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

Временная метка как одна из задач аутентификации

КУРСОВАЯ РАБОТА

студентки 2 курса 232 группы
специальности 10.05.01 Компьютерная безопасность
факультета компьютерных наук и информационных технологий
Ежовой Елены Дмитриевны

Научный руководитель
доцент к. ф.-м. н. _____ А. В. Жаркова
подпись, дата

Заведующий кафедрой
д. ф.-м. н., доцент _____ М. Б. Абросимов
подпись, дата

Саратов 2022

СОДЕРЖАНИЕ

ВВЕДЕНИЕ

Актуальность темы состоит в том, что с формированием сети Интернет многие сферы современного мира такие, как образование, бизнес, культура движутся в виртуальное пространство, что влечет за собой такую проблему, как фальсификация информации, предаваемой пользователями по сети. Данные, которыми обмениваются пользователи, должны быть защищены от внешних угроз и от злонамеренных поступков таких, как изменение текста сообщения получателем, фальсификация результатов испытаний или других подмен, связанных с документами, которые имеют юридическую силу. В виртуальном мире проверить подлинность документов можно с помощью временной метки.

Целью данной курсовой работы является изучение временной метки как одной из задач аутентификации, в результате чего требуется написать программу, реализующую один из рассмотренных протоколов.

Исходя из поставленной цели, необходимо решить следующие задачи:

1) осветить в работе теоретические аспекты, необходимые для реализации;

2) написать программу на высокоуровневом языке программирования, применив полученные теоретические знания на практике.

1 Необходимые определения

Для успешного выполнения поставленной цели необходимо ознакомиться с основными теоретическими понятиями, касающимися темы курсовой работы.

Криптография (от греческого тайнопись) – это совокупность идей и методов, связанных с преобразованием информации с целью ее защиты от непредусмотренных пользователей. Информация считается представленной в виде некоторого текста (сообщения).

Сообщение – открытый текст.

Шифр – способ преобразования сообщения в защищенную форму.

Шифрование – процесс применения шифра

Криптограмма – измененный текст, полученный в результате шифрования.

Дешифрование/*расшифрование* – перевод криптограммы в исходный открытый текст.

Ключ – это дополнительная информация, с помощью которой осуществляются взаимно обратные действия шифрования и расшифрования.

Криптология – наука о передаче информации в виде, защищенном от несанкционированного доступа.

Идентификация – это назначение объекту системы уникальной условной метки, которая позволяет однозначно определить этот объект.

Аутентификация – это проверка подлинности объекта, предъявившего данный идентификатор. Аутентификация основана на информации, которая может быть известна только истинному пользователю системы.

Хеш-функция предназначается для компактного представления длинных последовательностей (слов). Она преобразует сообщение произвольной длины над данным алфавитом в блок фиксированной длины над тем же алфавитом, т.е. производит свертку всех сообщений (слов) в сообщения (слова) одной и той же заданной длины [1].

Информация – сведения о лицах, предметах, фактах, событиях, явлениях и процессов независимо от формы их представления.

Защищаемая информация – это собственность какого-либо субъекта (государства, предприятия, группы лиц или отдельного гражданина) и подлежит защите в соответствии с требованиями собственника информации или с требованиями правовых документов.

Защита информации – это деятельность, направленная на предотвращение утечки защищаемой информации, несанкционированных и непреднамеренных воздействий на защищаемую информацию.

Целостность информации – это ее свойство оставаться неизменной при непреднамеренных или несанкционированных воздействиях или возможность выявлять случаи такого воздействия со стороны владельца.

Подлинность информации – это свойство оставаться неизменной при несанкционированных воздействиях или возможность выявлять случаи такого воздействия со стороны владельца (пользователя) информации.

Идентификатор – это уникальный признак данной информации, на основе которого можно доказательно установить ее подлинность.

Претендент – субъект, доказывающий свою аутентичность (свою, сели решается задача опознавания, или сформированных данных, если решается задача аутентификации сообщения).

Верификатор – субъект, проверяющий аутентичность [2].

Временная метка – достоверная информация о моменте подписания документа, которая присоединена к указанному электронному документу или иным образом связана с ним [3].

2 Об аутентификации и временной метке

1) Аутентификация.

Задача аутентификации стоит перед людьми с древних времен, так как функционирование юридических отношений между людьми предполагает уверенность в подлинности документов, выносимых на обсуждение. Пока все предприятия малого и среднего бизнеса, все вопросы, касающиеся образования, культуры, спорта, банковские дела, не переступили порог глобальной цифровизации, решить проблему аутентификации было довольно просто с помощью печатных подписей, но в виртуальном мире все оказалось намного сложнее. Компьютеры предоставлены сами себе и не умеют отличать одного пользователя от другого. Чтобы уверенно соотносить физическое лицо с пользователем сети, используют механизмы аутентификации. Надежная система аутентификации пользователей – важнейший компонент корпоративной системы информационной безопасности [4].

Существует достаточно большое количество способов аутентификации, но их можно условно поделить на несколько категорий:

1) что-то, что вы знаете – пароли и ПИН-коды. Максимально доступным способом аутентификации является использование паролей, но этот метод все больше теряет свою популярность из-за большого количества взломов, ведь пароль можно подобрать с помощью специальных программ;

2) то, кем вы являетесь – подход, основанный на биометрических данных человека. Идентификаторами могут служить как более простые характеристики человека: рост, вес, цвет волос, так и более сложные, например, запись голоса, отпечаток пальца, узоры радужной оболочки глаз, геометрия ладони. Наиболее надежный способ аутентификации, нежели пароли или ПИН-коды, так как физические характеристики сложно фальсифицировать, а некоторые из них и вовсе невозможно;

3) то, что у вас есть – подход, основанный на владении претендентом какого-то носителя информации такие, как банковские карты, удостоверения

личности. К тому же этот метод может распространяться и на нематериальные вещи, например, электронные карты;

4) то, что вы делаете – метод основан на индивидуальных действиях человека таких, как походка или особый интервал времени между нажиманием клавиш;

5) то, где вы находитесь – метод основан на определении местоположения претендента [5].

В процессе аутентификации участвуют как минимум две стороны: претендент и верификатор. В наиболее сложных системах безопасности их может быть больше. Претендент передает верификатору служебную информациюp (в случае аутентификации сообщения претендент передает еще и сообщение m). Информация p зависит от секретного ключа k претендента, а в случае диалоговой аутентификации и от запроса v верификатора. Используя свой ключ k', верификатор проверяет правильность информации p [6].

2) Временная метка: описание.

Для аутентификации документооборота в юридических отношениях фундаментальной стала цифровая подпись. Но часто требуется знать дату ознакомления, создания или изменения документа, которую невозможно было бы изменить без того, чтобы это стало явным.

Целью аутентификации электронных документов является их защита от возможных угроз таких, как:

1) активный перехват – нарушитель, подключившийся к сети, перехватывает документы (файлы) и изменяет их;

2) маскарад – абонент С посылает документ абоненту В от имени абонента А;

3) ренегатство – абонент А заявляет, что не посылал сообщения абоненту В, хотя сам на самом деле послал;

4) подмена – абонент В изменяет или формирует новый документ и заявляет, что получил его от абонента А;

5) повтор – абонент С повторяет ранее переделанный документ, который абонент А посылал В [7].

Защиту от этих угроз может обеспечить временная метка. Установка и проверка временной метки является одной из задач аутентификации. Временная метка может быть реализована с помощью цифровой подписи двумя способами: централизованно (при наличии центра установления меток) и децентрализовано (с распределенным механизмом проверки временной метки).

3) Централизованная реализация временной метки.

При централизованной реализации временной метки иногда требуется, чтобы обеспечивалась конфиденциальность (центр не должен знать содержание документа) и анонимность (центр не должен знать, кто прислал документ).

Протокол предусматривает участие трех легальных пользователей: претендента, который хочет добавить к документу временную метку, центра временных меток и верификатора, проверяющего правильность документа и времени его создания. Используемый криптографический примитив – электронная цифровая подпись центра временных меток. Достоинством этой реализации является ее простота.

Протокол. Централизованное создание и проверка временной метки.

Вход претендента. Документ m; открытый ключ проверки подписи; хеш-функция h.

Вход_верификатора. Открытый ключ проверки подписи; хеш-функция h.

Вход центра временных меток. Код текущего времени; секретный ключ формирования подписи.

Метод.

Для создания временной метки выполняются следующие действия.

1) Претендент, создавший (изменивший) документ m, вычисляет хеш-функцию $h(m)$ и посылает ее в доверенный центр.

2) Центр вырабатывает подпись $s(T)$ для проверочного текста $T = h(m) \,||\, t$, где t – код времени, и возвращает сертификат, состоящий из проверочного текста и подписи для него претенденту.

3) Претендент проверяет правильность проверочного текста, подписи и времени.

Для проверки временной метки выполняются следующие действия.

1) Претендент посылает документ и сертификат $(m, T, s(T))$ верификатору.

2) Верификатор выполняет следующие действия.

2.1) Вычисляет хеш-функцию $h(m)$ и проверяет ее соответствие проверочному тексту T.

2.2) Проверяет время t формирования подписи.

2.3) Проверяет правильность подписи.

При успешном прохождении всех проверок документ и время его подписи считаются подлинными.

Цифровая подпись может быть реализована, например, по ГОСТ Р 34.10-2001.

Поскольку участники протокола не вполне доверяют друг другу, могут возникать ситуации сговора некоторых из них между собой. Это требует соответствующего усиления протокола централизованного создания и проверки временной метки.

Так, центр иногда может быть не доверенной стороной и нарушать протокол, чтобы вместе с претендентом P_i обмануть верификатора. Для противодействия этому обычно применяется сцепка нескольких документов. Например, в сертификат включается имя претендента и имя следующего претендента P_{i+1} или нескольких следующих претендентов. Верификатор обращается к этим претендентам с просьбой предъявить свои сертификаты. Если они верные, то и сертификат проверяемого претендента верен.

Однако в этом случае число легальных участников данного этапа протокола увеличивается, и приходится рассматривать дополнительные случаи сговора, то есть это может привести не к усилению, а к ослаблению протокола. Например, если претендент P_{i+1} откажется представлять свой сертификат или преднамеренно предъявит подложный сертификат, то он может отменить

аутентификацию претендента P_i, даже если документ и сертификат, предъявленные претендентом P_i, являются подлинными.

4) Децентрализованная реализация временной метки.

Рассмотрим основные положения децентрализованной реализации. Предполагается, что число претендентов, подписывающих документы, достаточно велико для того, чтобы они не вступали в сговор против какого-то претендента или против верификатора. В основу протокола положена цифровая подпись RSA.

Пусть $n = pq$ – составное число, разложения которого не знает никто. Предположим, что претендент P_i с участием претендентов $P_1, \dots, P_m$, обладающих документами $x_1, \dots, x_m$, хочет подписать свой документ так, чтобы можно было соотнести подписанный документ со временем подписи. Для этого на каждом цикле протокола выполняются следующие действия.

Протокол. Децентрализованное создание и проверка временной метки.

Метод.

1) Претенденты по коду времени t вычисляют: $x_0 \leftarrow t^2 (mod\ n)$.

2) Претенденты вычисляют значения хеш-функции для своих документов: $y_i \leftarrow h(x_i)$.

3) Претенденты вычисляют проверочный текст (x_0, a) для данного цикла протокола, где $a \equiv x_0^{y_1 \cdots y_m} (mod\ n)$, и опубликовывают его.

4) Претендент P_i вычисляет свой сертификат $(z_i \equiv x_0^{y_1 \cdots y_{i-1} y_{i+1} \cdots y_m} (mod\ n), y_i)$.

5) Для проверки правильности документа x_i и времени подписи претендент P_i предъявляетверификатору текст, сертификат и время t.

6) Верификатор вычисляет хеш-функцию $h(x_i)$ и проверяет условия: $h(x_i) = y_i;\ a \equiv z_i^{y_i} (mod\ n);\ x_0 \equiv t^2 (mod\ n)$. Если все проверки проходят успешно, то претендент P_i действительно подписал документ в указанный момент времени.

Недостатком протокола децентрализованного создания и проверки временной метки является необходимость совместных действий нескольких претендентов. Кроме того, этот протокол уязвим к атаке на основе эндоморфизмов [2].

5) Цифровая подпись.

Курсовая работа предполагает написание программной реализации одного из протоколов временной метки. В работе будет представлена реализация протокола централизованного создания и проверки временной метки. В этом протоколе центр вырабатывает цифровую подпись для претендента, для реализации цифровой подписи воспользуемся алгоритмом Эль-Гамаля.

Протокол подписи Эль-Гамаля строится следующим образом. Секретным ключом формирования подписи служит целое число $x,\ 0 < x < r$, открытым ключом проверки подписи – простое число p, образующая a подгруппы простого порядка r мультипликативной группы F_p^* простого поля и экспонента $b \equiv a^x (mod\ p)$.

Протокол. Схема подписи Эль-Гамаля.

Вход отправителя. Секретный ключ x.

Вход получателя. Характеристика поля p, образующая a подгруппы простого порядка r, экспонента $b \equiv a^x (mod\ p)$.

Результат. Формирование и проверка подписи.

Метод.

Для формирования подписи для сообщения $m,\ 0 < m < r$, отправитель выполняет следующие действия.

1) Генерирует случайное число $k,\ 0 < k < r$.

2) Полагает $w \leftarrow a^k (mod\ p)$.

3) Находит число s, решая сравнение $m \equiv xw + ks (mod\ r)$:

$$s \leftarrow (m - xw)\ k^{-1} (mod\ r).$$

Подписью для сообщения m является пара (w, s).

Для проверки подписи получатель выполняет следующие действия.

1) Проверяет неравенство $w < p$. Если оно не выполнено, то результат: подпись недействительна.

2) Проверяет сравнение для экспонент: $a^m \equiv b^w w^s (mod\ p)$. Если оно выполняется, то результат: подпись подлинная, иначе результат: подпись недействительна [2].

6) Хеш-функция.

Создание цифровой подписи невозможно без хеш-функции. Чтобы хеш-функция была криптостойкой, необходимо выполнение следующих условий ее программной реализации:

1) (противодействие определению прообраза) если известно, что q является сверткой некоторого слова, то практически невозможно найти слово p, для которого $h(p) = q$;

2) (противодействие обнаружению второго прообраза) для данного слова p невозможно найти другое слово p'с такой же сверткой: $h(p') = h(p)$;

3) (противодействие коллизии) невозможно найти два разных слова pи p'с одинаковой сверткой: $h(p) = h(p')$.

Конкретно в алгоритме электронной цифровой подписи хеш-функция применяется для аутентификации сообщения. Код проверки подлинности сообщения, или MAC (Message Authentication Code) – это зависящая от секретного ключа криптографическая хеш-функция. Если абоненты сети A и B используют общий секретный ключ k, то A, посылая для B сообщение m, прикрепляет к нему MAC – хеш-значение $h(k||m)$ (к сообщению впереди приписывается ключ, создавая единый массив). Так как B знает ключ k, то, получив сообщение, скажем m', он вычислит $h(k||m')$ и, сравнив это значение с присланным MAC $h(k||m)$, увидит, изменилось или нет исходное сообщение в ходе передачи [1].

3 Программная реализация протокола централизованного создания и проверки временной метки

В ходе работы была выполнена реализация протокола централизованного создания и проверки временной метки на языке программирования высокого уровня C++. Программа состоит из файла Main.cpp – файла для запуска выбора режима работы программы, и 5 заголовочных файлов: Pretender.h – файл для имитации работы претендента; Center.h – файл для имитации работы центра; Verifier.h – файл для имитации работы верификатора; Keys.h – файл для генерации ключей для формирования и проверки цифровой подписи; Hash.h – файл, содержащий функцию хеширования; Checking.h – файл для проверок на корректность данных. В каждом заголовочном файле описан класс, который выполняет задачи, связанные с названием файла. Программа реализована для двух режимов работы: централизованное создание и проверка временной метки. При советующем выборе запускается либо работа претендента через функцию $getStamp()$, описанную в заголовочном файле Pretender.h, либо работа верификатора через функцию $checkDocument()$, описанную в файле Verifier.h. Рассмотрим режимы работы подробнее.

1) Описание кода программы для централизованного создания временной метки.

После получения запроса из Main.cpp претендент начинает свою работу. Ему необходимо передать доверенному центру хеш-значение документа, который нужно подписать. Поэтому сначала он применяет хеш-функцию $getHash()$ к документу m.

Рассмотрим реализацию хеш-функции в файле Hash.h. Хеш-функция разрабатывалась исходя из простоты и скорости, так как целью данной работы является программная реализация протокола временной метки. Функция состоит из двух этапов: подсчета суммы по модулю ASCII-кодов символов (в программе локальная переменная hex) и взятия остатка от деления этой суммы на длину документа. ASCII-коды берутся неслучайно. У каждого символа свой код,

поэтому при изменении документа хеш-код тоже изменится, что будет сигналом об ошибке аутентификации. Взятие остатка от деления на длину документа нужно для сокращения числа для более простых подсчетов. При некоторых значениях времени hex может быть кратен длине строки $slength$, что противоречит условию $1 < hex < p - 1$. Это объясняет добавление цикла $while$, в котором при кратности увеличиваем длину строки. Может быть так, что необходимо будет захешировать пустой текстовый документ, чтобы хеширование прошло успешно, прибавляем символ пробел каждый раз к строке s, в которую считывается файл.

Вернемся к файлу Pretender.h. После получения результата хеш-функции претендент запрашивает у центра сертификат с помощью функции $setCertificate()$, передавая в качестве аргумента $h(m)$. После получения запроса от претендента центр сначала формирует время t с помощью функции $clock()$. Реализация этой функции представляет собой получения системного времени от Windows с помощью библиотеки Time.h.

После получения времени t центр формирует проверочный текст $T = h(m) \,||\, t$. Формирование происходит конкатенацией строго через пробел хеш-кода $h(m)$ от претендента и времени t.

Затем центр формирует подпись $s(T)$. Для этого необходимо вызвать хеш-функцию, передав в качестве аргумента T. Полученное значение в программе присвоено локальной переменной hex.

Далее происходит формирование открытых ключей p, g, y, закрытого ключа x (в программе $closeKey$) и сессионного ключа k. Генерация ключей происходит в файле Keys.h. Диапазон для p выбран от h до $h + 22$, потому что по условию алгоритма Эль-Гамаля h должно быть меньше чем $p - 1$, так как h генерируется раньше чем p и не рандомно, то для корректной работы необходимо, чтобы p генерировалось в заданном диапазоне. Число 20 выбрано исходя из упрощения подсчетов.

Открытый ключ g, закрытый ключ x и сессионный ключ k зависят от p. В зависимости от p формируются диапазоны рандомизации, в которых, в свою очередь, формируются ключи g, x, k. Диапазоны рандомизации ключей выбраны минимальными для упрощения подсчетов, так как на более простых числах можно точно проверить правильность работы программы, иначе результат арифметических операций может быть отрицательным, что в силу логики программы невозможно. Отрицательные значения возникают из-за того, что диапазоны значений в C++ ограничены. Также минимальные диапазоны снижают к нулю зацикливание, которое возникает, когда интерпретатор не может подобрать число. Открытый ключ y считается по формуле $y = g^x \ mod \ p$.

Возвращаемся в центр, куда были переданы все сгенерированные ключи. Теперь начинается формирование электронной цифровой подписи, которая по алгоритму Эль-Гамаля состоит из двух компонент (w, s).

Первый компонент w вычисляется по формуле $w = g^k \ mod \ p$.

Компонент s находится по формуле $s = (m - xw)k^{-1}(mod\ r)$. В этой формуле все известно кроме k^{-1}, найдем его по формуле $k^{-1} * \ k \ \equiv \ 1 \ (mod\ r)$. Для удобства перепишем формулу, и получим $(k^{-1} * \ k - \ 1) \ mod \ r = 0$. Найдём k^{-1} подбором, исходя из двух условий:

1) разность $k^{-1} * \ k - \ 1$ должна быть больше нуля;

2) разность $k^{-1} * \ k - \ 1$ должна делиться на r без остатка.

После нахождения элемента k^{-1} подставляем его в формулу для s и получаем результат – второй компонент цифровой подписи.

После сформирования проверочного текста и подписи, центр заносит эти данные в текстовые файлы TextFromCenter.txt и Stamp.txt соответственно. Также необходимо передать претенденту посредством текстового файла открытые ключи, поэтому заносим открытые ключи в текстовый файл OpenKeys.txt.

Возвращаемся к претенденту, который, в свою очередь, начинает проверку данных, переданных от центра. Проверка состоит из трех этапов:

1) проверить на корректность проверочный текст в части хеш-кода: хеш-код должен соответствовать хеш-коду, переданному в центр претендентом;

2) проверить на корректность проверочный текст в части временной метки: временная метка должна находиться в одном файле с хеш-значением. Проверка реализована исходя из утверждения: если метка корректна, то она существует. Проверка этого и предыдущего пунктов осуществлена в функции $checkVerText()$. Принцип проверки временной метки состоит в следующем, если формат метки соответствует стандартному представлению, то она корректна;

3) проверка подписи на корректность с помощью открытых ключей. Для этого претендент хеширует проверочный текст и передает его в качестве аргумента функции $checkStamp()$ из Checking.h. В функции проверка происходит посредством сравнения двух величин, вычисленных по формулам $check1 = y^w * w^s \, mod \, p$ и $check2 = g^h mod \, p$. Если они равны, то подпись корректна. В этом файле подключаем библиотеку Large.h для работы с длинной арифметикой.

Если данные от центра прошли все проверки на корректность, то претендент выдает сообщение о том, что временная метка создана успешно. Иначе выдает ошибку. На этом режим централизованного создания временной метки завершает свою работу.

2) Описание кода программы для проверки временной метки.

По протоколу от верификатора требуется проверка на подлинность временной метки. Чтобы осуществить эту проверку, претендент должен передать верификатору файлы с подписанным центром документом m, с проверочным текстом $T = h(m) \,||\, t$, с подписью (w, s) и с открытыми ключами p, g, y. Названия файлов с расширениями вводятся с клавиатуры. Так как пользователь сам заносит в документы данные для проверки, нужно дополнительно проверить корректность введенных данных.

Проверка верификатора состоит из четырех этапов:

1) проверить на корректность проверочный текст в части хеш-кода: хеш-код должен соответствовать хеш-коду, вычисленному от документа m;

2) проверить на корректность проверочный текст в части временной метки: временная метка должна быть логичной и удовлетворять формату ввода. Метод проверки этого и предыдущего пунктов аналогичен 1-ому и 2-ому этапам проверки проверочного текста претендентом;

3) проверить подпись и открытые ключи на корректный формат ввода. Проверка происходит посредством функции $checkStampOut()$ и функции $checkKeys()$. Принцип проверки аналогичен проверке временной метки на корректность: если подпись и открытые ключи соответствуют стандартному формату, то они корректны;

4) проверка подписи на корректность с помощью открытых ключей. Для этого он хеширует проверочный текст и передает его в качестве аргумента функции $checkStamp()$ из Checking.h.

Если все данные верны, то выдается сигнал о том, что документ и время его подписи считаются подлинными, иначе выдается ошибка. На этом режим проверки временной метки завершает свою работу.

Важно подчеркнуть, что во время работы обеих программ в них поступают текстовые файлы. Поэтому каждый раз их необходимо проверять на открытие (существование файла) и на корректность расширения. Эти проверки осуществлены в файле Checking.h. Проверка на открытие будет произведена с помощью функции $checkFile()$. Если файл не открыт, т. е. его не существует, происходит аварийное завершение программы и вывод ошибки «ВНИМАНИЕ! Данного текстового документа не существует».

Проверка на корректность расширения будет производиться с помощью функции $checkExt()$. В строке с именем файла необходимо найти индекс последней точки, после чего происходит копирование в дополнительную строку $testString$ всех символов после этой точки, включая ее. Затем происходит сравнение строки $testString$ со строкой «.txt». Если строки не совпали, то

происходит аварийное завершение программы и вывод ошибки «ВНИМАНИЕ! Некорректное расширение».

3) Инструкция для пользователя.

1) Программа работает в двух режимах с файлами, у которых расширение .txt. Вводить имена текстовых документов нужно с расширением. Например, создаем документ с название m и с расширением .txt, в консоль вводим m.txt.

2) Запустить программу пользователь может, нажав на клавиатуре клавишу «F5», или через отладчик.

3) После запуска программы откроется консоль, в которой пользователю необходимо выбрать режим работы:

3.1) 1 – централизованное создание временной метки;

3.2) 2 – проверка временной метки.

4) В зависимости от выбора программа переходит в один из режимов:

4.1) если пользователь ввел «1», то он перейдет в режим централизованного создания временной метки, в котором необходимо сделать следующие действия:

4.1.1) в папке с проектом пользователю нужно создать текстовый документ, который нужно подписать;

4.1.2) в консоли необходимо ввести имя текстового документа, созданного на предыдущем пункте;

4.1.3) после ввода необходимо нажать клавишу «Enter»;

4.1.4) если программа выполнилась правильно, т. е. документ был подписан, то после нажатия клавиши «Enter» в консоли появится надпись: «Временная метка создана успешно»;

4.1.5) иначе после нажатия клавиши «Enter» будет выдана ошибка;

4.2) если пользователь ввел «2», то он перейдет в режим проверки временной метки, в котором необходимо сделать следующие действия:

4.2.1) пользователь должен либо перенести в папку с проектом, либо создать в ней следующие текстовые файлы: подписанный документ,

документ, в котором содержится проверочный текст, документ, в котором содержится подпись, документ, в котором содержатся открытые ключи;

4.2.2) данные в этих файлах должны соответствовать определенному формату. Например, если цифровая подпись $(w, s) = (5, 6)$, то в файле для подписи эти числа должны быть записаны строго через один пробел: 5 6. Например, если открытые ключи $p = 5,\ g = 2,\ y = 6$, то в файле для открытых ключей эти числа должны быть записаны строго через один пробел: 5 2 6. Например, временная метка, сделанная в воскресение 5 июня 2022 года в 17:16:48, должна быть представлена, как $Sun\ Jun\ \ 5\ 17{:}\ 16{:}\ 48\ 2022$. Если число месяца меньше 10, то его и название месяца должны разделять строго два пробела. Все остальные части, включая хеш-код, должны быть разделены одним пробелом;

4.2.3) в консоли необходимо ввести имена документов, описанных на 1-ом шаге, учитывая последовательность ввода, отраженную в консоли;

4.2.4) после ввода нужно нажать клавишу «Enter»;

4.2.5) после нажатия клавиши «Enter» начинается проверка документа и времени его подписи на подлинность по следующим пунктам:

4.2.5.1) соответствие вычисленной хеш-функции $h(m)$ проверочному тексту T;

4.2.5.2) корректность времени t формирования подписи;

4.2.5.3) правильность подписи;

4.2.6) если программа выполнилась правильно, то документ и время его подписи считаются подлинными, а значит в консоли появится надпись: «Документ и время его подписи подлинные»;

4.2.7) иначе после нажатия клавиши «Enter» будет выдана ошибка.

5) Если пользователь захочет снова проверить или подписать документ после окончания успешной проверки, то, не закрывая консоль, он может выбрать режим, так как выбор запускается автоматически после последнего запуска. Если

же была выведена ошибка, то нужно повторить все действия, начиная с 1-го пункта.

4) Примеры работы программы.

1) Корректный запуск программы для централизованного создания временной метки представлен на рисунках 1, 2, 3, 4, 5, 6.

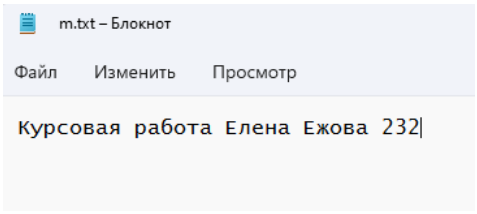

Рисунок 1 – Содержание текстового документа, на который нужно поставить временную метку

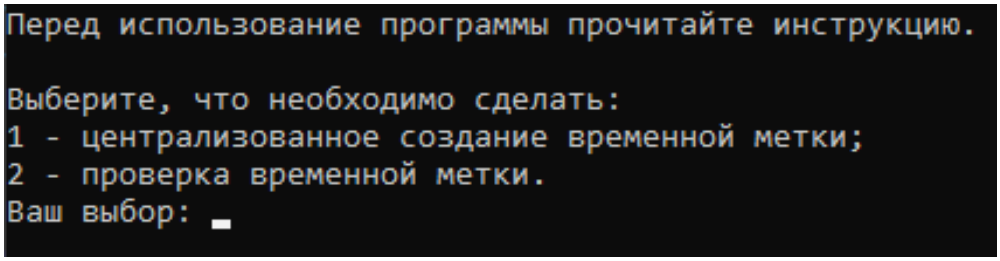

Рисунок 2 – Запуск программы

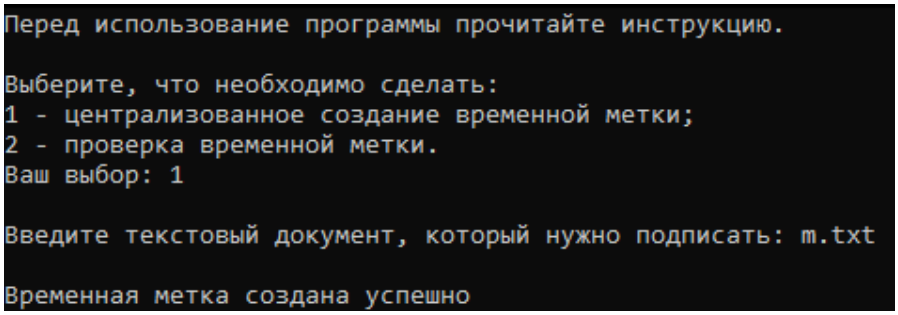

Рисунок 3 – Успешное создание временной метки

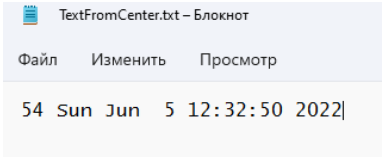

Рисунок 4 – Файл, в который был записан проверочный текст

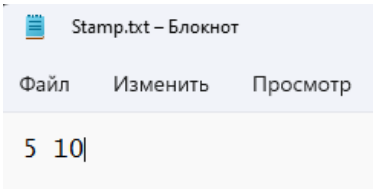

Рисунок 5 – Файл, в который была записана цифровая подпись

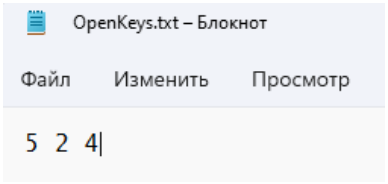

Рисунок 6 – Файл, в который были записаны открытые ключи

2) Корректный запуск программы для проверки временной метки представлен на рисунках 7, 8.

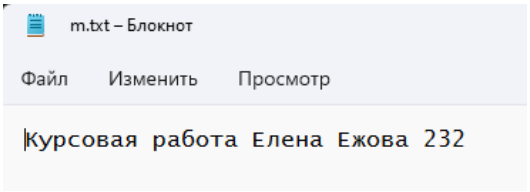

Рисунок 7 – Содержание текстового документа, подлинность и временную метку которого нужно проверить

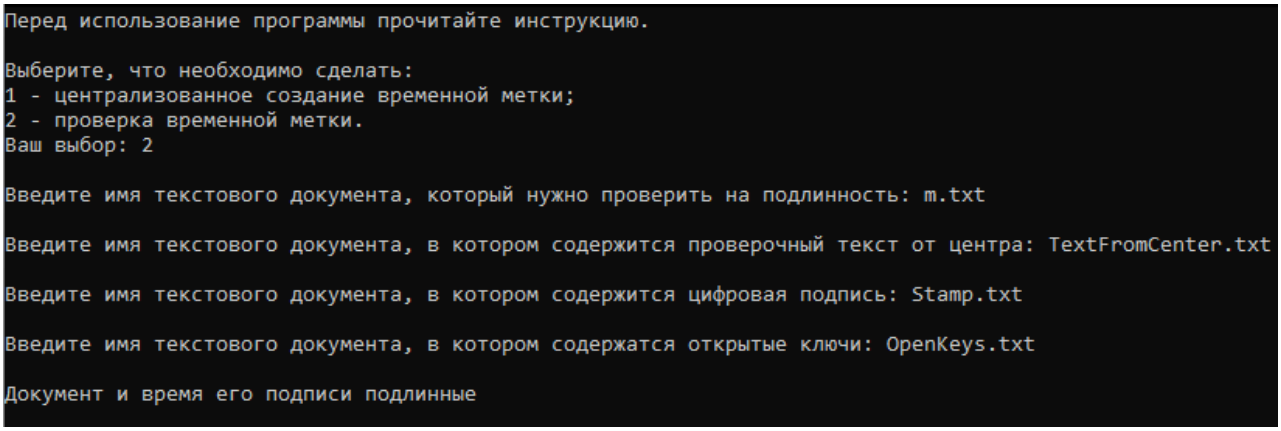

Рисунок 8 – Успешное завершение программы

3) Некорректный запуск программы из-за фальсификации документа представлен на рисунках 9, 10, 11.

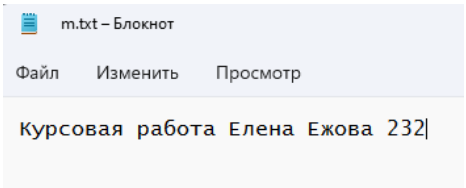

Рисунок 9 – Содержание текстового документа, который был подписан

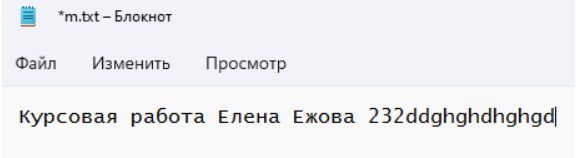

Рисунок 10 – Содержание текстового документа после изменения

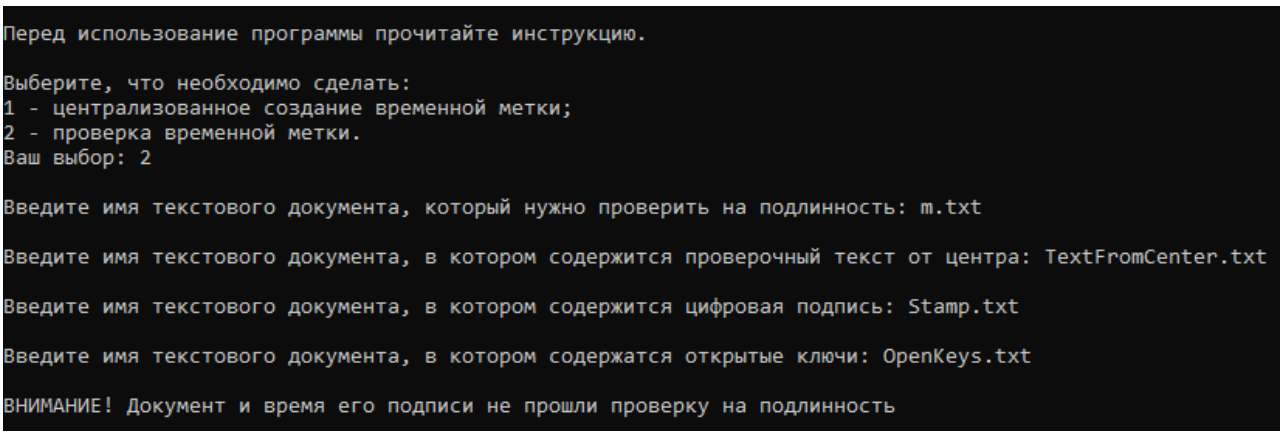

Рисунок 11 – Программа выдала ошибку

4) Некорректный запуск программы из-за некорректного расширения представлен на рисунках 12, 13.

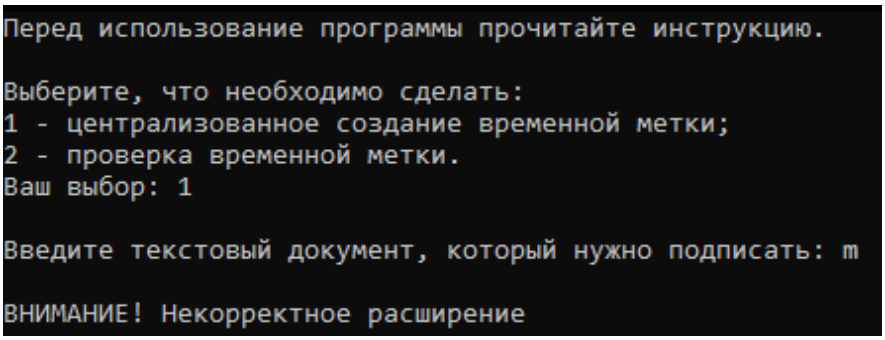

Рисунок 12 – Ошибка из-за отсутствия расширения

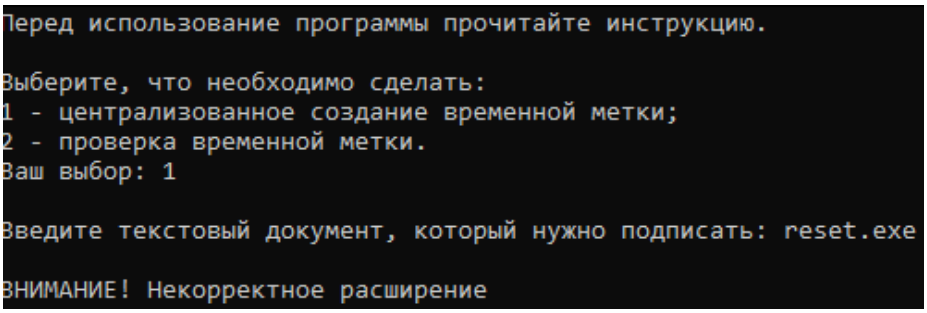

Рисунок 13 – Ошибка из-за неподходящего расширения

5) Некорректный запуск программы из-за ввода не существующего файла представлен на рисунке 14.

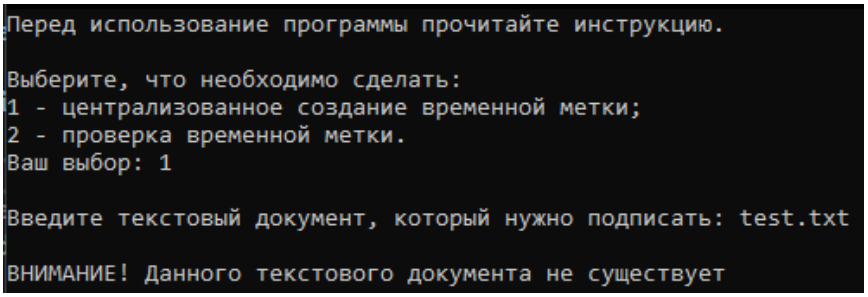

Рисунок 14 – Ошибка ввода

6) Некорректный запуск программы для проверки на подлинность из-за некорректного содержания файлов (на рисунках 15, 16, 17, 18 представлены ошибки, связанные с содержанием файла с подписью, но для файлов с проверочным текстом и с открытыми ключами программа будет работать аналогично).

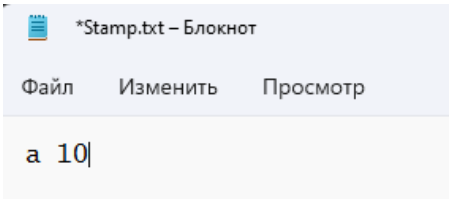

Рисунок 15 – Содержание текстового файла, содержащего подпись

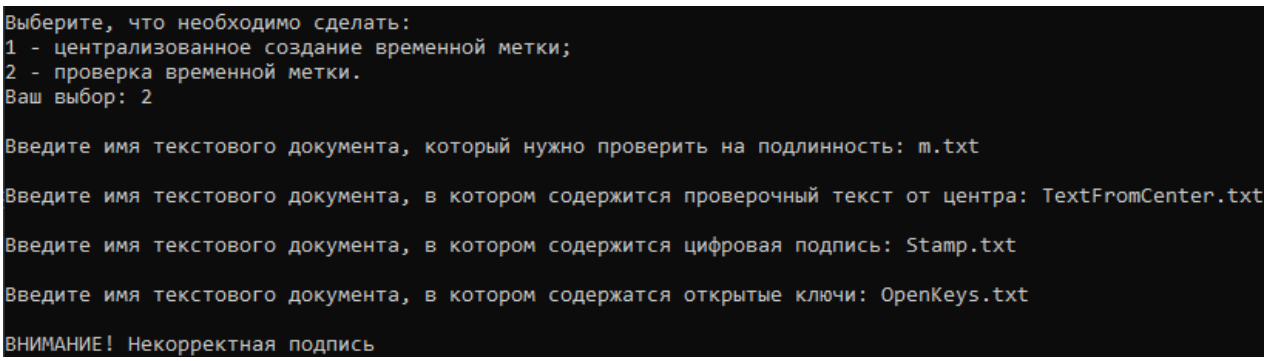

Рисунок 16 – Ошибка из-за ввода символов, отличных от цифр

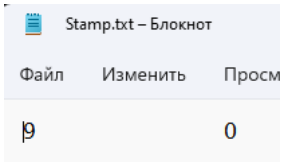

Рисунок 17 – Содержание текстового файла, содержащего подпись

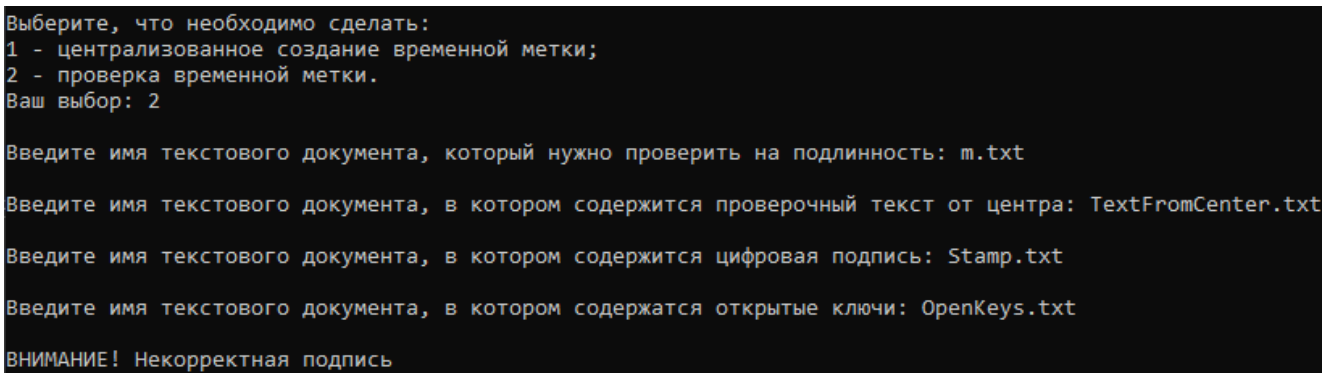

Рисунок 18 – Ошибка из-за некорректного расстояния между компонентами

ЗАКЛЮЧЕНИЕ

Временная метка в составе электронной цифровой подписи является важным компонентом, без которого аутентификация практически невозможна в системах с повышенным уровнем безопасности и в повседневных делах, связанных с документооборотом. Действительно, временная метка – это одна из задач аутентификации, потому что в современном мире особенно актуальна проблема фальсификации документов, но описанные в курсовой работе протоколы и схемы во многом предотвращают возможные угрозы.

Реализация протокола централизованного создания и проверки временной метки в работе не может использоваться в серьезных проектах, связанных с криптографической защитой и проверкой данных, но иллюстрирует основную концепцию создания и проверки временной метки для защиты от подмены.

Таким образом, все поставленные задачи решены, цель работы достигнута.

СПИСОК ИСПОЛЬЗОВАННЫХ ИСТОЧНИКОВ

1) Салий, В. Н. Криптографические методы и средства защиты информации [Электронный ресурс] : учеб. пособие / В. Н. Салий // Саратовский национальный исследовательский государственный университет имени Н. Г. Чернышевского [Электронный ресурс]. – Саратов, 2017. – 43 с. – URL: https://www.sgu.ru/sites/default/files/textdocsfiles/2017/10/18/saliy_v.n._kriptograficheskie_metody_i_sredstva_zashchity_informacii.pdf (дата обращения: 04.06.2022). – Загл. с экрана. – Яз. рус.

2) Ростовцев, А. Г. Теоретическая криптография [Электронный ресурс] / А. Г. Ростовцев, Е. Б. Маховенко. – СПб. : НПО «ПРОФЕССИОНАЛ», 2005. – 464 с. – Загл. с экрана. – Яз. рус.

3) Постановление Правительства РФ от 09.02.2012 № 111 (ред. от 20.11.2018) [Электронный ресурс]. Доступ из справочно-правовой системы «КонсультантПлюс».

4) Основы информационной безопасности [Электронный ресурс] : учеб. пособие для вузов / Е. Б. Белов [и др.]. – М. : Горячая линия – Телеком, 2006. – 544 с. – Загл. с экрана. – Яз. рус.

5) Иванов, М. А. Криптографические методы защиты информации в компьютерных системах и сетях [Электронный ресурс] / М. А. Иванов : учеб. пособие. – М. : КУДИЦ-ОБРАЗ, 2001. – 368 с. – Загл. с экрана. – Яз. рус.

6) Теоретические основы компьютерной безопасности [Электронный ресурс] : учеб. пособие для вузов по спец. «Компьютерная безопасность», «Компьютерное обеспечение информационной безопасности автоматизированных систем» / Девянин П. Н. [и др.]. – М. : Радио и связь, 2000. – 190 с. – Загл. с экрана. – Яз. рус.

7) Рябко, Б. Я. Криптографические методы защиты информации [Электронный ресурс] : учеб. пособие для вузов / Б. Я. Рябко, А. Н. Фионов. – М. : Горячая линия – Телеком, 2005. – 229 с. – Загл. с экрана. – Яз. рус.

ПРИЛОЖЕНИЕ А

Листинг программы

Main.cpp

```
#include <iostream>
#include <string>
#include <vector>
#include "Pretender.h"
#include "Verifier.h"
using namespace std;
using namespace check;
using namespace pretender;
using namespace hash;
using namespace verifier;
int main() {
	setlocale(LC_ALL, "Rus");
	Check check;
	Pretender pretender;
	Hash hash;
	Verifier verifier;
	int choose;

	cout << "Перед использование программы прочитайте инструкцию." <<
endl;
	cout << endl;
	cout << "Выберите, что необходимо сделать: " << endl;
	cout << "1 - централизованное создание временной метки; " << endl;
	cout << "2 - проверка временной метки.  " << endl;
	cout << "Ваш выбор: "; cin >> choose;
	if (choose == 1) {
		cout << endl;
		string stamp = pretender.getStamp();
		cout << stamp;
		cout << endl;
		cout << endl;
	}
	else if (choose == 2) {
		cout << endl;
		string check = verifier.checkDocument();
		cout << check;
		cout << endl;
		cout << endl;
	}
	return main();
}
```

Verifier.h

```
#pragma once
#include <iostream>
#include <string>
#include <vector>
using namespace std;
```

```
using namespace check;
using namespace pretender;
using namespace hash;
namespace verifier {
	class Verifier {
	public:
		string checkDocument();
	};
	string Verifier::checkDocument() {
		setlocale(LC_ALL, "Rus");
		Check check;
		Hash hash;
		string nameDocument;
		cout << "Введите имя текстового документа, который нужно
проверить на подлинность: ";
		cin >> nameDocument;
		cout << endl;
		string nameT;
		cout << "Введите имя текстового документа, в котором содержится
проверочный текст от центра: ";
		cin >> nameT;
		cout << endl;
		string nameP;
		cout << "Введите имя текстового документа, в котором содержится
цифровая подпись: ";
		cin >> nameP;
		cout << endl;
		string openKeys;
		cout << "Введите имя текстового документа, в котором содержатся
открытые ключи: ";
		cin >> openKeys;
		cout << endl;
		if ((check.checkExt(nameDocument) == true) &&
			(check.checkExt(nameT) == true) &&
			(check.checkExt(nameP) == true) &&
			(check.checkExt(openKeys) == true)) {
			string s = check.checkFile(nameDocument);
			long long hex = hash.getHash(s);
			if ((check.checkVerText(nameT, hex) == true) &&
				(check.checkStampOut(nameP) == true) &&
				(check.checkKeys(openKeys) == true)) {
				string str = check.checkFile(nameT);
				string stamping = check.checkFile(nameP);
				string openKeyss = check.checkFile(openKeys);
				string* stampp = new string[2];
				for (int j = 0; j < 2; j++) {
					for (int i = 0; i < stamping.length(); i++) {
						if (stamping[i] != ' ') stampp[j] +=
stamping[i];
						else j++;
					}
				}
				string* keys = new string[3];
				for (int j = 0; j < 3; j++) {
					for (int i = 0; i < openKeyss.length(); i++) {
						if (openKeyss[i] != ' ') keys[j] +=
openKeyss[i];
```

```
                              else j++;
                        }
                  }
                  long w = stoi(stampp[0]);
                  long s = stoi(stampp[1]);
                  long p = stoi(keys[0]);
                  long g = stoi(keys[0]);
                  long y = stoi(keys[0]);
                  long long checkText = hash.getHash(str);
                  bool res = check.checkStamp(w, s, p, g, y,
checkText);
                  if (res == true) {
                        return "Документ и время его подписи
подлинные";
                  }
                  else return "ВНИМАНИЕ! Документ и время его подписи
не прошли проверку на подлинность";
            }
            else return "ВНИМАНИЕ! Документ и время его подписи не
прошли проверку на подлинность";
      }
}
```

Pretender.h

```
#pragma once
#include <iostream>
#include <string>
#include <vector>
#include <cstdlib>
#include <ctime>
#include <cmath>
#include "Center.h"
#include "Checking.h"
using namespace std;
using namespace timestamp;
using namespace hash;
using namespace check;
namespace pretender {
      class Pretender {
            Timestamp timestamp;
            Hash hash;
            Check check;
      public:
            string getStamp();
      };
      string Pretender::getStamp() {
            string nameDocument;
            cout <<"Введите текстовый документ, который нужно подписать:";
            cin >> nameDocument;
            cout << endl;
            if (check.checkExt(nameDocument) == true) {
                  string s = check.checkFile(nameDocument);
                  long long hex = hash.getHash(s);

                  if (timestamp.setCertificate(hex) == true) {
```

```
                    if (check.checkVerText("TextFromCenter.txt",hex) ==
true) {
                        string                  str                 =
check.checkFile("TextFromCenter.txt");
                        string               stamping               =
check.checkFile("Stamp.txt");
                        string               openKeys               =
check.checkFile("OpenKeys.txt");

                        string* stampp = new string[2];
                        for (int j = 0; j < 2; j++) {
                            for (int i = 0; i < stamping.length();
i++) {
                                if (stamping[i] != ' ') stampp[j] +=
stamping[i];
                                else j++;
                            }
                        }
                        string* keys = new string[3];
                        for (int j = 0; j < 3; j++) {
                            for (int i = 0; i < openKeys.length();
i++) {
                                if (openKeys[i] != ' ') keys[j] +=
openKeys[i];
                                else j++;
                            }
                        }
                        long w = stoi(stampp[0]);
                        long s = stoi(stampp[1]);
                        long p = stoi(keys[0]);
                        long g = stoi(keys[0]);
                        long y = stoi(keys[0]);
                        long long checkText = hash.getHash(str);
                        bool res = check.checkStamp(w, s, p, g, y,
checkText);
                        if (res == true) {
                            return "Временная метка создана успешно";
                        }
                        else {
                            return "ВНИМАНИЕ! Временную метку создать
не удалось";
                        }
                    }
                }
            }
        }
}
```

Center.h

```
#pragma warning(disable : 4996)
#pragma once
#include <iostream>
#include <string>
#include <vector>
#include <cstdlib>
#include <ctime>
```

```
#include <cmath>
#include "Keys.h"
#include <conio.h>
#include <time.h>
#include <stdio.h>
#include "Hash.h"
using namespace std;
using namespace keys;
using namespace hash;
namespace timestamp {
      Keys keys;
      Hash hash;
      class Timestamp {
            char* clock();
      public:
            bool setCertificate(long h);
      };
      char* Timestamp::clock() {
            time_t td;
            td = time(0);
            return ctime(&td);
      }
     bool Timestamp::setCertificate(long h)
      {
            ofstream out1;
            out1.open("TextFromCenter.txt");
            char* timeS = clock();
            out1 << h << " ";
            out1 << timeS;
            out1.close();
            long hex = hash.getHash("TextFromCenter.txt");
            vector<long long> keyss = keys.getKeys(hex);
            long p = keyss[0];
            long g = keyss[1];
            long y = keyss[2];
            long closeKey = keyss[3];
            long k = keyss[4];
            long long g_pow_k = pow(g, k);
            long w = g_pow_k % p;
            long z = p - 1;
            bool il = false;
            int t = 0;
            while (il == false) {
                  if ((k * t - 1 > 0) && ((k * t - 1) % z == 0))
                        il = true;
                  else t = t + 1;
            }
            long s = ((abs(hex - closeKey * w)) * t) % (p - 1);
            ofstream out2;
            out2.open("Stamp.txt");
            out2 << w << " " << s;
            out2.close();
            ofstream out3;
            out3.open("OpenKeys.txt");
            out3 << p << " " << g << " " << y;
            out3.close();
            return true;
```

```
	}
}
```

Keys.h

```
#include <iostream>
#include <string>
#include <vector>
#include <cmath>
using namespace std;
namespace keys {
	class Keys {
		int random(long h);
		bool simple(unsigned long n);
		int NOD(int a, int b);
	public:
		vector<long long> getKeys(long h);
	};
	int Keys::NOD(int a, int b)
	{
		while (a != 0 and b != 0) {
			if (a > b) a = a % b;
			else b = b % a;
		}
		return (a + b);
	}
	int Keys::random(long h) {
		long long p;
		srand(time(NULL));
		for (unsigned long i = 1; i <= 100000; i++)
		{
			p = h + 2 + rand() % (h + 22 - h + 1);
			while (!simple(p))
	p = h + 2 + rand() % (h + 22 - h + 1);
			break;
		}
		return p;
	}
	bool Keys::simple(unsigned long n)
	{
		if (n < 4) return false;
		for (unsigned long j = 2; j * j <= n; j++)
			if (n % j == 0) return false;
		return true;
	}
	vector<long long> Keys::getKeys(long h) {
		long p = random(h);
		long long g;
		long long x;
		long long k;
		if (p > 10) {
			g = 2 + rand() % (10 - 2 + 1);
			while (!(NOD(g, p) == 1))
	g = 2 + rand() % (10 - 2 + 1);
			x = 2 + rand() % (10 - 2 + 1);
			k = 2 + rand() % (10 - 2 + 1);
			while (!(NOD(k, p - 1) == 1))
```

```
        k = 2 + rand() % (10 - 2 + 1);
            }
            else {
                g = 2 + rand() % (p - 2 + 1);
                while          (!(NOD(g,          p)          ==          1))
        g = 2 + rand() % (p - 2 + 1);
                x = 2 + rand() % (p - 1 - 2 + 1);
                k = 2 + rand() % (p - 1 - 2 + 1);
                while      (!(NOD(k,      p      -      1)      ==      1))
        k = 2 + rand() % (p - 1 - 2 + 1);
            }
            long long g_pow_x = pow(g, x);
            long long y = g_pow_x % p;
            vector<long long> keys = { p, g, y, x, k };
            return keys;
        }
}
```

Hash.h

```
#pragma once
#include <iostream>
#include <cmath>
#include <iomanip>
#include <sstream>
#include <vector>
#include <fstream>
using namespace std;
namespace hash {
    class Hash {
    public:
        long getHash(string s);
    };
    long Hash::getHash(string s) {
            s += " ";
            long long slength = s.length();
            long long hex = 0;
            for (int i = 0; i < slength; i++) {
                hex = hex + abs(s[i]);
            }
            while (hex % slength == 0) slength++;
            return hex % slength;
    }
}
```

Checking.h

```
#pragma once
#include <iostream>
#include <string>
#include <vector>
#include <cstdlib>
#include <cmath>
#include "Large.h"
#include <fstream>
```

```
using namespace std;
namespace check {
	class Check {
		string* mas(string s);
	public:
		string checkFile(string name);
		bool checkExt(string name);
		bool checkDate(string name);
		bool checkStampOut(string name);
		bool checkKeys(string name);
		bool checkVerText(string name,long long hex);
		bool checkStamp(long w, long s, long p, long g, long y, long
h);
	};
	string* Check::mas(string s) {
		bool ilhash = true;
		bool ilmonth = true;
		bool ilyear = true;
		int n = 0;
		int kol = 0;
		int check = 0;
		int check2 = 0;
		int position = 0;
		for (int i = 0; i < s.length(); i++) {
			if (s[i] == ' ') { cout << "ВНИМАНИЕ! Некорректный
проверочный текст" << endl; exit(1); }
			else break;
		}
		for (int i = 0; i < s.length(); i++) {
			if (s[i] == ' ') {
				if (n == 0) { check++; check2 = 0; }
				n++;
			}
			else if (s[i] != ' ') { n = 0; check2++; }
			if ((check != 3) && (n > 1))
			{
				cout << "ВНИМАНИЕ! Некорректный проверочный текст"
<< endl; exit(1);
			}
			if ((check == 3) && (n == 2)) position++;
			else if ((check == 3) && (n > 2)) {
				cout << "ВНИМАНИЕ! Некорректный проверочный текст"
<< endl; exit(1);
			}
			if ((check == 3) && (check2 > 1) && (position != 0))
			{
				cout << "ВНИМАНИЕ! Некорректный проверочный текст"
<< endl; exit(1);
			}
		}
		for (int i = 0; i < s.length(); i++) {
			if (s[i] == ' ') n++;
			else if (s[i] != ' ')  n = 0;
			if (n > 1)  s[i] = 0;
		}
		for (int i = s.length() - 1; i >= 0; --i) if (s[i] == 0)
s.erase(i, 1);
```

```
            for (int i = 0; i < s.length(); i++) if (s[i] == ' ') kol++;
            kol++;
            if (kol != 6) { cout << "ВНИМАНИЕ! Некорректный проверочный
текст"; exit(1); }
            string* massivDate = new string[kol];
            for (int j = 0; j < kol; j++) {
                  for (int i = 0; i < s.length(); i++) {
                        if (s[i] != ' ') {
                              if ((j == 0) || (j == 3) || (j == 5)) {
                                    if (s[i] >= '0' && s[i] <= '9' && s[i] !=
0)
                                          massivDate[j] += s[i];
                                    else {
                                          if (j == 0) ilhash = false;
                                          if (j == 3) ilmonth = false;
                                          if (j == 5) ilyear = false;
                                    }
                              }
                              else massivDate[j] += s[i];
                        }
                        else j++;
                  }
            }
            if ((ilhash == false) || (ilmonth == false) ||
                  (ilyear == false)) {
                  if (ilhash == false)
                        cout << "ВНИМАНИЕ! Некорректное значение хэша" <<
endl;
                  if ((ilmonth == false) || (ilyear == false))
                        cout << "ВНИМАНИЕ! Некорректное временная метка" <<
endl;
                  exit(1);
            }
            return massivDate;

      }
      string Check::checkFile(string name) {
            ifstream in;
            in.open(name);
            string s;
            getline(in, s);
            if (in.is_open()) {
                  in.close();
                  return s;
            }
            else
            {cout << "ВНИМАНИЕ! Данного текстового документа не существует"
<< endl; exit(1);}
      }
      bool Check::checkExt(string name) {
            int col = 0;
            char buf[255];
            strcpy(buf, name.c_str());
            for (int i = 0; i < name.length(); i++)
            if (buf[i] == '.') col = i;
            char testChar[5];
            strcpy(testChar, &buf[col]);
```

```
            string testString = string(testChar);

            if (testString != ".txt")
            { cout << "ВНИМАНИЕ! Некорректное расширение" << endl; exit(1);
}
            else return true;
      }
      bool Check::checkDate(string name) {
            if (checkExt(name) == true) {
                  string s = checkFile(name);
                  string* massivDate = mas(s);
                  if ((massivDate[1] == "Mon") ||
                        (massivDate[1] == "Tue") ||
                        (massivDate[1] == "Wed") ||
                        (massivDate[1] == "Fri") ||
                        (massivDate[1] == "Sat") ||
                        (massivDate[1] == "Sun")) {}
                  else { cout << "ВНИМАНИЕ! Некорректная временная метка"
<< endl; exit(1); }
                  if (((massivDate[2] == "Dec") ||
                         (massivDate[2] == "Jan")
                        || (massivDate[2] == "Mar") ||
                           (massivDate[2] == "May")
                        || (massivDate[2] == "Sep") ||
                           (massivDate[2] == "Nov"))
                        && (1 <= stoi(massivDate[3]) &&
                           (stoi(massivDate[3]) <= 31))) {
                  }
                  else if (((massivDate[2] == "Apr" || massivDate[2] ==
"Jun"
                        || massivDate[2] == "Aug" || massivDate[2] == "Jul"
                        ||    massivDate[2]    ==    "Oct"))    &&    (1    <=
stoi(massivDate[3])
                              && (stoi(massivDate[3]) <= 30))) {
                  }
                  else   if   ((massivDate[2]   ==   "Feb")   &&   (1   <=
stoi(massivDate[3])
                        && (stoi(massivDate[3]) <= 29))) {
                  }
                  else {cout << "ВНИМАНИЕ! Некорректная временная метка" <<
endl; exit(1);}
                  int kol = 0;
                  for (int i = 0; i < massivDate[4].length(); i++) if
(massivDate[4][i] == ':') kol++;
                  if (kol > 2) { cout << "ВНИМАНИЕ! Некорректная временная
метка" << endl; exit(1); }
                  else kol++;
                  string* timeh = new string[kol];
                  for (int j = 0; j < kol; j++) {
                        for (int i = 0; i < massivDate[4].length(); i++) {
                              if (massivDate[4][i] != ':') {
                                    if (massivDate[4][i] >= '0' &&
                                          massivDate[4][i] <= '9' &&
                                          massivDate[4][i] != ' ' &&
                                          massivDate[4][i] != 0)
                                          timeh[j] += massivDate[4][i];
```

```
                                    else { cout << "ВНИМАНИЕ! Некорректная
временная метка" << endl; exit(1); }
                              }
                              else j++;
                        }
                  }
                  if ((0 <= stoi(timeh[0])) &&
                        (stoi(timeh[0]) <= 23) &&
                        (0 <= stoi(timeh[1])) &&
                        (stoi(timeh[1]) <= 59) &&
                        (0 <= stoi(timeh[2])) &&
                        (stoi(timeh[2]) <= 59)) {}
                  else { cout << "ВНИМАНИЕ! Некорректная временная метка";
exit(1); }
                  return true; }}
      bool Check::checkStampOut(string name) {
            if (checkExt(name) == true) {
                  string stamp = checkFile(name);
                  int n = 0;
                  bool il = true;
                  int kol = 0;
                  for (int i = 0; i < stamp.length(); i++) {
                        if (stamp[i] == ' ') { cout << "ВНИМАНИЕ!
Некорректная подпись" << endl; exit(1); }
                        else break;
                  }
                  for (int i = 0; i < stamp.length(); i++) {
                        if (stamp[i] == ' ') n++;
                        else if (stamp[i] != ' ') n = 0;
                        if (n > 1) { cout << "ВНИМАНИЕ! Некорректная подпись"
<< endl; exit(1); }
                  }
                  for (int i = 0; i < stamp.length(); i++) {
                        if (stamp[i] == ' ') kol++;
                  }
                  kol++;
                  if (kol != 2) { cout << "ВНИМАНИЕ! Некорректная подпись"
<< endl; exit(1); }
                  string* NumberforStamp = new string[kol];
                  for (int j = 0; j < kol; j++) {
                        for (int i = 0; i < stamp.length(); i++) {
                              if (stamp[i] != ' ') {
                                    if (stamp[i] >= '0' && stamp[i] <= '9' &&
stamp[i] != 0)
                                          NumberforStamp[j] += stamp[i];
                                    else { cout << "ВНИМАНИЕ! Некорректная
подпись" << endl; exit(1); }
                              }
                              else {
                                    j++;
                              }
                        }
                  }
                  return true;
            }
      }
      bool Check::checkKeys(string name) {
```

```
            if (checkExt(name) == true) {
                  string keys = checkFile(name);
                  int n = 0;
                  bool il = true;
                  int kol = 0;
                  for (int i = 0; i < keys.length(); i++) {
                        if (keys[i] == ' ') { cout << "ВНИМАНИЕ! Некорректные
значения открытых ключей" <<endl; exit(1); }
                        else break;
                  }
                  for (int i = 0; i < keys.length(); i++) {
                        if (keys[i] == ' ') n++;
                        else if (keys[i] != ' ') n = 0;
                        if (n > 1) { cout << "ВНИМАНИЕ! Некорректные значения
открытых ключей" << endl; exit(1); }
                  }
                  for (int i = 0; i < keys.length(); i++) if (keys[i] == '
') kol++;
                  kol++;
                  if (kol != 3) { cout << "ВНИМАНИЕ! Некорректные значения
открытых ключей" << endl; exit(1); }
                  string* NumberforKeys = new string[kol];
                  for (int j = 0; j < kol; j++) {
                        for (int i = 0; i < keys.length(); i++) {
                              if (keys[i] != ' ') {
                                    if (keys[i] >= '0' && keys[i] <= '9' &&
keys[i] != 0) NumberforKeys[j] += keys[i];
                                    else { cout << "ВНИМАНИЕ! Некорректные
значения открытых ключей" << endl; exit(1); }
                              }
                              else j++; }
                  }
                  return true;
            }
      }
      bool Check::checkVerText(string name, long long hex) {
            string* massivDate{};
            if (checkExt(name) == true) {
                  string s = checkFile(name);
                  massivDate = mas(s);
            }
            if ((checkDate(name) == true) &&
                  (stoi(massivDate[0]) == hex))
                  return true;
            else return false;
      }
      bool Check::checkStamp(long w, long s, long p, long g, long y, long
h) {
            string y_pow_w = (large(y).pow(large(w))).ToString();
            string w_pow_s = (large(w).pow(large(s))).ToString();
            string g_pow_h = (large(g).pow(large(h))).ToString();
            string   check1   =   ((large(y_pow_w)   *   large(w_pow_s))   %
large(p)).ToString();
            string check2 = (large(g_pow_h) % large(p)).ToString();
            if ((check1 == check2) && (w < p)) return true;
            else return false;}
}
```