Geração Caótica de Chaves para QKD

Queiroz, R. S., Y. Martínez-Camejo, G. F. Guimarães, J. S. de Andrade e Gouveia, Daniel Xavier

Resumo— **Neste trabalho uma proposta computacional empírica é apresentada para a geração de chaves de criptografia para distribuição quântica de chaves (QKD). O método proposto combina geradores de números pseudoaleatórios, mapa caótico e teste de NIST 800-22 para certificação de aleatoriedade. A taxa de geração de sequências aleatórias obtidas no sistema proposto não é diferente dos obtidos de geradores reais de números aleatórios ou dos pseudoaleatórios, porém duas camadas de segurança acrescentadas dificultam a ação de espionagem em sistemas QKD que usam detectores de fótons únicos como APDs.**

Palavras-Chave— QKD, testes de NIST, aleatoriedade, mapa caótico, criptografia.

***Abstract*— In this work an empirical computational proposal is presented for the generation of encryption keys for quantum key distribution (QKD). The proposed method combines pseudorandom number generators, chaotic map and NIST 800-22 test for randomness certification. The generation rate of random sequences obtained in the proposed system is not different from those obtained from real random number generators or pseudorandom ones, but two added security layers make it difficult to spy on QKD systems that use single photon detectors as APDs.**

Keywords— QKD, NIST tests, randomness, chaotic map, cryptography.

I. INTRODUÇÃO

Na era pós-quântica, a necessidade de criptografia com maior segurança está aumentando [1, 2]. Os geradores de números aleatórios desempenham um papel crucial na segurança da criptografia, onde a aleatoriedade é essencial para garantir a confidencialidade e integridade dos dados. O caos pode contribuir com acrescimo de segurança em sistemas aleatórios pois gera um comportamento dinâmico complexo com aperiodicidade, transitividade de fase e imprevisibilidade de longo prazo.

A criptografia depende da aleatoriedade para criar chaves seguras, chaves previsíveis podem ser quebradas por criptoanálise [4]. Há outras aplicações relacionadas a criptografia, que fazem uso de geradores de números aleatórios, porém neste trabalho o foco é a preparação para QKD.

Nas últimas décadas, tem havido um constante aumento de estudos de estimativas para quebrar criptografias primitivas baseadas em produtos de grandes números primos, usando computadores quânticos [3]. O algoritmo quântico de Shor [4] abriu as portas para resolver fatoração de números primos com solução em tempo exponencial intratáveis em computadores clássicos, mostrando ser possível resolvê-los em tempo polinomial ($O(n^3)$).

Quanto a criptografia de chave pública, cerca 6n qubits seriam necessários para quebrar a criptografia de curva elíptica, onde n é o número de bits do grupo, assim como seriam necessários 6*255 = 1530 qubits lógicos para quebrar ECC em uma curva 255 bits (por exemplo, Curve25519, Ed25519) [5]. Otimizações em projetos de circuitos quânticos estão sendo feitas para reduzir a complexidade desse ataque [6], enquanto que em um estudo recente [7], o qubit lógico necessário para fatorar um número de n bits é mostrado como $3n + 0.002n*lg\ n$. De forma razoável supõe-se, considerando a taxa de erro de porta, tempo de ciclo e outros parâmetros, a hipótese de que um inteiro RSA de 2048 bits possa ser fatorado em 8 horas, se 20 milhões de qubits físicos estiverem disponíveis [3].

Comparados à criptografia de chave pública, os algoritmos criptográficos de chave privada permaneceram mais resilientes a um adversário quântico. A abordagem padrão tem sido, primeiro, aplicar o algoritmo de busca de Grover para força bruta da chave secreta, o que dá uma aceleração quadrática. Isso foi melhorado em [8]. Ataques *meet-in-the-middle*, levados do cenário clássico para o quântico, são estudados para AES [9], [10]. Um ingrediente central dos ataques quânticos em cifras de chave simétrica é um circuito quântico eficiente para a própria cifra, uma vez que a operação de criptografia é chamada várias vezes durante o ataque. Consequentemente, há um crescimento recente de estudos sobre circuitos quânticos eficientes para cifras de chave simétrica [11]. Apesar desses avanços, a complexidade do ataque quântico permaneceu na ordem exponencial para cifras de chave privada.

Distribuição quântica de chaves (QKD) é uma abordagem alternativa para troca de chaves, que funciona aproveitando as propriedades fundamentais da mecânica quântica [12]. QKD oferece segurança teórica da informação (ITS) e, portanto, incondicionalmente segura contra um invasor com poder computacional ilimitado. Isso está em forte contraste com os esquemas criptográficos tradicionais, que são apenas condicionalmente seguros contra um invasor com capacidades computacionais bem definidas e finitas. Assim, QKD também serve como uma alternativa natural para troca de chaves contra um adversário quântico. Um ponto importante a ser observado é que não é necessário um computador quântico para implementar um sistema QKD [3].

Inicialmente faremos uma revisão bibliográfica sobre as sequências de números aleatórios e os seus geradores, como são construídos os tipos clássicos e quânticos e quais as ferramentas usadas para testar essas sequências. Também será revisado a teoria do caos e suas propriedades aproveitáveis como fonte de criptografia. Em seguida, serão apresentados os passos adotados como proposta para obter sequências binárias, codificá-las com

Renata Silva de Queiroz, Programa de Pós-Graduação em Engenharia de Telecomunicações, IFCE, Fortaleza-CE, e-mail: renata.squeiroz8@gmail.com; Yosdan Martinez Camejo, Programa de Pós-Graduação em Engenharia de Telecomunicações, IFCE, Fortaleza-CE, e-mail: yosdan@ppget.ifce.edu.br; Glendo de Freitas Guimarães, Programa de Pós-Graduação em Engenharia de Telecomunicações, IFCE, Fortaleza-CE, e-mail: glendo@fotonica.ifce.edu.br; Joacir Soares de Andrade, Programa de Pós-Graduação em Engenharia de Telecomunicações, IFCE, Fortaleza-CE, e-mail: joacirsoares@yahoo.com; Daniel Xavier Gouveia, Programa de Pós-Graduação em Engenharia de Telecomunicações, IFCE, Fortaleza-CE, e-mail: dxgouveia@gmail.com; Este trabalho foi parcialmente financiado pela Fundação Coordenação de aperfeiçoamento de Pessoal de Nível Superior – CAPES(88887.950067/2024-00).

mapa logístico e testar sua aleatoriedade para uso em QKD. Por fim, resultados de eficiência de chaves geradas pelo método proposto são comparados com outros geradores, evidenciando a equivalência com os mesmos, porém destacando os ganhos reais de segurança no método proposto, acrescentada no processo de geração das chaves e na posterior distribuição das mesmas em canal óptico por modulação de amplitude e detecção por fotodiodo de avalanche (APD de Avalanche Photodiode), abordagem comum em QKD.

II. GERADORES DE CHAVES

A criptografia é uma técnica para converter informações em um formato ilegível e tem bases no desenvolvimento e aprimoramento de algoritmos que conseguem cifrar a informação de forma eficiente e viável computacionalmente. De forma simplificada, o objetivo da criptografia é transformar um texto original, chamado de texto claro, em um texto cifrado utilizando uma chave. Tal transformação também deve prever a recuperação da informação original, isto é, de posse do texto cifrado deve ser possível obter o texto claro utilizando também uma chave. Portanto, é essencial gerar chaves com qualidade aleatória testada para garantir a segurança dos sistemas criptográficos.

Os geradores de números aleatórios (RNG de Random Number Generators) estão divididos em três categorias principais: os geradores clássicos, baseados em fontes físicas como ruído eletrônico, decaimento nuclear, movimento browniano e outros; os geradores pseudoaleatórios (PRNG de Pseudo-Random Numbers Generators) são baseados em algoritmos, como por exemplo Mersenne Twister, Java Script Generate, Blum Blum Shub e outros; e geradores quânticos, baseados em fenômenos quânticos como polarização da luz, efeito fotoelétrico, flutuação do vácuo e outros.

Os RNGs usam fonte não determinística, ou seja, a fonte de entropia, acompanhado de algumas funções de processamento, isto é, o processo de destilação da entropia para produzir aleatoriedade. As saídas desse tipo de gerador podem ser usadas diretamente como um número aleatório, nesse caso a saída precisa satisfazer critérios rigorosos de aleatoriedade ou pode servir como entrada para geradores de números pseudoaleatórios. Geradores de números pseudoaleatórios usam uma ou mais entradas e geram múltiplos "pseudo" números, sendo que as entradas para PRNGs são chamadas de sementes. Em contextos em que a imprevisibilidade é necessária, a própria semente deve ser aleatória e imprevisível, assim, por padrão, um PRNG deve obter suas sementes a partir das saídas de um RNG [13].

Uma vez que as fontes verdadeiramente aleatórias se tornam muitas vezes inviáveis em uma aplicação, comumente se admite o uso de geradores de números pseudoaleatórios para gerar os números que tem características de números aleatórios[14].

III. CAOTICIDADE

A teoria do caos estuda os sistemas não lineares e determinísticos que são altamente sensíveis às condições iniciais, isto é, quando até a menor perturbação no sistema pode levar a resultados divergentes, tornando-os imprevisíveis e até aparentemente aleatórios[15]. Um exemplo emblemático de sistemas caóticos é o mapa logístico, representando os sistemas de tempo discreto, que utilizaremos como fonte de aperiodicidade na 2ª etapa da distribuição de chaves. O mapa logístico é uma equação de segundo grau que apresenta comportamento caótico, mesmo para uma equação não-linear tão simples como equação 1

$$X_{a+1} = \lambda X_a(1-X_a) \quad (1)$$

Esse mapa é parte de um modelo discreto de crescimento demográfico, onde X_a é um número entre 0 e 1 e representa a população em um ano a, X_0 é a população no ano inicial ($a=0$), λ é um número positivo e representa a taxa combinada de reprodução e mortalidade. O comportamento caótico advém da competição entre reprodução e mortalidade, já que a reprodução aumenta a uma taxa proporcional à população atual, para pequenos valores de população, e a taxa de mortalidade aumenta conforme a população aumenta. Um exemplo ilustrativo de valores caóticos gerados pela equação 1 pode ser visto na figura 1.

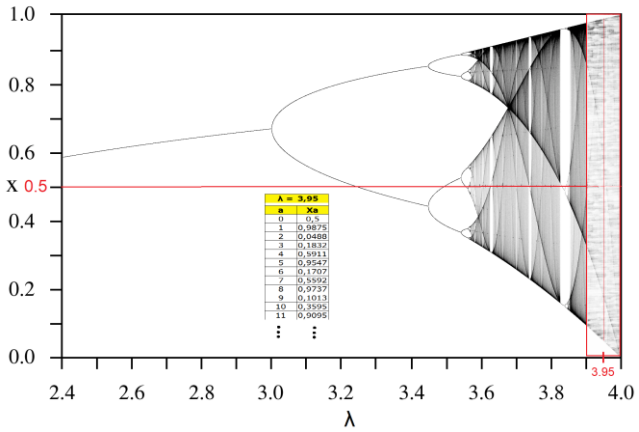

Fig. 1. Exemplo de caoticidade usando mapa logístico com X_0=0.5 e λ=3.95.

A partir de 1997, o *National Institute of Standard and Techology* (NIST) passou a recomendar uma bateria de testes estatísticos para verificar, em primeira instância, se o sistema em desenvolvimento se porta como um gerador de números aleatórios, como é o teste NIST 800-22. O NIST desenvolveu um pacote com 15 testes estatísticos para testar a existência ou não de aleatoriedade em uma sequência binária produzida por hardware e software de criptografia baseada em RNGs ou PRNGs. O tamanho da sequência adotado nesse trabalho é de 10^6 bits, por ser este tamanho o mínimo para submeter a bateria de testes de aleatoriedade do NIST. Em caso de aprovação em todos os testes da bateria, a sequência submetida será considerada nesse trabalho como chave, ou seja, uma sequência com aleatoriedade certificada para ser usada em processos de criptografia de dados. O NIST disponibiliza o código fonte da bateria de testes que servem de base para vários outros pacotes independentes com implementações para web, Windows e Linux. Em nosso caso usamos o pacote de https://randomness-tests.fi.muni.cz/ de Zdenek Ríha e Marek Sýs e foi feito a carga dos arquivos em 05/06/2024. Com base na documentação dos autores e algumas adaptações conseguimos fazer o programa ser executado direto do prompt do CMD no Windows 10.

A aprovação de uma amostra em algum teste envolve o cálculo de um valor de probabilidade (p-valor). Ele é definido como a probabilidade de se obter uma estatística de teste igual ou mais extrema quanto aquela observada em uma amostra, assumindo verdadeira a hipótese nula. Fixado um valor de

significância α, um teste falha se *p*-valor < α. Geralmente, o valor α escolhido está no intervalo [0.001, 0.01]. Em nosso sistema foi escolhido α = 0.01.

A descrição detalhada dos testes foge ao escopo deste trabalho, que os utiliza como ferramenta reconhecida no meio científico para teste de aleatoriedade, todavia, os detalhes teóricos sobre todos os testes podem ser observados em [16]. Porém, sua utilização na plataforma Windows e com aplicações no Matlab pode ser facilmente assimilada num tutorial que disponibilizamos no apêndice.

IV. ASPECTOS COMPUTACIONAIS

Discutiremos detalhadamente os passos para obter as sequências binárias e a codificação do método proposto. O processo lógico pode ser executado em ambiente computacional como MATLAB ou Octave em nosso caso usamos o MATLAB. As sequências binárias geradas ao final do processo são submetidas aos testes de NIST no mesmo ambiente computacional, porém em códigos distintos.

O Matlab possui funções para gerar números aleatórios como a função *rand* e *randi*, que utilizam por padrão o algoritmo Mersenne Twister. Esse algoritmo gera números pseudoaleatórios de alta qualidade. O Mersenne Twister foi desenvolvido especificamente para resolver falhas encontradas em algoritmos mais antigos.

Para melhor compreensão do método, nomeamos as funções realizadas pelo código gerador das sequências aleatórias em: Gerador de aleatoriedade primário (GA1), Gerador de aleatoriedade secundário (GA2) e Gerador de Caoticidade (GC).

A. *Gerador de Aleatoriedade primária(GA1)*

O objetivo do gerador de aleatoriedade primário (GA1), é de gerar uma sequência aleatória que permita descartar alguns de seus elementos como estratégia de camuflagem dessa sequência. Porém, quando se gera diretamente a sequência binária e se descarta alguns termos selecionados aleatoriamente, a sequência aleatória resultante, conforme observamos, dificilmente passa nos testes de NIST, o que inviabiliza a sua utilização como chave.

Dessa forma, num processo empírico de tentativas e erros, um método foi desenvolvido para se gerar uma sequência binária no Matlab, a partir de outra sequência não binária também gerada com função do Matlab e que, permita descarte aleatória de alguns de seus elementos e ainda assim tenha uma boa taxa de aprovação nos testes de NIST. Segue abaixo a descrição do método.

O sistema deve ser iniciado com as seguintes informações já definidas: o tamanho da chave desejada e o número de chaves a se obter. Em seguida, uma sequência aleatória não binária de dimensão *5x* maior que o tamanho da chave desejada é gerada e armazenado num vetor linha, a qual chamamos de *sequência primária*. O motivo para essa dimensão inicial é que elementos desse vetor serão descartados intencionalmente. O intervalo dos números aleatórios gerados é de 0 até 14 e o motivo de usarmos esse intervalo será mostrado adiante.

B. *Gerador de Aleatoriedade secundário(GA2)*

O GA2 usa como semente a *sequência primária* aleatória e não binária gerada pelo GA1. Inicialmente o GA2 escolhe um número qualquer de 0 a 14 que será descartado nas posições da *sequência primária* onde ele ocorre. No Matlab, GA2 faz as substituições nas posições onde aparece o número escolhido pela representação *NaN*(*Not a Number*), apenas marcando o descarte daquela informação. Os demais elementos do vetor são binarizados da seguinte forma, os números pares são substituídos por *0* e os ímpares por *1*. O vetor resultante dessa transformação é um novo vetor constituído dos 3 tipos de informação: *0*(zeros), *1*(uns) e *NaNs*. Chamamos este vetor de *sequência secundária*, lembrando que cada informação ocupa apenas uma posição dele. A sequência binária candidata a uma chave é exatamente um novo vetor obtido da *sequência secundária*, que é criado apenas retirando as posições que possuem *NaNs*, a qual denominamos de *sequência final*.

Porém, o vetor que será transferido para a próxima etapa da geração de chaves é o vetor de *pré-codificação*, que é criado a partir da *sequência secundária*. O vetor de *pré-codificação* tem 2 elementos representativos para cada 1 elemento da *sequência secundária*, onde os elementos *0s* (zeros) da sequência secundária são representados por *10* no vetor de pré-codificação, os *1s* (uns) são representados por *01* e os *NaNs* são representados por *00*. A justificativa específica para a *pré-codificação* em pares é criar mais uma dificuldade para espionagem, visto que, a posição exata do pulso representativo da informação muda em uma unidade temporal quando codificada em pares. Tal situação pode ser vista na figura 5 com a representação do trem de pulso da *mensagem* a ser enviado de Alice(transmissor) para Bob(receptor) em modulação de amplitude pelo canal óptico no processo de distribuição quântica de chaves quando ele for implementado.

A figura 2 contém um exemplo para representar toda transformação ocorrida no GA1 e GA2.

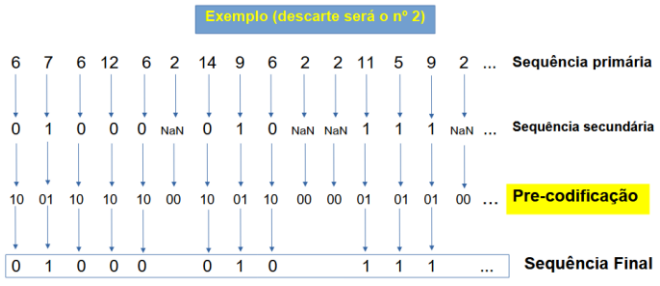

Fig. 2. Exemplo de transformações ocorridas devido a ação dos geradores GA1 e GA2 para geração de uma sequência aleatória de *n* bits

C. *Considerações sobre GA1 e GA2*

Simulações foram realizadas para testar intervalos de sequências aleatórias de 0 a 2, 0 a 3, 0 a 4, até de 0 a 30, aplicando sempre GA1 e GA2 para obter a *sequência final* de tamanho 10^6 bits num total de 1000 amostras de *sequências finais* geradas.

Concluímos empiricamente que o melhor intervalo numérico da *sequência primária* é o de 0 a 14 para se obter a melhor taxa de aprovação de *sequências finais* nos testes de NIST, conforme se observa na figura 3.

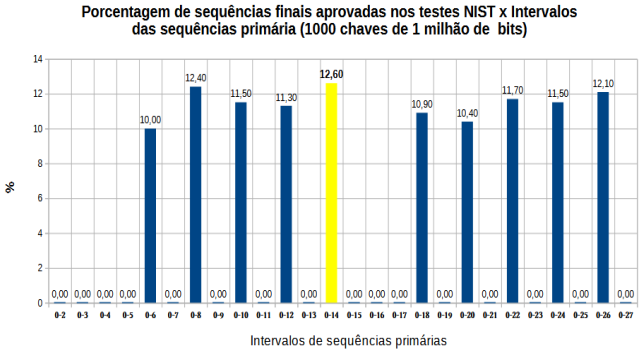

Fig. 3. Intervalos de *sequências primárias* que possibilitam melhor taxa de aprovação nos testes de NIST para *sequências finais*.

D. Gerador caótico GC

O gerador caótico GC tem a função de gerar valores caóticos que representem a quantidade de 0s (zeros) que devem se interpor entre os pares da sequência da *pré-codificação*. Para isso o GC usa o mapa logístico da equação (1), onde o parâmetro X deve estar no intervalo de 0 a 1 e o parâmetro λ neste trabalho fica entre [3.9, 4], o que permite ao GC maior sensibilidade às condições iniciais conforme visto no mapa caótico da figura 1. Os valores da tabela em destaque na figura 1 estão sobre a linha vertical passando sobre o ponto onde $\lambda=3.95$, oscilando acima ou abaixo de $X_0=0.5$. A área em cinza nessa figura representa a região de caoticidade, que cresce ao se aproximar de $\lambda=4$.

Aproveitando a tabela de valores caóticos da figura 1, juntando ao vetor de *pré-codificação* da figura 2, podemos montar um exemplo de funcionamento do GC para obter a *mensagem* que é enviada pelo canal óptico em uma QKD, como visto na figura 4.

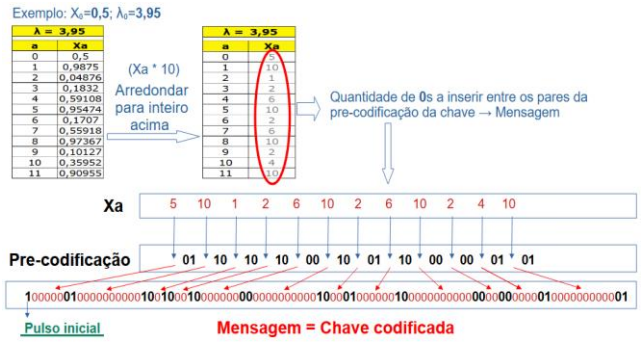

Fig. 4. Esquema de funcionamento do Gerador Caótico GC.

Na figura 4, os parâmetros iniciais $X_0=0.5$ e $\lambda=3.95$ geram a caoticidade e a cada iteração a, o resultado é multiplicado por 10 e arredondado para cima de forma a nunca resultar em X_a=0. O vetor de resultados X_a contém a quantidade de zeros a se interpor entre os elementos do vetor de pré-codificação apresentado na figura 1. O resultado é a *mensagem*, que é um vetor que contém a chave codificada e pronto para ser usada em QKD. O primeiro bit da *mensagem* é sempre 1, pois representa o *pulso inicial* (barra verde na figura 5) que desperta o receptor de no processo QKD, quando o pacote de dados é enviado em modulação de amplitude, conforme visto na figura 5.

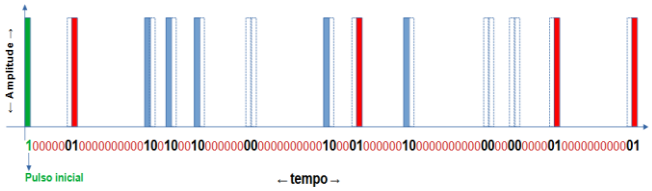

Fig. 5. Representação do trem de pulso do exemplo da figura 3 a ser enviado em modulação de amplitude por canal óptico em QKD.

Importante observar na figura 5 como os itens de segurança da informação foram acrescentados aos poucos, desde a *pré-codificação* dos bits em pares binários até a inserção de quantidades caóticas de zeros no trem de pulso. Na intenção de frustrar a espionagem, cujo principal ataque físico é desviar o sinal do canal óptico entre Alice e Bob para uma estação de criptoanálise de dados (chamada de *Eva*), os pares representativos do bit 0 (barras azul e vazio) e 1 (barras vazio e vermelho) se mostram deslocados no tempo. Tal deslocamento confunde a previsibilidade de uma sequência aleatória, que se complica ainda mais considerando as posições de perda(barras vazio e vazio) inseridas com o GA1. A inserção de intervalos temporais caóticos (inserção de zeros entre os pares) feitos pelo GC amplifica ainda mais a dificuldade em prever quando ocorrerá um novo pulso representativo de informação.

V. RESULTADOS

Foram gerados pelo método proposto 1000 chaves de tamanho 10^6 bits que foram submetidas aos testes de NIST. Para fazer uma comparação de qualidade no âmbito da geração de chaves, foi gerado 1000 chaves de tamanho 10^6 bits, em outros sistemas diretamente em binário, a saber: no algoritmo Mersene Twister no Matlab, no algoritmo Java Script(muito usados na criptografia de celulares) e num gerador quântico de números aleatórios GNAQ com base nas flutuações do vácuo, através do site https://qrng.anu.edu.au/random-binary/, acessado no dia 10/07/2023. Os resultados do comparativo estão apresentados na Tabela I, onde é mostrado a taxa de aprovação nos testes de NIST e tempo decorrido para obter as amostras, dos geradores de números pseudoaleatório GNPA e gerador quântico de números aleatórios GNAQ com base nas flutuações do vácuo.

TABELA I. COMPARATIVO ENTRE GERADORES DE NÚMEROS ALEATÓRIOS(1000 AMOSTRAS DE 10^6 BITS).

Fonte geradora das chaves	Chaves aprovadas	Taxa de aprovação NIST(%)
GNPA (sistema caótico proposto)	115	11.50
GNPA (Algoritmo Mersenne Twister)	114	11.40
GNPA (Algoritmo Java Script)	118	11.80
GNAQ (flutuações do vácuo)	114	11.40

Se o sistema proposto for otimizado para gerar apenas chaves aprovadas nos testes de NIST, então o tempo para realizar a geração de chaves será, em média 3 minutos por chave gerada.

VI. CONCLUSÕES

Foi mostrado neste trabalho que é possível gerar de forma empírica sequências aleatórias com taxa de aprovação nos testes de NIST próxima a dos demais geradores aqui testados, como os geradores pseudoaleatórios(algoritmos Mersenne Twister e Java Script) e até mesmo de um gerador quântico de números aleatórios. Mesmo a do gerador real como a do GNAQ aqui analisado, seus dados brutos extraídos de suas fontes geradoras passam por algoritmos de extração da aleatoriedade [17].

Mostramos também que usando uma pré-codificação das chaves e o mapa logístico é possível obter uma codificação bem qualificada para QKD, ainda no domínio clássico e com baixo custo de hardware envolvido.

O fator mais relevante do método empírico aqui proposto para geração de chaves é que durante o processo da geração se acrescenta mais camadas de segurança que podem ser imediatamente aproveitados na etapa de distribuição de chaves. Especificamente, o método proposto pode ser combinado com o protocolo de QKD como o BB84 [12], implementado em rede óptica com laser fortemente atenuado ao regime quântico, em que a codificação por modulação de amplitude imposta pelo novo método se soma as outras possíveis codificações de polarização ou fase, podendo ser utilizado em um link de comunicação quântica onde o receptor utilize detectores do tipo APD.

AGRADECIMENTOS

Este trabalho foi parcialmente financiado pela agência CAPES, processo 88887.950067/2024-00.

REFERÊNCIAS

[1] L. Chen, S. Jordan, Y.-K. Liu, D. Moody, R. Peralta, R. Perlner, and D. Smith-Tone, *Report on post-quantum cryptography*, U. S. Dept. Commerce, Nat. Inst. Standards Technol., NIST Interagency Rep. 8105, 2016.

[2] G. Alagic *et al.*, *Status report on the second round of the NIST postquantum cryptography standardization process*. U. S. Dept. Commerce, Nat. Inst. Standards Technol., NIST Interagency Rep. 8309, 2020.

[3] P. Ravi, A. Chattopadhyay e S. Bhasin, "Security and Quantum Computing: An Overview," 2022 *IEEE 23rd Latin American Test Symposium (LATS)*, doi: 10.1109/LATS57337.2022.9936966.

[4] Peter W Shor. "Algorithms for quantum computation: Discrete logarithms and factoring". *In Foundations of Computer Science, 1994 Proceedings., 35th Annual Symposium on*, p. 124–134. IEEE, 1994.

[5] John Proos and Christof Zalka. "Shor's discrete logarithm quantum algorithm for elliptic curves". *arXiv preprint quant-ph*/0301141, 2003.

[6] Thomas Häner, Samuel Jaques, Michael Naehrig, Martin Roetteler, and Mathias Soeken. *Improved quantum circuits for elliptic curve discrete logarithms*. In Jintai Ding and Jean-Pierre Tillich, editors, Post-Quantum Cryptography, p. 425–444, Cham, 2020. Springer International Publishing.

[7] Craig Gidney and Martin Ekerå. "How to factor 2048 bit RSA integers in 8 hours using 20 million noisy qubits". *Quantum*, 5:433, April 2021.

[8] Andre Chailloux, Maria Naya-Plasencia, and Andre Schrottenloher. "An efficient quantum collision search algorithm and implications on symmetric cryptography". *Cryptology ePrint Archive*, Paper 2017/847, 2017. https://eprint.iacr.org/2017/847.

[9] Kyungbae Jang, Anubhab Baksi, Gyeongju Song, Hyunji Kim, Hwajeong Seo, and Anupam Chattopadhyay. "Quantum analysis of AES". *IACR Cryptol. ePrint Arch*., p. 683, 2022.

[10] Zhenyu Huang and Siwei Sun. "Synthesizing quantum circuits of AES with lower t-depth and less qubits". *IACR Cryptol. ePrint Arch.*, p. 620, 2022.

[11] Kyungbae Jang, Anubhab Baksi, Jakub Breier, Hwajeong Seo, and Anupam Chattopadhyay. "Quantum implementation and analysis of default". *Cryptology ePrint Archive, Paper* 2022/647, 2022. https://eprint.iacr.org/2022/647.

[12] Charles H Bennett and Gilles Brassard. "Quantum cryptography: Public key distribution and coin tossing". *arXiv preprint arXiv*:2003.06557, 2020.

[13] NIST Special Publication 800-133: *Recommendation for Cryptographic Key Generation*, draft, August 2011.

[14] Hellekalek, P. "Good random number generators are (not so) easy to find". *Mathematics and Computers in Simulation*, v. 46, n. 5–6, p. 485–505, 1998.

[15] E. R. V. Jr. *Parameter spaces for an experimental chaotic circuit* 2010. 177p. Thesis (Master's degree) – Department of Physics Institute of Exact Sciences Minas Gerais, Belo Horizonte.

[16] Bassham, L., Rukhin, A., Soto, J., Nechvatal, J., Smid, M., Barker, E., Leigh, S., Levenson, M., Vangel, M., Banks, D., Heckert, N., Dray, J. and Vo, S. (2010), *A statistical test suite for random and pseudorandom number generators for cryptographic applications: National Institute of Standards and Technology*. Gaithersburg, MD, [online], https://doi.org/10.6028/NIST.SP.800-22r1a.

[17] T. Symul, S. M. Assad, P. K. Lam. "Real time demonstration of high bitrate quantum random number generationwith coherent laser light". *Appl. Phys. Lett.* 98, 231103/1-4 (2011). https://doi.org/10.1063/1.3597793.

APÊNDICE

Roteiro para rodar os testes de NIST

O roteiro abaixo usa o pacote de https://randomness-tests.fi.muni.cz/ de **Zdenek Ríha** e **Marek Sýs** e foi feito a carga dos arquivos em 13/06/2023. Com base na documentação dos autores e algumas adaptações conseguimos fazer o programa ser executado direto do prompt do CMD no Windows 10.

É preciso baixar do GitHub os arquivos customizados: https://github.com/JoacirSo/Teste_NIST
Ou leia o QR code abaixo e depois faça download dos arquivos

Assista os vídeos explicativos de como usar este pacote e aproveitar de mais alguns exemplos que disponibilizamos.