UNIVERSIDAD NACIONAL SAN ANTONIO ABAD DEL CUSCO

FACULTAD DE INGENIERIA ELÉCTRICA, ELECTRÓNICA, INFORMÁTICA Y MECÁNICA

ESCUELA PROFESIONAL DE INGENIERIA INFORMATICA Y DE SISTEMAS

DESCIFRAR RSA CON COMPUTACIÓN CUÁNTICA

CURSO: Computación Cuántica

DOCENTE: Ing. Hans Harley Ccacyahuillca Bejar

ESTUDIANTES:

- Llasa Yucra, Ruth Margot (191892)
- Montes Huillca Franklin Jesus (182922)
- Gonzalo Cusi Fuentes (184196)
- Mendoza Huaillapuma Elisban (113562)
- Condori Huaychay Cesar Aparicio

SEMESTRE: 2024-I

CUSCO-PERÚ

2024

Contenido

1. INTRODUCCIÓN

La criptografía, a lo largo de la historia, ha sido fundamental para asegurar la comunicación y proteger la información. Su evolución ha seguido de cerca los avances en el cómputo y la tecnología, adaptándose a nuevas amenazas y oportunidades. En este contexto, la criptografía cuántica surge como una disciplina emergente que promete revolucionar la forma en que entendemos y aplicamos la seguridad de la información.

Esta monografía tiene como objetivo explorar el impacto de la computación cuántica en la criptografía, centrándose específicamente en el cifrado RSA, uno de los algoritmos criptográficos más utilizados en la actualidad.

"El nombre RSA proviene de las iniciales de sus tres creadores, Rivest, Shamir y Adleman, allá por 1997. Se trata de un algoritmo de cifrado asimétrico, o de clave pública, y es uno de los más utilizados en la actualidad. De hecho, la mayor parte de los sitios web hoy integran seguridad SSL/TLS, y permiten la autenticación mediante RSA." (Cordova, 2016)

RSA utiliza un par de claves: una pública para el cifrado y una privada para el descifrado. Este sistema asegura que, incluso si la clave pública es conocida, la clave privada sigue siendo confidencial. RSA se emplea ampliamente en aplicaciones como el protocolo SSL/TLS para asegurar la comunicación en Internet, y en firmas digitales para autenticar la identidad y garantizar la integridad de los mensajes.

2. HISTORIA Y DESARROLLO

El algoritmo RSA lleva el nombre de sus inventores: Ron Rivest, Adi Shamir y Leonard Adleman, quienes lo describieron por primera vez en 1977 en un artículo seminal titulado "A method for obtaining digital signatures and public-key cryptosystems". Fue uno de los primeros sistemas de cifrado asimétrico ampliamente disponible para el público, marcando un hito significativo en la historia de la criptografía moderna.

El desarrollo de RSA fue un avance crucial porque introdujo el concepto de utilizar dos claves diferentes para cifrar y descifrar datos. Esto contrasta con el cifrado simétrico, donde una sola clave se utiliza tanto para cifrar como para descifrar datos.

A lo largo de la historia, la humanidad ha tenido, por diversas razones, la necesidad de transmitir mensajes cuyo contenido permanezca oculto. La criptografía nació, en principio, como la habilidad para esconder información a cualquier persona que no le estuviera permitido leerla. A través de los siglos, se desarrollaron distintas técnicas, métodos e instrumentos que permitieron el desarrollo de este arte; la criptografía clásica abarca estas técnicas.

En 1948, Claude Shannon propuso la Teoría de la Información (la cual establece que la información es mesurable). Con la aparición de esta teoría, la criptografía dio un salto importante: dejó de ser un arte para convertirse en una ciencia considerada como una rama de las matemáticas. Ahora es llamada Criptografía Moderna. Este hecho dio pie al surgimiento de diversos algoritmos criptográficos que se valen del uso de la computadora para su implementación. Algunos métodos de cifrado de este tipo son: DES (apareció en 1976 y fue aceptado como un estándar por el gobierno de Estados Unidos en 1977), RSA (1977), CCE (1985), 3DES (1998) y, finalmente, AES (2002).

Tanto la Criptografía Clásica como la Criptografía Moderna tienen como objetivo principal cifrar la información (por medio de métodos matemáticos simples o complejos) para que pueda ser transmitida a un receptor y que solo este sea capaz de descifrarla. Básicamente, la criptografía es la ciencia encargada de transformar la información de tal manera que esta quede encubierta y sea

incomprensible para todo aquel que no cuente con la autorización y los medios correspondientes para acceder a ella.

3. COMPUTACIÓN CUÁNTICA

La computación cuántica es una disciplina emergente que utiliza principios de la mecánica cuántica para realizar cálculos de manera significativamente más rápida y eficiente que las computadoras clásicas. Esta tecnología tiene el potencial de revolucionar numerosos campos, desde la criptografía y la inteligencia artificial hasta la simulación de sistemas físicos complejos. La computación cuántica aprovecha fenómenos cuánticos como la superposición y el entrelazamiento para procesar información de maneras que no son posibles con la computación clásica, lo que permite resolver problemas de gran escala que actualmente son inabordables.

3.1. Principios de la mecánica cuántica

La mecánica cuántica es la rama de la física que estudia el comportamiento de las partículas a escalas subatómicas. A diferencia de la física clásica, que describe el comportamiento de objetos macroscópicos, la mecánica cuántica revela que las partículas pueden existir en múltiples estados a la vez (superposición) y que pueden estar interconectadas de maneras que no tienen paralelo en el mundo macroscópico (entrelazamiento). Estos principios desafían nuestra intuición pero han sido consistentemente confirmados por experimentos. La superposición permite que una partícula, como un electrón, exista en múltiples estados simultáneamente, mientras que el entrelazamiento crea una conexión instantánea entre partículas separadas, permitiendo que el estado de una afecte instantáneamente el estado de otra, sin importar la distancia que las separe.

3.2. Qubits, Superposición y Entrelazamiento

En la computación cuántica, la unidad básica de información es el qubit, que puede representar tanto un 0 como un 1 simultáneamente gracias a la superposición. Esto contrasta con los bits clásicos, que solo pueden ser 0 o 1 en un momento dado. Además, los qubits pueden entrelazarse, creando correlaciones entre ellos que permiten realizar operaciones complejas de manera mucho más eficiente que en una computadora clásica. Esta capacidad de procesar múltiples estados a la

vez y de operar con qubits entrelazados da a las computadoras cuánticas su enorme potencial para resolver problemas que serían impracticables para las máquinas clásicas, como la factorización de grandes números enteros, la simulación de moléculas y materiales complejos, y la optimización de sistemas.

4. FUNCIONAMIENTO DEL ALGORITMO RSA

El algoritmo RSA se basa en la teoría matemática de los números primos y la dificultad de factorización de números grandes. Aquí está el proceso básico de cómo funciona:

4.1. Generación de Claves

Antes de que podamos enviarle un mensaje a alguien más, ese destinatario habrá generado las claves.

Si lo igualamos al ejemplo del uso de la caja fuerte, sería equivalente a que el destinatario creara las llaves, una para poder cerrar la caja (llave pública) y otra para poder abrirla (llave privada). Este esquema sirve para lograr el denominado servicio de confidencialidad, pero puede usarse también para lograr autenticidad.

- Se eligen dos números primos.
- Calcula el producto n=p*q.
- Se calcula el producto de p y q, denotado como n (n = p * q).
- Se elige un número entero e tal que 1 < e < φ(n) y e sea coprimo con φ(n). La clave pública es el par (n, e).
- se calcula un número d tal que (d * e) % φ(n) = 1. La clave privada es (n, d)

4.2. Cifrado del Mensaje

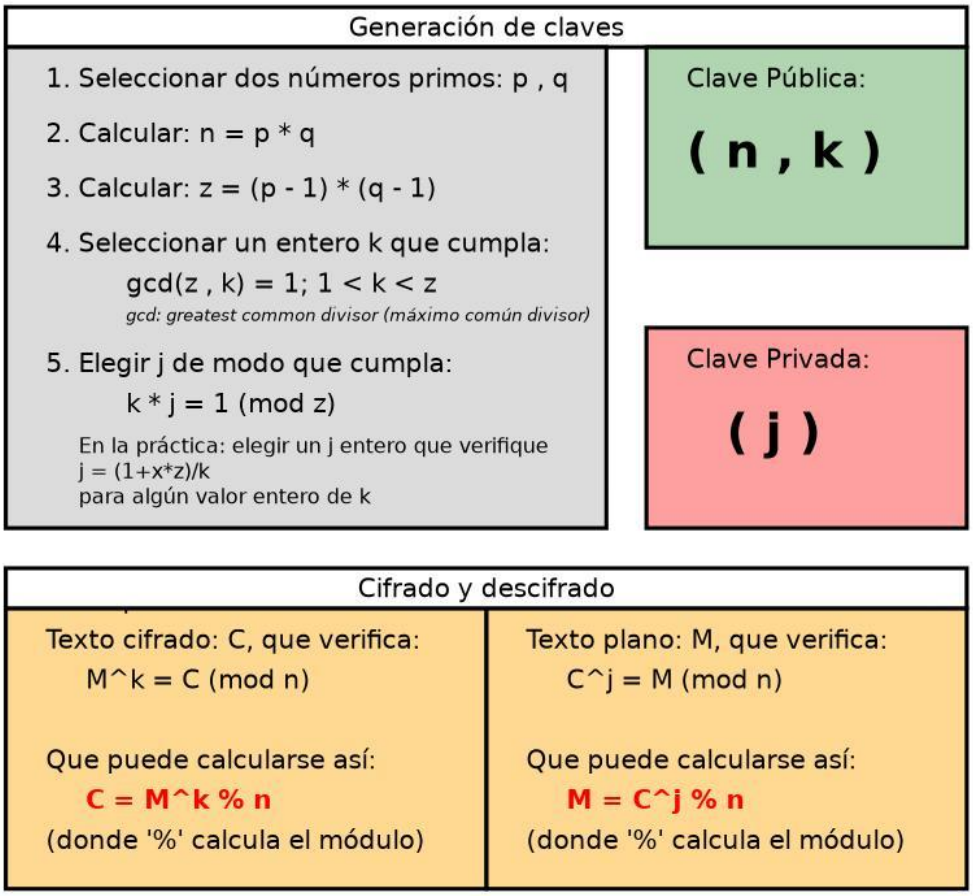
Dado un mensaje M, el emisor utiliza la clave pública del receptor (n, e) para cifrar el mensaje. El mensaje cifrado C se calcula como:

$$C = M^k \bmod n$$

4.3. **Descifrado del Mensaje**

El receptor utiliza su clave privada (n, d) para descifrar el mensaje. El mensaje original M se recupera como:

```
M = C^j mod n
```



5. APLICACIONES DEL ALGORITMO RSA

5.1. **Aplicaciones en el entorno**

- **Firmas Digitales:** RSA se emplea para crear firmas digitales que verifican la autenticidad y la integridad de los mensajes o documentos electrónicos. Una firma digital generada con RSA permite a los destinatarios confirmar que el mensaje proviene del remitente legítimo y que no ha sido alterado.

- **Autenticación y Verificación de Identidad:** RSA se usa en sistemas de autenticación para verificar la identidad de los usuarios. Esto es común en servicios bancarios en línea y en aplicaciones de acceso seguro a redes corporativas.
- **Distribución de Claves:** RSA permite el intercambio seguro de claves de cifrado simétrico a través de canales inseguros. Esto es fundamental en sistemas de cifrado híbrido, donde RSA se utiliza para cifrar una clave de sesión que luego se usa para el cifrado de datos con un algoritmo simétrico más eficiente.
- **Sistemas de Mensajería Segura:** RSA se emplea en aplicaciones de mensajería segura para cifrar mensajes y asegurar que solo el destinatario previsto pueda leer el contenido del mensaje.
- **Cifrado de Datos en Almacenamiento:** RSA puede utilizarse para cifrar datos almacenados en discos duros o dispositivos de almacenamiento, protegiendo la información sensible en caso de pérdida o robo del dispositivo.

5.2. Aplicaciones en seguridad más importantes que usan RSA

- **SSH (Secure Shell).** Es un protocolo de red con el que podemos acceder a una computadora remota, e incluso ejecutar aplicaciones en ella. Permite una comunicación segura sobre un canal inseguro. Se utiliza RSA para realizar la conexión, pero para el cifrado de los datos se utilizan procedimientos simétricos como AES. (Cantabria, 2003)
- **SSL (Secure Socket Layer).** Es un protocolo de cifrado utilizado en todas las conexiones https que hay en la Web. Fue desarrollado por Netscape para la transmisión de datos por Internet en 1994. La versión más utilizada de SSL es la 3.0. (Willems, 2009)
- **PGP (Prety Goog Privacy).** Es un software de uso privado para cifrar mensajes electrónicos. Fue desarrollado por Phil Zimmermann, quien al principio utilizó RSA para el intercambio de claves. Desde 2002 utiliza el procedimiento de ElGamal (1985).

6. RSA Y LA COMPUTACIÓN CUÁNTICA

RSA (Rivest-Shamir-Adleman) es uno de los sistemas de criptografía de clave pública más utilizados en el mundo. Introducido en 1978, RSA se basa en la dificultad de factorizar números grandes en sus factores primos, una tarea que, hasta ahora, ha demostrado ser computacionalmente inviable con los algoritmos y la tecnología clásicos. La seguridad de RSA radica en esta dificultad, lo que lo ha convertido en la base de muchas aplicaciones de seguridad, como el cifrado de datos, las firmas digitales y la autenticación.

La computación cuántica, sin embargo, plantea una amenaza fundamental a la seguridad de RSA. La computación cuántica utiliza principios de la mecánica cuántica, como la superposición y el entrelazamiento, para procesar información de una manera radicalmente diferente a las computadoras clásicas. Un algoritmo cuántico en particular, el algoritmo de Shor, propuesto por Peter Shor en 1994, es capaz de factorizar números grandes de manera eficiente, lo que podría romper la seguridad de RSA. En un ordenador cuántico suficientemente potente, este algoritmo puede factorizar números grandes en tiempo polinomial, haciendo obsoletos los esquemas de cifrado basados en la dificultad de la factorización.

Aunque los ordenadores cuánticos capaces de ejecutar el algoritmo de Shor a gran escala aún no están disponibles, el rápido progreso en el desarrollo de la computación cuántica, impulsado por empresas como Google e IBM, sugiere que es solo una cuestión de tiempo antes de que estos sistemas sean una realidad. Este avance tecnológico pone en riesgo la infraestructura de seguridad global que actualmente depende de RSA y otros algoritmos de clave pública similares.

7. ALGORITMO DE SHOR

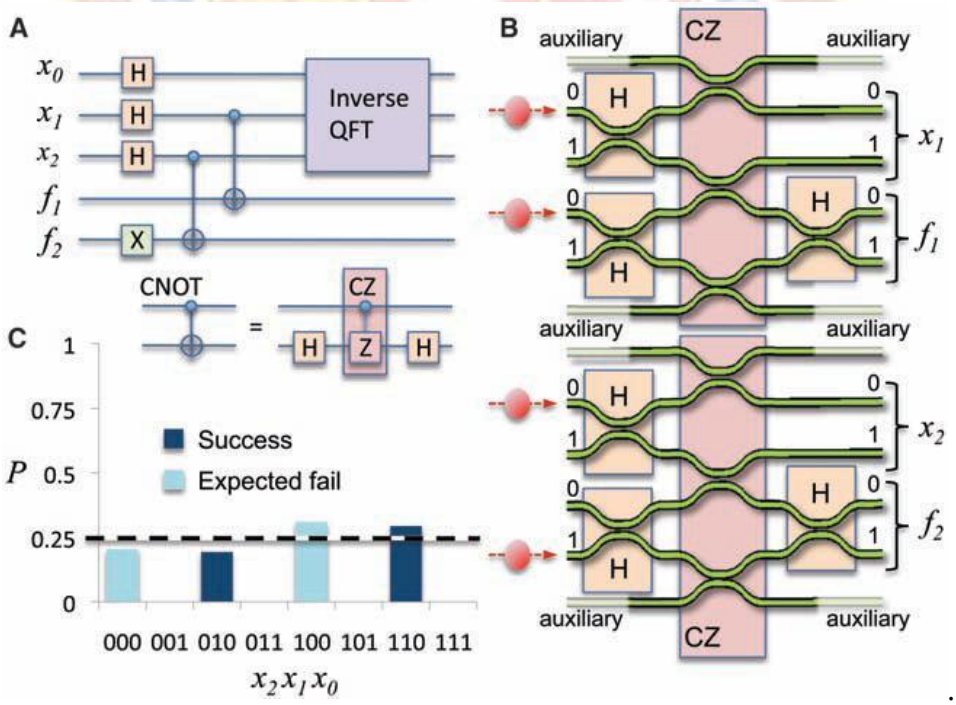
La factorización de números grandes es un problema computacionalmente difícil, fundamental para la seguridad de muchos algoritmos de cifrado, incluido RSA. Mientras que factorizar un número pequeño, como 51, es sencillo, factorizar un número de 100 dígitos puede llevar años usando computadoras tradicionales. La seguridad de las transacciones financieras depende de esta dificultad.

El Algoritmo de Shor, desarrollado para computadoras cuánticas, puede factorizar números grandes eficientemente, amenazando la seguridad de RSA. Aunque ha habido intentos de implementar el Algoritmo de Shor en sistemas cuánticos, la falta de suficientes qubits ha sido un obstáculo significativo. Este algoritmo requiere muchos qubits y una ejecución coherente para ser efectivo, lo que lo convierte en uno de los algoritmos cuánticos más desafiantes.

La capacidad del Algoritmo de Shor para factorizar números de 1024 bits en segundos con una computadora cuántica de 100 MIPS resalta la necesidad de desarrollar nuevas técnicas criptográficas, como la criptografía postcuántica, para proteger la información en un futuro donde la computación cuántica sea una realidad.

7.1. **Aplicaciones del Algoritmo de Shor con un ordenador cuántico**

- El algoritmo de Shor puede realizar la factorización de números primos muy grandes en un ordenador cuántico.
- El algoritmo de Shor puede utilizarse potencialmente para hackear RSA y otras formas de datos seguros
- El algoritmo de Shor puede resolver el problema de encontrar el periodo de una función



8. DESCIFRAR RSA CON COMPUTACIÓN CUÁNTICA

La computación cuántica promete revolucionar muchos campos, incluida la criptografía. Uno de los avances más significativos en esta área es el Algoritmo de Shor, desarrollado por el matemático Peter Shor en 1994. Este algoritmo tiene la capacidad de factorizar números enteros grandes en tiempo polinómico, una tarea que es extremadamente difícil para las computadoras clásicas y que constituye la base de la seguridad del cifrado RSA.

8.1. El Algoritmo de Shor y la Factorización de Números Grandes

El Algoritmo de Shor aprovecha dos principios fundamentales de la mecánica cuántica: la superposición y el entrelazamiento. Mediante estos principios, puede realizar cálculos en paralelo a una velocidad exponencialmente superior a la de los algoritmos clásicos. En términos simples, el algoritmo utiliza un proceso de transformaciones cuánticas para encontrar los factores primos de un número grande, esencialmente resolviendo el problema de factorización en tiempo polinómico. Para implementar el Algoritmo de Shor, se requiere un computador cuántico con un gran número de qubits funcionando de manera coherente, lo cual sigue siendo un desafío técnico significativo en la actualidad.

8.2. Implicaciones para el Cifrado RSA

El cifrado RSA se basa en la dificultad de factorizar grandes números compuestos de dos números primos. La clave pública de RSA contiene este número grande, n, mientras que la clave privada se basa en los factores primos de n. Si un atacante puede factorizar n, puede determinar la clave privada y descifrar los mensajes cifrados. El Algoritmo de Shor, ejecutado en un computador cuántico suficientemente poderoso, podría realizar esta factorización de manera eficiente, comprometiendo la seguridad de RSA. Esta capacidad plantea una amenaza existencial para los sistemas criptográficos actuales y está impulsando el desarrollo de nuevas formas de criptografía, conocidas como criptografía postcuántica, que sean resistentes a ataques cuánticos.

8.3. Estado Actual y Futuro de la Computación Cuántica

Aunque el Algoritmo de Shor ha sido probado en sistemas cuánticos experimentales, la implementación práctica en números lo suficientemente grandes para romper RSA aún no es posible debido a las limitaciones tecnológicas actuales, como el control de errores cuánticos y la estabilidad de los qubits. Sin embargo, los rápidos avances en la investigación cuántica sugieren que es solo cuestión de tiempo antes de que se logren los avances necesarios. Por esta razón, es crucial que la comunidad de seguridad de la información empiece a adoptar y desarrollar estándares de criptografía postcuántica para proteger la información en un futuro donde la computación cuántica sea una realidad operativa.

9. SEGURIDAD Y DESAFIOS

RSA (Rivest-Shamir-Adleman) ha sido un pilar de la criptografía moderna debido a su robustez en la protección de datos mediante la encriptación de clave pública. RSA se basa en la dificultad computacional para factorizar números grandes en sus factores primos, un problema que los algoritmos clásicos encuentran extremadamente difícil de resolver eficientemente. La seguridad de RSA se deriva de la complejidad de estos cálculos, lo que ha permitido su adopción generalizada en aplicaciones como la seguridad de internet, firmas digitales y protección de datos sensibles.

Sin embargo, la llegada de la computación cuántica y el desarrollo del algoritmo de Shor plantean serios desafíos:

- **Algoritmo de Shor**: Propuesto por Peter Shor en 1994, el algoritmo cuántico de Shor es capaz de factorizar enteros grandes en tiempo polinomial en un ordenador cuántico. Esto significa que, en teoría, un ordenador cuántico suficientemente poderoso podría descifrar rápidamente mensajes cifrados con RSA al calcular los factores primos de las claves públicas utilizadas .
- Impacto Potencial: Aunque los ordenadores cuánticos capaces de ejecutar el algoritmo de Shor a gran escala aún están en desarrollo, se espera que avances significativos en esta

tecnología puedan hacer que RSA sea vulnerable en un futuro no muy lejano. Empresas como Google e IBM están invirtiendo fuertemente en el desarrollo de computadoras cuánticas, acercándonos cada vez más a la realidad de máquinas capaces de romper RSA.

La preocupación es que la capacidad de un ordenador cuántico para factorizar números grandes eficientemente haría que los esquemas de criptografía basados en RSA y otros algoritmos de clave pública actuales se vuelvan inseguros. Esto plantea un riesgo significativo para la infraestructura de seguridad global, que depende en gran medida de la criptografía basada en RSA.

10. AVANCES Y VARIANTES

Criptografía Post-Cuántica: Ante la perspectiva de la amenaza cuántica, la comunidad científica está explorando y desarrollando activamente nuevas técnicas criptográficas que sean resistentes a los ataques cuánticos. Estas nuevas técnicas se conocen colectivamente como criptografía post-cuántica y están diseñadas para ser seguras tanto contra ataques cuánticos como clásicos. Entre las técnicas más prometedoras se encuentran:

- Criptografía Basada en Redes: Esta técnica utiliza problemas matemáticos relacionados con la estructura de redes complejas, como el problema del isomorfismo de grafos, que son difíciles de resolver incluso para los algoritmos cuánticos más avanzados. Los esquemas de criptografía basados en redes, como los criptosistemas de NTRU, se consideran fuertes candidatos para la criptografía post-cuántica debido a su resistencia a los ataques cuánticos .
- Criptografía Basada en Códigos de Corrección de Errores: Los códigos de corrección de errores son fundamentales en la transmisión de datos digitales, pero también pueden ser utilizados en la criptografía para complicar la decodificación de información. Los esquemas basados en códigos, como el criptosistema McEliece, utilizan grandes matrices de códigos que son difíciles de descifrar tanto para los ordenadores clásicos como para los cuánticos. Esta complejidad añadida proporciona una capa adicional de seguridad frente a los ataques cuánticos .

- Criptografía Multivariable: Utiliza sistemas de ecuaciones polinomiales no lineales que son difíciles de resolver, incluso para los algoritmos cuánticos. Los esquemas criptográficos multivariables, como el criptosistema Rainbow, se basan en la dificultad de resolver sistemas de ecuaciones no lineales, proporcionando una fuerte resistencia frente a ataques futuros.

11. IMPLEMENTACION Y PRUEBAS

11.1. **Instalación de librerías:**

Obteniendo nuestras dependencias

```
!pip install qiskit==0.46.0

Collecting qiskit==0.46.0
  Downloading qiskit-0.46.0-py3-none-any.whl.metadata (12 kB)
Collecting qiskit-terra==0.46.0 (from qiskit==0.46.0)
  Downloading qiskit_terra-0.46.0-cp38-abi3-manylinux_2_17_x86_64.manylinux2014_x86_64.whl.me
Collecting rustworkx>=0.13.0 (from qiskit-terra==0.46.0->qiskit==0.46.0)
  Downloading rustworkx-0.15.1-cp38-abi3-manylinux_2_17_x86_64.manylinux2014_x86_64.whl.metac
Requirement already satisfied: numpy<2,>=1.17 in /usr/local/lib/python3.10/dist-packages (frc
Collecting ply>=3.10 (from qiskit-terra==0.46.0->qiskit==0.46.0)
  Downloading ply-3.11-py2.py3-none-any.whl.metadata (844 bytes)
Requirement already satisfied: psutil>=5 in /usr/local/lib/python3.10/dist-packages (from qis
Requirement already satisfied: scipy>=1.5 in /usr/local/lib/python3.10/dist-packages (from qi
Requirement already satisfied: sympy>=1.3 in /usr/local/lib/python3.10/dist-packages (from qi
Collecting dill>=0.3 (from qiskit-terra==0.46.0->qiskit==0.46.0)
  Downloading dill-0.3.8-py3-none-any.whl.metadata (10 kB)
Requirement already satisfied: python-dateutil>=2.8.0 in /usr/local/lib/python3.10/dist-packa
Collecting stevedore>=3.0.0 (from qiskit-terra==0.46.0->qiskit==0.46.0)
  Downloading stevedore-5.2.0-py3-none-any.whl.metadata (2.3 kB)
Collecting symengine>=0.11 (from qiskit-terra==0.46.0->qiskit==0.46.0)
  Downloading symengine-0.11.0-cp310-cp310-manylinux_2_12_x86_64.manylinux2010_x86_64.whl.met
```

```
pip install qiskit-aer

Collecting qiskit-aer
  Downloading qiskit_aer-0.14.2-cp310-cp310-manylinux_2_17_x86_64.manylinux2014_x86_64.whl.me
Requirement already satisfied: qiskit>=0.45.2 in /usr/local/lib/python3.10/dist-packages (frc
Requirement already satisfied: numpy>=1.16.3 in /usr/local/lib/python3.10/dist-packages (from
Requirement already satisfied: scipy>=1.0 in /usr/local/lib/python3.10/dist-packages (from qi
Requirement already satisfied: psutil>=5 in /usr/local/lib/python3.10/dist-packages (from qis
Requirement already satisfied: qiskit-terra==0.46.0 in /usr/local/lib/python3.10/dist-package
Requirement already satisfied: rustworkx>=0.13.0 in /usr/local/lib/python3.10/dist-packages (
Requirement already satisfied: ply>=3.10 in /usr/local/lib/python3.10/dist-packages (from qis
Requirement already satisfied: sympy>=1.3 in /usr/local/lib/python3.10/dist-packages (from qi
Requirement already satisfied: dill>=0.3 in /usr/local/lib/python3.10/dist-packages (from qis
Requirement already satisfied: python-dateutil>=2.8.0 in /usr/local/lib/python3.10/dist-packa
Requirement already satisfied: stevedore>=3.0.0 in /usr/local/lib/python3.10/dist-packages (f
Requirement already satisfied: symengine>=0.11 in /usr/local/lib/python3.10/dist-packages (fr
Requirement already satisfied: typing-extensions in /usr/local/lib/python3.10/dist-packages (
Requirement already satisfied: six>=1.5 in /usr/local/lib/python3.10/dist-packages (from pyth
Requirement already satisfied: pbr!=2.1.0,>=2.0.0 in /usr/local/lib/python3.10/dist-packages
Requirement already satisfied: mpmath<1.4,>=1.1.0 in /usr/local/lib/python3.10/dist-packages
Downloading qiskit_aer-0.14.2-cp310-cp310-manylinux_2_17_x86_64.manylinux2014_x86_64.whl (12.
   ━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━ 12.4/12.4 MB 45.5 MB/s eta 0:00:00
Installing collected packages: qiskit-aer
Successfully installed qiskit-aer-0.14.2
```

```
pip install qiskit-aer-gpu

Collecting qiskit-aer-gpu
  Downloading qiskit_aer_gpu-0.14.2-cp310-cp310-manylinux_2_17_x86_64.manylinux2014_x86_64.
Requirement already satisfied: qiskit>=0.45.2 in /usr/local/lib/python3.10/dist-packages (f
Requirement already satisfied: numpy>=1.16.3 in /usr/local/lib/python3.10/dist-packages (fr
Requirement already satisfied: scipy>=1.0 in /usr/local/lib/python3.10/dist-packages (from
Requirement already satisfied: psutil>=5 in /usr/local/lib/python3.10/dist-packages (from q
Collecting nvidia-cuda-runtime-cu12>=12.1.105 (from qiskit-aer-gpu)
  Downloading nvidia_cuda_runtime_cu12-12.5.82-py3-none-manylinux2014_x86_64.whl.metadata (
Collecting nvidia-nvjitlink-cu12 (from qiskit-aer-gpu)
  Downloading nvidia_nvjitlink_cu12-12.5.82-py3-none-manylinux2014_x86_64.whl.metadata (1.5
Collecting nvidia-cublas-cu12>=12.1.3.1 (from qiskit-aer-gpu)
  Downloading nvidia_cublas_cu12-12.5.3.2-py3-none-manylinux2014_x86_64.whl.metadata (1.5 k
Collecting nvidia-cusolver-cu12>=11.4.5.107 (from qiskit-aer-gpu)
```

```
pip install qiskit-aer-gpu-cu11

Collecting qiskit-aer-gpu-cu11
  Downloading qiskit_aer_gpu_cu11-0.14.2-cp310-cp310-manylinux_2_17_x86_64.manylinux2014_x8
Requirement already satisfied: qiskit>=0.45.2 in /usr/local/lib/python3.10/dist-packages (f
Requirement already satisfied: numpy>=1.16.3 in /usr/local/lib/python3.10/dist-packages (fr
Requirement already satisfied: scipy>=1.0 in /usr/local/lib/python3.10/dist-packages (from
Requirement already satisfied: psutil>=5 in /usr/local/lib/python3.10/dist-packages (from q
Collecting nvidia-cuda-runtime-cu11>=11.8.89 (from qiskit-aer-gpu-cu11)
  Downloading nvidia_cuda_runtime_cu11-11.8.89-py3-none-manylinux1_x86_64.whl.metadata (1.5
```

11.2. **Importación de herramientas**

```
[5]  import numpy as np
     from qiskit import *
     from math import sqrt,log,gcd
     import random
     from random import randint
     import rsa
```

11.3. **Implementación de módulos principales**

Calcular la inversa modular

```
[6] def mod_inverse(a, m):
        for x in range(1, m):
            if (a * x) % m == 1:
                return x
        return -1
```

Comprobando la primalidad

```
def isprime(n):
    if n < 2:
        return False
    elif n == 2:
        return True
    else:
        for i in range(1, int(sqrt(n)) + 1):
            if n % i == 0:
                return False
    return True
```

+ Código + Texto | Copiar en Drive

Generando pares clave-valor

0s

```
def generate_keypair(keysize):
    p = randint(1, 1000)
    q = randint(1, 1000)
    nMin = 1 << (keysize - 1)
    nMax = (1 << keysize) - 1
    primes = [2]
    start = 1 << (keysize // 2 - 1)
    stop = 1 << (keysize // 2 + 1)
    if start >= stop:
        return []
    for i in range(3, stop + 1, 2):
        for p in primes:
            if i % p == 0:
                break
        else:
            primes.append(i)
    while (primes and primes[0] < start):
        del primes[0]
    # Seleccione dos números primos aleatorios P y Q
    while primes:
        p = random.choice(primes)
        primes.remove(p)
        q_values = [q for q in primes if nMin <= p * q <= nMax]
        if q_values:
            q = random.choice(q_values)
            break
    # Calcule n
    n = p * q
    # Calcule phi
    phi = (p - 1) * (q - 1)
    # Seleccione e
    e = random.randrange(1, phi)
    g = gcd(e, phi)
    # Calcule d
    while True:
        e = random.randrange(1, phi)
        g = gcd(e, phi)
        d = mod_inverse(e, phi)
        if g == 1 and e != d:
            break

    return ((e, n), (d, n))
```

11.4. Cifrado y descifrado

```
## Paso de cifrado

[9]  def encrypt(plaintext, package):
         e, n = package
         ciphertext = [pow(ord(c), e, n) for c in plaintext]
         return ''.join(map(lambda x: str(x), ciphertext)), ciphertext

Paso de descifrado

[10] def decrypt(ciphertext, package):
         d, n = package
         plaintext = [chr(pow(c, d, n)) for c in ciphertext]
         return (''.join(plaintext))
```

11.5. Resultados

Se hace la prube cifrando la palabra “ejecutar”

```
Ahora probemos con un mensaje de muestra.

Generar claves

[11] !pip  install pycryptodome #Instale el paquete Pycryptodome, que proporciona funcionalidad RSA

Collecting pycryptodome
  Downloading pycryptodome-3.20.0-cp35-abi3-manylinux_2_17_x86_64.manylinux2014_x86_64.whl.metadata (3.4 kB)
Downloading pycryptodome-3.20.0-cp35-abi3-manylinux_2_17_x86_64.manylinux2014_x86_64.whl (2.1 MB)
   ━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━ 2.1/2.1 MB 29.4 MB/s eta 0:00:00
Installing collected packages: pycryptodome
Successfully installed pycryptodome-3.20.0

[12] from Crypto.PublicKey import RSA

     bit_length = int(input("Ingrese la longitud del bit: "))

     public_k, private_k = generate_keypair(2**bit_length)

Ingrese la longitud del bit: 3
```

Cifrado

```
[13] plain_txt = input("Ingrese un mensaje: ")
     cipher_txt, cipher_obj = encrypt(plain_txt, public_k)

     print("Mensaje encriptado: {}".format(cipher_txt))
```

```
Ingrese un mensaje: ejecutar
Mensaje encriptado: 18612318619011720715856
```

Descifrado

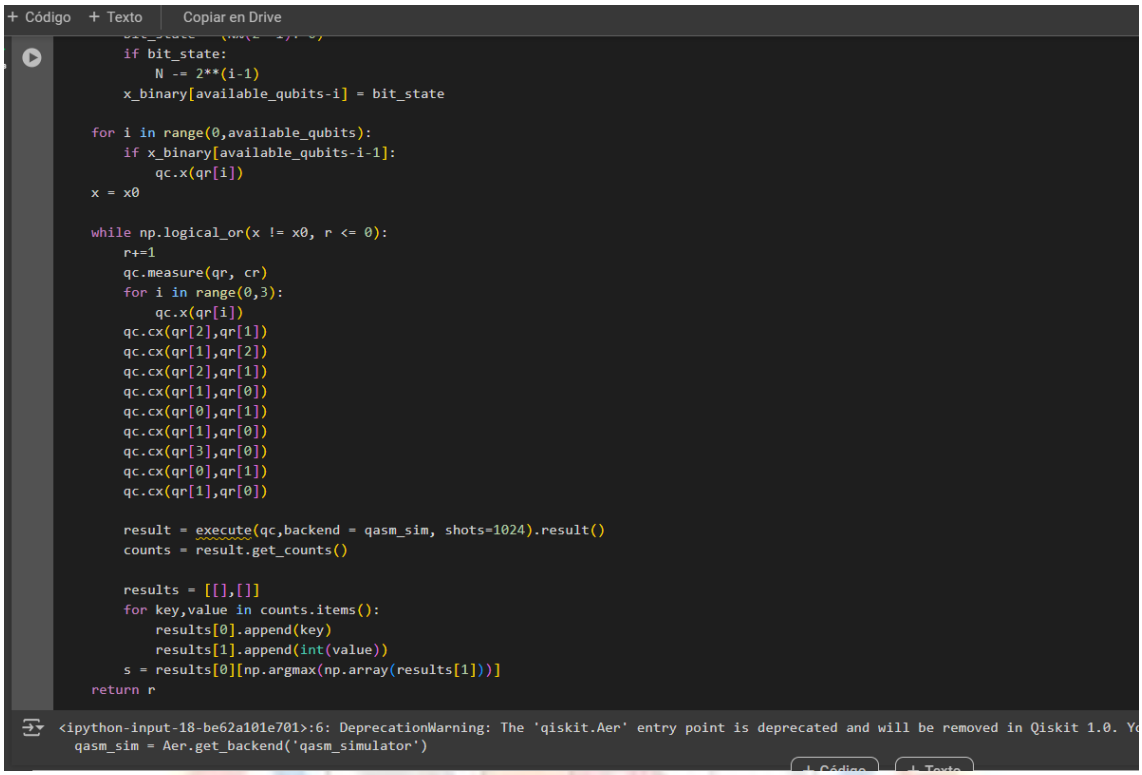
```
[14] print("Mensaje descifrado: {}".format(decrypt(cipher_obj, private_k)))
```

```
Mensaje descifrado: ejecutar
```

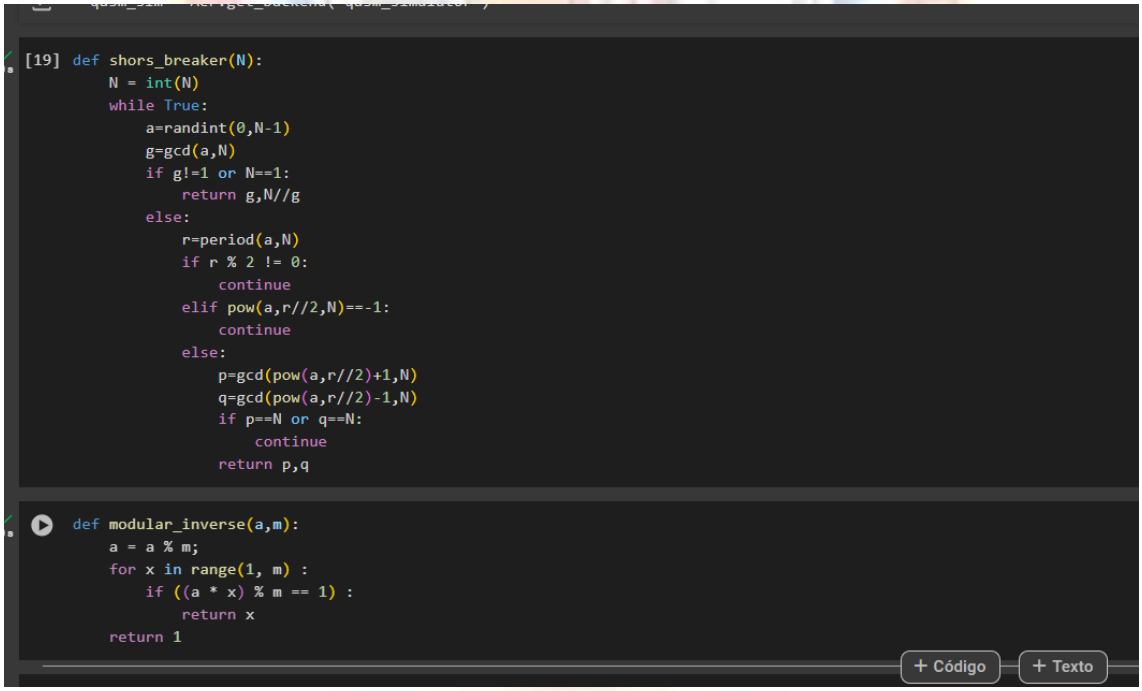
11.6. **Algoritmo Shor**

```
[17] from qiskit import QuantumCircuit, Aer
     from qiskit_aer import AerSimulator
```

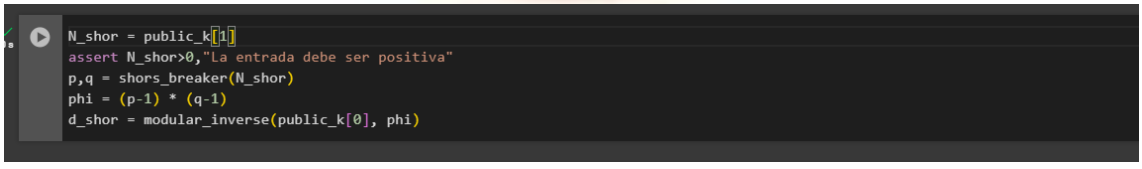
```
import numpy as np

from random import randint

from qiskit_aer import AerSimulator
qasm_sim = Aer.get_backend('qasm_simulator')
def period(a,N):

    available_qubits = 16
    r=-1

    if N >= 2**available_qubits:
        print(str(N)+' es demasiado grande para IBMQX')

    qr = QuantumRegister(available_qubits)
    cr = ClassicalRegister(available_qubits)
    qc = QuantumCircuit(qr,cr)
    x0 = randint(1, N-1)
    x_binary = np.zeros(available_qubits, dtype=bool)

    for i in range(1, available_qubits + 1):
        bit_state = (N%(2**i)!=0)
        if bit_state:
            N -= 2**(i-1)
        x_binary[available_qubits-i] = bit_state

    for i in range(0,available_qubits):
        if x_binary[available_qubits-i-1]:
            qc.x(qr[i])
    x = x0
```



```
+ Código + Texto  Copiar en Drive

        if bit_state:
            N -= 2**(i-1)
        x_binary[available_qubits-i] = bit_state

    for i in range(0,available_qubits):
        if x_binary[available_qubits-i-1]:
            qc.x(qr[i])
    x = x0

    while np.logical_or(x != x0, r <= 0):
        r+=1
        qc.measure(qr, cr)
        for i in range(0,3):
            qc.x(qr[i])
        qc.cx(qr[2],qr[1])
        qc.cx(qr[1],qr[2])
        qc.cx(qr[2],qr[1])
        qc.cx(qr[1],qr[0])
        qc.cx(qr[0],qr[1])
        qc.cx(qr[1],qr[0])
        qc.cx(qr[3],qr[0])
        qc.cx(qr[0],qr[1])
        qc.cx(qr[1],qr[0])

        result = execute(qc,backend = qasm_sim, shots=1024).result()
        counts = result.get_counts()

        results = [[],[]]
        for key,value in counts.items():
            results[0].append(key)
            results[1].append(int(value))
        s = results[0][np.argmax(np.array(results[1]))]
    return r

<ipython-input-18-be62a101e701>:6: DeprecationWarning: The 'qiskit.Aer' entry point is deprecated and will be removed in Qiskit 1.0. Yo
  qasm_sim = Aer.get_backend('qasm_simulator')
```

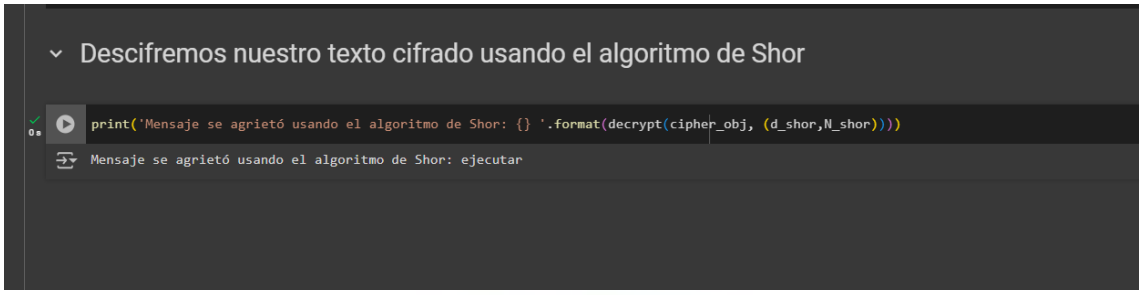


```
[19] def shors_breaker(N):
        N = int(N)
        while True:
            a=randint(0,N-1)
            g=gcd(a,N)
            if g!=1 or N==1:
                return g,N//g
            else:
                r=period(a,N)
                if r % 2 != 0:
                    continue
                elif pow(a,r//2,N)==-1:
                    continue
                else:
                    p=gcd(pow(a,r//2)+1,N)
                    q=gcd(pow(a,r//2)-1,N)
                    if p==N or q==N:
                        continue
                    return p,q

    def modular_inverse(a,m):
        a = a % m;
        for x in range(1, m) :
            if ((a * x) % m == 1) :
                return x
        return 1

+ Código + Texto
```



```
N_shor = public_k[1]
assert N_shor>0,"La entrada debe ser positiva"
p,q = shors_breaker(N_shor)
phi = (p-1) * (q-1)
d_shor = modular_inverse(public_k[0], phi)
```

11.7.**Desciframos**



12. CONCLUSIÓN

La evolución de la criptografía y la computación ha sido un viaje marcado por la constante adaptación a nuevos desafíos y avances tecnológicos. La aparición de la mecánica cuántica y su aplicación en la computación cuántica representa un cambio paradigmático que podría revolucionar la forma en que aseguramos la información. La capacidad de la computación cuántica para realizar cálculos exponencialmente más rápidos que las computadoras clásicas, especialmente en problemas de factorización, pone en riesgo los sistemas criptográficos tradicionales, como el cifrado RSA.

El cifrado RSA, uno de los algoritmos de seguridad más utilizados hoy en día, se basa en la dificultad de factorizar números grandes, una tarea que la computación cuántica, mediante el Algoritmo de Shor, podría realizar en tiempo polinómico. Aunque actualmente no existen computadoras cuánticas lo suficientemente avanzadas para implementar el Algoritmo de Shor a gran escala, los progresos en esta tecnología sugieren que es solo cuestión de tiempo antes de que esta capacidad se vuelva una realidad práctica.

La amenaza que la computación cuántica representa para la criptografía actual subraya la necesidad urgente de desarrollar y adoptar nuevas técnicas criptográficas, conocidas como criptografía postcuántica, que sean resistentes a ataques cuánticos. La transición hacia estos nuevos estándares de seguridad es crucial para garantizar la protección continua de la información en un futuro dominado por la computación cuántica.

Bibliografía

Cantabria, U. d. (2003). Proptocolos Crptográficos y Seguridad en Redes . En J. Gutierrez.

Cordova, D. (2016). *Junco TIC*. Obtenido de https://juncotic.com/rsa-como-funciona-este-algoritmo/

Willems, W. (2009). Introduccion a la critografía de clave Pública . En W. Willems. Colombia.

Rivest, R. L., Shamir, A., & Adleman, L. M. (1978). A method for obtaining digital signatures and public-key cryptosystems. Communications of the ACM, 21(2), 120-126.

Stallings, W. (2017). Cryptography and Network Security: Principles and Practice (7th ed.). Pearson Education.