Implementing Ring-LWE cryptosystems

Tore Vincent Carstens

December 16, 2016

Contents

1 Introduction

Here we study mainly two public key encryption schemes called LPR (described in [1]) and NTRU (described in [2]), which are based on Lattice based crypto and take a general look at Lattice based encryption schemes. The goal is to achieve 80 bits of security in all cases.

1.1 Motivation

State-of-the-art public key encryption schemes like ElGamal and RSA have the following problems:

- Their security proof is based on average-case-assumptions, i.e. on the assumption that some problem is (in some specific way) "hard" to solve. But worst-case-assumptions (also called complexity theoretical assumptions) are more trusted as they more widely studied.
- The invention of a universal quantum computer would render all state-of-the-art public key encryption schemes insecure as there are efficient quantum algorithms, that can break the encryption schemes.
- Sometimes it would be nice to be able to do operations on encrypted data, without decrypting it first.

Lattice based crypto can help in that sense, as

- It can in many cases be proven secure using only worst-case-assumptions.
- Lattice based crypto is believed to be secure against attacks, that make use of quantum effects.
- Some lattice based encryption schemes enable some doing operations mostly addition and sometimes also multiplication on the encrypted data. (so called homomorphic encryption)

2 Lattice Based Crypto

2.1 General Idea

Imagine you are given a set of linear independent vectors (called basis). The integer linear combinations of these vectors form a lattice. If the dimension is big it seems nearly impossible to find for one point in the vector space the closest point in the lattice. This is known as the closest vector problem (CVP) and it is assumed to be hard for big dimensions. On the other hand if the vectors in the basis are pairwise orthogonal it is trivial to find the closest point in the lattice.

Suppose someone secretly knows such an orthogonal basis. Then he can solve the CVP, which normal people can't solve. So he has some kind of a backdoor. That's the basic idea that helps to come up with a public key encryption scheme. The orthogonal basis corresponds to the private key, the non-orthogonal basis corresponds to the public key, the point in the vector space to the cipher text and the point in the lattice to the plain text. Everyone can pick a point in the lattice and add some small "error" to it, but only the person in possesion of the private key can get the point back.

2.2 Going to the ring

Dealing with entire basises as private and public key is very inefficient, because the number of vectors in the basis is equal to the dimension of the lattice. A possibility is needed to represent the whole basis by only one vector, and indeed it exists and is done in the case of LPR.

We want the vector to "wrap around" through all dimensions (i.e. we want that one vector somehow describes a whole basis of the vector space). The observation here is, that there is a freedom in which ring of scalars to choose. So nobody stops us from using a polynomial ring as the ring of scalars. It is used

$$R_q = \mathbb{Z}_q \Big/ (X^n + 1)$$

Now by repeatedly multiply with the polynomial

$$f(X) = X$$

we can "wrap around" the whole dimension and starting with some g after n step we are back with

$$X^n g = (X^n + 1 - 1)g = (X^n + 1)g - g \equiv -g$$

3 Algorithms

3.1 LPR

(see [1] for more details)

Parameters

$$n, q, m, s \in \mathbb{N}; \alpha, \sigma, \eta \in \mathbb{R}$$

Here the "good" orthogonal basises will be the r_i and their shifts around by multiplying x. And the "bad" non-orthogonal basises will be the a_i and their shifts around.

For the key generation m polynomial r_i with random 0-1-coefficients and m polynomials a_i with totally random coefficients are chosen. a_m and r_m are picked in a way such that

$$\sum_i r_i \cdot a_i = 0$$

precisely

KeyGen()
$\lceil$ **for** $i = 1$ to m
$r_i \xleftarrow{\$} \{0,1\}^n$
for $i = 1$ to m
$a_i \xleftarrow{\$} \{0,\ldots,q-1\}^n$
$r_{m+1} \leftarrow -1$
$a_{m+1} \leftarrow \sum_i r_i \cdot a_i$
return $(\mathrm{pk} = (a)_i, \mathrm{sk} = (r)_i)$

In the encryption the elements of the cipher text b_i are perputated with some small error of the Gaussian distribution e_i. It is made sure, that:

$$\sum_i r_i \cdot a_i \approx z \cdot \lceil \frac{q}{2} \rceil$$

precisely

Enc($\mathrm{pk} = (a)_i, \mathrm{m} = z$)
$\lceil$ $s \xleftarrow{\$} \{0,\ldots,q-1\}^n$
for $i = 1$ to $m+1$
$e_i \xleftarrow{\$} \mathrm{Gauss}(\sigma)$
for $i = 1$ to m
$b_i \leftarrow a_i \cdot s + e_i$
$b_{m+1} \leftarrow a_{m+1} \cdot s + e_{m+1} + z \cdot \lfloor \frac{q}{2} \rfloor$
return $\mathrm{c} = (b)_i$

The decryption is done by rounding

$$\sum_i r_i \cdot b_i$$

3.2 NTRU

(see [2] for more details)
Parameters

$$n, p, q, m, s \in \mathbb{N}; \alpha, \sigma, \eta \in \mathbb{R}$$

Here the "good" orthogonal basises will be the f and their shifts around by multiplying x. And the "bad" non-orthogonal basises will be the h and their shifts around.

KeyGen()
$\lceil$ $f' \xleftarrow{\$} \mathrm{Gauss}(\sigma)$
$f \leftarrow pf' + 1$
if f not invertible then start over
$g \xleftarrow{\$} \mathrm{Gauss}(\sigma)$
if g not invertible then start over
$h \leftarrow pg \cdot (f)^{-1}$
return $(\mathrm{pk} = h, \mathrm{sk} = f)$

Enc($\mathrm{pk} = .., \mathrm{m} = z$)
$\lceil$ $s, e \xleftarrow{\$} \mathrm{Gauss}(\sigma)$
$c \leftarrow h \cdot s + pe + z$
return $(\mathrm{c} = c)$

$$\begin{array}{l}\text{Dec}(\text{sk} = .., \text{c} = c)\\ \left[\begin{array}{l} z' \leftarrow f \cdot c \\ \textbf{return } (\text{m} = z') \end{array}\right.\end{array}$$

4 Implementation

To compare the runtimes of LPR, NTRU, RSA-OAEP and ElGamal in C++ based on the library NflLib [3] is used.

4.1 The Library NflLib

NflLib is C++ library that provides efficient algorithms for adding and multiplying polynomials and sampling polynomiahels from a discrete Gaussian distribution (the so-called error distribution). In [3] the underlying tricks are described and in [4] the algorithm for sampling the discrete Gaussian is described. NflLib enables the developer to use simply to use "+" and "*" to add and multiply polynomials. The multiplication of polynomials is done in $O(n\log(n))$ by the help of the so-called Fast Fourier Transform as opposed to $O(n^2)$ as the trivial approach would take.

4.2 Operations in the polynomial ring

Since NflLib doesn't support addition and multiplication modulo q and also doesn't compute modulo the polynomial $X^n + 1$. Hence the polynomials are represented as polynomial of degree $< 2n$ and some method called "keepInSpace" is needed that makes sure, that the polynomial has coefficients in the range from 0 to $q-1$ and has degree smaller then n. In other words every coefficient has to be modular divided by q and the whole polynomial has to be modular divided by $X^n + 1$. This both can efficiently be done in one go by the following algorithm:

$$\begin{array}{l}\text{keepInSpace}(\sum_i a_i x^i)\\ \left[\begin{array}{l} \textbf{for } i = 0 \text{ to } n-1 \\ a_i \leftarrow (a_i - a_{n+i}) \bmod q \\ \textbf{for } i = n \text{ to } 2n-1 \\ a_i \leftarrow 0 \end{array}\right.\end{array}$$

The algorithm reads each coefficient before before it writes it. So the computation done by the algorithm is:

$$a'_i = \begin{cases} (a_i - a_{n+i}) \bmod q, & i < n \\ 0, & i \geq n \end{cases}$$

It is easy to check, that this approach is sound. Namely

$$\begin{aligned} & a_i x^i + a_{n+i} x^{n+i} \\ =& (a_{n+i} + a_i - a_{n+i}) x^i + a_{n+i} x^{n+i} \\ =& (a_i - a_{n+i}) x^i + a_{n+i}(x^i + x^{n+i}) \\ =& (a_i - a_{n+i}) x^i + a_{n+i} x^i (x^n + 1) \\ \equiv& (a_i - a_{n+i}) x^i \qquad\qquad (x^n + 1) \end{aligned}$$

4.3 Structure of the Code

To structure the code in a better way, classes for the public key, the private key, the plain text, the cipher text and the key pair are being created.

Example

```
class LPR_Key_Pair {
  public:
  LPR_Public_Key pk;
  LPR_Private_Key sk;
};
```

To avoid copying polynomials for method calls, the technique "call by reference" is used where only a pointer to the polynomial is passed to the method and not the polynomial itself.

Example

```
int read_plaintext_from_poly(
    int q_quarter,
    LPR_Plaintext plaintext,
    poly_type* poly_pointer
) {
  ...
}
```

4.4 Encoding and Decoding the Plain Text

The encryption schemes LPR and NTRU deal with polynomials, but it rarely happens that a polynomial is what you want to encrypt. So some encoding of messages into polynomials is necessary. In the given implementation an ascii-string is encoded into the polynomial, but in real-life of cause an AES-key would be needed to be encrypted, since nobody wants to use public key encryption to encrypt longer messages, instead public key encryption is only used to encrypt the key for the symmetric encryption. To convert ascii-string to polynomials and back, the intermediate state of bytes (in C++ the datatype "unsigned char") is used. Ascii-strings can be quite directly converted to bytes and backwards, since strings are just arrays of characters. For that purpose method "bytesToString" and "stringToBytes" were created. The plain text object contains already the bytes. Bytes can be interpreted as bits and viewed as coefficients of a polynomial. This is done by methods "writePlaintextIntoPoly" and "readPlaintextFromPoly" which directly work with the plaintext object.

Example

```
for(int i = 0; i < 64; i++) {
  int value;
  value = 0;
  for(int j = 0; j < 8; j++) {
    if((*array_pointer)[8*i+j] > q_quarter) {
      value = (value << 1) + 1;
    } else {
      value = (value << 1) + 0;
    }
  }
  (*bytes_pointer)[i] = (unsigned char)value;
}
return 0;
```

4.5 Encoding and Decoding the Cipher Text

Also the cipher text consists of polynomials so in order to send it somewhere it has to be encoded into bits. The current implementation sheets a bit in this point. Since encryption and decryption are done on the same machine the encoding and decoding form binary is not neccessary. In real life the polynomials would have to be written and read in binary from some stream.

5 Comparing Security and Performance

In accordance to the parameters it is persumed in the performance of the algorithm, as it is described in the protocol.

5.1 Bit security and parameters for LPR

The algorithm involves several parameters namely n, α, q and m. Where

- n is the degree of the polynomial, so modulo $X^n + 1$, i.e. the dimension of the lattice.
- $q \in \mathbb{N}$ is the modulus.
- α is related to how big the error are that are added, α needs to be sufficiently big, so that encryption will be secure.
- m is related to how many ring-elements (i.e. polynomials) form the public key or private key or cipher text respectively

For simplicity other values are added:

- $\sigma = \alpha \cdot q$ the standard deviation of the discrete Gaussian
- $w = \frac{n}{8}$ the (maximal) byte-length of the plain text.
- $s = (m+1) \cdot \lceil \log(q) \rceil \cdot n$ bit size of the key.

Further as we are interested in bit-security-level we introduce the bit security parameter η.
As the currently best known attack takes $O(e^{(\sigma^2)})$ runtime [5] we define:

$$\eta = \log(e^{(\sigma^2)}) = \frac{1}{\ln(2)} \cdot \sigma^2$$

The the encryption scheme to be functional it is needed, that:

- n is a power of 2.
- q is prime (because it is using as the modulus)
- $q \equiv 1 (\mathrm{mod} n)$ (for cyclotonity i.e. the "wrapping around")
- $\sqrt{mn} \cdot \sigma \ll \frac{q}{2}$ (because otherwise the decryption will fail too often)

In order for the reduction proof to work, i.e. for "security", it is needed, that:

- $\sigma > \sqrt{n}$

Now two new parameters d and l can be introduced to make the conditions easier:

$$n = 2^d$$

$$q = l \cdot n + 1$$

So, all in all we have the 9 parameters

$$n, q, m, w, d, l, s \in \mathbb{N}; \alpha, \sigma, \eta \in \mathbb{R}$$

and the 7 conditions

$$\begin{aligned}
&\sigma = \alpha \cdot q \\
&w = \frac{n}{8} \\
&\eta = \log(e^{(\sigma^2)}) = \frac{1}{\ln(2)} \cdot (q \cdot \alpha)^2 \\
&\sqrt{mn} \cdot \sigma \ll \frac{q}{2} \\
&\sigma > \sqrt{n} \\
&n = 2^d \\
&n = l \cdot q + 1 \\
&s = (m+1) \cdot \lceil \log(q) \rceil \cdot n
\end{aligned}$$

According to that the following values were chosen/computed:

$$
\begin{aligned}
n &= 512 \\
q &= 617473 \\
m &= 1 \\
w &= 64 \\
d &= 10 \\
l &= 1206 \\
s &= 20480 \\
\alpha &= 0.00001 \\
\sigma &= 0.31 \\
\eta &= 55
\end{aligned}
$$

5.2 Bit security and parameters for NTRU

The difference between LPR and NTRU is that NTRU has an additional parameter $p \in \mathbb{N}$ and that the key consists of only one ring element so the key size here will be 6660.

$$p = 7$$

with respect the the new bit size:

$$s = 6600$$

as it reaches that far, over the other parameters as well.

5.3 Runtime Comparison

Different cryptosystems with different parameters can now be compared. When achieving similar efficiency for LPR as with state-of-the-art crypto like RSA and ElGamal, LPR still falls behind in the bit security. For NTRU however there exist already efficient implementations.

cryptosytem	bit security	key size	key generation (ms)	encryption (ms)	decryption (ms)
RSA	80	1024	180	12	11
ElGamal	128	512	210	1	1
LPR	55	20480	76	125	55

References

[1] Lyubashevsky, Vadim and Peikert, Chris and Regev, Oded, "On Ideal Lattices and Learning with Errors over Rings", Advances in Cryptology – EUROCRYPT 2010: 29th Annual International Conference on the Theory and Applications of Cryptographic Techniques, French Riviera, May 30 – June 3, 2010. Proceedings, 2010, Springer Berlin Heidelberg, Berlin, Heidelberg 978-3-642-13190-5

[2] Stehle, Damien and Steinfeld, Ron, "Making NTRU as Secure as Worst-Case Problems over Ideal Lattices", Advances in Cryptology – EUROCRYPT 2011: 30th Annual International Conference on the Theory and Applications of Cryptographic Techniques, Tallinn, Estonia, May 15-19, 2011. Proceedings, 2011, Springer Berlin Heidelberg, Berlin, Heidelberg, 978-3-642-20465-4, http://dx.doi.org/10.1007/978-3-642-20465-4

[3] Aguilar-Melchor, Carlos and Barrier, Joris and Guelton, Serge and Guinet, Adrien and Killijian, Marc-Olivier and Lepoint, Tancrede, "NFLlib: NTT-Based Fast Lattice Library", Topics in Cryptology - CT-RSA 2016: The Cryptographers' Track at the RSA Conference 2016, San Francisco, CA, USA, February 29 - March 4, 2016, Proceedings, 2016, Springer International Publishing, Cham, 978-3-319-29485-8, http://dx.doi.org/10.1007/978-3-319-29485-8

[4] Nagarjun C. DwarakanathSteven D. Galbraith "Sampling from discrete Gaussians for lattice-based cryptography on a constrained device" Dwarakanath, N.C. and Galbraith, S.D. AAECC (2014) 25: 159. doi:10.1007/s00200-014-0218-3

[5] Lindner, Richard and Peikert, Chris, "Better Key Sizes (and Attacks) for LWE-Based Encryption", Topics in Cryptology – CT-RSA 2011: The Cryptographers' Track at the RSA Conference 2011, San Francisco, CA, USA, February 14-18, 2011. Proceedings", Springer Berlin Heidelberg, isbn 978-3-642-19074-2, http://dx.doi.org/10.1007/978-3-642-19074-2-21