Universidad de Valladolid

ESCUELA DE INGENIERÍA INFORMÁTICA DE SEGOVIA

Grado en Ingeniería Informática de Servicios y Aplicaciones

Estudio del protocolo BB84 de criptografía cuántica

Alumno: Jorge Martín Villafruela

Tutor/a/es: Juan José Álvarez Sánchez

Estudio del protocolo BB84 de computación cuántica

Jorge Martín Villafruela

Índice general

Índice de figuras

Índice de cuadros

Dedicado a
mi familia

Agradecimientos

Muchas gracias a todos

Resumen

La computación cuántica es un paradigma de computación que hasta hace poco solo tenía utilidad práctica. Con la reciente construcción de cada vez más ordenadores cuánticos, este nuevo paradigma está obteniendo una utilidad práctica. Entre las ramas donde encuentra esa utilidad, es en la de la seguridad.

El protocolo BB84 solventa una de las grandes debilidad que tiene la seguridad actual: la generación de claves privadas. En este proyecto, se estudiará una implementación de dicho protocolo en python con ayuda de la librería PennyLane, así como el estudio de la fiabilidad (y mejora) del protocolo al trabajar sobre un canal con ruido, donde la detección de un posible atacante es más discreta.

Palabras claves: computación cuántica, protocolo BB84, seguridad, estudio teórico, PennyLane, python, Jupyter.

Parte I

Memoria del Proyecto

Capítulo 1

Descripción del proyecto

El proyecto consiste en la realización de una serie de cuadernos *Jupyter* explicando el funcionamiento del protocolo BB84, un protcolo de generación de claves privadas en criptografía cuántica. Por ello, también se realizará una breve introducción a la computación. Además, utilizando la librería `PennyLane`, se implementará y explicará el protocolo en Phyton. También se analiza la aplicación de dicho protocolo en un canal con ruido, y cómo en este entorno su implementación inicial se vuelve obsoleta. Por último, para completar el proyecto, se sugiere una modificación del protocolo para que puede ser utilizable en este entorno, junto a la generación de varias muestras estadísticas que justifican dicho razonamiento.

En esta Memoria se desarrollará tanto los aspectos teóricos del proyecto, donde se explicará tanto la base teórica como la programación del mismo, así como analizar el desarrollo de los cuadernos como proyecto software; explicando la metodología utilizada, así como su planificación temporal y desarrollo de los presupuestos, y la documentación del proyecto realizada a partir del análisis de requisitos y realización de diagramas. Al final de esta Memoria además se encontrarán unos breves manuales de instalación y uso de cuadernos *Jupyter*.

1.1. Introducción

La computación cuántica es todavía una tecnología emergente bastante desconocida y en pleno auge. Por ello, la creación de contenido didáctico que pueda servir como introducción a este paradigma de computación (tanto a nivel personal como general) es necesaria actualmente, sobre todo en español donde toda la bibliografía relacionada es escasa por no decir inexistente.

Entre las ramas de la informática donde la computación cuántica parece cobrar más relevancia, se encuentra el campo de la seguridad informática, debido a su naturaleza probabilística. Este Trabajo de Fin de Grado se centra en el estudio del protocolo de cifrado BB84, de manera de que sirva como introducción a este nuevo paradigma, añadiendo para ello simulaciones en *Python* de circuitos cuánticos. También se analiza cómo afecta

al protocolo el hecho de que actúe sobre un canal con ruido, donde la detección de un intruso es más costoso al poder camuflarse el atacante con el propio ruido.

Para implementar dichas simulaciones, se hace uso de la librería *PennyLane*, desarrollada por **Xanadu**(sección 6.1.3).

En esta memoria se discutirá cómo se ha planificado y ejecutado el desarrollo de estos cuadernos *Jupyter*, así como explicar los contenidos teóricos de los cuadernos y el funcionamiento de las librerías de *Python* utilizadas, con especial atención a la librería de *Pennylane*. A continuación, aparecerá la documentación del proyecto, compuesta por los requisitos del proyecto y los distintos diagramas que han dictado la elaboración del mismo.

En una sección aparte se comentará las librerías utilizadas en el proyecto, así como el razonamiento del desarrollo de las principales funciones que lo componen. Asimismo, se comentarán las pruebas de integridad realizadas sobre el código desarrollado. Finalmente, se incluirá un manual de instalación y uso de los cuadernos *Jupyter*.

1.2. Objetivos del trabajo

Al ser un estudio sobre una nueva tecnología emergente, en el ámbito personal este trabajo ha servido para introducirme al manejo de este nuevo paradigma de computación. Además, he podido asentar mis conocimientos en *Python*. Más específicamente, la creación de cuadernos *Jupyter* y el uso de la librería *pennylane*, utilizada para la simulación en *Python* de circuitos cuánticos.

A nivel teórico, el objetivo principal del proyecto es explicar la mejora que significa la aplicación de la computación cuántica en general y del protocolo BB84 en particular, en el problema de la generación del claves privadas, poniendo énfasis en la efectividad del protocolo en un canal con ruido. En concreto, el objetivo final de los cuadernos es llegar a conseguir una mejora del protocolo para que su rendimiento en un entorno con ruido mejore.

Por último, no hay que olvidar el carácter didáctico de los cuadernos que conforman el estudio, puesto que también se quiere que puedan servir como introducción al paradigma de la computación cuántica nivel teórico, usando el protocolo BB84 como guía para ir aprendiendo sobre las características y conceptos únicos de la computación cuántica. De la misma manera, se puede utilizar como ejemplo de implementación de circuitos cuánticos utilizando la librería de PennyLane.

1.3. Entorno del proyecto

1.3.1. Entorno general

Dentro del entorno general del proyecto; al ser un estudio teórico, cobran mayor relevancia los factores tecnológicos. En concreto, el estado actual de la computación cuántica.

Hasta antes de la anterior década, todo el avance sobre la computación cuántica era solamente a nivel teórico puesto que no había medios para para la creación de ordenadores cuánticos. Sin embargo, actualmente se están poniendo en funcionamiento los primeros ordenadores cuánticos, teniendo por fin la oportunidad de poner en práctica todos esas resultados anteriormente obtenidos. En España, ya hay un funcionamiento dos ordenadores cuánticos en Barcelona. Uno de ellos es parte de una red de ordenadores cuánticos formada por una red de ordenadores cuánticos situados por toda Europa, principalmente para finalidades de I+D [12].

También es importante considerar los factores económicos que envuelven a la computación cuántica. Pese a que ya haya empresas que han sacado a la venta ordenadores cuánticos para nivel comercial, su coste de fabricación sigue siendo muy elevado para su uso masificado en la población[11].

También es importante comentar los problemas de ser un paradigma de computación completamente diferente al actual. El cambio inmediato a una nueva tecnología nunca es inmediato, más aún en algo tan extendido y generalizado como los ordenadores y demás sistemas informáticos; además de que por comodidad para el usuario siempre se tiene tiene que garantizar cierta retrocompatibilidad del sistema moderno con el anterior.

1.3.2. Entorno específico: Xanadu

Xanadu Quatum Tecnologies es una empresa de computación cuántica (tanto a nivel hardware como software).

Por la parte hardware, *Xanadu* desarrolla *photonic quatum computers* accesibles a través de la nube, fomentando el desarrollo de software para ordenadores cuánticos de otras empresas tecnológicas. Este servicio permite una mayor divulgación y desarrollo de este nuevo paradigma que, pese a lo que se ha comentado, la fabricación de su hardware sigue siendo excesivamente costosa.

A nivel software, es la empresa desarrolladora de la librería de `PennyLane`, no solo utilizada para la implementación de circuitos cuánticos (como es el caso de este proyecto) sino también para poder desarrollar sistemas de Inteligencia Artificial Cuántica (*Quantum Machine Learning*). Se darán más detalles sobre el uso específico de esta librería en este proyecto en la sección 6.1.3.

Capítulo 2

Conceptos teóricos relevantes

La computación cuántica es un nuevo paradigma de computación en pleno desarrollo actualmente, comenzado en estos últimos años. Sin embargo, este nuevo paradigma todavía sigue siendo demasiado desconocido, ya sea por puro desconocimiento, o por lo abrumado que puede ser adentrarse en un nuevo paradigma, sobre todo tan diferente y complejo como es la computación cuántica. Este proyecto quiere servir como punto de inicio para las programadores interesados en aprender sobre este paradigma, tomando como ejemplo y motivación el protocolo BB84, un sistema de generación de claves compartidas que utiliza herramientas básicas de la computación cuántica para resolver una de las mayores vulnerabilidades que tienen los métodos de cifrado actuales.

El protocolo BB84 es un protocolo de generación de claves compartidas sobre un canal cuántico público. Para comprender lo que significa ello, hay que explicar primero lo que es la computación cuántica y para qué son necesarias las claves compartidas en la seguridad.

2.1. La computación cuántica

La computación cuántica es un paradigma de computación alternativo al tradicional, basado en la modificación y transmisión de bits. En la computación cuántica, el papel de los bits lo cumplen los **qubits**. A diferencia de los bits, que solo pueden tomar los valores 0 y 1, los qubits pueden tomar cualquier valor de la forma $\phi = a|0\rangle + b|1\rangle$, con $a, b \in \mathbb{C}$ que cumplan $a^2 + b^2 = 1$. Dentro de todos los valores, destacan los valores asociados a 0 y 1 de la computación tradicional, $|0\rangle$ ($a = 1$ y $b = 0$) y $|1\rangle$ ($a = 0$ y $b = 1$), denominados los estados/valores estáticos. Son llamados así ya que, si el qubit es medido, su valor pasa a ser uno de ellos, siendo a^2 y b^2 las probabilidades en las que se mida en cada estado estático respectivo.

Esto implica que, a no ser el qubit tenga un estado estático, es imposible predecir el valor que tendrá (pero sí intuir en base a los valores de a y b).

El hecho de un qubit pueda tener infinitos estados, hace que a la hora de construir circuitos cuánticos haya infinita variedad en las puertas lógicas que existen, además de que, mientras que no se mida el valor, se trabajará en un espacio más grande (se utiliza

álgebra compleja en vez de binaria). En cuanto a las puertas más importantes son las rotacionales (entre las que destacan las puertas de Pauli), y la puerta de Hadamard.

Las puertas lógicas tiene propiedades matemáticas muy interesantes, debido a que sus representaciones matriciales cumplen y tiene que cumplir la propiedad de que son unitarias (si su matriz inversa es su matriz hermítica[1]). En concreto, la puerta Hadamard cumple que la inversa de su matriz es ella misma. Este resultado se utilizará en protocolo BB84, ya que da una manera de cambiar el estado del qubit transmitido y de recuperarlo simétrico.

Otro resultado en el que se sostiene el funcionamiento del protocolo BB84 es en el *No Clonning Theorem* (Teorema de la No-Clonación), que determina que es imposible clonar el valor de un qubit a otro dado, a diferencia de lo que sucede en la computación tradicional.

Además de en el campo de la seguridad, donde el protocolo BB84 es un claro ejemplo de su aplicación, otra rama de la informática que la computación cuántica pone en jaque es en el aprendizaje automático. En concreto, la efectividad de los algoritmos de elección de la informática tradicional frente a los posibles algoritmos cuánticos. Un ejemplo de ello es el problema de Deutsch-Jotza[14, p.33-36].

El objetivo es construir un algoritmo que determine si una función $f(x_1, x_2, \ldots, x_n) \longrightarrow \{0, 1\}$ es constante (siempre da 0 ó 1) o es balanceada. Para la computación tradicional, el algoritmo se necesita de 2^{n-1} iteraciones de la función f para resolverlo (es decir tiempo exponencial) (teniendo un coste computacional enorme). Mientras tanto, en computación cuántica el algoritmo de Deutsch-Jozsa resuelve el problema mediante una única aplicación de f (luego tiempo constante).

2.2. El problema de la generación de la clave privada

Al ser el BB84 un protocolo de seguridad, también es importante hablar de entorno en el que se aplica: la generación de claves privadas en un canal público. Supongamos que Alice y Bob quieren establecer una comunicación privada dentro de una red pública (por ejemplo, a través de internet). Ambos en un principio se encuentran sobre un canal público, por lo que el primer paso debe ser que Alice localice a Bob dentro de la red. Tras ello, deben de cifrar la información, para que Bob sea el único capaz de saber el contenido de los mensajes. Entre los algoritmos de cifrado, existen varios que funcionan a través de una clave privada inicial, como por ejemplo el cifrado AES (*Advanced Encryption Standard*). De esta manera, Alice y Bob deben obtener la misma clave privada para que puedan enviarse información entre ellos sin que nadie (siempre que no sepa la clave) pueda conseguir la información.

[1]Extensión de matriz transpuesta para trabajar en el plano complejo

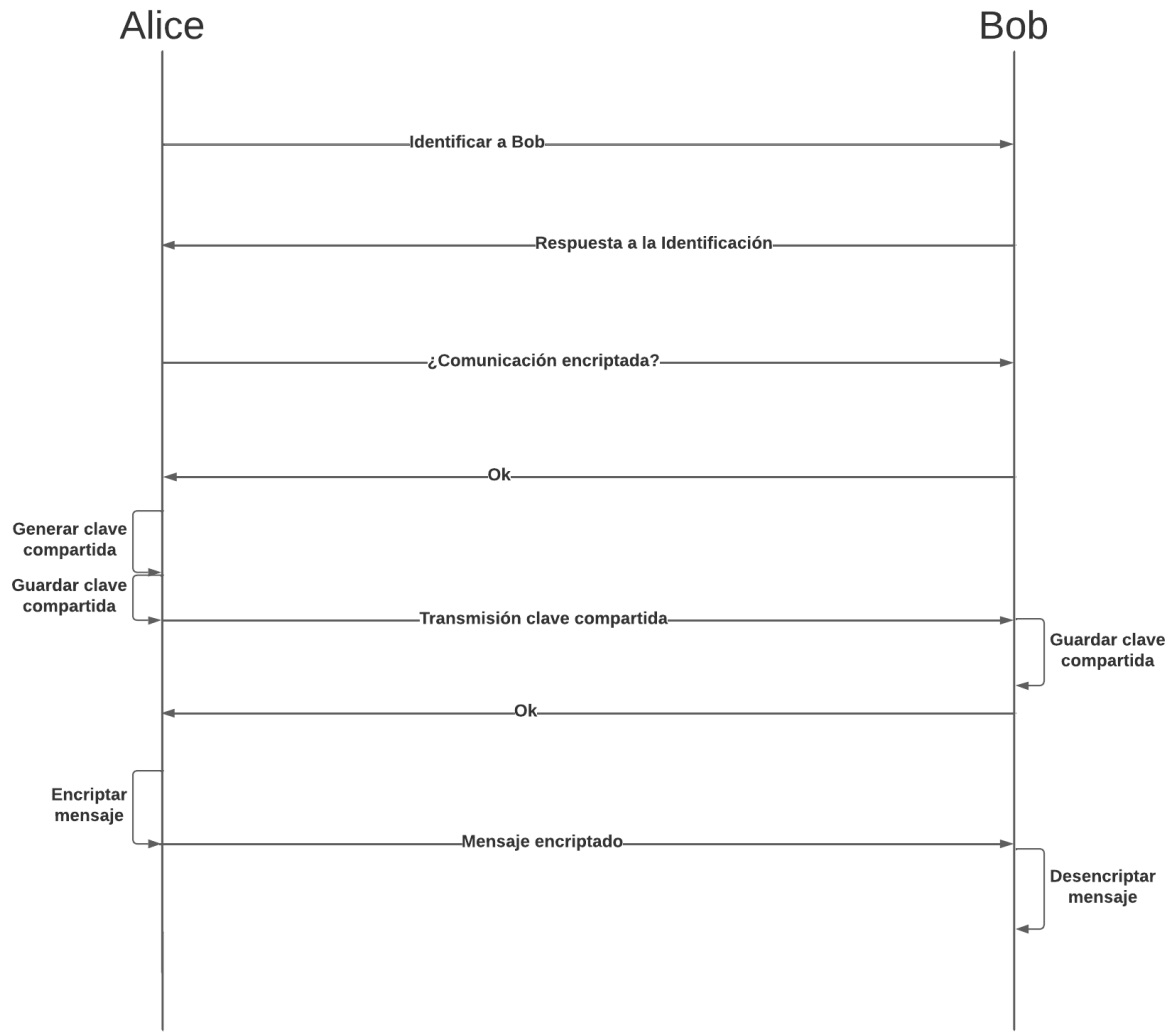


Figura 2.1: Generación de un ambiente cifrado en un canal público.

El problema se encuentra en que la única manera que tienen de comunicarse en un principio es de manera pública, lo que implica que una persona externa que se encuentre a su vez conectada a la red, Eve, puede conseguir la clave de cifrado, y con ello ver el contenido de los mensajes los mensajes del mismo modo que Bob. Esto es inevitable, puesto que cualquier cifrado que se pone sobre la transmisión de la clave también tiene que ser conocida por Eve, ya que Alice tiene que poder comunicarse con Eve y Bob de igual manera. A este ataque se le denomina **Eavesdropping**.

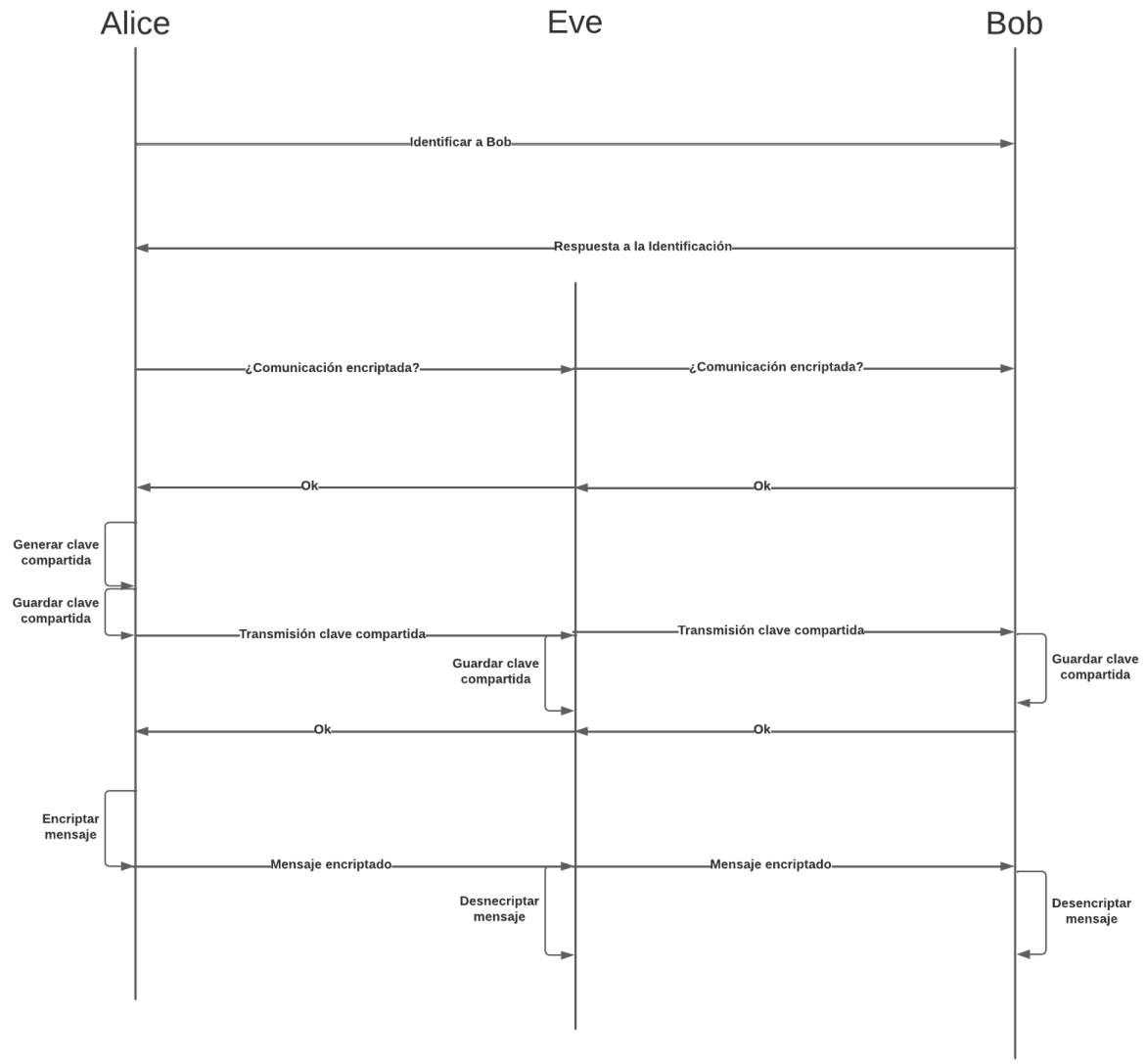


Figura 2.2: Esquema del ataque de eavesdropping

El protocolo BB84 es un protocolo para la obtención de una clave compartida en un canal público, es decir, es un protocolo que permite que Alice y Bob obtengan una misma clave de manera de que Eve no (siempre) consiga la clave. Aunque no sea 100 % fiable, es una gran mejora respecto al caso de trabajar en un canal tradicional, donde no puede existir ninguna protección debido a las restricciones del propio paradigma. No obstante, con esta estrategia Eve lo que hace en cambio es llevar a cabo un ataque de DoS, puesto que puede hacer que la comunicación entre Alice y Bob sea imposible.

En el estudio se considera además el caso en el que el protocolo se aplica en un canal cuántico con ruido, donde el detectar a un intruso en el canal será más complicado debido al poder camuflarse con el propio ruido.

2.3. El protocolo BB84

El protocolo BB84 se basa en "cifrar" la clave privada a transmitir enviando los qubits con la clave en una de dos posibles polarizaciones[2], y el receptor midiendo el qubit a su vez en una de esas dos posibles polarizaciones. Esto es posible ya que al medir el qubit en la misma polarización que la enviada siempre da el mismo valor, mientras que en el caso contrario no.

Tras insertar el bit de la clave original al circuito (C_A), Alice lo polariza según una polarización marcada por el valor del bit de P_A. Tras ello, envía el qubit; Bob recibe qubit, y lo mide en la polarización marcada por P_B, obteniendo así su bit para generar la clabe C_B. Si Alice y Bob no han elegido la misma polarización, el valor medido por Bob es aleatorio, pero si $P_A = P_B$, entonces $C_A = C_B$, pudiendo entonces generar ALice y Bob la misma clave si se quedan con los bits de cuando las polarizaciones hayan coincidido. El circuito cuántico asociado al protocolo BB84 es el siguiente:

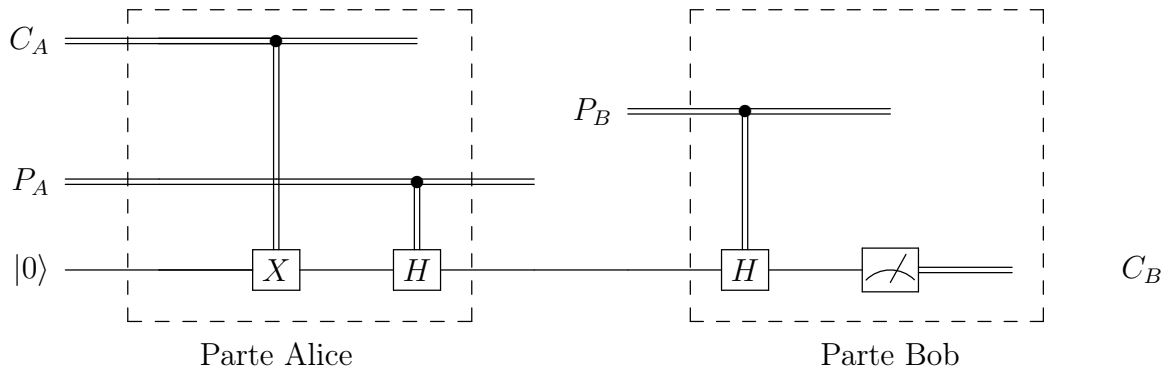


Figura 2.3: Circuito del Protocolo BB84

Sin embargo, como se ha comentado anteriormente, ya que las polarizaciones que generan Alice y Bob son independientes. En concreto, solo la mitad de las veces coincidirán, aproximadamente. Es por ello, que habría que enviar más qubits de los necesarios para formar la clave (se suele considerar el doble).

Por ahora, Alice y Bob no pueden generar una clave privada, puesto que no saben las polarizaciones que ha escogido el otro. Es por ello que, una vez enviado los qubits, comparten sus polarizaciones para descartar los bits "contaminados".

De esta manera, si hay un atacante Eve escuchando en el canal, no sabrá en que polarización medir los qubit para obtener el valor correcto a la hora de la transmisión de los qubits, y cuando se hagan públicas las polarizaciones elegidas, ya habrá medido incorrectamente varios qubits, haciendo que la clave privada que genere Bob sea distinta a la que genere Alice.

[2]En mecánica cuántica, dirección de propagación de la radiación electromagnética. En nuestro contexto, indica los estados estáticos.

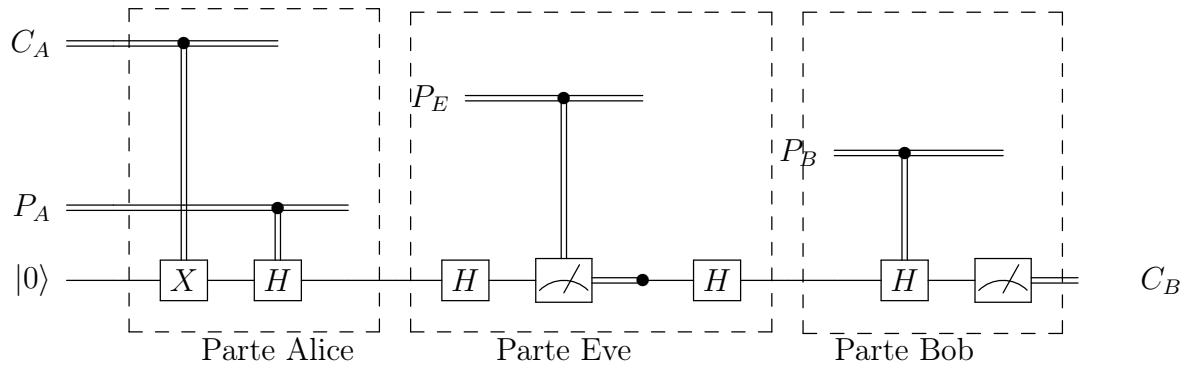


Figura 2.4: Ataque de Eve al protocolo BB84

No obstante, no se ha comprobado que la clave generada por ambos sea la correcta, y comprobándola transmitiéndola por el canal haría inútil la aplicación del protocolo. Es por ello que el protocolo necesita que se realice un test de integridad: además de tener que generarse bits necesarios para generar la clave, también será necesario generar bits que compartir públicamente para comprobar que efectivamente no ha habido problemas en la transmisión.

Sin embargo, sabiendo el funcionamiento del protocolo Eve puede actuar en consecuencia. La estrategia más sencilla que puede considerar es elegir una polarización al azar para medir el qubit, guardar el bit medido, y enviarlo el qubit polarizado con la polarización medida. De esta manera Eve acertará 67.5 % , y el 75 % de las veces no será detectada (aunque algunas no obtendrá el bit correctamente.)

- Siempre que Alice y Eve elijan ambas la polarización A (25 %), Eve obtendrá el valor, y además no lo cambiará, por lo que no será detectada.

- Lo mismo sucede cuando eligen B de polarización (25 %) (polariza en B, obtiene el valor correcto del bit, vuelve a lanzar el qubit polarizado en B, enviado a Bob el mismo estado que ella ha recibido).

- El resto de casos (Eve y Alice eligen distinta base) implica que Eve polariza en la base equivocada. Es decir, Eve medirá un estado cuántico no básico: $|+\rangle$ o $|-\rangle$, cambiando la polarización del bit recibido (si Alice le envía el qubit en la base A, Eve le enviará a Bob en la base B o viceversa). Esto implica que Bob medirá también la base equivocada, obteniendo por tanto el vlor incorrecto la mitad de las veces (25 % del total). Del mismo modo, dentro de ese 25 % de las veces Eve solamente conseguirá el valor correcto la mitad de ellas, cuando la polarización le haya dado el valor correcto de manera aleatoria.

Parece entonces que el protocolo BB84 no es tan efectivo, cuando hay un método que tiene un 67.5 % de éxito de obtener el bit sin ser detectado.

Sin embargo, ese porcentaje de éxito es para cada bit obtenido. Es decir, el método tiene un éxito del 75 % cuando es usado para concertar una clave de un solo valor. Este nunca sería un caso real, puesto que bajo cualquier circunstancia una clave determinada por un único bit es demasiado vulnerable (sin ninguna estrategia, hay 50 % de conseguir la clave). Un caso más real sería por ejemplo usarla para obtener una clave de 16 bits (como se hará más adelante). En ese caso, para no ser detectada tiene que acertar cada una de las 16 veces. La probabilidad es de $0{,}75^{16} \approx 0,01 \Rightarrow 1\,\%$. Esa es su probabilidad de no ser detectada: la probabilidad de que el ataque sea un éxito es mucho menor, del $0{,}625^{16} \approx 0,0005 \Rightarrow 0,05\,\%$.

Esta será la metodología que tomará Eve en el estudio, aunque eso no quiere decir que sea la más eficiente (hay una que en el mismo caso de antes consigue una probabilidad de éxito del 0,65 %).

Además de esa (aunque pequeña) tasa de éxito que hay frente al ataque de Eavesdropping, el test de integridad ofrece otra manera de atacar la comunicación entre Alice y Bob: Esto, sin embargo, crea la posibilidad a Eve de realizar un ataque de DoS, modificando los bits de integridad para que nunca coincidan.

Una explicación más detallada de su funcionamiento se puede encontrar en los propios cuadernos del proyecto, así como en el libro de Nielsen[14, p. 587], o en el artículo de H. Bennett [2].

2.4. Mejora del protocolo

La mejora del protocolo sugerida consiste en comparar el caso a analizar con los datos obtenidos en un conjunto de entrenamiento anterior, de manera de que si los valores del caso a analizar no es suficientemente similar a valores obtenidos previamente, es descartado. Este mecanismo (pero más elaborado) es en cierta manera el utilizado en el aprendizaje supervisado y en inteligencia artificial para crear mecanismos de categorización, pero modificado puesto que nuestro objetivo es categorizar, sino descartar[10].

En concreto, se compara el número de intentos que lleva para intentar pasar el test de integridad, así como cuántos de esos intentos han tenido un número de fallos significante (que, por lo estudiado en los cuadernos se ha considerado que ese número sea 5). En los cuadernos se justifica que ambos atributos son mayores en caso en que el canal esté siendo atacado, lo que implica que una forma de detectar si el caso a estudiar es sospecho de no sea debido al ruido, es que uno de esos valores sea suficientemente elevados; en caso contrario, se le permite seguir intentándolo (ya que debido al ruido es muy probable que haya que repetir el test de integridad varias veces).

La siguiente cuestión es qué valores considerar "suficientemete altos". En nuestro caso, esos valores será dado por la media de los valores medios de cada tipo. Una mejora sería considerar un algoritmo de agrupamiento -*clustering*- más optimizado; o simplemente optimizar los pesos de los parámetros/ponderaciones a través de algún algoritmo propio del aprendizaje supervisado, como la validación cruzada.

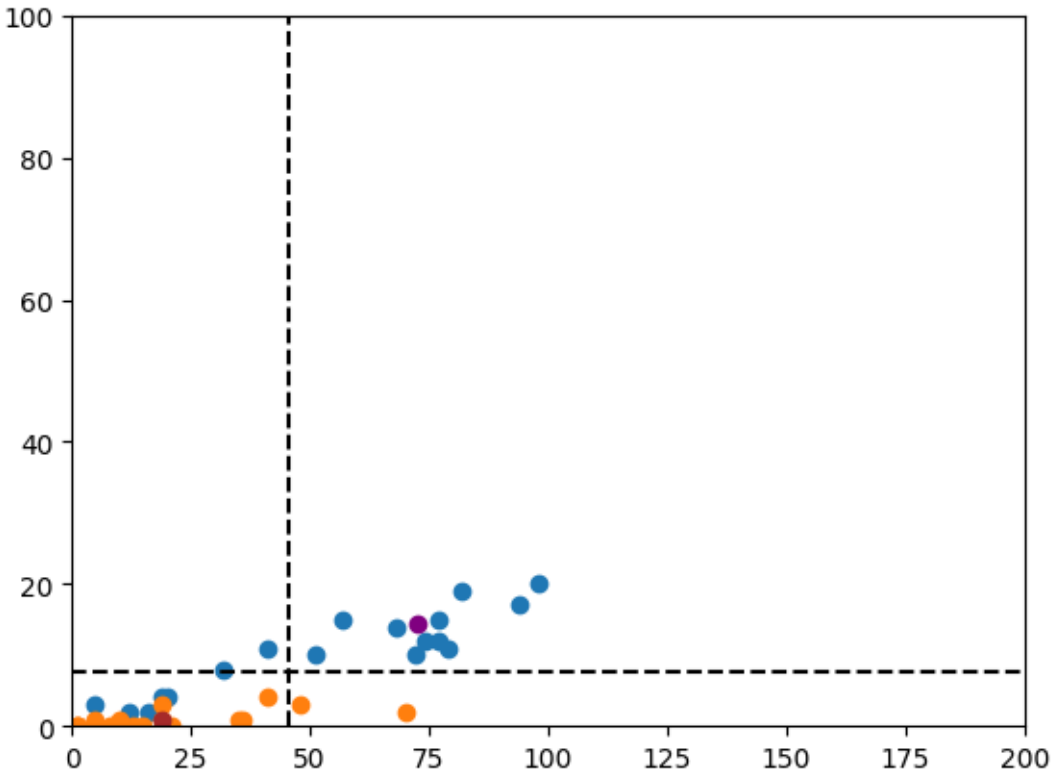


Figura 2.5: Los valores cercanos a la media de los valores del caso con ruido (punto marrón), serán considerados ruido. El resto, se tratarán como posibles ataques de Eve.

Capítulo 3

Metodología

Pese a no ser un producto software habitual, la metodología con la que se ha llevado a cabo el proyecto ha sido el **Proceso Unificado**. El hecho de que el producto no sea una aplicación software no implica que las técnicas de ingeniería aprendidas en la carrera no tengan utilidad en un proyecto de estas características.

En esta sección también se comentarán las herramientas software utilizadas, tanto para la realización del proyecto como de la propia memoria, así como un esquema que la arquitectura de trabajo decidida para el proyecto, además de la justificación de la programación estructurada como paradigma de programación empleado.

Por último, también se indicará ciertas siglas y expresiones utilizadas en el cuaderno.

3.1. Proceso de desarrollo

Siguiendo como referencia las fases del ciclo de vida del Proceso Unificado, el desarrollo del proyecto se puede dividir en tres fases perfectamente diferenciadas (no se considera la fase de Transición):

Fase I. Esta primera etapa consiste en el estudio personal sobre el tema que tratará en el proyecto: la computación cuántica y en concreto el funcionamiento del protocolo BB84. No hay mucho más que añadir a esta fase, ya que simplemente toma el papel de la formación del personal. También durante esta fase se comenzará a leer la documentación de la librería PennyLane.

Fase II. Esta etapa consiste en la creación de una simulación software del protocolo BB84 (tanto con como sin ruido en el canal) en scripts de *Python*. El objetivo de esta etapa es que, a partir de los programas en *Python* creados en esta fase, la creación de los cuadernos *Jupyter* posteriores resulte más sencilla. Se podría considerar la fase de Elaboración del PU. Esta fase se puede subdividir a su vez en tres iteraciones, donde en cada una se trabajará en la implementación del protocolo en un entorno diferente:

- En un canal seguro.

- En un canal con ruido.
- En un canal vulnerable.

Como se verá posteriormente en los apartados de planificación, en un principio iba a existir un cuarto cuaderno donde se analizaba más detenidamente el protocolo desde la situación de Eve. Por problemas de planificación, al final se decanto por no añadir dicho cuaderno, no afectando sustancialmente al los objetivos finales del proyecto.

Fase III. Durante esta etapa se creará el producto propiamente dicho: se redactarán los diversos cuadernos que componen el estudio (la fase de implementación del PU). La mayor parte de trabajo a realizar en esta parte se encontrará, en el último cuaderno, donde haya que razonar y justificar (mediante herramientas estadísticas) las mejoras realizadas al protocolo.

Como se verá próximamente, la duración de esta etapa acabará siendo más larga de lo esperada. Esto es debido a que también hubo que realizar actividades propias de la fase de Elaboración para la redacción de los cuadernos que no se habían considerado inicialmente.

Durante el desarrollo en su conjunto se abogará por un modelo evolutivo, donde se evaluará el progreso en reuniones bisemanales con el tutor para cada versión. Puesto que es un proyecto de investigación, la evaluación de las iteraciones es sencilla y los requisitos del proyecto están más o menos claros, por lo que la parte de desarrollo (no solo en la parte de programación) y el análisis son la más importantes dentro del ciclo de versiones.

En cuanto al paradigma de programación utilizado, se trabajará en programación estructurada. La ventaja de utilizar este paradigma de programación frente a uno más relacional como la programación orientada a objetos es debido a que el código deberá estar inscrito dentro de los cuadernos *Jupyter*, implicando tener que exponer los resultados intermedios de las funciones, que resulta más sencillo en este paradigma. El problema que conlleva el uso de este paradigma de programación es la falta de organización en comparación a otros. Sin embargo, gracias al hecho de que *Python* sea un lenguaje multiparadigma (como se comentará más adelante), hace que el cambio del código a uno u otro sistema (para, por ejemplo, su mejor utilización como módulo), sea relativamente sencilla.

Por otra parte, también tiene coherencia con el hecho de cómo se usará la librería `PennyLane`, ya que en ella los circuitos cuánticos se simulan como funciones, haciendo que el uso de la librería sea más fluido y natural.

Como se verá más adelante, esto no implica que no haya ninguna clase. La clase **Person** simulará cada una de las personas que entran en acción a la hora de simular el protocolo (Alice, Bob, y, posteriormente, Eve)Sin embargo, su papel es actuar como una estructura de datos, que sirve para saber qué información sabe cada persona en cada momento. Además, se implementará una clase para realizar el cifrado AES, utilizado en los cuadernos (ver sección 8).

3.2. Herramientas de desarrollo utilizadas

En cuanto a las herramientas que se han utilizado durante el desarrollo caben destacar:

Visual Studio Code. Visual Studio Code es el editor de código fuente que suelo utilizar por sus altos niveles de estilización y adaptación a prácticamente cualquier lenguaje que se vaya a utilizar, gracias a su funcionamiento a base de paquetes. Además, también tiene la posibilidad de vincularlo a un repositorio *git*, y suele ofrecer herramientas para debuggear y testear.

LaTeX. LaTeX es un procesador de textos, que sirve para crear documentos con cierta complejidad tipográfica (en concreto,o está centrado en escribir fórmulas matemáticas, para lo que tiene el *math mode*) a la vez de facilitar su estructuración. Aparte de su uso indispensable en la creación de la memoria del proyecto, también cabe mencionar **MathJax**, que como se comentó anteriormente, es una librería de javascript que implementa el modo matemático de LaTeX, y que *Jupyter* utiliza.

Otra herramienta relacionada con LaTeX es **Overleaf**, que es el un editor en línea de éste que ofrece bastantes facilidades a la hora de la edición y organización del *.tex* que genera el documento, con la ventaja (o desventaja, depende de cómo se mire) de que sea una aplicación en línea y que no se trabaje en local.

StarUML StarUML es un programa que permite la organización y realización de diagramas UML. Será la utilizada para crear los diagramas destinados al análisis del proyecto. En su mayoría serán diagramas de secuencia o de actividad, debido a que lo más importante del proyecto son los los propios algoritmos.

Lucidchart Lucidchart es una aplicación web que permite el diseño de diagramas (y esquemas) de manera fácil y sencilla. A diferencia de StarUML, es menos técnica y avanzada, pero en cambio es más intuitivo. Dependiendo del objetivo del diagrama se utilizará una u otra aplicación.

GitHub Para realizar el control de versiones del proyecto, pudiendo saber qué se ha añadido en cada versión y cuándo, se ha utilizado el repositorio web de GitHub. Al ser este proyecto un proyecto individual que además es un estudio teórico, no hace falta llevar un control exhaustivo de versiones ni el proyecto se compondrá de una gran cantidad de archivos, por lo que la actualizaciones (*push*) al repositorio se han podido realizar de manera "manual", subiendo los archivos nuevos.

Milanote Milanote es una herramienta de organización en línea muy general y sencilla. Básicamente sirve como una pizarra digital donde poder situar desde lista de tareas hasta esquemas de trabajo. Es muy versátil ayuda mucho a la organización de proyectos, en especial a aquellos desarrollados bajo una metodología ágil.

OpenProj OpenProj es una herramienta gratuita de gestión que intenta simular a MS Project. Aunque tuviera un diseño bastante similar, su uso resulta muy tosco muy

poco pulido, siendo definitivamente MS Project superior en todos los sentidos. Sin embargo, pocas herramientas de gestión gratuitas hay disponibles actualmente, siendo OpenProj una de las mejores dentro de esa categoría (siempre y cuando se desee realiza algún diagrama de Gantt o gestión de recursos, sino cualquier aplicación tipo Excel es mucho más cómoda de utilizar).

Microsoft Excel La herramienta para el manejo de hojas de cálculo por excelencia. Toda las tablas (y sus consecuentes operaciones) para por las estimaciones han sido realizadas con ella por la facilidad que ofrece para desplegar datos y operar con ellos.

Lenguajes de programación. Se explicarán detenidamente en el capítulo 6.

3.3. Arquitectura

Este proyecto no necesita de arquitectura física, ya que la única necesaria es la propia de los cuadernos *Jupyter*, explicada en la sección 6.2.

En cuanto al patrón de arquitectura del proyecto, las propias funciones que conforman el cuaderno se encontrarán definidas en el mismo, puesto que se quiere que el usuario comprenda su construcción. Dicho esto, las funciones definidas en los cuadernos también se encuentran en unos scripts de *Python* separados, que tiene dos objetivos:

- Poder usar funciones definidas en un cuaderno o en otro sin tener que volverlas a definir.
- Poder utilizar las funciones de manera independiente de los cuadernos.

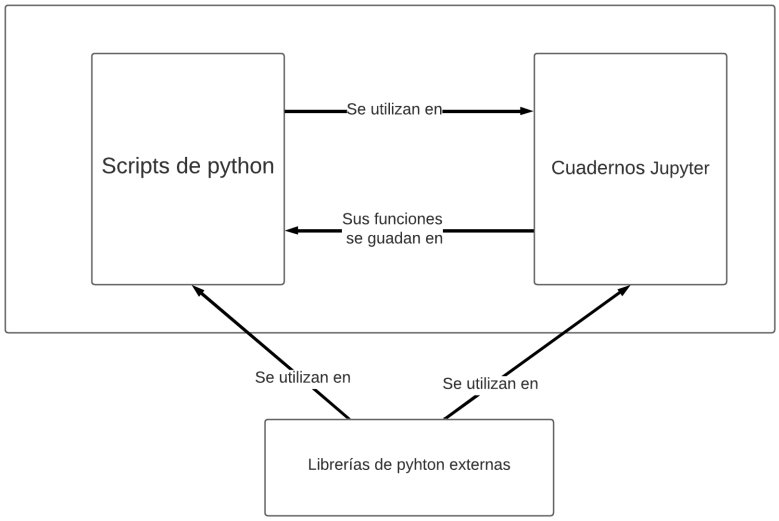


Figura 3.1: Patrón de arquitectura

3.4. Definición de siglas y abreviaturas

Además de varias siglas utilizadas a lo largo de la Memoria, aquí también se situarán varias expresiones o palabras que, a lo largo de la Memoria, se utilizarán bajo la jerga de la seguridad informática o de los propios cuadernos, teniendo por tanto que explicar su significado:

- **PU.** Proceso unificado.
- **MitM.** Man in the Middle.
- **DoS.** Denial of Service.
- **NCT.** Non-Cloning Theorem.
- **Alice.** Nombre con el que se suele denominar a aquel que juega el papel del emisor en los escenarios de distribución de claves.
- **Bob.** Nombre con el que se suele denominar a aquel que juega el papel del receptor en los escenarios de distribución de claves.
- **Eve.** Nombre con el que se suele denominar a aquel que juega el papel del atacante en los escenarios de distribución de claves.
- **Clave.** Bits generados por Alice a partir de los que se sacará la clave privada, y bits de Bob obtenidos a partir del circuito cuántico del protocolo BB84.
- **Polarizaciones.** Normalmente se referirá a polaridad/polarizaciones, no la polaridad en sí sino a la cadena de bits por la que Alice/Bob decidirán que polarización utilizar al qubit asociado transmitido.
- **Clave Final.** Bits de Alice y/o Bob de la cadena tras haber descartado los bits mal polarizados. Está compuesta por los Bits de integridad y la clave Privada.
- **Bits de integridad.** Se refiere a los bits que considerarán Alice y/o Bob para realizar el test de integridad del protocolo.
- **clave Privada.** Cadena utilizada en un cifrado que solo debe ser conocida por Alice o Bob. En este caso, es la cadena resultante de aplicar el protocolo BB84, y la de Alice y Bob coincide (es simétrica).

Puertas

Dentro de las siglas utilizadas durante el cuaderno, cabe destacar la simbología utilizada para referirse a las distintas puestas lógicas cuánticas.

- H. Puerta Hadamard
- X, Y, Z. Puertas de Pauli

- $R_X(\varphi)$, $R_Y(\varphi)$, $R_Z(\varphi)$. Puerta rotacional sobre el eje X, Y o Z respectivamente con ángulo φ
- C_P. Puerta controladora sobre la puerta P.

Capítulo 4

Planificación

En este capítulo se abordarán las cuestiones relativas a la planificación del trabajo, como son el diagrama de Gantt y la creación de los presupuestos, que se comparará con el obtenido. Para ello, se comenzará realizando una estimación temporal del tiempo que consumirán las tareas que componen el proyecto. Si bien es cierto que la mayor parte de las técnicas estudiadas, tales como el método *COCOMO II*, o los puntos de función son demasiado específicas para una aplicación software como para poder ser eficaces en este caso, técnicas y metodologías más generales como los diagramas de Gantt o el método PERT de estimación resultan sorprendentemente aplicables en este proyecto.

Por otra parte, debido a diversas causas que se comentarán más adelante, la estimación realizada del proyecto no ha sido muy acertada, y ha sufrido un severo retraso respecto al plan original, viéndose reflejado en el sobrecoste del presupuesto final, así como en el hecho de tener que descartar cierta parte del proyecto.

4.1. Estimación del esfuerzo

La estimación del esfuerzo a partir de los métodos tradicionales, tales como la estimación puntos de Caso de Uso o puntos de función, no tiene sentido para este proyecto, puesto que se basa en las funcionalidad para el usuario y este es un estudio teórico y no una aplicación software. Por tanto, pese haber considerado un proceso de desarrollo, a la hora de las estimaciones de esfuerzo se tendrán que considerar métodos alternativos.

Por ello, se va a considerar la estimación PERT (la estimación por tres valores). Para cada tarea que se realizará en el proyecto, se considerarán la **duración más optimista**, la **duración más probable** y la **duración más pesimista** para completarla. De esta manera se puede tratar la duración de la tarea como la variable con distribución β que pasa por eso tres puntos, obteniendo:

$$T_{esperado} = \frac{T_{Opt} + 4\,T_{Prob} + T_{Pes}}{6}$$

La estimación obtenida en la tabla A.1 da que el proyecto se terminará en unos 116 días de trabajo (de 4 horas). Es decir, sin considerar días festivos, **5 meses y medio**. Si

se considera el peor de los casos, tomando la predicción más pesimista siempre, el proyecto terminaría en 171 días de trabajo, lo que equivale a poco más de 8 meses. Como también se muestra en la tabla, el tiempo que se he tardó finalmente en realizar el proyecto fue de 150 días de trabajo (**7 meses y una semana** aproximadamente). Las mayores diferencias de tiempo han sido mayoritariamente por 3 razones:

- La implementación de la clase Persona y modelización de los canales no cuánticos, ya que surgieron ciertas dificultades al no haber entendido partes de la documentación correctamente.
- El retraso mencionado anteriormente además produjo que se decidiese descartar la sección sobre la perspectiva del Eavesdropper.
- Anaconda comenzó a dar problemas, por lo que se optó por cambiar y utilizar *Python* de manera independiente de Anaconda. El tiempo que figura en la tabla es el que se tardó en desinstalar Anaconda completamente más la instalación de *Python* y todas las librerías que se han utilizado (incluidas *Jupyter* y *PennyLane*, por ejemplo).

Sin embargo, pese a estas dificultades que hicieron que el proyecto durase más de lo estimado, el tiempo obtenido sigue siendo menor que la estimación más pesimista, estando lo obtenido dentro del margen estipulado.

4.2. Planificación temporal

En la planificación de tareas del proyecto se mostrará el diagrama de Gantt diseñado al inicio del proyecto, y el obtenido al finalizar el proyecto.

Aunque se haya realizado la estimación por el método de tres valores de PERT, no se mostrará el diagrama temporal de PERT, pues carece de importancia al haber sido un trabajo individual. Además, en cambio sí que se realizarán un diagrama de Gantt mucho más ilustrativo y completo (sobre todo teniendo en cuenta que ha sido un proyecto individual y que por tanto es imposible que se trabaje en varias tareas paralelamente).

4.2.1. Roles en el proyecto

Recordemos que el proyecto se dividía en tres fases: fase de aprendizaje, fase de elaboración y fase de redacción. La fase de aprendizaje, dedicada al estudio teórico necesario para la realización del proyecto, no se tiene en cuenta en la estimación temporal pese a que siempre es importante tenerla en cuenta, al ser formación necesaria para el proyecto. Esto es debido a que fue realizada anteriormente a estas estimaciones .

Pese a que el proyecto a sido realizado por solo una persona, también es de utilidad catalogar qué tareas realizará cada rol (cuando una tarea sea necesaria de varios roles, se considerará que el tiempo se reparte equitativamente por simplificación). Los roles que se considerarán son:

- **Jefe de Proyecto**. Rol de la persona encargada de la documentación y organización del proyecto.

- **Programador**. Rol de la persona encargada de la creación de los programas necesarios para el estudio.

- **Investigador**. Rol de la persona con conocimientos sobre computación cuántica y matemáticas. Es quien redactará los cuadernos.

En este caso, los tres roles han sido ejercidos por la misma persona, aunque en un principio no tiene por qué (dicho esto, es bastante recomendable que el rol de Programador y de Investigador lo haga la misma persona, debido a la gran comunicación y entendimiento que debe de haber entre ambas partes). Estos datos servirán principalmente para estimar y calcular los recursos humanos utilizados en el proyecto.

En las tablas A.2 y A.3 se vislumbra más claramente cómo han afectado al desarrollo los problemas comentados anteriormente. Haciendo la distinción entre los diferentes roles, se ve que el mayor aumento de horas ha surgido en tareas de programación. También es relevante observar la excesiva escasez de horas empleadas para tareas de "Jefe de proyecto". El tiempo dedicado a tareas de este rol ha sido insuficiente, teniendo como consecuencia que los percances ocurrido hayan necesitado de tanto tiempo para salvaguardarse (por ejemplo, un retraso de una semana y media de trabajo en terminar la implementación de la clase `Person`).

4.2.2. Diagrama de Gantt

Puesto que el proyecto ha sido realizado por una sola persona, el diagrama de Gantt del proyecto no ofrece mucha información adicional, al no necesitar de una optimización de la ruta crítica del proyecto (no se pueden realizar tareas de forma paralela ni reasignaciones a recursos con el mismo rol). Así, pues, a partir de los tiempos de las tablas A.2 y A.3 se consiguen los siguientes diagramas de Gantt

Como se puede observar, varias tareas o se han sustituido por otras o han desaparecido. Eso ha sido producto de cambios intermedios que ha habido, admitiendo que varios de los cuales han sido por un mal planteamiento de los requisitos iniciales. Además, también se puede ver que la implementación de Eve de manera más detallada se desechó, debido a la cierta complejidad que supondría (se detallará más detenidamente en el apartado de implementación), además de ir más relajado a los plazos de entrega. Otra desviación interesante al plan original es el hecho de que finalmente se optó por comenzar la redacción de los cuadernos antes de terminar la simulación de los circuitos. Esto no supone ningún problema, puesto que los cuadernos que necesitaban de esa implementación fueron redactados posteriormente.

4.3. Presupuesto económico

Para la estimación del presupuesto se tendrán en cuenta las herramientas utilizadas (hardware y software, con los factores de impacto que correspondan al duración del proyecto), junto con los recursos humanos necesarios, según la planificación de tareas, y el tipo de rol (analista, programador, etc) correspondiente a cada tarea.

4.3.1. Hardware y software

Para realizar los cálculos, se considerará que la vida útil de ambos elementos hardware es de 5 años. Además, por simplificación, se considerará que tanto las herramientas software y hardware únicamente se han utilizado para realización de este proyecto durante ese tiempo, y se seguirán usando durante el resto de su licencia/vida útil (es decir, su gasto para el proyecto será proporcional al tiempo estimado del mismo). Del mismo modo, no se considerarán la contratación de servicios fundamentales para el trabajo, tales como la electricidad o el internet.

En cuanto a los recursos software utilizados, se ha usado la versión gratuita de todos ellos. Sin embargo, debido a que para la mayoría de ellos la versión gratuita está destinada para su uso no comercial, para este presupuesto se tendrán en consideración la versión de pago/prémium de ellas (suponiendo el caso de que el estudio se haya hecho para con una empresa):

Recurso	Valor total	Tipo de pago	Coste estimado en el proyecto		Coste real	
			Tiempo de uso	Coste	Tiempo de uso	Coste
LucidChart	27,00 €	mensual	5,533730159	149,41 €	7,178571429	193,82 €
Microsoft Office (Excel)	300,00 €	por dispositivo	1	300,00 €	1	300,00 €
MS Project	34,00 €	mensual	5,533730159	188,15 €	7,178571429	244,07 €
Milanote	- €	-		- €		- €
Overleaf	179,00 €	anual	0,460833333	82,49 €	0,59825	107,09 €
Visual Studio	- €	-		- €		- €
GitHub	- €	-		- €		- €
				720,05 €		**844,98 €**

Cuadro 4.1: Presupuestos del software.

En cuanto a los recursos hardware utilizados, solo es necesario un ordenador de trabajo. Las especificaciones de este ordenador no son importantes, basta con que tenga suficiente potencia como para que no haya problemas de rendimiento al programar, y que preferiblemente sea un portátil para mayor movilidad del entorno de trabajo. Además, también se tendrán en cuenta periféricos esenciales, como un ratón.

Para estimar el coste del ordenador, se considerará un ordenador portátil de gama media, como el HP 15S-FQ5013NS Intel Core i5-1235U con 8GB de disco duro y una

tarjeta RAM 512GB SSD, que viene con el sistema operativo ya instalado. Su precio es de 700 € (es por ello que en el apartado software no se ha tenido en cuenta el costee del sistema operativo).

En cuanto al ratón, no hay mucha diferencia de precio entre distintas marcas y modelos, así que se considerará un precio de 15€para simplificar la cifras monetarias obtenidas.

Recurso	Valor total	Tiempo de vida	Coste estimado		Coste real	
			Tiempo de uso	Coste	Tiempo de uso	Coste
Ordenador portátil	700,00 €	5 años	0,461 años	64,52 €	0,598 años	83,76 €
Ratón	15,00 €	5 años	0,461 años	1,38 €	0,598 años	1,79 €
				65,90 €		85,55 €

Cuadro 4.2: Presupuestos del hardware.

4.3.2. Recursos humanos

Gracias a los cálculos realizados en la sección anterior, podemos entonces calcular el coste de los recursos humanos de nuestra estimación, así como del desarrollo real del proyecto. Para hacerlo verosímil, se considerarán los sueldos obtenidos en `indeed.com`, una página que recopila datos sobre los salarios de distintas profesiones[1]

- **Jefe de Proyecto**: 16€/hora.
- **Programador**. Se considerará el sueldo de programador junior: 10 €/hora .
- **Investigador**: 12 €/hora.

Sin embargo, el coste real para la empresa de la contratación a cada uno de los empleados es mayor, ya que hay que sumarle el coste de la Seguridad Social. Éste dependerá del contrato, pero para simplificar los cálculos y puesto que estamos solo analizando un caso hipotético, se supondrá un 30 % del salario. Los costes finales de los recursos humanos entonces serán:

- **Jefe de Proyecto**: 20.8 €/hora.

[1]Los verdaderos sueldos medios que figuran la página son:

- Jefe de Proyecto: 16,77€/hora.
- Programador: 10.76€/hora.
- Investigador: 11,66 €/hora.

Sin embargo, como el objetivo simplemente era asignar un salario realista a cada rol, se ha optado por considerar un entero cercano para facilitar las operaciones.

- **Programador**. Se considerará el sueldo de programador junior: 13 €/hora .
- **Investigador**: 15,6 €/hora.

Estos costes, junto con las estimaciones temporales obtenidas en las tablas A.2 (para los presupuestos) y A.3 (para el coste real), dan lugar a los siguientes coste en los recursos humanos del proyecto:

Recurso	Salario	Coste	Coste estimado		Coste real	
			trabajo (horas)	Coste	trabajo (horas)	Coste
Jefe de Proyecto	16,00 €	20,80 €	57,3 horas	1.192,26 €	59,3 horas	1.233,86 €
Prgramador	10,00 €	13,00 €	171,3 horas	2.226,64 €	272,4 horas	3.541,46 €
Investigador	12,00 €	15,60 €	253,6 horas	3.956,16 €	290,4 horas	4.530,55 €
				7.375,06 €		**9.305,87 €**

Cuadro 4.3: Presupuestos de los RRHH.

4.3.3. Presupuesto total

Juntando los presupuestos económicos obtenidos en las secciones anteriores, se obtiene el siguiente presupuesto total para el proyecto

	Estimación	Real	Sobrecoste
Recursos software	720,05 €	844,98 €	117%
Recursos hardware	65,90 €	85,55 €	130%
Recursos humanos	9.305,87 €	11.318,38 €	122%
TOTAL	**10.091,81 €**	**12.248,91 €**	121%

Cuadro 4.4: Presupuestos totales

Como se puede observar en la tabla anterior, tanto el presupuesto estimado como el obtenido son bastante elevados (unos 10,000 € y 12,000 € respectivamente) para lo

que es el proyecto. Esto es debido que el proyecto se ha estado realizando de manera relajada, sin considerar el tiempo (y, por tanto, el presupuesto) que se está necesitando para desarrollarlo. Esto se ha podido hacer puesto es que al ser un proyecto a nivel personal los gastos son meramente teóricos. Lo que sí que preocupante sin embargo es la gran diferencia del presupuesto económico estimado al real (de un aumento del 25 % del gasto), debido principalmente a los grandes retrasos que ha habido en el proyecto ya comentados anteriormente.

Capítulo 5

Conclusiones

Como se ha estado mencionado, la realización de este proyecto ha sido muy lenta, arrastrada tanto por un mal entendimiento inicial de los objetivos del proyecto, como por no haber utilizado correctamente las funciones de PennyLane en un inicio. Esto ha provocado que parte del proyecto inicial (análisis de los resultados obtenidos por Eve) se haya tenido que descartar, siendo una posible mejora del producto terminar dicha sección.

Además de su propósito divulgativo y didáctico, El producto final cumple su cometido de servir como material didáctico de introducción a la computación cuántica, no sólo a nivel teórico sino también como guía introductoria para la librería PennyLane. Sin embargo, puede considerarse que como estudio podría estar más completo, o haber tenido una mejor estructuración de los scripts auxiliares.

A nivel personal, el estudiar sobre este paradigma emergente ha servido para informarme sobre él personalmente, además de mejorar mi desenvoltura sobre *Python*, sobre todo de cara a la redacción de cuadernos *Jupyter* y el uso de la librería PennyLane para la creación de simulaciones de circuitos cuánticos. Sin embargo, cabe destacar que la dificultad del proyecto ha estado meramente en la asimilación de los conceptos teóricos sobre computación cuántica y el aprender a utilizar la librería de PennyLane, el proyecto no necesitó de una implementación complicada: los cuadernos no necesitan de una base de datos que les soporte, ni de servidores, por ejemplo. Esto no tiene por qué quitar mérito a la importancia del proyecto, ya que su reto se encuentra en tratar una tecnología emergente.

Además de las mejoras ya comentadas, se podría optimizar el método de distinción entre el caso con ruido y el caso con Eve, a partir de en vez de considerar un método de categorización entre el caso de con ruido e Eve, descartar aquellos que no sean suficientemente "ruido". Una idea de ello sería considerar cierta elipse con centro en el punto medio de las muestras de ruido obtenidas, descartando aquellos valores fuera de ella. Además, también el algoritmo de selección (tanto el actual como el sugerido) se podría mejorar a partir de la optimización de los parámetros que definen la división (ponderación de las componentes/los radios de la elipse) a partir de técnicas de inteligencia artificial; en concreto, la **validación cruzada**.

Parte II

Documentación técnica

Capítulo 6

Lenguajes de programación

6.1. Librerías de *Python*

Python es un lenguaje multiparadigma basado en la implementación por librerías. como su nombre indica, el ser un lenguaje multiparadigma significa que puede utilizarse en varios paradigmas de programación. En el caso específico de *Python*, éstos son esencialmente la programación estructurada y la orientada a objetos. Para que pueda utilizarse en este último, a nivel interno en Python todo es un objeto incluso las propias clases y las funciones (lo que permite indistintamente programar bajo un paradigma u otro). Como se dijo anteriormente, el principal motivo por el que se eligió *Python* como lenguaje de programación es por el hecho de utilizar la librería `PennyLane` para computación cuántica. Además de `PennyLane`, de la que se hablará detenidamente a continuación, otras librerías utilizadas han sido `Cryptodome` (para la importación de algoritmos de cifrado) y `matplot` (para la creación de histogramas y gráficos).

6.1.1. Crytodome

`Crytodome` es una librería de *Python* que contiene los principales algoritmos de cifrado (y descifrado) utilizados. En el estudio, se quiere poner en práctica la implementación del protocolo BB84 a la hora de generar una clave privada. Como el protocolo de cifrado AES utiliza una clave privada simétrica (que no es lo mismo a que sea un cifrado simétrico).

En concreto, el módulo `Cipher` permite crear un objeto en el que se transmiten varios elementos, para crear la "interfaz de cifrado" en caso del protocolo AES:

`cipher = new(key, AES.MODE_CBC, iv)` donde `key` es la clave de cifrado y `iv` es el vector inicial que utilizan en el algoritmo del protocolo.

Esta interfaz contiene las funciones `encrypt` y `decrypt` para cifrar y descifrar respectivamente. Puesto que su uso "en bruto" puede resultar pesado, se realizó el script `AES.py` para realizar su uso en los cuadernos:

Listing 6.1: AES.py

```
import base64
```

```
import hashlib

# pycryptodome es un paquete que incorpo
from Cryptodome import Random
from Cryptodome.Cipher import AES

class AESCipher(object):

    def __init__(self, key):
        self.bs = AES.block_size
        self.key = hashlib.sha256(key.encode()).digest()

    def encrypt(self, raw):
        raw = self._pad(raw)
        iv = Random.new().read(AES.block_size)
        cipher = AES.new(self.key, AES.MODE_CBC, iv)
        return base64.b64encode(iv + cipher.encrypt(raw.encode()))

    def decrypt(self, enc):
        enc = base64.b64decode(enc)
        iv = enc[:AES.block_size]
        cipher = AES.new(self.key, AES.MODE_CBC, iv)
        return self._unpad(cipher.decrypt(enc[AES.block_size:])).decode('utf-8')

    def _pad(self, s):
        return s + (self.bs - len(s) % self.bs) * chr(self.bs - len(s) % self.bs)

    @staticmethod
    def _unpad(s):
        return s[:-ord(s[len(s)-1:])]
```

6.1.2. Matplotlib

Como su nombre indica, la librería de `Matplotlib` sirve para mostrar gráficos y animaciones. En el caso específico de este proyecto, será utilizado para mostrar histogramas de las muestras obtenidas. La aparte de esa librería donde se encuentran las funciones para la visualización de gráficas es `pyplot`. En concreto, la función utilizada para eso es `matplotlib.pyplot.hist(x, bins=None, range=None, histtype='bar', ..., **kwargs)` (puede recibir más parámetros, pero solo se comentarán los más importantes).

- `x`. Datos de los que realizar el histograma
- `bins`. Puede ser un entero, o una array
  - Si es un entero, indica el número de columnas equiespaciadas que tendrá el histograma.
  - Si es un array indica en qué valores se hacen las divisiones para realizar las columnas.

- `range`. Dupla de valores `(a,b)` que indica el mínimo y el máximo considerados para crear el histograma. No tiene efecto en el caso en el que `bins` sea un array.
- `histtype`. Tipo de histograma, puede ser
  - `'bar'` Histograma de barras
  - `'barstacked'` Escalonado
  - `'step'` De línea
  - `'stepfilled'` De línea
- `color`. Color del histograma

Además, `hist` devuelve una estructura de datos formada por dos array, que indican los datos obtenidos en cada división, y las divisiones consideradas (el array `bins`), respectivamente.

Esta librería también se utiliza en la elaboración del mapeo utilizado, a partir del cual se realizará el proceso de *clustering* para dictar si un caso es ruido o de canal vulnerado. En concreto, la función `scatter(x,y)` dibuja en el gráfico los puntos (x, y) (`x`, `y` pueden ser arrays de valores para dibujar varios puntos a la vez), y la función `axline([x1,y1],[x2,xy], ...)` dibuja la recta que pasa por los puntos $(x1, y1)$ y $(x1, y2)$.

6.1.3. PennyLane

PennyLane[3] es la librería que se utilizará para simular los circuitos cuánticos del protocolo BB84 y derivados, siendo parte del cometido del estudio el mostrar la usabilidad básica de esta librería.

Dispositivos y circuitos

En `PennyLane`, los circuitos cuánticos son definidos como unas funciones especiales, marcadas a través del decorador `qnode`. A este decorador ha que pasarle un objeto de la clase `device`, que indica en qué "dispostivo" se va a correr el circuito (tipo del circuito, cuántos cables va a tener, qué se va a enviar, etc). Por ejemplo, esta sería la definición de un circuito compuesto únicamente por una puerta Hadamard (y en el que el estado inicial del qubit es $|0\rangle$).

Listing 6.2: Ejemplo de uso de la librería PennyLane

```
import pennylane as qml
from pennylane import numpy as np

# Se define el 'dispostivo' donde se trabaja.
dev = qml.device('default.qubit', wires=1)

# Decorador para indicar que la siguiente funcion
# va a ser un circuito cuantico sobre el dispositivo 'dev'.
```

```
@qml.qnode(dev)
def HadamardGate():
    qml.Hadamard(0)

# state() devuelve las componenetes en la base canonica
# del estado final del circuito. En el caso de un solo qubit,
# son los valores de a y b del estado final del qubit.
    return qml.state()
```

A lo largo de los distintos cuadernos, se utilizarán dos tipos distintos de dispositivos: `'default.qubit'` y `'default.mixed'`. La principal diferencia entre ambos que el `'default.mixed'` soporta las funciones que se utilizarán para implementar el ruido, `BitFlip()` y `DepoalirizingQubit()`. A cambio, `'default.mixed'` pierde la posibilidad de acceder a información sobre el gradiente el qubit, utilizado en *quantum machine learning* (pero no importante para el tema tratado en este proyecto).

Puertas lógicas

`PennyLane` tiene implementadas además las puertas más comúnmente utilizadas, como las puertas de Pauli (`PauliX()`) o la de Hadamard (`Hadamard()`). Además, si sabes la descomposición en rotaciones de la puerta a aplicar, puedes aplicar cualquier puerta gracias a la función `Rot()`.

Todas estas funciones, comparten los mismos parámetros:

- Los propios argumentos de la puerta (por ejemplo, para la función texttRX), se requiere el ángulo de rotación sobre el eje x.
- `wires`. Cable del circuito donde se encuentra situada la puerta.
- `doQeue`. Si la puerta quiere aplicarse directamente puede ser `True` o `False`
- `id`. Identificador que se quiera colocar a la puerta.

Un caso curioso es el de las puertas controladoras (*controlled*), en el se pasa como parámetro la función de la puerta a aplicar el controlador. Esto hace que a los propios argumentos de la función haya que enviarles argumentos:

Listing 6.3: Ejemplo puerta controladora

```
import pennylane as qml
from pennylane import numpy as np

dev = qml.device('default.qubit', wires=1)

@qml.qnode(dev)
def CNOTGate():
    qml.ctrl(qml.PauliX(1), (0))
    return qml.state()
```

Respuesta del circuito

En los ejemplos mostrados hasta ahora, las funciones de simulación del circuito siempre devuelven `state()`, que devuelve las componentes (en la base canónica) del estado final del qubit. Otros métodos utilizados en el cuaderno para estructurar la respuesta de las funciones circuitos son `probs()` y `sample()`.

Como su nombre sugiere, `probs()` devuelve la probabilidad del resultado al medir el qubit final sea el valor estático asociado (es decir, el cuadrado del módulo de `state()`. Se puede restringir y solo considerar parte de los cables del circuito mediante el parámetro `wires`.

Por otra parte, el método `sample()` devuelve la medición de los estados finales (en la polarización pasada por `op`). Para poder aplicar este método, es necesario que el dispositivo del circuito donde se utiliza tenga el argumento `shots`, que indica el número de muestras del circuito que se devuelven [1].

6.1.4. Numpy

`Numpy`[8] es la librería de *Python* por excelencia para trabajar sobre "estructuras matemáticas" complejas, sobre todo para el manejos de arrays numéricos (tiene su propia implementación de `array`) así como funciones para resolución de matrices, algoritmos de estimación, etcétera.

En nuestro caso, no se utiliza directamente esta librería, sino la modificación creada y utilizada en PennyLane. Esta versión ha sido alterada para que sea más cómodo operar con las puertas lógicas y estados cuánticos. Además de utilizarla de manera indirecta a través de `PennyLane`, también es utilizada en los cuadernos en varias ocasiones de manera directa, desde la generación de números aleatorios (con las funciones de su módulo `random`), hasta herramientas estadísticas como `percentile(v,p)`, que devuelve el/los percentiles del vector de muestras `v` o `_r()` (que permite construir arrays como se haría en R).

6.1.5. TensorFlow

TensorFlow es una librería destinada a la construcción de redes neuronales y del aprendizaje automático en general (recordemos que Pennylane es una librería específica para *quantum machine learning*). Para nuestro proyecto solo es interesante la estructura de datos que utiliza esta librería: los **tensores**.

De manera puntual se ha necesitado la función `squeeze(t)`, que elimina las dimensiones del tensor `t` de tamaño 1 (es decir, si `t` es un tensor con array de dimensión `[3,1,2]`), te devuelve un tensor cuyo array de valores es el mismo que el de `t`, pero su array de dimensiones es `[3,2]` (para la preparación de datos a la hora de la creación del histograma).

[1] Del mismo modo `probs()` y `state()` necesitan de que el dispositivo NO tenga el argumento `shots`.

6.2. *Jupyter Notebook*

Los cuadernos *Jupyter* (*Jupyter Notebook*)[13] se definen como una "plataforma de computación interactiva". Eso significa que es una interfaz web (es decir, que se puede visualizar el archivo creado en formato de página web desde un navegador web), en el que se puede mezclar texto, multimedia y código ejecutable, como si de un "cuaderno de código" se tratase (de ahí el nombre). Los cuadernos están divididos en **celdas**, que pueden ser de código ejecutable o no (Markdown).

El formato que ofrecen los cuadernos *Jupyter* es perfecto para la creación de informes, estudios u cualquier tipo de documento en el que se haya tenido que programar, puesto que, a diferencia de en un documento tradicional, en un cuaderno *Jupyter* se puede integrar el código en el documento y ejecutarlo personalmente.

6.2.1. Arquitectura y funcionamiento

Los cuadernos *Jupyter* son una interfaz de visualización de tanto archivos multimedia como de código ejecutado. Esa visualización se hace a través de un navegador web.

El lugar donde se ejecutan las celdas que componen los cuadernos (tanto las de "código" como las de `Markdown`) es el kernel, que compila el código de la celda en el lenguaje indicado (siendo `Markdown` uno de estos posibles lenguajes).

El propio archivo `.ipynb` (que es un archivo en formato JSON que genera el cuaderno [2]), el navegador y el kernel se encuentran conectados entre sí por el servidor *Jupyter*, de manera de que cada uno funciona de manera independiente del resto: el kernel lo único que hace es ejecutar los comandos transmitidos desde el servidor a partir las celdas contenidas en el archivo `.ipynb` como si fuesen entradas por la consola de comandos, y el navegador web lo único que hace es mostrar en el navegador la compilación del archivo `.ipynb` (como JSON) por el servidor enviada, a través de WebSockets.

[2]Para poder observar un cuaderno *Jupyter* como JSON, basta con abrirlo como texto plano con cualquier editor de texto (por ejemplo, Notepad), dando como resultado lo mostrado en la figura 6.1.

```
{
 "cells": [
  {
   "cell_type": "markdown",
   "id": "f95c33a5",
   "metadata": {},
   "source": [
    "Anterior Notebook: [Conceptos generales de computación Cuántica](Introduccion.ipynb)"
   ]
  },
  {
   "cell_type": "markdown",
   "id": "37fd425b",
   "metadata": {},
   "source": [
    "En esta sección se explicarán los principios de computación en los que se fundamenta el protocolo BB84, así como su funcionamiento
    "\n"
   ]
  },
  {
   "cell_type": "markdown",
   "id": "72233c25",
   "metadata": {
    "vscode": {
     "languageId": "html"
    }
   },
   "source": [
    "<div>\n",
    "Listado de contenidos:\n",
    "<ul>\n",
    "<li><a href=#NoCloning>No-Cloning Theorem</a></li>\n",
    "<li><a href=#Bolch>Funcionamiento del protocolo BB84 </a></li>\n",
    "<ul>\n",
    "<li>\n",
    "    <a href=#Aplicacion>Aplicación</a> </li>\n",
    "</ul>\n",
    "<li><a href=#Evesdropping>Evitamiento del Evesdropping</a></li>\n",
    "<li><a href=#Integridad>Importancia del test de integridad</a></li>\n",
    "</ul>\n",
    "\n",
    "</div>"
   ]
  },
  {
   "cell_type": "markdown",
   "id": "f958d320",
   "metadata": {},
   "source": [
    "# <div id=\"NoCloning\"> No-Cloning Theorem </div>"
   ]
  ]
```


Figura 6.1: Visualización de un cuaderno *Jupyter* como archivo JSON.

La comunicación entre el kernel y el servidor de *Jupyter* se basa en la transmisión de las celdas con y sin compilar por el kernel, y se hace mediante MQ (llamado también ZeroMQ), un middleware de comunicaciones orientado a mensajes[1].

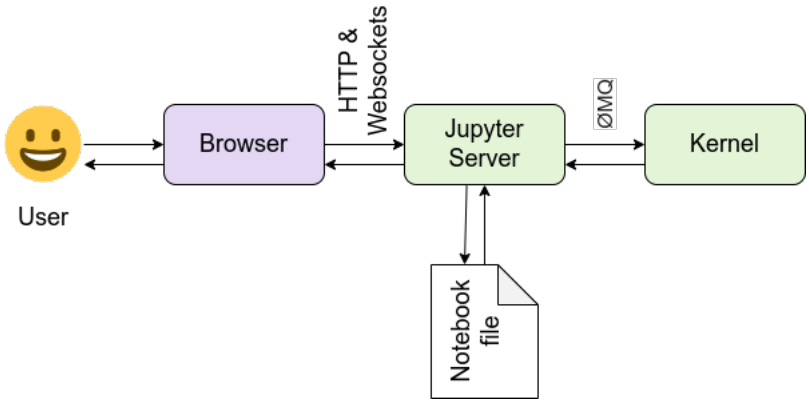
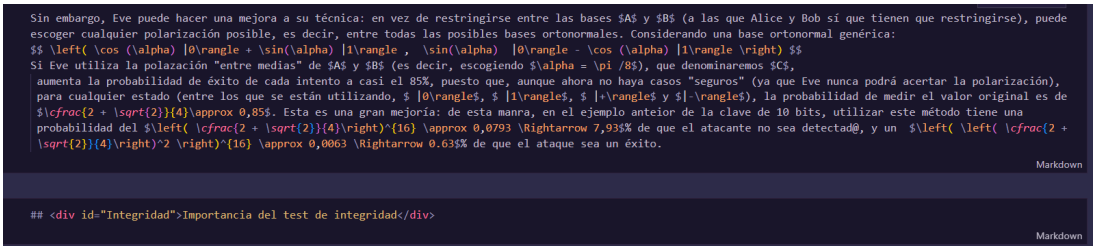


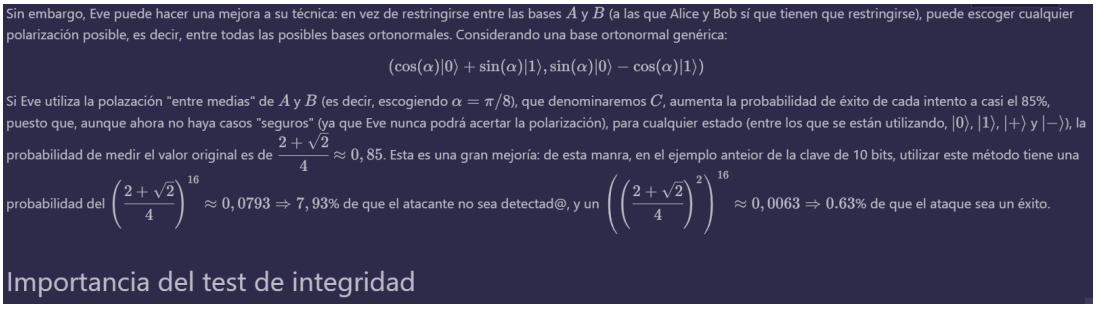
Figura 6.2: Arquitectura de *Jupyter Notebook*

6.2.2. Entorno de Markdown

Para la redacción de las celdas de no-código (Markdown), cabe destacar las herramientas que ofrece para inclusión de todo tipo de archivos multimedia y texto, desde fórmulas matemáticas hasta archivos de audio. Esto es posible debido a que en realidad estas celdas en realidad sí que se compilan. El núcleo de Markdown se compila en **JavaScript** (lo que tiene sentido, puesto que el cuaderno se tiene que visualizar en un navegador web), lo que permite utilizar etiquetas HTML para la edición del texto. Además, también tiene varias expresiones propias, para la inserción de imágenes o títulos, por ejemplo.



(a) Markdown sin compilar



(b) Markdown compilado

Figura 6.3: Ejemplo de celdas de Markdown

Los cuadernos *Jupyter* suelen utilizarse -como en nuestro caso- como herramienta de divulgación, por lo que hará falta poder presentar y excribir expresiones matemáticas dentro del cuaderno.¿Cómo hace entonces para poder insertar fórmulas matemáticas? La respuesta es `MathJax`. `MathJax` es una biblioteca de JavaScript que *Jupyter* tiene incorporada que permite escribir fórmulas matemáticas tal y como se haría en LaTeX[5].

En cuanto a las celdas de código, cabe destacar la variedad de lenguajes en la que se es posible compilar en los cuadernos. Entre los lenguajes de programación más importantes, caben destacar Julia, *Python* y R, que son de los que recibe el nombre (**Ju**lia-**pyt**hon-**R**). Entre ellos, se decidió utilizar *Python*, para poder hacer uso de la librería `PennyLane`, que es necesaria para realizar el estudio, como se ha comentado.

Capítulo 7

Análisis

7.1. Requisitos

En este apartado se describen los requisitos del sistema que se va a desarrollar: requisitos funcionales, de interfaz de usuario, de información, etc.

Recordemos que los objetivos del proyecto son dos:

1. Servir como material introductorio para el aprendizaje sobre computación cuántica.
2. Explicar el funcionamiento del protocolo BB84, tanto en un canal "limpio" como en uno con ruido.
3. Obtener una posible mejora del protocolo para cuando es aplicado sobre un canal con ruido, acompañado de una implementación en `PennyLane` del mismo.

7.1.1. Requisitos funcionales

En caso de la redacción de un documento, sus requisitos funcionales son simplemente que en él se encuentren los contenidos de los que se quiera hablar de él. De esta manera, se pueden dividir los requisitos funcionales del proyecto en cuatro bloques, correspondientes a los temas que se tiene que tratar en el estudio:

Conceptos básicos de la computación cuántica. En este cuaderno se expondrán los conceptos introductorios a la computación cuánticas necesarios para la comprensión del protocolo, haciendo hincapié en los circuitos cuánticos y las puertas lógicas que lo componen.

Nombre del Requisito	RF-01 Explicación de conceptos básicos.
Descripción	Se debe explicar al lector el concepto de qubit, así como otros conceptos básicos en la computación cuántica como la esfera de Bloch .

Prioridad	Alta
Otra info	

Nombre del Requisito	RF-02 Visualización de circuitos de puertas lógicas.
Descripción	El usuario debe poder visualizar la implementación en *Python* de las puertas lógicas.
Prioridad	Media
Otra info	

Nombre del Requisito	RF-03 Mostrar las puertas lógicas.
Descripción	El lector debe obtener nociones básicas de las puertas lógicas utilizadas en el protocolo.
Prioridad	Alta
Otra info	Las puertas de Pauli son unas puertas rotaciones específicas. Una imagen de cómo se representan en la esfera de Bloch es bastante visual para explicar tanto unas como otras. También conviene comentar las puertas controladoras.

Teoría protocolo BB84. En este cuaderno se explicará todo lo relacionado con el protocolo BB84 a nivel teórico, su aplicación, su funcionamiento, su circuito cuántico asocia, etcétera.

Nombre del Requisito	RF-11 Demostración de No-Cloning Theorem.
Descripción	El lector debe comprender la importancia del NCT para la computación cuántica, sobre todo en su aplicación para la seguridad.
Prioridad	Alta
Otra info	La demostración del NCT es bastante compleja y requiere de conceptos matemáticos elevados. Por ello su papel debe ser que se comprenda el resultado más que explicar su porqué.

Nombre del Requisito	RF-12 Mostrar circuito cuántico del protocolo BB84.
Descripción	El usuario debe poder ver la expresión gráfica del circuito asociado al protocolo BB84, además de comprender su funcionamiento
Prioridad	Alta
Otra info	Complementar con un ejemplo ayudaría a su comprensión.

Nombre del Requisito	RF-13 Visualización del problema de la clave privada.
Descripción	El lector debe visualizar el problema que supone el problema de la clave privada, y por qué es importante buscar manera para resolverlo
Prioridad	Alta
Otra info	Una imagen que ilustre el ataque facilitará su comprensión.

Nombre del Requisito	RF-14 Explicación del test de Integridad.
Descripción	El lector debe entender por qué es necesario la realización de un test de Integridad complementario al protocolo BB84 para que este funcione.
Prioridad	Alta
Otra info	

Nombre del Requisito	RF-14 Probabilidad de fallo.
Descripción	Se debe comentar al lector el hecho de que haya una probabilidad de que el protocolo falle, y el porqué de ello.
Prioridad	
Otra info	Aprovechar para demostrar que la implementación realizada obtiene estadísticas acorde a este resultado.

Implementación básica del protocolo En este cuaderno se desarrollará la implementación del protocolo BB84, explicando cada una de sus partes.

Nombre del Requisito	RF-21 Explicación de la implementación del circuito.
Descripción	El lector debe comprender cómo y por qué se ha simulado el protocolo en PennyLane de la manera realizada.
Prioridad	Alta
Otra info	

Nombre del Requisito	RF-22 Consideraciones de implementación.
Descripción	Se debe explicar las consideraciones esenciales que se han tenido a la hora de implementar el protocolo, para cada una de sus partes.
Prioridad	Media
Otra info	

Nombre del Requisito	RF-23 Mostrar implementación del circuito del protocolo BB84.
Descripción	El usuario debe poder visualizar la implementación en *Python* del circuito del protocolo BB84.
Prioridad	Media
Otra info	

Nombre del Requisito	RF-24 Mostrar ejemplo de ejecución .
Descripción	Se debe mostrar al lector el funcionamiento de la simulación del protocolo BB84, mediante un ejemplo de ejecución.
Prioridad	Alta
Otra info	

Implementación del protocolo en un canal con ruido /con Eve. En este cuaderno se harán modificaciones a la implementación anterior del protocolo para simular que se aplica tanto para un canal con ruido como para un canal espiado. Además, se estudiará como modificar el protocolo para que sea más eficaz en diferenciar ambos casos.

Nombre del Requisito	RF-31 Implementación del protocolo con ruido.
Descripción	El lector debe ver el código de la implementación del protocolo en un canal con ruido, así como la justificación de su correcto funcionamiento.
Prioridad	Alta
Otra info	Se trabajará considerando que el ruido máximo que admite el canal es del 15 %.

Nombre del Requisito	RF-32 Implementación del protocolo con Eve.
Descripción	El lector debe ver el código de la implementación del protocolo en un canal vulnerado, así como la justificación de su correcto funcionamiento.
Prioridad	Alta
Otra info	

Nombre del Requisito	RF-33 Visualización de estadísticas de fallo.
Descripción	Se debe mostrar al lector estadísticas de fallo en el caso con Ruido y con Eve.
Prioridad	Alta
Otra info	Esto puede realizarse a través de la creación de un histograma. Conviene además comparar ambos histogramas.

Nombre del Requisito	RF-34a Razonamiento de mejora del protocolo.
Descripción	Se debe explicar cómo mejorar el protocolo para que funcione en un canal con ruido.
Prioridad	Alta
Otra info	Esto puede realizarse a través de la creación de un histograma. Conviene además comparar ambos histogramas.

Nombre del Requisito	RF-34b Implementación de mejora del protocolo.

Descripción	El lector debe ver el código de la implementación mejorada, así como la justificación de su correcto funcionamiento.
Prioridad	Media
Otra info	

Nombre del Requisito	RF-35 Conclusión final.
Descripción	El documento debe contener una conclusión final en la que se suma todo lo comentado en los cuadernos y que se comenten las implicaciones de lo estudiado.
Prioridad	Alta
Otra info	

Además, también existen otros requisitos funcionales de hábito más general, que se deberán de tener en cuenta en el desarrollo de todos los cuadernos:

Nombre del Requisito	RF-40 Comprobación de la implementación .
Descripción	Se debe demostrar al lector que los circuitos propios creados son correctos.
Prioridad	Alta
Otra info	

Nombre del Requisito	RF-41 Rigor biblográfico.
Descripción	En cada cuaderno debe aparecer las fuentes bibliográficas utilizadas, además de estar citadas en formato APA, añadiendo enlace a la cita dentro del texto cuando sea necesario.
Prioridad	Media
Otra info	

Nombre del Requisito	RF-42 Tabla de contenidos.

Descripción	En cada cuaderno deben aparecer un esquema que indique los contenidos que se van a tratar en el mismo.
Prioridad	Baja
Otra info	Enlazar dicha entrada de la entrada con el contenido en sí daría más completitud y cohesión al documento.

7.1.2. Requisitos no funcionales

Además de los requisitos funcionales (que en este caso son prácticamente a los contenidos que se deben elaborar en el cuaderno) también hay que tener en cuenta ciertos requisitos no funcionales. Como material didáctico y de investigación, debe de estar lo suficientemente bien redactado y estructurado.

El texto debe tener una buena redacción y cohesión.

En cada cuaderno deben aparecer las fuentes bibliográficas utilizadas, además de estar citadas en formato APA.

Links entre los cuadernos.

Nombre del Requisito	RNF-01. Buena redacción
Descripción	El texto que conforma el estudio debe estar bien redactado y cohesionado
Prioridad	Alta
Otra info	Hay que hacer especial hincapié en una buena ortografía y que sea comprensible para cualquier lector.

Nombre del Requisito	RNF-03 Datos de salida limpios.
Descripción	Las respuestas que devuelven cada una de las celdas de los cuadernos Jupyter deberá de ser en una estructura de datos legible, y comprensible para el usuario (no simplemente una cifra/array, por ejemplo)
Prioridad	Baja
Otra info	En casos excepcionales comola generación de arrays para estadísticas se considerará obviar este requisito.

7.1.3. Atributos de calidad

Los atributos de calidad técnicos de los cuadernos como aplicación software (como la robustez) vienen dados por la arquitectura y el funcionamiento de *Jupyter* como software, por lo que no incumben en la elaboración del proyecto. Sin embargo, esto da lugar a que otros atributos de calidad cobren mayor importancia, a saber:

Portabilidad: Los cuadernos deben ser capaces de ser utilizados en cualquier sistema operativo y navegador (esto es inherente al funcionamiento de los cuadernos *Jupyter*).

Fiabilidad: La variables de entorno usadas en los cuadernos y/o archivos .py asociados tendrán que ser limpiadas correctamente de manera de que siempre aparezca en los cuadernos un resultado verídico. En concreto, con las celdas donde el resultado obtenido depende del azar.

Usabilidad: El código de los cuadernos debe estar debidamente comentado para su buena comprensión. Además, tendrá que resultar sencilla pasar de un cuaderno a otro cunado se requiera (en relación con el requisito...).

Robustez: tiene que ver con el comportamiento, la fiabilidad y la estabilidad de la aplicación ante posibles incidencias en tiempo real.

7.2. Diseño

En esta sección se hablará por una parte sobre la estructuración de los cuadernos que componen el estudio, así como las relaciones de los scripts de *Python* en los que se encuentran alojadas las funciones utilizadas en ellos.

7.2.1. Cuadernos *Jupyter*

Como se ha comentado en la sección de Planificación, el estudio se compone de cuatro cuadernos:

Introduccion.ipynb: En este cuaderno se presentan las nociones básicas de computación cuántica. Entre los contenidos del cuaderno, se encuentran:

- El concepto de qubit
- La esfera de Bloch.
- Puertas lógicas uni-qubit
- Puertas *Controlled*

Este cuaderno contiene una gran carga teórica, así que a penas habrá celdas de código. No obstante, se ilustrará la presentación de las puertas lógicas comentadas con sencillos circuitos utilizando la librería `PennyLane`, en concreto con las funciones que que implementan dichas puertas y el método `state()`, para devolver las componentes en la base canónicas del qubit al final del circuito.

TeoriaBB84.ipynb: En ese cuaderno se explicará cómo funciona el protocolo BB84 a nivel teórico. Esto implica tener que hablar de los **resultados teóricos** que utiliza el protocolo, presentar su **escenario de aplicación**, además explicar **el protocolo en sí y su funcionamiento**. Igual que el anterior, a penas habrá celdas de código, puesto que el resto de cuadernos se dedican a la implementación de lo comentado en este. Su contenido es similar a lo comentado en la sección 2.3.

ImplementacionBB84Basico.ipynb En este cuaderno se dan dos posibles implementaciones del circuito cuántico que utiliza el protocolo BB84. En una de las implementaciones se considera tratar en el circuito con la cadena de bits para generar la clave directamente (el caso perteneciente al script `BB84Alternativo.py`). Mientras, la otra implementación se decanta porque en el circuito se traten los bits -qubits- de manera individual(`BB84Basic.py`). Para el siguiente cuaderno, se decantará por continuar por la segunda implementación al ser más sencilla y cómoda.

ImplementacionBB84Complejo.ipynb: En el último cuaderno del estudio, se intenta expandir la implementación del circuito comentada en el cuaderno anterior al caso donde hay ruido en el canal, o un atacante en él (lo relacionado con `BB84Ruido.py` y `BB84Eve.py`, respectivamente). A partir de ello, se estudia qué posibles herramientas tienen Alice y Bob para diferenciar entre el caso benigno (que haya ruido) y el peligroso (que haya una tercera persona que escuche el canal), para lo cual se realiza un muestreo se hace uso de las librerías `Matplotlib` y `Numpy` para calcular las estadísticas de casa caso. Los resultados se darán partiendo de la suposición de que se considera que la saturación del canal se da para un ruido del 15 % (es decir, que el máximo ruido tolerable para la funcionalidad del canal es para $p = 0{,}15$).

Por último, esta parte da lugar a buscar una variación del protocolo BB84 mejorado, donde se pueda diferenciar entre ambos casos (contenido del script `BB84Extra.py`). Este cuaderno es el más largo y extenso del conjunto

7.2.2. Scripts de *Python*

Además de los cuadernos que componen el proyecto, el trabajo también contiene una serie de scripts de pyhton que se utilizarán en los cuadernos y/o par preservar las funciones mostradas y explicadas en los mismos. Estos archivos se pueden dividir en tres subgrupos:

Útiles En estos scripts Se encuentran las funciones y clases necesarias para la implementaciones de los circuitos cuánticos en sí. Esta formado por dos archivos:

- `Person.py` Este script se contiene la implementación de la clase `Person`, encargada de simular a Alice y a Bob.
- `OperacionesBB84.py`. Contiene las funciones que realizan las acciones del protocolo BB84 que no utilizan el circuito cuántico, como el cribado de bits o la realización del test de integridad. En el diagrama de actividad de figura 7.1 se muestra todo lo que es necesario realizar por el protocolo BB84, además de transmitir por el circuito asociado.

  Dentro de los algoritmos a implementar en este script a nivel de diseño destaca el cribado de bits a partir de las polarizaciones

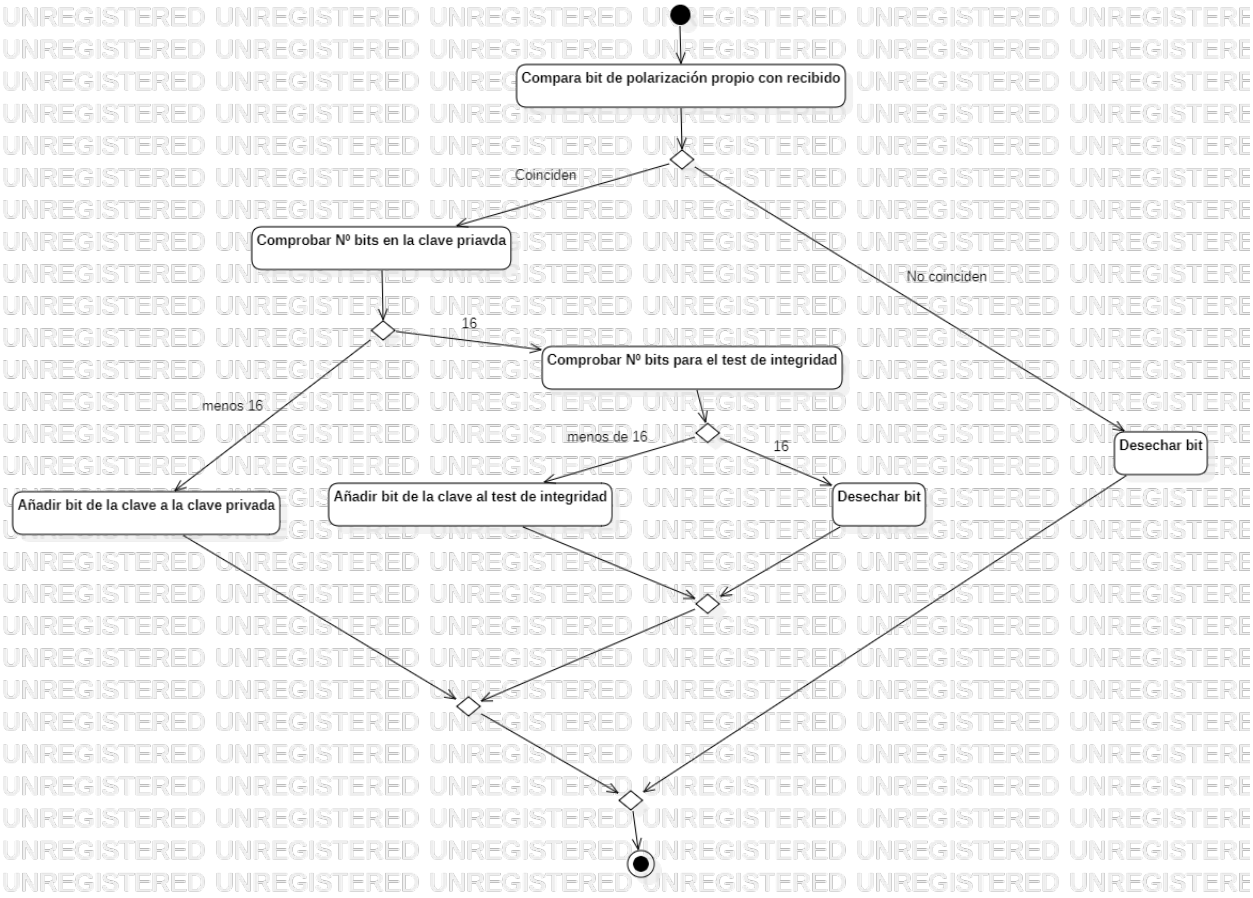


Figura 7.2: Diagrama del proceso de cribado de bits de los bits transmitidos

Implementaciones del circuito . En estos scripts se realizan las implementaciones de los circuitos cuánticos para cada uno de los casos a analizar. Para ello, hace uso de la librería `PennyLane`; y son la parte principal del proyecto. Además de métodos privados que necesitan específicamente cada variante, estos cuadernos contienen la implementación del circuito cuántico, así como funciones que realizan la ejecución entera del protocolo (tanto llegando solo a haber realizado el test de integridad,

como hasta haber generado la clave privada[1]).

- `BB84Basic.py` Contiene la implementación del circuito del protocolo BB84 sin ruido y/o atacante. Además, también se encuentran funciones que realizan la ejecución entera del protocolo (tanto llegando solo a haber realizado el test de integridad, como hasta haber generado la clave privada).
- `BB84Alternativo.py` Contiene la implementación del circuito del protocolo BB84 sin ruido y/o atacante, realizando la implementación del circuito cuántico para transmitir la cadenas de bits directamente en vez de uno en uno. Además, también se encuentran funciones que realizan la ejecución entera del protocolo.
- `BB84Ruido.py`. Contiene la implementación del circuito para el caso en el que el canal tenga cierto porcentaje de ruido de cambio de polarización y/o de bit. Al igual que para el caso básico, también contiene funciones que realizan la ejecución
- `BB84Eve.py` Contiene la implementación del circuito para el caso en el que el canal esté siendo espiado (y, por tanto, medido) por un atacante Eve. Al igual que para el caso básico, también contiene funciones que realizan la ejecución entera del protocolo (tanto llegando solo a haber realizado el test de integridad, como hasta haber generado la clave privada).
- `BB84Extra.py`. Contiene una función que implementa el circuito BB84 de manera generalizada, que aplicará uno u otro de los circuitos cuánticos implementados en los archivos anteriormente mencionados dependiendo de los parámetros transmitidos,

AES . Además, de manera independiente, también se encuentra el script `AES.py`, donde está implementada una clase con el mismo nombre que, importando métodos de las librerías `Cryptodome`, `hashlib` y `base64`, una clase cuyo cometido es simplificar el proceso de cifrado y descifrado AES, utilizado en los cuaderno como ejemplo de uso del protocolo BB84.

[1]la ejecución del protocolo intermedia (la que llega hasta después del test de integridad) es necesaria, puesto que es la que en los cuadernos es utilizada para obtener estadísticas, además de ser la parte que se reutiliza al final para la implementación mejorada sugerida.

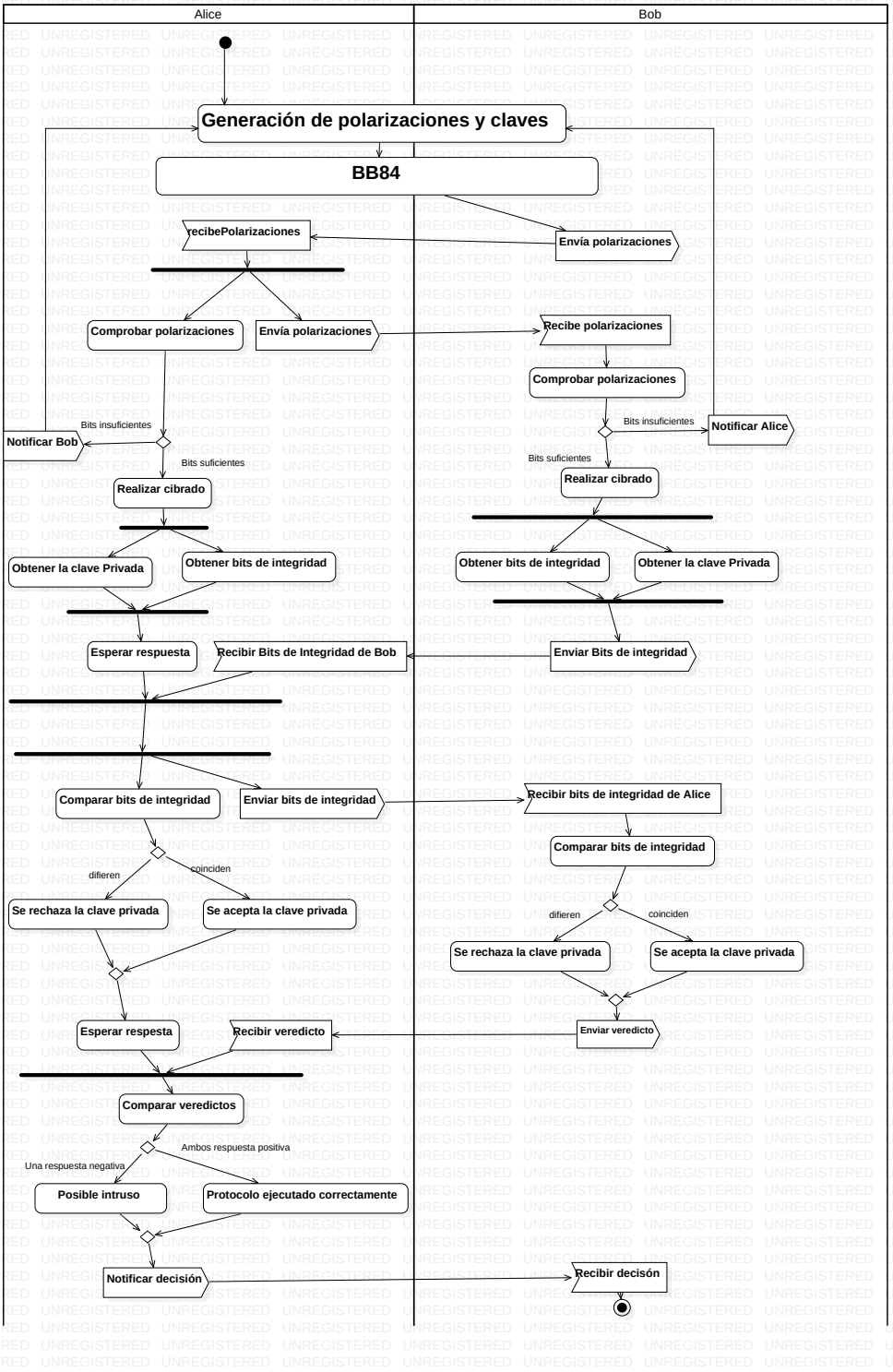


Figura 7.1: Diagrama de actividad del algoritmo BB84

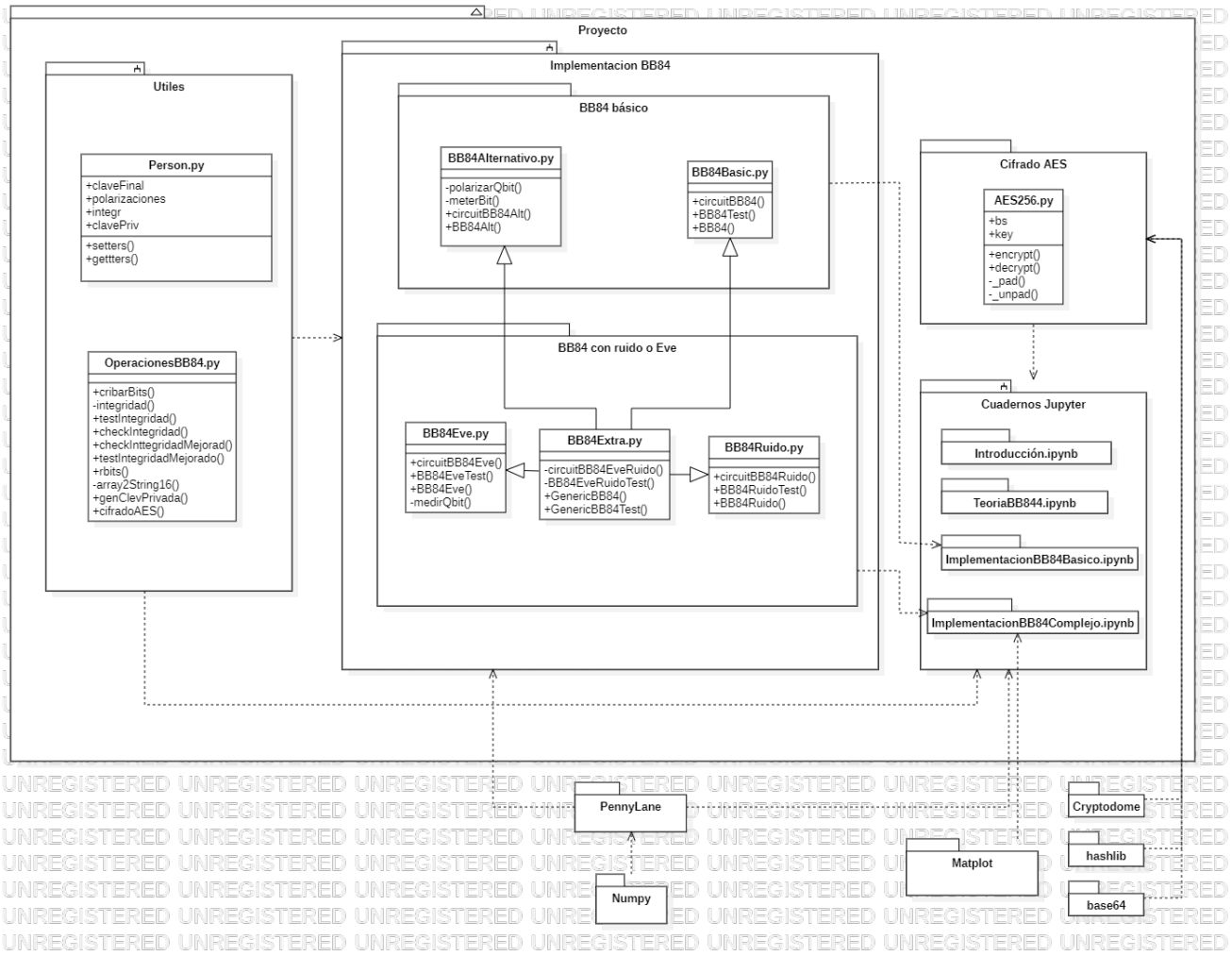


Figura 7.3: Diagrama de paquetes del proyecto

Capítulo 8

Implementación

En este capítulo se explicará el razonamiento seguido para la implementación de la simulación del circuito cuántico asociado al protocolo BB84, así como las variaciones construidas durante los cuadernos (con ruido, con Eve, y el mejorado).

8.1. `Person.py`

En cuanto a los métodos pertenecientes a la clase, solo serán necesarios los "set" y "get" de cada uno de sus atributos. En una implementación más orientada a objetos, la mayor parte de las funciones construidas podrían pasar a ser métodos de esta clase. Sin embargo, como se explicó anteriormente, de cara a un mejor seguimiento y división del código en los cuadernos se abogó por la programación estructurada. Lo que sí que es importante son los atributos pertenecientes de esta clase, puesto que contendrán las cadenas de bits que están utilizado cada una de las personas (Alice y Bob) en cada momento:

- `_pol`: Polarizaciones
- `_clave`: Bit iniciales de dónde sacar la clave final y los bits para el test de integridad
- `_claveFinal`: Bits de `_clave` tras cribar según las polarizaciones. Este array se divide en `_integr` y el array que formará `__clavePriv`
- `_integr`: Bits utilizados para comprobar la integridad del canal.
- `_clavePriv`: Clave privada obtenida (es String).
- `claveObtenida`: Booleano que indica si se ha pasado o no el test de integridad.

8.2. `OperacionesBB84.py`

Como se ha comentado ya anteriormente, el protocolo BB84 consta de, a grandes rasgos, los siguientes pasos:

1. Crear y asignar las cadenas de bits que se utilizaran para dictar tanto las polarizaciones de Alice y Bob como la cadena inicial que utilizará Alice (es decir `Alice._pol`, `Alice._clave` y `Bob._pol`) .

2. Ejecución del circuito cuántico del protocolo, a partir del cual Alice obtiene tanto su cadena de polarizaciones P_A, como los bits para la clave C_A (lo mismo para Bob, obteniendoP_B y C_B).

3. Bob hace pública P_B y Alice hace pública P_A, de manera que Alice y Bub puedan, individualmente, desechar los valores donde no han escogido la mismas polarizaciones, obteniendo P'_A y P'_B, desechar los mismos bits de las claves, obteniendo C'_A y C'_B.

4. Los n primeros bits de C'_A y C'_B se utilizarán para certificar que la integridad de la comunicación, mientras que los (en nuestro caso) 16 últimos serán la clave compartida generada.

Creación de las cadenas iniciales. Lo primero de todo entonces será crear cadenas aleatorias de bits. Para ello, se crea la función `rbits()`, que simplemente adapta la función `random.choice(i,n)` (que da n números enteros aleatorios menores que i) de la librería Numpy.

Con ello, se pueden ya crear las cadenas de bits iniciales necesarias

Listing 8.1: rbits()

```
def rbits(n:int):

    return np.random.choice(2,n)
```

Circuito cuántico. El siguiente paso es implementar el circuito cuántico. Esta parte es la que varía según el caso que se esté estudiando, y , como se comentó en el apartado de Diseño, las distintas implementaciones de esto se encuentran en el resto de archivos, por lo que se verá más adelante.

Cribar Bits Para este paso, simplemente tienen que compartir las polarizaciones entre ellos [1] para poder cribar `_clave` y conseguir así `_claveFinal`, cada uno de manera independiente.

Para ello, cada vez que la polarización de uno coincida con la propio, añadirán el valor del bit de `_clave` en la misma posición al array `_claveFinal`:

[1] Si se tuviera que prestar más atención al canal en la simulación -que no es el caso-, éste paso tendría que ser mucho más complejo y minucioso: tras Bob haber comprobado que ha recibido **todos** los bits, y haberlo notificado a Alice, Bob envía (y solo entonces) **primero** sus polarizaciones y **posteriormente** Alice las suyas; no las comparten de manera simultánea.

En caso de que se quiera realizar una simulación del protocolo haciendo hincapié en el carácter distribuido, o se quiera prestar más atención al papel del posible atacante, habrá que tener esto en cuenta.

Listing 8.2: cribarBits()

```
def cribarBits(p:Person, pol:int):
    pFinal=[]
    for i in range(len(p._pol) ):
        if (p._pol[i]==pol[i]):
            pFinal.append(p._clave[i])
    p.set_claveFinal(pFinal)
```

Se puede observar que no se presta mucha atención a qué ocurre en el canal. Esto es debido a que como en el estudio nunca se presta atención a si la clave obtenida por Eve es correcta o no, es irrelevante lo que ocurre en este punto. En caso de que se considere hacer el cuaderno adicional sobre las tasas de éxito de Eve, sí que se necesitará simular una clase `Canal` que indique cuál es el mensaje transmitido en cada momento, y quién es su emisor.

Test de Integridad. A continuación, se necesita realizar el test de Integridad. Recordemos que esto lo tienen que hacer de manera independiente ambos objetos `Person` Alice y Bob.

Lo primero que se necesita es saber cuáles son los bits que se van a utilizar para comprobar la integridad (es decir qué elementos del array de `claveFinal` componen los array `_integr`). Esto lo realiza la función `integridad()`:

Listing 8.3: integridad()

```
def integridad(Person:Person):

    qbitsFinales = Person.get_claveFinal()[16:]
    Person.set_integr( qbitsFinales[:16])

    return Person.get_integr()
```

Obsérvese en la propia función `integridad()` ya se asigna el determinado valor al atributo de la clase. Pese a ello se sigue devolviendo su valor, para poder utilizarlo dentro del cuaderno.

Tras ello, se tiene que realizar el test de integridad propiamente dicho, es decir, compartir entre ellos los bits de integridad (`_integr`) y comprobar que son lo mismo, de manera similar a como se hizo para obtener las `__claveFinal` mediante la función de Numpy `array_equal()`:

Listing 8.4: Test de integridad

```
def testIntegridad(Alice, Bob):

    # Test de integridad
    intAlice = integridad(Alice)
    intBob = integridad(Bob)
```

```
    # Comprobaci n Alice
    testAlice = checkIntegridad(Alice, intBob)
    # Comprobaci n Bob
    testBob = checkIntegridad(Bob, intAlice)

    return testAlice, testBob

def checkIntegridad(Person:Person, clveIntegridad):
    if (np.array_equal( clveIntegridad ,Person.get_integr())):
        return True
    else:
        return False
```

Para obtener estadísticas sobre los casos avanzados del protocolo (así como la implementación mejorada sugerida del protocolo), se necesitará que no se devuelva si se supera el test de integridad o no, sino el número de fallos que se han obtenido.

Listing 8.5: Test de integridad Mejorado

```
def checkIntegridadMejorado(Person, clveIntegridad):
    suma = np.sum(abs(np.subtract(Person.get_integr(), clveIntegridad)))

    return suma

def testIntegridadMejorado(Alice:Person , Bob:Person):
    # Test de integridad
    intAlice = integridad(Alice)
    intBob = integridad(Bob)

    # Comprobaci n Alice
    testAlice = checkIntegridadMejorado(Alice, intBob)

    # Comprobaci n Bob
    testBob = checkIntegridadMejorado(Bob, intAlice)

    return testAlice, testBob
```

Obtención de la clave privada. Por último, falta obtener `_clavePriv`. Para ello, se tiene que construir una string a partir de los 16 primeros elementos de un array (es por ello que para el test de integridad se consideró a partir del decimosexto elemento):

Listing 8.6: array2string16()

```
def array2string16(password:str):

        #Primero, recortamos la contrase a obtenida para quedarnos con los
        primeros 16 bits para crear el iv

        password = password[:16]
        #En caso de que la contrase a tenga menos de 16 bits,
```

```
        ceros = False
        iv= ''

        for digit in password:
            iv = iv + str(digit)


        while(len(iv)< 16 ):
            #De manera temporal, rellenamos por 0
            iv = iv + '0'
            ceros = True
        return iv, ceros
```

Observemos que `array2string16()` devuelve, además de la String, si se han tenido que añadir ceros (lo que implica que el array pasado tenía menos de 16 elementos). En este caso habría que volver a repetir el protocolo, porque esto implica que no ha habido test sobrantes en `_claveFinal` para realizar el test de integridad.

Listing 8.7: Obtención de la clave privada

```
def genClavePrivada(p:Person):
        clave, ceros = array2string16( p._claveFinal[16:])
        p.set_clavePriv(clave)
        if(ceros):
            return False
        else:
            return True
```

Utilización de la clave generada Finalmente, también en este script se encuentra resumido la trasmisión de mensajes cifrados, utilizado en el último cuaderno simplemente para simplificar el código de los cuadernos.

Listing 8.8: cifradoAES()

```
def cifradoAES(Alice:Person,Bob:Person,msg):
    '''
    Alice env a a Bob el mensaje msg a Bob cifrado

    return:
        Fale si no ha habido error en el cifrado
        msg recibido por Bob si no ha habido error
    '''
    msgFinal = ""
    msgOg = msg
    msgEncriptado=""
    AES_Alice= aes(Alice._clavePriv)
    msgEncriptado =AES_Alice.encrypt(msg)
    AES_Bob= aes(Bob._clavePriv)

    try:
        msgFinal = AES_Bob.decrypt(msgEncriptado)
```

```
        return msgFinal
    except:
        print("ERROR EN EL CIFRADO")
        return False
```

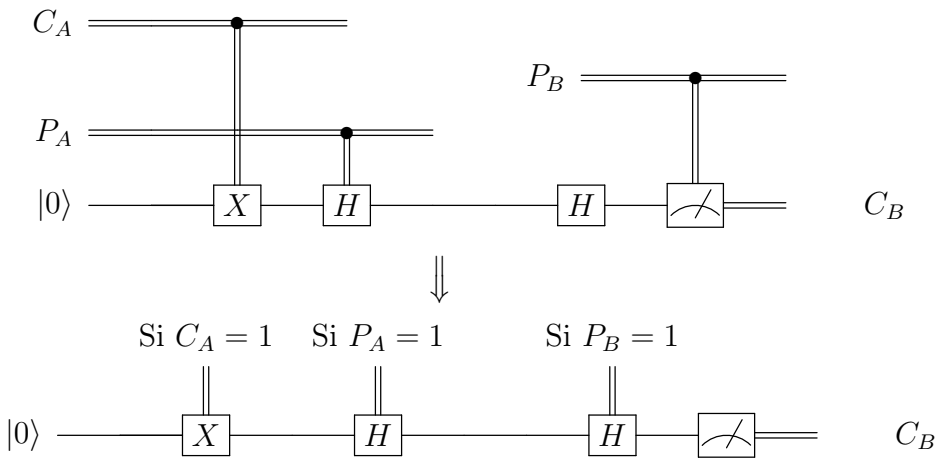
8.3. BB84Basic.py

El primer cuaderno sobre la implementación del protocolo es `ImplementacionBB84Basico.ipynb`. Como se comentó anteriormente, en este cuaderno se estudia cómo implementar en *Python* el protocolo BB84. En este archivo, se encontrará una implementación del circuito en la que cada bit de la cadena `clave` se transmite individualmente. En el script `BB84Alt.py` se encontrará una implementación del circuito alternativa en la que se transmite la cadena 'clave' por el circuito entera de manera conjunta.

En este archivo se encuentra lo relacionado con la primera implementación, mientras que en la siguiente sección se comentará la implementación alternativa.

Para los escenarios más avanzados comentados en el último cuaderno del proyecto, se descartará la implementación alternativa y solo se implementará de esta forma, para que su longitud no sea muy elevada.

La implementación del circuito en este script se basa en implementar el circuito en PennyLane, utilizando los condicionales para simular las *controlled* del circuito:



Con esta implementación, los qubits se transmiten por el circuito cuántico individualmente. Esto implica que haya que repetir el circuito $16 \times 4 = 64$ veces (16×2 para conseguir la clave compartida, 16×2 para conseguir suficentes bits para realizar el test de integridad). Sin embargo, a cambio de ello, la implementación del circuito es mucho más simple, puesto que nos podemos apoyar en condicionales para simular la aplicación o no de las puertas en vez de fabricar funciones por parámetros. Además, eso hace que esta implementación sea más natural al circuito cuántico real, siendo más comprensible su funcionamiento:

Listing 8.9: circuitBB84()

```
dev = qml.device('default.qubit', wires=1, shots=1)

@qml.qnode(dev)
def circuitBB84(bit:int,AliceP:int, BobP:int):

#Parte de Alice
    if(bit):
        qml.PauliX(0)

    if(AliceP):
        qml.Hadamard(0)

#Parte de Bob
    if(BobP):
        qml.Hadamard(0)

    return qml.sample(None, [0])
```

Añadiendo las funciones creadas en `OperacionesBB84.py` para implementar el protocolo hasta el test de integridad, se obtiene una función para realizar el protocolo BB84 hasta comprobar que se cumple el test de integridad:

Listing 8.10: BB84Test()

```
def BB84Test(Alice:Person , Bob:Person):
    Alice.set_integr([])
    # Reiniciar contador de ceros
    testAlice =0
    testBob =0
    while (len(Alice.get_integr()) < 16):

        claveAlice = OpBB84.rbits(16*4)
        polAlice = OpBB84.rbits(16*4)
        polBob = OpBB84.rbits(16*4)

        claveBob = []

        Alice.anadirClave(claveAlice.numpy())
        Alice.anadirPol(polAlice.numpy())
        Bob.anadirPol(polBob.numpy())

        for i in range(len(claveAlice)):
            claveBob.append(circuitBB84(claveAlice[i],polAlice[i],polBob[i]))
        Bob.anadirClave(tf.squeeze(claveBob).numpy())

        polB = Bob.get_pol()
        polA = Alice.get_pol()

        OpBB84.cribarBits(Alice,polB)
```

```
        OpBB84.cribarBits(Bob, polA)

        testAlice, testBob = OpBB84.testIntegridad(Alice, Bob)

    return(testAlice, testBob)
```

Finalmente, añadiendo la adjudicación de la clave privada:

Listing 8.11: Ciruito básico

```
def BB84(Alice:Person, Bob:Person):

    testAlice,testBob = BB84Test(Bob, Alice)

    if (testAlice & testBob):
        if (OpBB84.genClavePrivada(Bob) and OpBB84.genClavePrivada(Alice)):
            return True

    else:
        print("El canal es inseguro")
        return False
```

8.4. BB84Alt.py

Otra posible implementación del circuito del protocolo BB84 es el intentar tratar con toda la cadena de bits a transmitir a la vez en el circuito. Esto supone que no se podrán utilizar los condicionales como en la implementación anterior, sino que habrá que utilizar las propias cadenas de bits para parametrizar las funciones.

Recordemos que el circuito a simular consta de cuatro partes:

1. Transmitir el valor de la cadena de bits originales al cable cuántico (al qubit).
2. Polarizar los qubits según el valor de los bits generados por Alice P_A.
3. Polarizar los qubits según el valor de los bits generados por Bob P_A.
4. Medición de los qubits por parte de Bob para obtener su cadena

Para las tres primeras necesitaremos entonces crear dos métodos `meterBit` y `polarizarQbit`. mientras, el método `sample(wire:int)` de la librería de PennyLane mide el qubit en el cable *wire* del circuito.

Recordemos que en los circuitos implementados con PennyLane el valor inicial de los qubits es siempre $|0\rangle$. Como se observa en el circuito, hay que implementar una puerta C_{NOT} :

- Si el bit es 0, en la C_{NOT} entra $|00\rangle$ y se transforma en $|00\rangle$, luego sale $|0\rangle$.

- Si el bit es 1, en la C_{NOT} entra $|10\rangle$ y se transforma en $|11\rangle$, luego sale $|1\rangle$.

Como se comentó en la sección 6.1.3, en la librería PennyLane se puede encontrar el método `CNOT(wires)` que implementa dicha puerta. Sin embargo, eso implicaría que el circuito cuántico en el que se simularía el protocolo tuviera que utilizar también canales cuánticos para los bits tradicionales. Aunque sea una solución factible, esta implementación cuadriplicaría los recursos que se necesita para el circuito (el circuito sería de 4 canales cuánticos en vez de solo uno).

Una mejor opción a la hora de realizar la implementación es que se aplique una puerta u otra dependiendo de unos parámetros (que simulen a los bits) pasados al circuito.

Se tiene entonces que buscar una función para que cuando se envíe el valor 0 se aplique la puerta identidad I y cuando se envíe el valor 1 se aplique la puerta Pauli X (cuando el bit vale 1).

- Como ya hemos dicho anteriormente la identidad equivale al caso donde todos los parámetros son nulos. Es es el caso que asociaremos con que el bit valga 0.
- Como queremos hacer una rotación en el plano X, equivale a una rotación de π tanto en y como en Z: $P_X = iR_y(\pi)R_z(\pi)$ (Más luego la multiplicación por el escalar i, que se puede obviar).

Por tanto, como $I = R_y(0)R_z(0)$, podemos considerar la función

$$\begin{aligned} f : \{0,1\} &\longrightarrow C \\ f(x) &= e^{ix\pi/2}R_Y(\pi x)R_Z(\pi x) \end{aligned}$$

Esto implica que el método `meterBit()` debe consistir en aplicar primero la puerta $R_Z(\pi x)$ y luego $R_Y(\pi x)$ (con las funciones de PennyLnae `RY()` y `RZ()`), siendo x el valor del bit a insertar (en ese orden, puesto primero se aplican las puertas "a la derecha").

Listing 8.12: meterBit()

```
def meterBit(bit):

    # Para ello, hay que transformar |0> a
    #     - |0>: hay que aplicar la funcion identidad = RY(0)*RZ(0)
    #     - |1>: hay que aplicar la puerta PauliX = RY(pi)*RY(pi)
    # luego tenemos que aplicar una puerta u otra, en cada caso

    qml.RZ(bit*np.pi ,(0))
    qml.RY(bit*np.pi ,(0))
    qml.Rot(bit*np.pi,0,0,0)
```

De manera similar, se tiene que implementar cómo Alice y Bob polarizan un bit. En el circuito, esto se consigue por las dos *Controlled Hadamard* (C_H) (una en la parte de Alice y otra en la parte de Bob), que son las que cambia la polarización.

Por lo mismo que antes, para no tener que representar en el circuito el cable que envía las polarizaciones, tenemos que encontrar una manera de sustituir la puerta C_H por una parametrización que dependa del bit enviado.

Ya vimos que la puerta Hadamard se puede descomponer en:

$$H = iR_Y(\pi/2)R_Z(\pi)$$

Además, como $I = R_Y(0)\,R_Z(0)$, podemos considerar la función:

$$\begin{aligned} f : \{0,1\} &\longrightarrow C \\ f(x) &= e^{ix\pi/2}R_Y(x\pi/2)R_Z(x\pi) \end{aligned}$$

Que cumple que $f(0) = I$ y $f(1) = H$ (luego asocia los 0s de la cadena con la polarización A y los 1 con la B), así que podemos representar las polarizaciones, construyendo así la función `polarizarBit()`.

Con ambos métodos, ya se puede construir el circuito, que tiene como atributos de entrada dos objetos `Person` que harán el papeles de Alice y Bob respectivamente:

Listing 8.13: circuitBB84Alt()

```
dev = qml.device('default.qubit', wires=1, shots=1)

@qml.qnode(dev)
def circuitBB84(Alice, Bob):

#Parte de Alice
    meterBit(Alice._clave)
    polarizarQbit(Alice._pol)

#Parte de Bob
    polarizarQbit(Bob._pol)
    return (qml.sample(None, (0) ))
```

8.5. BB84Ruido.py

La principal diferencia en la implementación que se debe considerar respecto al caso básico será, obviamente, la implementación del ruido.

En computación cuántica hay dos tipos principales de ruido: el debido al cambio de polarización y el cambio de valor del bit (llamado *bitflip*). La librería de 'pennylane', contiene ya una implementación de dichos ruidos, mediante los métodos `DepolarizingChannel (prob, wires)` y `BitFlip(prob, wires=0)` respectivamente. Esto implica que entre la parte de Alice y la parte de Bob del circuito habrá que añadir ambos métodos. Sin embargo, no es tan sencillo. Estos métodos no son compatibles con el dispositivo básico utilizado para emular el circuito en el caso básico `dafult.qubit`. En cambio, se tendrá que utilizar el dispositivo `default.mixed`. Este tipo también se utilizará para el caso de Eve, aunque no sea necesario.

Listing 8.14: Circuito del protocolo BB84 con ruido en el canal en cadena

```
devMix = qml.device('default.mixed', wires=1, shots=1)

@qml.qnode(devMix)
def circuitBB84Ruido(Alice, Bob, ruido):

#Parte de Alice
    BB84.meterBit(Alice._clave)

    BB84.polarizarQbit(Alice._pol)

# Canal
    qml.DepolarizingChannel(ruido, wires=0)
    qml.BitFlip(ruido, wires=0)

#Parte de Bob
    BB84.polarizarQbit(Bob._pol)

    return (qml.sample(None, (0) ))
```

Listing 8.15: circuitBB84Ruido()

```
devMix = qml.device('default.mixed', wires=1, shots=1)

@qml.qnode(devMix)
def circuitBB84Ruido(bit, AliceP, BobP, ruido, ruido2=None):
    '''
    Circuito cu ntico hecho usando la librer a de pennylane, que ejecuta
    el protocolo BB84

    Alice: Objeto Person que env a la cadena de bit de la que se
    sacara la clave

    Bob: Objeto Person que recibe la cadena

    ruido: probabilidad de que se generere la distorsi n: fuerza del ruido

    Retrun:
    clave de Bob

    '''


    if( ruido2 is None):
        ruido2 = ruido

#Parte de Alice
    if(bit):
        qml.PauliX(0)

    if(AliceP):
        qml.Hadamard(0)
```

```
# Canal
    qml.DepolarizingChannel(ruido, wires=0)
    qml.BitFlip(ruido2, wires=0)

#Parte de Bob
    if(BobP):
        qml.Hadamard(0)

    return (qml.sample(None, (0) ))
```

A partir de esta función, se puede construir de *BB84RuidoTest()* y `BB84Ruido()` de la misma manera que `BB84Test()` y `BB84()`. Por ese motivo, no se mostrarán explícitamente aquí.

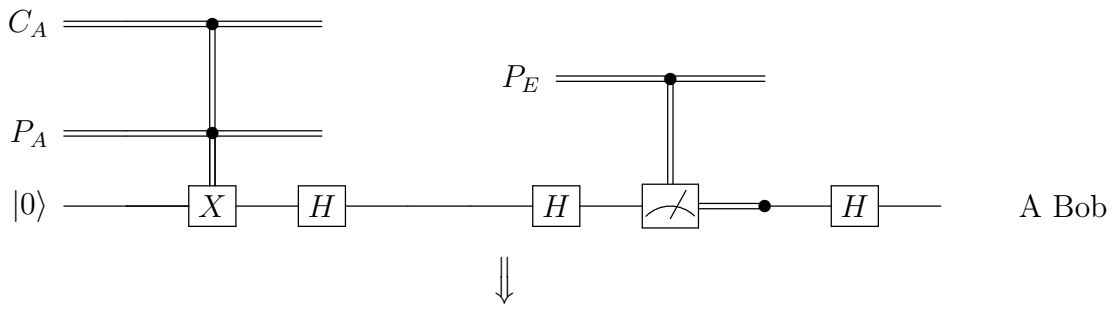
8.6. BB84Eve.py

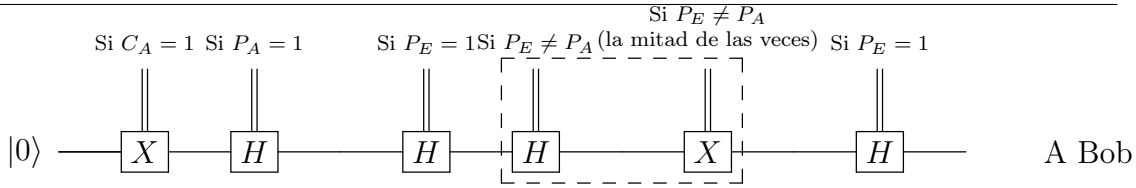
En esta sección, la función más importante es la que consiste en la implementación del efecto sobre el qubit en la medición de Eve. De las implementaciones consideradas, la que se ha implementado es la básica (donde la polarización elegida por Eve es al azar entre A y B).

Para ver cómo implementar esto, analicemos los pasos de cómo Eve interacciona con el qubit:

- Primero, polariza el qubit (puerta Hadamard condicional) según sus polarizaciones.
- A continuación, Eve mide el qubit:
  - En caso de que haya acertado la polarización, el valor del qubit se mide correctamente.
  - Si no es el caso, como se ha medido mal su valor, se polariza el bit (aplicar una puerta Hadamard). Además, la mitad de estas veces se "invertirá" su valor (una puerta de Pauli X la mitad de estas veces).
- Finalmente, se envían los qubits con los valores medidos, y con las polarizaciones de Eve (puerta Hadamard condicional).

Visualmente, el circuito necesario a implementar en PennyLane es:





Simulación de una mala polarización

Pasando este circuito a PennyLane, se obtiene esta implementación que simula el efecto de la medición en el canal:

Listing 8.16: medirQbit().

```
def medirQbit(polEve, polOG):
    cambioBit = BB84.rbits(1)

    if(polEve):
        qml.Hadamard(0)

    if not(polOG==polEve):
        qml.Hadamard(0)
        if(cambioBit):
            qml.PauliX(0)

    if(polEve):
        qml.Hadamard(0)
```

A partir de esta función, solo falta entonces añadirla al circuito general para obtener el circuito para el caso en el que el canal sea escuchado:

Listing 8.17: Circuito para el caso de Eve.

```
@qml.qnode(devMix)
def circuitBB84Eve(bit,AliceP, BobP, polEve):

#Parte de Alice
    if(bit):
        qml.PauliX(0)

    if(AliceP):
        qml.Hadamard(0)

# Canal
    medirQbit(polEve, AliceP)

#Parte de Bob
    if(BobP):
        qml.Hadamard(0)

    return (qml.sample(None, (0) ))
```

De la misma manera que sucedía con `BB84Ruido.py`, a partir de esta función se puede construir de `BB84RuidoEve()` y `BB84Eve()` de la misma manera que se construyeron `BB84Test()` y `BB84()` en `BB84Basic.py`.

8.7. BB84Extra.py

En este archivo se encuentra la implementación de la mejora que se sugiere en los cuadernos para mejorar la efectividad del protocolo en un canal con ruido.

Durante el cuaderno `ImplementacionBB84Complejo.ipynb`, se llega a la conclusión de que un factor clave para diferenciar entre cuándo el canal esta siendo atacado a cuándo el canal solamente se ve afectado con cierto nivel de ruido son tanto el número de intentos necesarios para pasar el test de integridad, como veces en las que se ha obtenido más de 5 fallos en el test de integridad a lo largo de todos los intentos. Esto lleva a encontrar una modificación del protocolo a partir de los valores de esa dupla para el caso a analizar.

Este proceso consiste en, partiendo de los valores de la muestra obtenida anteriormente en el cuaderno, decidir si tras cada intento de pasar el test de integridad, se ha de comprobar si la dupla `(intentos, fallosTotales)` del caso a analizar es suficientemente similar a los datos para el ruido en la muestra, siendo:

`intentos`. Nº de intentos (de realización del protocolo) requeridos hasta ese momento para pasar el test (entre todos los intentos).

`fallosTotales`. Veces que se han obtenido más de 5 errores en el test de integridad

Puesto que para ambos atributos el ruido suele tener valores menores, se le permite a la muestra volver a intentar pasar el test de integridad siempre y cuando ambos atributos sean suficientemente pequeños. Para ello, se considerará "suficientemente ruido" si está más cercano a la media de los valores obtenidos para el ruido, que a los obtenidos para cuando el canal ha sido atacado. Implementar esto es sencillo, puesto que, ya que para ambos datos los valores para el caso Ruido son menores que para el caso Eve, con solo detectar que los valores del caso a analizar son menores a la media de las medias obtenidas (denotadas en el cuaderno por las variables (`tM`, `fM`)), es suficiente para ver si el caso actual se encuentra en la **zona de influencia** de los valores del ruido o no[2].

Además, con ayuda de los archivos anteriormente comentados, se aprovechará para que esta función actúe como un polimorfismo, en el que se aplicará uno u otro circuito según los parámetros transmitidos a la función:

Listing 8.18: GenericBB84Test()

```
def GenericBB84Test(Alice:Person, Bob:Person,polEve=None,
    ruido1=None, ruido2=None):
    '''
    Dependiendo los arguntos enviados, realiza una versi n delprotocolo BB84 u otra
    Parametros:
            Alice: Persona emisora
            Bob: Persona receptora
            polEve: Polizaci n inducida por Eve
```

[2]Para el script, los puntos medios que se han considerado son los generados a partir del conjunto de entrenamiento de la figura 2.5. Esto da con los valores `tM` (media de los tiempos) 45 y `fM` (media de veces con más de 5 fallos) 7.7, respectivamente.

```
            ruido1: Porcentaje de ruido de cambio de bit (BitFlip)
            ruido2: Porcentaje dedepolarizacion
    return

    '''
    if (polEve is None and ruido1 is None and ruido2 is None):
        return not(BB84Test(Alice, Bob))
        # Se invierte su resultado, puesto que devulve True cuando se pasa
        el test a diferencia del resto
    elif(ruido1 is None and ruido2 is None):
        return BB84EveTest(Alice,Bob)
    elif(polEve is None ):
        return BB84RuidoTest(Alice,Bob, ruido1, ruido2)
    else:
        BB84EveRuidoTest(Alice,Bob, polEve, ruido1)
```

Listing 8.19: GenericBB84()

```
def GenericBB84(Alice:Person, Bob:Person,polEve=None,ruido1=None, ruido2=None):
    '''
    Realiza el protoclo BB84 entre Alice y Bob para intentar generar
    una clave Privada entre ellos, con posibilidad de pueda existir
    Eve espiando, o haya cierto ruido en el canal

    Parametros:
        Alice: Persona emisora
        Bob: Persona receptora
        polEve: Polizaci n inducida por Eve
        ruido1: Porcentaje de ruido de cambio de bit (BitFlip)
        ruido2: Porcentaje dedepolarizacion

    Return:
        True si se logr  generar clave privada

    '''

    #Valores empiricos obtenidos de una muestra obtenida
    tM = 45
    fM = 7.7

    intentos = 1
    fallosTotales=0
    pasaTest = False

    timeout=False
    while not(pasaTest):

        testAlice, testBob = GenericBB84Test(Alice,Bob, polEve, ruido1, ruido2)

        if not(testAlice & testBob):
            if (genClavePrivada(Bob) & genClavePrivada(Alice)):
```

```
                pasaTest = True
            else:
                pasaTest = False
                intentos = intentos + 1
        else:
            if testAlice>5:
                fallosTotales = 1 +fallosTotales #Si mas de 6 veces pasa
            intentos = intentos +1
            pasaTest = False

        if( intentos > tM or
            fallosTotales > fM):
            print() #para que no se borre el anterior
            print("Posible atacante")
            timeout= True
            break

    if (not(timeout)):

        msgFinal = cifradoAES(Alice, Bob, "Mensaje Prueba")

        if (msgFinal):
            print("Cifrado correcto")
            return True
        else:
            print("Cifrado no correcto")
            Alice.__init__()
            Bob.__init__()
            return GenericBB84(Alice,Bob, polEve, ruido1, ruido2)

    return False
```

8.8. AES.py

Librería auxiliar para poder cifrar y descifrar con el cifrado AES256. Ver la sección 6.1.1.

8.9. Generación de histogramas

Aunque no sean parte de los archivos `.py`, también es importante comentar el proceso de creación de los histogramas utilizados en los cuaderno.

Para este propósito, se utiliza la función `hist()` de la librería `Matplotlib` (6.1.2). Para la obtención de los datos, se suman la diferencia entre los elementos de los array `Alice._integr` y `Bob._integr` en valor absoluto (que será 1 cuando difieran y 0 cuando coincidan) para cada una de las iteraciones, obteniendo finalmente el array `dif`.

Listing 8.20: Obtención de estadísticas del test de integridad

```
pasa=0
noPasa=0
dif = []
for i in range(1000):

    #Reiniciar las Personas
    Alice = p.Person()
    Bob = p.Person()
    claveAlice = BB84.rbits(16*4)
    polAlice = BB84.rbits(16*4)

    polBob = BB84.rbits(16*4)

    Alice.anadirClave(claveAlice.numpy() )
    Alice.anadirPol(polAlice.numpy())
    Bob.anadirPol(polBob.numpy())

    prob= np.random.random(16*4)
    claveBob = circuitBB84Ruido(Alice,Bob ,.15)
    Bob.anadirClave(claveBob.numpy())

    testAlice, testBob = testIntegridad(Alice, Bob)

    diferencia = np.sum(abs(np.subtract(Alice._integr, Bob._integr)) )
    dif.append( diferencia )

    if (testAlice & testBob):
        Alice._clavePriv,cerosA = BB84.array2string16(Alice._claveFinal[16:])
        Bob._clavePriv,cerosB = BB84.array2string16( Bob._claveFinal[16:])
        pasa=pasa+1
    else:
        noPasa=noPasa+1
```

En realidad, por cómo se ha construido y por cómo funcionan los tensores, `dif` es una matriz 1×500. Para convertirlo en array unidimensional, se necesita utilizar el método `squeeze()` de TensorFlow. una vez preparado el array `dif`, ya se puede crear el historial mediante el método `pyplot.hist()` de `matplotlib`.

Listing 8.21: Creación de histograma.

```
import tensorflow as tf
import matplotlib.pyplot as plt
from pennylane import numpy as np

dif=tf.squeeze(dif)
his =plt.hist(x=dif, bins=np.max(dif)-1 ,rwidth=1., color='#0504aa')
```

Para el mapeado de la sección final, se sigue un proceso similar, utilizando la función `scatter()` para los puntos y la función `axline()` para las líneas.

Capítulo 9

Pruebas

Este capítulo es muy importante, y muestra las pruebas que se han realizado para demostrar el funcionamiento del sistema.

Para las pruebas unitarias sobre las funciones de los circuitos de los cuadernos, basta con modificarlas para que los circuitos devuelvan `state()` en vez de `sample()`, y así observar el estado cuántico final (determinista) en vez de uno aleatorio (el valor medido).

Por ejemplo, para el circuito BB84, el estado final que mide Bob tiene siempre que ser o $(1, 0) \rightarrow 0$, o $(1, 0) \rightarrow 1$, o $(1/\sqrt{2}, 1/\sqrt{2}) \rightarrow$ mide 0 o 1 (polarización incorrecta).

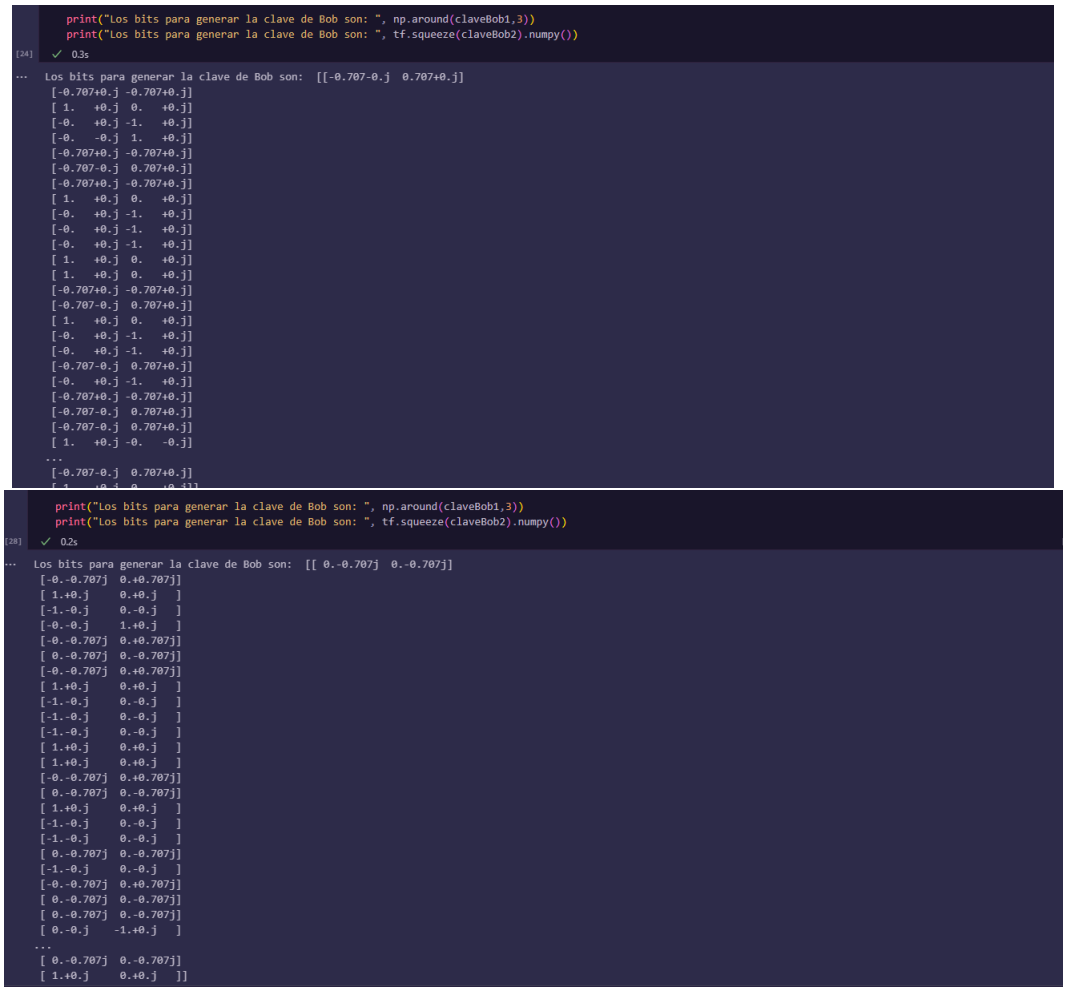


Figura 9.1: Utilización del método `state()` para la creación de pruebas de caja negra para el método `circuitBB84Alt()`. A la izquierda, la función está programa correctamente, deducido de que los estados devueltos sea los indicados. A la derecha, en vez de devolver $(1/\sqrt{2}, 1/\sqrt{2})$ devuelve $(1/\sqrt{2}, 1/\sqrt{2})i$, lo que implica que no se ha realizado una implementación correcta.

En cuanto a otras pruebas de caja negra para el resto de funciones, cabe destacar el comprobar que se se ha hecho correctamente la división de los bits de `claveFinal`, corroborando que `claveFinal[32:]` coincide con `tf.concat([clavePriv,integr], 0)` (sin considerar `clavePriv` como String todavía).

Para las pruebas de caja blanca, los propios cuadernos se pueden considerar como una prueba de caja blanca y de integración, ya que durante los mismos se va implementando cada una de las partes que componen el protocolo observando los resultando que se obtienen; además de comprobar de que efectivamente la aplicación del protocolo cumple su cometido.

Aunque no sea una comprobación propiamente dicha, una comprobación de la correcta implementación del protocolo ha sido comprobar que para mismos datos de entrada ambas implementaciones obtienen la misma clave privada (no se obtienen exactamente lo mismo por la aleatoriedad inherente del protocolo), de la misma manera que se hace dentro de

los cuadernos con las implementaciones de `BB84Basic.py` y `BB84Alt.py`.

Finalmente, cabe destacar el uso del Debugger de Visual Studio, que ha sido clave para la detección de errores y depuración del código.

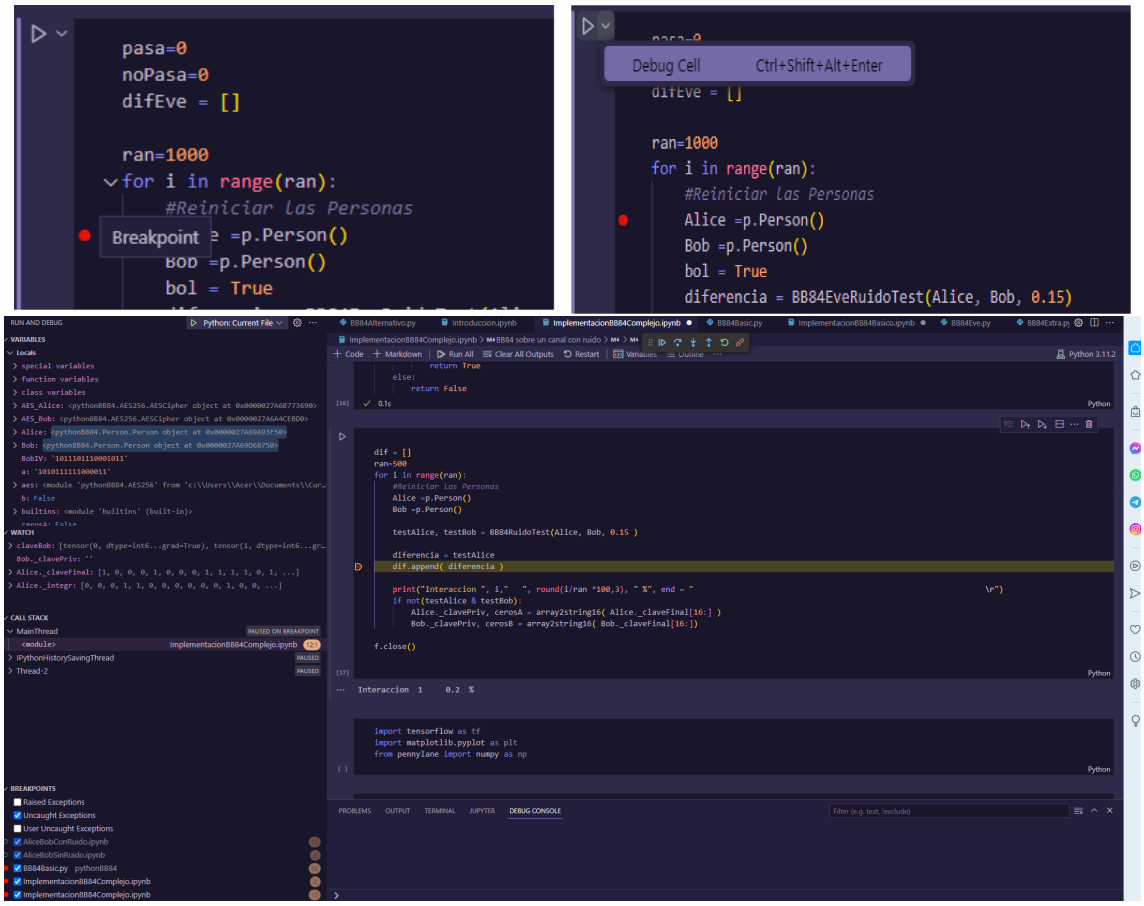


Figura 9.2: Utilización del Debugger.

Los pasos para la utilización del Debbuger

1. Marcar las líneas de código donde se quiere que se pare la ejecución.

2. Ejecutar la celda que se quiera comprobar la ejecución (los *Breakpoints* no tienen por qué estar en la celda que se ejecuta, pueden estar en código que es llamado en la celda).

3. Analizar el valor de las variables en ese momento de la ejecución. En el apartado **WATCH** se pueden añadir variables específicas que se quieran visualizar.

4. Se puede avanzar a través del código a partir de los controles situados arriba en el centro.

Parte III

Manuales de la Aplicación

Capítulo 10

Manuales

10.1. Manual de Instalación

Lo único necesario apara poder visualizar correctamente un cuaderno Jupyter en un dispositivo es tener la librería de Jupyther Notebook instalada -lo que implica tener instalado *Python*-, y un navegador web.

Puesto que, al fin y al cabo, Jupyter es una librería de *Python*, lo primero será instalar *Python* en el sistema en el que se vaya a utilizar. En caso de solo querer leer el cuaderno, por lo explicado en 6.2. Sin embargo, en caso de querer poder compilar personalmente las celdas, será necesarias también las librerías y paquetes utilizados en el código. La instalación de *Python* y de paquetes puede realizarse de dos maneras:

Instalación Independiente. Una manera de instalar *Python* en un dispositivo es simplemente instalando el ejecutable propio de *Python* para ello, localizado en `www.python.org/downloads`. Cualquier versión de *python3* o superior es suficiente para los cuadernos del proyecto.

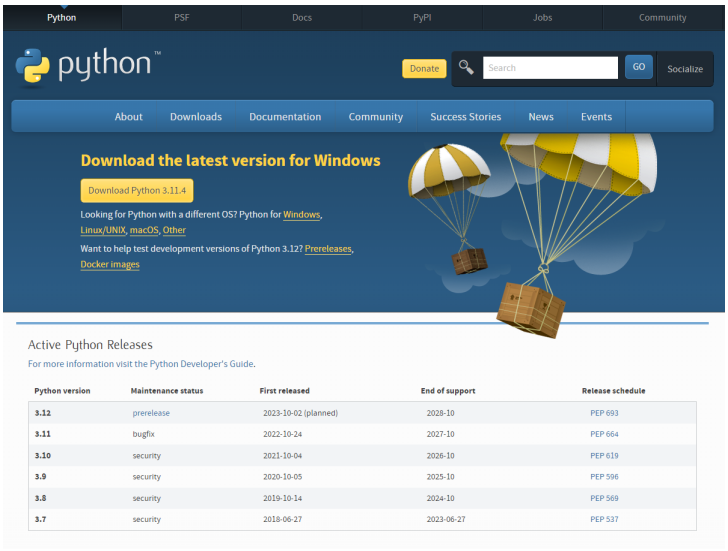


Figura 10.1: Página de descarga del ejecutable de *Python*.

Tras ejecutarlo (eligiendo ”añadir a la variable PATH” en caso de encontrarse en Windows), se puede comprobar que se ha instalado correctamente viendo que se ejecuta correctamente el comando

```
Python - -version
```

Una vez comprobado que se ha instalado *Python* correctamente, se tienen que instalar los paquetes. Esto se puede hacer mediente el comando `pip` en la consola de comandos. En el caso concreto de Jupyter Notebook, es

```
pip install notebook
```

Una vez instalado, para iniciar Jupyter se tiene que escribir en la consola de comandos

```
jupyter notebook
```

Instalación por Anaconda. Una manera alternativa de instalar *Python* y los paquetes es instalando la distribución de trabajo Anaconda. Anaconda es una distribución de trabajo, centrada en *machine learning* y en el análisis de datos, para Python y R, que permite instalar y gestionar los paquetes de manera cómoda y sencilla, además de que la instalación del propio Anconda incluye la instalación de *Python* en su distribución.

Pese a la comodidad que ofrece trabjar sobre Anaconda, también tiene sus desventajas. Puesto que Anaconda es una distribución aislada, crea un entorno de desarrollo local, pudiendo no usarse ni *Python* ni los paquetes instalados fuera de él. Además, suelen surgir problemas si existe una distribución de *Python* a nivel global en el sistema. Más información sobre este proceso de instalación se puede encontrar en `https://docs.anaconda.com`.

10.2. Manual de Uso

Se recomienda que contenga la información relevante, lo más completa y clara posible, como si fuese el manual de un producto comercial, dirigido a un usuario no avanzado. Esta documentación pudiera coincidir (de hecho, se recomienda) con la AYUDA del programa.

Dependiendo de la instalación elegida, se abre la interfaz de jupyter de una manera distinta:

- Si se ha realizado la instalación por la consola de comandos, basta con ejecutar en `cmd` (o `PowerShell` o similares) el comando

  ```
  jupyter notebook
  ```

  Para se abra directamente en un directorio determinado, se debe añadir su ruta (o ruta relativa) al comando:

  ```
  jupyter notebook .\Documents\Cuadernos
  ```

- En Anaconda, basta con abrir el entorno de Jupyter:

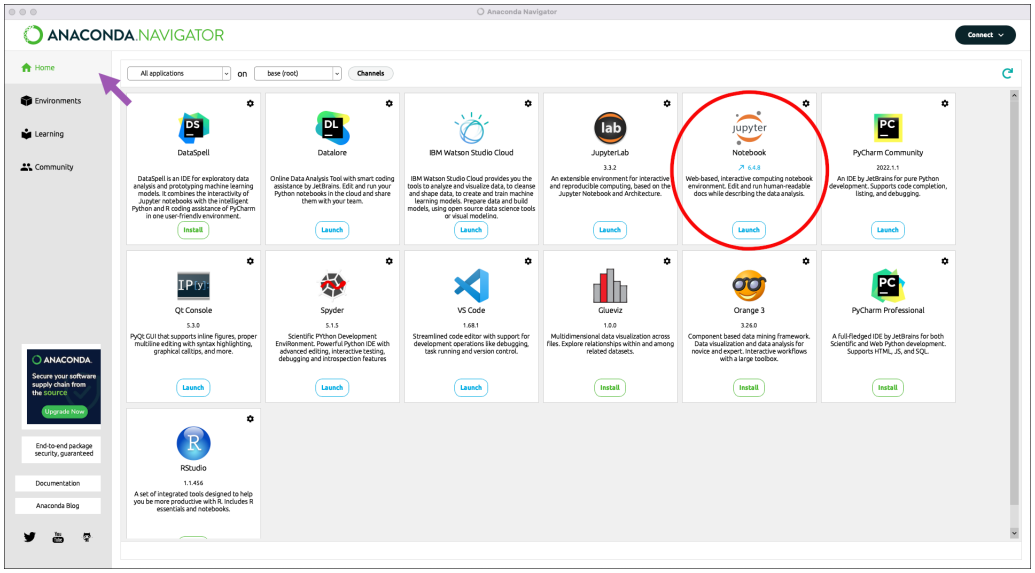


Figura 10.2: Abrir Jupyter en Anaconda

Una vez abierto, hay que ejecutar el cuaderno deseado. En este proyecto, el cuaderno inicial, y por el que se recomienda comenzar, es `Introduccion.ipynb`, aunque puede seleccionarse cualquiera si así se desea.

La barra superior contiene herramientas para la creación y compilación de nuevas celdas:

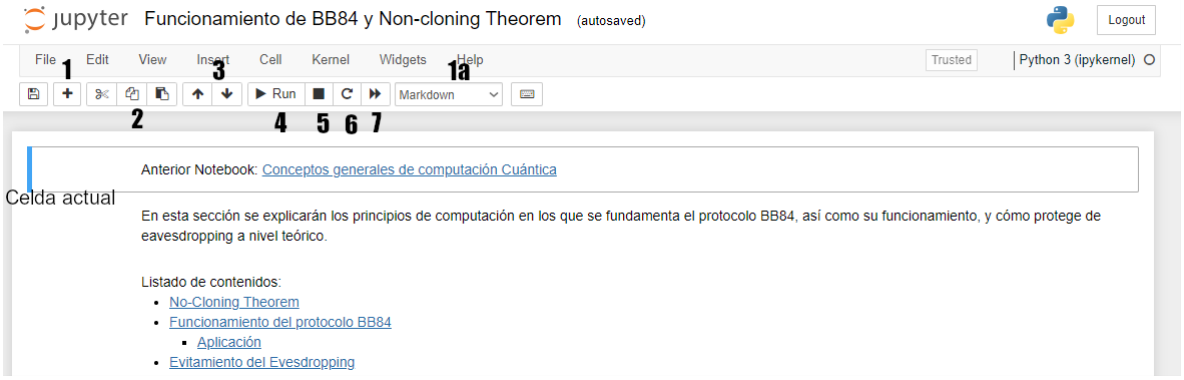


Figura 10.3: Barra de herramientas de los cuadernos Jupyter

1 Crea una nueva celda, compilada según lo indicado en **1a**.

2 Herramientas de copiado y pegado.

3 Mueve la ubicación de las celdas.

4 Ejecución de la celda seleccionada.

5 Parada de la ejecución de la celda

6 Reiniciar el kernel.

7 Ejecutar todo el cuaderno desde el inicio.

Por otra parte, la primera celda de cada cuaderno contiene las importaciones que se necesitan para dicho cuaderno. Si se quiere ejecutar una celda en específica, será necesario primero ejecutar esta celda.

Parte IV

Apéndices

Apéndice A

Anexos

A.1. Información complementaria

Si se quiere profundizar sobre el uso de la librería de `PennyLane`, se sugiere consultar su documentación en su página web (`https://pennylane.ai/qml/documentation`), además de varios ejemplos de aplicación, accesibles desde la dirección `https://pennylane.ai/qml/demonstrations`.

Como ya se ha comentado en la memoria, para profundizar sobre el funcionamiento del protocolo BB84, entre las fuentes bibliográficas se encuentran varias opciones. El libro *Quantum Comuptation and Quatum Information[14]* sirve para ahondar sobre la computación cuántica en general, siendo su mayor problema el ser demasiado técnico. Un enfoque más práctico es dado en el artículo *Quantum cryptography: Public key distribution and coin tossing*[2], donde su explicación es mucho más comprensible.

A.2. Diagramas y tablas

A.2.1. Estimaciones temporales del proyecto

Cuadro A.1: Estimaciones obtenidas mediante el método PERT

Recurso	Valor total	Tipo de pago	Coste estimado en el proyecto		Coste real	
			Tiempo de uso	Coste	Tiempo de uso	Coste
LucidChart	27,00 €	mensual	5,533730159	149,41 €	7,178571429	193,82 €
Microsoft Office (Excel)	300,00 €	por dispositivo	1	300,00 €	1	300,00 €
MS Project	34,00 €	mensual	5,533730159	188,15 €	7,178571429	244,07 €
Milanote	- €	-		- €		- €
Overleaf	179,00 €	anual	0,460833333	82,49 €	0,59825	107,09 €
Visual Studio	- €	-		- €		- €
GitHub	- €	-		- €		- €
				720,05 €		**844,98 €**

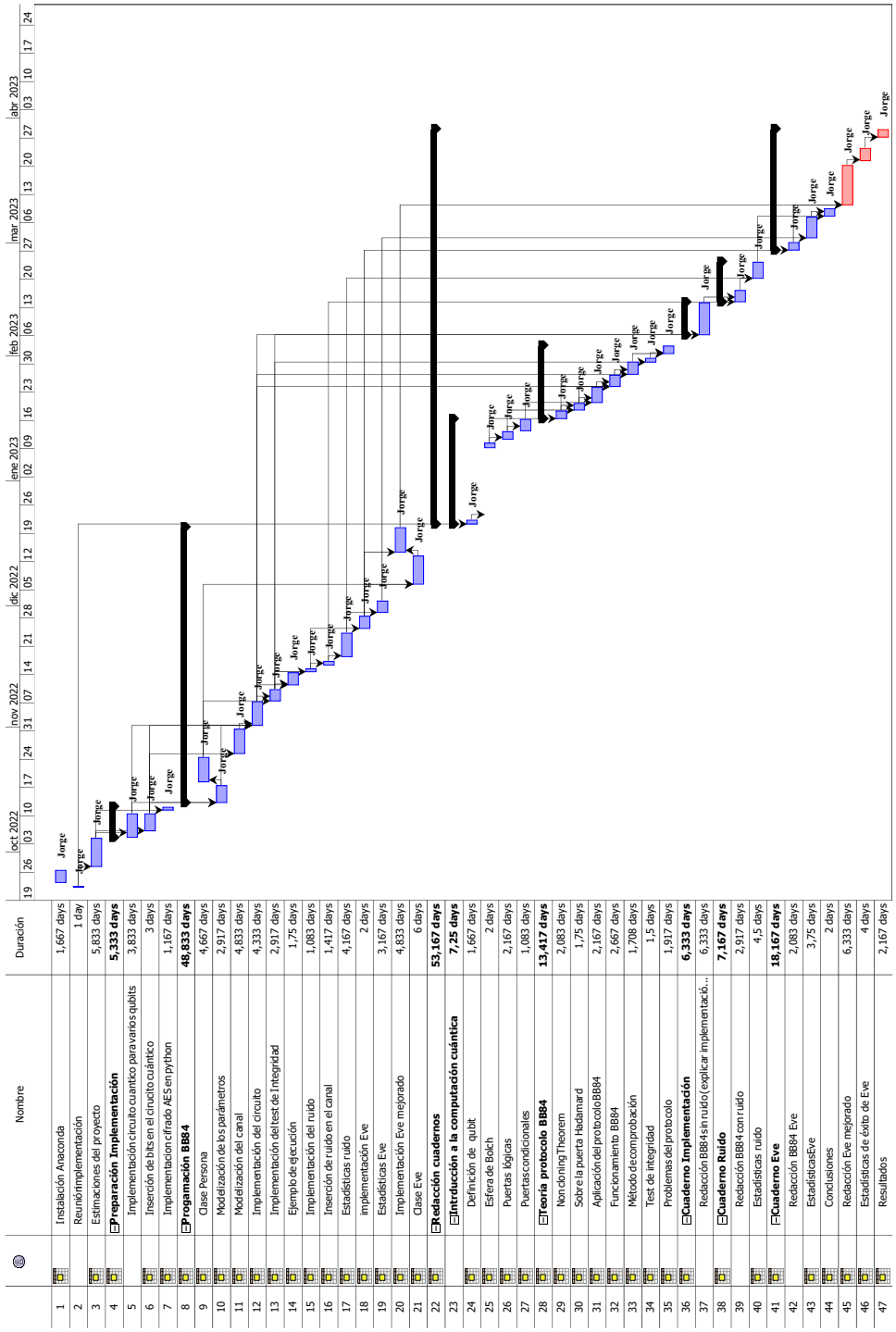
Cuadro A.2: División de la estimación en roles

	Tiempo estimado	Jefe de Proyecto		Programador		Investigador	
Instalación Anaconda	1,67	0%	0	100%	1,67	0%	0
Reunión implementación	1	100%	1	0%	0	0%	0
Estimaciones del proyecto	5,83	100%	5,83	0%	0	0%	0
Reinstalación Python	-	-	-	-		-	
Preparación Implementación							
Implementación circuito cuantico para varios qubits	3,833333333	0%	0	100%	3,833333333	0%	0
Inserción de bits en el circucito cuántico	3	0%	0	100%	3	0%	0
Implementacion cifrado AES en python	1,166666667	0%	0	100%	1,166666667	0%	0
Progamación BB84							
Clase Persona	4,666666667	0%	0	50%	2,333333333	50%	2,333333333
Modelización de los parámetros	2,916666667	33%	0,9625	33%	0,9625	33%	0,9625
Modelización del canal	4,833333333	0%	0	50%	2,416666667	50%	2,416666667
Implementación del circuito	4,333333333	0%	0	50%	2,166666667	50%	2,166666667
Implementación del test de Integridad	2,916666667	0%	0	50%	1,458333333	50%	1,458333333
Ejemplo de ejecución	1,75	0%	0	100%	1,75	0%	0
Implementación del ruido	1,083333333	0%	0	50%	0,541666667	50%	0,541666667
Inserción de ruido en el canal	1,416666667	0%	0	100%	1,416666667	0%	0
Estadísticas ruido	4,166666667	0%	0	100%	4,166666667	0%	0
implementación Eve	2,833333333	0%	0	50%	1,416666667	50%	1,416666667
Estadísticas Eve	3,166666667	0%	0	100%	3,166666667	0%	0
Implementación Eve mejorado	4,833333333	0%	0	50%	2,416666667	50%	2,416666667
Clase Eve	6	0%	0	100%	6	0%	0
Redacción cuadernos							
Intrducción a la computación cuántica							
Definición de qubit	1,666666667	0%	0	0%	0	100%	1,666666667
Esfera de Bolch	2	0%	0	0%	0	100%	2
Puertas lógicas	2,166666667	0%	0	0%	0	100%	2,166666667
Puertas condicionales	1,083333333	0%	0	0%	0	100%	1,083333333
Teoría protocolo BB84							
Non cloning Theorem	2,083333333	0%	0	0%	0	100%	2,083333333
Sobre la puerta Hadamard	1,75	0%	0	0%	0	100%	1,75
Aplicación del protocolo BB84	2,166666667	0%	0	0%	0	100%	2,166666667
Funcionamiento BB84	2,666666667	0%	0	50%	1,333333333	50%	1,333333333
Método de comprobación	1,708333333	0%	0	50%	0,854166667	50%	0,854166667
Test de integridad	1,5	0%	0	50%	0,75	50%	0,75
Problemas del protocolo	1,916666667	0%	0	0%	0	100%	1,916666667
Cuaderno Implementación							
Redacción BB84 sin ruido (explicar implementación hecha)	6,333333333	0%	0	0%	0	100%	6,333333333
Cuaderno Ruido							
Redacción BB84 con ruido	2,916666667	0%	0	0%	0	100%	2,916666667
Estadísticas ruido	4,5	0%	0	0%	0	100%	4,5
Cuaderno Eve							
Redacción BB84 Eve	2,083333333	0%	0	0%	0	100%	2,083333333
Estadísticas Eve	3,75	0%	0	0%	0	100%	3,75
Conclusiones	2	0%	0	0%	0	100%	2
Redacción Eve mejorado	6,333333333	0%	0	0%	0	100%	6,333333333
Estadísticas de éxito de Eve	4	0%	0	0%	0	100%	4
Resultados	2,166666667	0%	0	0%	0	100%	2,166666667
	TOTAL	14,33		42,82		63,4	
	en horas	**57,32**		**171,28**		**253,6**	

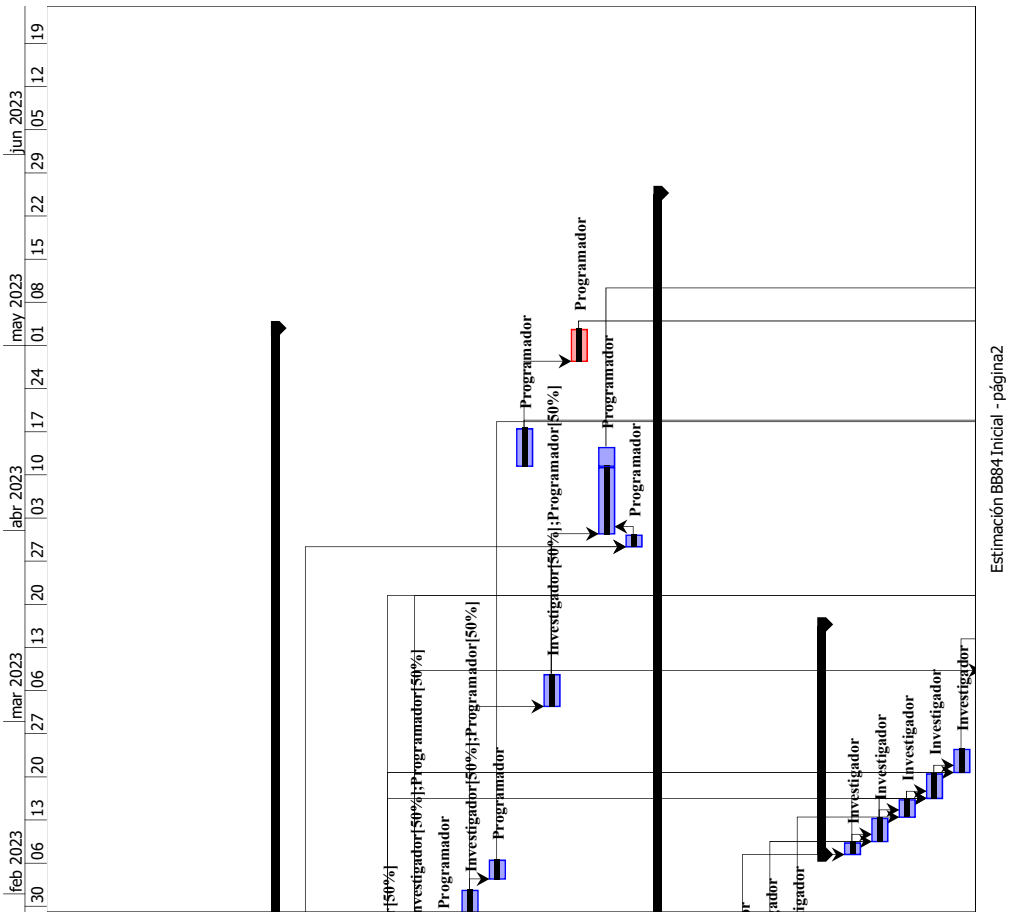
Cuadro A.3: División del tiempo utilizado en roles

	Tiempo real	Jefe de Proyecto		Programador		Investigador	
Instalación Anaconda	3	0%	0	100%	3	0%	0
Reunión implementación	1	100%	1	0%	0	0%	0
Estimaciones del proyecto	6	100%	6	0%	0	0%	0
Reinstalación Python	2	50%	1	50%	1	0%	0
Preparación Implementación							
Implementación circuito cuantico para varios qubits	7	0%	0	100%	7	0%	0
Inserción de bits en el cirucito cuántico	8	0%	0	100%	8	0%	0
Implementacion cifrado AES en python	3	0%	0	100%	3	0%	0
Progamación BB84							
Clase Persona	15	0%	0	50%	7,5	50%	7,5
Modelización de los parámetros	6	33%	1,98	33%	1,98	33%	1,98
Modelización del canal	8	0%	0	50%	4	50%	4
Implementación del circuito	7	0%	0	50%	3,5	50%	3,5
Implementación del test de Integridad	3	0%	0	50%	1,5	50%	1,5
Ejemplo de ejecución	2	0%	0	100%	2	0%	0
Implementación del ruido	4	0%	0	50%	2	50%	2
Inserción de ruido en el canal	2	0%	0	100%	2	0%	0
Estadísticas ruido	5	0%	0	100%	5	0%	0
implementación Eve	5	0%	0	50%	2,5	50%	2,5
Estadísticas Eve	4	0%	0	100%	4	0%	0
Implementación Eve mejorado	4,75	0%	0	50%	2,375	50%	2,375
Clase Eve	4	0%	0	100%	4	0%	0
Redacción cuadernos							
Intrducción a la computación cuántica							
Definición de qubit	3	0%	0	0%	0	100%	3
Esfera de Bolch	4	0%	0	0%	0	100%	4
Puertas lógicas	2	0%	0	0%	0	100%	2
Puertas condicionales	2	0%	0	0%	0	100%	2
Teoría protocolo BB84							
Non cloning Theorem	2	0%	0	0%	0	100%	2
Sobre la puerta Hadamard	2	0%	0	0%	0	100%	2
Aplicación del protocolo BB84	3	0%	0	0%	0	100%	3
Funcionamiento BB84	2,5	0%	0	50%	1,25	50%	1,25
Método de comprobación	3	0%	0	50%	1,5	50%	1,5
Test de integridad	2	0%	0	50%	1	50%	1
Problemas del protocolo	3	0%	0	0%	0	100%	3
Cuaderno Implementación							
Redacción BB84 sin ruido (explicar implementación hecha)	6	0%	0	0%	0	100%	6
Cuaderno Ruido							
Redacción BB84 con ruido	4	0%	0	0%	0	100%	4
Estadísticas ruido	4,5	0%	0	0%	0	100%	4,5
Cuaderno Eve							
Redacción BB84 Eve	2	0%	0	0%	0	100%	2
Estadísticas Eve	4	0%	0	0%	0	100%	4
Conclusiones	2	0%	0	0%	0	100%	2
Redacción Eve mejorado	-	0%	-	0%	-	100%	-
Estadísticas de éxito de Eve	-	0%	-	0%	-	100%	-
Resultados	-	0%	-	0%	-	100%	-
	TOTAL	14,83		68,105		72,605	
	en horas	**59,32**		**272,42**		**290,42**	

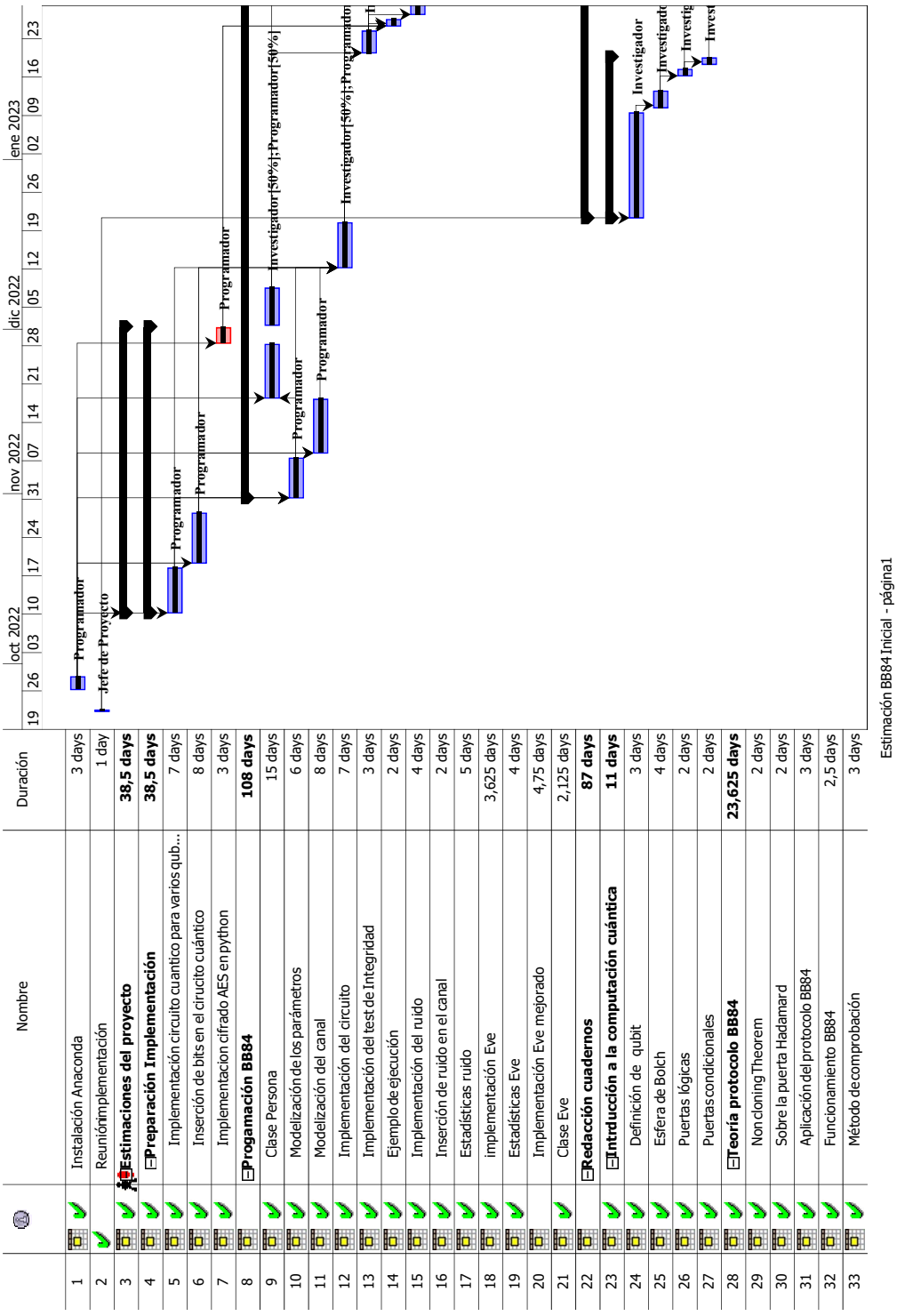
A.2.2. Diagramas de Gantt



Cuadro A.4: Diagrama de Gantt de la estimación



Cuadro A.5: Diagrama de Gantt final



		Nombre	Duración
1		Instalación Anaconda	3 days
2		Reunión implementación	1 day
3		**Estimaciones del proyecto**	**38,5 days**
4		**Preparación Implementación**	**38,5 days**
5		Implementación circuito cuantico para varios qub...	7 days
6		Inserción de bits en el circuito cuántico	8 days
7		Implementacion cifrado AES en python	3 days
8		**Progamación BB84**	**108 days**
9		Clase Persona	15 days
10		Modelización de los parámetros	6 days
11		Modelización del canal	8 days
12		Implementación del circuito	7 days
13		Implementación del test de Integridad	3 days
14		Ejemplo de ejecución	2 days
15		Implementación del ruido	4 days
16		Inserción de ruido en el canal	2 days
17		Estadísticas ruido	5 days
18		implementación Eve	3,625 days
19		Estadísticas Eve	4 days
20		Implementación Eve mejorado	4,75 days
21		Clase Eve	2,125 days
22		**Redacción cuadernos**	**87 days**
23		**Intrducción a la computación cuántica**	**11 days**
24		Definición de qubit	3 days
25		Esfera de Bolch	4 days
26		Puertas lógicas	2 days
27		Puertas condicionales	2 days
28		**Teoría protocolo BB84**	**23,625 days**
29		Non cloning Theorem	2 days
30		Sobre la puerta Hadamard	2 days
31		Aplicación del protocolo BB84	3 days
32		Funcionamiento BB84	2,5 days
33		Método decomprobación	3 days

Cuadro A.6: Diagrama de Gantt final

Apéndice B

Contenido del entregable

El entregable se compondrá de los cuadernos que componen el proyecto, así como de los scripts (guardados en la carpeta `pythonBB84`) e imágenes (guardados en la carpeta `Imagenes`) necesarios para compilarles.

Para abrir los cuadernos, se hará como cualquier otro cuaderno *Jupyter*, como se ha indicado en los manuales de uso (sección 10.2).

Bibliografía

[1] URL: https://zeromq.org/get-started/.

[2] Charles H. Bennett y Gilles Brassard. "Quantum cryptography: Public key distribution and coin tossing". En: *Theoretical Computer Science* 560 (2014). Theoretical Aspects of Quantum Cryptography – celebrating 30 years of BB84, págs. 7-11. ISSN: 0304-3975. DOI: https://doi.org/10.1016/j.tcs.2014.05.025. URL: https://www.sciencedirect.com/science/article/pii/S0304397514004241.

[3] Ville Bergholm et al. *PennyLane: Automatic differentiation of hybrid quantum-classical computations*. 2022. arXiv: 1811.04968 [quant-ph].

[4] P. Bourque y R.E. Fairley. *Guide to the Software Engineering Body of Knowledge, Version 3.0*. ACM-IEEE. 2014.

[5] MathJax Consortium. *Mathjax*. URL: https://www.mathjax.org/.

[6] *Cryptodome documentation*. PyCryptodome. Mayo 2023. URL: https://www.pycryptodome.org.

[7] Bryan Eastin y Steven T. Flammia. *Q-circuit Tutorial*. 2004. eprint: arXiv:quant-ph/0406003.

[8] Charles R. Harris et al. "Array programming with NumPy". En: *Nature* 585 (2020), 357–362. DOI: 10.1038/s41586-020-2649-2.

[9] J. D. Hunter. "Matplotlib: A 2D graphics environment". En: *Computing in Science & Engineering* 9.3 (2007), págs. 90-95. DOI: 10.1109/MCSE.2007.55.

[10] Anil K. Jain. "Data clustering: 50 years beyond K-means". En: *Pattern Recognition Letters* 31.8 (2010). Award winning papers from the 19th International Conference on Pattern Recognition (ICPR), págs. 651-666. ISSN: 0167-8655. DOI: https://doi.org/10.1016/j.patrec.2009.09.011. URL: https://www.sciencedirect.com/science/article/pii/S0167865509002323.

[11] J.M.S. "IBM presenta el primer ordenador cuántico comercial: ¿puedo tener ya un «súper-PC» en casa?" En: *ABC* (10 de ene. de 2019). URL: https://www.abc.es/tecnologia/informatica/hardware/abci-presenta-primer-ordenador-cuantico-comercial-puedo-tener-super-pc-casa-201901101757_noticia (visitado 29-10-2019).

[12] Sara Sans Josep Corbella. "Barcelona liderará el desarrollo de un ordenador cuántico". En: *La Vanguardia* (29 de oct. de 2021). URL: https://https://www.lavanguardia.com/local/barcelona/20211029/7825701/barcelona-liderara-desarrollo-ordenador-cuantico.html (visitado 29-10-2021).

[13] Thomas Kluyver et al. "Jupyter Notebooks – a publishing format for reproducible computational workflows". En: *Positioning and Power in Academic Publishing: Players, Agents and Agendas*. Ed. por F. Loizides y B. Schmidt. IOS Press. 2016, págs. 87 -90.

[14] Michael A. Nielsen e Isaac L. Chang. *Quantum Comuptation and Quatum Information*. Cambridge, 2010.

[15] R. Pressman. *Ingeniería del Software*. McGraw-Hill, 2014.

[16] Adi Shamir. "How to share a secret". En: *Communications of the ACM* 22 (1979).