Escuela Superior
de Ingeniería y Tecnología
Universidad de La Laguna

Trabajo de Fin de Grado

’Simulación y Criptografía Cuántica en una Aplicación Web’

’Simulation and Quantum Cryptography in a Web Application’

Andrea Hernández Martín

La Laguna, 26 de mayo de 2023

Dña. **Pino Caballero Gil**, con N.I.F. 45.534.310-Z Catedrática de Universidad adscrita al Departamento de Ingeniería Informática y de Sistemas de la Universidad de La Laguna, como tutora.

D. **Daniel Escánez Expósito**, con N.I.F. 79.159.491-T miembro del Grupo de Investigación en Criptología de la Universidad de La Laguna (CryptULL), como cotutor

C E R T I F I C A (N)

Que la presente memoria titulada:

”*Simulación y Criptografía Cuántica en una Aplicación Web*”

ha sido realizada bajo su dirección por Dña. **Andrea Hernández Martín**, con N.I.F. 45.853.565-Y.

Y para que así conste, en cumplimiento de la legislación vigente y a los efectos oportunos firman la presente en La Laguna a 26 de mayo de 2023

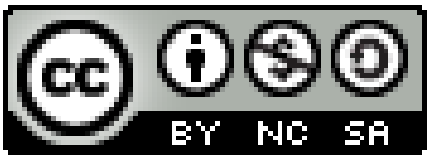
Agradecimientos

A mis tutores, Pino y Daniel, por la dedicación y apoyo que me han dado en todo momento para resolver los problemas teóricos y prácticos que me han surgido.

A mi madre, a José Manuel, a mi hermana, a Cristian y a Diego, por ser quienes me han apoyado día a día y han logrado que llegue a donde estoy y a ser quien soy, y sobre todo por la paciencia que han tenido durante estos años.

Y también a mis abuelos, familiares y amigos, sobre todo a Álvaro, Meli y Kabir, que me acompañaron todos los años durante el desarrollo del Grado de Ingeniería Informática pues sin ellos nunca lo hubiera terminado.

Licencia



Resumen

QuantumSolver es una herramienta propuesta y desarrollada recientemente desde la Universidad de La Laguna con el objetivo de dar respuesta al creciente interés por la computación cuántica. Debido a que QuantumSolver intenta satisfacer a una amplia variedad de posible público de los programas principales ejecutables disponibles, se ofrece la ejecución del software por medio de dos interfaces diferentes: Interfaz por Línea de Comandos e Interfaz Web. Esta última está más orientada al público general, pues permite que cualquier usuario pueda ejecutar los algoritmos cuánticos en hardware cuántico real, sin tener ningún tipo de experiencia previa con la programación informática. La idea principal de este trabajo ha sido conseguir que se pueda ejecutar QuantumSolver mediante ambas interfaces para facilitar su uso tanto para usuarios que con experiencia en el tema como también para usuarios sin experiencia. Además, en la herramienta se ha implementado la oferta de uso de simuladores cuánticos de imitación. Finalmente, una segunda finalidad ha sido añadir un nuevo protocolo criptográfico para enriquecer la herramienta.

Palabras clave: Docker, React, Node, Back-End, Front-end, Computación cuántica, Protocolo Six-State, Criptografía cuántica, Qiskit.

Abstract

QuantumSolver is a tool recently proposed and developed from the University of La Laguna with the aim of responding to the growing interest in quantum computing. Due to the fact that QuantumSolver aims to satisfy a wide range of possible audiences of the main executable programs available, software execution can be carried out through two different interfaces: Command Line Interface and Web Interface. The latter is more oriented towards the general public because it allows any user to execute quantum algorithms on real quantum hardware without any prior experience in computer programming. The main idea of this work has been to make QuantumSolver capable of running through both interfaces, to facilitate its use for both experienced and inexperienced users. In addition, the offer to use imitation quantum simulators has been implemented in the tool. Finally, a second goal has been to add new cryptographic protocols to enrich the tool.

Keywords: Docker, React, Node, Back-End, Front-End, Quantum Computing, Six-State Protocol, Quantum Cryptography, Qiskit.

Índice general

Índice de Figuras

Índice de Tablas

Capítulo 1

Introducción

1.1. Estado del arte

La computación y criptografía cuánticas son temas que recientemente han ido ganando importancia debido a su rápido avance apoyado por grandes empresas tecnológicas como IBM [1], Google [2] o Amazon [3], que están invirtiendo en el desarrollo e investigación de esta tecnología.

En 2022, IBM presentó su último procesador cuántico, Osprey [4], que cuenta con 433 cúbits, lo que representa un gran adelanto en la capacidad de procesamiento cuántico. Este computador es parte de la próxima generación del sistema *IBM Quantum System Two*, que está diseñado para ofrecer mayor rendimiento y capacidades avanzadas de computación cuántica. Desde 2019 habían publicado una hoja de ruta de desarrollo [5] en la que se incluía el lanzamiento de nuevos servicios y herramientas de software de computación cuántica. Hasta el momento se han cumplido los objetivos establecidos en dicha hoja, enfocados hacia nuevas formas de integrar la computación cuántica con la informática clásica para lograr el máximo aprovechamiento de la tecnología. Esa planificación predice la construcción de sistemas cuánticos de próxima generación con más de 1000 cúbits para abordar problemas cada vez más complejos.

Recientemente, la revista científica *Nature* ha publicado el segundo gran hito de Google en computación cuántica [6], lo que representa un importante paso hacia la creación de esa máquina cuántica capaz de realizar cálculos complejos de manera más rápida que cualquier supercomputadora actual. Concretamente el logro de Google se basa en el concepto de ”supremacía cuántica”, anunciado por Google desde 2019 [7]. Ese concepto se refiere al punto en el que una máquina cuántica puede realizar una tarea que sería prácticamente imposible de realizar por una computadora clásica. Un ejemplo es la tarea para generar un patrón aleatorio de números y verificar si este patrón es realmente aleatorio.

Todos esos avances evidencian el creciente interés en las tecnologías cuánticas. Por ese motivo, con el objetivo de fomentar su uso y estudio por usuarios con o sin experiencia en informática nació la propuesta *QuantumSolver* [8]. Con esa meta en mente, se potenciaron dos características de la herramienta, referentes a los modos de acceso al software desarrollado. Por un lado, ofrece una interfaz web en la que se pueden ejecutar los algoritmos disponibles predefinidos obteniendo los resultados de manera visual e intuitiva. Por otro lado, tiene una interfaz por línea de comandos, que está más orientada a usuarios con algún conocimiento informático.

1.2. Objetivos

En el presente proyecto se ha estudiado y analizado el trabajo previo realizado de la librería *QuantumSolver*, con el objetivo de desarrollar nuevas propuestas de mejora para la misma.

Esta herramienta es una librería que permite la simulación de algoritmos y protocolos cuánticos, y desde su lanzamiento hasta el día de hoy ha ido creciendo notablemente. La herramienta se podía usar a través de una interfaz web pero su instalación es bastante compleja para usuarios sin conocimientos informáticos. Por ello el principal objetivo de este trabajo ha sido realizar el lanzamiento público de la aplicación web, para que todos los usuarios puedan hacer uso de ella. Del mismo modo, es también una meta la publicación del módulo de *Python* en *PyPi*, consiguiendo una mejora considerable en la accesibilidad del proceso de descarga e instalación de la librería.

En esta misma línea de trabajo, un objetivo ha sido optimizar el diseño del entorno de la aplicación web, mejorando la UX mediante cambios en la UI, la navegación, así como el diseño visual, etc.

Por otro lado, se han añadido nuevos simuladores cuánticos a la librería para que cuente con una amplia variedad en la que probar los protocolos y algoritmos, obteniendo así una gran diversidad de resultados. Además, también se ha añadido un nuevo protocolo de distribución de claves cuánticas denominado *Six-State*, lo que ha supuesto un aporte científico más amplio al trabajo.

1.3. Fases

1.3.1. Estudio bibliográfico

Durante los primeros meses del transcurso del proyecto (noviembre y diciembre), se realizó la revisión y el análisis de la librería *QuantumSolver*, para conocer su funcionamiento y poder trabajar con ella. Además también se revisó la bibliografía de contenidos introductorios en el marco teórico de la computación cuántica. Posteriormente, tras tomar decisiones acerca de las tecnologías a utilizar, también se revisaron contenidos sobre dichas tecnologías como *docker*, *docker compose* y *heroku*. Además se investigó sobre el uso de PyPi y su funcionamiento para añadir el módulo de la librería.

1.3.2. Planificación

En esta etapa, que transcurrió durante los meses de enero y febrero, se fijaron los requisitos principales del proyecto. Se realizaron los *mockups* para la mejora visual de la aplicación web y se comenzó con el lanzamiento público de la web. Además se comenzó a investigar sobre el protocolo cuántico *Six-State* escogido para añadir a la librería y sobre los simuladores cuánticos de imitación proporcionados por IBM.

1.3.3. Desarrollo

La implementación se llevó a cabo una vez finalizada la investigación sobre todas las tecnologías necesarias para su desarrollo. Para ello se realizó un *fork* en Github del proyecto original, para realizar la continuación del proyecto sin modificar el original, y se crearon tres ramas separadas, una para la parte del lanzamiento público de la web, otra

para el desarrollo del protocolo cuántico y otra para añadir los simuladores de imitación. Una vez terminada cada parte y comprobado que funcionaban correctamente, se hizo un *Pull Request* sobre el proyecto original.

1.4. Estructura de la memoria

Este documento comienza con la descripción de algunos de los avances más importantes en el campo de la computación cuántica. A continuación se describen los objetivos y fases del desarrollo del trabajo. En el segundo capítulo se incluye la explicación del procedimiento para realizar un lanzamiento público de la web. También se indica la publicación del módulo para la facilitación de la accesibilidad de este. La descripción de los simuladores cuánticos de imitación que se seleccionaron para *QuantumSolver* se hallan en el tercer capítulo, nombrando todos los que proporciona IBM e incluyendo los resultados de un algoritmo ejecutado en alguno de ellos. El siguiente capítulo recoge la investigación e implementación realizada del protocolo criptográfico cuántico *Six-State*, detallando sus entidades, fundamento, ejecución y resultados. Los capítulos quinto y sexto realizan un recorrido por las conclusiones y líneas futuras, tanto en español como en inglés. El séptimo capítulo recoge una estimación del presupuesto desglosado del total desarrollo de la implementación e investigación de este trabajo de fin de grado. Finalmente, en un apéndice se incluye un artículo producto de este trabajo, que ha sido aceptado en un congreso nacional, y que será presentado en las próximas semanas.

Capítulo 2

Lanzamiento público de la web

2.1. Preparación

La librería *QuantumSolver* contaba con una aplicación web. Sin embargo, el problema principal es que su instalación es bastante compleja sobre todo para usuarios sin conocimientos informáticos. Por ello, un objetivo de este trabajo ha sido realizar un lanzamiento público de la web para que esté accesible sin la necesidad de instalación.

En primer lugar se procedió a crear una rama aparte en *github* del proyecto principal para la realización de esta tarea. Luego, se tuvo que separar el directorio que contiene la parte del *backend* de la parte que contiene el *frontend*, ya que es más sencillo trabajar con ambas partes separadas. En la Fig. 2.1 se puede observar cómo quedó la estructura de directorios una vez separadas, teniendo en cuenta que el directorio *quantum_solver_web* corresponde a la parte del *frontend* y *flask-server* corresponde al *backend*.

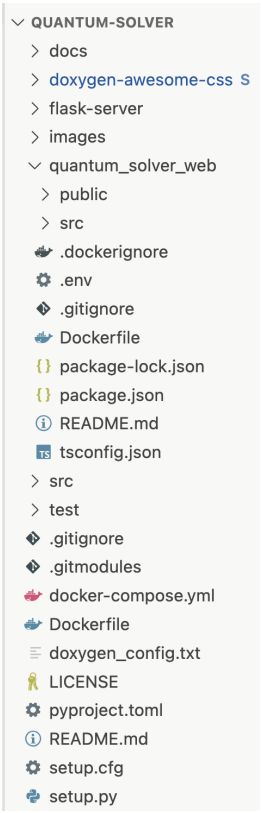


Figura 2.1: Estructura de directorios.

En un principio se trabaja con la herramienta *docker* para el funcionamiento de ambas partes por separado, ya que *docker* crea contenedores y en el caso de esta aplicación, cada contenedor se encarga de una parte diferente, para luego finalmente conectarlos entre sí. A continuación se explican detalladamente los pasos que se han realizado para el funcionamiento.

2.2. Docker

2.2.1. Frontend

La parte de *frontend* de la aplicación está realizada mediante *React*, la cual es una biblioteca de código abierto diseñada para crear interfaces de usuario con el objetivo de facilitar el desarrollo de aplicaciones, y el lenguaje de programación *Typescript*.

Para utilizar la herramienta *docker*, en primer lugar hay que crear un fichero denominado *'Dockerfile'* en el directorio raíz en la que se encuentra esta parte. Este fichero contiene los pasos de instalación y ejecución de la herramienta. En la Fig. 2.2 se puede observar el contenido.

quantum_solver_web > Dockerfile > ...

```
1  FROM node:19
2
3  WORKDIR /usr/src/app
4
5  # Install app dependencies
6  # A wildcard is used to ensure both package.json
7  # AND package-lock.json are copied
8  # where available (npm@5+)
9  COPY package*.json ./
10 COPY tsconfig.json ./
11
12 # If you are building your code for production
13 # RUN npm ci --only=production
14
15 # Bundle app source
16 COPY src/ ./src/
17 COPY public/ ./public/
18 COPY .env ./env
19
20 RUN npm install
21 EXPOSE 3000
22
23 ARG HOST=0.0.0.0
24
25 CMD [ "npm", "start" ]
```

Figura 2.2: Dockerfile frontend.

En la Fig. 2.2 se observa que en primer lugar se crea un contenedor con la imagen de *Node* con versión 19, que es compatible con la librería *QuantumSolver*. Luego se establece el directorio de trabajo dentro del contenedor. A continuación, se copia el *package.json*, *package-lock.json* y el compilador de *typescript* ya que sin ellos, la aplicación no compila y también, se copian los directorios que contienen el código de la aplicación. En ese punto, se procede a instalar todas las dependencias, establecer el puerto de escucha de la aplicación a 3000 con el *host* a 0.0.0.0, para que se escuche en todas las interfaces de red disponibles en el contenedor y finalmente, ejecutar el comando '**npm start**' que lanza la aplicación.

En segundo lugar, una vez realizado el fichero *Dockerfile*, hay que crear la imagen de *docker* para luego ejecutar el contenedor a través de los comandos que ofrece esta aplicación. Hay que tener en cuenta que se tiene que realizar con permisos *root*. El primer

comando a ejecutar es el que crea la imagen: '**docker build -t frontend .**', usando '-t' para etiquetar la imagen con un nombre para que sea más fácil encontrarla, y luego se ejecuta el contenedor con: '**docker run -p 3000:3000 -d frontend**', en el que usando '-p' se le redirige un puerto público a un puerto privado del contenedor y '-d' para que el contenedor se ejecute en segundo plano. Una vez se ha realizado esto, primero hay que comprobar que efectivamente el contenedor ha sido creado y está en funcionamiento sin ningún error. Para esto hay que ejecutar el comando: '**docker ps**', el cual se puede ver en la Fig. 2.3, mostrando el estado actual del contenedor, así como su identificador y el puerto de escucha, entre otras cosas.

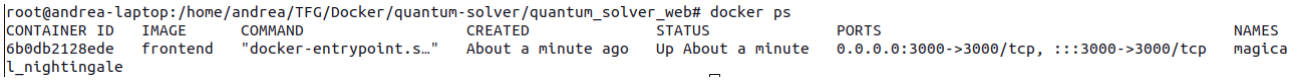


Figura 2.3: Ejecución del comando *docker ps* cuando se crea el frontend.

Una vez comprobado que efectivamente todo ha funcionado correctamente, si se accede a la ruta *'localhost:3000'* en un navegador, se debería observar la página principal de *QuantumSolver*, aunque no se puede interactuar con ella, ya que faltaría el *backend* funcionando. En caso de que no se mostrase dicha página, hay que comprobar los *logs* que ha generado el contenedor. Para ello se tiene que ejecutar el comando: '**docker logs <id>**', donde <id>tiene que ser el identificador del contenedor que se obtuvo anteriormente. De esta forma se podrá saber el error que ha ocurrido.

2.2.2. Backend

La parte de *backend* de la aplicación está realizada mediante *Flask*, el cual es un *framework* minimalista escrito en el lenguaje de programación *Python* que permite crear aplicaciones web rápidamente y con un mínimo número de líneas de código.

Una vez se tiene en funcionamiento el *frontend*, se comienza con la parte del *backend*, la cual es muy parecida a la anterior, teniendo en cuenta que el fichero *Dockerfile* es distinto.

Se crea el fichero *Dockerfile* en el directorio raíz donde está esta parte de la aplicación, con el contenido que se puede ver en la Fig. 2.4. En esa figura en primer lugar, se crea una imagen de *Python* con la versión 3.8.16, que es la compatible con la librería, luego se establece el directorio de trabajo en el contenedor y dos variables de entorno. A continuación, se copian los directorios necesarios para el funcionamiento del *backend*, como por ejemplo el que contiene el código de los algoritmos y protocolos, o el que tiene la configuración del proyecto. Una vez se tienen todos los directorios y ficheros necesarios, se instalan todas las dependencias que necesita el programa, se establece el puerto de escucha en 5000, ya que es el que utiliza *flask* en la aplicación y finalmente se compila la aplicación con el comando '**python3 flask-server/server.py**'

Una vez preparado el fichero *Dockerfile*, se procede a seguir los mismos pasos que en el caso anterior, cambiando algunos datos de los comandos. Para crear la imagen en este caso, se crea de la misma forma pero cambiando el nombre, quedando de la siguiente manera: '**docker build -t backend .**', luego se ejecuta el contenedor con: '**docker run -p 5000:5000 -d backend**', en el que el puerto público y privado en este caso es 5000 como se estableció en el *Dockerfile* así como el nombre de *backend*. Una vez se ha realizado, primero se comprueba que efectivamente el contenedor ha sido creado y está en funcionamiento sin ningún error, para esto se vuelve a ejecutar este comando:

```
Dockerfile > ...
1  FROM python:3.8.16-bullseye
2
3  WORKDIR /app
4
5  ENV FLASK_APP=flask-server/server.py
6  ENV FLASK_DEBUG=development
7
8  # Bundle app source
9  COPY flask-server/ ./flask-server
10 COPY pyproject.toml/ ./pyproject.toml
11 COPY setup.cfg ./setup.cfg
12 COPY README.md ./README.md
13 COPY src/ ./src
14
15
16 RUN python -m pip install -e .
17 RUN pip install flask && pip install -U flask-cors && pip install APScheduler && pip install pylatexenc
18
19 COPY . .
20
21 EXPOSE 5000
22
23 CMD [ "python3", "flask-server/server.py" ]
```

Figura 2.4: Dockerfile backend.

'**docker ps**', el cual se puede ver en la Fig. 2.5 y si todo ha ido bien también se puede observar que el creado anteriormente para el *frontend* sigue estando en funcionamiento.

```
root@andrea-laptop:/home/andrea/TFG/Docker/quantum-solver# docker ps
CONTAINER ID   IMAGE      COMMAND                  CREATED          STATUS          PORTS                                       NAMES
f0544b08a8a5   backend    "python3 flask-serve…"   4 seconds ago    Up 2 seconds    0.0.0.0:5000->5000/tcp, :::5000->5000/tcp   funny_brown
6b0db2128ede   frontend   "docker-entrypoint.s…"   42 minutes ago   Up 42 minutes   0.0.0.0:3000->3000/tcp, :::3000->3000/tcp   magical_nighti
ngale
```

Figura 2.5: Ejecución del comando *docker ps* cuando se crea el backend.

2.2.3. Resultado

Ahora que se han creado los dos contenedores, si se accede a la ruta *'localhost:3000'* en un navegador web, se puede observar la página principal de *QuantumSolver* pero no se puede interactuar con ella todavía. Esto es debido a que aunque estén ambos contenedores en funcionamiento, no se están comunicando entre sí. Para ello, se va a utilizar *docker-compose*, el cual es una herramienta para definir y ejecutar aplicaciones de *docker* de varios contenedores.

En primer lugar, hay que crear un fichero *docker-compose.yml* en el directorio raíz del proyecto. El contenido del fichero se puede observar en la Fig. 2.6. Este fichero contiene en primer lugar la versión de docker-compose, luego crea unos servicios en los que por un lado tenemos el *backend (server)*, que se crea en la raíz, con el puerto público 5000 y el privado igual, y el volumen que indica el directorio de trabajo del contenedor *docker* en el que está alojado. Por otro lado, crea el servicio *frontend (web)*, el cual se crea en el directorio *quantum_solver_web*, con el puerto público 3000 y el privado igual, el volumen indicando el directorio de trabajo del contenedor *docker* y por último, a diferencia del anterior, hay que especificar que depende del *server* para que se comuniquen entre sí.

De esta forma hemos conseguido que se comuniquen ambos contenedores, obteniendo así como resultado que la página web funcione correctamente pudiendo interactuar con ella. Además también con este procedimiento se está automatizando el proceso de crear contenedores, ya que al usar *docker-compose* también se crean, evitando tener que utilizar los comandos explicados en las secciones pasadas. Para hacer que los contenedores estén en funcionamiento hay que ejecutar este comando en el directorio en el que se encuentra el fichero *docker-compose*: '**docker-compose up -d**', en el que '-d' significa que se ejecuten en segundo plano. Si se quisiera seguir teniendo el contenedor creado pero apagado, se ejecutaría '**docker-compose stop**'.

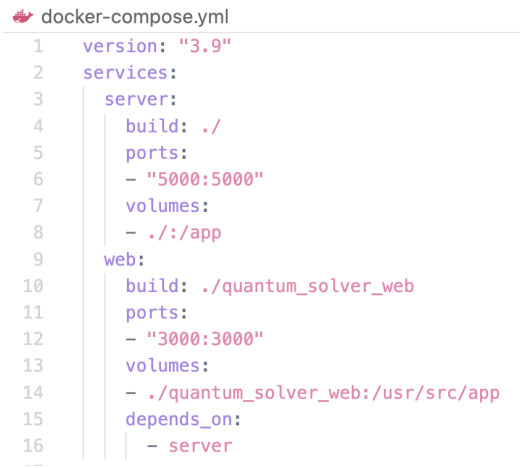


Figura 2.6: Fichero docker-compose.yml.

2.3. Hosting

Para el lanzamiento público de la web se necesita utilizar herramientas de *hosting* que proporcionan alojamiento y un dominio público para la web. En este caso solo ha sido posible poner en funcionamiento la parte del *frontend*, pero además de detallar la puesta en funcionamiento de este, también se van a detallar las opciones que se han intentado y los fallos que se han obtenido a la hora de intentar desplegar el *backend*.

2.3.1. Heroku

La primera plataforma en la que se intentó el funcionamiento del lanzamiento público, fue *Heroku*, ya que es compatible con *docker* y permite tener una cuenta gratuita siempre y cuanto no utilices ciertos recursos de la herramienta. Se probaron formas de desplegar la web en *Heroku*, una utilizando el *docker* ya implementado y otra sin él a través de las ramas creadas en *Github*.

Con *docker* se crearon dos proyecto en *Heroku*, uno para el *frontend* y otro para el *backend*. Se comenzó por el *frontend* y lo que se hizo es subir el código de esta parte con el *Dockerfile*, ya que *Heroku* tiene comandos específicos en caso de querer subir contenedores, estos comandos son: '**heroku container:push web**' para subir el código a la aplicación y crear la imagen, y el comando '**heroku container:release web**' para lanzar los cambios nuevos a la web desplegada. A continuación, al acceder al dominio que proporciona *Heroku* [13] se puede observar la página principal de *QuantumSolver* pero no se puede interactuar con ella debido a la falta del *backend*.

Una vez desplegada la parte del *frontend* se comenzó con la otra parte, para esta se hizo exactamente lo mismo, pero subiendo el fichero *Dockerfile* correspondiente. Luego, se ejecutan los comandos, la imagen se crea correctamente y no se advierte de ningún fallo, pero a la hora de acceder al dominio que proporciona *Heroku* para esta parte, se muestra que la aplicación no funciona. Y los *logs* de la aplicación indican que el puerto no se pudo vincular a la aplicación. Se puede ver en la Fig. 2.7 el error.

Investigando sobre dicho error para buscar la manera de arreglarlo, se sacaron dos conclusiones: por un lado, se puede dar debido a que el propio *Heroku* rechaza la conexión entre las dos aplicaciones debido a los puertos de escucha de *docker* por lo que no puede vincular los puertos, y no hubo forma de conseguir solucionarlo, y por otro lado, también se podría dar porque el servidor está desarrollado en *Flask Server*,

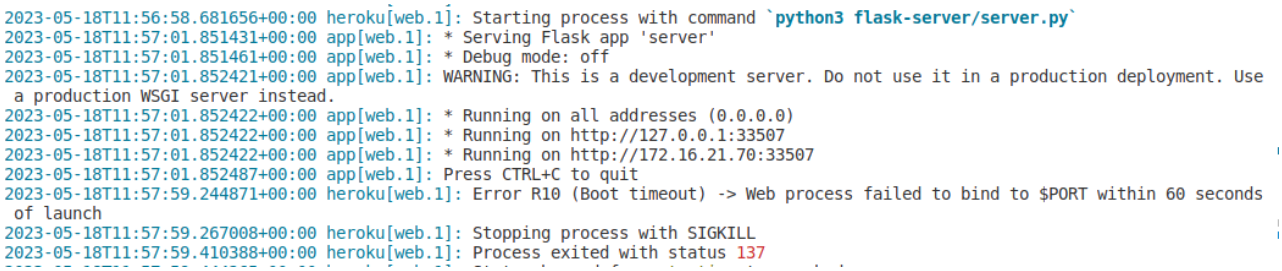


Figura 2.7: Error al desplegar el backend en Heroku mediante Docker.

y según la documentación, es un servidor de desarrollo que no se puede utilizar para la implementación en producción por lo que aconsejan utilizar un servidor *WSGI* de producción en su lugar para su funcionamiento. Por falta de tiempo, esta última opción no se pudo contemplar.

Una vez se intentaron todas las opciones viables para el despliegue mediante los contenedores que se habían creado anteriormente. Se intentó sin ellos, para ello, se eliminaron todos los ficheros *DockerFile* y *docker-compose* de las ramas y se creó un archivo *Procfile* en cada rama, fichero que utiliza *Heroku* para el despliegue de aplicaciones a través de *Github*, en el cual se tiene que especificar el comando que ejecuta la aplicación.

En el caso del *frontend* dentro del fichero se escribía lo siguiente: '**web: cd quantum_solver_web && npm start**', ya que se tiene que acceder al directorio en el que se encuentra el proyecto creado con *React* para luego ejecutar la aplicación. Una vez los cambios se suben a *Github*, se puede hacer un despliegue de esta desde la interfaz web de *Heroku*, una vez se selecciona el repositorio y la rama, se comienza con el despliegue, pero este falla por la instalación de la aplicación ya que no encuentra unas liberías de *Python* que necesita, pero que se están instalando con el comando, ya que en local funciona de esta forma. Por tanto, el despliegue no es satisfactorio para el *frontend* de esta forma. Se puede observar el error en la Fig. 2.8

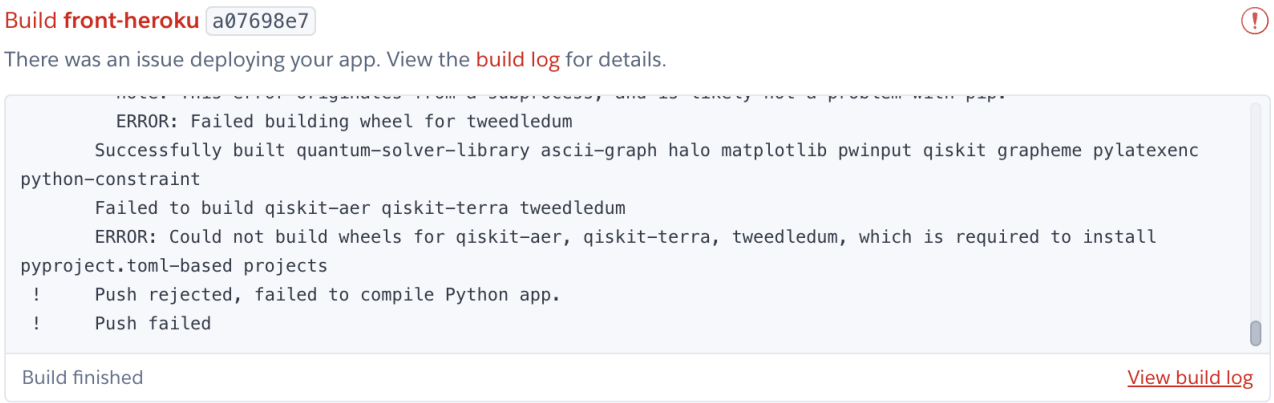


Figura 2.8: Error al desplegar el frontend en Heroku mediante Github.

Luego, en la rama del *backend* dentro del archivo creado *Procfile* se añade la siguiente línea: '**web: python -m pip install -e . && python3 flask-server/server.py**' para indicarle que esta aplicación se ejecuta con *Python* y el archivo de ejecución. Pero al hacer el despliegue pasa exactamente lo mismo que en intento anterior, ya que falla con la misma librería y a la hora de hacer el despliegue no se consigue instalar. En la Fig. 2.9 se puede ver el error en la interfaz gráfica de *Heroku*. No se encontró ninguna solución a esto, ya que no se llegó a entender bien porque la ejecución en local no falla ejecutando

exactamente los mismos comandos y luego en la ejecución dentro de la aplicación de *hosting* sí.

Figura 2.9: Error al desplegar el backend en Heroku mediante Github.

2.3.2. Railway

Una vez agotados todos los intentos en *Heroku*, se pasó a otra aplicación de *hosting* llamada *Railway*, la cual permite 500 horas de despliegue gratuito y es compatible con *docker* también.

Para este caso se utilizaron las ramas separadas para el *frontend* y para el *backend* que contienen el *Dokerfile* correspondiente. Se empezó creando un proyecto para el *frontend*, en el que se especificó el repositorio *Github* a utilizar, así como la rama, la ruta del directorio y el dominio a utilizar, ya que esta aplicación a diferencia de *Heroku* permite establecer el que se quiera siempre y cuando esté disponible. Una vez configurados todos los parámetros necesarios, se lanzó el despliegue ya que cuando comienza a realizarlo, detecta automáticamente el fichero *Dockerfile*, de esta forma crea la imagen de *docker* y publica dicha imagen correctamente para el despliegue. En los *logs* de construcción del despliegue no se muestra ningún error por lo que se despliega la web. Pero al acceder al dominio establecido anteriormente aparece un error en la pantalla de que la aplicación no responde y cuando accedemos a los *logs* de la aplicación se ve como la aplicación se ejecuta pero luego no llega a establecer ni siquiera la conexión con los puertos ni la dirección. En la Fig. 2.10 se muestra el error, en el que se explica que fue un error porque el puerto de escucha está cerrado. Finalmente, después de buscar información acerca de este error, no se pudo solucionar ya que no hay mucha documentación acerca de esta aplicación de *hosting*.

Aunque la parte del *frontend* no funcionó, igualmente se intentó desplegar el *backend*. Para ello, se creó otro proyecto igual que el anterior pero en este caso con la rama que contiene esta parte así como con otro dominio. Se procedió a lanzar el despliegue, la propia aplicación creó internamente la imagen de *docker* especificada en el *Dockerfile* instalando todas las dependencias necesarias para el funcionamiento y publicó la imagen sin ningún tipo de error. Pero como sucedió en el anterior, al acceder al dominio de la aplicación muestra el error de que la aplicación no responde. Pero en este caso, en los *logs* de la aplicación no hay ningún error, de hecho se muestra como el servidor está ejecutado, se puede observar en la Fig. 2.11, pero luego se le intentan hacer peticiones y sale el error 503 indicando que el servicio no está disponible. Tras buscar información, tampoco

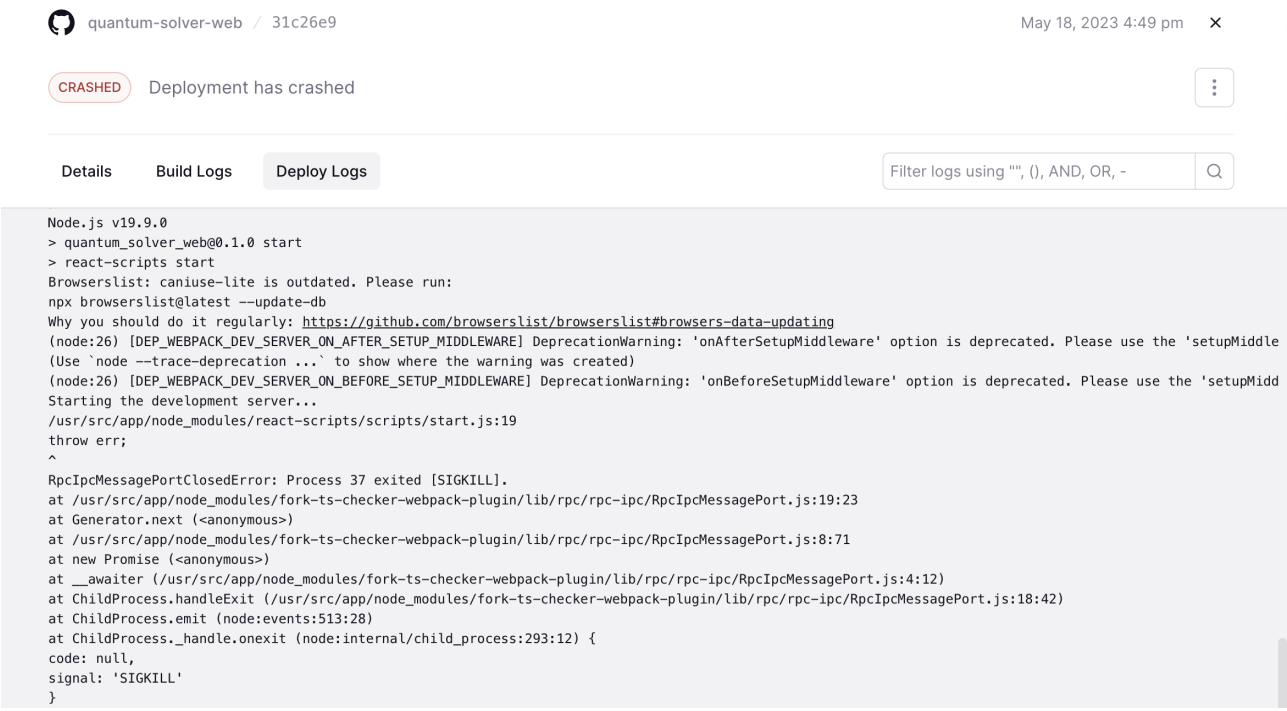


Figura 2.10: Error al desplegar el frontend en Railway.

se supo resolver el problema y finalmente no se consiguió el *backend* desplegado.

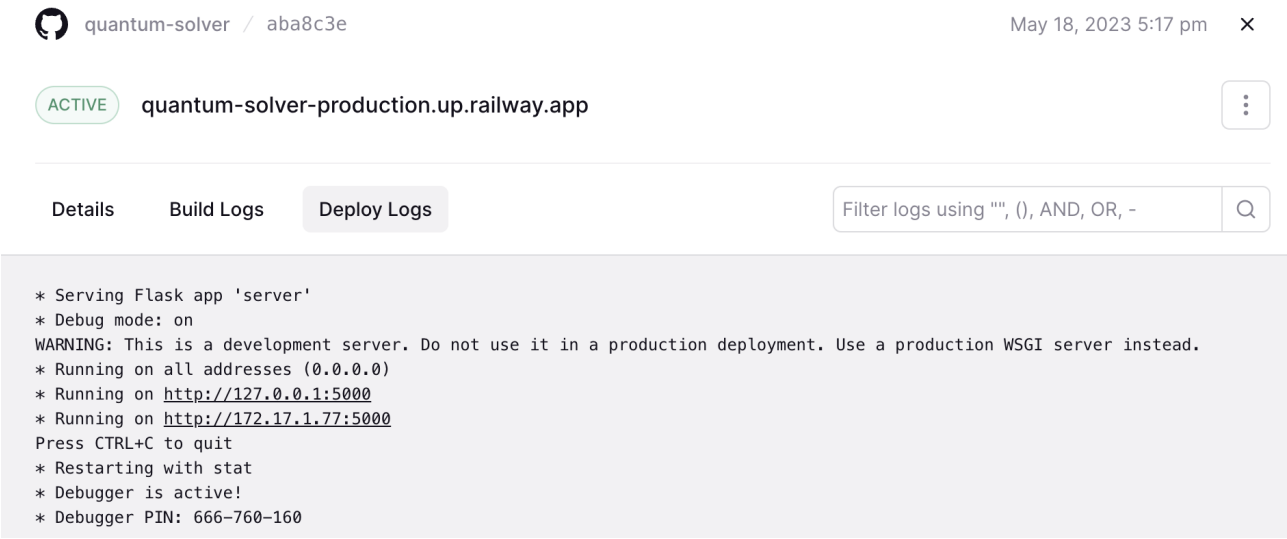


Figura 2.11: Error al desplegar el backend en Railway.

2.3.3. Conclusiones

Aunque por el momento no se ha logrado un lanzamiento público de la web completo, ya que solo se consiguió el *frontend* en el primer intento con *docker* y *Heroku*, se han adquirido nuevos conocimientos, como es el caso de los contenedores *docker*, debido a que estos no se habían utilizado previamente y es una tecnología muy interesante y muy útil. Gracias al intento del despliegue, se ha podido trabajar con ella y se ha podido ver

qué son, cómo funciona, la amplitud de cosas que se pueden realizar, etcétera. También aunque no se ha podido solucionar el error de los puertos y sus conexiones, han aportado bastantes conocimientos nuevos a la hora de manejarlos y probar diferentes soluciones, que aunque no fuesen las acertadas para este caso, a lo mejor para otros sí que puede funcionar, y gracias a esto ya se tienen los conocimientos para poder realizarlo.

2.4. PyPi

Por otro lado, se pretende publicar el módulo de *Python* en *PyPi*, consiguiendo una mejora considerable en la accesibilidad del proceso de descarga e instalación de la librería *QuantumSolver*. En primer lugar se subirá a *PypiTest*, ya que es el entorno de pruebas de *PyPi* en el cual se puede poner a prueba las herramientas y los procesos de distribución sin afectar al real.

Para ello, primero debemos crear una cuenta en *PypiTest*, luego una vez creada se debe crear en la raíz del directorio del módulo que se quiere subir un archivo *setup.py* el cual contiene todos los datos necesarios para crear dicho módulo. Estos datos son: el nombre del módulo, los autores, la descripción, la licencia, los requisitos de instalación, la versión, etc. En la Fig. 2.12 se muestran todos los datos incluidos en el módulo.

setup.py > ...

```python
import setuptools

readme = open("./README.md", "r")

setuptools.setup(
    name = "quantumSolverTool",
    version = "0.0.5",
    author = "Andrea Hernández",
    author_email = "alu0101119137@ull.edu.es",
    description = "A little quantum toolset developed using Qiskit",
    long_description =  readme.read(),
    long_description_content_type = "text/markdown",
    url = "https://github.com/alu0101119137/quantum-solver",
    project_urls = {
        "Bug Tracker": "https://github.com/alu0101119137/quantum-solver/issues",
    },
    classifiers = [
        "Programming Language :: Python :: 3",
        "License :: OSI Approved :: MIT License",
    ],
    package_dir = {"": "src"},
    packages = setuptools.find_packages(where="src"),
    python_requires = ">=3.6",
    install_requires = [
        'numpy>=1.19.5',
        'matplotlib==3.4.2',
        'qiskit==0.34.1',
        'pwinput==1.0.2',
        'halo==0.0.31',
        'alive-progress==2.3.1',
        'ascii_graph==1.5.1',
        'werkzeug>=0.11.15',
        'oauthlib>=3.0.0',
        'pytz>=2017.3',
        'setuptools',
        'flask',
        'flask-cors',
        'APScheduler',
        'pylatexenc'
    ],
    include_package_data=True
)
```

Figura 2.12: Fichero setup.py.

Una vez realizado el fichero *setup.py*, se procede a subir el módulo a *PyPiTest*, para ello, se utiliza la línea de comandos. En primer lugar se tiene que ejecutar '**python3 setup sdist/**', con el que se crea un directiorio denominado dist con el archivo comprimido que contiene el código fuente. Una vez realizado lo anterior se ejecuta el siguiente: '**twine upload —respository pypitest dist/***', el cual en primer lugar pide las credenciales del usuario para luego subir los archivos del directorio *dist* creado anteriormente, si todo se ha realizado bien, se podrá observar el módulo en la web de *PyPiTest*.

Por último, una vez subido el paquete hay que probar que cuando se importa el módulo este funciona correctamente. Hasta ahora, el paquete solo funciona si una de sus librerías de los requisitos de instalación se instalan manualmente antes de instalar el paquete.

Aunque esto no es lo esperado, por falta de tiempo no se ha podido indagar más en la instalación de dicha librería automáticamente, ya que la propia documentación de esta recomienda que sea instalada de manera manual. Se estaba buscando como solución evitar utilizar dicha librería para poder instalar el paquete de *QuantumSolver* sin necesidad de instalar manualmente nada adicional. Como propuesta de futuro se pretende indagar más y conseguir este objetivo.

2.5. Mockups

Para finalizar esta sección, se muestran los *mockups* que se realizaron para una mejora visual de *QuantumSolver* que aunque este trabajo de fin de grado no trata sobre mejorar la interfaz gráfica ni su visualización, se ha propuesto una ayuda para una futura mejora de esta mediante una maquetación realizada con la aplicación *Figma*. La maquetación realizada fue sobre la página principal, la página que genera el *token* de acceso y por último, la página que contiene el menú principal, para la selección de algoritmos, la selección de los *backends* y la selección de los parámetros necesarios, estas páginas se pueden observar en las Fig. 2.13, Fig. 2.14 y Fig. 2.15 respectivamente. Aunque no están todas las páginas que utiliza *QuantumSolver*, se da una pequeña idea de una mejor visualización.

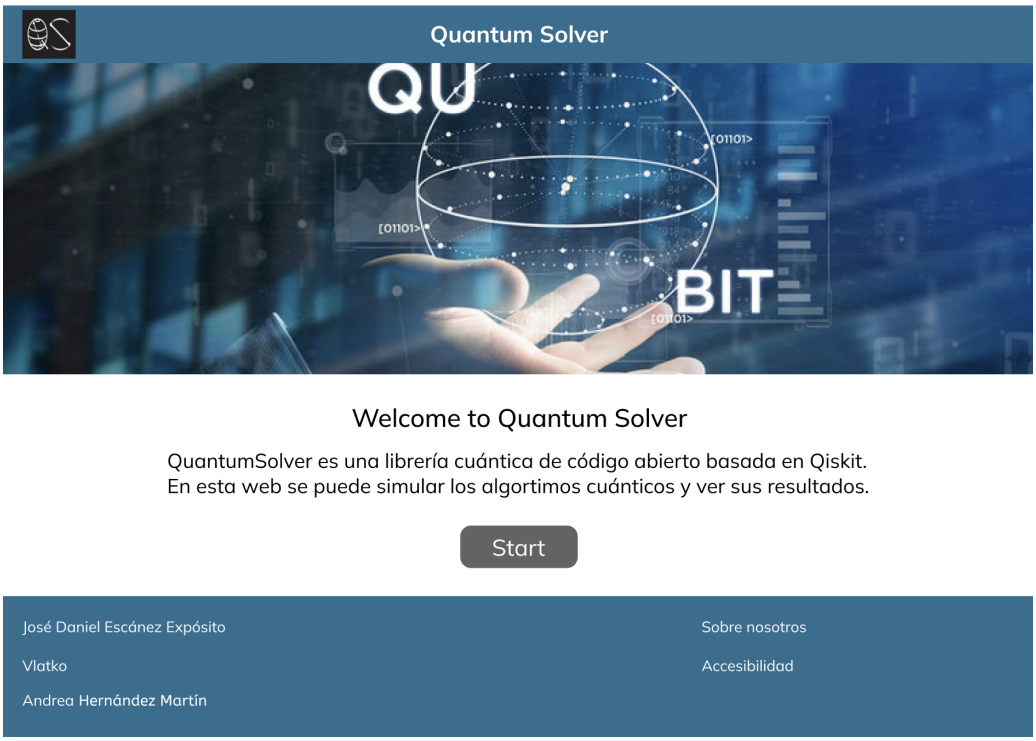


Figura 2.13: Maquetación de la página principal de QuantumSolver.

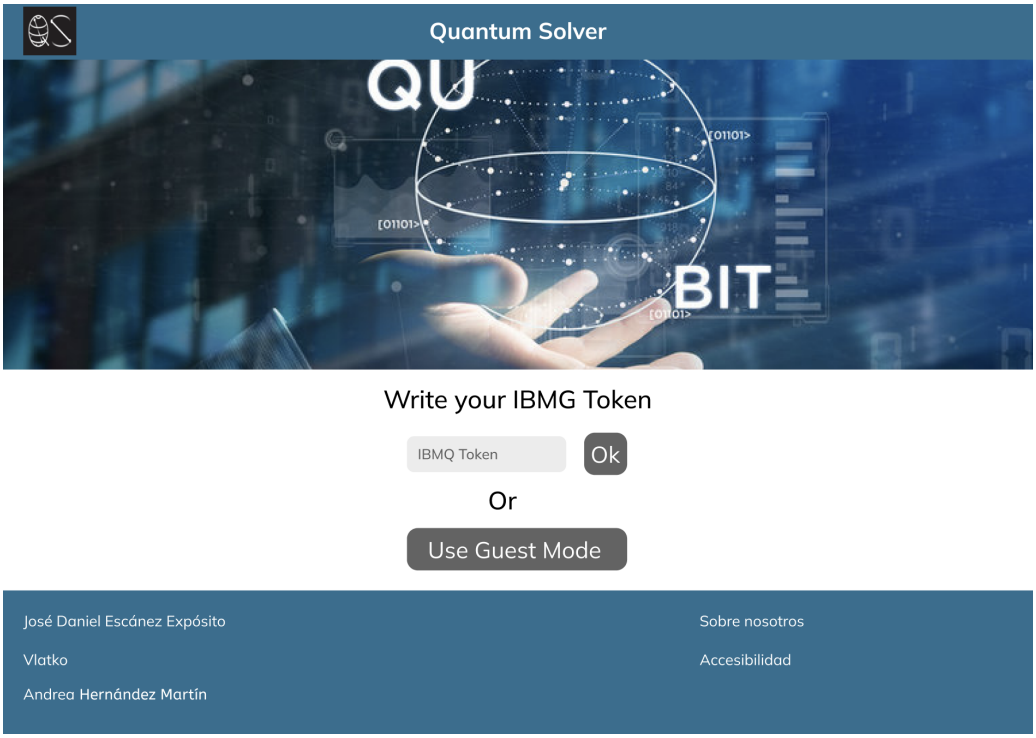


Figura 2.14: Maquetación de la página del token de acceso de QuantumSolver.

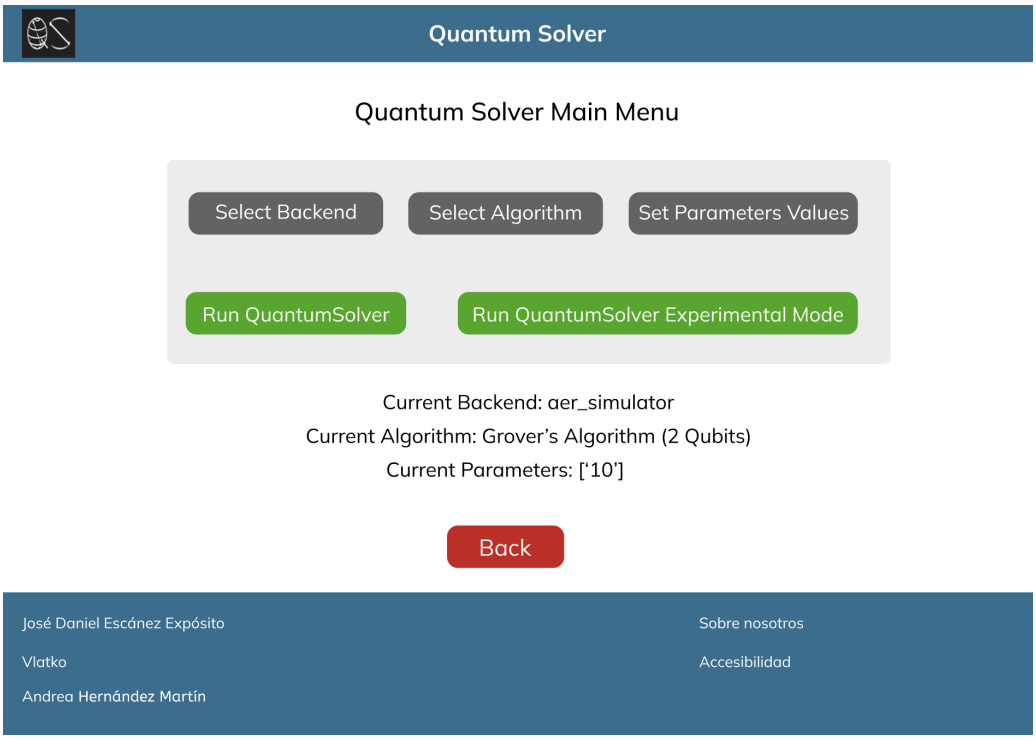


Figura 2.15: Maquetación de la página del menú de QuantumSolver.

Capítulo 3

Simuladores cuánticos de imitación

3.1. Selección de simuladores cuánticos

Hasta ahora en el modo invitado de la librería *QuantumSolver* solo se permitía ejecutar algoritmos con un único simulador cuántico ,'*aer_simulator*'. Sin embargo, este simulador es demasiado fiel al marco teórico dado que carece de ruido y da una aproximación demasiada exacta de lo que predice la formulación matemática. Es por ello que se aleja del estado actual de la tecnología, que tiene interferencias y otros tipos de errores cuánticos como errores de fases de los cúbits, errores de amplitud, etc. De hecho, la detección y corrección de esos errores es todo un sector de investigación.

Por ese motivo se ha decidido introducir nuevos simuladores cuánticos con ruido en la ejecución de la librería *QuantumSolver*. Esos simuladores son proporcionados por *IBM*, que los denomina *Fake Backends*, ya que se trata de simuladores que imitan el comportamiento de los *backends* cuánticos reales de *IBM* [14].

IBM ofrece las versiones *fakes* de los siguientes simuladores: Almaden, Armonk, Athens, Belem, Boeblingen, Bogota, Brooklyn, Burlington, Cairo, Cambridge, Casablanca, Essex, Guadalupe, Hanoi, Jakarta, Johannesburg, Kolkata, Lagos, Lima, London, Manhattan, Manila, Melbourne, Montreal, Mumbai, Nairobi, Ourense, Paris, Poughkeepsie, Kito, Rochester, Rome, Rueschlikon, Santiago, Singapore, Sydney, Tenerife, Tokyio, Toronto, Valencia, Vigo, Washington y Yorktown.

Este listado es exhaustivo, pero en este trabajo se ha decidido realizar una selección reducida para añadir a la librería *QuantumSolver*. Los criterios que se han utilizado para llevar a cabo esta selección vienen dados por las diferencias en cuanto al número de cúbits y el número de repeticiones en la ejecución en un circuito. Este listado se puede encontrar en la Tabla 3.1. Hay que tener en cuenta que hay simuladores que tienen un asterisco en su nombre porque no fue posible su ejecución debido a su número de cúbits, ya que simular un ordenador cuántico en un ordenador clásico es bastante costoso a medida que aumenta el número de cúbits, y la máquina en la que se ejecutaron no lo soportó.

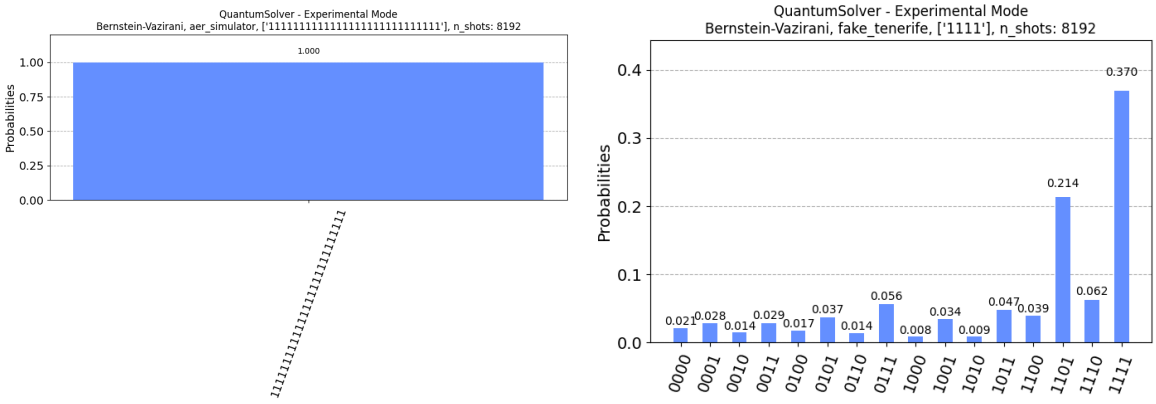
3.2. Resultados

Una vez se añadieron los simuladores a *QuantumSolver*, se decidió probar un algoritmo cuántico ya implementado en la librería, en cada uno de los simuladores con el número máximo de cúbits de cada uno. Se realizó el mismo número de repeticiones en el circuito para todos, 8192, ya que todos los simuladores aceptan ese número de repetición, para

Tabla 3.1: Simuladores cuánticos de imitación escogidos para QuantumSolver

Nombre del simulador	Número de cúbits	Shots
FakeArmonk	1	8192
FakeTenerife	5	65536
FakeCasablanca	7	8192
FakeMelbourne *	14	65536
FakeGuadalupe *	16	8192
FakeTokyo *	20	65536
FakeParis *	27	8192
FakeCambridge *	28	8192
FakeRochester *	53	8192
FakeBrooklyn *	65	8192

poder comparar los distintos resultados. Además, también se ejecutó en el simulador sin ruido *'aer_simulator'* para poder comparar y ver las diferencias que tiene el resultado según dónde sea ejecutado. El algoritmo escogido fue *Bernstein-Vazirani*, cuyo objetivo es determinar una cadena oculta binaria. El resultado iddeal del algoritmo cuando se ejecuta en un entorno sin ruido ni interferencias es 1 para todas las repeticiones, como se puede observar en la Fig. 3.1a, ya que en todos los casos sin ruido debe acertar la clave oculta. Hay que tener en cuenta que el tamaño de clave probado fue 28, ya que el número de cúbits del simulador es 29. Sin embargo, si observamos la Fig. 3.1b, se puede observar que solo acierta en un 37 % de los intentos debido al ruido. El simulador que se ha utilizado es el Fake Tenerife, el cual cuenta con 5 cúbits, por lo que la cadena secreta es de 4. En este caso, en el que la clave es 1111, ha sido más probable que también se determinen como candidatas a clave: 1101, 1110 y 0111.



(a) Ejecución en aer_simulator.

(b) Ejecución en Fake Tenerife.

Figura 3.1: Ejecución del algoritmo Bernstein-Vazirani en distintos simuladores cuánticos.

También en la Fig. 3.2 se puede observar la ejecución del algoritmo en el simulador Fake Casablanca con un tamaño de la cadena de 6, ya que el número de cúbits de este es de 7. En el histograma se puede ver cómo solo el 60 % de los intentos fueron aceptados debido a las interferencias y ruido que tiene este simulador. Para este caso, en el que la clave secreta es 111111, ha sido más probable que se determinen como candidatas a clave secreta: 1111101, 110111 y 0111111. Además, se puede también observar cómo para este simulador muchas de las claves tienen una probabilidad de 0 % de ser aceptadas.

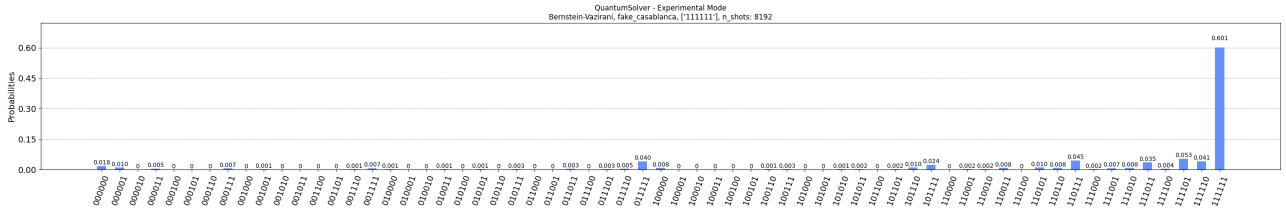
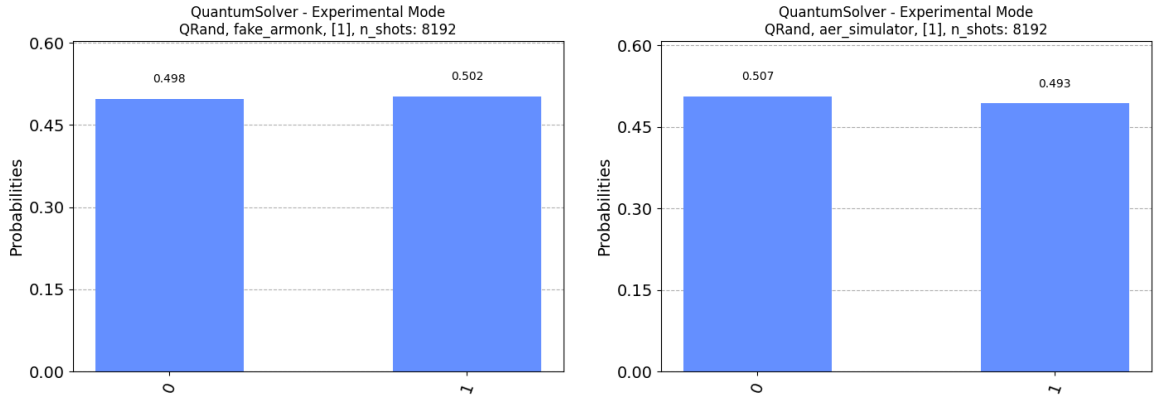


Figura 3.2: Ejecución del algoritmo Bernstein-Vazirani en el simulador Fake Casablanca.

Por último, se utilizó el simulador Fake Armonk, el cual se puede observar en la Fig. 3.3a. Para este caso se tuvo que utilizar otro algoritmo ya que este simulador cuenta con un único cúbit, por lo que la cadena secreta sería imposible probarla en dicho simulador. Por eso se escogió otro algoritmo implementado en *QuantumSolver*, denominado QRand. Este algoritmo genera un número aleatorio entre 0 y $2^n - 1$, donde n es el número de cúbits y el resultado esperado es que el número de repeticiones de cada elemento se distribuya uniformemente. En este caso se ha utilizado un parámetro de un cúbit y el resultado es prácticamente uniforme.

Para obtener resultados comparativos, también se ejecutó el mismo algoritmo con el mismo parámetro en el simulador sin ruido aer_simulator. Su resultado se puede observar en la Fig. 3.3b y como se puede ver, son resultados bastantes parecidos, no hay gran diferencia entre ejecutar este algoritmo con este parámetro en un entorno con ruido que sin él.



(a) Ejecución en Fake Armonk.

(b) Ejecución en aer_simulator.

Figura 3.3: Ejecución del algoritmo Bernstein-Vazirani en distintos simuladores cuánticos.

Capítulo 4

Implementación del protocolo Six-State

4.1. Fundamento teórico del protocolo Six-State

El protocolo criptográfico *Six-State* es un método utilizado en la computación cuántica para la transmisión segura de información entre dos usuarios [9]. De todos los estados posibles de un cúbit, este protocolo utiliza los estados que están proyectados sobre los ejes z, x e y, es decir $\{|0\rangle, |1\rangle\}$, $\{|+\rangle, |-\rangle\}$ y $\{|i\rangle, |-i\rangle\}$, respectivamente.

En este algoritmo dos participantes, Alice y Bob, desean comunicarse de manera segura utilizando un canal de comunicación cuántico unidireccional y un canal clásico bidireccional, ambos públicos. Primero, Alice genera una cadena aleatoria binaria para enviársela a Bob, utilizando los estados básicos $\{|0\rangle, |1\rangle\}$. Después, Alice elige aleatoriamente, para cada cúbit, una de las tres bases de medición (eje z, eje x, eje y), con una probabilidad de $\frac{1}{3}$ para cada una. En el caso de usar el eje z no se aplican transformaciones al cúbit en cuestión. Sin embargo, si se elige el eje x se le aplica la puerta *Hadamard*, también llamada puerta H, cuya representación matricial se puede observar en la Ec. (4.1).

$$H = \frac{1}{\sqrt{2}} \begin{pmatrix} 1 & 1 \\ 1 & -1 \end{pmatrix} \tag{4.1}$$

Por último, si usa el eje y se le aplica una puerta creada a partir de la puerta Z y la puerta Y, cuyas representaciones matriciales se pueden observar en la Ec. (4.2) y (4.3) respectivamente. Además, la puerta creada a partir de ambas se muestra en la Ec. (4.4).

$$Z = \begin{pmatrix} 1 & 0 \\ 0 & -1 \end{pmatrix} \tag{4.2}$$

$$Y = \begin{pmatrix} 0 & -i \\ i & 0 \end{pmatrix} \tag{4.3}$$

$$H_Y = \frac{1}{\sqrt{2}} (Y + Z) = \frac{1}{\sqrt{2}} \begin{pmatrix} 1 & -i \\ i & -1 \end{pmatrix} \tag{4.4}$$

Después de que Alice envíe estos cúbits aleatorios preparados en las bases aleatorias, Bob realiza una medición en cada cúbit que recibe. Dicha medición se realiza de manera aleatoria entre los ejes disponibles (z, x, y). Aproximadamente $\frac{1}{3}$ de los cúbits serán medidos en el mismo eje que el emisor, por lo que se consideran correctos. Los restantes $\frac{2}{3}$, deberán ser descartados dado que al medirlos en el eje equivocado, se tendrá exactamente un 50 % de probabilidad de emitir el valor codificado correcto, lo que implica la perdida total de la correspondiente información. Para realizar este descarte se hacen públicos

los ejes en los que se prepararon y midieron los cúbits, ya que no hay riesgo al realizar tal acción después de haber realizado las mediciones, y se eliminan los que no están medidos en el mismo eje por Alice y Bob. Los valores restantes tras el descarte podrían ser considerados la clave generada, pero antes hay que comprobar su seguridad. Si existen resultados incoherentes entre la clave del emisor y la del receptor, se evidencia la presencia de un atacante. Por ello, Bob debe compartir públicamente una parte de la clave recibida, para que Alice se asegure de que coincide con lo que ella tiene. Si es así, Alice comunica a Bob que puede utilizar el resto de la clave como secreto compartido seguro. En caso contrario, se deduce que se han interceptado los cúbits enviados y que han sido medidos antes de remitirlos al receptor, por alguna entidad intermedia que ha lanzado un ataque de escucha secreta *(eavesdropping)*, y por tanto toda la clave es descartada y se reinicia todo el proceso.

En la *Tabla 4.1* se observan todos los posibles casos, distribuidos en tres agrupaciones de comportamiento similar, en las que se presentan: casos en los que el protocolo es abortado por detectar una comunicación comprometida, y casos en los que no. Concretamente se tienen nueve casos distintos, cada uno con una probabilidad de suceder de $\frac{1}{9}$.

Tabla 4.1: Casos posibles de mediciones de un cúbit en fase de verificación de Six-State

Medición de emisor y receptor legítimo	Medición del atacante intermedio	Conclusión sobre ese cúbit	Probabilidad de suceder
Mismo eje	Mismo eje que ambos	Ha sido interceptado por el atacante sin que se aborte el protocolo, con un 100 % de probabilidad de que el valor final sea correcto	Sucede en $\frac{1}{9}$ casos
Mismo eje	Distinto eje que ambos	En el 50 % de estos casos se abortará el protocolo al detectar la interceptación	Sucede en $\frac{2}{9}$ casos
Distinto eje	Cualquier eje	Es desechado en la fase de descarte de los valores, sin tener en cuenta al atacante	Sucede en $\frac{6}{9}$ casos

En este protocolo se gana en seguridad con respecto a sus predecesores debido a que las tres bases abarcan toda la *Esfera de Bloch* mostrada en la Fig. 4.1 con los 6 estados que utiliza el protocolo. Por ejemplo, en el caso del protocolo *BB84*[10] solo se utiliza un plano bidimensional.

4.2. Implementación del protocolo Six-State

El desarrollo del protocolo *Six-State* se ha realizado dentro del módulo *QuantumSolver Crypto*. Para implementarlo se han diseñado tres entidades: *Participant, Sender* y *Receiver*; siendo la primera la entidad principal base y las demás sus entidades derivadas. La finalidad es que una instancia de *Sender* pueda comunicarse con una instancia de *Receiver* haciendo que la comunicación sea privada gracias al protocolo. Se comprueba

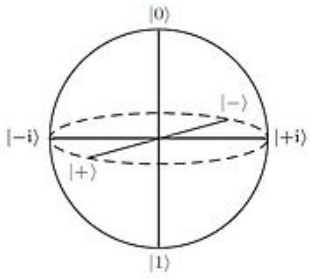


Figura 4.1: Esfera de Bloch con 3 bases

que la comunicación es privada gracias a una libreta de un solo uso que se genera en cada comunicación y que además solo la comparten emisor y receptor.

La clase principal *Participant* contiene los métodos para la generación y muestra de valores, ejes, claves y libretas de un solo uso. También tiene un método encargado de crear una puerta cuántica capaz de medir o preparar los cúbits enviados respecto al eje y, cuya implementación se puede observar en el Código (4.1). Esta puerta se ha implementado gracias a las funcionalidades de Qiskit, utilizando la clase *Operator*, para representar los operadores matriciales que actúan sobre el sistema cuántico. Se ha utilizado obteniendo el valor la suma de las puertas cuánticas Y y Z, multiplicada por el factor $\frac{1}{\sqrt{2}}$ (véase Ec. (4.4)). Para la implementación se crea un circuito cuántico y se utiliza el método *unitary* para añadir la puerta necesaria a partir de la matriz calculada. Por último, se convierte el circuito en una puerta mediante el método *to_gate*.

Listing 4.1: Creación de la puerta Hy

```
def set_hy(self):
  hy_op = qi.Operator(1/np.sqrt(2)*
    (YGate().to_matrix()+
    ZGate().to_matrix()))
  hy_gate = QuantumCircuit(1)
  hy_gate.unitary(hy_op,[0],label='h_y')
  self.hy = hy_gate.to_gate()
```

Las clases *Sender* y *Receiver* son bastante similares. La primera tiene un método para enviar un mensaje, el cual inicializa el circuito cuántico; y la segunda tiene otro para recibirlo añadiendo la fase de medición al circuito. Además hay que tener en cuenta que ambas clases para la preparación o medición de los cúbits, usan la puerta cuántica vinculada al eje y explicada anteriormente en el código (4.1).

Todas las simulaciones realizadas en el canal cuántico se realizan mediante la utilización de circuitos cuánticos de *Qiskit*. La implementación del protocolo de manera más detallada se puede encontrar en sus respectivos ficheros de código fuente [12].

4.3. Integración del protocolo en la librería

Al ejecutar el programa principal de *QuantumSolver Crypto*, se muestra un menú que permite elegir *Six-State* como protocolo a utilizar. A continuación, se da la opción de utilizar el token de IBM o entrar como usuario invitado, así como de elegir el *backend* a utilizar. Una vez elegido, se puede simular el protocolo de dos maneras.

La primera forma permite ejecutar el algoritmo una única vez con los parámetros

necesarios: una cadena de caracteres como mensaje y un valor entre 0 y 1 como densidad de interceptación, que es la probabilidad con la que el receptor intermedio ilegítimo medirá cada cúbit. Una vez establecidos los parámetros que se utilizarán en la simulación, el programa muestra una traza con los resultados obtenidos. En esta traza se muestran los ejes de preparación o medición de Alice, Bob y Eva (receptor ilegítimo) con valores aleatorios entre 0 y 2, ya que tienen tres ejes para elegir. También se pueden observar los valores preparados o medidos de 0 y 1, y en el caso de Eva toman -1 cuando no se haya realizado esa medición. Además, se muestra la clave privada de Alice y Bob, y la clave que comparte Bob para verificar la seguridad de la misma. Por último, se observa un mensaje de error en caso de que se detectase la interceptación del mensaje, abortando el protocolo; y en el caso contrario, se observa un aviso de que la comunicación fue segura.

La segunda manera de ejecutar el programa es mediante el modo experimental, en el que el algoritmo se ejecuta varias veces, mostrando un mapa de calor que representa mediante colores más claros las condiciones en las que ha habido más ocasiones de comunicación considerada segura. Con colores más oscuros se muestran los casos donde ha habido comunicaciones en las que se han detectado más interceptaciones. Los parámetros a especificar son distintos a los del modo anterior. En este caso hay que establecer: la longitud máxima del mensaje en número de bits como número positivo múltiplo de 5 hasta ese máximo, conformando el eje x en el mapa de calor; la magnitud de la distancia entre casos de la densidad de interceptación, que toma valores entre 0 y 1, y que define el eje y en el mapa de calor; y el número de repeticiones para cada instancia. La traza que se obtiene del modo experimental es diferente a la anterior. En este caso se observan los valores que van tomando los diferentes parámetros durante la ejecución, así como el tiempo estimado de ejecución de las instancias restantes del protocolo.

4.4. Resultados

4.4.1. Protocolo Six-State

Si ejecutamos el protocolo una única vez como se explicó en el apartado anterior con el *backend* y los parámetros seleccionados, se puede apreciar en las Fig. 4.2 , Fig. 4.3 y Fig. 4.4 una misma traza en la que se genera una libreta de un solo uso (al no existir interceptación en la comunicación) suficiente para enviar el mensaje “quantum”.

Además en la Fig. 4.4 al final se puede observar cómo se determina la comunicación segura, por lo que los participantes de la comunicación comparten la libreta de un solo uso. El emisor transmite su mensaje codificado con la total confianza de que solo el receptor legítimo será capaz de decodificarlo.

En la Fig. 4.5 no se llega a generar la libreta de un solo uso, dado que se encuentran las discrepancias entre los inicios de las claves generadas y se detecta la intercepción ocurrida.

Por otra parte, si se ejecuta el modo experimental se puede observar el mapa de calor como resultado. A continuación, en la Fig. 4.6 se muestran dos ejemplos de mapas de calor generados, correspondientes a los parámetros de la Tabla 4.2 ilustrando dos casos del modo experimental, uno de ellos más sencillo y rápido de computar; y otro más detallado y lento de procesar.

En las Fig. 4.7 y Fig. 4.8 se muestra la interfaz de ejecución de este modo expermiental, tras la selección del backend deseado y el establecimiento de los parámetros exigidos.

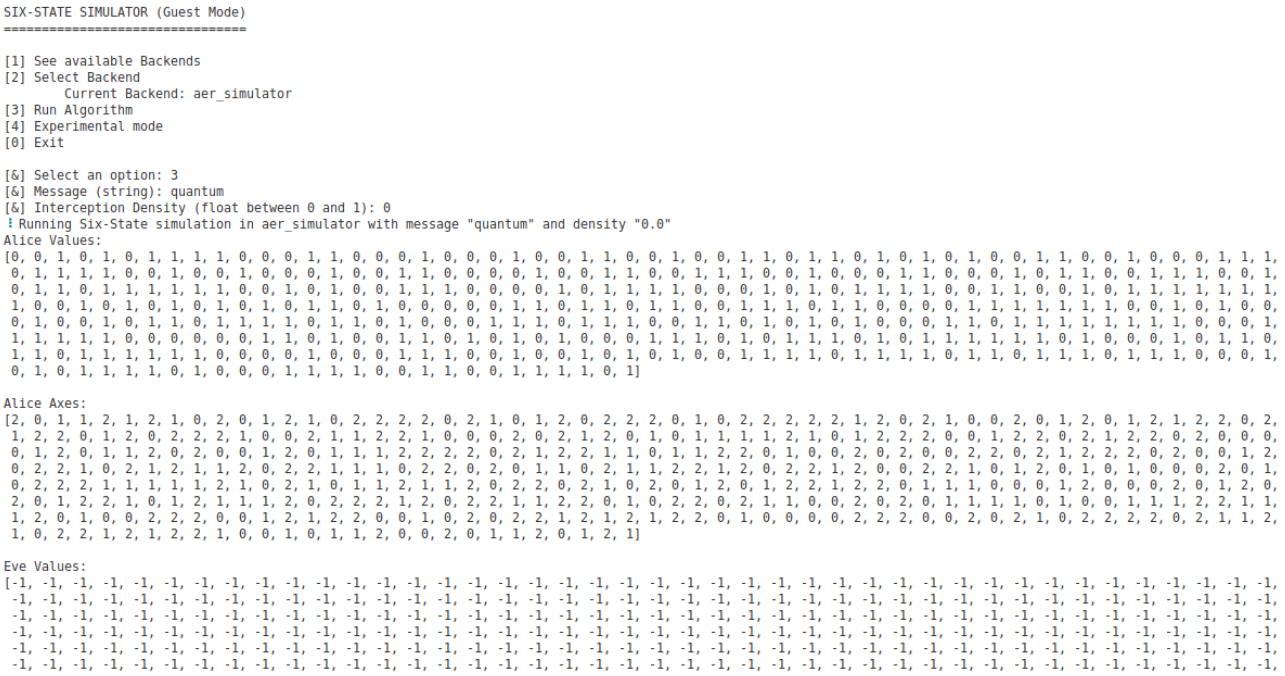


Figura 4.2: Inicio del protocolo.

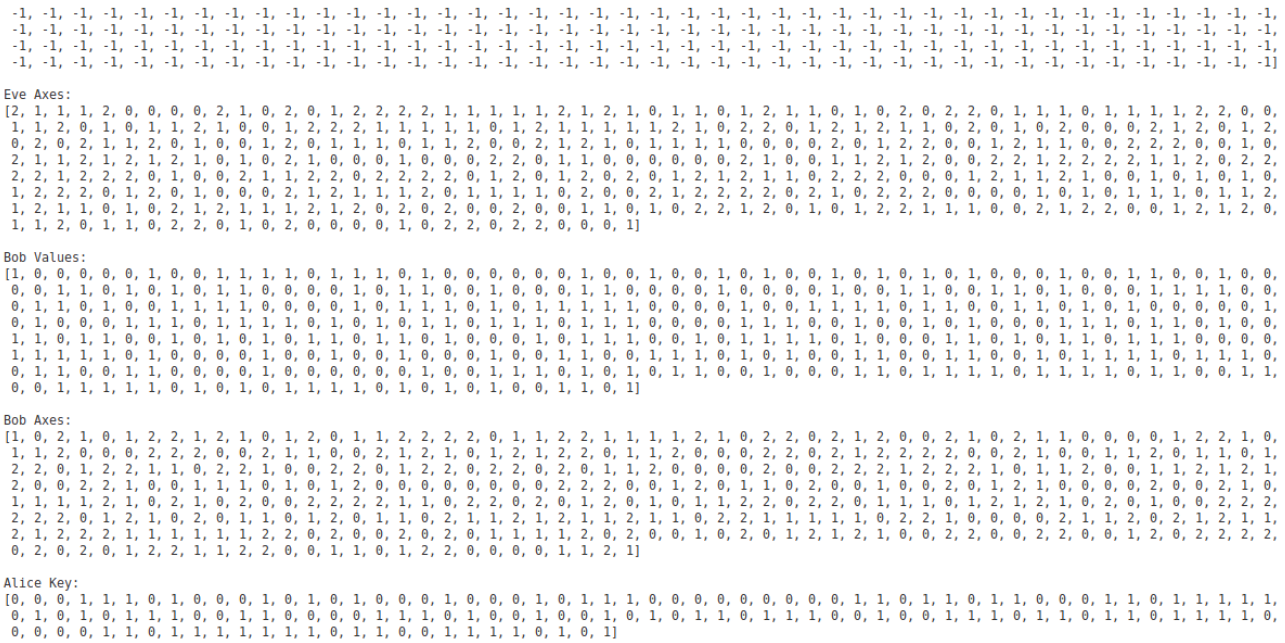


Figura 4.3: Continuación del protocolo.

Tabla 4.2: Parámetros de los mapas de calor en el modo experimental

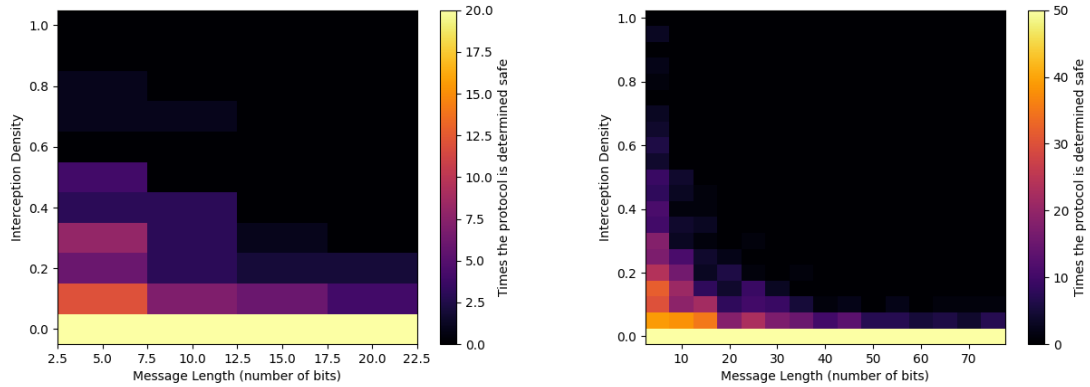
Figura	Longitud máxima del mensaje	*Step* de densidad de interceptación	Repeticiones de cada instancia	Tiempo de ejecución (aprox.)
4.6a	20 bits	0.1	20	24 minutos
4.6b	75 bits	0.05	50	1320 minutos

```
Bob Key:
[0, 0, 0, 1, 1, 1, 0, 1, 0, 0, 0, 1, 0, 1, 0, 1, 0, 0, 0, 1, 0, 0, 0, 1, 0, 1, 1, 1, 0, 0, 0, 0, 0, 0, 0, 0, 1, 1, 0, 1, 1, 1, 0, 1, 1, 0, 0, 0, 1, 1, 0, 1, 1, 1, 1, 1,
 0, 1, 0, 1, 0, 1, 1, 1, 0, 0, 1, 1, 0, 0, 0, 0, 1, 1, 1, 0, 1, 0, 0, 1, 0, 0, 1, 0, 1, 0, 1, 1, 0, 1, 1, 1, 0, 0, 1, 0, 0, 1, 1, 1, 0, 1, 1, 0, 1, 1, 0, 1, 1, 1, 1, 0,
 0, 0, 0, 0, 1, 1, 0, 1, 1, 1, 1, 1, 1, 1, 0, 1, 1, 0, 0, 1, 1, 1, 1, 0, 1, 0, 1]

Shared Bob Key:
[0, 0, 0, 1, 1, 1, 0, 1, 0, 0, 0, 1, 0, 1, 0, 1, 0, 0, 0, 1, 0, 0, 0, 1, 0, 1, 1, 1, 0, 0, 0, 0, 0, 0, 0, 0, 1, 1, 0, 1, 1, 1, 0, 1, 1, 0, 0, 0, 1, 1, 0, 1, 1, 1, 1, 1,
 0, 1, 0, 1, 0, 1, 1, 1, 0, 0, 1, 1, 0, 0]

Final Keys

Alice Key:
[0, 0, 1, 1, 1, 0, 1, 0, 0, 1, 0, 0, 1, 0, 1, 0, 1, 1, 0, 1, 1, 1, 0, 0, 1, 0, 0, 1, 1, 1, 0, 1, 1, 0, 1, 1, 0, 1, 1, 1, 1, 0, 0, 0, 0, 0, 1, 1, 0, 1, 1, 1, 1, 1, 1, 1,
 0, 1, 1, 0, 0, 1, 1, 1, 1, 0, 1, 0, 1]

Bob Key:
[0, 0, 1, 1, 1, 0, 1, 0, 0, 1, 0, 0, 1, 0, 1, 0, 1, 1, 0, 1, 1, 1, 0, 0, 1, 0, 0, 1, 1, 1, 0, 1, 1, 0, 1, 1, 0, 1, 1, 1, 1, 0, 0, 0, 0, 0, 1, 1, 0, 1, 1, 1, 1, 1, 1, 1,
 0, 1, 1, 0, 0, 1, 1, 1, 1, 0, 1, 0, 1]

Secure Communication!

Alice OTP:
[14, 36, 43, 28, 39, 27, 30, 3, 31, 54, 30, 5]

Bob OTP:
[14, 36, 43, 28, 39, 27, 30, 3, 31, 54, 30, 5]

Initial Message:
quantum
Encoded Message:
QJrSns
💡 Decoded Message:
quantum

✅ The initial message and the decoded message are identical
✔ Running Six-State simulation in aer_simulator with message "quantum" and density "0.0"
  Six-State simulation runned in 3825.6924152374268 ms
```

Figura 4.4: Resultado del protocolo.

```
Alice Key:
[0, 0, 1, 1, 1, 1, 1, 1, 0, 0, 0, 1, 1, 1, 1, 0, 1, 1, 0, 1, 1, 1, 0, 0, 1, 0, 0, 1, 1, 0, 1, 1, 1, 0, 1, 1, 1, 0, 1, 1, 1, 0, 0, 0, 0, 1, 0, 0, 1, 0, 0, 0, 1, 1,
 0, 1, 1, 0, 1, 1, 1, 0, 1, 1, 1, 1, 1, 0, 1, 0, 0, 0, 0, 0, 1, 0, 1, 1, 0, 1, 0, 1, 0, 1, 0, 1, 0, 0, 0, 0, 0, 1, 1, 1, 0, 1, 1, 0, 1, 0, 0, 1, 1, 1, 1, 1, 1, 0, 1, 1, 0, 1,
 1, 0, 1, 1, 0, 1, 1, 1, 0, 1, 0, 1, 0, 0, 0, 0, 0, 1, 1, 1, 0, 1, 0, 0, 0, 1, 0, 0, 1, 1, 0, 0, 1, 0, 0]

Bob Key:
[1, 0, 0, 1, 0, 1, 1, 0, 0, 0, 0, 1, 0, 1, 1, 0, 0, 1, 1, 0, 1, 1, 0, 1, 0, 1, 0, 1, 0, 0, 0, 1, 0, 1, 0, 1, 0, 0, 0, 0, 1, 1, 0, 0, 1, 0, 1, 0, 0, 1, 0, 1, 1, 1, 0, 0,
 0, 1, 1, 0, 0, 1, 0, 1, 1, 0, 1, 0, 1, 1, 1, 1, 1, 1, 0, 1, 1, 1, 0, 1, 0, 0, 1, 1, 0, 1, 0, 1, 0, 0, 1, 1, 1, 1, 1, 1, 0, 0, 1, 0, 1, 1, 0, 1, 1, 1, 1, 1, 1, 0, 0, 1,
 0, 0, 1, 1, 0, 1, 0, 0, 1, 0, 1, 1, 1, 1, 1, 1, 0, 1, 1, 0, 0, 1, 0, 0, 1, 0, 1, 0, 1, 1, 1, 1, 1, 1, 1, 0, 0]

Shared Bob Key:
[1, 0, 0, 1, 0, 1, 1, 0, 0, 0, 0, 1, 0, 1, 1, 0, 0, 1, 1, 0, 1, 1, 0, 1, 0, 1, 0, 1, 0, 0, 0, 1, 0, 1, 0, 1, 0, 0, 0, 0, 1, 1, 0, 0, 1, 0, 1, 0, 0, 1, 0, 1, 1, 1, 0, 0,
 0, 1, 1, 0, 0, 1, 0, 1, 1, 0, 1, 0, 1, 1, 1, 1, 1, 1]

Unsecure Communication! Eve has been detected intercepting messages

❌ Message not send
✔ Running Six-State simulation in aer_simulator with message "insecure" and density "1.0"
  Six-State simulation runned in 9295.258283615112 ms
```

Figura 4.5: Resultado del protocolo.



(a) Primer ejemplo.

(b) Segundo ejemplo.

Figura 4.6: Mapas de calor generados por *Six-State*.

```
SIX-STATE SIMULATOR (Guest Mode)
================================

[1] See available Backends
[2] Select Backend
        Current Backend: aer_simulator
[3] Run Algorithm
[4] Experimental mode
[0] Exit

[&] Select an option: 4
[&] Specify maximum message length (number of bits): 20
[&] Specify density step: 0.1
[&] Specify number of repetitions for each instance: 5

Running Six-State Simulator Experiment (in aer_simulator):
|                                        | ▄▄ 39/220 [18%] in 45s (0.9/s, eta: 3:23)
```

Figura 4.7: Modo experimental *Six-State* con la barra de carga.

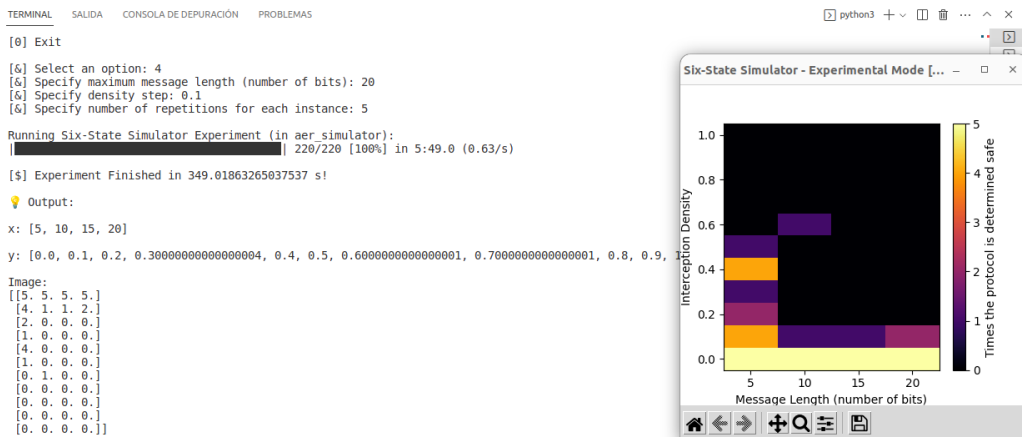


Figura 4.8: Resultado del modo experimental de *Six-State*.

4.4.2. Comparación entre el protocolo Six-State y BB84

Aprovechando que el protocolo *BB84* también está implementado en la librería *QuantumSolver* se ha realizado un análisis de resultados que permite demostrar la mejora que representa el protocolo *Six-State* en comparación con el protocolo *BB84*.

En las Fig. 4.9 y 4.10 se muestran dos ejemplos de mapas de calor generados. Por un lado, es importante destacar el tiempo de ejecución de ambos protocolos. Mientras que el *Six-State* tardó aproximadamente 22 horas en finalizar su ejecución para generar el mapa de calor, el *BB84* tardó 22 horas aproximadamente en generar el mapa de calor con los mismos parámetros que el anterior. Esto es debido a la nueva puerta que se tuvo que crear para poder medir en el eje y, ya que esta puerta no está definida dentro de las instrucciones primarias del simulador cuántico que se está utilizado.

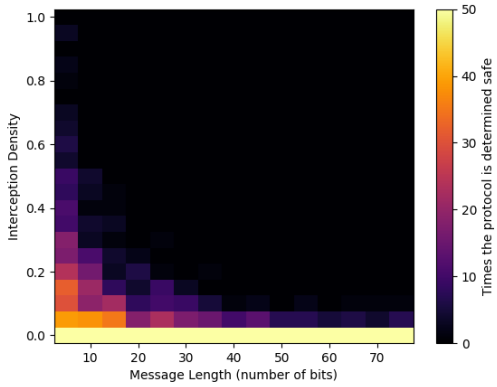


Figura 4.9: Ejemplo de mapa de calor del protocolo *Six-State*.

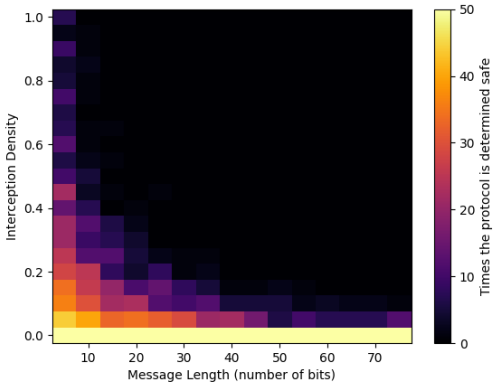


Figura 4.10: Ejemplo de mapa de calor del protocolo *BB84*

Si se observan ambas imágenes se puede ver cómo la curva del *Six-State* es ligeramente mejor que la del *BB84*, ya que esta última es ligeramente más alta, por lo que se consideran como buenos casos que no lo son, es decir, se está detectando menos al interceptor. Esto es debido a que el protocolo *Six-State* utiliza tres estados cuánticos no ortogonales que permiten obtener información adicional sobre el estado enviado [11]. Además hace que sea más difícil descifrar el mensaje para el interceptor ya que tiene tres ejes en los que medir, a diferencia del *BB84* que tiene dos. Por tanto, la probabilidad de que Eva mida en el mismo eje que Alice y Bob es menor.

Por otro lado, se debe tener en cuenta que cuanto más claro es el color, en más ocasiones se determina segura la comunicación, y cuanto más oscuro, más insegura. Al observar los gráficos, se considera que cuánto más crece el número de bits del mensaje a enviar se tiene que la probabilidad de detectar al interceptor es mayor, ya que como se puede observar, el gráfico es todo prácticamente oscuro según crece la variable del eje x.

Sin embargo, se ha comprobado el comportamiento en caso de que el número de bits sea pequeño. En ese caso el protocolo *Six-State* es más seguro ya que cuando se tienen pocos bits y la densidad de interceptación es mayor, la curva es más pequeña y los colores son más oscuros. Esto es debido a que se están desechando los casos en los que se intercepta el mensaje, a diferencia del *BB84* que cuando hay casos en los que con un número muy pequeño de bits y la densidad de interceptación es cercana a 1, la comunicación se da como segura cuando en realidad se está interceptando el mensaje prácticamente completo.

Cabe destacar el caso en el que la densidad de interceptación es cero para todos los tamaños del número de bits, observándose en el gráfico como el protocolo se considera seguro en el 100 % de los casos, desde que el protocolo se ha ejecutado en un entorno sin ruido.

Finalmente, se puede llegar a la conclusión de que el protocolo *Six-State* es más seguro que el protocolo *BB84* debido a su capacidad para detectar a los interceptores.

Capítulo 5

Conclusiones y líneas futuras

La librería *QuantumSolver* es una herramienta muy útil ya que permite la ejecución de software cuántico de manera amigable para todo tipo de público interesado. Gracias a ella el aprendizaje sobre criptografía y computación cuánticas resulta más sencillo por la posibilidad que ofrece de simular algoritmos y protocolos cuánticos, obteniendo resultados visuales. Además, permite manejar el correspondiente código software, lo que facilita que esté en constante evolución, pudiéndose prever que en un futuro próximo contenga muchos más algoritmos y protocolos cuánticos.

En este trabajo se ha presentado una propuesta de mejora del acceso a la librería *QuantumSolver*, que facilita el uso a todo tipo de usuarios. Además se ha descrito una implementación original de un nuevo protocolo añadido a la librería: el protocolo *Six-State* para distribución de claves cuánticas, que incluye el diseño de una puerta lógica cuántica para vincular el eje 'y'. Este protocolo es un método criptográfico cuántico que se espera que en el futuro sea utilizado en aplicaciones prácticas de ciberseguridad dado que, como se ha podido comprobar en los resultados de la implementación realizada, presenta una mayor fiabilidad que el conocido protocolo *BB84*.

Por otro lado, se han obtenido resultados sobre algoritmos ejecutados en entornos con ruido, gracias a que se añadieron simuladores de imitación en la librería, permitiendo obtener una gran variedad de resultados diferentes.

Algunas propuestas de líneas de trabajo futuro podrían ser optimización de la puerta propuesta vinculada al eje 'y' para reducir los tiempos de ejecución de las simulaciones. También se pretende ampliar los estudios sobre el protocolo Six-State para intentar obtener resultados más relevantes. Además se propone la futura implementación de otros algoritmos criptográficos como puede ser el *BBM92* y el *SARG04*.

Capítulo 6

Summary and Conclusions

The library *QuantumSolver* is a very useful tool as it allows the execution of quantum software in a user-friendly manner for all types of interested audiences. Thanks to it, learning about quantum cryptography and computing becomes simpler due to the possibility it offers to simulate quantum algorithms and protocols, obtaining visual results. Additionally, it allows handling the corresponding software code, facilitating its constant evolution, and it is expected to include many more quantum algorithms and protocols in the near future.

In this work, a proposal has been presented to improve access to the *QuantumSolver* library, making it easier for all types of users to use. Furthermore, an original implementation of a new protocol added to the library has been described: the *Six-State* protocol for quantum key distribution, along with the design of a quantum logic gate to link the ’y’ axis. This protocol is a quantum cryptographic method expected to be used in practical cybersecurity applications in the future, as demonstrated by the implementation results, showing higher reliability than the well-known *BB84* protocol.

Moreover, results have been obtained for algorithms executed in noisy environments, thanks to the addition of simulation emulators in the library, allowing for a wide variety of different results.

Some proposed directions for future work could include optimizing the proposed gate linked to the ’y’ axis to reduce simulation execution times. Furthermore, there is a plan to further expand studies on the Six-State protocol to achieve more relevant results. Additionally, the future implementation of other cryptographic algorithms such as *BBM92* and *SARG04* is proposed.

Capítulo 7

Presupuesto

A continuación, se realiza una estimación de los costes del proyecto. Se estiman las horas utilizadas en la realización del mismo, así como el coste de componentes físicos utilizados.

7.1. Costes de personal

Tabla 7.1: Costes de personal

Tarea	Tiempo	Coste	Coste total
Investigación realizada sobre el marco teórico cuántico y el uso de las tecnologías necesarias	50h	24€/h	1200€
Planificación del proyecto	15h	16€/h	288€
Lanzamiento público de la web	100h	20€/h	2000€
Implementación simuladores cuánticos	25h	30€/h	750€
Implementación protocolo cuántico	150h	30€/h	4500€
Obtención de resultados y redacción de la memoria	60h	30€/h	1800€
Total	400h		10538€

7.2. Costes de componentes

El equipo utilizado durante la realización del proyecto está valorado en, aproximadamente, 700€ y tiene las siguientes características:

- Procesador: Intel(R) Core(TM) i7-7500U CPU @2.7GHz, 4 procesadores principales, 8 procesadores lógicos
- Tipo de sistema: PC basado en x64
- Memoria física instalada (RAM): 16,00 GB
- Factor de forma de la Memoria RAM: DDR4
- Unidad del disco duro: 1 TB HDD
- Nombre del SO: Ubuntu 22.04.2 LTS

7.3. Coste total

Tabla 7.2: Coste total

Presupuesto	Coste
Personal	10538€
Componentes	700€
Total	11238€

Apéndice A

Artículo enviado a congreso

A.1. VIII Jornadas Nacionales de Investigación en Ciberseguridad JNIC (aceptado)

Implementación del protocolo criptográfico Six-State.
Andrea Hernández-Martín, Pino Caballero-Gil, Daniel Escánez-Expósito.
VIII Jornadas Nacionales de Investigación en Ciberseguridad (JNIC).
Vigo, 21 a 23 de junio de 2023.

Bibliografía

[1] IBM Research, “Quantum Computing - IBM Research”. [Online]. Available: https:research.ibm.com/quantum-computing. [Accessed: 27-Mar-2027].

[2] Google LLC, “Google Quantum AI”. [Online]. Available: https://quantumai.google/. [Accessed: 27-Mar-2023].

[3] Amazon Inc., “Amazon Braket”. [Online]. Available: https://aws.amazon.com/es/braket/. [Accessed: 27-Mar-2023].

[4] IBM España Newsroom, “IBM presenta su procesador cuántico de más de 400 qubits y la próxima generación de IBM Quantum System Two”. [Online]. Available: https://es.newsroom.ibm.com/122752. [Accessed: 27-Mar-2023].

[5] Gambetta, J., "Expanding the IBM Quantum roadmap to anticipate the future of quantum-centric supercomputing". IBM Research Blog, 2022.

[6] Castelvecchi, D., “Google’s quantum computer hits key milestone by reducing errors”. Nature, 2023.

[7] Gibney, E., “Hello quantum world! Google publishes landmark quantum supremacy claim”, Nature, vol. 574, no. 7779, pp. 461-463, 2019.

[8] Escanez-Exposito, D., “QuantumSolver”. [Online]. Available: https://github.com/jdanielescanez/quantum-solver. [Accessed: 15-Mar-2023].

[9] Bruß, D., “Optimal eavesdropping in quantum cryptography with six states”. Physical Review Letters, 81(14), 3018, 1998.

[10] C. H. Bennett and G. Brassard, “Quantum cryptography: Publickey distribution and coin tossing,” arXiv, 1984.

[11] Senekane, M., Mafu, M., Petruccione, F., “Six-State Symmetric Quantum Key Distribution Protocol”. Journal of Quantum Information Science, 5(02), 33-40, 2015.

[12] Hernández, A., Escanez-Exposito, D., “Six-State Implementation with QuantumSolver”. [Online]. Available: https://github.com/jdanielescanez/quantum-solver/tree/main/src/crypto/six_state. [Accessed: 17-Mar-2023].

[13] Hernández, A. “QuantumSolver Web en Heroku”. [Online]. Available: https://quantum-solver-web.herokuapp.com/. [Accessed: 17-Mar-2023].

[14] IBM, “Qiskit Fake Provider”. [Online]. Available: https://qiskit.org/documentation/apidoc/providers_fake_provider.html. [Accessed: 19- May-2023].