



**MEJORA DEL NETWORKING, LA SEGURIDAD Y LA  
PRIVACIDAD DE LA INFORMACIÓN PARA  
[COMPAÑÍA DE VUELOS COMERCIALES]**

**Fecha de realización:** 24/04/2024

**Consultoría realizada por:**

Juan Luis Ruano Muriedas

José Joaquín Rojas Romero

Antonio José Suárez García

# Índice

|                                      |          |
|--------------------------------------|----------|
| <b>Índice.....</b>                   | <b>2</b> |
| <b>Resumen ejecutivo.....</b>        | <b>3</b> |
| <b>Metodología y resultados.....</b> | <b>4</b> |
| Consulta 1.....                      | 4        |
| Consulta 2.....                      | 8        |
| Consulta 3.....                      | 22       |

## Resumen ejecutivo

En la *consulta 1* se ha debatido sobre las opciones comerciales de **SD-WAN** y servicios **SASE** para poder cumplir los objetivos que propone. Se ha decantado con la opción de **Palo Alto Networks** como la mejor opción a elegir de las recomendaciones del cuadrante mágico de *Gartner*.

En la *consulta 2* se ha resuelto la privacidad de los datos de los clientes en nubes públicas mediante el uso de **criptografía homomórfica**, en concreto mediante el uso del algoritmo de **Pailliet** obteniendo pruebas positivas. En cuanto a la privacidad de los datos en colaboración con las autoridades gubernamentales contra la delincuencia, se ha desarrollado un **algoritmo de cifrado personalizado** que compara los hashes generados de los pasajeros con una base de datos de criminales compartida por la policía. Para resolver la parte final de esta consulta y recuperar los datos empresariales de sus diferentes centros de datos hemos aportado un algoritmo personalizado de tipo **Prueba de Conocimiento Cero (Zero Knowledge Proofs (ZKPs))** utilizando criptografía asimétrica **RSA**.

En la *consulta 3*, para la primera parte se ha implementado, ejecutado y probado un script basado en el **algoritmo de Grover**, empleando las librerías **Quikit** del portal **IBM** a través de una instancia de un ordenador cuántico con **127 qubits** en **10000 shots**. Los resultados se muestran en una gráfica de **distribución de probabilidad** y se ha aportado una valoración técnica sobre el estado de madurez de estas tecnologías.

Para la segunda parte se enseñan ejemplos de scripts en python usando el Quikit del IBM sobre las utilidades de usar la computación cuántica en ciberseguridad, en concreto se detallan aspectos de **generación de números aleatorios cuánticos (QRNG)**, **distribución de claves cuánticas (QKD)** y **corrección de errores cuánticos (QEC)**.

En la tercera parte se han expuesto las características del algoritmo **CRYSTALS-KYBER**, algoritmo finalista del concurso del NIST que permite abordar los **problemas de cifrado y firmado** sobre los que nos consultaba. En resumidas cuentas, la **complejidad computacional** propia de los algoritmos quantum-resistant es **mucho mayor** que la de algoritmos clásicos como RSA y AES, aunque **el tamaño de clave no sea mayor**.

# Metodología y resultados

## Consulta 1

### 1. Propuesta tecnológica de *SD-WAN*

En esta consulta hemos decidido elegir los servicios de la empresa *Palo Alto Networks* para la licencia del servicio *SD-WAN* que necesita su compañía de vuelos comerciales. El motivo de esta elección se basa en que **cumple con todas las condiciones requeridas** de path selection, QoS, seguridad y reporting que vienen descritas en la documentación que podrá ver en el siguiente enlace: <https://docs.paloaltonetworks.com/sd-wan>.

Figure 1: Magic Quadrant for SD-WAN



**Figura 1.** Cuadrante mágico de *SD-WANs* por Gartner. Otra razón para elegir *Palo Alto Networks*. Si necesita alguna ayuda con cualquier servicio de *SD-WAN*, tenemos varios técnicos que estarán encantados de ayudarle. Este *SD-WAN* con la configuración adecuada también

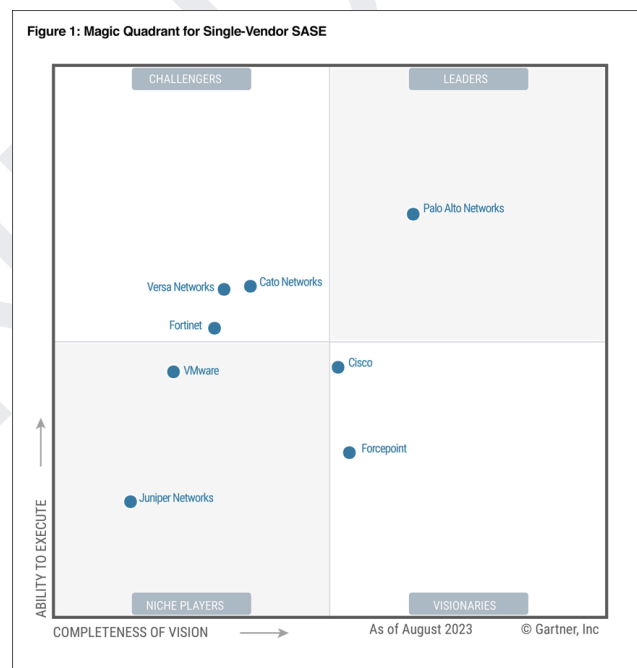
será capaz de **mejorar el rendimiento y disponibilidad** de las aplicaciones y servicios críticos tales como ERP, CRM, Bases de Datos, Office365, Salesforce y AWS.

El ancho de banda de WAN se podrá adaptar al de su empresa según la configuración y podrá llegar a los 450 Mbps sin problema. Un motivo para usar un *SD-WAN* es **ahorrar** en costes de hardware e infraestructura ya que permiten una comunicación más **ágil** usando el software especializado para definir políticas de enrutamiento, optimización de tráfico y seguridad en la red.

Y para poder incrementar la escalabilidad y flexibilidad del networking permitiendo la adición o eliminación de sitios o aplicaciones hacia *SD-WAN* o desde la misma, Palo Alto Networks tiene la posibilidad de **crear un servicio centralizado** de tal manera que pueda organizar todas las ramas sucursales de su empresa de forma centralizada.

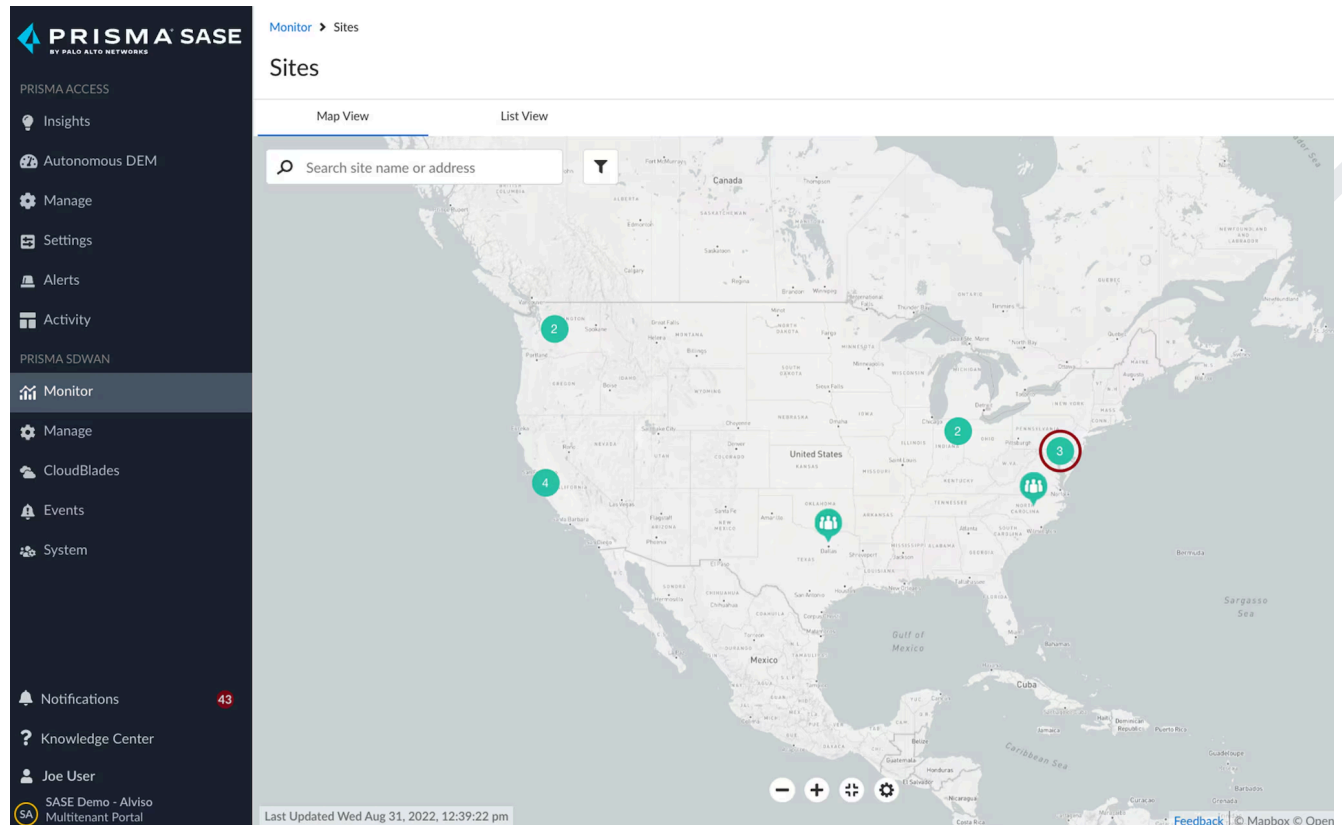
## 2. Servicios SASE

Aquí es donde encontramos la mayor ventaja de elegir **Palo Alto Networks** como proveedor *SASE*, es uno de los **líderes del mercado** según la consultora Gartner y tener el servicio conjunto será de gran ayuda.



**Figura 2.** Cuadrante mágico de *SASE* por Gartner. Palo Alto Networks se encuentra como líder del mercado.

La opción *SASE* de *Palo Alto Networks* se trata de ***SASE Prisma*** y cuenta con un servicio en la nube que **complementa** el sistema *SD-WAN* implementado anteriormente.



**Figura 3.** Interfaz Gráfica de *SASE Prisma*, *Palo Alto Networks*.

En cuanto a sus puntos de accesos, cuentan con PoPs por todo el globo y podrá verlos en su documentación a través del siguiente enlace:

<https://docs.paloaltonetworks.com/prisma/prisma-access/3-1/prisma-access-panorama-admin/prepare-the-prisma-access-infrastructure/list-of-prisma-access-locations>

Cuentan con un servicio eficaz y difundido que le será muy útil en el servicio *SASE*.

### 3. Costes del *SASE* y el *SD-WAN*

*Palo Alto Networks* posee una **calculadora** para dichos casos la cual vamos a utilizar para ver los costes de dicha implementación:

<https://tools.totaleconomicimpact.com/go/paloalto/sase/index.html>

Select the solution(s) you'd like to invest in:

Prisma SASE (Security and SD-WAN) ☒

Prisma Access (Security only) ☐

Prisma SD-WAN (SD-WAN only) ☐

Enter the total number of employees and contractors in your organization ⓘ

1,000 250,000 25,000

What percentage of employees work remotely?

0% 100% 100%

How many branches/sites does your organization have? ⓘ

25 2,500 138

**\$13,653,552**

Your estimated three-year net present value (NPV)

**279%**

Your estimated three-year return on investment (ROI)

Download **your custom business case (PDF)** to see all calculations, tables, charts, and ROI!

**COOKIE ACCEPTANCE IS REQUIRED TO REGISTER**

Accept cookies

**Figura 4.** Aproximación con la calculadora, incluyendo ambos servicios *SASE* y *SD-WAN*.

También le ofrece la opción de entregar los resultados del análisis tras rellenar un formulario si desea más información. La aproximación está hecha con **138 sucursales** como indicaron y hemos aproximado 25 mil empleados entre todas ellas y todos trabajando en remoto. De este modo obtenemos que el coste de contratar los servicios de *Palo Alto Networks* sería de **13.653.552 dólares cada tres años**, es decir, unos **4.551.184 al año**.

## Consulta 2

### 1. Privacidad del procesamiento de datos personales de los clientes de la compañía de vuelos comerciales en nubes públicas.

Para implementar un algoritmo básico de suma de gastos de pasajeros en la nube con técnicas de privacidad, se ha considerado el uso de **criptografía homomórfica**. La criptografía homomórfica permite realizar operaciones matemáticas directamente sobre datos cifrados sin necesidad de descifrarlos previamente. En concreto, para este caso, se ha utilizado el algoritmo de **Paillier**, que es un esquema de criptografía homomórfica aditiva. A continuación, se detallan los pasos para implementar un sistema de suma de gastos utilizando el algoritmo de Paillier.

```
1  from Crypto.Util.number import getPrime
2  from Crypto.Util.number import getRandomInteger
3  from gmpy2 import mpz
4  import math
5  def generate_keypair(bits):
6      p = getPrime(bits)
7      q = getPrime(bits)
8      n = p * q
9      g = n + 1
10     lambda_ = (p - 1) * (q - 1)
11     mu = pow(lambda_, -1, n)
12     return n, g, lambda_, mu
13 def encrypt(n, g, m):
14     r = getRandomInteger(n)
15     c = pow(g, m, n**2) * pow(r, n, n**2) % (n**2)
16     return c
17 def decrypt(n, lambda_, mu, c):
18     return ((pow(c, lambda_, n**2) - 1) * pow(n, -1, n**2) * mu) % n
19 def sum_gastos(n, g, c1, c2):
20     return (c1 * c2) % (n**2)
21 def main():
22     n, g, lambda_, mu = generate_keypair(1024)
23     # Pasajero 1
24     gasto1 = 1250
25     c1 = encrypt(n, g, gasto1)
26     # Pasajero 2
27     gasto2 = 1500
28     c2 = encrypt(n, g, gasto2)
29     # Suma homomórfica
30     c_sum = sum_gastos(n, g, c1, c2)
31     # Obtener el resultado
32     gasto_total = decrypt(n, lambda_, mu, c_sum)
33     print("Gasto total: ", gasto_total)
34 if __name__ == "__main__":
35     main()
```



**Figura 5.** Implementación del algoritmo de Paillier.

El tamaño de la clave recomendado para el algoritmo de Paillier es de **1024 bits o superior** para garantizar un nivel de seguridad adecuado. Con este sistema, los datos de los gastos de cada pasajero se mantienen privados en todo momento, ya que solo se trabaja con datos cifrados en la nube. Junto con el envío de esta consultoría se ha adjuntado el correspondiente sistema desarrollado para cubrir este aspecto.

Las pruebas realizadas tras ejecutar el *script* muestran resultados íntegros a pesar del cifrado. El sistema es eficiente, pues no aparecen incrementos de tiempos de procesamiento con datos cifrados frente al procesamiento con datos no cifrado.

Tras las pruebas, hemos redactado el siguiente párrafo sobre la Política de Privacidad, contemplando cómo se tratarán los datos relacionados con los gastos de los pasajeros de su compañía:

- Recopilación y uso de datos: Nuestra empresa de vuelo recopila información relacionada con los gastos realizados por los pasajeros durante sus viajes, con el fin de gestionar transacciones financieras y proporcionar servicios relacionados con la reserva y el vuelo.
- Uso lícito de datos: Los datos de gastos de los pasajeros se utilizan exclusivamente para procesar transacciones financieras y facilitar la prestación de servicios asociados con el viaje, como la emisión de facturas, reembolsos o gestión de reclamaciones.
- Envío a terceros: En algunos casos, los datos de gastos de los pasajeros pueden ser enviados a terceros proveedores de servicios financieros, como entidades bancarias, procesadores de pagos o plataformas de gestión de gastos, para facilitar el procesamiento de transacciones y cumplir con las obligaciones financieras.
- Nubes públicas: Para el almacenamiento de datos, nuestra empresa puede utilizar servicios de nubes públicas proporcionados por proveedores confiables y que cumplan con los estándares de seguridad y privacidad establecidos por las regulaciones aplicables.
- Seguridad de datos: Implementamos medidas de seguridad técnicas y organizativas para proteger los datos de los clientes, incluidos los datos de gastos, contra el acceso no autorizado, la divulgación o la alteración.

- Consentimiento del cliente: Al proporcionar sus datos de gastos, los pasajeros consienten expresamente el procesamiento de los mismos de acuerdo con esta política de privacidad.
- Retención de datos: Mantenemos los datos de gastos de los pasajeros solo durante el tiempo necesario para los fines para los que fueron recopilados y de acuerdo con los requisitos legales y regulatorios aplicables.
- Derechos de los pasajeros: Los pasajeros tienen derecho a acceder, corregir o eliminar sus datos de gastos, así como a oponerse al procesamiento de los mismos, de conformidad con las leyes de protección de datos aplicables.
- Cambios en la política de privacidad: Nos reservamos el derecho de modificar esta política de privacidad en cualquier momento. Cualquier cambio significativo será comunicado a los pasajeros de manera adecuada.

## 2. Privacidad de datos empresariales en procesos de colaboración contra la delincuencia y el terrorismo con autoridades gubernamentales.

Para completar esta segunda parte de la consultoría, se han establecido las siguientes pautas previas a la implementación del sistema del sistema:

- La policía y la empresa de vuelo establecerán, previo acuerdo, un sistema de hashing seguro y eficiente. Dicho sistema está comprendido por el uso de **SHA256** junto con **salt** y una **clave compartida** entre el jefe de departamento y el personal autorizado de la compañía de vuelo, la cual será renovada en cada vuelo.
- La **policía** contará con una **base de datos** actualizada donde se almacenan los **nombres** de los criminales junto con el **hash** del sistema de hashing comentado aplicado a dicho nombre, a modo de tuplas.
- La **compañía de vuelo** contará con una **base de datos**, generada en cada vuelo, donde se encontrará **únicamente el hashing** de los nombres de los viajeros, para preservar su confidencialidad. Los detalles del uso de esta base de datos podrán contemplarse en el apartado correspondiente de la Política de Privacidad.
- Antes de que el vuelo comience, la compañía de vuelo ejecutará un **programa o script**, adjuntado junto con la entrega de esta consultoría, el cual **comparará los**

**hashes** de los viajeros con la **base de datos policial** de los criminales, y en caso de coincidencia, se mostrará el nombre y hash de la coincidencia para intervenir con la persona correspondiente, detectando así posibles criminales en el vuelo sin violar la confidencialidad del resto de pasajeros.

El sistema implementado, empleando el lenguaje de programación *Python*, cuenta con dos partes importantes: el programa o *script* empleado por la policía para generar las bases de datos de criminales y el programa o *script* que ejecutará la compañía de vuelo antes de cada vuelo. A continuación se mostrarán los detalles de la implementación del sistema.

### **Script de la policía:**

```
1  import hashlib
2  import os
3  import getpass
4  # Limpiamos terminal para mostrar el script correctamente
5  os.system('clear')
6  # Mostramos el banner del script policial
7  with open("banner_police.txt", "r") as f:
8      banner = f.read()
9      print(banner)
10 # Nombre + salt en el cifrado con Sha256
11 clave = getpass.getpass('Introduce la contraseña: ').encode('utf-8')
12 salt = str(9843670983456938453).encode('utf-8')
13 # Lista vacía para guardar las diferentes entradas generadas
14 hashed_values = []
```

**Figura 6.** Primera parte del script policial.

Importamos la librería **hashlib** para realizar *hashing* seguro. También importamos la librería **os** para limpiar la terminal de anteriores tareas, y finalmente importamos la librería **getpass** para evitar ver en claro la contraseña que se solicita por terminal para ingresar al programa y generar la base de datos. Esta primera parte se encarga de inicializar el programa con la clave, *salt* y lista vacía para introducir los nombres de criminales hasheados.

En la segunda parte del *script* policial (ver **Figura 7**) se realiza el algoritmo de cifrado, donde se realiza un hashing del criminal (nombre completo) + salt + clave en las líneas de código de la 22 a la 25. El resto del *script* se encarga de almacenar en la base de datos *hashed\_criminals* el nombre en texto claro y hash del nombre de cada criminal.

```
15 # Acceso al sistema mediante la clave "clave1"
16 if clave != "" and clave == "clave1".encode('utf-8'):
17     # Nombres de criminales en el archivo criminales.txt
18     with open("criminales.txt", "r") as c:
19         # Realizamos lectura del archivo y generamos hashes
20         for line in c:
21             crim = line.strip()
22             nombre = str(crim).encode('utf-8')
23             cadena = nombre + salt + clave
24             hash_obj = hashlib.sha256(cadena)
25             hash_value = hash_obj.hexdigest()
26             # Tupla con el nombre del criminal, salt y hashing
27             hashed_store = (nombre.decode('utf-8'), hash_value)
28             # Mostramos las entradas a registrar en pantalla
29             print(hashed_store)
30             # Guardamos las entradas en la lista vacía inicial
31             hashed_values.append(hashed_store)
32             # Abrimos la base de datos de criminales y hashes
33             with open("hashed_criminals.txt", "a") as h:
34                 # Escribimos en la base de datos las nuevas entradas
35                 h.write(f"{hashed_store[0]},{hashed_store[1]}\n")
36 else:
37     # En caso de fallar la contraseña mostramos el mensaje
38     print('Contraseña inválida...')
```

**Figura 7.** Segunda parte del script policial.

### Script de la compañía de vuelo:

```
4 # Limpiamos terminal de anteriores tareas
5 os.system('clear')
6 # Mostramos banner de inicialización del script
7 with open("banner_fly.txt", "r") as f:
8     banner = f.read()
9     print(banner)
10 # Nombre + salt en el cifrado con Sha256
11 clave = getpass.getpass('Introduce la contraseña: ').encode('utf-8')
12 salt = str(9843670983456938453).encode('utf-8')
```

**Figura 8.** Primera parte del script de la compañía de vuelo.

El *script* de la compañía de vuelo tiene emplea las mismas librerías y comienza idénticamente al *script* policial.

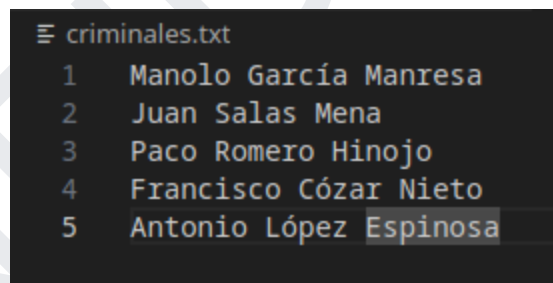
```
13 # Algoritmo de comprobación de coincidencias viajero/criminal
14 if clave != "" and clave == "clave1".encode('utf-8'):
15     # Pasajeros del actual viaje
16     with open("viajeros.txt", "r") as c:
17         for line in c:
18             l = line.strip()
19             nombre = str(l).encode('utf-8')
20             cadena = nombre + salt + clave
21             hash_obj = hashlib.sha256(cadena)
22             hash_value = hash_obj.hexdigest()
23             # Guardamos el hashing del pasajero dado
24             hashed_store = hash_value
25             # Mostramos por pantalla los hashes generados
26             print(hashed_store)
27             # Abrimos lista de criminales compartida por la policía
28             with open("hashed_criminals.txt", "r") as h:
29                 for line in h:
30                     # Separamos en variables el nombre y el hash
31                     c_hash = line.strip().split(",")[1]
32                     c_name = line.strip().split(",")[0]
33                     # si coinciden el hash del pasajero y el del criminal
34                     if c_hash == hashed_store:
35                         print("\n")
36                         print("¡POSIBLE CRIMINAL!\n")
37                         print(f"Nombre: {c_name}\n")
38                         print(f"HASH: {c_hash}\n")
39
40 else:
41     # Mensaje de fallo de autenticación por contraseña incorrecta
42     print('Contraseña inválida...')
```

**Figura 9.** Segunda parte del script de la compañía de vuelo.

La segunda parte del *script* se encarga de tomar los nombres de los pasajeros del vuelo actual y realizarle un hashing (líneas de la 19 a la 24 de la **Figura 9**). Dicho hash es comparado con la base de datos de criminales compartida por la policía (línea 34 de la **Figura 9**) y en caso de coincidencia alerta sobre un posible problema con el pasajero en cuestión, del cual se rescata el nombre que estaba guardado en la base de datos policial y se muestra un aviso (líneas de la 36 a la 38 de la **Figura 9**).

Para aclarar cualquier posible duda, queremos remarcar que **en ningún momento se almacenan los nombres de los pasajeros**. Tan solo se establece una lista temporal de los pasajeros del vuelo actual, y cada entrada es hasheada para comparar dichas entradas con la base de datos de criminales. Cada vez que se produce un vuelo distinto las listas temporales son **eliminadas** por Política de Privacidad y garantizamos que los hashes que se generan con los nombres de los pasajeros **no son almacenados** en ningún momento, pues son comparados *on the fly*.

Se han realizado pruebas del sistema con un conjunto de datos de ejemplo para garantizar al cliente el correcto funcionamiento del mismo. Para ello se ha creado la base de datos de criminales que muestra la Figura 10.



```
criminales.txt
1  Manolo García Manresa
2  Juan Salas Mena
3  Paco Romero Hinojo
4  Francisco Cózar Nieto
5  Antonio López Espinosa
```

**Figura 10.** Base de datos con los criminales de los archivos policiales

Desde el lado de la policía se ha ejecutado el *script* ***police.py*** sobre la correspondiente base de datos. Dicha ejecución requiere de una clave la cual es ***clave1*** en este ejemplo.

```
CAI5 : python -
* * * * *
*
*      POLICE CRIMINAL DATABASE SCRIPT      *
*
* * * * *
Introduce la contraseña: █
```

**Figura 11.** Mensaje inicial que nos pide la contraseña, en este caso *clave1*, para acceder.

Tras ingresar la contraseña requerida, se ejecuta el script que realiza el correspondiente hashing de los criminales de la base de datos policial, generando por un lado un log con los hashes obtenidos (Figura 12), y por otro lado otra base de datos (Figura 13) con el nombre del criminal y su hashing, la cual será la base de datos compartida con la compañía de vuelo.

```
CAI5 : bash — Konsole
* * * * *
*
*      POLICE CRIMINAL DATABASE SCRIPT      *
*
* * * * *
Introduce la contraseña:
('Manolo García Manresa', 'bc60b5010e16f48831ef3a7850c9a44923ee3bd0100f349225caf6179c53953e')
('Juan Salas Mena', '7f5baeee8e858266f8ad2c6c95dffcf7365fa904cbd91eb41266cf9775ff1ead2')
('Paco Romero Hinojo', '4e42e321afb7be13245e8c27a688fb7b4b97ce4add4f8b42aa5b968a53d0c600')
('Francisco Cózar Nieto', 'ad11cb6fe7062cf77fc5908abdf91addc77a47ee5946a0ed527df9b81a479ece')
('Antonio López Espinosa', '212e7b915ee02f8552d1cc4f06ae9a856548b61e3a9620ab273ec559e19fc5ec')
[tric0@arch CAI5]$ █
```

**Figura 12.** Log generado tras la ejecución del *script* policial sobre la BD de criminales.

```
hashes_criminals.txt
1  Manolo García Manresa,bc60b5010e16f48831ef3a7850c9a44923ee3bd0100f349225caf6179c53953e
2  Juan Salas Mena,7f5baeee8e858266f8ad2c6c95dffcf7365fa904cbd91eb41266cf9775ff1ead2
3  Paco Romero Hinojo,4e42e321afb7be13245e8c27a688fb7b4b97ce4add4f8b42aa5b968a53d0c600
4  Francisco Cózar Nieto,ad11cb6fe7062cf77fc5908abdf91addc77a47ee5946a0ed527df9b81a479ece
5  Antonio López Espinosa,212e7b915ee02f8552d1cc4f06ae9a856548b61e3a9620ab273ec559e19fc5ec█
```

**Figura 13.** Base de datos completada con el nombre del criminal y su hash.

Para continuar con la prueba, se ha ejecutado el *script* de la compañía de vuelo con la misma clave compartida (Figura 15) sobre la BD de pasajeros (Figura 14).

```
≡ viajeros.txt
1  Genoveva Toledo Rojas
2  Luis Díaz Gómez
3  Jose Alberto Martinez
4  Cesar Alcalá del Valle
5  Juan Salas Mena
```

**Figura 14.** Base de datos de pasajeros temporal.

```
> ✎ CAIS :
* * * * *
*
*   FLY COMPANY DATABASE SCRIPT   *
*
* * * * *
Introduce la contraseña: █
```

**Figura 15.** Ejecución del *script* de la compañía de vuelo ingresando la contraseña *clave1*.

Y se ha obtenido un resultado positivo al existir un pasajero cuyo hash coincide con uno de la base de datos policial, como se muestra en la Figura 16.



```

*****
*
*      FLY COMPANY DATABASE SCRIPT      *
*
*
*****
Introduce la contraseña:
6c68993c6d275fe0fe646c0acff210d3ed52c0f7f6911314480aca67e810edb2
1e94da049e2fa9cf112b803500aed0f89350488f260e63697d708d23a519e192
1e13dcc7259665885ea7701576712705ac5cc3e0e68461d77ece4f68e89336ae
9fec8eb051136813f253862db1992553b3500172e4dd81f835192f8277226863
7f5baeee8e858266f8ad2c6c95dfffc7365fa904cbd91eb41266cf9775ff1ead2

¡POSSIBLE CRIMINAL!

Nombre: Juan Salas Mena

HASH: 7f5baeee8e858266f8ad2c6c95dfffc7365fa904cbd91eb41266cf9775ff1ead2

[tric0@arch CAI5]$ █

```

**Figura 16.** Ejecución del *script* de la compañía de vuelo con resultado positivo.

Se ha realizado otra prueba para comprobar el tiempo que pueda tardar el sistema en comprobar una lista de 350 pasajeros (media de pasajeros en vuelos comerciales) y detectar, en el peor de los casos, un criminal en la última entrada, en una lista de criminales de 300 entradas, obteniendo un tiempo de respuesta máximo de 0.11 segundos (Figura 17), lo cual es una marca que prueba la eficacia del algoritmo diseñado.

```

6c68993c6d275fe0fe646c0acff210d3ed52c0f7f6911314480aca67e810edb2
1e94da049e2fa9cf112b803500aed0f89350488f260e63697d708d23a519e192
1e13dcc7259665885ea7701576712705ac5cc3e0e68461d77ece4f68e89336ae
9fec8eb051136813f253862db1992553b3500172e4dd81f835192f8277226863
6c68993c6d275fe0fe646c0acff210d3ed52c0f7f6911314480aca67e810edb2
1e94da049e2fa9cf112b803500aed0f89350488f260e63697d708d23a519e192
1e13dcc7259665885ea7701576712705ac5cc3e0e68461d77ece4f68e89336ae
9fec8eb051136813f253862db1992553b3500172e4dd81f835192f8277226863
6c68993c6d275fe0fe646c0acff210d3ed52c0f7f6911314480aca67e810edb2
7f5baeee8e858266f8ad2c6c95dfffc7365fa904cbd91eb41266cf9775ff1ead2

¡POSSIBLE CRIMINAL!

Nombre: Juan Salas Mena

HASH: 7f5baeee8e858266f8ad2c6c95dfffc7365fa904cbd91eb41266cf9775ff1ead2

Tiempo de ejecución: 0.11084413528442383 segundos
[tric0@arch CAI5]$ █

```

**Figura 17.** Tiempo de ejecución del *script* de la compañía de vuelo con resultado positivo.

Después de las pruebas realizadas hemos determinado que el sistema proporcionado cumple con los requisitos mínimos que garantizan la integridad, seguridad y confidencialidad de los datos de los pasajeros que pedía usted, por lo que podemos concluir afirmando que el sistema es eficaz con una buena eficiencia debido al algoritmo empleado (*SHA256*).

Después del resultado de esta consulta, se ha decidido añadir el párrafo siguiente a la Política de Privacidad de la web de la compañía sobre el uso de los datos requeridos de los pasajeros:

- **Recopilación de datos:** La empresa de vuelos recopila información personal, incluidos nombres completos y otros datos pertinentes, con el propósito de garantizar la seguridad de nuestros servicios y cumplir con los requisitos legales y regulatorios.
- **Uso de la información:** Los datos recopilados se utilizan exclusivamente para verificar la identidad de los pasajeros y garantizar que no representen una amenaza para la seguridad del vuelo. Esta verificación puede incluir la comparación de nombres con bases de datos de seguridad y antecedentes criminales.
- **Seguridad de datos:** Implementamos medidas de seguridad técnicas y organizativas para proteger los datos personales de nuestros clientes contra el acceso no autorizado, la divulgación, la alteración o la destrucción accidental o ilícita.
- **Acceso a terceros:** En ciertos casos, podemos compartir información con autoridades gubernamentales o agencias de seguridad para cumplir con las regulaciones de aviación y garantizar la seguridad de nuestros vuelos.
- **Retención de datos:** Mantenemos los datos personales solo durante el tiempo necesario para los fines para los que se recopilaron y de acuerdo con los requisitos legales aplicables.
- **Derechos de los usuarios:** Los usuarios tienen derecho a acceder, corregir o eliminar sus datos personales, así como a oponerse al procesamiento de los mismos, de acuerdo con las leyes de protección de datos aplicables.
- **Consentimiento del usuario:** Al proporcionar sus datos personales, los usuarios consienten expresamente el procesamiento de los mismos para los fines descritos en esta política de privacidad.

- Cambios en la política de privacidad: Nos reservamos el derecho de modificar esta política de privacidad en cualquier momento. Cualquier cambio significativo será notificado a los usuarios de manera adecuada.

### 3. Privacidad en la recuperación de datos empresariales de sus diferentes centros de datos.

Le proporcionamos la implementación de un protocolo de Recuperación de Información Privada (PIR) basado en un esquema de Prueba de Conocimiento Cero (*Zero Knowledge Proofs* (ZKPs)) utilizando criptografía asimétrica RSA, empleando el lenguaje de programación Python. La siguiente Figura 18 muestra una captura del código implementado en el sistema.

```
1 from Crypto.Util.number import getPrime
2 from Crypto.Random import get_random_bytes
3 # En este ejemplo, se genera un esquema de Prueba de Conocimiento Cero utilizando criptografía asimétrica
4 # RSA. Se eligen números primos grandes, se calculan claves pública y privada, se genera un mensaje aleatorio,
5 # se cifra con la clave pública y se verifica con la clave privada para demostrar el conocimiento del mensaje
6 # sin revelar su contenido. Este es un ejemplo básico para ilustrar cómo se puede implementar un esquema de
7 # Prueba de Conocimiento Cero en Python. Para aplicaciones reales, se recomienda un análisis más detallado y
8 # la adaptación a los requisitos específicos del sistema de precios de vuelos de la compañía aérea.
9
10 # Generar números primos grandes
11 p = getPrime(128)
12 q = getPrime(128)
13 # Calcular n y phi
14 n = p * q
15 phi = (p - 1) * (q - 1)
16 # Elegir un número aleatorio coprimo con phi
17 e = 65537
18 # Calcular la clave privada
19 d = pow(e, -1, phi)
20 # Función para generar una prueba de conocimiento cero
21 def zero_knowledge_proof():
22     # Generar un mensaje aleatorio
23     m = get_random_bytes(16)
24     # Calcular el mensaje cifrado
25     c = pow(int.from_bytes(m, 'big'), e, n)
26     # Verificar la prueba de conocimiento cero
27     decrypted_m = pow(c, d, n)
28     if int.from_bytes(m, 'big') == decrypted_m:
29         return True
30     else:
31         return False
32 # Ejecutar la prueba de conocimiento cero
33 result = zero_knowledge_proof()
34 print("¿La prueba de conocimiento cero es válida?", result)
```

Figura 18. Script que emplea la librería *Crypto* de Python para una prueba ZKPs.

Se ha realizado una prueba con un mensaje aleatorio para demostrar la eficacia y eficiencia del sistema proporcionado. Dicha prueba muestra la salida por pantalla una vez ejecutada que muestra la Figura 19.

```
(entorno_zkps) [tric0@arch part 3]$ python ZKPs.py
¿La prueba de conocimiento cero es válida? True
(entorno_zkps) [tric0@arch part 3]$
```

**Figura 19.** Resultado positivo de la prueba ZKPs con un mensaje aleatorio.

Como podemos comprobar, el sistema funciona de una manera eficaz y eficiente, pues no requiere de un gran tiempo de ejecución y funciona con cualquier tipo de dato requerido, esperando haber satisfecho las necesidades que usted requiere.

Se ha redactado el siguiente párrafo que debería ser incluido en el apartado correspondiente de la Política de Privacidad de usted:

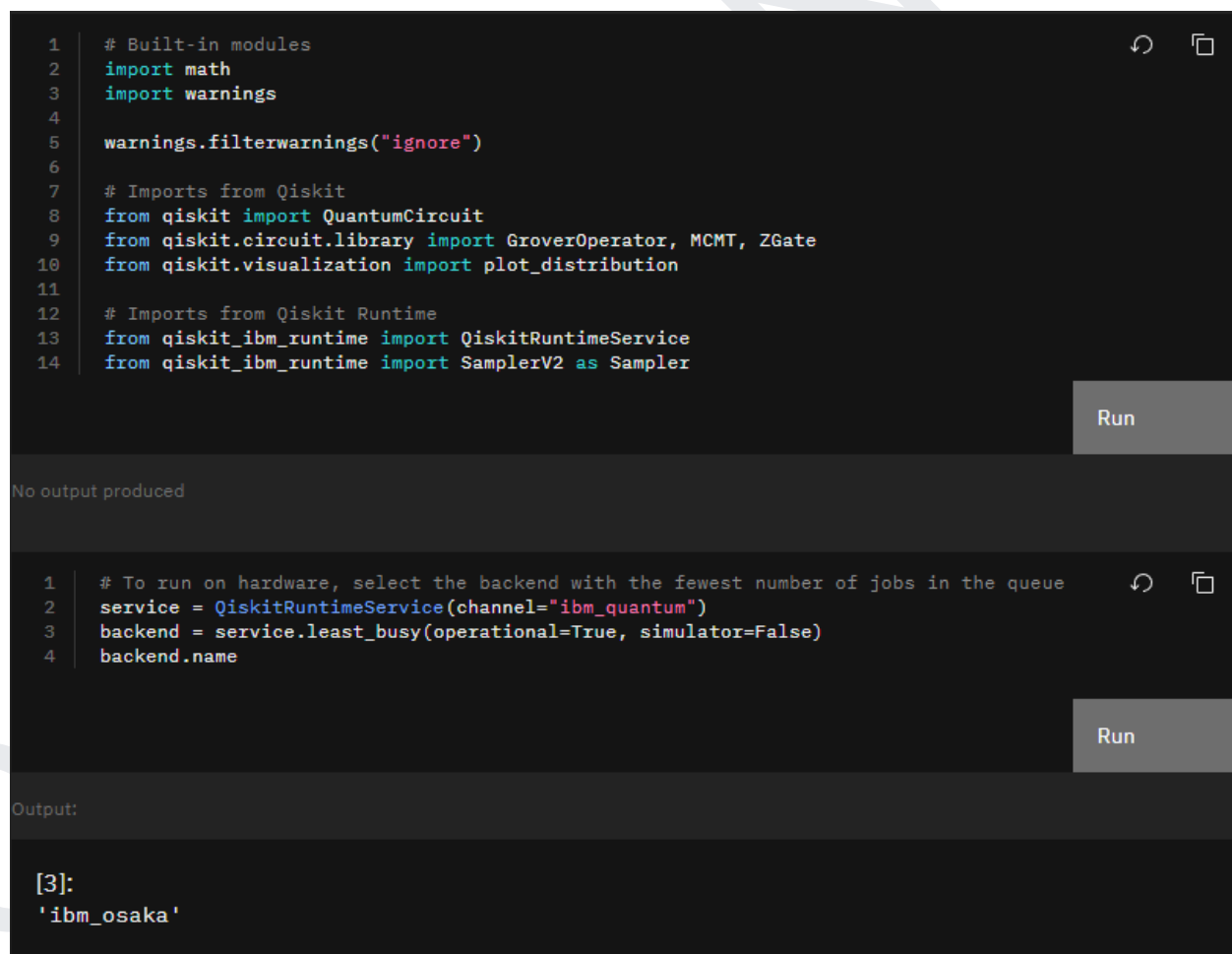
- **Recopilación de datos:** La empresa de vuelos recopila información personal de los clientes, como nombres y detalles de contacto, exclusivamente con el propósito de facilitar la reserva y compra de vuelos.
- **Uso de la información:** Los datos proporcionados por los clientes se utilizan únicamente para procesar sus solicitudes de precios de vuelos y para fines relacionados con la reserva y compra de billetes aéreos. No se realiza ningún seguimiento de las solicitudes individuales de los clientes, y los servidores no almacenan información sobre vuelos específicos solicitados por los clientes.
- **Minimización de la información:** Nos comprometemos a presentar a los clientes únicamente la información necesaria para llevar a cabo sus reservas de vuelo de manera eficiente y segura. No proporcionamos información adicional más allá de lo necesario para el proceso de reserva, a menos que sea requerida por las regulaciones de aviación o la legislación aplicable.
- **Seguridad de datos:** Implementamos medidas de seguridad robustas para proteger los datos personales de los clientes contra el acceso no autorizado, la divulgación o la alteración.
- **Acceso a terceros:** No compartimos los datos personales de los clientes con terceros, excepto en los casos en que sea necesario para cumplir con las regulaciones de aviación, las leyes aplicables o a solicitud de autoridades gubernamentales.

- Retención de datos: Mantenemos los datos personales de los clientes solo durante el tiempo necesario para los fines para los que fueron recopilados, a menos que se requiera un período de retención más largo por razones legales o regulatorias.
- Consentimiento del cliente: Al proporcionar sus datos personales, los clientes consienten el procesamiento de los mismos de acuerdo con esta política de privacidad.
- Cambios en la política de privacidad: Nos reservamos el derecho de modificar esta política de privacidad en cualquier momento. Cualquier cambio significativo será comunicado a los clientes de manera oportuna.

### Consulta 3

1. Implementación, ejecución y pruebas de dos desarrollos de software cuántico que representan amenazas a la seguridad de la información empresarial través de la computación cuántica

Para dar solución a esta consulta, se ha usado como recurso la plataforma de **IBM Quantum** (<https://learning.quantum.ibm.com/tutorial/grovers-algorithm>) donde existe un ejemplo del algoritmo de **Grover**, el cual será tomado como ejemplo base para el desarrollo de un script básico con el que demostrar cómo puede representar una amenaza a la seguridad de la información el uso de estas técnicas a través de la computación cuántica.



```
1 # Built-in modules
2 import math
3 import warnings
4
5 warnings.filterwarnings("ignore")
6
7 # Imports from Qiskit
8 from qiskit import QuantumCircuit
9 from qiskit.circuit.library import GroverOperator, MCMT, ZGate
10 from qiskit.visualization import plot_distribution
11
12 # Imports from Qiskit Runtime
13 from qiskit_ibm_runtime import QiskitRuntimeService
14 from qiskit_ibm_runtime import SamplerV2 as Sampler
```

No output produced

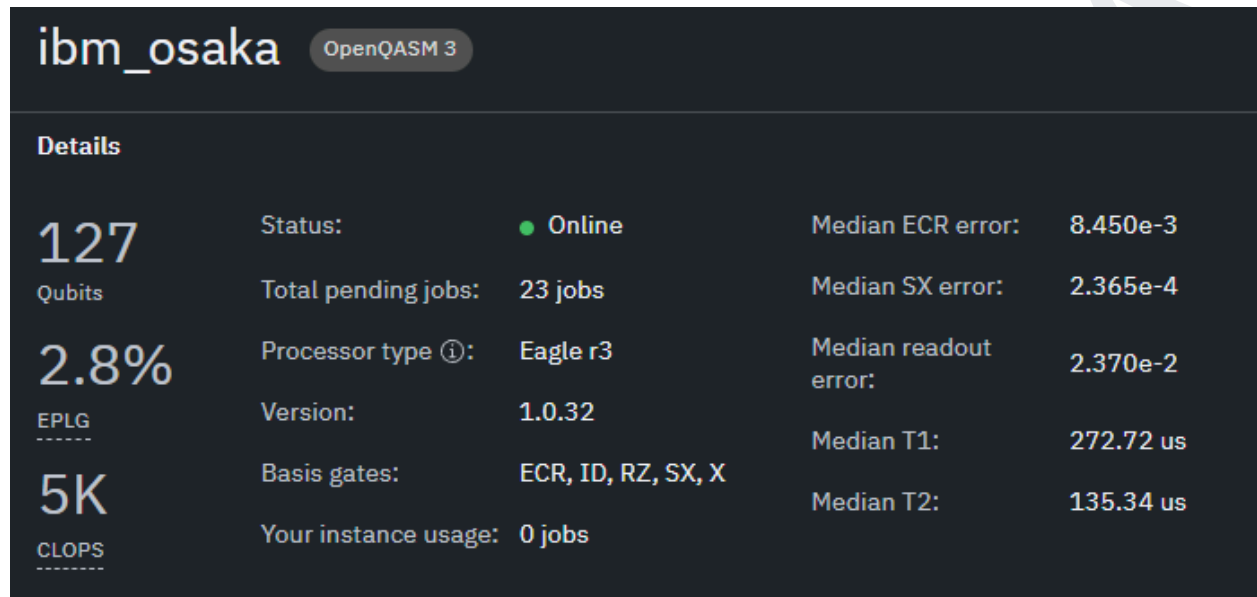
```
1 # To run on hardware, select the backend with the fewest number of jobs in the queue
2 service = QiskitRuntimeService(channel="ibm_quantum")
3 backend = service.least_busy(operational=True, simulator=False)
4 backend.name
```

Output:

```
[3]:
'ibm_osaka'
```

Figura 20. Librerías y módulos importados para el algoritmo de **Grover**.

En la **Figura 19** vemos como resultado de la ejecución del código el nombre de la máquina del backend que se encargará de ejecutar nuestro código, la máquina **ibm\_osaka**. Dicha máquina cuenta con **127 Qubits**, que es la cantidad máxima permitida para planes gratuitos dentro de la plataforma de **IBM**. Puede ver más detalles en la **Figura 20**.



**Figura 21.** Detalles sobre la máquina IBM donde se ejecutará nuestro código y pruebas.

Seguidamente se mostrará la definición de la función de **Grover**. El algoritmo de Grover requiere un oráculo que especifique uno o más estados de base computacional marcados, donde "marcado" significa un estado con una fase de  $-1$ . Una puerta  $Z$  controlada, o su generalización multicontrolada sobre  $N$  qubits, marca el estado  $2^N - 1$  ('1' \*  $N$  cadena de bits). Marcar los estados base con uno o más '0' en la representación binaria requiere la aplicación de puertas  $X$  en los qubits correspondientes antes y después de la puerta  $Z$  controlada; equivalente a tener un control abierto en ese cúbit. En el siguiente código, definimos un oráculo que hace precisamente eso, marcando uno o más estados básicos de entrada definidos a través de su representación de cadena de bits. La puerta MCMT se utiliza para implementar la puerta  $Z$  multicontrolada. Ver **Figura 21**.

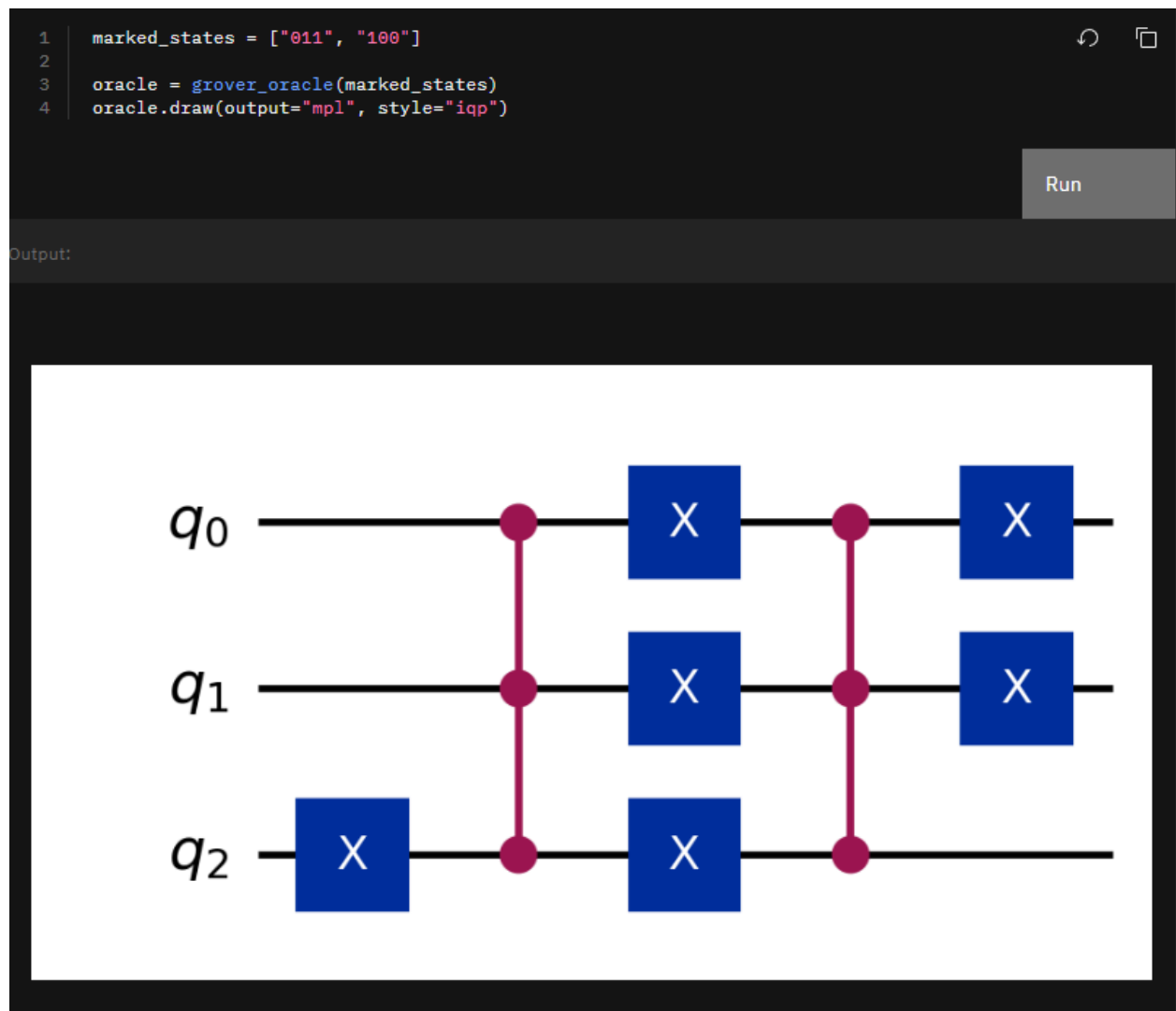
```
1 def grover_oracle(marked_states):
2     """Build a Grover oracle for multiple marked states
3
4     Here we assume all input marked states have the same number of bits
5
6     Parameters:
7         marked_states (str or list): Marked states of oracle
8
9     Returns:
10         QuantumCircuit: Quantum circuit representing Grover oracle
11     """
12     if not isinstance(marked_states, list):
13         marked_states = [marked_states]
14     # Compute the number of qubits in circuit
15     num_qubits = len(marked_states[0])
16
17     qc = QuantumCircuit(num_qubits)
18     # Mark each target state in the input list
19     for target in marked_states:
20         # Flip target bit-string to match Qiskit bit-ordering
21         rev_target = target[::-1]
22         # Find the indices of all the '0' elements in bit-string
23         zero_inds = [ind for ind in range(num_qubits) if rev_target.startswith("0", ind)]
24         # Add a multi-controlled Z-gate with pre- and post-applied X-gates (open-controls)
25         # where the target bit-string has a '0' entry
26         qc.x(zero_inds)
27         qc.compose(MCMT(ZGate(), num_qubits - 1, 1), inplace=True)
28         qc.x(zero_inds)
29     return qc
```

Run

Figura 22. Detalles de la función de *Grover*.

Ahora que tenemos la función oracle, podemos definir una instancia específica de la búsqueda de Grover. En este ejemplo, marcaremos dos estados computacionales de los ocho disponibles en un espacio computacional de tres qubits. Ver **Figura 23**.





**Figura 23.** Dos estados computacionales para tres qubits.

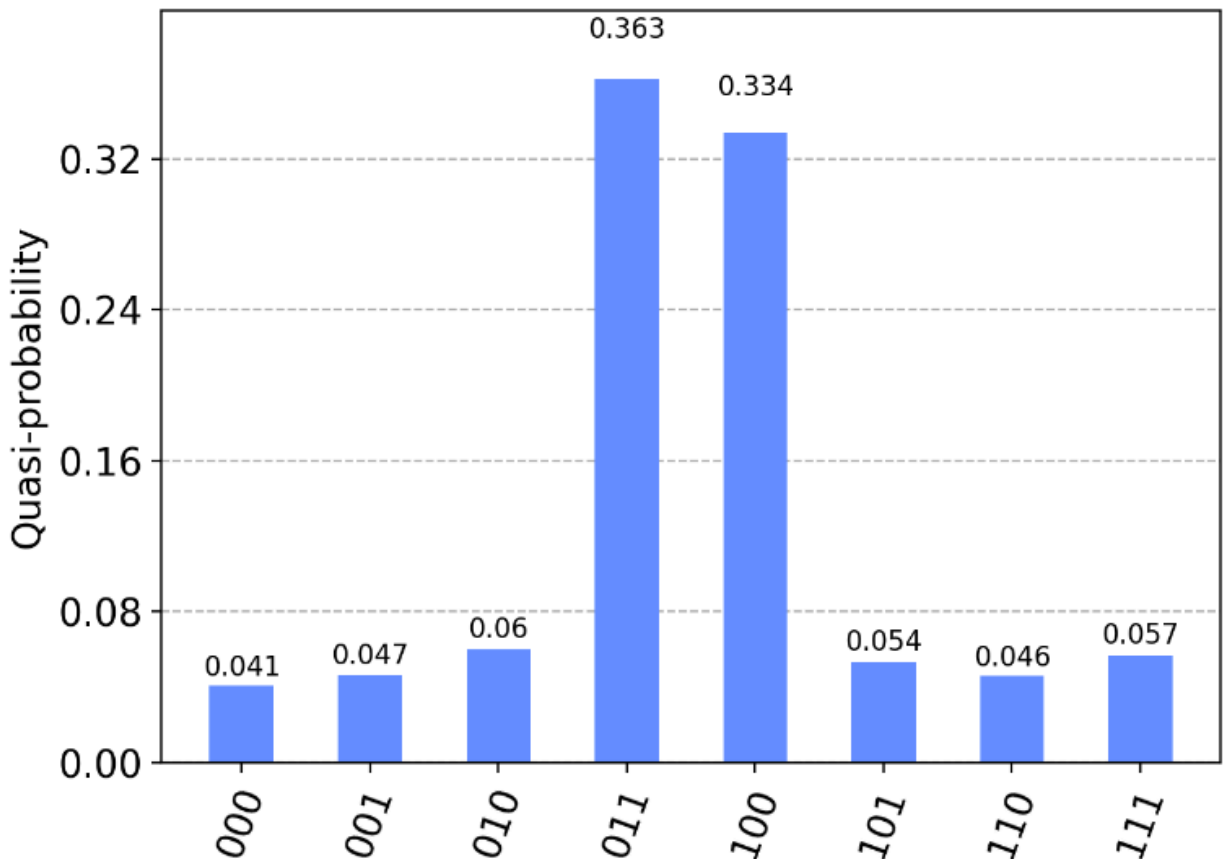
Para terminar de entender el funcionamiento de este algoritmo, se ha realizado una prueba mediante las primitivas de Qiskit, con un número de **10000 shots** (*mediciones que se realizarán en el algoritmo de Grover. El número de shots determina la precisión del resultado. Un mayor número de shots puede proporcionar resultados más precisos, pero también aumenta el tiempo de ejecución*). Ver **Figura 24**.

```
1 # To run on local simulator:
2 # 1. Use the Sampler from qiskit.primitives instead
3 sampler = Sampler(backend=backend)
4 sampler.options.default_shots = 10_000
5 result = sampler.run([circuit_isa]).result()
6 dist = result[0].data.meas.get_counts()
```

Run

**Figura 24.** Ejemplo de uso del algoritmo anterior con 10000 shots.

Este código extrae la distribución de probabilidad de los resultados de la ejecución del algoritmo de Grover. La distribución de probabilidad se obtiene a partir del resultado de la ejecución, que se almacena en la variable **result**. El método **get\_counts** devuelve un diccionario que contiene la distribución de probabilidad de los resultados, donde las claves son los resultados posibles y los valores son la frecuencia de cada resultado. Mediante el comando **plot\_distribution(dist)** podemos ver los resultados en la siguiente gráfica sobre la distribución de probabilidad, en la siguiente **Figura 25**.



**Figura 25.** Gráfico de la distribución de probabilidad del algoritmo de Grover con 10000 shots.

En este caso, el gráfico mostraría la distribución de los resultados de la búsqueda cuántica, que es la salida del algoritmo de *Grover*. El gráfico muestra la frecuencia de cada resultado posible, lo que permitiría analizar y comprender mejor la efectividad de la búsqueda cuántica. Por ejemplo, si el algoritmo de Grover está buscando un elemento en una base de datos cuántica, el gráfico podría mostrar la frecuencia de cada elemento en la base de datos, lo que permitiría identificar los elementos más comunes o los que están más cerca del objetivo de búsqueda.

Para demostrar la teoría del algoritmo de Grover aplicada a un **algoritmo que amenace la seguridad de la información empresarial**, se ha diseñado el siguiente script de prueba:

```

script_1.py > ...
1  from qiskit_ibm_runtime import QiskitRuntimeService
2  from qiskit import QuantumCircuit, execute, Aer
3  from qiskit.visualization import plot_histogram
4  import numpy as np
5
6  QiskitRuntimeService(channel="ibm_quantum", token="<fa09bd7859c643ac0c84aec2f4d8123d
7  service = QiskitRuntimeService()
8  backend = service.backend(name="ibm_brisbane")
9
10 # Definir el número de qubits
11 n = 3
12 # Crear un circuito cuántico con n qubits
13 qc = QuantumCircuit(n)
14 # Aplicar la transformada de Hadamard a todos los qubits
15 qc.h(range(n))
16 # Aplicar la función de oráculo (en este caso, buscar el estado '101')
17 qc.z(2)
18 qc.cx(0, 1)
19 qc.cx(1, 2)
20 # Aplicar la transformada de Grover
21 qc.h(range(n))
22 qc.z(range(n))
23 qc.h(range(n))
24 # Medir los qubits
25 qc.measure_all()
26 # Ejecutar el circuito en un simulador de IBM
27 backend = Aer.get_backend('qasm_simulator')
28 job = execute(qc, backend, shots=1024)
29 result = job.result()
30 # Obtener la distribución de probabilidad de los resultados
31 counts = result.get_counts(qc)
32 # Mostrar la distribución de probabilidad
33 plot_histogram(counts)

```

Figura 26. Script aportado que hace uso del algoritmo de Grover en un simulador IBM.

Al ejecutar este script, el gráfico resultante mostrará una distribución de probabilidad sesgada hacia el estado '101', lo que indica que el algoritmo de Grover ha encontrado el estado objetivo de manera más eficiente que una búsqueda clásica.

Este ejemplo representa una **amenaza real** para la seguridad de la información, ya que el algoritmo de Grover puede ser utilizado para romper ciertos algoritmos criptográficos de manera más eficiente que los métodos clásicos. Por lo tanto, es importante tener en cuenta los avances en la computación cuántica y desarrollar nuevos métodos de cifrado que puedan resistir estos ataques, pues las pruebas se hicieron en un ordenador cuántico de **127 qubits** y en la actualidad existen máquinas de hasta **2000 qubits**.

## 2. Implementación, ejecución y pruebas de dos desarrollos de software cuántico que muestren las posibles oportunidades de la computación cuántica para mejorar la seguridad de la información de la compañía.

En este apartado se procederá a usar el *computador cuántico del IBM* (<https://quantum.ibm.com/>) para probar la seguridad en aspectos de **generación de números aleatorios cuánticos (QRNG)**, **distribución de claves cuánticas (QKD)** y **corrección de errores cuánticos (QEC)** con especial énfasis en los dos primeros pues es donde se basan los scripts que probaremos.

En el ámbito computacional es notable la **carencia del software de crear números aleatorios**, por ello es necesario crear scripts que generan números aleatorios, sin embargo, nunca podrán ser 100% aleatorios por las limitaciones de la máquina y por tanto tienen **pueden llegar a ser predecibles**, lo que pone en riesgo los sistemas de seguridad que se basen en estos. Sin embargo, la física ofrece una **solución alternativa en la computación cuántica**. En un ordenador cuántico, un ejemplo común de un dispositivo **QRNG** es un generador basado en la polarización de fotones individuales. En este enfoque, los fotones individuales se envían a través de un divisor de haz que los separa en dos caminos diferentes. Luego, se aplican filtros polarizadores que pueden estar en diferentes orientaciones. La polarización cuántica de cada fotón se mide, y esta información se utiliza para generar bits aleatorios, es decir, **utilizan fotones y las reglas naturales de la física para generar bits aleatorios**.

Un ejemplo de script de **QRNG** en python usando **qiskit** del IBM es el siguiente:

```

from qiskit import QuantumRegister, ClassicalRegister, QuantumCircuit

service = QiskitRuntimeService(channel="ibm_quantum", token="token_ibm")
backend = service.backend("ibm_brisbane")

q = QuantumRegister(16, 'q')
c = ClassicalRegister(16, 'c')
circuit = QuantumCircuit(q, c)
circuit.h(q) # Applies hadamard gate to all qubits
circuit.measure(q, c) # Measures all qubits

job = backend.run(circuit, shots = 5)

print('Ejecutando...\n')
result = job.result()
counts = result.get_counts(circuit)

for i in counts.keys():
    counts[i] = int(i, 2)

print('Resultados: ', counts, '\n')

```

Figura 27. Ejemplo de script de generador de números aleatorios.

En el siguiente ejemplo estamos usando el ordenador cuántico situado en Brisbane, RU, para generar **5 números aleatorios de 16 qubits**, equivalente a 16 bits. Los resultados después de esperar en cola, fueron los siguientes:

Ejecutando...

Resultados: {'10110111111001001': 47049, '0100010100100100': 17700, '0010111010110110': 11958, '11110110111110000': 63216, '01011101111101110': 24046}

Figura 28. Ejemplo de script de generador de números aleatorios.

En cuanto a **QKD**, la base de la transmisión fiable es saber si el **mensaje ha sido modificado**, sin embargo, es siempre posible en computación editar mensajes sin dejar rastro de ello. Por ello, en computación cuántica una **QKD** utiliza propiedades cuánticas, como la superposición y el entrelazamiento, para **garantizar la seguridad de la clave** compartida. Un protocolo QKD típico implica la transmisión de fotones individuales que representan bits de la clave a través de un canal de comunicación. Debido a las leyes de la física cuántica, cualquier intento de un tercero de interceptar o medir estos fotones sin ser

detectada **perturbará el estado cuántico de los fotones**, lo que los usuarios de los extremos podrían detectar durante el proceso de distribución de la clave.

Se ha diseñado el siguiente ejemplo de QKD en Qiskit usando el protocolo BB84:

```
from qiskit import QuantumCircuit
from qiskit.visualization import plot_histogram, plot_bloch_multivector
from numpy.random import randint
import numpy as np
```

```
num_qubits = 32
```

```
alice_basis = np.random.randint(2, size=num_qubits)
alice_state = np.random.randint(2, size=num_qubits)
bob_basis = np.random.randint(2, size=num_qubits)
```

```
print(f"Alice's State:\t {np.array2string(alice_state)}")
print(f"Alice's Bases:\t {np.array2string(alice_basis)}")
print(f"Bob's Bases:\t {np.array2string(bob_basis)}")
```

```
def bb84_circuit(state, basis, measurement_basis):
```

```
    num_qubits = len(state)
```

```
    circuit = QuantumCircuit(num_qubits)
```

```
    # Sender prepares qubits
```

```
    for i in range(len(basis)):
```

```
        if state[i] == 1:
```

```
            bb84_circuit.x(i)
```

```
        if basis[i] == 1:
```

```
            bb84_circuit.h(i)
```

```
    # Measuring action performed by Bob
```

```
    for i in range(len(measurement_basis)):
```

```
        if measurement_basis[i] == 1:
```

```
            bb84_circuit.h(i)
```

```
    bb84_circuit.measure_all()
```

```
    return bb84_circuit
```

```
service = QiskitRuntimeService(channel="ibm_quantum", token="token")
```

```
backend = service.backend("ibm_brisbane")
```

```
circuit = bb84_circuit(alice_state, alice_basis, bob_basis)
```

```
key = backend.run(circuit.reverse_bits(), backend, shots=1).result().get_counts().most_frequent()
```

```
for i in range(num_qubits):
```

```
    if alice_basis[i] == bob_basis[i]:
```

```
        encryption_key += str(key[i])
```

```
print(f"Key: {encryption_key}")
```

**Figura 29.** Ejemplo de script de distribución de claves cuánticas.

Por último está el tema de la **fragilidad de los qubits frente a los errores**. Los qubits, que son los análogos cuánticos de los bits clásicos, son extremadamente sensibles a las perturbaciones ambientales y a los errores inherentes a los dispositivos cuánticos, lo que puede **comprometer la precisión de los cálculos cuánticos**.

El objetivo de la **corrección de errores cuánticos (QEC)** es desarrollar técnicas y algoritmos que puedan detectar y corregir errores en los estados cuánticos sin destruir la información cuántica. A diferencia de la corrección de errores en sistemas clásicos, donde se pueden duplicar bits o utilizar códigos de redundancia para detectar y corregir errores, la **corrección de errores cuánticos es mucho más desafiante** debido a la fragilidad de la superposición y el entrelazamiento cuánticos.

Si desea saber más sobre medidas de seguridad basadas en la **computación cuántica** tiene a su disposición nuestro grupo de técnicos de Insegus.

### **3. Eficacia y eficiencia de un algoritmo quantum-resistant para el cifrado o firmado de la información empresarial frente a los algoritmos clásicos.**

Empezando por el firmado, el algoritmo más comúnmente usado es **RSA**, el cual necesita de un tamaño de clave de al menos **2048 bits** para considerarse criptográficamente seguro. Entre los algoritmos quantum-resistant ganadores del concurso del NIST, para la firma digital destaca **CRYSTALS-KYBER**, un algoritmo de cifrado de propósito general que abarca dos primitivas criptográficas: **Kyber** y **Dilithium**. Ambos mecanismos también fueron finalistas del concurso, siendo el primero un mecanismo para **encapsulación de claves** y el segundo para la **firma digital**. En cuanto al tamaño de clave, este algoritmo destaca por tener una clave bastante pequeña, ya que la seguridad no la impone la clave, sino la difícil resolución de los **problemas de celosías** en los que se basa.



|   |                 |            | NIST | los           | clave   | clave   |                                   |
|---|-----------------|------------|------|---------------|---------|---------|-----------------------------------|
|   |                 |            |      | algoritmos    | pública | privada |                                   |
|   |                 |            |      |               | (bytes) | (bytes) |                                   |
| 1 | CRISTALES-KYBER | Finalistas | 1    | Kyber512      | 800     | 1632    | Mecanismos clave de encapsulación |
|   |                 |            |      | Kyber512-90s  | 800     | 1632    |                                   |
|   |                 |            | 3    | Kyber768      | 1184    | 2400    |                                   |
|   |                 |            |      | Kyber768-90s  | 1184    | 2400    |                                   |
|   |                 |            | 5    | Kyber1024     | 1568    | 3168    |                                   |
|   |                 |            |      | Kyber1024-90s | 1568    | 3168    |                                   |

**Figura 30.** CRYSTALS-KYBER en distintos niveles de seguridad del NIST.

Este algoritmo además permite cierta **configuración de sus parámetros**, permitiendo así elegir entre una versión más robusta pero pesada o una más ligera pero menos segura. Se definen 3 sets de parámetros: Kyber (el recomendado), Light (algo más inseguro) y Paranoid (más seguro), los cuáles varían la dimensión de la celosía.

| Nivel de seguridad | $k$ | $\eta$ | $(d_u, d_v, d_t)$ | $\delta$   | $sk$ (bytes) | $pk$ (bytes) |
|--------------------|-----|--------|-------------------|------------|--------------|--------------|
| Kyber              | 3   | 4      | (11, 3, 11)       | $2^{-142}$ | 2400         | 1088         |
| Light              | 2   | 5      | (11, 3, 11)       | $2^{-145}$ | 2048         | 736          |
| Paranoid           | 4   | 3      | (11, 3, 11)       | $2^{-169}$ | 2752         | 1440         |

**Figura 31.** Parámetros de Kyber

Para la configuración recomendada, **CRYSTALS-KYBER no tiene un tamaño de clave muy superior a RSA**, aunque, como todos los algoritmos quantum-resistant, requiere de **muchos ciclos de CPU**, lo cual aumenta la complejidad computacional, haciendo que estos algoritmos sean un **gran desafío** para dispositivos móviles y IoT.

En cuanto al cifrado, el algoritmo más comúnmente usado es **AES**, el cual **no se considera quantum-resistant**. De nuevo, recomendamos **CRYSTALS-KYBER** como algoritmo para el cifrado. En **primer lugar**, por las ventajas que le hemos expuesto anteriormente y, en **segundo lugar**, porque es el único algoritmo aprobado por el NIST

capaz de hacerlo, ya que los otros tres ganadores son algoritmos enfocados a la firma digital. Sin embargo, en este caso, **el tamaño de clave sí que varía mucho**, ya que el tamaño de clave de AES más usado es de **256 bits**, mientras que el de CRYSTALS-KYBER se recomienda uno de **1088 bits**. Además, la complejidad computacional de CRYSTALS-KYBER también es **mayor**, como ocurre con todos los algoritmos quantum-resistant.