

The Incomplete Technical Documents on Pedagogical QKD Implementation

Experimental Session

QUANTUM CAMP 2018

May 18, 2018

Dedicated to the people that make this happens.

Contents

1	Preliminaries	3
1.1	About the computer	3
1.1.1	Arduino	3
1.1.2	Linux Terminal	7
1.1.3	Python	8
1.1.4	Bit, Bytes, Hexadecimals and ASCII	11
1.2	About the electronics	14
1.2.1	Precaution	14
1.2.2	Arduino UNO Board	14
1.2.3	Breadboard	17
1.2.4	Resistor	18
1.2.5	Transistor	19
1.2.6	IR LED	19
1.2.7	Stepper motor	22
1.3	About laser and polarisation	24
1.3.1	Precaution	24
1.3.2	Laser	24
1.3.3	Photodiode	25
1.3.4	Polarisation	25
1.3.5	Some background story	25
2	Classical Channel	27
2.1	IR signals	27
2.1.1	Electrical connections	27
2.1.2	From NEC to the protocol	29
2.2	Programs	30
2.2.1	Arduino program	30
2.2.2	BinaryComm package	31
2.2.3	Python programs	31
3	Quantum Channel	33
3.1	Laser signal	33
3.1.1	General schematics	33

3.1.2	In the sequence	35
3.1.3	Synchronisation	35
3.2	Key generation	35
3.2.1	Random numbers	36
3.2.2	Representation	36
3.2.3	BB84 QKD scheme	36
3.3	Implementation	37
3.3.1	Polarisation basis alignment	37
3.3.2	Light generation and measurement	39
3.3.3	Secure key generation	39
3.4	Programs	39
3.4.1	Arduino programs	39
3.4.2	Python programs	41
4	Putting It Altogether	43
4.1	Overview	43
4.1.1	Secure key generation with both channels	43
4.1.2	Message encoding and decoding	43
4.1.3	Summary of steps	43
4.2	Programs	43
4.2.1	Arduino programs	43
4.2.2	Python programs	43

Preface

Dear Qcumbers,

This book of technical documents is an accompaniment to the series of gamified do-it-yourself experiments that takes place in Quantum Camp 2018 at Centre for Quantum Technologies, Singapore. The purpose of this book is to give a cohesive picture on the technical implementation of the experiments, such that the reader may be able to reconstruct (some parts of) the experiment elsewhere.

The experiments themselves are sort-of-pedagogical implementations of Quantum Key Distribution (QKD), designed to convey ideas and steps to realise QKD systems, along with its classical channel counterpart, with a small remark at the end about some security issues. It is imperative to note that this is not a QKD implementation, even though they share a lot of properties and similarities. It is up to the reader to ponder and understand the previous statement.

In itself, the book will help to clarify the nitty-gritty details of the implementation, and will serve as an excellent resources to the students in the Quantum Camp Experimental Session. As this is the first iteration of the event, and probably even the first iteration of the writing, the authors would like to express apologies should there be mistakes in the writings or explanations.

Best of luck to all the participants and have fun!

Qcamp2018 Team

1

Preliminaries

“Try not. Do, or do not. There is no try.”

– Yoda, *The Empire Strikes Back*

1.1 About the computer

It is easy to feel overwhelmed by commercialised solutions and esoteric computer codes and conclude that programming a computer should be left only to the ‘real computer experts’ in the industry. This is somewhat true of software whose source-code is kept private. However, there is a plethora of open-source code that allow one to program the computer and customise it to perform tasks that commercial solutions might not provide. Collections of code called ‘libraries’, collect codes with specific functionalities that can be used as building blocks for your own code. Lightweight computers with digital-to-analog interfaces, and their accompanying libraries, allow the computer to sense and interact with the environment. This section attempts to describe non-exhaustively some of tools and concepts to kick-start you in your journey of becoming a tinkerer/hacker/programmer.

1.1.1 Arduino

Arduino is an open-source electronics platform based on easy-to-use hardware and software. Arduino boards are able to read inputs - light on a sensor, a finger on a button, or a Twitter message - and turn it into an output - activating a motor, turning on an LED, publishing something online. You can tell your board what to do by sending a set of instructions to the microcontroller on the board. To do so you use the Arduino programming language (based on C++), and the Arduino Software (IDE), based on Processing.¹

¹<https://www.arduino.cc/en/Guide/Introduction>

Workflow: How we will use the Arduino Programming Language together with the Python Programming Language

Arduino programs are written in the Arduino programming language, which is a variant of the C++ programming language. The syntax is often not reader-friendly but has the advantage of being incredibly fast at runtime. As such, we write basic 'building-block' programs in this language but avoid using it for more complex tasks. You can recognise them as the `.ino` files.

For complex tasks, we use a more readable language called Python to get the computer to execute the 'building-block' programs written in the Arduino language. You can recognise them as the `.py` files.

For example, in the folder `~/Qcamp2018/1_Classical/ArduinoClassical/`, the `ArduinoClassical.ino` arduino program contains C++ code that performs the basic function of (A) causing a light source to blink (B) detecting a blink using a photo-detector. However, we use the `chatting.py` Python code that uses a combination of blinks with program A to send a message between two parties.

To see how Python and Arduino works together in detail, see more in Section 1.1.3.

Uploading Your First Arduino Program: `ArduinoClassical.ino`

We will first attempt to upload the program '`ArduinoClassical.ino`' to Arduino.² `ArduinoClassical.ino` defines the four 'building block' functions `"HELP"`, `"SBLINK"`, `"RBLINK"`, `"SEND"`, `"RCV"` that can be called either an external program, which could be a (i) Python script, (ii) Serial Monitor or (iii) even through the terminal using the 'echo' command or (iv) any other program that can communicate with a serial device.³ You can read the code for the description of the four functions. But they basically help you to send and receive blinks from the connected LED.

If you watched Ironman 1, recall that Tony Stark had to upload some programs to his first prototype armor before his suit could even begin to function - the `ArduinoClassical.ino` does that for us here. Were you to build your own project, you can write the 'building block' functions relevant to your purposes.

²This code relies on the library available on <https://github.com/z3t0/Arduino-IRremote> which has already been installed in your computers.

³A serial port is a serial communication interface through which information transfers in or out one bit at a time (in contrast to a parallel port)

Opening the Arduino IDE: Start the Arduino IDE by typing the following in terminal:

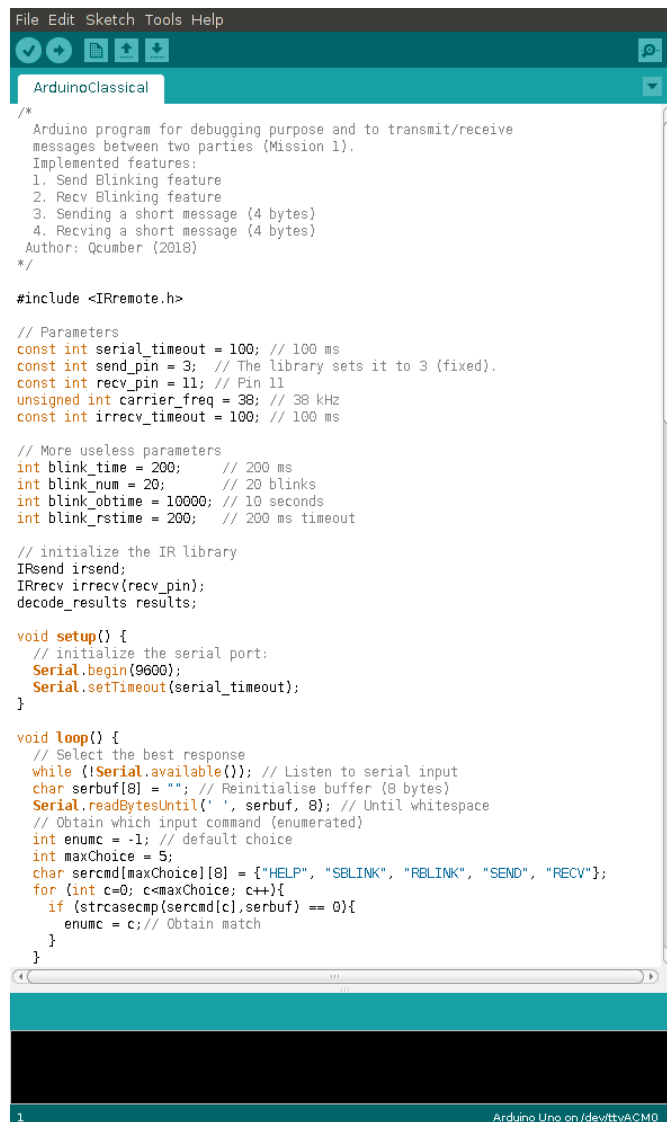
```
$ arduino
```

Or you can double click .ino files to load them immediately.

Try opening up the program located at:

```
~/Qcamp2018/1_Classical/ArduinoClassical/ArduinoClassical.ino
```

You should see the IDE loaded as shown:



```
File Edit Sketch Tools Help
ArduinoClassical
/*
  Arduino program for debugging purpose and to transmit/receive
  messages between two parties (Mission 1).
  Implemented features:
  1. Send Blinking feature
  2. Recv Blinking feature
  3. Sending a short message (4 bytes)
  4. Recvng a short message (4 bytes)
  Author: Qcumber (2018)
*/

#include <IRremote.h>

// Parameters
const int serial_timeout = 100; // 100 ms
const int send_pin = 3; // The library sets it to 3 (fixed).
const int rcv_pin = 11; // Pin 11
unsigned int carrier_freq = 38; // 38 kHz
const int irrecv_timeout = 100; // 100 ms

// More useless parameters
int blink_time = 200; // 200 ms
int blink_num = 20; // 20 blinks
int blink_obtime = 10000; // 10 seconds
int blink_rstime = 200; // 200 ms timeout

// initialize the IR library
IRsend irsend;
IRrecv irrecv(rcv_pin);
decode_results results;

void setup() {
  // initialize the serial port:
  Serial.begin(9600);
  Serial.setTimeout(serial_timeout);
}

void loop() {
  // Select the best response
  while (!Serial.available()); // Listen to serial input
  char serbuf[8] = ""; // Reinitialise buffer (8 bytes)
  Serial.readBytesUntil(' ', serbuf, 8); // Until whitespace
  // Obtain which input command (enumerated)
  int enumc = -1; // default choice
  int maxChoice = 5;
  char sercmd[maxChoice][8] = {"HELP", "SBLINK", "RBLINK", "SEND", "RCV"};
  for (int c=0; c<maxChoice; c++){
    if (strcasemp(sercmd[c], serbuf) == 0){
      enumc = c; // Obtain match
    }
  }
}
```

1 Arduino Uno on /dev/ttyACM0

Figure 1.1: The Arduino IDE with the program ArduinoClassical.ino loaded.

Check that Arduino is connected: The 'Arduino Uno' on /dev/ttyACM00

message on the bottom-right corner of the window verifies that your Arduino is connected properly.

Button Layout: The buttons on the top row perform these functions:

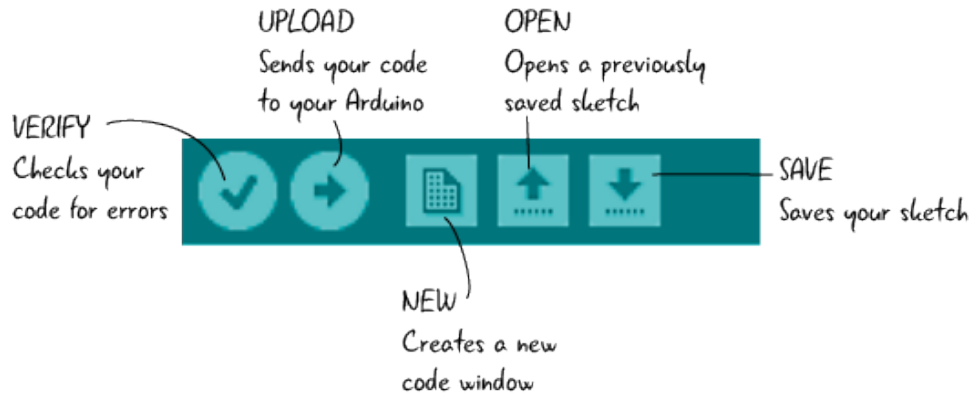


Figure 1.2: Arduino IDE button layout

Uploading and Verifying: To upload a program, click on the upload icon. It should return a **done uploading** message when successful. To ensure that the functions "HELP", "SBLINK", "RBLINK", "SEND", "RCV" that you have just uploaded to Arduino actually perform their task, you can do a quick test through Arduino's Serial Monitor.

Open the Serial Monitor by pressing **Ctl-Shift-M** or clicking through **Tools > Serial Monitor** on the Menu Bar.

Key in **HELP** and check if Arduino recognises this function. It should return:

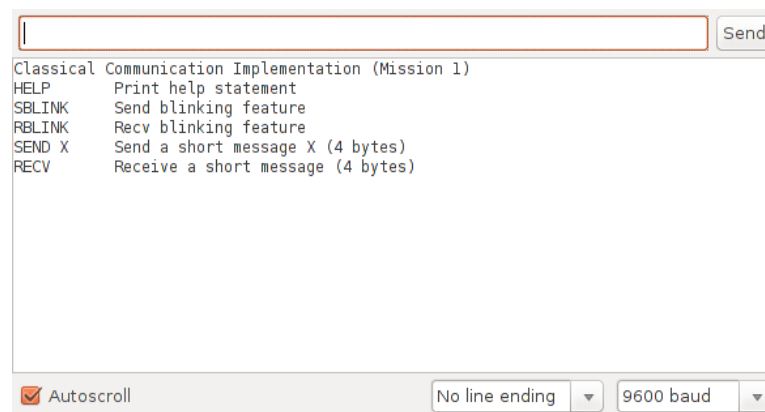


Figure 1.3: Serial Monitor returning the output for the **HELP** function.

Try keying in the other functions to observe the expected output. Know-

ing what each basic function outputs in return gives you an idea of how to process the output when they are being passed onto another programs.

Talk about the external libraries that we use

1.1.2 Linux Terminal

The Linux console is a system console internal to the Linux kernel.⁴ (a system console is the device which receives all kernel messages and warnings and which allows logins in single user mode). The Linux console provides a way for the kernel and other processes to send text output to the user, and to receive text input from the user.

General Commands

This section lists some of the common commands.

We will be using square brackets [...] to represent variables. For example, [FILE] represents the name of the file you wish to perform a command on.

- Changing directory

```
$ cd [DIRECTORY]
```

- The home folder can be assessed via:

```
$ cd ~
```

The camp folder is found in the location ~/QCamp/

- Changing to the parent directory:

```
$ cd ..
```

- Listing a directory

```
$ ls [DIRECTORY]
or
$ ls
```

- Listing a directory with details

⁴The kernel is a computer program that is the core of a computer's operating system, with complete control over everything in the system eg. CPU, RAM, Input/Output devices

```
$ ls -l
```

The '-' symbol indicates an OPTION that modifies the main command 'ls'. In this case, '-l' indicates that the option 'l' is being used.

The available options for a program is usually listed in the manual.

- Manual: getting help for a command

```
$ man [COMMAND]
```

There is even a manual for the manual command that you can access by typing, you guessed it, 'man man'.

Device Locations

When a device is connected and powered, successful detection by Linux results in a listing under the devices folder `/dev/`.

You can verify that a device is connected by listing the device folder.

```
$ ls /dev/
```

Depending on the manufacturer, USB devices are listed either as `/dev/ttyACMx` or `/dev/ttyUSBx`.

Kernel Messages

Displays all messages (or control) from the kernel ring buffer.⁵

Use this command to figure out what Linux is doing in the background.

Useful for debugging.

```
$ dmesg
```

For example, running `dmesg` after connecting a mouse returns the following message:

```
[ 79.577341] hid-generic 0003:046D:C534.0005:  input,hiddev0,hidraw1:
USB HID v1.11 Mouse [Logitech USB Receiver] on usb-0000:00:14.0-1/input1
```

1.1.3 Python

Python is a language of choice for quick development and creating human-friendly readable code. It is considered a 'high-level' language in that you do not have to worry about memory management (what size a variable is allowed to be, is the number a decimal point or an integer, etc...) and other

⁵The kernel ring buffer is a data structure that records messages related to the operation of the kernel. A ring buffer is a special kind of buffer that is always a constant size, removing the oldest messages when new messages come in.

'nut-and-bolt' issues - Python will handle them in the background. But it comes at a cost - because it has to figure these out by itself it will take time. Consequently, C++, a 'lower-level' language, wherein you have to define every 'nut-and-bolt', tends to be faster. Another reason why C++ is faster is because the computer converts the code from C++ language to machine readable form consisting of 0's and 1's. The computer executes this machine code very efficiently. Python however, has to convert line by line every time the code runs and is therefore less efficient.

“Hello, World”

- **C**

```
#include <stdio.h>

int main(int argc, char ** argv)
{
    printf("Hello, World!\n");
}
```
- **Java**

```
public class Hello
{
    public static void main(String argv[])
    {
        System.out.println("Hello, World!");
    }
}
```
- **now in Python**

```
print "Hello, World!"
```

2

Monday, June 14, 2010

Figure 1.4: A comparison between different coding languages all executing the same task: printing the statement "Hello World" on the screen.

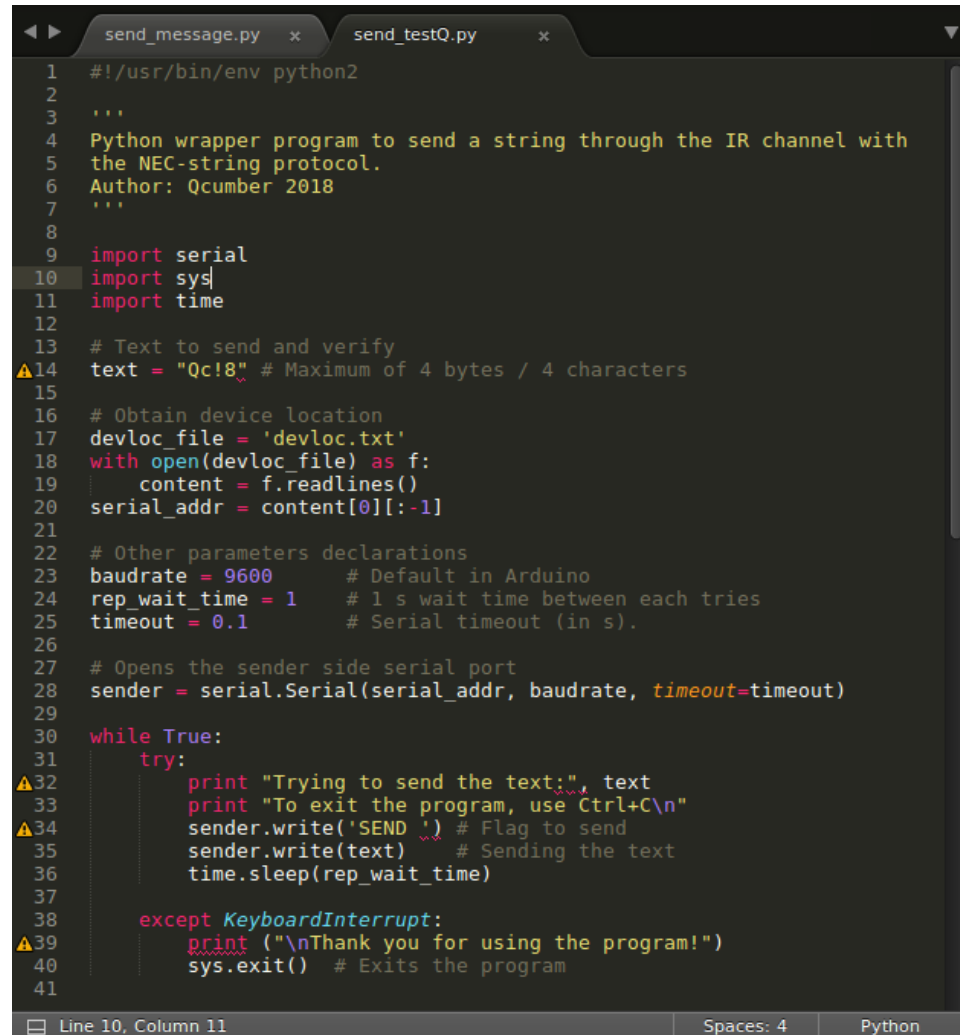
Editing Python Code

Opening Code in a Text Editor In this workshop, we will use the 'Sublime' text editor. It has the advantage of being able to compile / run code within the editor without going to the Linux Terminal, and also comes with nifty add-ons such as code auto-completes. You can open code by typing `subl [FILENAME.py]` in Terminal.

Understanding Python Code Most code can be broken up into 3 parts: (i) importing libraries, (ii) declaring variables & devices (iii) defining functions (even more building blocks) (iv) main body of code.

Editing Python Code

Let's look at one of the python scripts that we will be using to interact with the Arduino device:



```

1  #!/usr/bin/env python2
2
3  '''
4  Python wrapper program to send a string through the IR channel with
5  the NEC-string protocol.
6  Author: Qcumber 2018
7  '''
8
9  import serial
10 import sys
11 import time
12
13 # Text to send and verify
14 text = "Qc!8" # Maximum of 4 bytes / 4 characters
15
16 # Obtain device location
17 devloc_file = 'devloc.txt'
18 with open(devloc_file) as f:
19     content = f.readlines()
20 serial_addr = content[0][:-1]
21
22 # Other parameters declarations
23 baudrate = 9600 # Default in Arduino
24 rep_wait_time = 1 # 1 s wait time between each tries
25 timeout = 0.1 # Serial timeout (in s).
26
27 # Opens the sender side serial port
28 sender = serial.Serial(serial_addr, baudrate, timeout=timeout)
29
30 while True:
31     try:
32         print "Trying to send the text:", text
33         print "To exit the program, use Ctrl+C\n"
34         sender.write('SEND ') # Flag to send
35         sender.write(text) # Sending the text
36         time.sleep(rep_wait_time)
37
38     except KeyboardInterrupt:
39         print ("\nThank you for using the program!")
40         sys.exit() # Exits the program
41

```

Line 10, Column 11 Spaces: 4 Python

Figure 1.5: Python code ‘send_testq.py’ which uses the `SEND` command described in the `ArduinoClassical.ino` file, to send a message “Qc!8” via a series of LED blinks.

In the line
`sender = serial.Serial(serial_addr, baudrate, timeout=timeout)`
 we are defining a ‘sender’ by using the library ‘serial’. i.e. we are interacting with Arduino as if it were a serial device. Also notice that we had to define

the variable `serial_addr`, which is the location of the Arduino device. Notice that this was achieved by reading the file `devloc.txt`.

If your program is unable to detect the device location, open up the `devloc.txt` file and update it with the correct device location. You can determine the location by executing `ls /dev/` in Terminal.

Running Python Code

You can either press `Ctl-B` while in the Sublime text editor to build/run the code

or type the following in Terminal:

```
$ python [FILENAME.py]
```

Workflow Summary

Step 0: Ensure that Arduino device is connected and recognised by your computer Check the location of the device using either the Arduino IDE or typing `ls /dev/` in Terminal. Ensure this device location is defined in the `devloc.txt` file.

Step 1: Upload .ino files which define basic functions that Arduino recognises. These functions can be called by an external program such as those written in Python.

Step 2: Run the .py file to execute more complex tasks that are made up of a combination of the basic tasks defined in the `.ino` files.

1.1.4 Bit, Bytes, Hexadecimals and ASCII

Bits and Bytes

Here is a sort of glossary of computer buzzwords you will encounter in computer use:

Bit

Computer processors can only tell if a wire is on or off. Luckily, they can look at lots of wires at a time, and react to a complex pattern of ons and offs in pretty sophisticated ways. To translate these patterns into something that makes sense to humans, we consider a wire that is on to be a “1” and a wire that is off to be a “0”. Then we can look at the wires leading into a computer and read something like 00110111 00010000. We don’t know what that represents to the processor, it’s just a pattern. Each place in the

pattern is a bit, which may be 1 or 0. If it means a number to the processor, the bits make up a binary number.

Binary Numbers

Most of us count by tens these days. Ancient cultures used to count by 5s or 12s or 24s, but for the last thousand years, counting by tens has been the norm. When you see the number 145, you just know it includes one group of ten tens, plus four groups of ten, and five more. Ten tens is a hundred or ten squared. Ten hundreds is a thousand, or ten to the third. There's a pattern here. Each digit represents the number of tens raised to the power of the position of the digit, provided you start counting with zero and count right to left.

If you do the same thing with bits that can only be 1 or 0, each position in the list of bits represents some power of two. 1001 means one eight plus no fours plus no twos plus one extra. This is called binary notation. You can convert numbers from binary notation to decimal notation, but you seldom have to.

Bytes

Numbers like 00110111 10110000 are a lot easier to read if you put spaces every 8 bits. In decimal notation, we use commas every three digits for the same reason. There's nothing special about 8 bits, it just kind of got started that way. Hardware is easier to build if you group the wires consistently from one piece to another. Some older hardware used to group wires in 10s, but in the 70s the idea of working in groups of 8 really took over, especially in the design of integrated circuits. Somebody made a joke about a group carrying a byte of the data, and the term stuck. Sometimes you hear a group of four bits called a nibble.

The largest number you can represent with 8 bits is 11111111, or 255 in decimal notation. Since 00000000 is the smallest, you can represent 256 things with a byte. (Remember, a bite is just a pattern. It can represent a letter or a shade of green.) The bits in a byte have numbers. The rightmost bit is bit 0, and the left hand one is bit 7. Those two bits also have names. The rightmost is the least significant bit or lsb. It is least significant, because changing it has the smallest effect on the value. Which is the msb? (Bytes in larger numbers can also be called least significant and most significant.)

Hexadecimal Numbers

Even with the space, 00110111 10110000 is pretty hard to read. Software writers often use a code called hexadecimal to represent binary patterns. Hexadecimal was created by taking the decimal to binary idea and going the other way. Someone added six digits to the normal 0-9 so a number up

Hex	Binary	Decimal
0	0000	0
1	0001	1
2	0010	2
3	0011	3
4	0100	4
5	0101	5
6	0110	6
7	0111	7
8	1000	8
9	1001	9
A	1010	10
B	1011	11
C	1100	12
D	1101	13
E	1110	14
F	1111	15

Table 1.1: Hexadecimal Conversion Table

to 15 can be represented by a single symbol. Since they had to be typed on a normal keyboard, the letters A-F were used. One of these can represent four bits worth, so a byte is written as two hexadecimal digits. For example, these two bytes in binary representation “00110111 10110000” becomes “37B0” in hexadecimal representation. A handy-dandy table is included in Table 1.1.

With three different schemes running around, it’s easy to confuse numbers. 1000 can translate to a thousand, eight, or four thousand and ninety six. You have to indicate which system you are using. The fact that you still sometimes see an obsolete system called octal (digits 0-7. You can work it out) adds to the potential for confusion. Hexadecimal numbers can be indicated by writing them 1000hex 1000h or 0x1000. Binary numbers can be written 1000bin . Octal numbers were just written with an extra leading 0. Decimal numbers are not indicated, unless there’s some possibility of confusion, such as one in a page of hex numbers. ⁶

ASCII

Pronounced ask-ee, ASCII (American Standard Code for Information Interchange) is a code for representing English characters as numbers, with each letter assigned a number from 0 to 127. For example, the ASCII code for uppercase M is 77. Most computers use ASCII codes to represent text,

⁶http://artsites.ucsc.edu/ems/music/equipment/computers/bits_bytes/bits_bytes.html

which makes it possible to transfer data from one computer to another⁷.

Dec	Bin	Hex	Char	Dec	Bin	Hex	Char	Dec	Bin	Hex	Char	Dec	Bin	Hex	Char
0	0000 0000	00	[NUL]	32	0010 0000	20	space	64	0100 0000	40	@	96	0110 0000	60	`
1	0000 0001	01	[SOH]	33	0010 0001	21	!	65	0100 0001	41	A	97	0110 0001	61	a
2	0000 0010	02	[STX]	34	0010 0010	22	"	66	0100 0010	42	B	98	0110 0010	62	b
3	0000 0011	03	[ETX]	35	0010 0011	23	#	67	0100 0011	43	C	99	0110 0011	63	c
4	0000 0100	04	[EOT]	36	0010 0100	24	\$	68	0100 0100	44	D	100	0110 0100	64	d
5	0000 0101	05	[ENQ]	37	0010 0101	25	%	69	0100 0101	45	E	101	0110 0101	65	e
6	0000 0110	06	[ACK]	38	0010 0110	26	&	70	0100 0110	46	F	102	0110 0110	66	f
7	0000 0111	07	[BEL]	39	0010 0111	27	'	71	0100 0111	47	G	103	0110 0111	67	g
8	0000 1000	08	[BS]	40	0010 1000	28	(72	0100 1000	48	H	104	0110 1000	68	h
9	0000 1001	09	[TAB]	41	0010 1001	29)	73	0100 1001	49	I	105	0110 1001	69	i
10	0000 1010	0A	[LF]	42	0010 1010	2A	*	74	0100 1010	4A	J	106	0110 1010	6A	j
11	0000 1011	0B	[VT]	43	0010 1011	2B	+	75	0100 1011	4B	K	107	0110 1011	6B	k
12	0000 1100	0C	[FF]	44	0010 1100	2C	,	76	0100 1100	4C	L	108	0110 1100	6C	l
13	0000 1101	0D	[CR]	45	0010 1101	2D	-	77	0100 1101	4D	M	109	0110 1101	6D	m
14	0000 1110	0E	[SO]	46	0010 1110	2E	.	78	0100 1110	4E	N	110	0110 1110	6E	n
15	0000 1111	0F	[SI]	47	0010 1111	2F	/	79	0100 1111	4F	O	111	0110 1111	6F	o
16	0001 0000	10	[DLE]	48	0011 0000	30	0	80	0101 0000	50	P	112	0111 0000	70	p
17	0001 0001	11	[DC1]	49	0011 0001	31	1	81	0101 0001	51	Q	113	0111 0001	71	q
18	0001 0010	12	[DC2]	50	0011 0010	32	2	82	0101 0010	52	R	114	0111 0010	72	r
19	0001 0011	13	[DC3]	51	0011 0011	33	3	83	0101 0011	53	S	115	0111 0011	73	s
20	0001 0100	14	[DC4]	52	0011 0100	34	4	84	0101 0100	54	T	116	0111 0100	74	t
21	0001 0101	15	[NAK]	53	0011 0101	35	5	85	0101 0101	55	U	117	0111 0101	75	u
22	0001 0110	16	[SYN]	54	0011 0110	36	6	86	0101 0110	56	V	118	0111 0110	76	v
23	0001 0111	17	[ETB]	55	0011 0111	37	7	87	0101 0111	57	W	119	0111 0111	77	w
24	0001 1000	18	[CAN]	56	0011 1000	38	8	88	0101 1000	58	X	120	0111 1000	78	x
25	0001 1001	19	[EM]	57	0011 1001	39	9	89	0101 1001	59	Y	121	0111 1001	79	y
26	0001 1010	1A	[SUB]	58	0011 1010	3A	:	90	0101 1010	5A	Z	122	0111 1010	7A	z
27	0001 1011	1B	[ESC]	59	0011 1011	3B	;	91	0101 1011	5B	[123	0111 1011	7B	{
28	0001 1100	1C	[FS]	60	0011 1100	3C	<	92	0101 1100	5C	\	124	0111 1100	7C	
29	0001 1101	1D	[GS]	61	0011 1101	3D	=	93	0101 1101	5D]	125	0111 1101	7D	}
30	0001 1110	1E	[RS]	62	0011 1110	3E	>	94	0101 1110	5E	^	126	0111 1110	7E	~
31	0001 1111	1F	[US]	63	0011 1111	3F	?	95	0101 1111	5F	_	127	0111 1111	7F	[DEL]

1.2 About the electronics

1.2.1 Precaution

In general, we want to avoid shorting circuits and burning components:

Voltage Terminals: Don't short the + terminal to GND without any electrical component in between, as it can burn the chips.

LEDs: Don't ever connect them without a protection resistor in series with it. They are very easy to burn.

Integrated Circuits (IC): A lot of IC are very easy to burn if connected in a wrong polarity.

1.2.2 Arduino UNO Board

The Arduino UNO is a widely used open-source microcontroller board based on the ATmega328P microcontroller and developed by Arduino.cc. The board is equipped with sets of digital and analog input/output (I/O) pins that may be interfaced to various expansion boards (shields) and other circuits. The board features 14 Digital pins and 6 Analog pins. It is programmable with the Arduino IDE (Integrated Development Environment)

⁷<https://www.sparkfun.com/news/2121>

via a type B USB cable (see Section 1.1.1). It can be powered by a USB cable or by an external 9 volt battery, though it accepts voltages between 7 and 20 volts.⁸

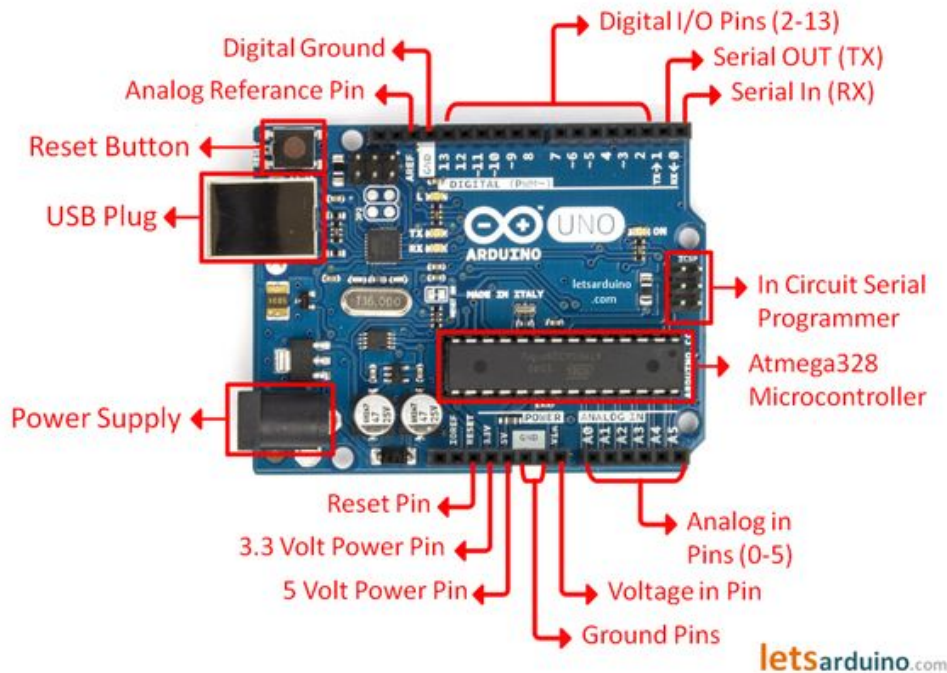


Figure 1.6: Arduino-Uno Board Layout

General Pin Functions

Figure 1.6 shows the Arduino board layout⁹.

- LED
There is a built-in LED driven by digital pin 13. When the pin is HIGH value, the LED is on, when the pin is LOW, it's off.
- VIN
The input voltage to the Arduino/Genuino board when it's using an external power source (as opposed to 5 volts from the USB connection or other regulated power source). You can supply voltage through this pin, or, if supplying voltage via the power jack, access it through this pin.
- 5V
This pin outputs a regulated 5V from the regulator on the board. The

⁸https://en.wikipedia.org/wiki/Arduino_Uno

⁹<https://www.pinterest.co.uk/pin/190910471683828392>

board can be supplied with power either from the DC power jack (7 - 20V), the USB connector (5V), or the VIN pin of the board (7-20V). Supplying voltage via the 5V or 3.3V pins bypasses the regulator, and can damage the board.

- **3.3V**
A 3.3 volt supply generated by the on-board regulator. Maximum current draw is 50 mA.
- **GND**
Ground pins.
- **IOREF**
This pin on the Arduino/Genuino board provides the voltage reference with which the microcontroller operates. A properly configured shield can read the IOREF pin voltage and select the appropriate power source or enable voltage translators on the outputs to work with the 5V or 3.3V.
- **Reset**
Typically used to add a reset button to shields which block the one on the board.

Special Pin Functions

Each of the 14 digital pins and 6 Analog pins on the Uno can be used as an input or output, using `pinMode()`, `digitalWrite()`, and `digitalRead()` functions. They operate at 5 volts. Each pin can provide or receive 20 mA as recommended operating condition and has an internal pull-up resistor (disconnected by default) of 20-50k ohm. A maximum of 40mA is the value that must not be exceeded on any I/O pin to avoid permanent damage to the microcontroller. The Uno has 6 analog inputs, labeled A0 through A5, each of which provide 10 bits of resolution (i.e. 1024 different values). By default they measure from ground to 5 volts, though is it possible to change the upper end of their range using the AREF pin and the `analogReference()` function.

In addition, some pins have specialized functions:

- **Serial pins 0 (RX) and 1 (TX)**
Used to receive (RX) and transmit (TX) TTL serial data. These pins are connected to the corresponding pins of the ATmega8U2 USB-to-TTL Serial chip.
- **External Interrupts pins 2 and 3**
These pins can be configured to trigger an interrupt on a low value, a rising or falling edge, or a change in value.

- PWM(Pulse Width Modulation) 3, 5, 6, 9, 10, and 11
Can provide 8-bit PWM output with the `analogWrite()` function.
- SPI(Serial Peripheral Interface): 10 (SS), 11 (MOSI), 12 (MISO), 13 (SCK)
These pins support SPI communication using the SPI library.
- TWI(Two Wire Interface): A4 or SDA pin and A5 or SCL pin
Support TWI communication using the Wire library.
- AREF(Analog REference
Reference voltage for the analog inputs

1.2.3 Breadboard

Breadboards are an essential tool for prototyping and building temporary circuits. These boards contain holes for inserting wire and components. Because of their temporary nature, they allow you to create circuits without soldering.¹⁰ The holes in a breadboard are connected in rows both horizontally and vertically as shown below.

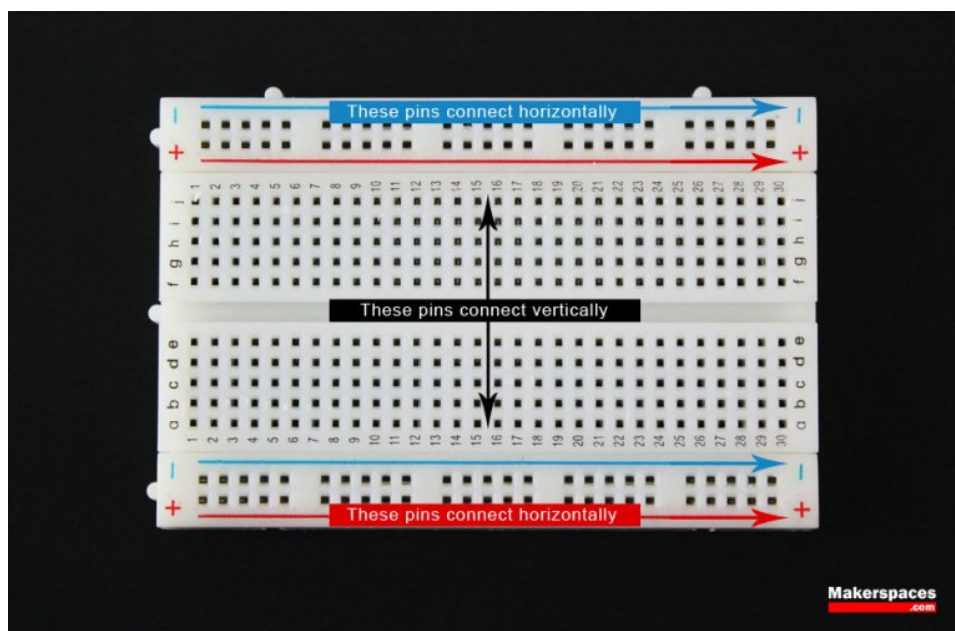


Figure 1.7: The holes in the breadboard are connected to its neighbours as shown in the figure.

¹⁰<https://www.makerspaces.com/basic-electronics/>

Example: Simple LED circuit

Suppose we want to implement a simple LED circuit in Figure 1.8.

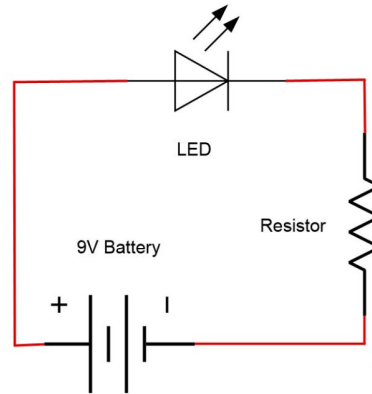
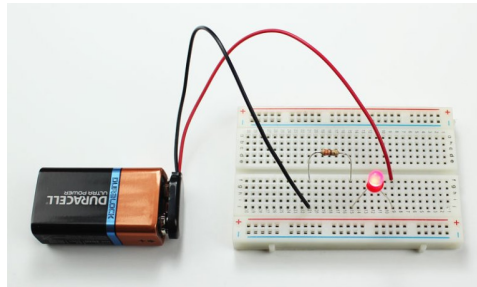
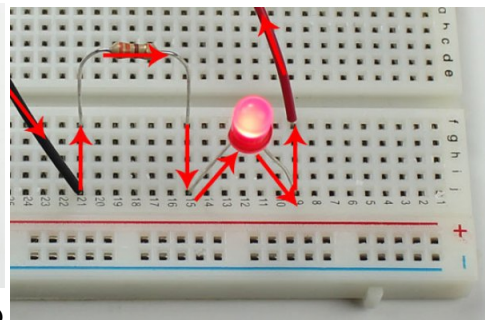


Figure 1.8: LED circuit with protective resistor. The resistor limits the current so that the LED does not burn out.

We may implement it using the breadboard layout as shown in Figure 1.9a. The flow of current shown in Figure 1.9b obeys the direction described in Figure 1.7.



(a) Breadboard implementation of LED circuit.



(b) LED circuit current path.

1.2.4 Resistor

We will use the resistor mainly to limit current to the LED. Other typical uses are to convert a current signal to a voltage signal through your all too familiar $V = IR$ formulae.¹¹

¹¹Typically, it is much easier to measure voltage than current at a particular location within a complex circuit since measuring current entails modifying the circuit and inserting an ammeter in the location of interest, whereas measuring voltage simply requires an electrical contact with the location.

1.2.5 Transistor

A transistor basically acts as a ‘gate’ for current to flow.

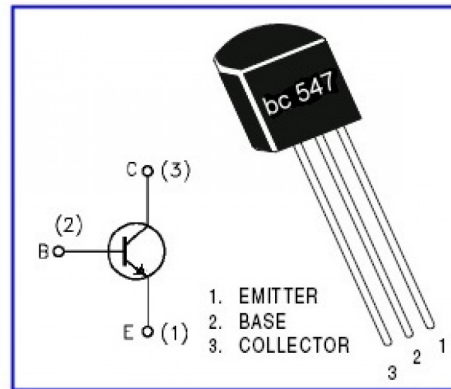


Figure 1.10: Transistor schematic: When current is sent to the Base, the transistor allows current to flow from the Collector to the Emitter. This allows the Base to function as a switch.

To operate the IR LED, we will use the Arduino to send current to the Base so as to switch on the current along CE wherein our led resides. We are unable to connect the Arduino to the LED directly since the Arduino pins have a maximum output of 40 mA. In the Arduino case, it is because the current from the pins (max 40 mA) are not enough for IR led and lasers to be in normal operation. We use BC547 or BC547A¹² transistor. We will operate our transistor in two modes modes¹³:

Saturation Mode Allows all available collector current to flow through.

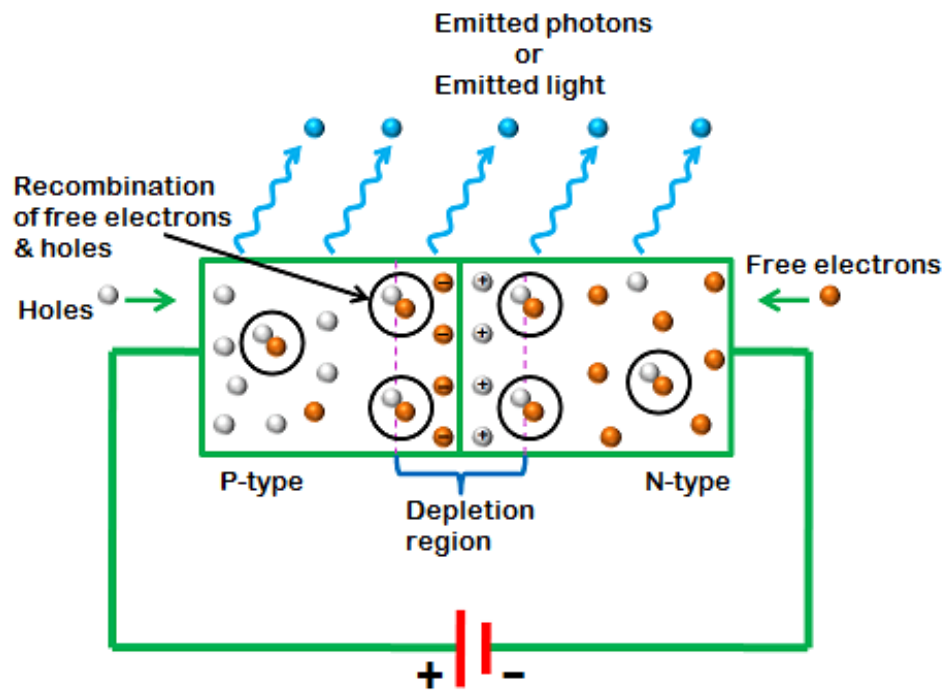
Cutoff Mode No current flows.

1.2.6 IR LED

A Light Emitting Diode (LED) consists of three layers: p-type semiconductor, n-type semiconductor and depletion layer. The p-type semiconductor

¹²<https://www.arduino.cc/documents/datasheets/BC547.pdf>

¹³<https://learn.sparkfun.com/tutorials/transistors/operation-modes>



Light Emitting Diode (LED)

Physics and Radio-Electronics

Figure 1.11: How an LED works: light is emitted when electrons and holes are recombined.

and the n-type semiconductor are separated by a depletion region or depletion layer.¹⁴

P-type semiconductor When trivalent impurities are added to the intrinsic or pure semiconductor, a p-type semiconductor is formed. In p-type semiconductor, holes are the majority charge carriers and free electrons are the minority charge carriers. Thus, holes carry most of the electric current in p-type semiconductor.

N-type semiconductor When pentavalent impurities are added to the intrinsic semiconductor, an n-type semiconductor is formed. In n-type semiconductor, free electrons are the majority charge carriers and holes are the minority charge carriers. Thus, free electrons carry most of the electric current in n-type semiconductor.

Depletion layer or region Depletion region is a region present between the p-type and n-type semiconductor where no mobile charge carriers (free electrons and holes) are present. This region acts as barrier to the electric current. It opposes flow of electrons from n-type semiconductor and flow of holes from p-type semiconductor.

How the LED works Light Emitting Diode (LED) works only in forward bias condition. When Light Emitting Diode (LED) is forward biased, the free electrons from n-side and the holes from p-side are pushed towards the junction. When free electrons reach the junction or depletion region, some of the free electrons recombine with the holes in the positive ions. We know that positive ions have less number of electrons than protons. Therefore, they are ready to accept electrons. Thus, free electrons recombine with holes in the depletion region. In the similar way, holes from p-side recombine with electrons in the depletion region.

Because of the recombination of free electrons and holes in the depletion region, the width of depletion region decreases. As a result, more charge carriers will cross the p-n junction.

Some of the charge carriers from p-side and n-side will cross the p-n junction before they recombine in the depletion region. For example, some free electrons from n-type semiconductor cross the p-n junction and recombines with holes in p-type semiconductor. In the similar way, holes from p-type semiconductor cross the p-n junction and recombines with free electrons in the n-type semiconductor.

¹⁴<http://www.physics-and-radio-electronics.com/electronic-devices-and-circuits/semiconductor-diodes/lightemittingdiodeledconstructionworking.html>

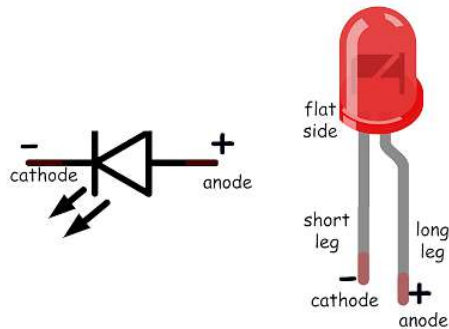


Figure 1.12: Polarity of an LED. Current flows and light is emitted when a higher voltage is connected to the anode. Otherwise, you will not observe any light.

IR receiver Refer to the pin assignment below. This device is very easy to burn. Please do not burn more than your instructor.

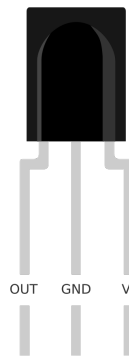
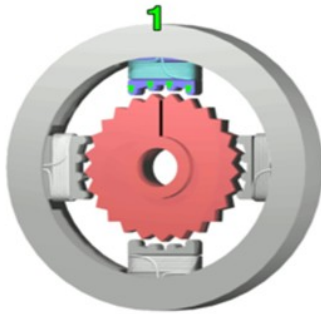


Figure 1.13: Connections for the TSOP38238 IR receiver

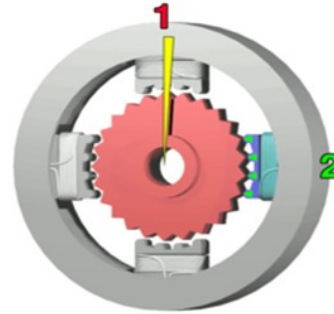
From our experience playing with the led and receiver: 1. It will reflect a few times off the walls and still be detected by the receiver, but the SNR might suffers. 2. The best position is to align them sort of facing each other, but not necessarily line of sight

1.2.7 Stepper motor

Stepper motors effectively have multiple "toothed" electromagnets arranged around a central gear-shaped piece of iron. The electromagnets are energized by an external driver circuit or a micro controller. To make the motor shaft



(a) The top electromagnet (1) is turned on, attracting the nearest teeth of a gear-shaped iron rotor. With the teeth aligned to electromagnet 1, they will be slightly offset from electromagnet.



(b) The top electromagnet (1) is turned off, and the right electromagnet (2) is energized, pulling the nearest teeth slightly to the right. This results in a rotation of 3.6° in this example.

turn, first, one electromagnet is given power, which magnetically attracts the gear's teeth. When the gear's teeth are aligned to the first electromagnet, they are slightly offset from the next electromagnet. This means that when the next electromagnet is turned on and the first is turned off, the gear rotates slightly to align with the next one. From there the process is repeated. Each of those rotations is called a "step", with an integer number of steps making a full rotation. In that way, the motor can be turned by a precise angle.¹⁵

How the stepping works. A visual depiction of how it works is shown in Figures 1.14a to 1.15b¹⁶

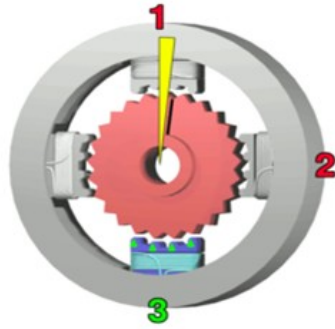
Usage of the motor I/C can be found at¹⁷. The motor needs an external power supply to power the stepper motor (5V). The required current of running the motor is around 220-280 mA (depending on the sequence and or speed), and can spike even more (particularly during on and off sequence: remember that the connection in stepper motor are mostly inductors). Thus, the 5V supply from the Arduino is probably not enough (depending on condition, max output around 250 mA). We use another Power supply adapter of 5V just to drive the motor

All pins must be connected properly (easy to miss a few connection or cable swapped). If even one pin is not connected, it might appear to be working, but it will give erratic and inconsistent results. There are a total of 2048 steps in 1 revolution: 32 internal steps + 64:1 gear ratio. The

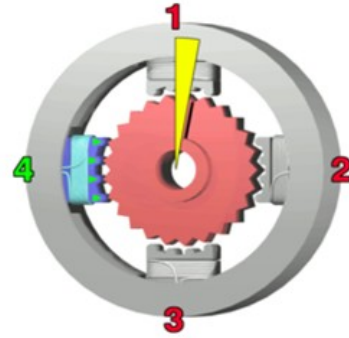
¹⁵https://en.wikipedia.org/wiki/Stepper_motor

¹⁶<http://slideplayer.com/slide/6419773/>

¹⁷<https://www.diodes.com/assets/Datasheets/ULN200xA.pdf>



(a) The bottom electromagnetic (3) is energized; another 3.6° rotation occurs.



(b) The left electromagnetic (4) is enabled, rotating again by 3.6° . When the top electromagnetic (1) is again enabled, the teeth in the sprocket will have rotated by one tooth position; since there are 25 teeth, it will take 100 steps to make a full rotation in this example.

fastest that we can drive each step is 1.5 ms. But we set it be 2 ms for safety tolerance. Thus, one full revolution will take $2048 * 2\text{ms} = 4.1\text{ s}$

1.3 About laser and polarisation

Talks about how this is important in the QKD. Motivate why use laser

1.3.1 Precaution

General precaution about the laser. In the end, we use the sanyo 780 nm (but operate it within the safe 5 mW limit). As the wavelength belongs to the near-infrared, it might not appear bright to your eyes but it can still be highly intense.

We will put some beam blockers for the safety, but make sure (can refer to the laser safety manual thingy, mostly need commonsense thingy).

The setup would have been already pre-aligned, so make sure don't touch them. The students might need to connect some electrical wires, make sure be careful.

1.3.2 Laser

Describe type of laser, and how you can obtain them. Talks about the circuit (with transistor) -> similar to the discussion above. Because sanyo diode only starts lasing at 30 - 40 mA, to protect the arduino pin, we used also

transistor to regulate the max current. Resistor values and laser power to be given by Mathias.

1.3.3 Photodiode

Describe the type of photodiode (I need to look at this again). We also use a lens to focus the light (correct for pointing error etc etc). Describe the reverse bias and how to read the voltage of the photodiode They can change sensitivity by adjusting the variable resistor knob. The voltage will be read with arduino analog pin with 10-bit ADC $\rightarrow 2^{10} = 1024$ quantization value thingy. Reading speed X

1.3.4 Polarisation

Describe what is the polarisation of the light. Important, need to unwrap the plastic of those linear polarisers if not yet unwrapped. Polarisers can be cleaned with IPA (or actually simple water might just do the job).

How do we generate the polarisation from the sender, i.e. initially the laser is already polarised, so we need to “depolarise” it with a wave plate (i.e. make it circularly polarised), which can then be polarised with the linear polariser attached to the stepper motor.

To read the polarisation, there will also be another linear polariser attach to the stepper motor on the other side.

1.3.5 Some background story

Initially, we want to do the experiment using the 632 cheap red laser pointer. But we can't find a suitable wave plate and the circular polariser (i.e. you can use circular polariser as sort of a wave plate) that is usually sold does not work very well on that wavelength unfortunately, as the depolarisation is not good enough (i.e. still more or less elliptical with intensity ratio 1:2).

The situation is actually not that bad, and with some modification to the programs we can still perform the experiment, but in the end we choose to use 780 nm as we have the perfect waveplates to make it circularly polarised (depolarised).

We probably need to try other wavelength that match the circular polariser wavelength, or we find a circular polariser wavelength that match the wavelength. We have not successfully done that due to time constraint. Another method is to rotate the laser instead (as the laser itself is already linearly polarised), but it is mechanically very challenging.

2

Classical Channel

“Probably a common misconception to the general public about QKD is that the whole process runs on the quantum channel. On the contrary, quantum channel is only used to distribute keys securely, and the encrypted message itself needs to run along the classical channel.”

– Alice and Bob, *Mind you, quantum!*

The first part of this series is about building a classical channel. With the advent of technology and internet, we might already take for granted the pervasiveness of this classical communication. Nowadays, it is very easy to tap into the extensive web of internet. Every connection, whether to any other parties or to the internet, is by itself a classical channel.

This pedagogical exercise uses a “less common, but probably more visually pleasing” approach to establish a classical channel, which is infrared (IR) signals. Nowadays, IR signals see a lot of application in TV, AC and projector remote controls.

2.1 IR signals

In this section, we discuss how we implement classical communication channel with IR signals. The process itself is very similar with how TV remote control tells the TV to turn on/off, change channels, or change volumes. In the IR signal that the remote sends, there will be packets of information, i.e. strings of ones and zeros that are sent to the TV.

2.1.1 Electrical connections

Sender

The sender circuit consists of an IR LED (TSAL7400), a few resistors, and a transistor (BC547). The purpose of the transistor is to provide higher

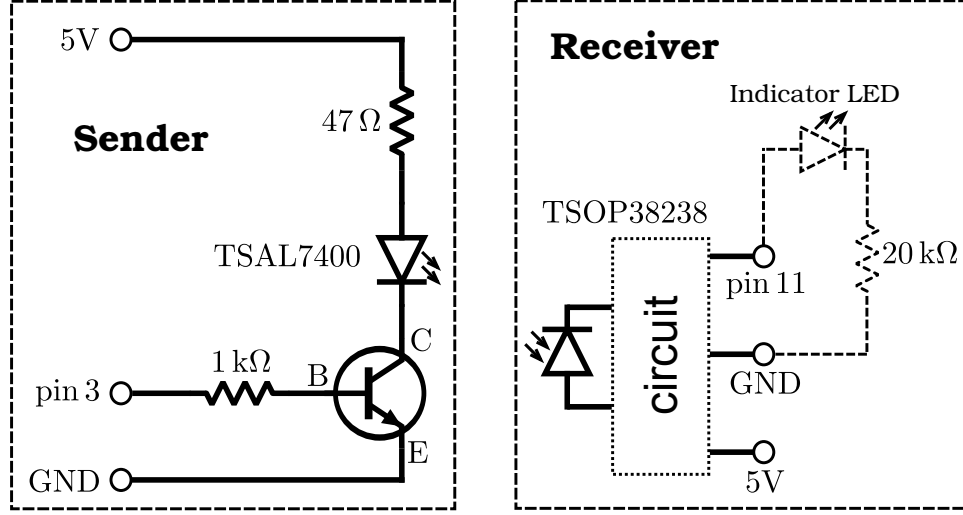


Figure 2.1: Electrical circuit of the IR sender and receiver

current to the IR LED ($\sim 60\text{mA}$) than the maximum allowable rating of the Arduino pin (40mA). To achieve this, we can use $R_B \approx 1\text{k}\Omega$ as the base resistor and $R_C \approx 47\Omega$ as the collector resistor. We use pin 3 on the Arduino to switch on/off the IR LED. Please consult Section 1.2 on how to connect the components properly. *Warning: The transistor can burn easily if connected in the wrong polarity.*

To test whether the circuit works, you can try sending blinking signals (see Section 2.2.1). You should be able to see the infrared light using your phone camera (except iPhone of course, which has an excellent IR filter).

Receiver

The IR receiver module (TSOP38238) consists of an IR photodiode with a built-in demodulation circuitry. The output of integrated circuit / photodiode module can be directly connected to Arduino pin 11. Please consult Section 1.2 on the pin assignments. *Warning: The IR receiver module can burn easily if connected in the wrong polarity.*

You may add in an (optional) indicator LED to monitor the output of the IR receiver module (indicated as dashed components in Figure 2.1). Note that you might need to use quite a high resistance value ($\sim 20\text{k}\Omega$) as the demodulation circuitry can not output more than a few mAs. Also, note that the pin logic is opposite, i.e. it turns OFF (0V pin reading or LED off) when it receives an ON signal from the IR sender, and vice versa.

To test the circuit, you can try receiving the blinking signals sent by another party (see Section 2.2.1). If you succeed, the connection between the sender and receiver is successfully established.

Modulation

There is still another ingredient to this IR channel: signal modulation. In simple terms, the IR light is switched on and off (modulated) repeatedly with a very high frequency (38 kHz). On the receiver side, there is an array of electronics that demodulates the signal, i.e. reading only the signal that has been repeatedly switched on and off. Thus, the receiver can only detect signal of frequency 38 kHz ($\pm 10\%$, according to the datasheet).

This process will suppress unwanted noise in the IR range that might come from elsewhere, i.e. room fluorescent lamps (~ 100 Hz modulation due to power grid).

2.1.2 From NEC to the protocol

We derive our communication protocol from the commonly known NEC (Nippon Electric Company) IR protocol. The logical ones and zeros are defined by the length of the signal in the ON and OFF position (Figure 2.2):

- Logical 1: ON for $560\ \mu\text{s}$ and OFF for $1690\ \mu\text{s}$
- Logical 0: ON for $560\ \mu\text{s}$ and OFF for $560\ \mu\text{s}$



Figure 2.2: The definition of logical 1s and 0s in the NEC protocol. Image source: <http://www.snrelectronicsblog.com/8051/nec-protocol-and-interfacing-with-microcontroller/>

Every time you press on a button on the remote, it will send a train of pulses, consisting of:

1. Header: ON for 9 ms and OFF for 4.5 ms.
2. Address byte: 8 bits of 1s and 0s containing the information to which device should this signal be delivered to, i.e. to this model of TV.
3. Inverse address byte: The logical opposite of the address byte, for redundancy purpose.
4. Command byte: 8 bits of information of what the device should do, i.e. turn ON/OFF, change channel, etc.

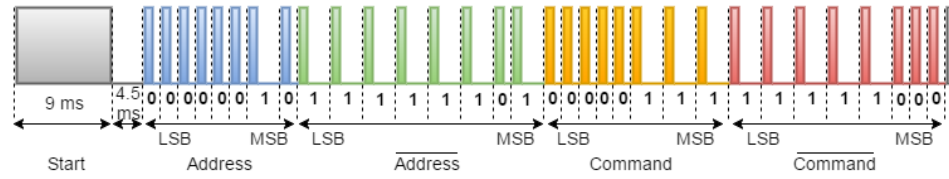


Figure 2.3: Typical example of a pulse train in the NEC protocol. Image source: <http://www.snrelectronicsblog.com/8051/nec-protocol-and-interfacing-with-microcontroller/>

5. Inverse command byte: The logical opposite of the command byte, for redundancy purpose.
6. Termination pulse: Such that the receiver can determine whether the last bit is 1 or 0.

For our experiment, there is only Alice and Bob, so we don't really need the address byte. Also, as we are not operating in a very noisy environment, we don't need the redundancy bytes either. Thus, we will use all four available bytes (32 bit) in the NEC protocol to send our data. Messages longer than 4 bytes will be cut into 4-byte chunks and send consecutively.

2.2 Programs

The programs to perform Mission 1 are located in the folder "1_Classical".

2.2.1 Arduino program

The Arduino program "ArduinoClassical.ino" consists of several subroutines to perform tasks related to testing and communicating IR signals, which can be accessed via serial communication. It can be used for both the sender and the receiver. There are some set parameters to the program, which can be modified at wish (and if you really know what you are doing). The default values for some of the parameters are given below as: (def: #).

To distinguish between serial sent by computer to Arduino (TX) and the serial sent by Arduino to computer (RX), we use the flags (TX) and (RX) in the discussion below. List of serial (TX) commands:

- **HELP**
Prints list of serial commands and their short descriptions (in case you forget). Alternatively, you can keep referring to this notes.
- **SBLINK**
Sends a blinking signal with a half-period of "blink_time" (def: 200 ms) for a total of "blink_num" (def: 20) periods.

- **RBLINK**
Listens for any blinking signal for “blink_obtime” (def: 10 s). Serial (RX) of “BLINK!\n” will be sent for each blinking period detected.
- **SEND X**
Sends a 4-byte character X with the protocol defined in Section 2.1.2.
- **RECV**
Listen to any 4-byte character sent by other devices sharing the same protocol, and sends the result to the computer via serial (RX). This process will not continue if no signal is received. To cancel this process, you need to send a termination character “#” via serial (TX).

2.2.2 BinaryComm package

These packages are only used to demonstrate the process of sending and receiving 8 bit binary sequences. There are two programs: “send_binary.ino” and “recv_binary.ino”. The former should be used by Alice, and the latter by Bob. The 8 bit binary sequence (for example the letter A: 01000001) should be written on the serial monitor of “send_binary.ino”, which will then be sent through the IR channel and appears on the serial monitor of “recv_binary.ino”.

2.2.3 Python programs

The Arduino program only serves as the backbone for the classical communication. To perform more complicated tasks, we have written several python programs that will aid in performing Mission 1.

devloc.txt

This is a simple text file that contains the information of the Arduino device location (usually “/dev/ttyACM0” or “/dev/ttyUSB0”). The python programs will look into this file prior to running to determine the device location of Arduino, and hence the file needs to be updated should there be any changes in the device or its location.

To check the location of the device after plugging in the USB hub, you might want to run the “dmesg” program in the Linux terminal.

conv_ascii.py

This program converts between different representation of ASCII characters: ASCII string, hex representation, and binary representation.

send_testQ.py

This program sends a test message of “text” (def: “Qc!8”) repeatedly every “rep_wait_time” (def: 1 s). This program can be used to ensure that the correct message is being transmitted and received by “recv_testQ.py”.

recv_testQ.py

This program listens to a message of “text” (def: “Qc!8”). It should print “Mission successful” if the message is received successfully and the “text” is the same with that in “send_testQ.py”. If the “text” is not the same, it will print “Receiving something that does not seem correct”.

Both “recv_testQ.py” and “recv_testQ.py” can be used to check for the quality of the transmission or for troubleshooting purposes.

send_message.py

This program asks the user to input the message (“tosend_string”), which will be sent through the IR channel afterwards. The message is split into 4-byte chunks (4 characters each time), with a message header of “[STX]×4” and a message footer of “[ETX]×4” to signify the start and end of the message transmission.

Each chunk takes around “rep_wait_time” (def: 0.3 s) to transmit. We have experimented with different waiting time in our computer, and so far 0.3 s is a safe choice (0.2 s is a bit on the edge as some chunks might get lost). This is in part due to serial communication between computer and Arduino (~ 80 ms), sending and receiving of IR signals (~ 70 ms), and operating system overhead. For slower computers, you might need to increase “rep_wait_time” to prevent packet loss.

recv_message.py

This program listens to any incoming message sent with our protocol (with “send_message.py”). It first looks for “[STX]×4” message header, and joins 4-byte message chunks. The listening process will terminate when it receives “[ETX]×4” message footer.

chatting.py

This program is a combination of both the programs “send_message.py” and “recv_message.py” in a looping sequential order. Thus, it enables for some form of chatting to happen. The program starts in a sending mode, which can be switched to receiving mode by pressing [Enter].

3

Quantum Channel

“If (A1) quantum mechanics is correct, and (A2) authentication is secure, and (A3) our devices are reasonably secure, then with high probability the key established by quantum key distribution is a random secret key independent (up to a negligible difference) of input values.”

– Douglas Stebila, *The Case for Quantum Key Distribution* ¹

In the second part of this series, we describe how to implement a variant ² of quantum key distribution (QKD) experiment by using lasers, polarisers, and photodetectors. Particularly, we will explore how the secret key can be formed with random keys, random basis choices, and the sifting process.

3.1 Laser signal

The main component of this experiment is the laser signal. Contrary to the previous chapter, which uses the ON/OFF sequence to encode the information, the laser signal described in this chapter uses light polarisation instead. For a review on light polarisation, please consult Section 1.3.4.

The section below describes the nitty-gritty details on how the laser signal is produced and detected, along with the protocol that we utilised to ensure that the receiver can receive the signal sent by the sender faithfully.

3.1.1 General schematics

In this section we describe the electrical connections as well as the general schematics of the experiment. The laser light is generated with a laser module (via pin 4), which then passes through a quarter wave plate and a linear polariser (which angle can be controlled with a stepper module). This

¹<https://arxiv.org/abs/0902.2839>

²Recall that this is not QKD. For more information, read the Preface.

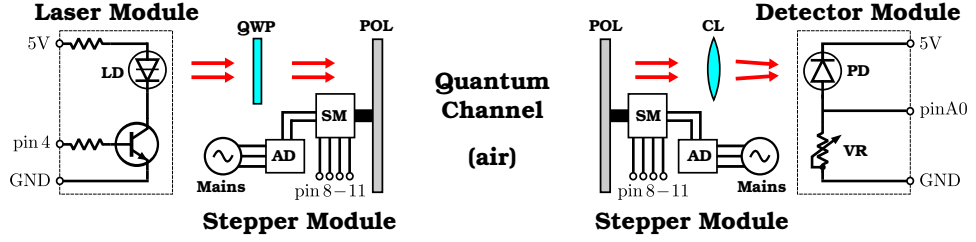


Figure 3.1: The schematics of the QKD experiment. Labels: (LD) Laser Diode, (QWP) Quarter Wave Plate, (POL) Linear Polariser, (SM) Stepper Motor, (AD) Electrical Adapter, (CL) Converging Lens, (PD) Photodiode, and (VR) Variable Resistor.

linear polariser will produce different polarisation corresponding to different key and basis choices (or different quantum states).

On the receiver side, another linear polariser will filter the light and project it into a measurement basis. The light will then be focused onto the photodiode with a converging lens. The detector module will output the light power via pin A0.

Sender

The electrical circuit for the sender consists of two main parts: laser module and stepper module. The laser module consists of a 780 nm laser that can be turned on/off with pin 4. *Be careful, as the laser is actually quite bright, even though it appears to be pretty dim, as the wavelength is in near-infrared regime. The laser can actually damage your eyes (similar power with a fancy laser pointer, Class 3A, below 5 mW). Please refer to safety precaution in Section 1.3.1.*

The stepper module consists of a stepper motor (with its electronics) which rotates a linear polariser. As the stepper motor requires relatively high current and ($\sim 220\text{-}280\text{ mA}$) and relatively fast switching ($\sim 2\text{ ms}$), we used a Darlington transistor array chip (ULN2002A) powered by an external 5V power adapter. The stepper motor is driven with pin 8 - 11 of Arduino.

As the laser output is linearly polarised, we introduce a 780 nm quarter wave plate to ‘depolarise’ the laser output, i.e. ‘depolarise’ in this case means making it circularly polarised, which has very similar effect with depolarisation. Then, after going through the linear polariser, the light will be linearly polarised with similar intensity at any polarisation axis angle.

Receiver

The receiver also consists of two main parts: stepper module (similar to the sender) and detector module. The detector module consists of a photodiode (any ‘generic’ near-infrared photodiode can do, i.e. SFH213) operated in

reverse biased mode. Before saturation, the light irradiance on the photodiode is proportional to the voltage at A0, and the sensitivity can be tuned with the variable resistor. To produce optimal result and to reduce misalignment sensitivity, we use a converging lens to focus the laser light on the photodiode.

3.1.2 In the sequence

For each polarisation configuration (each bit of ‘data’), the stepper motor needs to move to the correct angle, and the laser (detector) needs to send (receive) the light. The main bottleneck of this process is the movement of the stepper motor, which requires around 1.07 s to move 90° . We limit the turning angle³ between each bit to be 90° , and set that each bit uses 1.5 s, consisting of 1.1 s to rotate the polarisers, 0.1 s empty space⁴, 0.2 s to turn on the lasers, and end with 0.1 s empty space. The detector is set to read at the 1.3 s mark of each bit, right in the middle when the laser is on.

3.1.3 Synchronisation

To start each sequence of bits, Alice sends a synchronisation pulse to Bob (200 ms on, 200 ms off, and set to D polarisation). When Bob detects this pulse, he starts his own sequence. From our observation, there is a small offset (less than 5 ms) between Alice and Bob’s sequences. Also, Alice and Bob’s clock tick slightly in a different rate, so there is a maximum length that Alice can send before the sequences go out of sync. We set the length of our sequence to be 16 bits, but it can well be extended to 128 or 256 bits.

3.2 Key generation

In this section, we will describe each steps that need to be undertaken to generate the secure key. In this experiment, we employ a variant of BB84 quantum key distribution (QKD) scheme, which relies on some basis comparison and sifting process. To generate the key that is ‘independent on the input’⁵, we need to find some methods to generate ‘true’ random numbers. We also need to find a way to represent the keys and random numbers in light polarisation.

³Because the polariser in, i.e. 0° and 180° is basically the same polarisation, you can reduce the range of motion of the polariser to be max 90° , which reduces time of motion.

⁴The empty space is just an added waiting period where Arduino does nothing and only waits. The purpose of this is to time-separate the different steps (easier for debugging).

⁵Contrast this with pseudo random number generator, in which case after you know the initial seed, you can obtain the whole sequence.

3.2.1 Random numbers

For the random numbers, we use the Entropy library that was developed by Walter Anderson ⁶. This library harvests random numbers from the timing jitter between two different clocks (the watchdog RC timer v.s. the system clock). The author does not claim that this is the true random source ⁷, but some quick tests show that the numbers are quite random, unpredictable and pretty superior to other libraries ⁸, albeit with a relatively low bit generation rate ($\sim 2 \times 32$ bits per second).

3.2.2 Representation

There are 4 polarisation axes in this experiment: (H) horizontal, (D) diagonal, (V) vertical, and (A) antidiagonal, each of them separated by 45° . We represent each polarisation direction with a basis bit and a value bit according to Table 3.1.

Basis	Value	Polarisation
0	0	H
1	0	D
0	1	V
1	1	A

Table 3.1: Representation of polarisation on the basis bit and value bit

To make sense of the table, one can think of the basis bit as either Horizontal-Vertical or Diagonal-Antidiagonal, which we then assign to be 0_B and 1_B respectively ⁹. In each basis, there can be two values: 0_V (assigned to H or D) and 1_V (assigned to V or A).

3.2.3 BB84 QKD scheme

In the experiment, we implement a variant of BB84 QKD scheme which follows the following steps:

1. Alice generate 2 sequences of random numbers (basis and value), while Bob generate a sequence of random numbers (basis). The length of the sequence is 16 bits.
2. For each bit, Alice sends the polarised light depending on the basis and value bit sequences, i.e. if the basis bit is 1_B and the value bit is 0_V , then she sends diagonally polarised (D) light.

⁶<https://sites.google.com/site/astudyofentropy/home>

⁷Probably not when you compare it with randomness from quantum processes

⁸There are some other libraries that make use of the ‘random’ fluctuation in the analog pin input, for example.

⁹We use the subscript notation X_y , where X is either 0 or 1, and y is its representation.

3. Bob measures the light sends by Alice according to his basis sequence ¹⁰. He then takes note of the measurement result. His measurement result should be binary, i.e. above or below some threshold value. We denote this as 0_R and 1_R respectively ¹¹.
4. Bob sends his measurement basis via classical public channel to Alice. Alice then compares her basis and Bob's basis. She takes note which basis matches, which she represents with 0_M if the basis does not match, and 1_M if the basis match. She sends this information to Bob via classical public channel.
5. Both Alice and Bob go through a process called sifting, i.e. removing the value bits (for Alice) and measurement result bits (for Bob) which bases do not match (0_M).
6. Alice and Bob then compiles the rest of bits which bases match (1_M) by appending them next to each other. If the measurement is done properly, for matched bases, $0_V = 0_R$ and $1_V = 1_R$. Thus, in the end they would have obtained the same symmetric key.

3.3 Implementation

This section gives a brief summary of the implementation of this QKD scheme. In principle, one needs to perform these three steps:

1. Polarisation basis alignment. This step is to make sure that Alice's polarisation is aligned to Bob's polarisation.
2. Light generation and measurement. This step deals with the process of sending the polarised light and measuring them. This process occupies the quantum channel and takes the bulk of the time.
3. Secure key generation. This step comes after the transmission through the quantum channel, and is performed through the classical channel. This process is essential to ensure that both Alice and Bob obtain the same key.

3.3.1 Polarisation basis alignment

The first thing to do in any QKD system is to make sure that the polarisation axis are aligned with each other. In our experiment, we perform this in two steps:

¹⁰We assume that the measurement is done in the 0_V bases, i.e. H and D polarisation

¹¹There is no typo: above the threshold means the polarisation is correct, which would be 0_V . If the polarisation is incorrect, it would be 1_V .

1. Aligning the angle of the H polarisation axis between Alice and Bob. There are two ways to do this:
 - Alice fixes her H polarisation, and Bob finds and sets an angle offset which maximise the light transmission. This angle offset for maximum transmission makes Bob's H polarisation to be aligned with that of Alice.
 - Similar to the above, but now with Alice finding and setting her angle offset instead. This way, Alice's H polarisation is aligned to that of Bob.
2. After the alignment procedure, we construct the intensity matrix with various polarisation settings set by Alice and Bob. The theoretical values for the intensity matrix is given in Table 3.2. Due to imperfections of the devices (lasers, quarter wave plates, polarisers), the measured intensity matrix deviates from the expected value. We can define this deviation ("signal degradation") with the formula:

$$\eta = \frac{\sum_{ij} |d_{ij} - d_{ij}^{\text{exp}}|}{\sum_{ij} d_{ij}} \quad (3.1)$$

where d_{ij} is the measured intensity on i -th Alice's polarisation setting and j -th Bob's polarisation setting, and d_{ij}^{exp} is the expected intensity with the aforementioned settings, which is obtained from Table 3.2 and normalised to the measured values with the following relation:

$$\sum_{ij} d_{ij}^{\text{exp}} = \sum_{ij} d_{ij} \quad (3.2)$$

		Bob			
		H	D	V	A
Alice	H	1	0.5	0	0.5
	D	0.5	1	0.5	0
	V	0	0.5	1	0.5
	A	0.5	0	0.5	1

Table 3.2: Expected value of intensity of different polarisation settings between Alice and Bob, normalised to the maximum transmission.

To interpret the value of η , we can consider the following cases:

- (a) When the measured intensity matrix follows the expected matrix very closely. In this case, $|d_{ij} - d_{ij}^{\text{exp}}| \approx 0$ for all i and j . Thus, the signal degradation is very small: $\eta \approx 0$.

- (b) When the measured intensity matrix is constant for all cases, i.e. the polariser does nothing to the light ('unpolarised'). In this case, $d_{ij} \approx 0.5$ for all i and j , and thus $\eta \approx 0$
- (c) When the measured intensity matrix 'anti-correlates' with the expected matrix, i.e. when there is an offset of 45° or 90° between Alice and Bob's H polarisation.

3.3.2 Light generation and measurement

3.3.3 Secure key generation

3.4 Programs

The programs necessary to perform Mission 2 are located in the folder "2_QuantumKey".

3.4.1 Arduino programs

Similar to the previous chapter, the Arduino program "ArduinoQuantum.ino" consists of several subroutines to perform tasks related to rotating the stepper motors, turning the laser on/off, and generate and run the random sequence, which can be accessed via serial communication. The program is compatible with both Alice and Bob. There are some set parameters to the program, which can be modified at wish (and if you really know what you are doing). The default values for some of the parameters are given below as: (def: #).

To distinguish between serial sent by computer to Arduino (TX) and the serial sent by Arduino to computer (RX), we use the flags (TX) and (RX) in the discussion below. List of serial (TX) commands:

- **HELP**
Prints list of serial commands and their short descriptions (in case you forget). Alternatively, you can keep referring to this notes.
- **SETANG X**
Rotates the stepper motor to a specified angle X in degrees. The value of the latest set angle will be stored in the Arduino EEPROM, so that one does not lose the value after a shutdown cycle or any accidental misconnection. This program assumes that the stepper motor has "stepsPerRevolution" (def: 2048) steps in each revolution, and that the stepper motor is connected through pin 8, 9, 10, and 11. The serial (RX) prints "OK" after the task is finished.
- **ANG?**
Asks for the current angle. The serial (RX) prints the value in degrees.

- SETPOL X

Rotates the stepper motor to a specified polarisation X: 0 (H), 1 (D), 2 (V), and 3 (A). It does not understand any values besides 0, 1, 2, and 3, and will assume it is 0 (H). The angle set with a specific polarisation is calculated with the following formula:

$$\text{ANG} = 45^\circ \times \text{POL} + \text{HOF} \quad (3.3)$$

where ANG is the set angle in degrees, POL is the set polarisation, and HOF is angle offset of the H polarisation (see below).

The serial (RX) prints “OK” after the task is finished.

- POL?

Asks for the current polarisation. The serial (RX) prints the value in floating point number based on equation 3.3.

- SETHOF X

Sets the angle offset of the H polarisation (HOF) in degrees (see equation 3.3). This value will also be stored in EEPROM.

The serial (RX) prints “OK” after the task is finished.

- POLSEQ X

Sets the polarisation sequence according to X, a sequence of characters (consisting of 0, 1, 2, and 3) with length of “seqLength” (def: 16 bits). This function will not run the sequence.

The serial (RX) prints “OK” after the task is finished.

- RNDSEQ X

Generates the polarisation sequence (POLSEQ) with random numbers (0 to 3) with the Entropy library. This is due to the following relation:

$$\text{POL} = 2 \times X_V + X_B \quad (3.4)$$

where X_B is the basis bit of the sequence and X_v is the value bit of the sequence (see Section 3.2.2).

The serial (RX) prints “OK” after the task is finished.

- RNDBAS X

Similar to RNDSEQ, but with only random numbers 0 and 1 (X_B). This will set Bob’s measurement basis to be on either H or D.

The serial (RX) prints “OK” after the task is finished.

- SEQ?

Asks for the set sequence. The serial (RX) prints the sequence in string with values of 0 to 3.

- LASON

Turns on the laser connected to “pinLsr” (def: 4).

The serial (RX) prints “OK” after the task is finished.

- **LASOFF**
Turns off the laser connected to “pinLsr” (def: 4).
The serial (RX) prints “OK” after the task is finished.
- **VOLT?**
Asks for the voltage at “sensorLoc” (def: A0). The serial (RX) prints the result in floating point number (in units of V).
- **CATCH**
Waits until the voltage at “sensorLoc” (def: A0) gets higher than “catchTh” (def: 400 units, which is around 2 V). This command blocks until the condition is fulfilled. The serial (RX) prints the time when it happens (in milliseconds).
- **RUNSEQ**
Runs the sequence set by either POLSEQ, RNDSEQ, or RNDBAS. The sequence is defined in Section , but the ‘laser’ in this case is “pinDeb” (def: 13). This command is useful for debugging purpose. The serial (RX) prints “OK” after the task is finished.
- **TXSEQ**
Similar with RUNSEQ, but is customised for Alice’s sequence (TX). It switches on/off the laser at “pinLsr” (def: 4). The serial (RX) prints “OK” after the task is finished.
- **TXSEQ**
Similar with RUNSEQ, but is customised for Alice’s sequence (TX). It measures the voltage “sensorLoc” (def: A0) at the “seqReadTime” (def: 1300 ms) mark.
The serial (RX) prints the measurement result in sequences of integers (0 - 1023 for 0 - 5 V), separated with spaces, after the task is finished.

3.4.2 Python programs

4

Putting It Altogether

4.1 Overview

4.1.1 Secure key generation with both channels

4.1.2 Message encoding and decoding

4.1.3 Summary of steps

4.2 Programs

4.2.1 Arduino programs

4.2.2 Python programs