertext c and the secret key sk (equivalently a boolean circuit of small depth).

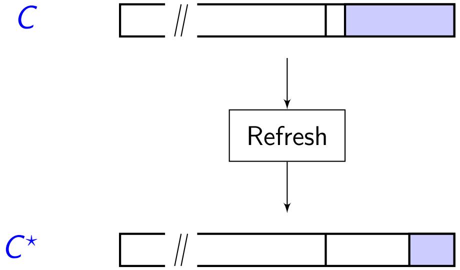
$$\begin{tikzcd} \text{Ciphertexts} & \mathcal{C}^{2t} \arrow[r, "V_{pk}(f,\cdots)"] \arrow[d, "D_{sk}(\cdots)"] & \mathcal{C} \arrow[d, "D_{sk}(\cdot)"] \\ \text{Plaintexts} & (\mathbb{Z}_2)^{2t} \arrow[r, "f"] & \mathbb{Z}_2 \end{tikzcd}$$

- Evaluating the decryption function homomorphically
  - We use $f = D(\cdot, \cdot)$.
  - We must have $f \in \mathcal{F}$: f must be a low-degree polynomial in the inputs
  - !!! This is not the case with $D(p, c) = (c \bmod p) \bmod 2$
- "Squash" the decryption procedure:
  - express the decryption function as a low degree polynomial in the bits of the ciphertext c and the secret key sk (equivalently a boolean circuit of small depth).

$$\begin{tikzcd}
\text{Ciphertexts} & \mathcal{C}^{2t} \arrow[r, "V_{pk}(f, \cdots)"] \arrow[d, "D_{sk}(\cdots)"] & \mathcal{C} \arrow[d, "D_{sk}(\cdot)"] \\
\text{Plaintexts} & (\mathbb{Z}_2)^{2t} \arrow[r, "f"] & \mathbb{Z}_2
\end{tikzcd}$$

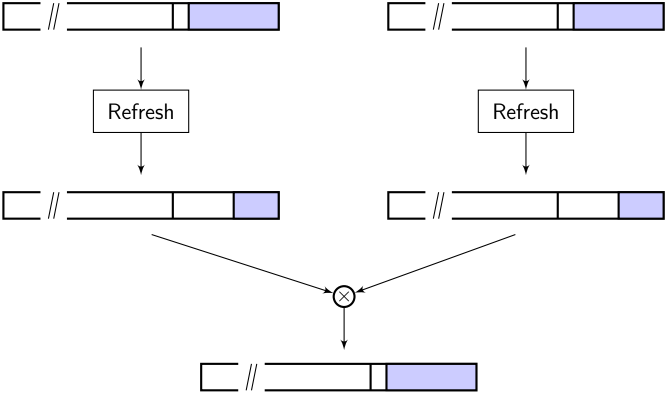
Ciphertext refresh

- 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.



The squashed scheme from DGHV

- The basic decryption $m \leftarrow (c \bmod p) \bmod 2$ cannot be directly expressed as a boolean circuit of low depth.
- Alternative decryption formula for $c = q \cdot p + 2r + m$
  - We have $q = \lfloor c/p \rceil$ and $c = q + m \pmod 2$
  - Therefore

$$m \leftarrow [c]_2 \oplus [\lfloor c \cdot (1/p) \rceil]_2$$

- Idea (Gentry, DGHV). Secret-share $1/p$ as a sparse subset sum:

$$1/p = \sum_{i=1}^{\Theta} s_i \cdot y_i + \varepsilon$$

The squashed scheme from DGHV

- The basic decryption $m \leftarrow (c \bmod p) \bmod 2$ cannot be directly expressed as a boolean circuit of low depth.
- Alternative decryption formula for $c = q \cdot p + 2r + m$
  - We have $q = \lfloor c/p \rceil$ and $c = q + m \pmod 2$
  - Therefore
$$m \leftarrow [c]_2 \oplus [\lfloor c \cdot (1/p) \rceil]_2$$
- Idea (Gentry, DGHV). Secret-share $1/p$ as a sparse subset sum:
$$1/p = \sum_{i=1}^{\Theta} s_i \cdot y_i + \varepsilon$$

The squashed scheme from DGHV

- The basic decryption $m \leftarrow (c \bmod p) \bmod 2$ cannot be directly expressed as a boolean circuit of low depth.
- Alternative decryption formula for $c = q \cdot p + 2r + m$
  - We have $q = \lfloor c/p \rceil$ and $c = q + m \pmod 2$
  - Therefore

$$m \leftarrow [c]_2 \oplus [\lfloor c \cdot (1/p) \rceil]_2$$

- Idea (Gentry, DGHV). Secret-share $1/p$ as a sparse subset sum:

$$1/p = \sum_{i=1}^{\Theta} s_i \cdot y_i + \varepsilon$$

Squashed decryption

- Alternative equation

$$m \leftarrow [c]_2 \oplus [\lfloor c \cdot (1/p) \rceil]_2$$

- Secret-share $1/p$ as a sparse subset sum:

$$1/p = \sum_{i=1}^{\Theta} s_i \cdot y_i + \varepsilon$$

with random public y_i with precision $2^{-\kappa}$, and sparse secret $s_i \in \{0,1\}$.

- Decryption becomes:

$$m \leftarrow [c]_2 \oplus \left[\left\lfloor \sum_{i=1}^{\Theta} s_i \cdot (y_i \cdot c) \right\rceil \right]_2$$

Squashed decryption

- Alternative decryption equation:

$$m \leftarrow [c]_2 \oplus \left[\left\lfloor \sum_{i=1}^{\Theta} s_i \cdot z_i \right\rceil \right]_2$$

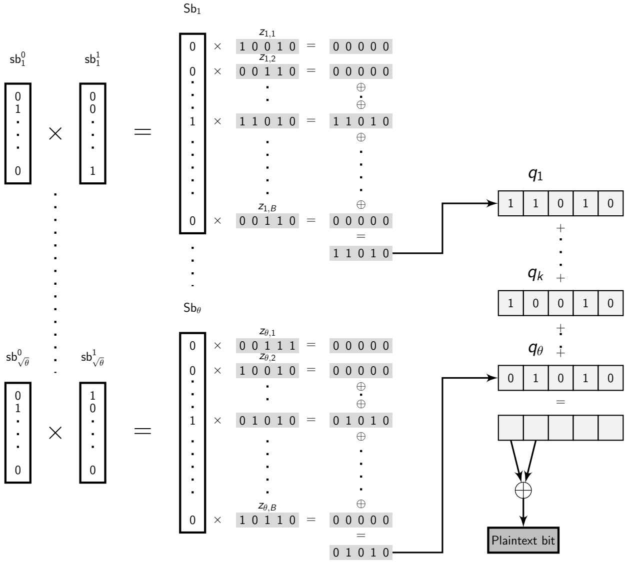
where $z_i = y_i \cdot c$ for public y_i's

- Since s_i is sparse with $H(s_i) = \theta$, only $n = \lceil \log_2(\theta + 1) \rceil$ bits of precision for $z_i = y_i \cdot c$ is required
  - With $\theta = 15$, only $n = 4$ bits of precision for $z_i = y_i \cdot c$
- The decryption function can then be expressed as a polynomial of low degree (30) in the s_i's.

The decryption circuit

- We must compute: $m \leftarrow [c]_2 \oplus \left[\left\lfloor \sum_{i=i}^{\Theta} s_i \cdot z_i \right\rceil\right]_2$
- Trick from Gentry-Halevi:
  - Split the Θ secret key bits into θ boxes of size $B = \Theta/\theta$ each.
  - Then only one secret key bit inside every box is equal to one
- New decryption formula: $m \leftarrow [c]_2 \oplus \left[\left\lfloor \sum_{k=1}^{\theta} \left(\sum_{i=1}^{B} s_{k,i} z_{k,i} \right) \right\rceil\right]_2$
  - The sum $q_k \stackrel{\text{def}}{=} \sum_{i=1}^{B} s_{k,i} z_{k,i}$ is obtained by adding B numbers, only one being non-zero.
  - To compute the j-th bit of q_k it suffices to xor all the j-th bits of the numbers $s_{k,i} \cdot z_{k,i}$.

The decryption circuit

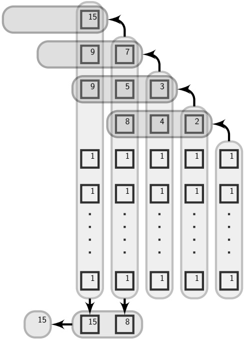


Grade School addition

- The decryption equation is now:

$$m \leftarrow [c]_2 \oplus \left[\left\lfloor \sum_{k=1}^{\theta} q_k \right\rceil \right]_2$$

  - where the q_k's are rational in $[0, 2)$ with n bits of precision after the binary point.



Gentry's Bootstrapping

- The decryption circuit
  - Can now be expressed as a polynomial of small degree d in the secret-key bits s_i, given the $z_i = c \cdot y_i$.

$$m = C_{z_i}(s_1, \ldots, s_\Theta)$$

- To refresh a ciphertext:
  - Publish an encryption of the secret-key bits $\sigma_i = E_{pk}(s_i)$
  - Homomorphically evaluate $m = C_{z_i}(s_1, \ldots, s_\Theta)$, using the encryptions $\sigma_i = E_{pk}(s_i)$
  - We get $E_{pk}(m)$, that is a new ciphertext but possibly with less noise (a "recryption").
  - The new noise has size $\simeq d \cdot \rho$ and is independent of the initial noise.

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

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

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

$$\begin{tikzcd}
\text{Ciphertexts} & \mathbb{Z}_q[\vec{x}] \times \mathbb{Z}_q[\vec{x}] \arrow[r, "{+,\times}"] \arrow[d, "{\delta,\delta}"] & \mathbb{Z}_q[\vec{x}] \arrow[d, "\delta"] \\
\text{Plaintexts} & \mathbb{Z}_q \times \mathbb{Z}_q \arrow[r, "{+,\times}"] & \mathbb{Z}_q
\end{tikzcd}$$

- 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.

Regev's scheme based on LWE [R05]

- Key generation
  - Secret-key: $\vec{s} \in (\mathbb{Z}_q)^n$
  - Public-key: $f_i(\vec{x})$ such that $f_i(\vec{s}) = 2e_i$ with $e_i \ll q$
- Encryption of $m \in \{0, 1\}$
  - $c(\vec{x}) = m + \sum_{i=1}^{\tau} b_i \cdot f_i(\vec{x})$ for random $b_i \leftarrow \{0, 1\}$
- Decryption
  - Compute $v = c(\vec{s}) = m + 2 \cdot \sum_{i=1}^{\tau} b_i \cdot e_i \pmod q$
  - Recover $m = v \bmod 2$

The BV scheme: relinearization [BV11]

- Regev's ciphertext:
  - $c(\vec{x})$ such that $c(\vec{s}) = m + 2e \bmod q$, with $\vec{s} \in (\mathbb{Z}_q)^n$.
- Multiplication of Regev's ciphertext
  - $c(\vec{x}) = c_1(\vec{x}) \cdot c_2(\vec{x})$
  - $c(\vec{s}) = (m_1 + 2e_1) \cdot (m_2 + 2e_2) = m_1 m_2 + 2e \pmod q$
- Problem: $c(\vec{x})$ is a quadratic polynomial with $(n+1)^2$ coefficients !
  - instead of $n+1$ for the original ciphertexts $c_1(\vec{x})$ and $c_2(\vec{x})$
- Relinearization [BV11]:
  - Publish polynomials $p_{j,k,t}(\vec{x}) = 2^t x_j x_k + L_{j,k,t}(\vec{x})$
  - with $p_{j,k,t}(\vec{s}) = 2e_{j,k,t} \bmod q$
  - remove the quadratic terms $a_{jk} x_j x_k$ by subtraction, using a binary decomposition of a_{jk}.
  - Only linear terms remain, so ciphertext size remains constant

The BV scheme: relinearization [BV11]

- Regev's ciphertext:
  - $c(\vec{x})$ such that $c(\vec{s}) = m + 2e \bmod q$, with $\vec{s} \in (\mathbb{Z}_q)^n$.
- Multiplication of Regev's ciphertext
  - $c(\vec{x}) = c_1(\vec{x}) \cdot c_2(\vec{x})$
  - $c(\vec{s}) = (m_1 + 2e_1) \cdot (m_2 + 2e_2) = m_1 m_2 + 2e \pmod q$
- Problem: $c(\vec{x})$ is a quadratic polynomial with $(n+1)^2$ coefficients !
  - instead of $n+1$ for the original ciphertexts $c_1(\vec{x})$ and $c_2(\vec{x})$
- Relinearization [BV11]:
  - Publish polynomials $p_{j,k,t}(\vec{x}) = 2^t x_j x_k + L_{j,k,t}(\vec{x})$
  - with $p_{j,k,t}(\vec{s}) = 2e_{j,k,t} \bmod q$
  - remove the quadratic terms $a_{jk} x_j x_k$ by subtraction, using a binary decomposition of a_{jk}.
  - Only linear terms remain, so ciphertext size remains constant

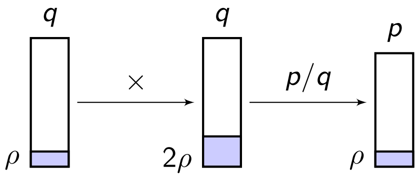
The BGV scheme: modulus switching [BGV11]

- Modulus switching of $c(\vec{x}) = \langle \vec{c}, (1, \vec{x}) \rangle \bmod q$ to modulo p
  - Let $\vec{c'}$ be the integer vector closest to $p/q \cdot \vec{c}$ such that $\vec{c'} = \vec{c} \bmod 2$
  - Then $[\vec{c'}, \vec{s}]_p = [\vec{c}, \vec{s}]_q \bmod 2$: decryption remains the same
  - and $\langle \vec{c'}, \vec{s} \rangle \simeq (p/q) \cdot \langle \vec{c}, \vec{s} \rangle$: noise is reduced by a factor q/p.
- Application: reducing noise growth. Assume $p/q = 2^{-\rho}$.

  - Noise reduction without bootstrapping !

The BGV scheme: modulus switching [BGV11]

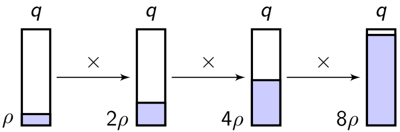
- Modulus switching of $c(\vec{x}) = \langle \vec{c}, (1, \vec{x}) \rangle \bmod q$ to modulo p
  - Let $\vec{c'}$ be the integer vector closest to $p/q \cdot \vec{c}$ such that $\vec{c'} = \vec{c} \bmod 2$
  - Then $[\vec{c'}, \vec{s}]_p = [\vec{c}, \vec{s}]_q \bmod 2$: decryption remains the same
  - and $\langle \vec{c'}, \vec{s} \rangle \simeq (p/q) \cdot \langle \vec{c}, \vec{s} \rangle$: noise is reduced by a factor q/p.
- Application: reducing noise growth. Assume $p/q = 2^{-\rho}$.



  - Noise reduction without bootstrapping !

Leveled fully homomorphic encryption

- Previous model: exponential growth of noise

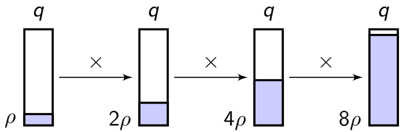


  - Only bootstrapping can give FHE
- New model: modulus switching after each multiplication layer
  - with a ladder of moduli p_i such that $p_{i+1}/p_i = 2^{-\rho}$

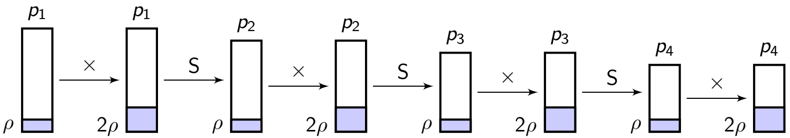
- Leveled FHE
  - Size of p_1 linear in the circuit depth
  - Parameters depend on the depth
  - Can accommodate polynomial depth

Leveled fully homomorphic encryption

- Previous model: exponential growth of noise

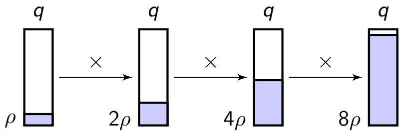


  - Only bootstrapping can give FHE
- New model: modulus switching after each multiplication layer
  - with a ladder of moduli p_i such that $p_{i+1}/p_i = 2^{-\rho}$

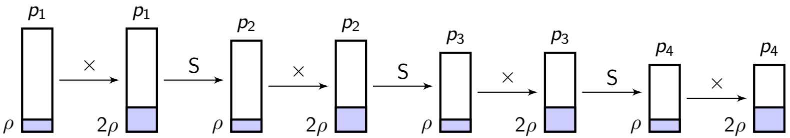


- Leveled FHE
  - Size of p_1 linear in the circuit depth
  - Parameters depend on the depth
  - Can accommodate polynomial depth

- 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

RLWE-based schemes

- Regev's scheme based on LWE
  - Secret-key: $\vec{s} \in (\mathbb{Z}_q)^n$
  - Public-key: $f_i(\vec{x})$ such that $f_i(\vec{s}) = 2e_i$ with $e_i \ll q$
  - $c(\vec{x}) = m + \sum_{i=1}^{\tau} b_i \cdot f_i(\vec{x})$ for random $b_i \leftarrow \{0,1\}$
  - $m = (c(\vec{s}) \bmod q) \bmod 2$
- RLWE-based scheme
  - We can replace $\mathbb{Z}_q$ by the polynomial ring $R_q = \mathbb{Z}_q[x]/ < x^k + 1 >$, where k is a power of 2.
  - Addition and multiplication of polynomials are performed modulo $x^k + 1$ and prime q.
  - We can take $n = 1$.
  - We can take $m \in R_2 = \mathbb{Z}_2[x]/< x^k + 1 >$ instead of $\{0,1\}$: more bandwidth

RLWE-based schemes

- Regev's scheme based on LWE
  - Secret-key: $\vec{s} \in (\mathbb{Z}_q)^n$
  - Public-key: $f_i(\vec{x})$ such that $f_i(\vec{s}) = 2e_i$ with $e_i \ll q$
  - $c(\vec{x}) = m + \sum_{i=1}^{\tau} b_i \cdot f_i(\vec{x})$ for random $b_i \leftarrow \{0,1\}$
  - $m = (c(\vec{s}) \bmod q) \bmod 2$
- RLWE-based scheme
  - We can replace $\mathbb{Z}_q$ by the polynomial ring $R_q = \mathbb{Z}_q[x] / < x^k + 1 >$, where k is a power of 2.
  - Addition and multiplication of polynomials are performed modulo $x^k + 1$ and prime q.
  - We can take $n = 1$.
  - We can take $m \in R_2 = \mathbb{Z}_2[x]/<x^k + 1>$ instead of $\{0,1\}$: more bandwidth

Third generation of FHE: ciphertext matrices

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

$$\begin{tikzcd}
\text{Ciphertexts} & \mathbb{Z}^{N\times N} \times \mathbb{Z}^{N\times N} \arrow[r, "{+,\times}"] \arrow[d, "{\delta,\delta}"] & \mathbb{Z}^{N\times N} \arrow[d, "\delta"] \\
\text{Plaintexts} & \mathbb{Z} \times \mathbb{Z} \arrow[r, "{+,\times}"] & \mathbb{Z}
\end{tikzcd}$$

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

Third generation of FHE: ciphertext matrices

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

$$
\begin{tikzcd}
\text{Ciphertexts} & \mathbb{Z}^{N\times N} \times \mathbb{Z}^{N\times N} \arrow[r, "{+,\times}"] \arrow[d, "{\delta,\delta}"] & \mathbb{Z}^{N\times N} \arrow[d, "\delta"] \\
\text{Plaintexts} & \mathbb{Z} \times \mathbb{Z} \arrow[r, "{+,\times}"] & \mathbb{Z}
\end{tikzcd}
$$

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

Ciphertext matrices: slow noise growth

- Noise grow of ciphertext multiplication [GSW13]:
  - $C_1 \cdot \vec{v} = \mu_1 \cdot \vec{v} + \vec{e}_1 \pmod{q}$, $C_2 \cdot \vec{v} = \mu_2 \cdot \vec{v} + \vec{e}_2 \pmod{q}$
  - $(C_1 \cdot C_2) \cdot \vec{v} = C_1 \cdot (\mu_2 \cdot \vec{v} + \vec{e}_2) = (\mu_2 \cdot \mu_1) \cdot \vec{v} + \vec{e}_3$
  - with $\vec{e}_3 = \mu_2 \cdot \vec{e}_1 + C_1 \cdot \vec{e}_2$
- Slow noise growth:
  - Ensure $\mu_i \in \{0, 1\}$, using only NAND gates $\mu_3 = 1 - \mu_1 \cdot \mu_2$
  - Ciphertext flattening: ensure $C_i \in \{0, 1\}^{N \times N}$, using binary decomposition and $\vec{v} = (s_1, \ldots, 2^\ell s_1, \ldots, s_n, \ldots, 2^\ell s_n)$.
  - If $\|\vec{e}_1\|_\infty \leq B$ and $\|\vec{e}_2\|_\infty \leq B$, $\|\vec{e}_3\|_\infty \leq (N+1) \cdot B$
- Leveled FHE
  - At depth L, $\|\vec{e}\|_\infty \leq (N+1)^L \cdot B$
  - One can take $q > 8 \cdot B \cdot (N+1)^L$ and accommodate polynomial depth L.

Ciphertext matrices: slow noise growth

- Noise grow of ciphertext multiplication [GSW13]:
  - $C_1 \cdot \vec{v} = \mu_1 \cdot \vec{v} + \vec{e}_1 \pmod q$, $C_2 \cdot \vec{v} = \mu_2 \cdot \vec{v} + \vec{e}_2 \pmod q$
  - $(C_1 \cdot C_2) \cdot \vec{v} = C_1 \cdot (\mu_2 \cdot \vec{v} + \vec{e}_2) = (\mu_2 \cdot \mu_1) \cdot \vec{v} + \vec{e}_3$
  - with $\vec{e}_3 = \mu_2 \cdot \vec{e}_1 + C_1 \cdot \vec{e}_2$
- Slow noise growth:
  - Ensure $\mu_i \in \{0, 1\}$, using only NAND gates $\mu_3 = 1 - \mu_1 \cdot \mu_2$
  - Ciphertext flattening: ensure $C_i \in \{0, 1\}^{N \times N}$, using binary decomposition and $\vec{v} = (s_1, \ldots, 2^\ell s_1, \ldots, s_n, \ldots, 2^\ell s_n)$.
  - If $\|\vec{e}_1\|_\infty \leq B$ and $\|\vec{e}_2\|_\infty \leq B$, $\|\vec{e}_3\|_\infty \leq (N+1) \cdot B$
- Leveled FHE
  - At depth L, $\|\vec{e}\|_\infty \leq (N+1)^L \cdot B$
  - One can take $q > 8 \cdot B \cdot (N+1)^L$ and accommodate polynomial depth L.

Ciphertext matrices: slow noise growth

- Noise grow of ciphertext multiplication [GSW13]:
  - $C_1 \cdot \vec{v} = \mu_1 \cdot \vec{v} + \vec{e}_1 \pmod{q}$, $C_2 \cdot \vec{v} = \mu_2 \cdot \vec{v} + \vec{e}_2 \pmod{q}$
  - $(C_1 \cdot C_2) \cdot \vec{v} = C_1 \cdot (\mu_2 \cdot \vec{v} + \vec{e}_2) = (\mu_2 \cdot \mu_1) \cdot \vec{v} + \vec{e}_3$
  - with $\vec{e}_3 = \mu_2 \cdot \vec{e}_1 + C_1 \cdot \vec{e}_2$
- Slow noise growth:
  - Ensure $\mu_i \in \{0,1\}$, using only NAND gates $\mu_3 = 1 - \mu_1 \cdot \mu_2$
  - Ciphertext flattening: ensure $C_i \in \{0,1\}^{N \times N}$, using binary decomposition and $\vec{v} = (s_1, \ldots, 2^\ell s_1, \ldots, s_n, \ldots, 2^\ell s_n)$.
  - If $\|\vec{e}_1\|_\infty \le B$ and $\|\vec{e}_2\|_\infty \le B$, $\|\vec{e}_3\|_\infty \le (N+1) \cdot B$
- Leveled FHE
  - At depth L, $\|\vec{e}\|_\infty \le (N+1)^L \cdot B$
  - One can take $q > 8 \cdot B \cdot (N+1)^L$ and accommodate polynomial depth L.

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: $\delta : \mathbb{Z}_q[\vec{x}] \to \mathbb{Z}_q$ given by evaluation at $\vec{s}$

$$\begin{tikzcd}
\text{Ciphertexts} & \mathbb{Z}_q[\vec{x}] \times \mathbb{Z}_q[\vec{x}] \arrow[r, "{+,\times}"] \arrow[d, "{\delta,\delta}"] & \mathbb{Z}_q[\vec{x}] \arrow[d, "\delta"] \\
\text{Plaintexts} & \mathbb{Z}_q \times \mathbb{Z}_q \arrow[r, "{+,\times}"] & \mathbb{Z}_q
\end{tikzcd}$$

- One must add some noise, otherwise broken by linear algebra.
  - $f(\vec{s}) = m + e \bmod q$, 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

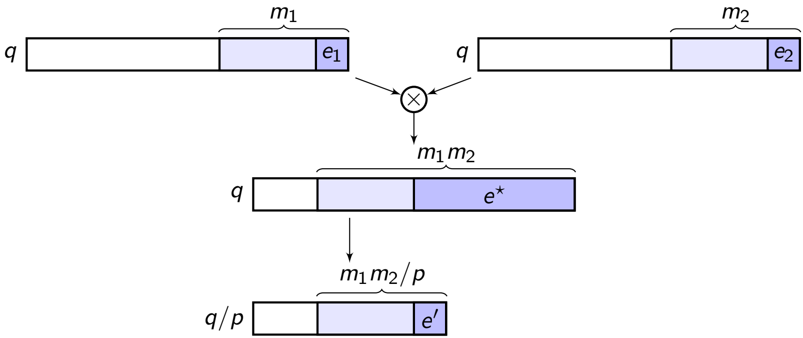
- Homomorphic encryption for real numbers [CKKS17]
  - Floating point arithmetic, instead of exact arithmetic.
  - Starting point: Regev's scheme.
  - Homomorphism: $\delta : \mathbb{Z}_q[\vec{x}] \to \mathbb{Z}_q$ given by evaluation at $\vec{s}$

$$\begin{tikzcd}
\text{Ciphertexts} & \mathbb{Z}_q[\vec{x}] \times \mathbb{Z}_q[\vec{x}] \arrow[r, "{+,\times}"] \arrow[d, "{\delta,\delta}"] & \mathbb{Z}_q[\vec{x}] \arrow[d, "\delta"] \\
\text{Plaintexts} & \mathbb{Z}_q \times \mathbb{Z}_q \arrow[r, "{+,\times}"] & \mathbb{Z}_q
\end{tikzcd}$$

- One must add some noise, otherwise broken by linear algebra.
  - $f(\vec{s}) = m + e \bmod q$, 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.

[CKKS17]: ciphertext multiplication and rescaling

- Ciphertext multiplication $c(\vec{x}) = c_1(\vec{x}) \cdot c_2(\vec{x})$
  - $c(\vec{s}) = (m_1 + e_1) \cdot (m_2 + e_2) = m_1 m_2 + e^\star \pmod q$
  - with $e^\star = m_1 e_2 + e_1 m_2 + e_1 e_2$.
- Rescaling of ciphertext:
  - $c'(\vec{x}) = \lfloor \vec{c}(x)/p \rceil \pmod{q/p}$
  - Valid encryption of $\lfloor m/p \rceil$ with noise $\simeq e/p$
  - Similar to modulus switching



Conclusion

- Main challenge: make FHE pratical !
  - New primitives
  - Libraries (HElib)
  - Compiler to homomorphic evaluation
- Applications
  - Homomorphic machine learning: evaluate a neural network without revealing the weights.
  - Genome-wide association studies: linear regression, logistic regression.

References

AP14 Jacob Alperin-Sheriff, Chris Peikert. Faster Bootstrapping with Polynomial Error. IACR Cryptol. ePrint Arch. 2014: 94 (2014)

BGV11 Zvika Brakerski, Craig Gentry, Vinod Vaikuntanathan. Fully Homomorphic Encryption without Bootstrapping. Electron. Colloquium Comput. Complex. 18: 111 (2011)

BV14 Zvika Brakerski, Vinod Vaikuntanathan. Lattice-based FHE as secure as PKE. ITCS 2014: 1-12

CCK+13 Jung Hee Cheon, Jean-Sébastien Coron, Jinsu Kim, Moon Sung Lee, Tancrède Lepoint, Mehdi Tibouchi, Aaram Yun: Batch Fully Homomorphic Encryption over the Integers. EUROCRYPT 2013: 315-335

CKKS17 Jung Hee Cheon, Andrey Kim, Miran Kim, Yong Soo Song. Homomorphic Encryption for Arithmetic of Approximate Numbers. ASIACRYPT (1) 2017: 409-437

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

References

GH11 Craig Gentry, Shai Halevi. Implementing Gentry's Fully-Homomorphic Encryption Scheme. EUROCRYPT 2011: 129-148

GSW13 Craig Gentry, Amit Sahai, Brent Waters. Homomorphic Encryption from Learning with Errors: Conceptually-Simpler, Asymptotically-Faster, Attribute-Based. CRYPTO (1) 2013: 75-92

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

SV10 Nigel P. Smart, Frederik Vercauteren. Fully Homomorphic Encryption with Relatively Small Key and Ciphertext Sizes. Public Key Cryptography 2010: 420-443