Министерство образования Республики Беларусь

Учреждение образования

БЕЛОРУССКИЙ ГОСУДАРСТВЕННЫЙ УНИВЕРСИТЕТ ИНФОРМАТИКИ И РАДИОЭЛЕКТРОНИКИ

КАФЕДРА ИНФОРМАТИКИ

Лабораторная работа №2

**Информационные сети. Основы безопасности. Информационные сети. Основы безопасности.**

Тема:   
**«Идентификация и аутентификация пользователей. Протокол Kerberos»**.

Выполнил студент гр. 753503

Муха Д.С.

Проверил

Протько М.И.

Минск, 2020

# 1. Введение

Целью данной лабораторной было реализовать программные средства протокола распределения ключей Kerberos вместе с процедурой, которая реализует Алгоритм DES.

**ЗАДАНИЕ:**

1. Изучить теоретические сведения.
2. Создать приложение, реализующее протокол распределения ключей Kerberos, включая процедуру, реализующую Алгоритм DES.

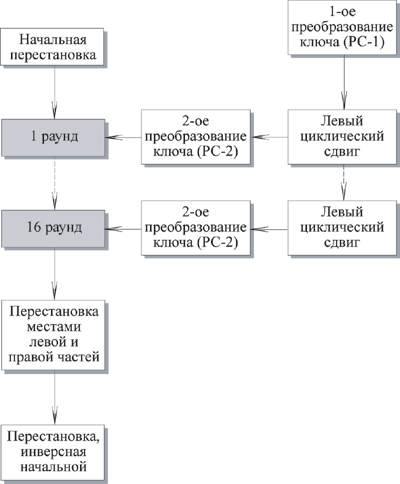
В интерфейсе приложения должны быть наглядно представлены:

* Исходные данные протокола (модули, ключи, секретные данные и т.п.);
* Данные, передаваемые по сети каждой из сторон;
* Проверки, выполняемые каждым из участников.

Процесс взаимодействия между сторонами протокола может быть реализован при помощи буферных переменных. Также необходимо выделить каждый из этапов протоколов для того, чтобы его можно было отделить от остальных.

# 2**. Блок-схемы алгоритмов**

**Блок-схема алгоритма DES:**



**Блок-схема протокола Kerberos:**

# Картинки по запросу "блок-схема Керберос"3. Теоретические сведения

**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:

**Рабочий этап:**

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. **C -> AS: {c}.**

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

1. **AS -> C: {{TGT}KAS\_TGS, KC\_TGS}KC,**

где:

* KC - основной ключ C;
* KC\_TGS - ключ, выдаваемый C для доступа к серверу выдачи разрешений *TGS*;
* {TGT} - *Ticket* Granting *Ticket* - билет на доступ к серверу выдачи разрешений

{TGT} = {c, *tgs*, t1, p1, KC\_TGS}, где *tgs* - идентификатор сервера выдачи разрешений, t1 - отметка времени, p1 - *период действия* билета.

Запись \{ \cdot \} K_{X}здесь и далее означает, что содержимое фигурных скобок зашифровано на ключе KX (Алгоритм шифрования приводится ниже).

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.

1. **C ->** ***TGS*: {TGT}KAS\_TGS, {Aut1} KC\_TGS, {ID}**

где {Aut1} - аутентификационный блок - Aut1 = {с, t2}, t2 - метка времени; ID - идентификатор запрашиваемого сервиса (в частности, это может быть идентификатор сервера SS).

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**).

1. ***TGS* -> C: {{*****TGS*}KTGS\_SS, KC\_SS} KC\_TGS,**

где KC\_SS - ключ для взаимодействия C и SS, {*TGS*} - *Ticket* Granting Service - билет для доступа к SS (обратите внимание, что такой же аббревиатурой в описании протокола обозначается и сервер выдачи разрешений).  
{*TGS*} = {с, ss, t3, p2, KC\_SS}.

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.

1. **C -> SS: {*****TGS*}KTGS\_SS, {Aut2} KC\_SS**

где Aut2 = {c, t4}.

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*.

1. **SS -> C: {t4+1} KC\_SS**

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:

**Алгоритм DES Основные сведения**

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.

**Шифрование**

**Начальная перестановка**

Начальная перестановка и её 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.

**Последовательность преобразований отдельного раунда**

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

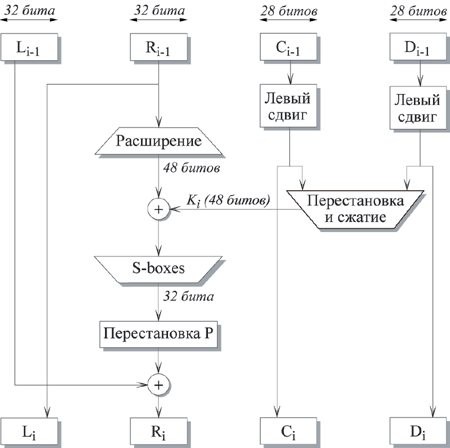


Рисунок 2. I-ый раунд DES

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.

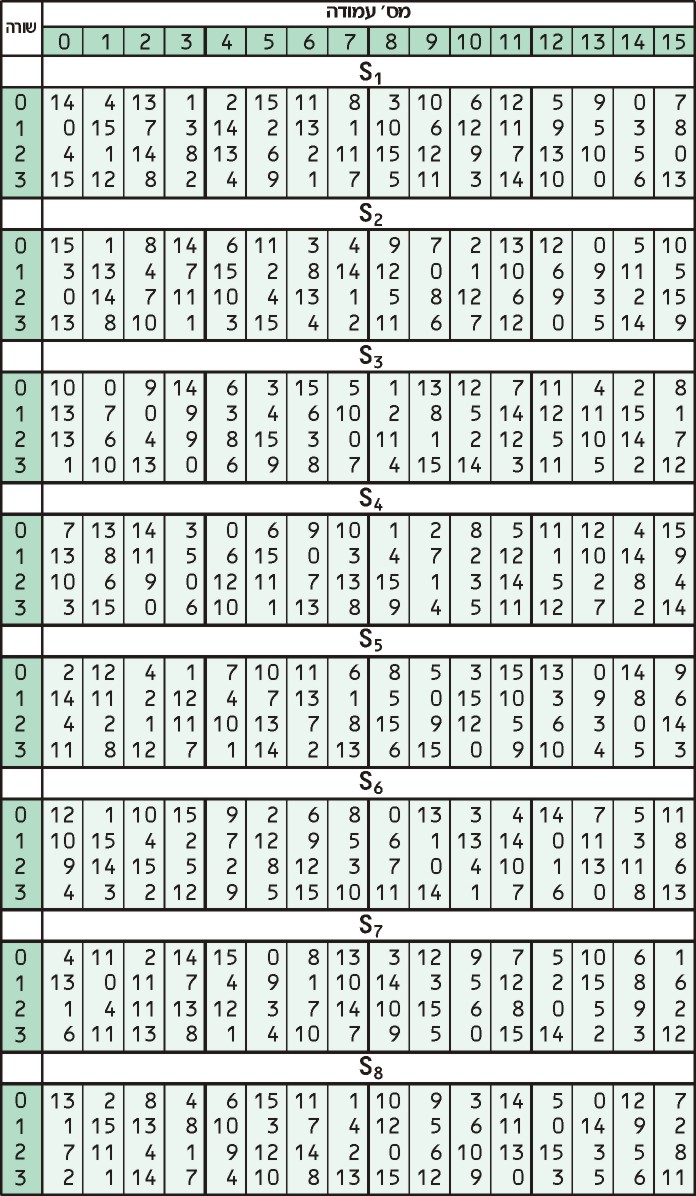


Рисунок 3. S-boxes

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*.

**Создание подключей**

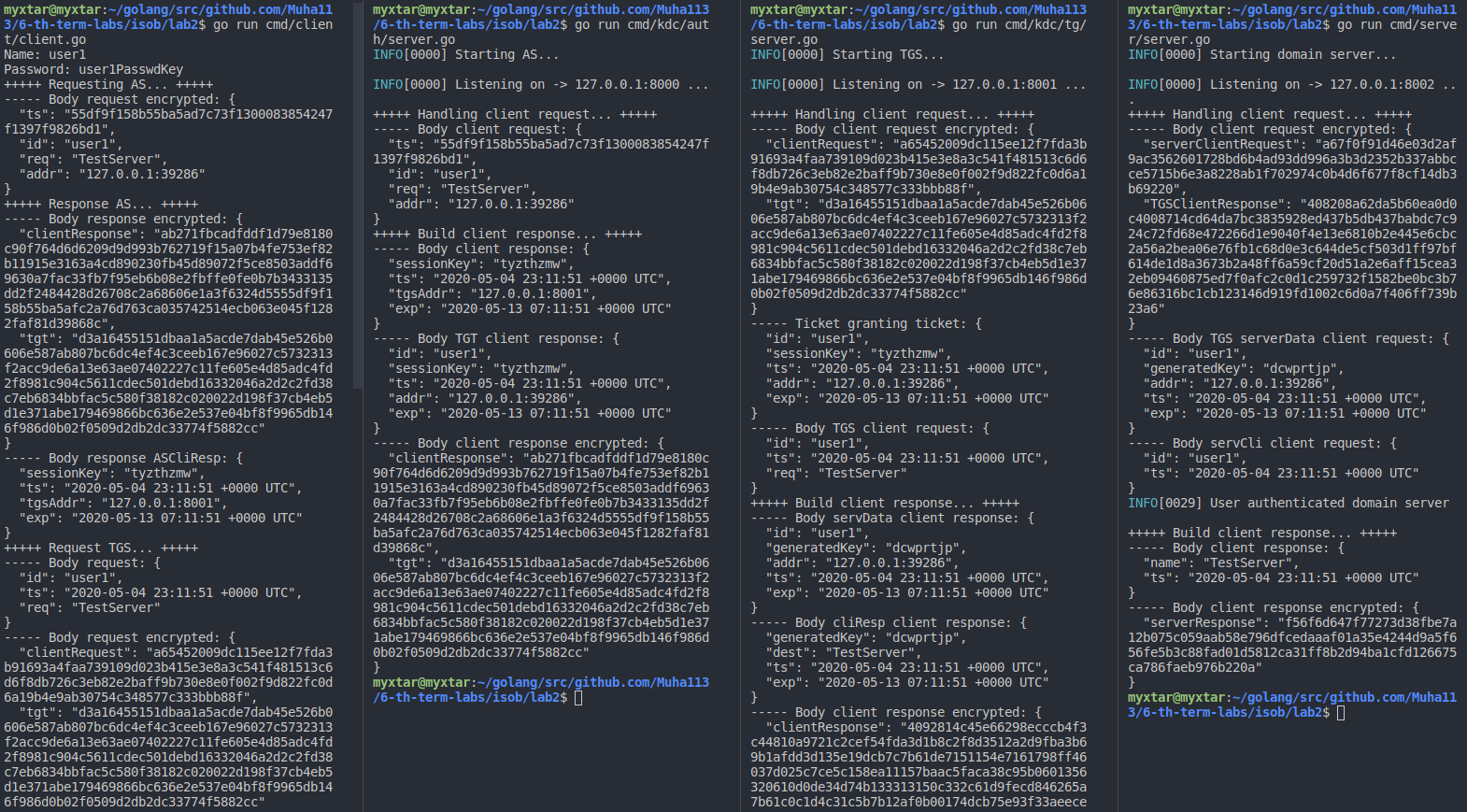
Ключ для отдельного *раунда Ki*состоит из 48 бит. Ключи *Ki* получаются по следующему алгоритму. Для 56-битного ключа, используемого на входе алгоритма, вначале выполняется перестановка в соответствии с таблицей Permuted Choice 1 (РС-1). Полученный 56-битный ключ разделяется на две 28-битные части, обозначаемые как C0 и D0 соответственно. На каждом *раунде Ci* и *Di*независимо циклически сдвигаются влево на 1 или 2 бита, в зависимости от номера *раунда*. 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*).

**Дешифрование**

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*.

# 4. Результаты выполнения программы

**Kerberos**



5. Выводы

DES был национальным стандартом США в 1977—1980 гг., но в настоящее время DES используется (с ключом длины 56 бит) только для устаревших систем, чаще всего используют его более криптоустойчивый вид (3DES, DESX). 3DES является простой эффективной заменой DES, и сейчас он рассмотрен как стандарт. В ближайшее время DES и Triple DES будут заменены алгоритмом AES (Advanced Encryption Standard — Расширенный Стандарт Шифрования).

Kerberos является одним из самых распространенных протоколов аунтефикации. В настоящее время множество ОС поддерживают данный протокол, в число которых входят: Windows 2000 и более поздние версии, которые используют Kerberos как метод аутентификации в домене между участниками, различные UNIX и UNIX подобные ОС (Apple Mac OS X, Red Hat Enterprise Linux 4, FreeBSD, Solaris, AIX, OpenVMS).

# 6. КОД ПРОГРАММЫ

# **Client:**

**package main**

import (

"bufio"

"encoding/json"

"fmt"

"net"

"os"

"time"

"github.com/sirupsen/logrus"

"github.com/Muha113/6-th-term-labs/isob/lab2/pkg/des"

"github.com/Muha113/6-th-term-labs/isob/lab2/pkg/kerberos/common"

)

type Client struct {

ID string

PasswordKey string

SessionKey string

GeneratedKey string

TGTEncrypted string

TimeStamp string

ExpTime string

TGSServerResponseEncrypted string

ASConn net.Conn

TGSConn net.Conn

ServerConn net.Conn

ASAddr string

TGSAddr string

ServerAddr string

}

func main() {

client := &Client{}

reader := bufio.NewReader(os.Stdin)

fmt.Print("Name: ")

name, err := reader.ReadString('\n')

common.HandleError(err, "Client-38: ")

fmt.Print("Password: ")

passwd, err := reader.ReadString('\n')

common.HandleError(err, "Client-41: ")

client.ID = name[:len(name)-1]

client.PasswordKey = common.Hash(passwd[:len(passwd)-1])

client.ASConn, err = net.Dial("tcp", "127.0.0.1:8000")

common.HandleError(err, "Client-45: ")

client.RequestAS()

client.ResponseAS()

client.ASConn.Close()

client.TGSConn, err = net.Dial("tcp", client.TGSAddr)

common.HandleError(err, "Client-50: ")

client.RequestTGS()

client.ResponseTGS()

client.TGSConn.Close()

client.ServerConn, err = net.Dial("tcp", "127.0.0.1:8002")

common.HandleError(err, "Client-55: ")

client.RequestServer()

client.ResponseServer()

client.ServerConn.Close()

}

func (c \*Client) RequestAS() {

fmt.Println("+++++ Requesting AS... +++++")

req := common.ASClientRequest{

TS: time.Now().Format("2006-01-02 15:04:05"),

ID: c.ID,

Req: "TestServer",

Addr: c.ASConn.LocalAddr().String(),

}

*//logrus.Error("KeyByte:", []byte(c.PasswordKey), " KeyStr:", c.PasswordKey, " Len:", len(c.PasswordKey))*

req.TS = des.Encode(req.TS, c.PasswordKey)

msg, err := json.Marshal(req)

common.HandleError(err, "Client-70: ")

fmt.Println("----- Body request encrypted:", string(common.PrettyPrint(msg)))

msg = append(msg, '\n')

c.ASConn.Write(msg)

}

func (c \*Client) ResponseAS() {

fmt.Println("+++++ Response AS... +++++")

buff := make([]byte, 1024)

bytes, err := c.ASConn.Read(buff)

common.HandleError(err, "Client-79: ")

fmt.Println("----- Body response encrypted:", string(common.PrettyPrint(buff[:bytes-1])))

var resp common.ASResponseEncrypted

err = json.Unmarshal(buff[:bytes-1], &resp)

common.HandleError(err, "Client-82: ")

fmt.Println("----- Body response ASCliResp:", string(common.PrettyPrint([]byte(des.Decode(resp.ASClientResponse, c.PasswordKey)))))

var cliResp common.ASClientResponse

err = json.Unmarshal([]byte(des.Decode(resp.ASClientResponse, c.PasswordKey)), &cliResp)

common.HandleError(err, "Client-85: ")

c.TGTEncrypted = resp.TGT

c.SessionKey = cliResp.SessionKey

c.TGSAddr = cliResp.TGSAddr

c.TimeStamp = cliResp.TS

c.ExpTime = cliResp.Exp

}

func (c \*Client) RequestTGS() {

fmt.Println("+++++ Request TGS... +++++")

req := common.TGSClientRequest{

ID: c.ID,

TS: c.TimeStamp,

Req: "TestServer",

}

reqJSON, err := json.Marshal(req)

common.HandleError(err, "Client-100: ")

fmt.Println("----- Body request:", string(common.PrettyPrint(reqJSON)))

*//logrus.Error([]byte(c.SessionKey), " str:", c.SessionKey)*

reqEncrypted := des.Encode(string(reqJSON), c.SessionKey)

msg := common.TGSRequestEncrypted{

TGSClientRequest: reqEncrypted,

TGT: c.TGTEncrypted,

}

msgJSON, err := json.Marshal(msg)

common.HandleError(err, "Client-107: ")

fmt.Println("----- Body request encrypted:", string(common.PrettyPrint(msgJSON)))

msgJSON = append(msgJSON, '\n')

c.TGSConn.Write(msgJSON)

}

func (c \*Client) ResponseTGS() {

fmt.Println("+++++ Response TGS... +++++")

buff := make([]byte, 1024)

bytes, err := c.TGSConn.Read(buff)

common.HandleError(err, "Client-115: ")

fmt.Println("----- Body response encrypted", string(common.PrettyPrint(buff[:bytes-1])))

var resp common.TGSResponseEncrypted

err = json.Unmarshal(buff[:bytes-1], &resp)

common.HandleError(err, "Client-118: ")

fmt.Println("----- Body reponse TGSCliResp:", string(common.PrettyPrint([]byte(des.Decode(resp.TGSClientResponse, c.SessionKey)))))

var cliResp common.TGSClientResponse

err = json.Unmarshal([]byte(des.Decode(resp.TGSClientResponse, c.SessionKey)), &cliResp)

common.HandleError(err, "Client-121: ")

c.GeneratedKey = cliResp.GeneratedKey

c.TGSServerResponseEncrypted = resp.TGSServerDataResponse

}

func (c \*Client) RequestServer() {

fmt.Println("+++++ Request domain server... +++++")

cliReq := common.ServerClientRequest{

ID: c.ID,

TS: c.TimeStamp,

}

cliReqJSON, err := json.Marshal(cliReq)

common.HandleError(err, "Client-132: ")

fmt.Println("----- Body request cliReq:", string(common.PrettyPrint(cliReqJSON)))

cliReqEncrypted := des.Encode(string(cliReqJSON), c.GeneratedKey)

msg := common.ServerRequestEncrypted{

ServerClientRequest: cliReqEncrypted,

TGSServerResponse: c.TGSServerResponseEncrypted,

}

msgJSON, err := json.Marshal(msg)

common.HandleError(err, "Client-139: ")

fmt.Println("----- Body request encrypted:", string(common.PrettyPrint(msgJSON)))

msgJSON = append(msgJSON, '\n')

c.ServerConn.Write(msgJSON)

}

func (c \*Client) ResponseServer() {

fmt.Println("+++++ Response domain server... +++++")

buff := make([]byte, 1024)

bytes, err := c.ServerConn.Read(buff)

common.HandleError(err, "Client-147: ")

fmt.Println("----- Body response encrypted:", string(common.PrettyPrint(buff[:bytes-1])))

var resp common.ServerResponseEncrypted

err = json.Unmarshal(buff[:bytes-1], &resp)

common.HandleError(err, "Client-150: ")

fmt.Println("----- Body response cliResp:", string(common.PrettyPrint([]byte(des.Decode(resp.ServerResponse, c.GeneratedKey)))))

var cliResp common.ServerResponse

err = json.Unmarshal([]byte(des.Decode(resp.ServerResponse, c.GeneratedKey)), &cliResp)

common.HandleError(err, "Client-153: ")

if cliResp.TS != c.TimeStamp {

logrus.Fatal("Unequal ts of cliResp and client")

} else {

logrus.Println("User authenticated domain server")

}

}

# **Authenticate server:**

**package as**

import (

"encoding/json"

"fmt"

"net"

"time"

"github.com/Muha113/6-th-term-labs/isob/lab2/pkg/des"

"github.com/Muha113/6-th-term-labs/isob/lab2/pkg/kerberos/common"

"github.com/sirupsen/logrus"

)

type AS struct {

TGSKey string

ClientsKeysMap map[string]string

ServersKeysMap map[string]string

ExpTime time.Duration

Listener net.Listener

}

*//supposed to user is registered*

func (a \*AS) HandleClientRequest() {

fmt.Println("+++++ Handling client request... +++++")

conn, err := a.Listener.Accept()

common.HandleError(err, "AS-23: ")

defer conn.Close()

buff := make([]byte, 1024)

bytes, err := conn.Read(buff)

common.HandleError(err, "AS-27: ")

fmt.Println("----- Body client request:", string(common.PrettyPrint(buff[:bytes-1])))

var req common.ASClientRequest

err = json.Unmarshal(buff[:bytes-1], &req)

common.HandleError(err, "AS-30: ")

tmp := req.TS

timeStamp := des.Decode(tmp, a.ClientsKeysMap[req.ID])

timeParsed, err := time.Parse("2006-01-02 15:04:05", timeStamp)

common.HandleError(err, "AS-34: ")

req.TS = timeParsed.String()

sessionKey := common.GenKey(8)

addr := "127.0.0.1:8001"

common.HandleError(err, "AS-38: ")

fmt.Println("+++++ Build client response... +++++")

resp := common.ASClientResponse{

SessionKey: sessionKey,

TS: req.TS,

TGSAddr: addr,

Exp: timeParsed.Add(a.ExpTime).String(),

}

tgt := common.TicketGrantingTicket{

ID: req.ID,

SessionKey: sessionKey,

TS: req.TS,

Addr: req.Addr,

Exp: resp.Exp,

}

respJSON, err := json.Marshal(resp)

common.HandleError(err, "AS-53: ")

fmt.Println("----- Body client response:", string(common.PrettyPrint(respJSON)))

tgtJSON, err := json.Marshal(tgt)

common.HandleError(err, "AS-55: ")

fmt.Println("----- Body TGT client response:", string(common.PrettyPrint(tgtJSON)))

respEncrypted := des.Encode(string(respJSON), a.ClientsKeysMap[req.ID])

tgtEncrypted := des.Encode(string(tgtJSON), a.TGSKey)

msg := common.ASResponseEncrypted{

ASClientResponse: respEncrypted,

TGT: tgtEncrypted,

}

msgJSON, err := json.Marshal(msg)

common.HandleError(err, "AS-63: ")

fmt.Println("----- Body client response encrypted:", string(common.PrettyPrint(msgJSON)))

msgJSON = append(msgJSON, '\n')

conn.Write(msgJSON)

}

func (a \*AS) HandleTGSRequest() {

conn, err := a.Listener.Accept()

common.HandleError(err, "AS-70: ")

defer conn.Close()

buff := make([]byte, 1024)

bytes, err := conn.Read(buff)

common.HandleError(err, "AS-74: ")

val, ok := a.ServersKeysMap[string(buff[:bytes-1])]

if !ok {

logrus.Fatal("No such server on AS")

}

conn.Write([]byte(val + "\n"))

}

# **Ticket granting server:**

**package tgs**

import (

"encoding/json"

"fmt"

"net"

"github.com/sirupsen/logrus"

"github.com/Muha113/6-th-term-labs/isob/lab2/pkg/des"

"github.com/Muha113/6-th-term-labs/isob/lab2/pkg/kerberos/common"

)

type TGS struct {

TGSKey string

SessionKey string

Listener net.Listener

}

func (t \*TGS) HandleClientRequest() {

fmt.Println("+++++ Handling client request... +++++")

conn, err := t.Listener.Accept()

common.HandleError(err, "TGS-20: ")

defer conn.Close()

buff := make([]byte, 1024)

bytes, err := conn.Read(buff)

common.HandleError(err, "TGS-24: ")

fmt.Println("----- Body client request encrypted:", string(common.PrettyPrint(buff[:bytes-1])))

var req common.TGSRequestEncrypted

err = json.Unmarshal(buff[:bytes-1], &req)

common.HandleError(err, "TGS-27: ")

fmt.Println("----- Ticket granting ticket:", string(common.PrettyPrint([]byte(des.Decode(req.TGT, t.TGSKey)))))

var tgt common.TicketGrantingTicket

err = json.Unmarshal([]byte(des.Decode(req.TGT, t.TGSKey)), &tgt)

common.HandleError(err, "TGS-30: ")

fmt.Println("----- Body TGS client request:", string(common.PrettyPrint([]byte(des.Decode(req.TGSClientRequest, tgt.SessionKey)))))

var cliReq common.TGSClientRequest

err = json.Unmarshal([]byte(des.Decode(req.TGSClientRequest, tgt.SessionKey)), &cliReq)

common.HandleError(err, "TGS-33: ")

if cliReq.ID != tgt.ID {

logrus.Fatal("Unequal tgt and client requests")

}

t.SessionKey = tgt.SessionKey

dialer, err := net.Dial("tcp", "127.0.0.1:8000")

common.HandleError(err, "TGS-40: ")

dialer.Write([]byte(cliReq.Req + "\n"))

bytes, err = dialer.Read(buff)

common.HandleError(err, "TGS-43: ")

dialer.Close()

fmt.Println("+++++ Build client response... +++++")

serverKey := string(buff[:bytes-1])

generatedKey := common.GenKey(8)

cliResp := common.TGSClientResponse{

GeneratedKey: generatedKey,

Dest: cliReq.Req,

TS: cliReq.TS,

Exp: tgt.Exp,

}

servData := common.TGSServerDataResponse{

ID: cliReq.ID,

GeneratedKey: generatedKey,

Addr: tgt.Addr,

TS: tgt.TS,

Exp: tgt.Exp,

}

servDataJSON, err := json.Marshal(servData)

common.HandleError(err, "TGS-61: ")

fmt.Println("----- Body servData client response:", string(common.PrettyPrint(servDataJSON)))

cliRespJSON, err := json.Marshal(cliResp)

common.HandleError(err, "TGS-63: ")

fmt.Println("----- Body cliResp client response:", string(common.PrettyPrint(cliRespJSON)))

servDataEncrypted := des.Encode(string(servDataJSON), serverKey)

cliRespEncrypted := des.Encode(string(cliRespJSON), tgt.SessionKey)

msg := common.TGSResponseEncrypted{

TGSClientResponse: cliRespEncrypted,

TGSServerDataResponse: servDataEncrypted,

}

msgJSON, err := json.Marshal(msg)

common.HandleError(err, "TGS-71: ")

fmt.Println("----- Body client response encrypted:", string(common.PrettyPrint(msgJSON)))

msgJSON = append(msgJSON, '\n')

conn.Write(msgJSON)

}

# **Domain server:**

**package main**

import (

"encoding/json"

"fmt"

"net"

"github.com/sirupsen/logrus"

"github.com/Muha113/6-th-term-labs/isob/lab2/pkg/des"

"github.com/Muha113/6-th-term-labs/isob/lab2/pkg/kerberos/common"

)

type Server struct {

ServerKey string

Listener net.Listener

}

func (s \*Server) HandleClientRequest() {

fmt.Println("+++++ Handling client request... +++++")

conn, err := s.Listener.Accept()

common.HandleError(err, "DomainServer-19: ")

defer conn.Close()

buff := make([]byte, 1024)

bytes, err := conn.Read(buff)

common.HandleError(err, "DomainServer-23: ")

fmt.Println("----- Body client request encrypted:", string(common.PrettyPrint(buff[:bytes-1])))

var req common.ServerRequestEncrypted

err = json.Unmarshal(buff[:bytes-1], &req)

common.HandleError(err, "DomainServer-26: ")

fmt.Println("----- Body TGS serverData client request:", string(common.PrettyPrint([]byte(des.Decode(req.TGSServerResponse, s.ServerKey)))))

var servReq common.TGSServerDataResponse

err = json.Unmarshal([]byte(des.Decode(req.TGSServerResponse, s.ServerKey)), &servReq)

common.HandleError(err, "DomainServer-34: ")

fmt.Println("----- Body servCli client request:", string(common.PrettyPrint([]byte(des.Decode(req.ServerClientRequest, servReq.GeneratedKey)))))

var cliReq common.ServerClientRequest

err = json.Unmarshal([]byte(des.Decode(req.ServerClientRequest, servReq.GeneratedKey)), &cliReq)

common.HandleError(err, "DomainServer-31: ")

if cliReq.ID != servReq.ID || cliReq.TS != servReq.TS {

logrus.Fatal("Unequal TGSResponse and ClientRequest to domain")

} else {

logrus.Println("User authenticated domain server")

}

fmt.Println("+++++ Build client response... +++++")

resp := common.ServerResponse{

Name: "TestServer",

TS: cliReq.TS,

}

respJSON, err := json.Marshal(resp)

common.HandleError(err, "DomainServer-40: ")

fmt.Println("----- Body client response:", string(common.PrettyPrint(respJSON)))

respEncrypted := des.Encode(string(respJSON), servReq.GeneratedKey)

msg := common.ServerResponseEncrypted{

ServerResponse: respEncrypted,

}

msgJSON, err := json.Marshal(msg)

common.HandleError(err, "DomainServer-46: ")

fmt.Println("----- Body client response encrypted:", string(common.PrettyPrint(msgJSON)))

msgJSON = append(msgJSON, '\n')

conn.Write(msgJSON)

}

func main() {

server := Server{

ServerKey: "ServMKey",

Listener: nil,

}

var err error

logrus.Println("Starting domain server...")

server.Listener, err = net.Listen("tcp", "127.0.0.1:8002")

common.HandleError(err, "DomainServer-58: ")

logrus.Printf("Listening on -> %s ...\n", server.Listener.Addr().String())

server.HandleClientRequest()

server.Listener.Close()

}

# **Des:**

# **1) cipher\_code:**

**package des**

import "strconv"

var initial\_replace\_matrix = [8][8]int{

{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},

}

var pc\_1 = [56]int{

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,

}

var pc\_2 = [48]int{

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,

}

var extended\_replacement\_matrix = [8][6]int{

{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},

}

var s\_box = [8][4][16]int{

{

{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},

},

}

var p\_box = [32]int{

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,

}

var reverse\_replace = [64]int{

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,

}

var displacements = [16]int{1, 1, 2, 2, 2, 2, 2, 2, 1, 2, 2, 2, 2, 2, 2, 1}

func getInitialReplaceMatrix() [8][8]int {

return initial\_replace\_matrix

}

func getExtendedReplacementMatrix() [8][6]int {

return extended\_replacement\_matrix

}

func getCN(round int) []int {

displacement := getDisplacements(round)

CN := make([]int, 28-displacement)

copy(CN, pc\_1[displacement:28])

for i := 0; i < displacement; i++ {

CN = append(CN, pc\_1[i])

}

return CN

}

func getDN(round int) []int {

displacement := getDisplacements(round)

DN := make([]int, 28-displacement)

copy(DN, pc\_1[28+displacement:])

for i := 0; i < displacement; i++ {

DN = append(DN, pc\_1[28+i])

}

return DN

}

func getCNDN(round int) []int {

CNDN := getCN(round)

DN := getDN(round)

for \_, ele := range DN {

CNDN = append(CNDN, ele)

}

return CNDN

}

func replace(CNDN []int, key string) string {

initial\_cipher\_code\_replace := ""

for i := 0; i < 56; i++ {

initial\_cipher\_code\_replace += string(key[CNDN[i]-1])

}

cipher\_code\_replace := ""

for i := 0; i < 48; i++ {

cipher\_code\_replace += string(initial\_cipher\_code\_replace[pc\_2[i]-1])

}

return cipher\_code\_replace

}

func getCipherCodeN(round int, key string) string {

CNDN := getCNDN(round)

return replace(CNDN, key)

}

func getKeys(key string) []string {

*// Encryption processing*

keys := make([]string, 0)

for i := 1; i <= 16; i++ {

keys = append(keys, getCipherCodeN(i, key))

}

return keys

}

func getDisplacements(round int) int {

displacement := 0

for i := 0; i < round; i++ {

displacement += displacements[i]

}

return displacement

}

func getSBoxN(index int) [4][16]int {

return s\_box[index]

}

func getPBox() [32]int {

return p\_box

}

func getReverseReplace() [64]int {

return reverse\_replace

}

func getRow(num1, num2 byte) int {

row, \_ := strconv.ParseInt(string(num1)+string(num2), 2, 64)

return int(row)

}

func getColumn(column string) int {

col, \_ := strconv.ParseInt(column, 2, 64)

return int(col)

}

# **2) des:**

**package des**

import (

"encoding/hex"

"fmt"

"os"

)

*// func Encode(text, key string) string {*

*// encoded := ""*

*// offset := len(key)*

*// for \_, v := range text {*

*// encoded += string(int(v) + offset)*

*// }*

*// return encoded*

*// }*

*// func Decode(text, key string) string {*

*// decoded := ""*

*// offset := len(key)*

*// for \_, v := range text {*

*// decoded += string(int(v) - offset)*

*// }*

*// return decoded*

*// }*

func Encode(clear\_text, key string) string {

extra := 8 - len(clear\_text)%8

for i := 0; i < extra; i++ {

clear\_text = clear\_text + string('0'+extra)

}

clear\_text = hex.EncodeToString([]byte(clear\_text))

return hexText(des(clear\_text, key, true))

}

func Decode(cipher\_text, key string) string {

clear\_text\_hex := hexText(des(cipher\_text, key, false))

clear\_text, \_ := hex.DecodeString(clear\_text\_hex)

clear\_text\_len := len(clear\_text)

return string(clear\_text[:clear\_text\_len-int(clear\_text[clear\_text\_len-1]-'0')])

}

func des(text, key string, tag bool) string {

*//logrus.Error("Key:", key, " Len:", len(key))*

if len(key) != 8 {

fmt.Println("The secret key need to be 8 bits.")

os.Exit(0)

}

key = formatKey(key)

keys := getKeys(key)

final\_text := ""

if !tag {

keys = reverse(keys)

}

for i := 0; i < len(text)/16; i++ {

textSub := binText(text[i\*16 : i\*16+16])

text\_init\_replace := initialReplace(textSub)

R\_16\_L\_16 := iteration(text\_init\_replace, keys)

final\_text += reverseReplace(R\_16\_L\_16)

}

return final\_text

}

# **3)initial\_replace:**

**package des**

func initialReplace(clear\_text string) string {

clear\_text\_init\_replace := ""

initial\_replace\_matrix := getInitialReplaceMatrix()

for i := 0; i < 8; i++ {

for j := 0; j < 8; j++ {

clear\_text\_init\_replace += string(clear\_text[initial\_replace\_matrix[i][j]-1])

}

}

return clear\_text\_init\_replace

}

# **4) iteration:**

**package des**

func iteration(clear\_text\_init\_replace string, keys []string) string {

L := make([]string, 0)

R := make([]string, 0)

L = append(L, clear\_text\_init\_replace[:32])

R = append(R, clear\_text\_init\_replace[32:])

for k := 0; k < 16; k++ {

*// Extended replacement*

R\_extended := extendedReplacement(R[k])

*// xor with keys[k]*

R\_extended\_xor := xorWithKeys\_K(R\_extended, keys[k])

*// S-box transfer*

R\_extended\_xor\_S\_trans := sBoxTransfer(R\_extended\_xor)

*// P-box transfer*

R\_extended\_xor\_S\_P\_trans := pBoxTransfer(R\_extended\_xor\_S\_trans)

*// xor with L[k]*

R\_extended\_xor\_S\_P\_trans\_xor := xorWithL\_K(R\_extended\_xor\_S\_P\_trans, L[k])

L = append(L, R[k])

R = append(R, R\_extended\_xor\_S\_P\_trans\_xor)

}

R\_16\_L\_16 := R[16] + L[16]

return R\_16\_L\_16

}

func extendedReplacement(R\_K string) string {

extended\_replacement\_matrix := getExtendedReplacementMatrix()

width := len(extended\_replacement\_matrix)

height := len(extended\_replacement\_matrix[0])

R\_extended := ""

for i := 0; i < width; i++ {

for j := 0; j < height; j++ {

R\_extended += string(R\_K[extended\_replacement\_matrix[i][j]-1])

}

}

return R\_extended

}

func xorWithKeys\_K(R\_extended, keys\_K string) string {

R\_extended\_xor := ""

for i := 0; i < len(keys\_K); i++ {

R\_extended\_xor += string((byte(R\_extended[i]-'0') ^ byte(keys\_K[i]-'0')) + '0')

}

return R\_extended\_xor

}

func sBoxTransfer(R\_extended\_xor string) string {

R\_extended\_xor\_S\_trans := ""

for i := 0; i < 8; i++ {

R\_extended\_xor\_slice := R\_extended\_xor[6\*i : 6\*(i+1)]

row := getRow(R\_extended\_xor\_slice[0], R\_extended\_xor\_slice[5])

column := getColumn(R\_extended\_xor\_slice[1:5])

S\_trans\_data := getSBoxN(i)[row][column]

*// fmt.Printf("Row: %d, Column: %d ", row, column)*

*// fmt.Printf("%d \n", S\_trans\_data)*

R\_extended\_xor\_S\_trans += decimalToBinary(S\_trans\_data)

}

return R\_extended\_xor\_S\_trans

}

func pBoxTransfer(R\_extended\_xor\_S\_trans string) string {

R\_extended\_xor\_S\_P\_trans := ""

p\_box := getPBox()

for i := 0; i < len(p\_box); i++ {

R\_extended\_xor\_S\_P\_trans += string(R\_extended\_xor\_S\_trans[p\_box[i]-1])

}

return R\_extended\_xor\_S\_P\_trans

}

func xorWithL\_K(R\_extended\_xor\_S\_P\_trans, L\_K string) string {

R\_extended\_xor\_S\_P\_trans\_xor := ""

for i := 0; i < len(L\_K); i++ {

R\_extended\_xor\_S\_P\_trans\_xor += string((byte(R\_extended\_xor\_S\_P\_trans[i]-'0') ^ byte(L\_K[i]-'0')) + '0')

}

return R\_extended\_xor\_S\_P\_trans\_xor

}

# **5) reverse\_replace:**

**package des**

func reverseReplace(R\_16\_L\_16 string) string {

*// reverse replace*

cipher\_text := ""

reverse\_replace := getReverseReplace()

for i := 0; i < len(reverse\_replace); i++ {

cipher\_text += string(R\_16\_L\_16[reverse\_replace[i]-1])

}

return cipher\_text

}

# **6) utils:**

**package des**

import (

"encoding/hex"

"strconv"

)

func decimalToBinary(data int) string {

switch data {

case 0:

return "0000"

case 1:

return "0001"

case 2:

return "0010"

case 3:

return "0011"

case 4:

return "0100"

case 5:

return "0101"

case 6:

return "0110"

case 7:

return "0111"

case 8:

return "1000"

case 9:

return "1001"

case 10:

return "1010"

case 11:

return "1011"

case 12:

return "1100"

case 13:

return "1101"

case 14:

return "1110"

case 15:

return "1111"

}

return ""

}

func hexadecimalToBinary(data byte) string {

switch data {

case '0':

return "0000"

case '1':

return "0001"

case '2':

return "0010"

case '3':

return "0011"

case '4':

return "0100"

case '5':

return "0101"

case '6':

return "0110"

case '7':

return "0111"

case '8':

return "1000"

case '9':

return "1001"

case 'a':

return "1010"

case 'b':

return "1011"

case 'c':

return "1100"

case 'd':

return "1101"

case 'e':

return "1110"

case 'f':

return "1111"

}

return ""

}

func reverse(keys []string) []string {

keys\_reverse := make([]string, 0)

for i := len(keys) - 1; i >= 0; i-- {

keys\_reverse = append(keys\_reverse, keys[i])

}

return keys\_reverse

}

func formatKey(key string) string {

hexKey := hex.EncodeToString([]byte(key))

binKey := ""

for i := 0; i < len(hexKey); i++ {

binKey += hexadecimalToBinary(hexKey[i])

}

return binKey

}

func binText(text string) string {

binText := ""

for i := 0; i < len(text); i++ {

binText += hexadecimalToBinary(text[i])

}

return binText

}

func hexText(text string) string {

hexText := ""

for i := 0; i < len(text)/4; i++ {

dec\_text, \_ := strconv.ParseInt(text[i\*4:i\*4+4], 2, 64)

hexText += strconv.FormatInt(dec\_text, 16)

}

return hexText

}