Kerberos 认证模型

原理概述

Kerberos是一种基于“tickets”的计算机网络认证协议，它允许通过非安全网络通信的节点以安全的方式相互证明其身份，是一种对称密钥体制进行密钥管理的系统。

主要针对客户机-服务器模型，它提供了相互身份验证-用户和服务器都验证彼此的身份，可以防止窃听和重播攻击。

总体结构设计

四个并发进程：AS、TGS、SS、Client

用户输入用户名和口令密码进行登录

Client将口令密码通过Hash转换成主密钥K_{Client}，该密钥是用户和AS预先协商好的；

进行身份认证：

Client向AS发送明文信息，发起请求

AS根据用户ID进行数据库确认，然后返回消息A、B

$$A:E(K_{Client},K_{Client-TGS})$$
$$B:E(K_{TGS},<clientID,clientaddress,validity,K_{Client-TGS}>)$$

Client解密消息A得的$K_{Client-TGS}$

$$D(K_{Client},A)=K_{Client-TGS}$$

进行服务认证：

Client向TGS发送信息C、D

$$C:serviceID,B$$
$$D:E(K_{Client-TGS},<clientID,timestamp>)$$

TGS通过

$$D(K_{TGS}, B) = K_{Client-TGS}$$

再通过

$$D(K_{Client-TGS}) =< clientID, timestamp >$$

TGS返回消息E、F

$$E : serviceID, ST$$
$$ST = E(K_{SS}, < clientID, client\ net\ address, validity, K_{Client-SS} >)$$
$$F : E(K_{Client-TGS}, K_{Client-SS})$$

Client通过

$$D(K_{Client-TGS}, F) = K_{Client-SS}$$

进行服务申请：

Client向TGS发送信息E、G

$$E : serviceID, ST$$
$$G : E(K_{Client-SS}, < clientID, timestamp >)$$

SS通过

$$D(K_{SS}, ST) = K_{Client-SS}$$
$$D(K_{Client-SS}, G) =< clientID, timestamp >$$

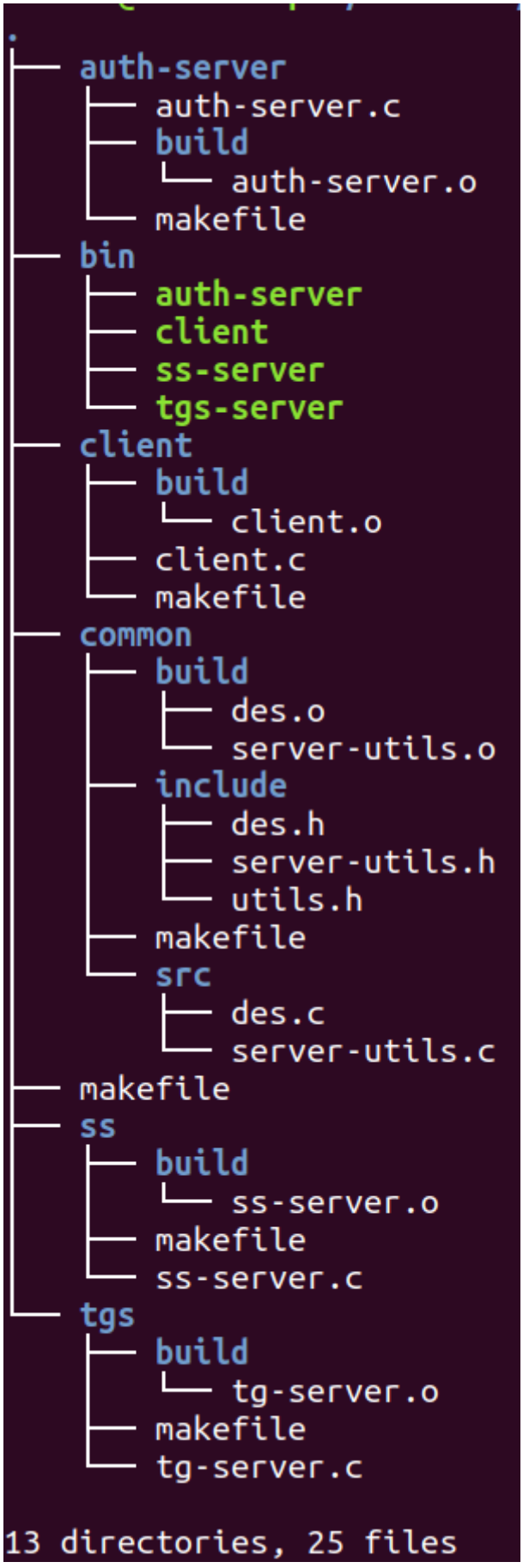
返回消息H用于进一步的确认

$$H : E(K_{Client-SS}, < clientID, timestamp + 1 >)$$

Client解密消息H得到时间戳，进行比对，如果时间戳被正确更新，则SS可以信赖，后续可以发送服务请求，而SS也会提供相应的服务。

模块分解

文件结构



- auth-server文件夹下存储AS的程序；
- client文件夹下存储client的程序；
- tgs文件夹下存储TGS的程序；
- ss文件夹下存储SS的程序；
- common文件夹下存储其它程序所使用的组件，包括

    - utils.h下存储IP地址和端口号；
    - server-utils下存储网络通讯相关的函数；
    - des下存储DES加解密相关的函数；
- bin文件夹下存储编译好的可执行文件；

server-utils

根据端口号生成服务端socket

```
int createServSocket(int port) {
    int listenfd = socket(AF_INET, SOCK_STREAM, 0);
    if (listenfd < 0) {
        fprintf(stderr, "create socket error: %s\n", strerror(errno));
        exit(errno);
    }

    struct sockaddr_in servAddr;
    memset(&servAddr, 0, sizeof(servAddr));
    servAddr.sin_family = AF_INET;
    servAddr.sin_addr.s_addr = htonl(INADDR_ANY);
    servAddr.sin_port = htons(port);

    int res;
    res = bind(listenfd, (struct sockaddr *)&servAddr, sizeof(servAddr));
    if (res < 0) {
        fprintf(stderr, "bind error: %s\n", strerror(errno));
        exit(errno);
    }

    res = listen(listenfd, 10);
    if (res < 0) {
        fprintf(stderr, "listen error: %s\n", strerror(errno));
        exit(errno);
    }
    return listenfd;
}
```

DES

基本沿用第一次作业的代码，对于函数进行了一定封装，不进行进一步的介绍；

根据主密钥生成各类子密钥

```
void generateSubKey(unsigned char *main_key);
```

加密

```
void encryption(unsigned char *message_piece, unsigned char *processed_piece);
```

解密

```
void decryption(unsigned char *message_piece, unsigned char *processed_piece);
```

Client

根据IP地址和端口号连接服务端socket

```
int connServSocket(const char *ip, int port) {
    int sockfd = socket(AF_INET, SOCK_STREAM, 0);
    if (sockfd == -1) {
        fprintf(stderr, "create socket error: %s\n", strerror(errno));
        exit(errno);
    }

    int res;
    struct sockaddr_in servAddr;
    memset(&servAddr, 0, sizeof(servAddr));
    servAddr.sin_family = AF_INET;
    servAddr.sin_port = htons(port);
    res = inet_pton(AF_INET, ip, &servAddr.sin_addr);
    if (res <= 0) {
        fprintf(stderr, "inet_pton error (%d)\n", res);
        exit(-1);
    }

    res = connect(sockfd, (struct sockaddr *)&servAddr, sizeof(servAddr));
    if (res < 0) {
        fprintf(stderr, "connect error: %s\n", strerror(errno));
        exit(errno);
    }
    return sockfd;
}
```

与AS服务端进行交互

```
int asAuth(const char *username, unsigned char *msgA, unsigned char *msgB);
```

与TGS服务端进行交互

```
int tgsAuth(unsigned char *msgC, unsigned char *cliTgsKey, unsigned char *msgE,
unsigned char *msgF);
```

与SS服务端进行交互

```
int ssAuth(unsigned char *msgE, unsigned char *cliSsKey, unsigned char *msgH);
```

主要过程

1. 发送字符串

```
int sockfd = connServSocket(ASIP, ASPORT);
int len = send(sockfd, username, strlen(username), 0);
if (len < 0) {
    fprintf(stderr, "send error: %s\n", strerror(errno));
    return errno;
}
```

2. 接收字符串

```
int msgLenA = recv(sockfd, msgA, BUFSIZE - 1, 0);
msgA[msgLenA] = '\0';

close(sockfd);
```

根据需要将多个信息打包成一条字符串

```
unsigned char *pkgMsgD(unsigned char *msgD);
unsigned char *pkgMsgG(unsigned char *msgG);
```

如msgD需要将clientID和timestamp打包在一起等，打包方式是字符串按序连接，一些特殊的部分如客户网络地址则会进行补位；

服务端

主要过程

1. 接收

```
int listenfd = createServSocket(ASPORT);
struct sockaddr cliAddr;
socklen_t addrLen;
int connfd = accept(listenfd, &cliAddr, &addrLen);
if (connfd == -1) {
    fprintf(stderr, "accept error: %s", strerror(errno));
    continue;
}
char buf[BUFSIZE];
int n = recv(connfd, buf, BUFSIZE - 1, 0);
buf[n] = 0;
```

2. 发送

```
int len;
len = send(connfd, msgA, BUFSIZE - 1, 0);
if (len < 0) {
    fprintf(stderr, "send error: %s\n", strerror(errno));
    return errno;
}

close(listenfd);
```

根据需要将多个信息打包成一条字符串

```
unsigned char *pkgMsgB(char *msgB, unsigned char *cliTgsKey, struct sockaddr
*cliAddr, unsigned char *clientID);
...
```

pkgMsgB

后续的时间戳、服务票据有效期都使用time函数生成，其返回自纪元 Epoch（1970-01-01 00:00:00 UTC）起经过的时间，以秒为单位。

转换为字符串后共10位。

```
unsigned char *pkgMsgB(char *msgB, unsigned char *cliTgsKey,
                       struct sockaddr *cliAddr, unsigned char *clientID) {
    strcpy(msgB, clientID);
    strcat(msgB, cliAddr->sa_data);
    for (int i = 0; i < 14 - strlen(cliAddr->sa_data); i++) {
        strcat(msgB, "0");
    }

    time_t validate;
    time(&validate);
    validate += 10 * 60;
    unsigned char timebuffer[50];
    sprintf(timebuffer, "%ld", validate);
    strcat(msgB, timebuffer);

    strcat(msgB, cliTgsKey);
    return msgB;
}
```

消息A

AS

```
unsigned char msgA[BUFSIZE];
...
generateSubKey(cliKey);
encryption(cliTgsKey, msgA);
...
len = send(connfd, msgA, BUFSIZE - 1, 0);
```

client：解密得到client-TGS密钥

```
unsigned char cliTgsKey[BUFSIZE];
generateSubKey(cliKey);
decryption(msgA, cliTgsKey);
```

消息B

AS

```
unsigned char msgB[BUFSIZE], buf[BUFSIZE];
generateSubKey(TGSKEY);
pkgMsgB(buf, cliTgsKey, &cliAddr, username);
encryption(buf, msgB);
...
len = send(connfd, msgB, BUFSIZE - 1, 0);
```

client：开头接上serviceID，直接转发给TGS，即消息C

```
unsigned char msgC[BUFSIZE], msgE[BUFSIZE], msgF[BUFSIZE];
strcpy(msgC, serviceID);
strcat(msgC, msgB);
tgsAuth(msgC, cliTgsKey, msgE, msgF);
```

消息C

TGS：解码，并保存serviceID、client-TGS密钥，校验validity

```
unsigned char msgB[BUFSIZE], leftB[BUFSIZE];

strcpy(leftB, buf + 6);
strncpy(serviceID, buf, 6);
serviceID[6] = 0;

generateSubKey(TGSKEY);
decryption(leftB, msgB);
...
unsigned char timebuffer[50];
memset(timebuffer, 0, sizeof(timebuffer));
strncpy(timebuffer, msgB + 22, 10);
timebuffer[10] = 0;
printf("msgB validity: %s\n", timebuffer);
time_t start = atol(timebuffer);
time_t end;
time(&end);
printf("nowtime: %ld\n", end);
double cost = difftime(start, end);
if (cost <= 0) {
    printf("\nRefuse: Validate is out of date.\n");
    return 0;
}

strcpy(cliTgsKey, msgB + 32);
```

消息D

client

```
unsigned char msgD[BUFSIZE], sendD[BUFSIZE];
generateSubKey(cliTgsKey);
pkgMsgD(msgD);
encryption(msgD, sendD);
...
len = send(sockfd, sendD, BUFSIZE - 1, 0);
```

TGS：解密得到clientID、timestamp

```
unsigned char msgD[BUFSIZE], pkgMsgD[BUFSIZE];
generateSubKey(cliTgsKey);

decryption(buf, msgD);
...
unsigned char timestamp[BUFSIZE];
strncpy(clientID, msgD, 8);
clientID[8] = 0;
strncpy(timestamp, msgD + 8, 10);
timestamp[10] = 0;
```

消息E

TGS

```
unsigned char msgE[BUFSIZE], enE[BUFSIZE], sendE[BUFSIZE];
pkgMsgE(msgE, getCliSsKey(), &cliAddr);
generateSubKey(SSKEY);
encryption(msgE, enE);

strcpy(sendE, serviceID);
strcat(sendE, enE);

len = send(connfd, sendE, BUFSIZE - 1, 0);
```

Client：转发给SS

```
int len = send(sockfd, msgE, BUFSIZE - 1, 0);
```

SS：解密ST，校验validity，得到client-SS密钥

```
unsigned char st[BUFSIZE], msgE[BUFSIZE];
strcpy(st, buf + 6);
generateSubKey(SSKEY);

decryption(st, msgE);

unsigned char timebuffer[50];
memset(timebuffer, 0, sizeof(timebuffer));
strncpy(timebuffer, msgE + 22, 10);
timebuffer[10] = 0;
printf("ST validity: %s\n", timebuffer);
time_t start = atol(timebuffer);
time_t end;
time(&end);
printf("nowtime: %ld\n", end);
double cost = difftime(start, end);
if (cost < 0) {
    printf("\nRefuse: Validate is out of date.\n");
    return 0;
}

strcpy(cliSsKey, msgE + 32);
```

消息F

TGS

```
unsigned char msgF[BUFSIZE];
generateSubKey(cliTgsKey);
encryption(getCliSsKey(), msgF);

len = send(connfd, msgF, BUFSIZE - 1, 0);
```

Client：解密得到client-SS密钥

```
unsigned char cliSsKey[BUFSIZE];
generateSubKey(cliTgsKey);
decryption(msgF, cliSsKey);
```

消息G

Client

```
unsigned char msgG[BUFSIZE], sendG[BUFSIZE];
generateSubKey(cliSsKey);
pkgMsgG(msgG);
encryption(msgG, sendG);
...
len = send(sockfd, sendG, BUFSIZE - 1, 0);
```

SS：解密获得clientID和时间戳

```
unsigned char msgG[BUFSIZE], pkgMsgD[BUFSIZE];
generateSubKey(cliSsKey);

decryption(buf, msgG);
...
unsigned char timestamp[50];
strncpy(clientID, msgG, 8);
clientID[8] = 0;
strcpy(timestamp, msgG + 8);
```

消息H

SS

```
unsigned char msgH[BUFSIZE], sendH[BUFSIZE];
pkgMsgH(msgH, timestamp);
generateSubKey(cliSsKey);
encryption(msgH, sendH);
...
int len;
len = send(connfd, sendH, BUFSIZE - 1, 0);
```

pkgMsgH函数将时间戳加1，打包信息

Client：解密，校验clientID和时间戳是否正确，正确则流程完成，否则报错；

```
unsigned char getH[BUFSIZE];
generateSubKey(cliSsKey);
decryption(msgH, getH);
...
unsigned char recCliID[BUFSIZE], recTimestamp[BUFSIZE];
strncpy(recCliID, getH, 8);
recCliID[8] = 0;
strcpy(recTimestamp, getH + 8);
if (strcmp(recCliID, clientID) != 0) {
    printf("recClientID: %s\n", recCliID);
    printf("Wrong clientID\n");
    return 0;
}

unsigned char timebuffer[50];
memset(timebuffer, 0, sizeof(timebuffer));
strncpy(timebuffer, recTimestamp, 10);
timebuffer[10] = 0;
printf("timestamp: %s\n", timebuffer);
time_t msgHTS = atol(timebuffer);

if ((msgHTS - msgGTS - 1.0) > 0.001) {
    printf("recTimestamp: %s\n", recTimestamp);
    printf("Wrong timestamp\n");
    return 0;
}

printf("Success!\n");
```

数据结构设计

密钥、文本、通讯的报文等都是保存在unsigned char数组中，后续的加解密、传输、校验等操作都通过数组进行；

C 语言源代码

Client

```
#include <arpa/inet.h>
#include <errno.h>
#include <memory.h>
#include <netinet/in.h>
#include <stdio.h>
#include <stdlib.h>
#include <sys/socket.h>
#include <time.h>
#include <unistd.h>

#include "des.h"
#include "utils.h"

#define BUFSIZE 1024

unsigned char *serviceID = "000000";
```

```c
unsigned char *clientID = "username";
unsigned char *username = "username";
unsigned char *clientKey = "12345678";

time_t msgGTS;

int connServSocket(const char *asIp, int asPort);
int asAuth(const char *username, unsigned char *msgA, unsigned char *msgB);
int tgsAuth(unsigned char *msgC, unsigned char *cliTgsKey, unsigned char *msgE,
            unsigned char *msgF);
int ssAuth(unsigned char *msgE, unsigned char *cliSsKey, unsigned char *msgH);
unsigned char *getCliKey(const char *username);

unsigned char *pkgMsgD(unsigned char *msgD);
unsigned char *pkgMsgG(unsigned char *msgG);

int main(int argc, char const *argv[]) {
    // char username[10];
    // strcpy(username, "username");
    printf("username: %s\n", username);
    printf("clientID: %s\n", username);
    printf("serviceID: %s\n", serviceID);

    unsigned char *cliKey = getCliKey(username);
    printf("clientKey: %s\n\n", cliKey);

    unsigned char msgA[BUFSIZE], msgB[BUFSIZE];
    asAuth(username, msgA, msgB);

    unsigned char cliTgsKey[BUFSIZE];
    generateSubKey(cliKey);
    // decryptionMsg(msgA, cliTgsKey);
    decryption(msgA, cliTgsKey);

    printf("msgA: ");
    for (int i = 0; msgA[i] != '\0'; i++) {
        printf("%02x", msgA[i]);
    }
    printf("\n");

    printf("msgB: ");
    for (int i = 0; msgB[i] != '\0'; i++) {
        printf("%02x", msgB[i]);
    }
    printf("\n");

    // printf("msgA: %02x\n", msgA);
    // printf("msgB: %02x\n", msgB);
    printf("cliTgsKey: %s\n", cliTgsKey);

    unsigned char msgC[BUFSIZE], msgE[BUFSIZE], msgF[BUFSIZE];
    strcpy(msgC, serviceID);
    strcat(msgC, msgB);
    tgsAuth(msgC, cliTgsKey, msgE, msgF);

    unsigned char cliSsKey[BUFSIZE];
    generateSubKey(cliTgsKey);
    decryption(msgF, cliSsKey);
```

```
    printf("msgE: ");
    for (int i = 0; msgE[i] != '\0'; i++) {
        printf("%02x", msgE[i]);
    }
    printf("\n");

    printf("msgF: ");
    for (int i = 0; msgF[i] != '\0'; i++) {
        printf("%02x", msgF[i]);
    }
    printf("\n");

    // printf("msgE: %s\n", msgE);
    // printf("msgF: %s\n", msgF);
    printf("cliSsKey: %s\n", cliSsKey);

    unsigned char msgH[BUFSIZE];
    ssAuth(msgE, cliSsKey, msgH);

    unsigned char getH[BUFSIZE];
    generateSubKey(cliSsKey);
    decryption(msgH, getH);

    printf("msgH: ");
    for (int i = 0; msgH[i] != '\0'; i++) {
        printf("%02x", msgH[i]);
    }
    printf("\n");
    // printf("msgH: %s\n", msgH);
    printf("msgH after dec: %s\n", getH);

    unsigned char recCliID[BUFSIZE], recTimestamp[BUFSIZE];
    strncpy(recCliID, getH, 8);
    recCliID[8] = 0;
    strcpy(recTimestamp, getH + 8);
    if (strcmp(recCliID, clientID) != 0) {
        printf("recClientID: %s\n", recCliID);
        printf("Wrong clientID\n");
        return 0;
    }

    unsigned char timebuffer[50];
    memset(timebuffer, 0, sizeof(timebuffer));
    strncpy(timebuffer, recTimestamp, 10);
    timebuffer[10] = 0;
    printf("timestamp: %s\n", timebuffer);
    time_t msgHTS = atol(timebuffer);

    if ((msgHTS - msgGTS - 1.0) > 0.001) {
        printf("recTimestamp: %s\n", recTimestamp);
        printf("Wrong timestamp\n");
        return 0;
    }

    printf("Success!\n");

    return 0;
```

```
}

int asAuth(const char *username, unsigned char *msgA, unsigned char *msgB) {
    int sockfd = connServSocket(ASIP, ASPORT);

    printf("\nsend request\n");
    int len = send(sockfd, username, strlen(username), 0);
    if (len < 0) {
        fprintf(stderr, "send error: %s\n", strerror(errno));
        return errno;
    }

    printf("\nreceive msgA & msgB\n");
    int msgLenA = recv(sockfd, msgA, BUFSIZE - 1, 0);
    msgA[msgLenA] = '\0';
    int msgLenB = recv(sockfd, msgB, BUFSIZE - 1, 0);
    msgB[msgLenB] = '\0';

    close(sockfd);
}

int tgsAuth(unsigned char *msgC, unsigned char *cliTgsKey, unsigned char *msgE,
            unsigned char *msgF) {
    int sockfd = connServSocket(TGSIP, TGSPORT);

    printf("\nsend msgC\n");
    // printf("msgC: %s\n", msgC);
    printf("msgC: ");
    for (int i = 0; msgC[i] != '\0'; i++) {
        printf("%02x", msgC[i]);
    }
    printf("\n");

    int len = send(sockfd, msgC, BUFSIZE - 1, 0);
    if (len < 0) {
        fprintf(stderr, "send error: %s\n", strerror(errno));
        return errno;
    }

    printf("\nsend msgD\n");
    unsigned char msgD[BUFSIZE], sendD[BUFSIZE];
    generateSubKey(cliTgsKey);
    pkgMsgD(msgD);
    encryption(msgD, sendD);

    // printf("sendD: %s\n", sendD);
    printf("msgD before enc: %s\n", msgD);

    printf("msgD: ");
    for (int i = 0; sendD[i] != '\0'; i++) {
        printf("%02x", sendD[i]);
    }
    printf("\n");

    len = send(sockfd, sendD, BUFSIZE - 1, 0);
    if (len < 0) {
        fprintf(stderr, "send error: %s\n", strerror(errno));
        return errno;
```

```
    }

    printf("\nreceive msgE & msgF\n");

    int msgLenE = recv(sockfd, msgE, BUFSIZE - 1, 0);
    msgE[msgLenE] = '\0';
    int msgLenF = recv(sockfd, msgF, BUFSIZE - 1, 0);
    msgF[msgLenF] = '\0';

    close(sockfd);
}

int ssAuth(unsigned char *msgE, unsigned char *cliSsKey, unsigned char *msgH) {
    int sockfd = connServSocket(SSIP, SSPORT);

    printf("\nsend msgE\n");
    // printf("msgE: %s\n", msgE);
    printf("msgE: ");
    for (int i = 0; msgE[i] != '\0'; i++) {
        printf("%02x", msgE[i]);
    }
    printf("\n");

    int len = send(sockfd, msgE, BUFSIZE - 1, 0);
    if (len < 0) {
        fprintf(stderr, "send error: %s\n", strerror(errno));
        return errno;
    }

    printf("\nsend msgG\n");
    unsigned char msgG[BUFSIZE], sendG[BUFSIZE];
    generateSubKey(cliSsKey);
    pkgMsgG(msgG);
    encryption(msgG, sendG);

    // printf("sendG: %s\n", sendG);
    printf("msgG before enc: %s\n", msgG);

    printf("msgG: ");
    for (int i = 0; sendG[i] != '\0'; i++) {
        printf("%02x", sendG[i]);
    }
    printf("\n");

    len = send(sockfd, sendG, BUFSIZE - 1, 0);
    if (len < 0) {
        fprintf(stderr, "send error: %s\n", strerror(errno));
        return errno;
    }

    printf("\nreceive msgH\n");

    int msgLenH = recv(sockfd, msgH, BUFSIZE - 1, 0);
    msgH[msgLenH] = '\0';

    close(sockfd);
}
```

```c
int connServSocket(const char *ip, int port) {
    int sockfd = socket(AF_INET, SOCK_STREAM, 0);
    if (sockfd == -1) {
        fprintf(stderr, "create socket error: %s\n", strerror(errno));
        exit(errno);
    }

    int res;
    struct sockaddr_in servAddr;
    memset(&servAddr, 0, sizeof(servAddr));
    servAddr.sin_family = AF_INET;
    servAddr.sin_port = htons(port);
    res = inet_pton(AF_INET, ip, &servAddr.sin_addr);
    if (res <= 0) {
        fprintf(stderr, "inet_pton error (%d)\n", res);
        exit(-1);
    }

    res = connect(sockfd, (struct sockaddr *)&servAddr, sizeof(servAddr));
    if (res < 0) {
        fprintf(stderr, "connect error: %s\n", strerror(errno));
        exit(errno);
    }
    return sockfd;
}

unsigned char *getCliKey(const char *username) { return clientKey; }

unsigned char *pkgMsgD(unsigned char *msgD) {
    strcpy(msgD, clientID);

    time_t start;
    time(&start);
    unsigned char timebuffer[50];
    sprintf(timebuffer, "%ld", start);
    printf("msgD timestamp: %s\n", timebuffer);

    strcat(msgD, timebuffer);
    return msgD;
}

unsigned char *pkgMsgG(unsigned char *msgG) {
    strcpy(msgG, clientID);

    time(&msgGTS);
    unsigned char timebuffer[50];
    sprintf(timebuffer, "%ld", msgGTS);
    printf("msgG timestamp: %s\n", timebuffer);

    strcat(msgG, timebuffer);
    return msgG;
}
```

AS

```c
#include <errno.h>
#include <memory.h>
#include <stdbool.h>
#include <stdio.h>
#include <string.h>
#include <sys/socket.h>
#include <time.h>
#include <unistd.h>

#include "des.h"
#include "server-utils.h"
#include "utils.h"

#define BUFSIZE 1024

unsigned char TGSKEY[] = "87654321";
unsigned char clientKey[] = "12345678";
unsigned char cliTgsKey[] = "23456789";

bool isValidUsername(const char *username);
int responseReq(const char *buf, int n, int connfd, struct sockaddr cliAddr,
                socklen_t addrLen);
unsigned char *getCliKey(const char *username);
unsigned char *getCliTgsKey();
unsigned char *pkgMsgB(char *msgB, unsigned char *cliTgsKey,
                       struct sockaddr *cliAddr, unsigned char *username);

int main(int argc, char const *argv[]) {
    printf("TgsKey: %s\n", TGSKEY);
    printf("clientKey: %s\n", clientKey);
    printf("cliTgsKey: %s\n\n", cliTgsKey);

    int listenfd = createServSocket(ASPORT);

    printf("======waiting for client's request======\n");
    while (1) {
        struct sockaddr cliAddr;
        socklen_t addrLen;
        int connfd = accept(listenfd, &cliAddr, &addrLen);
        if (connfd == -1) {
            fprintf(stderr, "accept error: %s", strerror(errno));
            continue;
        }
        char buf[BUFSIZE];
        int n = recv(connfd, buf, BUFSIZE - 1, 0);
        buf[n] = 0;
        if (n > 0) responseReq(buf, n, connfd, cliAddr, addrLen);
        close(connfd);

        printf("Success!\n");
    }

    close(listenfd);
    return 0;
}
```

```c
int responseReq(const char *buf, int n, int connfd, struct sockaddr cliAddr,
                socklen_t addrLen) {
    char username[n + 1];
    memcpy(username, buf, n);
    username[n] = '\0';
    printf("server: receive request from client (%s)\n", buf);
    if (isValidUsername(username)) {
        unsigned char *cliKey = getCliKey(username);
        unsigned char *cliTgsKey = getCliTgsKey();

        printf("\nsend msgA\n");

        unsigned char msgA[BUFSIZE];
        for (int i = 0; i < BUFSIZE; i++) {
            msgA[i] = '\0';
        }

        generateSubKey(cliKey);
        // encryptionMsg(cliTgsKey, msgA);
        encryption(cliTgsKey, msgA);

        printf("cliTgsKey: %s\n", cliTgsKey);
        // printf("len: %ld\n", strlen(msgA));
        // printf("msgA: %s\n", msgA);

        printf("msgA: ");
        for (int i = 0; msgA[i] != '\0'; i++) {
            printf("%02x", msgA[i]);
        }
        printf("\n");

        int len;
        len = send(connfd, msgA, BUFSIZE - 1, 0);
        if (len < 0) {
            fprintf(stderr, "send error: %s\n", strerror(errno));
            return errno;
        }

        printf("\nsend msgB\n");
        unsigned char msgB[BUFSIZE], buf[BUFSIZE];
        generateSubKey(TGSKEY);
        pkgMsgB(buf, cliTgsKey, &cliAddr, username);
        // encryptionMsg(pkgMsgB(buf, cliTgsKey, &cliAddr, username), msgB);
        encryption(buf, msgB);

        // unsigned char temp[BUFSIZE];
        // decryptionMsg(msgB, temp);

        // printf("msgB: %s\n", msgB);
        printf("msgB before enc: %s\n", buf);
        // printf("temp: %s\n", temp);

        printf("msgB: ");
        for (int i = 0; msgB[i] != '\0'; i++) {
            printf("%02x", msgB[i]);
        }
        printf("\n");
```

```
        len = send(connfd, msgB, BUFSIZE - 1, 0);
        if (len < 0) {
            fprintf(stderr, "send error: %s\n", strerror(errno));
            return errno;
        }
    }
    return 0;
}

bool isValidUsername(const char *username) { return true; }
unsigned char *getCliKey(const char *username) { return clientKey; }
unsigned char *getCliTgsKey() { return cliTgsKey; }

unsigned char *pkgMsgB(char *msgB, unsigned char *cliTgsKey,
                       struct sockaddr *cliAddr, unsigned char *clientID) {
    strcpy(msgB, clientID);
    strcat(msgB, cliAddr->sa_data);
    for (int i = 0; i < 14 - strlen(cliAddr->sa_data); i++) {
        strcat(msgB, "0");
    }

    time_t validate;
    time(&validate);
    validate += 10 * 60;
    unsigned char timebuffer[50];
    sprintf(timebuffer, "%ld", validate);
    strcat(msgB, timebuffer);

    strcat(msgB, cliTgsKey);
    return msgB;
}
```

TGS

```
#include <errno.h>
#include <memory.h>
#include <stdbool.h>
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include <sys/socket.h>
#include <time.h>
#include <unistd.h>

#include "des.h"
#include "server-utils.h"
#include "utils.h"

#define BUFSIZE 1024

unsigned char TGSKEY[] = "87654321";
unsigned char SSKEY[] = "76543210";
unsigned char cliSsKey[] = "98765432";
unsigned char cliTgsKey[BUFSIZE];
unsigned char clientID[BUFSIZE];
unsigned char serviceID[BUFSIZE];
```

```c
int responseReqC(unsigned char *buf, int n, int connfd, struct sockaddr cliAddr,
                 socklen_t addrLen);
int responseReqD(unsigned char *buf, int n, int connfd, struct sockaddr cliAddr,
                 socklen_t addrLen);
unsigned char *getCliSsKey();
unsigned char *pkgMsgE(char *msgB, unsigned char *cliTgsKey,
                       struct sockaddr *cliAddr);

int main(int argc, char const *argv[]) {
    printf("TgsKey: %s\n", TGSKEY);
    printf("SsKey: %s\n", SSKEY);
    printf("cliSsKey: %s\n\n", cliSsKey);

    int listenfd = createServSocket(TGSPORT);

    printf("======waiting for client's request======\n");
    while (1) {
        struct sockaddr cliAddr;
        socklen_t addrLen;
        int connfd = accept(listenfd, &cliAddr, &addrLen);
        if (connfd == -1) {
            fprintf(stderr, "accept error: %s", strerror(errno));
            continue;
        }

        unsigned char msgC[BUFSIZE];
        int n = recv(connfd, msgC, BUFSIZE - 1, 0);
        msgC[n] = '\0';
        if (n > 0) responseReqC(msgC, n, connfd, cliAddr, addrLen);

        // connfd = accept(listenfd, &cliAddr, &addrLen);
        // if (connfd == -1) {
        //     fprintf(stderr, "accept error: %s", strerror(errno));
        //     continue;
        // }
        unsigned char msgD[BUFSIZE];
        n = recv(connfd, msgD, BUFSIZE - 1, 0);
        msgD[n] = '\0';
        if (n > 0) responseReqD(msgD, n, connfd, cliAddr, addrLen);

        close(connfd);

        printf("Success!\n");
    }

    close(listenfd);
    return 0;
}

int responseReqC(unsigned char *buf, int n, int connfd, struct sockaddr cliAddr,
                 socklen_t addrLen) {
    printf("server: receive request from client\n");
    printf("\nreceive msgC\n");

    unsigned char msgB[BUFSIZE], leftB[BUFSIZE];

    strcpy(leftB, buf + 6);
```

```
    strncpy(serviceID, buf, 6);
    serviceID[6] = 0;

    generateSubKey(TGSKEY);
    decryption(leftB, msgB);

    printf("msgC: ");
    for (int i = 0; buf[i] != '\0'; i++) {
        printf("%02x", buf[i]);
    }
    printf("\n");

    // printf("msgC: %s\n", buf);
    // printf("leftB: %s\n", leftB);
    printf("serviceID: %s\n", serviceID);
    printf("TGSKEY: %s\n", TGSKEY);
    printf("msgB after dec: %s\n", msgB);

    unsigned char timebuffer[50];
    memset(timebuffer, 0, sizeof(timebuffer));
    strncpy(timebuffer, msgB + 22, 10);
    timebuffer[10] = 0;
    printf("msgB validity: %s\n", timebuffer);
    time_t start = atol(timebuffer);
    time_t end;
    time(&end);
    printf("nowtime: %ld\n", end);
    double cost = difftime(start, end);
    if (cost <= 0) {
        printf("\nRefuse: Validate is out of date.\n");
        return 0;
    }

    strcpy(cliTgsKey, msgB + 32);

    return 0;
}

int responseReqD(unsigned char *buf, int n, int connfd, struct sockaddr cliAddr,
                 socklen_t addrLen) {
    // printf("server: receive request from client\n");
    printf("\nreceive msgD\n");

    unsigned char msgD[BUFSIZE], pkgMsgD[BUFSIZE];
    generateSubKey(cliTgsKey);

    decryption(buf, msgD);

    // printf("%ld", strlen(buf));

    printf("msgD: ");
    for (int i = 0; buf[i] != '\0'; i++) {
        printf("%02x", buf[i]);
    }
    printf("\n");

    // printf("buf: %s\n", buf);
    printf("cliTgsKEY: %s\n", cliTgsKey);
```

```
    printf("msgD after dec: %s\n", msgD);

    unsigned char timestamp[BUFSIZE];
    strncpy(clientID, msgD, 8);
    clientID[8] = 0;
    strncpy(timestamp, msgD + 8, 10);
    timestamp[10] = 0;
    printf("clientID: %s\n", clientID);
    printf("timestamp: %s\n", timestamp);

    printf("\nsend msgE\n");
    unsigned char msgE[BUFSIZE], enE[BUFSIZE], sendE[BUFSIZE];
    pkgMsgE(msgE, getCliSsKey(), &cliAddr);
    generateSubKey(SSKEY);
    encryption(msgE, enE);

    strcpy(sendE, serviceID);
    strcat(sendE, enE);

    printf("ST before enc: %s\n", msgE);
    // printf("msgE: %s\n", sendE);
    printf("msgE: ");
    for (int i = 0; sendE[i] != '\0'; i++) {
        printf("%02x", sendE[i]);
    }
    printf("\n");

    int len;
    len = send(connfd, sendE, BUFSIZE - 1, 0);
    if (len < 0) {
        fprintf(stderr, "send error: %s\n", strerror(errno));
        return errno;
    }

    printf("\nsend msgF\n");

    unsigned char msgF[BUFSIZE];
    generateSubKey(cliTgsKey);
    encryption(getCliSsKey(), msgF);

    len = send(connfd, msgF, BUFSIZE - 1, 0);
    if (len < 0) {
        fprintf(stderr, "send error: %s\n", strerror(errno));
        return errno;
    }

    printf("cliSsKey: %s\n", getCliSsKey());

    return 0;
}

unsigned char *getCliSsKey() { return cliSsKey; }

unsigned char *pkgMsgE(char *msgE, unsigned char *cliSsKey,
                       struct sockaddr *cliAddr) {
    strcpy(msgE, clientID);
    strcat(msgE, cliAddr->sa_data);
    for (int i = 0; i < 14 - strlen(cliAddr->sa_data); i++) {
```

```
        strcat(msgE, "0");
    }

    time_t validate;
    time(&validate);
    validate += 10 * 60;
    unsigned char timebuffer[50];
    sprintf(timebuffer, "%ld", validate);
    strcat(msgE, timebuffer);

    printf("msgE validity: %s\n", timebuffer);

    strcat(msgE, cliSsKey);
    return msgE;
}
```

SS

```
#include <errno.h>
#include <memory.h>
#include <stdbool.h>
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include <sys/socket.h>
#include <time.h>
#include <unistd.h>

#include "des.h"
#include "server-utils.h"
#include "utils.h"

#define BUFSIZE 1024

unsigned char SSKEY[] = "76543210";
unsigned char cliSsKey[BUFSIZE];
unsigned char clientID[BUFSIZE];

int responseReqE(unsigned char *buf, int n, int connfd, struct sockaddr cliAddr,
                 socklen_t addrLen);
int responseReqG(unsigned char *buf, int n, int connfd, struct sockaddr cliAddr,
                 socklen_t addrLen);

unsigned char *pkgMsgH(unsigned char *msgH, unsigned char *timestamp);

int main(int argc, char const *argv[]) {
    printf("SsKey: %s\n\n", SSKEY);

    int listenfd = createServSocket(SSPORT);

    printf("======waiting for client's request======\n");
    while (1) {
        struct sockaddr cliAddr;
        socklen_t addrLen;
        int connfd = accept(listenfd, &cliAddr, &addrLen);
        if (connfd == -1) {
            fprintf(stderr, "accept error: %s", strerror(errno));
```

```
            continue;
        }
        unsigned char msgE[BUFSIZE];
        int n = recv(connfd, msgE, BUFSIZE - 1, 0);
        msgE[n] = '\0';
        if (n > 0) responseReqE(msgE, n, connfd, cliAddr, addrLen);

        // connfd = accept(listenfd, &cliAddr, &addrLen);
        // if (connfd == -1) {
        //     fprintf(stderr, "accept error: %s", strerror(errno));
        //     continue;
        // }
        unsigned char msgG[BUFSIZE];
        n = recv(connfd, msgG, BUFSIZE - 1, 0);
        msgG[n] = '\0';
        if (n > 0) responseReqG(msgG, n, connfd, cliAddr, addrLen);

        close(connfd);

        printf("Success!\n");
    }

    close(listenfd);
    return 0;
}

int responseReqE(unsigned char *buf, int n, int connfd, struct sockaddr cliAddr,
                 socklen_t addrLen) {
    printf("server: receive request from client\n");
    printf("\nreceive msgE\n");

    unsigned char st[BUFSIZE], msgE[BUFSIZE];
    strcpy(st, buf + 6);
    generateSubKey(SSKEY);

    decryption(st, msgE);

    // printf("%ld", strlen(buf));

    printf("msgE: ");
    for (int i = 0; buf[i] != '\0'; i++) {
        printf("%02x", buf[i]);
    }
    printf("\n");

    // printf("msgE: %s\n", buf);
    // printf("st: %s\n", st);
    printf("SSKEY: %s\n", SSKEY);
    printf("ST after dec: %s\n", msgE);

    unsigned char timebuffer[50];
    memset(timebuffer, 0, sizeof(timebuffer));
    strncpy(timebuffer, msgE + 22, 10);
    timebuffer[10] = 0;
    printf("ST validity: %s\n", timebuffer);
    time_t start = atol(timebuffer);
    time_t end;
    time(&end);
```

```
    printf("nowtime: %ld\n", end);
    double cost = difftime(start, end);
    if (cost < 0) {
        printf("\nRefuse: Validate is out of date.\n");
        return 0;
    }

    strcpy(cliSsKey, msgE + 32);

    printf("cliSsKey: %s\n", cliSsKey);

    return 0;
}

int responseReqG(unsigned char *buf, int n, int connfd, struct sockaddr cliAddr,
                 socklen_t addrLen) {
    printf("\nreceive msgG\n");

    unsigned char msgG[BUFSIZE], pkgMsgD[BUFSIZE];
    generateSubKey(cliSsKey);

    decryption(buf, msgG);

    // printf("%ld", strlen(buf));

    printf("msgG: ");
    for (int i = 0; buf[i] != '\0'; i++) {
        printf("%02x", buf[i]);
    }
    printf("\n");

    // printf("buf: %s\n", buf);
    printf("cliSsKey: %s\n", cliSsKey);
    printf("msgG after dec: %s\n", msgG);

    unsigned char timestamp[50];
    strncpy(clientID, msgG, 8);
    clientID[8] = 0;
    strcpy(timestamp, msgG + 8);

    printf("clientID: %s\n", clientID);
    printf("msgG timestamp: %s\n", timestamp);

    printf("\nsend msgH\n");

    unsigned char msgH[BUFSIZE], sendH[BUFSIZE];
    pkgMsgH(msgH, timestamp);
    generateSubKey(cliSsKey);
    encryption(msgH, sendH);

    printf("msgH before enc: %s\n", msgH);

    int len;
    len = send(connfd, sendH, BUFSIZE - 1, 0);
    if (len < 0) {
        fprintf(stderr, "send error: %s\n", strerror(errno));
        return errno;
    }
```

```
    return 0;
}

unsigned char *pkgMsgH(unsigned char *msgH, unsigned char *timestamp) {
    strcpy(msgH, clientID);

    unsigned char timebuffer[50];
    memset(timebuffer, 0, sizeof(timebuffer));
    strncpy(timebuffer, timestamp, 10);
    timebuffer[10] = 0;

    printf("msgG timestamp: %s\n", timebuffer);

    time_t start = atol(timebuffer);
    start++;
    memset(timebuffer, 0, sizeof(timebuffer));
    sprintf(timebuffer, "%ld", start);
    strcat(msgH, timebuffer);

    printf("msgH timestamp: %s\n", timebuffer);

    return msgH;
}
```

编译运行结果

注意：程序在Linux系统下编译运行，书写代码的系统是Ubuntu 18.04.5 LTS

注意：由于并无标准规范，所以报文中打包的信息是直接按序连接的；

注意：本程序目的是为了演示Kerberos流程，并未实现用户系统，所以系统中的用户ID等信息是写死在代码中的，用户名到client密钥的映射是直接完成的；

编译

进入到kerberos文件夹，在终端运行`make`指令即可完成编译；

运行

在`bin`文件夹下依次运行四个可执行文件即可；

注意：client文件需要最后一个启动；

运行结果及解释

每个程序运行时会在开头显示程序已知的信息，然后显示`send XXX`、`receive XXX`提示其运行流程，并显示其接收到的或生成的或使用的关键信息；

注意：由于消息均是被加密过的，所以以十六进制（`%02x`）展示消息内容；

各个程序依次运行后，输出与解释如下

下述截图中消息密文可以相互验证，说明消息传输、消息处理是准确无误的。

Client

开头显示的`username`、`clientKey`等是已知的信息；

后面可以看到接收消息A、B，发送消息C、D，接收消息E、F，发送消息E、G，接收消息H；

```
nonoli@nonoli-hp:~/Homework/info-safe/sysu-infoSecurity/hw4/kerberos/bin$ ./client
username: username
clientID: username
serviceID: 000000
clientKey: 12345678


send request

receive msgA & msgB
msgA: be272a9db2272fdbfeb959b7d4642fcb
msgB: 4d3711fd37b70deb6f44b03056bd56a6bd45537c1ffcf22bdc64fd53d9a62524ad7ca247619fe2afcb12a48c6b54c99e
cliTgsKey: 23456789

send msgC
msgC: 3030303030304d3711fd37b70deb6f44b03056bd56a6bd45537c1ffcf22bdc64fd53d9a62524ad7ca247619fe2afcb12a48c6b54c99e

send msgD
msgD timestamp: 1607603287
msgD before enc: username1607603287
msgD: 4a986f15dc936efd3214bb9de79c0e24949eb08ca93e9e18

receive msgE & msgF
msgE: 303030303030e255dbdcbf5165132162ae2b60390f9bcfc41003f5fbceefb612e49cd59dc9771b70636b92b17f34c1d85592a3be03a9
msgF: e4678d9c64b2b202b0ad95645c1bbae6
cliSsKey: 98765432

send msgE
msgE: 303030303030e255dbdcbf5165132162ae2b60390f9bcfc41003f5fbceefb612e49cd59dc9771b70636b92b17f34c1d85592a3be03a9

send msgG
msgG timestamp: 1607603287
msgG before enc: username1607603287
msgG: 4aeca6b8fc67c1095f6550b9e06fbc978c90a53a8d1a3cc0

receive msgH
msgH: 4aeca6b8fc67c1095f6550b9e06fbc9748f52189e9f0581c
msgH after dec: username1607603288▒▒▒▒▒▒▒
timestamp: 1607603288
Success!
```

AS

开头显示的`TgsKey`、`clientKey`、`cliTgsKey`是服务器已知的信息；

后面可以看到消息A的原文和密文，消息B的原文和密文；

消息B的原文开头8位是clientID、后面14位是client address及补0，后面10位是票据有效期，后面8位是`clientKey`（23456789）；

```
nonoli@nonoli-hp:~/Homework/info-safe/sysu-infoSecurity/hw4/kerberos/bin$ ./auth
-server
TgsKey: 87654321
clientKey: 12345678
cliTgsKey: 23456789

======waiting for client's request======
server: receive request from client (username)

send msgA
cliTgsKey: 23456789
msgA: be272a9db2272fdbfeb959b7d4642fcb

send msgB
msgB before enc: username↗◆0000000000160760388723456789
msgB: 4d3711fd37b70deb6f44b03056bd56a6bd45537c1ffcf22bdc64fd53d9a62524ad7ca24761
9fe2afcb12a48c6b54c99e
Success!
^C
```

TGS

开头显示的`TgsKey`、`SsKey`、`cliSsKey`是服务器已知的信息；

后面可以看到接收消息C、D，发送消息E、F；

其中消息D解密后最后出现的乱码是DES加密的补0，是正常现象；消息C和消息D开头都能看到303030303030，这是服务ID的2位十六进制显示；

```
nonoli@nonoli-hp:~/Homework/info-safe/sysu-infoSecurity/hw4/kerberos/bin$ ./tgs-server
TgsKey: 87654321
SsKey: 76543210
cliSsKey: 98765432

======waiting for client's request======
server: receive request from client

receive msgC
msgC: 3030303030304d3711fd37b70deb6f44b03056bd56a6bd45537c1ffcf22bdc64fd53d9a62524ad7ca247619fe2afcb12a48c6b54c99e
serviceID: 000000
TGSKEY: 87654321
msgB after dec: username↝◆00000000001607603887234567890
msgB validity: 1607603887
nowtime: 1607603287

receive msgD
msgD: 4a986f15dc936efd3214bb9de79c0e24949eb08ca93e9e18
cliTgsKEY: 23456789
msgD after dec: username1607603287▒▒▒▒▒▒▒
clientID: username
timestamp: 1607603287

send msgE
msgE validity: 1607603887
ST before enc: username◆f00000000000160760388798765432
msgE: 303030303030e255dbdcbf5165132162ae2b60390f9bcfc41003f5fbceefb612e49cd59dc9771b70636b92b17f34c1d85592a3be03a9

send msgF
cliSsKey: 98765432
Success!
^C
```

SS

开头显示的`SsKey`是服务器已知的信息；

后面可以看到接收消息E、G，发送消息H；

其中可以注意到服务票据和当前时间被打印出来、消息G时间戳和消息H时间戳被打印出来，这些时间相关的都被正确校验处理；

```
nonoli@nonoli-hp:~/Homework/info-safe/sysu-infoSecurity/hw4/kerberos/bin$ ./ss-server
SsKey: 76543210

======waiting for client's request======
server: receive request from client

receive msgE
msgE: 303030303030e255dbdcbf5165132162ae2b60390f9bcfc41003f5fbceefb612e49cd59dc9771b70636b92b17f34c1d85592a3be03a9
SSKEY: 76543210
ST after dec: username◆f00000000000160760388798765432
ST validity: 1607603887
nowtime: 1607603287
cliSsKey: 98765432

receive msgG
msgG: 4aeca6b8fc67c1095f6550b9e06fbc978c90a53a8d1a3cc0
cliSsKey: 98765432
msgG after dec: username1607603287▒▒▒▒▒▒▒
clientID: username
msgG timestamp: 1607603287▒▒▒▒▒▒▒

send msgH
msgG timestamp: 1607603287
msgH timestamp: 1607603288
msgH before enc: username1607603288
Success!
^C
```

验证用例

注意：由于没有用户系统，所以重复运行程序的显示唯一会变化的是时间戳、服务票据有效期，而其它信息基本不会发生改变；

Client

```
nonoli@nonoli-hp:~/Homework/info-safe/sysu-infoSecurity/hw4/kerberos/bin$ ./client
username: username
clientID: username
serviceID: 000000
clientKey: 12345678


send request

receive msgA & msgB
msgA: be272a9db2272fdbfeb959b7d4642fcb
msgB: 4d3711fd37b70deb2551ca6bbba338cdbd45537c1ffcf22bbed6093fca6c964aad7ca247619fe2afcb12a48c6b54c99e
cliTgsKey: 23456789

send msgC
msgC: 3030303030304d3711fd37b70deb2551ca6bbba338cdbd45537c1ffcf22bbed6093fca6c964aad7ca247619fe2afcb12a48c6b54c99e

send msgD
msgD timestamp: 1607604768
msgD before enc: username1607604768
msgD: 4a986f15dc936efd0ff319a0354f2858045eb5251d52d18b

receive msgE & msgF
msgE: 303030303030e255dbdcbf51651307e40c7cca2219e2cfc41003f5fbceefeb6b67eaf97d2e7a1b70636b92b17f34c1d85592a3be03a9
msgF: e4678d9c64b2b202b0ad95645c1bbae6
cliSsKey: 98765432

send msgE
msgE: 303030303030e255dbdcbf51651307e40c7cca2219e2cfc41003f5fbceefeb6b67eaf97d2e7a1b70636b92b17f34c1d85592a3be03a9

send msgG
msgG timestamp: 1607604768
msgG before enc: username1607604768
msgG: 4aeca6b8fc67c1090f9413de7220bb109c7d877dd86fa6f1

receive msgH
msgH: 4aeca6b8fc67c1090f9413de7220bb106d7f3111cd493411
msgH after dec: username1607604769[illegible]
timestamp: 1607604769
Success!
```

AS

```
nonoli@nonoli-hp:~/Homework/info-safe/sysu-infoSecurity/hw4/kerberos/bin$ ./auth
-server
TgsKey: 87654321
clientKey: 12345678
cliTgsKey: 23456789

======waiting for client's request======
server: receive request from client (username)

send msgA
cliTgsKey: 23456789
msgA: be272a9db2272fdbfeb959b7d4642fcb

send msgB
msgB before enc: username[illegible]0000000001607605368234567899
msgB: 4d3711fd37b70deb2551ca6bbba338cdbd45537c1ffcf22bbed6093fca6c964aad7ca24761
9fe2afcb12a48c6b54c99e
Success!
^C
```

TGS

```
nonoli@nonoli-hp:~/Homework/info-safe/sysu-infoSecurity/hw4/kerberos/bin$ ./tgs-server
TgsKey: 87654321
SsKey: 76543210
cliSsKey: 98765432

======waiting for client's request======
server: receive request from client

receive msgC
msgC: 3030303030304d3711fd37b70deb2551ca6bbba338cdbd45537c1ffcf22bbed6093fca6c964aad7ca247619fe2afcb12a48c6b54c99e
serviceID: 000000
TGSKEY: 87654321
msgB after dec: username◆▯◆00000000001607605368234567 89
msgB validity: 1607605368
nowtime: 1607604768

receive msgD
msgD: 4a986f15dc936efd0ff319a0354f2858045eb5251d52d18b
cliTgsKEY: 23456789
msgD after dec: username1607604768▯▯▯▯▯▯▯▯
clientID: username
timestamp: 1607604768

send msgE
msgE validity: 1607605368
ST before enc: username◆N00000000000160760536898765432
msgE: 303030303030e255dbdcbf51651307e40c7cca2219e2cfc41003f5fbceefeb6b67eaf97d2e7a1b70636b92b17f34c1d85592a3be03a9

send msgF
cliSsKey: 98765432
Success!
^C
```

SS

```
nonoli@nonoli-hp:~/Homework/info-safe/sysu-infoSecurity/hw4/kerberos/bin$ ./ss-server
SsKey: 76543210

======waiting for client's request======
server: receive request from client

receive msgE
msgE: 303030303030e255dbdcbf51651307e40c7cca2219e2cfc41003f5fbceefeb6b67eaf97d2e7a1b70636b92b17f34c1d85592a3be03a9
SSKEY: 76543210
ST after dec: username◆N00000000000160760536898765432
ST validity: 1607605368
nowtime: 1607604768
cliSsKey: 98765432

receive msgG
msgG: 4aeca6b8fc67c1090f9413de7220bb109c7d877dd86fa6f1
cliSsKey: 98765432
msgG after dec: username1607604768▯▯▯▯▯▯▯▯
clientID: username
msgG timestamp: 1607604768▯▯▯▯▯▯▯▯

send msgH
msgG timestamp: 1607604768
msgH timestamp: 1607604769
msgH before enc: username1607604769
Success!
^C
```

实验总结

本次实验主要是完成Kerberos的完整流程，涉及到网络通信和DES加解密的知识，通过在四个进程client、AS、TGS、SS间的通信来完成，整体难度中等，帮助我全面的了解了Kerberos这一认证模型。