LSB-Chat: plataforma segura de intercambio de datos

Diseño de Software Seguro

Valentín Alexandro Spataru Racotta
Universidad Aeronáutica en Querétaro

1 de abril de 2020

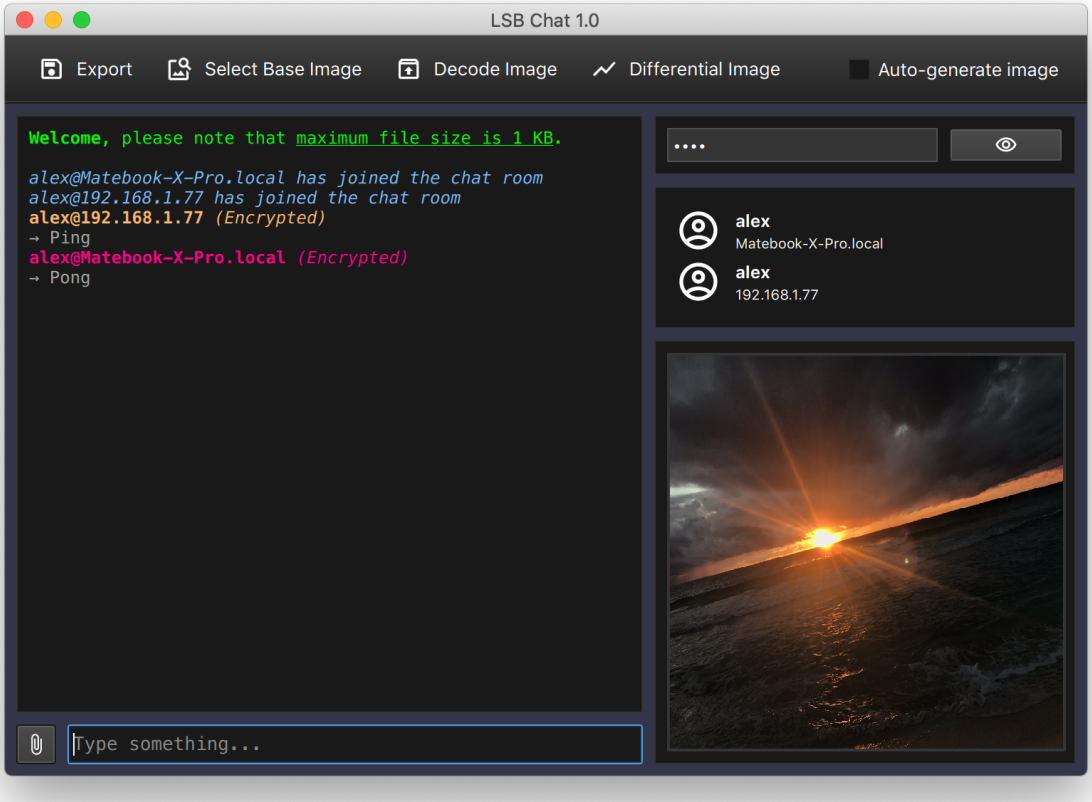
Índice

1. Introducción

El proyecto consiste en el diseño e implementación de una plataforma segura de intercambio de mensajes / archivos a través de una red local.

Los datos se cifran y posteriormente se ocultan con un algoritmo de esteganografía LSB dentro de una imagen, la cual se envía a un chat grupal en la red local. La imagen puede ser auto-generada por el programa con píxeles de colores aleatorios, o puede ser seleccionada por el usuario.

Para leer los datos originales, el usuario que recibe los datos debe conocer la contraseña de cifrado. De lo contrario, él o ella no podrá leer los datos recibidos.



Captura de pantalla del software desarrollado

A continuación, se desarrollan los requerimientos, arquitectura, evaluaciones de seguridad, y otros temas vistos en el curso. También se presenta un plan de desarrollo con fechas, etapas de desarrollo y responsabilidades. Al final del documento se presenta y documenta el proceso de desarrollo de la aplicación.

Nota: El código fuente y documentación actual del proyecto se pueden encontrar en el repositorio público de GitHub. El código y la documentación de este proyecto fue escrito exclusivamente en inglés, ya que que el inglés sigue siendo el idioma más común entre la mayoría de los desarrolladores de proyectos open-source e investigadores.

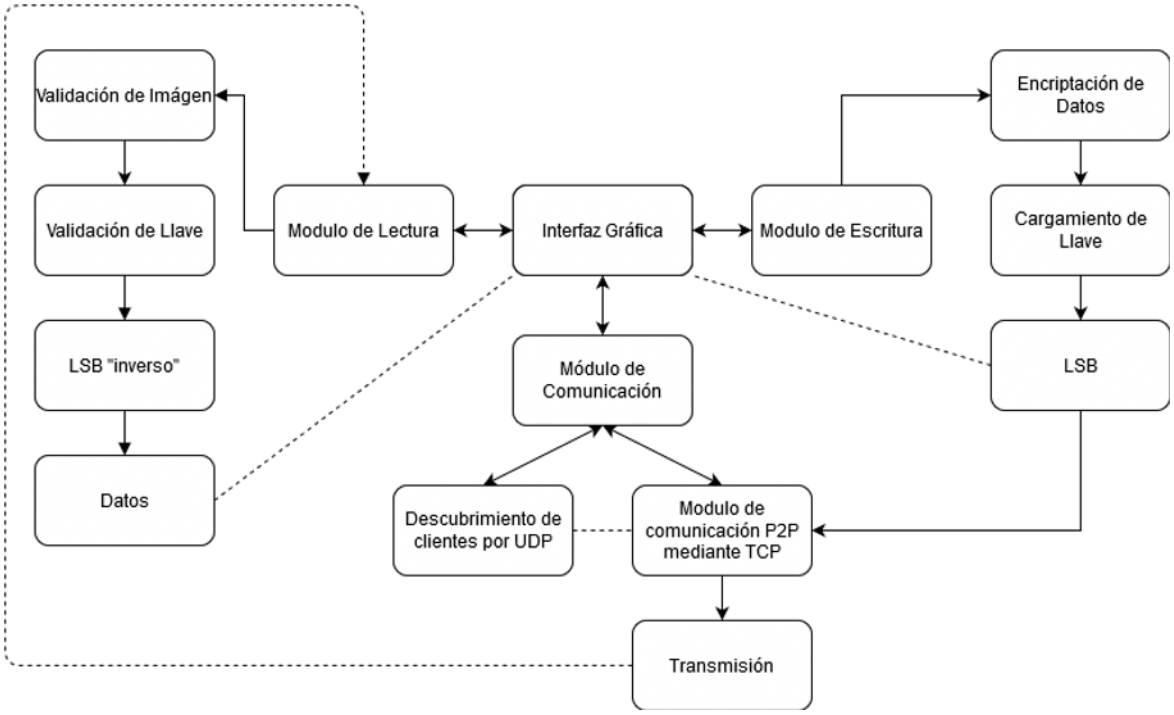
2. Requerimientos del Sistema

A continuación, se muestra la tabla de requerimientos del sistema:

Número de Req.	Descripción del Requerimiento
1.0	**Requerimientos de transmisión de datos**
1.1	La transmisión de datos se debe de realizar mediante TCP.
1.2	La transmisión de datos se debe de realizar utilizando el protocolo TCP para evitar perdida de información.
1.3	El descubrimiento de otros clientes de la aplicación se debe de realizar mediante tramas de broadcast UDP.
1.4	La comunicación de datos debe de ser P2P (peer-to-peer).
2.0	**Requerimientos del contenedor de datos**
2.1	Los datos deben de ser representados usando el formato JSON.
2.2	El documento JSON debe de tener los siguientes campos: Tipo de Mensaje, Tamaño de Mensaje, Datos de Mensaje y Nombre de Archivo.
2.3	Los datos del mensaje deben de ser convertidos a la representación de datos binarios “Base64”.
3.0	**Requerimientos de encriptación de datos**
3.1	La encriptación de los datos se debe de realizar utilizando cifrado cesar, combinado con un cifrado XOR.
3.2	Los datos descifrados no pueden ser guardados en la memoria permanente de la computadora. Es decir, el usuario debe de cargar la imagen con los datos cifrados al programa para poder leer los datos contenidos dentro de la imagen.
3.3	La encriptación cesar solo debe de operar sobre los carácteres imprimibles de la tabla ASCII.
3.3.1	El alfabeto válido para la encriptación cesar corresponde al siguiente rango de la tabla ASCII (33, 126).
4.0	**Requerimientos del módulo de LSB**
4.1	Los datos cifrados se deben de contener en la matriz diagonal de los datos RGB de la imagen, empezando desde el pixel inferior derecha.
4.2	Los datos escritos en la imagen deben de empezar con la siguiente cabecera $ TAMAÑO-DATOS $
4.3	El módulo LSB debe de ser capaz de generar una imagen con pixeles de valor aleatorio para acomodar los datos dados por el usuario.
4.4	El módulo LSB debe de ser capaz de cargar una imagen seleccionada por el usuario y usar esa imagen para ocultar los datos dados por el usuario.
4.5	Los datos de la imagen deben de ser exportados al sistema/red usando el formato “PNG” con un factor de calidad $q = 100$ para evitar perdida de información.
5.0	**Requerimientos de la interfaz gráfica**
5.1	La interfaz gráfica y los módulos del programa deben de ser implementados utilizando el framework Qt.
5.2	La interfaz gráfica debe de consistir de: ▪ Una vista previa de la imagen procesada mediante LSB. ▪ Una lista de los usuarios conectados mediante la red local. ▪ Un campo de texto con los mensajes mandados por los usuarios. ▪ Un campo de texto que permita al usuario establecer la llave de encriptación.

3. Arquitectura del Sistema

A continuación, se muestra la arquitectura del sistema a implementar.



Se escogió utilizar la interfaz gráfica como elemento central del programa. La interfaz gráfica interactúa con los distintos módulos del programa para generar una experiencia interactiva que cumpla con los requisitos del programa.

Al separar el programa en distintos módulos, se simplifica el desarrollo, mantenimiento y mejora de este. Al mismo tiempo, se simplifica el proceso de encontrar y mitigar errores en el desarrollo del proyecto.

Descripción de cada bloque/módulo del sistema:

- *Módulo de Comunicación:* Se encarga del descubrimiento de clientes en la red local (LAN) y administrar las conexiones individuales entre las distintas instancias de la aplicación.
- *Módulo de Escritura:* Se encarga de encriptar una cadena de datos e insertar los datos encriptados en una imagen mediante LSB.
- *Módulo de Lectura:* Se encarga de obtener los datos encriptados dentro de una imagen y - posteriormente - descifrar los datos y mostrarlos al usuario.

4. Evaluación de Seguridad

Mediante SNA, se identificaron los siguientes riesgos:

- Si la llave es compartida entre los clientes del programa, es posible efectuar un ataque MID (man in the middle), resultando en el intercambio de información con personas fuera de confianza.
- Se puede generar un ataque DDOS al solicitar que la computadora de cualquier usuario realice mas conexiones de las cuales puede manejar.
- Si el cliente recibe una imagen invalida, se puede generar un error al intentar procesarla para obtener los datos.
- Si el usuario intenta escribir información demasiado grande para la imagen, se puede generar:
  - Perdida de información.
  - Crash al intentar acceder a memoria no asignada.

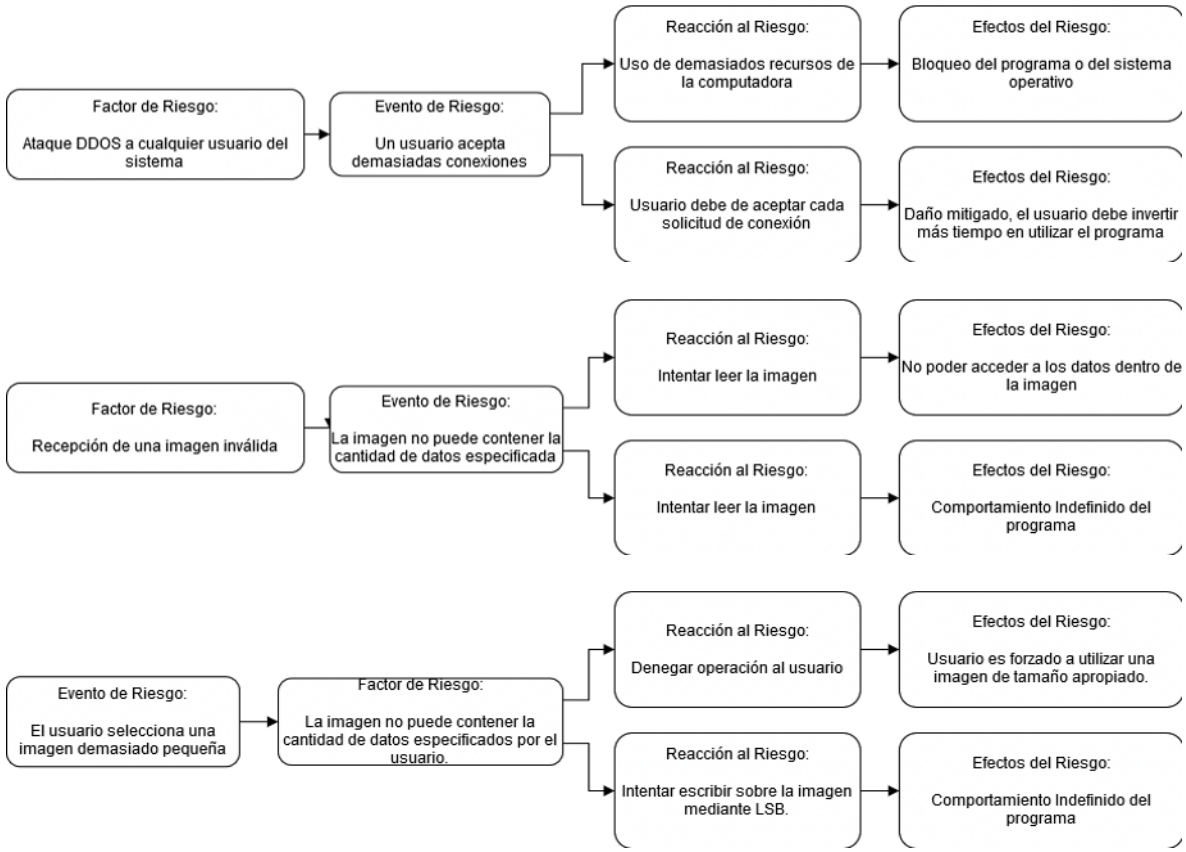
Para mitigar tales riesgos, se decidió tomar las siguientes acciones:

- La llave debe de ser previamente conocida por ambos usuarios antes de intercambiar información.
- Para poder realizar una solicitud de conexión con cualquier usuario, es necesario saber la contraseña de conexión para ese usuario especifico.
- La imagen debe de ser validada antes de ser procesada. Por lo cual, los primeros 4 bytes del LSB corresponden al tamaño de los datos. Si la imagen es demasiado pequeña para soportar el LSB de tales datos, la imagen no va a ser procesada.
- De manera similar, el programa debe de validar que los datos pueden caber dentro de una imagen al realizar el LSB sobre tal imagen.

5. Análisis de Riesgo

5.1. Escenarios de Riesgo

A continuación, se muestran algunos escenarios de riesgo que se identificaron:



5.2. Matriz de Riesgos

A continuación, se muestra una matriz tentativa de riesgos. Sin embargo, es importante tomar en consideración que probablemente se vayan a identificar mas riesgos y problemas de seguridad conforme se avance en las etapas de desarrollo del proyecto.

Alto	-	Ataque MID	Ataque DDOS
Medio	Recepción de Imagen Invalida	-	-
Bajo	Procesamiento de imagen invalida	-	-
Impacto/Probabilidad	**Alto**	**Medio**	**Bajo**

5.3. Características de Seguridad

Debido al diseño modular del programa, se deben de considerar los factores de riesgo para cada módulo. Por lo tanto, se debe de tomar en consideración cada modulo por separado para poder "garantizar" la seguridad.

Ventajas de la arquitectura:

- Al hacer un análisis separado por cada módulo, se pueden mitigar los factores de riesgo desde su origen.
- Cada módulo se encarga de una tarea en específico. Al realizar las pruebas adecuadas sobre cada módulo y posteriormente las pruebas de integración, se puede obtener un producto confiable y que cumpla los requisitos de calidad de los usuarios.

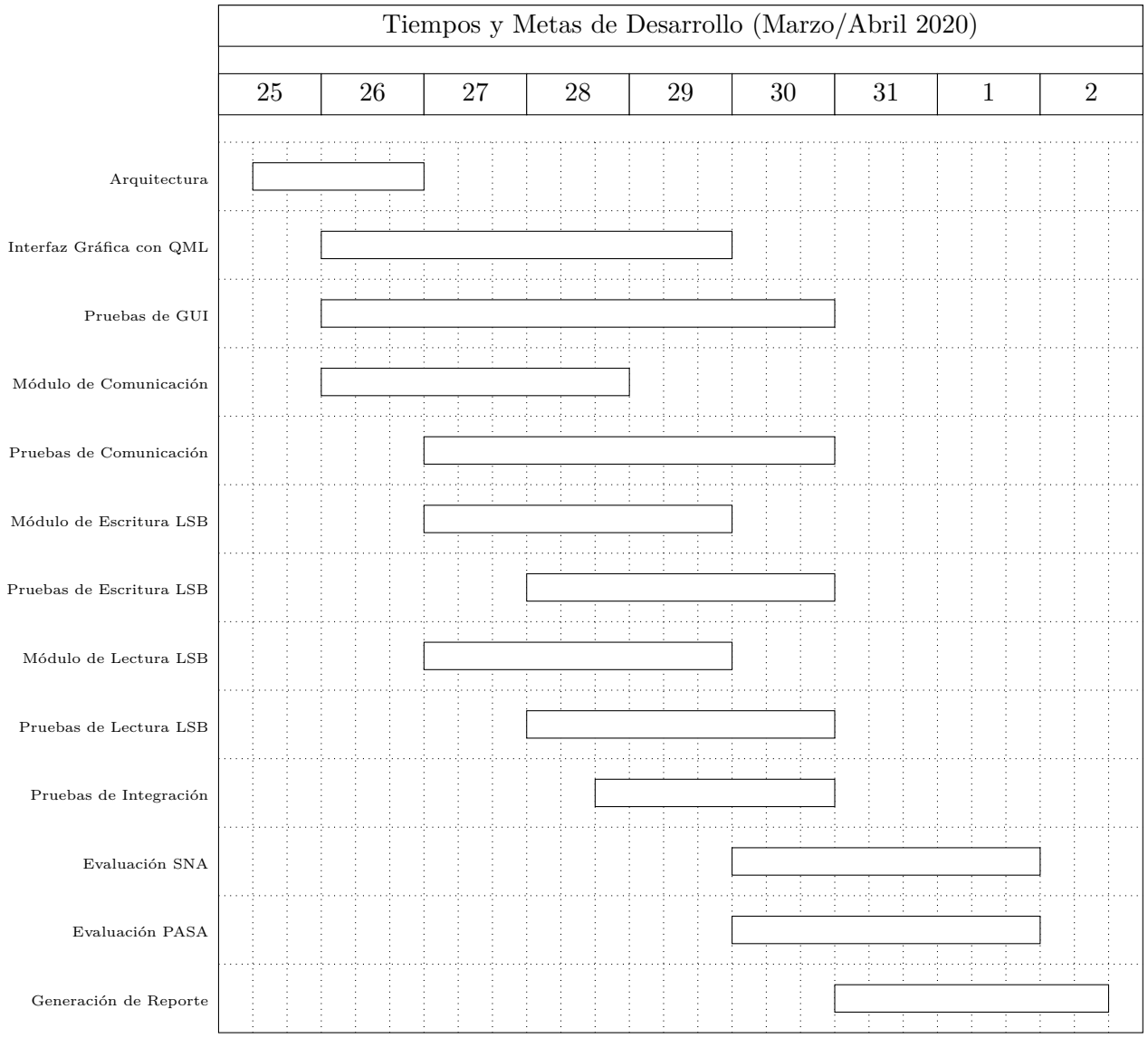
6. Propuesta de Programación del Sistema

El programa será desarrollado utilizando la plataforma Qt, debido a los siguientes motivos:

- Qt es un framework multi-plataforma, y contiene varias bibliotecas que simplifican el desarrollo de aplicaciones enormemente. (más información...).
- Qt integra clases para poder trabajar, modificar y procesar imágenes, lo cual puede simplificar el desarrollo del módulo de LSB. (más información...).
- Qt cuenta con un sistema de procesamiento de datos binarios muy robusto, lo cual puede servir tanto para el modulo de encriptación, como para el módulo de comunicación. (más información...).
- Qt cuenta con las bibliotecas necesarias para implementar interfaces gráficas nativas. (más información...).
- Qt cuenta con un sistema de eventos, el cual se puede integrar fuertemente a todos los módulos que hagan uso de la clase QObject, tales como elementos gráficos, administradores de archivos y sockets para redes. (más información...).
- Qt implementa un robusto módulo de comunicaciones por redes, y incluye soporte para trabajar directamente con sockets TCP y sockets UDP. (más información...).
- Finalmente, Qt es un proyecto open-source, por lo tanto, el framework tiene muchas contribuciones por parte de empresas y expertos en áreas específicas, lo cual lo hace adecuado para el desarrollo de aplicaciones de todo tipo.

6.1. Plan de Desarrollo

Debido a que el proyecto se desarrolla individualmente, todas las responsabilidades caen en el autor del proyecto. A continuación, se muestra un diagrama de Gantt con los principales hitos del proyecto y las fechas estimadas de finalización.



Actualmente, se esta utilizando GitHub para manejar el proyecto, con la finalidad de:

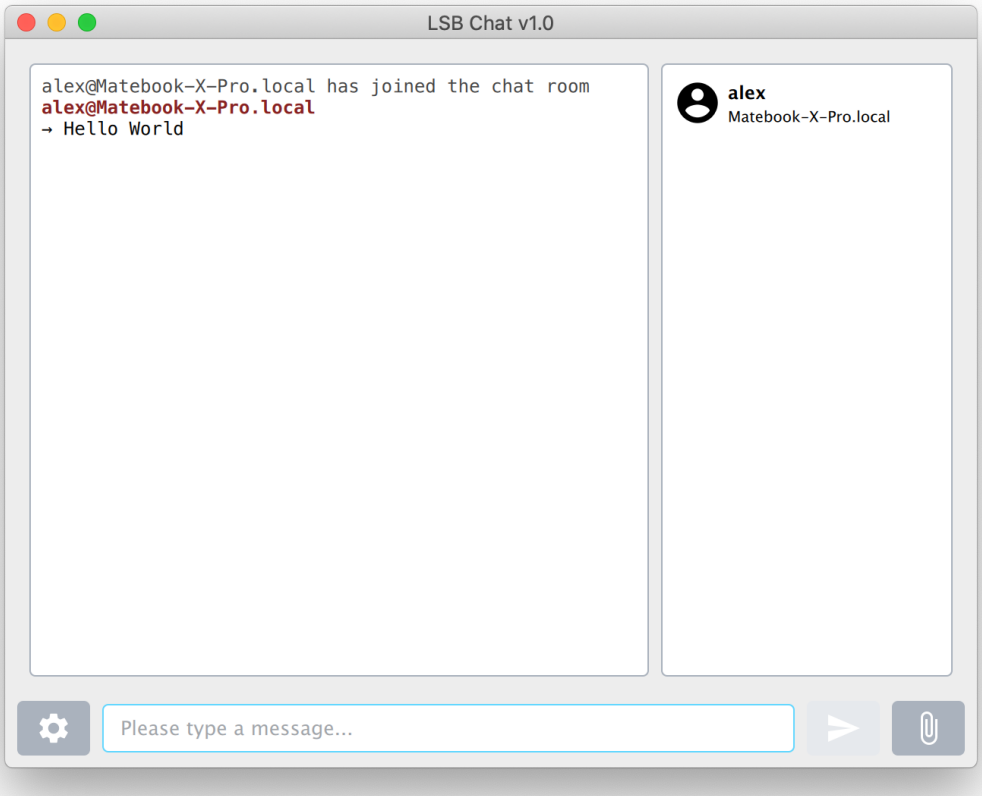
- Tener un respaldo del código del proyecto y tener a la mano las diferentes versiones del proyecto.
- Evidenciar las distintas fases de desarrollo del proyecto.
- Invitar a la participación de otras personas para reportar problemas y/o mejorar el proyecto.

7. Proceso y Notas de Desarrollo

En este apartado, se documenta el proceso de desarrollo de la aplicación y los cambios que han surgido durante la etapa de codificación y pruebas.

La mayoría de los cambios fueron en la interfáz gráfica y en el procesamiento necesario para poder asegurar que los distintos tipos de mensajes mandados por los usuarios (incluyendo archivos binarios) puedan ser mandados sin perdida de información.

Por ejemplo, aquí se muestra una captura de pantalla de la interfáz gráfica preliminar. A comparación de la captura de pantalla mostrada en la introducción de este documento, podemos observar que la GUI experimentó cambios muy significativos.



Primera versión de la interfaz gráfica

7.1. Módulo de comunicaciones P2P

El módulo de comunicaciones cumple con los siguientes objetivos:

- Descubrir otras instancias de la aplicación en la red local del dispositivo en donde se esta ejecutando la aplicación.
- Coordinar dos instancias de la aplicación de tal forma que se pueda realizar el intercambio de datos entre dos computadoras.

7.1.1. Funcionamiento General

El módulo consiste de los siguientes sub-módulos:

- **Conexión P2P**, se encarga de intercambiar y validar paquetes/tramas entre dos computadoras
- **Administrador de Conexiones**, se encarga de mandar tramas mediante un *socket* UDP para descubrir otras instancias la aplicación en la red local y se asegura que no existan conexiones duplicadas entre dos clientes.
- **Servidor TCP**, "escucha" las solicitudes de conexión y actúa como el primer paso para establecer una conexión P2P entre dos computadoras.
- **Cliente**, interfaz entre el resto de la aplicación y los sub-módulos de comunicación.

Nota: se eligió utilizar un *socket* TCP para implementación de la conexión P2P para asegurar que no existan perdidas de datos durante la transmisión de las imágenes. Aunque este tipo de sócalo puede ser más lento, uno de los objetivos del programa es resguardar la información transmitida y asegurar que la información recibida por un usuario sea la misma que otro usuario haya mandado.

Los archivos que conforman este módulo son:

- `git://program/src/Comms/NetworkComms.h`
- `git://program/src/Comms/NetworkComms.cpp`
- `git://program/src/Comms/P2P_Connection.h`
- `git://program/src/Comms/P2P_Connection.cpp`
- `git://program/src/Comms/P2P_Manager.h`
- `git://program/src/Comms/P2P_Manager.cpp`
- `git://program/src/Comms/TCP_Listener.h`
- `git://program/src/Comms/TCP_Listener.cpp`

7.2. Módulo de LSB

El módulo de LSB se encarga de codificar los datos cifrados dentro de una imagen, la cual puede ser auto-generada por el programa o seleccionada por el usuario.

7.2.1. Escritura de datos

Para la escritura de datos, se utiliza el siguiente procedimiento:

```
Image lsbWrite(Image image, Byte[] data) {
        // Agregar tamaño de datos al arreglo
        Byte[] outData = '$' + data.length() + '$' + data;
        int bytesWritten = 0;

        // Calcular tamaño de la diagonal usando teo. de pitagoras
        int hyp = sqrt(2) * min(image.width(), image.height());

        // Escribir datos sobre la diagonal de la imagen
        for (i = 0, i < hyp, i += 3) {
                // Salir del loop si ya escribimos todos los datos
                if (bytesWritten >= outData.length())
                        break;

                // Obtener lista de bits
                Bit[8] bits = outData.at(i).bits;

                // Obtener pixeles actuales
                Pixel p1 = image.pixelAt(i, i);
                Pixel p2 = image.pixelAt(i + 1, i + 1);
                Pixel p3 = image.pixelAt(i + 2, i + 2);

                // Cambiar bit menos significativo en cada pixel
                p1.r = change_lsb(p1.r, bits[0]);
                p1.g = change_lsb(p1.g, bits[1]);
                p1.b = change_lsb(p1.b, bits[2]);
                p2.r = change_lsb(p2.r, bits[3]);
                p2.g = change_lsb(p2.g, bits[4]);
                p2.b = change_lsb(p2.b, bits[5]);
                p3.r = change_lsb(p3.r, bits[6]);
                p3.g = change_lsb(p3.g, bits[7]);

                // Escribir pixeles en imagen
                image.setPixel(i, i, p1);
                image.setPixel(i + 1, i + 1, p2);
                image.setPixel(i + 2, i + 2, p3);

                // Incrementar num. bytes escritos
                ++bytesWritten;
        }

        return image;
}
```

7.2.2. Lectura de datos

Para la lectura de datos, se utiliza el siguiente procedimiento:

```
Byte[] lsbRead(Image image) {
        // Init. variables
        int dataLen = 0;
        String dataLenStr;
        int headerCodeCount = 0;

        // Calcular longitud de diagonal
        int hyp = sqrt(2) * min(image.width(), image.height());

        // Decodificar informacion
        Byte[] data;
        for (i = 0, i < hyp, i += 3) {
                // Obtener pixeles
                Pixel p1 = image.pixelAt(i, i);
                Pixel p2 = image.pixelAt(i + 1, i + 1);
                Pixel p2 = image.pixelAt(i + 2, i + 2);

                // Obtener los bits menos significativos
                Bits[8] bits ;
                bits[0] = get_lsb(p1.r);
                bits[1] = get_lsb(p1.g);
                bits[2] = get_lsb(p1.b);
                bits[3] = get_lsb(p2.r);
                bits[4] = get_lsb(p2.g);
                bits[5] = get_lsb(p2.b);
                bits[6] = get_lsb(p3.r);
                bits[7] = get_lsb(p3.g);

                // Convertir cadena de bits a byte
                Byte byte = 0;
                for (i = 0, i < 8, ++i)
                        byte += (bits[i] << i);

                // Long. de datos desconocida, falta leer header
                if (dataLen == 0 && headerCodeCount < 2) {
                        if (byte == '$')
                                ++headerCodeCount;

                        else if (byte >= '0' && byte <= '9')
                                dataLenStr += byte;
                }

                // Ya leimos el header, actualizar long. datos
                // y leer primer byte
                else if (dataLen == 0 && headerCodeCount == 2) {
                        dataLen = dataLenStr.toInt();
                        data += byte;
                }
```

```
                // Leer datos
                else if (data.length() < dataLen)
                        data += byte;

                // Ya leimos toda la informacion, salir del ciclo
                else
                        break;
        }

        // Regresar datos obtenidos
        return data;
}
```

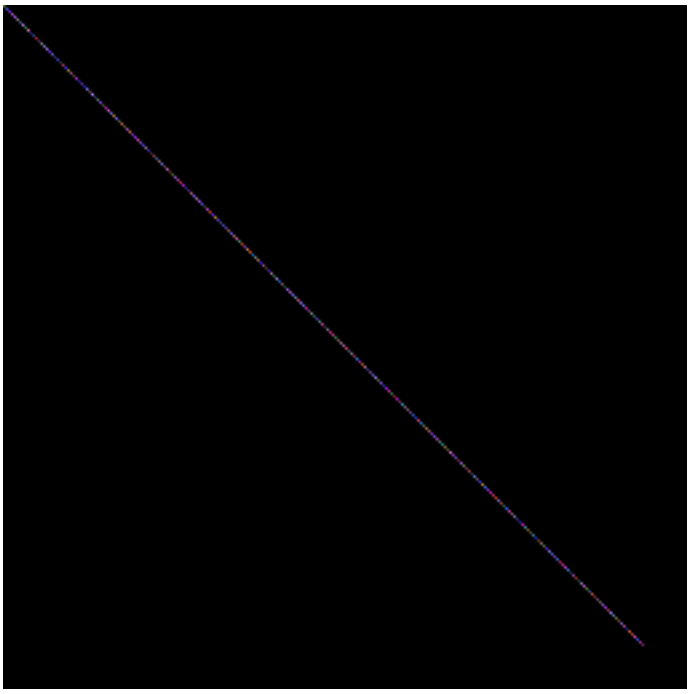
7.2.3. Generación de imágenes aleatorias y la imagen diferencial

El módulo LSB cuenta con la funcionalidad de generar imágenes aleatorias con el tamaño apropiado para los datos seleccionados por el usuario. Esta función puede ser muy útil, ya que incluso agregando la misma información dentro de la imagen, vamos a obtener imágenes diferentes entre las diferentes ejecuciones de la función "LSB-Write", lo cuál puede aumentar la seguridad.

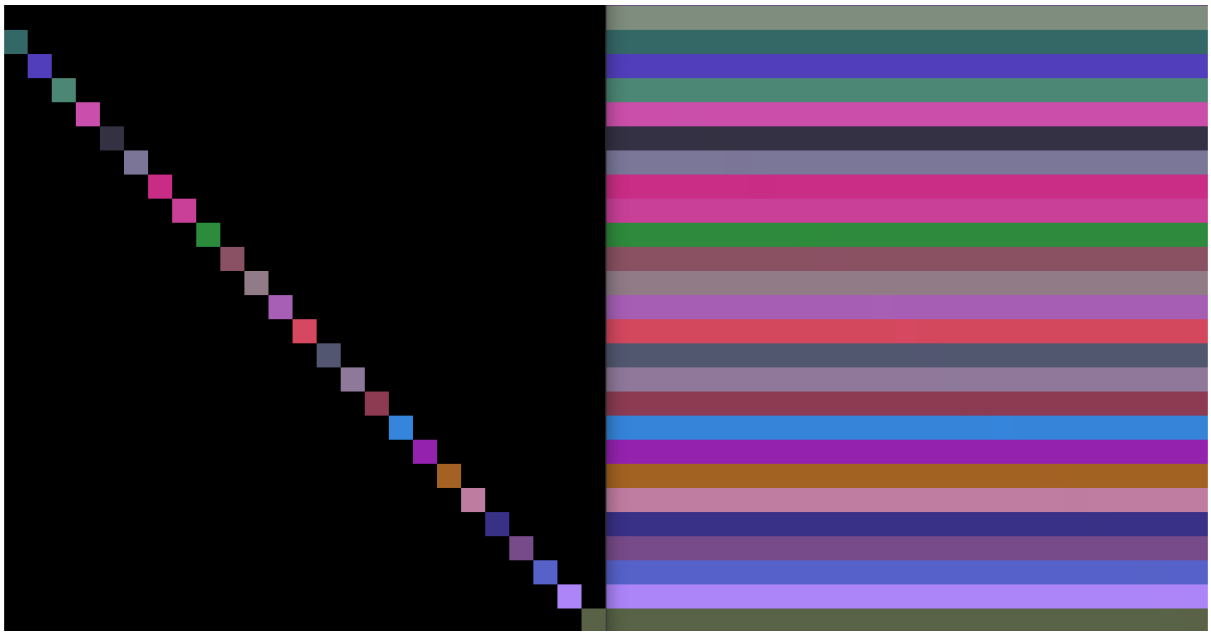
Sin embargo, el costo de esta funcionalidad es que existe cierta penalización en el tiempo de ejecución para mandar un mensaje. Debido a esto, esta funcionalidad es completamente opcional.

Como ejemplo, se muestra una imagen aleatoria generada por el programa. Dentro de esta imagen, se codifica el mensaje "Hola Mundo".

Para reducir el tiempo de ejecución de la función que genera la imagen aleatoria, solo se calculan los colores aleatorios para cada renglón de la imagen.
Adicionalmente, el módulo genera una imagen "diferencial", la cuál tiene un fondo negro y solo muestra los pixeles que han sido modificados por el algoritmo de escritura LSB. A continuación, se muestra el diferencial de la imagen anterior:



Finalmente, se muestra una comparación entre la imagen generada y la imagen diferencial:



Como ultima nota, la imagen diferencial se genera tanto para las imágenes autogeneradas como para las imágenes seleccionadas por el usuario. La imagen diferencial nunca se manda por la red local, ya que contiene la información necesaria para decodificar la imagen sin conocimiento de los algoritmos de escritura/lectura LSB. Con un poco de esfuerzo, la imagen diferencial puede ser la *llave* de cualquier algoritmo LSB.

7.3. Módulo de Encriptación

Antes de que los datos sean guardados dentro de una imagen usando el módulo LSB, el programa protege la información dada por el usuario utilizando dos algoritmos de cifrado consecutivamente.

7.3.1. Cifrado Cesar

El primer algoritmo de cifrado que se aplica a la información es el cifrado Cesar, el cual consiste de "rotar" el abecedario utilizado por la información. Una limitante de este algoritmo es que realmente no se ocupa una llave, si no solamente un *shift*, el cuál es un número que indica las posiciones a "rotar" el abecedario.

Otra limitante que tenemos es que el cifrado Cesar solo se puede aplicar a cierto rango de la tabla ASCII, además, el algoritmo no funciona bien con caracteres especiales, tales como "ñ".

A continuación, se muestra el pseudo-código para el cifrado Cesar:

```
#define ALFA_START  33
#define ALFA_END    126
#define ALFA_SIZE   ALFA_END - ALFA_START

Byte[] Caesar(Byte[] data, Byte[] key) {
        // Obtener y validar long. de llave
        int k = min(key.length(), ALFA_SIZE);

        // Rotar cada caracter de los datos
        Byte[] dataOut;
        for (int i = 0; i < data.length(); ++i) {
                // Obtener byte actual
                Byte byte = data[i];

                // El byte esta dentro del rango valido
                if (byte >= ALFA_START && byte <= ALFA_END) {
                        // Rotar byte
                        byte += k;

                        // Asegurar que seguimos en rango valido
                        if (byte > ALFA_END)
                                byte = byte - ALFA_END + ALFA_START - 1;
                }

                // Agregar byte a datos de salida
                dataOut[i] = byte;
        }

        // Regresar datos cifrados
        return dataOut;
}
```

Para el decifrado Cesar, se aplica el siguiente procedimiento:

```
Byte[] CaesarInverse(Byte[] data, Byte[] key) {
        // Obtener y validar long. de llave
        int k = min(key.length(), ALFA_SIZE);

        // Rotar cada caracter de los datos
        Byte[] dataOut;
        for (int i = 0; i < data.length(); ++i) {
                // Obtener byte actual
                Byte byte = data[i];

                // El byte esta dentro del rango valido
                if (byte >= ALFA_START && byte <= ALFA_END) {
                        // Rotar byte en direccion opuesta
                        byte -= k;

                        // Asegurar que seguimos en rango valido
                        if (byte < ALFA_START)
                                byte = byte + ALFA_END - ALFA_START + 1;
                }

                // Agregar byte a datos de salida
                dataOut[i] = byte;
        }

        // Regresar datos decifrados
        return dataOut;
}
```

Como se puede notar el cifrado y decifrado son procedimientos casi iguales, lo único que cambia es el sentido de "rotación".

7.3.2. Cifrado XOR

Para contrarestar las deficiencias del cifrado Cesar, se implementa también el cifrado XOR, el cual es ampliamente utilizado como componente para cifrados más complejos. Debido a las propiedades de la operación XOR, utilizando una sola función podemos cifrar y decifrar un bloque de información.
El procedimiento utilizado por el programa para implementar el cifrado XOR es:

```
Byte[] xorCrypto(Byte[] data, Byte[] key) {
        Byte[] outData;

        int k = key.length();
        for (int i = 0; i < data.length(); ++i) {
                Byte byte = data[i];
                byte ^= key[i %k];
                outData[i] = byte;
        }

        return outData;
}
```

7.4. Interfaz Gráfica e Integración de Módulos

El módulo que se encarga de integrar los demás módulos y presentarlos de manera amigable al usuario es el módulo de la interfaz gráfica.

En Qt, existen dos maneras principales de construir interfaces gráficas:

- Utilizando QtWidgets para construir interfaces tradicionales y estáticas, utilizando C++ de manera exclusiva.
- Utilizando QML, el cuál es un lenguaje para la descripción de interfaces gráficas similar al Javascript, y relacionando el la interfáz gráfica con los elementos de C++ mediante una clase intermediaria.
- Utilizando Qt WebEngine, el cuál sigue siendo un proyecto en sus primeras etapas de desarrollo.

A continuación, se muestra una comparación entre los sistemas mencionados, cortesía de la documentación de Qt:

	Qt Quick / Qt Quick Controls	Qt Widgets	Qt WebEngine	Comments
Used language(s)	QML/JS	C++	HTML/CSS/JS	
Native look'n'feel	✓	✓		Qt Widgets and Qt Quick Controls 1 integrate well to the underlying platform, providing a native look'n'feel on Windows, Linux, and macOS.
Custom look'n'feel	✓	✓	✓	Qt Widgets provide means for customization via style sheets, but Qt Quick is a better performing choice for user interfaces that do not aim to look native.
Fluid animated UIs	✓		✓	Qt Widgets do not scale well for animations. Qt Quick offers a convenient and natural way to implement animations in a declarative manner.
Touch screen	✓		✓	Qt Widgets often require a mouse cursor for good interaction, whereas Qt Quick only provides primitive building blocks that were designed with touch interaction in mind. The WebView Qt Quick component has support for multi-touch gestures to interact with web content.
Standard industry widgets		✓		Qt Widgets provide all the bells and whistles, developed over two decades, needed for building standard industry type applications. Qt WebEngine Widgets provide widgets and additional classes to render and interact with web content.
Model/View programming	✓	✓		Qt Quick provides convenient views, but Qt Widgets provide more convenient and complete framework. In addition to Qt Quick views, Qt Quick Controls provide a TableView.
Rapid UI development	✓	✓	✓	Qt Quick is an excellent choice for rapid UI prototyping and development.
HW accelerated graphics	✓	✓	✓	Qt Widgets provide QGLWidget for rendering OpenGL graphics, and Qt WebEngine supports WebGL, but the OpenGL ES 2.0 or OpenGL 2.0 based Qt Quick Scene Graph has proven to provide the best performance for UIs and for integrating with OpenGL content.
Graphical effects	✓			The particle system and shader effects available in Qt Quick are more flexible. Qt Widgets offer very little in this area.
Rich text processing	✓	✓		Qt Widgets currently provide the most comprehensive base for implementing text editors. Qt's rich text document classes can also be utilized in Qt Quick and Qt Quick Controls' TextArea, but may require some C++ implementation.
Existing web content			✓	Both Qt Quick and Qt Widgets provide components for presenting simple rich text, but Qt WebEngine is the best choice for presenting full-blown web content.

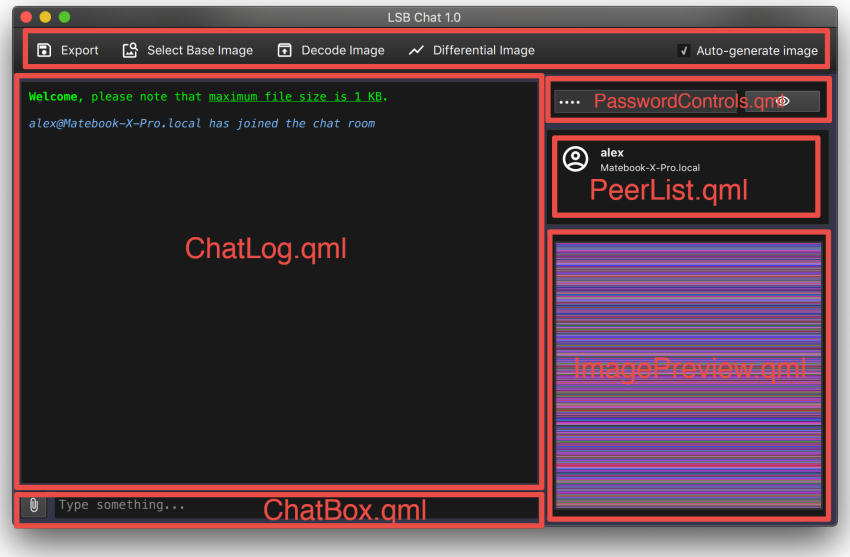
Debido a la arquitectura del proyecto, QML es la opción más adecuada. Además, construir una interfaz gráfica fácil de usar en QML es mucho más rápido que en C++.

7.4.1. Interfaz QML

La interfáz en QML se divide en los siguientes archivos:

- *main.qml:* Se encarga de cargar de manera asíncrona el archivo *MainWindow.qml* y define variables globales.
- *MainWindow.qml:* Consiste de una barra de herramientas y el archivo *ChatRoom.qml* como componente central.
- *ChatRoom.qml:* Define el acomodo de los siguientes componentes:
  - *ChatLog.qml:* Control de texto HTML que muestra los mensajes intercambiados entre los usuarios y las notificaciones del programa.
  - *PeerList.qml:* Implementa una lista de usuarios conectados y sus respectivas direcciones IP.
  - *ImagePreview.qml:* Muestra un renderizado rápido de la imagen generada/modificada por el módulo LSB. Además se da la opción de visualizar la imagen diferencial.
  - *ChatTextBox.qml:* Consta de un botón para adjuntar archivos al chat y un campo de texto que permite al usuario ingresar un mensaje.
  - *PasswordControls.qml:* Consta de un campo de texto configurado para la introducción de contraseñas, y un botón que permite al usuario ver la contraseña actual temporalmente.

A continuación, se muestra una captura de pantalla de la interfaz gráfica y sus componentes:



7.4.2. Puente entre interfaz QML y C++

Para integrar la interfaz QML y el resto de los módulos, se utiliza una clase intermediaria con el nombre de *QmlBridge*. Además de efectuar la integración, esta clase se ocupa de administrar todas las funciones que tienen que ver con la carga de exportación de información entre la aplicación y el almacenamiento permanente de la computadora.
Finalmente, cabe resaltar que esta clase es la más extensa de todas, pero no hace uso de algún algoritmo en especifico, simplemente se "adaptan" las entradas y salidas de cada módulo para efectuar la integración - y al mismo tiempo mantener la independencia entre cada módulo del programa, con el fin de facilitar el mantenimiento y desarrollo de cada módulo por separado.

7.4.3. Generación e interpretación de mensajes

Por la misma razón mencionada anteriormente, el módulo de comunicaciones solo se encarga de asegurar la correcta transmisión de información entre dos instancias de la aplicación.

Sin embargo, para poder transmitir mensajes y archivos cifrados dentro de una imagen, se decidió adaptar un formato estándar para representar la información mandada. Por ejemplo, al mandar un archivo, nos es útil saber su nombre original, el tamaño del archivo y su extensión.

Para lograr esto, se eligió utilizar el formato JSON para representar un mensaje y la información que contiene. El formato JSON es muy similar al XML, pero tiene la ventaja de que es más simple de leer y se puede codificar de manera binaria.

A continuación, se muestra una imagen generada por Sergio E. Ledesma (Universidad de Guanajuato), la cual considero que es una buena introducción a como sirve JSON:

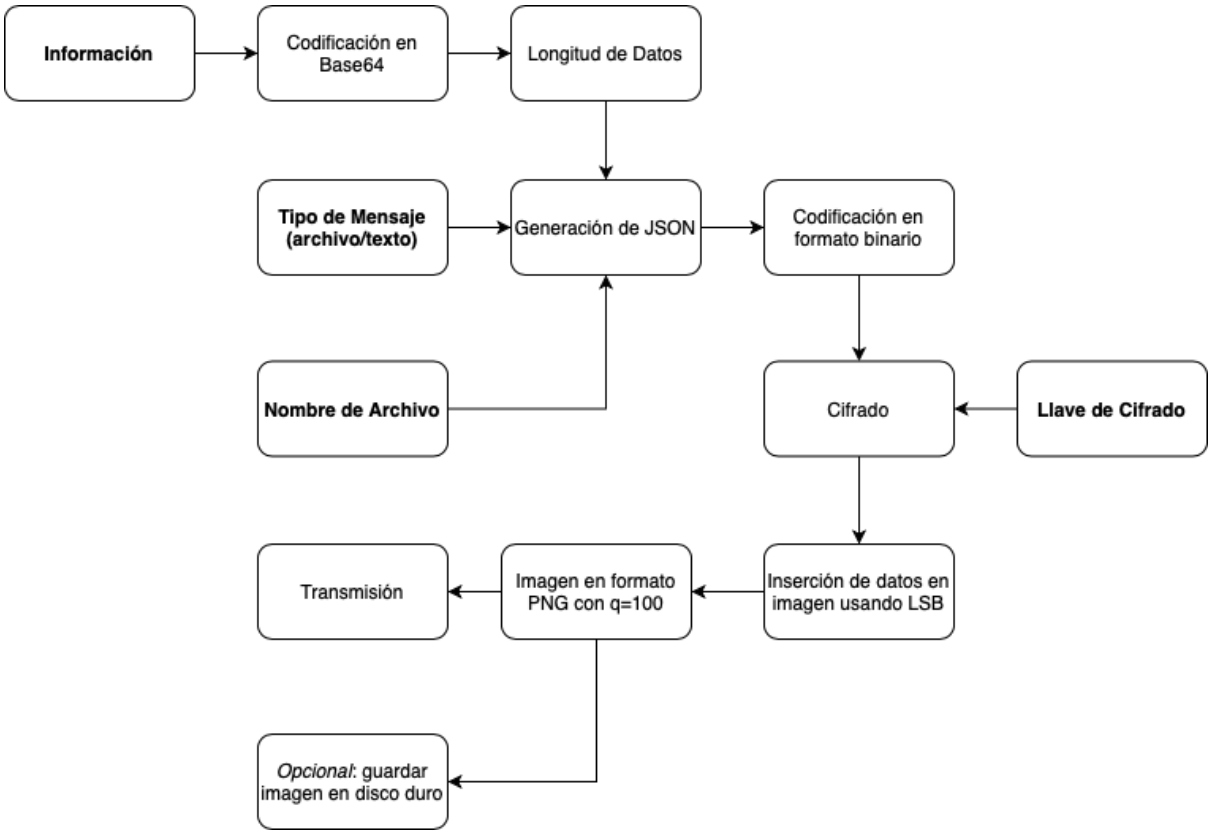
Como podemos observar, JSON básicamente representa las propiedades de un objeto como una serie de "llaves" y "valores".

Siguiendo esta linea de pensamiento, podemos representar un mensaje como un objeto con las siguientes propiedades:

- *Tipo de Mensaje* por el tiempo, los siguientes valores son aceptados por el programa:
  - *Archivo:* indica que el mensaje contiene un archivo adjuntado por el usuario
  - *Texto:* indica que el mensaje simplemente contiene un texto escrito por el usuario.
- *Nombre de Archivo:* en caso de que el mensaje contenga un archivo, este campo indica el nombre y la extensión del archivo. En caso contrario, este campo se mantiene vacío.
- *Tamaño de Datos:* este campo contiene la longitud de la información mandada dentro del mensaje, se utiliza para una validación final antes de mostrar el mensaje al usuario o guardar el archivo en el almacenamiento del sistema.
- *Datos:* contiene la información "cruda" mandada por el usuario en formato *Base64*. Se eligió convertir los datos en *Base64* para evitar perdida de información al mandar archivos con caracteres UTF-8 o secuencias de control.

Una vez generado el texto en JSON, este se encripta utilizando los algoritmos de cifrado descritos y se insertan los datos cifrados dentro de una imagen utilizando LSB.

A continuación, se muestra un diagrama que representa como se prepara y se manda un mensaje, haciendo uso de todos los módulos del programa:



Finalmente, se muestra el diagrama de como interpretamos un mensaje a partir de una imagen:

8. Desarollo de Pruebas

Para el proyecto, se desarrollaron casos de prueba utilizando el framework de *Unit Tests* de Qt. Debido a mi falta de experiencia utilizando ese modulo, solo se han desarrollados pruebas en los módulos de LSB y de Encriptación debido a que esos módulos no hacen uso del sistema de señales.

Por lo tanto, el flujo lógico de esos módulos es lineal y el comportamiento no depende de las interacciones del usuario con el programa o de factores externos a las entradas.

El procedimiento de pruebas del módulo de comunicaciones fue:

1. Lanzar dos instancias del programa en la misma red (y/o en la misma computadora).
2. Esperar a que se muestre el mensaje de "El usuario X se a unido a la sala de chat".
3. Mandar cualquier mensaje de una de las instancias de la aplicación y confirmar recepción del mensaje en la otra instancia.

Los demás módulos (excepto la interfaz gráfica) tienen pruebas automatizadas para comprobar su funcionamiento correcto.

8.1. Implementación de casos de prueba

8.1.1. Módulo LSB

Para el módulo de LSB, se implementan las siguientes pruebas:

- Validación de imagen autogenerada por el módulo el LSB.
- Codificar una cadena de datos usando LSB sobre la imagen autogenerada.
- Decodificar la imagen autogenerada y comparar los datos obtenidos con los datos originales.

A continuación, se muestra el código que implementa las pruebas:

```
void testLSB() {
    // Enable auto-image generation
    LSB::enableGeneratedImages(true);

    // Set data to encode
    const QByteArray data = "The quick brown fox jumped over the lazy dog";

    // Encode data
    const QImage lsbImage = LSB::encodeData(data);

    // Validate image size
    QVERIFY(lsbImage.width() > 0);
    QVERIFY(lsbImage.height() > 0);

    // Decode data from image
    const QByteArray decoded = LSB::decodeData(lsbImage);

    // Validate that input and output data is the same
    QVERIFY(data == decoded);
}
```

8.1.2. Módulo de Encriptación

Para el módulo de encriptación se implementan las siguientes pruebas:

- Verificar que se avienta una variable de error cuando el usuario intenta cifrar información utilizando una llave vacía.
- Verificar que al cifrar un dato con una cierta llave, y posteriormente realizar el descifrado obtengamos los datos originales.
- Verificar que al cifrar el mismo dato utilizando llaves diferentes, obtengamos datos cifrados diferentes entre si.
- Verificar que al cifrar un dato con una llave A, y al descifrar los datos generados con una llave B, no se pueda obtener acceso a los datos originales.
    - Caso 1: Llaves A y B tienen longitudes diferentes
    - Caso 2: Llaves A y B tienen la misma longitud y los mismos caracteres, pero en orden diferente.

A continuación, se muestra el código que implementa las pruebas:

```
void testCrypto() {
    // Define original data
    const QByteArray data = "The quick brown fox jumped over the lazy dog";

    // Define test keys
    const QByteArray key1 = "123";
    const QByteArray key2 = "132";
    const QByteArray key3 = "1234567890abcdefghijklmnopqrstuvwxyz!#$%&/()=?";

    // Init. error variable
    CryptoError error, error1, error2, error3;

    // Checar que el error se cambie cuando metemos una contraseña incorrecta
    Crypto::encryptData(data, "", &error);
    QVERIFY(error == kPasswordEmpty);

    // Hacer encriptaciones para tres llaves diferentes
    const QByteArray cipher1 = Crypto::encryptData(data, key1, &error1);
    const QByteArray cipher2 = Crypto::encryptData(data, key2, &error2);
    const QByteArray cipher3 = Crypto::encryptData(data, key3, &error3);

    // Checar que no haya errores de cifrado
    QVERIFY(error1 == kNoError);
    QVERIFY(error2 == kNoError);
    QVERIFY(error3 == kNoError);

    // Checar que los byte arrays cifrados sean diferentes entre si y que sean dif. a datos
    // originales
    QVERIFY(cipher1 != data);
    QVERIFY(cipher2 != data);
    QVERIFY(cipher3 != data);
    QVERIFY(cipher1 != cipher2);
    QVERIFY(cipher1 != cipher3);
    QVERIFY(cipher2 != cipher3);

    // Descifrar los archivos con las llaves adecuadas
    const QByteArray decipher1 = Crypto::decryptData(cipher1, key1, &error1);
    const QByteArray decipher2 = Crypto::decryptData(cipher2, key2, &error2);
    const QByteArray decipher3 = Crypto::decryptData(cipher3, key3, &error3);

    // Checar que no haya errores de descifrado
    QVERIFY(error1 == kNoError);
    QVERIFY(error2 == kNoError);
    QVERIFY(error3 == kNoError);

    // Checar que todos los archivos descifrados sean iguales a los datos originales
    QVERIFY(decipher1 == data);
    QVERIFY(decipher2 == data);
    QVERIFY(decipher3 == data);

    // Verificar que al usar la llave inadecuada el decif. falla
    const QByteArray badDecipher12 = Crypto::decryptData(cipher1, key2, &error);
    const QByteArray badDecipher13 = Crypto::decryptData(cipher1, key3, &error);
    const QByteArray badDecipher21 = Crypto::decryptData(cipher2, key1, &error);
    const QByteArray badDecipher23 = Crypto::decryptData(cipher2, key3, &error);
    const QByteArray badDecipher31 = Crypto::decryptData(cipher3, key1, &error);
    const QByteArray badDecipher32 = Crypto::decryptData(cipher3, key2, &error);

    // Checar que no haya errores de descifrado
    QVERIFY(badDecipher12 != data);
    QVERIFY(badDecipher13 != data);
    QVERIFY(badDecipher21 != data);
    QVERIFY(badDecipher23 != data);
    QVERIFY(badDecipher31 != data);
    QVERIFY(badDecipher32 != data);
}
```

8.2. Resultados de pruebas

Los resultados de las pruebas fueron:

```
********* Start testing of Tests *********
Config: Using QtTest library 5.12.5,
            Qt 5.12.5 (x86_64-little_endian-lp64 shared
            (dynamic) release build;
            by Clang 10.0.0 (clang-1000.11.45.5) (Apple))
PASS   : Tests::initTestCase()
PASS   : Tests::testLSB()
PASS   : Tests::testCrypto()
PASS   : Tests::cleanupTestCase()
Totals: 4 passed, 0 failed, 0 skipped, 0 blacklisted, 33ms
********* Finished testing of Tests *********
```

9. Referencias

NA. (2014, October 4). How is XOR used for encryption? Retrieved March 31, 2020, from https://crypto.stackexchange.com/questions/19470/how-is-xor-used-for-encryption

Ledesma, S. (2019, September 5). JavaScript Object Notation (JSON). Retrieved March 31, 2020, from http://sintesis.ugto.mx/WintemplaWeb/02Wintempla/26Web/08JSON/index.htm

Base64.Guru. (n.d.). Retrieved March 31, 2020, from https://base64.guru/

Qt Test Overview: Qt Test 5.14.2. (n.d.). Retrieved March 31, 2020, from https://doc.qt.io/qt-5/qtest-overview.html