

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

**Протокол анонимного голосования Хэ-Су**

**ОТЧЁТ**

**ПО ДИСЦИПЛИНЕ**

**«КРИПТОГРАФИЧЕСКИЕ ПРОТОКОЛЫ»**

студента 5 курса 531 группы

специальности 10.05.01 Компьютерная безопасность

факультета компьютерных наук и информационных технологий

Алексеева Александра Александровича

Преподаватель

аспирант

\_\_\_\_\_

Р. А. Фарахутдинов

подпись, дата

Саратов 2023

## СОДЕРЖАНИЕ

1 Теоретическая часть.....	3
1.1 Описание алгоритма.....	3
2 Описание программы.....	5
2.1 Пример работы программы.....	6
3 Листинг кода.....	9

## 1 Теоретическая часть

Цель работы – изучение протокола анонимного голосования Хэ-Су.

### 1.1 Описание алгоритма

Обозначения: **V** – Регистратор, **E** – избиратель, **A** – ЦИК.

#### Шаг 1. V

- утверждает списки легитимных избирателей;
- создаёт  $v_{public}$  и  $v_{private}$  (используются для асимметричного шифрования);
- $v_{public}$  выкладывается в свободный доступ.

#### Шаг 2. E

- создаёт  $e_{public}$  и  $e_{private}$  (используются для подписей);
- вычисляет хеш-функцию от публичного ключа:  $h(e_{public})$ ;
- накладывает слой маскирующего шифрования на  $h(e_{public})$ . Так как шифруется только хеш от ключа, а не длинное сообщение, можно выбрать какой-нибудь простой способ. Например, **E** может сгенерировать случайное число  $x$  и вычислить  $f = \text{encrypt}(v_{public}, x) \cdot h(e_{public})$ ;
- отправляет  $f$  **V**.

#### Шаг 3. V

- проверяет легитимность избирателя;
- дешифрует  $f$ :  $g = \text{decrypt}(v_{private}, \text{encrypt}(v_{public}, x) \cdot h(e_{public})) = x \cdot \text{decrypt}(v_{private}, h(e_{public}))$ . Часть  $e_{public}^{signed} = \text{decrypt}(v_{private}, h(e_{public}))$  считается подписанным ключом;
- отправляет  $g$  **E**.

#### Шаг 4. E

- снимает слой ослепляющего шифрования (умножает на обратный элемент  $x$ ) и получает подписанный ключ  $e_{public}^{signed}$ ;
- проверяет подлинность подписи регистратора: выполняется ли  $\text{encrypt}(v_{public}, \text{decrypt}(v_{private}, h(e_{public}))) = h(e_{public})$ .
- отправляет **A** пару  $\{e_{public}, e_{public}^{signed}\}$ .

### Шаг 5. А

- как и **Е**, проверяет подлинность подписи регистратора;
- проверяет, совпадает ли хеш-функция от  $e_{public}$  в паре с той, что хранится в  $e_{public}^{signed}$ ;
- добавляет  $e_{public}$  в список авторизованных ключей и сообщает об этом **Е**.

### Шаг 6. Е

- создаёт  $e_{secret}$  (используется для шифровки бюллетеней, чтобы ни **А**, ни внешний злоумышленник до нужного времени не мог узнать содержимое бюллетени);
- подготавливает сообщение **В** с выбранным решением;
- отправляет **А** набор  $\{e_{public}, \text{encrypt}(e_{secret}, \mathbf{B}), \text{sign}(e_{private}, h(\text{encrypt}(e_{secret}, \mathbf{B})))\}$ .

### Шаг 7. А

- проверяет авторизованность ключа;
- проверяет подлинность сообщения сравнивая хеш зашифрованного сообщения и хеш, полученный при помощи  $e_{private}$ ;
- публикует тройку в открытом списке.

### Шаг 8

- появление тройки в открытом списке сигнализирует **Е** отправить **А** новый набор:  $\{e_{public}, e_{secret}, \text{sign}(e_{private}, h(e_{secret}))\}$ .

### Шаг 9. А

- проверяет подлинность сообщения, сравнивая хеши;
- расшифровывает ранее полученную бюллетень;
- публикует все данные;
- подсчитывает результат.

### Шаг 10

- после голосования **В** публикует список всех зарегистрировавшихся избирателей, а **А** – список всех авторизованных ключей.

## 2 Описание программы

Программа, представленная ниже, содержит следующие функции:

- $\text{powClosed}(x, y, \text{mod})$  – возводит число  $x$  в степень  $y$  по модулю  $\text{mod}$ ;
- $\text{funEuler}(n)$  – вычисление функции Эйлера от  $n$ ;
- $\text{decForm}(x)$  – представление числа  $x$  в 10-чной системе счисления;
- $\text{usualEuclid}(a, b)$  – вычисление НОД чисел  $a$  и  $b$  обычным алгоритмом Евклида;
- $\text{advancedEuclid}(a, b)$  – вычисление обратного элемента для  $a$  в поле  $b$ ;
- $\text{miller\_rabin}(n, k = 10)$  – проверка числа  $n$  на простоту с вероятностью  $\frac{1}{2^k}$ ;
- $\text{hashStr}(\text{str})$  – выдаёт хэш строки  $\text{str}$ ;
- $\text{generateSimpleNum}(m)$  – генерация простого числа размером  $\approx m$  бит;
- $\text{votingHaeSu}(\text{candidates}, \text{countVoters}, \text{voters})$  – проведение анонимного голосования с помощью протокола Хэ-Су среди кандидатов  $\text{candidates}$  и избирателями  $\text{voters}$ .

## 2.1 Примеры работы программы

```
Протокол тайного голосования Хэ-Су
Количество кандидатов: 5
Список кандидатов: {2, 3, 4, 5, 6}

Количество избирателей: 10
Список избирателей: {1, 2, 3, 4, 5, 6, 7, 8, 9, 10}
Список легитимных избирателей: {1, 3, 5, 6, 8, 10}

Регистратор создаёт ключи vPublic и vPrivate: {76251401763353, 9668962762836092560670861432354815925918671913}. vPublic выкладывается в открытый доступ
```

```
Избиратель 1:
    Создаёт ключи {ePublic, n} и {ePrivate, n}: {1071499171499993, 12744112960027494377596024218393579357050617678525681} и {3348543645929951279474453997689
219308176279020906337, 12744112960027494377596024218393579357050617678525681}
    Вычисляет хэш-функцию h(ePublic): 116042123937339219216195452170648037148
    Накладывает маскирующий слой на h(ePublic): f = 3114061221309630833442144898103557416486976935540871663190427042216221980084686336684
    Отправляет маскирующий слой f Регистратору

Регистратор дешифрует f: g = 15739583630691019253517472179942557415212604563
Отправляет дешифрованное сообщение g Пользователю

Пользователь снимает маскирующий слой и получает подписанный ключ eSigPub = decrypt(vPrivate, h(ePublic)): 217133670481791362798314554950706229218793301
31
    Проверяет подлинность подписи Регистратора (encryption(vPublic, eSigPub) = h(ePublic)): 116042123937339219216195452170648037148 = 1160421239373392192161
95452170648037148
    Проверка пройдена. Пользователь отправляет ЦИКу пару {ePublic, eSigPub} = {1071499171499993, 21713367048179136279831455495070622921879330131}

ЦИК также проверяет подлинность подписи Регистратора (encryption(vPublic, eSigPub) = h(ePublic)): 116042123937339219216195452170648037148 = 116042123937
339219216195452170648037148
    Проверяет, совпадает ли хэш-функция от ePublic в паре с той, что хранится в eSigPub: 116042123937339219216195452170648037148 = 11604212393733921921619545
2170648037148
    Проверки успешно пройдены. ЦИК добавляет авторизированный ключ ePublic в список!

Пользователь создаёт eSecret для шифрования бюллетени: eSecret = 83738313517
Подготавливает сообщение B с выбранным решением: B = 5
Отправляет ЦИКу тройку {ePublic, encrypt(eSecret, B), sign(ePrivate, h(encrypt(eSecret, B)))}: {1071499171499993, 418691567585, 731946331397351488525015
6593399438595483339157741032}

ЦИК проверяет подлинность сообщения (h(encrypt(eSecret, B)) = encrypt(ePublic, sign(ePrivate, h(encrypt(eSecret, B)))): 54111441745601823221383289461116
0621413 = 541114417456018232213832894611160621413
    Проверка пройдена! Тройка публикуется в открытом списке

После появления тройки в открытом доступе пользователь отправляет ЦИКу новую тройку {ePublic, eSecret, sign(ePrivate, h(eSecret))}: {1071499171499993, 8
3738313517, 38323818764032029418209311454230429867440569452724

ЦИК проверяет подлинность (h(eSecret) = encrypt(sign(ePrivate, h(eSecret)))): 24227698522889060272536156123558114836 = 2422769852288906027253615612355811
4836
eSecretRev = 11871931727771493265301841355441022646098552842268996
Расшифровывает полученную бюллетень (eSecret^-1 * encrypt(eSecret, B)): B = 5
Публикует данные и подсчитывает результат.
```

```
Избиратель 3:
    Создаёт ключи {ePublic, n} и {ePrivate, n}: {181146181519, 1616997964911429501175630554525901646485464331997} и {159165684713859918078634257360138141111
418439479, 1616997964911429501175630554525901646485464331997}
    Вычисляет хэш-функцию h(ePublic): 85904941279901415646121649619146549428
    Накладывает маскирующий слой на h(ePublic): f = 333188315051295394860715161631240440370666286807951804979402959243390364281179935572
    Отправляет маскирующий слой f Регистратору

Регистратор дешифрует f: g = 5041592260668843289496106143978307936313889752
Отправляет дешифрованное сообщение g Пользователю

Пользователь снимает маскирующий слой и получает подписанный ключ eSigPub = decrypt(vPrivate, h(ePublic)): 103176907337731716433632724221059363348191004
55
    Проверяет подлинность подписи Регистратора (encryption(vPublic, eSigPub) = h(ePublic)): 85904941279901415646121649619146549428 = 85904941279901415646121
649619146549428
    Проверка пройдена. Пользователь отправляет ЦИКу пару {ePublic, eSigPub} = {181146181519, 10317690733773171643363272422105936334819100455}

ЦИК также проверяет подлинность подписи Регистратора (encryption(vPublic, eSigPub) = h(ePublic)): 85904941279901415646121649619146549428 = 8590494127990
1415646121649619146549428
    Проверяет, совпадает ли хэш-функция от ePublic в паре с той, что хранится в eSigPub: 85904941279901415646121649619146549428 = 859049412799014156461216496
19146549428
    Проверки успешно пройдены. ЦИК добавляет авторизированный ключ ePublic в список!

Пользователь создаёт eSecret для шифрования бюллетени: eSecret = 2559916126277
Подготавливает сообщение B с выбранным решением: B = 5
Отправляет ЦИКу тройку {ePublic, encrypt(eSecret, B), sign(ePrivate, h(encrypt(eSecret, B)))}: {181146181519, 12799580631385, 12601330853177982979823933
44040120040760443943235}

ЦИК проверяет подлинность сообщения (h(encrypt(eSecret, B)) = encrypt(ePublic, sign(ePrivate, h(encrypt(eSecret, B)))): 14189002845840400628116893260905
76085855 = 1418900284584040062811689326090576085855
    Проверка пройдена! Тройка публикуется в открытом списке

После появления тройки в открытом доступе пользователь отправляет ЦИКу новую тройку {ePublic, eSecret, sign(ePrivate, h(eSecret))}: {181146181519, 25599
16126277, 852872694994627817119278303888569586988912158

ЦИК проверяет подлинность (h(eSecret) = encrypt(sign(ePrivate, h(eSecret)))): 1380910305476680677011124980239499194264 = 13809103054766806770111249802394
99194264
eSecretRev = 24527121116125018957455363622620525424338997118
Расшифровывает полученную бюллетень (eSecret^-1 * encrypt(eSecret, B)): B = 5
Публикует данные и подсчитывает результат.
```

```
Избиратель 5:
    Создает ключи {ePublic, n} и {ePrivate, n}: {118777850921, 4409546955899962645146329872908195306766714222728451} и {199788745887888014283014024129842938
5193924893484953, 4409546955899962645146329872908195306766714222728451}
    Вычисляет хэш-функцию h(ePublic): 562424340343239598814061293519972482808
    Накладывает маскирующий слой на h(ePublic): f = 12253027190372968414426334560978916352020347996687249982228093355534684726675714935480
    Отправляет маскирующий слой f Регистратору

    Регистратор дешифрует f: g = 15586092908121662979093686040673823143013368343
    Отправляет дешифрованное сообщение g Пользователю

    Пользователь снимает маскирующий слой и получает подписанный ключ eSigPub = decrypt(vPrivate, h(ePublic)): 208456902416128384346078168668708137996393398
39
    Проверяет подлинность подписи Регистратора (encryption(vPublic, eSigPub) = h(ePublic)): 562424340343239598814061293519972482808 = 5624243403432395988140
61293519972482808
    Проверка пройдена. Пользователь отправляет ЦИКу пару {ePublic, eSigPub} = {118777850921, 20845690241612838434607816866870813799639339839}

    ЦИК также проверяет подлинность подписи Регистратора (encryption(vPublic, eSigPub) = h(ePublic)): 562424340343239598814061293519972482808 = 562424340343
239598814061293519972482808
    Проверяет, совпадает ли хэш-функция от ePublic в паре с той, что хранится в eSigPub: 562424340343239598814061293519972482808 = 56242434034323959881406129
3519972482808
    Проверки успешно пройдены. ЦИК добавляет авторизированный ключ ePublic в список!

    Пользователь создает eSecret для шифрования биллjetени: eSecret = 1543704751
    Подготавливает сообщение B с выбранным решением: B = 2
    Отправляет ЦИКу тройку {ePublic, encrypt(eSecret, B), sign(ePrivate, h(encrypt(eSecret, B)))}: {118777850921, 3087409502, 610947954333682398573496889972
780828007541825670044}

    ЦИК проверяет подлинность сообщения (h(encrypt(eSecret, B)) = encrypt(ePublic, sign(ePrivate, h(encrypt(eSecret, B)))): 17764947731807438377196397997062
6466295 = 177649477318074383771963979970626466295
    Проверка пройдена! Тройка публикуется в открытом списке

    После появления тройки в открытом доступе пользователь отправляет ЦИКу новую тройку {ePublic, eSecret, sign(ePrivate, h(eSecret))}: {118777850921, 15437
04751, 3017720806217942016505770162353724869314864210890342

    ЦИК проверяет подлинность (h(eSecret) = encrypt(sign(ePrivate, h(eSecret)))): 173846085512062970154236498060939291948 = 173846085512062970154236498060939
291948
    eSecretRev = 3275042705532755739093250060103536372096304869643139
    Расшифровывает полученную биллjetень (eSecret^-1 * encrypt(eSecret, B)): B = 2
    Публикует данные и подсчитывает результат.
```

```
Избиратель 6:
    Создает ключи {ePublic, n} и {ePrivate, n}: {1043430229922243, 978203901727003821378212107516118520102704202521704156312927} и {250597633035798171278536
205978082192140244155269407269435627, 978203901727003821378212107516118520102704202521704156312927}
    Вычисляет хэш-функцию h(ePublic): 169067129586389395064185040210384626742
    Накладывает маскирующий слой на h(ePublic): f = 3731832827613177680957657647660740839873170871160378621625269428145364957619298736724
    Отправляет маскирующий слой f Регистратору

    Регистратор дешифрует f: g = 25428694535193317392790130250656703233854840723
    Отправляет дешифрованное сообщение g Пользователю

    Пользователь снимает маскирующий слой и получает подписанный ключ eSigPub = decrypt(vPrivate, h(ePublic)): 156247896520368249848610587640461241456226570
14
    Проверяет подлинность подписи Регистратора (encryption(vPublic, eSigPub) = h(ePublic)): 169067129586389395064185040210384626742 = 1690671295863893950641
85040210384626742
    Проверка пройдена. Пользователь отправляет ЦИКу пару {ePublic, eSigPub} = {1043430229922243, 15624789652036824984861058764046124145622657014}

    ЦИК также проверяет подлинность подписи Регистратора (encryption(vPublic, eSigPub) = h(ePublic)): 169067129586389395064185040210384626742 = 169067129586
389395064185040210384626742
    Проверяет, совпадает ли хэш-функция от ePublic в паре с той, что хранится в eSigPub: 169067129586389395064185040210384626742 = 16906712958638939506418504
0210384626742
    Проверки успешно пройдены. ЦИК добавляет авторизированный ключ ePublic в список!

    Пользователь создает eSecret для шифрования биллjetени: eSecret = 8606616316237
    Подготавливает сообщение B с выбранным решением: B = 2
    Отправляет ЦИКу тройку {ePublic, encrypt(eSecret, B), sign(ePrivate, h(encrypt(eSecret, B)))}: {1043430229922243, 17213232632474, 5598565535208115015895
47159822656013226530289958733232906273}

    ЦИК проверяет подлинность сообщения (h(encrypt(eSecret, B)) = encrypt(ePublic, sign(ePrivate, h(encrypt(eSecret, B)))): 13124642699301089118896427307963
9405667 = 131246426993010891188964273079639405667
    Проверка пройдена! Тройка публикуется в открытом списке

    После появления тройки в открытом доступе пользователь отправляет ЦИКу новую тройку {ePublic, eSecret, sign(ePrivate, h(eSecret))}: {1043430229922243, 8
606616316237, 76422558881389319679486712016367547392490266125176785575436

    ЦИК проверяет подлинность (h(eSecret) = encrypt(sign(ePrivate, h(eSecret)))): 17985344333916798611346457813992781623 = 1798534433391679861134645781399278
1623
    eSecretRev = 678094958421068476321064110442042947093574103602330353735250
    Расшифровывает полученную биллjetень (eSecret^-1 * encrypt(eSecret, B)): B = 2
    Публикует данные и подсчитывает результат.
```

Избиратель 8:  
Создаёт ключи {ePublic, n} и {ePrivate, n}: {465527003629, 87274980170323288980619034880751853985115189} и {59015213759035291955659146663788044003779133, 87274980170323288980619034880751853985115189}  
Вычисляет хэш-функцию h(ePublic): 27565196196326734346141992510451097633  
Накладывает маскирующий слой на h(ePublic): f = 429554498276841048362809972042384789870727938716687833141858075896520124464061156786  
Отправляет маскирующий слой f Регистратору  
  
Регистратор дешифрует f: g = 23822001249645866549532145669976977116667331488  
Отправляет дешифрованное сообщение g Пользователю  
  
Пользователь снимает маскирующий слой и получает подписанный ключ eSigPub = decrypt(vPrivate, h(ePublic)): 108845755242149823021375484895651572007033358  
63  
Проверяет подлинность подписи Регистратора (encryption(vPublic, eSigPub) = h(ePublic)): 27565196196326734346141992510451097633 = 27565196196326734346141992510451097633  
Проверка пройдена. Пользователь отправляет ЦИКу пару {ePublic, eSigPub} = {465527003629, 10884575524214982302137548489565157200703335863}  
ЦИК также проверяет подлинность подписи Регистратора (encryption(vPublic, eSigPub) = h(ePublic)): 27565196196326734346141992510451097633 = 27565196196326734346141992510451097633  
Проверяет, совпадает ли хэш-функция от ePublic в паре с той, что хранится в eSigPub: 27565196196326734346141992510451097633 = 27565196196326734346141992510451097633  
Проверки успешно пройдены. ЦИК добавляет авторизированный ключ ePublic в список!  
  
Пользователь создаёт eSecret для шифрования бюллетени: eSecret = 2177816265793951  
Подготавливает сообщение B с выбранным решением: B = 5  
Отправляет ЦИКу тройку {ePublic, encrypt(eSecret, B), sign(ePrivate, h(encrypt(eSecret, B)))}: {465527003629, 10889081328969755, 81800751441658736753084168660212138976343695}  
ЦИК проверяет подлинность сообщения (h(encrypt(eSecret, B)) = encrypt(ePublic, sign(ePrivate, h(encrypt(eSecret, B))))): 828428653408976171510051651234164788270 = 828428653408976171510051651234164788270  
Проверка пройдена! Тройка публикуется в открытом списке  
  
После появления тройки в открытом доступе пользователь отправляет ЦИКу новую тройку {ePublic, eSecret, sign(ePrivate, h(eSecret)))}: {465527003629, 2177816265793951, 56160202354609254616685978288749462637650975  
652138  
ЦИК проверяет подлинность (h(eSecret) = encrypt(sign(ePrivate, h(eSecret)))): 152326091726051024732085742891576652138 = 152326091726051024732085742891576652138  
eSecretRev = 68792070890964587489574596093554705944194833  
Расшифровывает полученную бюллетень (eSecret^-1 \* encrypt(eSecret, B)): B = 5  
Публикует данные и подсчитывает результат.

Избиратель 10:  
Создаёт ключи {ePublic, n} и {ePrivate, n}: {2598794399, 6028348942443463546507033714463781293026027742459} и {2399216632057373127491399436898130549183603169119, 6028348942443463546507033714463781293026027742459}  
Вычисляет хэш-функцию h(ePublic): 153889933113212063149385439769612179883  
Накладывает маскирующий слой на h(ePublic): f = 285875257117533074517185285118436907860837363512227373966957489010475086764464007433  
Отправляет маскирующий слой f Регистратору  
  
Регистратор дешифрует f: g = 20696801868552072142337193391131443951963828030  
Отправляет дешифрованное сообщение g Пользователю  
  
Пользователь снимает маскирующий слой и получает подписанный ключ eSigPub = decrypt(vPrivate, h(ePublic)): 182973771657431630256753892843077440541687718  
23  
Проверяет подлинность подписи Регистратора (encryption(vPublic, eSigPub) = h(ePublic)): 153889933113212063149385439769612179883 = 153889933113212063149385439769612179883  
Проверка пройдена. Пользователь отправляет ЦИКу пару {ePublic, eSigPub} = {2598794399, 18297377165743163025675389284307744054168771823}  
ЦИК также проверяет подлинность подписи Регистратора (encryption(vPublic, eSigPub) = h(ePublic)): 153889933113212063149385439769612179883 = 153889933113212063149385439769612179883  
Проверяет, совпадает ли хэш-функция от ePublic в паре с той, что хранится в eSigPub: 153889933113212063149385439769612179883 = 153889933113212063149385439769612179883  
Проверки успешно пройдены. ЦИК добавляет авторизированный ключ ePublic в список!  
  
Пользователь создаёт eSecret для шифрования бюллетени: eSecret = 1180400031011  
Подготавливает сообщение B с выбранным решением: B = 3  
Отправляет ЦИКу тройку {ePublic, encrypt(eSecret, B), sign(ePrivate, h(encrypt(eSecret, B)))}: {2598794399, 3541200093033, 3974100613247888154178009755191505712878092460062}  
ЦИК проверяет подлинность сообщения (h(encrypt(eSecret, B)) = encrypt(ePublic, sign(ePrivate, h(encrypt(eSecret, B))))): 1756768372629909385212093329481192440935 = 1756768372629909385212093329481192440935  
Проверка пройдена! Тройка публикуется в открытом списке  
  
После появления тройки в открытом доступе пользователь отправляет ЦИКу новую тройку {ePublic, eSecret, sign(ePrivate, h(eSecret)))}: {2598794399, 1180400031011, 337160012562185939479941411752603172535786008968  
8607  
ЦИК проверяет подлинность (h(eSecret) = encrypt(sign(ePrivate, h(eSecret)))): 54262405665652174415184809967786998607 = 54262405665652174415184809967786998607  
eSecretRev = 4951028593525676858523290638529557877174138201577  
Расшифровывает полученную бюллетень (eSecret^-1 \* encrypt(eSecret, B)): B = 3  
Публикует данные и подсчитывает результат.

После голосования:  
Регистратор публикует список всех зарегистрировавшихся избирателей: {1, 2, 3, 4, 5, 6, 7, 8, 9, 10}  
ЦИК публикует список всех авторизованных ключей: {2598794399, 11877850921, 181146181519, 465527003629, 104343022992243, 1071499171499993}  
Результаты голосования:  
Кандидат 2 = 2  
Кандидат 3 = 1  
Кандидат 4 = 0  
Кандидат 5 = 3  
Кандидат 6 = 0



### 3 Листинг кода

```
#include <iostream>
#include <vector>
#include <string>
#include <map>
#include <set>
#include <boost/multiprecision/cpp_int.hpp>

using namespace std;
using namespace boost::multiprecision;

class Pattern
{
private:
    static vector <cpp_int> deg2(cpp_int el, cpp_int n) { //Раскладываем
число на степени двойки
        vector <cpp_int> res;
        while (n != 0) {
            if (n / el == 1) {
                res.push_back(el);
                n -= el;
                el = 1;
            }
            else
                el *= 2;
        }
        return res;
    }

    static cpp_int multMod(cpp_int n, cpp_int mod, vector <pair <cpp_int,
cpp_int>> lst) { //Умножаем число по модулю
        if (lst.size() == 1) {
            cpp_int res = 1;
            for (int i = 0; i < lst[0].second; i++)
                res = res * lst[0].first % mod;
            return res;
        }
        else if (lst[0].second == 1) {
            cpp_int el = lst[0].first;
            lst.erase(lst.begin());
            return (el * multMod(n, mod, lst)) % mod;
        }
        else {
            for (int i = 0; i < lst.size(); i++)
                if (lst[i].second > 1) {
                    lst[i].first = (lst[i].first *
lst[i].first) % mod;
                    lst[i].second /= 2;
                }
            return multMod(n, mod, lst);
        }
    }

public:
    static cpp_int powClosed(cpp_int x, cpp_int y, cpp_int mod) { //Возводим
число в степени по модулю
        if (y == 0)
            return 1;

        vector <cpp_int> lst = deg2(1, y);
        vector <pair <cpp_int, cpp_int>> xDeps;
```

```

        for (int i = 0; i < lst.size(); i++)
            xDeps.push_back(make_pair(x, lst[i]));

        cpp_int res = multMod(x, mod, xDeps);
        return res;
    }

    static cpp_int funEuler(cpp_int n) {
        cpp_int res = 1;
        for (int i = 2; i < n; i++)
            if (usualEuclid(n, i) == 1)
                res++;
        return res;
    }

    static cpp_int decForm(string x) {
        cpp_int res = 0, deg = 1;
        if (!x.empty() && x.back() == '1')
            res += 1;
        for (short i = x.length() - 2; i >= 0; i--) {
            deg = deg * 2;
            if (x[i] == '1')
                res += deg;
        }
        return res;
    }

    static cpp_int usualEuclid(cpp_int a, cpp_int b) {
        if (a < b)
            swap(a, b);
        if (a < 0 || b < 0)
            throw string{ "Выполнение невозможно: a < 0 или b < 0" };
        else if (b == 0)
            return a;

        cpp_int r = a % b;
        return usualEuclid(b, r);
    }

    static pair <cpp_int, cpp_int> advancedEuclid(cpp_int a, cpp_int b) {
        if (a < 0 || b < 0)
            throw string{ "Выполнение невозможно: a < 0 или b < 0" };

        cpp_int q, aPrev = a, aCur = b, aNext = -1;
        cpp_int xPrev = 1, xCur = 0, xNext;
        cpp_int yPrev = 0, yCur = 1, yNext;
        while (aNext != 0) {
            q = aPrev / aCur;
            aNext = aPrev % aCur;
            aPrev = aCur; aCur = aNext;

            xNext = xPrev - (xCur * q);
            xPrev = xCur; xCur = xNext;

            yNext = yPrev - (yCur * q);
            yPrev = yCur; yCur = yNext;
        }

        return make_pair(xPrev, yPrev);
    }

```

```

}

static bool miller_rabin(cpp_int n, int k = 10) {
    if (n == 0 || n == 1)
        return false;

    cpp_int d = n - 1;
    cpp_int s = 0;
    while (d % 2 == 0) {
        s++;
        d = d / 2;
    }

    cpp_int nDec = n - 1;
    for (int i = 0; i < k; i++) {
        cpp_int a = rand() % nDec;
        if (a == 0 || a == 1)
            a = a + 2;

        cpp_int x = powClosed(a, d, n);
        if (x == 1 || x == nDec)
            continue;

        bool flag = false;
        for (int j = 0; j < s; j++) {
            x = (x * x) % n;
            if (x == nDec) {
                flag = true;
                break;
            }
        }
        if (!flag)
            return false;
    }

    return true;
}

static cpp_int hashStr(string str) {
    while (str.length() < 2)
        str += " ";
    string res = "";
    hash <string> hashStr;

    unsigned short offset = ceil(str.length() / 2.0);
    for (unsigned short i = 0; i < str.length(); i += offset)
        res += to_string(hashStr(str.substr(i, offset)));
    return cpp_int(res);
}

static cpp_int generateSimpleNum(unsigned short m) {
    cpp_int q = rand() % 1000;
    while (Pattern::funEuler(q) != q - 1)
        q++;

    cpp_int s, n = 2, nDec;
    while (!Pattern::miller_rabin(n)) {
        string sBin = "1";
        int sBinSize = rand() % (m / 2) + m / 2;
        for (int i = 0; i < sBinSize; i++)

```

```

        sBin = sBin + to_string(rand() % 2);
        s = Pattern::decForm(sBin);

        n = (q * s) + 1;
        nDec = n - 1;
    }

    return n;
}

};

class RSA
{
private:
    cpp_int generateSimpleNum(unsigned short m) {
        cpp_int q = rand() % 1000;
        while (Pattern::funEuler(q) != q - 1)
            q++;

        cpp_int s, n = 2, nDec;
        while (!Pattern::miller_rabin(n)) {
            string sBin = "1";
            int sBinSize = rand() % (m / 2) + m / 2;
            for (int i = 0; i < sBinSize; i++)
                sBin = sBin + to_string(rand() % 2);
            s = Pattern::decForm(sBin);

            n = (q * s) + 1;
            nDec = n - 1;
        }

        return n;
    }

public:
    cpp_int publicKey;
    cpp_int privateKey;
    cpp_int n, phiN;

    RSA(int m)
    {
        cpp_int p = generateSimpleNum(m), q = generateSimpleNum(m);
        this->n = p * q;
        this->phiN = (p - 1) * (q - 1);

        this->publicKey = generateSimpleNum(m / 2);
        this->privateKey = Pattern::advancedEuclid(publicKey,
phiN).first;
        while (privateKey < 0)
            privateKey += phiN;
    }

    cpp_int encrypt(cpp_int m)
    {
        if (m < 2 || m > n - 1)
            throw string("Ошибка при шифровании: сообщение должно
лежать в промежутке от 2 до " + to_string(this->n));
        return Pattern::powClosed(m, this->publicKey, this->n);
    }
}

```

```

        cpp_int decrypt(cpp_int c)
        {
            return Pattern::powClosed(c, this->privateKey, this->n);
        }
};

void votingHaeSu(int candidates, int countVoters, vector<int> voters)
{
    RSA v(100);
    cout << "\n\n\tРегистратор создаёт ключи vPublic и vPrivate: {" <<
v.publicKey << ", " << v.privateKey << "}. vPublic выкладывается в открытый
доступ";

    set<cpp_int> authorizedKeys;
    map<cpp_int, unsigned short> votes;
    for (unsigned short i = 2; i <= candidates; i++)
        votes.insert(make_pair(i, 0));

    for (unsigned short i = 0; i < voters.size(); i++)
    {
        cout << "\n\n\n\tИзбиратель " << voters[i] << ": ";
        RSA e(100);
        cout << "\n\t\tСоздаёт ключи {ePublic, n} и {ePrivate, n}: {" <<
e.publicKey << ", " << e.n << "} и {" << e.privateKey << ", " << e.n << "};

        cpp_int hashEpublic = Pattern::hashStr(to_string(e.publicKey));
        cout << "\n\t\tВычисляет хэш-функцию h(ePublic): " <<
hashEpublic;

        cpp_int x = Pattern::generateSimpleNum(50);
        cpp_int f = v.encrypt(x) * hashEpublic;
        cout << "\n\t\tНакладывает маскирующий слой на h(ePublic): f = "
<< f << "\n\t\tОтправляет маскирующий слой f Регистратору";

        cpp_int g = v.decrypt(f);
        cout << "\n\n\t\tРегистратор дешифрует f: g = " << g <<
"\n\t\tОтправляет дешифрованное сообщение g Пользователю";

        cpp_int xRev = Pattern::advancedEuclid(x, v.n).first;
        while (xRev < 0)
            xRev += v.n;
        cpp_int eSigPub = g * xRev % v.n;
        cout << "\n\n\t\tПользователь снимает маскирующий слой и получает
подписанный ключ eSigPub = decrypt(vPrivate, h(ePublic)): " << eSigPub;

        cpp_int leftSide = v.encrypt(eSigPub);
        cout << "\n\t\tПроверяет подлинность подписи Регистратора
(encryption(vPublic, eSigPub) = h(ePublic)): " << leftSide << " = " <<
hashEpublic;
        if (leftSide != hashEpublic)
            throw string("Регистратор не прошёл проверку!");
        cout << "\n\t\tПроверка пройдена. Пользователь отправляет ЦИКу
пару {ePublic, eSigPub} = {" << e.publicKey << ", " << eSigPub << "};

        cout << "\n\n\t\tЦИК также проверяет подлинность подписи
Регистратора (encryption(vPublic, eSigPub) = h(ePublic)): " << leftSide << " = "
<< hashEpublic;
        cout << "\n\t\tПроверяет, совпадет ли хэш-функция от ePublic в
паре с той, что хранится в eSigPub: ";
        cout << Pattern::hashStr(to_string(e.publicKey)) << " = " <<
v.encrypt(eSigPub);
    }
}

```

```

        cout << "\n\t\tПроверки успешно пройдены. ЦИК добавляет
авторизированный ключ ePublic в список!";
        authorizedKeys.insert(e.publicKey);

        cpp_int eSecret = Pattern::generateSimpleNum(50);
        cout << "\n\n\t\tПользователь создаёт eSecret для шифрования
биллютени: eSecret = " << eSecret;
        cpp_int B = rand() % (candidates - 2) + 2;
        cout << "\n\t\tПодготавливает сообщение B с выбранным решением: B
= " << B;

        cpp_int Bsecret = B * eSecret % e.n;
        cpp_int Bsign = e.decrypt(Pattern::hashStr(to_string(Bsecret)));
        cout << "\n\t\tОтправляет ЦИКу тройку {ePublic, encrypt(eSecret,
B), sign(ePrivate, h(encrypt(eSecret, B)))}: {"";
        cout << e.publicKey << ", " << Bsecret << ", " << Bsign << "}}";

        leftSide = Pattern::hashStr(to_string(Bsecret));
        cpp_int rightSide = e.encrypt(Bsign);
        cout << "\n\n\t\tЦИК проверяет подлинность сообщения
(h(encrypt(eSecret, B)) = encrypt(ePublic, sign(ePrivate, h(encrypt(eSecret,
B))))): ";
        cout << leftSide << " = " << rightSide;
        if (leftSide != rightSide)
        {
            cout << "\nСООБЩЕНИЕ НЕ ПРОШЛО ПРОВЕРКУ!";
            continue;
        }
        cout << "\n\t\tПроверка пройдена! Тройка публикуется в открытом
списке";

        cpp_int eSigSecret =
e.decrypt(Pattern::hashStr(to_string(eSecret)));
        cout << "\n\n\t\tПосле появления тройки в открытом доступе
пользователь отправляет ЦИКу новую тройку {ePublic, eSecret, sign(ePrivate,
h(eSecret))}: {"";
        cout << e.publicKey << ", " << eSecret << ", " << eSigSecret;

        leftSide = Pattern::hashStr(to_string(eSecret));
        rightSide = e.encrypt(eSigSecret);
        cout << "\n\n\t\tЦИК проверяет подлинность (h(eSecret) =
encrypt(sign(ePrivate, h(eSecret)))): " << leftSide << " = " << rightSide;
        if (leftSide != rightSide)
        {
            cout << "\nСООБЩЕНИЕ НЕ ПРОШЛО ПРОВЕРКУ!";
            continue;
        }

        cpp_int eSecretRev = Pattern::advancedEuclid(eSecret, e.n).first;
        while (eSecretRev < 0)
            eSecretRev += e.n;
        cout << "\neSecretRev = " << eSecretRev;
        cpp_int res = Bsecret * eSecretRev % e.n;
        cout << "\n\t\tРасшифровывает полученную бюллетень (eSecret^-1 *
encrypt(eSecret, B)): B = " << res;
        cout << "\n\t\tПубликует данные и подсчитывает результат.";

        votes[B]++;
    }

    cout << "\n\nПосле голосования: ";
    cout << "\n\tРегистратор публикует список всех зарегистрировавшихся
избирателей: {"";

```

```

for (unsigned short i = 1; i <= countVoters; i++)
{
    if (i == countVoters)
        cout << i << "}";
    else
        cout << i << ", ";
}

cout << "\n\tЦИК публикует список всех авторизованных ключей: {";
for (auto i = authorizedKeys.begin(); i != authorizedKeys.end(); i++)
{
    if (i == --authorizedKeys.end())
        cout << *i << "}";
    else
        cout << *i << ", ";
}

cout << "\n\tРезультаты голосования: ";
for (auto i = votes.begin(); i != votes.end(); i++)
    cout << "\n\t\tКандидат " << i->first << " = " << i->second;
}

int main()
{
    srand(time(NULL));
    setlocale(LC_ALL, "ru");
    cout << "Протокол тайного голосования Хэ-Су";

    int candidates;
    cout << "\n\tКоличество кандидатов: ";
    cin >> candidates;
    cout << "\tСписок кандидатов: {";
    candidates++;
    for (int i = 2; i <= candidates; i++)
    {
        if (i == candidates)
            cout << i << "}";
        else
            cout << i << ", ";
    }

    int countVoters;
    cout << "\n\n\tКоличество избирателей: ";
    cin >> countVoters;
    cout << "\tСписок избирателей: {";
    for (int i = 1; i <= countVoters; i++)
    {
        if (i == countVoters)
            cout << i << "}";
        else
            cout << i << ", ";
    }

    vector<int> voters;
    string str = "\n\tСписок легитимных избирателей: {";
    for (int i = 1; i <= countVoters; i++)
        if (rand() % 2 == 1)
        {
            voters.push_back(i);
            str += to_string(i) + ", ";
        }
    str.erase(str.length() - 2);
    cout << str << "}";
}

```

```
    votingHaeSu(candidates, countVoters, voters);  
  
    cout << endl;  
    return 0;  
}
```