Quantum Money

Shagun Khare

Defining The Problem

What Is Quantum Money?

Protocol to create and validate banknotes impossible to forge

Two Sectors:

- Private-key
- Public-key

Stephen Wiesner (1942-2021)

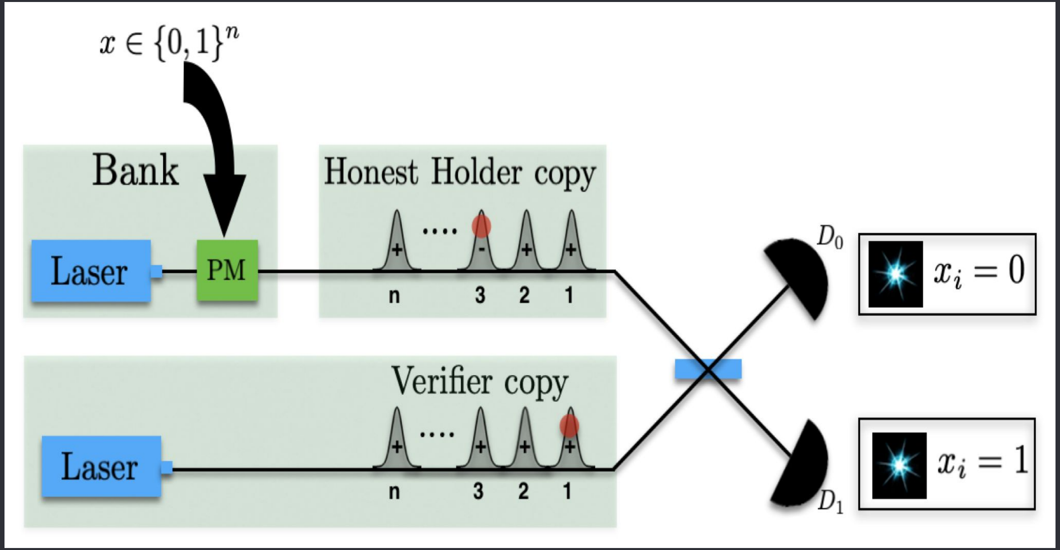
Wiesner's Quantum Money Scheme

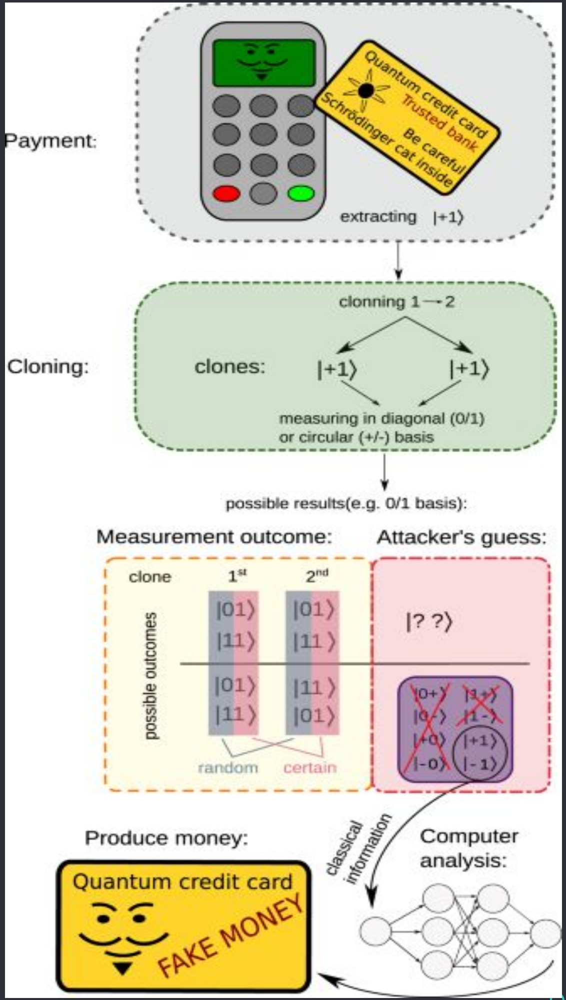
- ➢ Central bank prints "quantum bills"
  - » $|0\rangle$,
    » $|1\rangle$,
    » $|+\rangle = (|0\rangle+|1\rangle)/\sqrt{2}$, or
    » $|-\rangle = (|0\rangle-|1\rangle)/\sqrt{2}$.
- ➢ Measures each qubit in the correct basis
- ➢ Counterfeiter would fail with a a probability of $(3/4)^N$ with n qubits

Further Study In Weisner's Model

- Chance of getting "stolen" by an adversary
- Tokunada (2003) → prevent mint from tracking each bill
- Mosca and Stebila (2007) → *quantum coins*

Classical Variation





Qiskit Implementation

```python
def create_money(bits, bases):
    message = []
    for i in range(3):
        qc = QuantumCircuit(1,1)
        if bases[i] == 0: # Prepare qubit in Z-basis
            if bits[i] == 0:
                pass
            else:
                qc.x(0)
        else: # Prepare qubit in X-basis
            if bits[i] == 0:
                qc.h(0)
            else:
                qc.x(0)
                qc.h(0)
        qc.barrier()
        message.append(qc)
    return message
```

```python
bank_fixednote = np.array([1,0,0])
bank_fixedbase = np.array([0,1,1])

banknote = create_money(bank_fixednote, bank_fixedbase)

bank_bases = randint(2, size=3)
```

```python
def measure_message(message, bases):
    backend = Aer.get_backend('aer_simulator')
    measurements = []
    for q in range(3):
        if bases[q] == 0: # measuring in Z-basis
            message[q].measure(0,0)
        if bases[q] == 1: # measuring in X-basis
            message[q].h(0)
            message[q].measure(0,0)
        aer_sim = Aer.get_backend('aer_simulator')
        qobj = assemble(message[q], shots=1, memory=True)
        result = aer_sim.run(qobj).result()
        measured_bit = int(result.get_memory()[0])
        measurements.append(measured_bit)
    return measurements
```

```python
prob = 0
shots = 100

for i in range(100):
    forge_bits = randint(2, size=3)
    forge_bases = randint(2, size=3)
    forge_note = create_money(forge_bits, forge_bases)
    bank_measure = measure_message(forge_note, bank_bases)
    bank_fixed = measure_message(banknote, bank_fixedbase)
    if(bank_fixed==bank_measure):
        print("Bank Serial Number: " + str(bank_fixed))
        print("Verfied Bill: " + str(bank_measure))
    else:
        print("Bank Serial Number: " + str(bank_fixed))
        print("Forged Bill: " + str(bank_measure))
        prob = prob+1
    print("")

print("Probability of bills being forged:" + str(prob/shots))
```

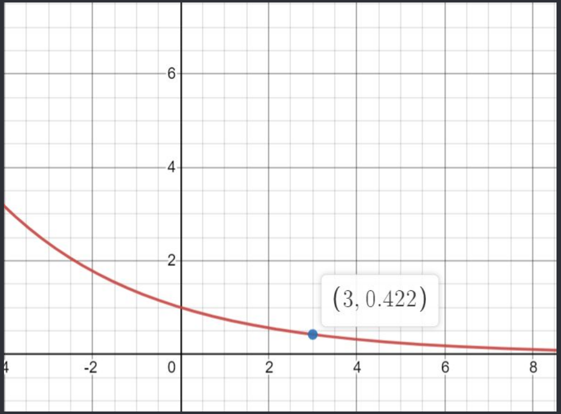
Example Result:

```
Bank Serial Number: [1, 1, 1]
Verfied Bill: [1, 1, 1]
Bank Serial Number: [1, 1, 1]
Forged Bill: [0, 1, 0]
Bank Serial Number: [1, 1, 0]
Forged Bill: [0, 1, 1]

Bank Serial Number: [1, 0, 1]
Forged Bill: [0, 1, 0]
Probability of bills being forged:0.42
```

Proof:

(¾)^3 = 0.421875

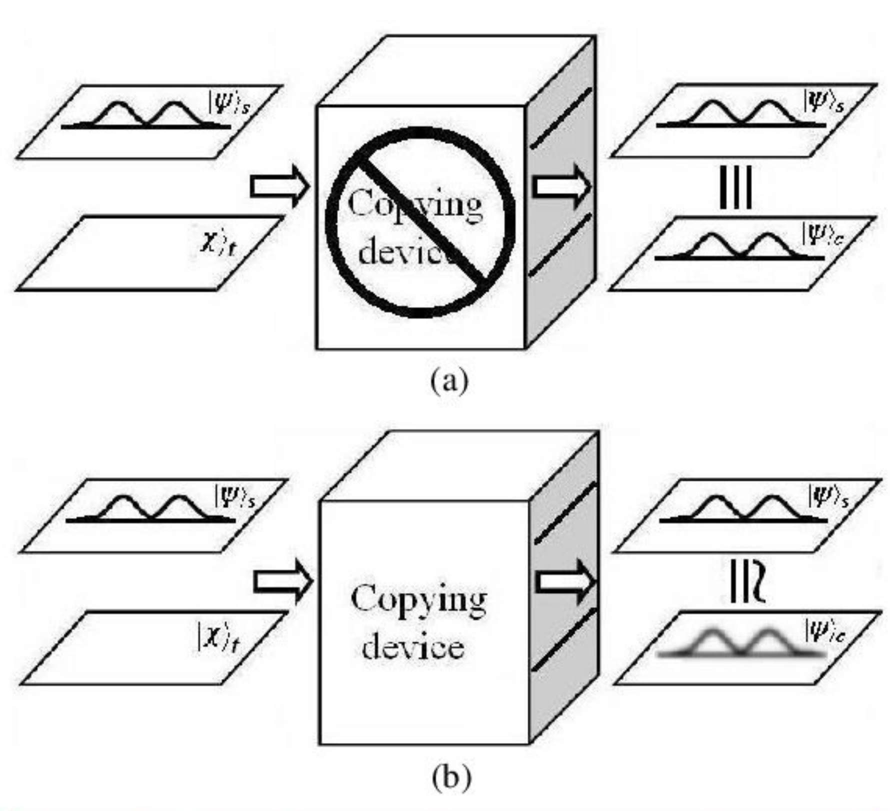


Advantages of this Scheme

- Coins become exponentially harder to counterfeit as attempts increase
- Bank's database can be static
- Does not require 3-party authentication
- Dependence between the number of verifications and the number of qubits it contains is optimal

No-Cloning Theorom

Cannot create a copy of an arbitrary quantum state

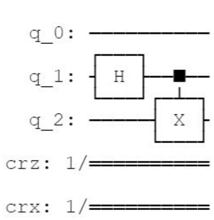


Qiskit Implementation

```python
bill = QuantumRegister(3, name="q")
crz = ClassicalRegister(1, name="crz")
crx = ClassicalRegister(1, name="crx")
bank_circuit = QuantumCircuit(bill, crz, crx)
```

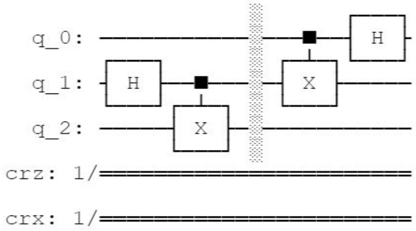
```python
def create_bell_pair(qc, a, b):
    qc.h(a)
    qc.cx(a,b)
```

```python
create_bell_pair(bank_circuit, 1, 2)
bank_circuit.draw()
```



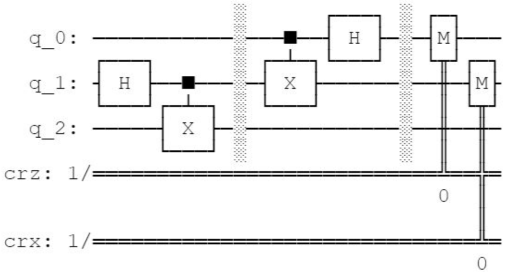

```python
def alice_bill(qc, psi, a):
    qc.cx(psi, a)
    qc.h(psi)
```

```python
bank_circuit.barrier()
alice_bill(bank_circuit, 0, 1)
bank_circuit.draw()
```



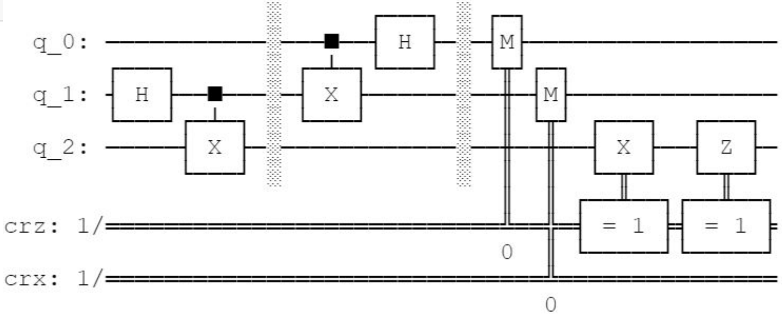

```python
def measure_and_send(qc, a, b):
    """Measures qubits a & b and 'sends' the results to Bob"""
    qc.barrier()
    qc.measure(a,0)
    qc.measure(b,1)
```

```python
measure_and_send(bank_circuit, 0 ,1)
bank_circuit.draw()
```




```python
def bank_apply(qc, qubit, crz, crx):
    qc.x(qubit).c_if(crx, 1)
    qc.z(qubit).c_if(crz, 1)
```

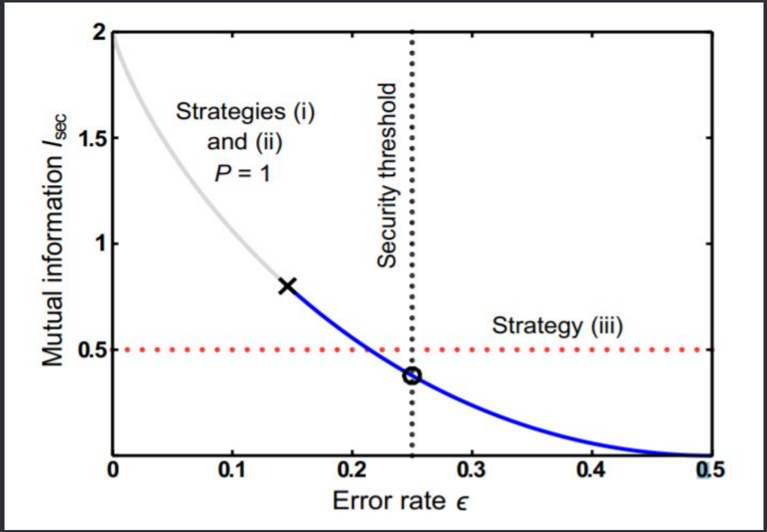
```python
bank_apply(bank_circuit, 2, crz, crz)
bank_circuit.draw()
```

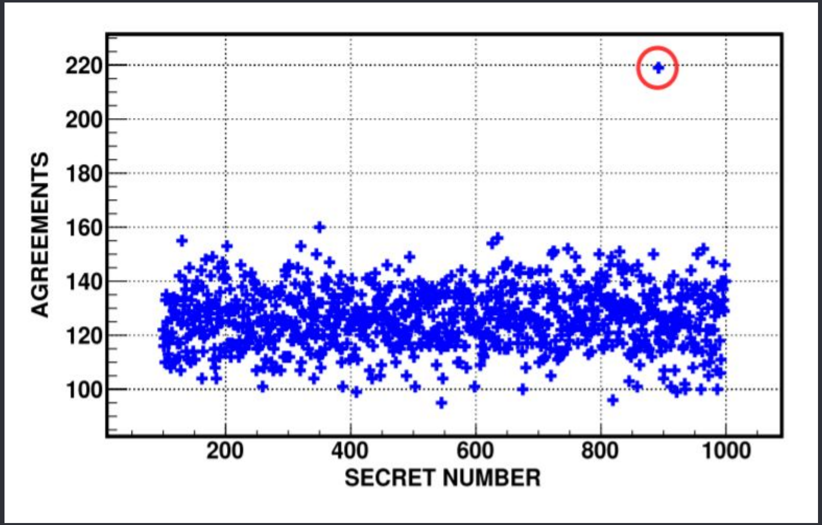


Benefits of this Model

- ➢ Obtain exponential security
- ➢ Produce notes that are publicly verifiable
- ➢ Cannot copy even with access to a verifier

Possible Attacks and Security Measures





(Jiráková, K. and Lemr, K.)

Future Outlook on Quantum Money

- ➢ Implementations of Quantum Key Distribution
- ➢ Practical implementation not proved yet
- ➢ Government and commercial sectors

References

1. Aaronson, S. Quantum copy-protection and quantum money. In Annual IEEE Conference on Computational Complexity (2009), 229242.
2. S. Aaronson and P. Christiano. Quantum money from hidden subspaces. In preparation, 2012.
3. A. Lutomirski. An Online Attack Against Wiesner's Quantum Money. http://arxiv.org/abs/1010.0256, 2010.
4. Bennett, C.H., Brassard, G., Breidbart, S. and Wiesner, S. Quantum cryptography, or unforgeable subway tokens. In Advances in CryptologyProceedings of Crypto (1983), volume 82, 267275.
5. Jiráková, K., Bartkiewicz, K. Černoch, A. and Lemr,K. Experimentally attacking quantum money schemes based on quantum retrieval games. (2019)
6. Mosca, M. and Stebila, D. A framework for quantum money. Poster at Quantum Information Processing (QIP) (Brisbane, Australia, 2007).

Thank You!

Questions?