ФЕДЕРАЛЬНОЕ ГОСУДАРСТВЕННОЕ БЮДЖЕТНОЕ
ОБРАЗОВАТЕЛЬНОЕ УЧРЕЖДЕНИЕ ВЫСШЕГО ОБРАЗОВАНИЯ
«СИБИРСКИЙ ГОСУДАРСТВЕННЫЙ УНИВЕРСИТЕТ
ТЕЛЕКОММУНИКАЦИЙ И ИНФОРМАТИКИ»

РАСЧЕТНО-ГРАФИЧЕСКАЯ РАБОТА

по дисциплине “Защита информации”

на тему

Доказательство с нулевым знанием

Выполнил:

студент гр. ИС-641

Сазонов П. Е.

Проверил:

Ассистент кафедры ПМиК

Петухова Я.В.

Новосибирск – 2019

СОДЕРЖАНИЕ

Оглавление

Описание алгоритма доказательства с нулевым знанием

Доказательство с нулевым знанием (информации) — это интерактивный протокол, позволяющий одной из сторон (проверяющему) убедиться в достоверности какого-либо утверждения, не получив при этом никакой другой информации от второй стороны (доказывающего).

Доказательство с нулевым разглашением должно обладать тремя свойствами:

1. **Полнота**: если утверждение действительно верно, то доказывающий убедит в этом проверяющего.
2. **Корректность**: если утверждение неверно, то даже нечестный доказывающий не сможет убедить проверяющего за исключением пренебрежимо малой вероятности.
3. **Нулевое разглашение**: если утверждение верно, то даже нечестный проверяющий не узнает ничего кроме самого факта, что утверждение верно.

Реализация алгоритма

Алгоритм реализован на языке C, в трех файлах: server – исполняющий роль доказывающего, client – проверяющий и graph_color в котором хранятся вспомогательные функции.

Входные данные: файл graph, в первой строке которого содержатся два числа n и m, количество вершин графа и количество ребер соответственно, в последующих m строках содержится информация о ребрах графа, каждое из которых описывается с помощью двух чисел (номера вершин, соединяемых этим ребром), к данному файлу имеют доступ и доказывающий, и проверяющий; файл color, в котором перечислены цвета вершин графа, к этому файлу имеет доступ только доказывающий.

После запуска исполняемых файлов, между ними создается связь на основе TCP сокетов. Такой выбор был сделан с учетом специфики задачи. Протокол TCP гарантирует доставку пакетов данных в неизменных виде, последовательности и без потерь.

Проверяющий отправляет доказывающему запрос о начале проверки. Доказывающий случайным образом меняет цвета вершин графа, так, чтобы логика раскраски осталась прежней, но цвета вершин изменились. Это делается для выполнения свойства нулевого разглашения: если не менять цвета, при большом количестве проверок, проверяющий сможет узнать раскраску графа, если будет запоминать цвета выбранных вершин, но т.к. цвета меняются случайным образом, информация о раскраске вершин с прошлой итерации устаревает.

Доказывающий создает новый ключ шифра Вернама и с его помощью шифрует массив цветов вершин. Шифр Вернама используется для того, чтобы у каждой вершины был свой уникальный ключ.

Проверяющий получает зашифрованный массив цветов и случайным образом выбирает две вершины, связанные одним ребром. Если проверяющий запросит у доказывающего ключи для этих вершин, нет никакой гарантии, что доказывающий не воспользуется свойством шифра Вернама и не создаст ложные ключи, такие что при расшифровке вершины будут раскрашены в разные цвета, вне зависимости от того, как они были раскрашены в оригинале. Поэтому проверяющий шифрует выбранные вершины своим ключом Вернама и пересылает их доказывающему вместе с номерами выбранных вершин.

Доказывающий расшифровывает полученные вершины своим ключом и отправляет полученную последовательность бит обратно. Проверяющий расшифровывает с помощью своего ключа и выводит на экран номера выбранных вершин и номера цветов, если цвета вершин совпадает об этом сообщается отдельно.

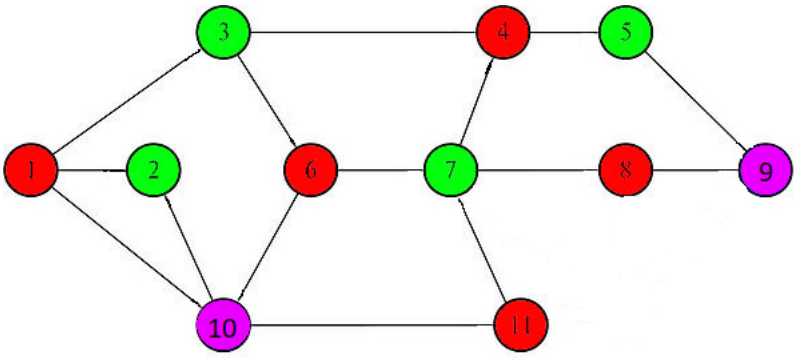
В данном алгоритме существует уязвимость: доказывающий на основе полученного от проверяющего зашифрованного массива цветов, и своего зашифрованного массива, может

вычислить ключ Вернама проверяющего, и вместо расшифровки, сгенерировать такие последовательности бит, которые после исключающего ИЛИ, выполненного проверяющим превратятся в два разных цвета, вне зависимости от оригинальной раскраски.

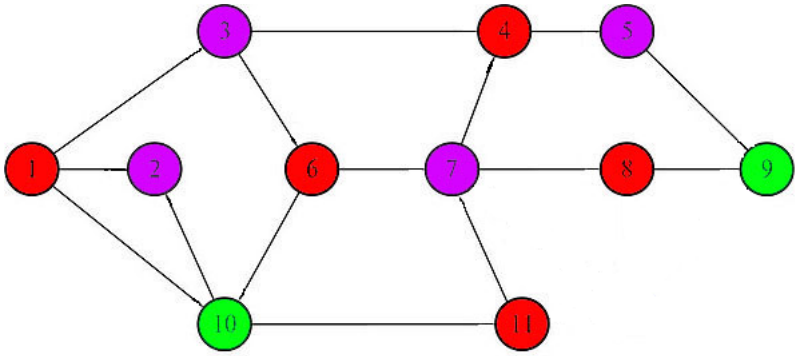
Поэтому в алгоритм добавлена вторая проверка: проверяющий на основе зашифрованных массивов из двух цветов, один из которых отправлен доказывающему, в второй получен от него, находит часть ключа Вернама доказывающего, использовавшуюся для шифрования этих вершин. С помощью этого ключа проверяющий расшифровывает полученные в начале обмена вершины и сравнивает полученные цвета вершин, с результатами первой проверки. Если цвета не совпадают, делается вывод, что сервер подменил свой ключ. Так как для шифрования использовался ключ Вернама, клиент не сможет с помощью найденной части ключа расшифровать все цвета, благодаря чему выполняется свойство нулевого разглашения.

После выполнения алгоритма, проверяющему дается выбор: запустить еще одно проверку или завершить работу, если он удовлетворен результатами проверки или получил доказательства недостоверности раскраски графа.

Демонстрация работы программы



Первоначальная раскраска графа



Раскраска графа после случайной замены цветов

```
encrypted_vertexes:
1189853281 1463073820 718300438 1283298177 356494304 1777253288 2137167018 10911
40235 373023836 2089337660 1781013346
```

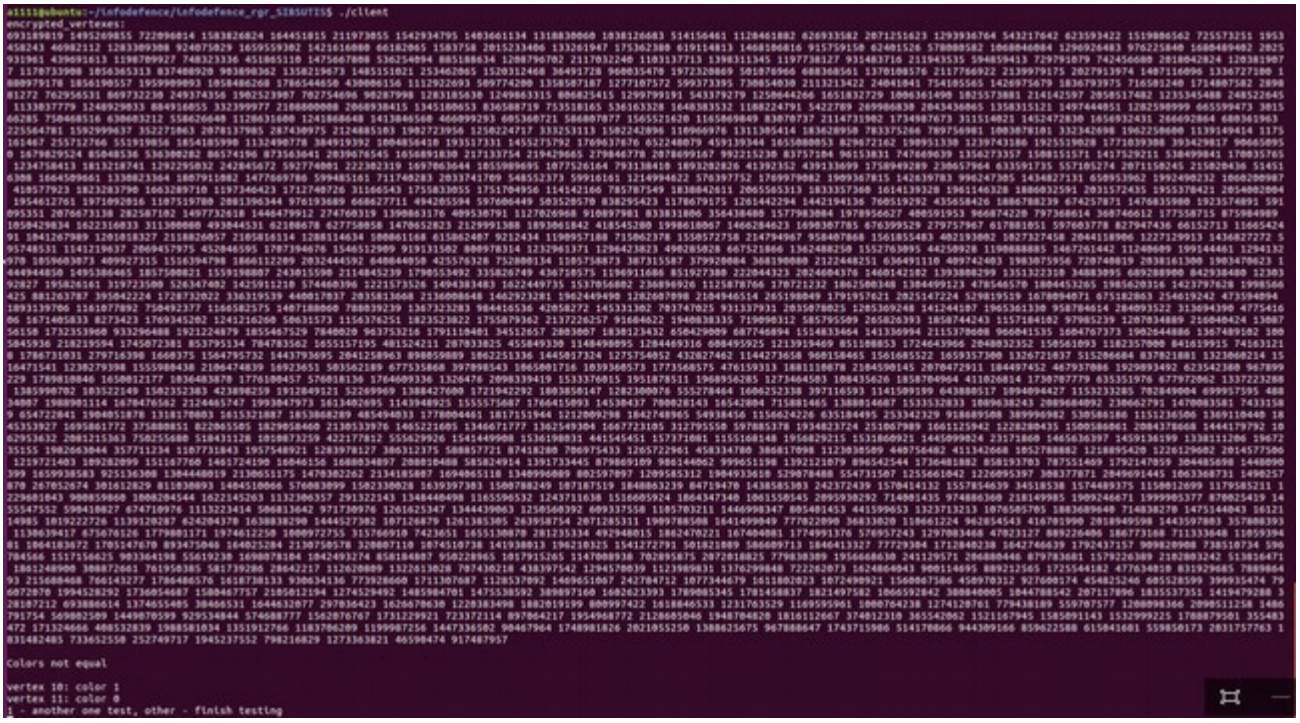
Информация о цветах вершин зашифрованная шифром Вернама

```
a1111@ubuntu:~/infodefence/rgz$ ./client
encrypted_vertexes:
14308849 1692779792 650508622 217054753 1596320874 2141833372 1825611554 5738735
25 492674977 926094374 939754580 910608756

Colors not equal

vertex 6: color 3
vertex 10: color 2
1 - another one test, other - finish testing
1
encrypted_vertexes:
491851980 2021901822 530124523 358407463 1090251774 910955670 751866212 18961347
41 1610377825 1178153113 550178159 546125015

Colors not equal

vertex 7: color 3
vertex 11: color 2
1 - another one test, other - finish testing
```

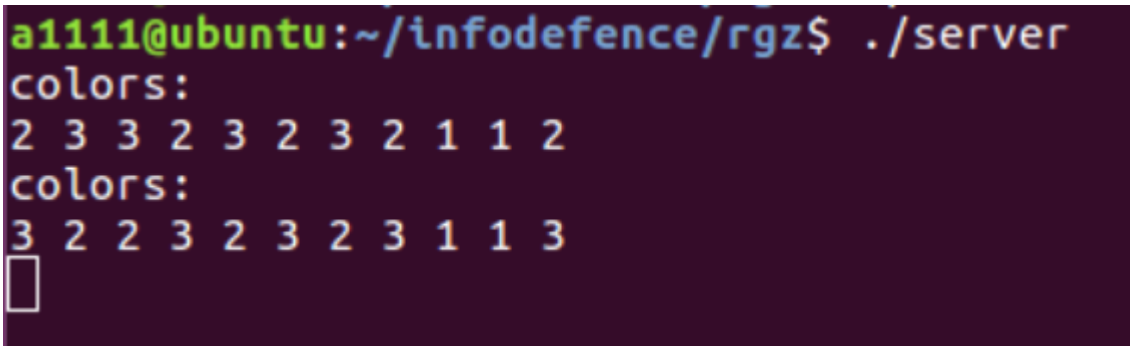
Демонстрация пройденных проверок

```
a1111@ubuntu:~/infodefence/rgz$ ./client
encrypted_vertexes:
1991553115 50254001 190179898 1257395618 1131003049 718926083 614462357 16876978
53 515773186 1724876383 479569444 1487931244

ERROR colors equal!

vertex 2: color 0
vertex 10: color 0
1 - another one test, other - finish testing
```

Демонстрация проваленной проверки



Демонстрация обработки 1000 вершин



Список использованных источников

- Доказательства с нулевым разглашением [Электронный ресурс]// https://neerc.ifmo.ru/wiki/index.php?title=Доказательства_с_нулевым_разглашением
- Рябко Б.Я. Криптография и стенография в информационных технологиях. – Новосибирск: Наука, 2015. - 239 с.

ПРИЛОЖЕНИЕ

Исходный код программы

Client.c

```
#include "graph.h"

int main(int argc, char *argv[])
{
   int i,*encrypted_vertexes, n, m, command = 1, **A;
   int sock, vertexes_for_test[2], vertexes_for_test_encrypt[2], key[2], vertexes_for_test_num[2];
   int vertexes_for_test_server_decrypt[2], *server_key, *vertexes_decrypt,
*vertexes_decrypt_2;
   struct sockaddr_in addr;
   char *end;
   char buf[2];
   sock = socket(AF_INET, SOCK_STREAM, 0);
   if(sock < 0)
   {
      perror("sock");
      exit(-1);
   }
   addr.sin_family = AF_INET;
   addr.sin_port = htons(3429);
   addr.sin_addr.s_addr = htonl(INADDR_LOOPBACK);

   A = load_graph(&m, &n, "tmp/graph");
   encrypted_vertexes = malloc(n * sizeof(int));
   if(connect(sock, (struct sockaddr *)&addr, sizeof(addr)) < 0)
   {
      perror("connect");
      exit(-1);
   }
   while(1)
   {
      send(sock, &command, sizeof(int), 0);
      if(command != 1)
      {
         break;
      }

     recv(sock, encrypted_vertexes, sizeof(int)*n, 0);
      generate_two_different_numbers(vertexes_for_test_num, n, A);
      printf("encrypted_vertexes:\n");
      for(i = 0; i < n; i++)
         printf("%d ", encrypted_vertexes[i]);
      printf("\n");
      for(i = 0; i < 2; i++)
         vertexes_for_test[i] = encrypted_vertexes[vertexes_for_test_num[i]];
```

```
        vernam_encrypt(2, vertexes_for_test, vertexes_for_test_encrypt, key);
        send(sock, vertexes_for_test_num, sizeof(int)*2, 0);
        send(sock, vertexes_for_test_encrypt, sizeof(int)*2, 0);
        recv(sock, vertexes_for_test_server_decrypt, sizeof(int)*2, 0);
        server_key = vernam_decrypt(2, vertexes_for_test_encrypt,
vertexes_for_test_server_decrypt);
        vertexes_decrypt = vernam_decrypt(2, vertexes_for_test_server_decrypt, key);
        if(vertexes_decrypt[0] == vertexes_decrypt[1])                //first test
        {
           printf("\nERROR colors equal!\n\n");
        }else
          printf("\nColors not equal\n\n");
        vertexes_decrypt_2 = vernam_decrypt(2, vertexes_for_test, server_key);          //second
test
        for(i = 0; i < 2; i++)
        {
          if(vertexes_decrypt_2[i] != vertexes_decrypt[i])
               printf("ERROR server key changed!\n");
        }
        printf("vertex %d: color %d\nvertex %d: color %d\n", vertexes_for_test_num[0],
vertexes_decrypt[0]-1, vertexes_for_test_num[1], vertexes_decrypt[1]-1);
        printf("1 - another one test, other - finish testing\n");
        do
        {
           if(!fgets(buf, sizeof(buf), stdin))
               break;
           command = strtol(buf, &end, 10);
        }while(end != buf + strlen(buf));
    }
    for(i = 0; i < n; i++)
        free(A[i]);
    free(A);
    free(encrypted_vertexes);
    close(sock);
    exit(0);
}
```

Server.c

```
#include "graph.h"

int main()
{
  int command, sock, child_sock, n, num_of_colors, *vertexes, *encrypted_vertexes, *key;
  int vertexes_for_test[2], vertexes_for_test_num[2], i;
  struct sockaddr_in addr, child;
  socklen_t size = sizeof(child);

  load_graph_size(&n, "tmp/graph");
  vertexes = load_colored(&num_of_colors, n, "tmp/graph2");     //array of vertex colors
  encrypted_vertexes = malloc(n * sizeof(int));
  key = malloc(n * sizeof(int));

  sock = socket(AF_INET, SOCK_STREAM, 0);   //tcp socket for communication
  if(sock < 0)
  {
```

```
    perror("socket");
    exit(-1);
  }
  addr.sin_family = AF_INET;
  addr.sin_port = htons(3429);
  addr.sin_addr.s_addr = htonl(INADDR_LOOPBACK);
  if(bind(sock, (struct sockaddr*)&addr, sizeof(addr)) < 0)
  {
    perror("bind");
    exit(-1);
  }
  listen(sock, 6);
  child_sock = accept(sock, (struct sockaddr*)&child, &size);
  while(1)
  {
    recv(child_sock, &command, sizeof(command), 0);
    if(command == 1)
    {
      relabeling(n, num_of_colors, vertexes);
      printf("colors:\n");
      for(i = 0; i < n; i++)
        printf("%d ", vertexes[i]);
      printf("\n");
      vernam_encrypt(n, vertexes, encrypted_vertexes, key);
      send(child_sock, encrypted_vertexes, sizeof(int)*n, 0);
      recv(child_sock, vertexes_for_test_num, sizeof(int)*2, 0);
//numbers of two vertexs for test
      recv(child_sock, vertexes_for_test, sizeof(int)*2, 0);
//values of two vertexs for test
      vernam_part_decrypt(2, vertexes_for_test, vertexes_for_test_num, key);    //decrypt two
vertexes for test
      send(child_sock, vertexes_for_test, sizeof(int)*2, 0);
    }else
      break;
  }
  free(vertexes);
  free(key);
  free(encrypted_vertexes);
  close(sock);
  exit(0);
}
```

Graph_coloring.c

```
#include "graph.h"

void generate_two_different_numbers(int *vertexes_for_test_num, int n, int **A)
{
  while(1)
  {
    randombytes(&vertexes_for_test_num[0], sizeof(vertexes_for_test_num[0]));
    vertexes_for_test_num[0] = fabs(vertexes_for_test_num[0] % n);
    randombytes(&vertexes_for_test_num[1], sizeof(vertexes_for_test_num[1]));
    vertexes_for_test_num[1] = fabs(vertexes_for_test_num[1] % n);
    if(vertexes_for_test_num[0] != vertexes_for_test_num[1] &&
       is_adjacent(A[vertexes_for_test_num[0]][vertexes_for_test_num[1]]))
      break;
```

```
  }
}

int* load_colored(int *num_of_colors, int n, char *filename)
{
  int i, *vertexes;
  vertexes = malloc(n * sizeof(int));
  FILE *fout = file_open(filename, "r");
  *num_of_colors = 0;
  for(i = 0; i < n; i++){
    fscanf(fout, "%d ", &vertexes[i]);
    *num_of_colors = MAX(vertexes[i], *num_of_colors);
  }
  fclose(fout);
  return vertexes;
}

void load_graph_size(int *n, char *filename)
{
  FILE *fout = file_open(filename, "r");
  int m;
  fscanf(fout, "%d %d", n, &m);
  fclose(fout);
}

void relabeling(int n, int num_of_colors, int *vertexes)
{
  int i = 0, flag, j, tmp, colors[num_of_colors];
  for(; i < num_of_colors; i++)
  {
    flag = 1;
    randombytes(&tmp, sizeof(tmp));
    tmp = fabs(tmp % num_of_colors) + 1;
    while(flag)
    {
      flag = 0;
      for(j = 0; j < i; j++)
        if(colors[j] == tmp)
        {
          flag = 1;
          break;
        }
      if(flag)
      {
        tmp = (tmp + 1) % num_of_colors;
        if(tmp == 0)
          tmp = num_of_colors;
      }
    }
    colors[i] = tmp;
  }
  for(i = 0; i < n; i++)
  {
    vertexes[i] = colors[vertexes[i] - 1];
  }
}

void save_colored(int *colored, int n, char *filename)
```

```
{
  int i;
  FILE *fout = file_open(filename, "w");
  for(i = 0; i < n; i++)
    fprintf(fout, "%d ", colored[i]);
  fclose(fout);
}
int **load_graph(int *m, int *n, char *filename)
{
  FILE *fout = file_open(filename, "r");
  int **A;
  int i = 0, vertex1, vertex2;
  fscanf(fout, "%d %d", n, m);
  if(*n > 1001 || *m > *n * *n)
  {
    fprintf(stderr, "n must be < 1001 and m must be <= n^2\n");
    exit(-1);
  }

  A = malloc(*n * sizeof(int*));
  for(; i < *n;i++)
    A[i] = calloc(*n, sizeof(int));
  for(i = 0; i < *m; i++)
  {
    fscanf(fout, "%d %d", &vertex1, &vertex2);
    A[vertex1][vertex2] = 1;
  A[vertex2][vertex1] = 1;
  }
  fclose(fout);
  return A;
}

inline int is_colored(int num)
{
  if(num != 0)
  return 1;
  return 0;
}

inline int is_adjacent(int num)
{
  if(num != 0)
    return 1;
  return 0;
}

int can_be_colored(int **A,  int *colored, int n, int num, int color)
{
  int i;
  for(i = 0; i < n; i++)
    if(colored[i] == color && is_adjacent(A[num][i]))
      return 0;
  return 1;
}
```

Graph.h

```
#ifndef graph_h
#define graph_h

#include <stdio.h>
#include <malloc.h>
#include <stdlib.h>
#include <unistd.h>
#include <string.h>
#include <sys/types.h>
#include <sys/socket.h>
#include <arpa/inet.h>
#include "randombytes.h"
#include "infodef.h"
#include "ciphers.h"

#define MAX(a,b) (((a)>(b))?(a):(b))

void save_colored(int *colored, int n, char *filename);
int **load_graph(int *m, int *n, char *filename);
extern int is_colored(int num);
extern int is_adjacent(int num);
int can_be_colored(int **A,  int *colored, int n, int num, int color);
void relabeling(int n, int num_of_colors, int *vertexes);
void load_graph_size(int *n, char *filename);
int* load_colored(int *num_of_colors, int n, char *filename);
void generate_two_different_numbers(int *vertexes_for_test_num, int n, int **A);
#endif
```