МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ
ХАРКІВСЬКИЙ НАЦІОНАЛЬНИЙ УНІВЕРСИТЕТ РАДІОЕЛЕКТРОНІКИ

Кафедра Програмної інженерії

Звіт
з лабораторної роботи № 2
з дисципліни: «Операційні системи»
з теми: «Створення та використання бібліотек. Частина 1»

Виконав:
ст. гр. ПЗПІ-23-2
Ситник Є. С.

Перевірила:
доц. каф. ПІ
Мельнікова Р. В.

Харків – 2025

2 СТВОРЕННЯ ТА ВИКОРИСТАННЯ БІБЛІОТЕК. ЧАСТИНА 1

2.1 Мета роботи

Вивчити прийоми та методи створення та використання динамічних бібліотек.

2.2 Хід роботи

2.2.1 Визначите необхідні функції для реалізації алгоритму RSA. Визначите серед цих функцій ті, які повинні використовуватись в зовнішніх програмах

RSA – це криптографічний алгоритм, який використовує пару ключів: відкритий і закритий. Відкритий ключ використовується для шифрування повідомлень, а закритий – для їх розшифрування.

Для реалізації алгоритму RSA знадобиться мінімум 3 функції:

а) функція генерації ключів;

б) функція шифрування даних;

в) функція розшифрування даних.

Також для зручності має сенс створити структуру, що буде зберігати пару ключів.

Ці функції та структура будуть доступні користувачу бібліотеки для зовнішнього використання.

2.2.2 Складіть файл заголовків таким чином, щоб цей файл заголовків можна було б використовувати для динамічної бібліотеки та програм, які використовують бібліотеку

Для операційної системи Windows створення динамічних бібліотек вимагає визначення певних розширених атрибутів класів зберігання (extended storage-class attributes), а саме – «dllimport» та «dllexport».

Для універсальності бібліотеки необхідно створити макрос, який буде автоматично встановлювати потрібні атрибути в залежності від режиму компіляції.

Можна виділити такі особливі випадки компіляції:

а) компіляція статичної бібліотеки для операційної системи Windows;

б) компіляція динамічної бібліотеки для операційної системи Windows;

в) компіляція бібліотеки для інших операційних систем;

г) компіляція бібліотеки для використання із C;

д) компіляція бібліотеки для використання із C++.

Для визначення режиму компіляції використаємо макроси, які повинен вказати користувач, та стандартні макроси та вирази, що надає компілятор.

Також не варто забувати про можливість підключення користувачем файлу заголовків в кількох місцях. Існує кілька способів запобігання подібним ситуаціям, найкращим для цієї бібліотеки буде використання макросу, який запобігатиме повторному включенню файлу заголовків, адже цей спосіб не залежить від компілятора, що використовується.

Всі файли, що стосуються бібліотеки (файл заголовків «rsa.h» та файл реалізації «rsa.cpp») знаходитимуться в директорії «library».

rsa.h

```
#ifndef RSA_LIB
#define RSA_LIB

#ifdef __cplusplus
extern "C" {
#endif

#ifdef _WIN32
  #ifdef BUILD_SHARED
    #define BUILD_SPEC __declspec(dllexport)
  #elif BUILD_STATIC
    #define BUILD_SPEC
  #else
    #define BUILD_SPEC __declspec(dllimport)
  #endif
#else
  #define BUILD_SPEC
#endif

typedef struct {
  __int128_t d;
  __int128_t e;
  __int128_t n;
} rsa_Pair;

BUILD_SPEC rsa_Pair rsa_gen_pair(__int128_t p, __int128_t q, __int128_t e);

BUILD_SPEC __int128_t rsa_encrypt(__int128_t data, __int128_t e, __int128_t n);
BUILD_SPEC __int128_t rsa_decrypt(__int128_t data, __int128_t d, __int128_t n);

#ifdef __cplusplus
}
#endif

#endif
```

Рисунок 2.1 – Файл заголовків «rsa.h»

Для запобігання повторному включенню файлу заголовків використано конструкцію із макросом «RSA_LIB». Файл буде включено тільки перший раз, всі наступні рази препроцесор не включить вміст файлу, через визначений макрос.

Для можливості використання бібліотеки як із C, так і з C++ використано конструкцію «extern C», що включається у файл тільки за умови визначення макросу «__cplusplus», що додається компілятором під час компіляції в режимі C++.

Для сумісності бібліотеки із різними операційними системами використано конструкцію із макросом «_WIN32», що визначає необхідні атрибути лише під час компіляції для цільової платформи Windows.

Для визначення атрибутів під час статичної та динамічної компіляції використано конструкцію із макросами «BUILD_SHARED» та «BUILD_STATIC».

В залежності від результату виконання вище вказаних конструкцій буде визначено макрос «BUILD_SPEC», який використовується під час оголошення функцій для додавання необхідних атрибутів.

Для всіх функцій в якості типу аргументів та результату було використано тип «__int128_t», що є найбільшим із стандартних типів на момент написання. Варто зазначити, що для шифрування RSA зазвичай використовують значно більші типи, реалізовані в спеціалізованих бібліотеках, таких як «OpenSSL BN» або «GNU Multiple Precision Arithmetic Library».

Оскільки C, на відміну від C++, не має вбудованої можливості ізолювати функції та змінні в просторах імен, кожна функція та структура починаються із назви бібліотеки, а саме – «rsa».

2.2.3 Реалізуйте функції бібліотек

Перш за все необхідно реалізувати функцію створення пари ключів на основі переданих користувачем простих чисел та відкритої експоненти (див. рис. 2.2).

rsa.cpp

```cpp
rsa_Pair rsa_gen_pair(__int128_t p, __int128_t q, __int128_t e) {
  rsa_Pair pair;
  pair.e = e;
  pair.n = p * q;

  __int128_t phi = (p - 1) * (q - 1);
  pair.d = mod_inverse(e, phi);

  return pair;
}
```

Рисунок 2.2 – Функція «rsa_gen_pair»

Далі реалізуємо функції шифрування та розшифрування (див. рис. 2.3).

rsa.cpp

```cpp
__int128_t rsa_encrypt(__int128_t data, __int128_t e, __int128_t n) {
  return pow_mod(data, e, n);
}

__int128_t rsa_decrypt(__int128_t data, __int128_t d, __int128_t n) {
  return pow_mod(data, d, n);
}
```

Рисунок 2.3 – Функції «rsa_encrypt» та «rsa_decrypt»

Оскільки стандартна бібліотека C не включає необхідні математичні функції їх також необхідно реалізувати (див. рис. 2.4). Ці функції розраховані на внутрішнє використання, тому їх нема в файлі заголовків.

rsa.cpp

```cpp
__int128_t mul_mod(__int128_t a, __int128_t b, __int128_t mod) {
  __int128_t res = 0;
  a %= mod;

  while (b > 0) {
    if (b & 1)
      res = (res + a) % mod;

    a = (a << 1) % mod;
    b >>= 1;
  }

  return res;
}

__int128_t pow_mod(__int128_t base, __int128_t exp, __int128_t mod) {
  __int128_t res = 1;
  base %= mod;

  while (exp > 0) {
    if (exp & 1)
      res = mul_mod(res, base, mod);

    base = mul_mod(base, base, mod);
    exp >>= 1;
  }

  return res;
}

__int128_t mod_inverse(__int128_t e, __int128_t phi) {
  __int128_t a = e, b = phi;
  __int128_t x = 1, y = 0;
  __int128_t x1 = 0, y1 = 1;
  __int128_t q, temp;

  while (b != 0) {
    q = a / b;

    temp = a % b;
    a = b;
    b = temp;

    temp = x - q * x1;
    x = x1;
    x1 = temp;

    temp = y - q * y1;
    y = y1;
    y1 = temp;
  }

  if (x < 0)
    x += phi;

  return x;
}
```

Рисунок 2.4 – Математичні функції «mul_mod», «pow_mod» та «mod_inverse»

2.2.4 Створіть динамічну бібліотеку

Оскільки я використовую операційну систему Arch Linux, у мене нема доступу до компілятора Microsoft MSVC. Натомість мною було використано компілятор Clang із проєкту LLVM, та набір інструментів MinGW, що забезпечують кросс-компіляцію для Windows на POSIX-сумісних операційних системах (Linux, MacOS, BSD і подібних).

Під час компіляції динамічної бібліотеки необхідно:

а) визначити макрос «BUILD_SHARED»;

б) вказати прапорець «shared» для компілятора;

в) встановити значення, що відповідає назві бібліотеки, для прапорця «out-implib» для компонувальника.

Компілятор автоматично викликає компонувальник, передати аргументи для нього можна за допомогою прапорця «Wl».

Динамічні бібліотеки для Windows складаються з 2 компонентів:

а) файл із розширенням «.dll» в якому знаходиться реалізація бібліотеки;

б) файл із розширенням «.lib» в якому знаходиться інформація, необхідна для зв’язування програми та бібліотеки.

Для зручнішого управління проєктом має сенс використовувати систему збірки, та визначати всі команди компіляції за її допомогою. Я використаю систему збірки «Just» (див. рис. 2.5). Всі скомпільовані файли будуть знаходитись в директорії «build».

```
cc := "x86_64-w64-mingw32-clang++"
flags := "-Wall -Wextra -std=c++17"

@clean:
  rm -rf build

@makedir-build: clean
  mkdir -p build

@build-lib-shared: makedir-build
  {{cc}} {{flags}} -D BUILD_SHARED --shared -Wl,--out-implib,build/rsa.lib library/rsa.cpp -o build/rsa.dll
```

Рисунок 2.5 – Вміст «Justfile» з інструкціями для збірки

Скомпілюємо бібліотеку та перевіримо результат (див. рис. 2.6).

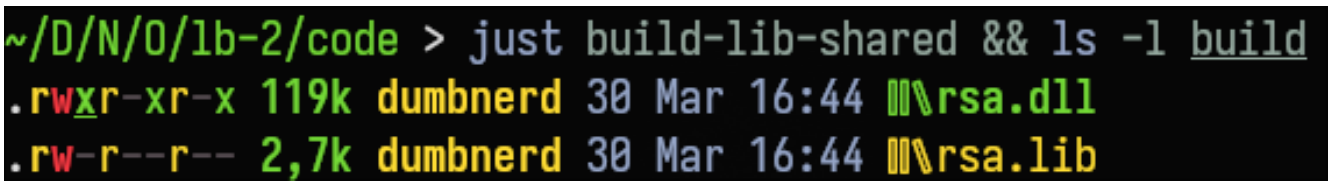

Рисунок 2.6 – Результат збірки динамічної бібліотеки

Можемо побачити, що компіляція пройшла без помилок та попереджень, та після компіляції, як і очікувалося, ми отримали 2 файли бібліотеки.

2.2.5 Реалізуйте головну програму для динамічної бібліотеки для першого способу її використання

Відповідно умові реалізуємо програму із використанням функцій бібліотеки. Для доступу до них необхідно підключити файл заголовків бібліотеки.

Оберемо пари простих чисел та відкриті експоненти для створення 2 пар ключів, створимо пари, почергово перевіримо роботу функцій шифрування та дешифрування із 2 парами ключів на різних даних друкуючу в консоль результат кожного етапу.

Оскільки бібліотека розрахована на роботу із даними типу «__int128_t», а стандартні функції друку в консоль не підтримують цей тип, додатково реалізуємо функцію для перетворення чисел такого типу на строки.

Всі файли, що стосуються програми (файл реалізації «main.cpp») знаходитимуться в директорії «application».

main.cpp

```cpp
#include <stdint.h>
#include <stdio.h>

#include "../library/rsa.h"

char *int128_to_str(__int128_t n) {
  static char str[41] = {0};
  char *s = str + sizeof(str) - 1;
  bool neg = n < 0;
  if (neg)
    n = -n;
  do {
    *--s = "0123456789"[n % 10];
    n /= 10;
  } while (n);
  if (neg)
    *--s = '-';
  return s;
}

int main() {
  uint64_t p0 = 8589934609, q0 = 2147483693, e0 = 65537;
  uint64_t p1 = 2147483659, q1 = 8589934621, e1 = 65537;

  rsa_Pair pair_0 = rsa_gen_pair(p0, q0, e0);
  rsa_Pair pair_1 = rsa_gen_pair(p1, q1, e1);

  printf("pair_0:\n");
  printf("- e:\t%s\n", int128_to_str(pair_0.e));
  printf("- d:\t%s\n", int128_to_str(pair_0.d));
  printf("- n:\t%s\n", int128_to_str(pair_0.n));

  printf("pair_1:\n");
  printf("- e:\t%s\n", int128_to_str(pair_1.e));
  printf("- d:\t%s\n", int128_to_str(pair_1.d));
  printf("- n:\t%s\n", int128_to_str(pair_1.n));

  uint64_t t[] = {22, 17, 2, 65500, 100};

  for (size_t i = 0; i < 5; i++) {
    printf("- - - - - - - - - - - - - - - - - -\n");

    printf("t[%zu]:\t\t%lld\n", i, t[i]);

    __int128_t e1t = rsa_encrypt(t[i], pair_0.e, pair_0.n);
    printf("e1t:\t\t%s\n", int128_to_str(e1t));

    __int128_t d1e1t = rsa_decrypt(e1t, pair_0.d, pair_0.n);
    printf("d1e1t:\t\t%s\n", int128_to_str(d1e1t));

    if (d1e1t != t[i]) {
      fprintf(stderr, "Error: decrypt[i]on failed - expected: %llu; got: %s\n",
              t[i], int128_to_str(d1e1t));
      return 1;
    }

    __int128_t e0d1e1t = rsa_encrypt(t[i], pair_1.e, pair_1.n);
    printf("e0d1e1t:\t%s\n", int128_to_str(e0d1e1t));

    __int128_t d0e0d1e1t = rsa_decrypt(e0d1e1t, pair_1.d, pair_1.n);
    printf("d0e0d1e1t:\t%s\n", int128_to_str(d0e0d1e1t));

    if (d0e0d1e1t != t[i]) {
      fprintf(stderr, "Error: decrypt[i]on failed - expected: %llu; got: %s\n",
              t[i], int128_to_str(d0e0d1e1t));
      return 1;
    }
  }

  return 0;
}
```

Рисунок 2.7 – Вміст файлу «main.cpp»

2.3 Висновки

Під час виконання даної лабораторної роботи я вивчив прийоми та методи створення та використання динамічних бібліотек.

ДОДАТОК А

Вміст файлу «rsa.h»

```
#ifndef RSA_LIB
#define RSA_LIB

#ifdef __cplusplus
extern "C" {
#endif

#ifdef _WIN32
  #ifdef BUILD_SHARED
    #define BUILD_SPEC __declspec(dllexport)
  #elif BUILD_STATIC
    #define BUILD_SPEC
  #else
    #define BUILD_SPEC __declspec(dllimport)
  #endif
#else
  #define BUILD_SPEC
#endif

typedef struct {
  __int128_t d;
  __int128_t e;
  __int128_t n;
} rsa_Pair;

BUILD_SPEC rsa_Pair rsa_gen_pair(__int128_t p, __int128_t q, __int128_t
e);

BUILD_SPEC __int128_t rsa_encrypt(__int128_t data, __int128_t e,
__int128_t n);
BUILD_SPEC __int128_t rsa_decrypt(__int128_t data, __int128_t d,
__int128_t n);

#ifdef __cplusplus
}
```

```
#endif

#endif
```

ДОДАТОК Б

Вміст файлу «rsa.cpp»

```
#include "rsa.h"

__int128_t mul_mod(__int128_t a, __int128_t b, __int128_t mod) {
  __int128_t res = 0;
  a %= mod;

  while (b > 0) {
    if (b & 1)
      res = (res + a) % mod;

    a = (a << 1) % mod;
    b >>= 1;
  }

  return res;
}

__int128_t pow_mod(__int128_t base, __int128_t exp, __int128_t mod) {
  __int128_t res = 1;
  base %= mod;

  while (exp > 0) {
    if (exp & 1)
      res = mul_mod(res, base, mod);

    base = mul_mod(base, base, mod);
    exp >>= 1;
  }

  return res;
}

__int128_t mod_inverse(__int128_t e, __int128_t phi) {
  __int128_t a = e, b = phi;
  __int128_t x = 1, y = 0;
```

```
  __int128_t x1 = 0, y1 = 1;
  __int128_t q, temp;

  while (b != 0) {
    q = a / b;

    temp = a % b;
    a = b;
    b = temp;

    temp = x - q * x1;
    x = x1;
    x1 = temp;

    temp = y - q * y1;
    y = y1;
    y1 = temp;
  }

  if (x < 0)
    x += phi;

  return x;
}

rsa_Pair rsa_gen_pair(__int128_t p, __int128_t q, __int128_t e) {
  rsa_Pair pair;
  pair.e = e;
  pair.n = p * q;

  __int128_t phi = (p - 1) * (q - 1);
  pair.d = mod_inverse(e, phi);

  return pair;
}

__int128_t rsa_encrypt(__int128_t data, __int128_t e, __int128_t n) {
  return pow_mod(data, e, n);
```

```
}

__int128_t rsa_decrypt(__int128_t data, __int128_t d, __int128_t n) {
  return pow_mod(data, d, n);
}
```

ДОДАТОК В

Вміст файлу «main.cpp»

```
#include <stdint.h>
#include <stdio.h>

#include "../library/rsa.h"

char *int128_to_str(__int128_t n) {
  static char str[41] = {0};
  char *s = str + sizeof(str) - 1;
  bool neg = n < 0;
  if (neg)
    n = -n;
  do {
    *--s = "0123456789"[n % 10];
    n /= 10;
  } while (n);
  if (neg)
    *--s = '-';
  return s;
}

int main() {
  uint64_t p0 = 8589934609, q0 = 2147483693, e0 = 65537;
  uint64_t p1 = 2147483659, q1 = 8589934621, e1 = 65537;

  rsa_Pair pair_0 = rsa_gen_pair(p0, q0, e0);
  rsa_Pair pair_1 = rsa_gen_pair(p1, q1, e1);

  printf("pair_0:\n");
  printf("- e:\t%s\n", int128_to_str(pair_0.e));
  printf("- d:\t%s\n", int128_to_str(pair_0.d));
  printf("- n:\t%s\n", int128_to_str(pair_0.n));

  printf("pair_1:\n");
  printf("- e:\t%s\n", int128_to_str(pair_1.e));
  printf("- d:\t%s\n", int128_to_str(pair_1.d));
```

```
  printf("- n:\t%s\n", int128_to_str(pair_1.n));

  uint64_t t[] = {22, 17, 2, 65500, 100};

  for (size_t i = 0; i < 5; i++) {
    printf("- - - - - - - - - - - - - - - - - -\n");

    printf("t[%zu]:\t\t%lld\n", i, t[i]);

    __int128_t e1t = rsa_encrypt(t[i], pair_0.e, pair_0.n);
    printf("e1t:\t\t%s\n", int128_to_str(e1t));

    __int128_t d1e1t = rsa_decrypt(e1t, pair_0.d, pair_0.n);
    printf("d1e1t:\t\t%s\n", int128_to_str(d1e1t));

    if (d1e1t != t[i]) {
      fprintf(stderr, "Error: decrypt[i]on failed - expected: %llu; got:
%s\n",
              t[i], int128_to_str(d1e1t));
      return 1;
    }

    __int128_t e0d1e1t = rsa_encrypt(t[i], pair_1.e, pair_1.n);
    printf("e0d1e1t:\t%s\n", int128_to_str(e0d1e1t));

    __int128_t d0e0d1e1t = rsa_decrypt(e0d1e1t, pair_1.d, pair_1.n);
    printf("d0e0d1e1t:\t%s\n", int128_to_str(d0e0d1e1t));

    if (d0e0d1e1t != t[i]) {
      fprintf(stderr, "Error: decrypt[i]on failed - expected: %llu; got:
%s\n",
              t[i], int128_to_str(d0e0d1e1t));
      return 1;
    }
  }

  return 0;
}
```

ДОДАТОК Г

Вміст файлу «Justfile»

```
cc := "x86_64-w64-mingw32-clang++"
flags := "-Wall -Wextra -std=c++17"

@clean:
  rm -rf build

@makedir-build: clean
  mkdir -p build

@build-lib-shared: makedir-build
  {{cc}} {{flags}} -D BUILD_SHARED --shared -Wl,--out-implib,build/
rsa.lib library/rsa.cpp -o build/rsa.dll

@build-app-shared: build-lib-shared
  {{cc}} {{flags}} application/main.cpp -L build -l rsa -o build/main.exe
```