A Project Report

On

# An Approach to Cryptography Based on Continuous-Variable Quantum Neural Network

BY

**SIDDHARTH SINGH**

**2020B5A32029H**

**DHRUV DESAI**

**2020B5AA2278H**

Under the supervision of

**Prof. T S L Radhika**

**BIRLA INSTITUTE OF TECHNOLOGY AND SCIENCE PILANI (RAJASTHAN)**

**HYDERABAD CAMPUS**

**(OCTOBER 2023)**

## Introduction

Cryptography is the process of hiding or coding information so that only the person a message was intended for can read it. It is an essential tool for protecting sensitive data. Modern encryption algorithms, such as RSA and AES, are designed to be computationally secure as they rely on the difficulty of solving complex mathematical problems, like factoring large numbers, which classical computers struggle to solve efficiently. The sheer size of the numbers involved makes brute-force attacks unfeasible. The development of quantum computers and quantum algorithms, poses a threat to such techniques, as they can exploit some quantum mechanical properties, such as superposition and entanglement, to perform some calculations much faster than classical computers. For example, Shor's algorithm can efficiently factor large numbers, which is used in many public-key cryptography algorithms.

This is why there is a need for techniques that are secure against both classical and quantum computers. We will train a Continuous Variable Quantum Neural Network to encrypt the data into cipher text, which is decrypted by the receiver by performing the inverse operations, which is unique to the CV-QNN.

**Literature Review:**

| Title | Objective |
|---|---|
| An Approach to Cryptography Based on Continuous-Variable Quantum Neural Network | The objective of this paper is to propose an efficient and secure quantum cryptography scheme using continuous-variable quantum neural network and demonstrate its feasibility through simulation experiments on the Strawberry Fields platform. |
| Quantum data compression by principal component analysis. | The objective of this paper is to introduce a quantum algorithm for data compression using principal component analysis and to demonstrate its exponential speedup over classical methods for compressing high-dimensional but approximately low-rank datasets. The paper also discusses the adaptation of this algorithm to existing quantum machine learning algorithms and explores potential applications in machine learning and data mining. |

**Objectives**

This project aims to present a novel method for quantum cryptography that generates, encrypts, and decrypts safe keys using a continuous-variable quantum neural network (CVQNN). The purpose of this project is to use simulation tests on Jupyter notebook using Strawberry Fields library to show the viability and effectiveness of this strategy. The suggested method overcomes the drawbacks of current quantum cryptography techniques, including high key rate requirements and massive key demands, and is made to be safe against both quantum and classical computers. The possible uses of CVQNN-based cryptography in a number of domains, such as financial transactions, data privacy, and secure communication, are also covered in this work. This paper's overall goal is to present a cutting-edge technology that has the potential to improve security and efficiency.

## Methodology

**Quantum State Preparation**

Initializing:Initializing a quantum program with the number of modes proportional to the binary data length is the first step in the procedure. Every mode is like a pathway that the quantum data will follow.

Encoding Operations: In the input data, for every binary digit:

The associated mode undergoes a squeezing operation (Sgate), the degree of which is dictated by a particular value ‘r’ from the params array. In order to prepare for the encoding of classical data, this process modifies the uncertainty in the quadratures of the quantum mode.

A displacement operation (Dgate) is then performed, which moves the quantum mode in the phase space. Another number (alpha) from the params array controls the size of this shift, which embeds the binary digit into the quantum state.

Entanglement and State Preparation: If the input data contains several bits, the modes are entangled using a beamsplitter operation (BSgate), establishing quantum correlations between them. This entanglement is critical for using quantum mechanical features effectively during subsequent calculations.

**Quantum Encryption Process**

Initialization: We begin by initialising a quantum program with a number of modes corresponding to the subsystems of the input quantum state, laying the groundwork for our quantum circuit.

Key-Dependent Modulation: We use displacement gates parameterized by the ASCII values of the cryptographic key's characters for each mode. These gates cause shifts in phase space, which encode the key into the quantum state.

Entanglement and Nonlinearity: The quantum state's modes are then entangled pairwise using beamsplitter gates, and nonlinearity is introduced using Kerr gates. These procedures are required for the complex changes of the quantum state, which ensures a high level of security.

**Decryption Process**

Inverse Nonlinearity: The function begins by applying inverse Kerr operations (Kgate) to each mode, which negates the nonlinear effects generated during the encryption phase.

Following nonlinearity reversal, the modes are disentangled by using beamsplitter gates (BSgate) in reverse order, progressively unwinding the entanglement between the modes.

Key-Specific Inverse Operations: Key-dependent displacement operations (Dgate) are used in the decryption process. To negate the key-specific shifts made during encryption, each mode undergoes a primary displacement followed by a secondary displacement with an inverse parameter.

**Measurement and Retrieval**

The quantum states are measured using the MeasureX operation after quantum processing, turning the quantum information back into conventional binary data. The measurement function (measure_quantum_state) is designed to handle possible errors, allowing for the identification and reporting of measurement issues.

**Optimization and Error Evaluation:**

By comparing the decrypted output to the original plaintext, a cost function (cost_function) is used to evaluate the accuracy of the encryption-decryption cycle. The primary function's optimisation procedure iteratively modifies the quantum gate parameters to minimise this cost, aiming for error-free encryption decryption.

## Code

```
import strawberryfields as sf
from strawberryfields.ops import Sgate, BSgate, MeasureFock, Dgate, Rgate,Kgate, MeasureX
import random
```

```
def generate_samples(num_samples, length):
    return [''.join(random.choice('01') for _ in range(length)) for _ in range(num_samples)]
samples = generate_samples(100, 2)
```

```
def cv_qnn_cryptography(plaintext, key, operation, params):
    quantum_state = encode_to_quantum_state(plaintext, params)
    processed_state = cv_qnn_process(quantum_state, key)  # Removed the 'params' argument

    if operation == 'encrypt':
        encrypted_data = measure_quantum_state(processed_state)
        return encrypted_data
    elif operation == 'decrypt':
        decrypted_data = cv_qnn_decrypt(processed_state, key)  # Removed the 'params' argument
        return measure_quantum_state(decrypted_data)
```

```
def encode_to_quantum_state(data, params):

    prog = sf.Program(len(data))

    with prog.context as q:
        for i, bit in enumerate(data):

            r = params[i * 2]
            Sgate(r) | q[i]


            alpha = params[i * 2 + 1]
            Dgate(alpha) | q[i]


        if len(data) > 1:
            BSgate() | (q[0], q[1])

    return prog
```

```python
def cv_qnn_process(state, key):

    num_modes = state.num_subsystems

    prog = sf.Program(num_modes)

    with prog.context as q:


        for i in range(num_modes):

            ascii_value = float(ord(key[i % len(key)])) / 100
            Dgate(ascii_value) | q[i]
            Sgate(ascii_value * 0.1) | q[i]


        for i in range(num_modes - 1):

            BSgate() | (q[i], q[i + 1])
            Kgate(0.5) | q[i]

    return prog
```

```python
def cv_qnn_decrypt(state, key):
    num_modes = state.num_subsystems

    prog = sf.Program(num_modes)

    with prog.context as q:

        for i in range(num_modes - 1):
            Kgate(-0.5) | q[i]


        for i in range(num_modes - 1, 0, -1):
            BSgate() | (q[i], q[i - 1])


        for i in range(num_modes):
            ascii_value = float(ord(key[i % len(key)])) / 100
            Dgate(-ascii_value) | q[i]
            Sgate(-ascii_value * 0.1) | q[i]

    return prog
```

```python
def measure_quantum_state(program):
    with program.context as q:
        for mode in q:
            MeasureX | mode


    eng = sf.Engine(backend='fock', backend_options={"cutoff_dim": 10})
    try:
        results = eng.run(program)
    except ValueError as e:
        print("Error during measurement:", e)
        return None


    if results is not None:
        measured_values = [results.samples[i][0] for i in range(len(results.samples))]
        return measured_values
    else:
        return None
```

```python
def cost_function(params, samples, key):
    total_cost = 0
    for plaintext in samples:
        encrypted_data = cv_qnn_cryptography(plaintext, key, 'encrypt', params)
        decrypted_data = cv_qnn_cryptography(encrypted_data, key, 'decrypt', params)
        decrypted_text = ''.join(str(int(x)) for x in decrypted_data)
        total_cost += sum(1 for x, y in zip(plaintext, decrypted_text) if x != y)
    return total_cost / len(samples)
```

```python
def main():
    key = "key"
    samples = generate_samples(100, 2)  # Generate 100 binary string samples
    params = [0.5] * 4  # Initial parameters for the gates

    learning_rate = 0.1
    for step in range(100):
        cost = cost_function(params, samples, key)
        print(f"Step {step}, Cost: {cost}")


        params = [p - learning_rate for p in params]

        if cost == 0:
            break

    print("Training completed. Optimized parameters:", params)

    # Test the trained model on new plaintext "101"
    new_plaintext = "11"
    encrypted_data = cv_qnn_cryptography(new_plaintext, key, 'encrypt', params)
    decrypted_data = cv_qnn_cryptography(encrypted_data, key, 'decrypt', params)
    decrypted_text = ''.join(str(int(x)) for x in decrypted_data)

    print(f"Original Plaintext: {new_plaintext}")
    print(f"Decrypted Text: {decrypted_text}")
    print(f"Match: {'Success' if new_plaintext == decrypted_text else 'Failure'}")
```

```
if __name__ == "__main__":
    main()
```

```
Step 0, Cost: 1.71
Step 1, Cost: 1.66
Step 2, Cost: 1.7
Step 3, Cost: 1.77
Step 4, Cost: 1.74
Step 5, Cost: 1.72
Step 6, Cost: 1.82
Step 7, Cost: 1.81
Step 8, Cost: 1.75
Step 9, Cost: 1.74
Step 10, Cost: 1.77
Step 11, Cost: 1.67
Step 12, Cost: 1.64
Step 13, Cost: 1.69
Step 14, Cost: 1.59
Step 15, Cost: 1.73
Step 16, Cost: 1.68
Step 17, Cost: 1.75
Step 18, Cost: 1.7
```

```
if __name__ == "__main__":
    main()
```

```
Step 84, Cost: 1.78
Step 85, Cost: 1.81
Step 86, Cost: 1.63
Step 87, Cost: 1.8
Step 88, Cost: 1.68
Step 89, Cost: 1.71
Step 90, Cost: 1.72
Step 91, Cost: 1.69
Step 92, Cost: 1.66
Step 93, Cost: 1.64
Step 94, Cost: 1.65
Step 95, Cost: 1.68
Step 96, Cost: 1.67
Step 97, Cost: 1.75
Step 98, Cost: 1.76
Step 99, Cost: 1.7
Training completed. Optimized parameters: [-9.499999999999982, -9.499999999999982, -9.499999999999982, -9.4999999999
99982]
```

## References


- An Approach to Cryptography Based on Continuous-Variable Quantum Neural Network. Jinjing Shi 1,4*, ShuhuiChen1,4, Yuhu Lu1, Yanyan Feng1*, Ronghua Shi 1, YuguangYang2 & Jian Li 3

- Quantum data compression by principal component analysis.
  Chao-Hua Yu,1, 2, 3, ∗ Fei Gao,1, † Song Lin,4 and Jingbo Wang3, ‡ 1State Key Laboratory of Networking and Switching Technology, Beijing University of Posts and Telecommunications, Beijing 100876, China 2State Key Laboratory of Cryptology, P.O. Box 5159, Beijing, 100878, China 3School of Physics, University of Western Australia, Perth 6009, Australia 4College of Mathematics and Informatics, Fujian Normal University, Fuzhou 350007, China (Dated: November 26, 2018)