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
профессор, д.ф.-м.н. _____ В. Е. Новиков
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 ассиметричного 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 = encrypt(v_{public}, x) · h(e_{public});

• отправляет f **V**.

Шаг 3. V

• проверяет легитимность избирателя;

• дешифрует f: g = decrypt($v_{private}$, encrypt(v_{public}, x) · h(e_{public})) = x · decrypt($v_{private}$, h(e_{public})). Часть e_{public}^{signed} = 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 encrypt(v_{public}, decrypt($v_{private}$, h(e_{public})) = h(e_{public}).

• отправляет **A** пару {e_{public}, e_{public}^{signed}}.

Шаг 5. A

• как и **E**, 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 авторизированных ключей и сообщает об этом **E**.

Шаг 6. E

• создаёт e_{secret} (используется для шифровки бюллетеней, чтобы ни **A**, ни внешний злоумышленник до нужного времени не мог узнать содержимое беллютени);

• подготавливает сообщение **B** с выбранным решение;

• отправляет **A** набор {e_{public}, encrypt(e_{secret}, **B**), sign($e_{private}$, h(encrypt(e_{secret}, **B**))}.

Шаг 7. A

• 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 **E** отправить **A** новый набор: {e_{public}, e_{secret}, sign($e_{private}$, h(e_{secret}))}.

Шаг 9. A

• проверяет подлинность сообщения, сравнивая хеши;

• расшифровывает ранее полученную бюллетень;

• публикует все данные;

• подсчитывает результат.

Шаг 10

• после голосования **V** публикует список всех зарегистрировавшихся избирателей, а **A** – список всех авторизованных ключей.

2 Описание программы

Программа, представленная ниже, содержит следующие функции:

• powClosed(*x*, *y*, *mod*) – возводит число *x* в степень *y* по модулю *mod*;

• funEuler(*n*) – вычисление функции Эйлера от *n*;

• decForm(*x*) – представление числа *x* в 10-чной системе счисления;

• usualEuclid(*a*, *b*) – вычисление НОД чисел *a* и *b* обычным алгоритмом Евклида;

• advancedEuclid(*a*, *b*) – вычисление обратного элемента для *a* в поле *b*;

• miller_rabin(*n*, *k* = 10) – проверка числа *n* на простоту с вероятностью $\frac{1}{2^k}$;

• hashStr(*str*) – выдаёт хэш строки *str*;

• generateSimpleNum(*m*) – генерация простого числа размером ≈ *m* бит;

• votingHaeSu(*candidates*, *countVoters*, *voters*) – проведение анонимного голосования с помощью протокола Хэ-Су среди кандидатов *candidates* и избирателями *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, 966896276283609256067086143235481592591867191 3}. vPublic выкладывается в открытый доступ
```

```
        Избиратель 1:
                Создаёт ключи {ePublic, n} и {ePrivate, n}: {1071499171499993, 127441129600274943775960242183935793570506176785256 81} и {33485436459299512794744539976892193081762790209063 37, 127441129600274943775960242183935793570506176785256 81}
                Вычисляет хэш-функцию h(ePublic): 116042123937339219216195452170648037148
                Накладывает маскирующий слой на h(ePublic): f = 311406122130963083344214489810355741648697693554087166319042704221622198008468633 6684
                Отправляет маскирующий слой f Регистратору

                Регистратор дешифрует f: g = 157395836306910192535174721799425574152126045 63
                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 = 116042123937339219216
954521706480371 48
                Проверка пройдена. Пользователь отправляет ЦИКу пару {ePublic, eSigPub} = {1071499171499993, 217133670481791362798314554950706229218793301 31}

                ЦИК также проверяет подлинность подписи Регистратора (encryption(vPublic, eSigPub) = h(ePublic)): 116042123937339219216195452170648037148 = 116042123937
339219216195452170648037148
                Проверяет, совпадет ли хэш-функция от ePublic в паре с той, что хранится в eSigPub: 116042123937339219216195452170648037148 = 116042123937339219216195452
170648037148
                Проверки успешно пройдены. ЦИК добавляет авторизированный ключ ePublic в список!

                Пользователь создаёт eSecret для шифрования биллютени: 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
659339943859548333915774103 2}

                ЦИК проверяет подлинность сообщения (h(encrypt(eSecret, B)) = encrypt(ePublic, sign(ePrivate, h(encrypt(eSecret, B)))): 541114417456018232213832894611160621413
 = 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, 383238187640320294182093114542304298674405694527 24

                ЦИК проверяет подлинность (h(eSecret) = encrypt(sign(ePrivate, h(eSecret))): 242276985228890602725361561235581148 36 = 242276985228890602725361561235581
14836
eSecretRev = 118719317277714932653018413554410226460985528422689 96
                Расшифровывает полученную бюллетень (eSecret^-1 * encrypt(eSecret, B)): B = 5
                Пубилкует данные и подсчитывает результат.
```

```
        Избиратель 3:
                Создаёт ключи {ePublic, n} и {ePrivate, n}: {181146181519, 161699796491142950117563055452590164648546433199 7} и {159165684713859918078634257360138141111
418439479, 161699796491142950117563055452590164648546433199 7}
                Вычисляет хэш-функцию h(ePublic): 859049412799014156461216496191465494 28
                Накладывает маскирующий слой на h(ePublic): f = 333188315051295394860715161631240440370666286807951804979402959243390364281179935572
                Отправляет маскирующий слой f Регистратору

                Регистратор дешифрует f: g = 504159226066884328949610614397830793631388975 2
                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)): 859049412799014156461216496191465494 28 = 859049412799014156461216
49619146549428
                Проверка пройдена. Пользователь отправляет ЦИКу пару {ePublic, eSigPub} = {181146181519, 103176907337731716433632724221059363348191004 55}

                ЦИК также проверяет подлинность подписи Регистратора (encryption(vPublic, eSigPub) = h(ePublic)): 859049412799014156461216496191465494 28 = 859049412799
014156461216496191465494 28
                Проверяет, совпадет ли хэш-функция от ePublic в паре с той, что хранится в eSigPub: 859049412799014156461216496191465494 28 = 859049412799014156461216496
19146549428
                Проверки успешно пройдены. ЦИК добавляет авторизированный ключ ePublic в список!

                Пользователь создаёт eSecret для шифрования биллютени: 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, 126013308531779829798239334
4404012004076044394 3235}

                ЦИК проверяет подлинность сообщения (h(encrypt(eSecret, B)) = encrypt(ePublic, sign(ePrivate, h(encrypt(eSecret, B)))): 141890028458404006281168932609057608585 5
 = 141890028458404006281168932609057608585 5
                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, 852872694994627817119278303888856958698891215 8

                ЦИК проверяет подлинность (h(eSecret) = encrypt(sign(ePrivate, h(eSecret))): 138091030547668067701112498023949919426 4 = 138091030547668067701112498023949
99194264
eSecretRev = 245271211161250189574555363622620525424338997118
                Расшифровывает полученную бюллетень (eSecret^-1 * encrypt(eSecret, B)): B = 5
                Пубилкует данные и подсчитывает результат.
```

```
        Избиратель 5:
                Создаёт ключи {ePublic, n} и {ePrivate, n}: {118777850921, 44095469558999626451463298729081953067667142227284 51} и {19978874588788801428304102412984293851939248934849 53, 44095469558999626451463298729081953067667142227284 51}
                Вычисляет хэш-функцию h(ePublic): 562424340343239598814061293519972482808
                Накладывает маскирующий слой на h(ePublic): f = 122530271903729684144263345609789163520203479966872499822280933555346847266757149354 80
                Отправляет маскирующий слой f Регистратору

                Регистратор дешифрует f: g = 155860929081216629790936860406738231430133683 43
                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 = 562424340343239598814061293519972482808
                Проверка пройдена. Пользователь отправляет ЦИКу пару {ePublic, eSigPub} = {118777850921, 208456902416128384346078168668708137996393398 39}

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

                Пользователь создаёт eSecret для шифрования биллютени: 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)))): 177649477318074383771963979970626466295 = 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, 1543704751, 301772080621794201650577016235372486931486421089034 2}

                ЦИК проверяет подлинность (h(eSecret) = encrypt(sign(ePrivate, h(eSecret))): 173846085512062970154236498060939291948 = 173846085512062970154236498060939291948
eSecretRev = 327504270553275573909325006010353637209630486946431 39
                Расшифровывает полученную бюллетень (eSecret^-1 * encrypt(eSecret, B)): B = 2
                Пубилкует данные и подсчитывает результат.
```

```
        Избиратель 6:
                Создаёт ключи {ePublic, n} и {ePrivate, n}: {1043430229922243, 97820390172700382137821210751611852010270420252170415631292 7} и {25059763303579817127853620597808219214024415526940726943562 7, 97820390172700382137821210751611852010270420252170415631292 7}
                Вычисляет хэш-функцию h(ePublic): 169067129586389395064185040210384626742
                Накладывает маскирующий слой на h(ePublic): f = 373183282761317768095765764766074083987317087116037862162526942814536495761929873672 4
                Отправляет маскирующий слой f Регистратору

                Регистратор дешифрует f: g = 254286945351933173927901302506567032338548407 23
                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 = 169067129586389395064185040210384626742
                Проверка пройдена. Пользователь отправляет ЦИКу пару {ePublic, eSigPub} = {1043430229922243, 156247896520368249848610587640461241456226570 14}

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

                Пользователь создаёт eSecret для шифрования биллютени: 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 4715982265601322653028995873323290627 3}

                ЦИК проверяет подлинность сообщения (h(encrypt(eSecret, B)) = encrypt(ePublic, sign(ePrivate, h(encrypt(eSecret, B)))): 131246426993010891188964273079639405667 = 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, 8606616316237, 7642255888813893196794867120163675473924902661251767855754 36}

                ЦИК проверяет подлинность (h(eSecret) = encrypt(sign(ePrivate, h(eSecret))): 179853443339167986113464578139927816 23 = 179853443339167986113464578139927816 23
eSecretRev = 678094958421068476321064110442042947093574103602330353735250
                Расшифровывает полученную бюллетень (eSecret^-1 * encrypt(eSecret, B)): B = 2
                Пубилкует данные и подсчитывает результат.
```

```
        Избиратель 8:
                Создаёт ключи {ePublic, n} и {ePrivate, n}: {465527003629, 872749801703232889806190348807518539851151 89} и {590152137590352919556591466637880440037791338
, 872749801703232889806190348807518539851151 89}
                Вычисляет хэш-функцию h(ePublic): 27565196196326734346141992510451097633
                Накладывает маскирующий слой на h(ePublic): f = 42955449827684104836280997204238478987072793871668783314185807589652012446406115678 6
                Отправляет маскирующий слой f Регистратору

                Регистратор дешифрует f: g = 2382200124964586654953214566997697711666733148 8
                Отправляет дешифрованное сообщение g Пользователю

                Пользователь снимает маскирующий слой и получает подписанный ключ eSigPub = decrypt(vPrivate, h(ePublic)): 1088457552421498230213754848956515720070333583
63
                Проверяет подлинность подписи Регистратора (encryption(vPublic, eSigPub) = h(ePublic)): 27565196196326734346141992510451097633 = 2756519619632673434614199251045109763 3
992510451097633
                Проверка пройдена. Пользователь отправляет ЦИКу пару {ePublic, eSigPub} = {465527003629, 1088457552421498230213754848956515720070333586 3}

                ЦИК также проверяет подлинность подписи Регистратора (encryption(vPublic, eSigPub) = h(ePublic)): 27565196196326734346141992510451097633 = 2756519619632
6734346141992510451097633
                Проверяет, совпадет ли хэш-функция от ePublic в паре с той, что хранится в eSigPub: 27565196196326734346141992510451097633 = 2756519619632673434614199251
10451097633
                Проверки успешно пройдены. ЦИК добавляет авторизированный ключ ePublic в список!

                Пользователь создаёт eSecret для шифрования биллютени: 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, 818007514416587367530841686602121389763436 95}

                ЦИК проверяет подлинность сообщения (h(encrypt(eSecret, B)) = encrypt(ePublic, sign(ePrivate, h(encrypt(eSecret, B)))): 828428653408976171510051651234164788270 =
4788270 = 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, 21778
16265793951, 5616020235460925461668597828874946263765097 5}

                ЦИК проверяет подлинность (h(eSecret) = encrypt(sign(ePrivate, h(eSecret))): 152326091726051024732085742891576652138 = 1523260917260510247320857428915766
52138
eSecretRev = 687920708909645874895745960935547059441948 33
                Расшифровывает полученную бюллетень (eSecret^-1 * encrypt(eSecret, B)): B = 5
                Пубилкует данные и подсчитывает результат.
```

```
        Избиратель 10:
                Создаёт ключи {ePublic, n} и {ePrivate, n}: {2598794399, 602834894244346354650703371446378129302602774245 9} и {2399216632057373127491399436898130549183603169119, 602834894244346354650703371446378129302602774245 9}
                Вычисляет хэш-функцию h(ePublic): 1538899331132120631493854397696121798 83
                Накладывает маскирующий слой на h(ePublic): f = 2858752571175330745171852851184369078608373635122227373966957489010475086764464007433
                Отправляет маскирующий слой f Регистратору

                Регистратор дешифрует f: g = 2069680186855207214233719339113144395196382803 0
                Отправляет дешифрованное сообщение g Пользователю

                Пользователь снимает маскирующий слой и получает подписанный ключ eSigPub = decrypt(vPrivate, h(ePublic)): 1829737716574316302567538928430774405416877182
3
                Проверяет подлинность подписи Регистратора (encryption(vPublic, eSigPub) = h(ePublic)): 1538899331132120631493854397696121798 83 = 1538899331132120631493
85439769612179883
                Проверка пройдена. Пользователь отправляет ЦИКу пару {ePublic, eSigPub} = {2598794399, 1829737716574316302567538928430774405416877182 3}

                ЦИК также проверяет подлинность подписи Регистратора (encryption(vPublic, eSigPub) = h(ePublic)): 1538899331132120631493854397696121798 83 = 153889933113
212063149385439769612179883
                Проверяет, совпадет ли хэш-функция от ePublic в паре с той, что хранится в eSigPub: 1538899331132120631493854397696121798 83 = 1538899331132120631493854 3
9769612179883
                Проверки успешно пройдены. ЦИК добавляет авторизированный ключ ePublic в список!

                Пользователь создаёт eSecret для шифрования биллютени: 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, 3974100613247888154178009755 1
915057128780924600 62}

                ЦИК проверяет подлинность сообщения (h(encrypt(eSecret, B)) = encrypt(ePublic, sign(ePrivate, h(encrypt(eSecret, B)))): 1756768372629909385212093329481192440935 =
92440935 = 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, 1180400
031011, 3371600125621859394799414117526031725357860089 68}

                ЦИК проверяет подлинность (h(eSecret) = encrypt(sign(ePrivate, h(eSecret))): 5426240566565217441518480996778699 8607
eSecretRev = 4951028593525676858523290638529557877174138201577
                Расшифровывает полученную бюллетень (eSecret^-1 * encrypt(eSecret, B)): B = 3
                Пубилкует данные и подсчитывает результат.
```

```
После голосования:
        Регистратор публикует список всех зарегистрировашихся избирателей: {1, 2, 3, 4, 5, 6, 7, 8, 9, 10}
        ЦИК публикует список всех авторизованных ключей: {2598794399, 118777850921, 181146181519, 465527003629, 1043430229922243, 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>> xDegs;
```

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

        cpp_int res = multMod(x, mod, xDegs);
        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\tПубилкует данные и подсчитывает результат.";

                votes[B]++;
        }

        cout << "\n\nПосле голосования: ";
        cout << "\n\tРегистратор публикует список всех зарегистрировашихся
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;
}
```