Extended Abstract   
Post-Quantum Cryptographic Primitive (PCP) Toolbox:  
 Lake Bu

Abstract:

The file “*11-20 Post-Quantum Crypto Primitives - Extended Abstract.docx*” introduces the general information of post-quantum primitives. In this report, we focus on the proposal of a Post-quantum Cryptographic Primitive (PCP) toolbox. The toolbox consists of:

* A Public-Key Cryptosystem (PKC)
* A Key Encapsulation Mechanism (KEM)
* An Oblivious Transfer (OT) mechanism
* A Zero-Knowledge Proof (ZKP) mechanism

It enables researchers to construct quantum attack-resistant modules/systems, by using the building blocks in this toolbox. The following sections briefly introduce the algorithms of the 4 primitives above.

The detailed implementation guides (on both software and hardware), are in the “*PCP Implementation*” folder.

1. **Public-key Cryptosystems (PKC) Based on Ring-LWE**

First, we define as a ring where . For computation simplicity, *n* is taken as a power of 2 and *q* being a prime such that . Also an error distribution is taken to be a discrete Gaussian noise of width *q*, and relative error rate .

**Algorithm I.1 Ring-LWE-based Public-key Cryptosystem (KPC) for Encryption [1]**

*Key Generation:*

1. Alice picks where as the private key, and as the random noise. Alice also picks as a portion of the public key. Alice computes:

(1)

where is the vector product operator.

1. Alice then sends {a, b} to Bob as the public key, and keeps {s} as the private key.

*Message encryption:*

1. Bob picks randomly, and converts his message to , where m should be binary only (but still treated as a regular digit within ).
2. Bob computes:

(2)

1. Bob sends {ca, cb} to Alice.

*Message decryption:*

1. Alice computes:

Denote as *w*, (3)

where stands for taking the nearest integer.

**Remark I.1**

It is notable that in order to perform efficient multiplication over the ring (modulo addition is trivial), a number theoretic transform (NTT) primitive needs to be constructed.

1. **Key Encapsulation Mechanism (KEM)**

First, is defined exactly the same as in Section I.

**Algorithm II.1 Ring-LWE-based Key Encapsulation Mechanism (KEM) – (Simplified) [2]**

*Preparation:*

1. Alice and Bob publicly shares

*First round:*

1. Alice picks and computes

(4)

1. Alice then sends {} to Bob.

*Second round:*

1. Bob picks and computes

(5)

1. Bob then sends {} to Alice.

*Third round:*

1. Alice and Bob compute (6) and (7) respectively:

(6)

(7)

where stands for taking the nearest integer, and they both get the same private key.

1. **Oblivious Transfer (OT)**

We propose a 1-out-of-n OT scheme using the KPC from Algorithm I.1, which is also over the ring Rp.

**Algorithm III.1 Ring-LWE-based Oblivious Transfer (OT)**

*Preparation:*

1. Alice generates the public key {a, b} and sends it to Bob. Alice keeps {s} as the private key.
2. Alice has n messages and generates n random vectors .

*First round:*

1. Alice sends to Bob;
2. Bob selects the *c*th random vector in order to acquire . Bob generates and then computes:

(8)

where Enc() is the encryption function from Algorithm I.1.

1. Bob sends *v* to Alice.

*Second round:*

1. For all , Alice computes

(9)

where Dec() is the decryption function from Algorithm I.1.

1. For all , Alice computes

(10)

And sends to Bob.

*Third round:*

1. Bob computes:

(11)

And then Bob acquires in an oblivious manner for both Alice and Bob.

1. **Zero-knowledge Proof (ZKP)**

We propose a zero-knowledge proof scheme, which is also over the ring Rp.

**Algorithm IV.1 Ring-LWE-based Zero-knowledge Proof (ZKP)**

*First round:*

1. Alice has a secret, and wants to proof to Bob she has it without revealing s.
2. Alice samples , and denotes t = . Alice generates {a, b} such that:

(12)

1. Alice sends {a, b} to Bob. (12) creates a Trap-door so that Bob does not know s.

*Second round:*

1. Alice picks a binary vector m, and samples and generates {a, c} such that:

(13)

1. Alice sends {c, m} to Bob. (13) also creates a Trap-door so that Bob does not know r.

*Third round:*

1. Bob samples and send it to Alice;
2. Alice computes x and sends it to Bob, such that:

(14)

Bob does not know s since he knows not r.

*Final round:*

1. Bob computes:

(15)

(16)

where stands for taking the nearest integer.

Since both and are “small”, if (16) achieves equality, then Alice successfully accomplishes her ZKP of s.

**Reference**

1. Vadim Lyubashevsky, Chris Peikert, and Oded Regev. On ideal lattices and learning with errors over rings. In Annual International Conference on the Theory and Applications of Cryptographic Techniques, pages 1{23. Springer, 2010.
2. Ding, Jintai, Xiang Xie, and Xiaodong Lin. "A Simple Provably Secure Key Exchange Scheme Based on the Learning with Errors Problem." IACR Cryptology EPrint Archive 2012 (2012): 688.

**Appendix**

**Code-based Public-key Cryptosystem**

Code-based cryptosystem can also be used to construct PKC. Below is a representative algorithm for public-key encryption.

**Algorithm 1**

Key Generation:

* 1. Alice selects a binary (n,k)-linear code C capable of correcting t errors. This code must possess an efficient decoding algorithm and generates a k×n generator matrix G for.
  2. Alice selects a random k×k binary non-singular matrix S.
  3. Alice selects a random n×n permutation matrix P.
  4. Alice computes the k×n matrix G’ = SGP.
  5. Alice's public key is (G’, t); her private key is (S,G,P).

Message encryption:

1. Suppose Bob wishes to send a message m to Alice whose public key is (G’, t).
2. Bob encodes the message m as a binary string of length k (the plaintext).
3. Bob computes the vector c’ = Mg’.
4. Bob generates a random n-bit vector z containing exactly t ones (a vector of length n and weight t, just as an error vector).
5. Bob computes the ciphertext as c = c’ + z.

Message decryption:

1. Upon receipt of c, Alice performs the following steps to decrypt the message.
2. Alice computes the inverse of P (i.e. P-1).
3. Alice computes c’’ = cP-1.
4. Alice uses the decoding algorithm for the code C to decode c’’ to m’’.
5. Alice computes m = m’’S-1.

For a quantum-secure implementation, binary Goppa codes with n = 6960, k = 5413, t = 119 were proposed, leading to a key size of approximately one megabyte (1 MB).

A question is, can we use other binary codes to replace Goppa codes to achieve a smaller key size? Some of the candidates are: OLSC and GTB codes, whose encoding matrices are binary, but codeword symbols can be non-binary. Thus a small matrix can encode a large number of bits.