Министерство образования Республики Беларусь

Учреждение образования

БЕЛОРУССКИЙ ГОСУДАРСТВЕННЫЙ УНИВЕРСИТЕТ ИНФОРМАТИКИ И РАДИОЭЛЕКТРОНИКИ

Факультет компьютерных систем и сетей

Кафедра информатики

Дисциплина: Информационные сети. Основы безопасности

**ОТЧЕТ**

к лабораторной работе № 2

на тему «Идентификация и аутентификация пользователей.  
Протокол Kerberos.»

Выполнил:

студент гр. 153503

Киселева Е.А.

Проверил:

Лещенко Е.А.

Минск 2024

СОДЕРЖАНИЕ

[1 Постановка задачи 3](file:///C:\Users\Acer\Downloads\Report1.docx#_Toc157353562)

[2 Краткие теоретические сведения 4](file:///C:\Users\Acer\Downloads\Report1.docx#_Toc157353563)

[2.1 Протокол Kerberos 4](file:///C:\Users\Acer\Downloads\Report1.docx#_Toc157353564)

[2.2 Алгоритм DES 6](file:///C:\Users\Acer\Downloads\Report1.docx#_Toc157353565)

[3 Результаты выполнения лабораторной работы 12](file:///C:\Users\Acer\Downloads\Report1.docx#_Toc157353566)

[3.1 Шифрование (дешифрование) при помощи алгоритма DES 12](file:///C:\Users\Acer\Downloads\Report1.docx#_Toc157353567)

[3.2 Реализация протокола распределения ключей Kerberos 12](file:///C:\Users\Acer\Downloads\Report1.docx#_Toc157353568)

[Выводы 13](file:///C:\Users\Acer\Downloads\Report1.docx#_Toc157353569)

Приложение А [(обязательное) Листинг исходного кода 14](file:///C:\Users\Acer\Downloads\Report1.docx#_Toc157353570)

## 1 ПОСТАНОВКА ЗАДАЧИ

Целью выполнения лабораторной работы является изучение теоретических сведений о протоколе Kerberos, а также создание приложения, реализующего протокол распределения ключей Kerberos, включая процедуру, реализующую Алгоритм DES. В интерфейсе приложения должны быть наглядно представлены исходные данные протокола (модули, ключи, секретные данные и т.п.), данные, передаваемые по сети каждой из сторон, проверки, выполняемые каждым из участников.

## 2 КРАТКИЕ ТЕОРЕТИЧЕСКИЕ СВЕДЕНИЯ

## 2.1 Протокол Kerberos

Протокол Kerberos является одной из реализаций протокола аутентификации с использованием третьей стороны, призванной уменьшить количество сообщений, которыми обмениваются стороны.

Протокол Kerberos, достаточно гибкий и имеющий возможности тонкой настройки под конкретные применения, существует в нескольких версиях. Мы рассмотрим упрощенный механизм аутентификации, реализованный с помощью протокола Kerberos версии 5 (рис. 1).



Рисунок 1 – Схема протокола Kerberos

Прежде всего стоит сказать, что при использовании Kerberos нельзя напрямую получить доступ к какому-либо целевому серверу. Чтобы запустить собственно процедуру аутентификации, необходимо обратиться к специальному серверу аутентификации с запросом, содержащим логин пользователя. Если сервер не находит автора запроса в своей базе данных, запрос отклоняется. В противном случае сервер аутентификации работает по следующему рабочему процессу:

Пусть клиент C собирается начать взаимодействие с сервером SS (англ. *Service* *Server* – *сервер*, предоставляющий сетевые сервисы). В несколько упрощенном виде, протокол предполагает следующие шаги:

1 Клиент C посылает серверу аутентификации AS свой идентификатор c (идентификатор передается открытым текстом).

2 На этом шаге сервер аутентификации AS, проверив, что клиент C имеется в его базе, возвращает ему билет для доступа к серверу выдачи разрешений и ключ для взаимодействия с сервером выдачи разрешений. Вся посылка зашифрована на ключе клиента C. Таким образом, даже если на первом шаге взаимодействия идентификатор с послал не клиент С, а нарушитель X, то полученную от AS посылку X расшифровать не сможет.

Получить доступ к содержимому билета TGT не может не только нарушитель, но и клиент C, т.к. билет зашифрован на ключе, который распределили между собой сервер аутентификации и сервер выдачи разрешений.

3 Клиент C на этот раз обращается к серверу выдачи разрешений ТGS. Он пересылает полученный от AS билет, зашифрованный на ключе KAS\_TGS, и аутентификационный блок, содержащий идентификатор c и метку времени, показывающую, когда была сформирована посылка. Сервер выдачи разрешений расшифровывает билет TGT и получает из него информацию о том, кому был выдан билет, когда и на какой срок, ключ шифрования, сгенерированный сервером AS для взаимодействия между клиентом C и сервером *TGS*. С помощью этого ключа расшифровывается аутентификационный блок. Если метка в блоке совпадает с меткой в билете, это доказывает, что посылку сгенерировал на самом деле С (ведь только он знал ключ KC\_TGS и мог правильно зашифровать свой идентификатор). Далее делается проверка времени действия билета и времени отправления посылки 3). Если проверка проходит и действующая в системе политика позволяет клиенту С обращаться к клиенту SS, тогда выполняется шаг 4).

4 Сейчас сервер выдачи разрешений *TGS* посылает клиенту C ключ шифрования и билет, необходимые для доступа к серверу SS. Структура билета такая же, как на шаге 2): идентификатор того, кому выдали билет; идентификатор того, для кого выдали билет; отметка времени; *период действия*; ключ шифрования.

5 Клиент C посылает билет, полученный от сервера выдачи разрешений, и свой аутентификационный блок серверу SS, с которым хочет установить сеанс защищенного взаимодействия. Предполагается, что SS уже зарегистрировался в системе и распределил с сервером *TGS* ключ шифрования KTGS\_SS. Имея этот ключ, он может расшифровать билет, получить ключ шифрования KC\_SS и проверить подлинность *отправителя сообщения*.

6 Смысл последнего шага заключается в том, что теперь уже SS должен доказать C свою подлинность. Он может сделать это, показав, что правильно расшифровал предыдущее сообщение. Вот поэтому, SS берет отметку времени из аутентификационного блока C, изменяет ее заранее определенным образом (увеличивает на 1), шифрует на ключе KC\_SS и возвращает C.

Если все шаги выполнены правильно и все проверки прошли успешно, то стороны взаимодействия C и SS, во-первых, удостоверились в подлинности друг друга, а во-вторых, получили *ключ* шифрования для защиты сеанса связи - *ключ* KC\_SS.

Нужно отметить, что в процессе сеанса работы клиент проходит шаги 1) и 2) только один раз. Когда нужно получить билет на *доступ* к другому серверу (назовем его SS1), клиент С обращается к серверу выдачи разрешений *TGS* с уже имеющимся у него билетом, т.е. протокол выполняется начиная с шага 3).

В алгоритме Kerberos могут применяться различные алгоритмы блочного симметричного шифрования. Для целей настоящей работы будем использовать алгоритм DES.

## 2.2 Алгоритм DES

Одной из наиболее известных криптографических систем с закрытым ключом является DES – Data Encryption Standard. Эта система первой получила статус государственного стандарта в области шифрования данных. Она разработана специалистами фирмы IBM и вступила в действие в США 1977 году. Алгоритм DES по-прежнему широко применяется и заслуживает внимания при изучении блочных шифров с закрытым ключом.

Стандарт DES построен на комбинированном использовании перестановки, замены и гаммирования. Шифруемые данные должны быть представлены в двоичном виде.

DES является классической сетью Фейстеля с двумя ветвями. Данные шифруются 64-битными блоками, используя 56-битный ключ. Алгоритм преобразует за несколько раундов 64-битный вход в 64-битный выход. Длина ключа равна 56 битам. Процесс шифрования состоит из четырех этапов. На первом из них выполняется начальная перестановка (IP) 64-битного исходного текста (забеливание), во время которой биты переупорядочиваются в соответствии со стандартной таблицей. Следующий этап состоит из 16 раундов одной и той же функции, которая использует операции сдвига и подстановки. На третьем этапе левая и правая половины выхода последней (16-й) итерации меняются местами. Наконец, на четвертом этапе выполняется перестановка IP-1 результата, полученного на третьем этапе. Перестановка IP-1 инверсна начальной перестановке.

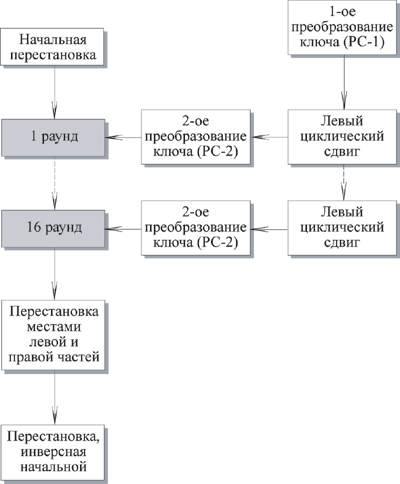


Рисунок 2 – Общая схема DES

Начальная перестановка и ее инверсия определяются стандартной таблицей. Если М – это произвольные 64 бита, то X = IP (M)-переставленные 64 бита. Если применить обратную функцию перестановки Y = IP-1 (X) = IP-1 (IP(M)), то получится первоначальная последовательность бит.

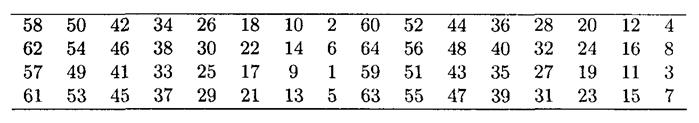


Рисунок 3 – DES. Начальная перестановка

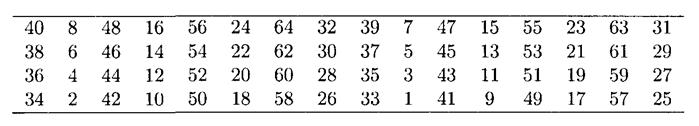


Рисунок 4 – DES. Заключительная перестановка

Теперь рассмотрим последовательность преобразований, используемую в каждом *раунде*.

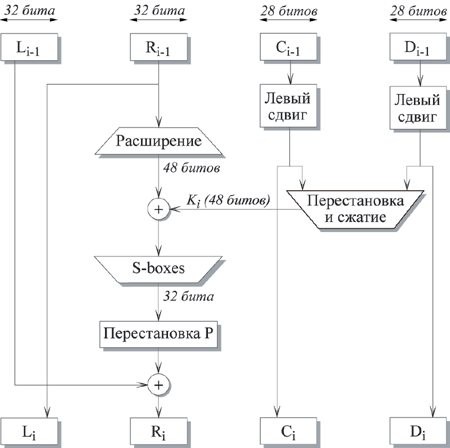


Рисунок 5 – I-ый раунд DES

64-битный входной блок проходит через 16 *раундов*, при этом на каждой итерации получается промежуточное 64-битное значение. Левая и правая части каждого промежуточного значения трактуются как отдельные 32-битные значения, обозначенные *L* и *R*. Каждую итерацию можно описать следующим образом:

Li = Ri-1

Ri = Li-1  F(Ri-1, Ki)

Где обозначает операцию XOR.

Таким образом, выход левой половины *Li* равен входу правой половины *Ri-1*. Выход правой половины *Ri*является результатом применения операции XOR к *Li-1* и функции *F*, зависящей от *Ri-1* и *Ki*.

Рассмотрим функцию *F* более подробно.

*Ri*, которое подается на вход функции *F*, имеет длину 32 бита. Вначале Ri расширяется до 48 бит, используя таблицу, которая определяет перестановку плюс расширение на 16 бит. Расширение происходит следующим образом. 32 бита разбиваются на группы по 4 бита и затем расширяются до 6 бит, присоединяя крайние биты из двух соседних групп. Например, если часть входного сообщения

. . . efgh ijkl mnop . . .

то в результате расширения получается сообщение

. . . defghi hijklm lmnopq . . .

После этого для полученного 48-битного значения выполняется операция XOR с 48-битным *подключом Ki*. Затем полученное 48-битное значение подается на вход функции подстановки, результатом которой является 32-битное значение.

Подстановка состоит из восьми *S-boxes,* каждый из которых на входе получает 6 бит, а на выходе создает 4 бита. Эти преобразования определяются специальными таблицами. Первый и последний биты входного значения *S-box* определяют номер строки в таблице, средние 4 бита определяют номер столбца. Пересечение строки и столбца определяет 4-битный выход. Например, если входом является 011011, то номер строки равен 01 (строка 1) и номер столбца равен 1101 (столбец 13). Значение в строке 1 и столбце 13 равно 5, т.е. выходом является 0101.

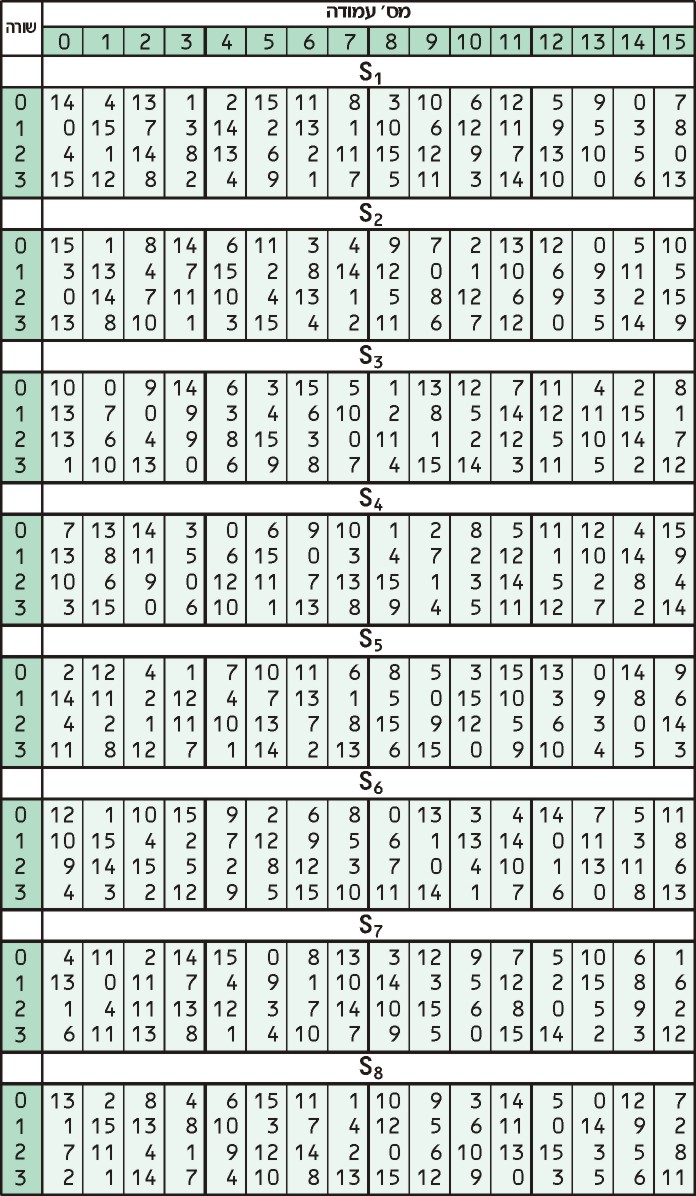


Рисунок 6 – S-boxes

Далее полученное 32-битное значение обрабатывается с помощью перестановки *Р*, целью которой является максимальное переупорядочивание бит, чтобы в следующем *раунде* шифрования с большой вероятностью каждый бит обрабатывался другим *S-box*.

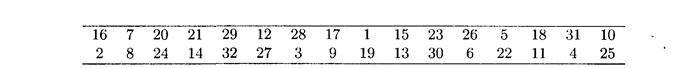


Рисунок 7 – Перестановка с помощью Р-блоков

Ключ для отдельного *раунда Ki*состоит из 48 бит. Ключи *Ki* получаются по следующему алгоритму. Для 56-битного ключа, используемого на входе алгоритма (если используется 64-битный ключ, то, как видно из рис. 5 убираются биты 64, 56, 48, 40, 32, 16, 8) , вначале выполняется перестановка в соответствии с таблицей Permuted Choice 1 (РС-1).

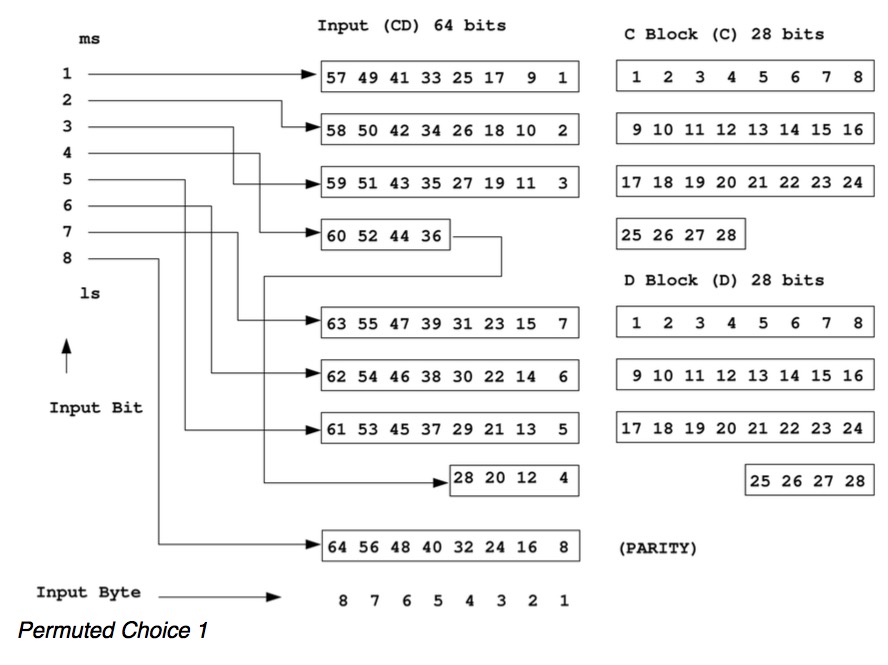


Рисунок 8 – Схема Permuted Choice

Полученный 56-битный ключ разделяется на две 28-битные части, обозначаемые как C0 и D0 соответственно. На каждом *раунде Ci* и *Di*независимо циклически сдвигаются влево на 1 или 2 бита, в зависимости от номера *цикла*.

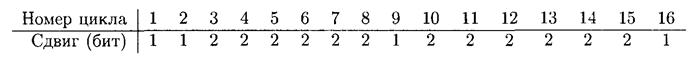


Рисунок 9 - Сдвиг ключа в зависимости от номера цикла

Полученные значения являются входом следующего *раунда*. Они также представляют собой вход в Permuted Choice 2 (РС-2), который создает 48-битное выходное значение, являющееся входом функции *F*(*Ri-1*, *Ki*).

Процесс дешифрования аналогичен процессу шифрования. На входе алгоритма используется зашифрованный текст, но ключи *Ki* используются в обратной последовательности. *K16* используется на первом *раунде*, *K1* используется на последнем *раунде*.

## 3 РЕЗУЛЬТАТЫ ВЫПОЛНЕНИЯ ЛАБОРАТОРНОЙ             РАБОТЫ

В ходе выполнения лабораторной работы было создано приложение, реализующее протокол распределения ключей Kerberos, включая процедуру, реализующую алгоритм DES.

## 3.1 Шифрование (дешифрование) при помощи алгоритма DES

При запуске программы пользователю предоставляется информация о том, как выглядит зашифрованный текст и как выглядит этот же расшифрованный текст. Текст, который будет шифровать, а после дешифровать программа задан внутри ее кода. Результат работы программы предоставлен на рисунке 3.1.

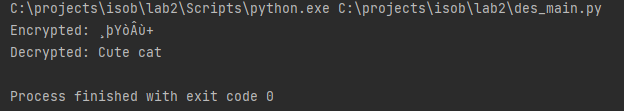


Рисунок 3.1 – Результат работы программы, реализующей алгоритм DES

## 3.2 Реализация протокола распределения ключей Kerberos

При запуске программы пользователю предоставляется информация о том, какие id и ключи имели сервера, к которым подключался клиент, а также id и ключ самого клиента. Результат работы программы представлен на рисунке 3.2.

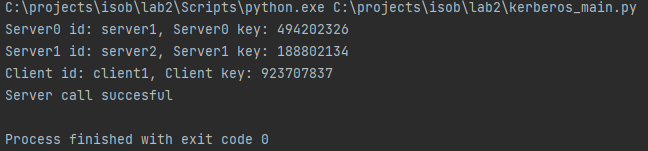


Рисунок 3.2 – Результат работы программы, реализующей протокол Kerberos

## ВЫВОДЫ

В ходе выполнения лабораторной работы было создано приложение, реализующее протокол распределения ключей Kerberos, включая процедуру, реализующую Алгоритм DES. В интерфейсе приложения были наглядно представлены исходные данные протокола (модули, ключи, секретные данные и т.п.), данные, передаваемые по сети каждой из сторон, проверки, выполняемые каждым из участников.

## ПРИЛОЖЕНИЕ А

## (обязательное)

## Листинг исходного кода

Листинг 1 – Программный код алгоритма DES

# Начальная матрица перестановки для данных

PI = [58, 50, 42, 34, 26, 18, 10, 2,

60, 52, 44, 36, 28, 20, 12, 4,

62, 54, 46, 38, 30, 22, 14, 6,

64, 56, 48, 40, 32, 24, 16, 8,

57, 49, 41, 33, 25, 17, 9, 1,

59, 51, 43, 35, 27, 19, 11, 3,

61, 53, 45, 37, 29, 21, 13, 5,

63, 55, 47, 39, 31, 23, 15, 7]

# Начальная перестановка ключа

CP\_1 = [57, 49, 41, 33, 25, 17, 9,

1, 58, 50, 42, 34, 26, 18,

10, 2, 59, 51, 43, 35, 27,

19, 11, 3, 60, 52, 44, 36,

63, 55, 47, 39, 31, 23, 15,

7, 62, 54, 46, 38, 30, 22,

14, 6, 61, 53, 45, 37, 29,

21, 13, 5, 28, 20, 12, 4]

# Перестановка после сдвига ключа для получения Ki+1

CP\_2 = [14, 17, 11, 24, 1, 5, 3, 28,

15, 6, 21, 10, 23, 19, 12, 4,

26, 8, 16, 7, 27, 20, 13, 2,

41, 52, 31, 37, 47, 55, 30, 40,

51, 45, 33, 48, 44, 49, 39, 56,

34, 53, 46, 42, 50, 36, 29, 32]

# Матрица расширения для получения 48-битовой матрицы данных для применения xor с Ki

E = [32, 1, 2, 3, 4, 5,

4, 5, 6, 7, 8, 9,

8, 9, 10, 11, 12, 13,

12, 13, 14, 15, 16, 17,

16, 17, 18, 19, 20, 21,

20, 21, 22, 23, 24, 25,

24, 25, 26, 27, 28, 29,

28, 29, 30, 31, 32, 1]

# SBOX

S\_BOX = [

[[14, 4, 13, 1, 2, 15, 11, 8, 3, 10, 6, 12, 5, 9, 0, 7],

[0, 15, 7, 4, 14, 2, 13, 1, 10, 6, 12, 11, 9, 5, 3, 8],

[4, 1, 14, 8, 13, 6, 2, 11, 15, 12, 9, 7, 3, 10, 5, 0],

[15, 12, 8, 2, 4, 9, 1, 7, 5, 11, 3, 14, 10, 0, 6, 13]],

[[15, 1, 8, 14, 6, 11, 3, 4, 9, 7, 2, 13, 12, 0, 5, 10],

[3, 13, 4, 7, 15, 2, 8, 14, 12, 0, 1, 10, 6, 9, 11, 5],

[0, 14, 7, 11, 10, 4, 13, 1, 5, 8, 12, 6, 9, 3, 2, 15],

[13, 8, 10, 1, 3, 15, 4, 2, 11, 6, 7, 12, 0, 5, 14, 9]],

[[10, 0, 9, 14, 6, 3, 15, 5, 1, 13, 12, 7, 11, 4, 2, 8],

[13, 7, 0, 9, 3, 4, 6, 10, 2, 8, 5, 14, 12, 11, 15, 1],

[13, 6, 4, 9, 8, 15, 3, 0, 11, 1, 2, 12, 5, 10, 14, 7],

[1, 10, 13, 0, 6, 9, 8, 7, 4, 15, 14, 3, 11, 5, 2, 12]],

[[7, 13, 14, 3, 0, 6, 9, 10, 1, 2, 8, 5, 11, 12, 4, 15],

[13, 8, 11, 5, 6, 15, 0, 3, 4, 7, 2, 12, 1, 10, 14, 9],

[10, 6, 9, 0, 12, 11, 7, 13, 15, 1, 3, 14, 5, 2, 8, 4],

[3, 15, 0, 6, 10, 1, 13, 8, 9, 4, 5, 11, 12, 7, 2, 14]],

[[2, 12, 4, 1, 7, 10, 11, 6, 8, 5, 3, 15, 13, 0, 14, 9],

[14, 11, 2, 12, 4, 7, 13, 1, 5, 0, 15, 10, 3, 9, 8, 6],

[4, 2, 1, 11, 10, 13, 7, 8, 15, 9, 12, 5, 6, 3, 0, 14],

[11, 8, 12, 7, 1, 14, 2, 13, 6, 15, 0, 9, 10, 4, 5, 3]],

[[12, 1, 10, 15, 9, 2, 6, 8, 0, 13, 3, 4, 14, 7, 5, 11],

[10, 15, 4, 2, 7, 12, 9, 5, 6, 1, 13, 14, 0, 11, 3, 8],

[9, 14, 15, 5, 2, 8, 12, 3, 7, 0, 4, 10, 1, 13, 11, 6],

[4, 3, 2, 12, 9, 5, 15, 10, 11, 14, 1, 7, 6, 0, 8, 13]],

[[4, 11, 2, 14, 15, 0, 8, 13, 3, 12, 9, 7, 5, 10, 6, 1],

[13, 0, 11, 7, 4, 9, 1, 10, 14, 3, 5, 12, 2, 15, 8, 6],

[1, 4, 11, 13, 12, 3, 7, 14, 10, 15, 6, 8, 0, 5, 9, 2],

[6, 11, 13, 8, 1, 4, 10, 7, 9, 5, 0, 15, 14, 2, 3, 12]],

[[13, 2, 8, 4, 6, 15, 11, 1, 10, 9, 3, 14, 5, 0, 12, 7],

[1, 15, 13, 8, 10, 3, 7, 4, 12, 5, 6, 11, 0, 14, 9, 2],

[7, 11, 4, 1, 9, 12, 14, 2, 0, 6, 10, 13, 15, 3, 5, 8],

[2, 1, 14, 7, 4, 10, 8, 13, 15, 12, 9, 0, 3, 5, 6, 11]],

]

# Перестановка после каждой замены S-блока для каждого раунда

P = [16, 7, 20, 21, 29, 12, 28, 17,

1, 15, 23, 26, 5, 18, 31, 10,

2, 8, 24, 14, 32, 27, 3, 9,

19, 13, 30, 6, 22, 11, 4, 25]

# Конечная перестановка для данных после 16 раундов

PI\_1 = [40, 8, 48, 16, 56, 24, 64, 32,

39, 7, 47, 15, 55, 23, 63, 31,

38, 6, 46, 14, 54, 22, 62, 30,

37, 5, 45, 13, 53, 21, 61, 29,

36, 4, 44, 12, 52, 20, 60, 28,

35, 3, 43, 11, 51, 19, 59, 27,

34, 2, 42, 10, 50, 18, 58, 26,

33, 1, 41, 9, 49, 17, 57, 25]

# Матрица, определяющая сдвиг для каждого раунда ключей

SHIFT = [1, 1, 2, 2, 2, 2, 2, 2, 1, 2, 2, 2, 2, 2, 2, 1]

def string\_to\_bit\_array(text):

return [int(bit) for char in text for bit in '{:08b}'.format(ord(char))]

def bit\_array\_to\_string(array):

return ''.join(chr(int(''.join(map(str, array[i:i+8])), 2)) for i in range(0, len(array), 8))

def binvalue(val, bitsize): # Возвращает двоичное значение как строку заданного размера

if isinstance(val, int):

binval = bin(val)[2:]

else:

binval = bin(ord(val))[2:]

binval = binval.zfill(bitsize) # Заполнение нулями до нужной длины

if len(binval) > bitsize:

raise ValueError("binary value larger than the expected size")

return binval

def nsplit(s, n): # Разбиение списка на подсписки размером "n"

return [s[i:i + n] for i in range(0, len(s), n)]

ENCRYPT = 1

DECRYPT = 0

class Des():

def \_\_init\_\_(self):

self.password = None

self.text = None

self.keys = list()

def run(self, key, text, action=ENCRYPT, padding=False):

key = self.validate\_key(key)

self.password = key[:8]

self.text = text

if padding and action == ENCRYPT:

self.addPadding()

elif len(self.text) % 8 != 0: # Если не указана дополнительная информация о размере, размер данных должен быть кратным 8 байтам

raise Exception("Data size should be multiple of 8")

self.generatekeys() # Генерация всех ключей

text\_blocks = nsplit(self.text, 8) # Разбиение текста на блоки по 8 байт, то есть 64 бита

result = []

for block in text\_blocks: # Цикл по всем блокам данных

block = string\_to\_bit\_array(block)

block = self.permut(block, PI) # Применение начальной перестановки

g, d = nsplit(block, 32)

tmp = None

for i in range(16): # 16 раундов

d\_e = self.expand(d, E) # Расширение d до размера Ki (48 бит)

if action == ENCRYPT:

tmp = self.xor(self.keys[i], d\_e) # Если шифрование, использовать Ki

else:

tmp = self.xor(self.keys[15 - i], d\_e) # Если дешифрование, начать с последнего ключа

tmp = self.substitute(tmp) # Применение S-блоков

tmp = self.permut(tmp, P)

tmp = self.xor(g, tmp)

g = d

d = tmp

result += self.permut(d + g, PI\_1) # Последняя перестановка и добавление результата

final\_res = bit\_array\_to\_string(result)

if padding and action == DECRYPT:

return self.removePadding(final\_res) # Удаление дополнения, если выполняется дешифрование и дополнение включено

else:

return final\_res # Возврат конечной строки данных

def substitute(self, d\_e): # Цикл по всем подспискам

result = []

for i, block in enumerate(nsplit(d\_e, 6)):

row = int(str(block[0]) + str(block[5]), 2) # Получение строки с первым и последним битом

column = int(''.join(map(str, block[1:5])), 2) # Столбец - 2, 3, 4, 5 биты

val = S\_BOX[i][row][column] # Получение значения в соответствующем S-блоке (i)

result.extend(map(int, binvalue(val, 4))) # Добавление в результирующий список

return result

def permut(self, block, table): # Перестановка блока по заданной таблице (универсальный метод)

return [block[x - 1] for x in table]

def expand(self, block, table): # То же, что и permut, но для ясности переименован

return [block[x - 1] for x in table]

def xor(self, t1, t2): # Применение XOR и возврат результата

return [x ^ y for x, y in zip(t1, t2)]

def validate\_key(self, key):

if len(key) < 8:

raise ValueError("Key should be 8 bytes long")

elif len(key) > 8:

return key[:8]

else:

return key

def generatekeys(self): # Генерация всех ключей

self.keys = []

key = string\_to\_bit\_array(self.password)

key = self.permut(key, CP\_1) # Начальная перестановка ключа

g, d = nsplit(key, 28) # Разделение на (g->LEFT), (d->RIGHT)

for i in range(16): # 16 раундов

g, d = self.shift(g, d, SHIFT[i]) # Применение сдвига, соответствующего раунду (не всегда 1)

tmp = g + d # Объединение

self.keys.append(self.permut(tmp, CP\_2)) # Получение Ki

def shift(self, g, d, n): # Сдвиг списка на заданное значение

return g[n:] + g[:n], d[n:] + d[:n]

def addPadding(self): # Добавление дополнения к данным в соответствии с PKCS5

pad\_len = 8 - (len(self.text) % 8)

self.text += pad\_len \* chr(pad\_len)

def removePadding(self, data): # Удаление дополнения (предполагается, что оно есть)

pad\_len = ord(data[-1])

return data[:-pad\_len]

def encrypt(self, key, text, padding=False):

return self.run(key, text, ENCRYPT, padding)

def decrypt(self, key, text, padding=False):

return self.run(key, text, DECRYPT, padding)

if \_\_name\_\_ == '\_\_main\_\_':

key = "secret\_k"

text = "Cute cat"

d = Des()

r = d.encrypt(key, text)

r2 = d.decrypt(key, r)

print("Encrypted:", r)

print("Decrypted:", r2)

Листинг 2 – Программный код протокола распределения ключей Kerberos

import random

import time

from des\_main import Des

from ast import literal\_eval as make\_tuple

# Получение текущего времени в миллисекундах

current\_milli\_time = lambda: int(round(time.time() \* 1000))

# Преобразование часов в миллисекунды

hours\_to\_milli = lambda hour: hour \* 3600 \* 10000

class DesEncryptor:

def encrypt\_data(self, data, key):

encrypted = Des().encrypt(key=str(key), text=str(data), padding=True)

return encrypted

def decrypt\_data(self, data, key):

decrypted = Des().decrypt(key=str(key), text=str(data), padding=True)

decrypted = make\_tuple(decrypted)

return decrypted

class KeyGenerator:

@staticmethod

def generate\_key():

return random.randint(100000000, 999999999)

class KeyDistributionCenter:

# Список доступных клиентов и серверов с их ключами

available\_clients = ['client1', 'client2']

clients\_keys = [KeyGenerator.generate\_key(), KeyGenerator.generate\_key()]

available\_servers = ['server1', 'server2']

servers\_keys = [KeyGenerator.generate\_key(), KeyGenerator.generate\_key()]

def \_\_init\_\_(self):

self.des = DesEncryptor()

self.tgs\_id = 1

# Генерация ключа Ticket Granting Server (TGS)

self.key\_tgs = KeyGenerator.generate\_key()

def get\_permission\_ticket(self, client\_id):

if client\_id not in self.available\_clients:

print('Unknown client id')

return

t = current\_milli\_time()

p = hours\_to\_milli(48)

key\_tgs\_c = KeyGenerator.generate\_key()

ticket = self.create\_permission\_ticket(client\_id, self.tgs\_id, t, p, key\_tgs\_c)

encrypted\_ticket = self.des.encrypt\_data(ticket, self.key\_tgs)

bundle = (encrypted\_ticket, key\_tgs\_c)

index = self.available\_clients.index(client\_id)

client\_key = self.clients\_keys[index]

encrypted\_bundle = self.des.encrypt\_data(bundle, client\_key)

return encrypted\_bundle

def get\_server\_ticket(self, permission\_ticket, authority, server\_id):

permission\_ticket = self.des.decrypt\_data(permission\_ticket, self.key\_tgs)

client\_id, \_, t, p, key\_tgs\_c = permission\_ticket

authority = self.des.decrypt\_data(authority, key\_tgs\_c)

auth\_client\_id, auth\_t = authority

if client\_id != auth\_client\_id:

print('Invalid client')

return None

if not (t <= auth\_t <= t + p):

print('Expired')

return None

t = current\_milli\_time()

p = hours\_to\_milli(48)

key\_ss\_c = KeyGenerator.generate\_key()

server\_ticket = self.create\_server\_ticket(client\_id, server\_id, t, p, key\_ss\_c)

index = self.available\_servers.index(server\_id)

server\_key = self.servers\_keys[index]

encrypted\_server\_ticket = self.des.encrypt\_data(server\_ticket, server\_key)

bundle = (encrypted\_server\_ticket, key\_ss\_c)

encrypted\_bundle = self.des.encrypt\_data(bundle, key\_tgs\_c)

return encrypted\_bundle

@staticmethod

def create\_permission\_ticket(client\_id, tgs, t, p, key\_tgs\_c):

return client\_id, tgs, t, p, key\_tgs\_c

@staticmethod

def create\_server\_ticket(client\_id, server\_id, t, p, key\_ss\_c):

return client\_id, server\_id, t, p, key\_ss\_c

class Client:

def \_\_init\_\_(self, client\_id, client\_key, kdc, servers):

self.client\_id = client\_id

self.client\_key = client\_key

self.kdc = kdc

self.servers = servers

self.des = DesEncryptor()

self.permission\_ticket = None

self.key\_tgs\_c = None

def make\_server\_call(self, server\_number):

server = self.servers[server\_number]

if self.permission\_ticket is None or self.key\_tgs\_c is None:

permission\_ticket\_bundle = self.kdc.get\_permission\_ticket(self.client\_id)

if permission\_ticket\_bundle is None:

return

permission\_ticket\_bundle = self.des.decrypt\_data(permission\_ticket\_bundle, self.client\_key)

self.permission\_ticket, self.key\_tgs\_c = permission\_ticket\_bundle[:2]

permission\_ticket = self.permission\_ticket

key\_tgs\_c = self.key\_tgs\_c

bundle = self.request\_tgs(permission\_ticket, key\_tgs\_c, server.server\_id)

if bundle is None:

return

server\_ticket, key\_ss\_c = self.des.decrypt\_data(bundle, key\_tgs\_c)

t = current\_milli\_time()

authority = (self.client\_id, t)

authority\_enctypted = self.des.encrypt\_data(authority, key\_ss\_c)

confirm\_t = server.connect(server\_ticket, authority\_enctypted)

if confirm\_t is None:

return

confirm\_t = self.des.decrypt\_data(confirm\_t, key\_ss\_c)

if confirm\_t != t + 1:

print('Server returns an incorrect time label')

return

print('Server call succesful')

def request\_tgs(self, permission\_ticket, key\_tgs\_c, server\_id):

encrypted\_authority = self.des.encrypt\_data((self.client\_id, current\_milli\_time()), key\_tgs\_c)

bundle = self.kdc.get\_server\_ticket(permission\_ticket, encrypted\_authority, server\_id)

return bundle

class Server:

def \_\_init\_\_(self, server\_id, server\_key):

self.server\_id = server\_id

self.server\_key = server\_key

self.des = DesEncryptor()

def connect(self, server\_ticket, authority):

client\_id, received\_server\_id, t, p, key\_ss\_c = self.des.decrypt\_data(server\_ticket, self.server\_key)

if received\_server\_id != self.server\_id:

print('Wrong server')

return None

auth\_client\_id, auth\_t = self.des.decrypt\_data(authority, key\_ss\_c)

if client\_id != auth\_client\_id:

print('Invalid client')

return None

if not (t <= auth\_t <= t + p):

print('Ticket is expired')

return None

confirm\_t = auth\_t + 1

encrypted\_confirm\_t = self.des.encrypt\_data(confirm\_t, key\_ss\_c)

return encrypted\_confirm\_t

def init():

kdc = KeyDistributionCenter()

servers = [Server(sid, skey) for sid, skey in zip(kdc.available\_servers, kdc.servers\_keys)]

client = Client(kdc.available\_clients[0], kdc.clients\_keys[0], kdc, servers)

for idx, server in enumerate(servers):

print(f'Server{idx} id: {server.server\_id}, Server{idx} key: {server.server\_key}')

print(f'Client id: {client.client\_id}, Client key: {client.client\_key}')

return client

def main():

client = init()

client.make\_server\_call(0)

if \_\_name\_\_ == "\_\_main\_\_":

main()