Nuestro proyecto consta de 4 clases que realizan la conexión entre computadoras (servidor-cliente) incluyendo una interfaz gráfica utilizada para la interacción con el usuario para realizar las peticiones, y una clase que aplica el algoritmo AES para la encriptación y desencriptación del mensaje. El mensaje viaja en forma de String en un archivo JSON, que es creado, leído y modificado utilizando con soporte de la librería json.simple.1.1.1.jar. A continuación, está el detalle de cada clase (códigos en anexos).

[**AES.java**](#AES)

Definimos los objetos necesarios para la encriptación y desencriptación utilizando el algoritmo AES. Se utiliza una llave (key) derivada de un “password” establecido en el código, y se establecen 2 objetos que son llamados por la máquina de estado y el cliente para que se encripte y desencripte el archivo JSON.

**[RemoteInterface.java](#remote)**

En este archivo se establece la interfaz remota necesaria para que el cliente se comunique con el servidor y tenga acceso a los métodos que requiera utilizar, ya sea lectura y actualización de las variables (read y update) o lectura y escritura del archivo JSON (operation). Para el rastreo de errores a la hora de ejecutar se utiliza RemoteException en los métodos (tanto en la interfaz como en el cliente, servidor y máquina de estado).

[**Server.java**](#Servidor)

Recibe la orden por parte del cliente a través de la interfaz remota y ejecuta la solicitud del cliente utilizando una máquina de estado (State\_Machine.java). Se establece el puerto necesario (utilizando la clase LocateRegistry) para realizar la conexión con el cliente y una “palabra clave” para identificarse como host de métodos (en este caso read y update) que se pueden solicitar, y comunica en consola un mensaje de texto si ya está listo para comunicarse con otra computadora.

[**State\_Machine.java**](#Maquinae)

Es la máquina de estado que utiliza el servidor para realizar las operaciones solicitadas por el cliente. Define las variables “a” y “b” y los métodos set, mult, add, read y update, para realizar las acciones de asignar valor, multiplicar, sumar, leer y actualizar respectivamente. Dependiendo de lo solicitado por el cliente, los valores de “a” o “b” se sumarán o multiplicarán al valor ingresado y guardarán actualizados, o solamente hará una lectura de sus valores actuales (el cliente recibe a través de la interfaz gráfica los valores actualizados de “a” y “b”). También incluye el método utilizado para escritura y lectura del archivo JSON, que lo desencripta, lee el contenido, lo modifica y lo convierte en String, para finalmente encriptarlo.

[**Client\_GUI\_Dialog.java**](#Cliente)

El código de este archivo se comunica con el servidor directamente utilizando el sistema RMI de java, la dirección IP, y la “palabra clave” establecida por el servidor para que se identifique como host de las acciones que desea realizar el cliente, este último recibiendo un objeto remoto con los métodos disponibles a utilizar.

Una vez solicitados los requerimientos a través de la interfaz gráfica, el servidor ejecuta el método remoto en su instancia de la máquina de estado (State\_Machine.java) para realizar la o las solicitudes que posteriormente el cliente recibe y muestra al usuario en la interfaz gráfica. Las solicitudes son enviadas al servido en forma de archivo JSON, que el cliente escribe y encripta para su envío.

Anexos.

**AES.java**

package Client;

import javafx.util.converter.BigIntegerStringConverter;

import javax.crypto.Cipher;

import javax.crypto.KeyGenerator;

import javax.crypto.SecretKey;

import javax.crypto.spec.GCMParameterSpec;

import java.io.ByteArrayOutputStream;

import java.io.DataOutputStream;

import java.math.BigInteger;

import java.util.Base64;

public class AES {

private SecretKey key;

private final int KEY\_SIZE = 128;

private final int T\_LEN = 128;

private Cipher encryptionCipher;

// private String prueba="Variable no válida.\n";

public void init() throws Exception {

KeyGenerator generator = KeyGenerator.getInstance("AES");

generator.init(KEY\_SIZE);

key = generator.generateKey();

}

public String encrypt(String message) throws Exception {

byte[] messageInBytes = message.getBytes();

encryptionCipher = Cipher.getInstance("AES");

encryptionCipher.init(Cipher.ENCRYPT\_MODE, key);

byte[] encryptedBytes = encryptionCipher.doFinal(messageInBytes);

return encode(encryptedBytes);

}

public String decrypt(String encryptedMessage) throws Exception {

byte[] messageInBytes = decode(encryptedMessage);

Cipher decryptionCipher = Cipher.getInstance("AES");

GCMParameterSpec spec = new GCMParameterSpec(T\_LEN, encryptionCipher.getIV());

decryptionCipher.init(Cipher.DECRYPT\_MODE, key, spec);

byte[] decryptedBytes = decryptionCipher.doFinal(messageInBytes);

return new String(decryptedBytes);

}

private String encode(byte[] data) {

return Base64.getEncoder().encodeToString(data);

}

private byte[] decode(String data) {

return Base64.getDecoder().decode(data);

}

public static void main(String[] args) {

try {

AES aes = new AES();

aes.init();

String encryptedMessage = aes.encrypt("Variable no válida.\n");

String decryptedMessage = aes.decrypt(encryptedMessage);

//String encryptedMessage = aes.encrypt("123");

System.err.println("Encrypted Message : " + encryptedMessage);

System.err.println("Decrypted Message : " + decryptedMessage);

} catch (Exception ignored) {

}

}

}

**RemoteInterface.java**

package src.Server;

import org.json.simple.parser.ParseException;

import java.rmi.\*;

public interface RemoteInterface extends Remote{

//Variables id number

static final int a\_ID = 0;

static final int b\_ID = 1;

//Operations id number

static final int set\_Oper = 0;

static final int add\_Oper = 1;

static final int mult\_Oper = 2;

static final int read\_Oper = 3;

// Metodos remotos

double read(int var) throws RemoteException;

boolean update(int var, int oper, double val) throws RemoteException;

String operation(String JSON\_Oper) throws Exception;

}

**Server.java**

package src.Server;

import java.rmi.\*;

import java.rmi.registry.LocateRegistry;

import java.rmi.registry.Registry;

import java.rmi.server.UnicastRemoteObject;

public class Server implements Remote{

public static void main(String[] args) throws RemoteException{

try {

String name = "Compute";

State\_Machine engine = new State\_Machine();

//State\_Machine stub =

// (State\_Machine) UnicastRemoteObject.exportObject(engine, 0);

Registry registry = LocateRegistry.createRegistry(5099);

registry.rebind(name, engine);

System.out.println("ComputeEngine bound");

} catch (Exception e) {

System.err.println("ComputeEngine exception:");

e.printStackTrace();

}

}

}

**State\_Machine.java**

package src.Server;

import java.rmi.RemoteException;

import java.rmi.registry.LocateRegistry;

import java.rmi.registry.Registry;

import java.rmi.server.UnicastRemoteObject;

import org.json.simple.\*;

import org.json.simple.parser.JSONParser;

import org.json.simple.parser.ParseException;

import src.AES;

public class State\_Machine extends UnicastRemoteObject implements RemoteInterface {

//Operation variables

private double a;

private double b;

private AES aes;

//Constructor

public State\_Machine() throws Exception {

super();

a = 0;

b = 0;

aes = new AES();

try {

aes.init();

} catch (Exception e) {

throw new RuntimeException(e);

}

}

// Métodos para a

private double get\_A() throws RemoteException{

return a;

}

private boolean set\_A(double val) throws RemoteException{

a = val;

return true;

}

private boolean add\_A(double val) throws RemoteException{

a = a + val;

return true;

}

private boolean mult\_A(double val) throws RemoteException{

a = a \* val;

return true;

}

// Métodos para b

private double get\_B() throws RemoteException{

return b;

}

private boolean set\_B(double val) throws RemoteException{

b = val;

return true;

}

private boolean add\_B(double val) throws RemoteException{

b = b + val;

return true;

}

private boolean mult\_B(double val) throws RemoteException{

b = b \* val;

return true;

}

public double read(int var) throws RemoteException{

if(var == a\_ID)

return get\_A();

else

return get\_B();

}

public boolean update(int var, int oper, double val) throws RemoteException{

switch (oper){

case set\_Oper:

if(var == a\_ID)

return set\_A(val);

else

return set\_B(val);

case add\_Oper:

if(var == a\_ID)

return add\_A(val);

else

return add\_B(val);

case mult\_Oper:

if(var == a\_ID)

return mult\_A(val);

else

return mult\_B(val);

}

return false;

}

public String operation(String JSON\_Oper) throws Exception {

JSONParser parser = new JSONParser();

String request = aes.decrypt(JSON\_Oper);

JSONObject json = (JSONObject)parser.parse(request);

JSONObject json\_out = new JSONObject();

int operation = Integer.parseInt((String)json.get("operation"));

int variable = Integer.parseInt((String)json.get("variable"));

double value = Double.parseDouble((String)json.get("value"));

double read\_result;

boolean update\_result;

json\_out.put("pad","");

if(operation == read\_Oper)

json\_out.put("result",read(variable)+"");

else{

json\_out.put("result",Boolean.toString(update(variable, operation, value)));

}

String output = aes.encrypt(json\_out.toString());

return output;

}

}

**Client\_GUI\_Dialog.java**

package src.Client;

import src.AES;

import src.Server.RemoteInterface;

import javax.swing.\*;

import java.awt.\*;

import java.awt.event.\*;

import java.rmi.Naming;

import java.rmi.RemoteException;

import org.json.simple.\*;

import org.json.simple.parser.JSONParser;

import org.json.simple.parser.ParseException;

public class Client\_GUI\_Dialog extends JDialog {

private JPanel contentPane;

private JButton conectarButton;

private JRadioButton rad\_a;

private JRadioButton rad\_b;

private JComboBox cmb\_oper;

private JTextField txt\_val;

private JButton btn\_exe;

private JTextArea txta\_console;

private JTextField txt\_IP;

private JTextField txt\_Port;

private JTextPane pnl\_console;

private int variable = 0;

private int operacion = 0;

private int value = 0;

private String var\_letra = "";

private RemoteInterface service;

private AES aes;

private void onOper() {

int index = cmb\_oper.getSelectedIndex();

if (index > 1) {

operacion = index - 2;

txt\_val.setEnabled(true);

btn\_exe.setEnabled(true);

} else if (index == 1) {

operacion = 3;

btn\_exe.setEnabled(true);

} else {

btn\_exe.setEnabled(false);

}

}

private void onExec() {

try {

JSONObject json = new JSONObject();

JSONParser parser = new JSONParser();

JSONObject answer\_json;

String answer;

String request;

json.put("operation", operacion + "");

json.put("variable", variable + "");

if (operacion == 3) {

//Se completa el json

json.put("value", "0");

//Se encripta el mensaje

request = aes.encrypt(json.toString());

//Se hace la solicitud al servidor

answer = aes.decrypt(service.operation(request));

//Se transforma el mensaje recibido a un json

answer\_json = (JSONObject) parser.parse(answer);

txta\_console.append("Variable " + var\_letra + "= " + answer\_json.get("result") + "\n");

} else if (operacion > 3) {

txta\_console.append("Operación no válida.\n");

} else {

if (variable == 0 || variable == 1) {

value = Integer.parseInt(txt\_val.getText());

//Se completa el json

json.put("value", value + "");

//Se encripta el mensaje

request = aes.encrypt(json.toString());

//Se hace la solicitud al servidor

answer = aes.decrypt(service.operation(request));

//Se transforma el mensaje recibido a un json

answer\_json = (JSONObject) parser.parse(answer);

txta\_console.append("Respuesta del servidor: " + answer\_json.get("result") + "\n");

json.clear();

json.put("operation", "3");

json.put("variable", variable + "");

json.put("value", "0");

request = aes.encrypt(json.toString());

answer = aes.decrypt(service.operation(request));

answer\_json.clear();

answer\_json = (JSONObject) parser.parse(answer);

txta\_console.append("Valor actual de la variable " + var\_letra + ": " + answer\_json.get("result") + "\n");

cmb\_oper.setSelectedIndex(0);

txt\_val.setText("");

txt\_val.setEnabled(false);

btn\_exe.setEnabled(false);

} else {

txta\_console.append("Variable no válida.\n");

}

}

} catch (RemoteException e) {

txta\_console.append("Error: " + e.toString() + "\n");

} catch (NumberFormatException e) {

txta\_console.append("Error: Entrada no válida. Escribir un número entero.\n");

} catch (ParseException e) {

txta\_console.append("Error: No se puede parsear el Json recibido.\n");

throw new RuntimeException(e);

} catch (Exception e) {

txta\_console.append("Error: No se puede encriptar/desencriptar.\n");

throw new RuntimeException(e);

}

}

private void onConnect() {

try {

String lookup\_Name = "rmi://" + txt\_IP.getText() + ":" + txt\_Port.getText() + "/Compute";

service = (RemoteInterface) Naming.lookup(lookup\_Name);

txta\_console.append("Status: Se estableció la conexión al servidor.\n");

aes = new AES();

aes.init();

rad\_a.setEnabled(true);

rad\_b.setEnabled(true);

conectarButton.setEnabled(false);

} catch (Exception e) {

txta\_console.append("Error: No se puede establecer la conexión al servidor.\n");

}

}

private void onRad\_A() {

variable = 0;

var\_letra = "a";

cmb\_oper.setEnabled(true);

}

private void onRad\_B() {

variable = 1;

var\_letra = "b";

cmb\_oper.setEnabled(true);

}

public static void main(String[] args) {

Client\_GUI\_Dialog dialog = new Client\_GUI\_Dialog();

dialog.pack();

dialog.setVisible(true);

System.exit(0);

}