Public-key cryptography

Part 3: secure cloud computing

Jean-Sébastien Coron

University of Luxembourg

Outline

- Lecture 1: introduction to public-key cryptography
  - RSA encryption, signatures and DH key exchange
- Lecture 2: applications of public-key cryptography
  - Security models.
  - How to encrypt and sign securely with RSA. OAEP and PSS.
  - Public-key infrastructure. Certificates, SSL protocol.
- Lecture 3: cloud computing (this lecture)
  - How to delegate computation thanks to fully homomorphic encryption
  - A fully homomorphic encryption scheme

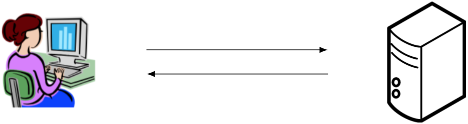
Introduction to Fully Homomorphic Encryption

Jean-Sébastien Coron

University of Luxembourg

Overview

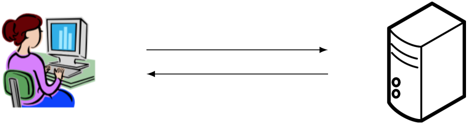
- What is Fully Homomorphic Encryption (FHE) ?
  - Basic properties
  - Cloud computing on encrypted data: the server should process the data without learning the data.



- 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

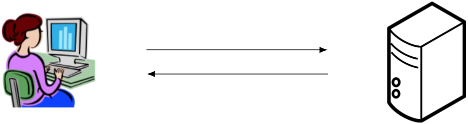
- What is Fully Homomorphic Encryption (FHE) ?
  - Basic properties
  - Cloud computing on encrypted data: the server should process the data without learning the data.



- 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

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

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

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

- Additively homomorphic: Paillier cryptosystem [P99]

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

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

Application of Paillier Cryptosystem

- Additively homomorphic: Paillier cryptosystem

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

- 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.

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

- Additively homomorphic: Paillier

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

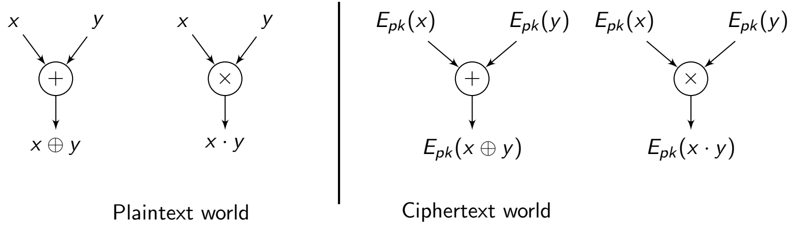
- Fully homomorphic: homomorphic for both addition and multiplication
  - Open problem until Gentry's breakthrough in 2009.

Fully homomorphic public-key encryption

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

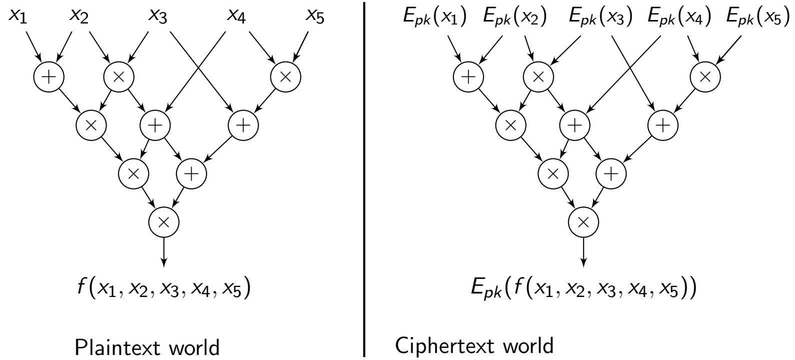
Fully homomorphic public-key encryption

- 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.

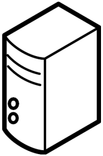


Evaluation of any function

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



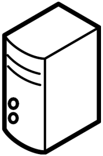
Outsourcing computation (1)



- 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)

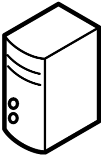
$$\xrightarrow{\quad c_i \;=\; E_{pk}(x_i) \quad}$$



- 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)

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

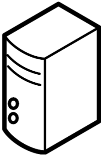


- The server homomorphically evaluates $f(x)$
  - by writing $f(x) = f(x_1, \ldots, x_n)$ as a boolean circuit.
  - Given $E_{pk}(x_i)$, the server eventually obtains $c = E_{pk}(f(x))$
- Finally Alice decrypts c into $y = f(x)$
  - The server does not learn x.
  - Only Alice can decrypt to recover $f(x)$.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

Outsourcing computation (2)

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

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

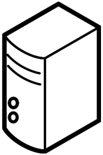


- The server homomorphically evaluates $f(x)$
  - by writing $f(x) = f(x_1, \ldots, x_n)$ as a boolean circuit.
  - Given $E_{pk}(x_i)$, the server eventually obtains $c = E_{pk}(f(x))$
- Finally Alice decrypts c into $y = f(x)$
  - The server does not learn x.
  - Only Alice can decrypt to recover $f(x)$.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

Outsourcing computation (2)

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

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



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

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

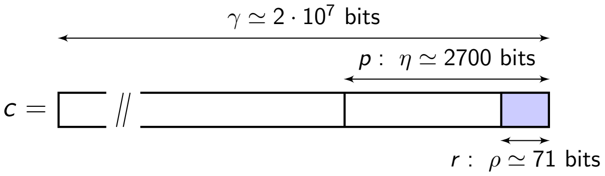
$$c = q \cdot p + 2r + m$$

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:

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

  - $c_1 + c_2$ 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

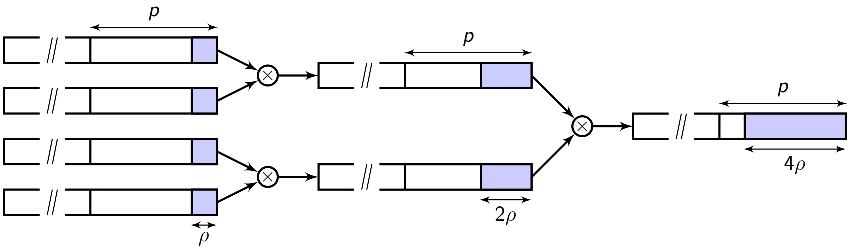
- DGHV ciphertext:

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

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

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

- 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 $\mathbb{Z}$

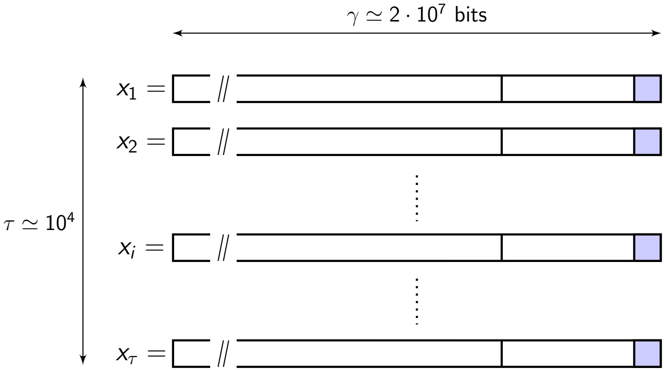
$$\begin{aligned} c_1 &= q_1 \cdot p + 2r_1 + m_1 \\ c_2 &= q_2 \cdot p + 2r_2 + m_2 \end{aligned} \Rightarrow c_1 \cdot c_2 = q' \cdot p + 2r' + m_1 \cdot m_2$$

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

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

  - Ciphertext size remains constant

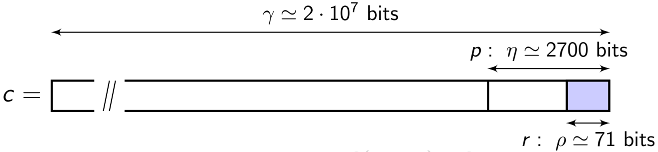
Public-key size



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

DGHV Ciphertext Compression

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



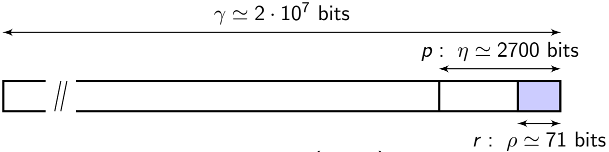
- Compute a pseudo-random $\chi = f(seed)$ of γ bits.

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

DGHV Ciphertext Compression

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

$c =$

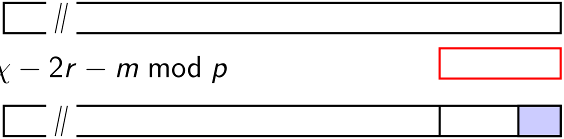


- Compute a pseudo-random $\chi = f(seed)$ of γ bits.

$\chi =$

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

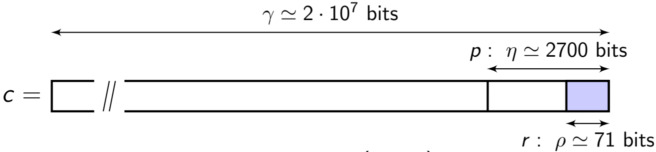
$c = \chi - \delta$



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

DGHV Ciphertext Compression

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

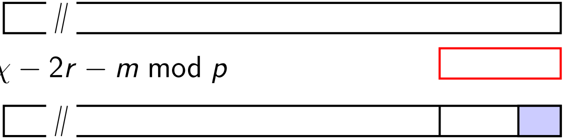


- Compute a pseudo-random $\chi = f(\mathit{seed})$ of γ bits.

$\chi =$

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

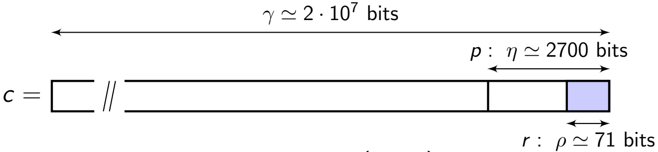
$c = \chi - \delta$



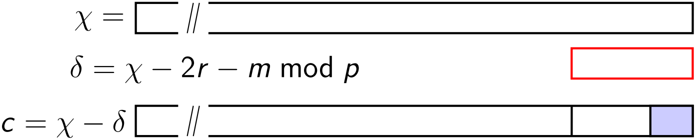
  - Only store *seed* and the small correction δ.
  - Storage: $\simeq 2\,700$ bits instead of $2 \cdot 10^7$ bits !

DGHV Ciphertext Compression

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

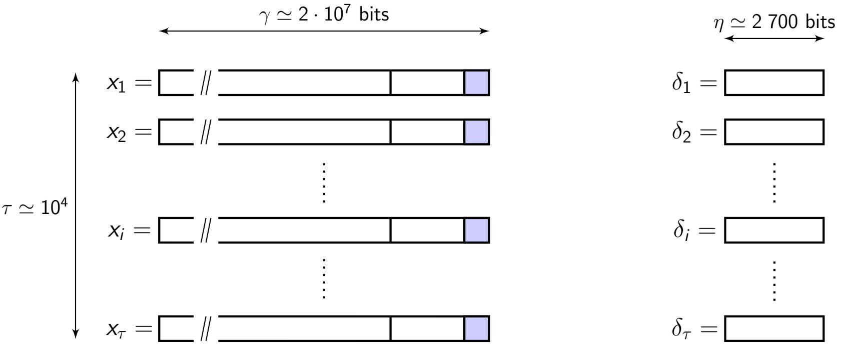


- Compute a pseudo-random $\chi = f(seed)$ of γ bits.



  - 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.

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