UNIVERSITEIT VAN PRETORIA
UNIVERSITY OF PRETORIA
YUNIBESITHI YA PRETORIA
Faculty of Engineering, Built Environment and
Information Technology

EHN 410

Network Security

Practical 3 Code

Group: 12

Name and Surname	Student Number	% Contribution
J.C.J. du Plessis	17010986	33.33
I. Eloff	17018082	33.33
R.S. Walters	17017051	33.33

By submitting this assignment we confirm that we have read and are aware of the University of Pretoria's policy on academic dishonesty and plagiarism and we declare that the work submitted in this assignment is our own as delimited by the mentioned policies. We explicitly declare that no parts of this assignment have been copied from current or previous students' work or any other sources (including the internet), whether copyrighted or not. We understand that we will be subjected to disciplinary actions should it be found that the work we submit here does not comply with the said policies.

June 14, 2020

Contents

1 Practical 3 Common Code

1.1 prac3.h

```
#ifndef EHN_PRAC3_H
#define EHN_PRAC3_H

// Common includes
#include <stdio.h>
#include <stdlib.h>
#include <unistd.h>
#include <string.h>
#include <stdbool.h>
#include <sys/types.h>
#include <sys/stat.h>
#include <sys/timeb.h>
#include <ctype.h>
#include <gmp.h>

// Common defines
#define U8 unsigned char


// RC4 functions
/// The struct used to store the current state of the RC4 encryption algorithm.
struct rc4ctx_t
{
    int S[256];
    int i, j;
};


/**
 * Sets up the RC4 algorithm variables using the key and performs the initial permutation.
 * @param rc4ctx A pointer to the struct that will hold all the state variables for the RC4 algorithm.
 * @param key An array of 8-bit values that represent the key.
 * @param keylen The length of the key in bytes.
 */
void rc4_init(struct rc4ctx_t *rc4ctx, U8 key[], int keylen);


/**
 * Returns the next byte of the stream cipher that can be used to encrypt a value and updates the state
↪  variables.
 * @param rc4ctx A pointer to the struct that will hold all the state variables for the RC4 algorithm.
 * @return The byte generated by the RC4 algorithm.
 */
U8 rc4_getbyte(struct rc4ctx_t *rc4ctx);


// RSA functions
/// The RSA struct to store all the key values.
struct rsactx_t
{
    mpz_t e, d, n;
    mpz_t p, q;
};


/**
 * Initialises the RSA context.
 * @param rsactx The RSA context struct.
 */
void rsa_init(struct rsactx_t *rsactx);


/**
 * Frees the memory associated with the RSA context.
 * @param rsactx The RSA context struct.
 */
void rsa_clean(struct rsactx_t *rsactx);


// Common functions
// Simply swap two values by reference
void swap(int *a, int *b);


// Convert hex to int, done because the system hex converter is unreliable
int hex_convert(char hex_string[], int length);


// Print a c-string up to a certain length in hex
void print_hex_string(U8 hex_string[], int message_len);


#endif // EHN_PRAC3_H
```

1.2 prac3.c

```
#include "prac3.h"


// RC4 functions
// Set up the RC4 cipher as done in "Network Security Essentials", William stallings, page 48
void rc4_init(struct rc4ctx_t *rc4ctx, U8 key[], int keylen)
{
    int i;
    int T[256];

    for (i = 0; i < 256; i++) // Initialise values
    {
        rc4ctx->S[i] = i;
        T[i] = key[i % keylen];
    }

    int j = 0;
    for (i = 0; i < 256; i++) // Do the initial permutation of S
    {
        j = (j + rc4ctx->S[i] + T[i]) % 256;
        swap(&(rc4ctx->S[i]), &(rc4ctx->S[j]));
    }

    rc4ctx->i = 0; // Set up permutation variables in the struct
    rc4ctx->j = 0;
}


// Generate a byte using the RC4 cipher as done in "Network Security Essentials", William stallings, page 48
U8 rc4_getbyte(struct rc4ctx_t *rc4ctx)
{
    // Increment the swap indexes
    rc4ctx->i = (rc4ctx->i + 1) % 256;
    rc4ctx->j = (rc4ctx->j + rc4ctx->S[rc4ctx->i]) % 256;

    // Swap the values in the S array
    swap(&(rc4ctx->S[rc4ctx->i]), &(rc4ctx->S[rc4ctx->j]));

    // Sum the swapped values
    int t = (rc4ctx->S[rc4ctx->i] + rc4ctx->S[rc4ctx->j]) % 256;
    return rc4ctx->S[t];
}


// RSA functions
// Initialises the RSA context
void rsa_init(struct rsactx_t *rsactx)
{
    mpz_init(rsactx->d);
    mpz_init(rsactx->e);
    mpz_init(rsactx->n);
    mpz_init(rsactx->p);
    mpz_init(rsactx->q);
}


// Frees the memory associated with the RSA context.
void rsa_clean(struct rsactx_t *rsactx)
{
    mpz_clear(rsactx->d);
    mpz_clear(rsactx->e);
    mpz_clear(rsactx->n);
    mpz_clear(rsactx->p);
    mpz_clear(rsactx->q);
}


// Common functions
// Simply swap two values by reference
void swap(int *a, int *b)
{
    int temp;
    temp = *a;
    *a = *b;
    *b = temp;
}


// Print a c-string up to a certain length in hex
void print_hex_string(U8 hex_string[], int message_len)
{
    int i;
    for (i = 0; i < message_len; i++)
        printf("%02X", hex_string[i]);
}


// Convert hex to int, done because the system hex converter is unreliable
int hex_convert(char hex_string[], int length)
{
    int result = 0;
    int base = 1;

    int i;
    for (i = length; i > 0; i--)
    {
        switch (hex_string[i - 1])
        {
            case '0': {break;}
            case '1': {result += base * 1; break;}
            case '2': {result += base * 2; break;}
            case '3': {result += base * 3; break;}
            case '4': {result += base * 4; break;}
            case '5': {result += base * 5; break;}
            case '6': {result += base * 6; break;}
            case '7': {result += base * 7; break;}
            case '8': {result += base * 8; break;}
            case '9': {result += base * 9; break;}
            case 'A': {result += base * 10; break;}
            case 'B': {result += base * 11; break;}
            case 'C': {result += base * 12; break;}
            case 'D': {result += base * 13; break;}
            case 'E': {result += base * 14; break;}
            case 'F': {result += base * 15; break;}
            case 'a': {result += base * 10; break;}
            case 'b': {result += base * 11; break;}
            case 'c': {result += base * 12; break;}
            case 'd': {result += base * 13; break;}
            case 'e': {result += base * 14; break;}
            case 'f': {result += base * 15; break;}
            default:
            {
                printf("The input given (\'%c\') is not a valid HEX character\nTerminating...\n",
                ↪  hex_string[i]);
                exit(EXIT_FAILURE);
            }
        }
        base *= 16;
    }

    return result;
}
```

2 RC4 Encryption and Decryption

2.1 rc4.h

```
#ifndef EHN_PRAC3_RC4_H
#define EHN_PRAC3_RC4_H

#include "prac3.h"

/// The maximum length of the key (in bytes) used for the RC4 encryption utility.
#define RC4_MAX_KEY_LEN 16 // This value can be no bigger than 255 for the RC4 algorithm to work correctly

// Uses the functions defined in prac3.h

#endif // EHN_PRAC3_RC4_H
```

2.2 rc4.c

```
#include "rc4.h"


/// The main function for the RC4 encryption/decryption utility. Uses the RC4 functions in prac3.h to
↪   encrypt/decrypt
/// an input file using a key file, or using a key entered into the terminal.
int main(int argc, char *argv[])
{
    int i;
    char *input_file_name = NULL;
    char *output_file_name = NULL;
    U8 key[RC4_MAX_KEY_LEN + 2];
    int keylen;
    char *key_file_name = NULL;
    //               fi     fo     key
    bool args[3] = {false, false, false};
    char help_message[] = "\t./rc4 -arg1 value1 -arg2 value2...\n"
                          "\t\n"
                          "\tThe following arguments should then be given in this order:\n\n"
                          "\t-fi <input file>\n"
                          "\t-fo <output file>\n"
                          "\t-key <key file> (optional)\n\n"
                          "\t\nThe use of the -e or -d arguments is optional and has no effect on the operation
                          ↪  performed"
                          "\t\nRemember to add \"double quotes\" if spaces are present in an argument\n"
                          "\t\nExample usage:\n"
                          "\t1.\t./rc4 -fi \"plain text.txt\" -fo encrypted.enc -key key.txt\n"
                          "\t2.\t./rc4 -fi encrypted.enc -fo decrypted.txt\n";
```

```
printf("EHN Group 12 Practical 3\n\n");

if (argc < 4)
{
    printf("Too few arguments were supplied\n"
           "Proper use of the program is as follows:\n\n%s\n", help_message);
    return EXIT_FAILURE;
}

int arg;
for (arg = 1; arg < argc; arg++)
{
    if (!strcmp(argv[arg], "-fi")) // Set the name of the input file
    {
        args[0] = true;
        input_file_name = argv[arg + 1];
        printf("Using \"%s\" as the input file\n", input_file_name);
        arg++; // Skip over the value parameter that follows this parameter
    }
    else if (!strcmp(argv[arg], "-fo")) // Set the name of the output file
    {
        args[1] = true;
        output_file_name = argv[arg + 1];
        printf("Using \"%s\" as the output file\n", output_file_name);
        arg++; // Skip over the value parameter that follows this parameter
    }
    else if (!strcmp(argv[arg], "-key")) // Set the name of the file containing the key
    {
        args[2] = true;
        key_file_name = argv[arg + 1];
        printf("Using \"%s\" as the key file\n", key_file_name);
        arg++; // Skip over the value parameter that follows this parameter
    }
    else if ((!strcmp(argv[arg], "-e")) || (!strcmp(argv[arg], "-d")))
        continue; // Encryption and decryption follow exact same process
    else
        printf("Invalid parameter supplied: \"%s\"\n", argv[arg]);
}

if (!args[0] || !args[1]) // -fi and -fo have to be specified
{
    printf("Too few arguments were supplied\n"
           "Proper use of the program is as follows:\n\n%s\n", help_message);
    return EXIT_FAILURE;
}

// Create variables to read the key
char buffer[RC4_MAX_KEY_LEN + 1];

for (i = 0; i < RC4_MAX_KEY_LEN + 1; i++)
    buffer[i] = '\0';

if (!args[2]) // Key file is not specified, read the key from the terminal
{
    printf("Please enter the key that should be used to encrypt/decrypt the input file (ASCII):  ");
    fgets(buffer, RC4_MAX_KEY_LEN + 1, stdin); // Read only up to the max number of characters
}
else // Read the key from the key file
{
    FILE *keyfile;
    keyfile = fopen(key_file_name, "r");
    if (keyfile == NULL) // Key file does not exist
    {
        printf("The key file could not be opened, please check that the name of the file is correct\n");
        return EXIT_FAILURE;
    }
    else // Read from the file
        fgets(buffer, RC4_MAX_KEY_LEN + 1, keyfile); // Read only up to the max number of characters
}

// If a password is entered in the terminal, a newline is appended, so remove it if present
char *newlinepos = strstr(buffer, "\n");
if (newlinepos != NULL)
    *newlinepos = '\0';

keylen = (int) strlen(buffer);
for (i = 0; i < RC4_MAX_KEY_LEN + 2; i++) // Fill to pad with zeroes if needed
    key[i] = '\0';
for (i = 0; i < keylen && i < (RC4_MAX_KEY_LEN + 1); i++) // Copy up to RC4_MAX_KEY_LEN characters
    key[i] = buffer[i];

printf("Using \"%s\" as the key.\n", key);

// Open the files to be read and written
FILE *infile;
infile = fopen(input_file_name, "r");
FILE *outfile;
outfile = fopen(output_file_name, "w");
struct rc4ctx_t rc4ctx;

if (infile == NULL) // Input file does not exist
{
    printf("The input file could not be opened, please check that the name of the file is correct\n");
    if (outfile != NULL)
        fclose(outfile);
    return EXIT_FAILURE;
}
else if (outfile == NULL) // Output file could not be created
{
    printf("The output file could not be created, please make sure the program has write privileges\n");
    fclose(infile);
    fclose(outfile);
    return EXIT_FAILURE;
}
else // Read a byte, encrypt, and write to output. Repeat until entire input file is read
{
    rc4_init(&rc4ctx, key, keylen); // Initialise the RC4 structure
```

```
        U8 character;
        struct timeb start_time, end_time;
        ftime(&start_time); // Get time before operation starts

        while (fread(&character, 1, 1, infile) > 0) // Read a byte and check if input file finished reading
        {
            character ^= rc4_getbyte(&rc4ctx); // XOR read byte to encrypt
            fwrite(&character, 1, 1, outfile); // Write encrypted byte
        }

        ftime(&end_time); // Get time after operation ends
        fclose(infile); // Close and save files
        fclose(outfile);

        // Calculate time elapsed in ms and print
        int elapsed_time = (int) (1000.0 * (end_time.time - start_time.time) + (end_time.millitm -
        ↪  start_time.millitm));
        printf("Operation took %u ms\n\n", elapsed_time);
        printf("Encryption/Decryption complete \n");
    }

    return EXIT_SUCCESS;
}
```

3 RSA Public/Private Key Pair Generation

3.1 rsakeygen.h

```
#ifndef EHN_PRAC3_RSAKEYGEN_H
#define EHN_PRAC3_RSAKEYGEN_H

#include "prac3.h"

struct rc4ctx_t rc4ctx;


/**
 * Gets the next prime from a randomly generated value from RC4 RNG.
 * @param prime The prime value output.
 * @param num_bits The length of the prime number in bits.
 */
void getprime(mpz_t prime, int num_bits);


/**
 * Create the RSA key pair.
 * @param rsactx Pointer to the main RSA struct.
 * @param key_len The length of the key that needs to be encrypted in bits.
 * @param e_selection Which common value for e will be chosen.
 */
void getkeys(struct rsactx_t *rsactx, int key_len, int e_selection);


#endif // EHN_PRAC3_RSAKEYGEN_H
```

3.2 rsakeygen.c

```
#include "rsakeygen.h"


/// This utility generates a public/private key pair to be used to encrypt and decrypt the RC4 key.
int main(int argc, char *argv[])
{
    int i;
    int num_bits = -1;
    char *private_key_file_name = NULL;
    char *public_key_file_name = NULL;
    //             bitLen  fopub fopriv  init
    bool args[4] = {false, false, false, false};
    U8 seed[17];
    int seedlen = 0;
    char help_message[] = "\t./rsakeygen -arg1 value1 -arg2 value2...\n"
                          "\t\n"
                          "\tThe following arguments should then be given in this order:\n\n"
                          "\t-bitLen <number of bits>\n"
                          "\t-fopub <public key file>\n"
                          "\t-fopriv <private key file>\n"
                          "\t-init <RC4 RNG string in ASCII> (optional)"
                          "\t\nRemember to add \"double quotes\" if spaces are present in an argument\n"
                          "\t\nExample usage:\n"
                          "\t1.\t./rsakeygen -bitLen 128 -fopub \"public key.txt\" -fopriv private_key.txt
                          ↪   -init \"ASCII key\"\n";

    printf("EHN Group 12 Practical 3\n\n");

    if (argc < 6)
    {
        printf("Too few arguments were supplied\n"
               "Proper use of the program is as follows:\n\n%s\n", help_message);
        return EXIT_FAILURE;
    }

    int arg;
    int e_val = 2;
    for (arg = 1; arg < argc; arg++)
    {
        if (!strcmp(argv[arg], "-bitLen")) // Set the number of bits to generate
        {
            args[0] = true;
            for (i = 0; i < strlen(argv[arg + 1]); i++)
            {
                if (!isdigit(argv[arg + 1][i]))
                {
                    printf("Argument \"%s\" is not a valid number\n", argv[arg + 1]);
                    return EXIT_FAILURE;
                }
            }
            num_bits = (int) strtol(argv[arg + 1], NULL, 10);
            if (num_bits < 128)
            {
                printf("%i is too small\nterminating...\n", num_bits);
                return EXIT_FAILURE;
            }
            else if (num_bits > 4096)
            {
                printf("%i is too large\nterminating...\n", num_bits);
                return EXIT_FAILURE;
            }

            printf("%i bits will be generated\n", num_bits);
            arg++; // Skip over the value parameter that follows this parameter
        }
        else if (!strcmp(argv[arg], "-fopub")) // Set the name of the output file
        {
            args[1] = true;
            public_key_file_name = argv[arg + 1];
            printf("Using \"%s\" as the public key file\n", public_key_file_name);
            arg++; // Skip over the value parameter that follows this parameter
        }
        else if (!strcmp(argv[arg], "-fopriv")) // Set the name of the output file
        {
            args[2] = true;
            private_key_file_name = argv[arg + 1];
            printf("Using \"%s\" as the private key file\n", private_key_file_name);
            arg++; // Skip over the value parameter that follows this parameter
        }
        else if (!strcmp(argv[arg], "-init")) // Set RC4 init seed
        {
            args[3] = true;
            char *rc4_seed = argv[arg + 1];
            seedlen = (int) strlen(rc4_seed);

            for (i = 0; i < 17; i++) // Clear the seed to pad with zeroes if needed
                seed[i] = '\0';

            for (i = 0; i < seedlen && i < 16; i++)
                seed[i] = rc4_seed[i];

            printf("Using \"%s\" as the RC4 RNG seed.\n", seed);
            arg++; // Skip over the value parameter that follows this parameter
        }
        else
            printf("Invalid parameter supplied: \"%s\"\n", argv[arg]);
    }

    if (!args[0] || !args[1] || !args[2])
    {
        printf("Too few arguments were supplied\n"
               "Proper use of the program is as follows:\n\n%s\n", help_message);
        return EXIT_FAILURE;
    }

    if (!args[3])
    {
```

```
    seed[0] = 0x01;
    seed[1] = 0x23;
    seed[2] = 0x45;
    seed[3] = 0x67;
    seed[4] = 0x89;
    seed[5] = 0xAB;
    seed[6] = 0xCD;
    seed[7] = 0xEF;
    for (i = 8; i < 17; i++)
        seed[i] = 0;
    seedlen = 8;

    printf("No RC4 RNG seed was specified, using the default value of 0123456789ABCDEF (HEX)\n");
}

struct rsactx_t rsactx;
rsa_init(&rsactx);
rc4_init(&rc4ctx, seed, seedlen);

getkeys(&rsactx, num_bits, e_val);

// Open the public key file to be written
FILE *pubkeyfile;
pubkeyfile = fopen(public_key_file_name, "w");
U8 temp = '\n';
if (pubkeyfile == NULL) // Output file could not be created
{
    printf("The public key file could not be created, please make sure the program has write
    ↪  privileges\n");
    return EXIT_FAILURE;
}
else
{
    mpz_out_str(pubkeyfile, 10, rsactx.n);
    fwrite(&temp, 1, 1, pubkeyfile);
    mpz_out_str(pubkeyfile, 10, rsactx.e);
    fwrite(&temp, 1, 1, pubkeyfile);
    fclose(pubkeyfile);
}

// Open the private key file to be written
FILE *privkeyfile;
privkeyfile = fopen(private_key_file_name, "w");
if (privkeyfile == NULL) // Output file could not be created
{
    printf("The private key file could not be created, please make sure the program has write
    ↪  privileges\n");
    return EXIT_FAILURE;
}
else
{
    mpz_out_str(privkeyfile, 10, rsactx.n);
    fwrite(&temp, 1, 1, privkeyfile);
    mpz_out_str(privkeyfile, 10, rsactx.d);
```

```
        fwrite(&temp, 1, 1, privkeyfile);
        fclose(privkeyfile);
    }

    rsa_clean(&rsactx);
    printf("\nDone\n");
    return EXIT_SUCCESS;
}


// Gets the next prime from a randomly generated value from RC4 RNG
void getprime(mpz_t prime, int num_bits)
{
    unsigned int result;
    mpz_t temp_result;
    mpz_init_set_ui(temp_result, 1);
    mpz_t val_2;
    mpz_init_set_ui(val_2, 2);
    mpz_t val_1;
    mpz_init_set_ui(val_1, 1);

    // Loop until right length
    for (int i = 0; i < num_bits - 1; i++)
    {
        mpz_mul(temp_result, temp_result, val_2);
        result = (rc4_getbyte(&rc4ctx) & 0b00000001);
        if (result == 1)
        {
            mpz_add(temp_result, temp_result, val_1);
        }
    }
    mpz_nextprime(prime, temp_result);
}


// Create the RSA key pair
void getkeys(struct rsactx_t *rsactx, int key_len, int e_selection)
{
    mpz_t phi;
    mpz_t p_1, q_1, val_1;
    mpz_t phi_1;
    mpz_t remain;
    unsigned long i_1 = 1;
    int p_q_bit_len = (key_len) / 2;
    unsigned long e[3] = {3, 17, 65537};

    do
    {
        do
        {
            getprime(rsactx->p, p_q_bit_len);
            getprime(rsactx->q, p_q_bit_len); // Random prime p and q
        } while (mpz_get_ui(rsactx->p) == mpz_get_ui(rsactx->q)); // p != q
        mpz_mul(rsactx->n, rsactx->p, rsactx->q); // Set n
        mpz_set_ui(rsactx->e, e[e_selection]); //set e from common e values
        mpz_init_set_ui(val_1, i_1); // Create a mpz struct with val 1 for subtraction.

        mpz_init(p_1);
        mpz_sub(p_1, rsactx->p, val_1); // (p-1)

        mpz_init(q_1);
        mpz_sub(q_1, rsactx->q, val_1); // (q-1)

        mpz_init(phi);
        mpz_mul(phi, p_1, q_1); // phi = (p-1)(q-1)

        mpz_init(phi_1);
        mpz_add(phi_1, phi, val_1);

        mpz_init(remain);
        mpz_t count;
        mpz_init_set_ui(count, 1);
        do
        {
            mpz_tdiv_qr(rsactx->d, remain, phi_1, rsactx->e);
            mpz_add(count, count, val_1);
            mpz_mul(phi_1, phi, count);
            mpz_add(phi_1, phi_1, val_1);
        } while ((mpz_get_ui(remain) != 0) && (mpz_cmp(rsactx->d, phi) < 0));

    } while ((mpz_get_ui(remain) != 0) || (mpz_cmp(rsactx->d, phi) >= 0));
}
```

4 RSA Encryption

4.1 rsaencrypt.h

```
#ifndef EHN_PRAC3_RSAENCRYPT_H
#define EHN_PRAC3_RSAENCRYPT_H

#include "prac3.h"


/**
 * Uses the GMP power function to encrypt a mpz_t value.
 * @param plain The value to be encrypted.
 * @param e The public exponent.
 * @param n The modulus.
 * @param cipher The output of the encrypt operation.
 */
void encrypt_rsa(mpz_t plain, mpz_t e, mpz_t n, mpz_t cipher);


#endif // EHN_PRAC3_RSAENCRYPT_H
```

4.2 rsaencrypt.c

```c
#include "rsaencrypt.h"


/// This utility encrypts the key used in the RC4 algorithm.
int main(int argc, char *argv[])
{
    int i;
    char *public_key_file_name = NULL;
    char *output_file_name = NULL;
    char key[17];
    //               key    fo    fopub
    bool args[3] = {false, false, false};
    char help_message[] = "\t./rsaencrypt -arg1 value1 -arg2 value2...\n"
                          "\t\n"
                          "\tThe following arguments should then be given in this order:\n\n"
                          "\t-key <key in ASCII>\n"
                          "\t-fo <output file>\n"
                          "\t-fopub <public key file>\n\n"
                          "\t\nRemember to add \"double quotes\" if spaces are present in an argument\n"
                          "\t\nExample usage:\n"
                          "\t1.\t./rsaencrypt -key \"ASCII key\" -fo cipher.key -fopub \"public key.txt\"\n";

    printf("EHN Group 12 Practical 3\n\n");

    if (argc < 6)
    {
        printf("Too few arguments were supplied\n"
               "Proper use of the program is as follows:\n\n%s\n", help_message);
        return EXIT_FAILURE;
    }

    int arg;
    for (arg = 1; arg < argc; arg++)
    {
        if (!strcmp(argv[arg], "-key")) // Set the name of the file containing the key
        {
            args[0] = true;
            int keylen = (int) strlen(argv[arg + 1]);
            for (i = 0; i < 17; i++) // Fill to pad with zeroes if needed
                key[i] = '\0';
            for (i = 0; i < keylen && i < 16; i++) // Copy up to 16 characters
                key[i] = argv[arg + 1][i];
            printf("Using \"%s\" as the key\n", key);
            arg++; // Skip over the value parameter that follows this parameter
        }
        else if (!strcmp(argv[arg], "-fo")) // Set the name of the output file
        {
            args[1] = true;
            output_file_name = argv[arg + 1];
            printf("Using \"%s\" as the output file\n", output_file_name);
            arg++; // Skip over the value parameter that follows this parameter
        }
        else if (!strcmp(argv[arg], "-fopub")) // Set the name of the public key file
        {
            args[2] = true;
            public_key_file_name = argv[arg + 1];
            printf("Using \"%s\" as the public RSA key file\n", public_key_file_name);
            arg++; // Skip over the value parameter that follows this parameter
        }
        else
            printf("Invalid parameter supplied: \"%s\"\n", argv[arg]);
    }

    if (!args[0] || !args[1] || !args[2])
    {
        printf("Too few arguments were supplied\n"
               "Proper use of the program is as follows:\n\n%s\n", help_message);
        return EXIT_FAILURE;
    }

    char buffer[2049];
    for (i = 0; i < 2049; i++)
        buffer[i] = '\0';

    struct rsactx_t rsactx;
    rsa_init(&rsactx);

    // Open the public key file to be read
    FILE *pubkeyfile;
    pubkeyfile = fopen(public_key_file_name, "r");
    if (pubkeyfile == NULL) // Key file could not be found
    {
        printf("The encrypted file could not be opened, please check that the name of the file is correct\n");
        return EXIT_FAILURE;
    }
    else
    {
        int result = -1;
        if (fgets(buffer, 2048, pubkeyfile) != NULL) // Get n from public key file
            result = mpz_set_str(rsactx.n, buffer, 10);

        if (result == -1) // Could not read or invalid
        {
            printf("Could not read n from the private key file\n");
            return EXIT_FAILURE;
        }

        result = -1;
        if (fgets(buffer, 2048, pubkeyfile) != NULL) // Get e from public key file
            result = mpz_set_str(rsactx.e, buffer, 10);

        if (result == -1) // Could not read or invalid
        {
            printf("Could not read e from the private key file\n");
            return EXIT_FAILURE;
        }
        fclose(pubkeyfile);
    }

    mpz_t plain, cipher, temp_val, total, byte;
    mpz_init(cipher);
    mpz_init(plain);
    mpz_init_set_ui(total, key[0]);
    mpz_init_set_ui(byte, 256);
    mpz_init(temp_val);
    for (int j = 1; j < 16; ++j)
    {
        mpz_mul(total, total, byte); // Shift byte
        mpz_set_ui(temp_val, key[j]);
        mpz_add(total, total, temp_val);
    }

    // Open the public key file to be written
    FILE *outfile;
    outfile = fopen(output_file_name, "w");
    if (outfile == NULL) // Output file could not be created
    {
        printf("The output file could not be created, please make sure the program has write privileges\n");
        return EXIT_FAILURE;
    }
    else
    {
        encrypt_rsa(total, rsactx.e, rsactx.n, cipher);
        mpz_out_str(outfile, 10, cipher);

        U8 new = '\n';
        fwrite(&new, 1, 1, outfile);
        new = '\0';
        fwrite(&new, 1, 1, outfile);
    }
    rsa_clean(&rsactx);
    printf("\nDone\n");
    return EXIT_SUCCESS;
}


// Uses the GMP power function to encrypt a mpz_t number
void encrypt_rsa(mpz_t plain, mpz_t e, mpz_t n, mpz_t cipher)
{
    mpz_powm(cipher, plain, e, n);
}
```

5 RSA Decryption

5.1 rsadecrypt.h

```
#ifndef EHN_PRAC3_RSADECRYPT_H
#define EHN_PRAC3_RSADECRYPT_H

#include "prac3.h"
```

```
/**
 * Uses the GMP power function to decrypt a mpz_t value.
 * @param plain The output of the decrypt operation.
 * @param d The secret exponent.
 * @param n The modulus.
 * @param cipher The value to be decrypted.
 */
void decrypt_rsa(mpz_t plain, mpz_t d, mpz_t n, mpz_t cipher);


#endif // EHN_PRAC3_RSADECRYPT_H
```

5.2 rsadecrypt.c

```
#include "rsadecrypt.h"


/// This utility decrypts the key used in the RC4 algorithm.
int main(int argc, char *argv[])
{
    int i;
    char *private_key_file_name = NULL;
    char *output_file_name = NULL;
    char *input_file_name = NULL;
    //               fi   fopriv   fo
    bool args[3] = {false, false, false};
    char help_message[] = "\t./rsadecrypt -arg1 value1 -arg2 value2...\n"
                          "\t\n"
                          "\tThe following arguments should then be given in this order:\n\n"
                          "\t-fi <input file>\n"
                          "\t-fo <output file>\n"
                          "\t-fopriv <private key file>\n\n"
                          "\t\nRemember to add \"double quotes\" if spaces are present in an argument\n"
                          "\t\nExample usage:\n"
                          "\t1.\t./rsadecrypt -fi cipher.key -fo plain.txt -fopriv \"private key.txt\"\n";

    printf("EHN Group 12 Practical 3\n\n");

    if (argc < 6)
    {
        printf("Too few arguments were supplied\n"
               "Proper use of the program is as follows:\n\n%s\n", help_message);
        return EXIT_FAILURE;
    }

    int arg;
    for (arg = 1; arg < argc; arg++)
    {
        if (!strcmp(argv[arg], "-fi")) // Set the name of the input file
        {
            args[0] = true;
            input_file_name = argv[arg + 1];
            printf("Using \"%s\" as the input file\n", input_file_name);
            arg++; // Skip over the value parameter that follows this parameter
        }
        else if (!strcmp(argv[arg], "-fopriv")) // Set the name of the private key file
        {
            args[1] = true;
            private_key_file_name = argv[arg + 1];
            printf("Using \"%s\" as the private RSA key file\n", private_key_file_name);
            arg++; // Skip over the value parameter that follows this parameter
        }
        else if (!strcmp(argv[arg], "-fo")) // Set the name of the output file
        {
            args[2] = true;
            output_file_name = argv[arg + 1];
            printf("Using \"%s\" as the output file\n", output_file_name);
            arg++; // Skip over the value parameter that follows this parameter
        }
        else
            printf("Invalid parameter supplied: \"%s\"\n", argv[arg]);
    }

    if (!args[0] || !args[1] || !args[2])
    {
        printf("Too few arguments were supplied\n"
               "Proper use of the program is as follows:\n\n%s\n", help_message);
        return EXIT_FAILURE;
    }

    char buffer[2049];
    for (i = 0; i < 2049; i++)
        buffer[i] = '\0';

    struct rsactx_t rsactx;
    rsa_init(&rsactx);

    // Open the private key file to be written
    FILE *privkeyfile;
    privkeyfile = fopen(private_key_file_name, "r");
    if (privkeyfile == NULL) // Key file could not be found
    {
        printf("The private key file could not be opened, please check that the name of the file is
        ↪  correct\n");
        return EXIT_FAILURE;
    }
    else
    {
        int result = -1;
        if (fgets(buffer, 2048, privkeyfile) != NULL) // Get n from public key file
            result = mpz_set_str(rsactx.n, buffer, 10);

        if (result == -1) // Could not read or invalid
        {
            printf("Could not read n from the private key file\n");
            return EXIT_FAILURE;
        }
        result = -1;
        if (fgets(buffer, 2048, privkeyfile) != NULL) // Get d from public key file
            result = mpz_set_str(rsactx.d, buffer, 10);

        if (result == -1) // Could not read or invalid
        {
            printf("Could not read d from the private key file\n");
            return EXIT_FAILURE;
        }
        fclose(privkeyfile);
    }

    char out_text[17];
    for (i = 0; i < 17; i++)
        buffer[i] = '\0';
    mpz_t plain, cipher, temp_val, shift, val_2;
    mpz_init(plain);
    mpz_init(cipher);
    mpz_init_set_ui(val_2, 2);
    mpz_init(temp_val);
    mpz_init(shift);

    // Open the encrypted file
    FILE *infile;
    infile = fopen(input_file_name, "r");
    if (infile == NULL) // Input file could not be found
    {
        printf("The encrypted file could not be opened, please check that the name of the file is correct\n");
        return EXIT_FAILURE;
    }
    else // Open output file, decrypt input and write to output
    {
        FILE *outfile;
        outfile = fopen(output_file_name, "w");
        if (outfile == NULL) // Output file could not be created
        {
            printf("The output file could not be created, please make sure the program has write
            ↪  privileges\n");
            return EXIT_FAILURE;
        }
        else // Read char, decipher char, write to output, repeat
        {
            int result = -1;
            if (fgets(buffer, 2048, infile) != NULL) // Get cipher value from the input file
                result = mpz_set_str(cipher, buffer, 10);

            if (result == -1) // Could not read or invalid
            {
                printf("Could not read the ciphertext from the input file\n");
                return EXIT_FAILURE;
            }

            decrypt_rsa(plain, rsactx.d, rsactx.n, cipher); // Decipher
```

```
            for (i = 0; i < 16; i++)
            {
                mpz_pow_ui(shift, val_2, 8 * (15 - i));
                mpz_tdiv_q(temp_val, plain, shift);
                out_text[i] = mpz_get_ui(temp_val);
                mpz_mul(temp_val, temp_val, shift);
                mpz_sub(plain, plain, temp_val);
            }
            out_text[16] = '\0';
            fputs(out_text, outfile);

            fprintf(outfile, "\n");
            fclose(infile);
            fclose(outfile);
        }
    }

    rsa_clean(&rsactx);
    printf("\nDone\n");
    return EXIT_SUCCESS;
}


// Uses the GMP power function to decrypt a mpz_t number
void decrypt_rsa(mpz_t plain, mpz_t d, mpz_t n, mpz_t cipher)
{
    mpz_powm(plain, cipher, d, n);
}
```