Харківський університет радіоелектроніки

Факультет комп'ютерних наук

Кафедра програмної інженерії

ЗВІТ

до лабораторної роботи №2 з дисципліни

"Безпека програм та даних"

на тему "ПРОГРАМНА РЕАЛІЗАЦІЯ ШИФРУВАННЯ ДАНИХ ЗА ДОПОМОГОЮ НЕСИМЕТРИЧНОГО КРИПТОАЛГОРИТМУ RSA"

Виконав ст. гр ПЗПІ-20-2

Овчаренко Михайло Миколайович

Перевірив

асистент кафедри ПІ

Олійник О. О.

Харків 2023

МЕТА

Отримати навички використання методики роботи асиметричних алгоритмів шифрування. Реалізувати програмно (на будь-якій мові програмування) роботу алгоритму RSA.

ЗАВДАННЯ

Відтворити двосторонню авторизацію клієнтської та серверної програм в мережі через алгоритм RSA.

ХІД РОБОТИ

Реалізуємо обидві програми мовою C, які взаємодіятимуть мережею за протоколом TCP. Для цього використаємо програмний інтерфейс сокетів стандарта POSIX. Це є функції socket; listen; connect; send; recv; bind; setsockopt, та інші.

Алгоритм двосторонньої авторизації працює таким чином:
1) Клієнт генерує ключі. Відкритий ключ C1 надсилає серверу, закритий C2 тримає у себе.

2) Сервер створює випадковий блок байт та підписує його публічним ключем клієнта C1. Надсилає його клієнтові.

3) Клієнт розшифровує повідомлення власним закритим ключем. Надсилає у відповідь.

4) Сервер перевіряє, чи збігається оригінальний блок з щойно отриманим. Якщо так, клієнта авторизовано. Якщо ні – розриває з’єднання.

5) Клієнт авторизований. Повторити шаги 1-4, але тепер вже по відношенню до сервера. Якщо сервер не авторизовано, розірвати з’єднання.

Програмі сервера відповідає файл server/server.c, клієнта – client/client.c. Реалізація алгоритму RSA міститься в rsa/rsa.c та була створена мною ще під час курсу «Операційні системи» на другому курсі. А net/net.c являє собою спрощену та кросплатформову обгортку інтерфейсу сокетів для Unix-like систем та Win32. Усі файли похідного коду містяться в додатку А.

ВИСНОВКИ

Імплементував двосторонню авторизацію клієнтської та серверної програм в мережі через алгоритм RSA.

ДОДАТОК А

Програмний код

Файл server.c

```
 1  #include <stdio.h>
 2  #include <string.h>
 3  #include <stdint.h>
 4  #include <stdlib.h>
 5  #include "../rsa/rsa.h"
 6  #include "../net/net.h"
 7  #define BUFF 512
 8
 9  typedef enum {false, true} bool;
10
11  int main (void) {
12          srand(time(0) + 1);
13      int listener = listen_net("127.0.0.1:8080");
14          if(listener < 0)
15          {
16                  fprintf(stderr, "Error: %d\n", listener);
17                  return listener;
18          }
19          printf("Server is listening ...\n");
20
21      char buffer[BUFF];
22
23          const int conn = accept_net(listener);
24
25          if (conn < 0) {
```

```
26                      fprintf(stderr, "Error: accept_net\n");
27                      return conn;
28              }
29
30              recv_net(conn, buffer, BUFF);
31              uint64_t ce = atoll(buffer);
32              int i = 0;
33              for(; buffer[i] != ' '; i++);
34              uint64_t cn = atoll(buffer + i + 1);
35              uint64_t test_block = die();
36              uint64_t i_test_block = rsa_encrypt(test_block, ce,
cn);
37              memset(buffer, 0, BUFF);
38              int len = sprintf(buffer, "%llu", i_test_block);
39              if(len < 0)
40              {
41                      fprintf(stderr, "Error formatting
test_block");
42                      return 4;
43              }
44              send_net(conn, buffer, BUFF);
45
46              memset(buffer, 0, BUFF);
47              recv_net(conn, buffer, BUFF);
48              uint64_t got_test_block = atoll(buffer);
49
50              if(got_test_block == test_block)
51              {
52                      static char* greet = "Successful client
authentication!";
53                      printf("Server says: %s\n", greet);
54              }
```

```
    55          else
    56          {
    57                  char* failure = "Client authentication
failed.";
    58                  printf("Server says: %s\n", failure);
    59                  close_net(conn);
    60          }
    61
    62          memset(buffer, 0, BUFF);
    63          uint64_t e,d,n;
    64          rsa_key_gen(&e, &d, &n);
    65
    66          len = sprintf(buffer, "%llu %llu", e, n);
    67          if(len < 0)
    68          {
    69                  fprintf(stderr, "Error formatting public
key");
    70                  return 4;
    71          }
    72          send_net(conn, buffer, BUFF);
    73
    74          memset(buffer, 0, BUFF);
    75          recv_net(conn, buffer, BUFF);
    76          uint64_t test_in = atoll(buffer);
    77          uint64_t test_out = rsa_decrypt(test_in, d, n);
    78          memset(buffer, 0, BUFF);
    79          len = sprintf(buffer, "%llu", test_out);
    80          if(len < 0)
    81          {
    82                  fprintf(stderr, "Error formatting test
block");
    83                  return 4;
```

```
84          }
85          send_net(conn, buffer, BUFF);
86
87          close_net(conn);
88
89      return 0;
90  }
```

Файл client.c

```
1  #include <stdio.h>
2  #include <stdlib.h>
3  #include <unistd.h>
4  #include <string.h>
5
6  #include <arpa/inet.h>
7  #include <sys/types.h>
8  #include <sys/socket.h>
9  #include <netinet/in.h>
10 #include "../rsa/rsa.h"
11 #include "../net/net.h"
12 #define BUFF 512
13
14 typedef enum {false, true} bool;
15
16 int main (void) {
17          srand(time(0));
18      const int conn = connect_net("127.0.0.1:8080");
19      if (conn < 0) {
20          fprintf(stderr, "Error: connect_net\n");
21          return conn;
```

```
    22      }
    23
    24      char buffer[BUFF];
    25          uint64_t e, d, n;
    26          rsa_key_gen(&e, &d, &n);
    27
    28          int len = sprintf(buffer, "%llu %llu", e, n);
    29          if (len < 0)
    30          {
    31                  fprintf(stderr, "Error: formatting
numbers\n");
    32                  return 3;
    33          }
    34          send_net(conn, buffer, BUFF);
    35          memset(buffer, 0, BUFF);
    36
    37          recv_net(conn, buffer, BUFF);
    38          uint64_t test_block_encr = atoll(buffer);
    39          uint64_t test_block = rsa_decrypt(test_block_encr, d,
n);
    40          len = sprintf(buffer, "%llu", test_block);
    41          if (len < 0)
    42          {
    43                  fprintf(stderr, "Error: formatting test
block\n");
    44                  return 3;
    45          }
    46          send_net(conn, buffer, BUFF);
    47          memset(buffer, 0, BUFF);
    48
    49          recv_net(conn, buffer, BUFF);
    50          uint64_t se = atoll(buffer);
```

```
51          int i = 0;
52          for(; buffer[i] != ' '; i++);
53          i++;
54          uint64_t sn = atoll(buffer + i);
55          uint64_t test_en = die();
56          uint64_t test_en_out = rsa_encrypt(test_en, se, sn);
57          memset(buffer, 0, BUFF);
58          len = sprintf(buffer, "%llu", test_en_out);
59          if (len < 0)
60          {
61                  fprintf(stderr, "Error: formatting test
block\n");
62                  return 3;
63          }
64          send_net(conn, buffer, BUFF);
65          memset(buffer,0,BUFF);
66          recv_net(conn, buffer, BUFF);
67          uint64_t got_test = atoll(buffer);
68          if(got_test == test_en)
69          {
70                  char* succ = "Successful server
authentication!";
71                  printf("Client says: %s\n", succ);
72          }
73          else
74          {
75                  char* fail = "Server authentication failed!";
76                  printf("Client says: %s\n", fail);
77                  close_net(conn);
78          }
79
80      close_net(conn);
```

```
 81
 82      return 0;
 83  }
```

Файл net.c

```
1  #ifdef __linux__
   2          #include <unistd.h>
   3          #include <arpa/inet.h>
   4  #elif __WIN32
   5          #include <winsock2.h>
   6  #else
   7          #warning "net.h: platform not supported"
   8  #endif
   9
  10  #if defined(__linux__) || defined(__WIN32)
  11
  12  #include <stddef.h>
  13  #include <stdint.h>
  14  #include <stdlib.h>
  15
  16  typedef enum error_t {
  17          WINSOCK_ERR = -1,
  18          SOCKET_ERR  = -2,
  19          SETOPT_ERR  = -3,
  20          PARSE_ERR   = -4,
  21          BIND_ERR    = -5,
  22          LISTEN_ERR  = -6,
  23          CONNECT_ERR = -7,
  24  } error_t;
  25
  26  #include "net.h"
```

```
27
28  static int8_t _parse_address(char *address, char *ipv4, char
*port);
29
30  // 127.0.0.1:8080
31  extern int listen_net(char *address) {
32  #ifdef __WIN32
33          WSADATA wsa;
34          if (WSAStartup(MAKEWORD(2,2), &wsa) != 0) {
35                  return WINSOCK_ERR;
36          }
37  #endif
38          int listener = socket(AF_INET, SOCK_STREAM, 0);
39          if (listener < 0) {
40                  return SOCKET_ERR;
41          }
42  #ifdef __linux__
43          if (setsockopt(listener, SOL_SOCKET, SO_REUSEADDR,
&(int){1}, sizeof(int)) < 0) {
44                  return SETOPT_ERR;
45          }
46  #else
47          if (setsockopt(listener, SOL_SOCKET, SO_REUSEADDR,
&(char){1}, sizeof(char)) < 0) {
48                  return SETOPT_ERR;
49          }
50  #endif
51          char ipv4[16];
52          char port[6];
53          if (_parse_address(address, ipv4, port) != 0) {
54                  return PARSE_ERR;
55          }
```

```
    56          struct sockaddr_in addr;
    57          addr.sin_family = AF_INET;
    58          addr.sin_port = htons(atoi(port));
    59          addr.sin_addr.s_addr = inet_addr(ipv4);
    60          if (bind(listener, (struct sockaddr*)&addr,
sizeof(addr)) != 0) {
    61                  return BIND_ERR;
    62          }
    63          if (listen(listener, SOMAXCONN) != 0) {
    64                  return LISTEN_ERR;
    65          }
    66          return listener;
    67  }
    68
    69  extern int accept_net(int listener) {
    70          return accept(listener, NULL, NULL);
    71  }
    72
    73  extern int connect_net(char *address) {
    74  #ifdef __WIN32
    75          WSADATA wsa;
    76          if (WSAStartup(MAKEWORD(2,2), &wsa) != 0) {
    77                  return WINSOCK_ERR;
    78          }
    79  #endif
    80          int conn = socket(AF_INET, SOCK_STREAM, 0);
    81          if (conn < 0) {
    82                  return SOCKET_ERR;
    83          }
    84          char ipv4[16];
    85          char port[6];
    86          if (_parse_address(address, ipv4, port) != 0) {
```

```
    87                  return PARSE_ERR;
    88          }
    89          struct sockaddr_in addr;
    90          addr.sin_family = AF_INET;
    91          addr.sin_port = htons(atoi(port));
    92          addr.sin_addr.s_addr = inet_addr(ipv4);
    93          if (connect(conn, (struct sockaddr*)&addr,
sizeof(addr)) != 0) {
    94                  return CONNECT_ERR;
    95          }
    96          return conn;
    97  }
    98
    99  extern int close_net(int conn) {
   100  #ifdef __linux__
   101          return close(conn);
   102  #elif __WIN32
   103          return closesocket(conn);
   104  #endif
   105  }
   106
   107  extern int send_net(int conn, char *buffer, size_t size) {
   108          return send(conn, buffer, (int)size, 0);
   109  }
   110
   111  extern int recv_net(int conn, char *buffer, size_t size) {
   112          return recv(conn, buffer, (int)size, 0);
   113  }
   114
   115  static int8_t _parse_address(char *address, char *ipv4, char
*port) {
   116          size_t i = 0, j = 0;
```

```
117         for (; address[i] != ':'; ++i) {
118                 if (address[i] == '\0') {
119                         return 1;
120                 }
121                 if (i >= 15) {
122                         return 2;
123                 }
124                 ipv4[i] = address[i];
125         }
126         ipv4[i] = '\0';
127         for (i += 1; address[i] != '\0'; ++i, ++j) {
128                 if (j >= 5) {
129                         return 3;
130                 }
131                 port[j] = address[i];
132         }
133         port[j] = '\0';
134         return 0;
135 }
136
137 #endif /* defined(__linux__) || defined(__WIN32) */
```

Файл rsa.c

```
1  #include <stdint.h>
   2  #include <stdio.h>
   3  #include <stdlib.h>
   4  #define LZ64 0x00000000FFFFFFFF
   5  #define die() ((rand() % (32768 - 7)) + 7)
   6
   7  uint64_t rsa_prime();
   8  int rsa_miller_rabin_test(uint64_t n, int k);
```

```
 9
10  uint64_t rsa_pow_mod(uint64_t a, uint64_t b, uint64_t c)
11  {
12          uint64_t power = b, d = a, res = 1;
13          while (power) {
14                  if (power & 1)
15                          res = (uint64_t)(res * d) % c;
16                  d = (d * d) % c;
17                  power = power >> 1;
18          }
19          return res % c;
20  }
21
22  uint64_t rsa_gcd(uint64_t u, uint64_t v)
23  {
24          // Обчислення найбільшого спільного дільника
25          // за допомогою двійкового метода Кнута.
26
27          int shift;
28
29          /* нСД(0,v) == v; НСД(u,0) == u, НСД(0,0) == 0 */
30          if (u == 0) return v;
31          if (v == 0) return u;
32
33          /* Нехай shift := lg K, де K - найбільша степінь
двійки, що ділить u і v */
34          for (shift = 0; ((u | v) & 1) == 0; ++shift) {
35                  u >>= 1;
36                  v >>= 1;
37          }
38
39          while ((u & 1) == 0)
```

```
    40                  u >>= 1;
    41
    42          /* Починаючи звідси, u завжди непарне. */
    43          do {
    44                  /* позбудемось всіх дільників 2 в v -- вони
не спільні */
    45                  /*   зауваження: v не 0, тож цикл завершиться
*/
    46                  while ((v & 1) == 0)  /* Loop X */
    47                          v >>= 1;
    48                  /* Тепер u і v - непарні. Якщо потрібно
обміняємо їх, щоб u <= v,
    49                     тоді встановимо v = v - u (парне число).
Для довгих чисел
    50                     обмін - всього переміщення вказівників, і
віднімання
    51                     можна зробити на місці */
    52                  if (u > v) {
    53                          uint64_t t = v; v = u; u = t; //
Обмін u і v.
    54                  }
    55
    56                  v = v - u; // Тут v >= u.
    57          } while (v != 0);
    58
    59          /* Відновлюємо спільні дільники 2 */
    60          return u << shift;
    61  }
    62
    63  uint64_t rsa_gen_E(uint64_t fi) {
    64
    65          uint64_t e;
    66
    67          for (e = 2; e < fi; e++)
```

```
68          {
69                  if (rsa_gcd(e, fi) == 1)
70                  {
71                          return e;
72                  }
73          }
74
75          return -1;
76  }
77
78  uint64_t rsa_D_calculate(uint64_t E, uint64_t fi) {
79
80          // Розширена теорема Евкліда
81          //uint64_t D = 2;
82          //while (((E * D) % fi) != 1) ++D;
83          //return D;
84
85          uint64_t d;
86          uint64_t k = 1;
87
88          while (1)
89          {
90                  k = k + fi;
91
92                  if (k % E == 0)
93                  {
94                          d = (k / E);
95                          return d;
96                  }
97          }
98  }
```

```
 99
100  void rsa_key_gen(uint64_t* E, uint64_t* D, uint64_t* n) {
101          uint64_t p = rsa_prime();
102          uint64_t q = rsa_prime();
103          // Перевіка нерівності p та q
104          while (p == q) { p = rsa_prime(); q = rsa_prime(); }
105          *n = p * q;
106          uint64_t fi = (p - 1) * (q - 1);
107          *E = rsa_gen_E(fi);
108          *D = rsa_D_calculate(*E, fi);
109  }
110
111  uint64_t rsa_encrypt(uint64_t a, uint64_t E, uint64_t n) {
112          uint64_t b = rsa_pow_mod(a, E, n);
113          return b;
114  }
115
116  uint64_t rsa_decrypt(uint64_t b, uint64_t D, uint64_t n) {
117          uint64_t c = rsa_pow_mod(b, D, n);
118          return c;
119  }
120
121  uint64_t rsa_prime() {
122          uint64_t nr = die();
123
124          // Ряд перших простих чисел, використовується для
відсіювання
125          // майже 80% непарних чисел, що не є простими, перш
ніж обробляти їх
126          // алгоритмом Міллера-Рабіна. Але оскільки числа не є
настільки великими,
127          // ми могли б використати звичайний алгоритм Евкліда,
не зважаючи на
```

```
128          // кількість машинних інструкцій.
129
130          uint64_t fewPrimes[] = { 2, 3,  5,      7,      11,
13,    17,      19,     23,     29,
131  31,     37,     41,     43,     47,     53,     59,     61,
67,    71, 73,  79,     83,     89,     97,     101,
132  103, 107, 109, 113, 127, 131, 137, 139, 149, 151, 157, 163,
167,
133  173, 179, 181, 191, 193, 197, 199, 211, 223, 227, 229,
134  233, 239, 241, 251
135          };
136
137          int div = 0;
138          while (1) {
139                  div = 0;
140                  for (uint64_t i = 0; i < sizeof(fewPrimes) /
sizeof(fewPrimes[0]); i++)
141                          if (nr % fewPrimes[i] == 0) div = 1;
142
143                  // 16 циклів перевіки тестом Міллера
144                  if (!div && rsa_miller_rabin_test(nr, 16))
return nr;
145                  nr = die();
146          }
147  }
148
149  int rsa_miller_rabin_test(uint64_t n, int k) {
150          // Тест Міллера-Рабіна є більш ефективним, ніж
151          // звичайний алгоритм Евкліда. З іншої сторони,
152          // точність цього алгоритму не є стопроцентною.
153          // 0 - не є простим числом, 1 - з великою
154          // імовірністю є простим числом.
155
```

```
156         uint64_t t = n - 1;
157         while (t % 2 == 0) { t /= 2; }
158
159         uint64_t a, t1;
160         while (k--) {
161                 a = (rand() % (n - 1 - 2)) + 2;
162                 uint64_t x = rsa_pow_mod(a, t, n);
163                 if (x == 1 || x == n - 1) continue;
164                 t1 = t;
165                 while (1) {
166                         if (t1 == n - 1) break;
167                         if (x == 1) break;
168                         if (x == n - 1) break;
169
170                         x = rsa_pow_mod(x, 2, n);
171                         t1 *= 2;
172                 }
173                 if (x != n - 1 && t1 % 2 == 0)
174                 {
175                         return 0;
176                 }
177         }
178         // Якщо усі етапи перевірки пройдені, число просте.
179
180         return 1;
181 }
```