

МИНОБРНАУКИ РОССИИ
Федеральное государственное бюджетное образовательное учреждение
высшего образования
**«САРАТОВСКИЙ НАЦИОНАЛЬНЫЙ ИССЛЕДОВАТЕЛЬСКИЙ
ГОСУДАРСТВЕННЫЙ УНИВЕРСИТЕТ ИМЕНИ Н.Г.
ЧЕРНЫШЕВСКОГО»**

Кафедра теоретических основ
компьютерной безопасности и
криптографии

Протоколы анонимности
ОТЧЁТ
ПО ДИСЦИПЛИНЕ
«КРИПТОГРАФИЧЕСКИЕ ПРОТОКОЛЫ»

студента 5 курса 531 группы
специальности 10.05.01 Компьютерная безопасность
факультета компьютерных наук и информационных технологий
Яхина Шамяля Илдусовича

Преподаватель

аспирант

Р. А. Фарахутдинов

подпись, дата

Саратов 2023

СОДЕРЖАНИЕ

ВВЕДЕНИЕ	3
1 Теоретические сведения	4
1.1 Описание алгоритмов протокола тайного голосования Фудзиоки — Окамото — Оты и Sensus	4
2 Практическая реализация	6
2.1 Описание программы	6
2.2 Тестирование программы.....	6

ВВЕДЕНИЕ

Цель работы - реализация протокола тайного голосования Sensus.

1 Теоретические сведения

Протокол Sensus является модификацией протокола Фудзиоки - Окамото - Оты. Его в 1996 году предложили Лорри Кранор и Рон Ситрон. Поэтому сначала рассмотрим протокол Фудзиоки - Окамото - Оты, а потом будет описано, в чем заключается отличие этих протоколов.

Схема Sensus, как и схема Фудзиоки — Окамото — Оты, основывается на протоколе двух агентств и криптографической подписи вслепую. Несмотря на усложнение протокола, эта схема частично решает проблему сговора двух агентств. Для работы протокола необходим заранее выбранный способ маскирующего шифрования, под которым избиратель присылает регистратору бюллетень. Слепяющее (маскирующее) шифрование — особый вид шифрования, позволяющее удостовериться в том, что документ подлинный и подписан авторизованным пользователем, но не даёт узнать содержащиеся в нём данные. Маскирующее шифрование должно быть коммутативным с электронной подписью, то есть $sign(blind(B)) = blind(sign(B))$.

1.1 Описание алгоритмов протокола тайного голосования Фудзиоки — Окамото — Оты и Sensus

Алгоритм протокола тайного голосования Фудзиоки - Окамото - Оты.

Вход: Количество избирателей, количество кандидатов, битовая длина числа для генерации ключей.

Выход: Результаты голосования.

1. V утверждает списки легитимных избирателей;
2.
 - а) E создаёт $e_{public}, e_{private}$ (для цифровой подписи) и e_{secret} (для того, чтобы ни A , ни посторонний злоумышленник не мог до нужного времени узнать содержимое бюллетеня);
 - б) E подготавливает сообщение B с выбранным решением;
 - в) E шифрует его e_{secret} ;
 - г) E накладывает слой слепяющего шифрования;
 - д) E подписывает его $e_{private}$;
 - е) E отправляет V подписанное сообщение:
 $blind(sign(e_{private}, encrypt(e_{secret}, B)))$.
3.
 - а) V создаёт v_{public} и $v_{private}$, публичный ключ выкладывается в общий доступ;

- б) V удостоверяется, что бюллетень действительный и принадлежит легитимному и не голосовавшему избирателю;
 - в) V подписывает его $v_{private}$;
 - г) V возвращает его E .
4. E снимает с бюллетени слой маскирующего шифрования (в силу коммутативности остаётся $sign(v_{private}, sign(e_{private}, encrypt(e_{secret}, B)))$) и отправляет её A ;
5. а) A проверяет подписи E и V ;
- б) A помещает всё ещё зашифрованную e_{secret} бюллетень в специальный список, который будет опубликован после того как все избиратели проголосуют или по истечении заранее оговорённого срока.
6. После того как список появляется в открытом доступе, E высылает A e_{secret} ;
7. а) A расшифровывает сообщение;
- б) A подсчитывает результаты.

Отличие протокола Sensus от описанного выше Фудзиоки - Окамото - Оты заключается в шагах 5 - 6. После того, как A получило зашифрованное сообщение от E , оно не только добавляет его в публикуемый список, а вдобавок отправляет подписанный бюллетень обратно избирателю в качестве квитанции. Таким образом E не нужно ждать, пока проголосуют все остальные, и он может закончить голосование за один сеанс. Это не только удобно для конечного пользователя, но ещё и предоставляет дополнительное доказательство, что E участвовал в выборах. Кроме того, в Sensus регламентированы дополнительные вспомогательные модули, упрощающие и автоматизирующие ход голосования.

2 Практическая реализация

2.1 Описание программы

Все действия алгоритма протокола тайного голосования выполняются в функции *sensus_main*.

Для подписывания сообщения с помощью *RSA* используется реализованная с помощью средств библиотеки *crypto++* функция *generateRSASign* и функция *checkRSASign* для проверки корректности этой подписи.

Шифрование с помощью *AES* реализовано в функции *encryptAES*, а дешифрование в функции *decryptAES*. Чтобы создать ключ для *AES* шифрования нужно воспользоваться функцией *generateAESKeyFromK*.

Для генерации маски из модуля *N*, который в данном случае достается из публичного ключа *RSA*, используется функция *generateMask*. Затем функция *blindSignature* применяет данную маску к сообщению. А чтобы снять маскирующее шифрование реализована функция *unblindSignature*.

Функция *generate_random* генерирует случайное число в заданном промежутке, а *rand_large_by_bit_length* генерирует число по заданной битовой длине числа.

Для возведения числа в степень по модулю реализована функция *powMod*.

Для вычисления обратного элемента используется расширенный алгоритм Евклида, который реализован в функции *algEuclidExtended*, а также в аналогичной функции *algEuclidExtendedInteger*, т.к. используются большие числа двух видов: *Integer* и *cpp_int*.

Подпись генерируется в функции *generateSign*.

Чтобы преобразовать определенное в библиотеке *crypto++* число типа *Integer* в строку реализована функция *IntegerToString*.

2.2 Тестирование программы

На рисунке 1 показан вызов параметра *help*, который выводит информацию о допустимых параметрах командной строки программы.

```
D:\cpp_projects\protocols\lab8_sensus\x64\Release>lab8_sensus.exe /h
Введена команда с /h. Допустимые параметры:

/l:<length> – Битовая длина для ключа AES

/c:<c> – Количество кандидатов

/v:<v> – Количество избирателей

/h – информация о допустимых параметрах командной строки программы.
```

Рисунок 1 – Вызов параметра *help*

На рисунке 2 запускается программа с параметрами $l = 2048$, $c = 5$, $v = 10$. Среди них только 4 избирателя оказались легитимными.

```
C:\WINDOWS\system32\cmd. x + v
D:\cpp_projects\protocols\lab8_sensus\x64\Release>lab8_sensus.exe /c:5 /v:10 /l:2048

Шаг 1
Легитимные избиратели: i2, i3, i7, i8
Кандидаты: 0, 1, 2, 3, 4

Шаг 2

Внимание! Избиратель i2 приступил к голосованию.
Сгенерированный eSecret: C4433CC47ECF61F037285157639C828EA196F1ADE66BA701876C76C9980B3E55
Избиратель сгенерировал себе ключи e_public и e_private
Избиратель i2 голосует за кандидата %0

Сообщение B = 0
Зашифрованное с помощью e_secret сообщение B: 115033478929764125312083165989888775020
Модуль RSA для маскирования подписи: N = 182272203963326415505769609814825044065182405346755869412322891003324092594561432682464
31858792826678483470691661823968044533566247785972403964699943390291981015379547096318501587874011953052436574785173681654075179
80470768601716559319173747114282931685138337724828177833686978003444830043172203965705708018385198609169644858755480321052494860
8488601883358999448402717808176850174173235241657608620388032359559956593487907709613526474490204218597216663503832829377508548
090100634541713503908220167569423017300603990425062722849297795036769151547674257035964414519779763932543027877315077362768440789
548161788779069471.
Зашифрованное B после наложения слоя ослепляющего шифрования: 108567673361886123549075600669639976131426748877241040642718915272
91722188785090643915538331737426823119472898561036861968672564142709862846882517599850988217289734851236407899643895882957388661
66793978920355597423514428131870466436683214321755785602358832832780390681206894117981527967837460841915842341703379341961304230
37143288328220129668271353107916958364922543548947660741243418474676992650522566430436296725298131011109991293741960727884552493
29628194582064136853775719956287142586229959445000741771049410387015509757737461195776639546718911200041601586639046610794303658
952322737104774966715079292445100638949.
Подписанное с помощью e_private замаскированное сообщение: 190900527271786.

Шаг 3

Регистратор V принял это сообщение, а также номер избирателя
V сгенерировал себе ключи v_public и v_private
V начинает проверку избирателя.
Проверки на легитимность и на то, голосовал ли уже кандидат, пройдены
Подписанное с помощью v_private сообщение: 8716389.
```

Рисунок 2 – Корректный запуск программы и начало шагов протокола для первого избирателя

На рисунках 3 - 9 показаны шаги протокола для каждого из избирателей

```

Шаг 4
Избиратель получает сообщение от регистратора
Сообщение после снятия маскирующего шифрования: 24869368623402459883359028580475810684999689354216394714107516439201377018581264
9276193741316163986214961545112405170413703906252053780604832249450784525872208918157003242062754529944772234024217673543837224
77282677989512577294250844447497762278195144679727136205619489914382415173470647585884679473778596561021703848245584776695586218
81645766879420681573837485928020104010936044787838977794493822897914423122456632476246070640910651283917083230155851660205546980
91187924225418858865674174590112118317603493932380504637878538464483904062769832628077121640356055926066988591987868249112083945
756766923026507894164279.

Шаг 5
А начинает проверку подписи V и Е.
Проверка V прошла успешно
Проверка Е прошла успешно
Зашифрованное сообщение В кладется в специальный список, Е получает квитанцию и может быть свободен.

```

Рисунок 3 – Шаги протокола для первого избирателя

```

C:\WINDOWS\system32\cmd. x + v

Шаг 2

Внимание! Избиратель i3 приступил к голосованию.

Сгенерированный eSecret: 780CBC2F96FCA16DF682669EB0E66A41D8035D378E20DF56399EC4B7CA680D2F

Избиратель сгенерировал себе ключи e_public и e_private

Избиратель i3 голосует за кандидата "4

Сообщение В = 4
Зашифрованное с помощью e_secret сообщение В: 60748836626856731409015140984178617975
Модуль RSA для маскирования подписи: N = 311389398630151866941572264156589251487288908005984795267504666312351861986348785084684
2767234555823376152729827377663595323397946212882464723968358359105279696286400934332145559934876271886503139864600799123698965
93864581551919368281098832000039281003657395684604341771485718297258201539360190727037272667378847279128988200622672557341899302
261966875136071153462384136196349427207364176791352133379464768223835828686822966780647942000536571916454325912787381277414912555
77085358425723227017316164527866570698175727670806344924629408035030049869799028068368023343206680455079212128095148456726271353
106092451273599529.
Зашифрованное В после наложения слоя ослепляющего шифрования: 306524650525739966260637461067122390298167477584293434336026268021
362963171320260876816632130647520120103876181328203685586494054474709084445788184309349044236859412023026151120008969025120697960
23029880003139685740914925411813492491385423996254958002995025549903350161159065658938764293103229379998046750781379382496348647
58135064052058545220631715950008024047801819395259195649547851386982921952239857923737674349682047060518109858787190366587435979
6642204530409375092508487808055497380357025687112679986077544277736953790333263595802002420771744241970250757119965774803655163
835301105692297197863343180998984474864.
Подписанное с помощью e_private замаскированное сообщение: 8746349.

```

Рисунок 4 – Шаги протокола для второго избирателя

```

Шаг 3

Регистратор V принял это сообщение, а также номер избирателя
V сгенерировал себе ключи v_public и v_private
V начинает проверку избирателя.
Проверки на легитимность и на то, голосовал ли уже кандидат, пройдены
Подписанное с помощью v_private сообщение: 9960.

Шаг 4
Избиратель получает сообщение от регистратора
Сообщение после снятия маскирующего шифрования: 17751973911052195748432082834961396778966971666001965882604959508455217389506567
84830522941902798028974309310329409277708804796359870074766630313840762843296037431097546092176372358421997200993753612699135186
6554341621933204116559439793573481522237568606659459237606838643305496450293194007287076122386273539606688081175252874652773950
64326475639163505373142847656379967747428198477935138214808627247601979063877472374020977996887810497133839412531495362859213064
9193011800815325756669377939838806987451320864051859000579965060367918615162235986359308640387526514552433818894381518699862840
1362942589460655160031444.

Шаг 5
А начинает проверку подписи V и Е.
Проверка V прошла успешно
Проверка Е прошла успешно
Зашифрованное сообщение В кладется в специальный список, Е получает квитанцию и может быть свободен.

```

Рисунок 5 – Шаги протокола для второго избирателя


```
C:\WINDOWS\system32\cmd. x + v

Шаг 2

Внимание! Избиратель i7 приступил к голосованию.

Сгенерированный eSecret: B87E0004F5EC68CC7FA4139683C4FBA1ED89874A5F6411E914DCFAE23615289E

Избиратель сгенерировал себе ключи e_public и e_private

Избиратель i7 голосует за кандидата %2

Сообщение B = 2
Зашифрованное с помощью e_secret сообщение B: 264926536517231080804328978144414347635
Модуль RSA для маскирования подписи: N = 226856114034750030356096694835650255379983268733462909306728897639319068417590539584858
982085139142218310970053158461667242577558187313865924321274669502572504951518015019502194908916023625751049810248279270845016820
94080177333677230789233694518903068527365923872947083371153295221254838035918008237611110326909269157745953103380026597797930354
46144699669067190977531180542196277473104534433101488205429616739417979345869663417598822849730040352376517765631446632099535985
21077983723163466339713838841815634836852789435057237413107467219733835180148387593019724145901237394644718831177904794344273335
224792745902431047.
Зашифрованное B после наложения слоя ослепляющего шифрования: 192675521238893291789416649708369741887586427653761344712891993300
42875409230919350582200277120138291087439409359033389095432096554256410145501576780370635808349558139001802156893587756766581581
86990136670162461151753497952155710371795882056167627912728555988701854750518962578934096741534773779690505577389127533081068736
77238622763234936639363944194658799482652980128756173330398617091024681042673658396642401931631277896728276239037606720412609792
2155706852192466534058544064712637661144521181926269617852518980812476751211885080550697490781244116329015769557724630747768627
019707850319544762188394388365871169620.
Подписанное с помощью e_private замаскированное сообщение: 3044.
```

Рисунок 6 – Шаги протокола для третьего избирателя

```
C:\WINDOWS\system32\cmd. x + v

Шаг 3

Регистратор V принял это сообщение, а также номер избирателя
V сгенерировал себе ключи v_public и v_private
V начинает проверку избирателя.
Проверки на легитимность и на то, голосовал ли уже кандидат, пройдены
Подписанное с помощью v_private сообщение: 1.

Шаг 4

Избиратель получает сообщение от регистратора
Сообщение после снятия маскирующего шифрования: 19242654757902005579015081448940619578309290584448316998503676889472334957892344
3897852321723450243846535121061005865670814867050702002228365600297823549310442134941515298549817352438973221950452119994417792
23704597110751398508750981587944438644980454921843080764122318147452078083909274477287856078489114714255820918525684528277432799
72125822648468810972699496076095426651630514280190951368445693881545255665506981491130107466805808058009449756656652517395134093
3917219920956049417410253901532755933555999087048081284513167499503865662448249309936764409841622123499233354933793686725760740
419243161045296765208060.

Шаг 5

A начинает проверку подписи V и E.
Проверка V прошла успешно
Проверка E прошла успешно
Зашифрованное сообщение B кладется в специальный список, E получает квитанцию и может быть свободен.
```

Рисунок 7 – Шаги протокола для третьего избирателя

```
C:\WINDOWS\system32\cmd. x + v

Шаг 2

Внимание! Избиратель i8 приступил к голосованию.

Сгенерированный eSecret: 9000981BF8A17971CC52DD2672D5315F140EEDA89F64AF3AE4C048FE8B3737DF

Избиратель сгенерировал себе ключи e_public и e_private

Избиратель i8 голосует за кандидата %0

Сообщение B = 0
Зашифрованное с помощью e_secret сообщение B: 7346103000192847248214020105824114361
Модуль RSA для маскирования подписи: N = 229332262640224397939598142076358334471855198378588953406879380651502773681171434106197
83446252741073469245039596402452217603983198345016452956914569769109099471738168725268425613599477113980274848924426541600586940
50858614038535697690133200005601132438568118857758535254171359753423169966516074114901921080612046379383425073421146367019412525
77479179339134923517375163231713165880294327319084287868738932804789884492509859865907122173002185114279085527800074254998701328
90325062444050117639652856913974793287345843479781709308577750926331956975879684264199137378688702724010005530469868820187776715
832467555143052641.
Зашифрованное B после наложения слоя ослепляющего шифрования: 509276928559741137232151636533973864511996044771605573290261148838
39285743741654812656415002881869556254701677337474011911028636498577877770095701163788021249654578974473012852118375773114293689
53628442693388033927919376240425350943445124638219991329203557207902777198548593480608154987231151710189158550652979690997116387
33572551971567703606206563728213195402804599703314005225522136733561706051019683955856490421359730094209693905216734297684405998
09001397466333080235027527077031487958047317786527671448599492906418670466991186978701979053254811714890277190125884559339399419
30266578643360346560506854445061333616.
Подписанное с помощью e_private замаскированное сообщение: 120018890.
```

Рисунок 8 – Шаги протокола для четвертого избирателя

```
C:\WINDOWS\system32\cmd. X + v

Шаг 3

Регистратор V принял это сообщение, а также номер избирателя
V сгенерировал себе ключи v_public и v_private
V начинает проверку избирателя.
Проверки на легитимность и на то, голосовал ли уже кандидат, пройдены
Подписанное с помощью v_private сообщение: 4.

Шаг 4

Избиратель получает сообщение от регистратора
Сообщение после снятия маскирующего шифрования: 71465259944450347538173270636008344244871583096401151760562072044381193814070481
50901145906494001642036752892325316567505639764044953448556301936200285646523409449083756580679379611637674311705137734501581077
72973754736309030927751626184755978831313085989618754497417486583689994382404671168014358044659475302714168958721373785307759154
00022102253178317326445122310947175371994042092262556448164010618442628843485227588730265416075906604731450495592480502928606490
11347165717505653290145352878948721395611124933924453363118330208678072439758313031097991043341737613565203470795998283511747127
32102044951719873785057.

Шаг 5

A начинает проверку подписи V и E.
Проверка V прошла успешно
Проверка E прошла успешно
Зашифрованное сообщение B кладется в специальный список, E получает квитанцию и может быть свободен.
```

Рисунок 9 – Шаги протокола для четвертого избирателя

На рисунке 10 показан результат выполнения протокола и итоги голосования.

```
C:\WINDOWS\system32\cmd. X + v

A начинает расшифровывать сообщения:
Легитимный избиратель i2 проголосовал за 0
Легитимный избиратель i3 проголосовал за 4
Легитимный избиратель i7 проголосовал за 2
Легитимный избиратель i8 проголосовал за 0

Всем приготовиться, A проводит подсчет результатов:

1...
2...
3...

Голоса за кандидатов :
Кандидат 0 набрал 2 голосов
Кандидат 1 набрал 0 голосов
Кандидат 2 набрал 1 голосов
Кандидат 3 набрал 0 голосов
Кандидат 4 набрал 1 голосов

D:\cpp_projects\protocols\lab8_sensus\x64\Release>
```

Рисунок 10 – Итоги голосования

Запустим данный тест для тысячи избирателей и пяти кандидатов. На рисунке 11 показан результат работы протокола с этими данными.

Всем приготовиться, А проводит подсчет результатов:

1...

2...

3...

Голоса за кандидатов :

Кандидат 0 набрал 101 голосов

Кандидат 1 набрал 99 голосов

Кандидат 2 набрал 100 голосов

Кандидат 3 набрал 95 голосов

Кандидат 4 набрал 88 голосов

Рисунок 11 – Итоги голосования для тысячи избирателей

ПРИЛОЖЕНИЕ А

Листинг программы

```
#include <iostream>
#include <vector>
#include <chrono>
#include <time.h>
#include <boost/random/random_device.hpp>
#include <boost/multiprecision/cpp_int.hpp>
#include <boost/random.hpp>
#include <sstream>
#include <fstream>
#include <unordered_map>
#include <string>
#include <windows.h>
#include <cryptlib.h>
#include "rijndael.h"
#include "modes.h"
#include "files.h"
#include "osrng.h"
#include "hex.h"
#include "rsa.h"
#include <base64.h>
#include <integer.h>
#include <filters.h>
#include <nbtheory.h>
#include <unordered_set>
#include <chrono>

using namespace std;
using namespace boost::multiprecision;
using namespace boost::random;
using namespace CryptoPP;

const int AES_KEY_SIZE = AES::DEFAULT_KEYLENGTH;
const int AES_BLOCK_SIZE = AES::BLOCKSIZE;
cpp_int pSize;

Integer generateMask(const Integer& N) {
    AutoSeededRandomPool rng;
    Integer r;
    do {
        r.Randomize(rng, Integer::One(), N - 1);
    } while (!RelativelyPrime(r, N));

    return r;
}

Integer blindSignature(const Integer& signature, const Integer& mask, const Integer& N) {
    return signature * mask % N;
}

Integer algEuclidExtendedInteger(Integer a, Integer b, Integer& x, Integer& y) {
    if (a == 0) {
        x = 0;
        y = 1;
        return b;
    }
    Integer xi, yi;
```

```

    Integer nod = algEuclidExtendedInteger(b % a, a, xi, yi);
    x = yi - (b / a) * xi;
    y = xi;
    return nod;
}

Integer unblindSignature(const Integer& blindedSignature, const Integer& mask, const
Integer& N) {
    Integer x1, y1;
    algEuclidExtendedInteger(mask, N, x1, y1);
    if (x1 < 0)
        x1 = x1 + N;
    Integer inverse = x1;
    return blindedSignature * inverse % N;
}

Integer generateRSASign(const std::string& message, const RSA::PrivateKey& privateKey,
AutoSeededRandomPool& rng) {
    RSASSA_PKCS1v15_SHA_Signer signer(privateKey);

    std::string signature;
    StringSource(message, true,
        new SignerFilter(rng, signer, new StringSink(signature))
    );

    return Integer(signature.c_str());
}

bool checkRSASign(const std::string& message, const Integer& signature, const
RSA::PublicKey& publicKey) {
    RSASSA_PKCS1v15_SHA_Verifier verifier(publicKey);

    ostringstream oss;
    oss << signature;
    string signatureStr = oss.str();

    try {
        StringSource(signatureStr + message, true,
            new SignatureVerificationFilter(verifier, nullptr)
        );
        return true; // Подпись верна
    }
    catch (const SignatureVerificationFilter::SignatureVerificationFailed&) {
        return false; // Подпись неверна
    }
}

cpp_int generate_random(cpp_int a, cpp_int b) {
    random_device rd;
    mt19937 gen(rd());
    uniform_int_distribution<cpp_int> dist(a, b);
    return dist(gen);
}

cpp_int rand_large_by_bit_length(int l) {
    random_device rd;
    mt19937 gen(rd());
    uniform_int_distribution<int> distribution(0, 1);
    cpp_int result = 0;
    for (int i = 1; i < l - 1; ++i) {
        result <<= 1;
        result += distribution(gen);
    }
    result |= (cpp_int(1) << (l - 1));
}

```

```

        result |= 1;
        return result;
    }

    string encryptAES(const string& plainText, const string& hexKey) {
        SecByteBlock key((const byte*)hexKey.data(), AES_BLOCK_SIZE);
        ECB_Mode<AES>::Encryption encryptor;
        encryptor.SetKey(key, key.size());

        string cipherText;
        StringSource(plainText, true, new StreamTransformationFilter(encryptor, new
StringSink(cipherText)));

        cpp_int decimalCipherText = 0;
        for (char c : cipherText) {
            decimalCipherText = decimalCipherText * 256 + static_cast<unsigned char>(c);
        }

        ostringstream oss;
        oss << decimalCipherText;
        return oss.str();
    }

    string decryptAES(const string& decimalCipherText, const string& hexKey) {

        SecByteBlock key((const byte*)hexKey.data(), AES_BLOCK_SIZE);
        ECB_Mode<AES>::Decryption decryptor;
        decryptor.SetKey(key, key.size());

        cpp_int decimalCipherTextValue(decimalCipherText);

        string cipherText;
        while (decimalCipherTextValue > 0) {
            unsigned char byte = static_cast<unsigned char>(decimalCipherTextValue % 256);
            cipherText.insert(cipherText.begin(), byte);
            decimalCipherTextValue /= 256;
        }

        string decryptedText;
        StringSource(cipherText, true, new StreamTransformationFilter(decryptor, new
StringSink(decryptedText)));

        return decryptedText;
    }

    cpp_int powMod(cpp_int x, cpp_int n, cpp_int m) {
        cpp_int N = n, Y = 1, Z = x % m;
        while (N != 0) {
            cpp_int lastN = N % 2;
            N = N / 2;
            if (lastN == 0) {
                Z = (Z * Z) % m;
                continue;
            }
            Y = (Y * Z) % m;
            if (N == 0)
                break;
            Z = (Z * Z) % m;
        }
        return Y % m;
    }

    cpp_int nod(cpp_int a, cpp_int m) {
        if (m == 0)

```

```

        return a;
    else
        return nod(m, a % m);
}

cpp_int algEuclidExtended(cpp_int a, cpp_int b, cpp_int& x, cpp_int& y) {
    if (a == 0) {
        x = 0;
        y = 1;
        return b;
    }
    cpp_int xi, yi;
    cpp_int nod = algEuclidExtended(b % a, a, xi, yi);
    x = yi - (b / a) * xi;
    y = xi;
    return nod;
}

string findInStr(string const& str, int n) {
    if (str.length() < n)
        return str;
    return str.substr(0, n);
}

vector<string> splitString(const string& input, char zn) {
    istringstream stream(input);
    string str1;
    vector<string> str;
    while (getline(stream, str1, zn)) {
        str.push_back(str1);
    }
    return str;
}

void helpFunc() {
    cout << "Введена команда с /h. Допустимые параметры:";
    cout << "\n\n/l:<length> - Битовая длина для ключа AES";
    cout << "\n\n/c:<c> - Количество кандидатов";
    cout << "\n\n/v:<v> - Количество избирателей";
    cout << "\n\n/h - информация о допустимых параметрах командной строки программы.\n";
}

string intToHexString(cpp_int K) {
    std::ostringstream stream;
    stream << std::hex << K;
    return stream.str();
}

string generateAESKeyFromK(cpp_int K) {
    string KString = intToHexString(K);
    SHA256 hash;
    byte digest[SHA256::DIGESTSIZE];
    hash.CalculateDigest(digest, (const byte*)KString.c_str(), KString.length());
    string hexKey;
    HexEncoder encoder(new StringSink(hexKey));
    encoder.Put(digest, sizeof(digest));
    encoder.MessageEnd();
    return hexKey;
}

string IntegerToString(Integer xInt) {
    ostringstream oss;
    oss << xInt;
    string xStr = oss.str();
}

```

```

    return xStr;
}

void printMessage(const string& message, int delaySeconds) {
    this_thread::sleep_for(chrono::seconds(delaySeconds));
    cout << "\n" << message << "\n";
}

void sensus_main(cpp_int countOfVoters, cpp_int countOfCandidates, int bitsLength) {

    AutoSeededRandomPool rng;

    //Шаг 1
    cout << "\n\tШаг 1\n";

    vector<cpp_int> legitimateVoters;
    cpp_int countOfLegitimateVoters;
    for (int i = 0; i < countOfVoters; i++)
    {
        if (generate_random(0, 1) == 0)
            legitimateVoters.push_back(i);
    }

    map<cpp_int, cpp_int> votersChoice;
    map<cpp_int, string> votersSecrets;
    cout << "\nЛегитимные избиратели: ";
    for (int i = 0; i < legitimateVoters.size(); i++) {
        votersChoice.insert(make_pair(legitimateVoters[i], -1));
        if (i == 0)
            cout << "i" << legitimateVoters[i];
        else
            cout << ", " << "i" << legitimateVoters[i];
    }
    cout << "\n";

    cout << "\nКандидаты: ";
    for (int i = 0; i < countOfCandidates; i++) {
        if (i == 0)
            cout << i;
        else
            cout << ", " << i;
    }
    cout << "\n";

    for (int i = 0; i < legitimateVoters.size(); i++) {
        //Шаг 2 E
        cout << "\n\tШаг 2\n";
        //Номер избирателя
        cpp_int me = legitimateVoters[i];
        cout << "\n\nВнимание! Избиратель i" << me << " приступил к голосованию.\n\n";

        string eSecret = generateAESKeyFromK(rand_large_by_bit_length(bitsLength));

        cout << "Сгенерированный eSecret: " << eSecret << "\n";

        RSA::PrivateKey ePrivate;
        ePrivate.GenerateRandomWithKeySize(rng, 2048);

        RSA::PublicKey ePublic(ePrivate);
        cout << "\nИзбиратель сгенерировал себе ключи e_public и e_private\n";

        cpp_int bNum = generate_random(0, countOfCandidates - 1);
        string B = boost::lexical_cast<string>(bNum);
    }
}

```



```

cout << "\nИзбиратель i" << me << " голосует за кандидата №" << B << "\n";
cout << "\nСообщение B = " << B << "\n";

//зашифровываем B
string encryptedB = encryptAES(B, eSecret);

Integer encryptedBInteger(encryptedB.c_str());

cout << "Зашифрованное с помощью e_secret сообщение B: " << encryptedB << "\n";
cpp_int encryptedBNum(encryptedB);

//применяем ослепляющее шифрование

// Получение модуля N из открытого ключа
const Integer& N = ePublic.GetModulus();

cout << "Модуль RSA для маскирования подписи: N = " << N << "\n";
// Генерация маски
Integer mask = generateMask(N);

// Маскирование подписи
Integer maskedBNum = blindSignature(encryptedBInteger, mask, N);

cout << "Зашифрованное B после наложения слоя ослепляющего шифрования: " <<
maskedBNum << endl;

string maskedB = IntegerToString(maskedBNum);

//подписываем maskedB

Integer signedB_EInteger = generateRSASign(maskedB, ePrivate, rng);
signedB_EInteger = signedB_EInteger.AbsoluteValue();

string signedB_E = IntegerToString(signedB_EInteger);

cout << "Подписанное с помощью e_private замаскированное сообщение: " << signedB_E
<< "\n";

//передали signedB регистратору V и me

//Шаг 3
cout << "\n\tШаг 3\n";
cout << "\nРегистратор V принял это сообщение, а также номер избирателя\n";

RSA::PrivateKey vPrivate;
vPrivate.GenerateRandomWithKeySize(rng, 2048);

RSA::PublicKey vPublic(vPrivate);

cout << "V сгенерировал себе ключи v_public и v_private\n";

cout << "V начинает проверку избирателя.\n";
//Проверяет избирателя на легитимность
bool isLegitimate = false;
for (int j = 0; j < legitimateVoters.size(); j++) {
    if (me == legitimateVoters[j]) {
        isLegitimate = true;
        break;
    }
}
if (!isLegitimate) {
    cout << "\nИзбиратель i" << me << " не легитимный\n";
    continue;
}

```

```

    }
    if (votersChoice[me] != -1) {
        cout << "\nИзбиратель i" << me << " уже голосовал\n";
        continue;
    }
    cout << "Проверки на легитимность и на то, голосовал ли уже кандидат, пройдены\n";

    Integer signedB_VEInteger = generateRSASign(signedB_E, vPrivate, rng);
    signedB_VEInteger = signedB_VEInteger.AbsoluteValue();

    string signedB_VE = IntegerToString(signedB_VEInteger);

    cout << "Подписанное с помощью v_private сообщение: " << signedB_VE << "\n";

    //передает signedB_VE обратно к избирателю E

    //шаг 4
    cout << "\n\tШаг 4\n";

    //снимаем маскирующее шифрование
    cout << "Избиратель получает сообщение от регистратора\n";

    // Демаскирование подписи
    Integer signedB_VEnum_withoutMask = unblindSignature(signedB_VEInteger, mask, N);

    cout << "Сообщение после снятия маскирующего шифрования: " <<
signedB_VEnum_withoutMask << "\n";

    //отправляем signedB_VEnum_withoutMask ЦИК A

    //шаг 5
    cout << "\n\tШаг 5\n";
    string signedB_VEnum_withoutMaskstr = IntegerToString(signedB_VEnum_withoutMask);

    cout << "A начинает проверку подписи V и E.\n";
    bool isValidVSign = checkRSASign(signedB_E, signedB_VEnum_withoutMask, vPublic); //
для проверки E

    if (isValidVSign)
        cout << "Проверка V прошла успешно\n";
    else {
        cout << "Проверка V не прошла\n";
        continue;
    }

    bool isValidESign = checkRSASign(maskedB, signedB_EInteger, ePublic);

    if (isValidESign)
        cout << "Проверка E прошла успешно\n";
    else {
        cout << "Проверка E не прошла\n";
        continue;
    }

    cout << "Зашифрованное сообщение B кладется в специальный список, E получает
квитанцию и может быть свободен.\n";
    votersChoice[me] = encryptedBNum;
    votersSecrets[me] = eSecret;
    //Теперь избиратель свободен

}

```

```

//Шаг 7
cout << "\nА начинает расшифровывать сообщения:";
vector <int> candidatesVoicesCount((int)countOfCandidates, 0);
auto it = votersChoice.begin();
for (int i = 0; i < votersChoice.size() && it != votersChoice.end(); ++i, ++it) {
    if (it->second != -1) {
        string message = boost::lexical_cast<string>(it->second);
        string decryptedAESCandidateNumber = decryptAES(message, votersSecrets[it-
>first]);
        cout << "\nЛегитимный избиратель i" << it->first << " проголосовал за " <<
decryptedAESCandidateNumber;
        cpp_int candidateNumber(decryptedAESCandidateNumber);
        if (candidateNumber >= 0 && candidateNumber < countOfCandidates)
            candidatesVoicesCount[(int)candidateNumber]++;
    }
}

cout << "\n\nВсем приготовиться, А проводит подсчет результатов:\n";
printMessage("1...", 1);
printMessage("2...", 1);
printMessage("3...", 1);
cout << "\nГолоса за кандидатов : \n";
for (int i = 0; i < countOfCandidates; i++) {
    cout << "Кандидат " << i << " набрал " << candidatesVoicesCount[i] << " голосов\n";
}
return;
}

int main(int argc, char* argv[]) {
    setlocale(LC_ALL, "rus");

    int bitsLength;
    int pBits, qBits, nBits;
    bool isN = false;
    string aliceMessage;
    cpp_int aliceSecretMessage;
    string signInfo;
    cpp_int countOfVoters, countOfCandidates;
    for (int i = 0; argv[i]; i++) {
        string checkStr = string(argv[i]);
        if (findInStr(checkStr, 2) == "/h") {
            helpFunc();
            return 0;
        }
        if (checkStr.length() > 2) {
            string ifStr = findInStr(checkStr, 3);
            char symbol = ',';
            if (ifStr == "/c:") {
                countOfCandidates = stoi(checkStr.substr(3, checkStr.length()));
            }
            if (ifStr == "/v:") {
                countOfVoters = stoi(checkStr.substr(3, checkStr.length()));
            }
            if (ifStr == "/l:") {
                bitsLength = stoi(checkStr.substr(3, checkStr.length()));
            }
        }
    }
    sensus_main(countOfVoters, countOfCandidates, bitsLength);
    return 0;
}

```