DISTRIBUIÇÃO DE CHAVES QUÂNTICAS COM O PROTOCOLO BB84 NOS SIMULADORES QUÂNTICOS DA IBM

QUANTUM KEY DISTRIBUITION WITH THE BB84 PROTOCOL INSIDE IBM'S QUANTUM SIMULATORS

Victor Luis Rodrigues Pereira Ferreira, Faculdade de Tecnologia de Americana – Ministro Ralph Biasi, victor.ferreira31@fatec.sp.gov.br
Sthéfanie Costa Amaro, Faculdade de Tecnologia de Americana – Ministro Ralph Biasi, sthefanie.amaro@fatec.sp.gov.br
Lucas Gomes Pinheiro, Faculdade de Tecnologia de Americana – Ministro Ralph Biasi, lucas.pinheiro7@fatec.sp.gov.br
Mariana Godoy Vazquez Miano, Faculdade de Tecnologia de Americana – Ministro Ralph Biasi, mariana.miano@fatec.sp.gov.br

Resumo

Devido ao valor atribuído às informações nas últimas décadas, sugiram e ainda surgem métodos de proteção à informação que visam assegurar que o único destinatário a ter acesso seja aquele o qual intendido. Atualmente, o método mais comum de cifragem de mensagens é a criptografia, particularmente o método RSA. Porém, esse método está sob risco devido ao novo paradigma emergente, a computação quântica, que é capaz de decifrar o método RSA em até segundos (o que previamente, no paradigma clássico demoraria centenas de anos). O objetivo desse artigo é demonstrar a importância da criptografia quântica através da teoria e apresentação de um código que simula a geração e distribuição de chaves quânticas através do protocolo quântico BB84 de maneira automática, escrito na linguagem Python juntamente com OpenQASM através da plataforma híbrida IBM *Quantum Lab*, utilizando o simulador quântico Qiskit AER *Simulator*. Conforme o novo paradigma computacional se torna popular, também a criptografia quântica.

Palavras-chave: Distribuição de chaves quânticas, Criptografia quântica, Protocolo BB84.

Abstract

Due to the value assigned to information in the last decades, protection methods that aim to assure that the only receiver to access the information is the one intended were and are still being created. Nowadays the most common method of message encryption is cryptography, RSA cryptography, particularly. However, this encryption method is at stake due to the new emerging computing paradigm, quantum computing, that can decipher the RSA cryptography method in mere seconds (something that previously, in classical computing, would take up to hundreds of years). This paper has the objective of demonstrating the importance of quantum cryptography through theory and presenting a program that simulates quantum key distribution through the BB84 protocol automatically, written in Python language in conjunction with OpenQASM, making use of IBM's quantum simulator Qiskit AER, within the hybrid platform IBM *Quantum Lab*. As quantum computing becomes popular, so does quantum cryptography.

Keywords: Quantum Key Distribuition, Quantum Cryptography, BB84 Protocol.

1. Introdução

Desde os primórdios a comunicação é essencial para a compreensão entre os seres humanos, a troca de informações se dá, de forma mais comum, através das diversas linguagens criadas para esse intuito, sejam elas verbais ou não.

Com o avanço tecnológico, as formas de comunicação, sejam entre indivíduos, organizações ou governos foi facilitada, de tal maneira que não apenas mensagens são trocadas por meios digitais. A emissão de dados sensíveis já se tornou algo consueto através de meios como a Internet, sejam esses dados correspondentes a informações bancárias, fotografias, dentre outras (CHAVES, 2018).

Diante deste contexto, entra em discussão a necessidade de garantir a segurança da informação, princípio o qual define que o acesso deve ser garantido somente aos usuários autorizados (confidencialidade), dentre outras características. Se tratando da proteção de dados e informação, a criptografia entra em cena com papel principal.

De acordo com Eswaran *et al.* (2022, p. 834): “atualmente, dentre a vastidão de sistemas criptográficos, os esquemas RSA (sistema de chave pública) e os esquemas de curva elíptica são os mais populares e comumente utilizados em aplicações de vários escopos”.

Apesar de populares e eficientes, esses sistemas criptográficos estão sob ameaça dos avanços tecnológicos emergentes do paradigma quântico de computação. Computadores quânticos possuem muito mais poder de processamento do que computadores denominados “clássicos” (máquinas convencionais baseadas no modelo de Alan Turing), essa alta eficiência faz com que a capacidade de quebra dos sistemas criptográficos existentes aumente significativamente, consequentemente aumentando também o risco de vulnerabilidades e ciberataques (ESWARAN *et al.*, 2022).

Algoritmos clássicos de criptografia como o RSA podem ser quebrados e possuem solução, porém são necessárias até centenas de anos para que computadores clássicos consigam solucioná-los, dado que o tamanho da chave seja adequado. A segurança da criptografia clássica vem da suposição de que não haja máquina com poder computacional o suficiente para solucionar o algoritmo em tempo prático, e não por possuir algoritmos insolúveis. Nos últimos anos essa suposição tem se provado errônea devido ao advento de máquinas quânticas e protocolos quânticos cada vez mais capazes (MIANO, 2020).

Existem duas classes de algoritmos quânticos com uso direcionado a solução de problemas clássicos, uma baseada na Transformada de Fourier Quântica (QFT) de Shor e outra baseada nos Algoritmos de Grover para a realização de busca quântica. Algoritmos baseados na QFT de Shor servem para fatorar números grandes em fatores primos, problema matemático tão complexo que foi implementado na criptografia RSA como medida de segurança. O Algoritmo de Shor em suas versões quânticas consegue "quebrar" a criptografia clássica, seu funcionamento envolve encontrar o período de uma função e em seguida os fatores do valor demandado (MIANO, 2020).

Visto que a criptografia clássica está sob ameaça do novo paradigma computacional, surge a necessidade de um método mais eficiente de proteção de dados e informações. Não por acaso, a computação quântica visa solucionar os problemas clássicos, apesar de criar um problema ao paradigma clássico de computação. O princípio conhecido como criptografia quântica ou distribuição de chaves quânticas é introduzido, o qual utiliza das propriedades da mecânica quântica para criar formas de distribuição de informação provadamente seguras (NIELSEN; CHUANG, 2010).

O objetivo geral deste trabalho é evidenciar a importância da computação quântica na segurança da informação, gerando e analisando um código de distribuição de chaves quânticas através do protocolo BB84 de maneira automática, além de detalhar a importância da criptografia quântica visando as ameaças à segurança dos criptossistemas clássicos.

O objeto deste projeto é um código *open source* escrito em Python que gera chaves criptográficas simétricas quânticas através do simulador quântico IBM Qiskit AER *Simulator*. Como plataformas e ambientes de desenvolvimento, foram selecionados o IBM *Quantum Lab* (parte do que era previamente conhecido como IBM *Quantum Experience*) e VSCode.

2. Referencial Teórico

Antes de mais nada, é essencial estabelecer conhecimentos sobre as áreas as quais serão discutidas.

2.1. Computação Clássica x Computação Quântica

Em 1936, o matemático Alan Turing anunciou um documento que, segundo Nielsen e Chuang (2010, p. 4) se tornou a encarnação moderna da ciência da computação:

> O documento explicava em detalhes o desenvolvimento da noção abstrata de uma máquina programável agora conhecida como Máquina de Turing, em sua homenagem.

> Turing mostrou que existe uma "Máquina Universal de Turing" que consegue simular qualquer outra máquina baseada em seu modelo. Ademais, o matemático afirmou que a Máquina Universal de Turing *captura completamente* o significado de realizar uma tarefa utilizando meios algorítmicos. Isto é, se um algoritmo consegue ser executado em *qualquer* hardware (um computador pessoal, por exemplo), então existe um algoritmo equivalente para uma Máquina Universal de Turing, que desempenha exatamente a mesma tarefa que o algoritmo rodando no computador pessoal.

Alonzo Church e Alan Turing criaram essa asserção sobre a relação que um algoritmo executado em qualquer dispositivo físico também pode ser executado na Máquina Universal de Turing. Essa tese ficou conhecida como a Tese Church-Turing, em homenagem aos dois cientistas, e sua aceitação "pavimentou" o caminho para uma teoria da ciência da computação mais rica (NIELSEN; CHUANG, 2010).

Computadores clássicos conseguem simular computadores quânticos de maneira eficiente, mas não de maneira holística, o que levou a Tese Church-Turing a sofrer diversas pequenas alterações que a deixassem mais forte, principalmente por David Deustch que sugeriu em 1985 que computadores quânticos conseguem resolver problemas impossíveis para máquinas clássicas. Peter Shor em 1994 demonstrou que dois grandes problemas para a computação clássica (fatoração de grandes números inteiros em fatores primos e o problema de "logaritmo discreto") poderiam ser facilmente resolvidos no paradigma quântico, fortalecendo ainda mais a hipótese de Deustch (NIELSEN; CHUANG, 2010).

Muitos outros exemplos surgiram com o passar dos anos provando que a computação quântica é mais eficiente que a computação clássica em diversas áreas, o suficiente para ser adotada sobre a sua predecessora, uma dessas áreas é a segurança da informação no ramo da criptografia.

Antes de partir para a área da criptografia, é necessário estabelecer princípios da computação quântica e mecânica quântica relevantes a esse estudo, leitores interessados em uma abordagem mais profunda do paradigma quântico podem se dirigir a obra de Nielsen e Chuang (2010) onde conceitos mais básicos como o Espaço Hilbert, notação de Dirac e sobreposição de estados são explorados em detalhe.

2.1.1. *Bit* quântico (*qubit*)

A unidade de medida de um computador clássico é denominada *bit* e pode assumir somente dois valores: 0 ou 1. Entretanto, em um computador quântico as coisas mudam, dado

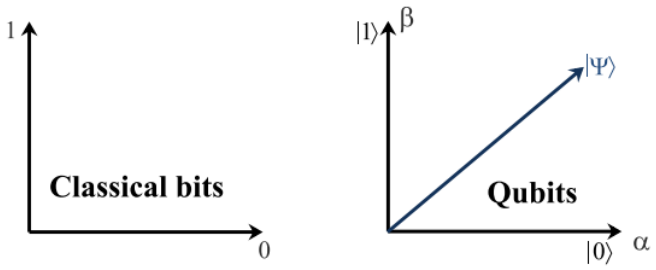
que as leis da física quântica permitem fenômenos como sobreposição de estados e emaranhamento (KOCKUM; NORI, 2019).

Na computação quântica, a unidade de medida mais simples é o *qubit* (abreviação de *quantum bit*) que possui dois estados definitivos |0⟩ (estado base) e |1⟩ (estado "animado"), onde | ⟩ correspondem a notação de Dirac. Contudo, diferentemente da computação clássica, além dos dois estados principais existem infinitos estados equivalentes a sobreposições de |0⟩ e |1⟩,

$$|\psi\rangle = \alpha\,|0\rangle + \beta\,|1\rangle,$$

onde α e β representam números complexos que satisfazem a equação $|\alpha|^2 + |\beta|^2 = 1$. A visualização desse esquema se torna mais fácil de compreender quando representado na Esfera de Bloch (KOCKUM; NORI, 2019). A figura 1 ilustra a representação gráfica vetorial que diferencia os *bits* clássicos e quânticos

Figura 1 - Representação vetorial de *bits* clássicos e *qubits*



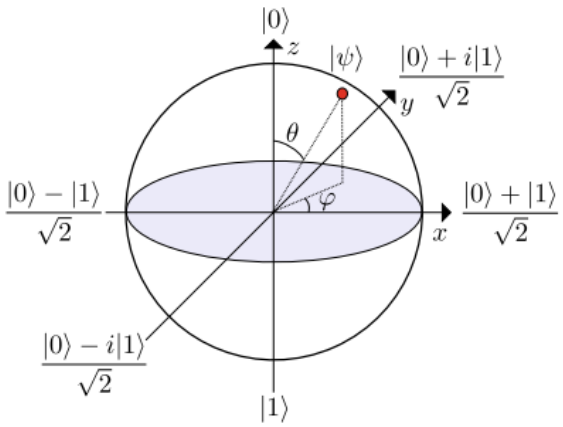
Fonte: (SHARMA, et al, p. 2053, 2021)

2.1.2. Esfera de Bloch

Nielsen e Chuang (2010, p. 15) explicam que a Esfera de Bloch provê uma ótima visualização do estado de um único *qubit*, servindo também como uma excelente base de testes para ideias sobre computação e informação quântica.

A esfera de Bloch também serve como representação geométrica de um *qubit* (LACAVA; MIANO, 2018). A figura 2 ilustra a Esfera de Bloch.

Figura 2 - Representação de um *qubit* na Esfera de Bloch



Fonte: (KOCKUM; NORI, p. 704, 2019)

De acordo com Kockum e Nori (2019, p. 704):

> "o polo norte da esfera de Bloch mostrada na figura (2) representa o estado base $|0\rangle$ e o polo sul o estado 'animado' $|1\rangle$. Para converter qualquer suposição arbitraria de $|0\rangle$ e $|1\rangle$ para um ponto na esfera, a parametrização: $\cos\frac{\theta}{2}\,|0\rangle + e^{i\varphi}\sin\frac{\theta}{2}\,|1\rangle$ é utilizada".

2.1.3. Polarização de Fótons

A polarização de um fóton se refere ao seu ângulo de orientação, porém esse conceito é melhor aplicado quando o ponto de vista observado é aquele em que o fóton é uma onda eletromagnética a qual vibra em uma direção específica (GUERRA *et al*., 2021).

2.2. Criptografia Clássica

A encriptação de mensagens não é a única função da criptografia, porém para o presente artigo é a mais relevante. É importante ressaltar que na maioria dos exemplos de esquemas criptográficos, o remetente da mensagem é referido como "Alice", o destinatário como "Bob" e um possível espião como "Eva".

Baseando-se na obra *Introduction to Modern Cryptography* (2007) de Jonathan Katz e Yehuda Lindell, Chaves (2018, p.13) descreve a encriptação de mensagens como um procedimento em que um remetente pretende enviar uma mensagem a um destinatário, de forma que a decifragem de seu conteúdo seja possível apenas se o receptor possuir a chave para decifrar a mensagem cifrada. O conteúdo da mensagem é codificado através de um algoritmo (uma cifra) que encripta o texto puro (mensagem) e a transforma num texto cifrado.

Esquemas de criptografia geralmente são estruturados utilizando três algoritmos:

I. Geração (*G*): algoritmo que gera de forma probabilística um par de chaves que serão utilizados para encriptação e decriptação da mensagem;

II. Encriptação (*E*): algoritmo que encripta a mensagem de maneira aleatória. (A aleatoriedade garante resultados diferentes para o mesmo texto puro);

III. Decifragem (*D*): algoritmo de decifragem do texto encriptado, onde não deve existir probabilidade do resultado ser diferente do texto puro original.

Esquemas de criptografia podem ser divididos em duas classes: de chave simétrica (ou chave privada) e de chave assimétrica (ou chave pública) (CHAVES, 2018).

2.2.1. Criptografia RSA

Baseado na dificuldade computacional clássica de fatorar grandes números (*N*) em fatores primos e na função totiente de Euler (MIANO, 2020), o protocolo RSA foi criado por Rivest, Shamir e Adleman em 1978. Para iniciar a definição do protocolo, é necessário assumir que $N = p \cdot q$, onde p e q são dois números primos. Segundo Chaves (2018, p. 20), o processo se inicia da seguinte maneira:

I. Geração (*G*): o algoritmo de geração *G* resulta em (*N, e, d*) tendo como parâmetros de entrada *n* no formato 1^n e (*e, d*) os quais são números inteiros. A chave pública será dada por (*N, e*) e a chave privada por (*N, d*).

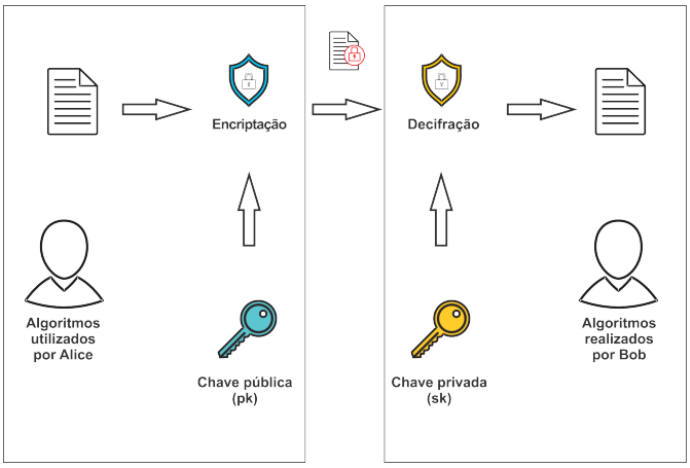
II. Encriptação (*E*): o algoritmo de encriptação *E* resulta em *c*, onde $c = [m^e \bmod N]$. *c* é o texto cifrado e o algoritmo recebe a chave pública e uma mensagem *m*.

III. Decifragem (*D*): o algoritmo de decifragem *D* utilizará a chave privada gerada anteriormente, bem como o texto cifrado *c* e resultará em *m* tal que $m = [c^d \bmod N]$.

Um dos problemas do protocolo já surge no fato do algoritmo de encriptação *E* não ser probabilístico, o que pode acarretar o envio do mesmo texto cifrado para a mesma mensagem sempre. Além disso, apesar de a computação clássica ainda não possuir uma maneira eficiente de fatorar números grandes em fatores primos, a computação quântica apresenta uma grande ameaça devido a eficiência do algoritmo de fatoração de Shor em computadores quânticos. Dessa maneira, a existência de computadores quânticos torna protocolos RSA inseguros (CHAVES, 2018).

A figura 3 ilustra como um protocolo de criptografia de chaves assimétrica (como o RSA) funciona.

Figura 3 – Representação da comunicação através de um esquema geral de criptografia assimétrica



Fonte: (CHAVES, 2018)

A necessidade de um novo formato de criptografia emerge, e uma ótima candidata é a criptografia quântica.

2.3. Criptografia Quântica

É importante entender que a fragilidade dos criptossistemas clássicos atuais não é uma ameaça em potencial apenas para o presente, mas também para o futuro. É possível que espiões interceptem criptogramas que ainda não são capazes de decriptar, contudo eles podem armazenar comunicações criptografadas e esperar até que um computador quântico suficientemente capaz esteja disponível (ou um novo algoritmo clássico seja descoberto). Isso significa que a confidencialidade das mensagens pode estar com os dias contados (PIRANDOLA *et al*., 2020).

Pirandola *et al*. (2020, p. 3) propõem que contramedidas de segurança ficam claramente necessárias. Um dos métodos conhecidos é a criptografia pós-quântica, que aborda o desenvolvimento de criptossistemas que são resistentes a fatoração e a outros tipos de algoritmos quânticos. Apesar de certamente ser uma opção, não é a solução para o problema como um todo. Ainda podem existir uma infinidade de algoritmos quânticos (ou até mesmo algoritmos clássicos) ainda não descobertos capazes de quebrar facilmente a segurança de novos criptossistemas. Explicando de outra maneira, criptografia pós-quântica provavelmente é apenas uma medida provisória, parcial e temporária para lidar com a situação. Em contrapartida, a Distribuição de Chaves Quânticas (DCQ ou QKD) oferece uma solução

definitiva: restaurar a segurança e confidencialidade utilizando princípios da natureza, como o princípio da incerteza e a monogamia do emaranhamento.

2.3.1. Distribuição de Chaves Quânticas (DCQ ou QKD)

A DCQ é baseada em duas leis fundamentais da mecânica quântica: o teorema da não-clonagem e o princípio da incerteza de Heisenberg. De forma simplificada, o teorema da não clonagem afirma que não é possível replicar um estado quântico desconhecido de um fóton de maneira exata, enquanto o princípio da incerteza diz que é impossível medir a posição e a velocidade de uma partícula simultaneamente. Esses dois princípios sugerem que *qubits* não podem ser clonados e qualquer tentativa de replicação poderia ser notada por ambos destinatário e remetente (Alice e Bob, respectivamente) (SHARMA *et al.*, 2021).

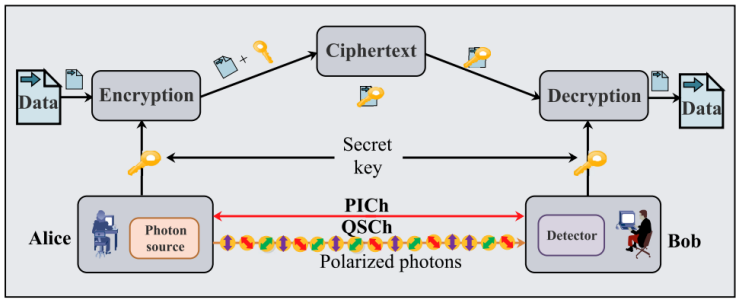
Um sistema simples de DCQ requer dois tipos de canais, um Canal de Sinal Quântico (QSCh) e um Canal de Interação Pública (PICh). Também são necessários blocos para cifragem/decifragem e um protocolo DCQ (SHARMA *et al.*, 2021).

Ainda segundo Sharma *et al.* (2021, p. 2053), as funcionalidades de cada elemento do sistema DCQ estão dispostas da seguinte maneira:

> - O QSCh será utilizado para mandar estados de luz quânticos (os fótons) entre os usuários (Alice e Bob, por exemplo).
> - O PICh será utilizado para enviar as bases de medição dos *qubits* e verificar as chaves secretas geradas e compartilhadas usando os métodos de pós-processamento.
> - Um protocolo de DCQ é utilizado para estabelecer uma conexão segura entre Alice e Bob. É o que gera as chaves secretas e analisa a quantidade de informação correta entre os usuários durante a geração das chaves.
> - Os blocos de cifragem/decifragem são necessários para cifrar as informações utilizando as chaves secretas e depois decifrá-las.

A figura 4 ilustra o funcionamento de um sistema básico de distribuição de chaves quânticas.

Figura 4 – Representação de um sistema DCQ simples



Fonte: (SHARMA et al, p.2054, 2021)

Protocolos quânticos possuem dois esquemas de *design*: o Preparar e Medir (P&M) e o Baseado em Emaranhamento (EB) (SHARMA *et al.*, 2021, p. 2053). Relevante para o presente artigo, somente o protocolo P&M será descrito.

O esquema P&M é baseado nos fundamentos da DCQ (princípio da incerteza e teorema da não-clonagem). Basicamente, Alice prepara as informações e depois as envia para Bob, que em seguida realiza a sua medição. A preparação que Alice realiza é nada mais que a polarização dos fótons. Um dos protocolos que utiliza o esquema P&M é o BB84 (SHARMA *et al.*, 2021).

2.3.2. Protocolo BB84

O protocolo BB84 é fundamentado nos princípios básicos da mecânica quântica e é provadamente seguro. Para a geração de fótons são utilizados pulsos de luz polarizada, onde há um único fóton para cada pulso de luz. O protocolo BB84 utiliza duas bases com dois estados de polarização cada, totalizando quatro estados de polarização. Sendo: Base retilínea (R) com os estados de polarização 0º e 90º; Base diagonal (D) com os estados 45º e 135º de polarização (SHARMA *et al.*, 2021).

O protocolo BB84 foi criado por Bennett e Brassard em 1984 e nomeado em homenagem a seus criadores, também foi o primeiro protocolo de distribuição de chaves quânticas desenvolvido. A primeira etapa do funcionamento do protocolo BB84 é realizada por um canal quântico e começa com a definição dos quatro possíveis estados para os *qubits*, para que os *bits* das chaves tanto de Alice quanto de Bob sejam codificados:

$$|\psi_0\rangle = |0\rangle,$$

$$|\psi_1\rangle = |1\rangle,$$

$$|\psi'_0\rangle = |+\rangle = \frac{1}{\sqrt{2}}(|0\rangle + |1\rangle),$$

$$|\psi'_1\rangle = |+\rangle = \frac{1}{\sqrt{2}}(|0\rangle - |1\rangle),$$

Onde $B = \{|\psi_0\rangle, |\psi_1\rangle\}$ e $B' = \{|\psi'_0\rangle, |\psi'_1\rangle\}$ são as bases. Os estados quânticos $|\psi_0\rangle$ e $|\psi'_0\rangle$ irão corresponder aos *bits* 0 da chave, e os estados $|\psi_1\rangle$ e $|\psi'_1\rangle$ aos *bits* 1. Após o estágio inicial, Alice gerará uma *string* de *bits* clássicos aleatórios, tendo quatro opções para polarização dos *qubits* e escolherá entre elas de maneira arbitrária para cada *bit*, enviando-a pra Bob logo em seguida. Assim que receber os *qubits* de Alice, Bob também irá escolher aleatoriamente entre

as bases *B* e *B'* e realizará uma medição para cada *qubit* recebido. Se as bases escolhidas por Bob e Alice forem iguais, os resultados irão se correlacionar perfeitamente, caso contrário, serão diferentes. É possível que ocorram erros na transmissão e/ou detecção o que acarretam Bob a não conseguir registrar nada (SHARMA *et al.*, 2021). Após a medição de todos os *qubits*, Bob cria uma *string* com os *bits* recebidos, a *chave bruta* (CHAVES, 2018).

É nesse momento que se inicia a fase clássica do processo. Através de um canal público Bob anunciará para Alice a base utilizada para a medição de cada *bit* de sua *chave bruta*. Alice irá comparar as bases envidas por Bob com as suas utilizadas no início do processo, os *bits* correspondentes a mesma base serão mantidos enquanto os correspondentes a bases diferentes descartados. O resultado dessa etapa é uma *string* chamada de *chave processada* (SHARMA *et al.*, 2021).

Por exemplo, Alice envia o estado $|\psi_0\rangle$ para Bob. Caso Bob escolha a base *B* para a medição, Alice o informará que a base está correta e o *bit* será aceito. Todavia, se Bob decidir medir o *qubit* de Alice na base *B'*, Alice perceberá que há uma possibilidade de erro, porque tanto o estado $|\psi_0\rangle$ quanto $|\psi_1\rangle$ poderão ser retornados com 50% de probabilidade. Ela então comunicará a Bob a incompatibilidade das bases e o *bit* será descartado. Teoricamente, se esse processo for realizado de maneira correta, a única possibilidade de haver um erro ocorre caso haja espionagem na comunicação, porém isso só é determinado após a comparação entre as chaves de Bob e Alice. Esse processo é ilustrado na tabela 1, onde Alice envia nove estados e somente quatro são aceitos, a chave gerada pelo algoritmo é 0010. (CHAVES, 2018)

A próxima etapa do processo é a estimativa de erro, aqui é realizada verificação de correspondência entre as chaves de Alice e Bob. Em um canal clássico, ambos selecionam *bits* de mesma posição em suas chaves e determinam um valor limiar de taxa de erro tolerável, pois canais quânticos podem apresentar ruído, o que interfere no processo de comunicação. Caso a taxa de erro exceda o valor estipulado pelos dois usuários, o processo é abortado e recomeçado (SHARMA *et al.*, 2021). Teoricamente essa é a etapa final do protocolo, porém existem duas etapas adicionais que ajudam a garantir mais segurança: correção ou reconciliação de erros e amplificação de privacidade.

A reconciliação de erros pode ocorrer através de diferentes métodos, usados para aumentar a capacidade de correção de erros em protocolos DCQ, essa etapa é realizada para

refinar ainda mais o processo de geração da *chave processada*, minimizando a chance de erro (SHARMA *et al.*, 2021).

A amplificação de privacidade gera uma chave secreta mais curta utilizando funções de *hash*, assim reduzindo a quantidade de informação da chave de uma forma negligenciável a algum usuário não autenticado. O produto é chamado de *chave secreta*. Ademais, um processo de autenticação é demandado para a garantia de segurança contra espionagem da chave secreta gerada (SHARMA *et al.*, 2021).

Tabela 1 – Exemplo de aplicação do protocolo BB84

Estado enviado por Alice	$\|\psi_0\rangle$	$\|\psi_1\rangle$	$\|\psi'_0\rangle$	$\|\psi'_0\rangle$	$\|\psi'_1\rangle$	$\|\psi'_1\rangle$	$\|\psi'_0\rangle$	$\|\psi_0\rangle$	$\|\psi_1\rangle$
Base usada por Bob	B	B'	B'	B	B	B'	B	B	B'
Compatibilidade entre as base	✓	✗	✓	✗	✗	✓	✗	✓	✗
Chave	0		0			1		0	

Fonte: (CHAVES, 2018)

2.4. Python

Python é uma linguagem de programação interativa orientada a objetos, que suporta outros paradigmas de programação como procedural e funcional. A linguagem possui uma sintaxe simples e incorpora exceções, módulos, classes, tipos de dados dinâmicos de alto nível e digitação dinâmica. É uma linguagem que funciona nas diversas variantes do sistema Unix, como Windows, macOS e Linux. Python também pode ser usada como uma extensão para programas que precisam de uma interface programável, além da própria linguagem ser extensível com C ou C++. (PYTHON, 2023)

A linguagem Python é utilizada em plataformas híbridas como intermediária entre a parte clássica e quântica da programação.

2.5. Qiskit

Segundo (QISKIT, 2023): “Qiskit é um software open-source usado para trabalhar com computadores quânticos a níveis de circuitos, pulsos e algoritmos. Complementarmente, muitas API’s de domínio específico existem sobre o núcleo do módulo”.

2.6. OpenQASM

Segundo Cross *et al.* (2017, p. 3): “QASM é uma linguagem de texto simples que descreve algoritmos quânticos”.

Cross *et al.* (2017, p.3) ainda elencam que a linguagem QASM possui elementos das linguagens clássicas C e Assembly. No código de um programa Open QASM, a primeira linha deve conter a sintaxe "OPENQASM M.m;" (M maiúsculo indica a versão atual e o m minúsculo em qual atualização, por exemplo: 2.0, 2.1 e etc.), que se trata da palavra-chave que define um arquivo OpenQASM. A versão utilizada é a 2.0. Sobre a sintaxe, a linguagem ignora espaços em branco e as linhas código são separadas por ponto e vírgula, além de ser *case sensitive*. Comentários são adicionados utilizando um par de barras (//) e terminam na linha seguinte.

2.7. VSCode

O Visual Studio Code, ou VSCode, é denominado como um poderoso editor de códigos que suporta as linguagens JavaScript, TypeScript e Node.js nativamente, porém possui uma vasta seleção de extensões para outras linguagens como Java, C#, C++, Python, PHP etc. O VSCode também é uma aplicação leve, o que significa que não demanda muitos recursos computacionais nem muito espaço em disco (MICROSOFT, 2023).

2.8. *IBM Quantum Experience*

Separada em IBM *Quantum Composer* e IBM *Quantum Lab*, a *Quantum Experience* é a plataforma de computação quântica e simuladores quânticos em nuvem da IBM. O IBM *Quantum Lab* foi utilizado para escrever o código.

3. Metodologia

A coleta de informações teóricas nesse projeto ocorreu através da revisão bibliográfica de artigos científicos, livros e documentação que abordem os temas relevantes ao projeto, como Computação Quântica, Criptografia Clássica e Quântica, Distribuição de Chaves Quânticas, Protocolos Quânticos, linguagens de programação QASM, Python etc.; o método de pesquisa utilizado foi a experimental, visando a coleta de resultados, dados, informações quantitativas, qualitativas, e pertinentes que agreguem aos tópicos escolhidos; o projeto foi desenvolvido na linguagem Python aliada a linguagem de circuitos quânticos OpenQASM, no simulador quântico da IBM, o IBM *Quantum Lab* (previamente parte do IBM *Quantum Experience*) e no VSCode juntamente com a biblioteca de simuladores quânticos também da IBM, o Qiskit, a fim de testar tanto um ambiente em nuvem como um ambiente local. Foram realizados 400 testes no total, 200 no ambiente local e 200 no ambiente em nuvem (Tabela 2). Os resultados do Projeto serão divulgados para a comunidade científica local e open-source.

4. Resultados e Discussões

Nessa seção do artigo realiza-se a análise de um código Python escrito pelos autores, que simula a distribuição de chaves quânticas através do protocolo quântico BB84. Esse protocolo utiliza quatro bases de polarização de fótons, duas retilíneas (0º e 90º) e duas diagonais (45º e 135º) para a encriptação de mensagens e faz a distribuição das chaves quânticas.

O código apresentado fornece uma perspectiva em terceira pessoa de como é realizada a DCQ através do protocolo BB84, quem protagoniza a troca de chaves são os personagens fictícios Alice e Bob. A troca de chaves é realizada de maneira automática, o que significa que o usuário que rodar o código observará dois indivíduos fictícios realizarem a troca de chaves quânticas entre si, determinada pelas linhas de código (desde o processo de polarização de cada bit da mensagem até a comparação das duas chaves).

Foram realizados 400 testes tanto no ambiente nuvem IBM *Quantum Lab* quanto no ambiente local VSCode, 200 em cada, onde as variáveis eram o *kernel* e a geração de arquivos. A figura 6 ilustra a disposição dos testes.

Após a execução da simulação, o código retorna *True* ou *False*, respectivamente, indicando se o procedimento funcionou ou não. Para o procedimento ocorrer com sucesso, é necessário que uma quantidade de bits verificadores seja comparada entre as duas chaves. O protocolo pode falhar diversas vezes devido a: erros quânticos, já que não estão sendo aplicados métodos específicos de correção de erros; a natureza do canal quântico, devido ao ruído quântico; o ambiente onde o código está sendo rodado, em nuvem ou local.

Os arquivos gerados são as chaves de Alice e de Bob com a extensão *pem* e a mensagem criptografada, tanto na extensão *pem* quanto na *qasm*.

Tabela 2 – Tabela de disposição dos testes

Número de Testes				
	Kernel "limpo", SEM geração de arquivos	Kernel "poluído", SEM geração de arquivos	Kernel "limpo", COM geração de arquivos	Kernel "poluído", COM geração de arquivos
Nuvem	50	50	50	50
Local	50	50	50	50

Fonte: Autores

Nos testes em nuvem, sem geração de arquivos, foram obtidos dois resultados *True* sem a limpeza do *kernel* (tentativas 17 e 31). Limpando o *kernel*, ainda sem a geração de arquivos, foi possível observar o resultado *True* cinco vezes (tentativas 6, 15, 20, 30 e 43).

Nos testes em nuvem, com geração de arquivos, foram obtidos cinco resultados *True* sem a limpeza do *kernel* (tentativas 24, 31, 39, 40 e 47). Limpando o *kernel*, com a geração de arquivos, foi possível observar o resultado *True* oito vezes (tentativas 3, 5, 15, 22, 24, 29, 35, e 50).

Nos testes locais, sem a limpeza de *kernel* e sem a geração de arquivos, foram obtidos dois resultados *True* (tentativas 10 e 12). Já com a limpeza, ainda sem a geração de arquivos, foi possível obter seis resultados *True* (tentativas 3, 5, 12, 16, 43 e 48).

Nos testes locais, sem a limpeza de *kernel* e com a geração de arquivos, foi obtido um resultado *True* (tentativa 27). Já com a limpeza, com a geração de arquivos, foi possível obter quatro resultados *True* (tentativas 17, 23, 40 e 45).

Vale ressaltar que o protocolo BB84 foi o primeiro protocolo de distribuição de chaves quânticas desenvolvido, portanto muitas de suas etapas foram utilizadas como base de outros protocolos de DCQ. O código desenvolvido se baseia em um exemplo de DCQ simples fornecido pelo Qiskit (QISKIT, 2023a) que utilizava apenas dois estados de polarização, um retilíneo (0º) e um diagonal (45º). Após a implementação das bases: retilínea 90º e diagonal 135º, foi possível simular o protocolo BB84 com suas etapas básicas, etapas adicionais como a reconciliação de erros e a amplificação de segurança não foram implementadas. Todo o processo está descrito na seção *2.3.2. Protocolo BB84* do presente artigo.

4.1 Análise do código

A seguir, será descrito como os trechos de código funcionam e a qual parte do protocolo BB84 estão relacionados, em um passo-a-passo para sua implementação.

I. Importação de bibliotecas necessárias: da biblioteca *qiskit*, são importados os módulos *QuantumCircuit*, o qual é responsável pela criação dos circuitos quânticos; da biblioteca *qiskit_aer* é importado o módulo do *AerSimulator*, o simulador quântico; da biblioteca *numpy*, é importado o módulo *randint* que é responsável pela aleatoriedade dos *bits* e bases que serão selecionados (Figura 5).

Figura 5 – Importações necessárias

```
from qiskit import QuantumCircuit
from qiskit_aer import AerSimulator
from numpy.random import randint
import numpy as np
```

Fonte: Autores

II. Depois, é definida a quantidade de *bits* que uma mensagem pode ter e atribuída a uma variável *n* (Figura 6):

Figura 6 – Quantidade de *bits* da mensagem

```
## Geração aleatoria e o tamanho da geração seria o n = 145
n = 145
```

Fonte: Autores

III. Aqui, é iniciado o protocolo BB84 (Figura 7). O primeiro passo é a seleção de *bits* aleatórios realizados por Alice: o método *randint* seleciona de maneira aleatória números entre 0 e 1 definidos pelo primeiro parâmetro (2), depois os armazena em uma variável de tamanho *n* (145, nesse caso).

Figura 7 – Seleção de *bits* aleatórios de Alice

```
## Passo 1
alice_bits = randint(2, size=n)
```

Fonte: Autores

IV. O próximo passo é a seleção das bases: o método *randint* seleciona de maneira aleatória qual dos 4 estados de polarização será utilizado para cada *bit* da mensagem, representado na figura 8.

V. Após a definição dos *bits* e bases, já é possível iniciar a codificação da mensagem, dada pela função *encode_message()* (figura 9):

Figura 8 – Geração de bases de Alice e parametrização do método *encode_message()* com os *bits* e bases de Alice:

```
## Passo 2
alice_bases = randint(4, size=n)
message_cod = encode_message(alice_bits, alice_bases)
```

Fonte: Autores

VI. A função *encode_message*() (Figura 9) utiliza os parâmetros *bits* e bases para definir qual polarização deverá ser aplicada a cada *qubit*, também gerando os circuitos quânticos para que isso seja possível. A função retorna a mensagem codificada.

Figura 9 – Função *encode_message*()

```
def encode_message(bits, bases):
    message_cod = []
    for i in range(n):
        qc = QuantumCircuit(1,1)
        if bases[i] == 0:  ## Preparação na polarização de 0º
            if bits[i] == 0:
                qc.x(0)
        elif bases[i] == 1:  ## Preparação na polarização de 45º
            if bits[i] == 0:
                qc.h(0)
            else:
                qc.x(0)
                qc.h(0)
        elif bases[i] == 2:  ## Preparação na polarização de 90º
            if bits[i] == 0:
                qc.h(0)
            else:
                qc.x(0)
                qc.h(0)
        elif bases[i] == 3:  ## Preparação na polarização de 135º ou -45º
            if bits[i] == 0:
                qc.h(0)
                qc.z(0)
            else:
                qc.x(0)
                qc.h(0)
                qc.sdg(0)
        qc.barrier()
        message_cod.append(qc)
    return message_cod
```

Fonte: Autores

VII. Bob prepara suas bases para poder realizar a medição dos *qubits* da mensagem de Alice. De forma similar a seleção aleatória de bases de Alice, o método *randint*() se encarrega de selecionar uma base de medição para cada *bit* da mensagem. Ver figura 10.

VIII. O simulador local é ativado através da variável *backend* e em seguida é exibido seu nome. Após isso a função de medição de *qubits* da mensagem é chamada e armazenada na variável *bob_results*, como mostra a figura 10.

Figura 10 - Seleção de bases de Bob

```
## Passo 3
bob_bases = randint(4, size=n)
backend = AerSimulator()
print('Seu dispositivo é o: {}' .format(backend.name))
bob_results = medicao_mensagem(message_cod, bob_bases, backend)
```

Fonte: Autores

IX. A função medição_mensagem(), ilustrada na figura 11, utiliza a mensagem codificada, as bases escolhidas de bob e o simulador quântico para montar os circuitos quânticos e medir os resultados dos *qubits* após a polarização selecionada de maneira aleatória, bem como transforma o *qubit* novamente em *bit* de acordo com o estado de polarização. Caso 0º ou 45º, o resultado será 0. Para 90º ou 135º, o resultado será 1. A função retorna as medições dos *qubits*.

Figura 11 – Função de medição dos *qubits* da mensagem

```
def medicao_mensagem(message_cod, bases, backend):
    measurements = []
    for q in range(n):
        if bases[q] == 0:  ## Preparação na polarização de 0º
          message_cod[q].x(0)
          message_cod[q].measure(0, 0)
        elif bases[q] == 1:  ## Preparação na polarização de 45º
            message_cod[q].x(0)
            message_cod[q].h(0)
            message_cod[q].measure(0, 0)
        elif bases[q] == 2:  ## Preparação na polarização de 90º
            message_cod[q].x(0)
            message_cod[q].h(0)
            message_cod[q].measure(0, 0)
        elif bases[q] == 3:  ## Preparação na polarização de 135º ou -45º
            message_cod[q].x(0)
            message_cod[q].h(0)
            message_cod[q].sdg(0)
            message_cod[q].measure(0, 0)
        result = backend.run(message_cod[q], shots=1, memory=True).result()
        measured_bit = int(result.get_memory()[0])
        measurements.append(measured_bit)
    return measurements
```

Fonte: Autores

X. O próximo passo é remover os *bits* divergentes da mensagem e deixar somente os *bits* considerados úteis através da função remover() ilustrada na figura 12, os "*bits* bons", armazenados no vetor *good_bits*. Para isso, são necessárias tanto as bases de Alice quanto de Bob, como também os *bits* de Alice e as medições de Bob. O produto do processo serão as chaves brutas de Alice e Bob, como ilustra figura 13.

Figura 12 – Função *remover*()

```
def remover(a_bases, b_bases, bits):
    good_bits = []
    for q in range(n):
        if a_bases[q] == b_bases[q]:
            good_bits.append(bits[q])
    return good_bits
```

Fonte: Autores

Figura 13 – Criação das chaves brutas através da função *remover*()

```
## Passo 4
chave_alice = remover(alice_bases, bob_bases, alice_bits)
chave_bob = remover(alice_bases, bob_bases, bob_results)
print()
print()
print("A chave de Alice: {}".format(chave_alice))
print("A chave de Bob:   {}".format(chave_bob))
print()
print()
```

Fonte: Autores

XI. As últimas etapas então se iniciam, é necessária uma quantidade de *bits* verificadores aleatórios para a comparação das chaves que validarão se o protocolo ocorreu com sucesso ou não. A quantidade de *bits* verificadores é definida pela variável amostra_bits_tamanho, nesse

caso 10. A seleção aleatória dos *bits* verificadores é realizada pelo método *randint*() e armazenado na variável selecao_bits. Por fim, a comparação de *bits* de Alice e Bob são definidas pelas variáveis comp_alice e comp_bob, através da função comp_bits (figura 15) que utiliza a chave bruta de Alice e Bob e a seleção aleatória de *bits* para realizar o procedimento. A figura 14 ilustra o processo de comparação de *bits*.

Figura 14 – Comparação de *bits*

```
## Passo 5
amostra_bits_tamanho = 10
selecao_bits = randint(n, size=amostra_bits_tamanho)
comp_bob = comp_bits(chave_bob,selecao_bits)
comp_alice = comp_bits(chave_alice, selecao_bits)
```

Fonte: Autores

Figura 15 – Função de comparação de *bits*

```
def comp_bits(bits, selection):
    amostra_bits = []
    for i in selection:
        i = np.remainder(i, len(bits))
        amostra_bits.append(bits.pop(i))
    return amostra_bits
```

Fonte: Autores

XII. Finalmente, são retornadas as chaves de Alice e Bob e se o protocolo funcionou ou não. Caso "*True*" houve sucesso. Caso "*False*", houve algum problema na execução, seja por conta de ruído ou fatores externos, portanto os *bits* devem ser descartados e o processo abortado e recomeçado. A figura 16 ilustra a verificação do protocolo, enquanto a figura 17 mostra as respectivas chaves com a validação *True* ou *False*.

Figura 16 – Validação do protocolo

```
## fazendo a verrificação do protocolo
verdade = comp_bob == comp_alice
print(verdade)
```

Fonte: Autores

Figura 17 – Chaves de Alice e Bob com o resultado da verificação do protocolo

```
Seu dispositivo é o: <bound method AerSimulator.name of AerSimulator('aer_simulator')>


A chave de Alice: [1, 1, 1, 1, 1, 1, 1, 1, 0, 0, 1, 1, 0, 1, 0, 1, 1, 0, 0, 0, 1, 0, 0, 1, 0, 1, 1, 0, 1, 1]
A chave de Bob:   [1, 1, 1, 1, 1, 1, 1, 1, 0, 0, 1, 1, 0, 1, 0, 1, 1, 0, 0, 1, 1, 0, 0, 1, 0, 1, 1, 1, 1, 1]


True
```

Fonte: Autores

5. Considerações Finais

Em conformidade com o objetivo do trabalho, os testes realizados transparecem a natureza do ambiente quântico que garante mais segurança devido a seus princípios, como o princípio da incerteza de Heisenberg e o teorema da não-clonagem, tendo vantagem significativa sobre sistemas de encriptação clássicos.

A partir da implementação do protocolo BB84 nos simuladores quânticos da IBM através do código Python e todos os testes realizados, foi possível observar que o sucesso do protocolo é bastante dependente do ambiente, ou seja, caso o canal quântico não tenha medidas de redução de ruído, a possibilidade de falha é alta. Observou-se ainda que quanto menor a quantidade de *bits*, maior é a chance de sucesso do protocolo, porém isso não significa que um protocolo com menos *bits* seja mais seguro. Uma grande quantidade de *bits*, apesar de possibilitar que erros aconteçam mais frequentemente, faz com que o protocolo sendo utilizado fique mais robusto e que seja mais difícil para um espião interceptar a comunicação sem ser notado. Também não há garantia que os canais quânticos utilizados nos testes possuam pouca quantidade de ruído, logo a alta quantidade de falhas era esperada.

Além de uma grande quantidade de *bits*, a utilização de quatro estados de polarização possibilita diversas outras formas de encriptação dos dados, pois são maiores as opções de polarização e medição dos *qubits*, dessa maneira garantindo mais segurança à operação.

Em uma situação real de troca de chaves quânticas, medidas preventivas de correção de erros quânticos e redução de ruído do canal são estabelecidas, portanto a alta taxa de não-correspondência entre as chaves e os *bits* verificadores de ambos os usuários deixa implícita uma tentativa de espionagem, já que os protocolos de correção garantem uma baixa quantidade de interferência. Porém, até o momento da conclusão deste artigo, nenhum canal quântico está livre de ruído, por esta razão há a necessidade de implementar uma quantidade máxima aceitável de erros nos protocolos quânticos.

Por utilizar leis da mecânica quântica, a distribuição de chaves quânticas através do protocolo BB84 mostra-se mais segura do que os criptossistemas clássicos, dadas as condições adequadas de ambiente, já que a interferência na comunicação causa uma medição implícita do *qubit*, ou seja, uma não-correspondência.

Como Pirandola *et al.* (2020, p. 92) discutem, apesar de ser uma tecnologia emergente, a DCQ é a mais madura das tecnologias quânticas e diversas medidas para aumentar sua segurança estão sendo criadas conforme os algoritmos se tornam mais robustos. Seus aspectos ficarão mais claros conforme a tecnologia quântica evolui e torna-se um produto mais comum. Como exemplo, recentemente (setembro de 2023) a instituição de educação SENAI de São Paulo adquiriu o primeiro computador quântico para fins educacionais, evidenciando que a acessibilidade ao novo paradigma computacional tende a crescer, possibilitando assim a difusão da criptografia quântica e os estudos da temática.

Referências

CHAVES, R. O. G. Táticas de ataque e protocolos quânticos de distribuição de chaves criptográficas utilizando estratégia de discriminação de estados. Dissertação (mestrado) – Universidade Federal de Minas Gerais – Departamento de Física. 2018.

CROSS, A. W. et al. Open quantum assembly language. arXiv preprint arXiv:1707.03429, 2017.

ESWARAN, S.; HONNAVALLI, P.; YALAMURI G. A Review of the Present Cryptographic Arsenal to Deal with Post-Quantum Threats, 2022. Disponível em: https://www.sciencedirect.com/science/article/pii/S1877050922021573 . Acesso em: 17 jun. 2023.

GUERRA, H.; TAVOLARO, C. R. C. Criptografia pós-quântica: protocolo dente de leão. Scientia Prima, v. 7, p. e111, 5 nov. 2021.

KOCKUM, A.F.; NORI, F. Quantum Bits with Josephson Junctions. In: Tafuri, F. (eds) Fundamentals and Frontiers of the Josephson Effect. Springer Series in Materials Science, vol 286. Springer, Cham, 2019. Disponível em: https://doi.org/10.1007/978-3-030-20726-7_17. Acesos em: 22 jun. 2023.

LACAVA, L.; MIANO, M. G. V. Implementação do algoritmo quântico Deustch-Josza em linguagem funcional e no simulador IBM Q Experience. Revista Tecnológica da Fatec Americana, Americana/SP, v. 6, n. 02, dez. 2018. Disponível em: https://fatec.edu.br/revista/index.php/RTecFatecAM/article/view/186. Acesso em 10 jun. 2023

MIANO, M. G. V. Aplicação de protocolos quânticos e algoritmo de Shor para a segurança da informação. Revista Tecnológica da Fatec Americana, Americana/SP, v. 8, n. 01, ago. 2020.

Disponível em: https://fatec.edu.br/revista/index.php/RTecFatecAM/article/view/233. Acesso em: 15 jun. 2023.

MICROSOFT. Visual Studio Code: Getting Started. 2023. Disponível em: https://code.visualstudio.com/docs. Acesso em: 20 mai. 2023.

NIELSEN, M. A; CHUANG, I. L. Quantum Computation and Quantum Information. Cambridge: Cambridge University Press, 2010. 698 p.

PIRANDOLA, S. et al. Advances in quantum cryptography. Advances in optics and photonics, v. 12, n. 4, p. 1012-1236, 2020.

PORTMANN, C.; RENNER, R. Security in quantum cryptography. Reviews of Modern Physics, v. 94, n. 2, p. 025008, 2022.

PYTHON. What is Python?. PYTHON, 2023. Disponível em: https://docs.python.org/3/faq/general.html#what-is-python. Acesso em: 19/05/2023.

QISKIT, T. D. Qiskit 0.43.0 documentation. 2023. Disponível em: https://qiskit.org/documentation/. Acesso em 07 mai. 2023.

QISKIT, T. D. Quantum key Distribuition. QISKITa. 2023. Disponível em: https://learn.qiskit.org/course/ch-algorithms/quantum-key-distribution. Acesso em 07 mai. 2023.

RESEARCH, Google. What is Colaboratory?. RESEARCH, 2023. Disponível em: https://research.google.com/colaboratory/faq.html. Acesso em: 20 mai. 2023.

SHARMA, P.; et al. Quantum key distribution secured optical networks: A survey. IEEE Open Journal of the Communications Society, v. 2, p. 2049-2083, 2021.