최적화문제연구

Microsoft Q#(Quantum Language)을
이용한 암호구현

정보시스템공학과

18211501 안규황

HSU
HANSUNG
UNIVERSITY

양자 컴퓨터란?

- 양자역학에서는 양자얽힘, 중첩, 텔레포테이션 등의 효과를 이용해 계산하는 컴퓨터를 양자 컴퓨터라 칭함

- 기존 컴퓨터가 0과 1만 구분할 수 있는 반면, 양자 컴퓨터는 0과 1을 동시에 공존 시킬 수 있음

- 양자 컴퓨터가 실제로 나온다면, 현존 최고의 슈퍼 컴퓨터가 수백 년이 걸려도 풀기 힘든 문제도 단 몇 초 이내로 풀 수 있을 것이라 예상

HSU
HANSUNG
UNIVERSITY

Issue

Microsoft Q# 이란?

- 2017년 12월 11일 Microsoft에서 양자 개발 킷을 공개 함
  →Quantum Development Kit[1]
- Q#을 이용한다면 양자 컴퓨터가 아닌 일반 컴퓨터에서 양자 개발 할 수 있음
- 양자 컴퓨터는 코어 프레서서를 사용하고 GPU 혹은 FPGA를 프로그래밍 한 다음 CPU에서 가속도를 함
  →Q#은 이와 비슷하게 디자인 되었음
- Windows, linux, macOS 모두 제공하지만, windows에서 설치 실패하여 linux에서 실험

HSU HANSUNG UNIVERSITY

[1] https://www.microsoft.com/en-us/quantum/development-kit

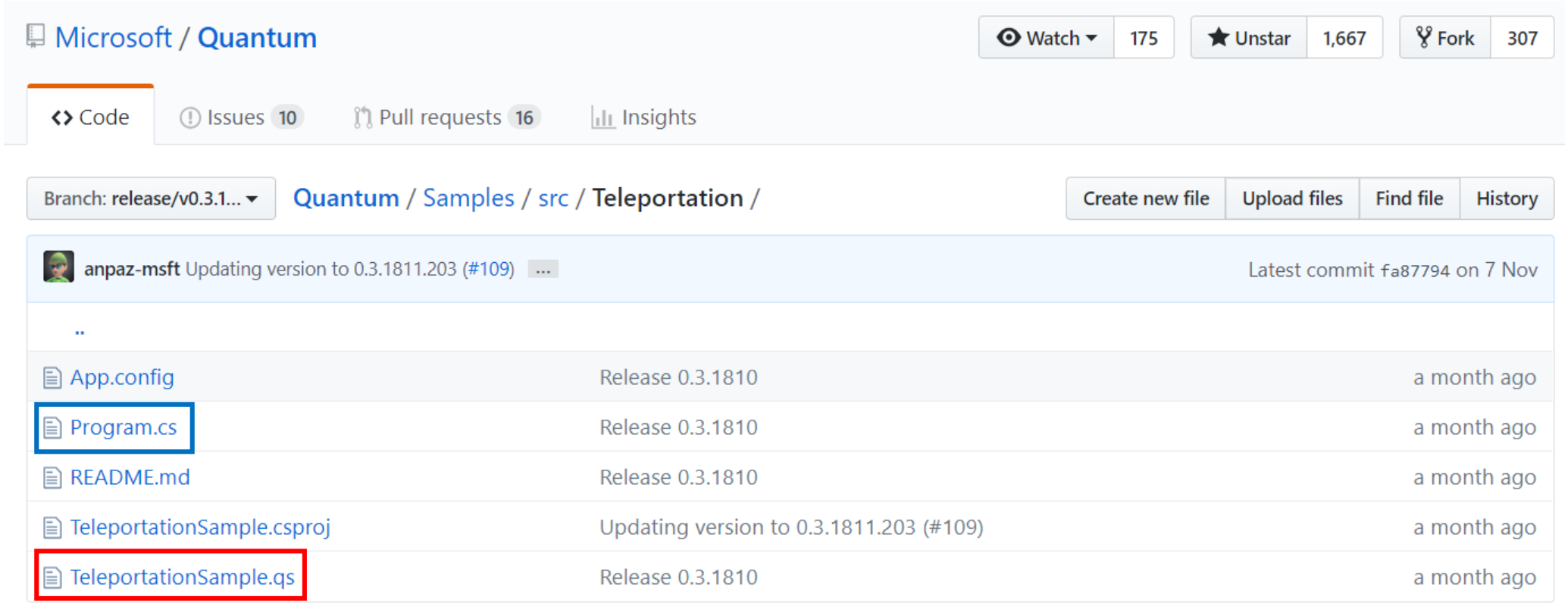
Qubit 란?

기존 프로그래밍에서는 'a'라는 변수를 선언하면, 해당 값에는 '0' 혹은 '1'로만 선언할 수 있음
그러나 Qubit의 경우 'a'라는 변수 안에 '0'과 '1'이 모두 들어있음

HSU
HANSUNG
UNIVERSITY

프로젝트 생성[2]

'.cs': 양자 언어로 작성된 것을 C#을 이용해 표현할 수 있도록 도와줌
'.qs': 양자 언어 Q#을 의미



HSU HANSUNG UNIVERSITY

[2] https://github.com/Microsoft/Quantum/tree/release/v0.3.1810/Samples/src/Teleportation

Program.cs

```
using Microsoft.Quantum.Simulation.Simulators;
using System.Linq;

namespace Microsoft.Quantum.Examples.Teleportation {
    class Program
    {
        static void Main(string[] args)
        {
            using (var sim = new QuantumSimulator())
            {
                var rand = new System.Random();

                foreach (var idxRun in Enumerable.Range(0, 8))
                {
                    var sent = rand.Next(2) == 0;
                    var received = TeleportClassicalMessage.Run(sim, sent).Result;
                    System.Console.WriteLine($"Round {idxRun}:\tSent {sent},\tgot {received}.");
                    System.Console.WriteLine(sent == received ? "Teleportation successful!!\n" : "\n");
                }
            }

            System.Console.WriteLine("\n\nPress Enter to continue...\n\n");
            System.Console.ReadLine();

        }
    }
}
```

HANSUNG UNIVERSITY

TeleportationSample.qs

```
operation TeleportClassicalMessage (message : Bool) : Bool {

    mutable measurement = false;

    using (register = Qubit[2]) {

        // Ask for some qubits that we can use to teleport.
        let msg = register[0];
        let there = register[1];

        // Encode the message we want to send.
        if (message) {
            X(msg);
        }

        // Use the operation we defined above.
        Teleport(msg, there);

        // Check what message was sent.
        if (M(there) == One) {
            set measurement = true;
        }

        // Reset all of the qubits that we used before releasing
        // them.
        ResetAll(register);
    }

    return measurement;
}
```

```
operation Teleport (msg : Qubit, there : Qubit) : Unit {

    using (register = Qubit[1]) {

        // Ask for an auxillary qubit that we can use to prepare
        // for teleportation.
        let here = register[0];

        // Create some entanglement that we can use to send our message.
        H(here);
        CNOT(here, there);

        // Move our message into the entangled pair.
        CNOT(msg, here);
        H(msg);

        // Measure out the entanglement.
        if (M(msg) == One) {
            Z(there);
        }

        if (M(here) == One) {
            X(there);
        }

        // Reset our "here" qubit before releasing it.
        Reset(here);
    }
}
```

HSU HANSUNG UNIVERSITY

.qs – 정의(함수 및 변수 선언)

```
operation Teleport (msg : Qubit, there : Qubit) : Unit {

    using (register = Qubit[1]) {

        // Ask for an auxillary qubit that we can use to prepare
        // for teleportation.
        let here = register[0];

        // Create some entanglement that we can use to send our message.
        H(here);
        CNOT(here, there);

        // Measure out the entanglement.
        if (M(msg) == One) {
            Z(there);
        }
```

operation은 기존의 function을 의미

let 및 if 명령 등을 사용하여 작업을 정의하고 호출.

cf) let의 경우 변수를 선언

HSU HANSUNG UNIVERSITY

TeleportationSample.qs

```
operation TeleportClassicalMessage (message : Bool) : Bool {

    mutable measurement = false;

    using (register = Qubit[2]) {

        // Ask for some qubits that we can use to teleport.
        let msg = register[0];
        let there = register[1];

        // Encode the message we want to send.
        if (message) {
            X(msg);
        }

        // Use the operation we defined above.
        Teleport(msg, there);

        // Check what message was sent.
        if (M(there) == One) {
            set measurement = true;
        }

        // Reset all of the qubits that we used before releasing
        // them.
        ResetAll(register);
    }

    return measurement;
}
```

```
operation Teleport (msg : Qubit, there : Qubit) : Unit {

    using (register = Qubit[1]) {

        // Ask for an auxillary qubit that we can use to prepare
        // for teleportation.
        let here = register[0];

        // Create some entanglement that we can use to send our message.
        H(here);
        CNOT(here, there);

        // Move our message into the entangled pair.
        CNOT(msg, here);
        H(msg);

        // Measure out the entanglement.
        if (M(msg) == One) {
            Z(there);
        }

        if (M(here) == One) {
            X(there);
        }

        // Reset our "here" qubit before releasing it.
        Reset(here);
    }
}
```

HSU HANSUNG UNIVERSITY

.qs - teleportationsaple 사용법

```
using (register = Qubit[2]) {
    // Ask for some qubits that we can use to teleport.
    let msg = register[0];
    let there = register[1];
```

using 명령어를 이용하여 2개의 Qubit을 상태 '0'으로 초기화

각 qubit을 특정 변수에 할당

```
// Encode the message we want to send.
if (message) { X(msg); }
```

'X'는 고전 논리에서 NOT과 유사

만약 'message == 1' 이라면 'X'를 이용해 'msg' qubit의 값은 뒤집어짐

HSU HANSUNG UNIVERSITY

TeleportationSample.qs

```
operation TeleportClassicalMessage (message : Bool) : Bool {

    mutable measurement = false;

    using (register = Qubit[2]) {

        // Ask for some qubits that we can use to teleport.
        let msg = register[0];
        let there = register[1];

        // Encode the message we want to send.
        if (message) {
            X(msg);
        }

        // Use the operation we defined above.
        Teleport(msg, there);

        // Check what message was sent.
        if (M(there) == One) {
            set measurement = true;
        }

        // Reset all of the qubits that we used before releasing
        // them.
        ResetAll(register);
    }

    return measurement;
}
```

```
operation Teleport (msg : Qubit, there : Qubit) : Unit {

    using (register = Qubit[1]) {

        // Ask for an auxillary qubit that we can use to prepare
        // for teleportation.
        let here = register[0];

        // Create some entanglement that we can use to send our message.
        H(here);
        CNOT(here, there);

        // Move our message into the entangled pair.
        CNOT(msg, here);
        H(msg);

        // Measure out the entanglement.
        if (M(msg) == One) {
            Z(there);
        }

        if (M(here) == One) {
            X(there);
        }

        // Reset our "here" qubit before releasing it.
        Reset(here);
    }
}
```

HSU HANSUNG UNIVERSITY

.qs - Teleport 사용법

```
operation Teleport(msg : Qubit, there : Qubit) : () {
    body {

        using (register = Qubit[1]) {
            // Ask for an auxillary qubit that we can use to prepare
            // for teleportation.
            let here = register[0];
```

하나의 qubit을 추가적으로 호출
here은 '0'으로 초기화 됨

```
// Create some entanglement that we can use to send our message.
H(here);
CNOT(here, there);
```

H: Hadamard gate
here qubit을 중첩 상태 H로 변경
-> 고전 프로그래밍과 다르게 양자 프로그래밍은 값의 중첩이 가능

HSU HANSUNG UNIVERSITY

TeleportationSample.qs

```
operation TeleportClassicalMessage (message : Bool) : Bool {

    mutable measurement = false;

    using (register = Qubit[2]) {

        // Ask for some qubits that we can use to teleport.
        let msg = register[0];
        let there = register[1];

        // Encode the message we want to send.
        if (message) {
            X(msg);
        }

        // Use the operation we defined above.
        Teleport(msg, there);

        // Check what message was sent.
        if (M(there) == One) {
            set measurement = true;
        }

        // Reset all of the qubits that we used before releasing
        // them.
        ResetAll(register);
    }

    return measurement;
}
```

```
operation Teleport (msg : Qubit, there : Qubit) : Unit {

    using (register = Qubit[1]) {

        // Ask for an auxillary qubit that we can use to prepare
        // for teleportation.
        let here = register[0];

        // Create some entanglement that we can use to send our message.
        H(here);
        CNOT(here, there);

        // Move our message into the entangled pair.
        CNOT(msg, here);
        H(msg);

        // Measure out the entanglement.
        if (M(msg) == One) {
            Z(there);
        }

        if (M(here) == One) {
            X(there);
        }

        // Reset our "here" qubit before releasing it.
        Reset(here);
    }
}
```

HSU HANSUNG UNIVERSITY

.qs - Teleport 사용법

```
CNOT(here, there);
```

CNOT(Control NOT)

만약 here qubit이 '1'일 경우 CNOT은 there의 값을 뒤집음, 0 -> 1 or 1 -> 0
만약 here qubit이 '0'일 경우 CNOT은 아무것도 하지 않음

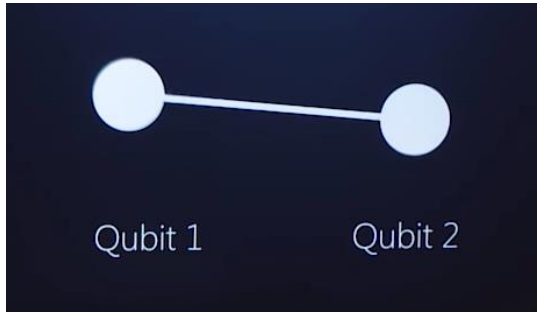
따라서, CNOT은 XOR 게이트와 같음

HSU HANSUNG UNIVERSITY

.qs - Teleport 사용법

```
H(here);
CNOT(here, there);
```

해당 두 줄은 qubit들 사이의 얽힘을 만들어 줌
두 가지 qubit가 일단 얽히게 되면, 비트는
왼쪽 아래와 같이 본질적으로 상관되게 됨



만약 Qubit 1의 상태를 바꾼다면,
Qubit 2의 상태 역시 즉시 바뀌게 됨

HSU HANSUNG UNIVERSITY

TeleportationSample.qs

```
operation TeleportClassicalMessage (message : Bool) : Bool {

    mutable measurement = false;

    using (register = Qubit[2]) {

        // Ask for some qubits that we can use to teleport.
        let msg = register[0];
        let there = register[1];

        // Encode the message we want to send.
        if (message) {
            X(msg);
        }

        // Use the operation we defined above.
        Teleport(msg, there);

        // Check what message was sent.
        if (M(there) == One) {
            set measurement = true;
        }

        // Reset all of the qubits that we used before releasing
        // them.
        ResetAll(register);
    }

    return measurement;
}
```

```
operation Teleport (msg : Qubit, there : Qubit) : Unit {

    using (register = Qubit[1]) {

        // Ask for an auxillary qubit that we can use to prepare
        // for teleportation.
        let here = register[0];

        // Create some entanglement that we can use to send our message.
        H(here);
        CNOT(here, there);

        // Move our message into the entangled pair.
        CNOT(msg, here);
        H(msg);

        // Measure out the entanglement.
        if (M(msg) == One) {
            Z(there);
        }

        if (M(here) == One) {
            X(there);
        }

        // Reset our "here" qubit before releasing it.
        Reset(here);
    }
}
```

HSU HANSUNG UNIVERSITY

.qs - Teleport 사용법

```
// Move our message into the entangled pair.
CNOT(msg, here);
H(msg);
```

Qubit msg에 다른 CNOT을 사용하여 페어링한 다음 H를 적용
해당 msg 값을 결론 짓기 위해선, 얽힘 값을 측정해야함

HSU HANSUNG UNIVERSITY

TeleportationSample.qs

```
operation TeleportClassicalMessage (message : Bool) : Bool {

    mutable measurement = false;

    using (register = Qubit[2]) {

        // Ask for some qubits that we can use to teleport.
        let msg = register[0];
        let there = register[1];

        // Encode the message we want to send.
        if (message) {
            X(msg);
        }

        // Use the operation we defined above.
        Teleport(msg, there);

        // Check what message was sent.
        if (M(there) == One) {
            set measurement = true;
        }

        // Reset all of the qubits that we used before releasing
        // them.
        ResetAll(register);
    }

    return measurement;
}
```

```
operation Teleport (msg : Qubit, there : Qubit) : Unit {

    using (register = Qubit[1]) {

        // Ask for an auxillary qubit that we can use to prepare
        // for teleportation.
        let here = register[0];

        // Create some entanglement that we can use to send our message.
        H(here);
        CNOT(here, there);

        // Move our message into the entangled pair.
        CNOT(msg, here);
        H(msg);

        // Measure out the entanglement.
        if (M(msg) == One) {
            Z(there);
        }

        if (M(here) == One) {
            X(there);
        }

        // Reset our "here" qubit before releasing it.
        Reset(here);
    }
}
```

HSU HANSUNG UNIVERSITY

.qs - Teleport 사용법

```
// Measure out the entanglement.
if (M(msg) == One)  { Z(there); }
if (M(here) == One) { X(there); }
```

- 'M' 연산자를 이용하여 msg와 here의 값을 확인할 수 있음

- 양자 알고리즘의 마지막에 'M' 연산을 수행할 경우 양자 정보를 고전적 상태로 투사하는 값을 측정하고 측정 정보를 읽을 수 있음

- 측정 출력을 조건부로 사용한다.
양자 이동 'Z'를 적용한 다음 'X'를 이용해 there 값을 반전

HSU HANSUNG UNIVERSITY

.qs - teleportationsaple 사용법

```
operation TeleportClassicalMessage(message : Bool) : Bool {
    body {
        mutable measurement = false;

        using (register = Qubit[2]) {
            // Ask for some qubits that we can use to teleport.
            let msg = register[0];
            let there = register[1];

            // Encode the message we want to send.
            if (message) { X(msg); }

            // Use the operation we defined above.
            Teleport(msg, there);

            // Check what message was sent.
            if (M(there) == One) { set measurement = true; }

            // Reset all of the qubits that we used before releasing
            // them.
            ResetAll(register);
        }

        return measurement;
    }
```

TeleportClassicalMessage에서 qubit를 측정하여 메시지가 있는지 확인

msg가 qubit에 실제로 텔레포팅 되었는지 확인할 수 있음

HSU HANSUNG UNIVERSITY

실행 결과

```
Round 0:        Sent True,      got True.
Teleportation successful!!

Round 1:        Sent True,      got True.
Teleportation successful!!

Round 2:        Sent True,      got True.
Teleportation successful!!

Round 3:        Sent False,     got False.
Teleportation successful!!

Round 4:        Sent False,     got False.
Teleportation successful!!

Round 5:        Sent False,     got False.
Teleportation successful!!

Round 6:        Sent True,      got True.
Teleportation successful!!

Round 7:        Sent False,     got False.
Teleportation successful!!
```

메모리 사용량

- 양자 컴퓨터가 아닌 컴퓨터에서 양자 프로그래밍을 돌리기 위해 약 30개의 시뮬레이션 된 qubit이 사용 됨

- 하나의 시뮬레이션으로 30 qubit을 사용하려면, 16GB RAM 소요
  -> 31 qubit을 사용하려면 2배의 메모리, 32GB RAM 필요
  -> 20 qubit을 사용하려면 ½배의 메모리, 8GB RAM 필요
  -> 40 qubit을 사용하려면 30배의 메모리, 16TB RAM 필요

Quantum Language Data Type

Type	Default
Int	0
Double	0.0
Bool	False
String	""
Qubit	Invalide qubit
Pauli	PauliI
Result	Zero
Range	The empty range, 1..1..0
Callable	Invalid callable
Array['T]	'T[0]

HSU HANSUNG UNIVERSITY

Quantum Language Operators

Operator	Description	Operand Types
!	Unwrap	Any user-defined type
-, ~~~, not	Numeric negative, bitwise complement, logical negation	Int, Double for – Int for ~~~ Bool for not
^	Integer power	Int
/ , * , %	Division, Multiplication, Integer modulus	Int, Double for / and * Int for %
+, -	Addition or string and array concatenation, subtraction	Int or Double, additionally String or any array type for +
<<<, >>>	Left shift, right shift	Int
<, <=, >, >=	Less-than, less-than-or-equal, greater-than, greater-than-or-equal comparisons	Int or Double
==, !=	Equal, not-equal comparisons	Any primitive type
&&&	Bitwise AND	Int
^^^	Bitwise XOR	Int
&&	Logical AND	Bool

HSU HANSUNG UNIVERSITY

초 경량 블록암호 CHAM 구현_64x128

```
function ROL_64_128 (k : Int, x : Int) : (Int)
{
    return ((k <<< x) ||| (k >>> 16-x)) % 0x10000;
}

function ROL1_64_128 (k : Int) : (Int)
{
    return ((k <<< 1) ||| (k >>> 15)) % 0x10000;
}

function ROL8_64_128 (k : Int) : (Int)
{
    return ((k <<< 8) ||| (k >>> 8)) % 0x10000;
}

function ROR_64_128 (k : Int, x : Int) : (Int)
{
    return ((k >>> x) ||| ((k <<< 16-x))) % 0x10000;
}

function KeyExpansion_64_128(k : Int[]) : (Int[])
{
    mutable arr = new Int[16];

    for (i in 0..7){
        //Message(ToStringI(i));
        set arr[i] = k[i] ^^^ ROL_64_128(k[i], 1) ^^^ ROL_64_128(k[i], 8);
        set arr[(i+8) ^^^ 0x0001] = k[i] ^^^ ROL_64_128(k[i], 1) ^^^ ROL_64_128(k[i], 11);
        // Message(ToStringI(arr[(i+8) ^^^ 0x0001]));
        // Message(ToStringI((i+8) ^^^ 0x0001));
    }

    return arr;
}
```

```
function Encryption_64_128(pt : Int[], rk : Int[]) : (Int[]){
    mutable tmp = 0;
    mutable local_pt = new Int[4];
    mutable ct = new Int[4];

    for (i in 0..3){
        set local_pt[i] = pt[i];
    }

    for (i in 0..19){
        // Message(ToStringI(i));
        set tmp = ROL_64_128(local_pt[1], 1) ^^^ rk[(4*i) % 16];
        set ct[0] = ((local_pt[0] ^^^ (4*i)) + tmp) % 0x10000;
        set ct[0] = ROL_64_128(ct[0], 8);
        // Message(ToStringI(ct[0]));

        set tmp = ROL_64_128(local_pt[2], 8) ^^^ rk[(4*i+1) % 16];
        set ct[1] = ((local_pt[1] ^^^ (4*i+1)) + tmp) % 0x10000;
        set ct[1] = ROL_64_128(ct[1], 1);
        // Message(ToStringI(ct[1]));

        set tmp = ROL_64_128(local_pt[3], 1) ^^^ rk[(4*i+2) % 16];
        set ct[2] = ((local_pt[2] ^^^ (4*i+2)) + tmp) % 0x10000;
        set ct[2] = ROL_64_128(ct[2], 8);
        // Message(ToStringI(ct[2]));

        set tmp = ROL_64_128(ct[0], 8) ^^^ rk[(4*i+3) % 16];
        set ct[3] = ((local_pt[3] ^^^ (4*i+3)) + tmp) % 0x10000;
        set ct[3] = ROL_64_128(ct[3], 1);
        // Message(ToStringI(ct[3]));

        set local_pt[0] = ct[0];
        set local_pt[1] = ct[1];
        set local_pt[2] = ct[2];
        set local_pt[3] = ct[3];
    }

    return ct;
}
```

HSU HANSUNG UNIVERSITY

추후 연구사항

- '.qs'에 필요 함수(AES의 경우 subByte, shiftRows, addRoundkey, mixColumns)를 정의하고 결과 값을 '.cs'에서 확인해야 하는 로직은 이해했음

- 초 경량 국산 블록암호인 CHAM[3]에 qubit을 적용하고자 함
  → 이를 논문으로…

- 이때 사용되는 qubit의 양에 따라 어떻게 차이가 날지 궁금

HSU HANSUNG UNIVERSITY

[3] https://github.com/kyu-h/QS_CHAM

참고 자료

[1] Microsoft “Quantum Development Kit” Available at https://www.microsoft.com/en-us/quantum/development-kit

[2] Microsoft “Q# Sample Code” Available at https://github.com/Microsoft/Quantum/tree/release/v0.3.1810/Samples/src/Teleportation

[3] Github Kyu-h “CHAM with Q#” Available at https://github.com/kyu-h/QS_CHAM

HSU HANSUNG UNIVERSITY

Q & A

Thank You!

HSU
HANSUNG
UNIVERSITY