Which Ring-Based SHE Scheme is best?

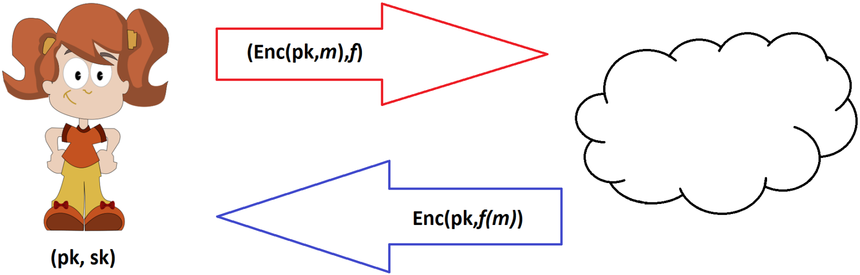
Anamaria Costache and Nigel P. Smart

University of Bristol

Fully Homomorphic Encryption

- Homomorphic encryption allows to compute on encrypted data.
- Allows to outsource computation to an untrusted server.
    - Signal processing satellite applications.
    - Analysing data (e.g. medical data) without compromising confidential information.

Fully Homomorphic Encryption



Fully Homomorphic Encryption

- A (fully) homomorphic encryption scheme $\mathcal{E}$ comprises of four algorithms: KeyGen, Enc, Dec and Evaluate.
- For $(sk, pk) \leftarrow \text{KeyGen}(\lambda)$, plaintext message m with corresponding ciphertext c and circuit $\mathcal{C}$, we say that $\mathcal{E}$ is **correct** if

$$\text{Dec}(sk, \text{Evaluate}(pk, \mathcal{C}, c)) = \mathcal{C}(m).$$

- $\mathcal{E}$ is
  - **Fully Homomorphic** if it is correct for all circuits $\mathcal{C}$.
  - **Somewhat Homomorphic** if it is correct for some circuits $\mathcal{C}$.

Fully Homomorphic Encryption

- RSA encryption is multiplicatively homomorphic [Rivest Shamir Adleman 77].
- Paillier is additively homomorphic [Paillier 99].
- A scheme both additively and multiplicatively homomorphic is more powerful, but also harder to obtain.

A History of Homomorphic Encryption

- First Generation: Gentry's first FHE scheme, bootstrappable [Gentry 09]
- Second Generation: Ring-Based leveled Somewhat Homomorphic Schemes, smaller ciphertexts. Use double-CRT to achieve a SIMD system and enhance efficiency. [Gentry Halevi Smart 11]
- Third Generation: Schemes such as [Gentry Sahai Waters 13]. Integer-based schemes, but slower computations and somewhat impractical.

The problem

- Different applications call for different parameters. For example plaintext spaces vary, or depth of the circuit we want to evaluate.
- Ideally we want an unbounded scheme, but not all applications require this.
- Even when restricted to a certain form of HE, there are many schemes available.

- We pick four of the most used Ring-Based schemes, BGV, FV, NTRU and YASHE and compare them against each other.
- On the face of it, YASHE and FV should be more efficient since they are scale-invariant, which should save in computation time.
- Similarly, NTRU and YASHE have fewer ring elements in the ciphertexts.
- What effect do the above have on the efficiency of the scheme?

A Noise Problem

- All messages are encrypted by adding a noise factor to a multiple of the original message.

$$Enc(pk, m) = c = \alpha \cdot m + e \pmod{q}.$$

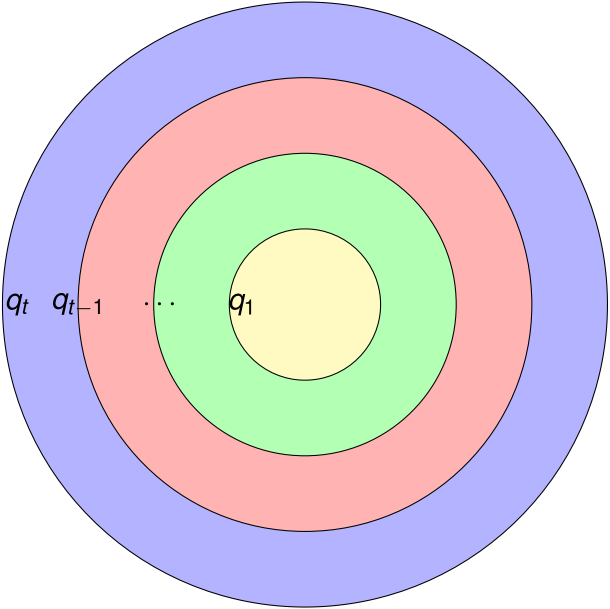
- But then $c \cdot c$ has noise $2 \cdot \alpha \cdot m + e^2$:

$$c \cdot c = (\alpha \cdot m + e) \cdot (\alpha \cdot m + e) = \alpha^2 \cdot m^2 + 2 \cdot \alpha \cdot m + e^2.$$

- This grows quickly, implying a need for a noise-management control.

A Noise Management Technique: SwitchModulus

- We use a chain of primes $p_0 < p_1 < \cdots < p_{L-1}$ and let $q_t = \prod_{i=0}^{t} p_i$.
- This gives a chain of moduli $q_0 < q_1 < \cdots < q_{L-1}$ such that $q_i \mid q_{i+1}$.


q_t
q_{t-1}
$\cdots$
q_1

The four schemes; $\mathsf{Dec}_{pk}^{\mathsf{BGV}}(\mathfrak{c})$

Decryption of a ciphertext $((c_0, c_1), t)$ at level t is performed by setting

$$m' \leftarrow [c_0 - sk \cdot c_1]_{q_t},$$

and outputting

$$m' \bmod p.$$

The four schemes; $\mathsf{Dec}_{pk}^{\mathsf{YASHE}}(\mathfrak{c})$

Decryption of a ciphertext (c, t) at level t is performed by setting

$$m' \leftarrow \left\lceil \frac{p}{q_t} \cdot [c \cdot sk]_{q_t} \right\rfloor,$$

and outputting

$$m' \bmod p.$$

How do we compare the four schemes?

- We follow the security analysis in [Gentry Halevi Smart 13], which itself follows on from Lindner-Peikert [Lindner Peikert 10].
- We assume that we encrypt, perform ζ additions, one multiplication, ζ additions, one multiplication and so on. We perform a SwitchKey operation and a Scale after each multiplication.
- We measure efficiency by the size of a ciphertext in kBytes.

Analysis

- Decryption is done by either modular reduction or a rounding operation. Thus if the noise is too large, we could decrypt erroneously.
- To ensure correct decryption, we require

$$4 \cdot c_m \cdot B^*_{\text{scale}} = 2 \cdot c_m \cdot B < \begin{cases} p_0 & \text{For BGV and NTRU} \\ \left\lfloor \frac{p_0}{p} \right\rfloor & \text{For FV and YASHE.} \end{cases} \tag{1}$$

- This gives us a lower bound on our bottom modulus.

Top modulus

- We want to find the sizes of the primes used in moduli. We start with the top level and calculate the primes we need with correct decryption in mind.
- We start off with a fresh ciphertext. We perform a number of additions, one multiplication and one scale operation, and calculate a noise bound B_2 on the resulting ciphertext.
- We require

$$p_{L-1} \approx \left\lceil \frac{B_2}{B^*_{\text{scale}}} \right\rceil.$$

Middle moduli

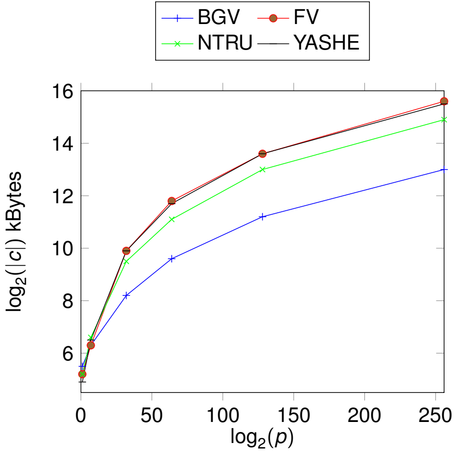
- For the middle moduli, we use the same methodology. The only difference is that that we do not start off with a fresh ciphertext, so the initial noise will be different.
- We call this bound $B'(t)$, and we require

$$p_t \approx \left\lceil \frac{B'(t)}{B^*_{\text{scale}}} \right\rceil.$$

- We can then iterate downwards, using

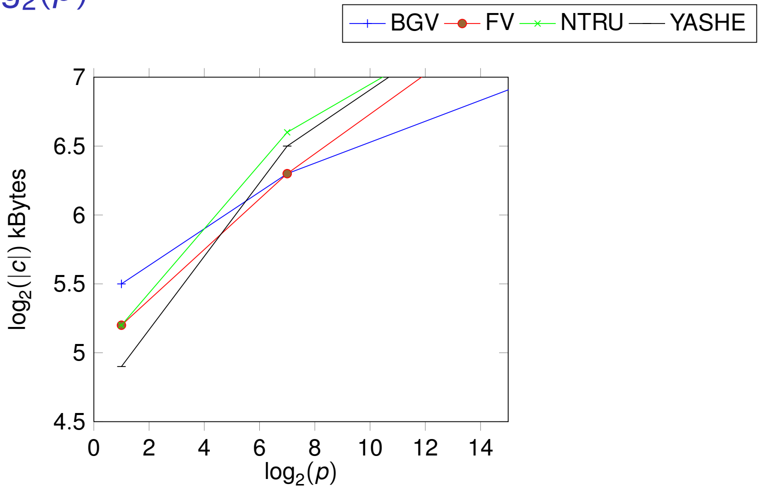
$$\log_2 q_t = \log_2 q_{t+1} - \log_2 p_{t+1}.$$

Results; $L = 5$ and varying plaintext modulus size $\log_2(p)$



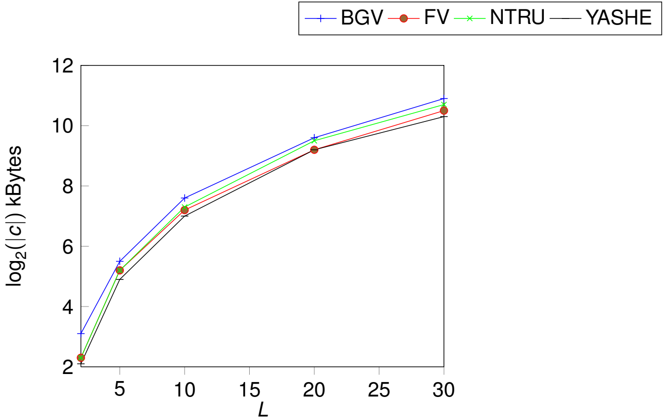
We see that the BGV scheme quickly takes over all other values.

Results; $L = 5$ and varying plaintext modulus size $\log_2(p)$



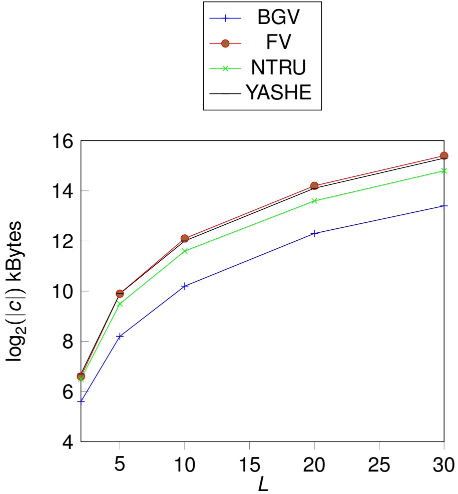
For small values of p, YASHE is indeed preferable. But as seen in the previous slide, BGV is better overall.

Results; plaintext modulus $p = 2$, for varying depth L



As previously, YASHE wins for small p...

Results; plaintext modulus $p = 2^{32}$, for varying depth L



... and BGV for large p. In fact, the size of L has no impact on the schemes' performance.

Open questions

- We have done a crude security analysis, in order to examine how the scheme parameters are affected by scaling the plaintext modulus p and the depth required of the scheme.

- A stricter security analysis would contribute to the survey. This would need to take into account attacks such as [Albrecht Bai Ducas 16].

Thank you!

Any questions?

CT-RSA Conference 2016

NFLlib

NTT-based Fast Lattice Library

Carlos Aguilar-Melchor[1] **Joris Barrier**[2] Serge Guelton[3] Adrien Guinet[3]
Marc-Olivier Killijian[2] Tancrède Lepoint[4]

[1]Université de Toulouse, CNRS, France, *carlos.aguilar@enseeiht.fr*

[2]Université de Toulouse, CNRS, France, *{joris.barrier,marco.killijian}@laas.fr*

[3]Quarkslab, France, *{sguelton,aguinet}@quarkslab.com*

[4]CryptoExperts, France, *tancrede.lepoint@cryptoexperts.com*

February 23, 2016

Outline

A Brief Overview

A Library...

NFLlib is a homemade C++ library to efficiently deal with polynomials.

...Specialized

Indeed, NFLlib works exclusively with polynomials usually considered in (ideal) lattice-based cryptography.

- polynomials of fixed degree (a power of two),
- with coefficient of fixed size (modular operations).

$$P(X) = a_0 + a_1X + a_2X^2 + \cdots + a_{n-1}X^{n-1} + a_nX^n$$

A Brief Overview

A Library...

NFLlib is a homemade C++ library to efficiently deal with polynomials.

...Specialized

Indeed, NFLlib works exclusively with polynomials usually considered in (ideal) lattice-based cryptography.

- polynomials of fixed degree (a power of two),
- with coefficient of fixed size (modular operations).

$$P(X) = a_0 + a_1 X + a_2 X^2 + \cdots + a_{n-1} X^{n-1} + a_n X^n$$

How to use NFLlib : Practice example

```
/* Set polynomial type with T the native type used
* such as uint16_t, uint32_t, uint64_t */
using poly_t = nfl::poly_from_modulus<T, degree, modulus>;
poly_t p1, p2, p3, p_res;

/*Fill polynomials with noise using different noise generators */
p1 = poly_t(nfl::uniform); //or p1 = nfl::uniform;
p2 = poly_t(nfl::gaussian<poly_t>(prng_instance));
p3 = poly_t(nfl::bounded(bound));

/*Overloaded operators for an easy use */
p_res = (p1 * p2) + p3 - p1;
```

NFLlib

What is in the box ?

Enabled Optimizations

NFLlib is a `C++` library with state of the art optimizations :

- Specific modulus ;
- NTT polynomial representation ;
- CRT representation to use big modulus ;
- NTT and iNTT optimized algorithm ;
- SSE and AVX2 processor instructions.

Remark : HElib

This kind of optimizations are implemented in HElib in the `DoubleCRT` class.

Modulus Optimizations

We choose our primes such as for an integer $1 \leq s_0 \leq s-1$, a chosen prime p verifies (Note that all our 62-bit primes verify Eq. 1) :

$$(1+1/2^{3s_0}) \cdot \beta/(2^{s_0}+1) \ < \ p \ < \ \beta/2^{s_0} \tag{1}$$

Algorithm 1: Modular reduction with a modulus verifying Eq. 1

Input: $u = \langle u_1, u_0 \rangle \in [0, p^2)$, p verifying Eq. (1), $v_0 = \lfloor \beta^2/p \rfloor$ mod β, $1 \leq s_0 \leq s-1$ margin bits

Output: $r = u$ mod p

$q \leftarrow v_0 \cdot u_1 + 2^{s_0} \cdot u$ mod β^2

$r \leftarrow u - \lfloor q/\beta \rfloor \cdot p$ mod β

if $r \geq p$ **then** $r \leftarrow r - p$

return r

Algo. 1 is a significantly improvement from Moller, N., Granlund, T., "*Improved division by invariant integers*". IEEE Trans. Computers (2011).

NTT form

Polynomials representation

In NFLlib polynomials are represented and handled in an evaluated form using the Number Theoretic Transform (Discrete Fourrier Transform).

Advantages

By the book, polynomials multiplication is in $O(n^2)$. In the NTT form, the multiplication is an element-to-element multiplication in (obviously) $O(n)$.
$\rightarrow$ Great performance improvement

NTT form

Polynomials representation

In NFLlib polynomials are represented and handled in an evaluated form using the Number Theoretic Transform (Discrete Fourrier Transform).

Advantages

By the book, polynomials multiplication is in $O(n^2)$. In the NTT form, the multiplication is an element-to-element multiplication in (obviously) $O(n)$.

$\rightarrow$ Great performance improvement

CRT Representation

Motivation

For performance reason we do not use specialized libraries to handle moduli that do not fit in native types when working directly with polynomials. However, we don't want to limit too strictly moduli sizes. So we use Chinese Theorem Representation (CRT) to deal with big moduli by splitting them in smaller integers.

Recover

To recover big moduli we call an external library because we cannot do a better implementation.

HElib

Note that in HElib they use FFT representation for big modulus instead of CRT.

Gaussian Random Generator

Description

```
unsigned int sigma = 20;
unsigned int security = 128;
unsigned int sample = 1 << 14;

FastGaussianNoise<uint8_t, T, 2> fg_prng(sigma, security, sample);
```

Distribution	Uniform	$D_{3.19}$	D_{300}
cycles / bit generated[1]	0.4	1.39	3.43

[1]We implement a constant time algorithm with a $\times 4$ overhead

Applications : Key Exchange & SFHE

High Performance Key Exchange

Key Exchange Protocol

To illustrate the performances of our library in a concrete setting we implement an equivalent of the key transport protocol RSASVE of NIST SP 800 56B. The client chooses a random message and encrypts it with the server public key then, the server decrypts this random value that is used to derivate (with a hashing function) a common secret.

Protocol	80 bits	128 bits	256 bits
RSA	7.95 Kops/s	0.31 Kops/s	N/A
ECDH	7.01 Kops/s	5.93 Kops/s	1.61 Kops/s
RLWE/NFLlib [2]	N/A	**1020 Kops/s**	**508 Kops/s**

[2] Enabled forward secrecy divides performances by 2

Somewhat Fully Homomorphic Encryption

SFHE

We modified the open-source implementation of the somewhat homomorphic encryption scheme of Fan and Vercauteren from [1] and directly replaced flint by NFLlib .

	`Encrypt`	`Decrypt`	`Hom. Add.`	`Hom. Mult.`
[1] with flint	26.7ms	13.3ms	1.1ms	91.2ms
[1] with NFLlib	**0.9ms**	**0.9 ms**	**0.01ms**	**17.2ms**
Gain	$\times 30$	$\times 15$	$\times 110$	$\times 5.5$

1. Tancrède Lepoint and Michael Naehrig. "*A Comparison of the Homomorphic Encryption Schemes FV and YASHE*"

Application : PIR

Private Information Retrieval

Computational Private Information Retrieval (PIR)

A PIR scheme is a protocol in which a user retrieves a record from a database while hiding which from the database administrators. A computational PIR protocol requires that the database server executes an homomorphic cryptography based algorithm over all the database content.

Protocol	[2]	[3]	[4]
Throughput	0.5 Gb/s	1 Gb/s	20 Gb/s

2. J. T. Trostle and A. Parrish, "*Efficient computationally private information retrieval from anonymity or trapdoor groups*," in ISC 2010

3. C. Aguilar Melchor and P. Gaborit, "*A Fast Private Information Retrieval Protocol*," in ISIT'08

4. cPIR based on Lipmaa scheme using lattice based cryptography implemented with NFLlib

Conclusion

NFLlib is an optimized and efficient library designed to handle polynomials over polynomials rings $\mathbb{Z}_p[x]/(x^n+1)$ in NTT form.

It can be used as a building block for ideal lattice based cryptography that can be more efficient than existing implementations based on NTL or flint .

Code available at : `https://github.com/quarkslab/NFLlib`