Algorithm Analysis and Design

Name: Jiang Bowen **Due Date:** May 30, 2022.
Student Number: 202117052 **Assignment:** Number 4

Problem In this assignment, you need to implement Qin's Algorithm (from Lecture 2) using Java or C/C++ (you need to use the package GMP for C/C++).

Solution RSA public key e and private key d is

$$ed \equiv 1 (mod\ \phi(n))$$

There exists a positive integer k, the equation

$$ed - 1 = k \cdot \phi(n)$$

is satisfied. By dividing both sides by $d\phi(n)$, it turns

$$\frac{e}{\phi(n)} - \frac{k}{d} = \frac{1}{d\phi(n)}$$

If e is too big, approximately equal to n and $n \approx \phi(n)$, $e \approx \phi(n)$. Applying Wiener's results to the RSA Cryptosystem, we are certain to find one of the convergents of the continued fraction expansion of e/n, which is exactly equal to k/d.

Now the first step is that how to modify Qin's algorithm to calculate the convergents of continued fraction. To achieve this, a simple method is to add a new column in state matrix:

$$\begin{bmatrix} x_{11} & x_{12} & x_{13} \\ x_{21} & x_{22} & x_{23} \end{bmatrix}$$

where the initial state is $\begin{bmatrix} 1 & e & 1 \\ 0 & n & n/e \end{bmatrix}$. The entries x_{13} and x_{23} update as the red part of *modify_qin_fun()* (Algorithm 1, Appendix A). The numerator of continued fractions is x_{21} and the denominator is $x_{13}\ mod\ x_{21}$.

The second step is to find the correct d from the entry x_{21} of Qin's state matrix. We test each denominator k'. If $k' | e \cdot x_{21} - 1$, it means that possible $\phi(n)$ can be calculated. Once n and $\phi(n)$ are known, we can factor n by solving the quadratic equation $x^2 - (n - \phi(n) + 1)x + n$ which two roots are p and q. The result is possibly more than one, so we need to try each one until $pq = n$ is satisfied. Blue part(Algorithm 1, Appendix A) and function *get_pq()*(Algorithm 2, Appendix A) implements this step.

The algorithm requires $O((\log_2 n)^4)$ steps so it is efficient. Finally, the procedure output d, p, q if attack successfully. The results are following and the full code is in Appendix B.

```
p = 14867132994240195614612448893358942179275318019184214169742723191230
48047842301139582064854327076454620356056462712742806057865452106752623867
83060000706225490710033813203994580538080520612702355239318915736363373202
37323389140170393827657953331593080152144741217135556393500390783662767
17999835780729080942467
```

```
q = 160577100375614564018341557641357114288004878328768285449283860863028
8567283929477948775814275770975955370361826425824326324412656409110830668
3888187675774743571970459026047807501138224282313456944860989266461891195
8144028778491530154987321607508754456707395614577095775911210216108867790
257353074868685910047
```

Appendix A

Algorithm 1 modify_qin_fun(e, n)

Input: RSA public key e, n with $1 < e < n$, $gcd(e, n) = 1$

Output: return *Attack Success* if find RSA private key d, otherwise return *Attack Fail*

$\begin{bmatrix} x_{11} & x_{12} & x_{13} \\ x_{21} & x_{22} & x_{23} \end{bmatrix} \leftarrow \begin{bmatrix} 1 & e & 1 \\ 0 & n & n/e \end{bmatrix}$

while $x_{12} > 1$ **do**

    if $x_{22} > x_{12}$ **then**

        $q \leftarrow \lfloor \frac{x_{22}-1}{x_{12}} \rfloor$

        $r \leftarrow x_{22} - qx_{12}$

        $x_{21} \leftarrow qx_{11} + x_{21}$

        $x_{13} \leftarrow qx_{23} + x_{13}$

        $x_{22} \leftarrow r$

    end if

    if $x_{12} > x_{22}$ **then**

        $q \leftarrow \lfloor \frac{x_{12}-1}{x_{22}} \rfloor$

        $r \leftarrow x_{12} - qx_{22}$

        $x_{11} \leftarrow qx_{21} + x_{11}$

        $x_{23} \leftarrow qx_{13} + x_{23}$

        $x_{12} \leftarrow r$

    end if

    $k' \leftarrow x_{13} \ mod \ x_{21}$

    $d' \leftarrow x_{21}$

    if k' is not zero and $k' | (d'e - 1)$ **then**

        $\phi'(n) \leftarrow \frac{(d'e-1)}{k'}$

        $(p, q) \leftarrow get\_pq(\phi'(n), n)$

        if p is not *None* and q is not *None* **then**

            $d \leftarrow d'$

            return Attack Success

        else

            $d \leftarrow None$

        end if

    end if

end while

return Attack Fail

Algorithm 2 get_pq($\phi'(n), n$)

Input: a possible $\phi'(n)$ and RSA public key n
Output: decompose n into p and q. Return (p, q) if success, return $(None, None)$

$b \leftarrow n - \phi'(n) + 1$
$\Delta \leftarrow b^2 - 4n$
if $\Delta \geq 0$ **then**
  $p \leftarrow \frac{-b+\sqrt{\Delta}}{2}$
  $q \leftarrow \frac{-b-\sqrt{\Delta}}{2}$
  if $pq == n$ **then**
    return (p, q)
  end if
end if
return $(None, None)$

Appendix B

```c
#include <stdio.h>
#include <stdlib.h>
#include <gmp.h>
#define ATTACK_SUCCESS 0
#define ATTACK_FAIL 1

int get_pq(mpz_t phi, mpz_t n){
    mpz_t b, par, t;
    mpz_t p, q;
    mpz_inits(b, par, t, NULL);
    mpz_inits(p, q, NULL);

    // b = n - phi + 1
    mpz_sub(b, n, phi);
    mpz_add_ui(b, b, 1);

    // par = b * b - 4 * n
    mpz_mul(par, b, b);
    mpz_mul_ui(t, n, 4);
    mpz_sub(par, par, t);

    if(mpz_cmp_ui(par, 0) >= 0){
        mpz_sqrt(par, par);
        // p = (-b + par) / 2
        mpz_sub(t, par, b);
        mpz_fdiv_q_ui(p, t, 2);
```

```c
        // q = (-b - par) / 2
        mpz_add(t, par, b);
        mpz_fdiv_q_ui(q, t, 2);
        mpz_mul_si(q, q, -1);

        mpz_abs(p, p);
        mpz_abs(q, q);

        mpz_mul(t, p, q);
        if(mpz_cmp(t, n) == 0) {
            gmp_printf("p = %Zd\n", p);
            gmp_printf("q = %Zd\n", q);
            return ATTACK_SUCCESS;
        }
    }
    mpz_clears(b, par, t, NULL);
    mpz_clears(p, q, NULL);
    return ATTACK_FAIL;
}

int qin_fun(mpz_t a, mpz_t m){
    mpz_t x11, x12, x21, x22;
    mpz_t x13, x23;

    mpz_inits(x11, x12, x21, x22, x13, x23, NULL);

    mpz_set_str(x11, "1", 10);
    mpz_set(x12, a);
    mpz_set_str(x21, "0", 10);
    mpz_set(x22, m);

    mpz_set_str(x13, "1", 10);
    mpz_fdiv_q(x23, m, a);

    mpz_t t, q, r, k, phi;
    mpz_inits(t, q, r, k, phi, NULL);

    while(mpz_cmp_ui(x12, 1)) {
        if (mpz_cmp(x22, x12)) {
            // q = (x22 - 1) // x12
            // r = x22 - q * x12
            mpz_fdiv_qr(q, r, x22, x12);
            // x21 = q * x11 + x21
            mpz_mul(t, q, x11);
```

```
            mpz_add(x21, t, x21);
            // c13 = q * x23 + x13
            mpz_mul(t, q, x23);
            mpz_add(x13, t, x13);
            // x22 = r
            mpz_set(x22, r);
        }

        if (mpz_cmp(x12, x22)) {
            mpz_fdiv_qr(q, r, x12, x22);
            // x21 = q * x21 + x11
            mpz_mul(t, q, x21);
            mpz_add(x11, t, x11);
            // x23 = q * x13 + x23
            mpz_mul(t, q, x13);
            mpz_add(x23, t, x23);
            // x12 = r
            mpz_set(x12, r);
        }

        mpz_mul(t, a, x21);
        mpz_sub_ui(t, t, 1);
        mpz_mod(k, x13, x21);

        if (mpz_cmp_ui(k, 0) == 0) {
            continue;
        }

        mpz_fdiv_qr(phi, t, t, k);
        if (mpz_cmp_ui(t, 0) == 0) {
            int ret = get_pq(phi, m);
            if (!ret) {
                gmp_printf("d = %Zd\n", x21);
                printf("attack success");
                return ATTACK_SUCCESS;
            }
        }
    }
    mpz_clears(x11, x12, x21, x22, x13, x23, NULL);
    mpz_clears(t, q, r, k, phi, NULL);
    return ATTACK_FAIL;
}

int main(int argc, char *argv[]) {
```

```
    mpz_t e, n;
    int ret = mpz_init_set_str(e,
    "151266542979647520130982367371427923433288858210438507175912122194161
    21876442886718732758175450185439250097909940465459448006358513001188
    31569595146784640326033719886971483790936192832798927285536243207113
    76031237668277173469217518859588677200153177482709215971610834903832
    78302493913553369743780995203398626939715733755155009681289940104874
    44413932542747401660887988848271779011553903823687558388970389112629
    36230099448221309246152536529123195434003038033959109873096548049896
    65931463997217772013338972458030372517101804468582533624114654185261
    016834363449414265885477703830170220100277420183737256953449921432442769",
    10);
    if (ret != 0){
        perror("mpz for e error");
        exit(EXIT_FAILURE);
    }

    ret = mpz_init_set_str(n,
    "238732110711371899306141663102673394874545380684224436579690412624604
    44429964816431426710183695755476413937115540614428835263204196063562
    20413644033208697559553412748961679463495811726292079735776038312121
    65275783685410407200978805112812657511485149705858525443570258925104
    19412343969413669659546355293839984987174937768419702376847760976489
    36552158704404855445241013030917418788178181734301491577591544256779
    63071201180500119889705228862793157243106529602603543177324082379498
    85912477148109782713365688772833978483650246761301470938166706596058
    470226807335420535020271139433490249384246034110447894017222870344265949",
    10);
    if (ret != 0){
        perror("mpz for n error");
        exit(EXIT_FAILURE);
    }

    ret = qin_fun(e, n);
    if (ret != 0){
        perror("attack fail");
        exit(EXIT_FAILURE);
    }

    mpz_clears(e, n, NULL);
    return EXIT_SUCCESS;
}
```