Учреждение образования

«БЕЛОРУССКИЙ ГОСУДАРСТВЕННЫЙ УНИВЕРСИТЕТ

ИНФОРМАТИКИ И РАДИОЭЛЕКТРОНИКИ»

Кафедра информационных технологий автоматизированных систем

Отчет по лабораторной работе №4

по курсу «ИТОКБ»

на тему: «Открытое распространение ключей»

Вариант 4

|  |  |
| --- | --- |
| Выполнил магистрант группы 025941: | Колесников В.Г. |
| Проверил: | Боброва Т.С. |

МИНСК

2021

**Задание 1:** Для заданного простого *P* = 1877 найти *g* — примитивный элемент конечного поля GF*(P)* и выполнить генерацию общего секрета.

Код программы приведен в конце лабораторной работы.

Результат работы программы поиска примитивного элемента и генерации секретов приведен на рисунке 1.1.

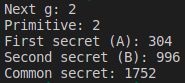


Рисунок 1.1 — Результат поиска примитивного элемента и генерации секретов

Алгоритм использования общего секрета:

1. Алиса генерирует целое число *a* и держит его в секрете, затем вычисляет

*A = g­amod P = 213771877mod 1877 = 304*

и пересылает его Бобу;

2. Боб генерирует целое число *b* и держит его в секрете, затем вычисляет

*B = gbmod P = 218771377mod 1877 = 996*

и пересылает его Алисе;

3. Алиса вычисляет значение

*Bamod P = gabmod P = 30413771877mod 1877 = 1752*

4. Боб вычисляет значение

*Abmod P = gabmod P = 99618771377mod 1877 = 1752*

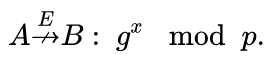
Конечное число для обоих участников и является общим секретом.

**Выводы:**

1. Модель атакующего и оценка длины ключа:

При атаке методом «человек посередине» атакующий должен заранее представиться сертифицированным источником (либо тем участником, которому предполагается пересылка ключа) для обоих сторон. Алгоритм атаки

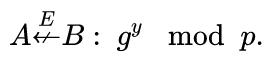
1) Сторона ***A*** отправляет сообщение стороне ***B***:



2) Криптоаналитик ***E*** перехватывает сообщение стороны ***A*** и подменяет его, отправляя стороне ***B*** уже другое сообщение :



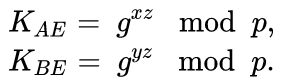
3) Сторона ***B*** отправляет сообщение стороне ***A***:



4) Криптоаналитик ***E*** перехватывает сообщение стороны ***B*** и подменяет его, отправляя стороне ***A*** какое-то своё сообщение:



5) Результатом данных действий является образование двух каналов связи криптоаналитика ***E*** со сторонами ***A*** и ***B***, причем сторона ***A*** считает что общается со стороной ***B*** при помощи секретного ключа ***KAE***, а сторона ***B*** отправляет сообщения при помощи ключа ***KBE***. При этом стороны ***A*** и ***B*** не подозревают, что обмен сообщениями происходит не напрямую, а через криптоаналитика ***E***:



Длина ключа напрямую влияет на устойчивость алгоритма к дешифровке. При малом размере ключа допускается возможность взлома за разумное время. Поэтому рекомендуется использовать ключ большего размера. На длину ключа влиет величина *P* и степень корня. Поэтому данные две величины рекомендуется увеличивать при реализации алгоритма.

2. Возможные угрозы протоколу и предложения по защите от них

Самая очевидная угроза — наличие «человека посередине». Для защиты от данного вида атак, разными алгоритмами предусмотрены различные механизмы. Один из них — использование доверенного центра сертификации, что практически исключает наличие третьей несертифицированной стороны.

Вторая угроза — малый размер ключа. При относительно малом размере, взлом возможен за разумное время, поэтому размер ключа рекомендуется увеличивать.

**Код программы:**

#include <iostream>

#include <string>

#include <bits/stdc++.h>

#include <cmath>

using namespace std;

// To reverse numbers bits

string reverseBits(string source)

{

string reversed = source;

reverse(reversed.begin(), reversed.end());

return reversed;

}

string getNumberBits(int number)

{

bool foundOne = false;

string bits = "";

for (int i = (sizeof(number) \* \_\_CHAR\_BIT\_\_ - 1); i >= 0; --i)

{

if (((number >> i) & 1) == 1)

{

foundOne = true;

}

if (foundOne)

{

bits.append(((number >> i) & 1) == 1 ? "1" : "0");

}

}

return bits;

}

const int getModulo(int g, int power, int P)

{

string powerBits = getNumberBits(power);

string powerBitsRev = reverseBits(powerBits);

unsigned int modulo = (g \* g) % P;

for (int i = powerBitsRev.length() - 1; i >= 0; i--)

{

if (powerBitsRev.at(i) == '1')

{

modulo \*= g;

}

if (i - 1 != -1)

{

modulo = (modulo \* modulo) % P;

}

else

{

modulo %= P;

}

}

return modulo;

}

const unsigned long getPrimitive(const unsigned int P)

{

unsigned int g = 1;

bool isAllPrimitive = false;

while (true)

{

g++;

cout << "Next g: " << g << endl;

if (P % g == 0 || g % P == 0)

{

continue;

}

isAllPrimitive = true;

for (int i = 2; i <= g; i++)

{

if (g % i == 0 && P % i == 0)

{

isAllPrimitive = false;

break;

}

}

if (!isAllPrimitive)

{

continue;

}

for (int power = 2; power < P - 1; power++)

{

const int modulo = getModulo(g, power, P);

if (modulo == 0)

{

isAllPrimitive = false;

break;

}

}

if (isAllPrimitive)

{

break;

}

}

return g;

}

int main(int argc, char \*\*argv)

{

setlocale(LC\_ALL, "");

srand(time(NULL));

const unsigned int P = 1877;

// - 1

// а

const unsigned long g = getPrimitive(P);

cout << "Primitive: " << g << endl;

// б

const unsigned int a = 13771877;

const unsigned int b = 18771377;

const unsigned int A = getModulo(g, a, P);

const unsigned int B = getModulo(g, b, P);

cout << "First secret (A): " << A << endl;

cout << "Second secret (B): " << B << endl;

const unsigned int Ab = getModulo(A, b, P);

const unsigned int Ba = getModulo(B, a, P);

cout << "Common secret: " << Ab << endl;

return 0;

}