Secure Cloud

Security in Networked Computing Systems project
Master of Science in Computer Engineering
University of Pisa

Candidates:
Micheloni Giulio
De Bianchi Luigi

1. System description

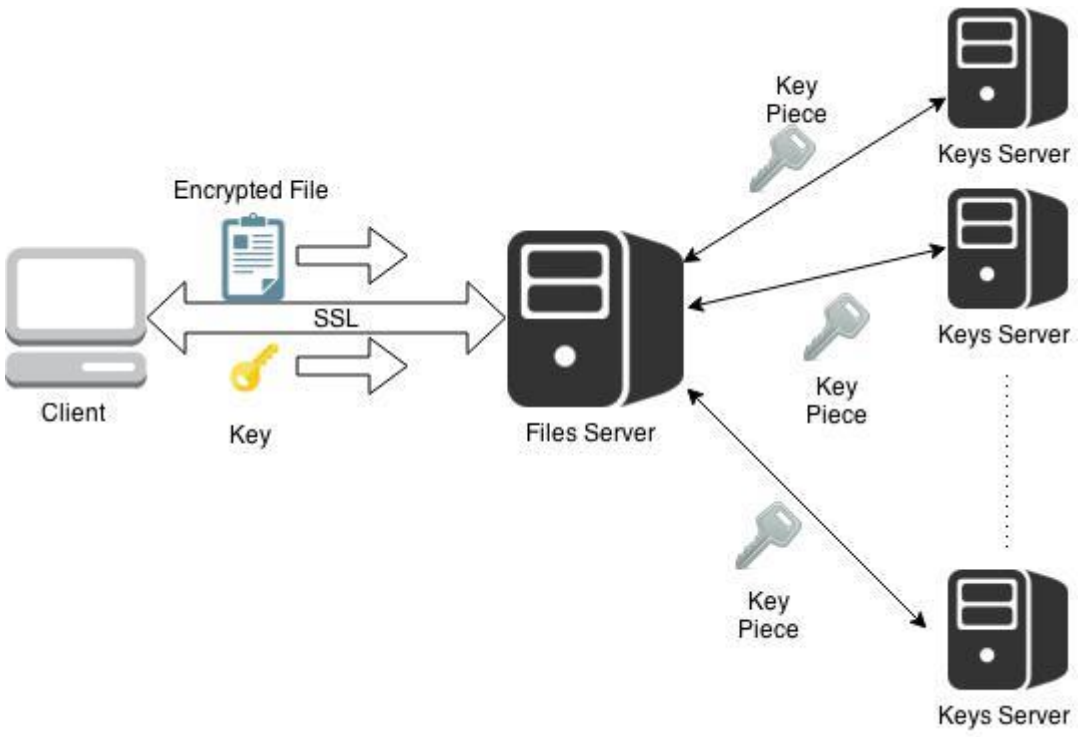
This is a secure file-storing application. It is composed basically of three entities:

- Clients: which can upload and download their own files, in a secure way.
- Files Server: that authenticates the clients, handles the encrypted files and splits the relative keys in multiple pieces, through Shamir Secret-Sharing algorithm.
- Keys Servers: their task is just to store, and send back, on request, the keys pieces.

The simplified behavior of the system is described below.

Upload phase:

1) The client authenticates itself to the File Server sending username and password;
2) The server sends a permit or deny response to the client;
3) If authentication is successful, the client both encrypts the chosen file with a random key, and sends both to the server;
4) The server stores the encrypted file, runs the Shamir Secret Sharing algorithm over the key, and then it sends one key-piece to each Key Server;

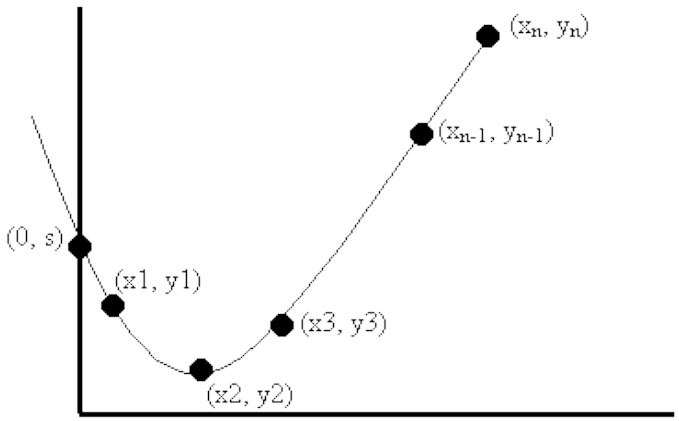


Download phase:

1) The client authenticates itself to the File Server sending username and password, also telling which file it wants to download;
2) The server collects all the key pieces from the Key Servers, and performs the recovery of the original key, then it sends the encrypted file and the key back to the client.

2. Shamir secret sharing

Now the secret sharing algorithm, used in this project, will be described. Thanks to this algorithm it is possible to split the secret in *n* unique parts, and when it is needed, only *k* over *n* parts are necessary to reconstruct correctly the original secret.
The main idea of the Shamir's scheme is that: to uniquely define a polynomial of degree *k*, are necessary *k+1* points.
Consider the secret as a number *S*, it is possible to generate a polynomial of degree *k-1*, imposing that crosses the constant term (*0, S*).
In this way, only who knows *k* points is able to reconstruct the polynomial through interpolation, and thus find the value at coordinate (*0, S*).



In practice the splitting algorithm is this:

1) Given *S* the secret, and the parameters (*k*,*n*)
2) Choose at random *k-1* positive integers $a_1, \ldots, a_{k-1}$
3) Define the polynomial $f(x) = S + a_1 x + a_2 x^2 + \ldots + a_{k-1} x^{k-1}$
4) With $i \in [1, n]$ compute $(i, f(i))$and give it to each participant
5) Destroy *S* and polynomial coefficients

The recovery algorithm is the following:

1) Collect *k* or more points $(i, f(i))$
2) Use Lagrange polynomial to interpolate these points
   i.e. $L(x) = \sum_{j=0}^{k} f(j) l(j)$
   where $l(j) = \prod_{m \in [0,k]}^{m \neq j} \frac{x - x_m}{x_j - x_m}$
3) The original secret is $f(0)$

3. Service guide

Needed library

The various services implemented are all based on common interfaces present in the standard libraries except for the OpenSSL library, which is needed for the encryption schemes, hash function and ssl connection.

File structure

In the root folder there are some utility files needed in all or some of the implemented services:

- security_ssl.h
  Contains all the functions needed to establish and manage ssl connection

- utils.h
  This library provides interfaces to serve three purposes:
  - symmetric encryption
  - asymmetric encryption
  - evaluate and verify hash

In addition to these files, there are these three folders; each one contains all the files needed to compile and run one of the three services developed for the system.

- Client
- Files_Server
- Key_Server

Compile

In the root folder there is a Makefile that automatically compiles each one of the three services, moreover in each folder there is a private Makefile to compile the specific service.

Run

Each one of the services must be started individually in the following order:

1. File server
   ./files_server [address] [port]
   Where [address] and [port] are mandatory parameters that define where the service is mounted.

2. Key Server
   ./key_server [address] 5555 [id] [fileServer_pubkey] [KeyServer_privkey]

Where 5555 is the predefined address where the file server waits for requests.
[address] and [id] are mandatory and define the address of the file server and the id of the started key server.
[fileServer_pubkey] [KeyServer_privkey] are not mandatory but it's not permitted to set only one of the two; as the name suggests, these parameters represent respectively the path to the file server public key and to the key server private key.

3. Client
   ./client [address] [port]
   Where [address] and [port] are mandatory parameters that define the address and the port where the file server is located.

It's also possible to run multiple key servers at once using the "run.sh" bash script that runs individually four key servers.

Configuration

It's obviously possible to set the address and the port where each service is mounted, varying the parameter.
Even if you prefer to use bash file it's possible to modify each parameter modifying the variable defined in the script itself; the only limit is that each private key for the varius key servers must be stored in the same directory.

As regards the behavior of the key servers it's possible to set up how many secret pieces a server is able to support simply by changing the value of the directive N_ELEMENTS; it's important to remember that the search within the server is done with a linear search algorithm.

Termination

Client terminates itself after one command, whereas File server and key server terminate when they receive a SIGINT signal; this can be sent either through the key combination ctrl + c or by using a kill command.
Like before, in the case of the key servers, this can be done either by hand or by launching the "kill_servers.sh" script.

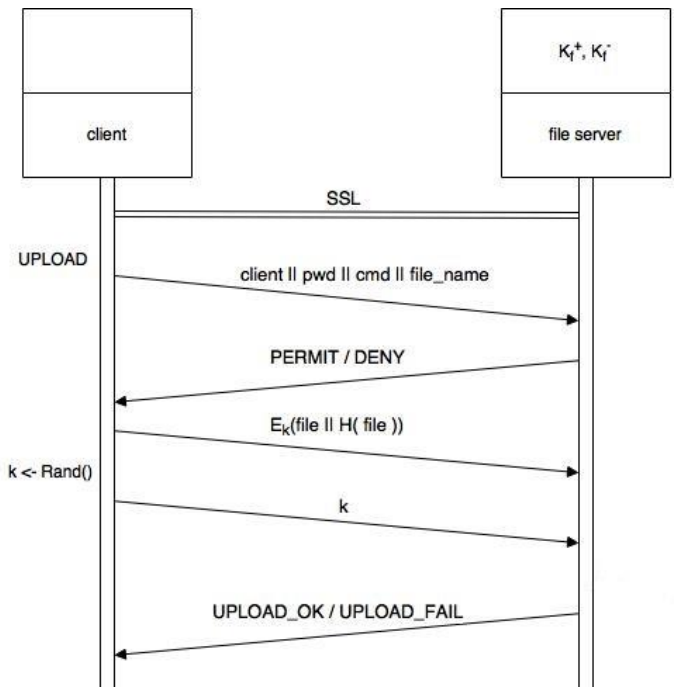
4. Communication Client – Files Server

The communication between Client and File Server is over an SSL connection, where only the client authenticates the server through certificates.
The server's certificate was created by a certification authority built up with OpenSSL.
The server authenticates the client through username and password, which are not stored in cleartext, but using hash value (SHA-256).

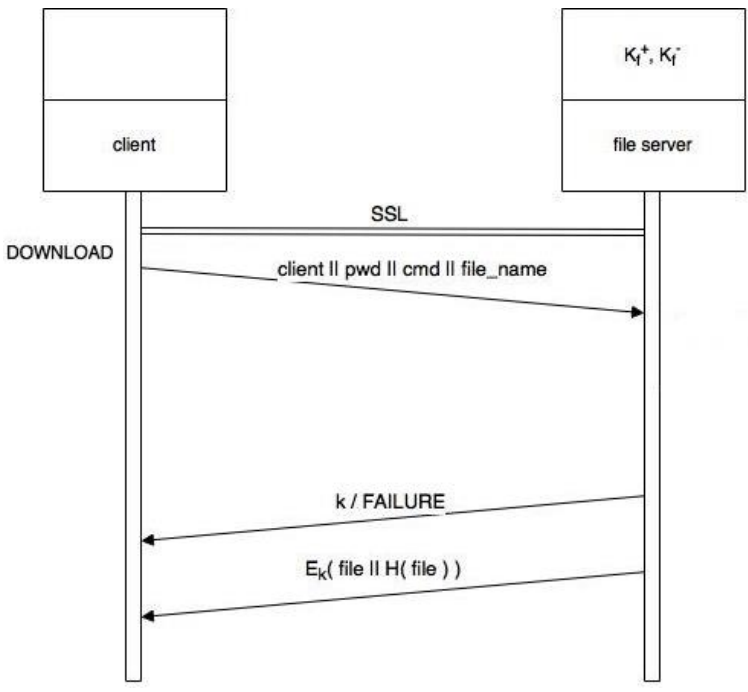
Applicative protocol
After the SSL connection establishment, the client initiates the communication:
At beginning the client sends to the server the username and password, the command that it wants to perform and the name of the file.
After this, the server checks if the pair [username, Hash(password)] is present in the "pwd.txt" file, and responds accordingly.
The subsequent operations depend on the kind of operation desired.

Upload protocol
After authentication the client generates a random key, encrypts the content of the file and its hash, and sends ciphertext and key to the server.

Download protocol
If the authentication is successful, the server fetches the encrypted file from its file system, performs operations in order to reconstruct the original key, then sends the key and the file, or an error message in case of error.



Upload Phase



Download Phase

5. Communication Files Server – Keys Server

The two servers communicate using messages, encrypted using symmetric encryption after a key establishment protocol that exploits public key encryption.
Note that in this implementation certificates are not used, thus it was assumed that each part believes that key pair (K_x^+, K_x^-) belongs to the entity x.

Key establishment protocol

M1	ks → fs:	$\{\text{ID File Server},\ \text{ID Key Server},\ \text{nonce}_A\}_{K_{fs}^+}$
M2	fs → ks:	$\{\text{ID File Server},\ \text{ID Key Server},\ \text{nonce}_A,\ \text{nonce}_B,\ \text{K}\}_{K_{ks}^+}$
M3	ks → fs:	$\{\text{nonce}_B\}_{\text{K}}$

where "ks" represents a key server and "kf" a file server and K^+ and K^- represent respectively public and private key.

Applicative protocol

After the key establishment protocol, each key server waits for a message that specifies the next operation; this can be key upload or download.

Upload protocol

The key server waits for a message in this format

$$\{\text{File ID} \parallel \text{x} \parallel \text{secret size} \parallel \text{secret} \parallel \text{hash(payload)}\}_{\text{K}}$$

which specifies the secret obtained by the Shamir secret sharing algorithm.
No message is sent in response because ideally the server splits the secret in so many pieces that even if something is lost, this doesn't imply failure in the systems.

Download protocol

The key server waits for a message in this format

$$\{\text{File ID}\}_{\text{K}}$$

If the id isn't saved, key server replies with a message that is interpreted by the file server as a "not found". This way, the file server can send the download request to another key server.
If the id is found, the key server replies with a message which contains all the data of the secret in this format:

$$\{\text{File ID} \parallel \text{x} \parallel \text{dim size} \parallel \text{secret} \parallel \text{hash(payload)}\}_{\text{K}}$$

After this, all the data are deleted in a secure way by the key server.