Algoritmo de Shor para quebra de criptografia RSA[1]

Evandro Fernandes de Uzeda [2]

Ricardo Vanni Dallasen[3]

RESUMO

O presente artigo busca explorar como a computação quântica tem a possibilidade de desestabilizar a segurança da informação, baseada em criptografia de chaves assíncronas. Deste modo, o mesmo discorre sobre os aspectos funcionais do algoritmo RSA (Rivest-Shamir-Adleman), onde serão vistos os passos necessários para gerar a chave pública e privada. Juntamente com o algoritmo de Shor, busca entender de que modo ele é capaz de encontrar os fatores primos de um produto, e suas etapas de funcionamento. Assim sendo, evidencia, na qualidade de experimento, o processo de quebra da criptografia RSA através da utilização da computação quântica. Com isso, foi realizada uma experiência capaz de efetuar a quebra de uma chave RSA. Porém, foram utilizadas chaves de valores pequenos, devido a limitação dos processadores quânticos atuais. Frente a quantidade de bits da criptografia atual, onde eles ainda não possuem qbits suficientes para executar o algoritmo de Shor em um número com tantos bits.

Palavras-chave: Computação Quântica, Criptografia. RSA, Algoritmo Shor

ABSTRACT

This article seeks to explore how quantum computing has the possibility to destabilize information security, based on asynchronous key cryptography. In this way, it discusses the functional aspects of the RSA (Rivest-Shamir-Adleman) algorithm, where the necessary steps to generate the public and private key will be seen. In the same way with Shor's algorithm, it seeks to understand how it is able to find the prime factors of a number, and its operating stages. Therefore, it shows, as an experiment, the process of breaking RSA encryption through the use of quantum computing. Thus, an experiment capable of breaking an RSA key was carried out. However, small-value keys were used, due to the limitation of current quantum processors. Faced with the amount of bits in current encryption, they still don't have enough qubits to run Shor's algorithm on a number with so many bits.

Keywords: Quantum Computation. Cryptography. RSA. Shor's Algorithm.

---

[1] Trabalho de Conclusão de Curso (TCC) apresentado ao Curso de Ciência da Computação do Instituto Federal Sul-rio-grandense, Câmpus Passo Fundo, como requisito parcial para a obtenção do título de Bacharel em Ciência da Computação, na cidade de Passo Fundo.

[2] Aluno do curso de Ciência da Computação, IFSul campus Passo Fundo.

[3] Professor do curso de Ciência da Computação, IFSul campus Passo Fundo.

1 INTRODUÇÃO

Atualmente grande parte das informações que são transferidas de um local para outro no âmbito digital utilizam criptografia para proteger os dados transferidos. Uma forma de criptografia que é empregada para comunicação é o RSA (Rivest-Shamir-Adleman), que utiliza chaves assimétricas para encriptar e desencriptar uma mensagem. Sendo assim, de posse da chave pública é possível encriptar a mensagem, e a mesma somente será desencriptada por quem possuir a chave privada.

O algoritmo RSA é o recomendado para ser usado em aplicações do tipo cliente-servidor, já que a chave pública pode ser enviada aos clientes e somente o servidor terá a chave privada para desencriptar as mensagens. Junto com a chave pública é enviado o produto dos coeficientes, que é a base para gerar tanto a chave pública como a chave privada. Isso não é necessariamente um problema, uma vez que para gerar as chaves ainda seria necessário saber quais são os fatores que concebem o produto. Entretanto, com um número suficientemente grande de tentativas, um computador clássico poderia um dia encontrar esses coeficientes e gerar a chave privada. Este trabalho aborda o entendimento do algoritmo de criptografia RSA juntamente com o algoritmo de Shor, que é utilizado para encontrar os fatores de um produto e neste caso, poder gerar a chave privada. Utilizando a computação quântica, é possível melhorar o tempo de execução do algoritmo de Shor, fazendo com que a obtenção do resultado seja conseguida em tempo polinomial.

O trabalho está organizado da seguinte forma: inicialmente é realizada uma breve introdução sobre computação quântica, Em seguida é apresentado como é o funcionamento da criptografia RSA e também o funcionamento do Algoritmo de Shor, que é capaz de melhorar o processo de adivinhação do coeficiente.

2 INTRODUÇÃO A COMPUTAÇÃO QUÂNTICA

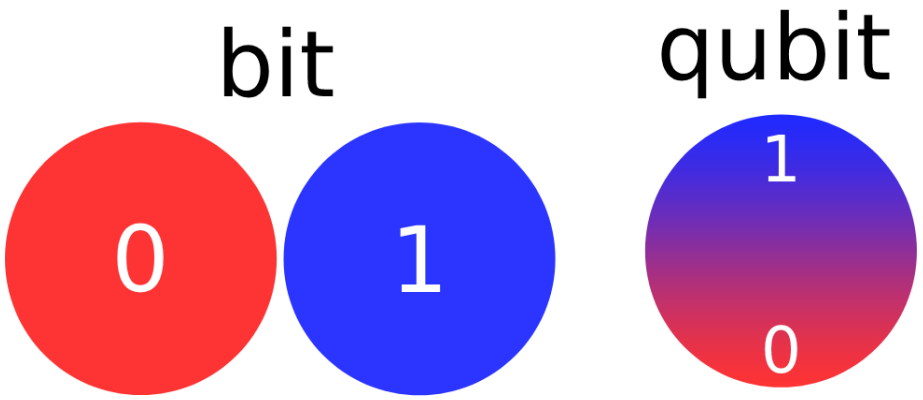
Na computação clássica, que remete a Turing e Von Neumann, as informações são reduzidas a bits e são capazes de assumir os valores binários de

“0” e “1”. Da mesma maneira, o processamento pode ser realizado por funções lógicas como AND ou OR, entre outras. Assim, o estado de um computador clássico é inteiramente determinado pelos estados de todos os seus bits, de modo que um computador com bits pode existir em um dos estados possíveis, variando de 00...0 a 11...1 (IBM, 2021).

Atualmente os computadores utilizam tensão elétrica para a representação da informação em formato digital. Para a realização de cálculos e operações lógicas são utilizados circuitos construídos com transistores que realizam chaveamento para realizar essas funções lógicas. O desempenho de um sistema computacional está ligado diretamente ao tamanho do transistor, quanto menor o transistor mais rápido ocorre a mudança de “0” para “1” e vice-versa para realizar as operações lógicas, e também menor o consumo de energia. No entanto, esse encolhimento do tamanho está chegando ao limite físico do silício, as estruturas dos transistores estão chegando em tamanhos mensuráveis em números de átomos.

Na computação quântica também é possível representar “0” e “1” com bits quânticos, ou qubits. O diferencial é a possibilidade da combinação linear de ambos, que é a propriedade denominada superposição. A superposição do computador quântico permite muitos estados lógicos simultaneamente, como mostra Figura 1, onde o qubit pode estar proporcionalmente nos dois estados.

Figura 1: Comparação entre bit e qubit



Fonte: do autor.

Segundo a IBM, a grande maioria das superposições quânticas, e as mais úteis para a computação quântica, estão emaranhadas. Estados emaranhados são estados que não correspondem a nenhuma atribuição de estados digitais ou analógicos, isso para cada qubits individualmente. Assim, o computador quântico é, portanto, significativamente mais poderoso do que qualquer computador clássico - seja ele determinístico, probabilístico ou analógico (BENNETTI, 2021).

3 FUNCIONAMENTO DO ALGORITMO RSA

A segurança da criptografia RSA se deve principalmente devido à dificuldade de se fatorar um grande número, para encontrar seus coeficientes que são dois números primos (MILANOV, 2009). O produto obtido pela multiplicação entre dois números primos é utilizado para gerar a chave pública e a privada. Com isso, a multiplicação entre dois números primos é fácil de ser calculada. No entanto, tendo o produto, saber quais são os multiplicadores é uma tarefa bastante dispendiosa. Dessa forma o RSA consegue manter seguro os números primos que são as bases das chaves.

Para encriptar um conteúdo é necessário possuir a chave pública, e para desencriptar o conteúdo é preciso dispor da chave privada. Com o objetivo de gerar tanto a chave pública como a chave privada, é preciso concluir os processos de geração de números primos e a função frequente de Carmichael.

3.1 GERAÇÃO DE NÚMEROS PRIMOS

A fim de gerar as chaves, é necessário obter dois números primos que sejam distantes um do outro e de grande valor. Deste modo, os números podem ser escolhidos utilizando o teste de primalidade, como o algoritmo de Rabin–Miller. Após encontrados os dois números primos, é possível calcular o produto entre eles. O produto será utilizado no processo de encriptação da mensagem.

3.2 FUNÇÃO FREQUENTE DE CARMICHAEL

A função de Carmichael representa o produto entre os dois números primos, como sendo o valor do menor múltiplo comum (MMC), com o decréscimo de um em cada número primo. Dado pela expressão $\lambda(n) = MMC((p - 1), (q - 1))$, onde n é o produto, p e q são os números primos respectivamente. O resultado da função será utilizado para realizar a desencriptação da mensagem. De igual modo, pode ser usado simplesmente o produto da multiplicação como $\lambda(n) = (p - 1) * (q - 1)$ (MILANOV, 2009).

3.3 CONCEPÇÃO DA CHAVE PÚBLICA

De posse de $\lambda(n)$, a chave pública é concebida com a escolha de um número primo, chamado de e, que seja maior que 1 e co-primo de $\lambda(n)$. À vista disso, é possível iniciar o processo de encriptação da mensagem.

A encriptação da mensagem deriva da aplicação da fórmula $c = m^e \bmod n$, onde m é a mensagem, n o produto, e a chave pública e c o *ciphertext*. O *ciphertext* é a mensagem criptografada, que pelo conceito do RSA, somente será descriptografada pela chave privada.

3.4 CONCEPÇÃO DA CHAVE PRIVADA

A fim de descriptografar a mensagem, é necessário saber o valor de d, que será a chave privada, sendo d o inverso de e tal como co-primo de $\lambda(n)$. Com isso, o último passo consistirá em utilizar a fórmula $m = c^d \bmod n$. Sendo assim, o resultado m representará a mensagem originalmente encriptada.

4 FUNCIONAMENTO DO ALGORITMO DE SHOR

Em 1994 Peter Shor propôs um algoritmo quântico capaz de solucionar o problema para encontrar os fatores de um produto. Por esse motivo o algoritmo é significante, já que seria possível quebrar uma chave pública. Como visto na seção anterior, o RSA possui um produto de dois números primos grandes, que por sua vez geram tanto chave pública como privada. O algoritmo de Shor foi demonstrado

em 2001 por um grupo da IBM, onde o número 15 foi fatorado em 3 e 5, utilizando um computador quântico com 7 qubits (QUANTIKI, 2015).

O problema que Shor foca em resolver é: dado um inteiro *N* é preciso encontrar outro inteiro *p* estando entre 1 e *N*, que divide *N*. Com isso, o algoritmo de Shor pode ser dividido em duas partes. A primeira parte consiste em reduzir o problema de fatoração ao problema de encontrar o período de uma função, o que pode ser feito por um computador clássico. Na segunda parte é solucionado o problema para encontrar o período de uma função. Essa parte é realizada em um computador quântico.

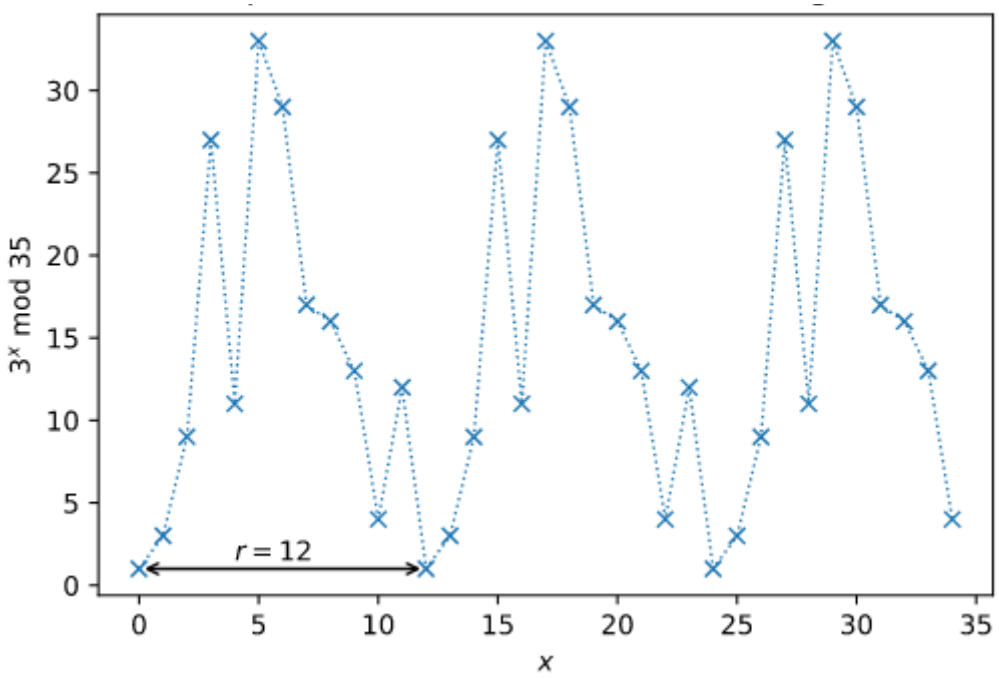
4.1 PRIMEIRA PARTE

Para iniciar o processo de redução é necessário escolher um número *a* que seja menor que o produto *N*. Em seguida, utilizando o algoritmo Euclidiano, calcula-se o maior divisor comum (MDC) entre *a* e *N*. Assim, se o resultado do MDC for diferente de 1, então encerra por aqui. No entanto, caso contrário, é preciso usar a subrotina para encontrar o período de uma função, desta forma encontrando r. Sendo a função, $f(x) = a^x \bmod N$, e *r* sendo o inteiro onde $f(x+r) = f(x)$.

4.2 SEGUNDA PARTE

Agora é necessário encontrar *r*, que é o menor inteiro, sem ser zero, demonstrado por $a^r \bmod N = 1$. Na Figura 2 é mostrado o gráfico de uma função periódica no algoritmo de Shor, com *N* de 35, *a* igual a 3 e *r* = 12.

Figura 2: Exemplo da função periódica no algoritmo de Shor



Fonte: Qiskit.

Utilizando a Estimativa de Fase Quântica, iniciando o estado em 1 temos que a cada aplicação consecutiva da função periódica, o estado será multiplicado por *a (mod N)*, e depois de *r* aplicações o estado retornará à 1. Isso pode ser visto na Figura 1. Até esse ponto seria possível usar um computador clássico para fazer um loop para encontrar r. No entanto, a superposição da computação quântica permite aplicar todas as funções periódicas de uma vez. Só estaria limitado pela quantidade de qubits disponível no processador para registrar os estados. Deste modo, usa-se a transformação quântica de Fourier para encontrar a frequência. Assim, para enfim obter os fatores de *N* é necessário calcular o maior divisor comum de $a^{r/2} \pm 1$ em relação a *N*.

5 METODOLOGIA

Após o entendimento da base de funcionamento das duas áreas de pesquisa, é preciso exercer a teoria aprendida. Para tal, foi utilizado o SDK (Software Development Kit) fornecido pela Qiskit que usa a linguagem Python como base. Assim, foi implementado um programa que consiste na implementação do algoritmo

RSA, que gera as chaves. Juntamente com a implementação do algoritmo de Shor para encontrar os fatores primos da multiplicação. Deste modo, após encontrados os fatores, gerar a chave privada a partir da chave pública, dessa forma descriptografando a mensagem.

Para a realização da implementação do algoritmo RSA, foi usado como base, a implementação de Jane Chan, disponível no GitHub (CHEN, 2017). A fim de melhorar o entendimento, foram utilizados os conceitos do *clean architecture* (MARTIN, 2017). Deste modo, o código foi dividido em duas entidades e em casos de uso. A implementação será detalhada na seção 6.1 Código do RSA.

O SDK do Qistik disponibiliza uma implementação do algoritmo de Shor, pronta para ser executada nos simuladores ou nos computadores quânticos. Para o experimento foi utilizada a classe Shor do SDK, que recebe uma instância de um computador quântico, seja simulado ou real. Para executar é preciso chamar o método factor e passando o *n* como parâmetro junto de um inteiro *a*. O retorno do factor é um objeto que contém os fatores e outras informações de execução do algoritmo. Com isso, é possível usar os fatores para gerar a chave privada usando a chave pública.

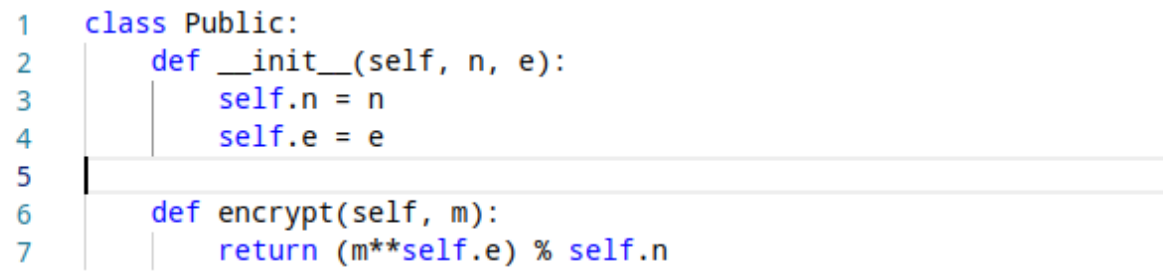
6 RESULTADOS

Nessa seção serão mostrados os códigos da implementação do RSA, juntamente com a execução do programa, para mostrar quais foram os resultados obtidos com o experimento.

6.1 CÓDIGO DO RSA

O código do RSA possui duas entidades que representam as chaves, e os casos de uso que criarão as chaves. Assim, a classe Public contém os atributos *n*, que é o produto dos primos, juntamente com *e*, que é um número maior que 1 e co-primo de *t*. Public possui o método encrypt que recebe a mensagem e retorna *c*, a mensagem encriptada. A Figura 3 mostra a implementação.

Figura 3: Implementação em Python do Public



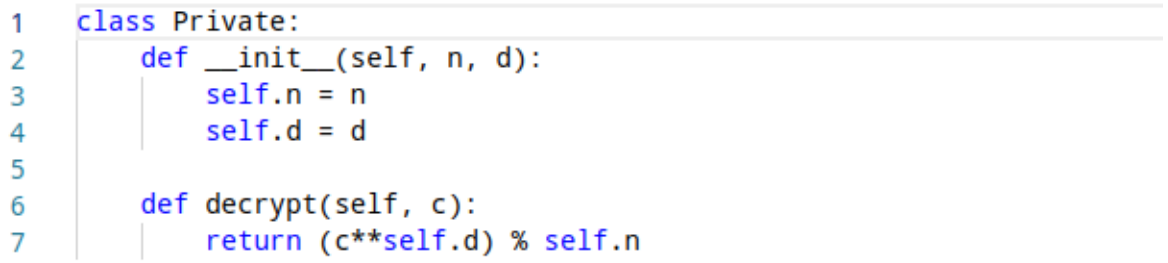
```
class Public:
    def __init__(self, n, e):
        self.n = n
        self.e = e

    def encrypt(self, m):
        return (m**self.e) % self.n
```

Fonte: do autor.

Na Figura 4 mostra que a classe Private também possui *n*, porém contém *d*, que é o inverso de *e* bem como co-primo de t. Private possui o método decrypt que recebe *c* e retorna a mensagem descriptografada.

Figura 4: Implementação em Python do Private



```
class Private:
    def __init__(self, n, d):
        self.n = n
        self.d = d

    def decrypt(self, c):
        return (c**self.d) % self.n
```

Fonte: do autor.

6.1.1 CASOS DE USO

Os casos de uso representam os cenários de criação das chaves, gerando os números primos ou recebendo-os por parâmetro, e geração da chave privada com base na chave pública. A classe CreateKeys, mostrada na Figura 5, possui um método *execute*, no qual recebe um valor mínimo e um máximo, representando um intervalo. Os números primos estarão dentro do intervalo, porém o segundo estará em um intervalo acima do primeiro. Depois de escolhidos, são calculados *n* e *t*, sendo *t* empregue para encontrar *e* tal como *d*. Para encontrar *e* é chamada a função getE com o parâmetro *t*, onde é executado um *loop* para obter o maior divisor comum (MDC) diferente de 1, entre *i* > *1* e *t*. Já na função getD, os parâmetros são *e*

e *t*, onde é executado o algoritmo Euclidiano estendido, que retornará o MDC, *a* e *b*, onde *a*, caso positivo será *d*, e caso negativo, será somado *t* para obter *d*.

Figura 5: Implementação da classe CreateKeys

```
61  class CreateKeys:
62      def __init__(self) -> None:
63          pass
64
65      def execute(min, max):
66          p = getPrime(min, max)
67          q = getPrime(p+min,p+max)
68          print("Prime p: ", p)
69          print("Prime q: ", q)
70          n = p * q
71          t = (p-1)*(q-1)
72          e = getE(t)
73          d = getD(e, t)
74
75          return Public(n, e), Private(n, d)
```

Fonte: do autor.

A classe PreSetPrimes é semelhante à anterior, no entanto ao invés de gerar os número primos recebe-os por parâmetro *p* e *q*. Assim, de igual modo retorna as chaves públicas e privadas.

Já o caso de uso GeneratePrivateFromPublic, no método execute os parâmetros são *p*, *q* e a chave pública. Nele é calculado *t*, para em seguida adquirir *d*, através da função getD, usando o *e* da chave pública. Deste modo, é retornada a chave privada usando o *n* da chave pública. Figura 6 mostra a implementação.

Figura 6: Implementação do GeneratePrivateFromPublic

```
6   class GeneratePrivateFromPublic:
7       def __init__(self) -> None:
8           pass
9
10      def execute(p, q, public: Public):
11          t = (p-1)*(q-1)
12          d = getD(public.e, t)
13
14          return Private(public.n, d)
```

Fonte: do autor.

6.2 EXECUÇÃO

Como mostrado na Figura 7, o programa main.py inicia atribuindo um valor inteiro à mensagem, em seguida são geradas as chaves para poder criptografar a mensagem. Depois, é criado um backend, que nesse caso será na máquina local, para conter a instância do simulador do computador quântico. Após, é instanciada a classe Shor, passando a instância, e executado o factor sobre o produto. Assim, tendo os fatores é executado o caso de uso GeneratePriveFromPublic, para então descriptografar e imprimir a mensagem. E obtém-se o resultado mostrado na Figura 8.

Figura 7: Código do arquivo main.py

```
10  msg = 9
11  public, private = PreSetPrimes.execute(3, 7)
12  c = public.encrypt(msg)
13
14  backend = Aer.get_backend('qasm_simulator')
15  quantum_instance = QuantumInstance(backend, shots=1000)
16  my_shor = Shor(quantum_instance=quantum_instance)
17  result = my_shor.factor(N=public.n, a=2)
18  factors = result._factors[0]
19
20  gPrivate = GeneratePrivateFromPublic.execute(factors[0], factors[1], public)
21  m = gPrivate.decrypt(c)
22  print(m)
```

Fonte: do autor.

Figura 8: Saída da execução do main.py

```
(tcc) root@0095cf86f447:~/TCC# /opt/conda/envs/tcc/bin/python /root/TCC/Quantum/main.py
Prime p:  3
Prime q:  7
original d:  5
ciphertext:  18
factors:  [3, 7]
generated d:  5
message:  9
(tcc) root@0095cf86f447:~/TCC#
```

Fonte: do autor.

Os testes foram executados no simulador em máquina local, por esse motivo a escolha dos primos de valor baixo, no caso 3 e 7. Os testes não puderam ser executados em computadores quânticos, pois as máquinas liberadas possuem apenas 5 qubits e para encontrar os fatores de 21 são necessários 22 qubits.

Contudo, foi possível utilizar os simuladores da IBM que possuem mais qbits. Deste modo, com os primos 5 e 23, n de 115, necessitando 30 qbits, conforme mostrado na Figura 9. Foi obtido o resultado esperado, porém sem a performance do Computador Quântico.

Figura 9: Saída da execução no simulador da IBM

```
(tcc) root@0095cf86f447:~/TCC# /opt/conda/envs/tcc/bin/python /root/TCC/Quantum/main.py
Prime p:  5
Prime q:  23
original d:  59
ciphertext:  39
n:  115
bit length:  7
qbits:  30
factors:  [5, 23]
generated d:  59
message:  9
(tcc) root@0095cf86f447:~/TCC#
```

Fonte: do autor.

7 CONCLUSÃO

No presente artigo foi apresentado o funcionamento da criptografia RSA e como o algoritmo de Shor tem a capacidade de encontrar os fatores primos de um produto. Deste modo, a parte prática exemplificou com chaves de poucos bits (devido a limitação de qubits disponível para utilização) que o resultado obtido está de acordo com o esperado. Dada a limitação de uso do computador quântico, os testes foram executados apenas em simulação, gerando resultados de forma determinística. Os resultados gerados a partir de um computador quântico real são probabilísticos e sujeitos a ruídos devido a sua característica de funcionamento. Sendo assim, não é possível afirmar que em um computador quântico seria obtido exatamente os mesmos resultados.

Outro ponto importante a ser analisado é que atualmente as chaves RSA usadas por exemplo em certificados SSL, possuem entre 2048 e 4096 bits. Assim, utilizando o SDK da Qiskit para gerar o circuito onde calcula a quantidade de qubits necessários, uma chave de 4096 bits necessita 16386 qubits. E segundo Matthew, o computador quântico mais avançado é o da IBM, com 127 qubits. Isso demonstra que, utilizando esse método, ainda estamos longe de quebrar as chaves amplamente utilizadas.

Com isso, fica para trabalhos futuros pesquisar alternativas para o algoritmo RSA. Um exemplo é a criptografia baseada no caos que pode ajudar a evitar o Apocalipse Quântico (RITSUMEIKAN, 2022). Haja visto que, em algum momento, os computadores quânticos terão qubits suficientes para quebrar criptografias baseadas no algoritmo RSA.

REFERÊNCIAS

QUANTIKI. Disponível em: <https://www.quantiki.org/wiki/shors-factoring-algorithm>. Acesso em: 07 out. 2021.

QISKIT. Disponível em: <https://qiskit.org/textbook/ch-algorithms/shor.html>. Acesso em: 07 out. 2021.

QISKIT. Disponível em: <https://learn.qiskit.org/course/introduction/why-quantum-computing>. Acesso em: 07 out. 2021.

IBM. Disponível em: <https://quantum-computing.ibm.com/composer/docs/iqx/guide/shors-algorithm>. Acesso em: 07 out. 2021.

BENNETT, Charlie. Learn quantum computing: a field guide. Disponível em: <https://quantum-computing.ibm.com/composer/docs/iqx/guide/>. Acesso em: 07 out. 2021.

SHOR, Peter W.. Polynomial-Time Algorithms for Prime Factorization and Discrete Logarithms on a Quantum Computer. Nov., 1994. Disponível em: <https://arxiv.org/pdf/quant-ph/9508027.pdf>. Acesso em: 07 out. 2021.

CHEN, Jane. RSA Implementation in Python, Abril 2017. Disponível em: <https://github.com/jchen2186/rsa-implementation>. Acesso em: 28 fev. 2022.

MILANOV, Evgeny. The RSA Algorithm. jun. 2009.

SPARKES, Matthew. IBM creates largest ever superconducting quantum computer. Disponível em: <https://www.newscientist.com/article/2297583-ibm-creates-largest-ever-superconducting-quantum-computer/>. Acesso em: 01 mar. 2022.

MARTIN, Robert C.. Clean Architecture: A Craftsman's Guide to Software Structure and Design. 1ª Edição, 2017.

RITSUMEIKAN. Criptografia baseada no caos pode evitar Apocalipse Quântico Disponível em: <https://www.inovacaotecnologica.com.br/noticias/noticia.php?artigo=criptografia-baseada-caos-evitar-apocalipse-quantico>. Acesso em: 23 mar. 2022.