Министерство образования Республики Беларусь

Учреждение образования «Белорусский государственный университет   
информатики и радиоэлектроники»

Факультет компьютерных систем и сетей

Кафедра информатики

Дисциплина: Информационные сети. Основы безопасности

ОТЧЁТ

к лабораторной работе №3

на тему

**«ИДЕНТИФИКАЦИЯ И АУТЕНТИФИКАЦИЯ ПОЛЬЗОВАТЕЛЕЙ. ПРОТОКОЛ KERBEROS»**

Выполнил: студент гр. 253505

Сенько Н. С.

Проверил: ассистент кафедры

информатики

Герчик А. В.

Минск 2025

**СОДЕРЖАНИЕ**

[1 Цель работы 3](#__RefHeading___Toc1348_1888910669)

[2 Ход работы 4](#__RefHeading___Toc1350_1888910669)

[Заключение 5](#__RefHeading___Toc1352_1888910669)

[Приложение А (обязательное](#__RefHeading___Toc3241_1888910669)) и[сходный код программного продукта 6](#__RefHeading___Toc3243_1888910669)

# 1 ЦЕЛЬ РАБОТЫ

Реализовать простейшую реализацию программного протокол Kerberos. В результате данной лабораторной работы будет реализован на классах Kotlin «сервер» аутентификации, выдачи разрешений, клиента и сервиса.

# 2 ХОД РАБОТЫ

Kerberos – в первую очередь протокол аутентификации, но при этом предусматривающий возможность транспортировки информации необходимой для авторизации. Он был разработан был разработан в MIT, как часть научно-исследовательского проекта Афина, предназначенного для создания распределенной образовательной среды. Основная идея протокола заключается в том, что пользователи и сервисы проходят аутентификацию через доверенного посредника – центр распределения ключей (KDC), который генерирует временные ключи и билеты для безопасного взаимодействия между участниками.

Протокол Kerberos состоит из нескольких ключевых компонентов. На рисунке 1 можно наблюдать схему работы протокола Kerberos.

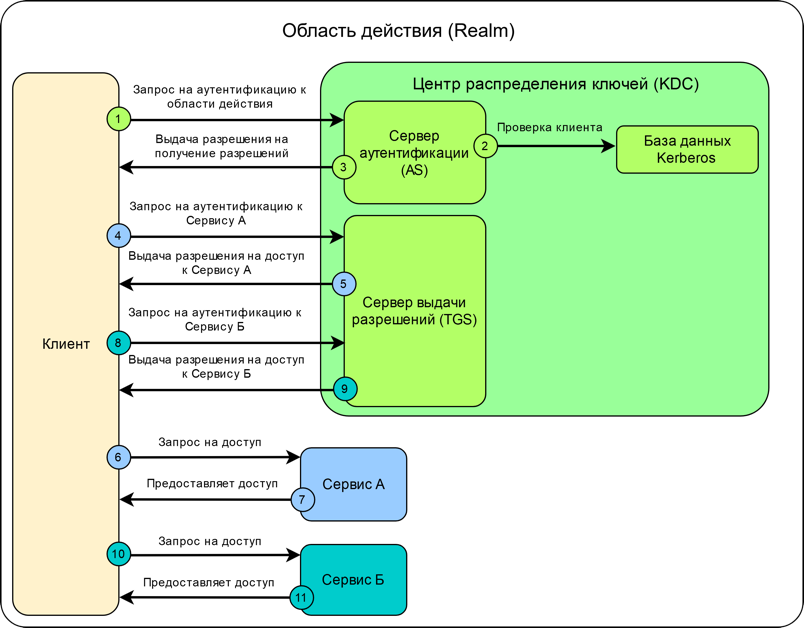


Рисунок 1 – Схема работы протокола Kerberos

В результате был разработан код, который эмитирует работу протокола Kerberos, который на базе ключа клиента, сервера и tgs выдавал разрешение на получение разрешения, а также по полученному разрешению выдавал разрешения на доступ к сервису.

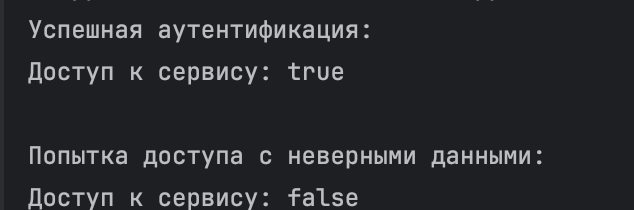


Рисунок 2 – результат работы программы

Результат выполнения программы с успешной и неуспешной аутентификацией представлен на рисунке 2.

# ЗАКЛЮЧЕНИЕ

Таким образом, в ходе выполнения лабораторной работы было успешно исследовано и реализовано взаимодействие по протоколу Kerberos с использованием языка программирования Kotlin. Созданные классы – клиент, сервер аутентификации, сервер выдачи разрешений и конечный сервер – были разработаны с учетом основных этапов протокола Kerberos. Основными достижениями работы является понимание принципов функционирования протокола Kerberos, включая процессы аутентификации, обмена билетами. Таким образом, выполнение лабораторной работы позволило успешно достичь поставленных целей и задач, а также приобрести практические навыки в области обеспечения информационной безопасности и применения протокола Kerberos.

# Приложение А

# (обязательное)

# Исходный код программного продукта

import java.util.\*

import kotlin.random.Random

fun main() {

val clientSecret = "client\_pass"

val tgsSecret = "tgs\_secret"

val serviceSecret = "service\_secret"

val authServer = AuthenticationServer(clientSecret, tgsSecret)

val ticketServer = TicketGrantingServer(tgsSecret, serviceSecret)

val service = ServiceServer(serviceSecret)

val client = Client("client1", clientSecret)

println("Успешная аутентификация:")

val success = client.accessService(authServer, ticketServer, service, "file\_service")

println("Доступ к сервису: $success\n")

println("Попытка доступа с неверными данными:")

val wrongClient = Client("client1", "wrong\_pass")

val fail = wrongClient.accessService(authServer, ticketServer, service, "file\_service")

println("Доступ к сервису: $fail")

}

class AuthenticationServer(

private val clientSecret: String,

private val tgsSecret: String

) {

fun requestTGT(clientId: String): Pair<String, String> {

val sessionKey = generateKey()

val tgtData = "$clientId:${sessionKey}:${System.currentTimeMillis()}"

// 1. генерируем сессионый ключ и шифруем нашим паролей

// 2. генерим тгт данные и шифруем тгс ключом

return Pair(

sessionKey.xor(clientSecret), // Для клиента

tgtData.xor(tgsSecret) // TGT для TGS

)

}

}

class TicketGrantingServer(

private val tgsSecret: String,

private val serviceSecret: String

) {

fun requestServiceTicket(tgt: String, authenticator: String, serviceId: String): Pair<String, String>? {

val decryptedTGT = tgt.unxor(tgsSecret) // 5. получаем тгт данные с помощью тгт секрета

val tgtParts = decryptedTGT.split(":")

if (tgtParts.size != 3) return null

val (clientId, sessionKey, timestamp) = tgtParts

if (System.currentTimeMillis() - timestamp.toLong() > 5000) return null

val decryptedAuth = authenticator.unxor(sessionKey) // 6. провеяем что тут клиуент

if (decryptedAuth != clientId) return null

val serviceSessionKey = generateKey() // 7. генериурем ключ доступа к сервисв

val serviceTicket = "$clientId:$serviceSessionKey:${System.currentTimeMillis()}".xor(serviceSecret) // 8. генерим тикет доступа и шифруем серверным сикретом

return Pair(

serviceSessionKey.xor(sessionKey), // Для клиента

serviceTicket // Для сервиса

)

}

}

class ServiceServer(private val serviceSecret: String) {

fun validateTicket(serviceTicket: String, authenticator: String): Boolean {

val decryptedTicket = serviceTicket.unxor(serviceSecret)

val ticketParts = decryptedTicket.split(":")

if (ticketParts.size != 3) return false

val (clientId, sessionKey, timestamp) = ticketParts

if (System.currentTimeMillis() - timestamp.toLong() > 5000) return false

val decryptedAuth = authenticator.unxor(sessionKey)

return decryptedAuth == "$clientId:${System.currentTimeMillis()}"

}

}

class Client(val id: String, private val secret: String) {

fun accessService(

auth: AuthenticationServer,

tgs: TicketGrantingServer,

service: ServiceServer,

serviceId: String

): Boolean {

val (encSessionKey, tgt) = auth.requestTGT(id)

val sessionKey = encSessionKey.unxor(secret) // 3. получаем сессионый ключ и расшифровываем нашим паролей

val authenticator = id.xor(sessionKey) // 4. генерим аунтентификатор на базе сессионого ключа

val serviceTicketData = tgs.requestServiceTicket(tgt, authenticator, serviceId) ?: return false

val (encServiceSessionKey, serviceTicket) = serviceTicketData // 9. Из ответы достаем ключ и тикет доступа к сервису

val serviceSessionKey = encServiceSessionKey.unxor(sessionKey) // 10. получаем ключ расшифроывая нашим сессионым ключам

val serviceAuth = "$id:${System.currentTimeMillis()}".xor(serviceSessionKey) // 11. генерим ключ дл] сервера и отпрвляетм на проверку

return service.validateTicket(serviceTicket, serviceAuth)

}

}