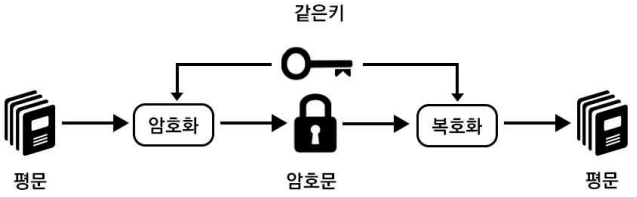
BB84 프로토콜

2025113574 권나현
2025.12.10
System Software Lab.

목차

1. 배경: 암호키

- 메시지 암호화/복호화 표준 2단계
  1. 거래의 두 당사자가 최초에 서로 합의한 키 값을 정함 (= 길이가 긴 이진수 문자열)
  2. 거래 당사자는 이후부터 이 키를 이용해 메시지 암호화&복호화



=> 보안성: 키에서 비롯됨
=> Alice&Bob의 목표: Eve가 모르도록 키 값을 합의

1. 배경: 현대의 암호 vs 양자 암호

- 현대 암호의 특징

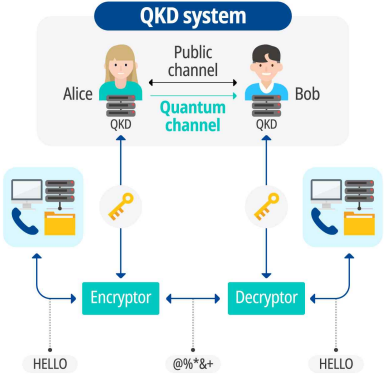
  - 안전성의 근거: 수학적 난이도
  ex) 큰 수 소인수분해, 이산 로그 문제
  - 양자컴퓨터 등장 시 취약

- 양자 암호의 특징

  - 안전성의 근거: 양자역학의 물리 법칙
  - 도청이 일어나면 측정이 상태를 교란하므로 자동으로 탐지 가능 -> BB84 프로토콜

1. 배경: 현대의 암호 vs 양자 암호

- 양자 암호의 물리적 구현 방식: QKD 시스템



- Public channel(공개 채널)

고전적인 정보 교환용 통로
도청이 가능하나 교환 내용 자체는 키가 아니므로 위험 없음
주요 사용: 수신자의 어떤 비트가 송신자와 일치하는지 기저 공개
: 오류율(Eve 존재 가능성) 체크

- Quantum channel(양자 채널)

실제 양자 상태(qubit)를 전송하는 통로
광섬유 케이블을 통해 개별 광자를 전송하며 각 광자는 데이터 비트(0 또는 1)를 나타냄
송신 측의 편광 필터: 각 광자의 방향 설정
수신 측: 빔 분할기를 사용하여 전송받은 광자를 읽음
송신 측과 수신 측은 광자의 방향을 비교하고 일치하는 방향을 암호화 키로 사용

2. 정의: BB84 프로토콜

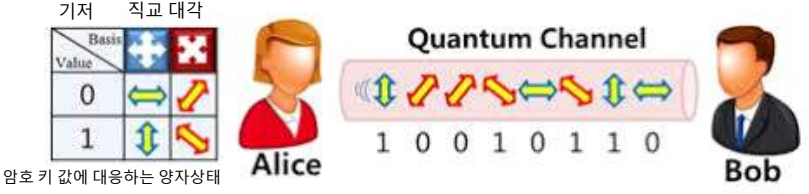
- BB84 = 찰스 베넷(B) + 질 브라사르(B) + 1984년(84)
- 최초의 QKD 프로토콜, 양자암호의 표준 모델
- 송신자(Alice)와 수신자(Bob)이 양자채널을 통해 비밀키를 생성하는 프로토콜

- 핵심 아이디어

  - Alice가 무작위 비트를 두 가지 기저 중 하나로 인코딩해 보낸다.
  - Bob은 무작위 기저로 측정한다.
  - 두 사람은 고전 채널을 통해 기저가 일치한 비트만 추려낸다.
  - 도청자가 있으면 에러율이 증가한다. -> 탐지 가능

3. 원리: BB84 프로토콜

- 광자의 편광 상태 -> 양자비트로 이용
- 2개의 기저(Z,X)를 이루는 4개의 양자상태(0,1,+,–) 사용
- 보안 원리: 복제 불가능성, 측정 교란



3. 원리: BB84 프로토콜

- BB84에서 4개의 양자상태:
  '광자의 편광 방향(편광 상태)'을 양자비트로 사용한 것
  (편광: 전자기파가 진행할 때 파를 구성하는 전기장이나 자기장이 특정한 방향으로 진동하는 현상)
  이유: 광자의 편광이 양자역학적으로 2차원 힐베르트 공간의 상태벡터이기 때문이다.

- 즉 BB84가 쓰는 4개의 양자상태는 광자가 다음 중 하나로 편광된 상태이다:
  수직, 수평, 45°, 135°.
  이것들을 각각 |0⟩, |1⟩, |+⟩, |−⟩로 기호화할 수 있다.

- Z 기저(직교 기저)
- |0⟩ = |↕⟩ (수직 편광)
- |1⟩ = |↔⟩ (수평 편광)

- X 기저(대각 기저)
- |+⟩ = |↗ 45° 편광
- |−⟩ = |↘ 135° 편광

4. 절차: BB84 프로토콜

무작위 기저 선택 → 송신/측정 → 기저 비교 → 오류율 확인

1단계: Alice – 비트와 기저 무작위 선택

2단계: Alice – 선택된 기저에 따라 광자 편광 상태로 인코딩하여 전송

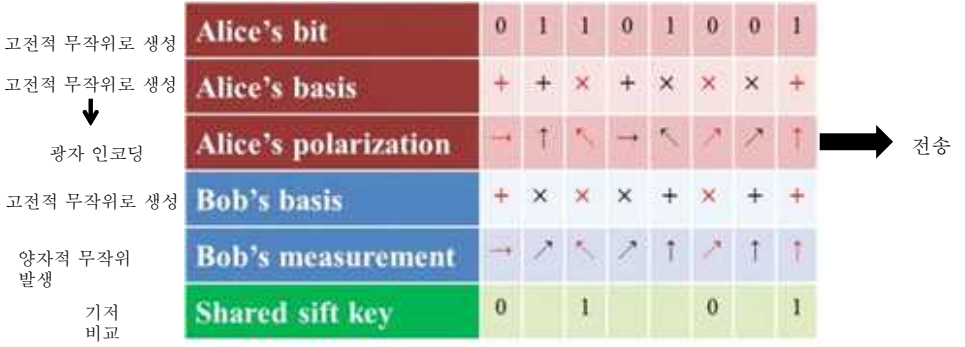
3단계: Bob – 무작위 기저로 측정

4단계: Alice&Bob – 공개 채널에서 사용한 기저 비교 –> 일치하는 경우만 키로 활용

5단계: 오류율(QBER) 공개 비교 (=도청 여부 판단)

6단계: 최종 암호키 생성

4. 절차: BB84 프로토콜



4. 절차: BB84 프로토콜

수학적 해석

1. 두 종류의 기저

- qubit

$$|0\rangle = \begin{pmatrix}1\\0\end{pmatrix},\ |1\rangle = \begin{pmatrix}0\\1\end{pmatrix}$$

- 정규 기저 (Z-basis)

$$B_Z = \{|0\rangle, |1\rangle\}$$

- 대각 기저 (X-basis)

$$|+\rangle = \frac{1}{\sqrt{2}}\begin{pmatrix}1\\1\end{pmatrix},\ |-\rangle = \frac{1}{\sqrt{2}}\begin{pmatrix}1\\-1\end{pmatrix}$$

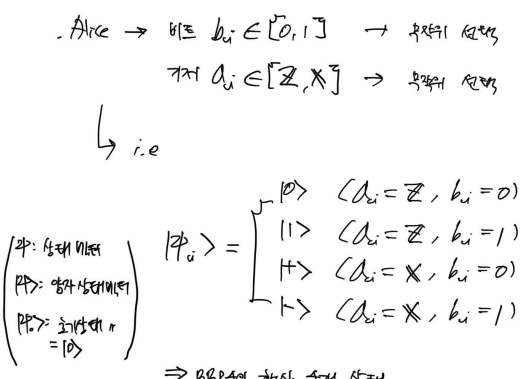
$$B_X = \{|+\rangle, |-\rangle\}$$

($|0\rangle$, $|1\rangle$의 중첩)

4. 절차: BB84 프로토콜

수학적 해석

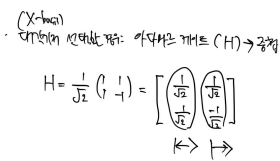
2. 앨리스의 비트 -> 양자상태 인코딩

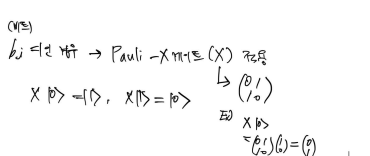


4. 절차: BB84 프로토콜

수학적 해석

2. 앨리스의 비트 -> 양자상태





4. 절차: BB84 프로토콜

수학적 해석

3. Bob의 측정 -> 양자상태
디코딩

- 큐비트 $|\psi\rangle$을 $\{|0\rangle, |1\rangle\}$ 기저 측정

→ 붕괴 → 고전적 비트 획득 → 고전비트 저장
(0 or 1)

- 측정 확률 (측정할 때 Alice와 일치할 확률)

$$P(m|\psi_i) = |\langle m|\psi_i\rangle|^2$$

(확률론적 해석)

Case 1: Bob 기저 = Alice 기저 → 확률 100%

$|\langle 0|0\rangle|^2 = 1, \quad |\langle +|+\rangle|^2 = 1$

Case 2: Bob 기저 ≠ Alice 기저 → 확률 50%

$|\langle +|0\rangle|^2 = \frac{1}{2}$

$|\langle -|0\rangle|^2 = \frac{1}{2}$

(-)

$\frac{1}{\sqrt{2}}(1\ \ 1)\begin{pmatrix}1\\0\end{pmatrix}$

$= \frac{1}{\sqrt{2}}\begin{bmatrix}1 & 1\\ 1 & -1\end{bmatrix}\begin{bmatrix}1\\0\end{bmatrix}$

$= \frac{1}{\sqrt{2}}(1+0) = \frac{1}{\sqrt{2}}$

$\frac{1}{\sqrt{2}}(1\ -1)\begin{pmatrix}1\\0\end{pmatrix}$

$= \frac{1}{\sqrt{2}}\begin{bmatrix}1 & -1\\ 1 & 1\end{bmatrix}\begin{bmatrix}1\\0\end{bmatrix} = \frac{1}{\sqrt{2}}(1+0) = \frac{1}{\sqrt{2}}$

4. 절차: BB84 프로토콜

수학적 해석

4. 기저 비교 -> 암호키 도출

- Alice & Bob : classical channel 에서 기저공개,

$a_i = b_i$ 인 비트만 선택 (a_i : Alice 기저, b_i : Bob 기저)

- 최종 키 : $k = [\text{비트 } i \mid a_i = b_i]$

5. 구현 실습 - Qiskit

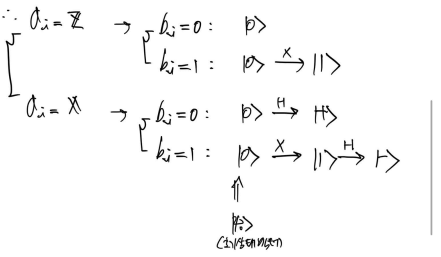
Qiskit:IBM에서 만든 파이썬 기반 양자컴퓨터 프로그래밍 프레임워크
(= 양자 알고리즘을 코딩하고 시뮬레이션하거나 실제 IBM 양자컴퓨터에서 실행할 수 있게 해주는 도구)

기본 문법

QuantumCircuit 클래스: Qiskit에서 양자 회로(Quantum Circuit)를 구축하는 데 사용되는 가장 기본적인 구성 요소.
- 회로 정의: 양자 비트(qubits)와 고전 비트(clbits)의 개수를 지정하여 회로 초기화
- 게이트 추가: 회로에 양자 게이트(ex: H(Hadamard), X(NOT), CNOT 등)와 측정(Measurement)을 추가하여 양자 연산 정의
- 프로그램 작성: 양자 컴퓨터에서 실행될 양자 프로그램의 설계도.
- ex: 2큐비트 회로를 만들고 첫 번째 큐비트에 H 게이트를 적용한 다음, CNOT 게이트를 적용하는 등의 작업 수행.
- QuantumCircuit(q, c) //문법- QuantumCircuit(1, 1) //1개의 큐비트와 1개의 고전 비트로 구성된 양자회로 생성

5. 구현 실습 - (2) 인코딩 함수

```python
def encode_message(bits, bases):
    message = []
    for i in range(n):
        qc = QuantumCircuit(1,1)
        if bases[i] == 0: # 직교 기저
            if bits[i] == 0:
                pass # 수평
            else:
                qc.x(0) # 수직
        else: # 대각 기저
            if bits[i] == 0:
                qc.h(0) # 대각
            else:
                qc.x(0)
                qc.h(0) # 반대각
        qc.barrier()
        message.append(qc)
    return message
```



5. 구현 실습 - (3) 디코딩(=측정) 함수

```
def decode_message(message, bases):
    decode_msg = []
    for q in range(n):
        if bases[q] == 0: # 직교 기저에서 측정
            message[q].measure(0,0)
        if bases[q] == 1: # 대각 기저에서 측정
            message[q].h(0)
            message[q].measure(0,0)   양자회로에 측정연산을 추가
        aer_sim = Aer.get_backend('aer_simulator')
        result = aer_sim.run(message[q], shots=1, memory=True).result()
        measured_bit = int(result.get_memory()[0])
        decode_msg.append(measured_bit)
    return decode_msg
```

파라미터: **message**(인코딩 함수의 반환값), **bases**(bob의 기저)
여기서 **message**의 type:
양자회로 객체(**Quantum Circuit**, 특정 양자상태를 정의하는 명령어 묶음)

Aer: **Qiskit**의 고성능 양자 시뮬레이터 백엔드를 제공하는 모듈.
실제 양자 하드웨어가 아닌 일반 컴퓨터 환경에서 양자 회로를 시뮬레이션하고 테스트하기 위해 설계됨.
실제 양자 컴퓨터에 회로를 보내지 않고도 결과 예측, 회로 디버깅이 가능.

5. 구현 실습 - (4) 암호키 생성 함수

```python
def generate_encryption_key(a_bases, b_bases,  bits):
    encryption_key = []
    for q in range(n):
        if a_bases[q] == b_bases[q]:
            # 같은 기저면, 암호키에 추가
            encryption_key.append(bits[q])
    return encryption_key
```

- Alice & Bob : classical channel 에서 기저공개,

  $a_i = b_i$ 인 비트만 저장 (a_i : Alice 기저, b_i : Bob 기저)

- 최종 키 : $k = [\text{비트 } i \mid a_i = b_i]$

6. 구현 실습 - 도청자(Eve) 유무

```
np.random.seed(seed=0)
n = 35
# Alice: 랜덤 비트 & 랜덤 기저 생성
alice_bits = randint(2, size=n)
alice_bases = randint(2, size=n)
print("Alice bases:", alice_bases)
# 특정 기저로 인코딩
message = encode_message(alice_bits, alice_bases)
# Bob: 랜덤 기저 생성
bob_bases = randint(2, size=n)
print("  Bob bases:", bob_bases)
# Bob: 랜덤 기저에 맞게 qubit 측정 -> message 읽기
bob_results = decode_message(message, bob_bases)
# Alice와 Bob의 암호키 생성
alice_key = generate_encryption_key(alice_bases, bob_bases, alice_bits)
print("Alice key:", alice_key)
bob_key = generate_encryption_key(alice_bases, bob_bases, bob_results)
print("  Bob key:", bob_key)
#암호키 비교
if alice_key == bob_key:
    print("전송 성공!")
else:
    print("도청 감지, 전송 실패!")
```

```
Alice bases: [1 0 1 1 0 0 1 0 1 1 1 1 1 0 1 0 1 1 1 1 0 1 0 0 1 1 0 1 0 1 0 0 0 0 0]
  Bob bases: [1 1 0 0 0 1 1 0 1 0 0 1 0 1 1 1 1 1 1 1 0 1 1 0 0 1 0 0 1 1 0 1 0 0 1 0]
Alice key: [np.int64(0), np.int64(1), np.int64(1), np.int64(1), np.int64(1), np.int64(1), np.int64(0), np.i
  Bob key: [0, 1, 1, 1, 1, 1, 0, 0, 0, 0, 0, 1, 1, 0, 0, 1, 1, 0, 0, 1, 1]
전송 성공!
```

```
np.random.seed(seed=4)
n = 35
# Alice: 랜덤 비트 & 랜덤 기저 생성
alice_bits = randint(2, size=n)
alice_bases = randint(2, size=n)
print("Alice bases:", alice_bases)
# 특정 기저로 인코딩
message = encode_message(alice_bits, alice_bases)

# 도청
eve_bases = randint(2, size=n)
eve_intercepted_msg = decode_message(message, eve_bases)
print("Intercepted message:", eve_intercepted_msg)

# Bob: 랜덤 기저 생성
bob_bases = randint(2, size=n)
print("  Bob bases:", bob_bases)
# Bob: 랜덤 기저에 맞게 qubit 측정 -> message 읽기
bob_results = decode_message(message, bob_bases)
# Alice와 Bob의 암호키 생성
alice_key = generate_encryption_key(alice_bases, bob_bases, alice_bits)
print("Alice key:", alice_key)
bob_key = generate_encryption_key(alice_bases, bob_bases, bob_results)
print("  Bob key:", bob_key)
#암호키 비교
if alice_key == bob_key:
    print("전송 성공!")
else:
    print("도청 감지, 전송 실패!")
```

```
Alice bases: [0 0 0 1 0 0 1 1 0 0 1 0 1 0 1 1 1 0 1 1 0 1 1 1 0 1 0 1 1 1 1]
Intercepted message: [0, 0, 0, 0, 1, 0, 1, 0, 0, 1, 1, 1, 0, 0, 1, 0, 0, 1, 0, 1, 0, 0, 0, 1, 1, 1, 1, 1, 0]
  Bob bases: [0 1 0 1 0 1 0 0 0 0 1 1 0 0 1 0 1 0 1 1 1 1 0 0 0 1 0 1 0 0 0 0 0 1 1 1]
Alice key: [np.int64(0), np.int64(1), np.int64(1), np.int64(1), np.int64(0), np.int64(0), np.int64(0), np.int64(1), np.int64(1), np
  Bob key: [0, 0, 1, 1, 1, 0, 0, 0, 1, 1, 0, 0, 1, 1, 0, 1, 0, 0]
도청 감지, 전송 실패!
```

7. 에러율(QBER) 계산

BB84에서는 앨리스(Alice)와 밥(Bob)이 서로 다른 기저(Z/X)를 랜덤하게 선택
같은 기저로 측정한 비트만 sifted key로 사용
도청자(Eve)나 채널 잡음이 있으면 측정 결과가 달라져 오류 발생

QBER (Quantum Bit Error Rate)= 오류 비트 비율
QBER = 잘못된 비트 수 / 총 비교한 비트 수

Eve가 잘못된 기저로 측정하면 밥의 결과와 불일치
Eve가 도청 시 평균적으로 전체 암호키의 25%가 오류 → QBER ≈ 25%
QBER가 일정 임계치 이상이면 채널 도청 감지 가능

오류가 낮으면 채널 안전
QBER > 임계치 → 도청 가능성 → 키 폐기
BB84 안전성의 핵심 지표

감사합니다.