may 13, 19 11:49 **server_thread.h** Page 1/1

```
#ifndef THREAD_H
#define THREAD_H

#include <iostream>
#include <vector>
#include <thread>
#include <mutex>

class Thread {
    private:
        std::thread thread;

    public:
        Thread()= default;
        void start();
        void join();
        virtual void run() = 0;
        virtual ~Thread() {}
        Thread(const Thread&) = delete;
        Thread& operator=(const Thread&) = delete;
        Thread(Thread∧ other);
        Thread& operator=(Thread∧ other);
};

#endif
```

may 13, 19 11:49 **server_thread.cpp** Page 1/1

```
#include "server_thread.h"

void Thread :: start() {
    thread = std::thread(&Thread::run, this);
}

void Thread :: join() {
    thread.join();
}

Thread :: Thread(Thread∧ other) {
    this→thread = std::move(other.thread);
}

Thread& Thread :: operator=(Thread∧ other) {
    this→thread = std::move(other.thread);
    return *this;
}
```

may 13, 19 11:49 **server_socket_accept.h** Page 1/1

```
#ifndef SOCKET_ACCEPT
#define SOCKET_ACCEPT

#include <string.h>
#include <stdio.h>
#include <errno.h>
#include <stdbool.h>
#include <sys/types.h>
#include <sys/socket.h>
#include <netdb.h>
#include <unistd.h>
#include "common_socket_connect.h"

class SocketAccept {
    private:
        int s;
        int skt;
        int opt;
        struct addrinfo hints;
        struct addrinfo *ptr;
        void setHints();

    public:
        SocketAccept();
        ~SocketAccept();
        SocketConnect acceptSocket();
        bool listenSocketAccept();
        bool bindSocketAccept();
        bool sktOpciones();
        bool addrinfo(char *srvn);
        bool sktSocketAccept();
        void cerrarSocket();
};

#endif
```

may 13, 19 11:49 **server_socket_accept.cpp** Page 1/2

```
#define _POSIX_C_SOURCE 200112L

#include "server_socket_accept.h"

SocketAccept :: SocketAccept() {
  s = 0;
  skt = 0;
  opt = 1;
}

SocketAccept :: ~SocketAccept() {
    skt = -1;
    s = -1;
}

void SocketAccept :: cerrarSocket() {
  shutdown(skt,SHUT_RDWR);
  close(skt);
}

void SocketAccept :: setHints() {
  memset(&hints, 0, sizeof(struct addrinfo));
  hints.ai_family = AF_INET;
  hints.ai_socktype = SOCK_STREAM;
  hints.ai_flags = 0;
}

bool SocketAccept :: addrinfo(char *srvn) {
  setHints();
  s = getaddrinfo(NULL, srvn, &hints, &ptr);
    if (s ≠ 0) {
          printf("Error in getaddrinfo: %s\n", gai_strerror(s));
          return false;
    }
    return true;
}

bool SocketAccept :: sktSocketAccept() {
  skt = socket(ptr→ai_family, ptr→ai_socktype,
  ptr→ai_protocol);
  if (skt ≡ -1) {
    printf("Error: %s\n", strerror(errno));
    freeaddrinfo(ptr);
    return false;
  }
  return true;
}

bool SocketAccept :: sktOpciones() {
  s = setsockopt(skt, SOL_SOCKET, SO_REUSEADDR,
   &opt, sizeof(opt));
  if (s ≡ -1) {
    printf("Error: %s\n", strerror(errno));
    close(skt);
    freeaddrinfo(ptr);
    return false;
  }
  return true;
}

bool SocketAccept :: bindSocketAccept() {
  s = bind(skt, ptr→ai_addr, ptr→ai_addrlen);
  freeaddrinfo(ptr);
  if (s ≡ -1) {
    printf("Error: %s\n", strerror(errno));
    close(skt);
```

```
67      return false;
68    }
69    return true;
70  }
71
72  bool SocketAccept :: listenSocketAccept() {
73    s = listen(skt, 20);
74    if (s == -1) {
75      printf("Error: %s\n", strerror(errno));
76      close(skt);
77      return false;
78    }
79    return true;
80  }
81
82  SocketConnect SocketAccept :: acceptSocket() {
83    int skt_aceptado = accept(skt, NULL, NULL);
84    return SocketConnect(skt_aceptado);
85  }
86
```



```
1   #ifndef INDICE_H
2   #define INDICE_H
3   #include <iostream>
4   #include <fstream>
5   #include <string>
6   #include <map>
7   #include <mutex>
8   #include "common_archivo.h"
9   #include "server_cliente.h"
10
11  class Indice {
12      private:
13          std::string nombre_archivo;
14          std::fstream archivo;
15          std::map<std::string,Cliente*> clientes;
16          std::map<std::string,Cliente*> mapa_de_espera;
17          std::mutex m;
18          uint32_t indice_archivo;
19
20          //Se encarga de agrega un nuevo cliente al mapa clientes
21          void agregarCliente(std::string &nombre
22          ,std::string exponente,std::string modulo);
23          //Indica con un booleano si un cliente pertenece o no al map
24          //indice. En caso de no perteneceser y que el booleano que
25          //recibe por parametro sea true, incrementa en uno el
26          //indice del archivo
27          bool clientePerteneceAlIndice(const std::string &cliente,
28              const bool modo_new);
29
30          //Elimina el cliente pasado por parametro del map clientes
31          void eliminarCliente(const std::string &nombre);
32
33          //Indica con un booleano si el cliente pasado por paremetro
34          //pertenece o no al map_de_espera
35          bool clientePerteneceAEspera(const std::string &cliente);
36
37      public:
38          explicit Indice(std::string &nombre_archivo);
39          ~Indice();
40
41          //Lee el archivo
42          void leerArchivo();
43
44          //Parsea el archivo de indice y los agrega al map clientes
45          void parser(std::string &bf_aux,std::string &nombre_cliente
46          ,std::string &exponente,bool &primer_palabra,
47          bool &es_el_exp, bool &es_el_indice);
48
49          //Escribe un archivo con la informacion que tiene el map
50          //clientes
51          void escribirArchivo();
52          uint16_t getModuloCliente(const std::string &subject);
53          uint8_t getExponenteCliente(const std::string &subject);
54
55          //Agrega en caso que no pertenesca un cliente al
56          //mapa_de_espera
57          bool agregarSiNoPertenece(std::string &nombre,
58              uint8_t exponente,uint16_t modulo);
59
60          //Elimina un cliente si pertenece al map de clientes
61          bool eliminarSiPertenece(const std::string &nombre);
62
63          //Agrega un cliente al mapa_de_espera
64          void agregarAEspera(std::string &nombre,
65              uint8_t exponente,uint16_t modulo);
66
```

may 13, 19 11:49 **server_indice.h** Page 2/2

```
67          //Indica si un cliente pertence al map clientes
68          void pasarClienteDeEsperaAIndice(std::string &nombre);
69
70          //Elimina un cliente del mapa_de_espera
71          void eliminarDeEspera(std::string &nombre);
72
73          int getIndiceCliente(const std::string &nombre);
74  };
75
76  #endif
```

may 13, 19 11:49 **server_indice.cpp** Page 1/3

```
1   #include <string>
2   #include <map>
3   #include <mutex>
4   #include "server_cliente.h"
5   #include "server_indice.h"
6
7   Indice :: Indice(std::string &nombre_archivo) :
8       archivo(nombre_archivo, std::fstream::in) ,clientes() {
9           leerArchivo();
10          this→nombre_archivo = nombre_archivo;
11  }
12
13  Indice :: ~Indice() {
14      archivo.close();
15  }
16
17  void  Indice :: parser(std::string &bf_aux,std::string &nombre_cliente
18      ,std::string &exponente,bool &primer_palabra, bool &es_el_exp
19      , bool &es_el_indice) {
20      if (es_el_indice) {
21          indice_archivo = stoi(bf_aux)−1;
22          es_el_indice = false;
23          return;
24      }
25      if (bf_aux.back() ≡ ';') {
26          bf_aux.pop_back();
27          nombre_cliente = nombre_cliente+" "+bf_aux;
28          primer_palabra = true;
29          return;
30      }
31      if (isdigit(bf_aux[0])) {
32          if (es_el_exp) {
33              exponente = bf_aux;
34              es_el_exp = false;
35          } else {
36              agregarCliente(nombre_cliente,exponente,bf_aux);
37              nombre_cliente = "";
38              es_el_exp = true;
39          }
40          return;
41      }
42      if (¬primer_palabra) {
43          nombre_cliente = nombre_cliente + " " + bf_aux;
44      } else {
45          nombre_cliente = bf_aux;
46          primer_palabra = false;
47      }
48  }
49
50  void Indice :: leerArchivo() {
51      std::string bf_aux;
52      std::string exponente;
53      bool primer_palabra = true;
54      bool es_el_exp = true;
55      bool es_el_indice = true;
56      this→indice_archivo = 0;
57      std::string nombre_cliente = "";
58      while (archivo >> bf_aux) {
59          parser(bf_aux,nombre_cliente,exponente,
60          primer_palabra,es_el_exp,es_el_indice);
61      }
62  }
63
64  void Indice :: eliminarCliente(const std::string &nombre) {
65      Cliente *cliente = this→clientes[nombre];
66      delete cliente;
```

```
67      clientes.erase(nombre);
68  }
69
70  bool Indice :: clientePerteneceAlIndice(const std::string &cliente,
71    const bool modo_new) {
72      for (std::map<std::string,Cliente*>::iterator
73      it=clientes.begin(); it≠clientes.end(); ++it) {
74          if (cliente ≡ it→second→getNombre()) {
75              return true;
76          }
77      }
78      if (modo_new) this→indice_archivo++;
79      return false;
80  }
81
82  bool Indice :: eliminarSiPertenece(const std::string &nombre) {
83      std::unique_lock<std::mutex> lck(m);
84      if (this→clientePerteneceAlIndice(nombre,false)) {
85          this→eliminarCliente(nombre);
86          return true;
87      }
88      return false;
89  }
90
91  bool Indice :: clientePerteneceAEspera(const std::string &cliente) {
92      return (mapa_de_espera.find(cliente) ≠ mapa_de_espera.end());
93  }
94
95  void Indice :: agregarAEspera(std::string &nombre,
96      uint8_t exponente,uint16_t modulo) {
97          std::string exp = std::to_string(exponente);
98          std::string mod = std::to_string(modulo);
99          Cliente *cliente = new Cliente(nombre,exp,mod,indice_archivo);
100         this→mapa_de_espera.insert({nombre,cliente});
101 }
102
103 bool Indice :: agregarSiNoPertenece(std::string &nombre,
104     uint8_t exponente,uint16_t modulo) {
105         std::unique_lock<std::mutex> lck(m);
106         if (¬this→clientePerteneceAlIndice(nombre,true) ∧
107         ¬this→clientePerteneceAEspera(nombre)) {
108             this→agregarAEspera(nombre,exponente,modulo);
109             return true;
110         }
111         return false;
112 }
113
114 void Indice :: pasarClienteDeEsperaAIndice(std::string &nombre) {
115     std::unique_lock<std::mutex> lck(m);
116     Cliente *cliente = this→mapa_de_espera[nombre];
117     this→clientes.insert({nombre,cliente});
118     this→mapa_de_espera.erase(nombre);
119 }
120
121 void Indice :: eliminarDeEspera(std::string &nombre) {
122     Cliente *cliente = this→mapa_de_espera[nombre];
123     delete cliente;
124     clientes.erase(nombre);
125 }
126
127 void Indice :: agregarCliente(std::string &nombre,
128   std::string exponente,std::string modulo) {
129     Cliente *cliente = new Cliente(nombre,exponente,modulo,indice_archivo);
130     this→clientes.insert({nombre,cliente});
131 }
132
```



```
133 void Indice :: escribirArchivo() {
134     archivo.close();
135     archivo.open(this→nombre_archivo, std::fstream::out | std::fstream::trunc);
136     archivo << (this→indice_archivo+1) << '\n';
137     for (std::map<std::string,Cliente*>::iterator
138     it=clientes.begin(); it≠clientes.end(); ++it) {
139         archivo << it→second→getNombre()<< ";" << it→second→getExponente()
140         << ' ' << it→second→getModulo() << '\n';
141         delete it→second;
142     }
143 }
144
145 uint16_t Indice :: getModuloCliente(const std::string &nombre) {
146     if (this→clientePerteneceAlIndice(nombre,false)) {
147         return (uint16_t)std::stoi(this→clientes[nombre]→getModulo());
148     }
149     return 0;
150 }
151
152 uint8_t Indice :: getExponenteCliente(const std::string &nombre) {
153     if (this→clientePerteneceAlIndice(nombre,false)) {
154         return (uint8_t)std::stoi(this→clientes[nombre]→getExponente());
155     }
156     return 0;
157 }
158
159 int Indice :: getIndiceCliente(const std::string &nombre) {
160     return this→mapa_de_espera[nombre]→getIndice();
161 }
```

may 13, 19 11:49 **server.cpp** Page 1/1

```cpp
#include "common_rsa.h"
#include "common_socket_connect.h"
#include "common_certificado.h"
#include "common_protocolo.h"
#include "common_claves.h"
#include "server_indice.h"
#include "server_socket_accept.h"
#include "server_aceptador_de_conexiones.h"
#include "server_comando.h"
#include <iostream>
#include <string>

int main(int argc, char *argv[]) {
    if (argc ≠ 4) {
        std::cout << "Error: argumentos invalidos." << std::endl;
        return 0;
    }

    std::string archivo_claves = argv[2];
    Claves claves(archivo_claves);
    claves.parser();

    std::string archivo_indice = argv[3];
    Indice indice(archivo_indice);

    SocketAccept socket_accept;

    if (¬socket_accept.addrinfo(argv[1])) return 1;

    if (¬socket_accept.sktSocketAccept()) return 1;

  if (¬socket_accept.sktOpciones()) return 1;

    if (¬socket_accept.bindSocketAccept()) return 1;

    if (¬socket_accept.listenSocketAccept()) return 1;

    bool continuar_ejecutando = true;
    AceptadorDeConexiones *hilo_aceptador =
      new AceptadorDeConexiones(socket_accept,indice,claves);
    hilo_aceptador→start();

    std::string entrada;
    while (continuar_ejecutando) {
        std::cin >> entrada;
        if (entrada ≡ "q") continuar_ejecutando = false;
    }
    hilo_aceptador→finalizarEjecucion();
    hilo_aceptador→join();
    delete hilo_aceptador;

    indice.escribirArchivo();
    return 0;
}
```

may 13, 19 11:49 **server_comunicador.h** Page 1/1

```cpp
#ifndef COMUNICADOR_H
#define COMUNICADOR_H
#include "server_thread.h"
#include "server_indice.h"
#include "common_claves.h"
#include "common_socket_connect.h"
#include "server_comando.h"

class Comunicador : public Thread {
        private:
        SocketConnect socket;
        Indice *indice;
        Claves *claves;
        bool termino;

        public:
        bool terminoEjecucion();
        Comunicador(SocketConnect socket,
        Indice *indice,Claves *claves);
        ~Comunicador() = default;
        virtual void run() override;
};

#endif
```

may 13, 19 11:49 **server_comunicador.cpp** Page 1/1

```cpp
#include "server_comunicador.h"


Comunicador :: Comunicador(SocketConnect socket,
    Indice *indice,Claves *claves) {
    this→socket = std::move(socket);
    this→indice = indice;
    this→claves = claves;
    termino = false;
}

bool Comunicador :: terminoEjecucion() {
    return this→termino;
}

void Comunicador :: run() {
    try {
        ComandoServidor comando(&socket);
        comando.inciarModo(*indice,*claves);
        socket.cerrarConexion();
        termino = true;
    } catch (const std::runtime_error& e) {
        std::cout << e.what() << std::endl;
    } catch (...) {
        std::cout << "Ocurrio un error desconocido" << std::endl;
    }
}
```

may 13, 19 11:49 **server_comando.h** Page 1/1

```cpp
#ifndef COMANDO_SERVIDOR_H
#define COMANDO_SERVIDOR_H
#include <iostream>
#include <string>
#include "common_protocolo.h"
#include "common_socket_connect.h"
#include "common_claves.h"
#include "server_indice.h"
#include "common_certificado.h"

class ComandoServidor {
    private:
        Protocolo protocolo;
        bool comandoNew(Indice &indice,Claves &claves);
        bool comandoRevoke(Indice &indice,Claves &claves);
        bool comandoNewEnviarRespuesta(
            Certificado &certificado,uint32_t &rsa);

        //Recibe un Certificado, un Indice, Claves y un uint32_t y verifica
        //si el hash del certificado coincide con la huella dada. En caso que
        //coincidan devuelve 0, si no coinciden devuelve 2. En caso que no halla
        //podido extraer los datos necesarios del indice para calcular el hash
        //devuelve 1.
        int verificarHash(Certificado &certificado,Indice &indice,
            Claves &claves,uint32_t huella);

    public:
        explicit ComandoServidor(SocketConnect *socket);
        ~ComandoServidor() = default;

        //Recibe como parametro un SocketConnect, un Indice y Claves. Espera
        //a recibir el modo en el que debe ser ejecutado el servidor (0 new o
        //1 revoke).
        bool inciarModo(Indice &indice,Claves &claves);
};

#endif
```

```
#include <string>
#include <mutex>
#include "server_comando.h"
#include "common_certificado.h"
#include "common_fecha.h"

ComandoServidor :: ComandoServidor(SocketConnect *socket) : protocolo(socket) {
}

int ComandoServidor :: verificarHash(Certificado &certificado,Indice &indice,
    Claves &claves, uint32_t huella) {

    std::string subject = certificado.getSubject();
    uint16_t mod_cliente = indice.getModuloCliente(subject);
    uint8_t exp_cliente = indice.getExponenteCliente(subject);

    if (mod_cliente ≡ 0 ∧ exp_cliente ≡ 0) return 1;

    huella = certificado.calcularRsa(huella, exp_cliente, mod_cliente);

    uint16_t mod_servidor = claves.getModulo();
    uint8_t exp_servidor = claves.getExponentePrivado();

    huella = certificado.calcularRsa(huella, exp_servidor, mod_servidor);

    uint32_t hash = certificado.calcularHash();
    if (hash ≠ huella) return 2;
    return 0;
}

bool ComandoServidor :: comandoRevoke(
    Indice &indice,Claves &claves) {
    uint32_t sn = 0;
    this→protocolo >> sn;

    std::string subject;
    this→protocolo >> subject;

    std::string issuer;
    this→protocolo >> issuer;

    std::string fecha_incial;
    this→protocolo >> fecha_incial;

    std::string fecha_final;
    this→protocolo >> fecha_final;

    uint16_t modulo(0);
    this→protocolo >> modulo;

    uint8_t exponente(0);
    this→protocolo >> exponente;

    uint32_t huella(0);
    this→protocolo >> huella;

    Certificado certificado;
    certificado.setAtributos(sn,subject,fecha_incial,fecha_final,
      modulo,exponente);
    certificado.parser();

    uint8_t rta = 0;

    if (this→verificarHash(certificado,indice,claves,huella) ≡ 2) {
        rta = 2;
        this→protocolo << rta;
```



```
        return false;
    }
    if (¬indice.eliminarSiPertenece(subject)) {
        rta = 1;
        this→protocolo << rta;
        return false;
    }
    this→protocolo << rta;
    return true;
}

bool ComandoServidor :: comandoNewEnviarRespuesta(
    Certificado &certificado,uint32_t &rsa) {

        uint32_t sn = certificado.getSerialNumber();
        this→protocolo << sn;

        std::string subject = certificado.getSubject();
        this→protocolo << subject;

        std::string issuer = certificado.getIssuer();
        this→protocolo << issuer;

        std::string f_inicial = certificado.getFechaInicio();
        this→protocolo << f_inicial;

        std::string f_final = certificado.getFechaFin();
        this→protocolo << f_final;

        uint16_t modulo = certificado.getModulo();
        this→protocolo << modulo;

        uint8_t exponente = certificado.getExponente();
        this→protocolo << exponente;

        this→protocolo << rsa;

        return true;
}

bool ComandoServidor :: comandoNew(Indice &indice,Claves &claves) {
    std::string subject;
    this→protocolo >> subject;

    uint16_t mod(0);
    this→protocolo >> mod;

    uint8_t exp(0);
    this→protocolo >> exp;

    std::string fecha_inicial;
    this→protocolo >> fecha_inicial;

    std::string fecha_final;
    this→protocolo >> fecha_final;

    uint8_t rta;
    if (¬indice.agregarSiNoPertenece(subject,exp,mod)) {
        rta = 1;
        this→protocolo << rta;
        return false;
    } else {
        rta = 0;
        this→protocolo << rta;
    }
    if (fecha_inicial ≡ "") {
```

may 13, 19 11:49 **server_comando.cpp** Page 3/3

```
133         Fecha fecha;
134         fecha_inicial = fecha.getFechaActual();
135         fecha_final = fecha.getFecha30DiasDespues();
136     }
137     uint32_t i = indice.getIndiceCliente(subject);
138     Certificado certificado;
139     certificado.setAtributos(i,subject,fecha_inicial,fecha_final,mod,exp);
140     certificado.parser();
141 
142     uint16_t hash = certificado.calcularHash();
143     uint32_t rsa = certificado.calcularRsa(hash,claves.getExponentePrivado(),
144       claves.getModulo());
145     rsa = certificado.calcularRsa(rsa,exp,mod);
146     if (¬this→comandoNewEnviarRespuesta(certificado,rsa)) return false;
147 
148     uint8_t rta_del_cliente;
149     this→protocolo >> rta_del_cliente;
150     if (rta_del_cliente ≠ 0) {
151         indice.eliminarDeEspera(subject);
152         return false;
153     }
154     indice.pasarClienteDeEsperaAIndice(subject);
155     return true;
156 }
157 
158 bool ComandoServidor :: inciarModo(Indice &indice,Claves &claves) {
159     uint8_t modo(2); //incio en modo invalido
160     this→protocolo >> modo;
161     if (modo ≡ 0) {
162         if (¬this→comandoNew(indice,claves)) return 1;
163         return 0;
164     } else if (modo ≡ 1) {
165         if (¬this→comandoRevoke(indice,claves)) return 1;
166         return 0;
167     } else {
168         return 1;
169     }
170 }
```

may 13, 19 11:49 **server_cliente.h** Page 1/1

```
1  #ifndef CLIENTE_S_H
2  #define CLIENTE_S_H
3  
4  #include <iostream>
5  #include <string>
6  
7  class Cliente {
8      private:
9          std::string nombre;
10         std::string modulo;
11         std::string exponente;
12         uint32_t indice;
13 
14     public:
15         Cliente(std::string nombre,std::string exp,std::string mod,
16             uint32_t indice);
17         ~Cliente() = default;
18         std::string getExponente();
19         std::string getNombre();
20         std::string getModulo();
21         uint32_t getIndice();
22         bool operator≡(const Cliente& otro) const;
23 };
24 
25 
26 #endif
```

may 13, 19 11:49 **server_cliente.cpp** Page 1/1

```cpp
#include "server_cliente.h"
#include <string>

Cliente :: Cliente(std::string nombre,std::string exp,std::string mod,
    uint32_t indice) {
    this→nombre = nombre;
    this→exponente = exp;
    this→modulo = mod;
    this→indice = indice;
}

std::string Cliente :: getNombre() {
    return this→nombre;
}

std::string Cliente :: getModulo() {
    return this→modulo;
}

std::string Cliente :: getExponente() {
    return this→exponente;
}

uint32_t Cliente :: getIndice() {
    return this→indice;
}

bool Cliente :: operator≡(const Cliente &otro) const {
    return this→nombre ≡ otro.nombre;
}
```

may 13, 19 11:49 **server_aceptador_de_conexiones.h** Page 1/1

```cpp
#ifndef ACEPTADOR_DE_CONEXIONES_H
#define ACEPTADOR_DE_CONEXIONES_H
#include "server_socket_accept.h"
#include "server_comunicador.h"
#include "server_indice.h"
#include "common_claves.h"
#include <vector>


class AceptadorDeConexiones : public Thread {
    private:
        SocketAccept *socket_aceptador;
        bool continuar_ejecutando;
        Indice *indice;
        Claves *claves;

    public:
        AceptadorDeConexiones(SocketAccept &socket_aceptador,
        Indice &indice,Claves &claves);
        ~AceptadorDeConexiones();
        virtual void run() override;
        void finalizarEjecucion();
};


#endif
```

may 13, 19 11:49 **server_aceptador_de_conexiones.cpp** Page 1/1

```cpp
#include "server_aceptador_de_conexiones.h"
#include <list>

AcceptadorDeConexiones :: AcceptadorDeConexiones(SocketAccept &socket_aceptador,
    Indice &indice,Claves &claves) {
    this→continuar_ejecutando = true;
    this→socket_aceptador = &socket_aceptador;
    this→indice = &indice;
    this→claves = &claves;
}

AcceptadorDeConexiones :: ~AcceptadorDeConexiones() {
}

void AcceptadorDeConexiones :: finalizarEjecucion() {
    this→continuar_ejecutando = false;
    this→socket_aceptador→cerrarSocket();
}

void AcceptadorDeConexiones :: run() {
    std::list<Comunicador*> comunicadores;
    while (this→continuar_ejecutando) {
        SocketConnect socket_connect = socket_aceptador→acceptSocket();
        if (¬this→continuar_ejecutando) {
            continue;
        }

        if (¬socket_connect.isValid()) {
            this→continuar_ejecutando = false;
        } else {
            Comunicador* comunicador = new Comunicador(std::move
            (socket_connect),indice,claves);
            comunicadores.push_back(comunicador);
            comunicadores.back()→start();
        }

        for (auto i = comunicadores.begin(); i ≠ comunicadores.end();) {
            if (¬(*i)→terminoEjecucion()) {
                ++i;
            } else {
                (*i)→join();
                delete (*i);
                i = comunicadores.erase(i);
            }
        }
    }

    for (auto i = comunicadores.begin(); i≠ comunicadores.end();) {
        (*i)→join();
        delete (*i);
        i = comunicadores.erase(i);
    }
}
```

may 13, 19 11:49 **common_socket_connect.h** Page 1/1

```cpp
#ifndef SOCKET_CONNECT_H
#define SOCKET_CONNECT_H
#define RESPONSE_MAX_LEN 50

#include <string>
#include <stdio.h>
#include <string.h>
#include <errno.h>
#include <stdbool.h>
#include <sys/types.h>
#include <sys/socket.h>
#include <netdb.h>
#include <unistd.h>

class SocketConnect {
    private:
        int s;
        int skt;
        struct addrinfo hints;
        struct addrinfo *result;
        struct addrinfo *ptr;
        void set_hints_socket();

    public:
        SocketConnect();
        explicit SocketConnect(int socket);
        bool addrinfo(const char *hostn, const char *srvn);
        bool conectar();

        //Cierra la conexion del socket.
        void cerrarConexion();
        ~SocketConnect() = default;
        //Recibe un string y un int.
        //Envia un mensaje al socket el cual esta conectado.
        int enviarMensaje(char *buf, int tam);

        //Recibe un string y un int.
        //Espera un mensaje del socket al cual esta conectado.
        int recibirMensaje(char *buf, int tam);

        //Devuelve un booleano indicando si el socket es valido
        bool isValid();

        SocketConnect& operator=(const SocketConnect&) = delete;
        SocketConnect(SocketConnect∧ other);
        SocketConnect& operator=(SocketConnect∧ other);
};

#endif
```

may 13, 19 11:49 **common_socket_connect.cpp** Page 1/2

```
#include <string.h>
#include <string>
#include <iostream>
#include "common_error.h"
#include "common_socket_connect.h"

void SocketConnect :: set_hints_socket() {
  memset(&hints, 0, sizeof(struct addrinfo));
  hints.ai_family = AF_INET;
  hints.ai_socktype = SOCK_STREAM;
  hints.ai_flags = 0;
}

SocketConnect :: SocketConnect() {
  s = -1;
  skt = -1;
}

SocketConnect :: SocketConnect(int socket) {
  s = -1;
  skt = socket;
}

bool SocketConnect :: addrinfo(const char *hostn,const char *srvn) {
  set_hints_socket();
  s = getaddrinfo(hostn, srvn, &hints, &result);
    if (s ≠ 0) {
          printf("Error in getaddrinfo: %s\n", gai_strerror(s));
          Error error;
          error.error_desconexion();
          return false;
    }
    return true;
}

bool SocketConnect :: conectar() {
    bool conexion_establecida = false;
    for (ptr = result; ptr≠NULL ∧ ¬conexion_establecida;
      ptr=ptr→ai_next) {
        skt = socket(ptr→ai_family, ptr→ai_socktype,
         ptr→ai_protocol);
        if (skt ≡ -1) {
            printf("Error: %s\n", strerror(errno));
        } else {
      s = connect(skt, ptr→ai_addr,
            ptr→ai_addrlen);
      if (s ≡ -1) {
        printf("Error: %s\n", strerror(errno));
        close(skt);
      }
      conexion_establecida = (s ≠ -1);
    }
    }
  freeaddrinfo(result);
  if (conexion_establecida ≡ false) {
    Error error;
    error.error_desconexion();
    return false;
  }
  return true;
}


int SocketConnect :: recibirMensaje(char *buf,int tam) {
  int recibido = 0, r= 0;
  bool el_socket_es_valido = true;
```

may 13, 19 11:49 **common_socket_connect.cpp** Page 2/2

```
  while (recibido < tam ∧ el_socket_es_valido) {
    r = recv(skt, &buf[recibido], tam-recibido, MSG_NOSIGNAL);
    if (r ≤ 0) {
      el_socket_es_valido = false;
    } else {
      recibido += r;
    }
  }
  if (el_socket_es_valido) {
    return recibido;
  } else {
    Error error;
    error.error_desconexion();
    return -1;
  }
}

int SocketConnect :: enviarMensaje(char *buf,int tam) {
  int total = 0,enviado = 0;
  bool el_socket_es_valido = true;
  while (total < tam ∧ el_socket_es_valido) {
    enviado = send(skt, &buf[total], tam-total, MSG_NOSIGNAL);
    if (enviado ≤ 0) {
      el_socket_es_valido = false;
    } else {
      total += enviado;
    }
  }
  if (el_socket_es_valido) {
    return total;
  } else {
    Error error;
    error.error_desconexion();
    return -1;
  }
}

void SocketConnect :: cerrarConexion() {
  shutdown(skt, SHUT_RDWR);
}

SocketConnect :: SocketConnect(SocketConnect∧ other) {
    this→s = std::move(other.s);
  this→skt = std::move(other.skt);
  other.skt = -1;
  other.s = -1;
}

SocketConnect& SocketConnect :: operator=(SocketConnect∧ other) {
    this→s = std::move(other.s);
  this→skt = std::move(other.skt);
  other.skt = -1;
  other.s = -1;
    return *this;
}

bool SocketConnect :: isValid() {
  return this→skt ≠ -1;
}
```

may 13, 19 11:49 **common_rsa.h** Page 1/1

```
#ifndef RSA_H
#define RSA_H

#include <iostream>

class Rsa {
    public:
        Rsa() = default;
        //Dado dos uint16_t y un uin8_t, encprita el uint16_t
        //con el exp y mod dados.
        uint32_t calcularRsa(
            const uint16_t &hash,
            const uint8_t &exp,
            const uint16_t &mod);
        ~Rsa() = default;
};

#endif
```

may 13, 19 11:49 **common_rsa.cpp** Page 1/1

```
#include "common_rsa.h"
#include <cstdlib>

uint32_t Rsa :: calcularRsa(
    const uint16_t &hash,
    const uint8_t &exp,
    const uint16_t &mod){
    uint32_t ret = 0;
    for (size_t i = 0; i<4; i++) {
        uint32_t result = hash >> (i * 8) & 0xff;
        uint32_t base = result;
        for (size_t j = 1; j<exp; j++) {
            result = (result * base) % mod;
        }
        ret = ret + (result << (i*8));
    }
    return ret;
}
```

may 13, 19 11:49 **common_protocolo.h** Page 1/1

```
#ifndef PROTOCOLO_H
#define PROTOCOLO_H

#include <string.h>
#include <string>
#include <iostream>
#include "common_socket_connect.h"

class Protocolo {
    private:
        SocketConnect *socket;
        void enviarInt32(uint32_t num);
        void recibirInt32(uint32_t *num);
    public:
        explicit Protocolo(SocketConnect *socket);
        ~Protocolo() = default;
        Protocolo& operator>>(std::string &cadena);
        Protocolo& operator>>(uint8_t &num);
        Protocolo& operator>>(uint16_t &num);
        Protocolo& operator>>(uint32_t &num);
        Protocolo& operator<<(uint8_t &num);
        Protocolo& operator<<(uint16_t &num);
        Protocolo& operator<<(uint32_t &num);
        Protocolo& operator<<(std::string &cadena);
};

#endif
```

may 13, 19 11:49 **common_protocolo.cpp** Page 1/2

```
#define UINT8_SIZE 1
#define UINT16_SIZE 2
#define UINT32_SIZE 4
#define TAM 10

#include "common_protocolo.h"
#include <arpa/inet.h>
#include <string>

void obtenerTam(uint32_t &tam_total, uint32_t &r) {
  if (tam_total - r - TAM > tam_total) {
    r = tam_total%TAM;
  } else {
    r = TAM;
  }
}

Protocolo& Protocolo :: operator<<(std::string &cadena) {
  uint32_t tam = (uint32_t)cadena.size();
  uint32_t enviado = 0;
  this->enviarInt32(tam);
  while (tam > enviado) {
    char cad[TAM];
    uint32_t e = 0;
    obtenerTam(tam,e);
    strncpy(cad, cadena.data()+enviado,e);
    this->socket->enviarMensaje(cad,e);
    enviado+=e;
  }
  return *this;
}

Protocolo& Protocolo :: operator>>(std::string &cadena) {
  uint32_t tam = 0;
  uint32_t recibido = 0;
  this->recibirInt32(&tam);
  while (tam > recibido) {
    char buffer[TAM];
    uint32_t r = 0;
    obtenerTam(tam,r);
    this->socket->recibirMensaje(buffer,r);
    buffer[r] = '\0';
    cadena += buffer;
    recibido += r;
  }
  return *this;
}

Protocolo :: Protocolo(SocketConnect *socket) {
  this->socket = socket;
}

Protocolo& Protocolo :: operator>>(uint8_t &num) {
  this->socket->recibirMensaje((char*)&num,UINT8_SIZE);
  return *this;
}

Protocolo& Protocolo :: operator>>(uint16_t &num) {
  uint16_t numero_red(0);
  this->socket->recibirMensaje((char*)&numero_red,UINT16_SIZE);
  num = ntohs(numero_red);
  return *this;
}

Protocolo& Protocolo :: operator>>(uint32_t &num) {
  this->recibirInt32(&num);
```

```
67    return *this;
68  }
69
70  Protocolo& Protocolo :: operator<<(uint8_t &num) {
71    this→socket→enviarMensaje((char*)&num,UINT8_SIZE);
72    return *this;
73  }
74
75  Protocolo& Protocolo :: operator<<(uint16_t &num) {
76    uint16_t numero_red = htons(num);
77    this→socket→enviarMensaje((char*)&numero_red,UINT16_SIZE);
78    return *this;
79  }
80
81  Protocolo& Protocolo :: operator<<(uint32_t &num) {
82    this→enviarInt32(num);
83    return *this;
84  }
85
86  void Protocolo :: enviarInt32(uint32_t num) {
87    uint32_t numero_red = htonl(num);
88    this→socket→enviarMensaje((char*)&numero_red,UINT32_SIZE);
89  }
90
91  void Protocolo :: recibirInt32(uint32_t *num) {
92    uint32_t numero_red(0);
93    this→socket→recibirMensaje((char*)&numero_red,UINT32_SIZE);
94    *num = ntohl(numero_red);
95  }
```



```
1   #ifndef HASH_H
2   #define HASH_H
3
4   #include <string>
5   #include <iostream>
6
7   class Hash {
8       public:
9           Hash() = default;
10          //Recibe como parametro un string y calcula su hash.
11          uint16_t calcularHash(const std::string &cadena);
12          ~Hash() = default;
13  };
14
15  #endif
```

may 13, 19 11:49 **common_hash.cpp** Page 1/1

```cpp
#include <string>
#include "common_hash.h"

uint16_t Hash :: calcularHash(const std::string &cadena) {
    uint16_t suma = 0;
    for (char c : cadena) {
        suma += (uint16_t) c;
    }
    return suma;
}
```

may 13, 19 11:49 **common_fecha.h** Page 1/1

```cpp
#ifndef FECHA_H
#define FECHA
#include <ctime>
#include <iostream>
#include <string>
#include <chrono>
#include <map>

class Fecha {
    private:
        std::tm* fecha_actual;
        std::chrono::system_clock::time_point t_actual;

    public:
        Fecha();
        ~Fecha() = default;
        //Devuelve la fecha actual.
        std::string getFechaActual();

        //Devuelve la fecha 30 dias despues de la fecha actual
        std::string getFecha30DiasDespues();
};

#endif
```

may 13, 19 11:49 **common_fecha.cpp** Page 1/1

```cpp
#include "common_fecha.h"
#include <string>
#define NAME_SIZE 4

Fecha :: Fecha() {
    t_actual = std::chrono::system_clock::now();
    std::time_t t = std::chrono::system_clock::to_time_t(t_actual);
    fecha_actual = std::localtime(&t);
}

std::string Fecha :: getFechaActual() {
    char buf[NAME_SIZE];
    strftime(buf,NAME_SIZE,"%b",fecha_actual);
    std::string nombre_mes(buf);
    std::string dia = std::to_string(fecha_actual→tm_mday);
    if (dia.size() ≡ 1) dia = '0'+dia;
    std::string hora = std::to_string(fecha_actual→tm_hour);
    if (hora.size() ≡ 1) hora = '0'+hora;
    std::string min = std::to_string(fecha_actual→tm_min);
    if (min.size() ≡ 1) min = '0'+min;
    std::string seg = std::to_string(fecha_actual→tm_sec);
    if (seg.size() ≡ 1) seg = '0'+ seg;
    std::string anio = std::to_string(fecha_actual→tm_year + 1900);
    return nombre_mes + ' ' + dia + ' ' + hora + ':'
    + min + ':' + seg + ' ' + anio;
}

std::string Fecha :: getFecha30DiasDespues() {
    std::chrono::duration<int,std::ratio<60*60*24*30> > treinta_dias(1);
    std::chrono::system_clock::time_point en_treinta_dias =
      t_actual+treinta_dias;
    std::time_t t = std::chrono::system_clock::to_time_t(en_treinta_dias);
    fecha_actual = std::localtime(&t);
    std::string tiempo = getFechaActual();
    std::chrono::system_clock::time_point t_actual = t_actual − treinta_dias;
    t = std::chrono::system_clock::to_time_t(t_actual);
    fecha_actual = std::localtime(&t);
    return tiempo;
}
```

may 13, 19 11:49 **common_error.h** Page 1/1

```cpp
#ifndef ERROR_H
#define ERROR_H
#include <iostream>
#include <stdexcept>
#include <exception>
#include <typeinfo>

class Error : public std::exception {
    public:
        int error_desconexion();
};

#endif
```

may 13, 19 11:49 **common_error.cpp** Page 1/1

```cpp
#include <iostream>
#include <stdexcept>
#include <exception>
#include <typeinfo>
#include "common_error.h"

int Error :: error_desconexion() {
    throw std::runtime_error("Error: la conexion fue perdida");
}
```

may 13, 19 11:49 **common_claves.h** Page 1/1

```cpp
#ifndef CLAVES_H
#define CLAVES_H
#include <iostream>
#include <fstream>
#include <string>
#include "common_archivo.h"

class Claves : public Archivo {
    private:
        uint8_t exp_privado;
        uint8_t exp_publico;
        uint16_t mod;

    public:
        using Archivo::Archivo;

        //Se encarga de parsear correctamente los datos del
        //archivo leido
        void parser();
        uint8_t getModulo();
        uint16_t getExponentePrivado();
        uint16_t getExponentePublico();
};

#endif
```

may 13, 19 11:49 **common_claves.cpp** Page 1/1

```cpp
#include <string>
#include "common_claves.h"

void Claves :: parser() {
    std::string buf;
    this→leerArchivo(buf);
    size_t pos = 0;
    int num_de_palabra = 0;
    std::string token;
    std::string delimitador = " ";
    while ((pos = buf.find(delimitador)) ≠ std::string::npos) {
        token = buf.substr(0, pos);
        if (num_de_palabra ≡ 0) exp_publico = (uint8_t)stoi(token);
        if (num_de_palabra ≡ 1) exp_privado = (uint8_t)stoi(token);
        buf.erase(0, pos + delimitador.length());
        num_de_palabra++;
    }
    mod = (uint16_t)stoi(buf);
}

uint8_t Claves :: getModulo() {
    return mod;
}

uint16_t Claves :: getExponentePrivado() {
    return exp_privado;
}

uint16_t Claves :: getExponentePublico() {
    return exp_publico;
}
```

may 13, 19 11:49 **common_certificado.h** Page 1/1

```cpp
#ifndef CERTIFICADO_H
#define CERTIFICADO_H

#include <iostream>
#include <string>

class Certificado {
    private:
        uint32_t serial_number;
        std::string subject;
        std::string issuer;
        std::string fecha_inicial;
        std::string fecha_final;
        uint16_t modulo;
        uint8_t exponente;
        std::string certificado_completo;

        void agregarSerialNumber();
        std::string getHexadecimal(int largo,int n);
        void agregarValidity();
        void agregegarPublicKeyInfo();
        void extraerSubject();
        void extraerSerialNumber();
        void extraerIssuer();
        void extraerFechas();
        void extraerModulo();
        void extraerExponente();
        void asignarCadena(size_t pos_inicial,size_t pos_final,
          std::string &cadena);
        void calcularPosicion(size_t &pos_inicial,size_t &pos_final,
          std::string &inicio, std::string &fin);

    public:
        Certificado() = default;
        ~Certificado() = default;

        //Arma el certificado en el formato pedido. El certificado
        //debe tener todos los campos con informacion valida.
        void parser();

        //Calcula el hash de certificado_completo
        uint16_t calcularHash();

        //Calcula el rsa del certificado.
        uint32_t calcularRsa(const uint32_t &hash ,
          const uint8_t &exp, const uint16_t &mod);

        uint32_t getSerialNumber();
        std::string getSubject();
        std::string getIssuer();
        std::string getFechaInicio();
        std::string getFechaFin();
        uint16_t getModulo();
        uint8_t getExponente();
        std::string getCertificado();
        void setAtributos(uint32_t sn, std::string &nombre,
         std::string &fecha_inicio ,std::string &fecha_fin,
         uint16_t mod, uint8_t exp);
        void setCertificado(std::string &certificado);
};

#endif
```

```
#include <string>
#include <iomanip>
#include <sstream>
#include "common_certificado.h"
#include "common_fecha.h"
#include "common_hash.h"
#include "common_rsa.h"

void Certificado :: setAtributos(uint32_t sn,std::string
    &nombre,std::string &fecha_inicio ,std::string &fecha_fin,
    uint16_t mod, uint8_t exp) {
        serial_number = sn;
        subject = nombre;
        if (fecha_inicio ≡ "") {
            Fecha fecha;
            fecha_inicial = fecha.getFechaActual();
            fecha_final = fecha.getFecha30DiasDespues();
        } else {
            fecha_inicial = fecha_inicio;
            fecha_final = fecha_fin;
        }
        modulo = mod;
        exponente = exp;
}

std::string Certificado :: getHexadecimal(int largo,int n) {
    std::stringstream stream;
    stream << std::hex << n;
    std::string result(stream.str());
    int tam = result.size();
    for (int i=0; i< largo-tam; i++) {
        result = "0"+result;
    }
    result = "(0x" + result + ")";
    return result;
}

void Certificado :: agregarSerialNumber() {
    certificado_completo += "serial number: "+ std::to_string(serial_number);
    std::string hexa = getHexadecimal(8,serial_number);
    certificado_completo += " "+hexa+"\n\t";
}

void Certificado :: agregarValidity() {
    certificado_completo += "validity:\n\t\tnot before: ";
    certificado_completo += fecha_inicial+"\n\t\t";
    certificado_completo += "not after: ";
    certificado_completo += fecha_final+"\n\t";
}

void Certificado :: agregegarPublicKeyInfo() {
    certificado_completo += "subject public key info:\n\t\t";
    certificado_completo += "modulus: " + std::to_string(modulo) + " ";
    std::string hexa_mod = getHexadecimal(4,modulo);
    certificado_completo += hexa_mod + "\n\t\t";
    certificado_completo += "exponent: " + std::to_string(exponente)+" ";
    std::string hexa_exp = getHexadecimal(2,exponente);
    certificado_completo += hexa_exp;
}

void Certificado :: parser() {
    certificado_completo = "certificate:\n\t";
    agregarSerialNumber();
    certificado_completo += "subject: "+subject+"\n\t";
    certificado_completo += "issuer: Taller de programacion 1\n\t";
    agregarValidity();
```



```
    agregegarPublicKeyInfo();
}

uint16_t Certificado :: calcularHash() {
    Hash hash;
    return hash.calcularHash(certificado_completo);
}

uint32_t Certificado :: calcularRsa(const uint32_t &hash ,
  const uint8_t &exp, const uint16_t &mod) {
    Rsa rsa;
    return rsa.calcularRsa(hash,exp,mod);
}

uint32_t Certificado :: getSerialNumber() {
    return this→serial_number;
}
std::string Certificado :: getSubject() {
    return this→subject;
}
std::string Certificado :: getIssuer() {
    return "Taller de programacion 1";
}
std::string Certificado :: getFechaInicio() {
    return this→fecha_inicial;
}
std::string Certificado :: getFechaFin() {
    return this→fecha_final;
}
uint16_t Certificado :: getModulo() {
    return this→modulo;
}
uint8_t Certificado :: getExponente() {
    return this→exponente;
}

std::string Certificado :: getCertificado() {
    return this→certificado_completo;
}

void Certificado :: asignarCadena(size_t pos_inicial,
  size_t pos_final,std::string &cadena) {
    for (size_t i=pos_inicial; i< pos_final; i++) {
        cadena+= this→certificado_completo[i];
    }
}
void Certificado :: calcularPosicion(size_t &pos_inicial,size_t &pos_final,
    std::string &inicio, std::string &fin) {
        pos_inicial = this→certificado_completo.find(inicio)+inicio.size();;
        pos_final = this→certificado_completo.find(fin,pos_inicial);
}

void Certificado :: extraerSubject() {
    std::string subject = "subject: ";
    std::string salto = "\n";
    size_t pos_inicial(0);
    size_t pos_final(0);
    this→calcularPosicion(pos_inicial,pos_final,subject,salto);
    this→asignarCadena(pos_inicial,pos_final,this→subject);
}

void Certificado :: extraerSerialNumber() {
    std::string sn = "serial number: ";
    std::string parentesis = "(";
    size_t pos_inicial(0);
    size_t pos_final(0);
```

may 13, 19 11:49 **common_certificado.cpp** Page 3/3

```
133     this→calcularPosicion(pos_inicial,pos_final,sn,parentesis);
134     sn = "";
135     this→asignarCadena(pos_inicial,pos_final,sn);
136     this→serial_number = (uint32_t)std::stoi(sn);
137 }
138
139 void Certificado :: extraerIssuer() {
140     std::string issuer = "issuer: ";
141     std::string salto = "\n";
142     size_t pos_inicial(0);
143     size_t pos_final(0);
144     this→calcularPosicion(pos_inicial,pos_final,issuer,salto);
145     this→asignarCadena(pos_inicial,pos_final,this→issuer);
146 }
147
148 void Certificado :: extraerFechas() {
149     std::string f_inicial = "not before: ";
150     std::string salto = "\n";
151     size_t pos_inicial(0);
152     size_t pos_final(0);
153     this→calcularPosicion(pos_inicial,pos_final,f_inicial,salto);
154     this→asignarCadena(pos_inicial,pos_final,this→fecha_inicial);
155     std::string f_final = "not after: ";
156     this→calcularPosicion(pos_inicial,pos_final,f_final,salto);
157     this→asignarCadena(pos_inicial,pos_final,this→fecha_final);
158 }
159
160 void Certificado :: extraerModulo() {
161     std::string modulus = "modulus: ";
162     std::string parentesis = "(";
163     size_t pos_inicial(0);
164     size_t pos_final(0);
165     this→calcularPosicion(pos_inicial,pos_final,modulus,parentesis);
166     std::string mod;
167     this→asignarCadena(pos_inicial,pos_final,mod);
168     this→modulo = (uint16_t)std::stoi(mod);
169 }
170
171 void Certificado :: extraerExponente() {
172     std::string exponent = "exponent: ";
173     std::string parentesis = "(";
174     size_t pos_inicial(0);
175     size_t pos_final(0);
176     this→calcularPosicion(pos_inicial,pos_final,exponent,parentesis);
177     std::string exp;
178     this→asignarCadena(pos_inicial,pos_final,exp);
179     this→exponente = (uint16_t)std::stoi(exp);
180 }
181
182 void Certificado :: setCertificado(std::string &certificado) {
183     this→certificado_completo = certificado.substr(0,certificado.size()−1);
184     this→extraerSerialNumber();
185     this→extraerSubject();
186     this→extraerIssuer();
187     this→extraerFechas();
188     this→extraerModulo();
189     this→extraerExponente();
190 }
```

may 13, 19 11:49 **common_archivo.h** Page 1/1

```
1  #ifndef ARCHIVO_H
2  #define ARCHIVO_H
3
4  #include <iostream>
5  #include <fstream>
6  #include <string>
7
8  class Archivo {
9      private:
10         std::fstream archivo;
11
12     public:
13         explicit Archivo(std::string &nombre_archivo);
14
15         //Se encarga de leer los datos dentro de archivo
16         //y colocarlos en el buffer recibido por parametro.
17         void leerArchivo(std::string &buf);
18         virtual void parser() = 0;
19         ~Archivo();
20 };
21
22 #endif
```

may 13, 19 11:49 **common_archivo.cpp** Page 1/1

```cpp
#include <string>
#include "common_archivo.h"

Archivo :: Archivo(std::string &nombre_archivo) :
    archivo(nombre_archivo, std::fstream::in | std::fstream::out) {
}

void Archivo :: leerArchivo(std::string &buf) {
    std::string buf_aux;
    while (std::getline(archivo, buf_aux)) {
        buf += buf_aux+"\n";
    }
}

Archivo :: ~Archivo() {
    archivo.close();
}
```

may 13, 19 11:49 **client_request.h** Page 1/1

```cpp
#ifndef REQUEST_CLIENTE_H
#define REQUEST_CLIENTE_H

#include <iostream>
#include <fstream>
#include <string>
#include "common_archivo.h"

class RequestCliente : public Archivo {
    private:
        std::string nombre;
        std::string fecha_inicial;
        std::string fecha_final;
        bool ingreso_fechas;

    public:
        using Archivo::Archivo;

        //Se encarga de parsear correctamente los datos
        //del archivo leido.
        void parser();
        bool ingresoFechas();
        std::string getNombre();
        std::string getFechaInicial();
        std::string getFechaFinal();
};

#endif
```

may 13, 19 11:49 **client_request.cpp** Page 1/1

```cpp
#include <string>
#include "client_request.h"

void RequestCliente :: parser() {
    std::string buf;
    leerArchivo(buf);
    size_t pos = 0;
    int num_de_palabra = 0;
    std::string token;
    std::string delimitador = "\n";
    ingreso_fechas = false;
    while ((pos = buf.find(delimitador)) ≠ std::string::npos) {
        token = buf.substr(0, pos);
        if (num_de_palabra ≡ 0) nombre.assign(move(token));
        if (num_de_palabra ≡ 1) fecha_inicial.assign(move(token));
        if (num_de_palabra ≡ 2) fecha_final.assign(move(token));
        if (num_de_palabra>0) ingreso_fechas = true;
        buf.erase(0, pos + delimitador.length());
        num_de_palabra++;
    }
}

std::string RequestCliente :: getNombre(){
    return nombre;
}

std::string RequestCliente :: getFechaInicial(){
    return fecha_inicial;
}

std::string RequestCliente :: getFechaFinal(){
    return fecha_final;
}

bool RequestCliente :: ingresoFechas() {
    return ingreso_fechas;
}
```

may 13, 19 11:49 **client.cpp** Page 1/2

```cpp
#include <iostream>
#include <string>
#include <string.h>
#include <vector>
#include "common_hash.h"
#include "common_rsa.h"
#include "common_claves.h"
#include "client_request.h"
#include "client_clave_server.h"
#include "common_socket_connect.h"
#include "common_protocolo.h"
#include "client_comando.h"
#include "client_archivo_certificado.h"
#define ARGS_ESPERADOS 7
#define HOST 1
#define PORT 2
#define MODO 3
#define REQUEST 4
#define CLAVES_CLIENTE 5
#define CLAVES_SERVER 6

int main(int argc, char *argv[]) {
    if (argc ≠ ARGS_ESPERADOS) {
        std::cout << "Error: argumentos invalidos." << std::endl;
        return 0;
    }
    try {
        std::string modo = argv[MODO];

        std::string archivo_clave_cliente = argv[CLAVES_CLIENTE];
        Claves claves_cliente(archivo_clave_cliente);
        claves_cliente.parser();

        std::string archivo_clave_pub_srv = argv[CLAVES_SERVER];
        ClavePublicaServer clave_pub_server(archivo_clave_pub_srv);
        clave_pub_server.parser();

        SocketConnect socket_cliente;
        if (¬socket_cliente.addrinfo(argv[HOST],argv[PORT])) {
            std::cout << "Problema en addrinfo" << std::endl;
            return 0;
        }

        if (¬socket_cliente.conectar()) {
            std::cout << "Problema en connectar" << std::endl;
            return 0;
        }

        ComandoCliente comando(&socket_cliente);

        if (modo ≡ "new") {
            std::string archivo_request = argv[REQUEST];
            RequestCliente request(archivo_request);
            request.parser();
            comando.comandoNew(request,claves_cliente,
            clave_pub_server);

        } else if (modo ≡ "revoke") {
            std::string archivo_certificado = argv[REQUEST];
            ArchivoCertificado arch_certificado(archivo_certificado);
            arch_certificado.parser();
            comando.comandoRevoke(arch_certificado,
            claves_cliente,clave_pub_server);

        } else {
            std::cout << "Error: argumentos invalidos." << std::endl;
```

may 13, 19 11:49 **client.cpp** Page 2/2

```
67              return 0;
68          }
69          socket_cliente.cerrarConexion();
70      } catch (const std::runtime_error& e) {
71          std::cout << e.what() << std::endl;
72          return 0;
73      } catch (...) {
74          std::cout << "Ocurrio un error inesperado" << std::endl;
75      }
76      return 0;
77  }
```

may 13, 19 11:49 **client_comando.h** Page 1/1

```
1   #ifndef CLIENT_COMANDO_H
2   #define CLIENT_COMANDO_H
3   #include "common_protocolo.h"
4   #include "common_certificado.h"
5   #include "common_socket_connect.h"
6   #include "client_request.h"
7   #include "client_clave_server.h"
8   #include "common_claves.h"
9   #include "client_archivo_certificado.h"
10  #include <string>
11
12
13  class ComandoCliente {
14      private:
15          Protocolo protocolo;
16
17          bool enviarModo(uint8_t m);
18          bool comandoNewRecibirRespuesta(Protocolo
19            &protocolo, ClavePublicaServer &clave_server,Claves
20            &clave_cliete);
21          void guardarCertificado(Certificado &certificado);
22          uint8_t respuestaDelServidor();
23
24
25      public:
26          explicit ComandoCliente(SocketConnect *socket);
27          ~ComandoCliente() = default;
28
29          //Recibe como parametro un SocketConnect, RequestCliente,
30          //Claves y ClavePublicaServer. Se encarga de que se
31          //ejecute correctamente el comando new del cliente.
32          bool comandoNew(RequestCliente &request
33          , Claves &claves_cliente,ClavePublicaServer &clave_server);
34
35          //Recibe como parametro un SocketConnect, RequestCliente,
36          //Claves y ClavePublicaServer. Se encarga de que se
37          //ejecute correctamente el comando revoke del cliente.
38          bool comandoRevoke(ArchivoCertificado
39            &certificado,Claves &claves_cliente,ClavePublicaServer
40            &clave_server);
41  };
42
43  #endif
```

```
1   #include "client_comando.h"
2   #include <string>
3
4   ComandoCliente :: ComandoCliente(SocketConnect *socket) : protocolo(socket) {
5   }
6
7   bool ComandoCliente :: enviarModo(uint8_t m) {
8       this→protocolo << m;
9       return true;
10  }
11
12  void ComandoCliente :: guardarCertificado(Certificado &certificado) {
13      std::string nombre_archivo = certificado.getSubject() + ".cert";
14      std::ofstream ofs(nombre_archivo, std::ofstream::out);
15      ofs << certificado.getCertificado();
16      ofs.close();
17  }
18
19  bool ComandoCliente :: comandoNewRecibirRespuesta(
20      Protocolo &protocolo, ClavePublicaServer &clave_server,
21      Claves &clave_cliete) {
22      uint8_t se_agrego = -1;
23      this→protocolo >> se_agrego;
24
25      if (se_agrego ≡ 1) {
26          std::cout << "Error: ya existe un certificado." << std::endl;
27          return false;
28      }
29
30      uint32_t sn=-1;
31      this→protocolo >> sn;
32
33      std::string subject;
34      this→protocolo >> subject;
35
36      std::string issuer;
37      this→protocolo >> issuer;
38
39      std::string fecha_inicial;
40      this→protocolo >> fecha_inicial;
41
42      std::string fecha_final;
43      this→protocolo >> fecha_final;
44
45      uint16_t mod(0);
46      this→protocolo >> mod;
47
48      uint8_t exp(0);
49      this→protocolo >> exp;
50
51      uint32_t huella(0);
52      this→protocolo >> huella;
53
54      Certificado certificado;
55      certificado.setAtributos(sn,subject,fecha_inicial,fecha_final,mod,exp);
56      certificado.parser();
57
58      uint8_t exp_privado_cliente = clave_cliete.getExponentePrivado();
59      uint8_t exp_servidor = clave_server.getExponente();
60      uint16_t mod_servidor = clave_server.getModulo();
61      uint32_t hash = certificado.calcularHash();
62      uint32_t rsa = certificado.calcularRsa(huella,exp_privado_cliente,mod);
63      rsa = certificado.calcularRsa(rsa,exp_servidor,mod_servidor);
64      std::cout << "Huella del servidor: " << huella << std::endl;
65      std::cout << "Hash del servidor: " << rsa << std::endl;
66      std::cout << "Hash calculado: " << hash << std::endl;
```



```
67      if (hash ≠ rsa) {
68          std::cout << "Error: los hashes no coinciden." << std::endl;
69          return false;
70      }
71      this→guardarCertificado(certificado);
72      return true;
73  }
74
75  bool ComandoCliente :: comandoNew(
76      RequestCliente &request, Claves &claves_cliente,
77      ClavePublicaServer &clave_server) {
78      uint8_t m = 0;
79      if (¬this→enviarModo(m)) return false;
80
81      std::string nombre = request.getNombre();
82      this→protocolo << nombre;
83
84      uint16_t mod = claves_cliente.getModulo();
85      this→protocolo << mod;
86
87      uint8_t exp_publico = claves_cliente.getExponentePublico();
88      this→protocolo << exp_publico;
89
90      if (request.ingresoFechas()) {
91          std::string fecha_inicial = request.getFechaInicial();
92          this→protocolo << fecha_inicial;
93
94          std::string fecha_final = request.getFechaFinal();
95          this→protocolo << fecha_final;
96      }
97
98      uint8_t rta;
99      if (this→comandoNewRecibirRespuesta(protocolo,
100       clave_server,claves_cliente)) {
101         rta = 0;
102     } else {
103         rta = 1;
104     }
105
106     this→protocolo << rta;
107
108     return true;
109 }
110
111 uint8_t ComandoCliente :: respuestaDelServidor() {
112     uint8_t rta = 3;//inicio invadio
113     this→protocolo >> rta;
114     return rta;
115 }
116
117
118 bool ComandoCliente :: comandoRevoke(
119     ArchivoCertificado &arch_certificado, Claves &claves_cliente
120     ,ClavePublicaServer &clave_server) {
121     Certificado certificado;
122     std::string certificado_texto = arch_certificado.getCertificado();
123     certificado.setCertificado(certificado_texto);
124
125     uint8_t m(1);
126     this→protocolo << m;
127
128     uint32_t sn = certificado.getSerialNumber();
129     this→protocolo << sn;
130
131     std::string subject = certificado.getSubject();
132     this→protocolo << subject;
```

```
133
134     std::string issuer = certificado.getIssuer();
135     this→protocolo << issuer;
136
137     std::string fecha_incial = certificado.getFechaInicio();
138     this→protocolo << fecha_incial;
139
140     std::string fecha_final= certificado.getFechaFin();
141     this→protocolo << fecha_final;
142
143     uint16_t modulo = certificado.getModulo();
144     this→protocolo << modulo;
145
146     uint8_t exponente = certificado.getExponente();
147     this→protocolo << exponente;
148
149     uint32_t hash = certificado.calcularHash();
150     std::cout << "Hash calculado: " << hash << std::endl;
151     uint8_t exp_cliente = claves_cliente.getExponentePrivado();
152     uint16_t mod_cliente = claves_cliente.getModulo();
153     uint32_t rsa = certificado.calcularRsa(hash,exp_cliente,mod_cliente);
154     std::cout <<"Hash encriptado con la clave privada: "<< rsa << std::endl;
155     uint8_t exp_servidor = clave_server.getExponente();
156     uint16_t mod_servidor = clave_server.getModulo();
157     rsa = certificado.calcularRsa(rsa,exp_servidor,mod_servidor);
158     std::cout << "Huella enviada: " << rsa << std::endl;
159
160     this→protocolo << rsa;
161
162     uint8_t rta = this→respuestaDelServidor();
163     if (rta ≡ 1) {
164         std::cout << "Error: usuario no registrado." << std::endl;
165         return false;
166     }
167
168     if (rta ≡ 2) {
169         std::cout << "Error: los hashes no coinciden." << std::endl;
170         return false;
171     }
172
173     return true;
174 }
```



```
1  #ifndef CLAVE_PUB_SERVER_H
2  #define CLAVE_PUB_SERVER_H
3
4  #include <iostream>
5  #include <string>
6  #include <fstream>
7  #include "common_archivo.h"
8
9  class ClavePublicaServer : public Archivo {
10     private:
11         uint16_t mod;
12         uint8_t exp;
13
14     public:
15         using Archivo::Archivo;
16
17         //Se encarga de parsear correctamente los datos
18         //del archivo leido.
19         void parser();
20         uint8_t getModulo();
21         uint16_t getExponente();
22 };
23
24 #endif
```

may 13, 19 11:49 **client_clave_server.cpp** Page 1/1

```cpp
#include <string>
#include "client_clave_server.h"


void ClavePublicaServer :: parser() {
    std::string buf;
    leerArchivo(buf);
    size_t pos = 0;
    int num_de_palabra = 0;
    std::string token;
    std::string delimitador = " ";
    while ((pos = buf.find(delimitador)) ≠ std::string::npos) {
        token = buf.substr(0, pos);
        if (num_de_palabra ≡ 0) exp = (uint8_t)stoi(token);
        buf.erase(0, pos + delimitador.length());
        num_de_palabra++;
    }
    mod = (uint16_t)stoi(buf);
}

uint8_t ClavePublicaServer :: getModulo(){
    return mod;
}

uint16_t ClavePublicaServer :: getExponente(){
    return exp;
}
```

may 13, 19 11:49 **client_archivo_certificado.h** Page 1/1

```cpp
#ifndef ARCHIVO_CERTIFICADO_H
#define ARCHIVO_CERTIFICADO_H

#include <iostream>
#include <fstream>
#include <string>
#include "common_archivo.h"

class ArchivoCertificado : public Archivo {
    private:
        std::string certificado;

    public:
        using Archivo::Archivo;

        //Se encarga de parsear correctamente los datos del
        //archivo leido
        void parser();

        std::string getCertificado();
};

#endif
```

may 13, 19 11:49 **client_archivo_certificado.cpp** Page 1/1

```cpp
#include <string>
#include "client_archivo_certificado.h"

void ArchivoCertificado :: parser() {
    std::string buf;
    leerArchivo(buf);
    this→certificado.assign(buf);
}

std::string ArchivoCertificado :: getCertificado(){
    return certificado;
}
```

may 13, 19 11:49 **Table of Content** Page 1/1

Table of Contents