ФЕДЕРАЛЬНОЕ ГОСУДАРСТВЕННОЕ БЮДЖЕТНОЕ ОБРАЗОВАТЕЛЬНОЕ УЧРЕЖДЕНИЕ ВЫСШЕГО ОБРАЗОВАНИЯ

«СИБИРСКИЙ ГОСУДАРСТВЕННЫЙ УНИВЕРСИТЕТ ТЕЛЕКОММУНИКАЦИЙ И ИНФОРМАТИКИ»

**РАСЧЁТНО-ГРАФИЧЕСКОЕ ЗАДАНИЕ**

по дисциплине “Защита информации” на тему

**Доказательство с нулевым знанием**

Вариант №1

|  |  |
| --- | --- |
| Выполнил студент | Гурулев Дмитрий Александрович |
|  | Ф.И.О. |

|  |  |
| --- | --- |
| Группы | ИВ – 621 |

|  |  |  |
| --- | --- | --- |
| Работу принял |  | ассистент кафедры ПМиК Я. В. Петухова |
| подпись | | |

|  |  |
| --- | --- |
| Защищена | Оценка |

Новосибирск – 2019

**СОДЕРЖАНИЕ**

[ПОСТАНОВКА ЗАДАЧИ 3](#_bookmark0)

[ТЕОРЕТИЧЕСКИЕ СВЕДЕНИЯ 4](#_bookmark1)

[ДЕМОНСТРАЦИЯ 5](#_TOC_250000)

[ПРИЛОЖЕНИЕ 7](#_bookmark4)

ПОСТАНОВКА ЗАДАЧИ

В рамках расчётно-графического задания необходимо разработать программу, реализующую протокол доказательства с нулевым знанием для задачи «Раскраска графа».

Абонент Алиса, связывающаяся с другим абонентом Бобом, перед доказательством, должен иметь связный граф с раскрашенными вершинами. Раскраска считается правильной, если любые смежные вершины имели разные цвета. Алисе необходимо доказать Бобу, что раскраска графа правильная, без раскрытия сведений о раскраске Бобу.

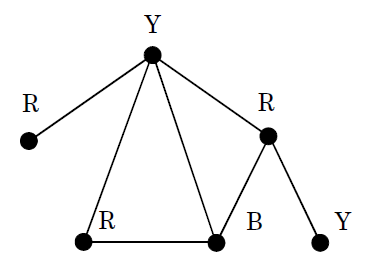
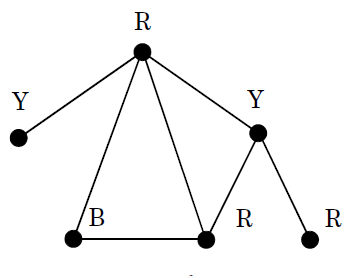
 

Рис.1 Правильная раскраска Рис.2 Неправильная раскраска

Перед реализацией протокола необходимо реализовать дополнительную программу для генерации графа. Граф считываться с 2 файлов в определенном формате:

1. Файл содержит информацию о вершинах графа (ID и Цвет)
2. Файл содержит информацию о ребрах графа

Для реализации протокола доказательства используются параметры RSA для всех вершин, которые передаются Бобу по верифицированному каналу. В работе должны быть использованы такие процедуры, как быстрое возведение в степень, расширенный алгоритм Евклида, а также генерация больших простых чисел.

ТЕОРЕТИЧЕСКИЕ СВЕДЕНИЯ

1. Алиса выбирает случайную перестановку имеющихся цветов и перекрашивает граф. Это не меняет корректность графа, так смежность вершин не меняется.

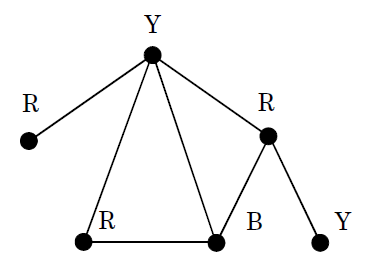
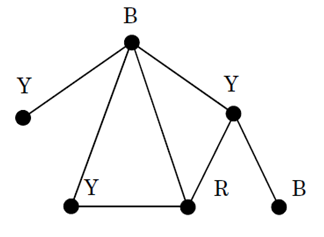
 

Рис.3 Исходный граф Рис.4 Граф после перестановки

1. Алиса для каждой вершины графа генерирует случайной большое число *r* и заменяет в нем младшие разряды кодом цвета. Например: 00 – R (красный), 01 – B (синий), 10 – Y (жёлтый).
2. Алиса для каждой вершины графа формирует данные из системы RSA: , и .
3. Алиса вычисляет

и посылает Бобу публичные ключи для каждой вершины графа.

1. Боб выбирает случайное ребро графа и сообщает его Алисе. В ответ Алиса передает Бобу ключи и . Боб вычисляет

и

и сравнивает младшие разряды, соответствующие кодам цвета вершины. Если значения совпали, то Алиса обманывает Боба, на этом этапе все заканчивается. Если нет, то возвращаемся на *Шаг 1*, до тех пор, пока все ребра графа не будут проверены.

ДЕМОНСТРАЦИЯ

Визуализация перекраски графа из 5 вершин и 9 ребер.

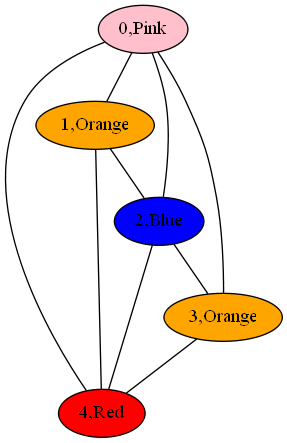
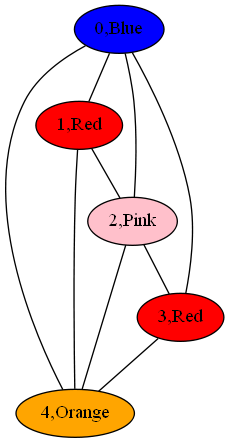
 

Рис. 5 Граф исходный Рис. 6 Граф после перекраски

Сохраненный граф в файлах nodes.json и edjes.json с разметкой JSON.

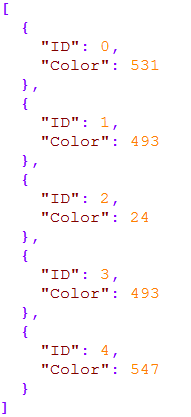


Рис. 7 Файл nodes.json



Рис. 8 Файл edjes.json

Демонстрация работы программы

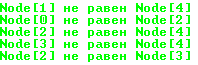


Рис. 9 Вывод программы

Демонстрация работы программы на большом графе

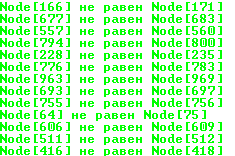


Рис. 10 Фрагмент вывода программы для графа из 1000 вершин и 5961 ребер.

Демонстрация неудачного выполнения программы на большом графе

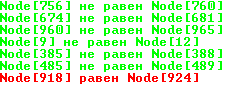


Рис. 11 Фрагмент вывода программы для графа из 1000 вершин и 5972 ребер. Неудачное выполнение.

ПРИЛОЖЕНИЕ

Класс Node:

class Node : RSAUser

{

[JsonProperty("ID")]

public long labeln;

[JsonIgnore]

public long z;

[JsonProperty("Color")]

public DotColor colorn;

[JsonIgnore]

public int adj;

public Node(DotColor color, long label) : base() { colorn = color; labeln = label; adj = 0; }

public override string ToString()

{

string n = String.Format("Label = {0:d}, Color = {1:d}, z = {2:d}, c = {3:d}, d = {4:d}", labeln, colorn, z, c, d);

return n;

}

}

Класс Graph:

class Graph

{

static RNGCryptoServiceProvider Rand = new RNGCryptoServiceProvider();

static long Randomizer(int min, int max)

{

uint scale = uint.MaxValue;

while (scale == uint.MaxValue)

{

byte[] four\_bytes = new byte[8];

Rand.GetBytes(four\_bytes);

scale = BitConverter.ToUInt32(four\_bytes, 0);

}

return (int)(min + (max - min) \* (scale / (double)uint.MaxValue));

}

public static string ReverseString(string s)

{

char[] arr = s.ToCharArray();

Array.Reverse(arr);

return new string(arr);

}

static DotColor[] shuffle(DotColor[] arr)

{

Random random = new Random();

List<KeyValuePair<int, DotColor>> list =

new List<KeyValuePair<int, DotColor>>();

foreach (DotColor s in arr)

{

list.Add(new KeyValuePair<int, DotColor>(random.Next(), s));

}

var sorted = from item in list

orderby item.Key

select item;

DotColor[] result = new DotColor[arr.Length];

int index = 0;

foreach (KeyValuePair<int, DotColor> pair in sorted)

{

result[index] = pair.Value;

index++;

}

return result;

}

public DotColor[] colors = { DotColor.Blue, DotColor.Orange, DotColor.Red, DotColor.Pink, DotColor.Green};

public List<Node> nodes;

public List<string> edges;

public long no\_nodes, no\_edges;

public Graph(long no\_nodess) { no\_nodes = no\_nodess; no\_edges = 0; nodes = new List<Node>(); edges = new List<string>(); }

public void generate()

{

create\_nodes();

create\_edges();

}

public void create\_nodes()

{

for(int i = 0; i < no\_nodes; i++)

{

nodes.Add(new Node(colors[Randomizer(0,colors.Length)], i));

}

}

public void create\_edges()

{

for (int i = 0; i < no\_nodes; i++)

{

for (int j = i; j < no\_nodes; j++)

{

if (nodes[i].colorn != nodes[j].colorn && nodes[i].adj <= 5)

{

nodes[i].adj++;

string[] v = { Convert.ToString(nodes[i].labeln), Convert.ToString(nodes[j].labeln) };

edges.Add(String.Join(":", v));

}

}

}

}

public void shuffle\_colors()

{

DotColor[] new\_colors = shuffle(colors);

Dictionary<DotColor, DotColor> colors\_h = new Dictionary<DotColor, DotColor>(5);

var old\_new = colors.Zip(new\_colors, (first, last) => new {first, last});

foreach(var c in old\_new)

{

colors\_h.Add(c.first, c.last);

}

List<Node> new\_nodes = new List<Node>();

foreach(var n in nodes)

{

DotColor c;

colors\_h.TryGetValue(n.colorn, out c);

new\_nodes.Add(new Node(c, n.labeln));

}

nodes = new\_nodes;

}

public void save()

{

string output1 = JsonConvert.SerializeObject(nodes, Formatting.Indented);

File.WriteAllText(@"C:\nodes.json", output1);

string output2 = JsonConvert.SerializeObject(edges);

File.WriteAllText(@"C:\edges.json", output2);

}

public void load()

{

string json1 = File.ReadAllText(@"C:\nodes.json");

nodes = JsonConvert.DeserializeObject<List<Node>>(json1);

string json2 = File.ReadAllText(@"C:\edges.json");

edges = JsonConvert.DeserializeObject<List<string>>(json2);

}

public void render(string file)

{

var dGraph = new DotGraph("dGraph", true);

for (int i = 0; i < nodes.Count; i++)

{

DotNode myNode = new DotNode($"{i}")

{

Shape = DotNodeShape.Ellipse,

Label = $"{nodes[i].labeln}",

FillColor = nodes[i].colorn,

FontColor = DotColor.White,

Style = DotNodeStyle.Filled,

Height = 0.5f

};

dGraph.Add(myNode);

}

for (int i = 0; i < edges.Count; i++)

{

DotArrow myArrow = new DotArrow(edges[i].Split(':')[0], edges[i].Split(':')[1])

{

ArrowHeadShape = DotArrowShape.None

};

dGraph.Add(myArrow);

}

var dot = dGraph.Compile();

File.WriteAllText(@"C:\Program Files (x86)\Graphviz2.38\bin\" + file, dot);

}

}

Класс RSAUser:

class RSAUser

{

[JsonIgnore]

public long p, q, N, d, c;

public RSAUser() {

key\_init();

}

static RNGCryptoServiceProvider Rand = new RNGCryptoServiceProvider();

static long Randomizer(int min, int max)

{

uint scale = uint.MaxValue;

while (scale == uint.MaxValue)

{

byte[] four\_bytes = new byte[8];

Rand.GetBytes(four\_bytes);

scale = BitConverter.ToUInt32(four\_bytes, 0);

}

return (int)(min + (max - min) \* (scale / (double)uint.MaxValue));

}

public static bool IsPrime(long num)

{

if (num == 2) return true;

if (num <= 1 || num % 2 == 0) return false;

var bound = (long)Math.Floor(Math.Sqrt(num));

for (long i = 3; i <= bound; i += 2)

{

if (num % i == 0) return false;

}

return true;

}

public void key\_init()

{

int MAX = Convert.ToInt32(1e4);

do

{

q = Randomizer(0, MAX);

p = 2 \* q - 1;

} while (!IsPrime(q) || !IsPrime(p));

N = p \* q;

long phi = (p - 1) \* (q - 1);

//(c \* d) mod (phi) = 1

do

{

d = Randomizer(0, MAX);

Lab1 l = new Lab1();

long[] hsh = l.evklid(d, phi);

if (hsh[0] == 1)

{

c = (hsh[1] % phi + phi) % phi;

break;

}

}

while (true);

}

}

Класс Client:

class Client

{

static RNGCryptoServiceProvider Rand = new RNGCryptoServiceProvider();

static long Randomizer(int min, int max)

{

uint scale = uint.MaxValue;

while (scale == uint.MaxValue)

{

byte[] four\_bytes = new byte[8];

Rand.GetBytes(four\_bytes);

scale = BitConverter.ToUInt32(four\_bytes, 0);

}

return (int)(min + (max - min) \* (scale / (double)uint.MaxValue));

}

public static bool IsPrime(long num)

{

if (num == 2) return true;

if (num <= 1 || num % 2 == 0) return false;

var bound = (long)Math.Floor(Math.Sqrt(num));

for (long i = 3; i <= bound; i += 2)

{

if (num % i == 0) return false;

}

return true;

}

public Graph gr;

public Client(Graph graph) { gr = graph; }

public void next()

{

int MAX = Convert.ToInt32(1e4);

Lab1 l = new Lab1();

gr.shuffle\_colors();

foreach (var n in gr.nodes)

{

long r = (Randomizer(0, MAX) & ~(0x7)) | (long)n.colorn;

long z = Convert.ToInt64(l.modexp((ulong)r, (ulong)n.d, (ulong) n.N));

n.z = z;

}

}

}

Класс Server:

class Server

{

public string c\_edge;

public bool checking = false;

public Graph gr;

public Server(Graph graph) { gr = graph; }

static RNGCryptoServiceProvider Rand = new RNGCryptoServiceProvider();

static long Randomizer(int min, int max)

{

uint scale = uint.MaxValue;

while (scale == uint.MaxValue)

{

byte[] four\_bytes = new byte[8];

Rand.GetBytes(four\_bytes);

scale = BitConverter.ToUInt32(four\_bytes, 0);

}

return (int)(min + (max - min) \* (scale / (double)uint.MaxValue));

}

public void choose\_edge()

{

c\_edge = gr.edges[(int)Randomizer(0, gr.edges.Count)];

}

public void check()

{

Lab1 l = new Lab1();

int left = Convert.ToInt32(c\_edge.Split(':')[0]);

int right = Convert.ToInt32(c\_edge.Split(':')[1]);

long lval = Convert.ToInt64(l.modexp((ulong)gr.nodes[left].z, (ulong)gr.nodes[left].c, (ulong)gr.nodes[left].N));

long rval = Convert.ToInt64(l.modexp((ulong)gr.nodes[right].z, (ulong)gr.nodes[right].c, (ulong)gr.nodes[right].N));

long l3 = lval & 7;

long r3 = rval & 7;

if(l3 != r3)

{

Console.ForegroundColor = ConsoleColor.Green;

Console.WriteLine($"Node[{left}] не равен Node[{right}]");

Console.ResetColor();

int numidx = Array.IndexOf(gr.edges.ToArray(), c\_edge);

if (numidx != -1)

{

gr.edges.RemoveAt(numidx);

}

}

else

{

Console.ForegroundColor = ConsoleColor.Red;

Console.WriteLine($"Node[{left}] равен Node[{right}]");

Console.ResetColor();

checking = true;

}

}

}

Класс Program:

class Program

{

static void Main(string[] args)

{

var gr = new Graph(1000);

var alice = new Client(gr);

var bob = new Server(gr);

gr.generate();

gr.render("g.dot");

gr.save();

for(int i = 0; i < bob.gr.edges.Count; i++)

{

alice.next();

bob.choose\_edge();

bob.check();

if (bob.checking) break;

}

gr.render("g1.dot");

Console.ReadLine();

}

}

}