National College of Ireland

Configuration Manual

Academic Internship
MSc Cybersecurity

Success Jimoh
Student ID: X20139471

School of Computing
National College of Ireland

Supervisor: Vikas Sahni

National College of Ireland

MSc Project Submission Sheet

School of Computing

Student Name: ……. ……Success Jimoh……

Student ID: ……20139471……..……

Programme: ……MSc. Cybersecurity…… **Year:** ..2021……..

Module: ……Academic Internship…….……

Lecturer: ……Vikas Sahni…….……

Submission Due Date: ……16/12/2021…….……

Project Title: QSTRU: A new variation of the NTRU public key cryptosystem……

Word Count: …… **Page Count:** ……..…….……

I hereby certify that the information contained in this (my submission) is information pertaining to research I conducted for this project. All information other than my own contribution will be fully referenced and listed in the relevant bibliography section at the rear of the project.
ALL internet material must be referenced in the bibliography section. Students are required to use the Referencing Standard specified in the report template. To use other author's written or electronic work is illegal (plagiarism) and may result in disciplinary action.

Signature: ……Success Jimoh……

Date: ……16/12/2021……

PLEASE READ THE FOLLOWING INSTRUCTIONS AND CHECKLIST

Attach a completed copy of this sheet to each project (including multiple copies)	□
Attach a Moodle submission receipt of the online project submission, to each project (including multiple copies).	□
You must ensure that you retain a HARD COPY of the project, both for your own reference and in case a project is lost or mislaid. It is not sufficient to keep a copy on computer.	□

Assignments that are submitted to the Programme Coordinator Office must be placed into the assignment box located outside the office.

Office Use Only	
Signature:	
Date:	
Penalty Applied (if applicable):	

Configuration Manual

Success Jimoh
Student ID: 20139471

1. Introduction

This configuration manual is a guide to show how the QSTRU algorithm could be reproduced along with his environments. This includes the tools and setup used for implementing and evaluating the algorithm.

2. Hardware specifications

The configuration of the system used to carry out the research is:

- Device: Apple MacBook Pro (13-inch, M1, 2020)
- Operating system: MacOS Big Sur
- Memory : 8GB
- Hard Drive: 500GB

3. Software Specifications

There were various softwares used in implementing this algorithm. They include :

- Sublime Text
- Command shell
- Python

4. Software Installation

This is how the various tools and software used for the algorithm implementation could be downloaded and installed.

- Download and Install python (Download Python, 2021)
- Download and Install sublime text (Download - Sublime Text, 2021)

5. Implementation

While this application is implemented with python, there were various python libraries used for this and they are :

- Sympy
- Numpy
- Math
- Counter

The following steps were used for the implementation:

1. Importing of the libraries

```
from docopt import docopt
from qstru.Qstrucipher import QstruCipher
from qstru.mathutils import random_poly
from sympy.abc import x
from sympy import ZZ, Poly
from padding.padding import *
import numpy as np
import sys
import logging
import math
import time
```

2. key Generation

```
def generate(N, p, q, priv_key_file, pub_key_file):
    qstru = QstruCipher(N, p, q)
    qstru.generate_random_keys()
    h = np.array(qstru.h_poly.all_coeffs()[::-1])
    f, f_p = qstru.f_poly.all_coeffs()[::-1], qstru.f_p_poly.all_coeffs()[::-1]
    np.savez_compressed(priv_key_file, N=N, p=p, q=q, f=f, f_p=f_p)
    log.info("Private key saved to {} file".format(priv_key_file))
    np.savez_compressed(pub_key_file, N=N, p=p, q=q, h=h)
    log.info("Public key saved to {} file".format(pub_key_file))
```

3. For Encryption of data

```
def encrypt(pub_key_file, input_arr, bin_output=False, block=False):
    start_time = time.time()
    pub_key = np.load(pub_key_file, allow_pickle=True)
    qstru = QstruCipher(int(pub_key['N']), int(pub_key['p']), int(pub_key['q']))
    qstru.h_poly = Poly(pub_key['h'].astype(np.int)[::-1], x).set_domain(ZZ)
    if not block:
        if qstru.N < len(input_arr):
            raise Exception("Input is too large for current N")
        output = (qstru.encrypt(Poly(input_arr[::-1], x).set_domain(ZZ),
                                random_poly(qstru.N, int(math.sqrt(qstru.q)))).all_coeffs()[::-1])
    else:
        input_arr = padding_encode(input_arr, qstru.N)
        input_arr = input_arr.reshape((-1, qstru.N))
        output = np.array([])
        block_count = input_arr.shape[0]
        for i, b in enumerate(input_arr, start=1):
            log.info("Processing block {} out of {}".format(i, block_count))
            next_output = (qstru.encrypt(Poly(b[::-1], x).set_domain(ZZ),
                                         random_poly(qstru.N, int(math.sqrt(qstru.q)))).all_coeffs()[::-1])
            if len(next_output) < qstru.N:
                next_output = np.pad(next_output, (0, qstru.N - len(next_output)), 'constant')
            output = np.concatenate((output, next_output))

    if bin_output:
        k = int(math.log2(qstru.q))
        output = [[0 if c == '0' else 1 for c in np.binary_repr(n, width=k)] for n in output]
    return np.array(output).flatten()
```

4. Decryption of data

```
def decrypt(priv_key_file, input_arr, bin_input=False, block=False):
    priv_key = np.load(priv_key_file, allow_pickle=True)
    qstru = QstruCipher(int(priv_key['N']), int(priv_key['p']), int(priv_key['q']))
    qstru.f_poly = Poly(priv_key['f'].astype(np.int)[::-1], x).set_domain(ZZ)
    qstru.f_p_poly = Poly(priv_key['f_p'].astype(np.int)[::-1], x).set_domain(ZZ)

    if bin_input:
        k = int(math.log2(qstru.q))
        pad = k - len(input_arr) % k
        if pad == k:
            pad = 0
        input_arr = np.array([int("".join(n.astype(str)), 2) for n in
                              np.pad(np.array(input_arr), (0, pad), 'constant').reshape((-1, k))])
    if not block:
        if qstru.N < len(input_arr):
            raise Exception("Input is too large for current N")
        log.info("POLYNOMIAL DEGREE: {}".format(max(0, len(input_arr) - 1)))
        return qstru.decrypt(Poly(input_arr[::-1], x).set_domain(ZZ)).all_coeffs()[::-1]

    input_arr = input_arr.reshape((-1, qstru.N))
    output = np.array([])
    block_count = input_arr.shape[0]
    for i, b in enumerate(input_arr, start=1):
        log.info("Processing block {} out of {}".format(i, block_count))
        next_output = qstru.decrypt(Poly(b[::-1], x).set_domain(ZZ)).all_coeffs()[::-1]
        if len(next_output) < qstru.N:
            next_output = np.pad(next_output, (0, qstru.N - len(next_output)), 'constant')
        output = np.concatenate((output, next_output))
    return padding_decode(output, qstru.N)
```

5. Implementation of the QSTRU class for key generation, encryption and decryption algorithm
Generating f and g polynomials

```
def generate_random_keys(self):
    g_poly = random_poly(self.NewN, int(math.sqrt(self.q)))
    log.info("g: {}".format(g_poly))
    log.info("g coeffs: {}".format(Counter(g_poly.coeffs())))


    tries = 10
    while tries > 0 and (self.h_poly is None):
        f_poly = random_poly(self.NewN, self.NewN//2, neg_ones_diff=-1)
        log.info("f: {}".format(f_poly))
        log.info("f coeffs: {}".format(Counter(f_poly.coeffs())))
        try:
            self.generate_public_key(f_poly, g_poly)
        except NotInvertible as ex:
            log.info("Failed to invert f (tries left: {})".format(tries))
            log.debug(ex)
            tries -= 1
    if self.h_poly is None:
        raise Exception("Couldn't generate invertible f")
```

Generating the public keys

```
def generate_public_key(self, f_poly, g_poly):

    self.f_poly = f_poly
    self.g_poly = g_poly
    self.f_p_poly = invert_poly(self.f_poly, self.R_poly, self.p)
    self.f_q_poly = invert_poly(self.f_poly, self.R_poly, self.q)
    print("f_p_poly",self.f_p_poly)
    print("f_q_poly",self.f_q_poly)
    log.debug("f*f_p mod (x^n - 1): {}".format(((self.f_poly * self.f_p_poly) % self.R_poly).trunc(self.p)))
    log.debug("f*f_q mod (x^n - 1): {}".format(((self.f_poly * self.f_q_poly) % self.R_poly).trunc(self.q)))
    p_f_q_poly = (self.p * self.f_q_poly).trunc(self.q)
    log.debug("p_f_q: {}".format(p_f_q_poly))
    h_before_mod = (p_f_q_poly * self.g_poly).trunc(self.q)
    log.debug("h_before_mod: {}".format(h_before_mod))
    self.h_poly = (h_before_mod % self.R_poly).trunc(self.q)
    log.info("h: {}".format(self.h_poly))
```

Encryption algorithm

```
def encrypt(self, msg_poly, rand_poly):
    log.info("r: {}".format(rand_poly))
    log.info("r coeffs: {}".format(Counter(rand_poly.coeffs())))
    log.info("msg: {}".format(msg_poly))
    log.info("h: {}".format(self.h_poly))
    return (((rand_poly * self.h_poly).trunc(self.q) + msg_poly) % self.R_poly).trunc(self.q)
```

Decryption algorithm

```
def decrypt(self, msg_poly):
    log.info("f: {}".format(self.f_poly))
    log.info("f_p: {}".format(self.f_p_poly))
    a_poly = ((self.f_poly * msg_poly) % self.R_poly).trunc(self.q)
    log.info("a: {}".format(a_poly))
    b_poly = a_poly.trunc(self.p)
    log.info("b: {}".format(b_poly))
    return ((self.f_p_poly * b_poly) % self.R_poly).trunc(self.p)
```

Trigintaduonions Multiplications

```
def qmult(x, y):
    return np.array([
        x[0]*y[0] - x[1]*y[1] - x[2]*y[2] - x[3]*y[3],
        x[0]*y[1] + x[1]*y[0] + x[2]*y[3] - x[3]*y[2],
        x[0]*y[2] - x[1]*y[3] + x[2]*y[0] + x[3]*y[1],
        x[0]*y[3] + x[1]*y[2] - x[2]*y[1] + x[3]*y[0]
    ])

    # quaternion conjugate
def qstar(x):
    return x*np.array([1, -1, -1, -1])

    # octonion multiplication
def omult(x, y):
        # Split octonions into pairs of quaternions
    a, b = x[:4], x[4:]
    c, d = y[:4], y[4:]

    z = np.zeros(8)
    z[:4] = qmult(a, c) - qmult(d, qstar(b))
    z[4:] = qmult(qstar(a), d) + qmult(c, b)
    return z

def ostar(x):
        mask = -np.ones(8)
        mask[0] = 1
        return x*mask

    # sedenion multiplicaton
def smult(x, y):
    # Split sedenions into pairs of octonions
    a, b = x[:8], x[8:]
    c, d = y[:8], y[8:]
    z = np.zeros(16)
    z[:8] = omult(a, c) - omult(d, ostar(b))
    z[8:] = omult(ostar(a), d) + omult(c, b)
    return z

def sstar(x):
    mask = -np.ones(16)
    mask[0] = 1
    return x*mask
```

```
def tmult(x,y):
    a, b = x[:16], x[16:]
    c, d = y[:16], y[16:]
    z = np.zeros(32)
    z[:16] = smult(a, c) - smult(d, sstar(b))
    z[16:] = smult(sstar(a), d) + smult(c, b)
```

Inverting polynomials

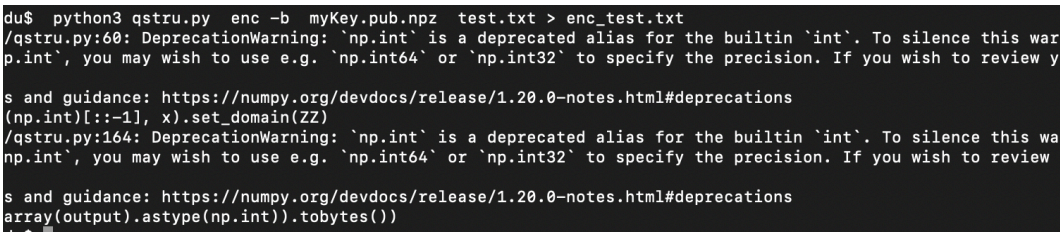
```
def invert_poly(f_poly, R_poly, p):
    inv_poly = None
    if is_prime(p):
        log.debug("Inverting as p={} is prime".format(p))
        inv_poly = invert(f_poly, R_poly, domain=GF(p))
    elif is_2_power(p):
        log.debug("Inverting as p={} is 2 power".format(p))
        inv_poly = invert(f_poly, R_poly, domain=GF(2))
        e = int(math.log(p, 2))
        for i in range(1, e):
            log.debug("Inversion({}): {}".format(i, inv_poly))
            inv_poly = ((2 * inv_poly - f_poly * inv_poly ** 2) % R_poly).trunc(p)
    else:
        raise Exception("Cannot invert polynomial in Z_{}".format(p))
    log.debug("Inversion: {}".format(inv_poly))
    return inv_poly
```

6. Software Execution and Result

To run the application the following command lines were ran :

Key Generation

Data Encryption



Data Decryption

```
du$  python3 qstru.py  dec -b  myKey.priv.npz  enc_test.txt > dec_test.txt
/qstru.py:87: DeprecationWarning: `np.int` is a deprecated alias for the bu
.int`, you may wish to use e.g. `np.int64` or `np.int32` to specify the pr

s and guidance: https://numpy.org/devdocs/release/1.20.0-notes.html#deprecat
e(np.int)[::-1], x).set_domain(ZZ)
```

To evaluate the encryption time of the algorithm

```
def encrypt(pub_key_file, input_arr, bin_output=False, block=False):
    start_time = time.time()
    print("--- %s seconds ---" % (time.time() - start_time))
    return np.array(output).flatten()
```

References

Download Python. (2021). Python.Org. Retrieved 15 December 2021, from https://www.python.org/downloads/

Download—Sublime Text. (2021). Retrieved 15 December 2021, from https://www.sublimetext.com/3