

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.....	3
Реализация алгоритма.....	4
Демонстрация работы программы.....	6
Список использованных источников.....	9
ПРИЛОЖЕНИЕ.....	10
Исходный код программы.....	10

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 неизменном 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 совпадают об этом сообщается отдельно.

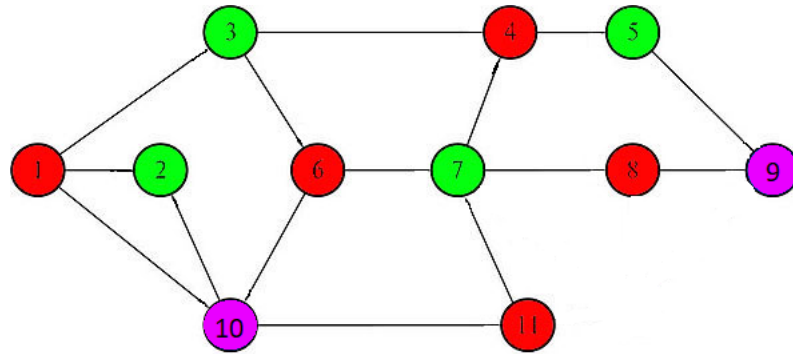
В данном алгоритме существует уязвимость: доказывающий на основе полученного от проверяющего зашифрованного массива цветов, и своего зашифрованного массива, может

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.

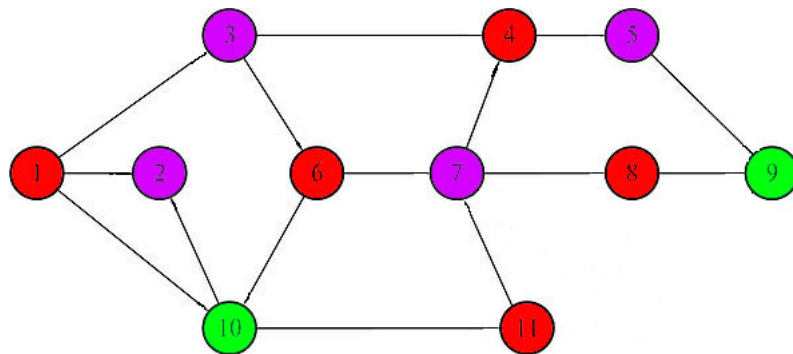
Поэтому в алгоритм добавлена вторая проверка: проверяющий на основе зашифрованных массивов из двух цветов, один из которых отправлен доказывающему, в второй получен от него, находит часть ключа Вернама доказывающего, использовавшуюся для шифрования этих вершин. С помощью этого ключа проверяющий расшифровывает 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
█

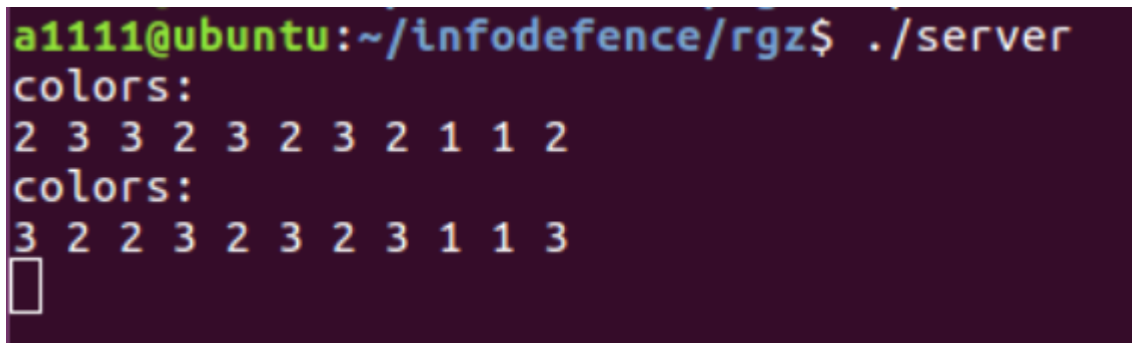
```

Демонстрация проваленной проверки

```
a1111@ubuntu:~/infodefence/infodefence_rgr_328507355_0_client
encrypted vertices:
0913198919 149520955 722896614 1583266824 644451815 211973855 1542934795 1483661134 1318838060 1938126683 514156461 1128461882 626933582 2071251623 1293936704 543217642 623593422 15198006562 725573255 1953
428243 460821212 1281389388 924075029 1659559382 1421616888 66818265 1583758 2015233466 113261947 37563260 61914833 1460584816 915759158 45401528 378888582 1660646484 279642483 976252448 1608464842 2025
931963 436695133 1198799627 746323336 451865519 1475667888 536254894 885186434 1268196782 2117032546 1183137713 1388313345 1197378327 931483716 213843555 584459413 729789879 742456688 2818642824 128381907
7 118733988 1056365313 837488928 963888362 1358213673 1485151021 253462885 1520312468 36491728 966035478 1972328869 581878966 688368561 1370188576 2117766922 2139979175 20279313974 1487116696 133672780 1
888979178 1856198557 1959999993 1054846268 1786544658 438086156 1112922693 589774288 1358587187 1271787572 599373395 730854830 2133213422 24041841 754105565 1426975679 1836793975 716581240 1748807582 2898
81272 762955531 609732236 249174356 1802527397 782754896 96287998 128318538 124866315 886622415 1129797913 541379279 1258642466 1651832729 1896361498 1815573881 2141432781 289513748 148322545
1333832779 124892953 884916855 352399977 1786888888 2648938415 1345106653 836588719 73518165 1385807132 1464838332 1186224791 5422789 249966438 2843436645 1358315121 1487444851 1358315121 487444851 655199471 3015
66281 738486516 638683212 158626648 1128631688 1241868648 1413846568 466899293 685368721 588887077 1565521620 1165896849 83870737 2114731982 1734987673 311514821 1454272830 1656932431 2866928264 688361983
125264781 1592999637 1522713863 287813985 287438975 1214885183 1982733956 1582224717 133253113 1582428898 1189690678 1311385614 183628938 783375268 789758684 1880878181 323242888 1586258688 1139149414 1173
161467 253173786 55319816 185418598 1132488778 384519392 1880458418 1935173314 1453273792 1706813738 653248879 459139364 1651888853 819872162 188931338 1239143186 1925118828 1772189386 393429817 96645895
0 1678292524 85848536 136388282 86574196 873563645 2839676545 1658891838 217333754 219428665 379846378 5078499167 982215238 83757184 96153811 747689639 1356273357 1588155171 1457529211 538698841 178085763
123475843 181321874 129229582 242856572 59274881 22282319 1697863644 3655888895 107525166 793118418 1693282826 41932152 43913699 1758949289 238657964 63591735 557185274 2071158245 215828436 55451
6388 1664599665 1338823364 1887911882 1477669788 599485101 711748283 2033741789 548552373 59916161 1214946422 576337572 1769979882 1909367815 142639783 996247385 1434827331 658953962 1992480232 1860280987
418877923 182323796 1663288719 187346423 1712748726 21166453 1755833051 1751784956 114452166 785787549 183884241 2885565313 1833357369 1614139328 5981146328 1886632591 2031572435 1955378421 2654882684
1954617261 1971892816 1107515788 2881366344 976193689 668627711 494265594 597666445 503528570 838295423 1178079315 1261442284 1442194136 780519292 435658426 188678829 874257871 1476835888 1923574891 591
895351 287667318 282587182 1497732618 1446479912 274768319 1598863176 409538791 112782698 910897981 833811886 356418488 1577983884 1978956627 408519353 966874228 797386614 368746612 177558715 87584989
1858429384 1623110833 131388888 493844531 42186678 627758858 147852823 212991388 1893861842 418545288 1998618867 1466284623 1698387785 676399529 279757867 617881051 597803778 827947436 66152713 11645424
91 1841267989 920183827 2118826957 2185816314 1288114634 588641168 618862487 921214 1190957188 715862378 1350572758 214794867 958487888 1561851483 402139882 182732748 2844118966 1227259913 1436827272 5
95748531 1114213837 2868457975 452864855 1787386678 1546512889 919131582 888978314 1212964317 1296423823 490265828 665721886 1562488258 1552763895 46258928 1198888881 1467261442 11248869 199144611 1294132
978 1858368873 409927315 1516394798 1866112289 2032444592 648844958 425578328 752888914 1185738873 387315587 379928884 388188888 2122448251 636491110 489474203 1983875956 728748819 203811388 1903478623 1
444944858 1495186485 1837588821 1559188887 243815598 2114845239 1798553492 335828748 36718573 159691688 851927388 222846432 2824684376 1460142182 193388839 1351322318 34883895 689288988 842938488 72603
92827 958426161 139723198 526347082 1425811218 174468396 1221573526 1494483483 1622448715 1537856882 238898926 1123878766 778721212 1882588348 1384499123 478146678 1884453265 19845629316 14237797628 1998566
425 861263787 395842224 1728732822 136319539 448017837 2035813688 2136888848 14662921281 1962449498 1282687988 2189446514 265188849 1791957621 2025147224 578819519 1678848071 675182863 254619242 473584884
189313788 1161077892 758492277 1164582575 1487188888 788839239 1367323283 984416536 42058272 145331382 783747025 931337931 2035878825 1285369248 141244587 1965361338 958786614 284893522 333694398 4775416
66 1671485813 8273423 1769345182 124116268 50631579 1156376251 1823522822 1755879362 2117228257 91848622 146083835 718998912 585795589 265826393 1858744143 1557184182 97982329 1287834888 2168464624 15087
56158 1732353868 93126488 1921224879 1855467529 78484826 96133216 1791114841 34512657 28833887 1838123432 458429889 487744884 1554833664 141338994 2115378688 966445155 368477373 1902444886 1367489182 188
5845936 218219584 174502781 857359134 784783562 1655517195 481524211 287833825 455489339 1144898895 1284469316 688495925 1213919469 851188853 1724643966 2648832352 158618093 1812357888 164159195 71463321
0 1786731831 279716398 16688735 1564795732 1443736895 2841258963 888858889 1862251336 1444573324 127574852 418274682 1144273858 968158465 1561885522 165935788 1326721837 1552866884 837821881 1323688214 15
16471541 1238273588 1555884834 2186474839 149236551 583562189 67751588 397894843 1865887176 1839386573 1771568575 476159313 188718878 2164598145 2878472911 184497452 467937884 1929893492 621542188 367899
229 1788818846 165881277 1836463878 1778198457 576818136 3746469136 1326478 2898338418 1533176015 3931878311 1984856283 1274464593 188435626 1858784884 41826814 1728778779 83551978 67972862 133723288
1385998782 183522149 1582352383 42388829 1912849121 3228988 1388427800 1721842282 1883858147 1282388878 555278484 1686252338 397116593 1141589195 643888157 1894894427 1153232285 782848884 499957595 488
34887 1588881114 1678476562 2124665747 1993847973 1613482895 1413948923 1555575887 1766415893 145384737 788828143 16542884 713881485 1388814687 1533532833 1888638243 2888648892 2386642791 1478883181 7433158
9 654722845 188481878 118178883 561321887 1853888889 48548483 1278888865 1817531844 121288888 1842748885 5893816 1156624226 635184895 253323239 9188888 389988882 13858888 1151261586 1384918448 18
43353927 1898461772 1738888851 822888588 1878888846 2138513976 1485221685 1346671777 13621489384 166723185 13795558 397888378 1784823784 251887888 1681125942 1228288435 1588888861 288437868 144417992 12
62953612 588125363 7582515688 518431128 1888873259 422177812 555629926 1584549988 1536198831 46154551 157371881 1155168184 1956829215 1531868921 1464898824 2317888 14656361397 145913619 133811288 19672
35155 1582683844 357713218 158771843 1957548821 1283978127 386312375 588857721 874188831 396218325 1265722661 458314788 388817098 1123838589 448756482 411347688 1851788882 1218895428 1226129682 2014577586
1229721483 189523899 151617768 1463724198 16846158 1688834887 28881848 585824914 1391713445 879865189 981148882 899651598 1392121879 896542544 3736481882 818193783 787514689 1792147858 38445888 1448888
389 1659385883 925513688 1384448819 2138651175 1478382282 2314188887 1684865318 1348998888 882597887 1289585312 1398453518 539878488 554731587 1255661842 1224895387 388377871 2849691445 1863388731 14892857
878 2678552674 381612828 81388893 1484518886 576883899 1582318828 1639397383 1588788249 187187158 1848883238 84719478 1438855383 242372439 1578414132 1557854639 34518538 157489315 1158812899 1179585211
1229681843 988858888 1888284544 1622145263 1132386357 291322143 1344448498 1165586532 1243711638 1556685924 1864347348 1861558545 2895938292 714881435 9374888368 218149985 1898246671 1999995377 878025419 14
55457552 598418827 674718976 113222414 588813642 971788976 1261625347 1384439863 1258168972 68937558 1188783211 1446889147 88248143 441598653 122373213 187485785 1883488848 714838278 1475148483 16121
198532 1819222276 1139128827 642884378 1688838298 1444527882 107126879 1281385385 263591874 2871385311 1989788588 1641498948 1778822898 36833828 188661224 96284543 416789358 2818849598 1443197881 357888193
1138639417 675676126 1778881171 1974612258 1888972735 315766919 2423651 1655138878 281233334 492948815 1864270221 167488881 1774991378 576957243 1297883468 47023127 88922688 18677318 71333648 11859394
81 1864182672 1785147878 893475866 164602584 2180758578 328887710 1874518738 128887110 1874518738 156218325 1431713445 879865189 981148882 899651598 1392121879 896542544 3736481882 818193783 787514689 1792147858 38445888 1448888
784566 15717156425 983364598 55619239 1688878184 1842493274 481184887 958228865 167951265 1217888938 782891875 78783389 195666638 24115977 288888466 87873381 1579226889 218281242 513824471
2312488888 388872641 761958385 581738286 28642217 112628889 1326518828 787438218 418397542 1294578839 1123888831 1376299848 722262873 1624868843 980114695 889212565 1725544182 47763889 831929685 7888864
96 125688888 1666143277 1786488576 1618738133 338634136 773928868 971387887 128537892 1468651887 2478872 1877344679 151882823 1872489921 1588887886 458978132 927688814 45825246 68552859 399953474 79
4872878 199452252 1738544687 1888487757 2185812194 1274523482 1485964701 1475358392 388887168 1682823393 1788885345 1781458837 1821497582 1866592842 388848885 1884788542 287117888 183553375 1419479288 3
28187212 693888614 1374651955 38464531 1644632877 297878423 1626678638 1228338898 1882819198 888997422 1618846533 1211733529 1169599861 3888764238 1274128761 774318189 558787577 1768888366 288891258 1486
791754 568882589 1448878589 529534844 554888777 1562876767 1731229921 723372174 897864217 1954988772 1594988772 1188658864 1948784828 1818112667 374812318 365512862 1521167945 1585891143 1532999925 1788879561 355483
472 173124666 488532839 1888581834 1351951726 1881786289 1199987256 1447336582 98467964 1748891826 2821855258 1388825675 987888647 1743715886 51478886 944389166 859582588 615841681 559858173 283157763 1
811482485 733652558 252477971 1945237552 798218829 1273363821 46598474 917487957
```

```
Colors not equal
vertex 18: color 1
vertex 12: color 8
1 - another one test, other - finish testing
```

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
```