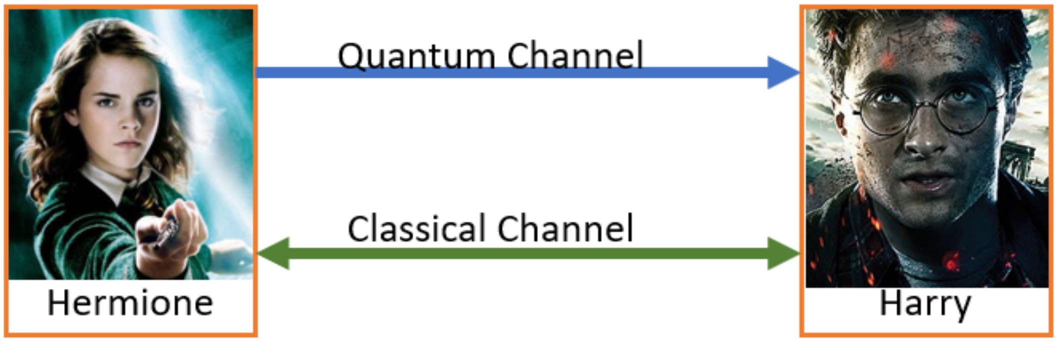
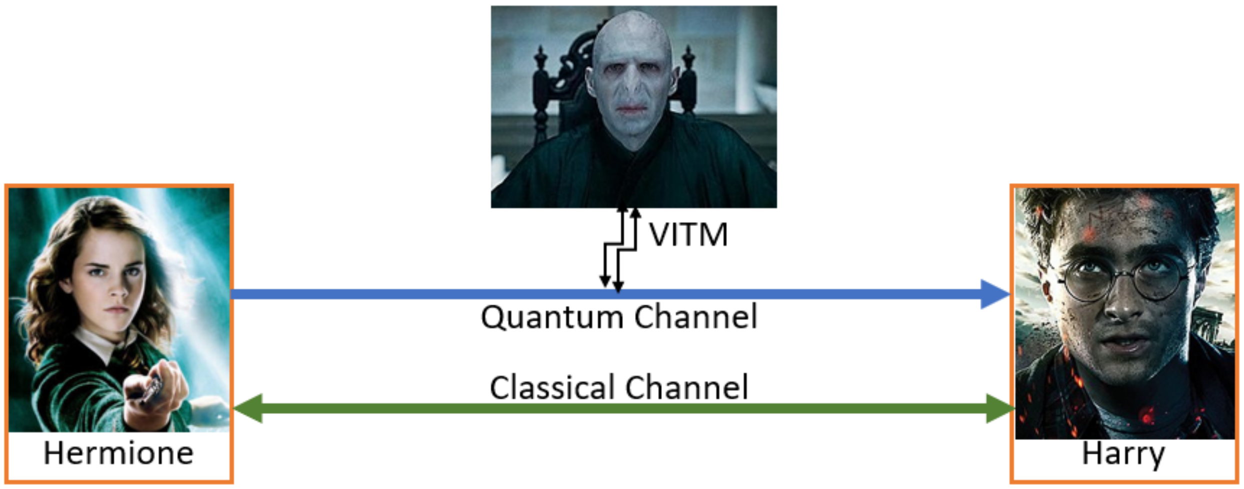
Learning Outcome: Students will gain an experience in implementing a quantum key distribution protocol.

Overview



Hermione and Harry create two connections: one quantum channel and one bidirectional classical channel. In this lab, you will implement the BB84 protocol by preparing and measuring a sequence of qubits and then perform classical operations to transform the measurement results to a sifted key.



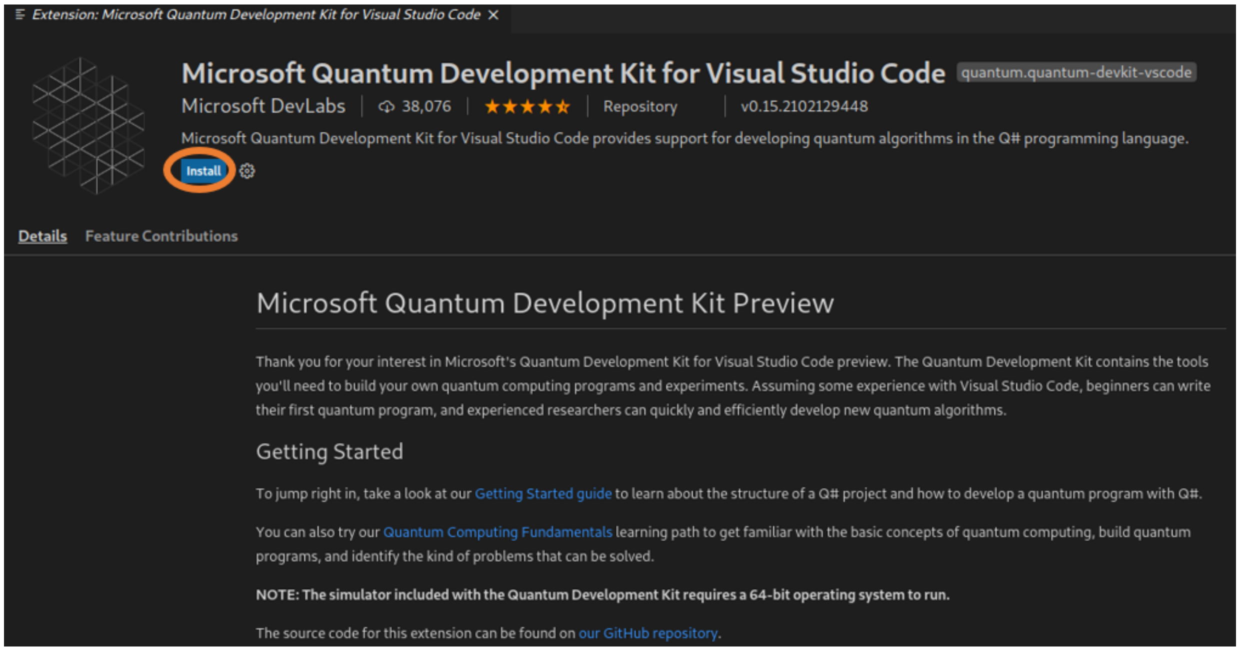
You will then replicate Tom Riddle's attempt to conduct VITM (Voldemort-in-the-middle!) against the quantum channel and observe the impact on the sifted key.

Setup

You'll need a VM running Microsoft Visual Studio Code, as in:
(NOTE: I recommend doing apt update after each install; you may even want to throw apt upgrade in there, but be prepared to wait a while if you're on RLES)

```
$ sudo apt update


$ sudo apt install software-properties-common apt-transport-https


$ wget -qO- https://packages.microsoft.com/keys/microsoft.asc | gpg --dearmor
> packages.microsoft.gpg


$ sudo install -o root -g root -m 644 packages.microsoft.gpg
/etc/apt/trusted.gpg.d/


$ sudo sh -c 'echo "deb [arch=amd64 signed-
by=/etc/apt/trusted.gpg.d/packages.microsoft.gpg]
https://packages.microsoft.com/repos/vscode stable main" >
/etc/apt/sources.list.d/vscode.list'


$ sudo apt install code


$ wget https://packages.microsoft.com/config/ubuntu/20.10/packages-microsoft-
prod.deb -O packages-microsoft-prod.deb


$ sudo dpkg -i packages-microsoft-prod.deb


$ sudo apt install -y apt-transport-https


$ sudo apt install -y dotnet-sdk-3.1


$ sudo apt install -y dotnet-sdk-5.0


$ sudo apt update
```

Once VSCode & the SDK are installed, open it on your VM and manually add the Microsoft Quantum Development Kit


Extension: Microsoft Quantum Development Kit for Visual Studio Code
Install
Details
Feature Contributions

Microsoft Quantum Development Kit for Visual Studio Code quantum.quantum-devkit-vscode

Microsoft DevLabs | 38,076 | ★★★★★ | Repository | v0.15.2102129448

Microsoft Quantum Development Kit for Visual Studio Code provides support for developing quantum algorithms in the Q# programming language.

Microsoft Quantum Development Kit Preview

Thank you for your interest in Microsoft's Quantum Development Kit for Visual Studio Code preview. The Quantum Development Kit contains the tools you'll need to build your own quantum computing programs and experiments. Assuming some experience with Visual Studio Code, beginners can write their first quantum program, and experienced researchers can quickly and efficiently develop new quantum algorithms.

Getting Started

To jump right in, take a look at our Getting Started guide to learn about the structure of a Q# project and how to develop a quantum program with Q#.

You can also try our Quantum Computing Fundamentals learning path to get familiar with the basic concepts of quantum computing, build quantum programs, and identify the kind of problems that can be solved.

NOTE: The simulator included with the Quantum Development Kit requires a 64-bit operating system to run.

The source code for this extension can be found on our GitHub repository.

Once MSQDK is installed, you'll need to reload VSCode & may need to install any necessary updates. You may also want to install the Python extension (or your preferred classical programming language) within VSCode. This is not required but you may find it easier than learning Q# for routine classical things like building an array of randomized bools. You'll just need to call the Q# code from within Python for the quantum-specific operations. Regardless, you can build a standalone Q# project using these steps:

To create a new project:

1. Click **View** -> **Command Palette** and select **Q#: Create New Project**.
2. Click **Standalone console application**.
3. Navigate to the location to save the project. Enter the project name and click **Create Project**.
4. When the project is successfully created, click **Open new project...** in the lower right.

Inspect the project. You should see a source file named `Program.qs`, which is a Q# program that defines a simple operation to print a message to the console. You'll be able to do the same thing to build a hello world program in Python (for those new to VSCode).

To run the application:

1. Click **Terminal** -> **New Terminal**.
2. At the terminal prompt, enter `dotnet run`.

3. You should see the following text in the output window `Hello quantum world!`

You can use that base file (`Program.qs`) to complete the steps in this lab, but you'll need to add a few libraries to the namespace:

```
    open Microsoft.Quantum.Canon;
    open Microsoft.Quantum.Intrinsic;
    open Microsoft.Quantum.Arrays;
    open Microsoft.Quantum.Measurement;
    open Microsoft.Quantum.Canon;
    open Microsoft.Quantum.Diagnostics;
    open Microsoft.Quantum.Intrinsic;
    open Microsoft.Quantum.Convert;
    open Microsoft.Quantum.Math;
    open Microsoft.Quantum.Random;
```

Some background on building BB84 using Q#

The first step for BB84 requires Hermione to prep the qubits and send them to Harry for measurement. The second step is using a classical channel to communicate basis selections.

In this example, Hermione can choose between two basis choices: horizontal and diagonal. She can also choose which bit value she wants to encode in each qubit. She'll then encode the qubit along that basis.

Hermione prepares the states along the horizontal basis, where l0⟩ represents the key bit value `0` and l1⟩ represents the key bit value `1`. She is also able to prepare states along the diagonal (or Hadamard) basis, where l+⟩ represents the key bit value `0` along the diagonal basis and l-⟩ represents the key bit value `1` along the diagonal basis.

You can map the diagonal basis in Q# using the following conversion (given N qubits, stored in the array qs[i]):

//if qs[i] was in state |0⟩, it should become |+⟩, where |+⟩ = (|0⟩ + |1⟩) / sqrt(2), and

//if qs[i] was in state |1⟩, it should become |-⟩, where |-⟩ = (|0⟩ - |1⟩) / sqrt(2).

```
   operation DiagonalBasis (qs : Qubit[]) : Unit {
      ApplyToEachA(H, qs); // this applies the Hadamard gate to each element of qs[]
   }
```

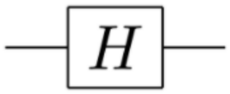
You can also simulate the preparation of the qubit into a superposition by adding relative phase to one of the states, as in converting it from the l0⟩ state to the |+⟩ state, by applying the **Hadamard gate**:

```
  H(q); //this applies the Hadamard gate on a single qubit, ‘q’
```

Note: The preparation of the qubit in superposition is conceptually equivalent to rotating the basis along the diagonal. The result of the Hadamard Gate phase shift to the diagonal basis is a measurement along the horizontal basis that is 50% likely to be |0⟩ and 50% likely to be |1⟩.
This is represented by the matrix:

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

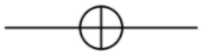
and the circuit:



There is one more critical gate in Quantum computing that you will need for this lab: the Pauli-**X gate**. This is conceptually equivalent to the NOT gate in classical computing, relative to the standard/horizontal basis.
This is represented by the matrix:

$$X = \begin{bmatrix} 0 & 1 \\ 1 & 0 \end{bmatrix}.$$

and the circuit:



You can implement the X Gate in Q# using:

```
  X(qs[i]); //where qs[] is the qubit array
```

Finally, you can measure a qubit in Q# along any of the three main axes (Z, X, Y) using:

```
  Measure([PauliZ], [q]); // where PauliZ is the standard/horizontal basis and q is a qubit ptr
```

You can see more Q# programming tips in the docs page, here: https://docs.microsoft.com/en-us/azure/quantum/user-guide/

For this lab, you will need to:

1) Prepare the qubits (Hermione's operation selects a random bit value and a random basis for encoding)–20%
2) Receive qubits (Harry's operation selects random basis and measures the qubit)–20%
3) Calculate the sifted key and use it to encrypt or encode a "hello quantum world" message (Hermione & Harry communicate via classical channel, exchange basis values, and discard the measurements where the bases did not match, encrypt/exchange/decrypt a "hello quantum world" message between Hermione & Harry)–20%
4) Introduce an evesdropper (Tom Riddle randomly selects a basis and measures the qubit before sending the qubit along to Harry). Your TomRiddle operation should log/output the basis & measurements. NOTE: If TomRiddle were to share basis choices with Hermione or Harry, they would effectively form a sifted key between TomRiddle & the other party. That said, VITM is meant to stay hidden, so TomRiddle would just relay the qubits without telling Hermione/Harry–20%
5) Detect the VITM attack (the sifted key does not allow encryption/decryption). When you catch the sifted key mismatch, your program should message a counterspell from defense against the dark arts (ie. "*Expelliarmis*!". You can just use that one or find an alternative favorite from the Hogwarts curriculum ref: https://harrypotter.fandom.com/wiki/Defence_Against_the_Dark_Arts#Spells)–20%

NOTE: You should use a single Q# program on a single VM to conduct these steps. Each of the steps could be one or more operations (callable subroutines) within the main program. This will allow for the preservation of qubit states as they are "passed along" from Hermione to Harry (and successfully reveal a VITM attack).

Signoff Page

1) Prepare the qubits. Hermione's operation selects a random bit value and a random basis for encoding. 20%

____

2) Receive the qubits. Harry's operation selects a random basis and measures the qubit stream sent by Hermione. 20%

____

3) Calculate the sifted key and use it to encrypt or encode a "hello quantum world" message. Hermione & Harry communicate via classical channel, exchange basis values, and discard the measurements where the bases did not match, encrypt/exchange/decrypt a "hello quantum world" message between Hermione & Harry. 20%

____

4) Introduce an evesdropper. Tom Riddle randomly selects a basis and measures the qubit before sending the qubit along again to Harry. 20%

____

5) Detect the VITM attack. Detect sifted key failure to encrypt/decrypt & Message(your_favorite_spell_from_*Defense_Against_the_Dark_Arts*). 20%

____