**МИНИСТЕРСТВО ТРАНСПОРТА РОССИЙСКОЙ ФЕДЕРАЦИИ**

**федеральное государственное автономное образовательное**

**учреждение высшего образования**

**«РОССИЙСКИЙ УНИВЕРСИТЕТ ТРАНСПОРТА»**

**РУТ(МИИТ)**

Кафедра «Вычислительные системы, сети и информационная безопасность»

**Отчет По Дисциплине**

**«ОСНОВЫ ИНФОРМАЦИОННОЙ БЕЗОПАСНОСТИ»**

**курсовая работа**

**На тему:**

**“Алгоритм на основе протокола Kerberos (Цербер) с применением алгоритма DES и других.”**

*Направление:* 10.03.01*Информационная безопасность*

*Профиль:**Безопасность компьютерных систем*

Выполнил:  
студент группы УИБ-212

Орлов Андрей Витальевич

Проверил:

Доцент, Голдовский Яков Михайлович

(должность, ФИО)

Ст. Преподаватель, Цыганова Наталия Алексеевна

(должность, ФИО)

**МОСКВА 2022**

**СОДЕРЖАНИЕ**

1 ТЕОРЕТИЧЕСКОЕ ОПИСАНИЕ3

2 БЛОК-СХЕМЫ РАБОТЫ АЛГОРИТМА14

3 ТЕСТОВЫЙ ПРИМЕР32

4 КОД ПРОГРАММЫ34

5 ИЛЛЮСТРАЦИЯ РЕЗУЛЬТАТА РАБОТЫ ПРОГРАММЫ45

6 СПИСОК ИСПОЛЬЗУЕМОЙ ЛИТЕРАТУРЫ46

1. **ТЕОРЕТИЧЕСКОЕ ОПИСАНИЕ**

Протокол безопасности Kerberos стал основой современной кибербезопасности. Фактически, он настолько хорошо интегрирован, что большинство пользователей или даже разработчиков вообще забывают о нем. Этот скрытый статус может затруднить получение информации о том, что такое Kerberos и как он работает, особенно со всеми различными формами, которые этот протокол принимает сегодня.

Kerberos — это протокол проверки подлинности компьютерной сети, предназначенный для упрощения и безопасности проверки подлинности. Основная идея Kerberos вращается вокруг использования локальной формы личной идентификации, называемой билетами, которые предоставляются сервером аутентификации. Каждый билет принадлежит определенным областям, которые определяют, к каким службам он предоставляет доступ. Эти билеты зашифрованы, и для их использования требуется несколько уровней дешифрования. Эта система билетов гарантирует, что конфиденциальная информация, такая как пароли, никогда не будет отправлена ​​по сети. Для шифрования билетов в данном случае использовался алгоритм DES.

Протокол Kerberos получил широкое признание с момента его создания в Массачусетском технологическом институте в 1980-х годах. Теперь он встроен в бесчисленное количество зависимых от безопасности реализаций в Интернете, и почти все компании ежедневно взаимодействуют по крайней мере с одной системой Kerberos.

Kerberos так широко используется из-за его простоты и непревзойденной безопасности данных. Вот лишь некоторые из его преимуществ:

1. Безопасность: Kerberos никогда не передает пароли по сети. Вместо этого Kerberos подтверждает личность пользователя, отправляя ограниченные по времени криптографические сообщения, которые становятся недействительными по истечении установленного периода. Даже если сообщения были перехвачены и расшифрованы, они были бы бесполезны в считанные минуты!
2. Единый вход: Kerberos требует, чтобы пользователь вводил пароль только один раз при первой аутентификации клиента. С этого момента пользователь имеет доступ ко всем керберизованным службам в области Kerberos без необходимости повторно вводить свой пароль. Единый вход упрощает работу с несколькими службами, устраняя необходимость в нескольких требованиях входа в систему.
3. Надежная третья сторона: Kerberos использует централизованный сервер аутентификации, известный как Центр распространения ключей (KDC), которому по умолчанию доверяют все другие устройства в сети. Все запросы аутентификации, такие как криптографические сообщения, маршрутизируются через этот сервер. Такой аутсорсинг гарантирует, что конфиденциальная информация не будет храниться на локальном компьютере.
4. Взаимная аутентификация: в Kerberos оба конца связи должны быть аутентифицированы, прежде чем соединение будет разрешено. Взаимная аутентификация резко снижает возможность мошенников обманным путем заставить системы отправлять конфиденциальную информацию.

Рассмотрим пример взаимной аутентификации.

Пользователь в сети, использующий Kerberos, может пройти аутентификацию на почтовом сервере, чтобы доказать, что он тот, за кого себя выдает. С другой стороны, почтовый сервер также должен подтвердить, что он действительно является почтовым сервером, а не какой-либо другой службой в сети, претендующей на роль почтового сервера. Если обе стороны аутентифицированы, соединение устанавливается. [1]

Основные компоненты Kerberos.

1. Центр распространения ключей

Центр распространения ключей (KDC) — это центральный процесс Kerberos, содержащий сервер аутентификации (AS) и службу выдачи билетов (TGS). Его основная функция — быть посредником между этими двумя, ретранслируя сообщения от AS, выдает билет на выдачу билетов (TGT), а затем передает его для шифрования с помощью TGS. После этого KDC мало влияет на процесс аутентификации.

1. Билет на выдачу билетов

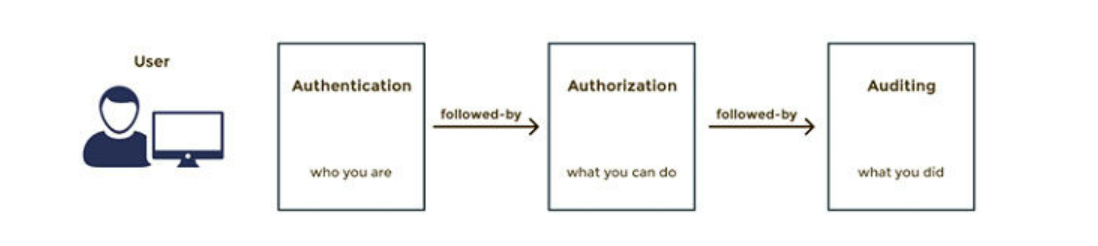
Этот билет выдается KDC после успешной аутентификации клиента. TGT зашифрован и содержит разрешения на то, к каким службам может получить доступ клиент, как долго предоставляется доступ, а также ключ сеанса, используемый для связи с клиентом. Клиенты не могут расшифровать TGT, так как у них нет ключа TGS. Следовательно, они должны слепо представить TGT желаемым службам (которые могут получить доступ к TGS) и позволить службам решить, может ли клиент получить к нему доступ. Скрывая TGT от клиента, Kerberos предотвращает мошенническое копирование или изменение разрешений клиентом.

1. Сервер аутентификации

Сервер аутентификации — это первая остановка при аутентификации с помощью Kerberos. Сначала клиент должен аутентифицироваться в AS, используя имя пользователя и пароль для входа.

После этого AS перенаправляет имя пользователя в KDC, который, в свою очередь, предоставляет TGT. Без выполнения этого первого шага клиент не сможет взаимодействовать с какой-либо другой частью системы Kerberos.  
Служба предоставления билетов действует как привратник между клиентами, владеющими TGT, и различными службами в сети. Когда клиент хочет получить доступ к услуге, он должен представить свой TGT в TGS.

Затем TGS аутентифицирует TGT и устанавливает сеансовый ключ, совместно используемый сервером и клиентом. Если TGS подтверждает, что клиентский TGT включает доступ к желаемой службе, клиенту предоставляется доступ для запроса услуги, что предоставлено на рисунке 1.

  
Рисунок 1 – Краткая схема алгоритма

Проверка подлинности Kerberos состоит из 4 этапов, в зависимости от того, какие компоненты взаимодействуют между собой:

1. Вход пользователя / клиента: на этом этапе происходит взаимодействие между пользователем и клиентом. Пользователь вводит в клиент свое имя пользователя и пароль. Затем клиент преобразует этот пароль в ключ шифрования, хранящийся локально. Если это завершится правильно, клиент может начать аутентификацию с помощью AS.

Аутентификация клиента / AS: на этом этапе клиент и сервер аутентификации подключаются для аутентификации имени пользователя и проверки того, что они являются частью системы. Затем AS проверяет, что имя пользователя уже задокументировано в системе. В этом случае Клиент и AS обмениваются зашифрованными проверочными сообщениями для проверки друг друга. В конце оба аутентифицируются, соединение устанавливается, и клиент может перейти к аутентификации с помощью службы.

1. Аутентификация клиента / службы: на этом этапе клиент и сервер должны аутентифицировать друг друга в соответствии с практикой взаимной аутентификации. Клиент и сервер обмениваются зашифрованными проверочными сообщениями, как на предыдущем этапе. Если все это проходит, клиент и служба аутентифицируются, и клиент получает разрешение запросить их службу.
2. Клиент / запрос службы: наконец, клиент может запросить именованную службу у сервера службы. Затем сервисный сервер проверяет, доступна ли запрошенная услуга. Если да, сервер службы предоставляет услугу клиенту. Поскольку клиент прошел аутентификацию на всех этапах этого процесса, он может продолжать использовать службу до истечения срока действия разрешений.

Рассмотрим пример процесса Kerberos.

Каждый из этих этапов состоит из нескольких этапов, но в режиме реального времени процесс происходит очень быстро. Чтобы поместить то, что мы узнали выше, в контекст, давайте рассмотрим пример из реальной жизни. В начале рабочего дня вы вводите пароль в свой клиент. Пароль аутентифицируется AS, а затем KDC предоставляет вам TGT. В этом билете есть набор ключей от dataScience службы.

Затем TGT кэшируется на вашем компьютере для дальнейшего использования. Этот доступ позволяет вам использовать любые услуги в dataScience области, например, доступ к покупательскому поведению клиентов. [2]

Затем вы можете получить доступ к этой службе в любое время без необходимости каждый раз аутентифицировать свои разрешения. Однако, если вы попытаетесь получить доступ к любой из служб из finances области, вам будет отказано, потому что ваш TGT не имеет ключей к этой области.

В конце рабочего дня срок действия вашего TGT истекает, и вы не сможете снова получить доступ к этим службам, пока не получите новый билет при входе в систему на следующий день.

Разбивка процесса Kerberos (16 шагов):

1. Войти

Пользователь вводит свое имя пользователя и пароль. Затем клиент с поддержкой Kerberos преобразует этот пароль в секретный ключ клиента.

1. Запросы клиентов на сервер выдачи билетов

Затем клиент отправляет серверу аутентификации текстовое сообщение, содержащее:

1. введенное имя пользователя;
2. название запрашиваемой услуги;
3. сетевой адрес пользователя;
4. на какое время они запрашивают доступ;  
    5) сервер проверяет имя пользователя.

Имя пользователя проверяется на соответствие проверенным именам пользователей, хранящимся в KDC. Если имя пользователя знакомо, программа продолжится.

1. Выдача билета. Билет возвращается клиенту.

Сервер аутентификации отправляет клиенту два зашифрованных сообщения:

1. MessageA может быть расшифровано с помощью секретного ключа клиента, созданного на шаге 1. Оно содержит имя TGS, временную метку, время жизни билета и вновь предоставленный сеансовый ключ сервера предоставления билетов.
2. MessageB является билетом на выдачу билета и может быть расшифрован только с помощью секретного ключа TGS. Он содержит ваше имя пользователя, имя TGS, метку времени, ваш сетевой адрес, время жизни билета и тот же ключ сеанса TGS.
3. Клиент получает сеансовый ключ TGS.

Теперь клиент расшифровывает messageA, используя секретный ключ клиента и предоставляя клиенту доступ к ключу сеанса TGS. Message B хранится локально в зашифрованном состоянии.

1. Клиент запрашивает доступ к службе с сервера

Теперь клиент отправляет обратно два сообщения:

1. MessageC представляет собой незашифрованное сообщение, содержащее имя запрошенной службы, время существования и все еще зашифрованное message B.
2. MessageD является аутентификатором, зашифрованным с помощью сеансового ключа TGS, содержит ваше имя и временную метку
3. Сервер проверяет службу.

Затем TGS проверяет, существует ли служба запросов в KDC. Если это так, программа продолжается.

1. Сервер получает сеансовый ключ TGS.

Теперь сервер получает все еще зашифрованные messageB, отправленные messageC. MessageB (TGT) затем расшифровывается с использованием секретного ключа TGS сервера, давая серверу сеансовый ключ TGS. Теперь с помощью этого сеансового ключа TGS сервер может расшифровать message D. Теперь у сервера есть отметка времени и имя из messageB и message D (сообщения аутентификатора). Сервер следит за тем, чтобы имена и временные метки совпадали, чтобы предотвратить мошеннические сообщения. Он также проверяет метку времени на соответствие времени жизни билета, чтобы убедиться, что время ожидания не истекло.

1. Сервер генерирует служебный сеансовый ключ.

Затем сервер генерирует случайный ключ сеанса службы и еще два сообщения.

1. MessageE зашифрован с помощью секретного ключа службы и содержит ваше имя, имя запрошенной службы, метку времени, ваш сетевой адрес, время жизни билета и ключ сеанса службы.
2. Message F шифруется с помощью сеансового ключа TGS, хранимого как клиентом, так и сервером. Это сообщение содержит всю ту же информацию, messageE кроме вашего имени пользователя и сетевого адреса.
3. Клиент получает ключ сеанса обслуживания.

Используя ключ сеанса TGS, кэшированный на шаге 5, клиент расшифровывает messageF, чтобы получить ключ сеанса службы.

1. Клиент связывается с Сервисом

Теперь клиент отправляет еще два сообщения, на этот раз службе:

Message G - еще одно сообщение аутентификатора, на этот раз зашифрованное с помощью сеансового ключа службы. Он содержит ваше имя и метку времени.

MessageH является копией messageE, которая все еще зашифрована служебным секретным ключом.

1. Расшифровка сервисов MessageG

Затем служба расшифровывает messageH своим секретным ключом службы, чтобы получить ключ сеанса службы изнутри. С помощью этого ключа сервис расшифровывает messageG.

1. Сервис проверяет запрос

Затем служба проверяет запрос, сравнивая имена пользователей, временные метки и время жизни из messageG и messageH.

1. Сервис аутентифицируется для клиента.

Затем служба отправляет messageI зашифрованные с помощью сеансового ключа службы, хранимого как службой, так и клиентом. Message I- аутентификатор, содержащий идентификатор службы и временную метку.

1. Клиент проверяет услугу.

Затем клиент расшифровывает messageI, используя ключ сеанса службы, кэшированный с шага 10. Затем клиент проверяет идентификатор и временные метки, содержащиеся в нем. Если оба соответствуют ожидаемым результатам, услуга считается безопасной.

1. Свободное общение между клиентом и службой:

Уверенный в том, что и клиент, и служба взаимно аутентифицированы, Kerberos позволяет клиенту связываться со службой.

Хотя Kerberos по-прежнему является самым безопасным протоколом, его можно взломать, как и любой другой. Kerberos, долгое время выступавший в качестве отраслевого стандарта, дал хакерам достаточно времени для преодоления системы. [1]

Хакеры нашли 5 основных способов обойти систему Kerberos, основанных на нацеливании на уязвимые системные настройки, слабые пароли или распространение вредоносного ПО. Рассмотрим каждый из 5 типов атак:

1. Передача билета: этот метод создает ложный сеансовый ключ путем подделки ложного TGT. Затем хакер может представить TGT службе как действительные учетные данные. Наличие сеансового ключа позволяет этой подделке обходить все этапы проверки Kerberos, которые предшествуют этапу предоставления сеансового ключа.
2. Золотой билет: этот метод подделывает билет со статусом администратора. Хакер имеет неограниченный доступ ко всему домену при использовании этого билета; доступны отдельные устройства, серверы, данные и настройки. Хакеры могут создать «золотой билет» только в том случае, если у них есть доступ к машине администратора, обычно через установленное вредоносное ПО.
3. Серебряный билет: подобно атаке Золотого билета, серебряные билеты — это поддельный билет проверки подлинности службы, который предоставляет доступ к службе. Этот метод дает меньший доступ, чем атака по золотому билету, но его также труднее обнаружить. Все взаимодействия на этом этапе являются взаимодействиями клиент/служба, что позволяет хакеру избежать мер безопасности, установленных в KDC.
4. Грубая сила: самый очевидный метод, грубая форсировка, включает использование автоматического подбора паролей для ввода тысяч паролей до тех пор, пока не будет найден правильный. Для него не требуются украденные учетные данные, но его легко обнаружить из-за нечеловеческого поведения при входе.
5. Вредоносное ПО со скрытым ключом бэкдора: в этом методе хакеры устанавливают скрытый ключ-ключ доступа к бэкдору в систему, чтобы позволить им войти в систему как любой пользователь в любое время в будущем. Этот метод требует ранее успешной атаки (2), поскольку эти скелетные ключи могут быть установлены только с административным доступом. Это одни из самых сложных атак для обнаружения, поскольку взлом и атака могут произойти с разницей в годы.

Также Kerberos имеет и несколько других недостатков. [4]

Единая точка отказа: требуется постоянное наличие центрального сервера. Когда сервер Kerberos падает, новые пользователи не могут войти. Это может быть устранено с помощью нескольких серверов Kerberos и резервных механизмов аутентификации.

Kerberos имеет строгие требования к времени, что означает, что часы участников должны быть синхронизированы в заданных пределах. Мандаты имеют время жизни и, если часы клиента не синхронизированы с часами сервера Kerberos, аутентификация не будет выполнена. Конфигурация по умолчанию требует, чтобы часы расходились не боле чем на пять минут друг от друга. На практике, как правило, используются демоны Network Time Protocol для синхронизации часов у клиентов.

Протокол администрирования не стандартизирован и зависит от конкретной реализации сервера. Смена пароля описана в RFC 3244.

В случае использования симметричной криптографии (Kerberos может работать с использованием как симметричной, так и асимметричной (с открытым ключом) криптографии), так как все способы аутентификации управляются централизованно центром распределения ключей (KDC), эта особенность инфраструктуры аутентификации позволит злоумышленнику выдавать себя за пользователя.

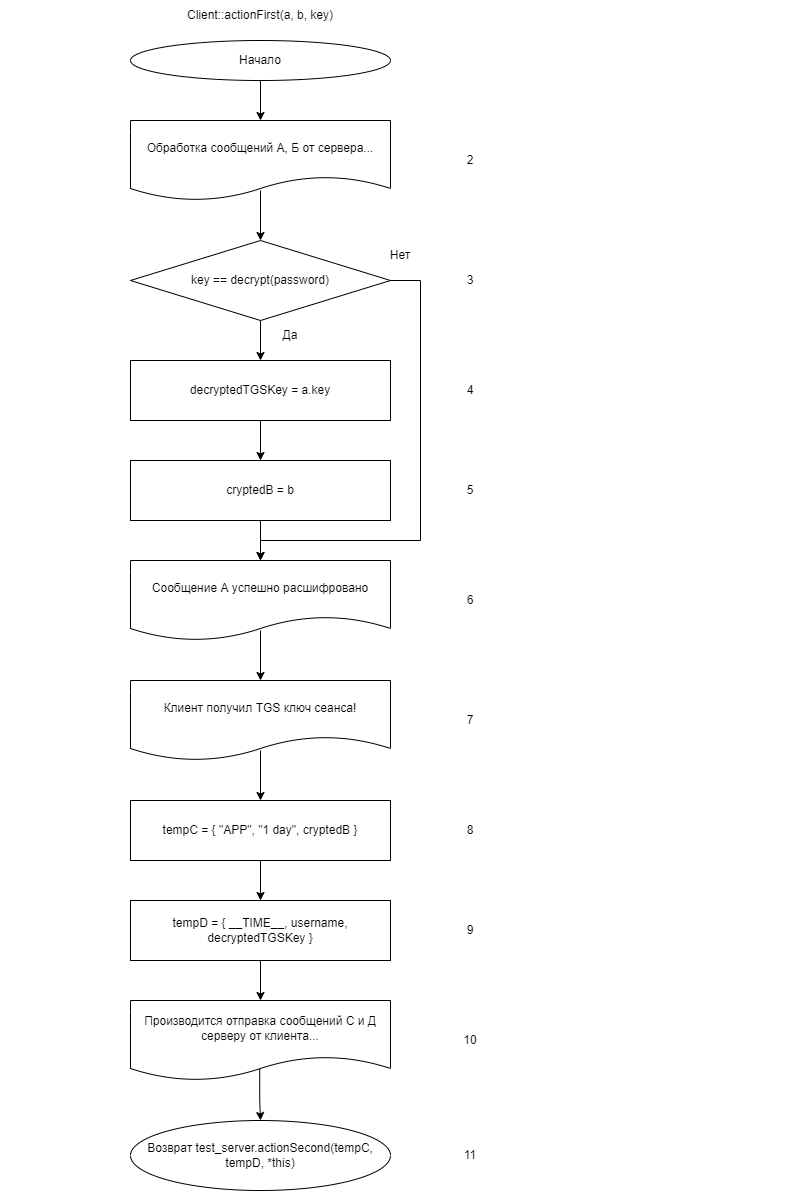
Каждый сетевой сервис, требующий смены имени хоста, должен будет обновить собственный набор ключей Kerberos. Это усложняет использование виртуального хостинга и кластеров.

Kerberos требует, чтобы учетные записи пользователей, клиенты и пользователи услуг на сервере, все доверяли серверу Kerberos (все должны быть в одном и том же домене с Kerberos или в доменах, имеющих доверительные отношения друг с другом). Kerberos не может использоваться в случаях, когда пользователи хотят подключаться к службам от неизвестных / ненадежных клиентов, как в обычном интернете.

1. **БЛОК-СХЕМЫ РАБОТЫ АЛГОРИТМА**



Рассмотрим блок-схемы функций программы:



actionFirst

Рисунок 2 – Блок-схема первого шага

actionThirddd

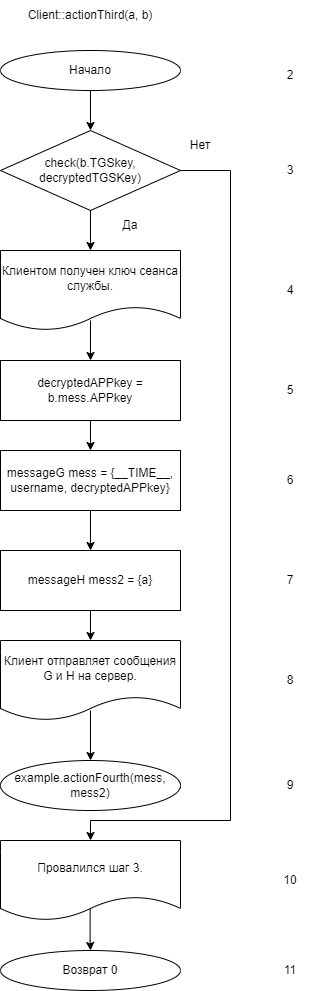


Рисунок 3 – Блок-схема третьего шага

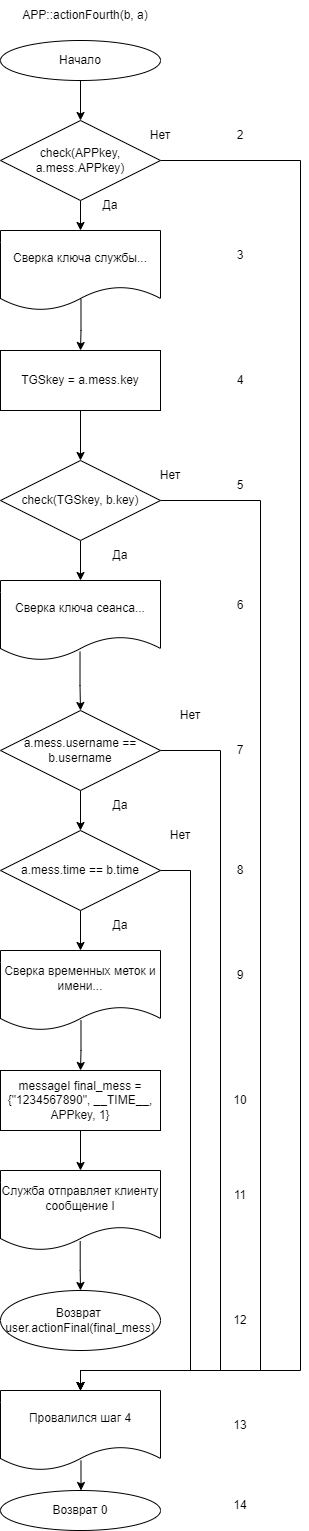
­­

Рисунок 4 – Блок-схема четвертого шага

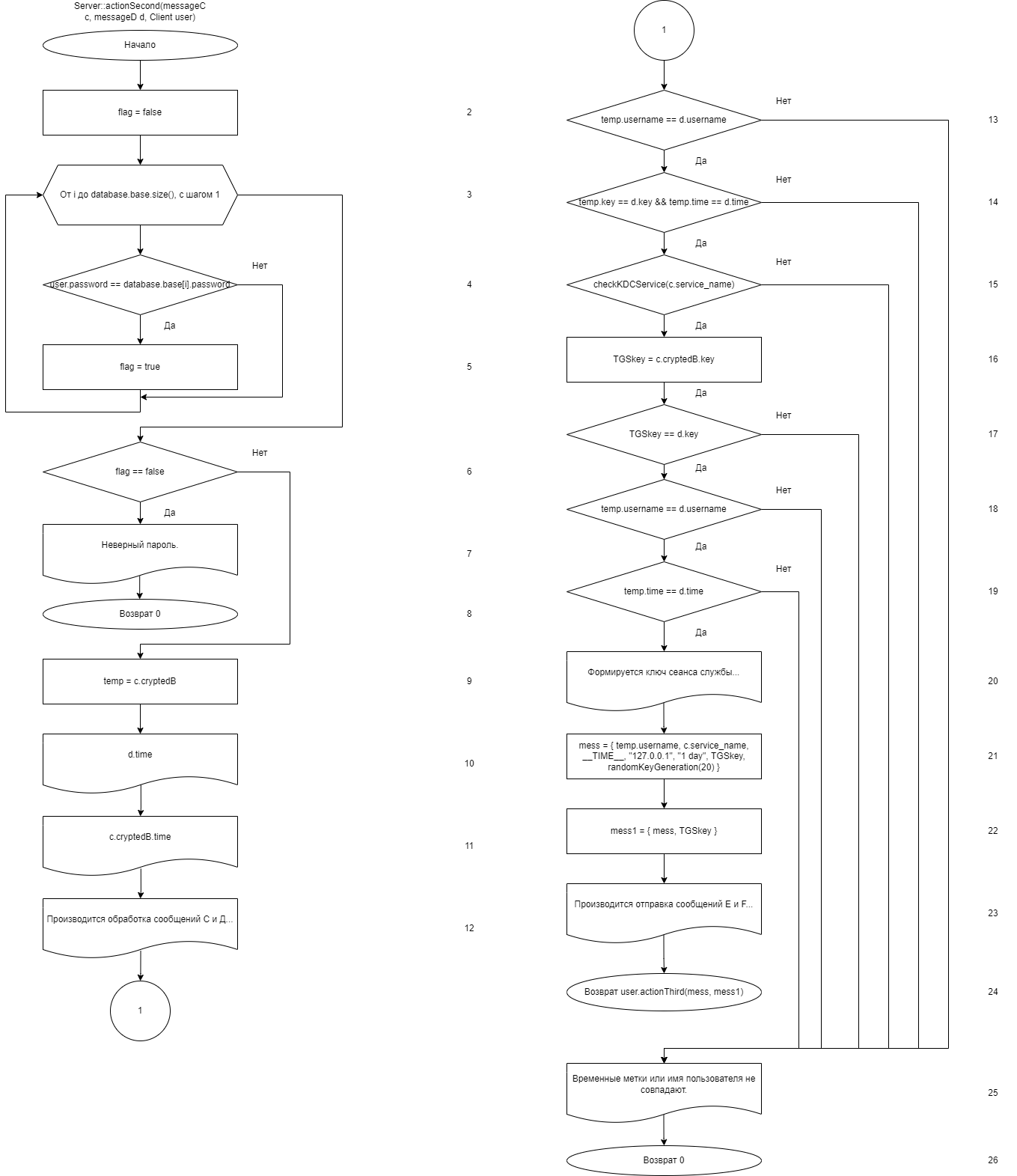


Рисунок 5 – Блок-схема второго шага

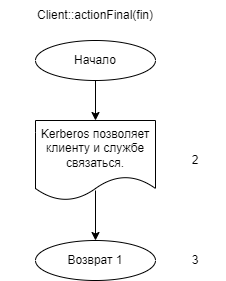


Рисунок 6 – Блок-схема пятого шага

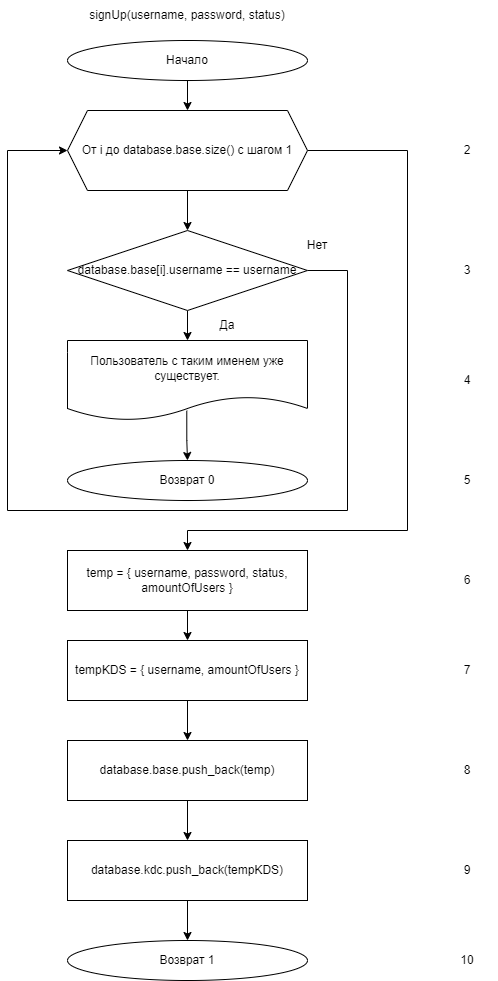


Рисунок 7 – Блок-схема регистрации

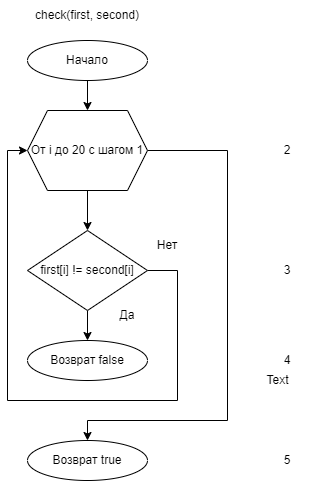


Рисунок 8 – Блок-схема проверки на идентичность

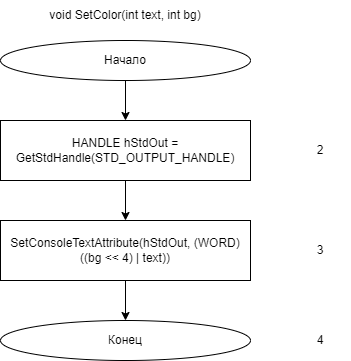


Рисунок 9 – Блок-схема задания цвета текста в консоли

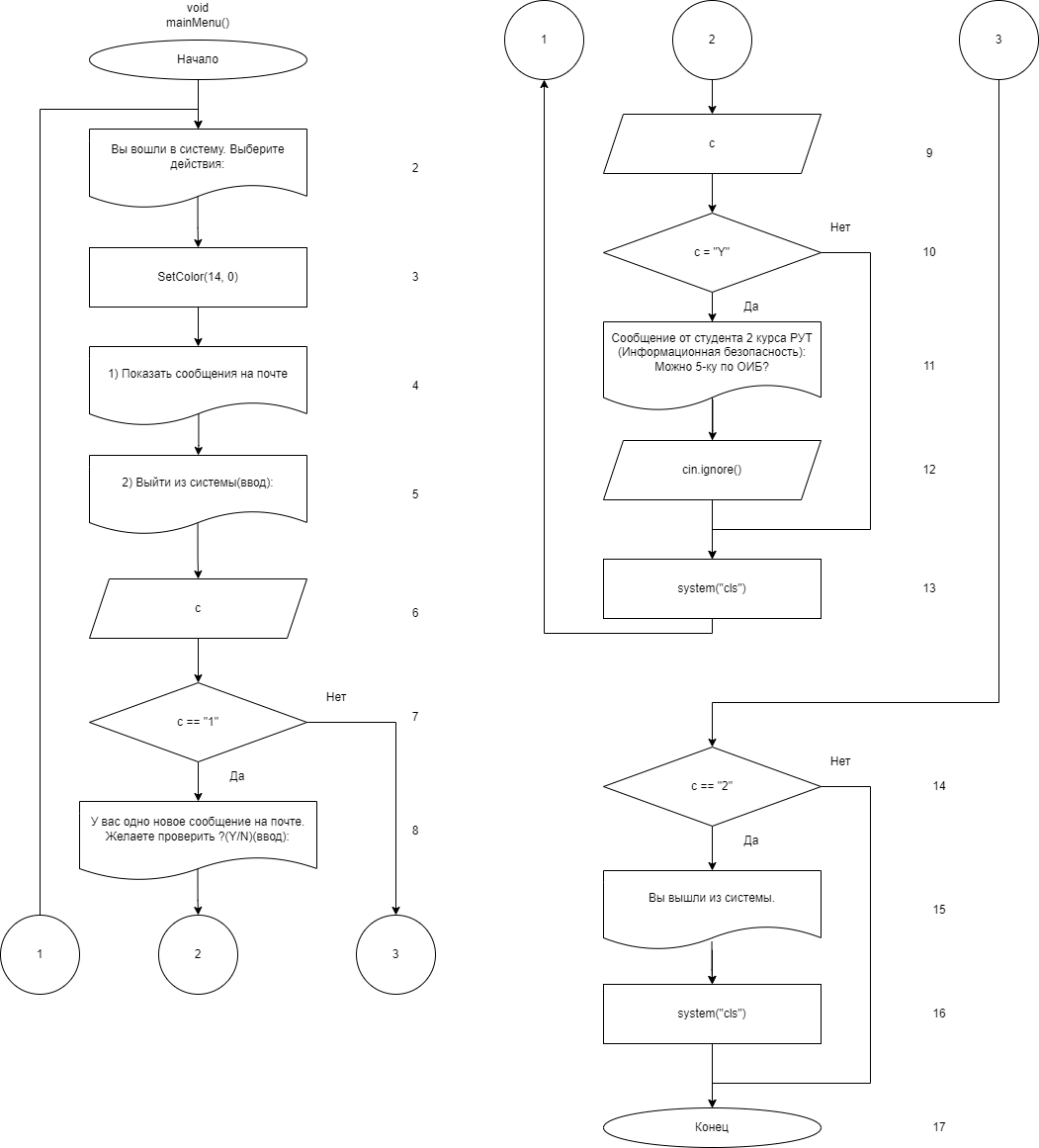
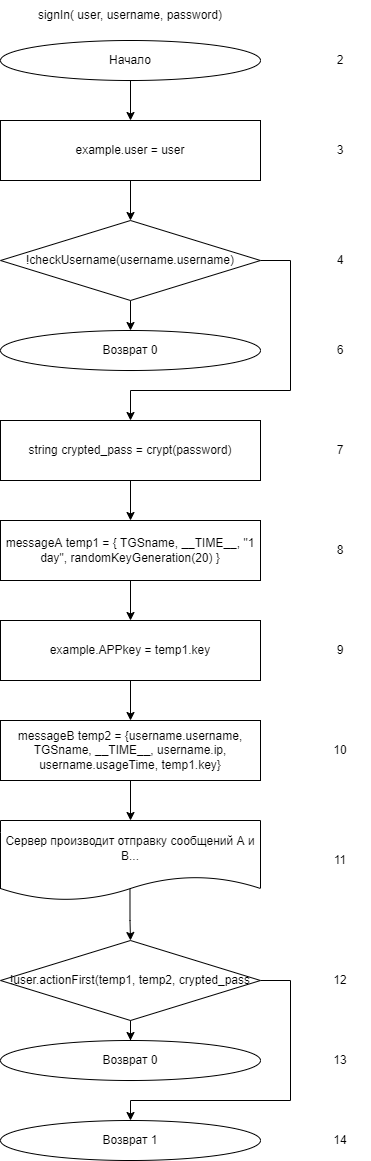
**­**

Рисунок 10 – Блок-схема меню программы (при входе)



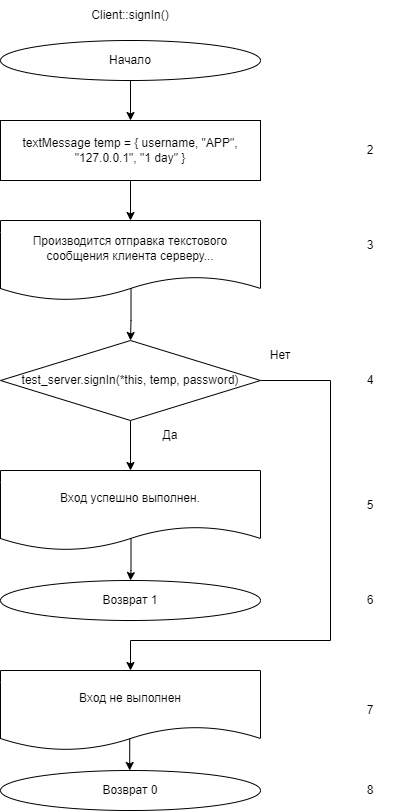


Рисунок 11 – Блок-схема функции входа (со стороны сервера)

Рисунок 12 – Блок-схема функции входа (со стороны клиента)

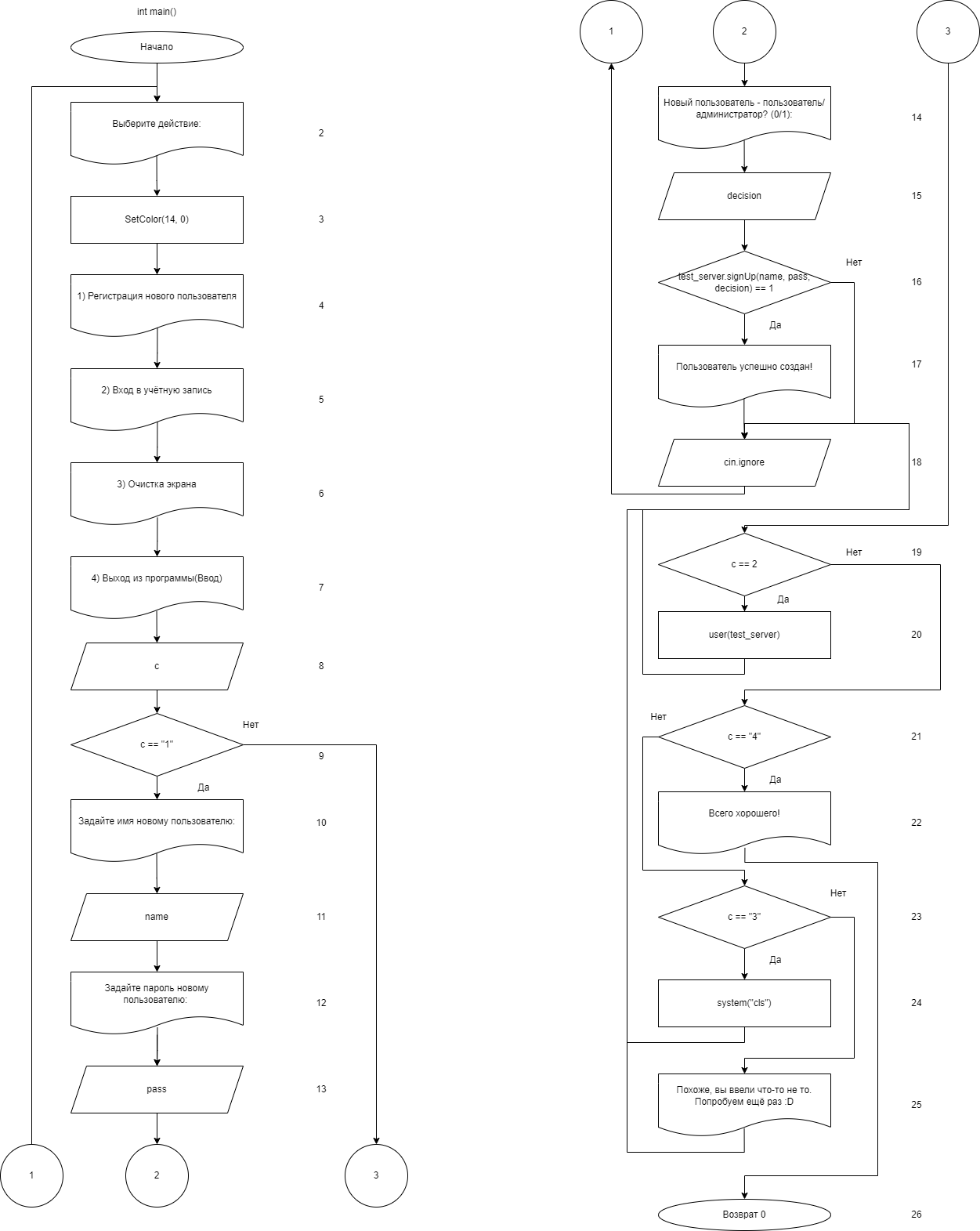


Рисунок 13 – Блок-схема меню программы (при авторизации)

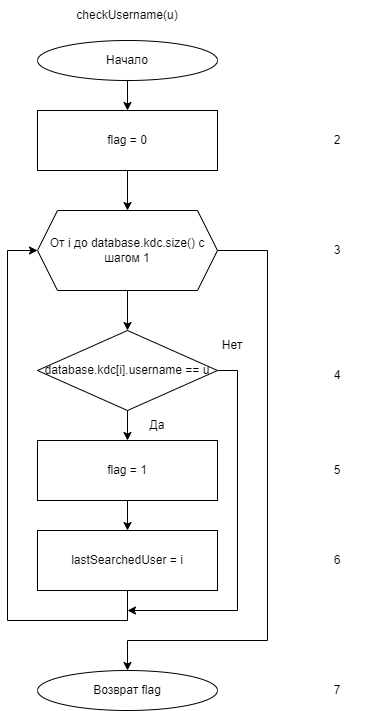
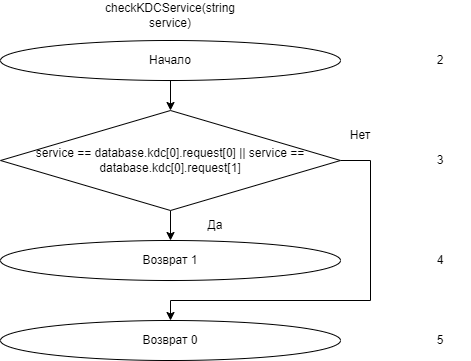
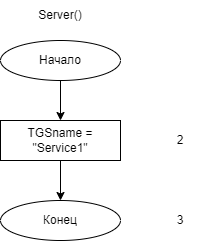
****

Рисунок 14 – Блок-схема проверки службы в кдс

Рисунок 15 – Блок-схема проверки имени пользователя в кдс

****

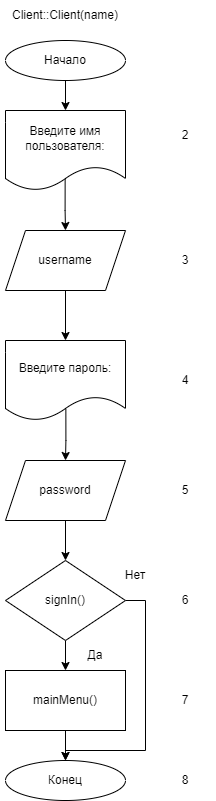
****

Рисунок 16 – Блок-схема конструктора сервера по умолчанию

Рисунок 17 – Блок-схема конструктора пользователя

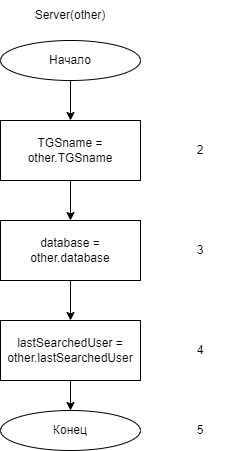
****

Рисунок 18 – Блок-схема конструктора копирования сервера

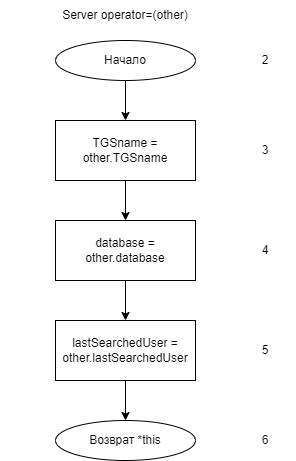
****

Рисунок 19 – Блок-схема перегрузки присваивания сервера

**3 ТЕСТОВЫЙ ПРИМЕР**



Local storage

Client

App

Server

K

D

C

(из F)

10

4

16

15

9

14

13

12

11

5

3

8

1

7

2

6

1. Отправка текстового сообщения серверу от клиента.
2. Проверка имени и пароля пользователя в KDC.
3. Отправка A, B от сервера клиенту.
4. Сохранение TGS из A в локальной памяти клиента.
5. Отправка C и D от сервера клиенту.
6. Проверка службы в KDC.
7. Сверка имён, меток из B, D.
8. Отправка E, F от сервера клиенту.
9. Расшифровка F хешированным TGS.
10. Сохранение TGT в локальную память клиента.
11. Отправка сообщений G, H от клиента службе.
12. Расшифровка G(TGT), расшифровка G(H)(TGT).
13. Сверка временных меток, имени пользователя.
14. Отправка I от службы клиенту.
15. Расшифровка I через TGT.
16. Вход в систему, разрешён Kerberos.

Разберём пошагово шифрование.

19В-Т-Шифрование Kerberos (16 шагов)

1. Клиент вводит логин, пароль (хэширование).

текстовое сообщение

1. Клиент Сервер.

поиск имени

1. Сервер КДС.

crypted by TGS

A

B

1. Сервер Клиент.

crypted by password

D

C

1. Клиент Сервер.

D crypted by TGS

1. Сохранение TGS в клиенте.

“App”

1. Сервер KDC.
2. Сервер получает TGS из B, расшифровывается C в D, сверка временных меток, имени.

crypted by TGS

F

E

1. Сервер Клиент.

crypted by service key (TGT)

1. Decrypt F with ключ TGS, кэшированного с 6 шага. (Клиент), кеширование ключа службы в клиенте.

H

G

1. Клиент Служба (App).

crypted by TGT

1. Служба сверяет временные метки, служба сверяет имена пользователей.
2. Расшифровка Службой сообщений G и H.

I

1. Служба Клиент

crypted by TGT

1. Клиент расшифровывает I через TGT, кэширован с шага 10
2. Kerberos разрешает

Message A

1. TGS.
2. \_\_TIME\_\_.
3. Lifetime.
4. Ключ.

Message B

1. Username.
2. TGS name.
3. \_\_TIME\_\_.
4. IP.
5. Lifetime.
6. TGS key.

Message C

1. App name.
2. Lifetime.
3. B.

Message D

1. Username.
2. \_\_TIME\_\_.

Message E

1. Username.
2. App name.
3. \_\_TIME\_\_.
4. IP.
5. Lifetime.
6. TGS.

Message G

1. \_\_TIME\_\_.
2. Username.

Message F

App key.

Message H

E still crypted.

Message I

Идентификатор входа в службу.

**4 КОД ПРОГРАММЫ**



//-------------------------------------------------------------------- database.h

#pragma once

#include <iostream>

#include <string>

#include <vector>

#include <time.h>

#include <math.h>

#include <stdio.h>

#include <stdlib.h>

using namespace std;

struct KDC {

string username;

int id;

vector<string> request = { "APP", "signUp" };

};

struct textMessage {

string username;

string service;

string ip;

string usageTime;

};

struct User {

string username;

string password;

bool isAdmin;

int id;

};

struct Database {

vector<KDC> kdc;

vector<User> base;

};

struct messageA {

string tgs\_name;

string time;

string lifetime;

string key;

};

struct messageB {

string username;

string tgs\_name;

string time;

string ip;

string lifetime;

string key;

};

struct messageC {

string service\_name;

string lifetime;

messageB cryptedB;

};

struct messageD {

string time;

string username;

string key;

};

struct messageE {

string username;

string service\_name;

string time;

string ip;

string lifetime;

string key;

string APPkey;

};

struct messageF {

messageE mess;

string TGSkey;

};

struct messageG {

string time;

string username;

string key;

};

struct messageH {

messageE mess;

};

struct messageI {

string id;

string time;

string key;

bool authenthication;

};

string crypt(string key) {

for (int i = 0; i < key.size(); i++)

swap(key[i], key[key.size() - i - 1]);

return key;

}

string decrypt(string key) {

for (int i = 0; i < key.size(); i++)

swap(key[i], key[key.size() - i - 1]);

return key;

}

// Function to randomly generates password

// of length N

string randomKeyGeneration(const int N)

{

srand((unsigned int)(time(NULL)));

char numbers[] = "0123456789";

char letter[] = "abcdefghijklmnoqprstuvwyzx";

char LETTER[] = "ABCDEFGHIJKLMNOQPRSTUYWVZX";

char symbols[] = "!@#$^&\*?";

char\* password = new char[N];

int randomizer = rand() % 4;

for (int i = 0; i < N; i++) {

switch (randomizer) {

case 1:

password[i] = numbers[rand() % 10];

randomizer = rand() % 4;

break;

case 2:

password[i] = symbols[rand() % 8];

randomizer = rand() % 4;

break;

case 3:

password[i] = LETTER[rand() % 26];

randomizer = rand() % 4;

break;

case 4:

password[i] = letter[rand() % 26];

randomizer = rand() % 4;

break;

}

}

return password;

}

//---------------------------------------------------------------- Главный файл

#include "database.h"

#include <Windows.h>

using namespace std;

void SetColor(int text, int bg) {

HANDLE hStdOut = GetStdHandle(STD\_OUTPUT\_HANDLE);

SetConsoleTextAttribute(hStdOut, (WORD)((bg << 4) | text));

}

bool check(string first, string second) {

for (int i = 0; i < 20; i++) {

if (first[i] != second[i]) {

return false;

}

}

return true;

}

class Server;

// Пользовательская сторона, клиент.

class Client {

friend class Server;

string username;

string password;

string decryptedTGSKey;

string decryptedAPPkey;

messageB cryptedB;

public:

bool signIn();

Client() { };

Client(Server&);

bool actionFirst(messageA, messageB, string);

bool actionThird(messageE, messageF);

bool actionFinal(messageI);

};

// Служба, к которой мы стараемся получить доступ

class APP {

public:

string APPkey;

string TGSkey;

Client user;

bool actionFourth(messageG, messageH);

APP() { };

};

APP example;

// Сервер аутенфикации

class Server {

public:

// Ключ TGS

string TGSkey;

// Ключ службы

string APPkey;

// Всё остальное

friend class Client;

static const int amountOfUsers = 0;

string TGSname;

Database database;

int lastSearchedUser;

// Проверка наличия пользователя в KDC

bool checkUsername(string u) {

bool flag = 0;

for (int i = 0; i < database.kdc.size(); i++)

if (database.kdc[i].username == u) {

flag = 1;

lastSearchedUser = i;

}

return flag;

}

bool actionSecond(messageC, messageD, Client);

// Вход пользователя в систему

bool signIn(Client& user, textMessage username, string password) {

example.user = user;

if (!checkUsername(username.username))

return 0;

string crypted\_pass = crypt(password);

messageA temp1 = { TGSname, \_\_TIME\_\_, "1 day", randomKeyGeneration(20) };

example.APPkey = temp1.key;

messageB temp2 = {username.username, TGSname, \_\_TIME\_\_, username.ip, username.usageTime, temp1.key};

cout << "Сервер производит отправку сообщений A и В..." << endl;

if (!user.actionFirst(temp1, temp2, crypted\_pass))

return 0;

return 1;

}

Server() {

TGSname = "Service1";

}

bool checkKDCService(string service) {

if (service == database.kdc[0].request[0] || service == database.kdc[0].request[1])

return 1;

return 0;

}

Server(const Server& other) {

TGSname = other.TGSname;

database = other.database;

lastSearchedUser = other.lastSearchedUser;

}

Server operator=(const Server& other) {

TGSname = other.TGSname;

database = other.database;

lastSearchedUser = other.lastSearchedUser;

return \*this;

}

// Регистрация пользователя в системе

bool signUp(string username, string password, bool status) {

for (int i = 0; i < database.base.size(); i++) {

if (database.base[i].username == username) {

cout << "Пользователь с таким именем уже существует.\n";

system("pause");

system("cls");

return 0;

}

}

User temp = { username, password, status, amountOfUsers };

KDC tempKDS = { username, amountOfUsers };

database.base.push\_back(temp);

database.kdc.push\_back(tempKDS);

return 1;

}

};

Server test\_server;

bool Client::signIn() {

textMessage temp = { username, "APP", "127.0.0.1", "1 day" };

cout << "Производится отправка текстового сообщения клиента серверу..." << endl;

if (test\_server.signIn(\*this, temp, password)) {

cout << "Вход успешно выполнен.\n";

system("pause");

return 1;

}

else {

cout << "Вход не выполнен\n";

system("pause");

system("cls");

return 0;

};

}

// Обработка присланных сообщений а и б, отправка С и Д.

bool Client::actionFirst(messageA a, messageB b, string key) {

cout << "Обработка сообщений А, Б от сервера..." << endl;

if (key == decrypt(password)) {

decryptedTGSKey = a.key;

cryptedB = b;

}

cout << "Сообщение А успешно расшифровано." << endl;

cout << "Клиент получил TGS ключ сеанса!" << endl;

messageC tempC = { "APP", "1 day", cryptedB };

messageD tempD = { \_\_TIME\_\_, username, decryptedTGSKey };

cout << "Производится отправка сообщений C и Д серверу от клиента..." << endl;

return test\_server.actionSecond(tempC, tempD, \*this);

};

// Обработка С и Д, отправка сообщений Е и Ф.

bool Server::actionSecond(messageC c, messageD d, Client user) {

bool flag = false;

for (int i = 0; i < database.base.size(); i++) {

if (user.password == database.base[i].password)

flag = true;

}

if (flag == false) {

cout << "Неверный пароль." << endl;

return 0;

}

messageB temp = c.cryptedB;

cout << d.time << endl;

cout << c.cryptedB.time << endl;

cout << "Производится обработка сообщений С и Д..." << endl;

if (temp.username == d.username)

if (temp.key == d.key && temp.time == d.time)

if (checkKDCService(c.service\_name)) {

TGSkey = c.cryptedB.key;

if (TGSkey == d.key) {

if (temp.username == d.username) {

if (temp.time == d.time) {

cout << "Формируется ключ сеанса службы...";

messageE mess = { temp.username, c.service\_name, \_\_TIME\_\_, "127.0.0.1", "1 day", TGSkey, randomKeyGeneration(20) };

messageF mess1 = { mess, TGSkey };

cout << "Производится отправка сообщений E и F..." << endl;

return user.actionThird(mess, mess1);

}

}

}

}

cout << "Временные метки или имя пользователя не совпадают." << endl;

return 0;

}

// Обработка сообщений E и F, отправка сообщений G, H.

bool Client::actionThird(messageE a, messageF b) {

if (check(b.TGSkey, decryptedTGSKey)) {

cout << "Клиентом получен ключ сеанса службы." << endl;

decryptedAPPkey = b.mess.APPkey;

messageG mess = {\_\_TIME\_\_, username, decryptedAPPkey};

messageH mess2 = {a};

cout << "Клиент отправляет сообщения G и H на сервер." << endl;

return example.actionFourth(mess, mess2);

}

cout << "Провалился шаг 3." << endl;

return 0;

}

// Обработка сообщений G, H, отправка сообщения I.

bool APP::actionFourth(messageG b, messageH a) {

if (check(APPkey, a.mess.APPkey)) {

cout << "Сверка ключа службы..." << endl;

TGSkey = a.mess.key;

if (check(TGSkey, b.key)) {

cout << "Сверка ключа сеанса..." << endl;

if (a.mess.username == b.username) {

if (a.mess.time == b.time) {

cout << "Сверка временных меток и имени..." << endl;

messageI final\_mess = {"1234567890", \_\_TIME\_\_, APPkey, 1};

cout << "Служба отправляет клиенту сообщение I." << endl;

return user.actionFinal(final\_mess);

}

}

}

}

cout << "Провалился шаг 4." << endl;

return 0;

}

// Авторизация прошла успешно!

bool Client::actionFinal(messageI fin) {

cout << "Kerberos позволяет клиенту и службе связаться." << endl;

return 1;

}

// Обработка главного меню программы

void mainMenu() {

while (true) {

system("cls");

system("color D");

cout << "Вы вошли в систему. Выберите действия:\n";

SetColor(14, 0);

cout << "1) Показать сообщения на почте\n";

cout << "2) Выйти из системы\n\n(ввод):";

string c; getline(cin, c);

if (c == "1") {

cout << "У вас одно новое сообщение на почте. Желаете проверить? (Y/N)\n\n(ввод):";

getline(cin, c);

if (c == "Y") {

cout << "Сообщение от студента 2 курса РУТ (Информационная безопасность):\nМожно 5-ку по ОИБ?\n";

cin.ignore();

}

}

else if (c == "2") {

cout << "Вы вышли из системы." << endl;

system("pause");

system("cls");

return;

}

system("cls");

}

}

// Обработка входа в учётную запись.

Client::Client(Server& name) {

cout << "Введите имя пользователя: ";

getline(cin, username);

cout << "Введите пароль: ";

getline(cin, password);

if (signIn())

mainMenu();

}

// Точка входа в программу.

int main() {

setlocale(0, "rus");

system("chcp 1251 >> null");

while (true) {

system("color D");

cout << "Выберите действие:\n";

SetColor(14, 0);

cout << "1) Регистрация нового пользователя\n";

cout << "2) Вход в учётную запись\n";

cout << "3) Очистка экрана\n";

cout << "4) Выход из программы\n\n(Ввод) ";

string c; getline(cin, c);

if (c == "1") {

cout << "Задайте имя новому пользователю: ";

string name; getline(cin, name);

cout << "Задайте пароль новому пользователю: ";

string pass; getline(cin, pass);

cout << "Новый пользователь - пользователь/администратор? (0/1): ";

bool decision; cin >> decision;

if (test\_server.signUp(name, pass, decision) == 1)

cout << "Пользователь успешно создан!\n\n";

system("pause");

system("cls");

}

else if (c == "2") {

Client user(test\_server);

}

else if (c == "4") {

cout << "Всего хорошего!" << endl;

break;

}

else if (c == "3") {

system("cls");

}

else {

cout << "Похоже, вы ввели что-то не то. Попробуем ещё раз :)\n\n";

system("pause");

system("cls");

}

cin.ignore();

}

system("pause");

return 0;

}

**5 ИЛЛЮСТРАЦИЯ РЕЗУЛЬТАТА РАБОТЫ ПРОГРАММЫ**

Рассмотрим результат работы программы:

Изображение выглядит как текст

Автоматически созданное описание

Рисунок 22 – Успешное создание учетной записи

Изображение выглядит как текст

Автоматически созданное описание

Рисунок 23 – Успешный вход в систему

Изображение выглядит как текст

Автоматически созданное описание

Рисунок 24 – Неудачное создание учетной записи

Изображение выглядит как текст

Автоматически созданное описание

Рисунок 25 – вход с неверным паролем

Изображение выглядит как текст

Автоматически созданное описание

Рисунок 26 – Вход с неверным именем пользователя

Изображение выглядит как текст

Автоматически созданное описаниеИзображение выглядит как текст

Автоматически созданное описание

Рисунок 27 – Тестирование функций учётной записи

**6 ЗАКЛЮЧЕНИЕ**

В ходе выполнения работы был разработан протокол сетевой аутентификации на основе Kerberos, в функционале была реализована последовательность из 16 шагов выполнения программы, которая реализует взаимодействие между службой, клиентом и сервером, а также выдачу билетов типа TGS/TGT. Равно с этим на этапах проводятся проверки жизни билетов на всех этапах жизненного цикла.

**7 СПИСОК ИСПОЛЬЗОВАННОЙ ЛИТЕРАТУРЫ**

1 Сайт о программировании: различные руководства и учебные материалы, статьи и примеры. [Электронный ресурс] – 2015. – URL: <https://bestprogrammer.ru/izuchenie/kerberos-za-5-minut-znakomstvo-s-setevoj-autentifikatsiej> (дата обращения 16.10.2022)

2 Качественная документация: службы каталогов и LDAP на русском языке, перевод официальной документации OpenLDAP. [Электронный ресурс] – 2011 – URL: <https://pro-ldap.ru/tr/zytrax/tech/kerberos.html> (дата обращения 16.10.2022)

3 Книги на любую тематику: различные новости и учебные материалы, примеры. [Электронный ресурс] – 2015 – URL: <https://litgu.ru/knigi/seti/277088-kerberos-the-definitive-guide.html> (дата обращения 16.10.2022)

4 Дэниел Г. Грэм “Этичный Хакинг: практическое руководство по взлому (Предисловие Хуана Гилберта)”//Наука. Инновации. - Москва: [б. и.], 2022 – 15 глава – перемещение по корпоративной сети Windows