Universidad Politécnica
de Madrid

Escuela Técnica Superior de
Ingenieros Informáticos

Grado en Ingeniería Informática

Trabajo Fin de Grado

Diseño e implementación de módulos de seguridad para la red MadQCI

Autor: Axel Abadías Fernández

Tutor: Juan Pedro Brito Méndez

Madrid, Junio 2025

Este Trabajo Fin de Grado se ha depositado en la ETSI Informáticos de la Universidad Politécnica de Madrid para su defensa.

Trabajo Fin de Grado
Grado en Ingeniería Informática
Título: Diseño e implementación de módulos de seguridad para la red MadQCI
Junio 2025

Autor: Axel Abadías Fernández

Tutor:
Juan Pedro Brito Méndez
"LENGUAJES Y SISTEMAS INFORMÁTICOS E INGENIERÍA DE SOFTWARE"
ETSI Informáticos
Universidad Politécnica de Madrid

Resumen

En un futuro, cuando las computadoras cuánticas, aunque no estén disponibles para el público, lleguen a ser una realidad, los atacantes podrán desencriptar la información de las redes fácilmente debido a la capacidad de los ordenadores cuánticos de resolver ciertos problemas con una rapidez inimaginable. Esto podrá causar grandes pérdidas económicas y filtraciones masivas de datos de millones de empresas y personas. Las computadoras logran resolver determinados problemas mucho más eficientemente que los sistemas clásicos, y eso impacta directamente en la seguridad. La tecnología de **Distribución Cuántica de Claves** (**QKD**, sus siglas en inglés), podría considerarse una de las soluciones más prometedoras para garantizar la seguridad de las comunicaciones en una época post-cuántica. Existen otras soluciones como PQC, pero este trabajo se centrará exclusivamente en QKD. Sin embargo, las implementaciones reales de QKD no están exentas de posibles vulnerabilidades, como ataques de canal lateral, denegaciones de servicio o intervenciones en el canal cuántico, que pueden comprometer tanto la integridad como la disponibilidad del sistema. La infraestructura clásica de la arquitectura QKD también es vulnerable, como cualquier otra infraestructura clásica, por ello es tan importante securizarla. Estos nodos que son vulnerables a ataques físicos pueden explotarse con ataques clásicos de hacking.

El objetivo de este trabajo es estudiar la seguridad de las redes QKD desde un enfoque integral que combine la evaluación de vulnerabilidades en los nodos físicos mediante herramientas de análisis de red y pentesting, con el desarrollo de un sistema capaz de detectar intrusiones en el canal cuántico y redirigir automáticamente el tráfico por una ruta segura. Para ello se ha diseñado un entorno simulado para comprobar la eficacia de las herramientas. Estas dos herramientas se implementarán dentro de un pipeline de detección y contramedidas donde se aprovecharán ambas soluciones conjuntamente para obtener una mejora significativa de la seguridad en la red.

Los resultados obtenidos muestran que aplicar las técnicas tradicionales de la ciberseguridad sobre los nodos QKD permite reducir significativamente la superficie de ataque, y que es posible desarrollar un sistema capaz de asegurar disponibilidad en las redes QKD incluso en caso de ataques al sistema. Este enfoque híbrido demuestra la viabilidad de aplicar mecanismos de defensa clásicos y dinámicos en un entorno cuántico, y de esta manera, sienta las bases para futuras redes QKD más robustas, adaptativas y seguras.

Abstract

In the future, when quantum computers, even if not available to the public, become a reality, attackers will be able to decrypt network information easily due to the ability of quantum computers to solve certain problems at unimaginable speeds. This could lead to major economic losses and massive data breaches affecting millions of companies and individuals. Quantum computers are capable of solving specific problems much more efficiently than classical systems, and this directly impacts security. **Quantum Key Distribution** (**QKD**) technology could be considered one of the most promising solutions to ensure secure communications in a post-quantum era. Other solutions such as PQC also exist, but this work will focus exclusively on QKD. However, real-world QKD implementations are not free from potential vulnerabilities, such as side-channel attacks, denial-of-service attacks, or interventions in the quantum channel, which can compromise both the integrity and availability of the system. The classical infrastructure of the QKD architecture is also vulnerable, just like any other classical infrastructure, which is why securing it is so important. These nodes, which are vulnerable to physical attacks, can be exploited using classical hacking techniques.

The objective of this work is to study the security of QKD networks from a comprehensive approach that combines the evaluation of vulnerabilities in physical nodes using network analysis and pentesting tools, with the development of a system capable of detecting intrusions in the quantum channel and automatically redirecting traffic through a secure route. For this purpose, a simulated environment has been designed to test the effectiveness of these tools. Both tools will be implemented within a detection and countermeasure pipeline, where both solutions will be used together to achieve a significant improvement in network security.

The results obtained show that applying traditional cybersecurity techniques to QKD nodes significantly reduces the attack surface, and that it is possible to develop a system capable of ensuring availability in QKD networks even in the event of attacks on the system. This hybrid approach demonstrates the feasibility of applying classical and dynamic defense mechanisms in a quantum environment, thereby laying the groundwork for future QKD networks that are more robust, adaptive, and secure.

Tabla de contenidos

Introducción **1**
1.1 Motivación y contexto 1
1.1.1 Limitaciones físicas y necesidad de nodos de confianza 2
1.2 Estructura de una red QKD 2
1.3 Objetivos del proyecto 3
1.4 Solución propuesta 4
1.5 Estructura del documento 5
Estado del Arte **6**
2.1 Cómo funciona los dispositivos QKD 7
2.2 Seguridad Cuántica 8
2.3 Redes QKD Actuales 9
2.3.1 Integración con infraestructuras existentes 9
2.4 Panorama de amenazas 10
2.4.1 Ataques al Canal Cuántico (Eavesdropping e Interferencia) 10
2.4.1.1 Ataque de división de fotones (PNS): 10
2.4.1.2 Ataque Man-in-the-Middle (MitM): 11
2.4.1.3 Ataques de Interferencia (Jamming): 11
2.4.2 Ataques a los nodos de confianza “Trusted Nodes” (Dispositivos Alice y Bob 11
2.4.2.1 Ataques a los detectores (Bob) 12
2.4.2.2 Ataques de Trojan-horse al emisor (Alice) 12
2.4.2.3 Canales laterales de detectores 12
2.4.2.4 Daño láser 12
2.4.3 Vulnerabilidades en el canal clásico de QKD 13
2.4.3.1 Suplantación o interceptación (MitM clásico) 13
2.4.3.2 Denegación de Servicio (DoS) 13
2.4.3.3 Filtración de información en reconciliación 13
2.4.3.4 Ataques a la infraestructura de los “trusted nodes” 14
2.5 Herramientas de Pentesting 14
2.5.1 Descubrimiento y mapeo de red 14
2.5.2 Escaneo de vulnerabilidades 14
2.5.3 Análisis de tráfico 15
2.6 Enrutamiento Dinámico 16
Métodos **18**
3.1 Sistema de enrutamiento dinámico 19
3.1.1 Simulador QKD 20
3.1.2 Arquitectura del simulador 21

3.1.2.1 QKDNode.py 21
3.1.2.2 SDNController.py 25
3.1.2.3 Network_manager.py 29
3.1.2.4 Admin.py 31
3.1.2.5 Topología.json 32
3.1.2.6 ascii_visualizer.py y visualizer.py 34
3.1.2.7 Flujo de Ejecución 34
3.1.3 Enrutamiento Dinámico 38
3.1.3.1 Implementación en el simulador 38
3.1.3.2 Ventajas ante soluciones actuales 40
3.2 Herramienta de análisis de vulnerabilidades 40
3.2.1 Arquitectura de la Herramienta 41
3.2.1.1 Main.py 42
3.2.1.2 cve_scanner.py 43
3.2.1.3 cli.py 44
3.2.1.4 port_scanner.py 44
3.2.1.5 mitm_detector.py 45
3.2.1.6 pdf_generator.py 46
Resultados y Conclusiones 48
4.1 Enrutamiento dinámico 48
4.1.1 Interfaz de Adminstración 52
4.1.2 Nodos de la topología 54
4.1.3 Controlador SDN 55
4.1.4 Representación ASCII de la topología 58
4.2 Pentesting Tras el Ataque 60
4.2.1 Ejecución del código 62
4.2.2 Reporte generado 63
4.3 Combinación de las soluciones 64
4.4 Conclusiones 65
4.4.1 Trabajos Futuros 66
Análisis de Impacto 67
5.1 Impacto del simulador y el enrutamiento dinámico 67
5.2 Impacto de la herramienta de análisis de red 68
Bibliografía 69
Anexos 73
7.1 Anexo A: Reporte Generado por la herramienta de análisis de vulnerabilidades 73

Capítulo 1

Introducción

1.1 Motivación y contexto

La irrupción de la computación cuántica ha supuesto un cambio de paradigma en múltiples disciplinas, y una de las más afectadas es la seguridad de las redes y las comunicaciones. En el mundo actual, las comunicaciones seguras son fundamentales para garantizar la integridad, confidencialidad y autenticidad de la información que se transmite. Esta seguridad se basa en algoritmos criptográficos que confían en la dificultad computacional de ciertos problemas matemáticos, como la factorización de números primos en el caso del algoritmo Rivest-Shamir-Adleman (RSA), o el problema del logaritmo discreto en el caso de Eliptic Curve Cryptography (ECC). Estos métodos han demostrado ser fiables en el contexto de la computación clásica, pero su resistencia se ve seriamente amenazada por las capacidades emergentes de los ordenadores cuánticos.

Los avances en computación cuántica han permitido resolver con eficiencia problemas que antes se consideraban imposibles de resolver. Esto incluye, precisamente, aquellos problemas matemáticos que constituyen la base de los sistemas criptográficos actuales. En consecuencia, ha surgido una necesidad urgente de desarrollar mecanismos de seguridad que sean resistentes a los ataques cuánticos. En este contexto, se han propuesto diversas soluciones, entre las que destacan los algoritmos de criptografía post-cuántica (PQC, por sus siglas en inglés), basados en principios matemáticos diferentes, y también la criptografía cuántica, que aprovecha directamente las propiedades de la física cuántica.

Una de las aplicaciones más prometedoras de la criptografía cuántica es la Distribución Cuántica de Claves (QKD, Quantum Key Distribution). QKD permite a dos partes generar una clave secreta compartida utilizando fenómenos cuánticos como la superposición y el entrelazamiento [3]. Lo que hace especial a esta técnica es su capacidad de detectar cualquier intento de interceptación: cualquier perturbación del canal cuántico modifica el estado de los qubits, permitiendo a las partes detectar la intrusión y, por tanto, garantizar la seguridad del intercambio [1] [2]. Esta propiedad otorga a QKD una solidez teórica superior a la de los métodos clásicos.

Sin embargo, la implementación práctica de QKD aún presenta importantes desafíos. Aunque el canal cuántico pueda ser teóricamente invulnerable, los sistemas reales deben apoyarse en una infraestructura clásica para gestionar el intercambio de información, y esta infraestructura puede ser vulnerable a diferentes tipos de ataques. Además, existen limitaciones físicas, como la distancia de transmisión y la sensibilidad del equipo, que dificultan su adopción a gran escala. Por ello, los esfuerzos actuales no solo se centran en mejorar la fiabilidad del canal cuántico, sino también en fortalecer los componentes clásicos del sistema.

En esta línea, se han desarrollado arquitecturas como la red MadQCI, una infraestructura diseñada para integrar la QKD en redes distribuidas. Esta red se estructura sobre tecnologías como las redes definidas por software (SDN), lo que permite una gestión flexible y programable, aunque también introduce nuevos riesgos debido a la complejidad del software. La importancia de proteger adecuadamente los nodos de la red es fundamental, ya que un fallo en cualquiera de ellos podría comprometer la seguridad global del sistema.

1.1.1 Limitaciones físicas y necesidad de nodos de confianza

Uno de los principales obstáculos técnicos de las redes QKD es que el canal cuántico tiene un alcance limitado, ya que es extremadamente sensible a las pérdidas y no puede ser amplificado sin alterar los estados cuánticos de los qubits. En la actualidad, y mientras no se disponga de memorias cuánticas funcionales capaces de almacenar y reenviar qubits sin colapsarlos, la única solución viable para extender el alcance de estas redes es el uso de trusted nodes o nodos de confianza. Estos nodos actúan como puntos intermedios donde se regeneran nuevas claves cuánticas que permiten enlazar tramos consecutivos de la red. Gracias a esta arquitectura escalonada, es posible implementar comunicación cuántica segura a larga distancia superando las limitaciones físicas del canal cuántico.

1.2 Estructura de una red QKD

Las redes QKD pueden estructurarse de distintas formas según el estándar o protocolo adoptado. En este trabajo se sigue el ETSI GS QKD 015, también utilizado en la red QKD de Madrid, donde se pretende aplicar la solución. Este estándar define una arquitectura modular y flexible que permite integrar sistemas de Distribución Cuántica de Claves en redes definidas por software (SDN), facilitando así la gestión eficiente de recursos cuánticos y clásicos, y su incorporación en infraestructuras de telecomunicaciones actuales.

La arquitectura SD-QKD propuesta divide las funcionalidades en bloques bien definidos: cuánticos, clásicos y de gestión. Esta segmentación permite localizar con precisión qué parte del sistema se ve afectada o mejorada por una solución específica, como la desarrollada en este trabajo.

La estructura que propone el estándar 015 de ETSI se basa en los siguientes elementos:

1. **Nodos SD-QKD:** Dispositivos que integran módulos cuánticos, agentes SDN y el sistema de gestión de claves (KMS). Actúan como punto de conexión entre los enlaces QKD y las aplicaciones que utilizan las claves generadas.

2. **Controlador SDN:** Coordina el funcionamiento de la red, gestiona la topología, configura los enlaces y distribuye las claves, tanto en el plano clásico como cuántico.

3. **Interfaces de control:** Establecen la comunicación entre nodos y el controlador mediante protocolos estandarizados, permitiendo descubrir, configurar y monitorizar los elementos de red.

4. **Canales Cuánticos y Clásicos:** El canal cuántico distribuye claves seguras usando protocolos como **BB84**, el clásico transmite datos y mensajes de control necesarios para completar el proceso.

La siguiente imagen representa la estructura básica de una red QKD enfocada específicamente en los componentes que conforman los nodos QKD.

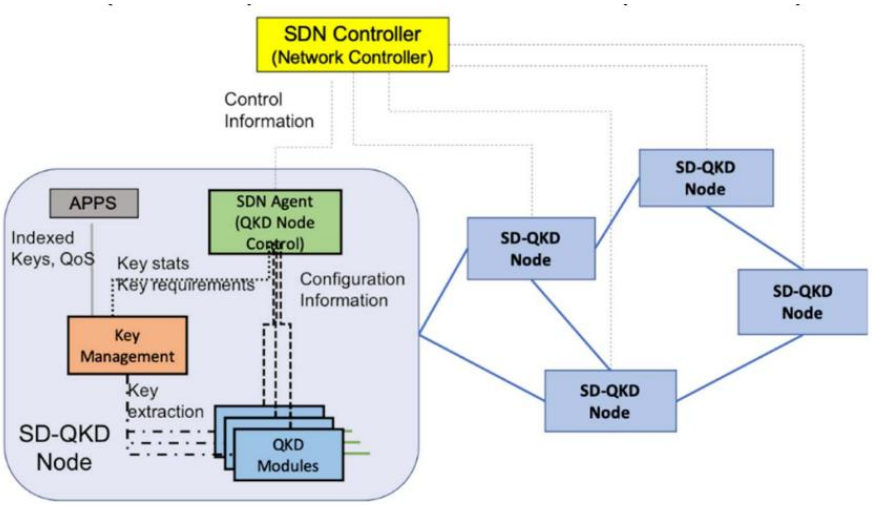


Imagen 1: Red QKD con énfasis en los NodosQKD. [25]

Este estándar propone un flujo de operación muy concreto. El flujo general **comienza** con el descubrimiento y registro de los nodos en el controlador SDN, **seguido** de la configuración de enlaces y rutas. **Posteriormente**, se genera y distribuye la clave cuántica entre los nodos. **Durante** toda la operación, el sistema permanece monitorizado para detectar anomalías o fallos, permitiendo aplicar ajustes o contramedidas según sea necesario.

1.3 Objetivos del proyecto

El objetivo general de este TFG es el diseño e implementación de unos módulos de detección de ataques en redes QKD, así como la aplicación de medidas de contingencia y análisis automatizados, con el fin de abordar el problema de los ciberataques en estas redes. Para ello, se combinan técnicas tradicionales de pentesting y mecanismos de enrutamiento dinámico aplicados a los nodos QKD, contribuyendo así a mejorar la seguridad de esta tecnología emergente.

Los objetivos específicos del proyecto son:

- Investigar las amenazas que afectan a las redes QKD, tanto a nivel físico como cuántico.

- Analizar el funcionamiento de las redes QKD, con especial atención a la arquitectura MadQCI basada en SDN.

- Estudiar la estructura de un nodo QKD conforme a la especificación ETSI GS QKD 015.

- Aplicar técnicas de análisis de seguridad sobre dicha estructura de nodo mediante herramientas de pentesting, identificando posibles vulnerabilidades explotables.

- Diseñar y aplicar contramedidas que permitan mantener el servicio activo ante incidentes de seguridad, mediante técnicas de reenrutamiento de aplicaciones.

1.4 Solución propuesta

La solución propuesta en este TFG se centra en el diseño e implementación de un procedimiento para la detección de un tipo específico de ataque en redes QKD: la intervención o manipulación de la línea de comunicación. Este tipo de ataque representa una amenaza relevante para las redes cuánticas actuales, ya que compromete la integridad de los enlaces. El procedimiento diseñado permite identificar esta intervención, y una vez detectada, notifica al controlador de red para que ejecute un enrutamiento dinámico, redirigiendo el tráfico actual por rutas alternativas seguras. Paralelamente, se inicia un análisis de los nodos implicados para diagnosticar el origen del problema.

Este procedimiento ha sido implementado y validado a través de un prototipo funcional, diseñado específicamente para este tipo de ataques. La idea es que en futuros trabajos pueda extenderse a otros vectores de ataque. Junto con esto, se ha desarrollado un entorno de simulación de enlaces de redes QKD que incluye generación de claves, representación visual del estado de la red mediante un simulador ASCII y módulos para evaluar el comportamiento del sistema bajo diferentes escenarios de ataque.

El enfoque combina dos herramientas clave: por un lado, una herramienta de pentesting aplicada a los nodos físicos QKD, que permite identificar y mitigar vulnerabilidades clásicas que puedan comprometer estos dispositivos. Por otro lado, un sistema de respuesta automática basado en reconfiguración dinámica del tráfico para mantener el servicio disponible incluso ante ataques de denegación de servicio o caídas en la red.

La integración de ambos mecanismos ofrece una solución robusta para proteger los “trusted nodes” o nodos de confianza en las redes QKD. Ante una intrusión, se redirige inmediatamente el tráfico para mantener la comunicación, y posteriormente se ejecuta un análisis de seguridad sobre los nodos afectados para verificar si persisten vulnerabilidades y confirmar la recuperación segura del sistema.

Todo este sistema ha sido validado en un entorno simulado que permite demostrar su eficacia y utilidad práctica, evidenciando mejoras en la resiliencia, disponibilidad y seguridad general de la red frente a ataques.

1.5 Estructura del documento

Este trabajo tiene la siguiente estructura:

- **Capítulo 2 (Estado del Arte):** Aquí se hablará del estado de la tecnología actualmente, en concreto cómo funciona la tecnología QKD, ataques y contramedidas conocidas, herramientas de pentesting para analizar la seguridad de los nodos, y la explicación de la técnica de enrutamiento.

- **Capítulo 3 (Métodos):** En esta sección se describen los métodos propuestos para crear una solución al problema. Se explicará en detalle el código y funcionamiento de la herramienta de pentesting y el sistema de enrutamiento dinámico.

- **Capítulo 4 (Resultados y Conclusiones):** En este capítulo se mostrará la ejecución del código para mostrar distintos casos de prueba y comprobar la efectividad. Finalmente, una breve conclusión de los resultados.

- **Capítulo 5 (Análisis de impacto):** Se analizará el impacto de las herramientas desarrolladas y su contribución con los ODSs.

- **Capítulo 6 (Bibliografía):** Se recoge la bibliografía utilizada en el proyecto.

- **Capítulo 7 (Anexos):** Sección que contiene los anexos del proyecto.

Capítulo 2

Estado del Arte

Las redes de distribución cuántica de claves (Quantum Key Distribution, QKD), son un tipo de red que se distinguen de las redes clásicas en su proceso de encriptar información.

Estas redes cuánticas se componen de un canal clásico para el envío de información entre nodos y para cada par de nodos, un canal cuántico. El canal cuántico se distingue de los clásicos en su propósito. Este se usa para determinar un par de claves compartidas entre dos nodos contiguos y luego la comunicación se encripta con estas claves para una comunicación segura por el canal clásico.

Lo que realmente hace segura y fascinante esta tecnología es que las claves secretas se generan de manera cuántica, es decir, aprovecha las leyes de la física cuántica para generar las claves y tener la capacidad de detectar un intruso en la red en caso de que hubiera alguna perturbación en el canal cuántico. En teoría, QKD ofrece una seguridad incondicional ya que cualquier intento de espionaje en el canal cuántico introduce perturbaciones detectables por los nodos (e.g. debido a un aumento de la tasa de error cuántico) lo que provoca la caída de la red en ese canal.

Sin embargo, en la práctica no existe ningún sistema perfecto. Los dispositivos y nodos QKD son equipos físicos, lo que los hace vulnerables a ataques clásicos. De hecho, todos los ataques contra QKD reportados hasta la fecha han sido causados por desviaciones o fugas de información en la implementación (lo que se conoce como “canales laterales”).

Por ello el estado del arte de la seguridad en las redes QKD tiene que enfocarse primeramente en identificar los posibles ataques reales y desarrollar contramedidas o protecciones efectivas.

A continuación, se revisará:

1. Cómo funciona una red QKD
2. Los principales tipos de ataques a las redes QKD y las soluciones conocidas para mitigar cada ataque.
3. Herramientas de pentesting aplicables ara analizar los nodos físicos de la red.
4. La técnica de enrutamiento dinámico, esencial para asegurar la disponibilidad y evitar caídas innecesarias de la red en caso de un ataque.

2.1 Cómo funciona los dispositivos QKD

Una red QKD se compone de dos canales principales.

1. **Un canal** de envío de información **tradicional**.

En lugar de encriptar la información por este canal mediante técnicas comunes como el de claves asimétricas (donde se emplean un par de claves, una pública y otra privada para la encriptación y la desencriptación del mensaje), cada par de nodos se ponen de acuerdo para usar una clave que usarán para encriptar y desencriptar los mensajes.

2. **Un canal cuántico**.

Este canal se usa para determinar una clave entre dos nodos que posteriormente se usarán para encriptar y desencriptar la información en futuras comunicaciones.

El canal cuántico es especial.

En este canal la clave se manda en forma de qbits, la unidad básica de información en computación cuántica. Estos qubits son una especie de bits que en lugar de tomar valores tradicionales de 0 o 1, pueden tomar el valor "0" y el valor "1" al mismo tiempo, son análogos al bit clásico, pero con propiedades cuánticas que permiten mayores capacidades de codificación y procesamiento de información [33]. Estos qubits se crean de manera que se encuentran en un estado de superposición entre los valores 0 y 1, y siguen las leyes de la mecánica cuántica, obedeciendo de esta manera a la función de onda de Schrödinger que predice su comportamiento.

A priori esto puede sonar extraño y anti intuitivo, pero esta es la maravillosa naturaleza de la física cuántica. Gracias a que entendemos cómo se comportan las partículas a niveles subatómicos, hemos sido capaces de grandes avances en la tecnología, como por ejemplo el propio desarrollo de nuestros teléfonos móviles o que podamos comunicarnos sin importar la distancia a la que nos encontremos.

Otra cosa muy importante del envío de estos qubits es que se generan de manera completamente aleatoria siguiendo las leyes de la cuántica, y de esta manera se le hace imposible a los atacantes poder adivinar la contraseña, ya que los sistemas clásicos de generación de contraseñas son relativamente inseguros ya que generan números aleatorios teniendo en cuenta parámetros medibles como la hora de ejecución, temperatura de GPU, relojes internos del PC, etc.

Además, gracias a que los qubits pueden codificarse mediante fotones ("partículas" de luz), es posible aprovechar las infraestructuras de comunicación actuales, como las fibras ópticas, para transmitirlos. Aun así, los qubits conservan sus propiedades cuánticas, como la superposición y el colapso del estado al ser medidos. Estos fotones se emiten con una energía muy controlada entre el emisor y el receptor, lo que permite detectar si hay alguien escuchando en la línea, ya que la más mínima perturbación cuántica, como una medición no autorizada, provocaría un aumento en las pérdidas (SKR) o errores detectados en la transmisión (QBER). Este fenómeno es la base que permite garantizar la seguridad de la comunicación en la distribución cuántica de claves (QKD) [34].

Una forma común de generar un qubit es mediante la polarización de los fotones (luz). Cada fotón representa un qubit y su estado cuántico se determina mediante un generador cuántico de números aleatorios que decide, por ejemplo, entre usar la base recta (horizontal/vertical) o la diagonal (45º/135º), y el valor del bit (0 o 1).

El emisor (**Alice** a partir de ahora) envía una secuencia de estos fotones a través de una fibra óptica (uno de los canales más comunes). Los fotones están en un estado de superposición antes de ser medidos. Esto significa como he mencionado antes que pueden representar un 0 y un 1 al mismo tiempo hasta que el receptor (**Bob** a partir de ahora) los mide, momento en el cual el estado del fotón colapsa a uno en concreto.

Finalmente, Bob mide cada fotón con una base elegida aleatoriamente. Solo cuando ambos usan la misma base, obtienen el mismo bit.

2.2 Seguridad Cuántica

El concepto de "colapso" que he mencionado antes es esencial para comprender por qué estos sistemas son tan seguros.

En la física cuántica el "colapso" es un concepto bastante polémico debido a que existen muchas interpretaciones de la cuántica, unos le llaman decoherencia, otros tienen interpretaciones como la de "muchos mundos" la cual es bastante aceptada, pero por simplicidad usaremos este término para referirnos al momento el cual **la partícula pasa de estar cuantificada en un estado de superposición, a un estado clásico de un valor en concreto al medirlo**.

Todas las partículas cuantificadas son extremadamente sensibles a colapsar debido a mediciones o pequeñas perturbaciones en el entorno (las mediciones causan perturbaciones que provocan que colapse la partícula). Para medir el estado de una partícula, se ha de interactuar con ella de alguna forma, es esta la interacción que provoca a la partícula colapsar en un estado en concreto.

Es esto lo que hace la tecnología QKD tan segura. Aprovecha una cualidad de las partículas (el colapso al ser medidas) para asegurar que ningún intruso está escuchando (sniffing) en la red para captar la clave secreta enviada. Si un atacante (**Eve**) intenta interceptar los fotones, el acto de medirlos modifica su estado (colapsa) y se introduce un error detectable por Alice o por Bob. Ellos comparan públicamente parte de sus datos para estimar la tasa de error. Si está dentro de límites aceptables, pueden extraer una clave secreta segura y se aplica una reconciliación de clave y amplificación de privacidad para asegurar que ambos tienen la misma clave y que es secreta.

Si no lo está, significa probablemente que existe un intruso en la red (o hay algún factor externo perturbando, como puede ser un camión ruidoso pasando cerca de la zona) por lo que Alice y Bob dejan de compartir la clave y se termina su conexión temporalmente.

2.3 Redes QKD Actuales

En los últimos años, la implementación de redes QKD ha avanzado notablemente, con diversos proyectos internacionales que han demostrado la viabilidad de esta tecnología en entornos reales. Entre ellos, destaca **SECOQC**, una de las primeras redes metropolitanas de QKD implementadas, creada en Viena (Austria). Integró nodos de distintos fabricantes y demostró la interoperabilidad y escalabilidad de la tecnología QKD en un entorno urbano, conectando bancos, organismos gubernamentales y universidades [28].

También sobresale la **Tokyo QKD Network** desplegada en Japón, que integró múltiples protocolos QKD y diferentes tipos de enlaces como fibra y enlaces ópticos libres. Esto permitió probar la gestión dinámica de claves en una red compleja y su integración con infraestructuras de telecomunicaciones ya existentes [29].

En Suiza, la **Quantum Network in Geneva** fue una de las primeras en ofrecer servicios comerciales de distribución de claves cuánticas, principalmente en el sector financiero [30].

A nivel europeo, la iniciativa **EuroQCI** (**European Quantum Communication Infrastructure**) tiene como objetivo desplegar una red paneuropea de comunicaciones cuánticas que conecte infraestructuras nacionales y centros de investigación. Esta red está actualmente en desarrollo y cuenta con la participación de múltiples países de la Unión Europea [31].

En el contexto nacional, destaca la red **MADRID QKD**, donde se pretende desplegar la solución propuesta en este trabajo. Esta red experimental conecta diferentes centros de investigación y universidades en la Comunidad de Madrid utilizando tecnología QKD sobre fibra óptica. Se considera un ejemplo de integración práctica con la infraestructura de telecomunicaciones existente. Además, forma parte de los esfuerzos por posicionar a España en la vanguardia de la comunicación cuántica europea y se integra dentro de las iniciativas del EuroQCI [32].

La evolución de estos proyectos está impulsando la estandarización y adopción de tecnologías cuánticas en el ámbito de las telecomunicaciones globales.

2.3.1 Integración con infraestructuras existentes

La integración con redes o infraestructuras de red existentes es una de las principales ventajas de la arquitectura ETSI GS QKD 015. Esto se debe a que esta arquitectura está diseñada para operar sobre redes definidas por software (SDN), utilizando tecnologías y protocolos estándar ampliamente adoptados en la industria, como los modelos YANG y el protocolo NETCONF. Gracias a esto, los nodos cuánticos pueden interoperar con equipos de red convencionales, sin requerir modificaciones profundas en la infraestructura subyacente.

Además, esta compatibilidad permite una adopción gradual de la tecnología QKD, integrándola en redes de telecomunicaciones actuales sin interrumpir los servicios existentes. El uso de estos estándares facilita tareas como la configuración remota, supervisión del estado de los enlaces, y gestión de recursos, lo cual reduce los costes de despliegue y acelera el proceso de incorporación de la seguridad cuántica en redes ya operativas.

2.4 Panorama de amenazas

La seguridad de las redes QKD, aunque sea teóricamente sólida gracias a los principios de la mecánica cuántica, sigue enfrentándose a múltiples amenazas en entornos reales. En esta sección se describen algunos de los ataques más representativos que pueden afectar tanto al canal cuántico, como al clásico, así como las contramedidas más reconocidas. Algunas de estas soluciones, especialmente aquellas relacionadas con el enrutamiento dinámico como defensa frente a ataque de denegación de servicio, han sido implementadas en este trabajo y se detallan más adelante en la sección de métodos.

2.4.1 Ataques al Canal Cuántico (Eavesdropping e Interferencia)

Cualquier intento de espionaje en el canal cuántico de QKD será detectado. Si un adversario intercepta los fotones en tránsito introducirá errores en las correlaciones cuánticas entre Alice y Bob. Este aumento en la tasa de error cuántico (QBER) alertará a Alice y a Bob de la presencia de Eve, permitiéndoles abortar la comunicación antes de que Eve pueda comprometer la red [4]. Sin embargo, existen varios ataques más sutiles en el canal cuántico que han sido identificados a lo largo de los años. A continuación, enumeraré y explicaré diversos ataques en el canal cuántico junto con sus contramedidas.

2.4.1.1 Ataque de división de fotones (PNS):

Si la fuente de qubits de Alice en lugar de ser un emisor de fotones realmente individuales, fuera un láser atenuado (fuentes coherentes débiles suelen emitir pulsos con distribución de Poisson) existiría la posibilidad de que algunos pulsos contengas más de un fotón. Debido a esto, Eve puede interceptar estos pulsos con varios fotones, separa uno de los fotones, los almacena en memoria cuántica y deja pasar el resto a Bob. Mas adelante cuando Alice y Bob revelan públicamente la base de codificación usada en cada posición, Eve mide usando la base pública, el valor decodificado de los fotones capturados, obteniendo así información parcial de la clave sin causar errores. Esto se debe a que Bob ha recibido todos sus fotones y nosotros solo hemos interceptado duplicados.

Este ataque no es sencillo ni trivial de ejecutar ya que requiere tecnología de memoria cuántica, pero muestra la importancia de tener un sistema que asegure el envío de fotones únicos [4].

La contramedida principal es el uso de **estados señuelo** (decoy states), propuesta por Hwang en 2003. Alice varía aleatoriamente la intensidad de sus pulsos enviando algunos fotones señuelo con menor intensidad conocida [3]. Cualquier intento de ataque PNS desequilibraría las tasas de detección de estos estados señuelo vs los pulsos señal, lo que revelaría la presencia de Eve.

Los protocolos QKD modernos ya incorporan esta técnica para garantizar seguridad incluso con fuentes coherentes débiles.

2.4.1.2 Ataque Man-in-the-Middle (MitM):

Este ataque no es realmente un ataque al canal cuántico por sí solo, ya que es precisamente la tecnología QKD la misma que neutraliza este ataque. Este se realiza por falta de autenticación en la comunicación. Eve en lugar de interceptar la comunicación entre Alice y Bob, se hace pasar por alguno de los dos, y de esta manera Alice cree que le está mandando la clave a Bob, cuando realmente se lo está mandando a Alice. QKD es vulnerable a ataques de MitM si no se acompaña de un canal clásico autenticado [4].

La solución sería usar métodos clásicos de autenticación, por ejemplo, códigos de autenticación "unconditionally secure" basados en familias de hash universales, como Carter-Wegman, usando una pequeña clave precompartida (PSK, Pre-Shared Key) para firmar todos los mensajes en la fase de sifting y reconciliación [1]. De esta manera Eve no podrá falsificar las etiquetas de autenticación sin ser detectada.

No obstante, dado que estas soluciones clásicas podrían verse comprometidas en un futuro por la computación cuántica, también se están explorando métodos de autenticación post-cuántica (PQC), que permitirían mantener la seguridad de este proceso incluso ante adversarios con capacidades cuánticas. Incluir autenticación PQC en el canal clásico complementaría el uso de QKD, haciendo todo el proceso verdaderamente quantum-safe.

2.4.1.3 Ataques de Interferencia (Jamming):

Este ataque no pretende leer información cuántica, sino crear interferencias en el canal cuántico para que Alice y Bob no se puedan comunicar ni enviar la clave cuántica.

Los sistemas de QKD operan con señales de muy baja potencia por lo que son vulnerables a ruido inyectado. Por ejemplo, se podría aplicar un campo magnético o introducir luz polarizada adicional para randomizar la generación de qubits, crear una tasa de errores muy alta y de esta manera se abortaría la sesión [2]. También de manera más simple, aplicar un láser en el canal cuántico crearía mucho ruido óptico y de la misma manera se abortaría la sesión.

Esto es muy peligroso ya que si se interfieren suficientes canales, se podría realizar un ataque de DoS (Denial of Service o Denegación de Servicio) lo que causaría una caída de la red y podría tener grandes implicaciones negativas en la organización [5].

No existe una contramedida directa a este ataque por ello es tan peligroso. Lo que sí que se podría realizar es una ampliación de la red y aplicar un enrutamiento dinámico (el objetivo del TFG) y de esta manera, aunque se ataquen varios nodos, si se amplía lo suficiente la red, habría menos posibilidades de tumbarla y se redirigiría la comunicación por otra ruta.

2.4.2 Ataques a los nodos de confianza "Trusted Nodes"

Los nodos físicos de una red QKD, son blancos potenciales de ataques de aprovechan desviaciones del comportamiento normal de los dispositivos. En la última década, han aparecido los llamados "hackers cuánticos" que aprovechan vulnerabilidades para comprometer la seguridad utilizando ataques como los de **Side Channel** que extraen información a través de canales no previsto o ataques

de **inyección de control**. A continuación, explicaré diversos ataques a los nodos físicos. [6]

2.4.2.1 Ataques a los detectores (Bob)

Los detectores de fotones individuales (típicamente fotodiodos de avalancha, APDs) en Bob son el punto débil más explotado en QKD. Un ataque muy complejo que se le puede realizar a Bob es el **Blinding Attack**. En 2010 se descubrió que es posible tomar el control de detectores de Bob mediante iluminación brillante, evitando que detecten fotones individuales. Este ataque es muy complejo, pero se consiguió realizar exitosamente en dos sistemas comerciales de QKD y robando la clave completa sin ser detectados. [7]

La contramedida inmediata recomendada fue incorporar medidores de potencia óptica en la entrada de Bob para detectar niveles anormalmente altos de luz [8]. También se propusieron otras medidas como realizar test-pulses aleatorios, un rediseño de los APDs y de forma radical migrar a la tecnología MDI-QKD donde los detectores pueden ser no confiables [9].

2.4.2.2 Ataques de Trojan-horse al emisor (Alice)

Estos ataques de Troyano aprovechan la luz que se refleja desde los moduladores internos de Alice, inyectando pulsos laser cortos y midiendo la luz devuelta. De esta manera Eve puede inferir la base o el bit codificado. Este ataque se llevó a cabo por primera vez en 2014, Jain *et al* [7] comprometió un sistema QKD real, y lograron obtener suficiente información sobre las bases utilizadas en cada pulso cuántico.

Existen defensas a este ataque como aisladores ópticos de alta relación de extinción y filtros espectrales que impidan la entrada de luz no autorizada, atenuadores fijos en la salida y sensores que disparan una alarma si detectan luz entrante no autorizada.

2.4.2.3 Canales laterales de detectores

Los canales laterales involuntarios añaden mas riesgo. Entre ellos está el backflash de los detectores de Bob. Reportado por Kurtsiefer *et al.* En 2001, después de cada detección hay una pequeña probabilidad de que el fotón salga de Bob por la fibra, y potencialmente yendo de vuelta por el canal cuántico [7]. Si Eve pudiera captar esos fotones y analizar sus características podría deducir que detector y que bit se activó en Bob, y así obtiene información de la clave.

La solución a esto es emplear **aisladores** en la salida de Bob y absorbentes anti reflejo que bloquean cualquier luz saliente [7].

2.4.2.4 Daño láser

Este ataque es más agresivo y fue demostrado por Bugge *et al.* En 2014. Se usan potencias de luz suficientemente altas para dañar permanentemente el sistema QKD. Este ataque puede crear nuevos desbalances y facilitar ataques posteriores ya que daña el sistema o lo altera [7].

Para solucionar esto se requiere seguridad física, como carcasas selladas, controles de acceso, sensores de temperatura y monitorización de potencia interna [7].

2.4.3 Vulnerabilidades en el canal clásico de QKD

El canal clásico es el encargado de transportar los mensajes de "sifting", reconciliación de errores, amplificación de privacidad y autenticación mutua. Si un atacante consigue manipular los mensajes críticos intercambiados por el canal, la solidez y seguridad del protocolo se deteriora.

2.4.3.1 Suplantación o interceptación (MitM clásico)

Este es el ataque más serio que se le puede realizar a un QKD. Si Eve consigue situarse entre Alice y Bob, y los mensajes carecen de un mecanismo de autenticación robusto, puede sustituir, inyectar o eliminar información [4].

Mantener una clave precompartida de autenticación y renovarla periódicamente empleando una sección de la clave cuántica recién generada es una solución sencilla al problema. También se ha de supervisar la caducidad y longitud de los tags.

En este TFG se propone una herramienta de análisis de red que escanea la red en búsqueda de vulnerabilidades y posibles brechas, pero en concreto, se implementará un módulo para la intercepción de este tipo de ataques MitM.

2.4.3.2 Denegación de Servicio (DoS)

Esta amenaza puede ser altamente crítica dependiendo de la organización a la que se ataque. El canal cásico puede saturarse mediante un ataque DDoS o por la inyección de comandos que obliguen a reiniciar repetidamente el protocolo de reconciliación.

Para solucionar este problema se puede aplicar filtrado y "rate limiting" o limitación de tasa, incluir mecanismos de reintento con *back-off* exponencial y si es posible, disponer de canales de control dedicados, o lógicamente aislados del público.

Como medida central en este TFG, se ha desarrollado un sistema de enrutamiento dinámico que actúa como contramedida ante este tipo de ataques, permitiendo redirigir automáticamente el tráfico a través de rutas seguras cuando se detecta la caída o interferencia de un enlace.

2.4.3.3 Filtración de información en reconciliación

En la fase de reconciliación de errores existe la posibilidad de fuga de información como el número de rondas, las paridades, el tiempo total de ejecución del protocolo, etc. Estos pueden relacionarse con la distribución de errores y reducir el espacio de búsqueda de la clave.

Se pueden implementar protocolos para tener tiempos de ejecución constantes, alternar el tamaño de los bloques o incluso introducir relleno o ruido en los intercambios para solucionar o prevenir el ataque.

2.4.3.4 Ataques a la infraestructura de los "trusted nodes"

En redes de varios saltos, la clave se suele enrutar a través de nodos de confianza (trusted nodes) o servidores KMS. El compromiso de estos equipos de manera clásica, mediante malware o técnicas de hacking, supondría la pérdida total de confidencialidad, por lo que si un atacante decide atacar un trusted node y lo consigue satisfactoriamente podría causar grandes estragos en la organización. Por ello es imprescindible endurecer su seguridad tanto física como lógica.

Uno de los objetivos de este TFG, es la solución de este problema. Securizar un trusted node no se realiza mediante una herramienta en concreto, es una tarea con un *scope* muy grande por lo que se ha de utilizar diversas herramientas a distintos niveles (físico, de red, de aplicación, etc.) para asegurar que el nodo se encuentra aislado de atacantes y lo más seguro posible.

2.5 Herramientas de Pentesting

A continuación, se describirán las herramientas que se podrían usar para realizar el pentesting a los trusted nodes. Para cada uno se indica su nombre, que apoyo me puede brindar, y cómo funciona.

2.5.1 Descubrimiento y mapeo de red

Para comenzar el análisis de seguridad es necesario identificar y mapear los nodos presentes en la red. A continuación, en la Tabla 1, se presentan algunas herramientas utilizadas habitualmente en tareas de descubrimiento y mapeo.

Herramienta	Que hace	Cómo funciona
Nmap	Escanea rangos de IP/puertos, detecta sistema operativo y versiones de servicios.	Envía paquetes TCP/UDP y analiza las flags de respuesta. Incluye un modo de Scripts para tests automáticos específicos. Por ejemplo, banner-grabbing de la API QKD 004. [10]
masscan	Scanner de puertos de alta velocidad	Reproduce la sintaxis de Nmap pero genera múltiples SYN en paralelo, se usa para descubrir puertos abiertos antes de un escaneo más en detalle. [11]

Tabla 1: Herramientas Pentesting, mapeo de la red

2.5.2 Escaneo de vulnerabilidades

Una vez identificados los dispositivos, el siguiente paso consiste en detectar posibles vulnerabilidades en los servicios que ofrecen. La Tabla 2 recoge distintas herramientas que permiten analizar configuraciones incorrectas o fallos de seguridad conocidos, a partir de bases de datos como CVE.

Herramienta	Que hace	Cómo funciona
API de NIST	API que proporciona acceso a la base de datos de vulnerabilidades CVE.	Funciona mediante peticiones HTTP (REST) y devuelve resultados en formato JSON
OpenVAS/GVM	Análisis de CVE y malas configuraciones.	Lanza plugins contra los servicios detectados. [12]
Nessus Essentials	Alternativa gratuita a Tenable	Feed de más de 75000 plugins. Devuelve un score CVSS, es útil para comparar resultados con OpenVAS y descartar falsos positivos. [13]

Tabla 2: Herramientas Pentesting, escaneo de vulnerabilidades

2.5.3 Análisis de tráfico

El análisis del tráfico de red permite estudiar en profundidad el comportamiento del canal clásico. En la Tabla 3 se resumen herramientas útiles para inspeccionar, capturar y generar paquetes, así como para simular posibles ataques mediante scripts personalizados.

Herramienta	Que hace	Cómo funciona
Wireshark	Inspección de paquetes del canal cásico	Captura en vivo los paquetes para luego analizarlos y buscar vulnerabilidades.
tcpdump	Similar a Wireshark, pero sin interfaz gráfica	Captura paquetes para luego analizarlos con Wireshark. Útil para entornos con CLI únicamente
Scapy	Generación/decodificación de paquetes y fuzzing de protocolos	Permite construir paquetes a medida y medir respuestas. Útil para scripts ad-hoc que simulan tráfico malicioso en el canal clásico [15]

Tabla 3: Herramientas Pentesting, análisis de tráfico

El pentesting tendrá distintas fases:

1. **Mapeo inicial:** Con nmap o con msscan para resultados más rápidos.
2. **Escaneo de vulnerabilidades**: Se podrían usar herramientas como nmap con scripts u otras alternativas para escanear la red en busca de vulnerabilidades.
3. **Asignar CVE:** Comparar las vulnerabilidades encontradas previamente con la base de datos de CVEs para asignarles uno si existiera.

4. **Documentación:** Se crea una documentación que recoja los datos escaneados en un documento sencillo para gente inexperta en la ciberseguridad.

2.6 Enrutamiento Dinámico

El enrutamiento dinámico es esencial en las redes QKD. Como hemos visto antes, un atacante podría intentar tumbar el servicio y la red escuchando en un canal. Si no existiera enrutamiento dinámico, esto impediría que Alice y Bob se comunicaran ya que el atacante está impidiendo su comunicación. Si se le aplica a la red un protocolo de enrutamiento dinámico, aunque el atacante esté bloqueando un camino a Bob, Alice podrá comunicarse desde otro camino que no esté siendo bloqueado. Esto aseguraría la disponibilidad de la red y minimizaría las posibilidades de un ataque de DoS.

Existen varios algoritmos que se pueden usar para realizar el enrutamiento dinámico. A continuación, se explican los distintos algoritmos que existen y luego una explicación detallada del algoritmo escogido.

Algoritmo	Idea clave	Cuando se prefiere	Complejidad
Bellman-Ford	Cada nodo envía periódicamente a sus vecinos la tabla de distancias [16]	Redes pequeñas	O(V E) por iteración
Floyd-Warshall	Programación dinámica que obtiene las distancias mínimas entre todos los pares de nodos en una sola pasada [17]	Redes de hasta unos cientos de nodos cuando se necesitan todas las combinaciones.	O(V^3)
A*	Extiende Dijkstra con una heurística admisible para dirigir la búsqueda hacia el destino [18]	Cuando se conoce un objetivo único y se dispone de una estimación heurística	O(E) a O(ElogV)
Yen K-Shortest Paths	Encuentra los k caminos más cortos, reutilizando Dijkstra para cada desviación [19]	Redes que necesitan rutas de respaldo pre-computadas	k·O(ElogV)
Suurballe	Calcula dos o más caminos enlaces disjuntos con coste mínimo global [20]	Requerimientos de alta disponibilidad o QKD paralela con canales independientes	k·O(ElogV)

Dijkstra	Calcula el camino de menor coste desde un origen a todos los demás nodos	Red de tamaño medio-pequeño donde enlaces se marcan UP/DOWN al superar un umbral	O((V+E)logV)

Tabla 4: Enrutamiento Dinámico

Capítulo 3

Métodos

A continuación, se explicará para a paso como se ha abordado el problema planteado (Intrusiones en los "Trusted Nodes" en las tecnologías QKD). Se explicará cómo se realizó el enrutamiento dinámico y el simulador de la red QKD correspondiente, luego se indagará en el desarrollo de la herramienta de análisis de vulnerabilidades.

Antes de comenzar a entrar en detalles técnicos, se muestra a continuación una representación a alto nivel de cómo es la estructura general del simulador.

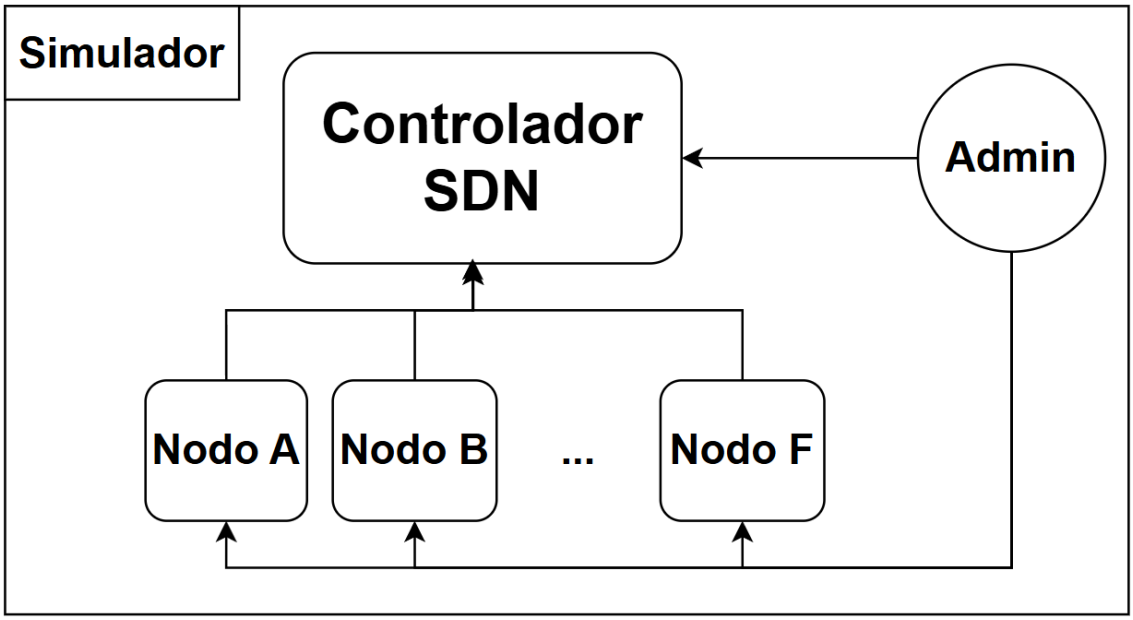


Imagen 2: Estructura general del simulador.

En la imagen se puede observar como el simulador se compone por X nodos, determinado en el fichero de entrada del programa; El controlador SDN, encargado de gestionar la red; y el Administrador, que es una interfaz que se encarga de simular acciones en la red, como intrusiones, envíos de mensaje, etc.

Cuando el controlador detecte cualquier tipo de intrusión en la red, se encarga de enrutar la comunicación por una ruta segura. Paralelamente se decidirá si se ejecuta la herramienta de pentesting, que analizará la red en busca de vulnerabilidades.

En la imagen 3 se puede ver la estructura en alto nivel de esta herramienta. Se puede observar cómo se compone de varios módulos, cada uno con su función específica en el programa, como buscar puertos, analizarlos en búsqueda de servicios y versiones, el detector de ataque MitM y el escáner de CVEs (vulnerabilidades conocidas). Todo esto finalmente se agrupa en un reporte formal en formato PDF.

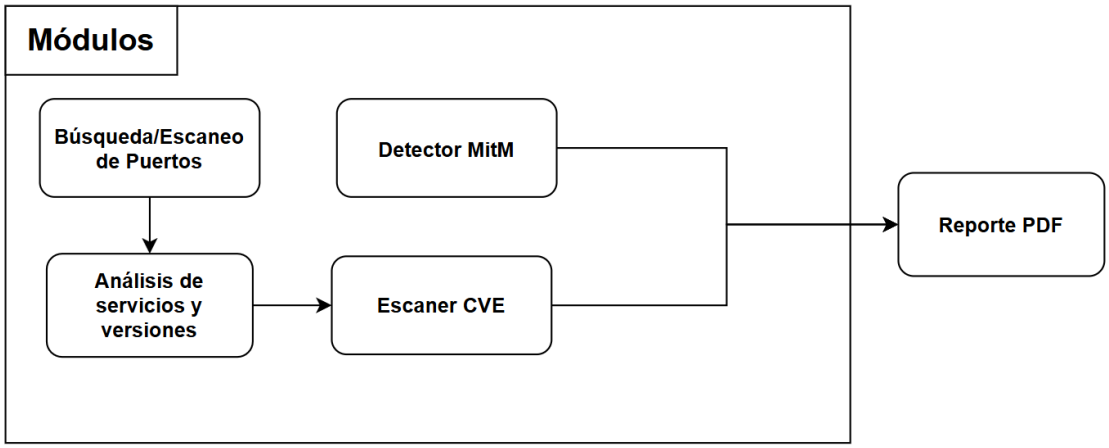


Imagen 3: Estructura general de la herramienta de Pentesting.

3.1 Sistema de enrutamiento dinámico

La primera mitad de la solución es el sistema de enrutamiento dinámico. Esta es la parte que se encarga de asegurar la disponibilidad de la red aún en casos críticos como ataques MitM o de denegación de servicio. Asegurar una alta disponibilidad en la red es una de las mayores preocupaciones en las organizaciones, especialmente en grandes compañías u organizaciones donde es especialmente importante, debido a que una caía de la red podría generar grandes pérdidas económicas. Esto puede ser especialmente crítico en proveedores de servicios en la nube o grandes empresas cuyos servicios se usan a diario por miles de personas (Netflix, Whatsapp, Youtube, etc.).

Este factor al ser tan importante, no solo se debería tener una fuerte seguridad lógica, sino una gran seguridad física que permita proteger a la red incluso de ataques locales. Este tipo de seguridad se suele pasar por alto y se suele pensar que los cibercriminales solamente actúan desde sus casas, pero nada más lejos de la realidad. Si un sistema está muy bien protegido en cuanto a seguridad lógica o de software (puertos cerrados, firewall, proxy, aplicaciones actualizadas…) pero no tiene buena seguridad física, en caso de ser atacado físicamente, puede ser casi tan vulnerable como si no tuviera medidas de protección. Por eso, aunque este trabajo trate solamente la ciberseguridad de las redes QKD, es fundamental no restarle importancia a la seguridad física.

Para implementar el mecanismo de reconfiguración de rutas ante una caída de enlace, ha sido necesario seleccionar un algoritmo de cálculo de rutas que permita adaptarse rápidamente a cambios topológicos. Entre las posibles opciones, se ha decidido usar el algoritmo de Dijkstra. La elección se basa en varias razones: la red objetivo es de tamaño pequeño o mediano (menos de varias decenas de nodos), lo que hace viable su ejecución en tiempo real. Además, los enlaces de la red se monitorizan continuamente y se marcan como UP o DOWN en función del umbral de QBER; si este umbral se supera, el enlace se considera no disponible. Esto encaja perfectamente con el modelo de grafos ponderados en los que opera Dijkstra.

Otra razón clave es su rapidez y simplicidad: implementado con un heap binario, el cálculo de una nueva ruta puede realizarse en milisegundos, y el código necesario en Python es conciso. Además, su carácter modular permite

extenderlo fácilmente en el futuro a variantes como k-shortest paths o Suurballe, lo que lo hace escalable y adaptable.

El algoritmo de Dijkstra resuelve el problema del camino más corto con pesos no negativos partiendo de un único origen [21] [22]:

1. Primero inicializa la distancia de todos los vértices a "infinito" salvo la de origen (0) y coloca cada vértice en una cola de prioridad, en mi caso un heap binario.

2. Luego itera: Extrae el nodo "u" con menor distancia provisional, lo incorpora al conjunto definitivo y "relaja" todas las aristas salientes (u → v). Relajar significa calcula la ruta alternativa **dist[u] + w(u,v)** y si resulta menor que **dist[v],** actualiza esta distancia y disminuye la clave de v en cola.

3. Repite el proceso hasta vaciar la cola, almacenando en un vector **prev[]** los predecesores para reconstruir cada ruta óptima.

La eficiencia depende de la estructura usada para la cola, en mi caso, al usar un heap binario, la complejidad es de **O((V+E)logV)** operaciones (extrae V veces el mínimo y realiza hasta E actualizaciones de clave) [23] [24]. Su naturaleza determinista y la facilidad que tiene de integración con métricas hibridas, hace que sea la opción más atractiva para la implementación.

3.1.1 Simulador QKD

Naturalmente, para poder diseñar el sistema es completamente necesario un entorno donde aplicarlo y realizar pruebas. Para esto, se decidió crear un simulador personalizado donde cada elemento fuera lo más fiel a la realidad posible, realizando cada elemento las mismas funciones que harían en un sistema real.

Python fue seleccionado como lenguaje principal para el desarrollo del simulador debido a varias razones. Primero por su **versatilidad y simplicidad.** Python es conocido por si sintaxis clara y concisa, lo que facilita el desarrollo rápido y la lectura del código para realizar el "debugging". También, Python cuenta con una **amplia gama de bibliotecas** lo que facilita el desarrollo enormemente y reduce significativamente la longitud del código. Finalmente, **es muy sencillo integrar** con otras herramientas y lenguajes. De esta manera si se quisiera crear una interfaz se podría realizar sencillamente.

En resumen, es una herramienta es modular, versátil y sencilla, es en general una muy buen lenguaje de programación si lo que buscamos son estas cualidades por lo que es una herramienta que permitirá una implementación efectiva del simulador.

3.1.2 Arquitectura del simulador

Como se ha mencionado antes, el objetivo del simulador es que sea lo más fiel a la realidad posible, por lo que he implementado una arquitectura que simula el comportamiento de los principales elementos de la red QKD.

El programa se compone por 6 clases de Python y un archivo JSON:

- **QKDNode.py:** Esta es la clase que modela el comportamiento de un nodo QKD, usando sus claves para encriptar y desencriptar los mensajes, mandando reportes de los valores "qber" y "skr" al SDNController, etc.

- **SDNController.py:** El Controlador SDN (Software Defined Networking o Redes Definidas por Software) es el "cerebro" de la red. Toma decisiones sobre el enrutamiento y configuración de la red. Recibe los reportes de los nodos QKD.

- **Network_manager.py:** Clase de control, sirve para lanzar los elementos de la red y del programa.

- **Admin.py:** Esta es la clase que sirve para administrar la red QKD. Sirve para simular los ataques a la red, restaurar conexiones y enviar mensajes entre nodos.

- **Topología.json:** El programa se ha de inicializar con una topología de la red. Este JSON modela como va a ser la estructura de la red y otras configuraciones adicionales para el correcto funcionamiento del programa.

- **Ascii_visualizer.py:** Este archivo se encarga de pintar visualmente la red en formato ascii.

- **Visualizer.py:** Script que se usa para ejecutar el visualizador y mantenerlo actualizado.

A continuación, se explicarán en detalle las clases que conforman el simulador y el redireccionamiento de datos. También se mostrarán fragmentos de código cortos. Estos serán una reducida representación del código real, por lo que se omitirán varias diversas funcionalidades para acortar la longitud del código.

3.1.2.1 QKDNode.py

El archivo QKDNode implementa el comportamiento de los componentes y funcionalidad principales de un nodo QKD real. Cada nodo simula la generación de claves compartidas con sus vecinos, reporta la calidad de los enlaces y permite simular ataques sobre la red. Todo esto siguiendo la estructura del estándar ETSI GS QKD 015 que define la arquitectura de la red [25].

Inicialización del nodo y generación de claves

```
def __init__(self, …):
        self.name = name
        self.port = port
        self.neighbors = neighbors
        self.keys = {n: self.generate_keys(seed)...
        ... for n,(_,_, seed) in neighbors.items()}
        self.intrusion_flags = {n: False for n in neighbors}
        self.report_interval = report_interval
        self.ctrl_ip = controller_ip
        self.ctrl_port = controller_port
        self.admin_port = admin_port
```

Cada nodo se instancia con los siguientes parámetros: nombre, puerto de comunicaciones, vecinos (con sus semillas para la generación de claves), y detalles de conexión al controlador SDN.

En la construcción del objeto, para cada vecino se genera un conjunto de claves simétricas a partir de la semilla compartida. Esta generación de claves se realiza mediante u generador de números pseudoaleatorios que están definidos por la semilla compartida entre dos nodos contiguos.

```
def generate_keys(self, seed):
    rng = random.Random(seed)
    return [rng.randint(0, 999999) for _ in range(100)]
```

De esta manera se asegura que dos nodos conectados generen el mimo conjunto de claves para cifrar y descifrar sus mensajes, y así permitir la comunicación entre dos nodos contiguos pero dos nodos que no estén conectados directamente no se puedan comunicar. Esto, en una implementación real del sistema es un poco distinto, ya que el SDNController forzaría a cada par de nodos contiguos (que formen parte de la ruta de envió del mensaje), que creen las claves, en este caso formadas por qbits.

Comunicación y cifrado entre nodos

La comunicación entre nodos se realza utilizando conexiones TCP para enviar mensajes cifrados. El cifrado por simplicidad es de tipo XOR, suficiente para la simulación.

```
def xor_encrypt_decrypt(self, message, key):
    return ''.join(chr(ord(c) ^ (key % 256)) for c in message)
```

Al enviar un mensaje al vecino, el nodo utiliza la clave correspondiente para cifrar el texto antes de establecer la conexión.

```
def send_to_neighbor(self, neighbor_name, message, index=0):
    key = self.keys[neighbor_name][index]
    encrypted = self.xor_encrypt_decrypt(message, key)
    # Se omite la gestión de sockets para no saturar el texto
```

Este mecanismo permite simular el intercambio de información protegida mediante claves QKD en cada salto de la red.

Reporte periódico de métricas de enlace

Un aspecto clave para la resiliencia de la red QKD es la monitorización constante del estado de los enlaces entre nodos. Cada nodo ejecuta hilos dedicados que, en intervalos configurables, reportan al controlador SDN los valores de **QBER** (Quantum Bit Error Rate) y **SKR** (Secret Key Rate) de cada enlace.

En la realidad QBER mide el porcentaje de bits que han sido recibidos incorrectamente durante la transmisión de claves cuánticas entre dos nodos. Un QBER bajo indica que la transmisión es segura, pero uno alto puede significar que hay interferencias, fallos técnicos, o un atacante en la red.

El SKR mide la velocidad a la que dos nodos puede generar una clave secreta. Esto importa ya que, en la presencia de una atacante, se disminuye el SKR debido a errores introducidos, porque se requiere más tiempo de procesamiento para corregirlos lo que disminuye la tasa de generación de claves.

En el contexto de la simulación de la red, estos valores reportados se degradan para reflejar el compromiso de seguridad.

```
def report_loop(self, neighbor, base_qber=0.006, base_skr=4700,
noise=0.002):
   while True:
       if self.intrusion_flags[neighbor]:
           qber = 0.15
           skr = 0
       else:
           qber = max(base_qber + random.uniform(-noise, noise), 0)
           skr = max(base_skr * (1 - qber), 0)
        # Envío del reporte al controlador
```

Esto permite el SDN tomar decisiones sobre si desactivar la conexión entre dos nodos y redirigir el mensaje por otra ruta, o no.

Interfaz administrativa y simulación de ataques

Cada nodo expone un puerto UDP dedicado a la administración remota. A través de una interfaz de administración, los usuarios pueden decidir si se realiza una intrusión sobre un enlace o si se restaura este mismo, así como solicitar el envío de mensajes por la red.

```
action = cmd.get("action")
            if action == "intrusion":
                nei = cmd.get("neighbor")
                self.intrusion_flags[nei] = True
                print(f"... Intrusión marcada en enlace {nei}")
            elif action == "restore":
                nei = cmd.get("neighbor")
                self.intrusion_flags[nei] = False
                print(f"... Intrusión restaurada en enlace {nei}")
            elif action == "send_message":
                if "dst" in cmd:
                    dst = cmd.get("dst"); msg = cmd.get("message")
```

Integración con el controlador SDN

Cuando se desea enviar un mensaje a un nodo destino no directamente conectado, el nodo de origen solicita al controlador SDN que calcule la ruta óptima. El controlador, que conoce el estado global de la red y la calidad de los enlaces, determina el mejor camino y coordina el envío paso a paso.

```
def send_to_controller(self, message, dst):
   cmd = {"action": "send_message", ...
          ... "src": self.name, "dst": dst, "message": message}
```

Ejecución del nodo

Al inicializar el programa, se inicializan los nodos arrancando un hilo para cada nodo que se encarga de escuchar mensajes, reportar métricas y administración.

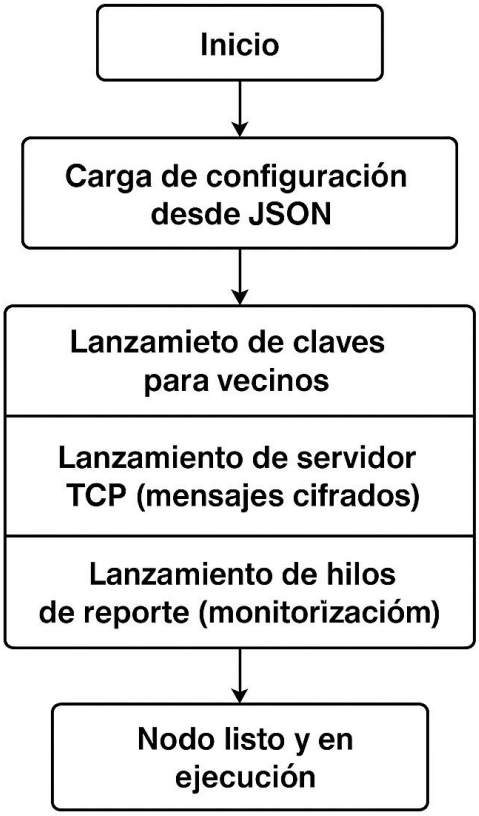


Imagen 4: Flujo ejecución de los nodos

Flujo de ejecución:

- Carga de configuración desde JSON
- Generación de claves para vecinos
- Lanzamiento de servidor TCP (mensajes cifrados)
- Lanzamiento de hilos de reporte (monitorización)
- Lanzamiento de servidor UDP administrativo

3.1.2.2 SDNController.py

El controlador SDN asume un papel esencial como una entidad globalmente informada que es responsable de gestionar la topología, supervisar el estado de los enlaces y decidir dinámicamente las rutas óptimas para la distribución de los mensajes. La existencia del controlador separa el plano de datos y el plano de control.

Inicialización y configuración

El controlador se inicializa con los siguientes parámetros:

Los umbrales de seguridad (QBER máximo y SKR mínimo), su puerto de escucha UDP, y la topología completa de la red que se lee desde un archivo de configuración JSON.

También se inicializa con variables de soporte para calcular las rutas como el destino actual, origen actual y camino mínimo actual.

```
class SDNController:
    def __init__(self, qber_max=0.11, skr_min=1, ...
        ... listen_ip="0.0.0.0", listen_port=6000, nodes_cfg=None):
        self.graph = {}
        self.nodes_cfg = nodes_cfg or {}
        self.current_source = None
        self.current_dest = None
        self.current_path = None
        self.QBER_MAX = qber_max
        self.SKR_MIN = skr_min
        self.listen_ip = listen_ip
        self.listen_port = listen_port
```

Inicializar de esta manera el controlador SDN permite que tenga una visión global de la topología de la red para así gestionarla eficientemente.

Construcción de la topología y rutas

La topología de la red se representa con un grafo no dirigido. Sus nodos están conectados mediante enlaces que pueden variar dependiendo de los reportes que manden ellos mismos al controlador. Los enlaces se pueden añadir o eliminar.

```
def add_link(self, n1, n2, cost=1):
        self.graph.setdefault(n1, {})[n2] = cost
        self.graph.setdefault(n2, {})[n1] = cost
```

```
 def remove_link(self, n1, n2):
        if n2 in self.graph.get(n1, {}):
            del self.graph[n1][n2]
            del self.graph[n2][n1]
            print(f"[Controller] Enlace {n1}-{n2} eliminado.")
        if self.current_path:
            for a, b in zip(self.current_path, ...
             ... self.current_path[1:]):
                if (a, b) == (n1, n2) or (a, b) == (n2, n1):
```

```
                            self.current_path = self.get_path(…)
                    break
        return self.current_path
```

Para la búsqueda de rutas se eligió utilizar el algoritmo de Dijkstra con pesos, lo que permite encontrar el camino más corto y seguro teniendo en cuenta los distintos pesos que se le pueden asignar a los nodos

Esto se explicará más en detalle posteriormente cuando se trate el apartado de enrutamiento dinámico.

Monitorización de reportes

El controlador ejecuta un servidor UDP que recibe periódicamente reportes de los nodos. Cada reporte está compuesto por el QBER y el SKR del enlace, cuando el valor de cualquiera de estos dos parámetros supere un límite preestablecido, el controlador sospechará de un potencial ataque por lo que eliminará la conexión entre los nodos afectados y recalculando la ruta por un camino seguro.

```
if all(k in packet for k in ('from','to','qber','skr')):
    frm = packet['from']
    to = packet['to']
    qber = packet['qber']
    skr = packet['skr']
    # Proceso de análisis...
```

```
def handle_report(self, n1, n2, qber, skr):
    if qber > self.QBER_MAX or skr < self.SKR_MIN:
        return (True, self.remove_link(n1, n2))
    # ...restauración si mejora...
```

Orquestación del envío de mensajes

Cuando un nodo solicita enviar un mensaje a otro nodo vecino, la petición llega al controlador, este, calcula la ruta actual y coordina el reenvío del mensaje "hop-by-hop" a través de los nodos intermedios presentes en la ruta calculada. Para esto el controlador SDN envía comandos a cada nodo a través de un puerto administrativo.

```
if packet.get('action') == 'send_message' and 'src' in packet:
    src = packet['src']
    dst = packet['dst']
```

```
    msg = packet['message']
    # Si hay ruta, se reparten instrucciones nodo a nodo
    for i in range(len(self.current_path) - 1):
        node = self.current_path[i]
        next_hop = self.current_path[i+1]
        # Se omite la gestión de sockets
```

Este diseño hace que el canal sea mas seguro ya que se adapta e tiempo real a cada situación de la red.

Ejecución del servidor y ciclo de vida

El servidor UDP del controlador se ejecuta en un hilo dedicado y así se posibilita un procesamiento concurrente de reportes y comandos.

```
def start_server(self):
    sock = socket.socket(socket.AF_INET, socket.SOCK_DGRAM)
    sock.bind((self.listen_ip, self.listen_port))
    def server_loop():
        while True:
            data, addr = sock.recvfrom(1024)
            # Procesamiento de cada paquete recibido
    threading.Thread(target=server_loop, daemon=True).start()
```

La inicialización del controlador, la construcción del grafo y el cálculo de la ruta inicial se realiza al lanzar el programa principal.

Logs del Programa

El programa en ejecución muestra por pantalla los datos que el controlador recibe, manda o calcula (si la opción -v de verbosidad está activa). Pero también estos reportes se pueden ver en el log de ejecución que crea el controlador al terminar el programa.

```
logfile = open("ControllerLog", "w")
    sys.stdout = Tee(sys.stdout, logfile)
    sys.stderr = Tee(sys.stderr, logfile)
```

3.1.2.3 Network_manager.py

Para realizar una simulación de una red QKD no solo necesitamos los componentes principales como los nodos QKD o controlador SDN, sino que necesitamos una forma de lanzar la infraestructura. Este es el rol de network_manager, automatizar el despliegue simultaneo de todos los componentes de la red, no solo los nodos y el controlador sino también la interfaz de administración que se definirá en el siguiente punto.

El controlador de la red inicia todos estos componentes en una sola sesión de terminal multiplexada (tmux). Tmux es una herramienta que se ha utilizado para poder visualizar en una misma terminal todos los nodos desplegados, el controlador SDN y la terminal de administración simultáneamente. Se ha elegido este enfoque ya que permite observar y controlar todos los procesos de la red en paralelo y porque es la manera más natural de eventualmente implementar este sistema en una red QKD real, ya que una interfaz gráfica o GUI no mejoraría la efectividad de aplicarlo en un entorno real, de hecho, solo retrasaría el desarrollo.

Carga de configuración y preparación del entorno

Este simulador se ejecuta de una manera intuitiva, sencilla y escalable, ya que soporta topologías complejas sin modificar el código, solamente habría que modificar el código de configuración. En concreto se inicial leyendo el archivo de configuración JSON donde se define la topología de la red, los parámetros de cada nodo y los detalles del controlador.

```
def _load_config(self):
    with open(self.config_path) as f:
        cfg = json.load(f)
    self.session = cfg.get("session_name", "qkd_network")
    self.controller_cfg = cfg.get("controller")
    self.nodes_cfg = cfg.get("nodes", {})
```

Así, network_manager.py es totalmente genérico y reutilizable para cualquier topología definida en el JSON, y se permite cambiar la configuración de la simulación con un simple cambio del archivo de configuración.

Construcción dinámica de los comandos de lanzamiento

A partir de la configuración cargada, el script ejecuta los comandos necesarios para ejecutar e iniciar cada componente. Primero se lanza el controlador SDN luego todos los nodos QKD y finalmente la herramienta de administración de la red.

```
ctrl_cmd = (
    f"python3 SDNController.py --config {self.config_path}"
    f" --ip {ip} --port {port}"
```

```
)
cmds.append(ctrl_cmd)


for name in self.nodes_cfg:
    node_cmd = f"python3 QKDNode.py –name {name} --config {self.config_path}"
    cmds.append(node_cmd)


cmds.append(f"python3 admin.py --config {self.config_path}")
```

Orquestación con tmux

Este script aprovecha la capacidad de tmux para lanzar todos los procesos en una única sesión de la terminal, dividiendo la ventana en múltiples paneles de forma cuadriculada.

```
self._run_tmux([
    "new-session", "-d",
    "-s", self.session,
    "-n", "main",
    "bash", "-lc", first
])
# Luego, para cada comando adicional:
self._run_tmux([
    "split-window", "-h", "-t", f"{self.session}:0", "bash", "-lc", cmd
])
# o en vertical si horizontal falla
```

Cada vez que se añade un nuevo panel, el script reorganiza la distribución en formato mosaico para optimizar el espacio.

Ejecución del programa

Este es el archivo que se encarga de levantar la red y todos sus componentes, es el archivo de ejecución del programa. Lanzar roda la red con una configuración en concreto se realiza de la siguiente forma.

```
python3 network_manager.py config.json
```

El programa está pensado para ejecutarse en Linux, por lo que, si el sistema es Windows, es un requisito tener algún entorno virtual de ejecución como WSL o una máquina virtual Linux.

3.1.2.4 Admin.py

Para interactuar con el entorno se simulación es esencial tener una herramienta de administración de la red. Admin.py proporciona una interfaz de línea de comandos (CLI) donde el usuario puede, de manera centralizada, inyectar y restaurar intrusiones en los enlaces existentes, y solicitar el envío de mensaje desde un nodo origen al destino.

Carga de configuración

Admin.py se apoya en el archivo de configuración JSON para conocer la topología de la red, parámetros de los nodos, puertos de administración y enlaces disponibles.

```
with open(args.config) as f:
    cfg = json.load(f)
nodes_cfg = cfg.get('nodes', {})
links = load_links(nodes_cfg)
```

Presentación de opciones

La CLI está diseñada para ser simpe y concisa. El menú permite elegir entre simular una intrusión, restaurar un enlace caído, enviar un mensaje o salir.

```
print("\n*** ADMIN QKD CLI ***")
print(" [1] Atacar/restaurar enlace")
print(f" [2] Solicitar nodo {src} envíe mensaje -> {dst}")
print(" [3] Salir")
```

Simulación de ataques

Cuando el usuario selecciona la opción de inyectar o restaurar una intrusión, este puede escoger entre los enlaces de la red. El script liego envía comandos UDP a los puertos de administración de los nodos implicados y así activa o desactiva la intrusión en un enlace en concreto.

```
for src_n,dst_n in ((a,b),(b,a)):
    info = nodes_cfg.get(src_n, {})
    port = info.get('admin_port', info.get('port')+10000)
    cmd = json.dumps({"action":action, "neighbor":dst_n})
```

```
    with socket.socket(socket.AF_INET, socket.SOCK_DGRAM) as s:
        s.sendto(cmd.encode(), (ip, port))
```

En los nodos afectados, se cambia su indicador interno de intrusión, lo que implica en la degradación de las métricas reportadas (QBER y SKR) y cuando el controlador SDN reciba el siguiente reporte, se dará cuenta de estos valores inusuales y eliminaría el enlace afectado.

Envío de mensaje

En cuanto a la funcionalidad de envío de mensajes, se envía un comando UDP al puerto administrativo del nodo origen, donde se solicita el envió del mensaje que está definido en la configuración. El nodo origen, inicia el proceso de comunicación a través del controlador que decidirá su ruta.

```
cmd = json.dumps({"action":"send_message","dst":dst,"message":message})
with socket.socket(socket.AF_INET, socket.SOCK_DGRAM) as s:
    s.sendto(cmd.encode(), (ip_src, admin_port_src))
```

3.1.2.5 Topología.json

La topología es el único input requerido en el programa. Ha que tener una estructura concreta para que el programa pueda digerirlo, a continuación la explicaré. En las pruebas se diseñaron 3 topologías distintas, una de prueba con una estructura sencilla en forma de rombo, y 2 mas complejas. La primera es una topología robusta que aunque se caigan varios enlaces, lo mas probable es que siga habiendo conexión directa del nodo origen al nodo destino. La segunda modela la topología real de la red QKD de Madrid.

Estructura general

El archivo se organiza en varias secciones clave. Un ejemplo simplificado sería el siguiente

```
{
  "session_name": "qkd_network",
  "report_interval": 2,
  "controller": { ... },
  "nodes": { ... },
  "simulation": { ... },
  "management": { ... }
}
```

Desglose de las secciones

1. **Parámetros de sesión y control**

```
"session_name": "qkd_network",
"report_interval": 2,
```

session_name: Nombre de la sesión para la orquestación en tmux

report_interval: Frecuencia con la que los nodos hacen reportes al controlador (en segundos)

2. **Controlador SDN**

```
"controller": {
  "ip": "0.0.0.0",
  "port": 6000,
  "qber_max": 0.11,
  "skr_min": 1
}
```

ip y puerto: socket el cual controlador escucha reportes y peticiones mediante UDP

qber_max y skr_min: Umbrales de seguridad usados para la detección de intrusiones

3. **Nodos y topología**

```
"nodes": {
  "A": {
    "port": 5000,
    "admin_port": 15000,
    "neighbors": {
      "C": { "ip": "localhost", "port": 5002, "seed": 11111 }
    }
  },
  ...
}
```

Cada nodo (A,B,C…) se define con

- **Port:** puerto tcp para recibir mensajes cifrados
- **admin_port:** puerto UDP para comandos administrativos
- **neighbors:** Vecinos directos, con dirección, puerto y semilla para generación de claves simétricas.

4. Configuración de la simulación

```
"simulation": {
  "source": "A",
  "destination": "H",
  "message": "clave123"
}
```

Source: nodo origen

Destination: nodo destino

Message: mensaje para enviar

3.1.2.6 ascii_visualizer.py y visualizer.py

Estos archivos son los encargados de pintar una representación de la red en ascii. Esta representación es dinámica, es decir, simula la red QKD y sus estados, como envíos de mensaje, intrusiones, etc.

Esta funcionalidad fue añadida específicamente para mostrar el estado y comportamiento de la red visualmente en la defensa del proyecto, aunque se podrá usar en un futuro también para realizar depuraciones o pruebas del código.

Representación gráfica

El dibujo se realiza en "ascii_visualizer.py" que contiene varias funciones para dibujar diferentes topologías. La función **print_topology()** dibuja la topología de la red, coloreando los enlaces que estén correctos en verde y aquellos que presenten una intrusión en rojo.

3.1.2.7 Flujo de Ejecución

El proceso se inicia con la ejecución del script **network_manager.py** junto con la topología, que se ejecuta de la siguiente manera:

```
python3 network_manager.py simple.json
```

Si el programa se ejecuta con la opción -h o --help, este devuelve las opciones que tenemos para la ejecución

```
usage: network_manager.py [-h] [-v] config

Lanza nodos QKD, SDNController y admin CLI en tmux según config JSON

positional arguments:
  config         Ruta al fichero JSON de configuración

options:
  -h, --help     show this help message and exit
  -v, --verbose  Lanza el controlador SDN en modo verbose
```

Imagen 5: Opciones para la ejecución del programa

Como podemos ver en la imagen superior, también existe la opción -v. Esta opción asigna el nivel de verbosidad del controlador. Si esta opción está presente, el controlador devolverá por pantalla los reportes que le mandan los nodos QKD, en cambio, si está desactivada la opción, estos reportes no aparecerán por terminal, pero se podrán ver en el fichero de logs del programa.

Continuando con el flujo de ejecución, primero se lee la configuración del JSON que define la topología, parámetros de cada nodo y detalles del controlador.

Luego se lanza una sesión de tmux donde crea un panel para el controlador SDN, uno separado para cada nodo y otro para la herramienta de administración.

Después se inicializan los componentes:

- El controlador SDN arranca un servidor UDP para escuchar reportes, construye el grafo interno de la red y calcula la ruta inicial
- Cada nodo QKD arranca un servidor TCP para mensajes cifrados y un UDP para la administración, genera para cada par de vecinos u conjunto de claves simétricas usando la semilla compartida y se inician los hilos que reportan el estado del QBER/SKR al controlador
- La herramienta de administración muestra por pantalla las opciones y espera interacción del usuario.

Ejemplo

A continuación, se explicará un ejemplo típico de ejecución del programa justo después de la inicialización antes mencionada.

1. Los nodos comienzan a reportar su estado al controlador.
2. El usuario solicita un envío de mensaje de A a H:
   - EL controlador coordina la transmisión segura según la ruta disponible.

3. El usuario inyecta una intrusión en el enlace C-D:

   - Los nodos C y D reportan métricas anómalas.
   - El controlador lo detecta y elimina el enlace C-D, recalculando las rutas.
   - Si se solicita de nuevo el envío de mensaje de A a H, la ruta será diferente si existe una alternativa segura.

4. El usuario restaura el enlace C-D:

   - Los nodos C y D comienzan a reportar métricas sanas.
   - El controlador lo detecta, y vuelve a recalcular la ruta por el camino mínimo.

En la siguiente imagen se muestra un ejemplo de ejecución del programa, desde que se inicia, hasta que comienzan los módulos a comunicarse entre ellos. El siguiente ejemplo muestra un ejemplo de intrusión en un nodo.

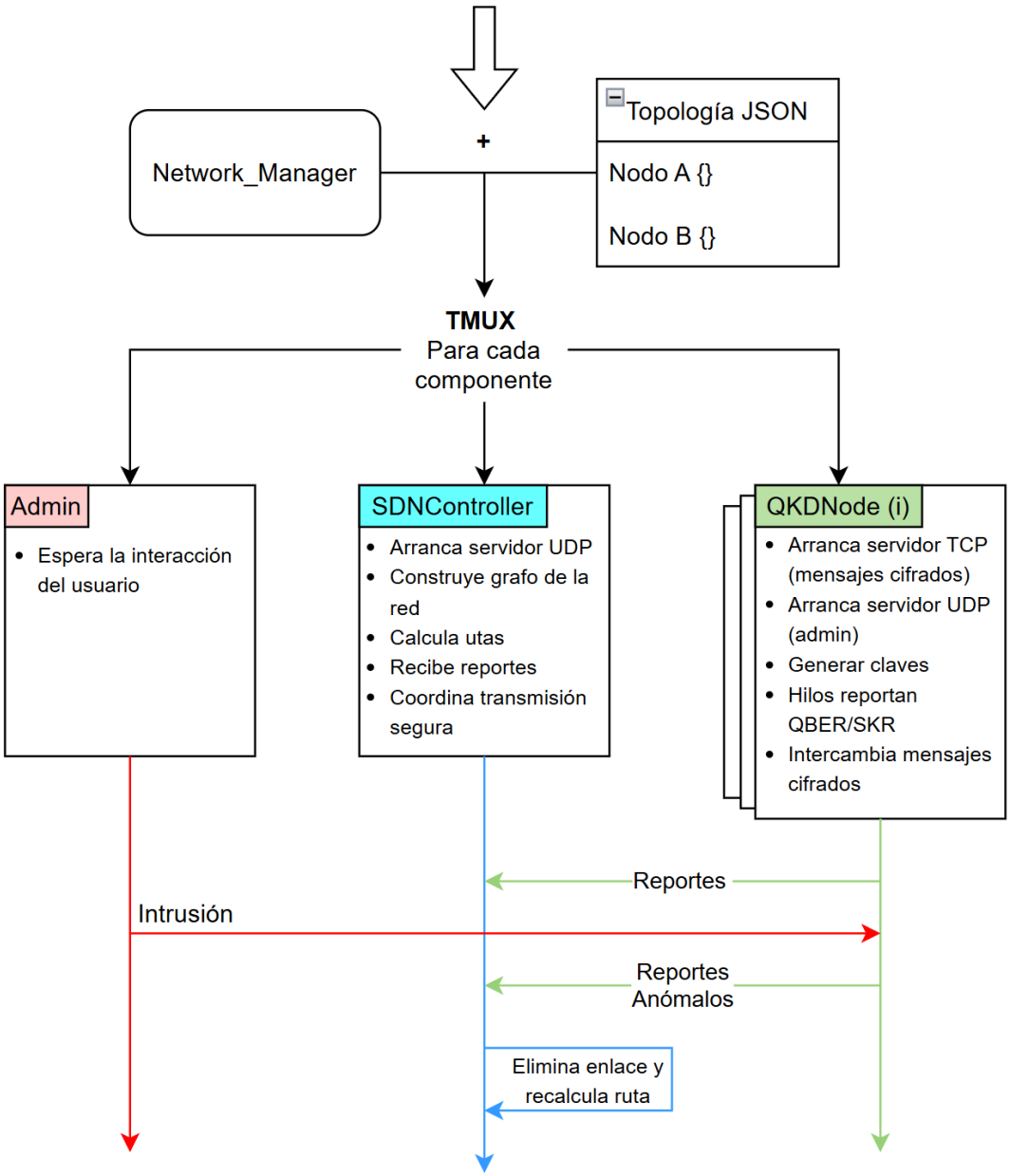


Imagen 6: Flujo de ejecución del programa

Comunicación entre procesos

Nodos QKD ↔ Controlador SDN: UDP para reportes de métricas.

Nodos QKD ↔ Nodos vecinos: TCP para transmisión de mensajes cifrados.

Herramienta de administración ↔ Nodos QKD: UDP para inyección/restauración de intrusiones y comandos de envío.

Nodo origen ↔ Controlador SDN: UDP para solicitar rutas y envío de mensajes.

3.1.3 Enrutamiento Dinámico

En las redes QKD la seguridad y eficiencia en la transmisión de información dependen en gran medida del estado de los enlaces y la capacidad de la red para adaptarse ante ataques o fallos. El **enrutamiento dinámico** es un mecanismo esencial para garantizar la continuidad y seguridad del servicio permitiendo que los mensajes lleguen a su destino incluso en casos de compromiso.

En la simulación, este mecanismo se implementa de manera centralizada en el controlador SDN, con todos los componentes interactuando según el estado de la red.

3.1.3.1 Implementación en el simulador

- **Supervisión y reporte del estado de los enlaces**

Como se ha explicado anteriormente, cada nodo QKD ejecuta hilos de monitorización que envían mensajes periódicos al controlador

```
def report_loop(self, neighbor, base_qber=0.01, base_skr=1000, noise=0.005):
    ...
    report = {"from": self.name, "to": neighbor, "qber": qber, "skr": skr}
    sock.sendto(json.dumps(report).encode(), (self.ctrl_ip, self.ctrl_port))
```

- **Decisión centralizada**

El controlador SDN recibe todos estos reportes y decide (usando el método "handle_report"), si un enlace sigue siendo seguro o debe ser eliminado.

```
def handle_report(self, n1, n2, qber, skr):
    if qber > self.QBER_MAX or skr < self.SKR_MIN:
        return (True, self.remove_link(n1, n2))
    # Restaurar enlace si vuelve a ser seguro...
```

Cuando un enlace es atacado, se elimina dinámicamente el grafo interno del controlador (con el método "remove_link"). Si en un futuro vuelve a ser seguro porque se reciben valores normales de QBER y SKR, se reincorpora el link (con "add_link")

- **Cálculo y actualización automática de rutas**

Como se puede observar, la función de añadir un nuevo enlace tiene un parámetro para el coste, en este caso el default es igual a 1.

```
def add_link(self, n1, n2, cost=1):
```

El controlador almacena la topología como un grafo con pesos y utiliza el algoritmo de **Dijkstra con pesos** en "get_path" para calcular la ruta más corta y segura entre el origen y el destino, cada vez que la topología cambia o se solicita un nuevo envío.

La función que modela el algoritmo de Dijkstra es la siguiente.

```
def get_path(self, start, end):
        """ Dijkstra."""
        pq = [(0, start, [])]
        visited = set()
        while pq:
            cost, node, path = heapq.heappop(pq)
            if node in visited:
                continue
            new_path = path + [node]
            if node == end:
                return new_path
            visited.add(node)
            for nei, w in self.graph.get(node, {}).items():
                if nei not in visited:
                    heapq.heappush(pq, (cost + w, nei, new_path))
        return None
```

- **Orquestación hop-by-hop**

Cuando se solicita enviar un mensaje desde el nodo origen este nodo llama a "send_to_controller".

```
def send_to_controller(self, message, dst):
    cmd = {"action": "send_message", "src": self.name, "dst": dst,
"message": message}
    sock.sendto(json.dumps(cmd).encode(),             (self.ctrl_ip,
self.ctrl_port))
```

El controlador al recibir la petición (action: send_message), utiliza la ruta actual calculada y envía las instrucciones a cada nodo de la ruta para reenviar el mensaje al siguiente salto. Esto se implementa en el bucle de reenvío dentro de "start_server" de "SDNController.py".

```
for i in range(len(self.current_path) - 1):
    node = self.current_path[i]
    next_hop = self.current_path[i+1]
    # Enviar comando 'send_message' al nodo correspondiente...
```

3.1.3.2 Ventajas ante soluciones actuales

El enrutamiento dinámico gestionado por SDN en este proyecto aporta varias ventajas frente a las soluciones tradicionales de enrutamiento estático en redes QKD:

- **Adaptabilidad automática:** La red se reconfigura en tiempo real ante ataques o incluso falos, sin intervención manual, por lo que garantizando continuidad y resiliencia.

- **Visión global:** El controlador centralizado toma decisiones óptimas usando información actualizada sobre todos los enlaces, mejorando la seguridad y eficiencia.

- **Reacción inmediata ante incidentes:** Los enlaces comprometidos se excluyen automáticamente del enrutamiento, reduciendo la exposición a ataques y minimizando el tiempo fuera de servicio.

- **Escalabilidad y flexibilidad:** Cambios en la topología o la red pueden realizarse fácilmente a través de la configuración, así, se facilita la experimentación y la adaptación a nuevos escenarios.

3.2 Herramienta de análisis de vulnerabilidades

La herramienta de pentesting que se va a explicar a continuación, ha sido desarrollada con el objetivo de facilitar la evaluación de seguridad en infraestructuras QKD. Puede usarse como una herramienta individual para analizar la red y buscar posibles vulnerabilidades de esta, tanto como una extensión al protocolo de enrutamiento dinámico ya descrito. La combinación de las dos herramientas proporcionaría una mejora significativa de la seguridad en estas redes ya que no solo implementarán un método de redirección automático para evitar ataques de denegación de servicio, sino que tendrán una capa de seguridad extra al poder segurizar aún más la parte más vulnerable a ataques de estas redes QKD (los nodos de confianza) ya que son igualmente vulnerables a ataques de hacking clásicos.

Esta herramienta está destinada a usuarios que no tengan experiencia previa con la ciberseguridad, ya que genera un informe detallado especificando los

puertos y servicios abiertos y expuestos, las vulnerabilidades asociadas a estos, el CVV asignado (si existe) y una posible solución al problema.

El enfoque principal de la herramienta es automatizar la fase de reconocimiento y análisis de vulnerabilidades, que puede ser un proceso tedioso y complicado si no se dispone de los suficientes conocimientos de ciberseguridad.

La herramienta realiza un análisis en varias fases:

- **Escaneo de puertos** (TCP/UDP) para identificar servicios expuestos.
- **Detección y enumeración de servicios** disponibles en el objetivo.
- **Consulta automática de vulnerabilidades conocidas** (CVEs) para los servicios detectados, utilizando la base de datos pública del NIST.
- **Detección de posibles ataques Man-in-the-Middle (MITM)** a través de análisis de certificados SSL, ARP spoofing y servicios no cifrados.
- **Generación automática de un informe en PDF**, comprensible para usuarios no expertos, que resume tanto los hallazgos técnicos como las recomendaciones de remediación priorizadas por severidad.

3.2.1 Arquitectura de la Herramienta

Cada funcionalidad clave de la herramienta se implementa como una clase independiente. Esta estructura modular no solo facilita el mantenimiento del código, sino que también permite y facilita futuras integraciones de nuevas técnicas de análisis o la adopción a diferentes entornos de red.

La herramienta se divide en los siguientes componentes que se explicarán en detalle en los siguientes apartados:

- **Programa principal (main.py):** Orquesta la ejecución de los diferentes módulos, gestiona los argumentos de entrada y coordina el flujo de trabajo general.
- **Escáner de puertos (port_scanner.py):** Se encarga de identificar los puertos TCP y UDP abiertos en el objetivo y de enumerar los servicios asociados a cada puerto.
- **Detector de ataques Man-in-the-Middle (mitm_detector.py):** Analiza el entorno de red en busca de indicios de ataques MITM, tales como certificados autofirmados, anomalías en la latencia y posibles ataques de ARP spoofing.
- **Escáner de vulnerabilidades (cve_scanner.py):** Consulta la base de datos pública de vulnerabilidades del NIST (NVD) para identificar CVEs relevantes a los servicios detectados.
- **Generador de informes PDF (pdf_generator.py):** Sintetiza todos los resultados obtenidos en un informe sencillo y completo.
- **Módulo de CLI (cli.py):** Gestiona la entrada de argumentos por línea de comandos, validando parámetros como la IP objetivo, el rango de puertos y la configuración de salida.

A continuación, se explicará cada módulo por separado. Se omitirán grandes fragmentos de código debido a su longitud, pero se explicará el funcionamiento fundamental de cada clase.

3.2.1.1 Main.py

El archivo main.py es el programa inicial y el núcleo orquestador de la aplicación. Coordina la ejecución secuencial de los distintos módulos que se expondrán posteriormente, como el escaneo de puertos, análisis de vulnerabilidades, detección de ataques MitM y generación del informe final.

Este archivo se ejecuta desde una terminal, ha de ser Linux o como se explicó anteriormente se puede usar herramientas como WSL o máquina virtual para ejecutarlo.

La herramienta contiene un manual si se aplica la bandera "-h" o "--help" o si se ejecuta sin argumentos.

Una ejecución simple del programa se ve de la siguiente manera:

```
python3 main.py <IP a escanear> -o reporte.pdf
```

Ejecución del programa

Al iniciar el script procesa los argumentos de la línea de comandos gracias a la función "parse_arguments" del módulo **cli.py**. También muestra un banner característico de las herramientas de pentesting que no le proporciona ninguna funcionalidad extra pero si un poco de personalidad.

Antes de iniciar el análisis, el script verifica y crea los directorios necesarios para crear el informe. Así se garantiza que no se produzcan errores al guardar el pdf.

```
    # Crear directorios de salida si no existen
    output_dir = os.path.dirname(args.output_file)
    if output_dir and not os.path.exists(output_dir):
        os.makedirs(output_dir)
```

A continuación, se instancian los objetos de cada módulo implementado en el programa. Este procesamiento modular permitirá en un futuro expandir la herramienta con más funcionalidades.

Y finalmente se ejecuta la secuencia completa del "vulnerability assesment"

```
    try:
        # Escaneo de puertos
        print(f"[+] Iniciando escaneo en {args.target_ip}")
        # Detección de vulnerabilidades
        vuln_results = cve_scanner.scan(port_results.get('services'
        # Detección de MITM
        mitm_results = mitm_detector.scan(port_results)
```

```
        # Generación de reporte
        report_file = report_gen.generate(port_results,...
                      ... vuln_results, mitm_results)
        print(f"[+] Escaneo completado.
```

3.2.1.2 cve_scanner.py

Cve_scanner se encarga de automatizar la detección de vulnerabilidades conocidas (CVEs) para los servicios descubiertos en el objetivo analizado. Para ello, se conecta a la API oficial de National Vulnerability Database (NVD) del NIST y extrae la información relevante, clasificándola por severidad.

Inicialización y parámetros

Al instanciar la clase se permite especificar una API Key para evitar límites de uso y retardo entre peticiones.

```
class CVEScanner:
    def __init__(self, nist_api_key=None,...
                  ... max_cves_per_service=5, delay=1.5):
        self.api_url = ...
         ... "https://services.nvd.nist.gov/rest/json/cves/2.0"
        self.nist_api_key = nist_api_key
        self.max_cves = max_cves_per_service
        self.delay = delay
```

Escaneo de vulnerabilidades

El método scan() recibe un diccionario de servicios y para cada uno consulta la API de NVD para obtener los CVEs relevantes. Si la consulta inicial (nombre + versión) no devuelve resultados se intenta una búsqueda por nombre únicamente.

También se incluyen funciones auxiliares para obtener la descripción, puntuación CVSS (puntuación de criticidad), la severidad traducida y sugerencias para la remediación de cada CVE encontrado.

```
    def scan(self, services_dict):
        results = {
            'vulnerabilities': [],
            'critical_count': 0,
            'high_count': 0,
            'medium_count': 0,
```

```
            'low_count': 0
        }
       # Lógica del programa llamando a la API y recolectando
información…
        return results
```

3.2.1.3 cli.py

El módulo cli.py gestiona los argumentos de entrada que permiten personalizar el análisis: IP objetivo, archivo de salida, rango de puertos, timeout, y nivel de verbosidad. Además, valida la IP y ajusta la extensión del informe para evitar errores comunes del usuario.

```
parser = argparse.ArgumentParser(
        description="Herramienta de pentesting para redes QKD )
    parser.add_argument("target_ip", ...
    parser.add_argument("-o", "--output", ...
    parser.add_argument("-p", "--ports", ...
    parser.add_argument("-t", "--timeout", ...
    parser.add_argument("-v", "--verbose", ...
    args = parser.parse_args()
```

3.2.1.4 port_scanner.py

Este módulo es seguramente el más importante de la herramienta, se encarga de descubrir los puertos abiertos (TCP y UDP) y enumerar los servicios que corres sobre ellos en el nodo QKD. Utiliza la herramienta de nmap para escanear la IP o el rango de IPs definido, y extrae información sobre nombres, versiones y posibles banners de los servicios encontrados.

En su ejecución se realiza primero un escaneo rápido de todos los puertos TCP

```
def scan_tcp(self):
        print("[+] Escaneo de puertos TCP por rangos...")
        ranges = ["1-1000", "1001-10000", "10001-30000", "30001-
65535"]
…
output = self.run_command(["nmap", "-p", port_range, "-T4", "--min-
rate=5000", self.target_ip])
…
```

Luego se realiza un escaneo detallado de los puertos encontrados mediante la búsqueda de versión, escaneo con scripts genéricos y escaneo con scripts concretos para amoldarse más a la tecnología QKD.

```
def scan_detailed(self):
        …
        output = self.run_command([
            "nmap", "-sC", "-sV",
            "--script", "ssl-cert,ssh-hostkey,snmp-info,banner",
            "-p", self.tcp_ports, self.target_ip
        ])
```

Finalmente, un escaneo UDP de los puertos más comunes ya que escanear UDP es un proceso que consume mucho tiempo, por lo que reducir la cantidad de puertos también reduce el tiempo de ejecución significativamente,

```
def scan_udp(self):
        udp_ports = "53,67,68,69,123,161,162,500,514,520"
        output = self.run_command(["nmap", "-sU", "-p", udp_ports,
self.target_ip])
```

3.2.1.5 mitm_detector.py

Este módulo, es un módulo complementario que se decidió implementar por su gran utilidad en la red. Está diseñado para detectar posibles vectores de ataque Man-in-the-Middle.

Primeramente, se analizan los certificados ssl. El método “check_ssl_certificate()” analiza los puertos SSL y verifica si los certificados son autofirmados o están caducados, que son condiciones que facilitan un ataque MitM.

```
    def check_ssl_certificates(self, ports=[443, 8443]):
        for port in ports:
            # ... conexión y extracción del certificado ...
            if self.is_self_signed(cert):
                self.results['ssl_issues'].append({
                    'port': port,
                    'issue': 'Certificado autofirmado detectado',
                    'risk': 'ALTO - Facilita ataques MITM'
                })
            if self.is_certificate_expired(cert):
```

```
                self.results['ssl_issues'].append({
                    'port': port,
                    'issue': 'Certificado caducado',
                    'risk': 'MEDIO - Podría facilitar ataques MITM'
                })
```

También se analiza la tabla ARP local para detectar su una misma IP aparece con distintas MACs, esto es un síntoma típico de ataques MitM en redes locales.

```
def check_arp_spoofing(self):
        arp_table = self.run_command(["arp", "-a"])
## Comprobaciones de arp spoofing
```

Luego se realizan pings repetidos para detectar picos de latencia, lo que puede indicar la presencia de un atacante intermedio.

```
def check_latency_anomalies(self):
...
        for _ in range(5):
            ping_output = self.run_command(["ping", "-c", "1", ...
                          ... self.target_ip])
...
```

Finalmente se buscan puertos típicamente inseguros como Telnet, FTP y HTTP y los marca como vectores de ataque MitM si están abiertos.

```
def check_vulnerable_services(self, services):
       vulnerable_protocols = {'telnet': 23, 'ftp': 21, 'http': 80}
```

3.2.1.6 pdf_generator.py

Esta clase se encarga de crear el informe final en un PDF. Su objetivo es presentar los resultados del análisis de forma simple, clara y entendible para una persona que no es experta en ciberseguridad.

Generación del informe

El método “generate()” recibe los resultados del análisis de puertos, vulnerabilidades y MitM y organiza los elementos del PDF (portada, tablas, resúmenes, recomendaciones)

```
    def generate(self, port_results, ...
                 ... vuln_results, mitm_results=None):
        doc = SimpleDocTemplate(self.output_file, pagesize=letter)
        elements = []
        self.add_cover(elements, port_results, ...
                       ... vuln_results, mitm_results)
        # ... añade secciones técnicas ...
        doc.build(elements)
        print(f"[+] Reporte generado: {self.output_file}")
        return self.output_file
```

Secciones clave

- **Portada y resumen ejecutivo:** Con un resumen del objetivo, fecha, nivel de riesgo global y principales hallazgos.
- **Puertos y servicios:** Muestra una tabla con los puertos y servicios abiertos.
- **Vulnerabilidades encontradas:** Lista los CVEs encontrados con su nivel de criticidad, descripciones y enlace directo al CVE.
- **Riesgos MITM:** Presenta los riesgos encontrados de MitM como certificados inseguros, ARP spoofing…
- **Recomendaciones:** Finalmente se ofrece una lista de medidas preventivas recomendadas para reducir los riesgos detectados.

Capítulo 4

Resultados y Conclusiones

El en presente capítulo, se realizará un análisis de la ejecución de las dos soluciones propuestas para comprobar su funcionamiento y eficacia ante escenarios simulados. Debido a que durante estos meses la red MadQCI no se encontraba disponible para realizar pruebas reales, se ha desarrollado un entorno simulado que reproduce sus características principales. Esta simulación ha permitido validar el comportamiento de las herramientas en condiciones realistas controladas. Se mostrará también un ejemplo de uso de ambas soluciones aplicadas de forma simultánea, para evaluar su efectividad combinada frente a ataques potenciales.

4.1 Enrutamiento dinámico

Esta sección evalúa la eficacia del sistema de enrutamiento dinámico desarrollado, centrándose en su capacidad para mantener la disponibilidad de la red frente a caídas de enlace provocadas por fallos o ataques. El objetivo es comprobar que, ante un enlace marcado como no disponible, el sistema es capaz de recalcular rutas alternativas válidas de forma automática y rápida, garantizando la continuidad de la comunicación en la red QKD.

La validación se ha realizado en un entorno simulado que permite modelar distintas topologías y simular el comportamiento de la red bajo condiciones adversas. Para ello, se han empleado tres configuraciones de red: una topología simple en anillo, una red mallada inspirada en SECOQC, y la estructura representativa de la red QKD de Madrid. Cada una de estas topologías permite analizar distintos aspectos del algoritmo, como la resiliencia frente a fallos o la flexibilidad ante múltiples rutas posibles.

Para entender mejor el funcionamiento del programa, a continuación, se muestra un diagrama de flujo de UCs (casos de uso). En la imagen 7 presentada a continuación se muestra paso a paso que acciones se toman en cada momento del programa junto con condiciones que lo limitan.

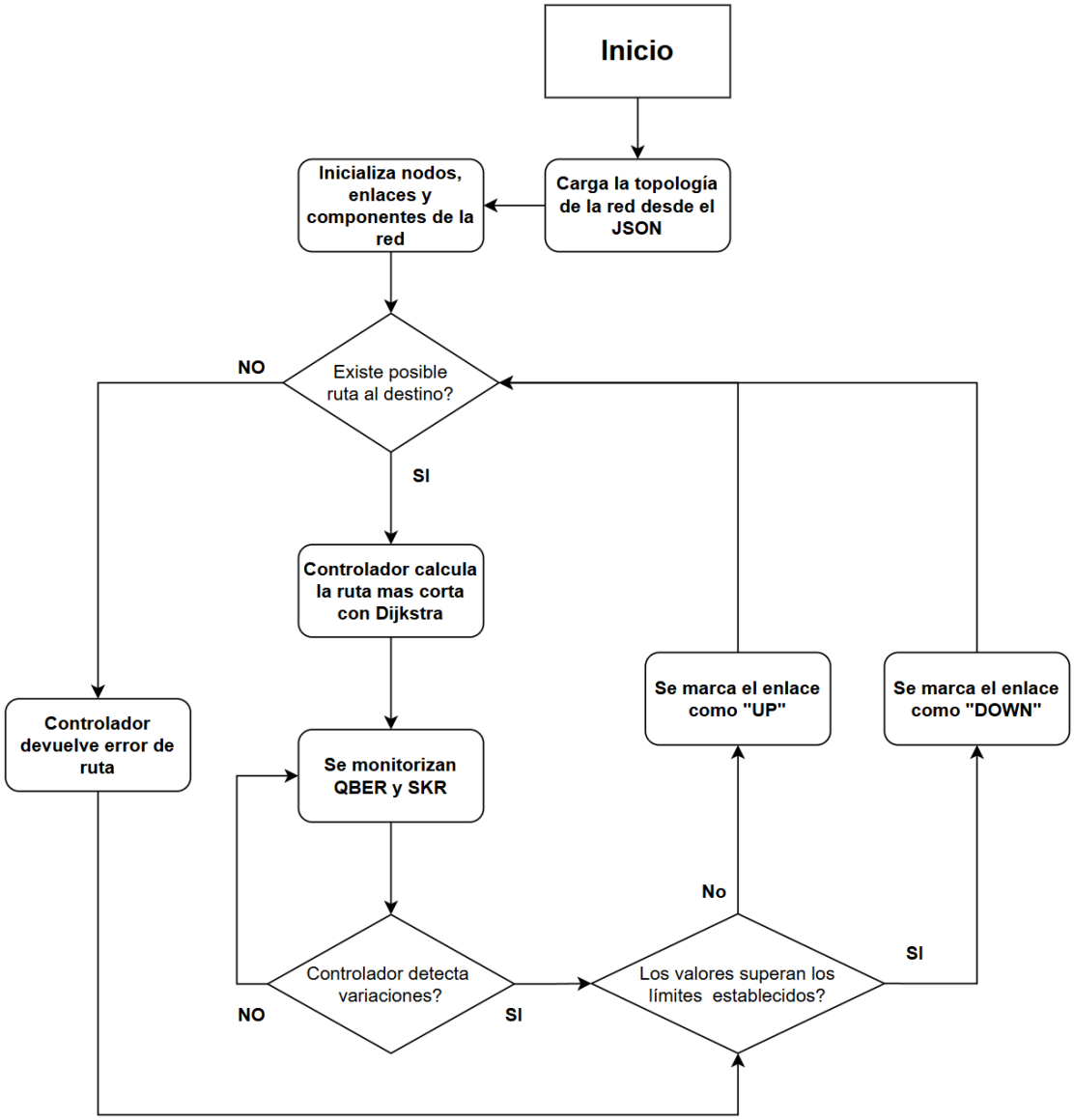


Imagen 7: Diagrama de UC del programa de enrutamiento dinámico.

Como hemos comentado anteriormente, estos programas están diseñados para ejecutarse en entornos Linux. En este caso se ejecutarán desde un sistema operativo Windows pero con WSL (Windows Subsystem for Linux) que permite ejecutar un entorno Linux directamente en Windows.

El fichero que se tiene que introducir como argumento es la topología de la red. Como se ha explicado antes, es un JSON donde está representado su configuración, nodos, mensajes, etc. A continuación, se muestra una representación gráfica de las topologías:

Simple.json:

Esta topología simple en forma de anillo está diseñada para que, si se compromete un camino, exista una única alternativa.

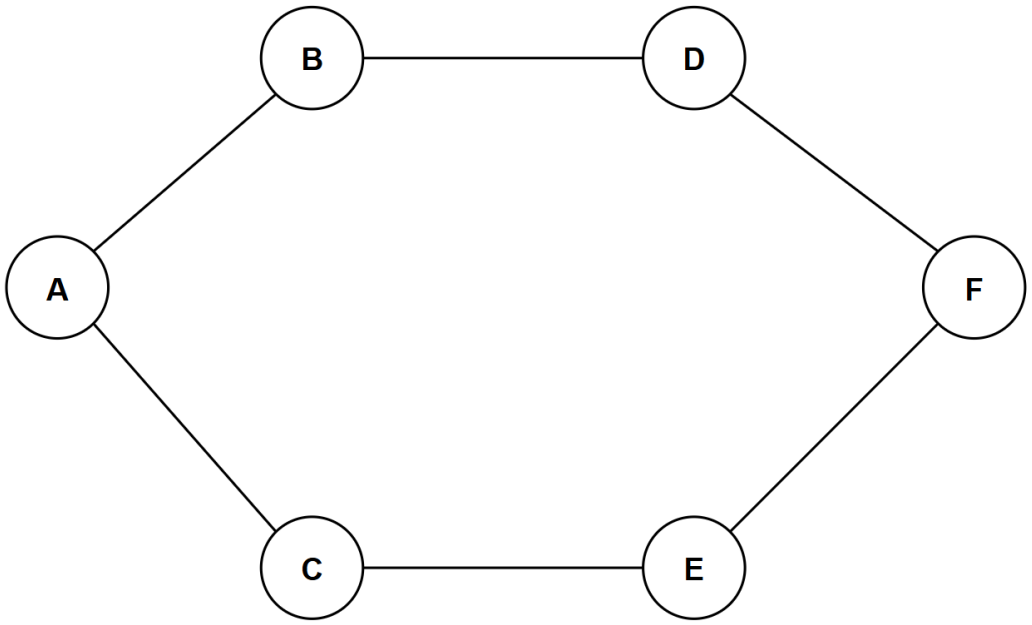


Imagen 8: Topología en forma de anillo.

Advanced.json:

Esta topología se ha incluido en el simulador por su similitud con la red SECOQC, una de las primeras redes metropolitanas de QKD. La ventaja de esta red es que existen muchas opciones de enrutar el mensaje en caso de caída de enlace.

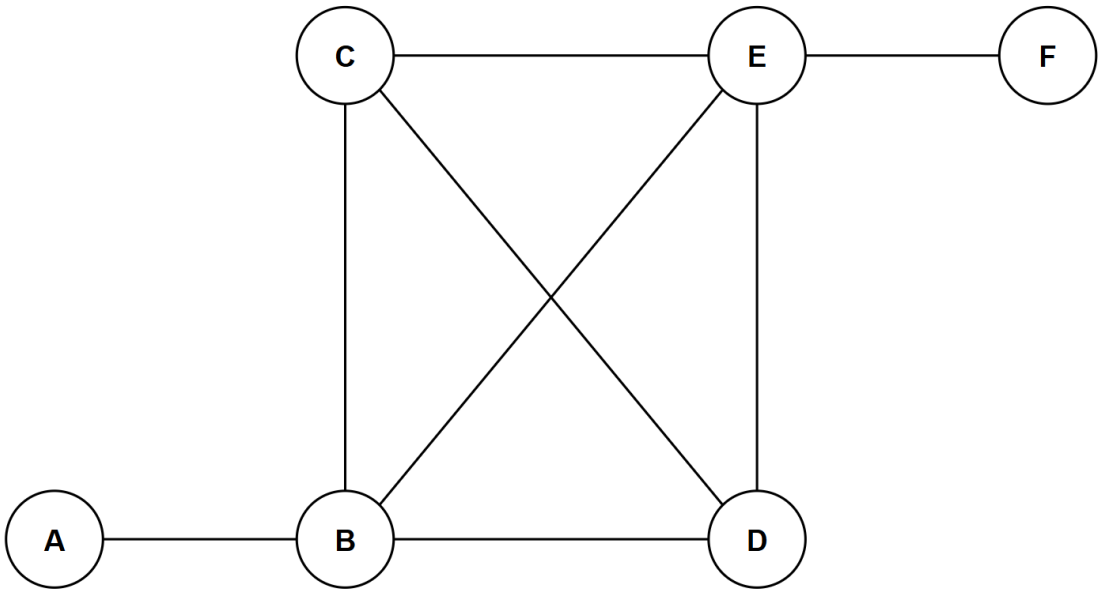


Imagen 9: Topología basada en SECOQC.

Madrid.json:

Esta topología representa la red QKD de Madrid.

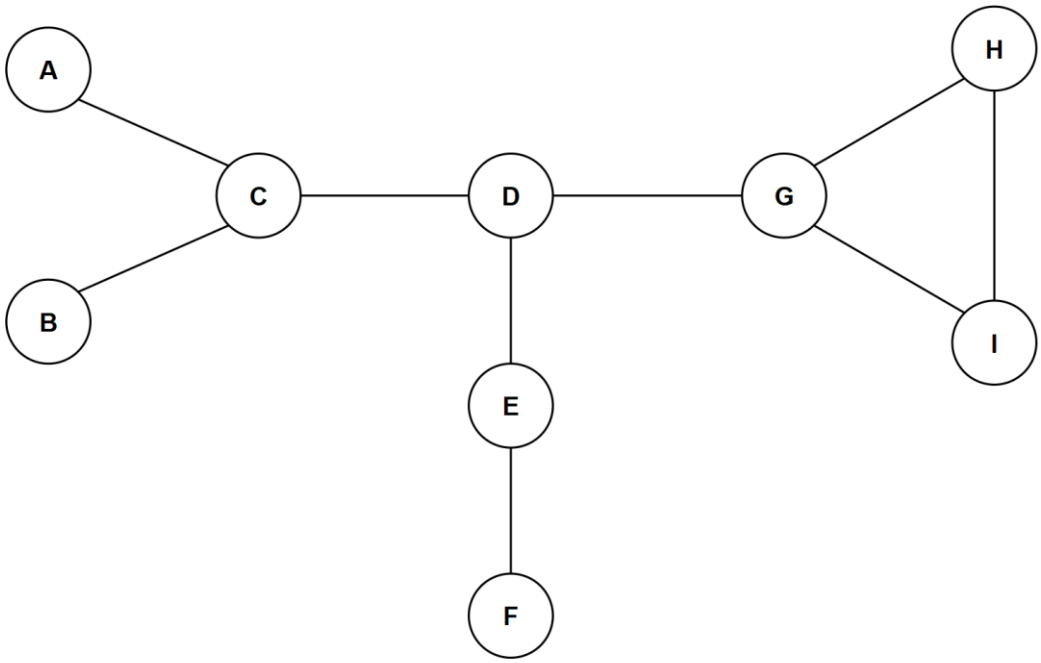


Imagen 10: Topología de la red QKD de Madrid.

Una vez ejecutado el programa, se puede ver la interfaz en formato de mosaico.

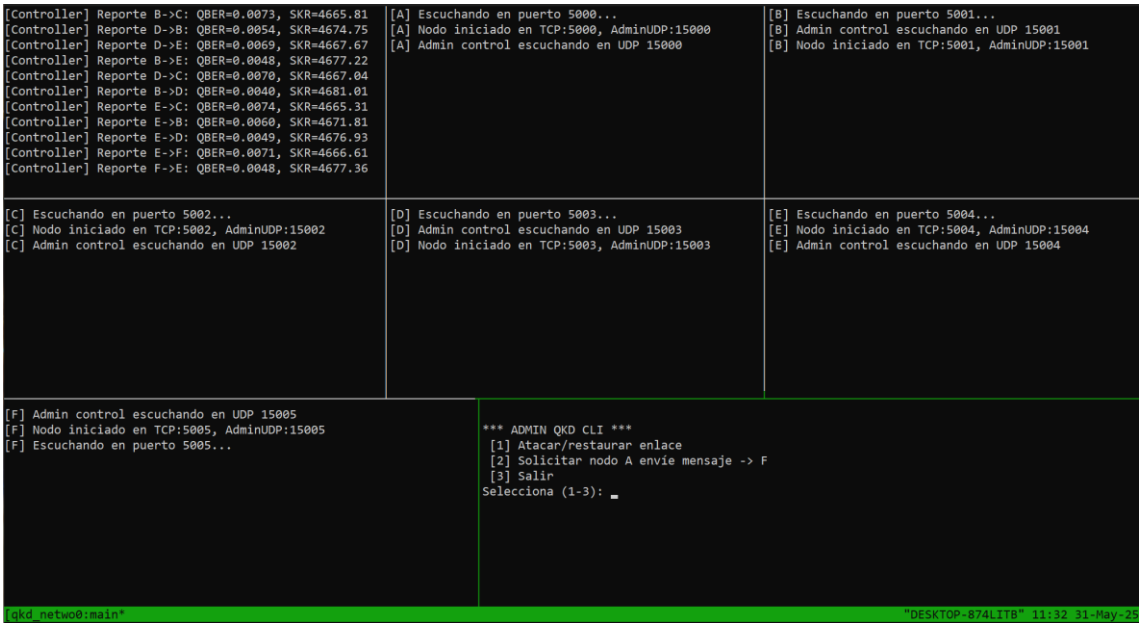


Imagen 11: Interfaz de línea de comandos del programa.

Nombrando de izquierda a derecha, y de arriba a abajo, se puede ver: El controlador recibiendo los reportes, los nodos A, B, C, D, E y F correspondientes

a la topología introducida (en este caso es advanced.json) y finamente la interfaz de Administración.

A continuación, se explica cada sección de la interfaz Tmux. En concreto se explicarán y realizarán pruebas con la interfaz del controlador, los nodos y la interfaz de administración de la red.

4.1.1 Interfaz de Administración

El funcionamiento de la interfaz de administración es sencillo. Esta interfaz ha sido desarrollada específicamente para comprobar el funcionamiento de la herramienta. Una vez asegurado el funcionamiento correcto esta interfaz no es útil ya que los ataques y envío de mensajes se realizan automáticamente por la red. Igualmente podrá servir como herramienta en un futuro si se deseara ampliar la funcionalidad o hacer pruebas.

Al ejecutar el programa se observan 3 opciones: Inyectar o restaurar el enlace, solicitar el envío del nodo A al F (se especifica en el archivo JSON) y la opción de terminar el programa.

En la imagen 12 se puede apreciar la interfaz de administración con las opciones previamente mencionadas. Esta interfaz es relativamente básica, pero es suficiente para el objetivo de la herramienta, que es comprobar el funcionamiento del enrutamiento dinámico en caso de ataques. De todas formas, debido a la estructura modular del código, es sencillo extenderla si en un futuro se deseara.

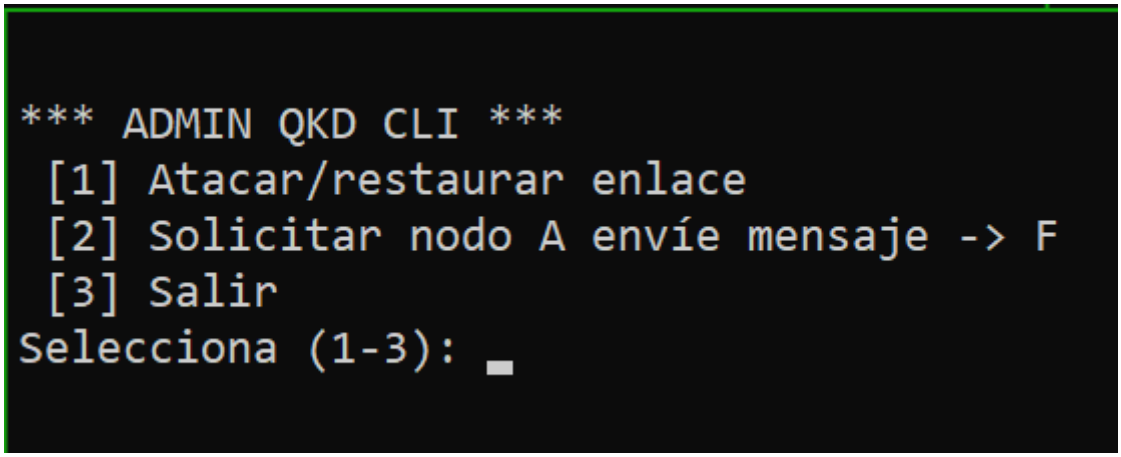


Imagen 12: Elección de opciones en CLI de Administración.

A continuación, comprobaremos el funcionamiento de la interfaz interactuando con las opciones.

Comencemos con la opción de atacar/restaurar el enlace. En la imagen 13 se pueden ver todas las opciones para atacar o restaurar, estos son todos los enlaces definidos en el JSON.

```
Selecciona (1-3): 1
Enlaces:
 [0] A-B
 [1] B-C
 [2] B-D
 [3] B-E
 [4] C-D
 [5] C-E
 [6] D-E
 [7] E-F
```

Imagen 13: Elección de enlaces en CLI de Administración.

En este caso se decide atacar el enlace A-B. En la imagen 14 se puede ver como al seleccionar en enlace A-B aparece la opción de inyectar o restaurar.

```
Índice: 0
1) Atacar  2) Restaurar
```

Imagen 14: Opciones de CLI de Administración.

Se elije a continuación atacar el enlace y una vez realizada la intrusión (índice 1), el controlador recibe valores extraños del SKR y QBER de los nodos afectados y elimina el enlace.

```
(1-2): 1
Acción 'intrusion' en enlace A-B
```

Imagen 15: Intrusión en CLI de Administración.

En el caso que se desee restaurar el enlace A-B hay dos opciones. La primera que el enlace esté caído, se restaure, y que el controlador calcule el nuevo camino mínimo, y la segunda, que el enlace no esté caído, en este caso el controlador ignora la petición.

En la siguiente imagen se muestra como se manda una petición de 'restore' al controlador para restaurar el enlace A-B.

```
1) Atacar  2) Restaurar
(1-2): 2
Acción 'restore' en enlace A-B
```

Imagen 16: Restaurar enlace en CLI de Administración.

Si se deseara enviar un mensaje, se ha de seleccionar la opción "[2] Solicitar nodo A envía mensaje -> F". Una vez realizada la solicitud de envío, el mensaje legará al destino si existe camino, si no existiera camino el controlador lo comunicará. Se puede ver el camino tomado por el mensaje en la interfaz de los nodos y en la del controlador que devuelve la ruta elegida.

La siguiente imagen muestra la solicitud de envío de mensaje que se realiza desde la interfaz de administración. En este caso se desea enviar desde el nodo A al nodo F (esto se define en el JSON de entrada).

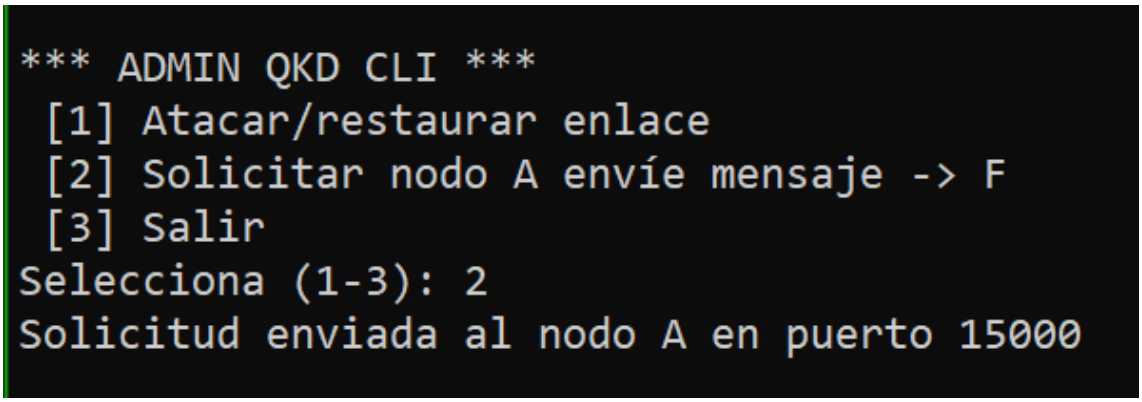


Imagen 17: Envío de Mensaje en CLI de Administración.

4.1.2 Nodos de la topología

La interfaz de los nodos muestra las intrusiones efectuadas, la restauración de los enlaces y el camino tomado por el mensaje.

En la imagen 18 se puede observar la interfaz Tmux para el nodo A y el nodo B. Se puede ver como los nodos comienzan a escuchar en los puertos 5000 y 5001 respectivamente e inician los puertos de administración y control. Posteriormente se puede observar como en el enlace A-B se aplica intrusión y aparece el mensaje correspondiente por pantalla en cada uno de los nodos afectados. Luego se restaura la intrusión y el mensaje aparece también en ambos nodos. Finalmente se desea enviar un mensaje de A a F, pasando por B, por lo que A crea la solicitud, que manda al controlador, y este, le envía a cada nodo afectado el siguiente salto que tienen que tomar.

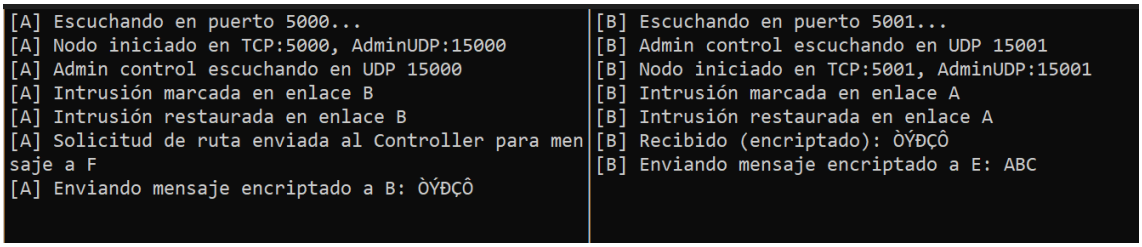


Imagen 18: CLI de los Nodos QKD.

Los mensajes concretos de intrusión y restauración son los siguientes:

- **Mensaje Intrusión:** [ID] Intrusión marcada en enlace <ID Vecino>
- **Mensaje Restauración:** [ID] Intrusión restaurada en enlace <ID Vecino>

En cuanto al camino tomando por los mensajes, se puede ver en la imagen 19, como los nodos que pertenecen a la ruta que estableció el controlador van recibiendo y mandando mensajes hasta llegar al destino, devolviendo por pantalla los mensajes:

- **Envió del mensaje:** [ID] Enviando mensaje encriptado a E: <Mensaje Encriptado>
- **Recepción del mensaje:** [ID] Recibido (Encriptado): <Mensaje Encriptado>

Se puede observar que la ruta tomada por el mensaje es la siguiente: A-B-E-F

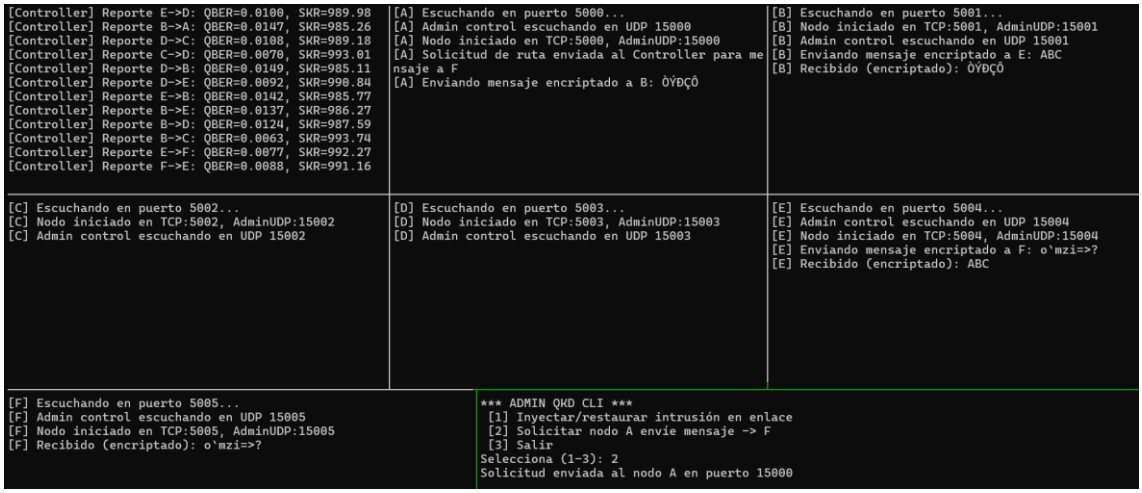


Imagen 19: Vista general de los nodos al enviar un menaje.

4.1.3 Controlador SDN

El controlador comienza calculando la ruta mínima inicial el empezar la ejecución del programa y también activa un servidor UDP para recibir los reportes de los nodos.

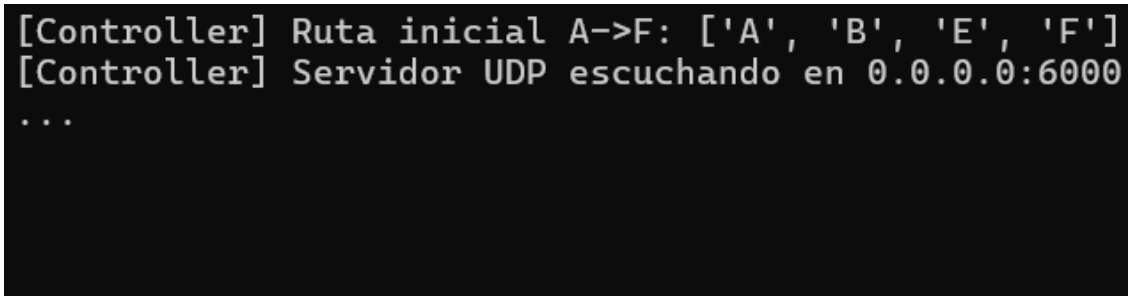


Imagen 20: CLI del Controlador SDN al iniciar el programa.

Luego comienzan a llegar los reportes. Estos incluyen el enlace origen, y valores del QBER y SKR.

```
[Controller] Reporte D->E: QBER=0.0070, SKR=4667.08
[Controller] Reporte B->C: QBER=0.0061, SKR=4671.30
[Controller] Reporte B->E: QBER=0.0048, SKR=4677.31
[Controller] Reporte B->D: QBER=0.0063, SKR=4670.46
[Controller] Reporte B->A: QBER=0.0065, SKR=4669.48
[Controller] Reporte A->B: QBER=0.0049, SKR=4677.04
[Controller] Reporte F->E: QBER=0.0044, SKR=4679.54
[Controller] Reporte E->F: QBER=0.0071, SKR=4666.40
[Controller] Reporte E->C: QBER=0.0044, SKR=4679.51
[Controller] Reporte E->B: QBER=0.0052, SKR=4675.35
[Controller] Reporte E->D: QBER=0.0055, SKR=4673.98
```

Imagen 21: Reportes enviados al controlador.

Cuando se ejecuta la solicitud de envío de mensaje, lo recibe el controlador, comprobando que exista una ruta y comunicando a los nodos correspondientes cual es el envío que deben realizar.

```
[Controller] Solicitud de mensaje de A a F: 'clave12
3'
[Controller] Enviado a nodo A: next B
[Controller] Enviado a nodo B: next E
[Controller] Enviado a nodo E: next F
```

Imagen 22: CLI del Controlador al enviar un mensaje.

Si el controlador recibe algún valor de QBER o SKR fuera de los límites, el controlador elimina el enlace y busca dinámicamente la ruta más corta con Dijkstra y a devuelve por pantalla.

```
[Controller] Enlace D-E eliminado.
[Controller] Ruta actualizada: ['A', 'B', 'E', 'F']
```

Imagen 23: Detección de intrusión en un enlace y actualización de ruta

Si se solicita restaurar en enlace desde la terminal administrativa, este se restaura y se actualiza la ruta.

```
[Controller] Enlace D-E restaurado.
[Controller] Ruta actualizada: ['A', 'B', 'E', 'F']
```

Imagen 24: Restauración del enlace y actualización de ruta.

Podría existir el caso en el que se elimine un enlace y no exista ruta posible. En este caso devuelve “None” como ruta.

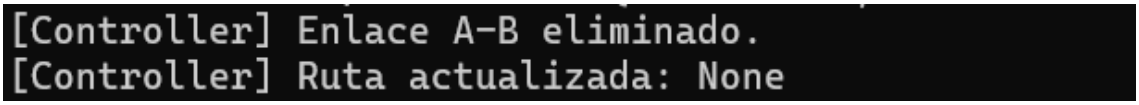


Imagen 25: Detección de intrusión, pero no existe ruta posible al destino.

Posteriormente si se desea enviar un mensaje, pero no hay ruta disponible, se devuelve un mensaje donde se puede ver que se ignora la solicitud.

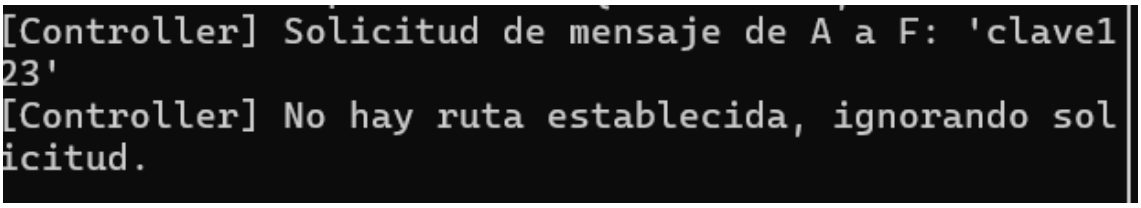


Imagen 26: Envió de mensaje al destino sin una posible ruta.

Finalmente, una vez termina la ejecución del programa, podemos comprobar el log que genera el controlador “ControllerLog.txt”. Aquí se muestran todos los mensajes enviados y recibidos por el controlador.

La imagen 27 muestra el fichero ControllerLog después de efectuar un envío de mensaje de A a F.

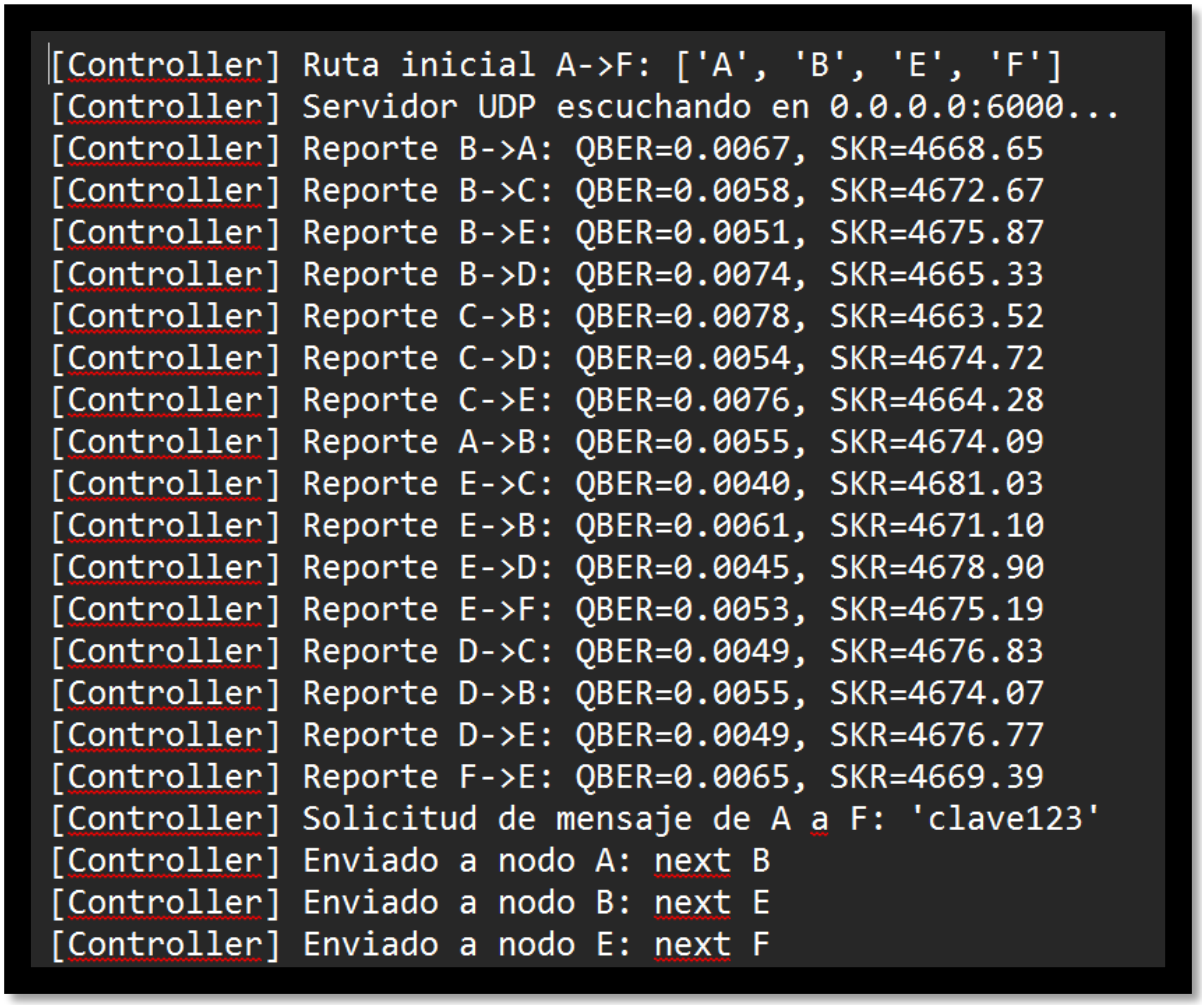


Imagen 27: Log del controlador.

4.1.4 Representación ASCII de la topología

La topología de la red es mostrada en otra ventana de tmux por separado, por simplicidad, se mostrará el funcionamiento del simulador ASCII correspondiente a la topología simple. Esta simulación es ASCII ya que las máquinas de las redes remotas en entornos reales se manejan en consola, no hay interfaz gráfica, por lo que se ha optado a realizar todo dentro de la misma terminal y dividir las funcionalidades en distintas ventanas de Tmux.

La imagen 28 muestra el simulador ASCII de la topología simple de la red. Los enlaces verdes indican que están activos actualmente.

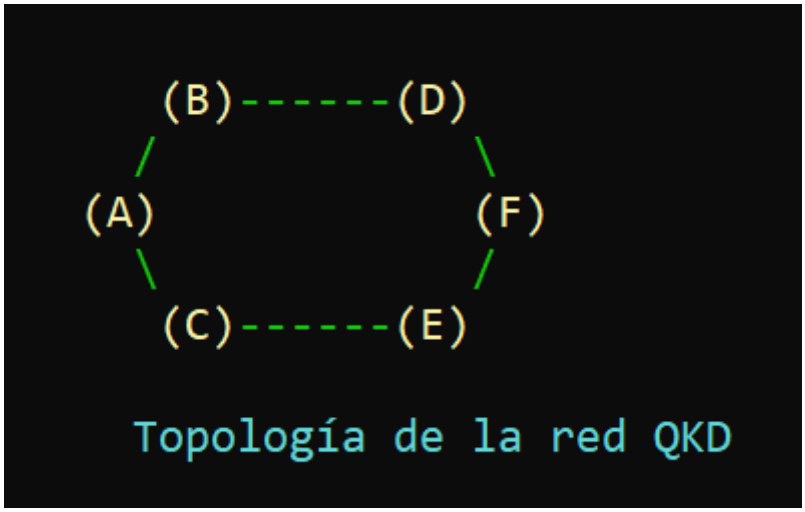


Imagen 28: Representación ASCII de la red.

Al enviar un mensaje desde “A” hasta “F” podemos ver como los nodos involucrados en el enrutamiento se dibujan de azul durante unos segundos.

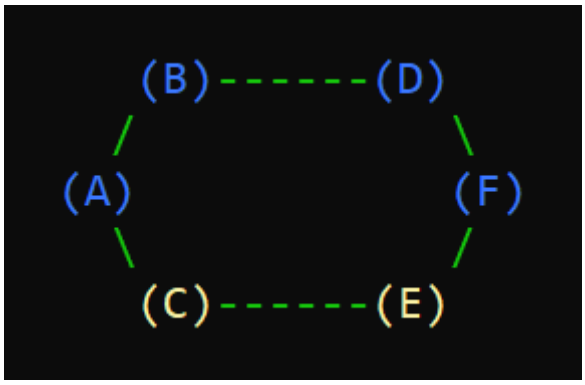


Imagen 29: Envío de mensaje de “A” a “F”

Al ejecutar el comando de atacar enlace desde la terminal de administración se puede observar que cuando el controlador recibe los reportes en enlace se marca de color rojo.

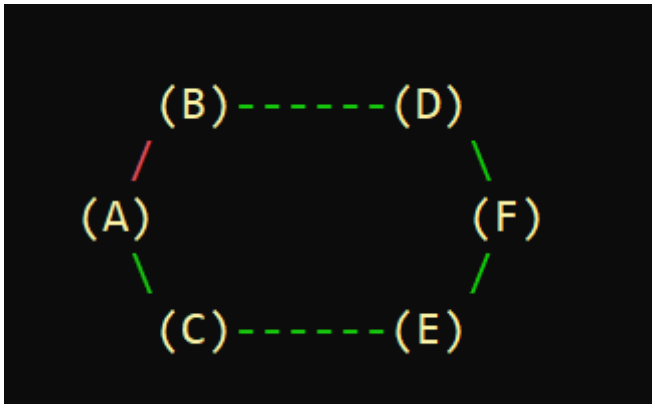


Imagen 30: Visualización de ataque de en A-B

Y posteriormente si se desea enviar un mensaje se enrutará el mensaje por otro camino.

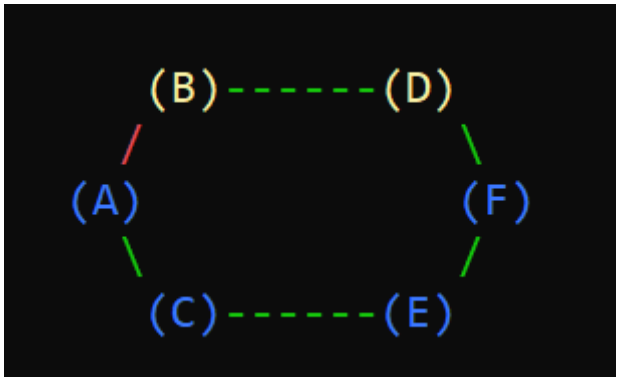


Imagen 31: Envío de mensaje con enrutamiento

4.2 Pentesting Tras el Ataque

Después de detectar la intrusión en la red y redirigir el envío de los datos, se usará la herramienta de pentesting para analizar la red en búsqueda de posibles vulnerabilidades, brechas de seguridad como puertos abiertos innecesarios y posibles indicios de que un atacante se encuentra en la red realizando el ataque de MitM.

A continuación, se muestra un diagrama de flujo de UCs correspondiente al programa de pentesting. Se puede ver que se sigue un flujo relativamente sencillo y directo sin muchas desviaciones, ya que el programa se dedica a buscar todas las vulnerabilidades posibles sin exclusión.

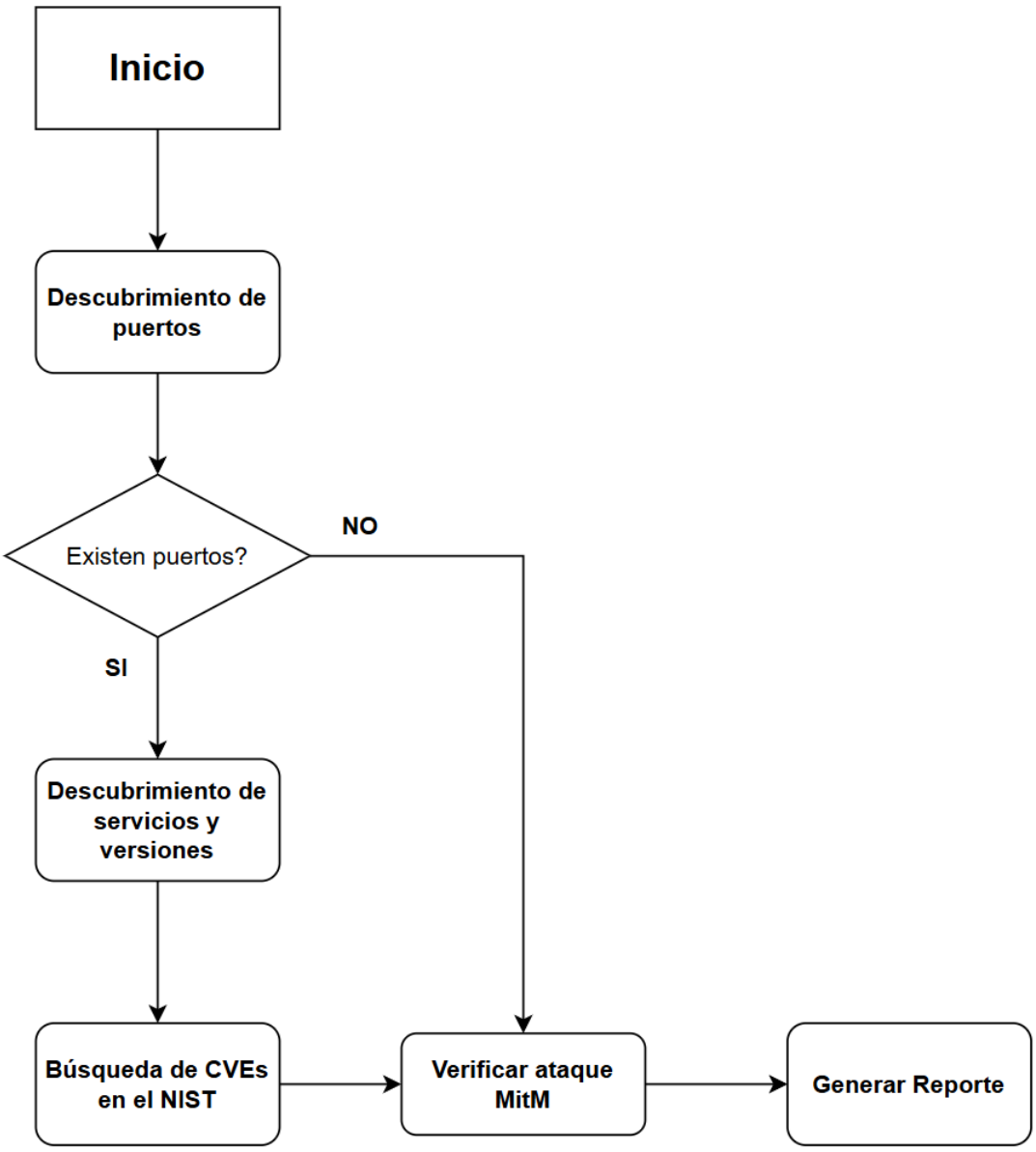


Imagen 32: Diagrama de UCs de la herramienta de pentesting

La herramienta comienza analizando los puertos abiertos para detectar servicios, versiones antiguas, puertos innecesarios o inseguros, etc. Luego utiliza las versiones y servicios encontrados para asignarle un CVE buscando en la base de datos del NIST, este CVE es el ID de la vulnerabilidad y va asociado a un nivel de riesgo. A continuación, realiza un escaneo de ataque MitM para analizar la red y averiguar si el ataque está siendo efectuado, y finalmente crea un reporte sencillo con todos los descubrimientos en un formato legible para personas inexpertas en la ciberseguridad.

La herramienta está diseñada para ser ejecutada con un solo comando, luego se genera un reporte cuyo nombre es indicado en el comando con la bandera "-o".

Para ver las opciones de ejecución ejecutamos con la bandera -h / --help o directamente sin argumentos.

```
usage: main.py [-h] [-o OUTPUT_FILE] [-p PORT_RANGE] [-t TIMEOUT] [-v] target_ip

Herramienta de pentesting para redes QKD y análisis de nodos confiables

positional arguments:
  target_ip             Dirección IP del nodo objetivo a escanear

options:
  -h, --help            show this help message and exit
  -o OUTPUT_FILE, --output OUTPUT_FILE
                        Nombre del archivo de salida para el reporte (default: reporte_qkd_scan.pdf)
  -p PORT_RANGE, --ports PORT_RANGE
                        Rango de puertos a escanear (ejemplo: 1-1024, 22,80,443) (default: 1-1024)
  -t TIMEOUT, --timeout TIMEOUT
                        Tiempo de espera para conexiones en segundos (default: 1.0)
  -v, --verbose         Mostrar información detallada durante el escaneo (default: False)
```

Imagen 33: Opciones de ejecución del programa.

4.2.1 Ejecución del código

El código se ejecuta de la siguiente manera:

```
python3 main.py scanme.nmap.org -o reporte.pdf
```

Scanme.nmap.org es el dominio que vamos a escanear en este caso. Es una IP que proporciona Nmap de forma gratuita para realizar pruebas de su herramienta. Contiene varios puertos abiertos y vulnerabilidades que se pueden escanear.

El código nada más empezar a ejecutar, muestra un “banner” característico de herramientas de ciberseguridad, junto con el nombre, versión y usos del programa, que se podrán ir ampliando en un futuro.

Imagen 34: Banner inicial del programa.

Luego comienza el escaneo, mostrando el dominio o IP a escanear.

Primero se realiza un **escaneo TCP** completo por todos los puertos. Estos están divididos en secciones para agilizar el escaneo y ver por pantalla el progreso.

Luego usando la herramienta de Nmap (**Nmap Scripting Engine**) se escanean los puertos encontrados en detalle, en este caso el 21, 22, 80, 9929. A continuación, se realiza un scan UDP con los puertos comunes para que no se demore mucho. Después de encontrar los puertos abiertos se realiza una **búsqueda de CVEs**, donde se realiza una búsqueda con los resultados obtenidos en la base de datos del NIST para encontrar los CVEs y nivel de riesgo asociados. También se realizan varias comprobaciones para ver si hay riesgo de **ataques MitM.** Finalmente genera un **reporte** con toda la información recopilada.

A continuación, se muestra el resultado de un escaneo completo en la CLI donde se puede observar el proceso completo descrito previamente.

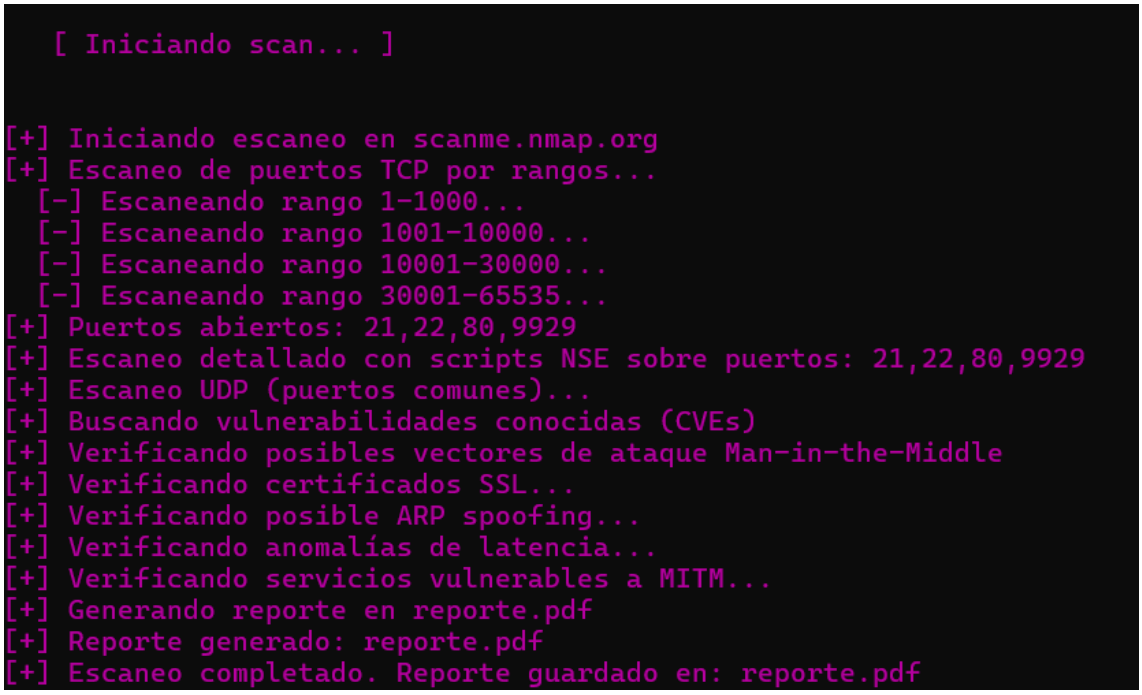


Imagen 35: Escaneo completo desde el CLI.

4.2.2 Reporte generado

El reporte generado tiene una estructura simple, con secciones claras para mejorar el entendimiento. Las imágenes del reporte están adjuntas en el anexo del documento.

Comienza con la fecha de creación y el domino o IP objetivo al que se ha realizado el escaneo. Contiene también una sección de Resumen Ejecutivo donde se mencionan por encima el número de vulnerabilidades encontradas con su peligrosidad asociada, y un nivel de riesgo general que tiene en cuenta lo previamente mencionado, junto con una breve recomendación. (La imagen correspondiente se encuentra en el Anexo A, Imagen 36).

La sección de **Puertos y Servicios Detectados** muestra una tabla con todos los puertos detectados en el escaneo, con sus servicios correspondientes y finalmente la versión. Estas dos últimas debido a limitaciones de la herramienta es posible que no siempre devuelvan un valor (Imagen en: Anexo A, Imagen 37).

La sección de **Vulnerabilidades Detectadas** es la sección que nos proporciona la herramienta de análisis por CVE. Aquí se muestran las vulnerabilidades encontradas, junto con su CVE (Imagen en: Anexo A, Imagen 38).

Luego se analizan las posibles vulnerabilidades para el ataque **Man-in-the-Middle** y se muestran en el reporte (Imagen en: Anexo A, Imagen 39).

Finalmente se muestra el apartado de **Recomendaciones** algunas soluciones básicas para remediar las posibles vulnerabilidades encontradas. Se puede observar que las vulnerabilidades con un CVE de nivel bajo no se muestran, esto es porque no tienen ningún impacto importante en el servicio, suelen ser falsos positivos y saturarían el documento demasiado con información innecesaria que gente inexperta podría ocupar su tiempo intentando remediar algo que realmente no es un problema (Imagen en: Anexo A, Imagen 40).

4.3 Combinación de las soluciones

La combinación de ambas herramientas proporciona una seguridad sólida en ataques clásicos a los nodos de confianza en la red. A continuación, se mostrará un ejemplo de flujo de uso sencillo de ambas herramientas simulando un entorno más realista.

El caso comienza con el programa de redirección corriendo en una red QKD real. El controlador estará escuchando los reportes que los nodos le mandan, pero repentinamente, uno de ellos sobrepasa los límites preestablecidos.

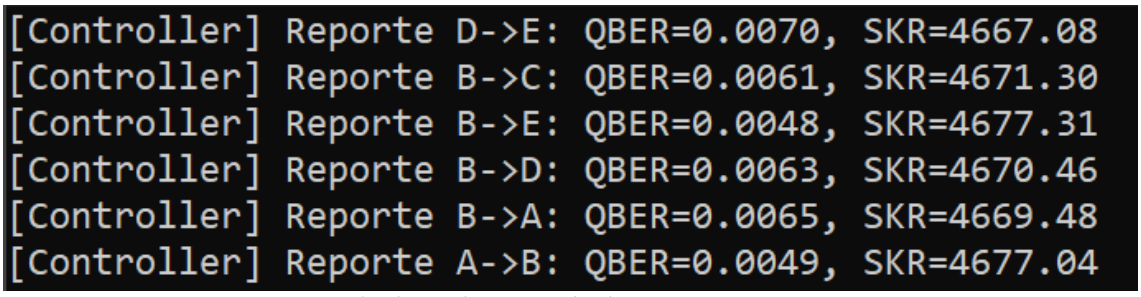


Imagen 41: Reportes enviados al controlador.

Por ello, el controlador elimina el enlace afectado, sea por un error, una interferencia o en el peor de los casos, un ataque.

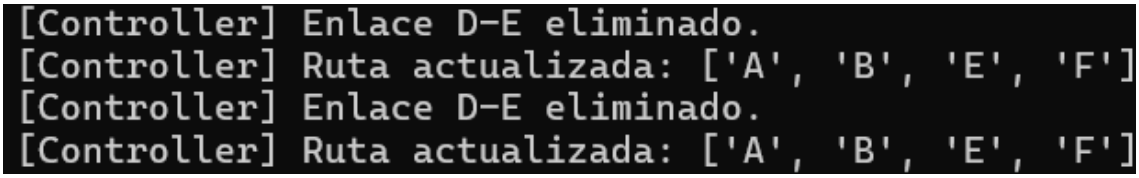


Imagen 42: Detección de intrusión en un enlace y actualización de ruta.

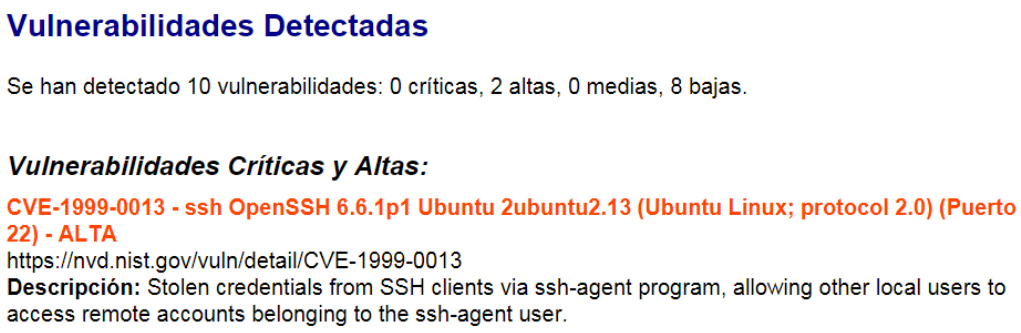
El mensaje de error llega a los operadores de la red y se les informa que el nodo está caído. Ellos tienen ahora varias opciones, recuperar inmediatamente el enlace si no tienen sospechas de ataque, esperar a que los niveles de QBER y

SKR vuelvan a niveles normales de nuevo y recuperar el enlace, o (el protocolo recomendado) esperar a niveles normales en los reportes y realizar un análisis de la red con la herramienta desarrollada.

Los técnicos deciden optar por la más segura ya que es la primera vez que el nodo da problemas y presenta un comportamiento bastante sospechoso. Por lo que ejecutan el código de la herramienta.

```
python3 main.py <IP Nodos afectados> -o reporte.pdf
```

Esta les devuelve un reporte con las principales vulnerabilidades y una sospecha de ataque MitM. Por lo que deciden no levantar el enlace y llamar a un especialista para que analice la red desde dentro y solucione las vulnerabilidades detectadas y cierre cualquier posible vía de ataque encontrada en el reporte, como puertos abiertos innecesarios, versiones antiguas, software inseguro, etc.



Vulnerabilidades Detectadas

Se han detectado 10 vulnerabilidades: 0 críticas, 2 altas, 0 medias, 8 bajas.

Vulnerabilidades Críticas y Altas:

CVE-1999-0013 - ssh OpenSSH 6.6.1p1 Ubuntu 2ubuntu2.13 (Ubuntu Linux; protocol 2.0) (Puerto 22) - ALTA
https://nvd.nist.gov/vuln/detail/CVE-1999-0013
Descripción: Stolen credentials from SSH clients via ssh-agent program, allowing other local users to access remote accounts belonging to the ssh-agent user.

Imagen 43: Vulnerabilidades detectadas en PDF.

4.4 Conclusiones

Los resultados obtenidos en este trabajo demuestran que es posible mejorar significativamente la seguridad y disponibilidad de las redes QKD mediante el uso combinado de dos soluciones complementarias: un sistema de enrutamiento dinámico y una herramienta de análisis automatizado de vulnerabilidades. Esta propuesta responde directamente a los objetivos planteados al inicio del proyecto, donde se buscaba, por un lado, investigar las amenazas más relevantes en redes QKD, y por otro, diseñar mecanismos para su detección y mitigación. La herramienta de pentesting ha cumplido con la función de identificar vulnerabilidades explotables en los nodos de confianza, mientras que el sistema de enrutamiento ha demostrado su eficacia a la hora de redirigir el tráfico de manera segura ante ataques de denegación de servicio o caídas en la red.

El entorno de simulación desarrollado ha sido clave para validar la solución de enrutamiento dinámico, permitiendo recrear escenarios realistas y estudiar su comportamiento sin depender de la disponibilidad de una red física. Esto representa una contribución adicional del trabajo, ya que este simulador puede

ser utilizado en futuras investigaciones, pruebas o incluso como herramienta didáctica en entornos académicos.

El impacto del proyecto se refleja en su capacidad para abordar tanto la prevención como la reacción ante ataques en infraestructuras cuánticas. A diferencia de otras propuestas centradas únicamente en la parte teórica o en aspectos aislados, esta solución integra protección física (a través del análisis de nodos) y lógica (a través de la gestión dinámica del tráfico), lo que la convierte en una propuesta integral y versátil. Además, su diseño modular permite escalarla, por ejemplo, añadiendo nuevos módulos de análisis de amenazas, más potencia o paralelismo a la hora de encontrar puertos, o adaptándola a redes más complejas.

A nivel personal, este proyecto ha supuesto una oportunidad para aplicar conocimientos de ciberseguridad clásica a un entorno emergente como la computación cuántica, permitiendo explorar la convergencia entre estas dos disciplinas. Más allá del resultado obtenido, el valor de este trabajo reside en haber demostrado que es posible aplicar soluciones prácticas a un ámbito aún en desarrollo, sentando así una base sobre la que seguir construyendo.

4.4.1 Trabajos Futuros

A partir de los resultados obtenidos, se pueden identificar múltiples líneas de trabajo que podrían continuar o extender la propuesta desarrollada. En primer lugar, una mejora directa sería desplegar y validar el sistema en una red QKD real como la red MadQCI, una vez esté operativa, para comprobar su funcionamiento frente a condiciones no simuladas y ajustar los parámetros del sistema (como los umbrales de QBER) a un entorno físico donde es crítico tener valores personalizados y muy concretos.

Por otro lado, el sistema de enrutamiento podría ampliarse con algoritmos más avanzados, como variantes de “k-shortest paths”, que permitirían mantener múltiples rutas alternativas en reserva y optimizar no solo la resiliencia sino también el rendimiento de la red.

En cuanto a la herramienta de pentesting, se podría integrar con sistemas de monitorización en tiempo real o incluir capacidades de respuesta automatizada (por ejemplo, aislamiento de nodos comprometidos). También sería posible ampliar el alcance del análisis incorporando escaneos sobre la capa de aplicación, y no solo sobre servicios de red.

Finalmente, el simulador desarrollado puede evolucionar hacia una plataforma más completa, con interfaz gráfica, inyección de eventos en tiempo real, y soporte para múltiples protocolos QKD, sirviendo como entorno de pruebas generalizable para desarrolladores e investigadores del área.

Todas estas direcciones abren la puerta a convertir la herramienta en una solución más completa, escalable y útil no solo en entornos académicos, sino también en contextos industriales donde la protección de redes cuánticas será cada vez más prioritaria.

Capítulo 5

Análisis de Impacto

En este capítulo se analiza en impacto práctico y estratégico de las dos soluciones propuestas. También se tendrá en cuenta el impacto que tienen en el Marco de los ODS. Especificando el ODS corresponda con una breve explicación.

5.1 Impacto del simulador y el enrutamiento dinámico

Experimentación segura y realista

El simulador permite recrear redes complejas QKD, emulando la distribución de claves y el comportamiento de la red ante caídas o ataques.

Resiliencia de la red

La función de enrutamiento dinámico gestionada por el SDN Controller garantiza que la red puede adaptarse a fallos o intrusiones, eligiendo rutas alternativas automáticamente y de esta forma ayudar a solucionar el gran problema de la denegación de servicio en las redes QKD.

Facilidad para la docencia e I+D

El simulador es una plataforma ideal para experimentar con arquitecturas y algoritmos sin riesgo sobre infraestructuras reales.

ODSs con los que contribuya

- **ODS 9: Industria, Innovación e Infraestructura**

  El simulador permite el desarrollo y prueba de nuevas tecnologías de comunicación cuántica, fomentando la innovación y la construcción de infraestructuras resilientes y sostenibles. [26]

- **ODS 4: Educación de Calidad**

  El simulador promueve la formación en tecnologías emergentes, contribuyendo a una educación inclusiva y de calidad. [27]

5.2 Impacto de la herramienta de análisis de red

Visibilidad y prevención

Automatiza la detección de vulnerabilidades en la infraestructura clásica, identificando servicios inseguros o desactualizados que podrían ser explotados para atacar la red QKD.

Reducción del error humano:

Permite que incluso usuarios no expertos obtengan diagnósticos claros y recomendaciones, minimizando los riesgos asociados a una mala configuración.

Ciclo de mejora continua

Facilita auditorías periódicas y la verificación de que los nodos y controladores mantienen un nivel de seguridad aceptable conforme evoluciona la red.

ODSs con los que contribuye

- **ODS 9: Industria, Innovación e Infraestructura**

  La herramienta contribuye a la construcción de infraestructuras resilientes al identificar y mitigar vulnerabilidades en redes híbridas, promoviendo una industrialización sostenible y fomentando la innovación en ciberseguridad. [26]

Capítulo 6

Bibliografía

[1] E. M. Ghourab, M. Azab y D. Gračanin, “A Quantum Key Distribution Routing Scheme for a Zero-Trust QKD Network System: A Moving Target Defense Approach,” Big Data Cogn. Comput., vol. 9, no. 4, p. 76, 2025 [En línea]. Disponible en: https://www.mdpi.com/2504-2289/9/4/76

[2] Cryptography Stack Exchange, “What are the current known weaknesses/attacks on quantum key distribution?,” Cryptography Stack Exchange, 2015 [En línea]. Disponible en: https://crypto.stackexchange.com/questions/27437/what-are-the-current-known-weaknesses-attacks-on-quantum-key-distribution

[3] Academia EITCA, “¿Cuáles son algunas de las contramedidas desarrolladas para combatir el ataque PNS y cómo mejoran la seguridad de los protocolos de distribución de claves cuánticas (QKD)?,” Academia EITCA, 2025 [En línea]. Disponible en: https://es.eitca.org/la-seguridad-cibern%C3%A9tica/eitc-es-qcf-fundamentos-de-criptograf%C3%ADa-cu%C3%A1ntica/distribuci%C3%B3n-pr%C3%A1ctica-de-claves-cu%C3%A1nticas/pirater%C3%ADa-cu%C3%A1ntica-parte-2/revisi%C3%B3n-del-examen-pirater%C3%ADa-cu%C3%A1ntica-parte-2/%C2%BFCu%C3%A1les-son-algunas-de-las-contramedidas-desarrolladas-para-combatir-el-ataque-pns-y-c%C3%B3mo-mejoran-la-seguridad-de-los-protocolos-qkd-de-distribuci%C3%B3n-de-claves-cu%C3%A1nticas%3F/

[4] Wikipedia, “Quantum key distribution,” Wikipedia, 2025 [En línea]. Disponible en: https://en.wikipedia.org/wiki/Quantum_key_distribution

[5] National Security Agency/Central Security Service, “Quantum Key Distribution (QKD) and Quantum Cryptography (QC),” NSA, 2025 [En línea]. Disponible en: https://www.nsa.gov/Cybersecurity/Quantum-Key-Distribution-QKD-and-Quantum-Cryptography-QC/

[6] M. Ivezic, “Quantum Hacking: Cybersecurity of Quantum Systems,” PostQuantum.com, Nov. 19, 2024 [En línea]. Disponible en: https://postquantum.com/post-quantum/quantum-hacking/

[7] S. Sun y A. Huang, “A Review of Security Evaluation of Practical Quantum Key Distribution System,” Entropy, vol. 24, no. 2, p. 260, Feb. 2022 [En línea]. Disponible en: https://pmc.ncbi.nlm.nih.gov/articles/PMC8870823/

[8] L. Lydersen, C. Wiechers, C. Wittmann, D. Elser, J. Skaar y V. Makarov, "Hacking commercial quantum cryptography systems by tailored bright illumination," Nature Photonics, vol. 4, no. 10, pp. 686–689, Oct. 2010 [En línea]. Disponible en: https://qudev.phys.ethz.ch/static/content/courses/QSIT12/pdfs/Lydersen2010.pdf

[9] L. Lydersen, C. Wiechers, C. Wittmann, D. Elser, J. Skaar y V. Makarov, "Thermal blinding of gated detectors in quantum cryptography," Opt. Express, vol. 18, no. 26, pp. 27938–27954, 2010 [En línea]. Disponible en: https://opg.optica.org/oe/fulltext.cfm?uri=oe-18-26-27938&id=209022

[10] The Nmap Project, "Nmap: the Network Mapper – Free Security Scanner," Nmap.org, 2025 [En línea]. Disponible en: https://nmap.org/

[11] R. D. Graham, "masscan," GitHub, 2025 [En línea]. Disponible en: https://github.com/robertdavidgraham/masscan

[12] Greenbone, "OpenVAS – The Open Vulnerability Assessment System," OpenVAS.org, 2025 [En línea]. Disponible en: https://www.openvas.org/

[13] Tenable Inc., "Nessus Essentials," community.tenable.com, 2025 [En línea]. Disponible en: https://community.tenable.com/s/article/Nessus-Essentials?language=en_US

[14] Chris Sullo, "Nikto web server scanner," GitHub, 2025 [En línea]. Disponible en: https://github.com/sullo/nikto

[15] Philippe Biondi et al., "Scapy: interactive packet manipulation library," Scapy.net, 2025 [En línea]. Disponible en: https://scapy.net/

[16] GeeksforGeeks, "Distance Vector Routing (DVR) Protocol," GeeksforGeeks, 2024 [En línea]. Disponible en: https://www.geeksforgeeks.org/distance-vector-routing-dvr-protocol/

[17] GeeksforGeeks, "Floyd Warshall Algorithm," GeeksforGeeks, 2025 [En línea]. Disponible en: https://www.geeksforgeeks.org/floyd-warshall-algorithm-dp-16/

[18] Wikipedia, "A* search algorithm," Wikipedia, 2025 [En línea]. Disponible en: https://en.wikipedia.org/wiki/A%2A_search_algorithm

[19] Wikipedia, "Yen's algorithm," Wikipedia, 2025 [En línea]. Disponible en: https://en.wikipedia.org/wiki/Yen%27s_algorithm

[20] Wikipedia, “Suurballe’s algorithm,” Wikipedia, 2025 [En línea]. Disponible en: https://en.wikipedia.org/wiki/Suurballe%27s_algorithm

[21] GeeksforGeeks, “Dijkstra’s Shortest Path Algorithm using priority_queue of STL,” GeeksforGeeks, 2024 [En línea]. Disponible en: https://www.geeksforgeeks.org/dijkstras-shortest-path-algorithm-using-priority_queue-stl/

[22] takeUforward, “Dijkstra’s Algorithm – Using Priority Queue : G-32,” takeUforward.org, 2025 [En línea]. Disponible en: https://takeuforward.org/data-structure/dijkstras-algorithm-using-priority-queue-g-32/

[23] Stack Overflow, “Why is the time complexity of Dijkstra O((V+E) logV)?,” StackOverflow.com, 2025 [En línea]. Disponible en: https://stackoverflow.com/questions/61890100/why-is-the-time-complexity-of-dijkstra-ov-e-logv

[24] Mathematics Stack Exchange, “Time complexity of Dijkstra’s algorithm,” Mathematics Stack Exchange, 2025 [En línea]. Disponible en: https://math.stackexchange.com/questions/3683910/time-complexity-of-dijkstras-algorithm

[25] European Telecommunications Standards Institute, “Quantum Key Distribution (QKD); Application Interface,” ETSI GS QKD 015 V2.1.1, 2022 [En línea]. Disponible en: https://www.etsi.org/deliver/etsi_gs/QKD/001_099/015/02.01.01_60/gs_QKD015v020101p.pdf

[26] Wikipedia, “Objetivo de Desarrollo Sostenible 9,” Wikipedia, 2024 [En línea]. Disponible en: https://es.wikipedia.org/wiki/Objetivo_de_Desarrollo_Sostenible_9

[27] Wikipedia, “Fundación Cibervoluntarios,” Wikipedia, 2025 [En línea]. Disponible en: https://es.wikipedia.org/wiki/Fundaci%C3%B3n_Cibervoluntarios

[28] A. Poppe, M. Peev, O. Maurhart, "Outline of the SECOQC Quantum-Key-Distribution Network in Vienna," arXiv:0804.0122, 2008. [En línea]. Disponible en: https://arxiv.org/abs/0804.0122

[29] M. Sasaki et al., "Field test of quantum key distribution in the Tokyo QKD Network," Optics Express, vol. 19, no. 11, pp. 10387-10409, 2011. [En línea]. Disponible en: https://opg.optica.org/oe/abstract.cfm?uri=oe-19-11-10387

[30] ID Quantique, "Quantum Key Distribution | QKD," [En línea]. Disponible en: https://www.idquantique.com/quantum-safe-security/quantum-key-distribution/

[31] Comisión Europea, "The European Quantum Communication Infrastructure (EuroQCI)," [En línea]. Disponible en: https://digital-strategy.ec.europa.eu/en/policies/european-quantum-communication-infrastructure-euroqci

[32] A. J. Sebastián Lombraña, "MadQCI: a metropolitan QKD network in Madrid," Jornadas Técnicas de RedIRIS 2023, [En línea]. Disponible en: https://tv.rediris.es/video/6489d61118d1ca0360740e33

[33] C. Orzel, How to Teach Quantum Physics to Your Dog, New York: Scribner, 2010.

[34] E. Diamanti, H.-K. Lo, B. Qi y Z. Yuan, “Practical challenges in quantum key distribution,” npj Quantum Information, vol. 2, no. 1, p. 16025, 2016. [En línea]. Disponible en: https://www.nature.com/articles/npjqi201625

Capítulo 7

Anexos

7.1 Anexo A: Reporte Generado por la herramienta de análisis de vulnerabilidades

A continuación se muestran las distintas imágenes correspondientes a los distintos apartados del reporte de seguridad generado por la herramienta de análisis de vulnerabilidades.

Reporte de Seguridad de Red QKD

Fecha: 26/05/2025 12:37

Objetivo: scanme.nmap.org

Resumen Ejecutivo

Este escaneo ha identificado 2 vulnerabilidades significativas:

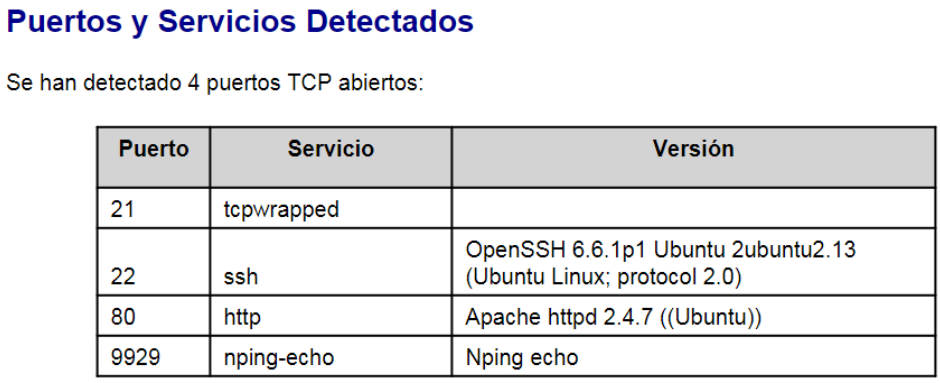
• 0 críticas, 2 altas, 0 medias

• 2 posibles vectores de ataque Man-in-the-Middle

El nivel de riesgo general es: ALTO

Se recomienda planificar la remediación de las vulnerabilidades altas en el corto plazo.

Imagen 36: Sección del Resumen ejecutivo en PDF.

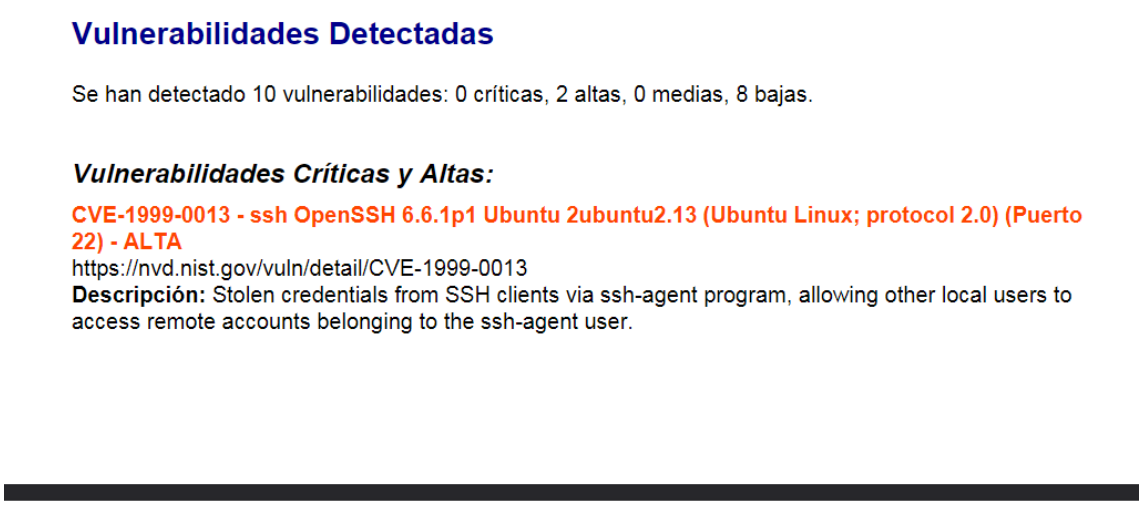


Puertos y Servicios Detectados

Se han detectado 4 puertos TCP abiertos:

Puerto	Servicio	Versión
21	tcpwrapped	
22	ssh	OpenSSH 6.6.1p1 Ubuntu 2ubuntu2.13 (Ubuntu Linux; protocol 2.0)
80	http	Apache httpd 2.4.7 ((Ubuntu))
9929	nping-echo	Nping echo

Imagen 37: Sección de Puertos y Servicios en PDF.



Vulnerabilidades Detectadas

Se han detectado 10 vulnerabilidades: 0 críticas, 2 altas, 0 medias, 8 bajas.

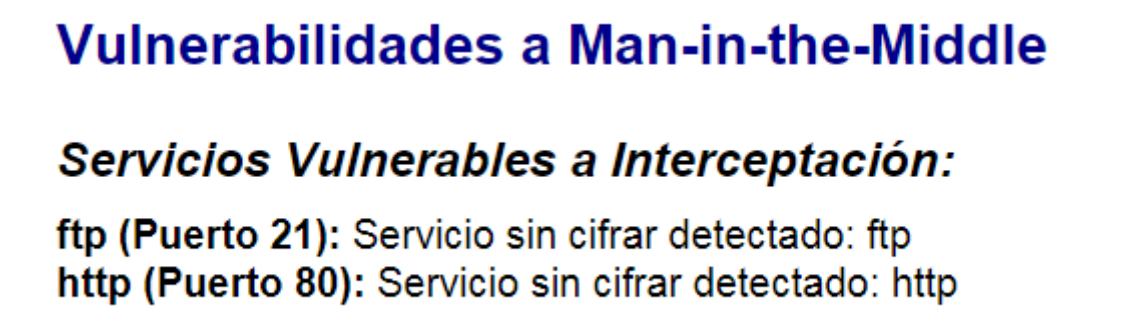
Vulnerabilidades Críticas y Altas:

CVE-1999-0013 - ssh OpenSSH 6.6.1p1 Ubuntu 2ubuntu2.13 (Ubuntu Linux; protocol 2.0) (Puerto 22) - ALTA
https://nvd.nist.gov/vuln/detail/CVE-1999-0013
Descripción: Stolen credentials from SSH clients via ssh-agent program, allowing other local users to access remote accounts belonging to the ssh-agent user.

Remediación: Actualizar a la última versión disponible.

CVE-1999-0236 - http Apache httpd 2.4.7 ((Ubuntu)) (Puerto 80) - ALTA
https://nvd.nist.gov/vuln/detail/CVE-1999-0236
Descripción: ScriptAlias directory in NCSA and Apache httpd allowed attackers to read CGI programs.
Remediación: Actualizar a la última versión disponible.

Imagen 38: Sección de Vulnerabilidades Detectadas en PDF.

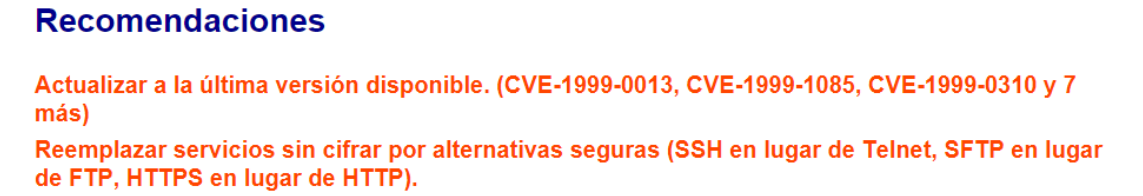


Vulnerabilidades a Man-in-the-Middle

Servicios Vulnerables a Interceptación:

ftp (Puerto 21): Servicio sin cifrar detectado: ftp
http (Puerto 80): Servicio sin cifrar detectado: http

Imagen 39: Sección de Vulnerabilidades de Man-in-the-Middle en PDF.



Recomendaciones

Actualizar a la última versión disponible. (CVE-1999-0013, CVE-1999-1085, CVE-1999-0310 y 7 más)

Reemplazar servicios sin cifrar por alternativas seguras (SSH en lugar de Telnet, SFTP en lugar de FTP, HTTPS en lugar de HTTP).

Imagen 40: Sección de recomendaciones en PDF.

Este documento esta firmado por

	**Firmante**	CN=tfgm.fi.upm.es, OU=CCFI, O=ETS Ingenieros Informaticos - UPM, C=ES
	Fecha/Hora	Tue Jun 03 23:35:05 CEST 2025
	Emisor del Certificado	EMAILADDRESS=camanager@etsiinf.upm.es, CN=CA ETS Ingenieros Informaticos, O=ETS Ingenieros Informaticos - UPM, C=ES
	Numero de Serie	561
	Metodo	urn:adobe.com:Adobe.PPKLite:adbe.pkcs7.sha1 (Adobe Signature)