THE UNIVERSITY OF ARIZONA

CSC 552 Advanced Operating Systems

Project Report

Securing Data from Illegal Access on Cloud

Tianyu Liang
Yang Liu
Yash Naik
Yuan Zhang

ABSTRACT

Cloud service has become more and more common for its convenience and efficiency. However, the secure issue has also become more and more important. Many people have put forward solutions for the service providers to secure the data, they think of many ways for the server to preserve the date securely. In our program, we come up with the specific situation: what if the server itself is not fully trusted? Following this thought, we put forward a method using AES Encryption file sharing and Diffie Hellman Key Exchange to project information from illegal access and untrusted server.

1. INTRODUCTION

Nowadays as the network becomes increasingly easy to access and the speed becomes faster, more and more people choose to use cloud to store their data.

Cloud services means services made available to users on demand via the Internet from a cloud computing provider's servers as opposed to being provided from a company's own on-premises servers. Cloud services are designed to provide easy, scalable access to applications, resources and services, and are fully managed by a cloud services provider.

For individuals, after using the cloud service, they don't need to carry a hard-driver with them and be afraid of losing it. For small business, they can reap the benefits of not having to deploy physical infrastructure like file and e-mail servers, storage systems or shrink-wrapped software. Cloud services also provide entrepreneurs, SOHOs, and mom-and-pop outfits access to sophisticated technology without the need of an IT consultant or tech worker on the payroll.

There are many companies providing cloud service, IBM, Amazon, Microsoft are trying to persuade small business to use their cloud service, and there are also numerous cloud service provider for individuals, such as google driver, dropbox and so on.

However, although the cloud service is developing rapidly, there remain several threats to it. According to the Cloud Security Alliance, these threats includes data breaches, data loss, account or service traffic hijacking, insecure APIs, denial of services, malicious insiders, abuse of cloud services, insufficient due diligence and shared technology. Among these threats, the key lies in the security of the sensitive informations. A cloud user stores their sensitive data to the cloud. Thus it becomes cloud service provider’s responsibility to establish secure communication mechanism. It should provide secure channels for sending and receiving data and also for storing data.

In our program, we try to secure data from illegal access on cloud, this illegal access include access from untrusted servers and from unauthorized clients. Our solution is that the clients encrypt the files locally and upload them to cloud, then exchange key with target clients, finally share target can download the files and decrypt. Through the process, neither the server nor other clients knows what is uploaded, thus we can assume the data is well protected.

2. BACKGROUND

More and more people are storing their files onto the cloud, even sensitive ones. While all cloud service providers (CSP) claim that they won’t look into user’s content in respect for privacy, it’s not valid since companies never publish their codes. Even though we cannot prove they are eavesdropping our information, we are pretty sure they are, or how could we receive so much spam in our email?

While some may say, we trust our CSPs, nor do I care if they read my privacy, unfortunately, CSPs can be compromised. In 2011 Sony’s database was breached by hackers and those gamers suffered a great loss by exposing their name, address, credit card number etc. to the malicious attackers. If the stored files are already encrypted, there’s no way you lose your privacy.

This brings us to focus on the insecure cloud concept. We try to protect user’s information from being read by any unauthorized unit.

3. RESEARCH

After did researches in the field, we found there are several main thoughts to deal with the security issue. First, some researchers proposed to take actions to secure the cloud network. However, most of them are still based on an assumption that the server administrator is trusted and reliable. For example, a thought is to set an automatic password distributor. But whoever controls the distributor will have the access to your file.

Then, a reasonable thought is to store encrypted file online and send the password through another secure channel. For instance, give it in person. But obviously, this is an inefficient way. If you want to achieve this through network, P2P communication is the only option. However, to build a secure communication in an insecure network, there are two common methods, D-H key exchange and RSA. Both of them are widely used. However, we think the DH key exchange has advantage in efficiency.

Thus, we think the best solution is to encrypt the file using AES algorithm, which is the most common and safe encryption method, and to transmit password using DH key exchange method via P2P.

4. DESIGNS AND IMPLEMENTATION

This is how to set up our experiment.

1. Use NFS to create a file sharing service (cloud), including one server and two linux clients. For convenience, we run both server and clients on virtual machine.

2. File owner encrypts file and upload it to cloud.

3. Physical ID identification

4. D-H Key Exchange using TCP communication.

5. Share target download file and decrypt it.

4.1 NFS Mount set up

We use standard NFS setup procedure to connect both clients to the file server, and let them share a common directory.

4.2 Interaction between two clients

After authentication is done, the file owner client1 starts a socket server program, which verifies the incoming socket client request and start DH key exchange with it.

If the file owner assumes this socket connection is safe and give the file key in plain text via this channel, then a malicious eavesdropper can compromise this by capturing and analyzing all the sent packets. In our case, the cloud can just be this eavesdropper.

That's why we choose to apply DH key exchange, which is secure against eavesdropping because the underlying idea is that, a key is generated from two random numbers, known to the two clients only. One generated by itself and the other unreadable by others.

The socket server program includes a parameter of the file key, which will be encrypted by the session key generated by DH key exchange.

The socket client program takes as parameters the address of server and the file name it wants to access. It will then ask the user to specify a path to store the file once the key is obtained. The client can run standard decryption algorithm to read the file's content.

5. EXPERIMENT RESULT

5.1 demo workflow

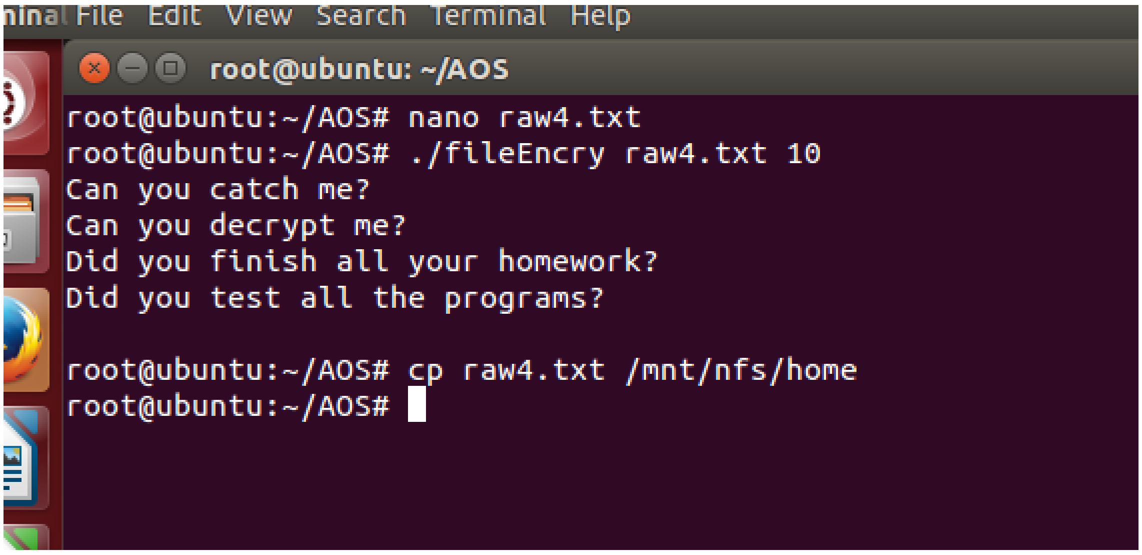
File names, Commands and useful notation:

1. raw4.txt -> user file to be uploaded
2. ./fileEncry <file_name> <keyvalue> -> command to encrypt file
3. ./server <key_value> <file_name> → To run tcp program on server side
4. ./client <server_ip_address> →To run tcp program client side
5. ./fileDecry <file_name> <key_value> -> To decrypt the file

Step 1: Encrypt the file and Put it on the cloud.

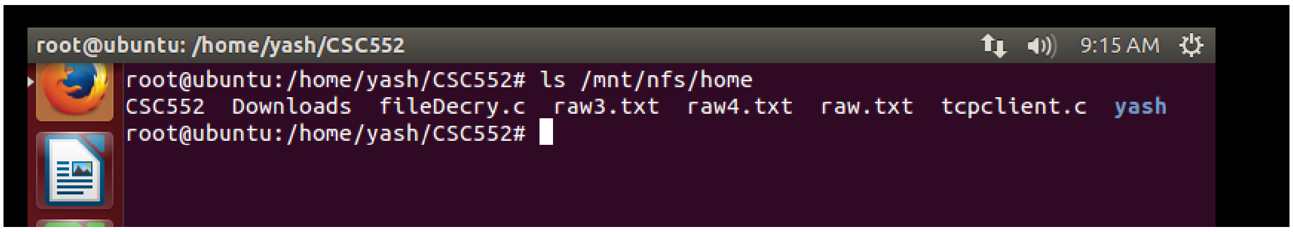
Command 1(Encryption): $ ./fileEncry raw4.txt 10

Command 2(Put it on the cloud): $ cp raw4.txt /mnt/nfs/home

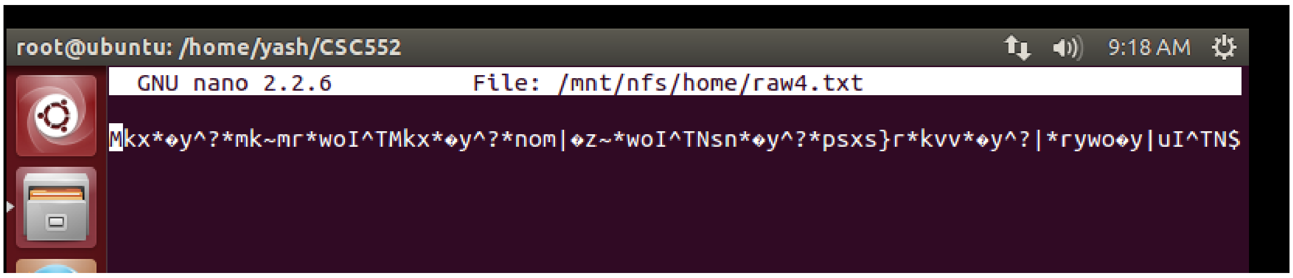


```
root@ubuntu: ~/AOS
root@ubuntu:~/AOS# nano raw4.txt
root@ubuntu:~/AOS# ./fileEncry raw4.txt 10
Can you catch me?
Can you decrypt me?
Did you finish all your homework?
Did you test all the programs?

root@ubuntu:~/AOS# cp raw4.txt /mnt/nfs/home
root@ubuntu:~/AOS#
```

Step 2: How do all others see the encrypted file on the shared folder



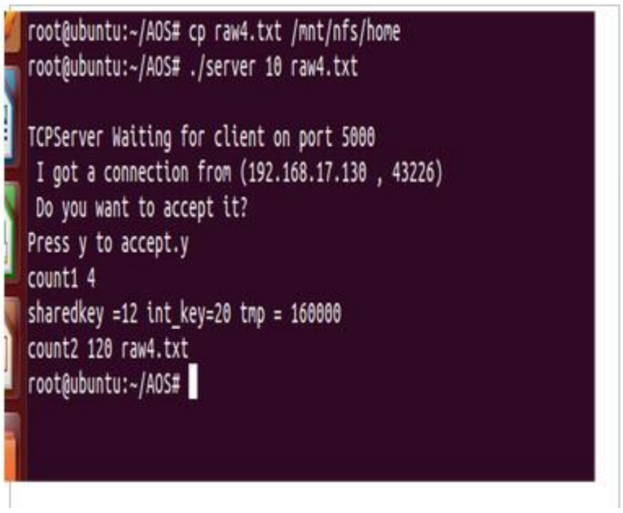
```
root@ubuntu: /home/yash/CSC552
root@ubuntu:/home/yash/CSC552# ls /mnt/nfs/home
CSC552  Downloads  fileDecry.c  raw3.txt  raw4.txt  raw.txt  tcpclient.c  yash
root@ubuntu:/home/yash/CSC552#
```



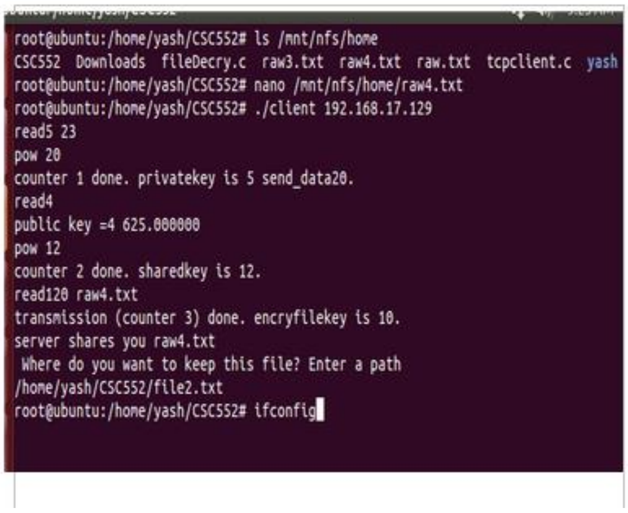
```
root@ubuntu: /home/yash/CSC552
  GNU nano 2.2.6            File: /mnt/nfs/home/raw4.txt

Mkx*�y^?*mk~mr*woI^TMkx*�y^?*nom|�z~*woI^TNsn*�y^?*psxs}r*kvv*�y^?|*rywo�y|uI^TN$
```

Step 3: Since the file is unreadable to any other user inside the network file system.We need to share the key with other users in order for the file to be readable to other users. For this we need to physically authenticate the each other using email, calling or even physically meeting. Once they are authenticated to each other then we can pair them. Pairing will mean that there wont be any need to authenticate each other again.

Step 4: Pairing and sharing



```
root@ubuntu:~/AOS# cp raw4.txt /mnt/nfs/home
root@ubuntu:~/AOS# ./server 10 raw4.txt

TCPServer Waiting for client on port 5000
 I got a connection from (192.168.17.130 , 43226)
 Do you want to accept it?
Press y to accept.y
count1 4
sharedkey =12 int_key=20 tmp = 160000
count2 120 raw4.txt
root@ubuntu:~/AOS# 
```



```
root@ubuntu:/home/yash/CSC552# ls /mnt/nfs/home
CSC552  Downloads  fileDecry.c  raw3.txt  raw4.txt  raw.txt  tcpclient.c  yash
root@ubuntu:/home/yash/CSC552# nano /mnt/nfs/home/raw4.txt
root@ubuntu:/home/yash/CSC552# ./client 192.168.17.129
read5 23
pow 20
counter 1 done. privatekey is 5 send_data20.
read4
public key =4 625.000000
pow 12
counter 2 done. sharedkey is 12.
read120 raw4.txt
transmission (counter 3) done. encryfilekey is 10.
server shares you raw4.txt
 Where do you want to keep this file? Enter a path
/home/yash/CSC552/file2.txt
root@ubuntu:/home/yash/CSC552# ifconfig
```

The fig shows the running of the tcp program both at server and client. Server is the machine which has the key of the file and client is the machine which needs the key to read the file.

As we can see here we see the Deffie-Hellman key exchange algorithm being used to pair the two devices and generate a shared secret key.

The shared key is “12”

Once the shared key is obtained the server of the tcp program will send the file key encrypting it using the shared key.

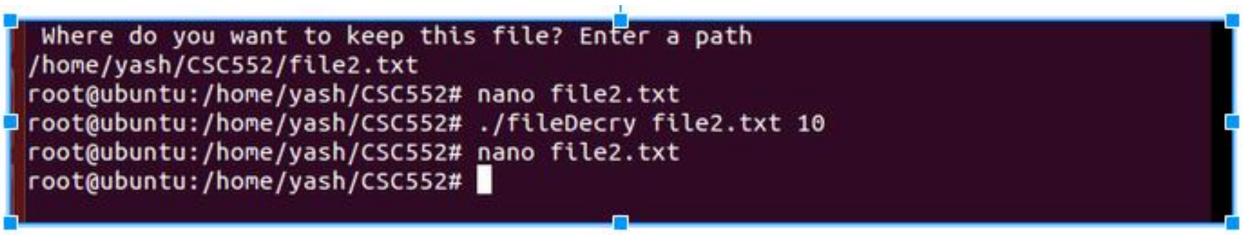
The file key is “10”

The server sends 10*12 =120

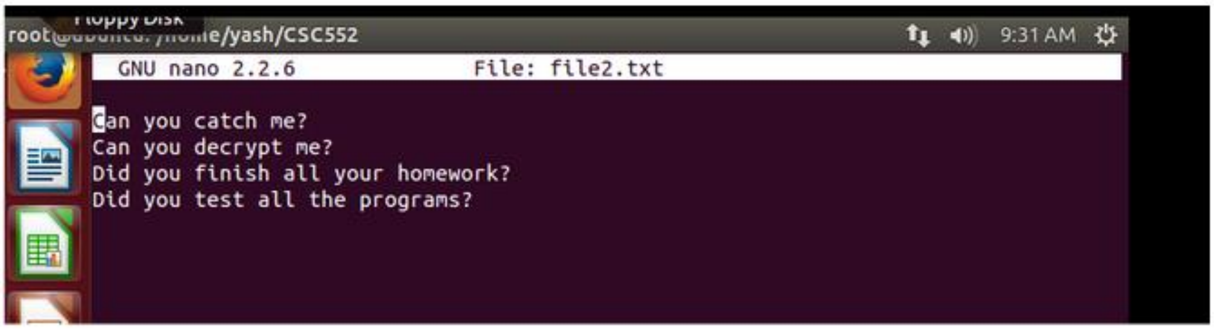
On the other side the client of the tcp program will decrypt the value “120” by dividing it by the shared key ie 120/12 =10 Thus the file key is obtained.

Step 5: Decrypt the file

Now using the file key the client can open and read the file. As shown in the figure below



```
 Where do you want to keep this file? Enter a path
/home/yash/CSC552/file2.txt
root@ubuntu:/home/yash/CSC552# nano file2.txt
root@ubuntu:/home/yash/CSC552# ./fileDecry file2.txt 10
root@ubuntu:/home/yash/CSC552# nano file2.txt
root@ubuntu:/home/yash/CSC552# 
```



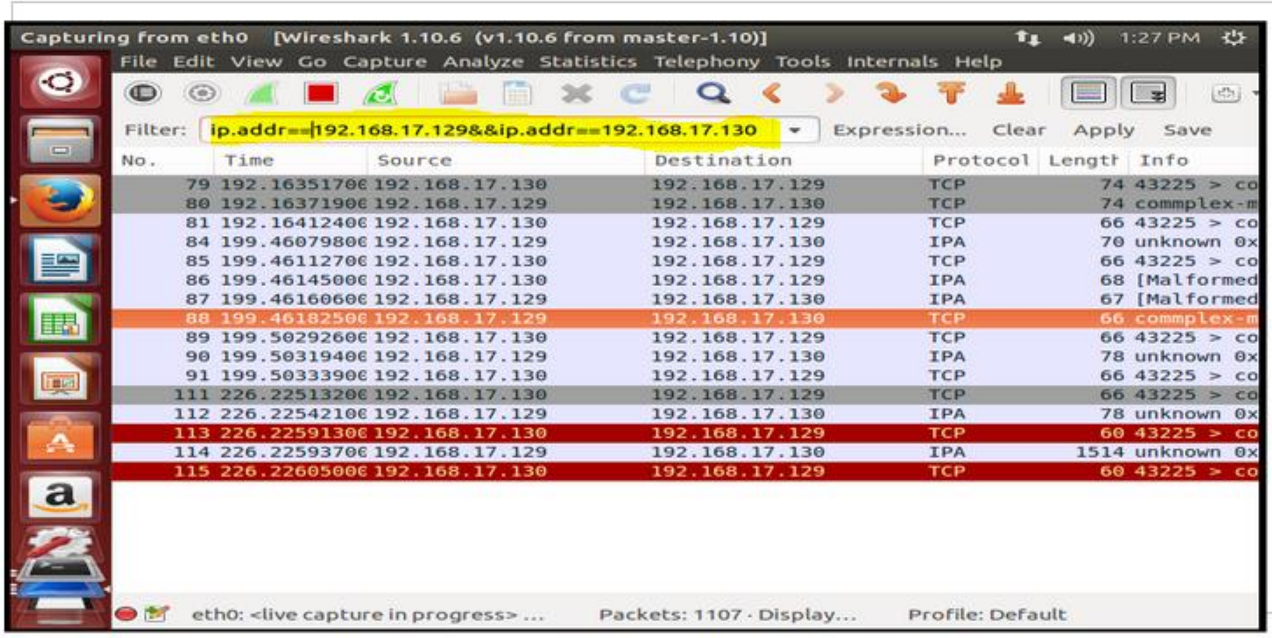
```
  GNU nano 2.2.6              File: file2.txt

Can you catch me?
Can you decrypt me?
Did you finish all your homework?
Did you test all the programs?
```

5.2 robustness against eavesdropper

To test that the cloud server can observe the communication we used “Wireshark”

software used for analyzing and reading packets send inside the network.i

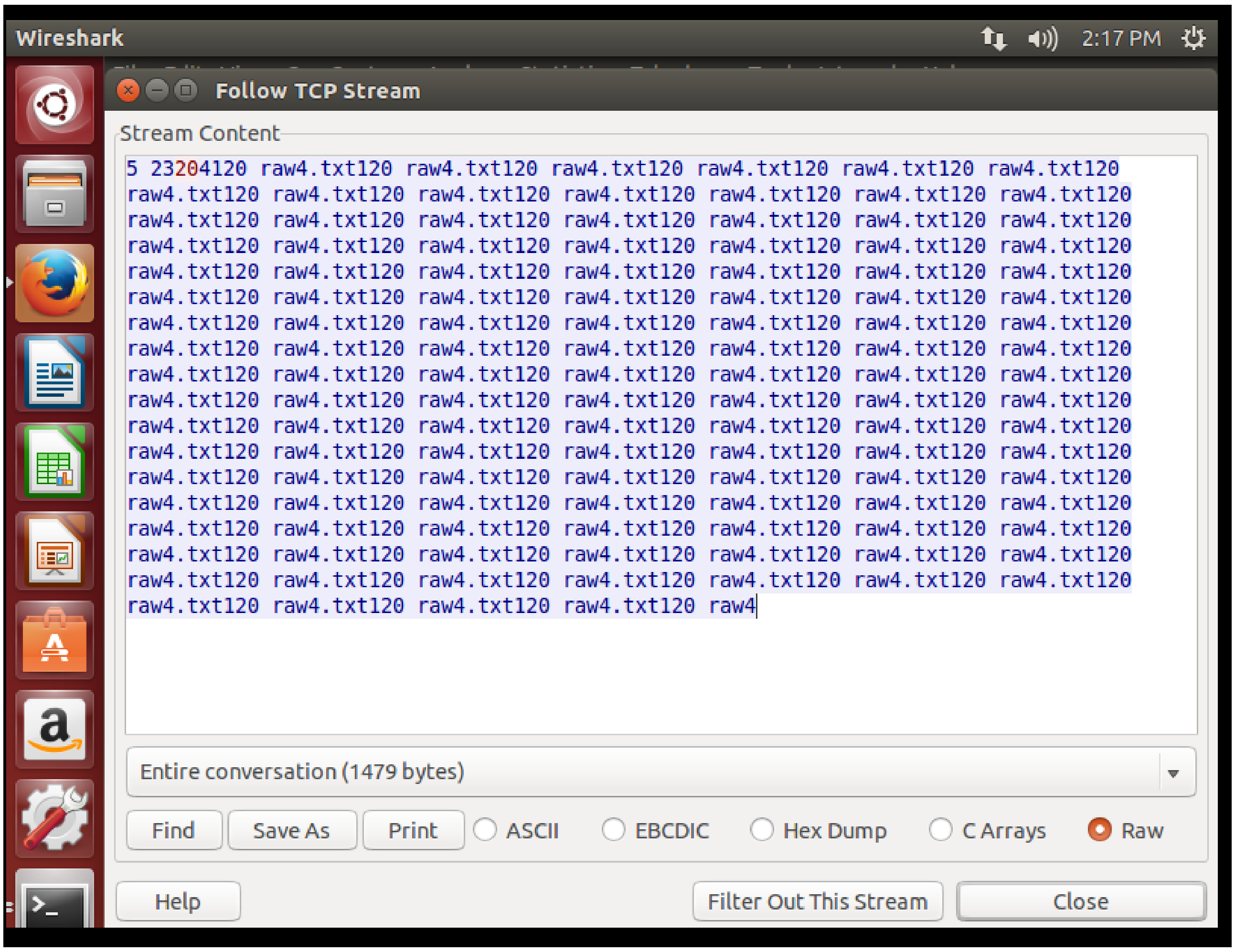
Step 1: Run the Wireshark inside the Cloud server machine. Set the filter to ip addresses of the two communicating devices.



```
Capturing from eth0   [Wireshark 1.10.6 (v1.10.6 from master-1.10)]
Filter: ip.addr==192.168.17.129&&ip.addr==192.168.17.130

No.  Time            Source          Destination     Protocol  Length Info
 79 192.1635170€ 192.168.17.130  192.168.17.129  TCP         74 43225 > co
 80 192.1637190€ 192.168.17.129  192.168.17.130  TCP         74 commplex-m
 81 192.1641240€ 192.168.17.130  192.168.17.129  TCP         66 43225 > co
 84 199.4607980€ 192.168.17.129  192.168.17.130  IPA         70 unknown 0x
 85 199.4611270€ 192.168.17.130  192.168.17.129  TCP         66 43225 > co
 86 199.4614500€ 192.168.17.130  192.168.17.129  IPA         68 [Malformed
 87 199.4616060€ 192.168.17.129  192.168.17.130  IPA         67 [Malformed
 88 199.4618250€ 192.168.17.129  192.168.17.130  TCP         66 commplex-m
 89 199.5029260€ 192.168.17.130  192.168.17.129  TCP         66 43225 > co
 90 199.5031940€ 192.168.17.129  192.168.17.130  IPA         78 unknown 0x
 91 199.5033390€ 192.168.17.130  192.168.17.129  TCP         66 43225 > co
111 226.2251320€ 192.168.17.130  192.168.17.129  TCP         66 43225 > co
112 226.2254210€ 192.168.17.129  192.168.17.130  IPA         78 unknown 0x
113 226.2259130€ 192.168.17.130  192.168.17.129  TCP         60 43225 > co
114 226.2259370€ 192.168.17.129  192.168.17.130  IPA       1514 unknown 0x
115 226.2260500€ 192.168.17.130  192.168.17.129  TCP         60 43225 > co

eth0: <live capture in progress> ...   Packets: 1107 · Display...   Profile: Default
```

Clearly, we see the cloud server can observe messages being passed over the network.

Step 2: The cloud tries to read the packets



```
5 23204120 raw4.txt120 raw4.txt120 raw4.txt120 raw4.txt120 raw4.txt120 raw4.txt120
raw4.txt120 raw4.txt120 raw4.txt120 raw4.txt120 raw4.txt120 raw4.txt120 raw4.txt120
raw4.txt120 raw4.txt120 raw4.txt120 raw4.txt120 raw4.txt120 raw4.txt120 raw4.txt120
raw4.txt120 raw4.txt120 raw4.txt120 raw4.txt120 raw4.txt120 raw4.txt120 raw4.txt120
raw4.txt120 raw4.txt120 raw4.txt120 raw4.txt120 raw4.txt120 raw4.txt120 raw4.txt120
raw4.txt120 raw4.txt120 raw4.txt120 raw4.txt120 raw4.txt120 raw4.txt120 raw4.txt120
raw4.txt120 raw4.txt120 raw4.txt120 raw4.txt120 raw4.txt120 raw4.txt120 raw4.txt120
raw4.txt120 raw4.txt120 raw4.txt120 raw4.txt120 raw4.txt120 raw4.txt120 raw4.txt120
raw4.txt120 raw4.txt120 raw4.txt120 raw4.txt120 raw4.txt120 raw4.txt120 raw4.txt120
raw4.txt120 raw4.txt120 raw4.txt120 raw4.txt120 raw4.txt120 raw4.txt120 raw4.txt120
raw4.txt120 raw4.txt120 raw4.txt120 raw4.txt120 raw4.txt120 raw4.txt120 raw4.txt120
raw4.txt120 raw4.txt120 raw4.txt120 raw4.txt120 raw4.txt120 raw4.txt120 raw4.txt120
raw4.txt120 raw4.txt120 raw4.txt120 raw4.txt120 raw4.txt120 raw4.txt120 raw4.txt120
raw4.txt120 raw4.txt120 raw4.txt120 raw4.txt120 raw4.txt120 raw4.txt120 raw4.txt120
raw4.txt120 raw4.txt120 raw4.txt120 raw4.txt120 raw4.txt120 raw4.txt120 raw4.txt120
raw4.txt120 raw4.txt120 raw4.txt120 raw4.txt120 raw4.txt120 raw4.txt120 raw4.txt120
raw4.txt120 raw4.txt120 raw4.txt120 raw4.txt120 raw4.txt120 raw4.txt120 raw4.txt120
raw4.txt120 raw4.txt120 raw4.txt120 raw4.txt120 raw4
```

Entire conversation (1479 bytes)

ASCII EBCDIC Hex Dump C Arrays [✓ Raw]

The cloud sees all the messages being passed but the crucial thing is the shared key ie "12" and file key is "10" are no transferred anywhere in the communication .Hence even if the cloud tries to read the conversation the secrecy of the file is still maintained.

6. CONCLUSION

We demonstrate that our secret scheme is secure as long as the guy is authenticated. However, in real life the cloud may fake a client and pretend to issue a normal request. Classical solutions rely on the use of secure channels or a trusted server. In our case, we skip this part and focus on eavesdropper attack.

7. FUTURE WORKS

In the future, we would like to implement our thoughts on mobile devices, especially on Android platform. At the same time, try to create a fully automatic application to replace this demo one, which still need manual operations. If possible, design a good GUI is really necessary to improve user experiences.

REFERENCE

[1] Madiseh, M.G. McGuire, M.L. Neville, S.W. Secret Key Generation within Peer-to-Peer Network Overlays 3PGCIC, page 156-163. (2012)

[2] Cong Tang and Ruichuan Chen and Zhuhua Cai and Anmin Xie and Jian-bin Hu and Liyong Tang and Zhong Chen. SKIP: A Secure Key Issuing Scheme for Peer-to-Peer Networks ICNS'09 page 296-302. (2009)

[3] Ya-Fen Chang and Jung-San Lee and Chin-Chen Chang. A secure and efficient authentication scheme for mobile users. IJMC page 581-594. (2006)

[4] Amit P. Sheth. Federated Database Systems for Managing Distributed, Heterogeneous, and Autonomous Databases. VLDB'91 (1991)

[5] Su Chen and Yi Chen and Hai Jiang and Laurence Tianruo Yang and Kuan-Ching Li. A Secure Distributed File System Based on Revised Blakley's Secret Sharing Scheme. TrustCom'12 page 310-317. (2012)

[6] Chachapara, K. Bhadlawala, S. Secure sharing with cryptography in cloud computing. NUiCONE (2013)

[7] Federal Information Processing Standards Publication 197. Announcing the ADVANCED ENCRYPTION STANDARD (AES). NIST (2001)

[8] Diffie, W.; Hellman, M. "New directions in cryptography". IEEE Transactions on Information Theory 22 (6): 644–654. (1976)

APPENDIX

1. **File Encryption Program**

```
#include <stdio.h>
#include <stdlib.h>
void encryptor(char* raw, int key);
int main(int argc, char* args[]){
//argv[1]: string, file path
//argv[2]: string, encrypting key
        FILE* fp = fopen(args[1],"r");
        int psw = atoi(args[2]);
        if(fp == 0){perror("no such file");exit(1);}
        FILE* writer = fopen("writeBuffer2","w");
        char buffer[1000];
        int i = 0;
        char* tmp;
        while (fgets(buffer,sizeof(buffer),fp) != 0){
                printf("%s",buffer);
                encryptor(buffer,psw);
                fprintf(writer,"%s",buffer);
        }
        fclose(fp);
        fclose(writer);
        unlink(args[1]);
        rename("writeBuffer2",args[1]);
        return 0;
}
void encryptor(char* raw, int key){
        int i = 0;
        while(raw[i] != '\0'){
                raw[i] = (raw[i] + key + 256) % 256;
                i++;
        }
}
```

2. **Decryption Program**

```
#include <stdio.h>
#include <stdlib.h>
void decryptor(char* raw, int key);
int main(int argc, char* args[]){
//argv[1]: string, file path
//argv[2]: decrypting key
        FILE* fp = fopen(args[1],"r");
        int psw = atoi(args[2]);
        if(fp == 0){perror("no such file");exit(1);}
        FILE* writer = fopen("writeBuffer2","w");
        char buffer[1000];
        int i = 0;
        char* tmp;
        while (fgets(buffer,sizeof(buffer),fp) != 0){
                printf("%s",buffer);
                decryptor(buffer,psw);
                fprintf(writer,"%s",buffer);
        }
        fclose(fp);
        fclose(writer);
        unlink(args[1]);
        rename("writeBuffer2",args[1]);
        return 0;
}
void decryptor(char* raw, int key){
        int i = 0;
        while(raw[i] != '\0'){
                raw[i] = (raw[i] - key + 256) % 256;
                i++;
        }
}
```

3. TCP client Program

```
/* tcpclient.c */

#include <sys/socket.h>
#include <sys/types.h>
#include <netinet/in.h>
#include <netdb.h>
#include <stdio.h>
#include <string.h>
#include <stdlib.h>
#include <unistd.h>
#include <errno.h>
#include <time.h>
#include <math.h>
void fileCopy(char* sourceP,char* targetP);
int main(int argc, char* argv[])
//argv[1]: string, ip address of server
{

        int sock, bytes_recieved;
        char send_data[1024],recv_data[1024];
        struct hostent *host;
        struct sockaddr_in server_addr;
        int mod,base;
        int privatekey;
        int counter = 1;
        int publickey;
        int sharedkey;
        int encryfilekey;

        host = gethostbyname(argv[1]);

        if ((sock = socket(AF_INET, SOCK_STREAM, 0)) == -1) {
                perror("Socket");
                exit(1);
        }

        server_addr.sin_family = AF_INET;
```

```
    server_addr.sin_port = htons(5000);
    server_addr.sin_addr = *((struct in_addr *)host->h_addr);
    bzero(&(server_addr.sin_zero),8);

    if (connect(sock, (struct sockaddr *)&server_addr,
                sizeof(struct sockaddr)) == -1)
    {
        perror("Connect");
        exit(1);
    }

    while(1)
    {
        bytes_recieved=recv(sock,recv_data,1024,0);
        recv_data[bytes_recieved] = '\0';
        int i;
        for (i = 0; i<bytes_recieved; i++)

            if (recv_data[i] == ' ')
                break;


if(i == 0) {printf("server rejects you\n");break;}
            printf("read%s\n",recv_data);
        if (counter == 1) {
            mod = atoi(&recv_data[i+1]);
            recv_data[i] = '\0';
            base = atoi(recv_data);
            srand(time(NULL));
            //privatekey = rand()%20;
            privatekey = 5;
            unsigned long int temp = (unsigned long int) pow(base, privatekey);
temp%=mod;

printf("pow %lu\n",temp);
            sprintf(send_data, "%lu", temp);

            send(sock,send_data,strlen(send_data), 0);

            counter ++;
            printf("counter 1 done. privatekey is %d send_data%s.\n",
privatekey,send_data);
        }
        else if (counter == 2)
        {
            publickey = atoi(recv_data);
printf("public key =%d %f\n",publickey,pow(privatekey,publickey));
unsigned long int temp = (unsigned long int) pow(publickey,privatekey);
temp%=mod;
printf("pow %lu\n",temp);
            sharedkey = temp;

            counter ++;

            printf("counter 2 done. sharedkey is %d.\n", sharedkey);
```

```
                    }
                    else if (counter == 3)
                    {
char* sourcePath = &recv_data[i+1];
recv_data[i] = '\0';
                              encryfilekey = atoi(recv_data);
                              encryfilekey = encryfilekey/sharedkey;
                              printf("transmission (counter 3) done. encryfilekey is %d. \n",
encryfilekey);
printf("server shares you %s\n Where do you want to keep this file? Enter a
path\n",sourcePath);
char targetPath[20];
scanf("%s",targetPath);
fileCopy(sourcePath,targetPath);

                              break;
                    }


          }
          return 0;
}
void fileCopy(char* sourceP,char* targetP){
FILE* source = fopen(sourceP,"r");
if (source == 0) {
perror("source file does not exist.");
exit(1);
}
FILE* target = fopen(targetP,"w");
char buffer[100];
while(fgets(buffer,sizeof(buffer),source) != 0){
          fprintf(target,"%s",buffer);
}
fclose(source);
fclose(target);
}
```

4. **Tcp server program**

```
/* tcpserver.c */

#include <sys/types.h>
#include <sys/socket.h>
#include <netinet/in.h>
#include <arpa/inet.h>
#include <stdio.h>
#include <stdlib.h>
#include <unistd.h>
#include <errno.h>
#include <string.h>
#include <time.h>
#include <math.h>
#define _BASE 5
#define _MOD  23
int main(int argc, char*argv[])
          //argv[1] string, key for the file, supposed to be integer
          //argv[2] path of sharing file
```

```
{       int count=0;
        int private_key;
        int int_key;
        int ses_key;
        int file_key = atoi(argv[1]);
        int sock, connected, bytes_recieved , true = 1;
        char send_data [1024] , recv_data[1024];

        struct sockaddr_in server_addr,client_addr;
        int sin_size;
        FILE* sharingFile = fopen(argv[2],"r");
        if(sharingFile == 0) {
                perror("target file does not exist. Restart with a valid one.");
                exit(1);
        }
        if ((sock = socket(AF_INET, SOCK_STREAM, 0)) == -1) {
                perror("Socket");
                exit(1);
        }

        if (setsockopt(sock,SOL_SOCKET,SO_REUSEADDR,&true,sizeof(int)) == -1) {
                perror("Setsockopt");
                exit(1);
        }

        server_addr.sin_family = AF_INET;
        server_addr.sin_port = htons(5000);
        server_addr.sin_addr.s_addr = INADDR_ANY;
        bzero(&(server_addr.sin_zero),8);

        if (bind(sock, (struct sockaddr *)&server_addr, sizeof(struct sockaddr))
                        == -1) {
                perror("Unable to bind");
                exit(1);
        }

        if (listen(sock, 5) == -1) {
                perror("Listen");
                exit(1);
        }

        printf("\nTCPServer Waiting for client on port 5000");
        fflush(stdout);


        while(1)
        {

                sin_size = sizeof(struct sockaddr_in);

                connected = accept(sock, (struct sockaddr *)&client_addr,&sin_size);

                printf("\n I got a connection from (%s , %d)\n Do you want to accept
it?\nPress y to accept.",
                                inet_ntoa(client_addr.sin_addr),ntohs(client_addr.sin_port));
                char buf;
                scanf("%c",&buf);
```

```
				if(buf != 'y'){close(sock);break;}

				while (1)
				{
					//printf("\n SEND (q or Q to quit) : ");
					//gets(send_data);


					if(count==0)
					{
						sprintf(send_data,"%d %d",_BASE,_MOD);
						count++;
					}
					else if(count==1)
					{
						private_key = 4;
						//srand(time(NULL));
						//private_key=rand()%20;
						unsigned long int temp;
						temp=(unsigned long int)pow(_BASE,private_key);
						temp=temp%_MOD;
						sprintf(send_data,"%lu",temp);
						count++;
						printf("count1 %s\n",send_data);
					}
					else if(count==2)
					{
						int temp=file_key*ses_key;
						sprintf(send_data,"%d %s",temp,argv[2]);
						printf("count2 %s\n",send_data);
						count++;
					}
					send(connected, send_data,strlen(send_data), 0);
					if(count!=1)
					{
						bytes_recieved = recv(connected,recv_data,1024,0);

						recv_data[bytes_recieved] = '\0';
						if(count==2)
						{
							int_key=atoi(recv_data);
							ses_key=(int)pow(int_key,private_key);
							ses_key%=_MOD;
							printf("sharedkey =%d int_key=%d tmp
= %d\n",ses_key,int_key,(int) pow(int_key,private_key));
						}
					}
				}
		}

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