TP: Fully Homomorphic Encryption Scheme

Jean-Sébastien Coron

Université du Luxembourg

1 DGHV Somewhat Homormorphic Encryption Scheme

Implement the basic DGHV encryption scheme [4], with encryption and decryption. You can use the SAGE library [1]. Check that homomorphic addition and multiplication works.

2 Optional: approximate GCD problem: GCD attack and lattice attack

The Partial Approximate Common Divisor problem consists in recovering p, given $x_0 = p \cdot q_0$ and polynomially many $x_i = p \cdot q_i + r_i$.

1. Implement the brute force attack on the noise.
2. Implement the improved attack from [3].
3. Implement the lattice attack.
4. Compare the practical complexities of these attacks.

3 Optional: LWE encryption

In [2], the authors described a fully homomorphic encryption scheme based on the learning with errors (LWE) assumption. In particular, they introduced a new relinearization technique for performing the ciphertext multiplication for a "somewhat homomorphic" encryption scheme based on LWE.

3.1 Basic LWE encryption

Let $q \in \mathbb{Z}$. Let $\vec{s} \in \mathbb{Z}$ be the secret-key. For simplicity we can take $\vec{s} \in \{0,1\}^n$. An LWE ciphertext for a message $m \in \{0,1\}$ is $\vec{c} \in \mathbb{F}_q^n$ such that

$$\langle \vec{c}, \vec{s} \rangle = 2e + m \pmod q$$

with $s_1 = 1$. For the error e, we can take the binomial distribution χ with parameter κ, for some small κ. One can let $e = h(u) - h(v)$ where $u, v \leftarrow \{0,1\}^\kappa$, where h is the Hamming weight function.

```
def binom(kappa=2):
  return hw(ZZ.random_element(2^kappa))-hw(ZZ.random_element(2^kappa))
```

To encrypt, one can use a matrix $\mathbf{A} \in \mathbb{F}_q^{m \times n}$ of row vectors $\vec{a}_i \in \mathbb{F}_q^n$, such that $\langle \vec{a}_i, \vec{s} \rangle = e_i$ for $e_i \leftarrow \chi$, for all $1 \le i \le m$. This can be written $\mathbf{A} \cdot \vec{s} = \vec{e} \pmod q$. To encrypt a message $\mu \in \{0,1\}$, one generates a linear combination of the vectors $\vec{a}_i$:

$$\vec{c} = (\mu, 0, \ldots, 0) + 2 \sum_{i=1}^{m} u_i \cdot \vec{a}_i = (\mu, 0 \ldots, 0) + 2\vec{u} \cdot \mathbf{A} \pmod q$$

where $\vec{u} \leftarrow \chi^m$. For decryption, we compute:

$$\langle \vec{c}, \vec{s} \rangle = \mu + 2\vec{u} \cdot \mathbf{A} \cdot \vec{s} = \mu + 2\langle \vec{u}, \vec{e} \rangle \pmod{q}$$

For correct decryption, we must have $|\langle \vec{u}, \vec{e} \rangle| < q/4$. We can fix the parameters so that this is the case, except with negligible probability. For a constant κ, the distribution of $\langle \vec{u}, \vec{e} \rangle$ looks like a Gaussian with standard deviation $\mathcal{O}(\sqrt{n})$. Hence we can take $q = \mathcal{O}(\sqrt{n})$. We can take $m = \mathcal{O}(n)$, for a proof of security based on the leftover hash lemma.

3.2 Ciphertext multiplication and relinearization

Let $\vec{c}_1$ and $\vec{c}_2$ be two LWE ciphertexts, with $\langle \vec{c}_1, \vec{s} \rangle = 2e_1 + m_1 \pmod{q}$ and $\langle \vec{c}_2, \vec{s} \rangle = 2e_2 + m_2 \pmod{q}$. We can write:

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

which gives:

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

for $e = 2e_1 e_2 + m_1 e_2 + m_2 e_1$. Hence $\vec{c}' = (c_{1,i} \cdot c_{2,j})_{i,j} \in \mathbb{F}_q^{n^2}$ is a new LWE ciphertext for the secret-key $\vec{s}^{'} = (s_i \cdot s_j)_{i,j} \in \mathbb{Z}^{n^2}$, with

$$\langle \vec{c}', \vec{s}^{'} \rangle = 2e + m_1 m_2 \pmod{q}$$

We would like to obtain a ciphertext with binary components only, so we define the function:

$$\mathsf{BitDecomp}(\vec{u}) = (\ldots, u_{ij}, \ldots)$$

that computes the binary decomposition of each component $u_i = \sum_{j=0}^{n_q - 1} 2^j u_{ij}$, with $n_q = \lceil \log_2 q \rceil$. Similarly we define the function $\mathsf{PowerOf2}$ with:

$$\mathsf{PowerOf2}(\vec{v}) = (\ldots, v_i, 2v_i, \ldots, 2^{n_q - 1} v_i, \ldots)$$

and we have for any $\vec{u}$, $\vec{v}$:

$$\langle \vec{u}, \vec{v} \rangle = \langle \mathsf{BitDecomp}(\vec{u}), \mathsf{PowerOf2}(\vec{v}) \rangle$$

Therefore we can define a new ciphertext $\vec{c}^{''} = \mathsf{BitDecomp}(\vec{c}^{'}) \in \{0,1\}^{n^2 \cdot n_q}$ which satisfies:

$$\langle \vec{c}'', \vec{s}^{''} \rangle = 2e + m_1 m_2 \pmod{q}$$

for the new secret-key $\vec{s}^{''} = \mathsf{PowerOf2}(\vec{s}^{'})$.

To get back a ciphertext in $\mathbb{F}_q^n$, we compute an encryption of each component of the secret-key $\vec{s}^{''}$. We define the LWE ciphertexts $\vec{t}_i$ for $1 \leq i \leq n^2 \cdot n_q$ such that $\langle \vec{t}_i, \vec{s} \rangle = 2f_i + s_i'' \pmod{q}$. We can then define the new relinearized ciphertext:

$$\vec{d} = \sum_{i=1}^{n^2 \cdot n_q} c_i'' \cdot \vec{t}_i$$

and we get:

$$\langle \vec{d}, \vec{s} \rangle = 2 \sum_{i=1}^{n^2 \cdot n_q} c_i'' f_i + 2e + m_1 m_2$$

and therefore $\vec{d}$ is an LWE encryption of $m_1 m_2$.

3.3 Instructions

You are asked to implement a cryptographic construction, the "somewhat homomorphic" BV11 scheme, in Python using the Sage library. Only an elementary version is required, working for small parameters only. The goal is to show that you have a basic understanding of the corresponding algorithms. Please keep your code readable and reasonably short (roughly 100-150 lines of code). More precisely, you must:

1. Write a short (3-4 pages) report explaining why the scheme works, based on the succinct description above. For clarity, you can embed some snippets of your code in the report.
2. Write an implementation in Sage of the scheme, including the key generation, public-key encryption, decryption, test of correct decryption, homomorphic multiplication with relinearization, and test of correct homomorphic multiplication.

References

1. Sage Mathematical Library, Available at `http://www.sagemath.org/`
2. Zvika Brakerski, Vinod Vaikuntanathan: Efficient Fully Homomorphic Encryption from (Standard) LWE. FOCS 2011: 97-106
3. Yuanmi Chen, Phong Q. Nguyen: Faster Algorithms for Approximate Common Divisors: Breaking Fully-Homomorphic-Encryption Challenges over the Integers. EUROCRYPT 2012: 502-519
4. Marten van Dijk, Craig Gentry, Shai Halevi, Vinod Vaikuntanathan: Fully Homomorphic Encryption over the Integers. EUROCRYPT 2010: 24-43.