Public Key Compression and Modulus Switching for Fully Homomorphic Encryption over the Integers

Jean-Sébastien Coron, David Naccache and Mehdi Tibouchi

University of Luxembourg & ENS & NTT

EUROCRYPT, 2012-04-18

Fully homomorphic encryption

- Multiplicatively homomorphic: RSA.

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

- Additively homomorphic: Paillier

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

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

Fully homomorphic encryption

- Multiplicatively homomorphic: RSA.

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

- Additively homomorphic: Paillier

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

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

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 Encryption Schemes

- 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.
- 3. RLWE schemes [BV11a,BV11b].
  - FHE without bootstrapping [BGV11]
  - Batch FHE (next talk !)
  - Implementation with homomorphic evaluation of AES [GHS12]
- This talk: smaller PK for DGHV (10 MB) and improved attack against DGHV.

Fully Homomorphic Encryption Schemes

- 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.
- 3. RLWE schemes [BV11a,BV11b].
  - FHE without bootstrapping [BGV11]
  - Batch FHE (next talk !)
  - Implementation with homomorphic evaluation of AES [GHS12]
- This talk: smaller PK for DGHV (10 MB) and improved attack against DGHV.

Fully Homomorphic Encryption Schemes

- 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.
- 3. RLWE schemes [BV11a,BV11b].
    - FHE without bootstrapping [BGV11]
    - Batch FHE (next talk !)
    - Implementation with homomorphic evaluation of AES [GHS12]
- This talk: smaller PK for DGHV (10 MB) and improved attack against DGHV.

Fully Homomorphic Encryption Schemes

- 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.
- 3. RLWE schemes [BV11a,BV11b].
    - FHE without bootstrapping [BGV11]
    - Batch FHE (next talk !)
    - Implementation with homomorphic evaluation of AES [GHS12]
- This talk: smaller PK for DGHV (10 MB) and improved attack against DGHV.

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:

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:

The DGHV Scheme

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

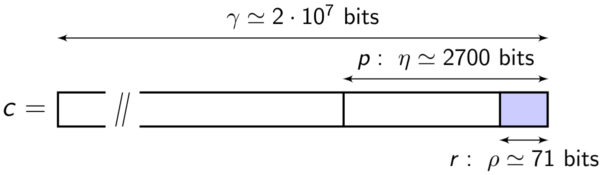
$$c = q \cdot p + 2r + m$$

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

- Decryption:

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

- Parameters:



Homomorphic Properties of DGHV

- Addition:

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

- Multiplication:

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

with

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

  - Noise becomes twice larger.

Homomorphic Properties of DGHV

- Addition:

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

- Multiplication:

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

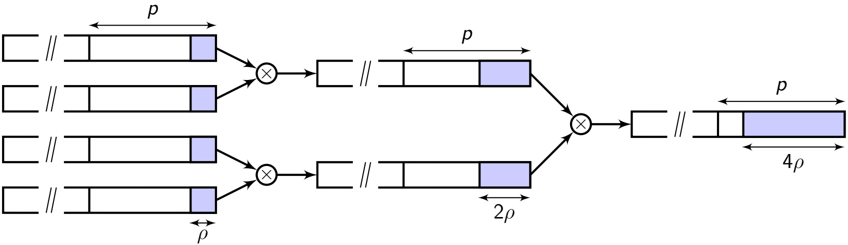
with

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

  - Noise becomes twice larger.

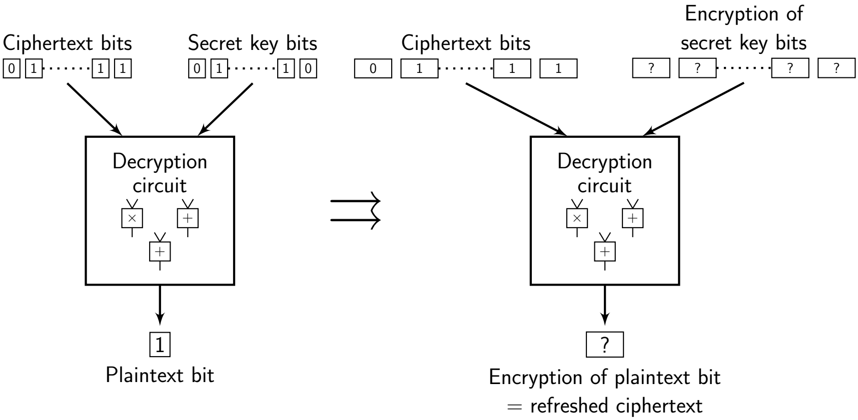
Somewhat homomorphic scheme

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



Fully Homomorphic Encryption

- Gentry's breakthrough idea: refresh the ciphertext by evaluating the decryption circuit homomorphically: bootstrapping.



Public-key Encryption with DGHV

- Ciphertext

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

- 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

- Ciphertext

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

- 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

- Ciphertext

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

- Public-key: a set of τ encryptions of 0's.

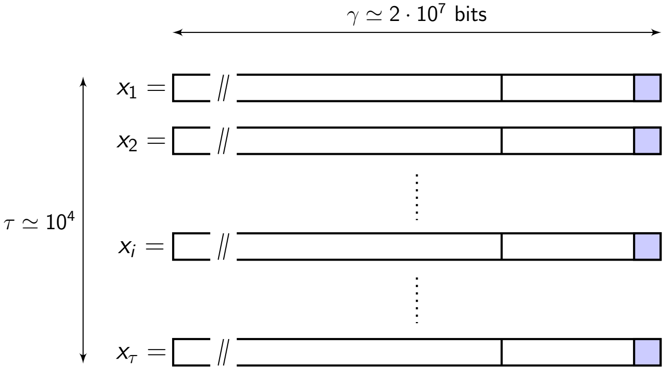
$$x_i = q_i \cdot p + 2r_i$$

- Public-key encryption:

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

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

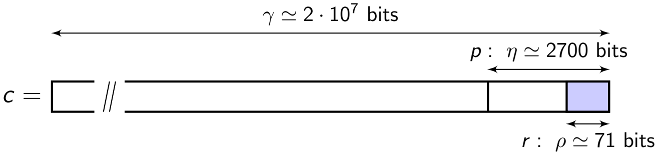
Public Key Size



- Public-key size: $\tau \cdot \gamma = 2 \cdot 10^{11}$ bits $= 25$ GB !
  - In [CMNT11], with quadratic encryption, PK size of 1 GB.

New: DGHV Ciphertext Compression

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

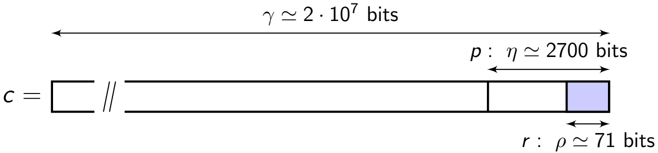


- Compute a pseudo-random $\chi = f(seed)$ of γ bits.

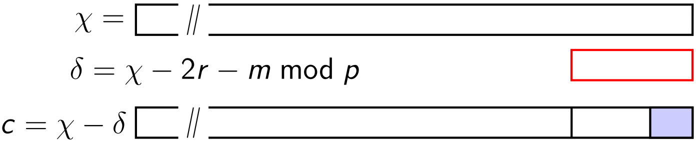
- Only store *seed* and the small correction δ.
- Storage: $\simeq 2700$ bits instead of $2 \cdot 10^7$ bits!

New: DGHV Ciphertext Compression

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



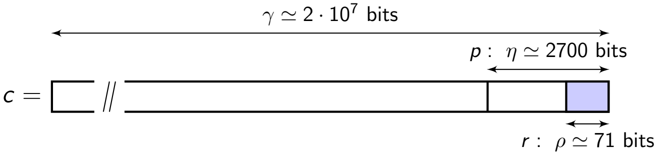
- Compute a pseudo-random $\chi = f(seed)$ of γ bits.



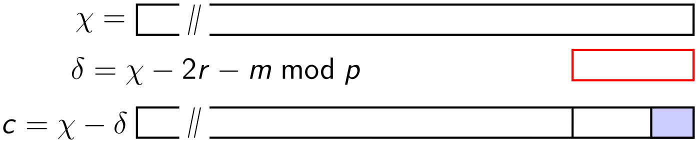
- Only store *seed* and the small correction δ.
- Storage: $\simeq 2\,700$ bits instead of $2 \cdot 10^7$ bits !

New: DGHV Ciphertext Compression

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



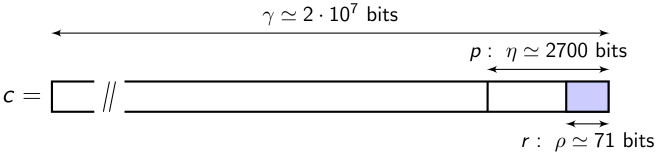
- Compute a pseudo-random $\chi = f(seed)$ of γ bits.



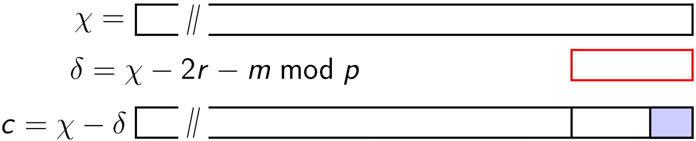
  - Only store *seed* and the small correction δ.
  - Storage: $\simeq 2\,700$ bits instead of $2 \cdot 10^7$ bits !

New: DGHV Ciphertext Compression

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

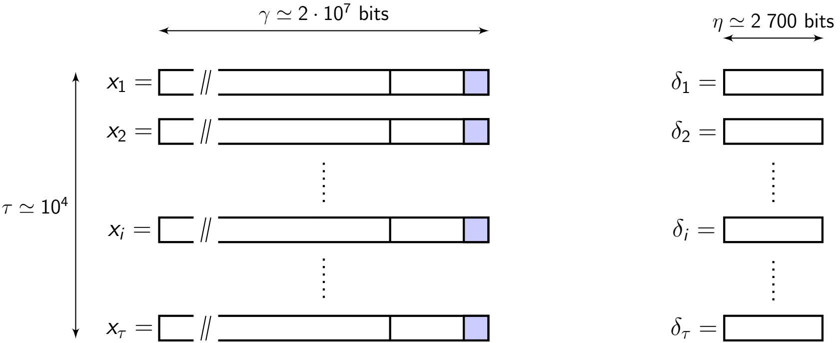


- Compute a pseudo-random $\chi = f(seed)$ of γ bits.

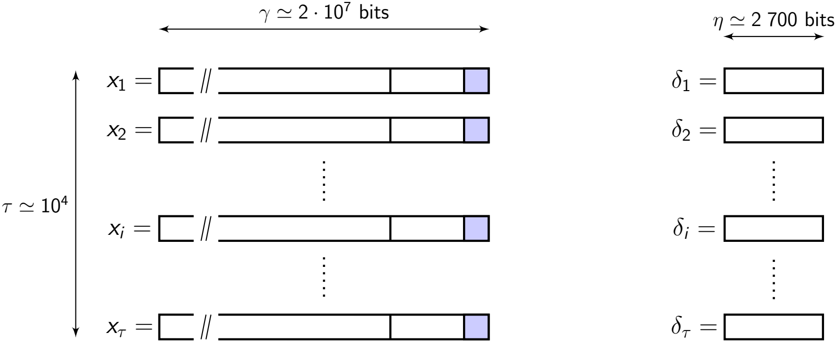


  - Only store *seed* and the small correction δ.
  - Storage: $\simeq 2\,700$ bits instead of $2 \cdot 10^7$ bits !

Compressed Public Key



Compressed Public Key



Old PK: 25 GB

New PK: 3.4 MB !

Security of Compressed PK

- Original 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.
- Compressed public key
  - *seed* is part of the public-key, to recover the x_i's, so we cannot argue that $f(seed)$ is pseudo-random.
  - Security in the random oracle model only, still based on approximate-gcd.

Security of Compressed PK

- Original 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.
- Compressed public key
  - *seed* is part of the public-key, to recover the x_i's, so we cannot argue that $f(seed)$ is pseudo-random.
  - Security in the random oracle model only, still based on approximate-gcd.

Security of Compressed PK

- Original 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.
- Compressed public key
  - *seed* is part of the public-key, to recover the x_i's, so we cannot argue that $f(seed)$ is pseudo-random.
  - Security in the random oracle model only, still based on approximate-gcd.

PK Generation

$$\chi_i = H(seed, i)$$
$$\delta_i = [\chi_i]_p + \lambda_i \cdot p - r_i$$
$$x_i = \chi_i - \delta_i$$

Security of Compressed PK

- Original 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.
- Compressed public key
  - *seed* is part of the public-key, to recover the x_i's, so we cannot argue that $f(seed)$ is pseudo-random.
  - Security in the random oracle model only, still based on approximate-gcd.

PK Generation (↓)

$\chi_i = H(seed, i)$
$\delta_i = [\chi_i]_p + \lambda_i \cdot p - r_i$
$x_i = \chi_i - \delta_i$

Simulation in ROM (↑)

$H(seed, i) \leftarrow x_i + \delta_i$
$\delta_i \leftarrow \{0,1\}^{\eta+\lambda}$
$x_i = q_i \cdot p + r_i$

PK size and timings

Instance	λ	ρ	η	γ	pk size	Recrypt
Toy	42	27	1026	$150 \cdot 10^3$	77 KB	0.41 s
Small	52	41	1558	$830 \cdot 10^3$	437 KB	4.5 s
Medium	62	56	2128	$4.2 \cdot 10^6$	2.2 MB	51 s
Large	72	71	2698	$19 \cdot 10^6$	10.3 MB	11 min

- Updated parameters to take into account the Chen-Nguyen attack.
- PK size: 10.3 MB instead of 1 GB in [CMNT11].

Hardness assumption for semantic security

- Original DGHV scheme: secure under the General Approximate Common Divisor (GACD) assumption.
  - Given polynomially many $x_i = p \cdot q_i + r_i$, find p.
- 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.
- PACD is clearly easier than GACD.
  - How much easier ?

Hardness assumption for semantic security

- Original DGHV scheme: secure under the General Approximate Common Divisor (GACD) assumption.
  - Given polynomially many $x_i = p \cdot q_i + r_i$, find p.
- 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.
- PACD is clearly easier than GACD.
  - How much easier ?

Hardness assumption for semantic security

- Original DGHV scheme: secure under the General Approximate Common Divisor (GACD) assumption.
  - Given polynomially many $x_i = p \cdot q_i + r_i$, find p.
- 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.
- PACD is clearly easier than GACD.
  - How much easier ?

Solving PACD

- Given $x_0 = p \cdot q_0$ and polynomially many $x_i = p \cdot q_i + r_i$, find p.
- Brute force attack: 2^ρ GCD computations.
  - with $x_0 = q_0 \cdot p$ and $x_1 = q_1 \cdot p + r_1$ and $0 \leq r_1 < 2^\rho$.
- Variant suggested by Phong Nguyen, still in $\mathcal{O}(2^\rho)$:

$$p = \gcd\left(x_0, \prod_{i=0}^{2^\rho - 1} (x_1 - i) \bmod x_0\right)$$

- Improved attack in $\tilde{\mathcal{O}}(2^{\rho/2})$ time and memory by Chen and Nguyen at Eurocrypt 2012.

Solving PACD

- Given $x_0 = p \cdot q_0$ and polynomially many $x_i = p \cdot q_i + r_i$, find p.
- Brute force attack: 2^ρ GCD computations.
  - with $x_0 = q_0 \cdot p$ and $x_1 = q_1 \cdot p + r_1$ and $0 \leq r_1 < 2^\rho$.
- Variant suggested by Phong Nguyen, still in $\mathcal{O}(2^\rho)$:

$$p = \gcd\left(x_0, \prod_{i=0}^{2^\rho-1}(x_1 - i) \bmod x_0\right)$$

- Improved attack in $\tilde{\mathcal{O}}(2^{\rho/2})$ time and memory by Chen and Nguyen at Eurocrypt 2012.

Solving PACD

- Given $x_0 = p \cdot q_0$ and polynomially many $x_i = p \cdot q_i + r_i$, find p.
- Brute force attack: 2^ρ GCD computations.
  - with $x_0 = q_0 \cdot p$ and $x_1 = q_1 \cdot p + r_1$ and $0 \le r_1 < 2^\rho$.
- Variant suggested by Phong Nguyen, still in $\mathcal{O}(2^\rho)$:

$$p = \gcd\left(x_0, \prod_{i=0}^{2^\rho - 1} (x_1 - i) \bmod x_0\right)$$

- Improved attack in $\tilde{\mathcal{O}}(2^{\rho/2})$ time and memory by Chen and Nguyen at Eurocrypt 2012.

Solving PACD

- Given $x_0 = p \cdot q_0$ and polynomially many $x_i = p \cdot q_i + r_i$, find p.
- Brute force attack: 2^ρ GCD computations.
  - with $x_0 = q_0 \cdot p$ and $x_1 = q_1 \cdot p + r_1$ and $0 \leq r_1 < 2^\rho$.
- Variant suggested by Phong Nguyen, still in $\mathcal{O}(2^\rho)$:

$$p = \gcd\left(x_0, \prod_{i=0}^{2^\rho - 1} (x_1 - i) \bmod x_0\right)$$

- Improved attack in $\tilde{\mathcal{O}}(2^{\rho/2})$ time and memory by Chen and Nguyen at Eurocrypt 2012.

Solving GACD

- Given polynomially many $x_i = p \cdot q_i + r_i$, find p.
  - Variant without $x_0 = q_0 \cdot p$.
- Brute force attack: $2^{2\rho}$ GCD computations.
  - From $x_1 = p \cdot q_1 + r_1$ and $x_2 = p \cdot q_2 + r_2$
- Using Chen-Nguyen attack: $\tilde{\mathcal{O}}(2^{3\rho/2})$ time.
  - Guess r_1 and apply Chen-Nguyen on r_2
  - $\mathcal{O}(2^{\rho}) \cdot \tilde{\mathcal{O}}(2^{\rho/2}) = \tilde{\mathcal{O}}(2^{3\rho/2})$ time and $\tilde{\mathcal{O}}(2^{\rho/2})$ memory.
- New attack: $\tilde{\mathcal{O}}(2^{\rho})$ time and memory.

Solving GACD

- Given polynomially many $x_i = p \cdot q_i + r_i$, find p.
  - Variant without $x_0 = q_0 \cdot p$.
- Brute force attack: $2^{2\rho}$ GCD computations.
  - From $x_1 = p \cdot q_1 + r_1$ and $x_2 = p \cdot q_2 + r_2$
- Using Chen-Nguyen attack: $\tilde{\mathcal{O}}(2^{3\rho/2})$ time.
  - Guess r_1 and apply Chen-Nguyen on r_2
  - $\mathcal{O}(2^{\rho}) \cdot \tilde{\mathcal{O}}(2^{\rho/2}) = \tilde{\mathcal{O}}(2^{3\rho/2})$ time and $\tilde{\mathcal{O}}(2^{\rho/2})$ memory.
- New attack: $\tilde{\mathcal{O}}(2^{\rho})$ time and memory.

Solving GACD

- Given polynomially many $x_i = p \cdot q_i + r_i$, find p.
  - Variant without $x_0 = q_0 \cdot p$.
- Brute force attack: $2^{2\rho}$ GCD computations.
  - From $x_1 = p \cdot q_1 + r_1$ and $x_2 = p \cdot q_2 + r_2$
- Using Chen-Nguyen attack: $\tilde{\mathcal{O}}(2^{3\rho/2})$ time.
  - Guess r_1 and apply Chen-Nguyen on r_2
  - $\mathcal{O}(2^{\rho}) \cdot \tilde{\mathcal{O}}(2^{\rho/2}) = \tilde{\mathcal{O}}(2^{3\rho/2})$ time and $\tilde{\mathcal{O}}(2^{\rho/2})$ memory.
- New attack: $\tilde{\mathcal{O}}(2^{\rho})$ time and memory.

Solving GACD

- Given polynomially many $x_i = p \cdot q_i + r_i$, find p.
  - Variant without $x_0 = q_0 \cdot p$.
- Brute force attack: $2^{2\rho}$ GCD computations.
  - From $x_1 = p \cdot q_1 + r_1$ and $x_2 = p \cdot q_2 + r_2$
- Using Chen-Nguyen attack: $\tilde{\mathcal{O}}(2^{3\rho/2})$ time.
  - Guess r_1 and apply Chen-Nguyen on r_2
  - $\mathcal{O}(2^{\rho}) \cdot \tilde{\mathcal{O}}(2^{\rho/2}) = \tilde{\mathcal{O}}(2^{3\rho/2})$ time and $\tilde{\mathcal{O}}(2^{\rho/2})$ memory.
- New attack: $\tilde{\mathcal{O}}(2^{\rho})$ time and memory.

New Attack against GACD

- Given polynomially many $x_i = p \cdot q_i + r_i$, find p.
- Variant of the previous equation with $x_1 = p \cdot q_1 + r_1$ and $x_2 = p \cdot q_2 + r_2$

$$p \mid \gcd\left(\prod_{i=0}^{2^\rho-1}(x_1 - i), \prod_{i=0}^{2^\rho-1}(x_2 - i)\right)$$

  - Product over $\mathbb{Z}$ can be computed in $\tilde{O}(2^\rho)$ time using a product tree.
  - $\tilde{O}(2^\rho)$ time and memory.
- Problem: many parasitic factors.
  - Can be eliminated by taking the gcd with more products,
  - and by dividing by $B!$ for $B \simeq 2^\rho$.

New Attack against GACD

- Given polynomially many $x_i = p \cdot q_i + r_i$, find p.
- Variant of the previous equation with $x_1 = p \cdot q_1 + r_1$ and $x_2 = p \cdot q_2 + r_2$

$$p \,\Big|\, \gcd\left(\prod_{i=0}^{2^\rho-1}(x_1 - i), \prod_{i=0}^{2^\rho-1}(x_2 - i)\right)$$

  - Product over $\mathbb{Z}$ can be computed in $\tilde{\mathcal{O}}(2^\rho)$ time using a product tree.
  - $\tilde{\mathcal{O}}(2^\rho)$ time and memory
- Problem: many parasitic factors.
  - Can be eliminated by taking the gcd with more products,
  - and by dividing by $B!$ for $B \simeq 2^\rho$.

New Attack against GACD

- Given polynomially many $x_i = p \cdot q_i + r_i$, find p.
- Variant of the previous equation with $x_1 = p \cdot q_1 + r_1$ and $x_2 = p \cdot q_2 + r_2$

$$p \mid \gcd\left(\prod_{i=0}^{2^\rho-1}(x_1 - i), \prod_{i=0}^{2^\rho-1}(x_2 - i)\right)$$

  - Product over $\mathbb{Z}$ can be computed in $\tilde{\mathcal{O}}(2^\rho)$ time using a product tree.
  - $\tilde{\mathcal{O}}(2^\rho)$ time and memory
- Problem: many parasitic factors.
  - Can be eliminated by taking the gcd with more products,
  - and by dividing by $B!$ for $B \simeq 2^\rho$.

New Attack against GACD

- Given polynomially many $x_i = p \cdot q_i + r_i$, find p.
- Variant of the previous equation with $x_1 = p \cdot q_1 + r_1$ and $x_2 = p \cdot q_2 + r_2$

$$p \,|\, \gcd\left(\prod_{i=0}^{2^\rho-1}(x_1 - i), \prod_{i=0}^{2^\rho-1}(x_2 - i)\right)$$

  - Product over $\mathbb{Z}$ can be computed in $\tilde{\mathcal{O}}(2^\rho)$ time using a product tree.
  - $\tilde{\mathcal{O}}(2^\rho)$ time and memory
- Problem: many parasitic factors.
  - Can be eliminated by taking the gcd with more products,
  - and by dividing by $B!$ for $B \simeq 2^\rho$.

New Attack against GACD

- Given polynomially many $x_i = p \cdot q_i + r_i$, find p.
- Variant of the previous equation with $x_1 = p \cdot q_1 + r_1$ and $x_2 = p \cdot q_2 + r_2$

$$p \mid \gcd\left(\prod_{i=0}^{2^\rho - 1}(x_1 - i), \prod_{i=0}^{2^\rho - 1}(x_2 - i)\right)$$

  - Product over $\mathbb{Z}$ can be computed in $\tilde{\mathcal{O}}(2^\rho)$ time using a product tree.
  - $\tilde{\mathcal{O}}(2^\rho)$ time and memory
- Problem: many parasitic factors.
  - Can be eliminated by taking the gcd with more products,
  - and by dividing by $B!$ for $B \simeq 2^\rho$.

Source Code in SAGE

```
def attackGACD(rho=12,gam=1000,eta=100):
  p=random_prime(2^eta)
  s=rho

  B=floor(2^(1.*rho*(s+1)/(s-1)))
  fa=factorial(B)

  for j in range(1,s):
    x=p*ZZ.random_element(2^(gam-eta))+ \
        ZZ.random_element(2^rho)
    z=prod([x-i for i in range(2^rho)])
    if j==1: g=z; continue
    g=gcd(g,z)
    g=prime_to_m_part(g,fa)
    if g.nbits()==p.nbits(): break
```

GACD Attack Running Time

Instance	ρ	γ	time	time [CN12]
Micro	12	10^4	40 s	
Toy (Section 8)	13	$61 \cdot 10^3$	13 min	
Toy' ([CN12])	17	$1.6 \cdot 10^5$	17 h	*3495 h* (est.)

- Chen-Nguyen attack: $\mathcal{O}(2^{3\rho/2})$ time and $\mathcal{O}(2^{\rho/2})$ memory.
- Our attack: $\mathcal{O}(2^{\rho})$ time and memory
- Time-memory tradeoffs are possible.

Conclusion

- Smaller public key size for the DGHV fully homomorphic encryption scheme.
  - 10 MB instead of 1 GB
- Better attack against approximate-gcd without $x_0 = q_0 \cdot p$
  - $\tilde{\mathcal{O}}(2^{\rho})$ complexity instead of $\tilde{\mathcal{O}}(2^{3\rho/2})$
- In the proceedings:
  - Generalization of [CMNT11] quadratic encryption technique to higher degrees.
  - DGHV without bootstrapping: analogous to RLWE without bootstrapping [BGV11].

Conclusion

- Smaller public key size for the DGHV fully homomorphic encryption scheme.
  - 10 MB instead of 1 GB
- Better attack against approximate-gcd without $x_0 = q_0 \cdot p$
  - $\tilde{\mathcal{O}}(2^{\rho})$ complexity instead of $\tilde{\mathcal{O}}(2^{3\rho/2})$
- In the proceedings:
  - Generalization of [CMNT11] quadratic encryption technique to higher degrees.
  - DGHV without bootstrapping: analogous to RLWE without bootstrapping [BGV11].

Conclusion

- Smaller public key size for the DGHV fully homomorphic encryption scheme.
  - 10 MB instead of 1 GB
- Better attack against approximate-gcd without $x_0 = q_0 \cdot p$
  - $\tilde{\mathcal{O}}(2^{\rho})$ complexity instead of $\tilde{\mathcal{O}}(2^{3\rho/2})$
- In the proceedings:
  - Generalization of [CMNT11] quadratic encryption technique to higher degrees.
  - DGHV without bootstrapping: analogous to RLWE without bootstrapping [BGV11].