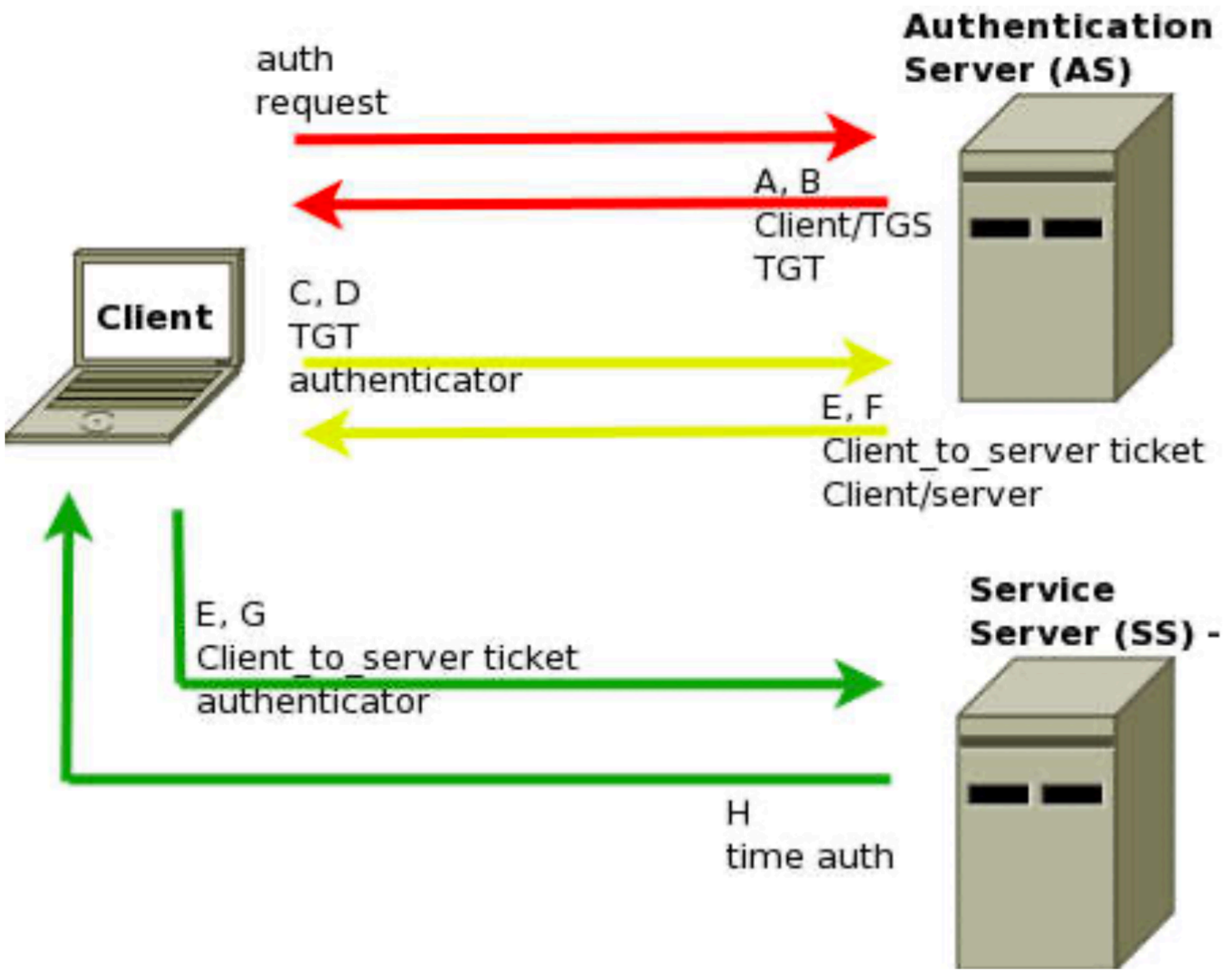
IS_Assignment 4 Kerberos

作业内容：在单主机并发环境下构建实现一个Kerberos认证模型

原理概述

- The client (machine) authenticates to the AS once using a long-term shared secret and receives a TGT from the AS.
- Later, when the client wants to contact some SS, it can (re)use this TGT to get additional tickets STs from TGS for SS, without resorting to using the shared secret. These STs can be used to prove authentication to SS.
  - AS = Authentication Server
  - TGT = Ticket-Granting Ticket
  - TGS = Ticket-Granting Server
  - ST = Service Ticket
  - SS = Service Server


auth request
Authentication Server (AS)
A, B Client/TGS TGT
Client
C, D TGT authenticator
E, F Client_to_server ticket Client/server
E, G Client_to_server ticket authenticator
Service Server (SS) -
H time auth

实验设计

分别在authserver.c, server.c, client.c 三个文件下实现验证服务器，服务器和客户端的内容。

大致流程如下：

- 客户端向AS发送as_req请求
- AS用as_rep响应客户端
- 客户端向服务器发送s_req消息
- 身份验证
- 服务器用s_rep响应客户端

另外，数据的传输使用pdata, pkt相关，加解密使用openssl中的aes。

上面提到的请求、响应以及验证等的数据结构设计如下：

```
struct as_req {
    short type;
    unsigned char client_name[40];
    unsigned char server_name[40];
    time_t time_stamp1;
};

struct as_rep {
    short type;
    short cred_length;
    unsigned char cred[CRED_SIZE];
};

struct auth{
    unsigned char client_name[40];
    time_t time_stamp3;
};

struct s_req{
    short type;
    short ticket_length;
    short auth_length;
    unsigned char ticket_enc[TICKET_SIZE];
    unsigned char auth[64];
};

struct s_rep{
    short type;
    short nonce_length;
    unsigned char nonce[16];
};

struct ticket{
  unsigned char AES_key[32];
```

```
    unsigned char client_name[40];
    unsigned char server_name[40];
    time_t time_stamp;
    int life_time;
};

struct credential {
  unsigned char AES_key[32];
    unsigned char server_name[40];
    time_t time_stamp;
    int life_time2;
    int ticket_length;
    unsigned char ticket_enc[TICKET_SIZE];
};

struct pkt {
    short type;
    short pkt_length;
    unsigned char pkt_enc[976];
};

struct pdata {
    short     type;
    short     packet_length;
    short     pid;
    unsigned char    data[960];
};
```

通信相关使用socket，如server中：

```
    // 创建socket
    int sock;
    if ((sock = socket(PF_INET, SOCK_DGRAM, IPPROTO_UDP)) < 0)
        printError("socket() failed");

    // 构造server完整地址
    struct sockaddr_in server_addr;
    struct sockaddr_in client_addr;
    memset(&server_addr, 0, sizeof(server_addr));
    server_addr.sin_family = AF_INET;
    server_addr.sin_addr.s_addr = htonl(INADDR_ANY);
    server_addr.sin_port = htons(server_port);

    // bind
    struct s_req s_request;
    if (bind(sock, (const struct sockaddr *) &server_addr, sizeof(server_addr))
< 0)
        printError("bind() failed");
    // 接收client请求
    unsigned int client_addr_len = sizeof(client_addr);
    int recv_msg_size;
    if ((recv_msg_size = recvfrom(sock, &s_request, sizeof(struct s_req), 0,
(struct sockaddr *) &client_addr, &client_addr_len)) < 0){
        printError("recvfrom() failed");
    }else{
        printf("recvfrom() succeed, from client.\n");
    }
```

加密解密：

```
// 使用key和iv加密长度为plaintext_len的plaintxt，并将密文放入ciphertext，返回ciphertext长度。
int encrypt(unsigned char *plaintext, int plaintext_len, unsigned char *key,
            unsigned char *iv, unsigned char *ciphertext);

//使用key和iv解密长度为ciphertext_len的ciphertext，并将解密结果存入plaintext，返回plaintext长度。
int decrypt(unsigned char *ciphertext, int ciphertext_len, unsigned char *key,
            unsigned char *iv, unsigned char *plaintext);
```

C语言源代码

openssl version：1.0.2g

存放在src文件夹下：

- header.h、header.c 头文件代码，对应各种include以及数据结构等
- authserver.c、server.c、client.c 对应AS、server、client

运行方法

- make
- 开三个terminal分别跑，先跑server和authserver，再跑client
- ./server server端口 server名字

  ./authserver auth端口 server名字 serverkey client名字 clientkey

  ./client auth_ip auth端口 server_ip server端口 server名字 client名字 clientkey

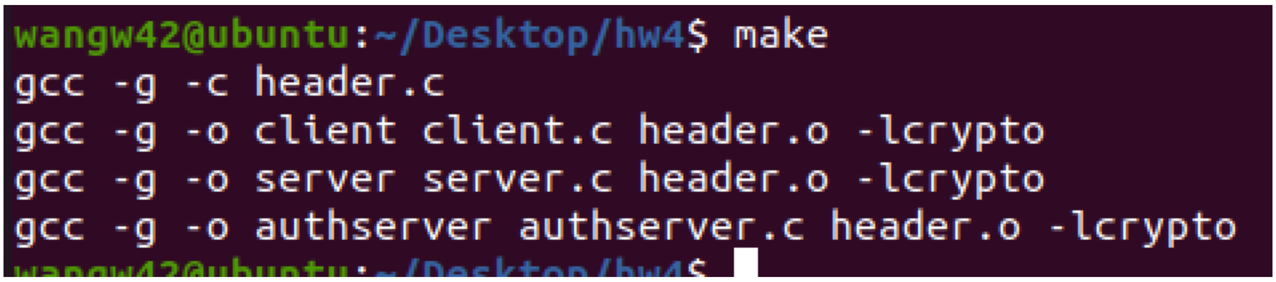
- 示例如下：

  ./server 4001 def

./authserver 4000 alice abc bob def

./client 127.0.0.1 4000 abc 127.0.0.1 4001 alice bob

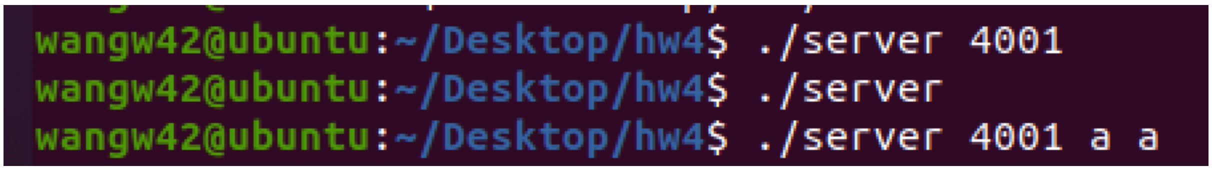
编译运行结果



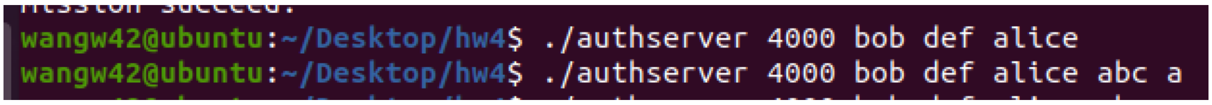
```
wangw42@ubuntu:~/Desktop/hw4$ make
gcc -g -c header.c
gcc -g -o client client.c header.o -lcrypto
gcc -g -o server server.c header.o -lcrypto
gcc -g -o authserver authserver.c header.o -lcrypto
wangw42@ubuntu:~/Desktop/hw4$
```

测试用例

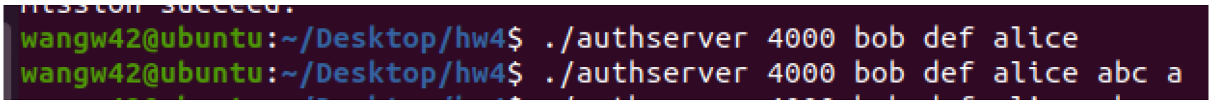
- 参数输入不正确，直接退出



```
wangw42@ubuntu:~/Desktop/hw4$ ./server 4001
wangw42@ubuntu:~/Desktop/hw4$ ./server
wangw42@ubuntu:~/Desktop/hw4$ ./server 4001 a a
```

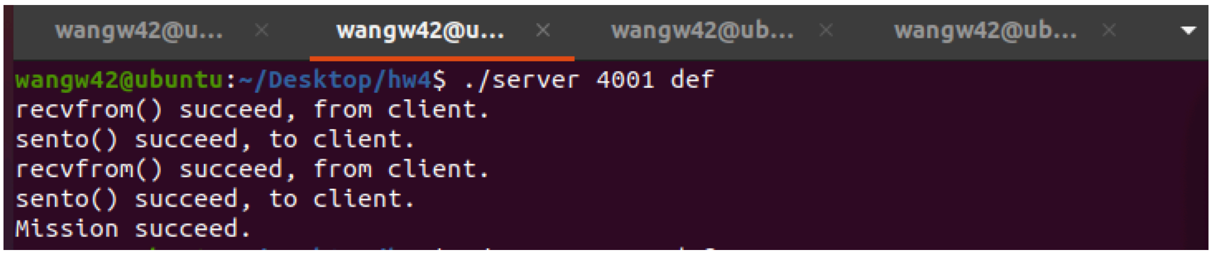


```
wangw42@ubuntu:~/Desktop/hw4$ ./authserver 4000 bob def alice
wangw42@ubuntu:~/Desktop/hw4$ ./authserver 4000 bob def alice abc a
```

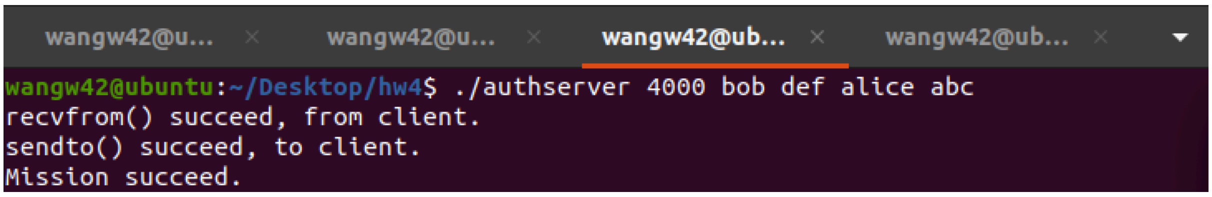


```
wangw42@ubuntu:~/Desktop/hw4$ ./authserver 4000 bob def alice
wangw42@ubuntu:~/Desktop/hw4$ ./authserver 4000 bob def alice abc a
```

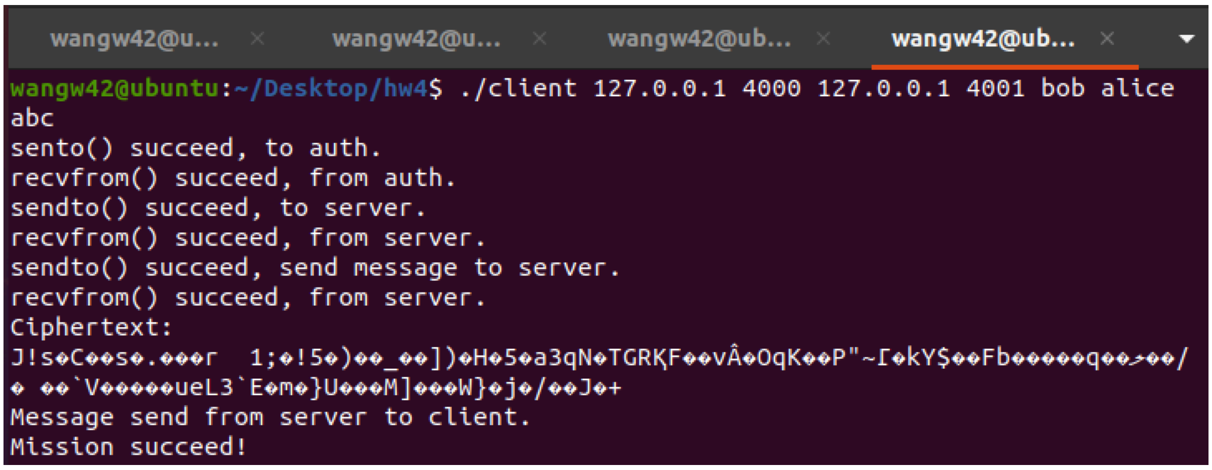
- 正确测试结果：



```
wangw42@ubuntu:~/Desktop/hw4$ ./server 4001 def
recvfrom() succeed, from client.
sento() succeed, to client.
recvfrom() succeed, from client.
sento() succeed, to client.
Mission succeed.
```



```
wangw42@ubuntu:~/Desktop/hw4$ ./authserver 4000 bob def alice abc
recvfrom() succeed, from client.
sendto() succeed, to client.
Mission succeed.
```



```
wangw42@ubuntu:~/Desktop/hw4$ ./client 127.0.0.1 4000 127.0.0.1 4001 bob alice
abc
sento() succeed, to auth.
recvfrom() succeed, from auth.
sendto() succeed, to server.
recvfrom() succeed, from server.
sendto() succeed, send message to server.
recvfrom() succeed, from server.
Ciphertext:
J!s�C��s�.���r  1;�!5�)��_��])�H�5�a3qN�TGRĶF��vÂ�OqK��P"~Ɪ�kY$��Fb�����q��↣��/
� ��`V�����ueL3`E�m�}U���M]���W}�j�/��J�+
Message send from server to client.
Mission succeed!
```