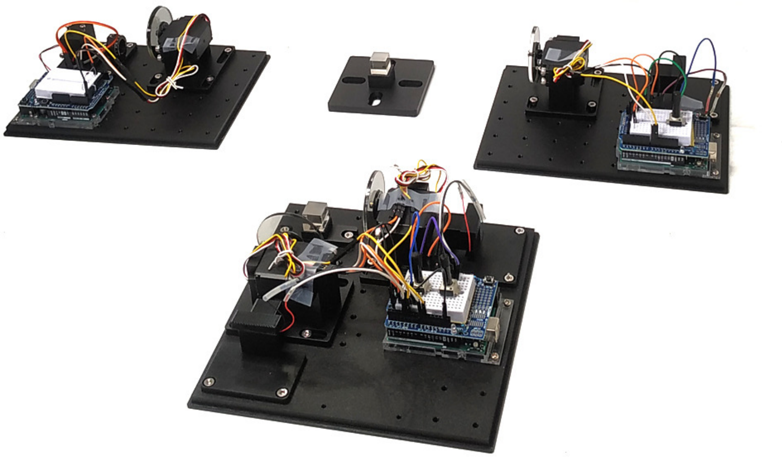
EKPQC

Educational Kit: Programmable Quantum Cryptography

Students' Guide
First Edition
July 10, 2023



S-Fifteen Instruments Pte. Ltd.
Singapore
https://s-fifteen.com/

Introduction

This manual describes an experimental workshop designed to teach Quantum Key Distribution (QKD). The workshop presumes that the participants have some rudimentary understanding of Quantum Mechanics: quantum states, bases, unitary operators, its axioms, and the no-cloning theorem.[1] Our aim is to go from abstract to concrete; from understanding these quantum-mechanical concepts, to implementing a working QKD system.

First, the students set up two communication channels: a classical channel using infrared pulses, and a "quantum channel" using polarization encoded photons. Next, they distribute a symmetric key between two parties with the BB84 protocol [3]. Finally, they use the key to encrypt secret messages and send them over to the other party via the classical channel.

To simplify implementation of the "quantum" channel, we use macroscopic light intensity levels, instead of transmitting single-photons, which require more specialized training and equipment to generate and detect. However, this comes at a cost of introducing a security loophole, which will be exploited experimentally.

Apart from this hardware implementation difference, the steps in the BB84 protocol are faithfully implemented – this provides an opportunity for students to understand that the security of QKD is not solely an intrinsic property of the protocol itself, but relies critically on exploiting quantum mechanical properties of the particles used to distribute the key. Incorrect physical implementation[2] leads to unintended security vulnerabilities [7].

The workshop has been peer-reviewed in Ref. [8], and successfully implemented with pre-university students at the Centre of Quantum Technologies, Singapore, from 2018 onwards. Students have reported that the highlight of the workshop was that, despite being informed about the formidable security that QKD promises, and following the BB84 protocol to the letter, they were surprised that a flaw in implementation resulted in a successful hacking attempt – we kept the presence of the hacking team a secret.

We hope that you will enjoy this educational kit. It is fully programmable – please feel free to modify the kit and write to us at `info@s-fifteen.com` if you feel that the kit can be improved to help spread the wonder and fun of learning about the applications of Quantum Physics!

[1] Excellent introductory texts include Refs. [1, 2].

[2] In addition, additional protocols are required in a practical implementation: e.g. authentication is required to secure against man-in-the-middle attacks, countermeasures are required to overcome device and source imperfections. There is currently no internationally-recognized standard for practical implementation security. However, many commercial QKD vendors, including S-Fifteen Instruments, follow the guidelines published by the European Telecommunications Standards Institute (ETSI) [4–6].

Organization of Manual

The organization of this manual is as follows:

Chapter 1 provides the theoretical foundations of QKD: namely, the use of randomly chosen encoding bases, and the no-cloning theorem. It will also provide a high-level overview of the experimental setup of the educational kit.

Chapter 2 describes the infrared transmitter and receiver circuits used to establish the classical channel. Students will follow the experimental layout to construct their own classical channel, with the accompanying software that enables the students to send and receive test messages. This channel is subsequently used as part of the QKD protocol.

Chapter 3 describes the 650 nm laser and photodiode used to send and receive 'quantum bits'. After constructing the quantum channel, students will understand and execute a calibration procedure that will help communicating parties establish a common coordinate reference used to define the polarization states of photons.

Chapter 4 combines the key distribution and key sifting steps – students use the quantum channel established in the previous chapter to send and receive bits of information that can be 'sifted' into the final key. The key-sifting procedure is facilitated by communication over the classical channel. The final key is then used to encrypt secret messages transmitted through the classical channel.

Chapter 5 describes the eavesdropping unit and techniques, used to intercept part of the distributed key. Students will construct the eavesdropping unit and attempt to use the intercepted key information to decrypt secret messages transmitted between the intended parties.

Discussion and Further exploration

We have included discussion sections that facilitate instructors to pose thoughtful questions pertaining to QKD implementation.

Further exploratory tasks, which may involve diving deeper into each experimental component, are also included. Some tasks are included to exploit the advantage that a programmable Arduino microcontroller provides, and can be used in conjunction with other electronic components to explore experiments not directly associated with QKD – the microcontroller is primarily used to actuate various parts of the QKD experiment. The level of theoretical or experimental difficulty associated with the tasks is indicated by the number of asterisk signs (*).

Safety

The < 1 mW 650 nm laser diode used in the experiment, is a Class 2 laser in accordance to Radiation Protection (Non-Ionising) Regulations 1991 [9]:

"Class 2 lasers emit visible light and are limited to a maximum output power of 1 milliwatt (mW). A person receiving an eye exposure from a Class 2 laser will be protected from injury by the person's natural aversion response – an involuntary response which causes the person to blink and turn their head, thereby avoiding further eye exposure."

Although there is no requirement for additional protective equipment, nonetheless, we would like to highlight the following safety precautions:

1. Ensure that the experimental setup is constructed below eye level.
2. Do not look directly at the laser, or point it at someone's eye.
3. Alert everyone on the beam path, so that they can avoid it.
4. Do not wear accessories that may accidentally reflect the beam into somebody's eye e.g. watches, bracelets and rings.
5. Turn off the laser after use.

Warning!

This programmable QKD system is designed for pedagogical purposes only: it implements the BB84 QKD protocol albeit with a slight modification, which introduces a security loophole. We do not recommend using the system for establishing QKD in a non-educational context.

Annotation

Note that the following set of polarization states $\{|H\rangle, |V\rangle, |D\rangle, |A\rangle\}$ are used interchangeably with $\{|\leftrightarrow\rangle, |\updownarrow\rangle\ |\swarrow\!\!\!\nearrow\rangle, |\nwarrow\!\!\!\searrow\rangle\}$, and they indicate horizontal, vertical, diagonal, and anti-diagonal polarization states, respectively. Where the ket notation $(|\quad\rangle)$ does not provide additional insight, they are dropped and the polarization states are indicated by {H, V, D, A} instead.

Related materials

Assembly and Installation Manual

A detailed inventory list, software installation and hardware assembly guide are available in a separate manual; the present manual focuses more on how the experimental system can be delivered in the form of an educational workshop.

Media

Kit assembly instructions are available at:
https://www.youtube.com/channel/UCRx-sMrlPE24LeWxHMTps4g.

Contents

Chapter 1

Theoretical Foundations & Experiment Overview

Objectives: This chapter provides a brief description of the BB84 QKD protocol. Most commercial QKD systems adopt this protocol, albeit with additional modifications that secure it against sophisticated eavesdropping attacks [10].

An overview of the experimental implementation is presented. Importantly, we highlight the key differences between this setup, which is meant to train participants on the foundations of the BB84 protocol, and its original implementation.

In addition, we will also present the installation instructions of the software used to control the classical and quantum communication channels of the setup.

1.1 Basics of QKD

Quantum Key Distribution (QKD) is the process where secret symmetric digital keys can be shared privately between two parties exploiting the concepts of Quantum Mechanics. The security of QKD does not rely on computational complexity and thus will not be vulnerable to future enhancements in algorithms, processing power or the emergence of quantum computers. QKD does not address the issue of authentication and how the keys will be used to encrypt data, however, typically, authentication can be performed by private-public key authentication while encryption via AES128 algorithm.

Suppose one party Alice, wants to communicate to another party Bob securely with the certainty that their communication is not being eavesdropped upon by a third party, Eve (See Figure 1.1). To do this, Alice prepares and transmits photons to Bob, polarized randomly in one of two non-orthogonal bases (Rectilinear or Diagonal), as shown in Table 1.1. The bases are used to encode logical bits of information. Bob, having no information about the state or basis of the incoming photon, randomly chooses one of two non-orthogonal bases from the same set as Alice, and measures the polarization state. The process is repeated until a desired number of measurements is obtained. After the measurements, they publicly discuss their measurement basis. For a particular polarization state measurement, if their basis coincide, they keep the results of the encoded bit, otherwise, the results are discarded. To check for the presence of an eavesdropper, some of the results are compared. Any error on the measurement can be due to the Eve performing her own measurements on the transmitted photons from Alice to Bob. Eve

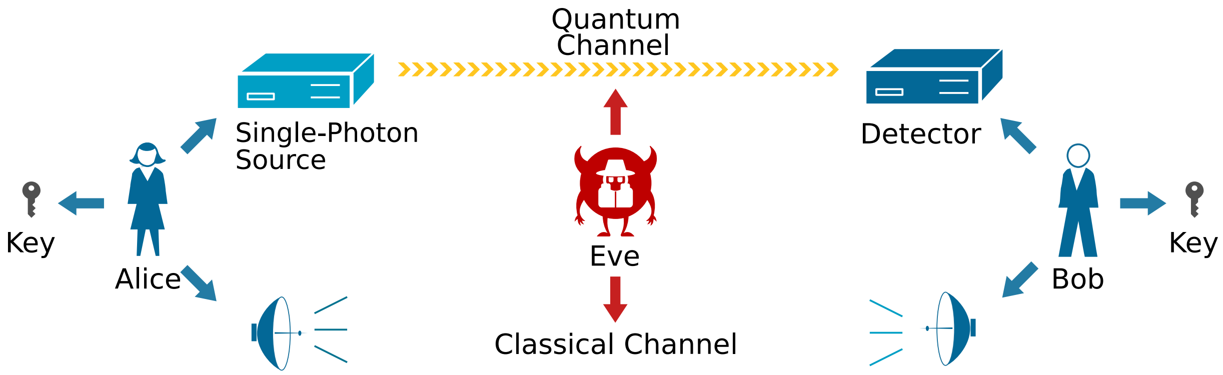


Figure 1.1: Alice and Bob wish to communicate securely where both of them are assured that any attempt by Eve to listen to their communication can be detected. Each party follows a QKD protocol using a classical and quantum communication channel, whose outcome is to provide them with a shared, private key. The key can be subsequently used to encrypt messages transmitted across the classical channel. QKD security relies on the properties of quantum states distributed across the quantum channel. Eavesdropping by Eve results in a quantifiable error in the fidelity of the transmitted quantum states. When the transmission error crosses an intolerable threshold, no private key will be generated by the protocol, i.e. while Eve can impose a denial-of-service attack, she is nonetheless prevented from possessing any useful portion of the shared key.

is unable to perform measurements without introducing errors in the measurements for Bob due to the no-cloning principle of Quantum Mechanics that states that an arbitrary quantum state cannot be cloned without knowledge of the state a-priori. Alice and Bob then perform error correction and privacy amplification to ensure that any knowledge of Eve of the shared key can be reduced to an arbitrarily small amount with the trade-off of final key length.

Basis	Alternate name	Polarization angle(°)	0	1
Rectilinear(+)	Horizontal Vertical (H-V)	0,90	↔	↕
Diagonal(×)	Diagonal Anti-Diagonal (D-A)	+45,-45	⤢	⤡

Table 1.1: Two non-orthogonal bases for polarization measurments. Measurements in the rectilinear (diagonal) basis will result in outcomes ↔(⤢) or ↕(⤡) which will encode bits of 0 or 1.

1.2 BB84 Protocol

In 1984, Charles Bennett and Gilles Brassard published their famous protocol for key distribution in a Proceedings of IEEE International Conference on Computers [3]. Here we highlight the implementation and elaborate slightly on the possible measurments and their outcomes (Refer to Table 1.2 for an example).

In BB84, Alice has single photons and prepares the state that is then sent to Bob. This state preparation can be in one of four states using two randomly chosen bases. Bob will also perform state measurement by choosing one of the two bases from the same set as Alice. Bob's measurement result will be correlated (same) as Alice when his measurement basis matches Alice's and uncorrelated (random) otherwise. This process is repeated for a string of transmitted states. Once the measurement sequence is completed, Bob will

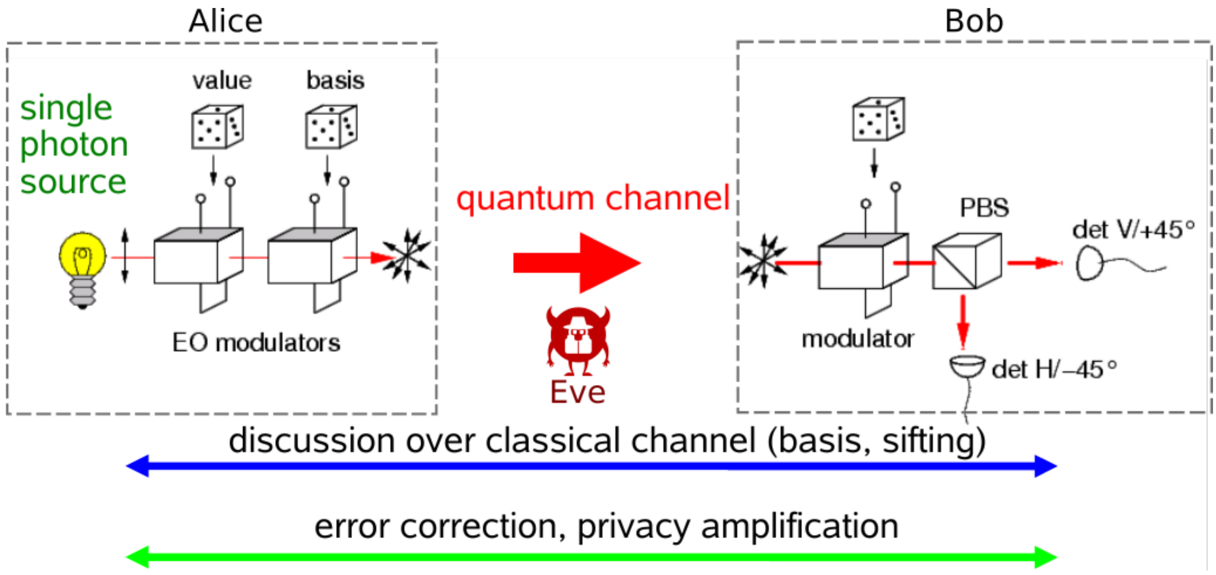


Figure 1.2: BB84 Protocol: Alice (left) and Bob (right) wish to establish a common secret key that they can use to encrypt secret messages. Alice sends Bob single-photon states which encode a bit in a basis determined by a random number generator (RNG). Bob measures the states in a basis that is also randomly determined by his RNG. Both parties communicate their bases over a public channel after a string of states are transmitted; revealing bases does not reveal the value encoded by Alice. When their measurement basis agree, the value is received accurately by Bob and is kept to generate a common secret key. Otherwise, Bob's measurement is uncorrelated (random) and the measurement round is discarded by both parties. This process of comparing bases is called key sifting. EO modulators/modulator: Electro-optic modulators used to set the polarization of light sent by Alice, and is also used to set the measurement basis at Bob – each modulator implements a polarization rotation to the photon. Dice: RNG. PBS: Polarization Beam Splitter reflects/transmits photons according to their polarization.

publicly announce the sequence of basis he has measured Alice's photon in and Alice will acknowledge for which measurements Bob's basis matched hers. On average, their basis will match for half of the measurement results, and Bob retains those results as he can be assured that they should be the same as the states prepared by Alice. The remaining results obtained when their basis do not match will be discarded. This process is called key-sifting.

After this they will have some shared information that may or may not contain errors due to measurement and/or the presence of an eavesdropper. To check for this, Bob reveals some of his keys at random and then Alice confirms them. If the keys match, then Alice and Bob can say that their communication was not intercepted with some confidence. If some of the keys don't match, they need to determine what is the error rate and whether their communication can still be secure by some error correction [11, 12] and privacy amplification techniques (e.g. by means of a compression matrix). Finally, they will establish a Quantum Bit Error Rate (QBER). Only if the QBER falls below a certain value will the QKD communication be considered secure [13] and the final secret key can be used.

Quantum Transmission															
Alice’s random bits	0	1	1	0	1	1	0	0	1	0	1	1	0	0	1
Random sending bases	D	R	D	R	R	R	R	R	D	D	R	D	D	D	R
Photons Alice sends	⤢	↕	⤡	↔	↕	↕	↔	↔	⤡	⤢	↕	⤡	⤢	⤢	↕
Random receiving bases	R	D	D	R	R	D	D	R	D	R	D	D	D	D	R
Bits as received by Bob	1		1		1	0	0	0		1	1	1		0	1
Public Discussion															
Bob reports bases of received bits	R		D		R	D	D	R		R	D	D		D	R
Alice says which bases were correct			OK		OK			OK				OK		OK	OK
Presumably shared information (if no eavesdrop)			1		1			0				1		0	1
Bob reveals some key bits at random					1									0	
Alice confirms them					OK									OK	
Outcome															
Remaining shared secret bits			1					0				1			1

Table 1.2: Example of the BB84 protocol from [3].

1.3 Theoretical foundations of BB84

The security of the quantum channel relies mainly on two quantum mechanical concepts: conjugate coding and the no-cloning theorem. We briefly describe each concept and explain how they contribute to the security of the protocol. [1]

1.3.1 Coding in different, randomly selected bases

In BB84, Alice encodes logical bits (0 or 1), in a qubit using different bases which she chooses at random (Table 1.1). This technique is known as conjugate coding [14]. When a qubit is prepared in one basis but measured with another basis, the outcome is completely randomized. Consequently, the receiver needs to know the bases in which an unknown qubit is prepared in, in order to receive the transmitted logical bit information without error.

Consider, for example, that Alice sends a logical bit 0 bit using the rectilinear basis i.e. she transmits a horizontally polarized photon in the polarization state $|\leftrightarrow\rangle$.

Since Eve does not know *a-priori* the basis with which Alice prepares the state, she would choose the incorrect basis (diagonal) half the time.

When Eve measures $|\leftrightarrow\rangle$in the diagonal basis, she would obtain either the $|\text{⤢}\rangle$outcome half the time:

$$|\langle \text{⤢}| \leftrightarrow\rangle|^2 = \left|\frac{\langle\leftrightarrow| + \langle\updownarrow|}{\sqrt{2}}\,|\leftrightarrow\rangle\right|^2 = \frac{1}{2} \tag{1.1}$$

[1] Understanding these proofs require understanding quantum states and measurements. An introduction to these concepts may be found in Ref. [2].

or the |⤡⟩outcome the other half of the time:

$$|\langle ⤡ | \leftrightarrow\rangle|^2 = \left|\frac{\langle\leftrightarrow| - \langle\updownarrow|}{\sqrt{2}}\,|\leftrightarrow\rangle\right|^2 = \frac{1}{2} \tag{1.2}$$

Since |⤢⟩and |⤡⟩, which represent a logical bit 0 and 1 respectively, occur completely at random, Eve's cannot determine the logical bit sent by Alice.

Moreover, if she attempts to resend the intercepted qubit to the intended receiver, she will do so in the incorrect basis half of the time. Consider the scenario when Eve resends a qubit in the diagonal basis to Bob. If Bob measures the qubit in the rectilinear basis, he will measure $|\updownarrow\rangle$ with a probability of 1/2. This is a marked contrast to the situation where Eve is absent: since Bob's measurement basis is the same as Alice's preparation bases, he is supposed to measure $|\updownarrow\rangle$ with zero probability, given that Alice sent $|\leftrightarrow\rangle$. Consequently, such measurement errors can be used to detect the presence of an eavesdropper. In typical operation, Alice and Bob will compare a random subset of their transmitted and measured qubits, and check for errors in measurement rounds where their basis is the same. Once they are satisfied that there is no eavesdropping attempt, they will use the remaining subset to generate the final encryption key.

1.3.2 No-cloning theorem

The no-cloning theorem prevents Eve from making identical copies of a quantum state. This effect is desirable since Eve would be unable to analyse a copy of the transmitted state [15, 16], which would have allowed her to gain information without the concomitant risk of introducing an error to the original, transmitted state.

Cast in the operator formalism of quantum mechanics, the no-cloning theorem essentially states that, there is no unitary operator $\hat{U}$ which acts on an arbitrary, unknown quantum state $|\psi\rangle$ and an ancillary qubit $|0\rangle$, so that $\hat{U}$ transforms the ancilliary qubit to an exact copy $|\psi\rangle$:

$$U(|\psi\rangle\,|0\rangle) = |\psi\rangle\,|\psi\rangle \tag{1.3}$$

Proof by contradiction:

Suppose that a unitary operator that could perform cloning exists. Then we would have:

$$\hat{U}\,|0\rangle\,|0\rangle = |0\rangle\,|0\rangle \tag{1.4}$$

and

$$\hat{U}\,|1\rangle\,|0\rangle = |1\rangle\,|1\rangle \tag{1.5}$$

Consider now an arbitrary state $\psi = \alpha\,|0\rangle + \beta\,|1\rangle$ with $\alpha, \beta \in \mathbb{C}$. Then,

$$\begin{aligned} \hat{U}\,|\psi\rangle\,|0\rangle &= \hat{U}(\alpha\,|0\rangle + \beta\,|1\rangle)\,|0\rangle && (1.6)\\ &= \alpha\hat{U}\,|0\rangle\,|0\rangle + \beta\hat{U}\,|1\rangle\,|0\rangle && (1.7)\\ &= \alpha\,|0\rangle\,|0\rangle + \beta\,|1\rangle\,|1\rangle\,. && (1.8)\end{aligned}$$

However, we also have the definition of what this cloning unitary operator should be able to do when it is applied directly to $|\psi\rangle$ from Eqn. 1.3:

$$
\begin{aligned}
\hat{U}\,|\psi\rangle\,|0\rangle &= |\psi\rangle\,|\psi\rangle & (1.9)\\
&= (\alpha\,|0\rangle + \beta\,|1\rangle)(\alpha\,|0\rangle + \beta\,|1\rangle) & (1.10)\\
&= \alpha^2\,|0\rangle\,|0\rangle + \alpha\beta(|0\rangle\,|1\rangle + |1\rangle\,|0\rangle) + \beta^2\,|1\rangle\,|1\rangle & (1.11)
\end{aligned}
$$

The results of Eqn. 1.6 and Eqn. 1.9 are different except for the very specific case when $\alpha = 1$ or $\beta = 1$, i.e. such a $\hat{U}$ does not exist.

1.4 Simplified BB84 implementation overview

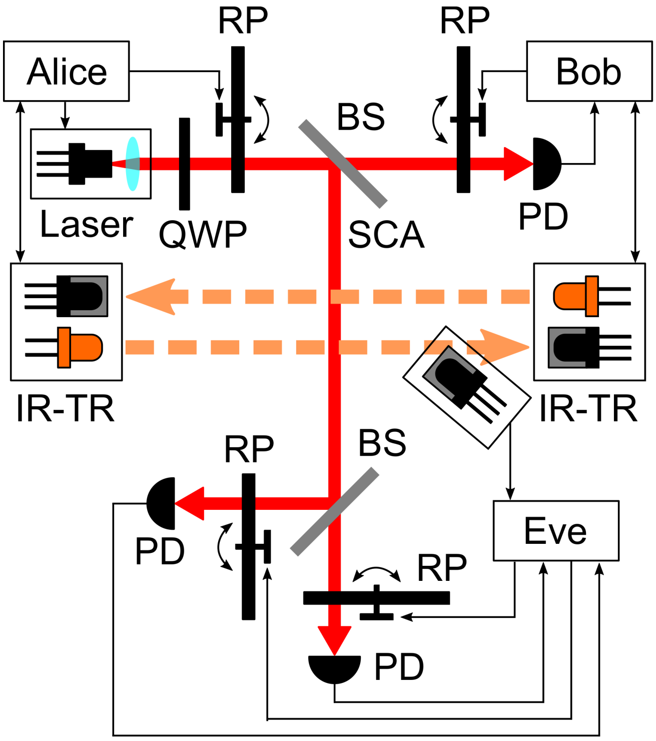


Figure 1.3: BB84 setup – quantum channel: Alice encodes a string of bits using different polarization choices set by her rotatable polarizer (RP). The quarter-wave plate (QWP) transforms the linearly polarized light from the laser diode into circular polarization such that the output from Alice's polarizer has equal intensity for all linear polarization choices. Bob projects Alice's photons into different polarization bases, and measures the corresponding intensity with a photodetector (PD).
Classical channel: Using infrared transceivers (IR-TR), Alice and Bob communicate the matched bases and the encrypted message.
Side channel attack (SCA): Using a beam splitter (BS), Eve intercepts and measures some of Alice's photons in two different bases simultaneously. As Eve's basis choice is *a priori* not aligned to Alice and Bob's, she may not be able to distinguish between polarization states optimally. However, by measuring in more than one basis choice simultaneously, she improves her ability to identify distinct polarization states even in the presence of laser intensity noise. She also intercepts the matched bases and encrypted message using her IR receiver from the classical channel.

In this experiment, we will implement a modified version of the original BB84 QKD protocol, shown in Fig. 5.1. Note that Fig. 5.1 shows a simplified schematic – more

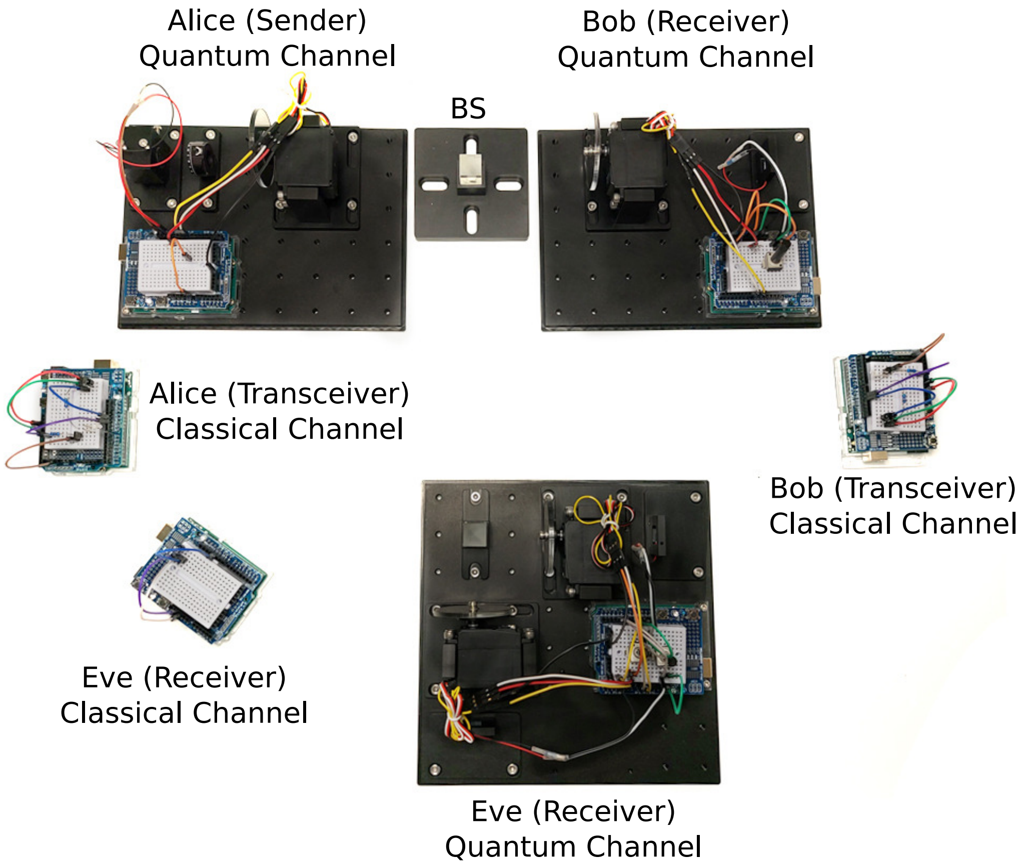


Figure 1.4: Photograph showing the layout of various components in the setup, without USB and power cables attached. BS: Beam splitter used by Eve to intercept some of Alice's photons in the quantum channel.

detailed schematic that facilitates the assembling of the constituent components, will be provided in the following chapters.

For ease of implementing the quantum channel, Bob will be measuring the polarization state of the photons using a motorized, rotatable linear polarizer, and a single photodetector. This contrasts with the EO-modulator, PBS and dual single-photon detector combination shown in Fig. 1.2.

Alice and Bob will also establish a public classical channel using infrared (IR) pulses by assembling IR transmitting and receiving circuits. Eve will attempt to eavesdrop on the quantum and classical channels using a beamsplitter and additional IR devices, respectively.

Apart from the implementation differences in the quantum channel, the steps involved in the BB84 protocol remains the same as the original protocol i.e. distributing and measuring polarization states in conjugate bases, and key sifting. However, we will not be performing any error correction or privacy amplification as the focus of the experiment is on the key distribution aspect of BB84. Nonetheless, the experiment is programmable and the end-user can in fact, write associated programs to perform these functions on the generated keys. Finally, you will also use the final key to encrypt a secret message. A key learning outcome is to determine if this simplified implementation compromises security in any significant way, despite following the steps outlined by the protocol.

1.5 Software

1.5.1 Installation

1. The software to operate this educational QKD system is hosted on Github at https://github.com/s-fifteen-instruments/s15_Qcamp2022. Download the software in your desired directory by opening command line and entering the following command. Assuming that you wish to install the program in C:

   ```
   C:\> git clone https://github.com/s-fifteen-instruments/s15_Qcamp2022
   ```

   This requires that Git is installed in your operating system – it is available at https://git-scm.com/book/en/v2/Getting-Started-Installing-Git.

2. The software requires a set of Python libraries to be installed. Navigate to the main programs folder s15_Qcamp2022 and execute the following command to download the relevant libraries:

   ```
   C:\s15_Qcamp2022> pip install -r requirements.txt
   ```

   Note that, if you are using the Raspbian distribution of Linux on a Raspberry Pi, please instead use:

   ```
   C:\s15_Qcamp2022> pip install -r requirements_raspbian.txt
   ```

Please do not hesitate to contact us at info@s-fifteen.com for assistance in this process if required.

1.5.2 Summary of programs

For convenience, we briefly list the the programs used to operate the QKD system in the subsequent chapters. Example execution codes are provided where appropriate. More information will be provided in the relevant chapters.

Note that, throughout the workshop, Alice will always take the role of the sender, and Bob the receiver during key distribution. Consequently, they will use different programs when the need arises.

The only scenario when they can take both roles is *after* the key distribution process, when they are using the final distributed key to send and receive encrypted messages over the classical channel. Of course, they may also chat over the classical channel in both directions at anytime, if only to test whether the channel is still working.

Chapter 2

`programs\1_Classical\led_on_off.py` – switches the infrared LED, used for establishing the classical channel, on and off. The appropriate COM port where the Arduino controlling the LED is connected to, should be provided. The command can be executed in the appropriate directory with:

```
programs\1_Classical> python led_on_off.py --serial COM5
```

In this example, the Arduino controller was located at COM5.

Note that, the path indicated by the example codes does not include the full path where the programs are installed. E.g. If the program folder is at `C:\s15_Qcamp2022`, you should for example, execute instead:

```
C:\s15_Qcamp2022\programs\1_Classical> python led_on_off.py --serial COM5
```

Please edit the codes accordingly to suit you purposes.

`programs\1_Classical\chatting.py` – Allows Alice and Bob to communicate unencrypted messages over the classical channel. Example execution:

```
programs\1_Classical> python chatting.py --serial COM5
```

Chapter 3

`programs\2_QuantumKey\AlignmentGUI\runReceiver.py` – Graphical user interface (GUI) used by Bob to receive polarized light transmitted by Alice. The GUI enables Alice and Bob to agree on a common coordinate axis used to define the orientation of horizontally polarized light.

`programs\2_QuantumKey\AlignmentGUI\runSender.py` – GUI used by Alice for the same purpose stated above.

`programs\2_QuantumKey\recv_calibrate.py` – Program used by Bob to generate the intensity matrix together with Alice. Start this code before Alice. Example execution:

```
programs\2_QuantumKey>python recv_calibrate.py -- serial COM5
```

`programs\2_QuantumKey\send_calibrate.py` – Program used by Alice to generate the intensity matrix together with Bob. Start this code after Bob. Example execution:

```
programs\2_QuantumKey>python send_calibrate.py -- serial COM5
```

Chapter 4

`programs\3_QKDComm\recv _key.py` – Program used to receive the raw QKD key by Bob. Example execution:

```
programs\3_QKDComm> python recv_key.py --serial COM5 --threshold 300
```

`programs\3_QKDComm\send _key.py` – Program used to send the raw QKD key by Alice. Example execution:

```
programs\3_QKDComm> python send_key.py --serial COM5
```

`programs\3_QKDComm\keysift _hint.py` – Program to assist in learning how the key-sifting process works.

`programs\3_QKDComm\encrypt .py` – Program to encrypt a plain text message with the final QKD key.

`programs\3_QKDComm\decrypt .py` – Program the decrypt the message with the final QKD key.

`programs\3_QKDComm\chatting .py` – Program to communicate encrypted messages over the classical channel. Example execution:

```
programs\3_QKDComm> python chatting.py --serial COM5
```

`programs\3_QKDComm\recv _32bitQKD.py` – Program to perform QKD with automatic key-sifting. Bob executes this program to initiate his receiver before Alice executes her program as the sender. Example execution:

```
programs\3_QKDComm> python recv_32bitQKD.py --Cserial COM5
 --Qserial COM8 --threshold 300
```

`programs\3_QKDComm\send _32bitQKD.py` Program to perform QKD with automatic key-sifting. Alice executes this program to perform her role as the sender over the quantum channel, but only after Bob standbys as the receiver by executing `recv_32bitQKD.py`. Example execution:

```
programs\3_QKDComm> python send_32bitQKD.py --Cserial COM5 --Qserial COM8
```

Chapter 5

`programs\4_HackTools\listener .py` – Program that Eve executes to listen in on the classical channel. Example execution:

```
programs\4_HackTools> python listener.py --serial COM5
```

`programs\4_HackTools\key _logger.py` – Program that Eve executes to eavesdrop on the quantum channel. Example execution:

```
programs\4_HackTools> python key_logger.py --serial COM5
```

`programs\4_HackTools\run _interceptor.py` – Program that Eve executes to analyse the data eavesdropped over the quantum channel.

Chapter 2

Building a classical channel

Objective: The objective of this part of the experiment is to set up a classical channel between Alice and Bob. You will send and receive messages encoded in a binary string sequence. The binary string sequences are translated into infrared (IR) pulsed sequences using a microcontroller and an IR LED. An IR receiver and another microcontroller are used to detect and decode the transmission. The classical channel is subsequently used in the QKD protocol for communicating basis choices during key-sifting, and for exchanging secret messages encrypted with the distributed quantum key.

2.1 Background

As described in Section 1.2, a key step in QKD requires a classical channel where Alice and Bob can use it to exchange their basis choices. Interestingly, since the security of QKD rests on the properties of the quantum channel, QKD remains secure even if Eve has access to the classical channel. The channel can even be publicly accessible, e.g. through a newspaper publication.

For ease of implementation, we have chosen to use IR transmitters and receivers, since they are ubiquitous components. Although not as convenient as communicating over commercially available chatting applications, the tangible aspect of constructing a classical communication channel from the ground-up, was also a factor in our choice in communication mode. In a typical QKD system, the classical channel can be realized by a standard TCP/IP connection.

2.2 Experimental Setup

Each party, Alice and Bob, will construct a sending and receiving unit for themselves, establishing a two-way communication channel. For each transmission direction, Fig. 2.1 shows the diagram of the circuit that you are to construct on top of a solder-less breadboard.

Each sending attempt comprises of the following encoding steps:

```
MESSAGE →ASCII ENCODED MESSAGE →NEC ENCODED MESSAGE
```

where an original MESSAGE is first digitized in ASCII, and is further encoded via the NEC protocol [17]. The NEC protocol encodes the ASCII message into a pulsed timing sequence. Each pulse is a 560µs long 38kHz carrier burst (Fig. 2.2), such that a 0-bit

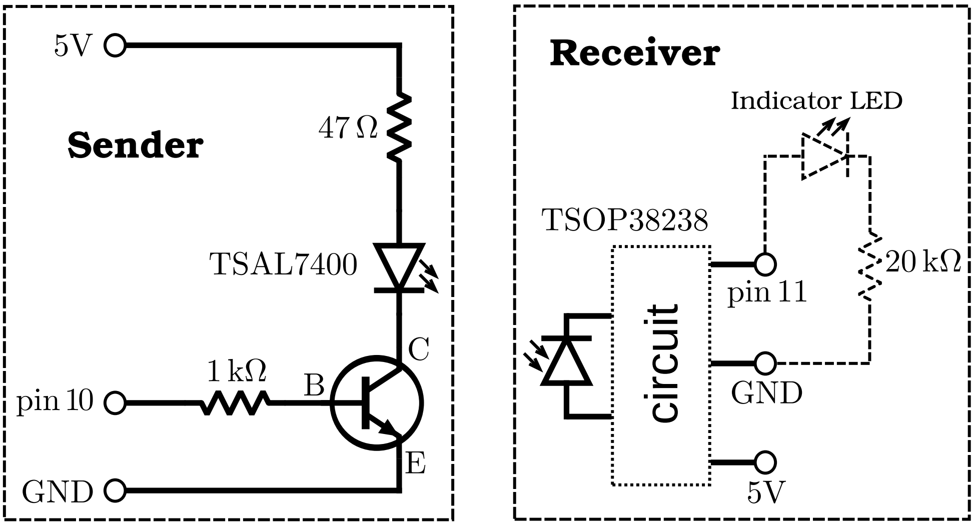


Figure 2.1: Electrical circuit of the IR sender and receiver: A series of IR pulses enables a sender and receiver to communicate with each other. The devices enable them to exchange classical information e.g. basis choices, encrypted messages. Information, encoded as binary strings, is translated into pulse sequences. A microcontroller uses the pulse sequences to switch the state of an IR LED to transmit the message. An IR receiver detects the pulses and decodes it with a microcontroller.

corresponds to a sequence of signal pulses with a timing characteristic distinct from that of 1-bit.[1]

Each receiving attempt comprises of the reversed process:

`NEC ENCODED MESSAGE →ASCII ENCODED MESSAGE →MESSAGE`

Both encoding processes have been implemented automatically whenever a classical message is sent over our system. You may find the encoding codes for ASCII, IR modulation, and NEC in `conv_ascii.py`, `IRremote.hpp`[2] and `IrSender.sendNECMSB`[3]

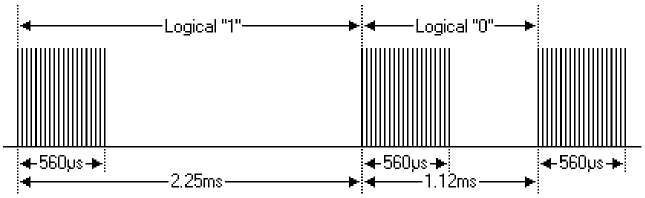


Figure 2.2: The NEC protocol uses pulse distance encoding of the bits. Each pulse is a 560µs long 38kHz carrier burst (about 21 cycles). A logical "1" takes 2.25ms to transmit, while a logical "0" is only half of that, being 1.125ms. Image credit [17].

[1] This modulation protocol, developed by the Nippon Electric Company (NEC), is ubiquitously adopted in remote control devices.

[2] Implemented as a library in `ArduinoClassical.ino`

[3] A class method belonging to the IRremote library.

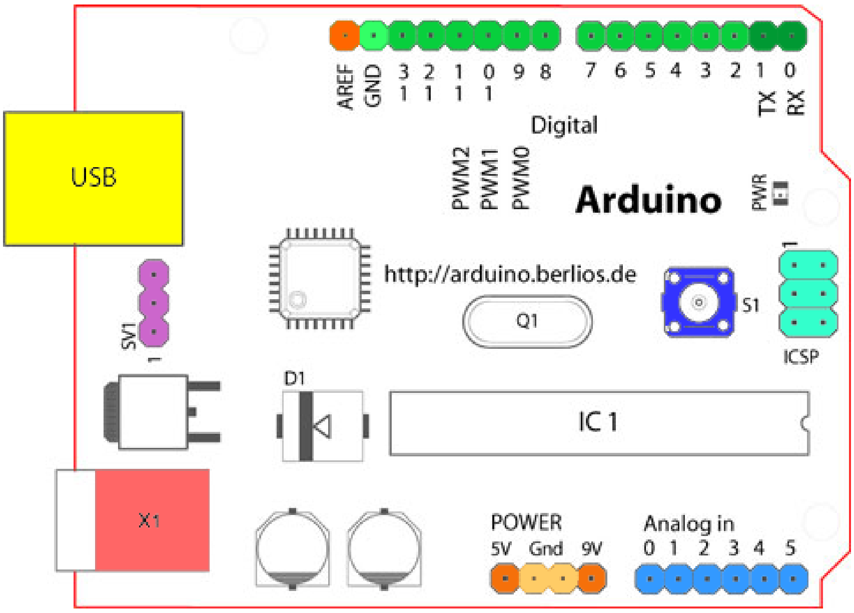


Figure 2.3: Parts of the Arduino microcontroller board.

2.3 Arduino Microcontroller & Breadboard

2.3.1 Arduino Microcontroller

Alice and Bob will each be given two Arduino microcontrollers, one for the classical channel, and the other for the quantum channel.

The board is based on the ATmega328P microcontroller chip, and comes with several digital and analog input/output (I/O) pins that may be connected to expansion boards (shields) and other circuits. The chip is programmable with the Arduino IDE (Integrated Development Environment) via a type B USB cable. The Arduino code used to control the various components have the file extensions `.ino`, and can be found in the `programs` folder. However, we will not be calling them directly when executing the QKD protocol. Instead, we are calling them from higher level functions written in Python, which we will describe throughout this manual.

Starting clockwise from the top center, the various parts of the board are [18]:

1. Analog Reference pin (orange)
2. Digital Ground (light green)
3. Digital Pins 2-13 (green)
4. Digital Pins 0-1/Serial In/Out - TX/RX (dark green) - These pins cannot be used for digital i/o (digitalRead and digitalWrite) if you are also using serial communication (e.g. Serial.begin).
5. Reset Button - S1 (dark blue)
6. In-circuit Serial Programmer (blue-green)
7. Analog In Pins 0-5 (light blue)
8. Power and Ground Pins (power: orange, grounds: light orange)

9. External Power Supply In (9-12VDC) - X1 (pink)
10. Toggles External Power and USB Power (place jumper on two pins closest to desired supply) - SV1 (purple)
11. USB (used for uploading sketches to the board and for serial communication between the board and the computer; can be used to power the board) (yellow)

More information about the Arduino microcontroller can be found at https://www.arduino.cc/en/reference/board.

2.3.2 Solderless Breadboard

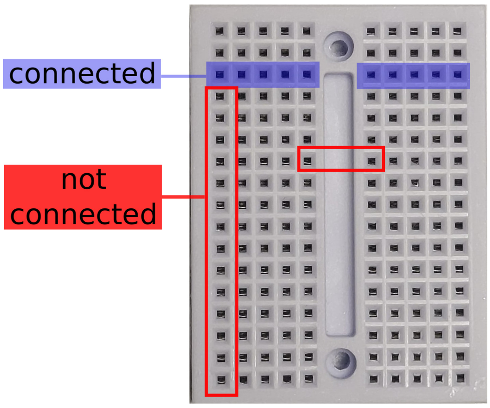
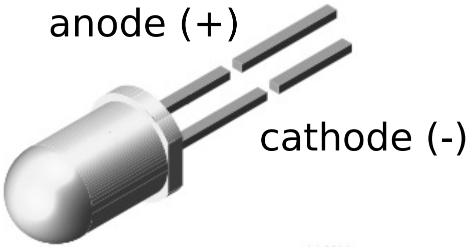


Figure 2.4: Breadboard: holes in the board are electronically connected to its neighbours as shown in the figure.

Alice and Bob will each assemble a sending and receiving unit (Fig. 2.1) on a solderless electronic breadboard. The holes on the board are electrically connected in the rows and columns as shown.

2.4 Task 1: Assembling the IR sending and receiving unit

2.4.1 Sender



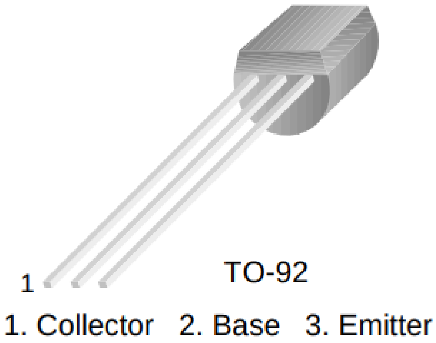


Figure 2.5: (Left) IR LED: Longer leg (anode) is to be connected to a higher potential. (Right) BC597 transistor, in a common TO-92 package, used to supply current to the IR LED.

```
programs\1_Classical>python led_on_off.py --serial COM5
Opening the serial port...
Done

LED basic on/off program
"LEDON" and "LEDOFF" to turn the LED on and off respectively
To exit the program, use Ctrl+C

Enter command here: ledon
Unknown command
Enter command here: ledon
LED ON!
Enter command here: ledoff
LED OFF!
Enter command here:
Keyboard Interrupt - Exiting...thank you for using the program!
```

Figure 2.6: Program to switch the IR LED on and off.

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 current to the IR LED ($\sim 60\,\text{mA}$) than the maximum allowable rating of the Arduino pin ($40\,\text{mA}$). 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. To switch on the IR LED via the transistor, we vary the base current using Arduino pin 10.

Warning: Note that the transistor needs to be connected at the correct polarity. An incorrect connection may result in damage to the component.

To test whether the circuit works, you can switch on the LED via the Arduino microcontroller, by executing `led_on_off.py`, (Fig. 2.6): you should be able to observe the infrared light using your mobile phone camera.[4] In the following example, the Arduino controller is connected to COM5.

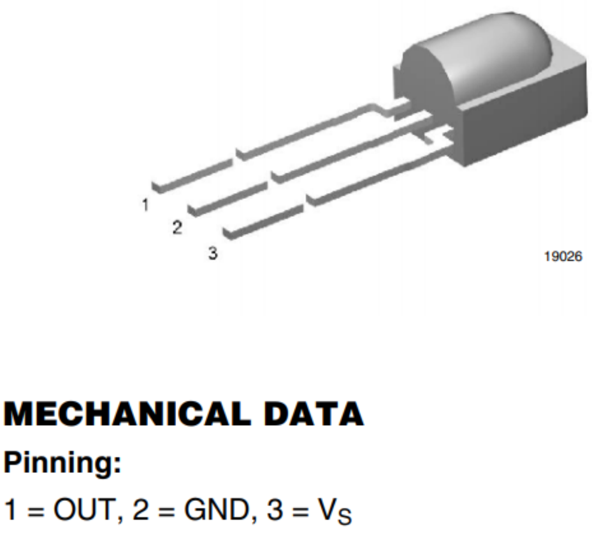
2.4.2 Receiver

Figure 2.7 shows the IR receiver module (TSOP38238). The module consists of an IR photodiode with a built-in demodulation circuitry.[5] The output (OUT) of integrated circuit / photodiode module can be directly connected to Arduino pin 11. The module is biased across the V_s and GRD pins by connecting them to the $5\,\text{V}$ and GRD pins of the Arduino board, respectively. *Warning: The IR receiver module may be damaged 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 cannot output more than a few milliamperes. Also, note that the pin logic is inverted, i.e. it turns OFF ($0\,\text{V}$ pin reading or LED off) when it receives an ON signal from the sender, and

[4] Certain models will be unable to detect IR light e.g. the iPhone which has an excellent IR filter.

[5] Whereas the pulse modulation from the sender is implemented on-board the Arduino.



MECHANICAL DATA

Pinning:

1 = OUT, 2 = GND, 3 = V_S

Figure 2.7: IR receiving unit. V_s connects to the voltage supply of 5 V, to be supplied on the Arduino board. GND can be assigned to the Arduino ground pin. The OUT pin is where the measured IR signal is read out using Arduino pin 11.

vice-versa.

2.5 Task 2: Sending messages back and forth between remote parties

The sending and receiving unit can now be tested by issuing the following commands:

We first prepare the receiver:

```
programs\1_Classical>python chatting.py --serial COM5
Opening the serial port...
Done

Qcumber ChatBox v1.00
To exit the program, use Ctrl+C

You are now in sending mode.
To change to listening mode, press ENTER.
Write the message you want to send below:
```

Once the receiver is ready, the sender may send a message e.g. “hello” using the following commands:

```
programs\1_Classical>python chatting.py --serial COM5
Opening the serial port...
Done

Qcumber ChatBox v1.00
To exit the program, use Ctrl+C

You are now in sending mode.
```

```
To change to listening mode, press ENTER.
Write the message you want to send below:
hello
Sending done!
```

Upon successful receipt of the message, the receiver should observe the following decoded message:

```
--- START OF TEXT ---
hello
--- END OF TEXT ---
```

Both senders and receivers should switch roles in order to test their corresponding sending and receiving circuits.

2.6 Discussion

1. While binary encoding was used to digitize the transmission, it was not intended to provide any encryption or authentication. Would implementing either encryption or authentication or both, on the classical channel, benefit the QKD protocol?
2. The classical channel uses light in the infrared regime, similar to the IR remote controllers in household electronic devices. Why is infrared light ubiquitously used for such applications? Why not say, visible or ultraviolet light?
3. Why does the IR light used to establish the classical channel have to be modulated? Can it function without modulation?

2.7 Further Exploration

1. * Measure the speed (in bytes per seconds) of the classical communication channel!
2. ** Measure the working distance of the classical communication channel. Explore ways to increase the working distance.
3. *** Measure the output from an IR remote controller when one of its functions is activated e.g. ON/OFF, volume, etc. With this knowledge, create your own custom programs and circuits to control the associated devices instead of using the intended remote controls!

Chapter 3

Building and characterising a simulated quantum channel

Objective: The objective of this part of the experiment is to set up a quantum channel between Alice and Bob. The quantum channel enables Alice to send a binary sequence, encoded in the polarization of laser pulses, to Bob. During the QKD protocol, we use this channel to transmit the raw key. There are two prerequisites to constructing this channel: First, Alice will have to construct the circuits required to generate laser pulses, and manipulate their polarization. Bob likewise, have to construct the corresponding circuits for measuring the polarization of the laser pulses. Second, Alice and Bob have to agree on the direction of polarization that is to be assigned the label 'Horizontal'. This direction serves as a reference for defining the other polarization directions 'Vertical', 'Diagonal' and 'Anti-diagonal'. To characterize the quantum channel, we will perform a visibility experiment which will tell us how distinguishable the polarization states are from each other.

3.1 Background

In this chapter, we establish a communication channel using polarization-state encoding. This encoding scheme differs from that for the classical channel, which instead encodes information using a timed sequence of LED pulses (Fig. 2.2).

The polarization-encoding scheme is as follows:

Basis	Alternate name	Polarization angle(°)	0	1
Rectilinear(+)	Horizontal Vertical (H-V)	0,90	↔	↕
Diagonal(×)	Diagonal Anti-Diagonal (D-A)	+45,-45	⤢	⤡

Table 3.1: Two non-orthogonal bases for polarization measurments. Measurements in the rectilinear (diagonal) basis will result in outcomes ↔(⤢) or ↕(⤡) which will encode bits of 0 or 1.

To measure the polarization state, we measure the intensity of the laser pulse after it passes through a linear polarizer. Note how this measurement scheme differs from that introduced in Fig. 1.2, where the result of the polarization state measurement is indicated by the beamsplitter port at which one of two detectors registers a photodetection event.

Measuring Polarisation

In this experiment, we will use a linear polarizer to determine the polarisation of light. A linear polarizer (also called an analyser) is allows only light polarized along its transmission axis to pass through (see Figure 3.1, left). Polarization orthogonal (90°) to its transmission axis is completely rejected (see Figure 3.1, right).

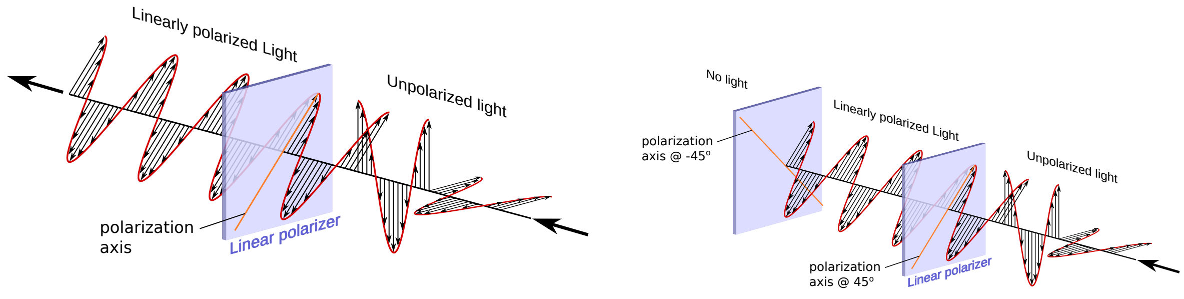


Figure 3.1: (Left) A linear polarizer allows only light polarized along its transmitting axis to pass through. (Right) Light oriented 90°to the transmitting axis of the second polarizer is not allowed to pass through.

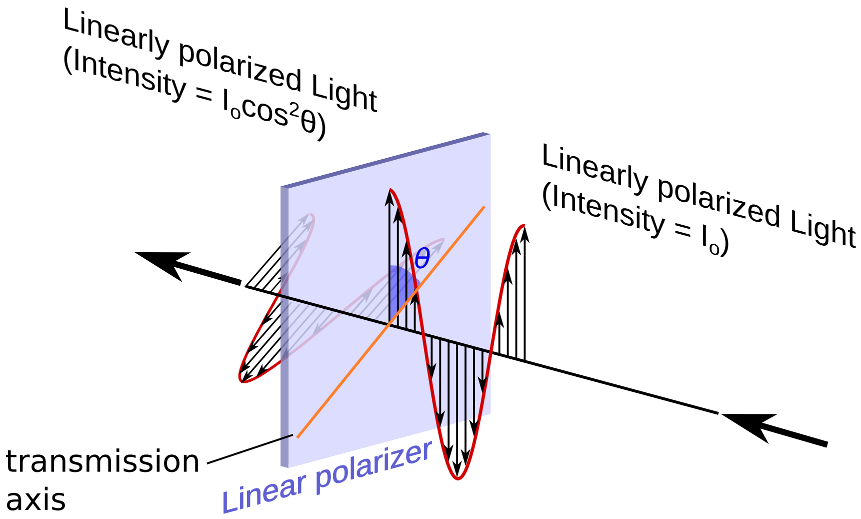


Figure 3.2: A setup similar to Figure 3.1, except that the linear polarizer (also known as an analyser) is used to ‘analyse’ how much of the light inpinging on it contains a component parallel to its transmission axis: the output intensity is a function of the relative angle θ between the input polarization and the transmission axis.

General Polarisation Measurement Linearly polarized light oriented θ degrees to the transmission axis will be transmitted with an intensity $I_o \cos^2\theta$, where I_o is the input intensity.

The output intensity $I_0 \cos^2\theta$ is due the following relations:
First, light intensity is proportional to the dot product of its electric field:

$$I_o \propto \vec{E_o} \cdot \vec{E_o}. \tag{3.1}$$

Second, the input electric field can be represented as:

$$\vec{E_o} = E_o \begin{pmatrix} \cos\theta \\ \sin\theta \end{pmatrix} \tag{3.2}$$

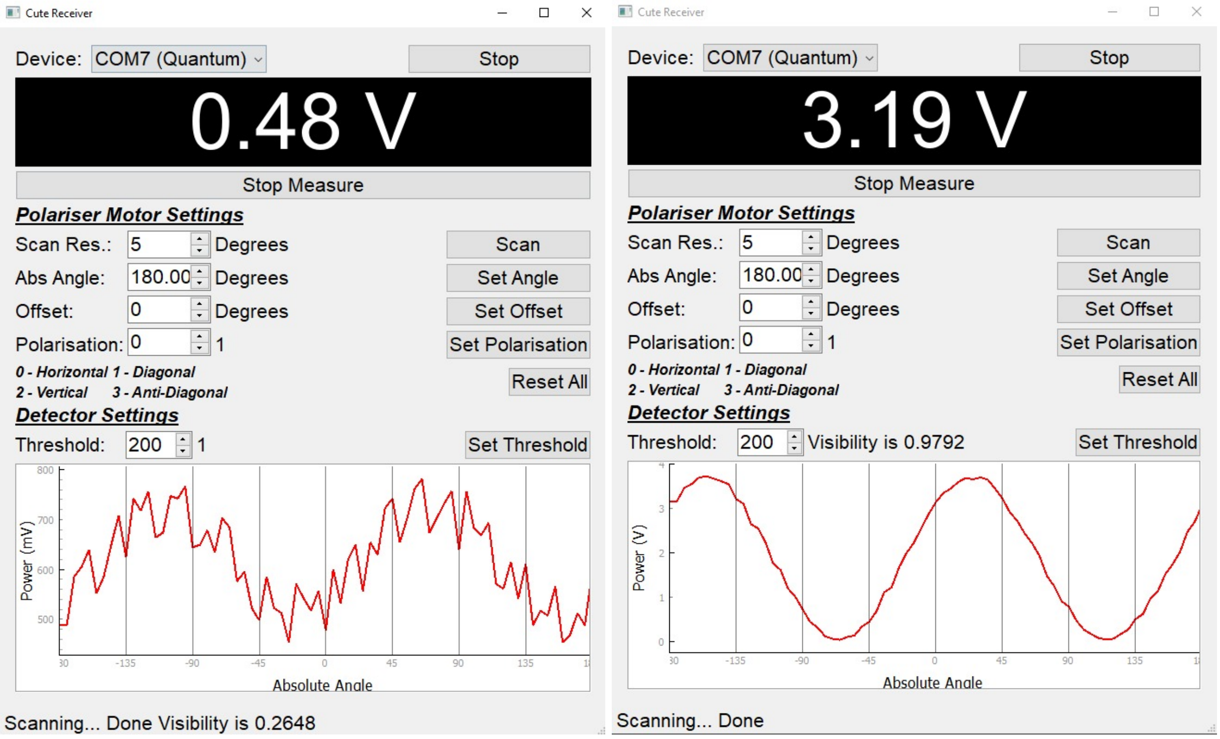
where the first row is the component of $\vec{E_o}$ parallel to the transmission axis, and the second row is the component orthogonal to the transmission axis. Since the analyser only transmits light parallel to its transmission axis, we only consider the first row $E_o \cos(\theta)$. The intensity associated with this electric field component is thus proportional to $\cos^2\theta$.

Thus, by making a relative comparison between the input and output intensity, we can infer the relative angle of the analyzer and the incoming light polarization.

Visibility measurement In a real experiment, the detected intensity will instead be:

$$I(\theta) = (I_o + I_{AC}) \cos^2\theta + I_{DC} + I'_{AC} \tag{3.3}$$

where I_{DC} is the detected light intensity to (i) background light unassociated with the source that we would like to detect (ii) light orthogonal to the transmission axis leaking through an imperfect analyzer, (iii) input light not perfectly linear e.g. eliptically polarized light whose component orthogonal to the transmission axis is small but still appreciable. Of particular concern is fluctuating noise, which is typically due to (iv) detector noise (I'_{AC}), (v) fluctuating energy output of the laser diode generating the incoming light (I_{AC}), or (vi) fluctuating transmission characteristics of the medium between the sender or receiver (I_{AC}).



Cute Receiver

Device: COM7 (Quantum)

0.48 V

Polariser Motor Settings

Scan Res.: 5 Degrees

Abs Angle: 180.00 Degrees

Offset: 0 Degrees

Polarisation: 0 1

0 - Horizontal 1 - Diagonal
2 - Vertical 3 - Anti-Diagonal

Detector Settings

Threshold: 200 1

Scanning... Done Visibility is 0.2648

Cute Receiver

Device: COM7 (Quantum)

3.19 V

Polariser Motor Settings

Scan Res.: 5 Degrees

Abs Angle: 180.00 Degrees

Offset: 0 Degrees

Polarisation: 0 1

0 - Horizontal 1 - Diagonal
2 - Vertical 3 - Anti-Diagonal

Detector Settings

Threshold: 200 Visibility is 0.9792

Scanning... Done

Figure 3.3: Visibility curves showing bad (left), and good (right) visibility. Each visibility curve shows the power (vertical axis) of Alice's horizontally polarized beam that passes through Bob's linear polarizer, at various angles (horizontal axis). The visibility measurements are performed using the setup (Fig. 3.4), controlled via the graphical-user-interface explained in Tables. 3.3 and 3.4. Notice that when visibility is poor, the signal to noise ratio is low. This is disadvantageous if we would like to infer the transmitted polarization state from Alice using intensity measurements. See the main text for tips to improve the setup in order to obtain a good visibility measurement.

To ensure that the noise level is tolerable enough to distinguish between the four polarization states used in the experiment, we perform a so-called visibility measurement.

Essentially, the output light intensity I is measured while the analyzer is rotated from $\theta = 0^\circ$ to 180°, so as to reconstruct the relation Eq. 3.3. We would like to observe that cosine term of the intensity curve is not obscured by intensity noise.

The figure of merit for this measurement is the visibility, which is defined as:

$$V = \frac{I_{max} - I_{min}}{I_{max} + I_{min}}, \tag{3.4}$$

where I_{max} and I_{min} are the maximum and minimum detected intensities, over the acquired intensity curve $I(\theta)$, respectively. In a simple, worse case scenario, $V = 0$, so that there is no way to distinguish between any of the polarization states.

When $V \sim 1$, the visibility measurement may serve as a tool to indicate when the transmission axis of both Alice and Bob's polarizers are aligned. This is useful when both parties would like to agree on a common coordinate system for their polarization states (Section 3.3).

		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.

Intensity matrix A particularly expedient alternative to performing a visibility measurement is to acquire a so-called intensity matrix.

This is the approach used to qualify the quantum channel for operation in this experiment. The intensity matrix is determined by measuring the intensity after the polarizer, at the various polarisation settings set by Alice and Bob.

The theoretical values for the intensity matrix is given in Table 3.2. As outlined in the previous section, due to imperfections of the devices (lasers, quarter wave plates, polarizers), the measured intensity matrix deviates from the expected value. We may define a metric called signal degradation η, which characterises the deviation of the measured with the ideal intensity matrix, with the following formulas:

First, given a measured intensity matrix whose elements are photodiode voltages d_{ij}, we define the mean of these elements by:

$$\bar{d} = \frac{\sum_{ij} d_{ij}}{N}, \tag{3.5}$$

where N is the total number of elements in the matrix.

We may now normalize the intensity matrix:

$$\eta_{ij} = \frac{d_{ij}}{\bar{d}} \tag{3.6}$$

Note that if we normalize the expected intensity matrix d_{ij}^{exp} (Table 3.2) in the same fashion, we obtain:

$$\eta_{ij}^{exp} = \frac{d_{ij}^{exp}}{\bar{d}^{exp}} = 2d_{ij}^{exp} \tag{3.7}$$

With the same normalization established between the measured and expected intensity matrix, we may now define the signal degradation as:

$$\eta = \frac{\sum_{ij} |\eta_{ij} - \eta_{ij}^{\mathrm{exp}}|}{N} \tag{3.8}$$

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

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

2. 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 $\eta \approx 0.5$.
3. 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. In this case, $\eta \approx 1$.

Thus, η represents, to some extent, the degree of misalignment and the 'visibility' of the signal. In our experiment, ideally we would require η to be as small as possible, but based on our experience, it is very hard to get $\eta \leq 0.1$. However, the experiment would still work fine if $\eta \leq 0.2$.

3.2 Task 1: Setup assembly and alignment

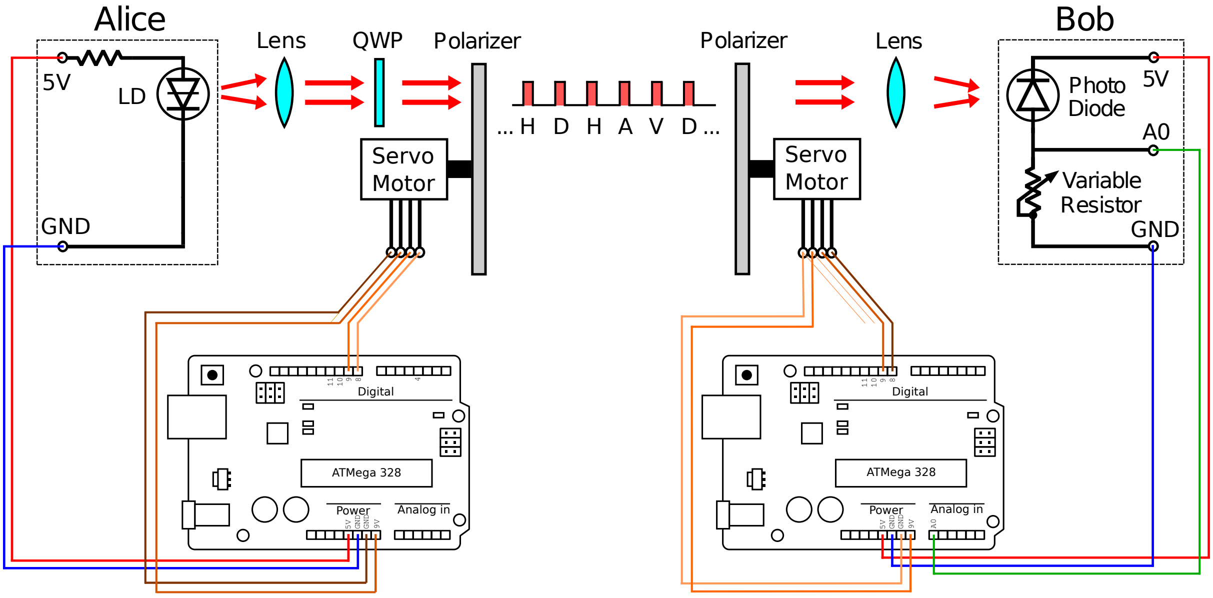


Figure 3.4: Schematic of the quantum channel. Alice transmits a series of polarized light pulses to Bob – created using a laser diode and a motorized polarizer. Bob projects the incoming states with his motorized polarizer and measures its intensity with a photodetector. The motors and laser diode are operated via the digital output on the microcontroller (DIG pins), while the photodiode readout is recorded by the analog-to-digital converter (ADC) on the microcontroller (A pins). QWP: Quarter-wave plate. GND: Ground connection.

Figure 3.4 illustrates the quantum channel. Alice prepares different polarization states by sending 650 nm laser diode pulses through a quarter-wave plate (QWP) and a motorized polarizer: The quarter-wave plate (QWP) converts linear-polarized light from the laser diode (LD) to circular-polarized light, which is subsequently converted into one of the four polarization states (H, V, D, A) using a motorized polarizer. To perform a measurement on incoming polarization states, Bob chooses one of two measurement bases (H/V or D/A) at random, and implements the basis with a motorized polarizer. Measuring the transmitted light intensity allows him to infer the polarization state.

Before using the quantum channel, both parties need to agree on a common coordinate system for their polarization states (Section 3.3). This is performed using a visibility measurement. First, Alice chooses an arbitrary rotation angle for her polarizer and defines it as the orientation of H-polarization. Then, she transmits a macroscopic beam with this polarization to Bob as a reference. To deduce Alice's orientation of H-polarization, Bob rotates his polarizer and identifies the position that maximises the transmission of the reference beam.

3.2.1 Polarized laser pulse sending and receiving unit

Sender

Figure 3.4 (Alice) shows the circuit diagram of Alice's sending unit. The electrical circuit for the sender consists of two main parts: laser module and polarizer module. The laser module consists of a 650 nm laser diode. The polarizer module consists of linear polariser mounted on and rotated by a servo motor [1].

Your task is to construct the circuit required to operate the laser and polarizer module with the Arduino controller using a breadboard and the components provided. Note the following descriptors for the wires of laser diode and servo motor. The pin assignments to Alice's Arduino are indicated in square braces [ ]:

Laser Diode:

1. Red: Anode (+) [Pin 12]
2. White: Cathode (-) [GND]

Motor:

1. Red: Voltage supply [Vin]
2. Black: Ground [GND]
3. White: Control signal [Pin 8]
4. Yellow: Feedback signal – reports current angle [Pin 9]

Receiver

Figure 3.4 (Bob) shows the circuit diagram of Bob's receiving unit. The receiver also consists of two main parts: the polarizer module (similar to the sender) and the detector module. At the polarizer module, light sent from Alice passes through the linear polarizer whose orientation, set by Bob's servo motor, implements the measurement basis. The incoming beam is focused using a converging lens, through the linear polarizer, and onto the photodiode in the detector module to reduce alignment sensitivity.

The detector module consists of a photodiode (SFH213) operating in reverse-biased mode, arranged in a voltage divider configuration in series with a variable resistor. The variable resistor allows the reverse-bias across the photodiode to be tuned, allowing you to change its sensitivity. When the photodiode is not saturated during normal operation, the light intensity on the photodiode is proportional to the voltage at A0.

Your task is to construct the circuit required to operate the detector and polarizer module with the Arduino controller using a breadboard and the components provided. Note the descriptors for the wires of the following devices. Their pin assignments to Bob's Arduino are indicated in square braces [ ]:

[1] https://www.pololu.com/product/3432

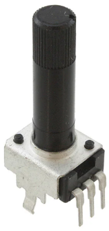


Figure 3.5: Variable resistor

Photodiode:

1. Red: Anode (+) [Pin A0]
2. White: Cathode (-) [5V]

Motor: Same as Alice's configuration.

Figure 3.5 illustrates the variable resistor. Note that you need only to connect the middle, and either of the two outer connectors, to the rest of the circuit. One of the connector leads to the GND pin on the Arduino, and the other connector, to the photodiode anode. This constructs a voltage divider circuit from which the photodiode signal is read out via its anode.

Note that a 1 kΩ resistor (not shown in Fig. 3.4) should be inserted in series with the photodiode and variable resister, in order to prevent the photocurrent from exceeding the recommended value.

3.3 Task 2: Aligning Alice's and Bob's coordinate systems

In this section, Alice and Bob will have to agree on a common coordinate system, so that their definition of the four polarization states, $\{|H\rangle, |V\rangle, |D\rangle, |A\rangle\}$, used to encode information in the quantum channel, is the same. i.e. the orientation of horizontally-polarized light for Alice is the same as Bob.

This alignment/calibration procedure requires Alice to send a polarized laser beam to Bob, who will perform the visibility measurement. The objective is to find the orientation of Bob's polarizer that corresponds to the maximum transmission of Alice's beam. This direction is then labeled 'Horizontal'. Note that this direction does not necessarily need to be parallel to the floor – it can be defined arbitrarily. However, once set, every other direction that is labeled 'Vertical', 'Diagonal' and 'Antidiagonal', will be defined with respect to this commonly agreed 'Horizontal' direction.

In the following, we will outline the steps that Alice and Bob performs to measure the visibility curve used to find this common orientation.

First, Alice will execute the following code to access the graphical interface (GUI) controlling the laser diode and motorised linear polarizer:

```
programs\2_QuantumKey\AlignmentGUI>python runSender.py
```

Likewise, Bob will execute the following to access the GUI controlling his polarization measurement module:

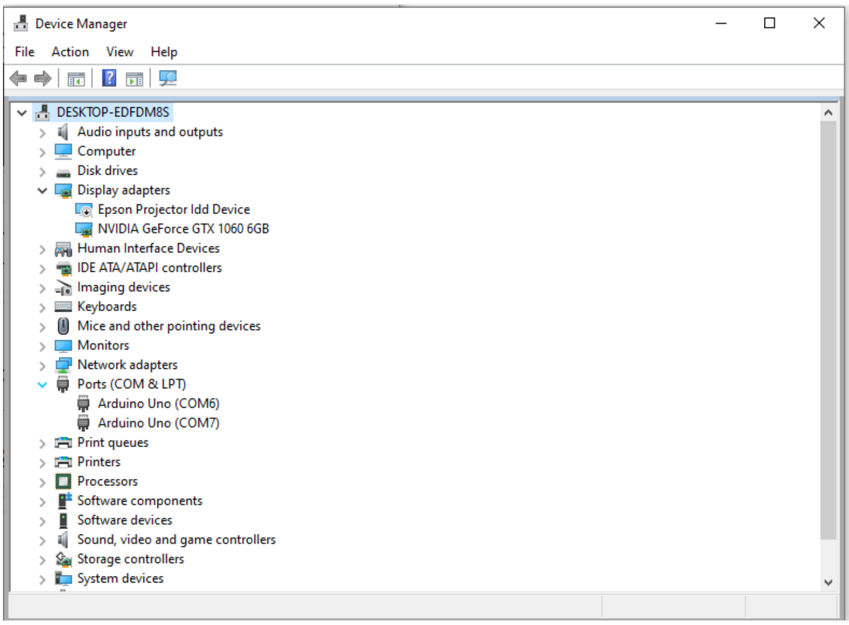


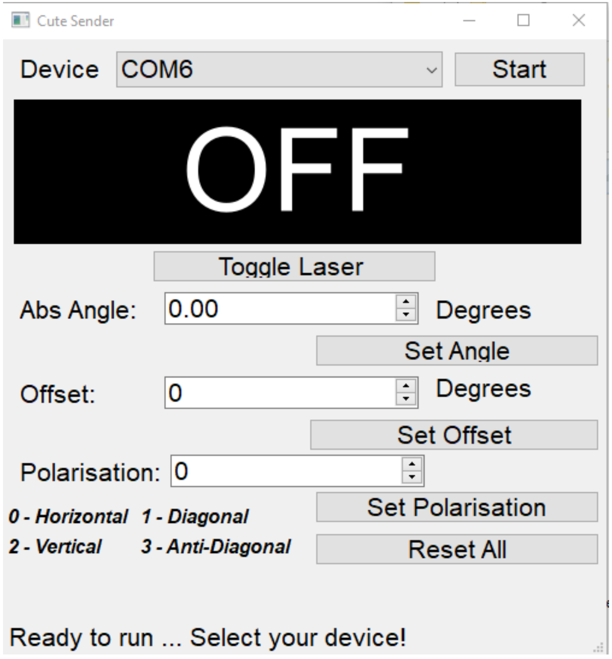
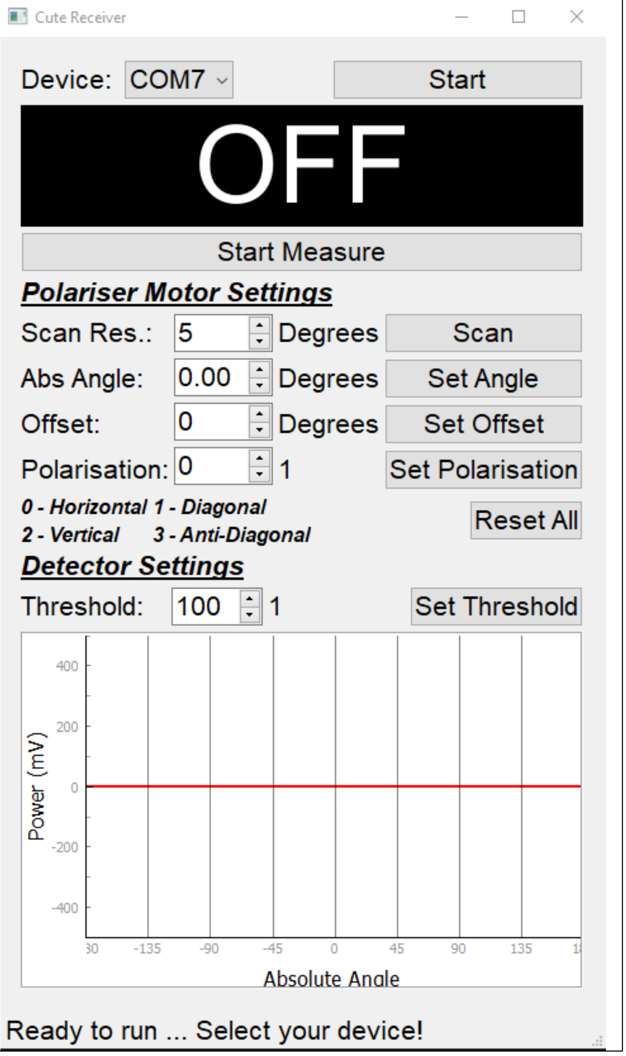
Figure 3.6: For Windows users: use the device manager to identify the COM port at which your Arduino device is located.

```
programs>python list_arduinos.py
The open ports are: ['COM3', 'COM5', 'COM6', 'COM7', 'COM8', 'COM9']
Identifying ports, please wait...
Ports identified!
['COM3 (Quantum)', 'COM5 (Classical)', 'COM6 (QuantumEve)', 'COM7 (Quantum)',
'COM8 (Classical)', 'COM9 (Classical)']
```

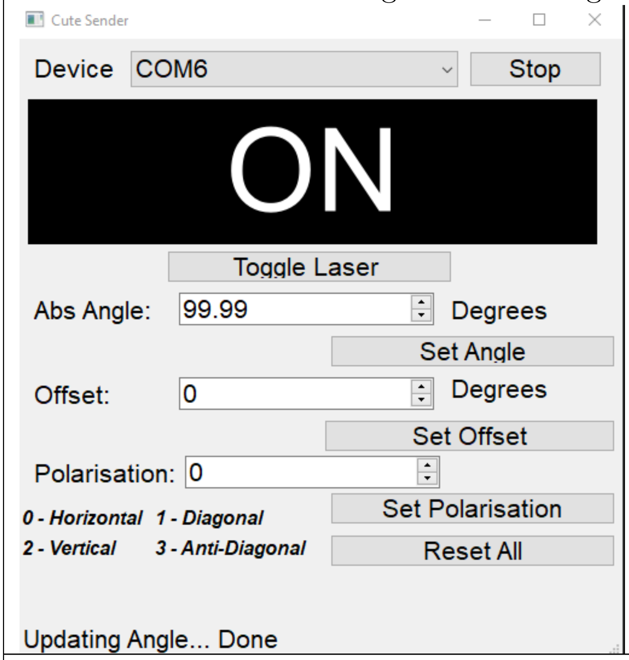
Figure 3.7: Program which identifies the COM port at which your Arduinos are connected to. Note that in this example, nine devices are connected and listed. For this workshop, Arduinos controlling the classical and quantum channel have their corresponding names listed in parenthesis. Whereas the Eve's quantum eavesdropping setup is listed as QuantumEve.

```
programs\2_QuantumKey\AlignmentGUI>python runReceiver.py
```

Each party will follow Steps 1–10 for the calibration procedure. Note that in Steps 1 and 2, Alice and Bob will have to locate the COM port (for Windows users) at which their Arduino microcontrollers are connected to. One way to do it is via the Windows device manager (Fig. 3.6). Alternatively, you may run `list_arduinos.py` to identify the devices (Fig. 3.7).

Alice	Bob
Step 1: Select the appropriate device from the com port assignment corresponding to Alice's Arduino.  Cute Sender Start OFF Toggle Laser Set Angle Set Offset Set Polarisation Reset All Device COM6 Abs Angle: 0.00 Degrees Offset: 0 Degrees Polarisation: 0 *0 - Horizontal 1 - Diagonal* *2 - Vertical 3 - Anti-Diagonal* Ready to run ... Select your device!	Step 2: Select the appropriate device from the com port assignment corresponding to Bob's Arduino.  Cute Receiver Start OFF Start Measure Scan Set Angle Set Offset Set Polarisation Reset All Set Threshold 400 200 0 -200 -400 Power (mV) 30 -135 -90 -45 0 45 90 135 Absolute Angle Device: COM7 ***Polariser Motor Settings*** Scan Res.: 5 Degrees Abs Angle: 0.00 Degrees Offset: 0 Degrees Polarisation: 0 1 *0 - Horizontal 1 - Diagonal* *2 - Vertical 3 - Anti-Diagonal* ***Detector Settings*** Threshold: 100 1 Ready to run ... Select your device!

Step 3: Click ’Start’, then click ’Toggle Laser’ to turn Alice’s laser diode on. Set the offset and polarization values to 0. It is not necessary to set the value for the absolute angle at this stage.



Cute Sender

Device: COM6

ON

Abs Angle: 99.99 Degrees

Offset: 0 Degrees

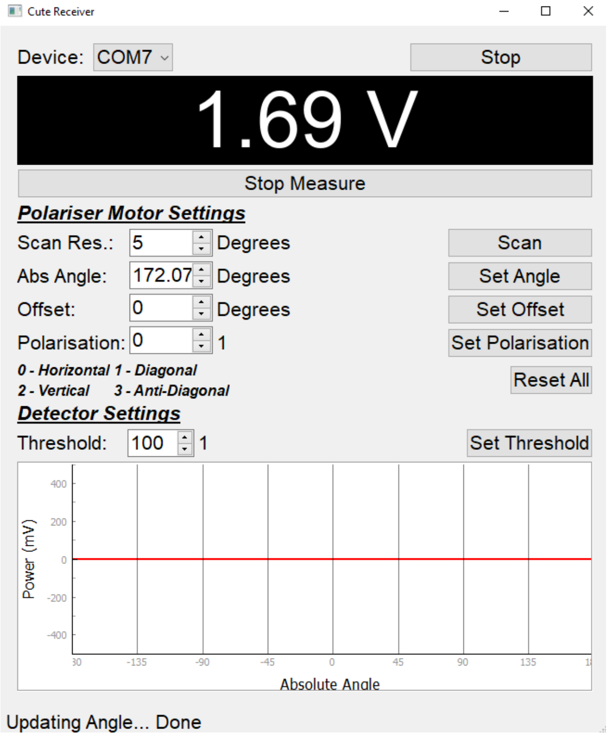
Polarisation: 0

0 - Horizontal 1 - Diagonal

2 - Vertical 3 - Anti-Diagonal

Updating Angle... Done

Step 4: Next, click ‘Start’, then click ’Start Measure’ to start the intensity measurement.



Cute Receiver

Device: COM7

1.69 V

Polariser Motor Settings

Scan Res.: 5 Degrees

Abs Angle: 172.07 Degrees

Offset: 0 Degrees

Polarisation: 0 1

0 - Horizontal 1 - Diagonal

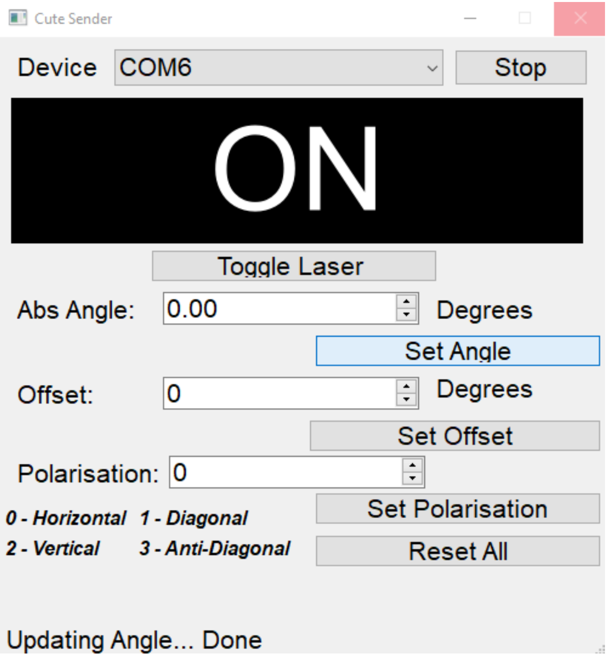
2 - Vertical 3 - Anti-Diagonal

Detector Settings

Threshold: 100 1

Updating Angle... Done

Step 5: Click ‘Set Angle’ to define the present transmission axis angle as 0°i.e. the ‘Horizontal’.



Cute Sender

Device: COM6

ON

Abs Angle: 0.00 Degrees

Offset: 0 Degrees

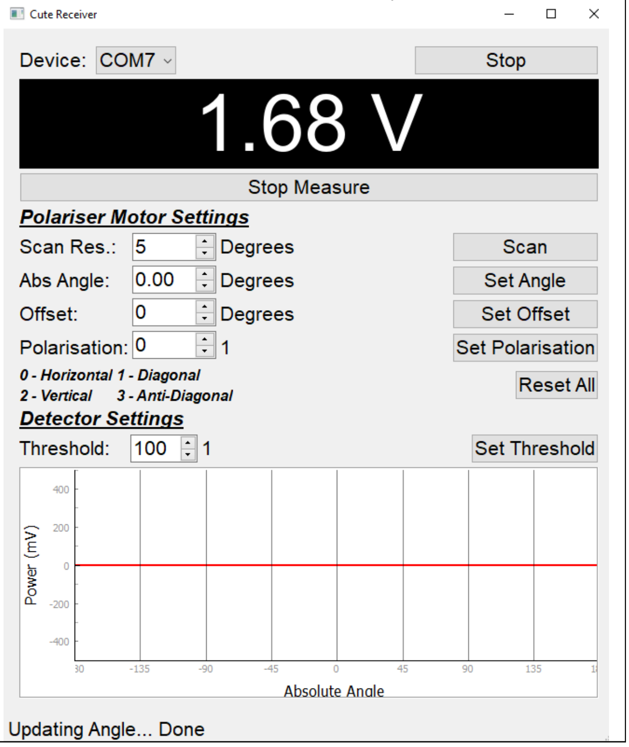
Polarisation: 0

0 - Horizontal 1 - Diagonal

2 - Vertical 3 - Anti-Diagonal

Updating Angle... Done

Step 6: Next, set the ’Abs Angle’ to 0°and click ‘Set angle’ to define Bob’s present transmission axis angle as 0°. Note that at this stage, Alice and Bob’s definition of 0°is not yet calibrated.



Cute Receiver

Device: COM7

1.68 V

Polariser Motor Settings

Scan Res.: 5 Degrees

Abs Angle: 0.00 Degrees

Offset: 0 Degrees

Polarisation: 0 1

0 - Horizontal 1 - Diagonal

2 - Vertical 3 - Anti-Diagonal

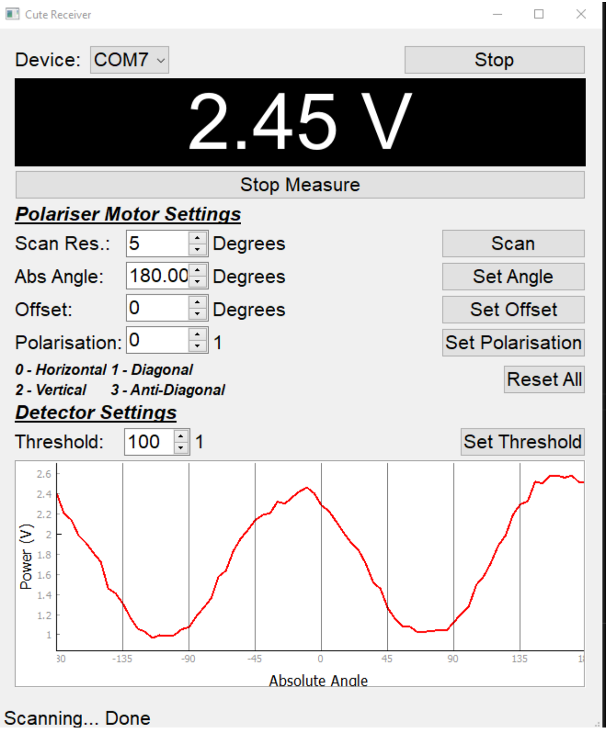
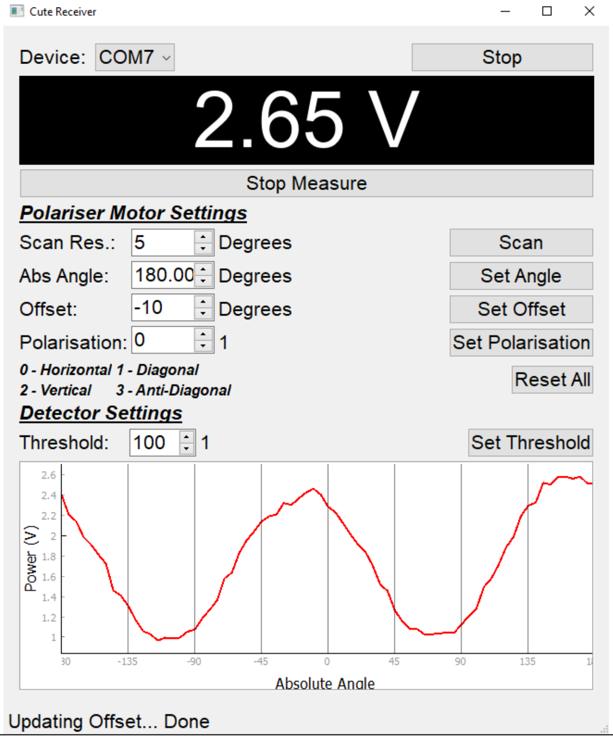
Detector Settings

Threshold: 100 1

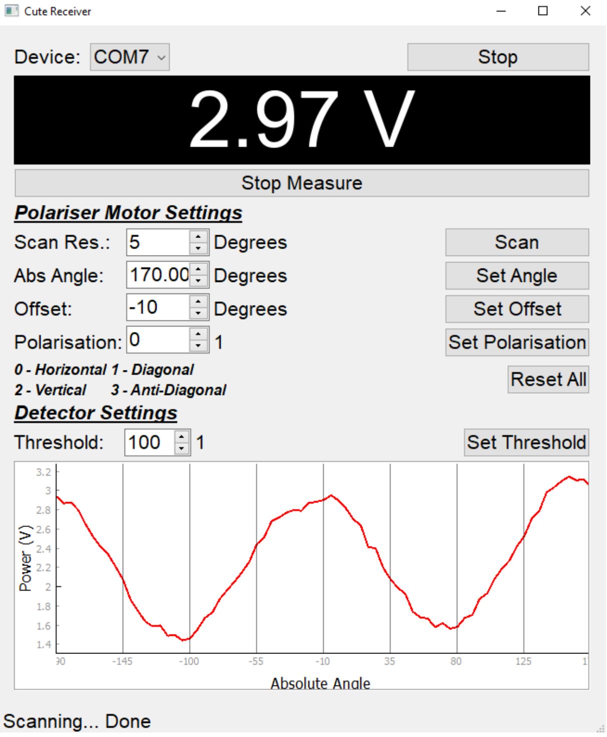
Updating Angle... Done

Table 3.3: Calibration Procedure (Part I) by Alice and Bob

Following which, Bob will continue to perform the following steps:

Bob	Bob
Step 7: Click 'Scan' to measure the intensity after the linear polarizer as it rotates through an angular range of 360°.  Cute Receiver Device: COM7 2.45 V *Polariser Motor Settings* Scan Res.: 5 Degrees Abs Angle: 180.00 Degrees Offset: 0 Degrees Polarisation: 0 1 *0 - Horizontal 1 - Diagonal* *2 - Vertical 3 - Anti-Diagonal* *Detector Settings* Threshold: 100 1 Scanning... Done Stop Stop Measure Scan Set Angle Set Offset Set Polarisation Reset All Set Threshold Power (V) 2.6 2.4 2.2 2 1.8 1.6 1.4 1.2 1 30 -135 -90 -45 0 45 90 135 1 Absolute Angle	Step 8: Observe the maximum of the transmission. In this example, it occurs at approximately −10°. Input this value in the 'offset' field, and click 'offset'. This informs Bob's Arduino that the angle corresponding to maximum transmission is on fact, not 0°, but -10°.  Cute Receiver Device: COM7 2.65 V *Polariser Motor Settings* Scan Res.: 5 Degrees Abs Angle: 180.00 Degrees Offset: -10 Degrees Polarisation: 0 1 *0 - Horizontal 1 - Diagonal* *2 - Vertical 3 - Anti-Diagonal* *Detector Settings* Threshold: 100 1 Updating Offset... Done Stop Stop Measure Scan Set Angle Set Offset Set Polarisation Reset All Set Threshold Power (V) 2.6 2.4 2.2 2 1.8 1.6 1.4 1.2 1 30 -135 -90 -45 0 45 90 135 1 Absolute Angle

Step 9: Click 'Scan' to measure the visibility curve again. You should observe that the maximum of the transmission occurs at 0°. i.e. Bob has indeed calibrated his coordinate system with Alice's.



Device: COM7

2.97 V

Polariser Motor Settings

Scan Res.: 5 Degrees

Abs Angle: 170.00 Degrees

Offset: -10 Degrees

Polarisation: 0 1

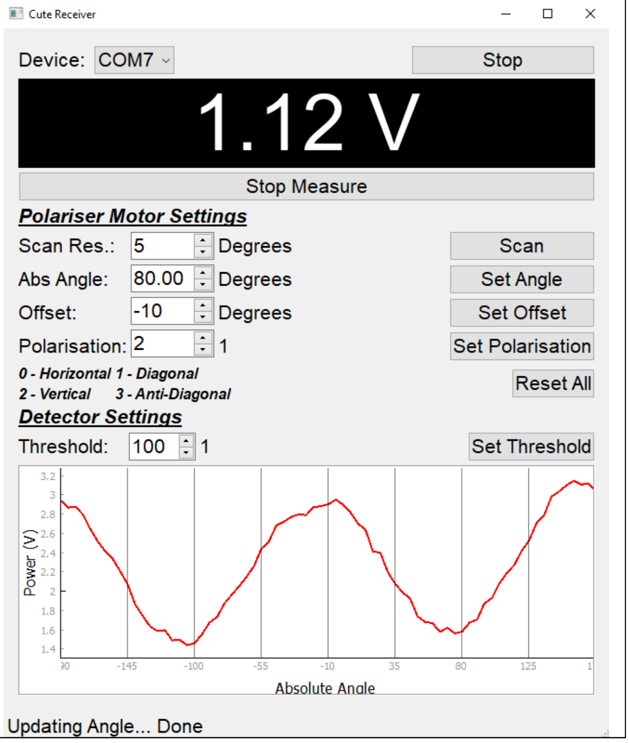
0 - Horizontal 1 - Diagonal
2 - Vertical 3 - Anti-Diagonal

Detector Settings

Threshold: 100 1

Scanning... Done

Step 10: Bob may set his polarization to '2', corresponding to setting the measurement direction to vertical polarization. He should observe a minimum intensity since the incoming polarization is horizontal if the calibration was done correctly.



Device: COM7

1.12 V

Polariser Motor Settings

Scan Res.: 5 Degrees

Abs Angle: 80.00 Degrees

Offset: -10 Degrees

Polarisation: 2 1

0 - Horizontal 1 - Diagonal
2 - Vertical 3 - Anti-Diagonal

Detector Settings

Threshold: 100 1

Updating Angle... Done

Table 3.4: Calibration Procedure (Part II) by Bob only

Improving visibility

Bob's ability to distinguish the various polarization states sent by Alice is indicated by the visibility V that you just measured. From Eqn. 3.4, we observe that (i) from the numerator $I_{max} - I_{min}$, a large amplitude indicates high visibility, while (ii) from the denominator $I_{max} + I_{min}$, a low mean position of the visibility curve indicates high visibility.

Guided by these "visual cues", attempt to improve the visibility with the following steps:

Reduce Ambient Light: Light from the sun, ceiling lights etc. will affect the readings on the photodetector. To reduce ambient light, reduce these ambient light sources, and/or use the beam blocker provided (little plastic piece that fits over the detector module).

Adjust the Quarter-wave Plate: The light entering Alice's rotating linear polarizer (and subsequently Bob's) should be circularly polarized. This ensures that linear polarized light generated at any orientation of polarization is generated with the same intensity. The circularly-polarized light is generated by transmitting the light emerging from the laser diode, which is at a fixed linear polarization, through a quarter-wave plate (QWP). The position of the QWP however, might be compromised by the end-user, or during device transportation. To ensure that the light after the QWP is indeed circularly polarized, first remove Alice's rotating linear polarizer from its motor with a M2.5 wrench. Then, on the receiver GUI, simply press 'Scan' to get the visibility reading. Ideally, visibility should be zero for perfectly circular light since intensity is the same in all polarization directions, and the graph would appear flat relative to detector noise. Rotate the QWP using its rotation mount and hit 'Scan'. Repeat until the visibility curve is as flat as possible. The alignment required is somewhat precise – a 1° rotation is sufficient to affect visibility measurements, use the markings on the QWP holder to keep track of your adjustments. In our tests, we find a visibility value < 0.1 is more than sufficient for normal functioning of the QKD Kit. After adjustment, replace Alice's polarizer and you're good to go!

Tune the bias of the photodetector: In the scanning step, if you observe that the peak of your visibility curve is much less than the saturation point of 5V, the bias on the photodetector can be adjusted to improve the amplitude of the visibility curve. First, use Bob's GUI to set the angle of the polarizer to coincide with peak transmission through his polarizer. This angle is indicated by the angle on the x-axis of the visibility curve when its amplitude is maximum. Next, rotate the knob on Bob's variable resistor to change its resistance. Do this until a value of about 4-4.5 V is measured with Bob's GUI. Doing so provides the maximum photodetector signal while avoiding signal saturation, since this value is close to the maximum bias that can been provided by the 5 V Arduino pin (Fig. 3.4).

3.4 Task 3: Pulse synchronization and intensity matrix calibration

In the previous section, Alice sent a continuous laser beam at a fixed polarization, defined as 'Horizontal', to Bob. Bob rotates his linear polarizer in order to observe the orientation at which the transmission of Alice's beam is maximum – this indicates to him Alice's 'Horizontal' direction.

In this section, Alice will send laser pulses of the four different polarization states, $\{|H\rangle, |V\rangle, |D\rangle, |A\rangle\}$, that will be used to encode information in the quantum channel. For each of these states, Bob will measure the intensity of the transmitted beam four different orientations of his linear polarizer, corresponding to $\{|H\rangle, |V\rangle, |D\rangle, |A\rangle\}$. The result of this measurement yields 16 intensities.

3.4.1 Synchronizing the transmission and measurement of pulse sequences

To prepare each polarized laser pulse to transmit each bit of data, the servo motor needs to move to the correct angle, and the laser (detector) needs to send (receive) each light pulse. The main timing bottleneck of this process is the rotation time of the servo motor (of the order of half a second) which depends on the rotation angle. This rotation time has already been reduced by rotating to the closest angle that reproduces the specified polarization, while limiting the maximum rotation angle to 90°.

After each rotation, Alice's laser produces a pulse for a duration of 0.3 s, while Bob's detector is set to measure at the 0.15 s mark (right in the middle of the laser pulse). In practise, Bob may not be able to measure precisely at the middle of the laser pulse, since his clock might not be perfectly synchronized with Alice. However, we have found that the above pulse parameters are sufficient for Bob to perform his measurements with Alice's pulses in a synchronized manner, given the typical frequency inaccuracies between their clocks, and the total measurement time for our pulse sequences. In other words, while the frequency inaccuracies of the clocks may cause Bob to detect the pulses off-center at each pulse, they are not sufficient to cause Bob to perform the detection outside of each pulse. The total measurement time for the pulse sequences allows for pulse sequences as long as 256 bits, but we have set it to 16 bits per pulse sequence for this experiment. These parameters have been implemented in our setup, and do not require further adjustment.

To start each sequence of bits, Alice sends a synchronisation pulse to Bob (500 ms on, 500 ,ms off, set to D polarisation). When Bob detects this pulse, he starts his own polarization measurement sequence. This synchronization procedure has been automated and does not need to be adjusted by the end-user.

3.4.2 Measuring the intensity matrix

To measure the intensity matrix, first, we prepare to receive the calibration signal. (In the following code examples, we assume Bob and Alice connect their individual Arduinos to COM5 of their respective computers.)

Bob:

```
programs\2_QuantumKey\AlignmentGUI>
python recv_calibrate.py -- serial COM5

Polarisation Calibrator (Receiver)
Uploading sequence to Arduino...
Opening the serial port...
Done
```

Next, Alice sends the calibration signal to Bob.

Alice:

```
programs\2_QuantumKey\AlignmentGUI>
python send_calibrate.py -- serial COM5

Polarisation Calibrator (Sender)
Uploading sequence to Arduino...
Opening the serial port...
Done
```

At the end of the procedure, you should observe an intensity matrix similar to Fig. 3.8.

Note that the matrix is normalized to a range of 0 to 1023, where 1023 corresponds to the maximum bias across the photodetector, which occurs when it is saturated. We recommend that the diagonal elements do not exceed 700–800. To achieve this, can adjust the variable resistor at the receiver unit to reduce the voltage bias across the photodetector.

Notice that in a practical scenario, the minimum value of the elements in the intensity matrix are not 0, even though they correspond to orthogonal polarization settings at the sender and receiver. This is due to polarizer and light source imperfections, in addition to ambient light impinging on the photodetector.

```
Receiver
           |  H  |  D  |  V  |  A  |
       | H | 802 | 634 | 116 | 319 |
Sender | D | 337 | 953 | 576 |  49 |
       | V |  60 | 407 | 912 | 512 |
       | A | 577 |  46 | 385 | 791 |

  The mean is 467.25
  Signal degradation is 0.192
```

Figure 3.8: Measured intensity matrix: the matrix is normalized to the range 0–1023. The mean value serves as a threshold to distinguish between the detection of orthogonal polarization states.

You can adjust the experimental setup using the checklist in Section 3.3 in order to reduce signal degradation. We recommend a signal degradation $\eta \leq 0.2$ to ensure that all the polarization states can be properly distinguished in the quantum channel.

Record the mean value calculated by the program. This value will be used as an **intensity threshold** which will serve to distinguish polarization states in Chapter 4. E.g. Notice that from the intensity matrix in Fig. 3.8, when H is the receiver's measurement setting, the intensities associated with the detection of H is 802, while the intensity associated with the detection of V is 116. With the intensity threshold 467.25, we may use the detected intensities to infer the polarization state sent.

3.5 Discussion

1. What are the properties of photons that make them good candidates for distributing quantum keys?
2. Besides using a rotatable linear polarizer, what other methods can we use to prepare and measure the polarization states?
3. What is the difference between the photodetectors implemented in this system, and single-photon detectors?
4. Not all the elements in the intensity matrix are required for determining if polarization states can be properly distinguished during key distribution. Which are these elements?

3.6 Further exploration

1. * The laser used in this experiment is a Class 2 laser (< 1 mW). Verify this claim by measuring the power output of the laser! How does the power vary with respect to the input voltage? Remember to not exceed 5V input voltage.
2. * Determine the conversion factor between the voltage across the photodiode and the incident power.
3. * Estimate how much error is introduced to the visibility measurement due to ambient light.
4. ** Write your own python program to automate the calibration procedure.

Chapter 4

Key distribution, sifting, and usage

Objective: In this chapter, we will use the experimental setup implemented thus far to perform QKD. We first distribute as many raw keys as required to create a final key of 32-bits[1].

The key sifting is first done manually with the aid of the “Key Sifting Worksheet” (included at the end of the manual) as an exercise, but will be automated later so that the QKD system can be used to quickly generate a 32-bit key useful for encrypting secret messages.

4.1 Task 1: Raw Key distribution

Bob first initiates the key distribution by preparing to receive the key. In this example, he prepares to receive 16 bits of information from Alice with 16 polarization basis settings, by initiating the `recv_key.py` program. These settings are encoded in a 16 bit string[2] referred to as the ‘basis bits’ in the program. Note that `recv_key.py` takes as input the threshold value measured in Section 3.4.2. (In the following code examples, we assume Bob and Alice connect their individual Arduinos to COM5 of their respective computers.)

Bob:

```
programs\2_QuantumKey>python recv_key.py --serial COM5 --threshold 468
Opening the serial port...
Done

Bob, Are you ready? This is the key receiver program.
Randomising basis bits using Arduino
OK

Arduino says he/she likes to choose the following bits:
('Basis bits (in hex):', '802e')
```

Now that Bob is ready, Alice sends a 16 bit string, referred to as the ‘value bits’ in the program. She does so with her own 16 polarization settings as reflected in her own value

[1] Recall that the final key tends to be shorter than the raw key, since for each bit of the raw key, it is kept for the final key only when the basis settings of Alice and Bob agree during its transmission.

[2] The 16 bit strings are represented in hex, to facilitate reading.

of the ‘basis bits’.

Alice:

```
programs\2_QuantumKey>python send_key.py --serial COM5
Opening the serial port...
Done

Alice, Are you ready? This is the key sender program.
Randomising key bits and basis bits using Arduino
OK

Arduino says he/she likes to choose the following bits:
Value bits (in hex): 8eb9
Basis bits (in hex): e151

Running the sequence...
OK

Task done. Please perform key sifting with Bob via public channel.
```

Once the transmission through the quanum channel is completed, Bob’s measurement result will automatically show up on the screen:

```
('Measurement result bits (in hex):', '8e80')

Task done. Please perform key sifting with Bob via public channel.
```

Note that at this stage, Bob’s measurement results differs from Alice’s measurement result, since there will be instances where Alice and Bob’s basis settings are different.

We proceed to the next step, key-sifting, which allows Alice and Bob to identify a subset of their value bits which are sent and received in the same basis.

4.2 Task 2: Key sifting (Manual)

4.2.1 Key Sifting Implementation

We will walk through the key sifting process in a “manual” step-by-step manner, by sifting as many keys as possible from a 16-bit long raw key. To facilitate learning, Bob will use a piece of paper to write and ‘transmit’ his basis bit string to Alice, who in turn, will transmit back the positions of his basis bits that correspond to her basis bit string. Note that the exercise in this section is done for pedagogical purposes only – in Section 4.4, where we will use the QKD system to send and receive encrypted messages, a 32-bit long final key will be distributed, with the key-sifting step done automatically.

Step 1: Bob ‘transmits’ his basis choice to Alice via a piece of paper.

Step 2: Alice receives Bob’s basis choice, and identities when his basis choices matches hers. By identifying these instances, she is able to ‘sift’ the value bits to arrive at the final, sifted key.

Step 3: Alice ‘transmits’ a 16 bit (hex encoded) string representing the instances when their basis match. This string is known as the 'matched basis' string in the program. Bob receives the matched basis string, and uses it to sift his measurement bit string in the same way Alice does (also known as the unsifted key). His final, sifted key should match Alice's.

After this process, Alice and Bob should have a symmetrical sifted final key. The key lesson here is to realize that, at no point during the protocol was the final, sifted key transmitted – both parties derived this key from the unsifted key by communicating their basis choices, which contain no information about the final key.

To aid the understanding of these steps, a program is used to (i) exhibit the hex strings in binary format, so that you may verify the sifting process bit-by-bit, and (ii) perform the key sifting by a logical AND operation between the unsifted key string, and the matched basis string. We encourage you to perform the bit-by-bit verification using the "Key Sifting Worksheet" found at the end of the manual. An example of how the form can be filled up, is included at the end of the manual as well.

An example of the program execution and output is provided:

Step 1

Bob converts his hex-formatted measurement result and basis bit string to binary format. He only sends his basis bit string to Alice through a public channel (piece of paper)

```
programs\2_QuantumKey>python keysift_hint.py
Key sifting simulator for our BB84 QKD Scheme
To exit the program, use Ctrl+C
1. Alice (sender)
2. Bob (receiver)
Your choice: 2

Bob, please input your measurement result (in hex, max 4 hex):
8e80
Binary representation: 1000 1110 1000 0000

Bob, please input your basis choice (in hex, max 4 hex):
802e
Binary representation: 1000 0000 0010 1110

Bob, send your basis choice to Alice through public channel!
Wait for the response from Alice! It contains the matched basis.
```

Step 2

Likewise, Alice converts her hex-formatted value bit and basis bit string to binary format. Next, upon receiving Bob's basis bit string, she inputs it into the program. The program automatically helps her to identify the bit positions where Bob's basis bits match her basis bits.

While the input is hex-formatted, its binary format helps students to match the basis manually by comparing both Alice and Bob's basis bit string, one bit at a time. This is faciliated by the "Key Sifting Worksheet" found at the end of the manual.

```
s15_Qcamp2022\programs\2_QuantumKey>python keysift_hint.py
programs\2_QuantumKey\keysift_hint.py

Key sifting simulator for our BB84 QKD Scheme
To exit the program, use Ctrl+C
1. Alice (sender)
2. Bob (receiver)
Your choice: 1

Alice, please input your value bits / unsifted key (in hex, max 4 hex):
8eb9
Binary representation: 1000 1110 1011 1001

Alice, please input your basis choice (in hex, max 4 hex):
e151
Binary representation: 1110 0001 0101 0001

Bob should have sent you something through public channel.

Alice, please input Bob's basis choice (in hex, max 4 hex):
802e
```

Once Alice inputs Bob's basis bit string into the program, it returns the string in binary format, and identifies the position where the basis bits match. 'Sifting' only her value bits at these matched basis bit positions, she obtains the sifted key bits.

```
Binary representation: 1000 0000 0010 1110

The matched basis is given below. Send this to Bob!
Hex representation: 9e80
Binary representation: 1001 1110 1000 0000

Matched key bits: 1XX0 111X 1XXX XXXX

Congrats! You obtain 6 bits key.
Sifted key bits: 0000 0000 0010 1111
Hex representation: 002f
```

Step 3

Alice's program automatically sends the matched key bits to Bob, and Bob's program sift only his measurement bits at the matched key positions, producing the following output:

```
Matched key bits: 1XX0 111X 1XXX XXXX

Congrats! You obtain 6 bits key.
Sifted key bits: 0000 0000 0010 1111
Hex representation: 002f
```

In the above example, 6 bits of final key was sifted from a 16 bit raw key. Due to the random nature of basis choice by both Alice and Bob, the number of final key that can be sifted from the raw key is also random.

In the subsequent section, we will (i) use the IR classical channel for key sifting and (ii) repeat and automate the above process such that we accumulate 32 bits of final key.

This 32 bit final key is then used to encrypt a secret message that will be transmitted via the IR classical channel.

4.3 Task 2: Automated key distribution and 32-bit key generation

In this section, we will generate a 32 bit final key between Alice and Bob, by automating the process outlined in Section 4.2.1. This key will be used to encrypt secret messages in the next section.

First, we prepare Bob to listen to Alice's messages sent through the IR classical public channel, by running `recv_32bitQKD.py`. Once Bob is ready, Alice will send a test message through the classical channel using `send_32bitQKD.py`. Once she receives an automatic acknowledgement from Bob that he receives her transmission, Alice commences sending raw key bits over the quantum channel.

An partial example output from both parties is as follows. Note that, in the steps below, we describe Bob on the left column since he first initiates the process by being ready to receive. (In the following code examples, we assume Bob and Alice connect their individual Arduinos to COM5 (Arduino associated with the classical channel) and COM8 (Arduino associated with the quantum channel) of their respective computers.)

Bob (receiver)

```
programs\3_QKDComm>
python recv_32bitQKD.py
--Cserial COM5 --Qserial COM8
--threshold 428

Opening the serial port...
Done

Hi Bob, are you ready?
Let's make the key!

Testing the public channel...
hex_string: 7070741
state: 0
hex_string: 54657374
state: 1
Test
Alice sends Test
You reply --OK!!--
OK!!

Public channel seems okay...
Sending the quantum keys...

Attempt 1
Generating random basis choices...
OK
```

Alice (sender)

```
programs\3_QKDComm>
python send_32bitQKD.py
--Cserial COM5 --Qserial COM8

Opening the serial port...
Done

Hi Alice, are you ready?
Let's make the key!

Testing the public channel...
You send --Test--
Test
hex_string: 7070742
state: 0
hex_string: 4F4B2121
state: 1
OK!!
Bob replies OK!!

Public channel seems okay...
Sending the quantum keys...

Attempt 1
Generating random polarisation
sequence...
OK
```

As described in Section 4.2.1, several rounds (attempts) is needed to accumulate 32-bit of final key. Once the procedure is completed, both parties should be presented with a final key:

```
DONE. The task is completed.
The 32 bit secret key is (in hex): 2237797c
```

4.4 Task 3: Secret message encryption and transmission

Encryption We implement an encryption method that belongs to a class of symmetric-key algorithm known as stream cipher (https://en.wikipedia.org/wiki/Stream_cipher). For the stream cipher method, the message is combined with a pseudo-random cipher, one at a time, typically with a bit-wise XOR operation. The pseudo-random cipher is generated from a random seed value supplied, in our case with the QKD key. This allows us to "expand" the shorter QKD key, into the longer pseudo-random cipher useful for encrypting longer messages.

However, since we used a small initial key length (seed value) for the stream cipher, is not cryptographically secure. Nonetheless, we chose this implementation as the focus of this educational workshop was to provide a demonstration of BB84, and not on secure implementation of the stream cipher method. An example of a stream cipher that is much more cryptographically secure is Salsa20 or ChaCha (https://en.wikipedia.org/wiki/Salsa20), which require at least 128 bits of key length. One of the most widely-used symmetric-key algorithm is AES, which is a block cipher, and require at least 128 bits of key length as well.

Decryption

To decrypt the message, we perform a bit-wise XOR between the "expanded key" and the encrypted text (represented in binaries). This process is basically identical to the encryption process (in the binary representation). This is because performing an XOR operation twice on a message gives back the message.

4.4.1 Encryption Implementation and Transmission

We first prepare Bob to receive any encrypted messages through the classical public channel using `chatting.py`:

```
programs\3_QKDComm>python chatting.py --serial COM5
Opening the serial port...
Done

Qcumber ChatBox v1.00
To exit the program, use Ctrl+C

You are now in sending mode. To change to listening mode, press ENTER.
Write the message you want to send below:
```

```
You are now in listening mode.
Waiting to receive the message...
```

Once Bob is ready, Alice first uses the 32-bit key generated in Section 4.3 to encrypt a desired secret message. The encryption process is executed using `encrypt.py`:

```
programs\3_QKDComm>python encrypt.py

Welcome to encrypt! Please enter the secure key (in 32 bit hex):
2237797c

Please write down the message to encrypt below:
I have the high ground

The encrypted message is:
8224f2c9901f6c4e1b8828a1dc7f67a520f60b53f60e

Task completed. Thank you for using the program!
```

Alice then transmits the encrypted message through the classical channel using `chatting.py`:

```
programs\3_QKDComm>python chatting.py --serial COM5
Opening the serial port...
Done

Qcumber ChatBox v1.00
To exit the program, use Ctrl+C

You are now in sending mode. To change to listening mode, press ENTER.
Write the message you want to send below:
8224f2c9901f6c4e1b8828a1dc7f67a520f60b53f60e
Sending done!
```

4.4.2 Decrypting Implementation

With Bob in the listening mode (see previous section), he would receive the encrypted message once Alice transmit it over the classical channel:

```
You are now in listening mode.
Waiting to receive the message...

--- START OF TEXT ---
8224f2c9901f6c4e1b8828a1dc7f67a520f60b53f60e
--- END OF TEXT ---

You are now in sending mode. To change to listening mode, press ENTER.
Write the message you want to send below:

Thank you for using the program!
```

To decrypt the message, he inputs the encrypted message and his copy of the 32-bit final key into decrypt.py, which returns the decrypted message.

```
programs\3_QKDComm>python decrypt.py

Welcome to decrypt! Please enter the encrypted text:
8224f2c9901f6c4e1b8828a1dc7f67a520f60b53f60e

Now, please enter the key (in 32 bit hex):
2237797c

The decrypted message is:
I have the high ground

Congratulations! Thank you for using the program!
```

4.5 Discussion

1. The 16-bit raw key, basis choices, and measurement results are presented in the program as 4 hexadecimal characters. Why do you think we presented them in this way, and are there other alternatives?
2. In this implementation, after the "quantum" channel transmission, Bob sends his basis choices to Alice, and Alice replies with the matched bases. Are there any implications if Alice replies with her basis choices instead?
3. Will QKD security be compromised if an adversary listens in on the classical channel?
4. In our setup, we generate approximately 8 bits of final key from 16 bits of raw key in every cycle. We repeat the cycles approximately four times to obtain a 32-bit key. Is it better to generate the 32-bit final key in a single cycle?
5. A common method for encrypting messages with a quantum key is the one-time pad. How does the method introduced in our setup differ with the one-time pad?
6. What role does a pseudo-random number generator play in message encryption? What are the possible drawbacks with this encryption method?

4.6 Further exploration

1. ** Use a random number generator testing suite to explore how random the basis settings are.
2. ** Estimate and measure the key generation rate of the setup! What are the limiting factors in our setup? Compare this with the other QKD systems!
3. *** Implement an error correction algorithm to correct for any errors in transmitting the raw key.
4. **** In our setup, we skip the information reconciliation and privacy amplification steps, as we have focused on the key distribution aspect of the QKD protocol. Design and implement a basic form of information reconciliation and privacy amplification in our setup!

Chapter 5

Eavesdropping of a weak quantum cryptography setup

Objective: In this chapter, we describe the setup used by Eve to eavesdrop on quantum channel, and listen in to the classical channel during the key distribution process. (Technically, the classical channel is public, so Eve is not really eavesdropping on the channel.) The objective of Eve is to be able to deduce the distributed key, and use that to decipher secret messages transmitted over the classical channel after the key distribution procedure.

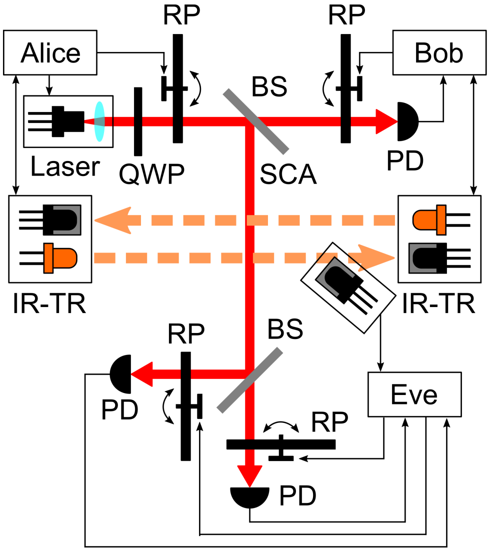


Figure 5.1: Side channel attack (SCA): Using a beam splitter (BS), Eve intercepts and measures some of Alice's photons in two different bases simultaneously. As Eve's basis choice is *a priori* not aligned to Alice and Bob's, she may not be able to distinguish between polarization states optimally. However, by measuring in more than one basis choice simultaneously, she improves her ability to identify distinct polarization states even in the presence of laser intensity noise. She also intercepts the matched bases and encrypted message using her IR receiver from the classical channel.

Figure 5.1 shows the overall setup which involves Alice and Bob executing the BB84 protocol using both the classical and quantum channels, and also Eve's attempt at intercepting both channels.

To intercept the basis choice sequence transmitted over the classical channel, Eve simply constructs the IR receiver already described in Chapter 2.

To eavesdrop on the quantum channel, Eve performs a so-called side-channel-attack (SCA), where she samples some of the photons from the quatum channel and attempts to measure the polarization state transmitted from Alice to Bob using a beam splitter (BS).

Eve performs the polarization state measurement simultaneously using two bases: first, by splitting the beam equally with a 50:50 beam-splitter, then by measuring the transmitted intensities through two linear polarizers, each oriented at different angles, using two photodiodes.

The reason why it is advantageous for Eve to perform her measurement in two different bases, instead of one, is that it allows Eve to associate a pair of intensity measurements to every polarization state she intercepts.

Recall that Eve did not participate in the polarization calibration that Alice and Bob did in Section 3.3. i.e. Eve's polarizers will not be optimally orientated to distinguish between the four polarization states used to encode the distributed key. Consequently, the signal-to-noise ratio of her measurements could compromise her ability to assign a polarization state to her intensity measurements accurately, especially if she relies on only a single linear polarizer and detector.

Automatically identifying clusters of measurements

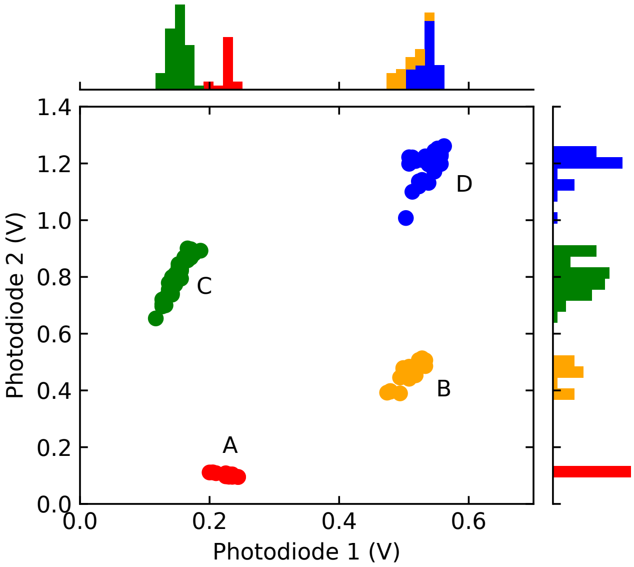


Figure 5.2: Four identified clusters of signal voltages measured by Eve's dual photodiodes. Each cluster represents a polarization state intercepted in the quantum channel and are arbitrarily assigned A, B, C, D. Each polarization state represents H, V, D or A sent from Alice to Bob. By assigning the correct polarization state through **trial and error**, Eve is able to derive the transmitted key. This is done via obtaining intelligible messages by decoding the ciphers transmitted through the classical channel.

Figure 5.2 shows the graph titled "Signal: Height Clusters", which plots the potential difference across the two photodiodes, Photodiode 1 and Photodiode 2. We notice that the intensity distribution (plotted along the marginals of Fig. 5.2) of Photodiode 1 does not allow the four polarization states to be fully distinguishable, whereas the intensity distribution of Photodiode 2 happens to allow good distinguishability.

To identify four clusters associated with the four polarization states in this 2-dimensional space, we use a K-means clustering algorithm to identify four distinct groups. The clustering algorithm essentially identifies four distinct clusters that correspond to the four polarization states intercepted. The algorithm iteratively computes the location of the four clusters in order to minimize the distance between each datapoint, and the average position of its assigned cluster [19].

The output of the clustering algorithm arbitrarily assigns a label A, B, C, D to the clusters, which serve as placeholders for the actual polarization states H, V, D, A. By permuting through the possible identities of A, B, C and D, we result in different polarization sequences associated with the sequence of 2-dimensional intensity values. The correct permutation results in a sequence of bits that, when sifted with the basis sequence communicated over the classical channel, results in a final key that deciphers secret messages into legibles messages.

The following sections describe Eve's circuit used for performing polarization state measurements on the intercepted transmission from the quantum channel (Task 1), and using the measured intensities to correctly decipher encrypted messages from the classical channel (Task 2) using the ideas discussed so far. Let's begin!

5.1 Task 1: Assembling the eavesdropping setup

Figure 5.3 illustrates Eve's dual polarization-state measurement units. Similar to Bob's measurement unit, each of Eve's measurement unit comprises of a detector module and polarizer module. Both modules are controlled by a single Arduino controller board. Please refer to Sections 2.3.1 and 2.3.2 for more details on the Arduino board and solderless breadboard.

Your task is to construct the circuit required to operate the polarization measurement unit with the Arduino controller using a breadboard and the components provided. For the servo motors in the polarizer modules, the control and feedback wires are to be assigned to the following Arduino board pins:

Motor 1:

1. Red: Voltage supply [Vin]
2. Black: Ground [GND]
3. White: Control signal [Pin 8]
4. Yellow: Feedback signal – reports current angle [Pin 9]

Motor 2:

1. Red: Voltage supply [Vin]
2. Black: Ground [GND]
3. White: Control signal [Pin 4]
4. Yellow: Feedback signal – reports current angle [Pin 5]

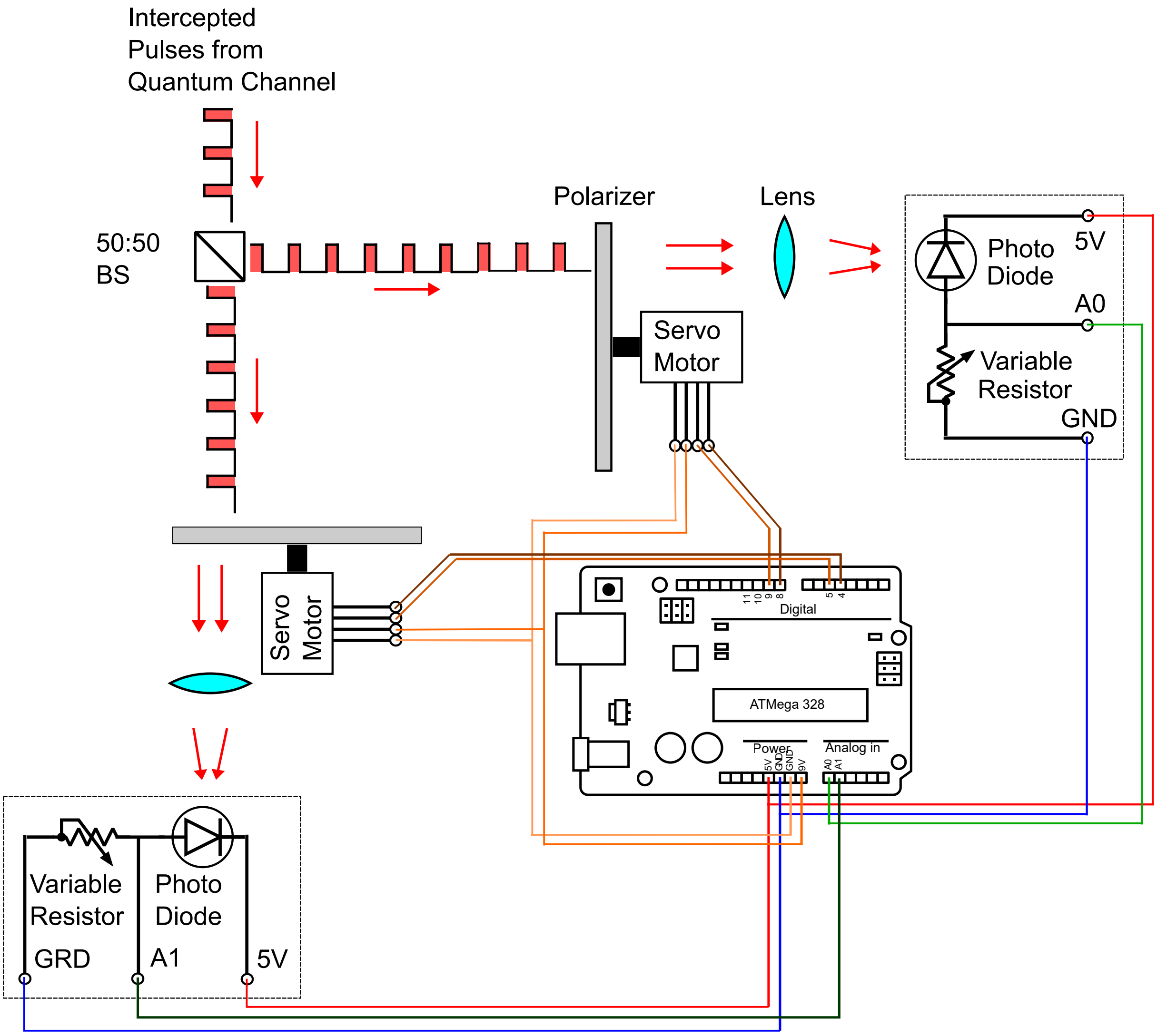


Figure 5.3: Schematic of Eve's unit for measuring the polarization state of the intercepted laser pulses from the quantum channel. A 50:50 beamsplitter(BS) allows Eve to measure the polarization of the laser pulses in two bases simulataneously.

whereas the output of photodiodes 1 and 2 in the detector modules, are to be connected to Arduino board pins A0 and A1, respectively.

5.2 Task 2: Eavesdropping on the channels, and reconstructing encrypted messages

5.2.1 Eavesdropping the classical channel for the matched basis string

Eve begins by listening, as soon as possible, for matched basis transmissions over the classical channel:

```
programs\4_HackTools\Classical_Listener>python listener.py
Opening the serial port...
Done

IR Listener for Qcumbers
To exit the program, use Ctrl+C
Waiting for data ...

Incoming message from Alice:
Test
Incoming message from Bob:
OK!!
Incoming message from Alice:
RDY!
```

Once Alice and Bob initiates their automated key distribution procedure, you will find that the programme returns the messages in their classical channel. The following exchange begins with several cycles of raw key distribution and key-sifting to accumulate a 32-bit final key, and ends with the use of this final key in transmitting an encrypted message:

```
Incoming message from Alice:
RDY!
Incoming message from Bob:
7075
Incoming message from Alice:
fcef
Incoming message from Alice:
RDY!
Incoming message from Bob:
b717
Incoming message from Alice:
8c4c
Incoming message from Alice:
RDY!
Incoming message from Bob:
33f8
Incoming message from Alice:
```

```
5c38
Incoming message from Alice:
RDY!
Incoming message from Bob:
e956
Incoming message from Alice:
6ae9
(...)
--- START OF TEXT ---
c31f015f5c9104af7adb244359fc382c93899d11347c1ebc42f
9b01f86da8766c34db2ce799d3a
--- END OF TEXT ---
```

The elipses (...) above indicate that some time has traversed after the key-sifting step has concluded. The header – `START OF TEXT` – indicates that Alice has begun transmitting encrypted messages over the classical channel using the final key constructed using the raw key (intercepted in the next section), and masking it with the incoming messages from Alice (matched basis string).

Note that, since the 32-bit key is constructed in multiple steps, involving multiple transmissions from Alice on what the matched basis string should be, the resulting matched basis string (hex) from all of these steps in this example is therefore: `fcef8c4c5c386ae9`.

5.2.2 Eavesdropping the quantum channel for the raw key string

Once the IR receivers has been set to listen in on the classical channel, Eve now eavesdrops on the quantum channel using `key_logger.py`, which records the signals registered by her two photodiodes. (In the following code examples, we assume Eve connects her Arduino to COM5 of her computer.)

```
programs\4_HackTools\Quantum_Listener\key_logger>
python key_logger.py --serial COM5
Eavesdropping... will record any voltages into a file
To exit the program, use Ctrl+C

Ready!

182456.dat
Logging the voltages into: 182456.dat
0.0020880699157714844 242 217 ←- these go to log file!!
0.06286954879760742 243 219
```

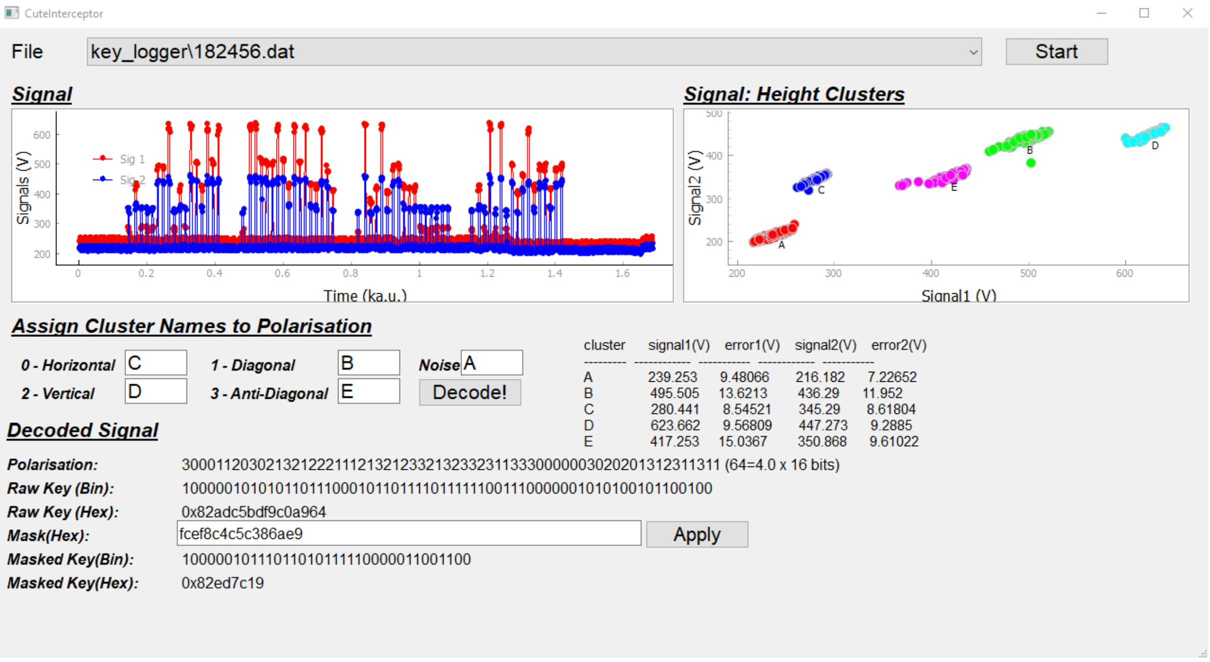
In Section 5.2.4, Eve uses `runInterceptor.py` to identify cluster of voltages that correspond to the four polarization states. This allows her to associate each pair of photodiode signals to a possible polarization state. The correct assignment allows her to translate the sequence of photodiode signals to a corresponding sequence of polarization states, which can be used to derive the correct final key. To determine if the correct key is obtained, we test the keys by decrypting secret messages collected in Section 5.2.3, and see if they result in intelligible messages.

5.2.3 Eavesdropping the classical channel during encrypted message transmission

Once Alice and Bob finishes distributing their key. They will use it to encrypt and transmit secret messages. If Eve is monitoring the transmission over the public channel using `listener.py` (Section 5.2.1), she will detect these messages. To facilitate the identification of the start and end of these messages in our educational kit, we have left the header `-START OF TEXT-` and footer `-END OF TEXT-` of the classical channel unencrypted.

```
--- START OF TEXT ---
c31f015f5c9104af7adb244359fc382c93899d11347c1ebc42f
9b01f86da8766c34db2ce799d3a
--- END OF TEXT ---
Thank you for using the program!
```

5.2.4 Guessing the final key and decrypting encrypted messages



File: key_logger\182456.dat

0 - Horizontal: C
2 - Vertical: D
1 - Diagonal: B
3 - Anti-Diagonal: E
Noise: A

```
cluster   signal1(V)   error1(V)   signal2(V)   error2(V)
--------  ----------   ---------   ----------   ---------
A          239.253      9.48066     216.182      7.22652
B          495.505      13.6213     436.29       11.952
C          280.441      8.54521     345.29       8.61804
D          623.662      9.56809     447.273      9.2885
E          417.253      15.0367     350.868      9.61022
```

Polarisation: 3000112030213212222111213212332132332311333000000302020131231131 1 (64=4.0 x 16 bits)
Raw Key (Bin): 1000001010101101110001011011110111111001110000001010100101100100
Raw Key (Hex): 0x82adc5bdf9c0a964
Mask(Hex): fcef8c4c5c386ae9
Masked Key(Bin): 10000010111011010111110000011001100
Masked Key(Hex): 0x82ed7c19

Figure 5.4: Guessing the sifted key by guessing the polarization states that correspond to the voltages, Signal1 and Signal2, recorded by Eve's photodiode pair. Left: Signal graph that plots the two photodiode signals over time. Right: XY-plot of Signal1 and Signal2 depicting five voltage clusters. The program assigns an arbitrary label A to E, which are assigned the four polarization states and to noise – the assignment is by trial-and-error. Use of the GUI is described in the main text.

To process the photodiode voltage sequences recorded in Section 5.2.2, we have provided a graphical user interface (GUI) (Fig. 5.4). To start the GUI, run `runInterceptor.py`.

The GUI automatically identifies clusters of photodiode signals, and allows you to assign a guess to the polarization state that each cluster represents[1]. This guess is then used to calculate the resulting sifted key using the intercepted matched basis bit string (Section 4.2.1) that is publicly available in the classical channel.

[1] The KMeans algorithm is implemented using Python's Sklearn library [20].

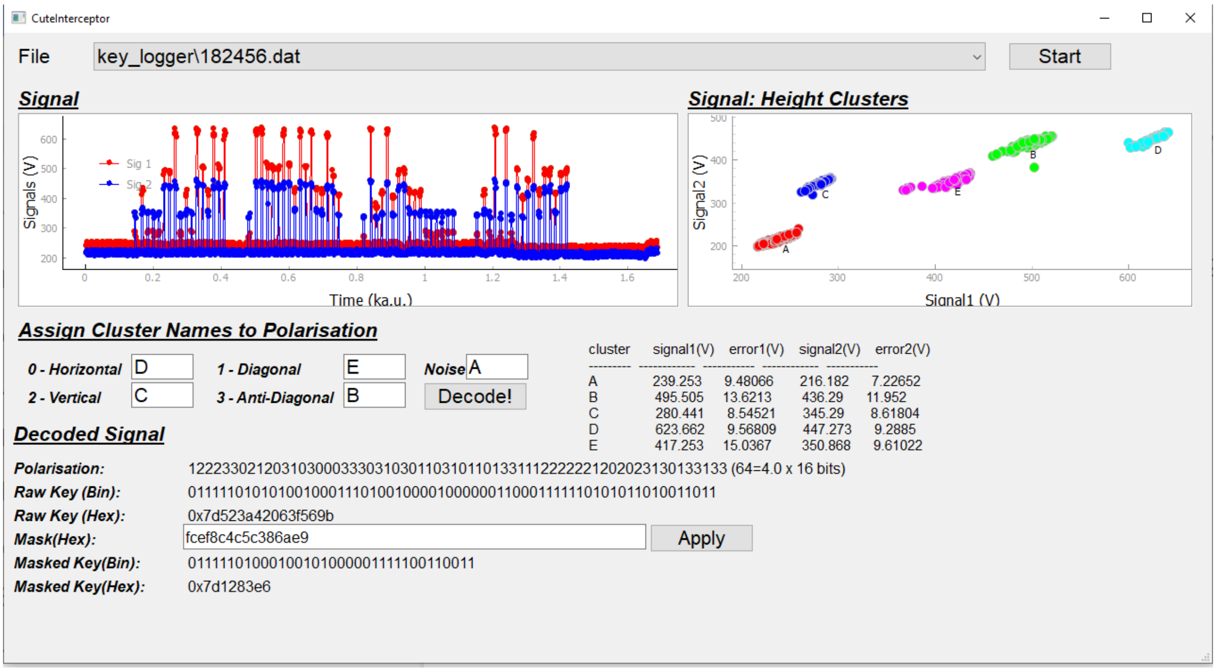
The use of the GUI is described in the following steps:

1. First, select the filename that records the photodiode signals performed in Section 5.2.2.
2. Click “Start”. This plots the signals from the two photodiodes, and organizes the datapoints to five clusters arbitrarily labelled A to E. Note that Fig. 5.4 shows five, instead of four voltage clusters, since in a practical implementation, one of the voltage clusters correspond to electrical noise.
3. Assign cluster names to polarization: make an initial guess to what the group labels A to E might mean: assign them to the four possible polarization states, and to noise.
4. Click “Decode!”, which labels the signal sequences with the corresponding polarization identities. The resulting “polarization” string is displayed in numerical format, where the polarizations H, V, D, A are represented by 0, 2, 1, 3, respectively. As each polarization state represents a binary bit, this polarization string can be translated into an assiociated raw key string.
5. The intercepted matched basis string sequence obtained in Section 5.2.1 is input under the field “Mask(Hex)” in hexadecimal form.
6. Clicking “Apply” applies the matched basis string sequence to the raw key, producing the final “Masked Key”.

Next, execute `decrypt.py` to use the “Masked key” to decipher secret messages obtained in Section 5.2.1:

```
programs\4_HackTools>python decrypt.py

Welcome to decrypt! Please enter the encrypted text:
c31f015f5c9104af7adb244359fc382c93899d11347c1ebc42f
9b01f86da8766c34db2ce799d3a

Now, please enter the key (in 32 bit hex):
82ed7c19

The decrypted message is:
When does the Sun rise on the East Coat

Congratulations! Thank you for using the program!
```

Note that if the sifted key happens to be incorrect, due to incorrect assignment of polarization states to the clusters (see example Fig. 5.5), we would obtain an illegible message (see example Fig. 5.6). We then continue to guess the other possible permutations for this assignment using the GUI, and repeat the process with `decrypt.py` until we obtain an intelligible message.



Assign Cluster Names to Polarisation

0 - Horizontal: D
1 - Diagonal: E
Noise: A
2 - Vertical: C
3 - Anti-Diagonal: B

```
cluster    signal1(V)   error1(V)   signal2(V)   error2(V)
---------  ------------ ----------- ------------ -----------
A           239.253      9.48066     216.182      7.22652
B           495.505      13.6213     436.29       11.952
C           280.441      8.54521     345.29       8.61804
D           623.662      9.56809     447.273      9.2885
E           417.253      15.0367     350.868      9.61022
```

Decoded Signal

Polarisation: 1222330212031030003330310301103101101331112222221202023130133133 (64=4.0 x 16 bits)
Raw Key (Bin): 0111110101010010001110100100001000000110001111110101011010011011
Raw Key (Hex): 0x7d523a42063f569b
Mask(Hex): fcef8c4c5c386ae9
Masked Key(Bin): 01111101000100101000001111100110011
Masked Key(Hex): 0x7d1283e6

Figure 5.5: An incorrect guess of the assignment of polarization statees corresponding to the photodiode voltages, leading to an incorrect guess of the final key, and subsequently, an illegible decrypted message in Fig. 5.6.

```
programs\4_HackTools>python decrypt.py

Welcome to decrypt! Please enter the encrypted text:
c31f015f5c9104af7adb244359fc382c93899d11347c1ebc42f
9b01f86da8766c34db2ce799d3a

Now, please enter the key (in 32 bit hex):
7d1283e6

The decrypted message is:


Congratulations! Thank you for using the program!
```

Figure 5.6: An illegible decrypted message due to incorrectly guessing the final key.

5.3 Discussion

1. In our setup, Eve uses two photodetectors to reconstruct the polarization of Alice's photons. Is it possible to just use one photodetector? Would using a third photodetector help?

2. The number of photons in a single laser pulse is enormous compared with a single photon required for QKD. How many photons are there in a single laser pulse?

3. While the quantum bit transmission is one-directional, the classical channel supporting the protocol requires a two-way communication. Is it necessary for an eavesdropper to listen to both Alice and Bob in the classical channel?

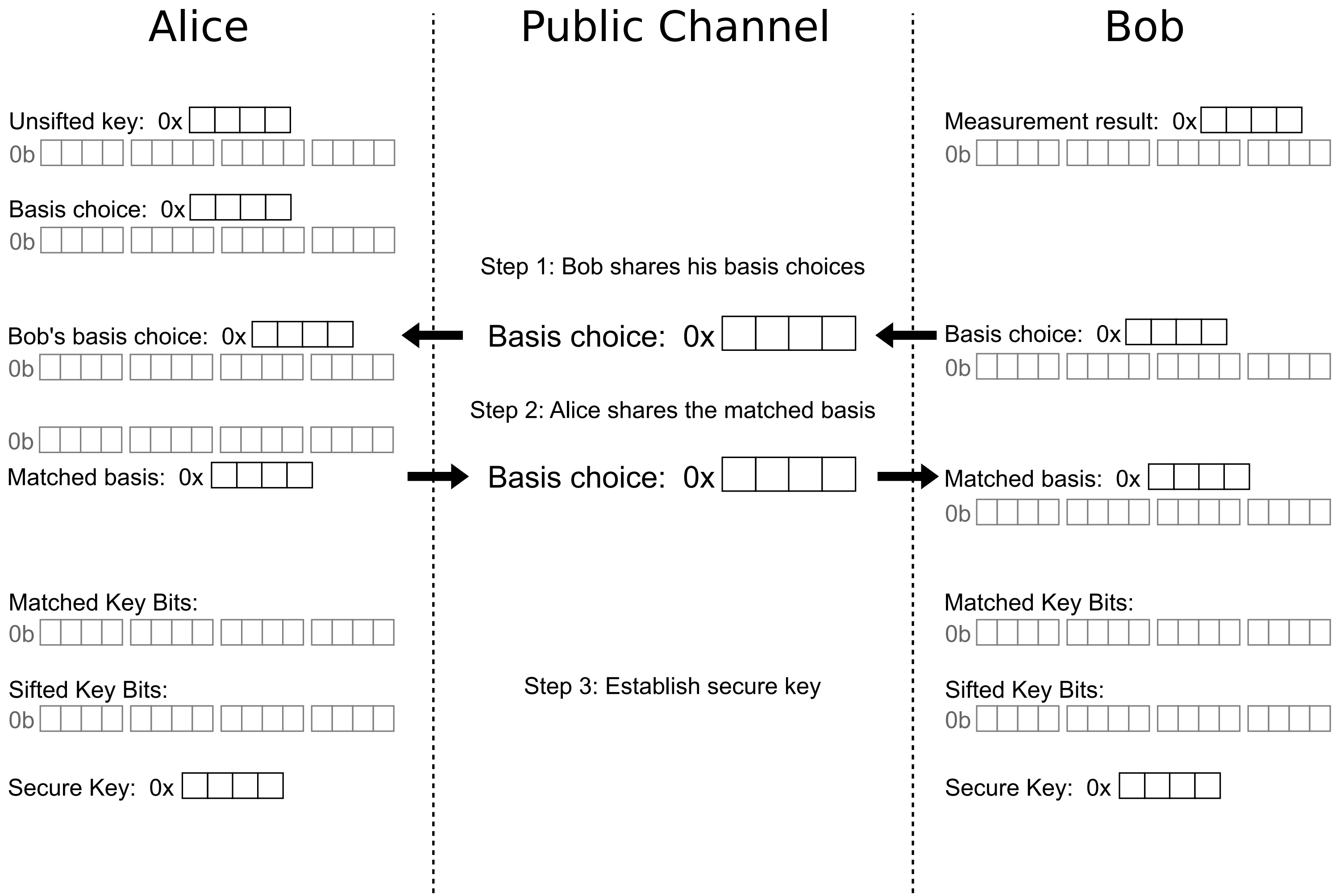
4. Without prior calibration, Eve would have to guess the polarization of each cluster, resulting in a few key combinations. How many plausible combinations are there?

5. Discuss why the QKD setup was vulnerable to hacking.

5.4 Further exploration

1. **** As the QKD key used to seed the longer expanded key is relatively short, the space of possible keys remains small (see Section 4.4). Consequently, the encrypted message is prone to a brute-force attack. Repeat the QKD protocol, eavesdrop again on the classical channel. This time, attempt to decrypt the secret message using a brute-force attack.

Key Sifting Worksheet



Alice

Unsifted key: 0x□□□□
0b□□□□ □□□□ □□□□ □□□□

Basis choice: 0x□□□□
0b□□□□ □□□□ □□□□ □□□□

Bob's basis choice: 0x□□□□
0b□□□□ □□□□ □□□□ □□□□

0b□□□□ □□□□ □□□□ □□□□
Matched basis: 0x□□□□

Matched Key Bits:
0b□□□□ □□□□ □□□□ □□□□

Sifted Key Bits:
0b□□□□ □□□□ □□□□ □□□□

Secure Key: 0x□□□□

Public Channel

Step 1: Bob shares his basis choices

← Basis choice: 0x□□□□ ←

Step 2: Alice shares the matched basis

→ Basis choice: 0x□□□□ →

Step 3: Establish secure key

Bob

Measurement result: 0x□□□□
0b□□□□ □□□□ □□□□ □□□□

Basis choice: 0x□□□□
0b□□□□ □□□□ □□□□ □□□□

Matched basis: 0x□□□□
0b□□□□ □□□□ □□□□ □□□□

Matched Key Bits:
0b□□□□ □□□□ □□□□ □□□□

Sifted Key Bits:
0b□□□□ □□□□ □□□□ □□□□

Secure Key: 0x□□□□

Key Sifting Worksheet (Sample Answer)

Alice

Unsifted key: 0x E199

0b 1110 0001 1001 1001

Basis choice: 0x 279D

0b 0010 0111 1001 1101

Bob's basis choice: 0x E269

0b 1110 0010 0110 1001

0b 0011 1010 0000 1011

Matched basis: 0x 3A0B

Matched Key Bits:

0b XX10 0X0X XXXX 1X01

Sifted Key Bits:

0b XXXX XXXX X100 0101

Secure Key: 0x XX45

$\underbrace{}_{\text{7 bits}}$

Public Channel

Step 1: Bob shares his basis choices

← Basis choice: 0x E269 ←

Step 2: Alice shares the matched basis

→ Basis choice: 0x 3A0B →

Step 3: Establish secure key

Bob

Measurement result: 0x 644D

0b 0110 0100 0100 1101

Basis choice: 0x E269

0b 1110 0010 0110 1001

Matched basis: 0x 3A0B

0b 0011 1010 0000 1011

Matched Key Bits:

0b XX10 0X0X XXXX 1X01

Sifted Key Bits:

0b XXXX XXXX X100 0101

Secure Key: 0x XX45

$\underbrace{}_{\text{7 bits}}$

Bibliography

[1] Valerio Scarani, Chua Lynn, and Shiyang Liu. *Six quantum pieces: A first course in quantum physics*. World Scientific, 2010 (page 1).

[2] Berthold-Georg Englert. *Lectures on Quantum Mechanics*. World Scientific Publishing Company, 2006. DOI: 10.1142/6093-vol1. eprint: https://www.worldscientific.com/doi/pdf/10.1142/6093-vol1. URL: https://www.worldscientific.com/doi/abs/10.1142/6093-vol1 (pages 1, 9).

[3] Charles H Bennett and Gilles Brassard. *Proceedings of the IEEE International Conference on Computers, Systems and Signal Processing*. 1984 (pages 1, 7, 9).

[4] *Industry Specification Group (ISG) on Quantum Key Distribution for Users (QKD)*. European Telecommunication Standard Institute. URL: https://www.etsi.org/committee/qkd (page 1).

[5] *Quantum Key Distribution (QKD); QKD Module Security Specification*. Vol. 2. European Telecommunication Standard Institute. 2019 (page 1).

[6] *Quantum Key Distribution (QKD); Device and Communication Channel Parameters for QKD Deployment*. Vol. 2. European Telecommunication Standard Institute. 2019. URL: https://www.etsi.org/deliver/etsi_gs/QKD/001_099/012/01.01.01_60/gs_QKD012v010101p.pdf (page 1).

[7] Valerio Scarani and Christian Kurtsiefer. “The black paper of quantum cryptography: real implementation problems”. In: *Theoretical Computer Science* 560 (2014), pp. 27–32 (page 1).

[8] Adrian Nugraha Utama, Jianwei Lee, and Mathias Alexander Seidler. “A hands-on quantum cryptography workshop for pre-university students”. In: *American Journal of Physics* 88.12 (2020), pp. 1094–1102 (page 1).

[9] National Environment Agency. *Frequently Asked Questions on Lasers*. https://www.nea.gov.sg/our-services/radiation-safety/lasers/frequently-asked-questions-on-lasers. [Online; accessed 5-Mar-2022]. 2022 (page 2).

[10] Hoi-Kwong Lo, Xiongfeng Ma, and Kai Chen. “Decoy State Quantum Key Distribution”. In: *Phys. Rev. Lett.* 94 (23 June 2005), p. 230504. DOI: 10.1103/PhysRevLett.94.230504. URL: https://link.aps.org/doi/10.1103/PhysRevLett.94.230504 (page 6).

[11] Tomohiro Sugimoto and Kouichi Yamazaki. “A study on secret key reconciliation protocol "cascade"”. In: *IEICE Trans. Fundamentals* E83-A.10 (Oct. 2000), pp. 1987–1991 (page 8).

[12] Gilles Brassard and L Salvail. “Advances in cryptology eurocrypt’93”. In: *Lecture Notes in Computer Science* 765 (1994), pp. 410–423 (page 8).

[13] Peter W. Shor and John Preskill. “Simple Proof of Security of the BB84 Quantum Key Distribution Protocol”. In: *Phys. Rev. Lett.* 85 (2 July 2000), pp. 441–444. DOI:

10.1103/PhysRevLett.85.441. URL: https://link.aps.org/doi/10.1103/PhysRevLett.85.441 (page 8).

[14] Stephen Wiesner. “Conjugate coding”. In: *ACM Sigact News* 15.1 (1983), pp. 78–88 (page 9).

[15] Alexander Ling, Kee Pang Soh, AntÍa Lamas-Linares, and Christian Kurtsiefer. “An optimal photon counting polarimeter”. In: *Journal of Modern Optics* 53.10 (2006), pp. 1523–1528 (page 10).

[16] GE Jellison. “Four-channel polarimeter for time-resolved ellipsometry”. In: *Optics letters* 12.10 (1987), pp. 766–768 (page 10).

[17] San Bergmans. *NEC Protocol*. https://www.sbprojects.net/knowledge/ir/nec.php. [Online; accessed 19-Feb-2022]. 2021 (pages 17, 18).

[18] Arduino.cc. *Introduction to the Arduino Board*. https://www.arduino.cc/en/reference/board. [Online; accessed 10-Mar-2022]. 2022 (page 19).

[19] Richard O Duda, Peter E Hart, et al. *Pattern classification*. John Wiley & Sons, 2006 (page 51).

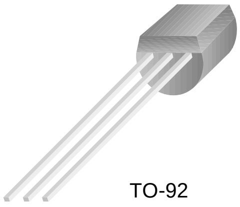
[20] scikit-learn developers. *sklearn.cluster.KMeans*. https://scikit-learn.org/stable/modules/generated/sklearn.cluster.KMeans.html. [Online; accessed 19-Mar-2022]. 2022 (page 55).

FAIRCHILD
SEMICONDUCTOR®

BC546/547/548/549/550

Switching and Applications

- High Voltage: BC546, V_{CEO}=65V
- Low Noise: BC549, BC550
- Complement to BC556 ... BC560



TO-92

1. Collector 2. Base 3. Emitter

NPN Epitaxial Silicon Transistor

Absolute Maximum Ratings T_a=25°C unless otherwise noted

Symbol	Parameter	Value	Units
V_{CBO}	Collector-Base Voltage : BC546	80	V
	: BC547/550	50	V
	: BC548/549	30	V
V_{CEO}	Collector-Emitter Voltage : BC546	65	V
	: BC547/550	45	V
	: BC548/549	30	V
V_{EBO}	Emitter-Base Voltage : BC546/547	6	V
	: BC548/549/550	5	V
I_C	Collector Current (DC)	100	mA
P_C	Collector Power Dissipation	500	mW
T_J	Junction Temperature	150	°C
T_{STG}	Storage Temperature	-65 ~ 150	°C

Electrical Characteristics T_a=25°C unless otherwise noted

Symbol	Parameter	Test Condition	Min.	Typ.	Max.	Units
I_{CBO}	Collector Cut-off Current	V_{CB}=30V, I_E=0			15	nA
h_{FE}	DC Current Gain	V_{CE}=5V, I_C=2mA	110		800	
V_{CE} (sat)	Collector-Emitter Saturation Voltage	I_C=10mA, I_B=0.5mA		90	250	mV
		I_C=100mA, I_B=5mA		200	600	mV
V_{BE} (sat)	Base-Emitter Saturation Voltage	I_C=10mA, I_B=0.5mA		700		mV
		I_C=100mA, I_B=5mA		900		mV
V_{BE} (on)	Base-Emitter On Voltage	V_{CE}=5V, I_C=2mA	580	660	700	mV
		V_{CE}=5V, I_C=10mA			720	mV
f_T	Current Gain Bandwidth Product	V_{CE}=5V, I_C=10mA, f=100MHz		300		MHz
C_{ob}	Output Capacitance	V_{CB}=10V, I_E=0, f=1MHz		3.5	6	pF
C_{ib}	Input Capacitance	V_{EB}=0.5V, I_C=0, f=1MHz		9		pF
NF	Noise Figure : BC546/547/548	V_{CE}=5V, I_C=200μA		2	10	dB
	: BC549/550	f=1KHz, R_G=2KΩ		1.2	4	dB
	: BC549	V_{CE}=5V, I_C=200μA		1.4	4	dB
	: BC550	R_G=2KΩ, f=30~15000MHz		1.4	3	dB

h_{FE} Classification

Classification	A	B	C
h_{FE}	110 ~ 220	200 ~ 450	420 ~ 800

Typical Characteristics

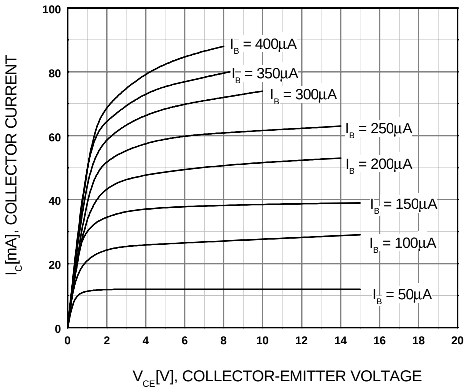


Figure 1. Static Characteristic

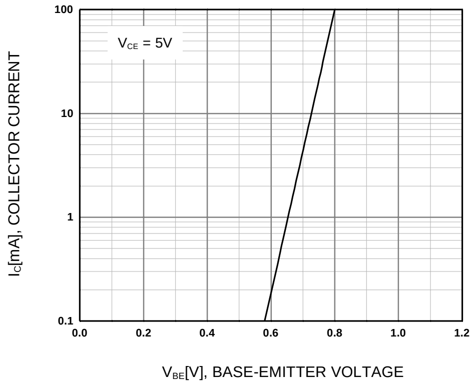


Figure 2. Transfer Characteristic

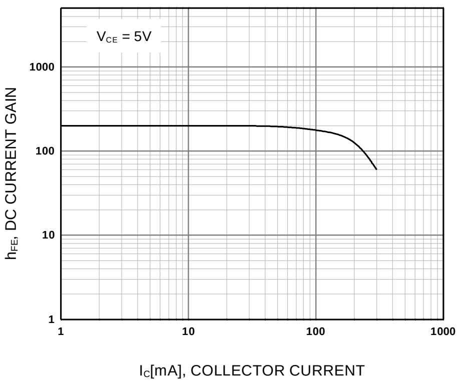


Figure 3. DC current Gain

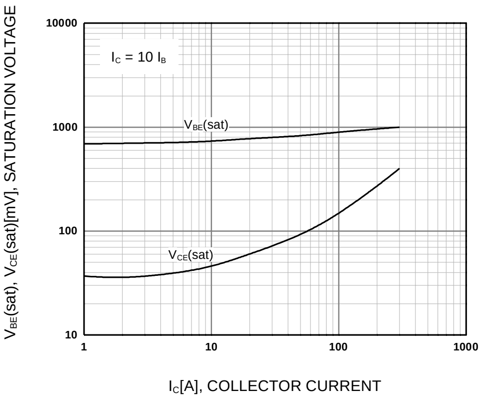


Figure 4. Base-Emitter Saturation Voltage
Collector-Emitter Saturation Voltage

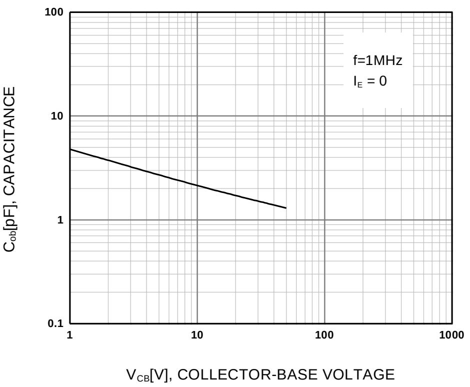


Figure 5. Output Capacitance

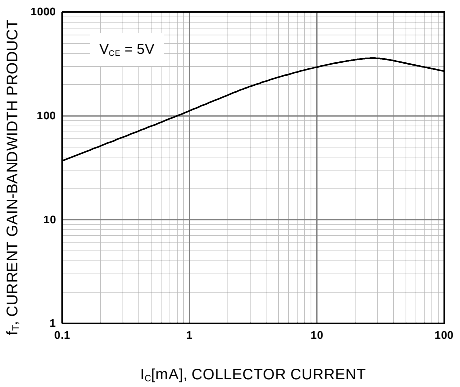
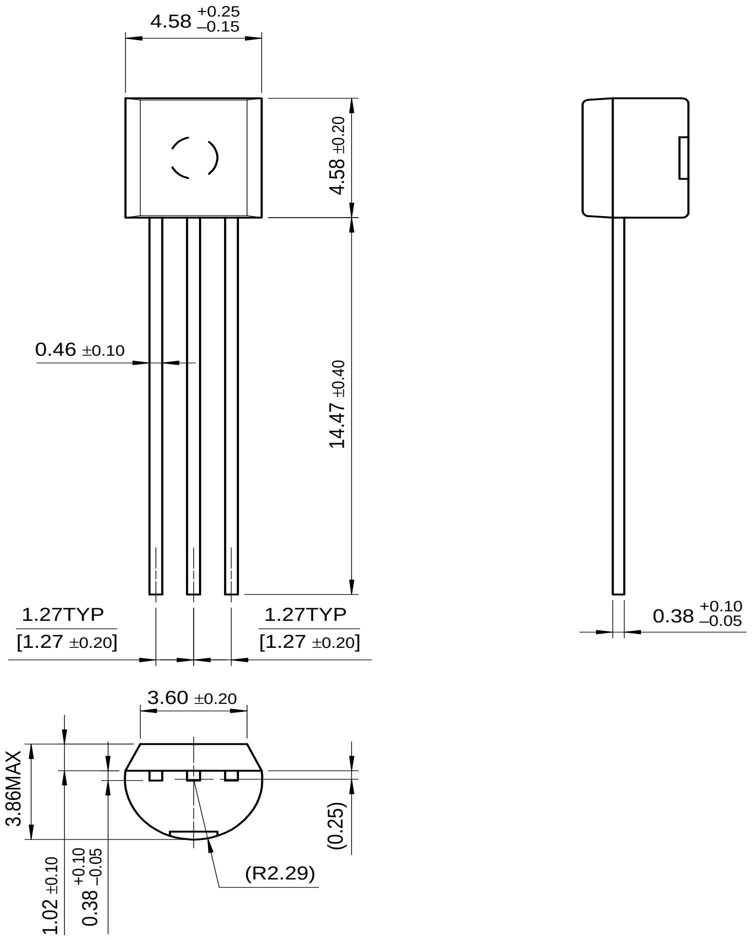


Figure 6. Current Gain Bandwidth Product

Package Dimensions

TO-92



Dimensions in Millimeters

TRADEMARKS

The following are registered and unregistered trademarks Fairchild Semiconductor owns or is authorized to use and is not intended to be an exhaustive list of all such trademarks.

ACEx™
ActiveArray™
Bottomless™
CoolFET™
CROSSVOLT™
DOME™
EcoSPARK™
E^{2}CMOS™
EnSigna™

FACT™
FACT Quiet series™
FAST®
FASTr™
FRFET™
GlobalOptoisolator™
GTO™
HiSeC™
I^{2}C™

Across the board. Around the world.™
The Power Franchise™
Programmable Active Droop™

ImpliedDisconnect™
ISOPLANAR™
LittleFET™
MicroFET™
MicroPak™
MICROWIRE™
MSX™
MSXPro™
OCX™
OCXPro™
OPTOLOGIC®
OPTOPLANAR™

PACMAN™
POP™
Power247™
PowerTrench®
QFET™
QS™
QT Optoelectronics™
Quiet Series™
RapidConfigure™
RapidConnect™
SILENT SWITCHER®
SMART START™

SPM™
Stealth™
SuperSOT™-3
SuperSOT™-6
SuperSOT™-8
SyncFET™
TinyLogic™
TruTranslation™
UHC™
UltraFET®
VCX™

DISCLAIMER

FAIRCHILD SEMICONDUCTOR RESERVES THE RIGHT TO MAKE CHANGES WITHOUT FURTHER NOTICE TO ANY PRODUCTS HEREIN TO IMPROVE RELIABILITY, FUNCTION OR DESIGN. FAIRCHILD DOES NOT ASSUME ANY LIABILITY ARISING OUT OF THE APPLICATION OR USE OF ANY PRODUCT OR CIRCUIT DESCRIBED HEREIN; NEITHER DOES IT CONVEY ANY LICENSE UNDER ITS PATENT RIGHTS, NOR THE RIGHTS OF OTHERS.

LIFE SUPPORT POLICY

FAIRCHILD'S PRODUCTS ARE NOT AUTHORIZED FOR USE AS CRITICAL COMPONENTS IN LIFE SUPPORT DEVICES OR SYSTEMS WITHOUT THE EXPRESS WRITTEN APPROVAL OF FAIRCHILD SEMICONDUCTOR CORPORATION.
As used herein:

1. Life support devices or systems are devices or systems which, (a) are intended for surgical implant into the body, or (b) support or sustain life, or (c) whose failure to perform when properly used in accordance with instructions for use provided in the labeling, can be reasonably expected to result in significant injury to the user.

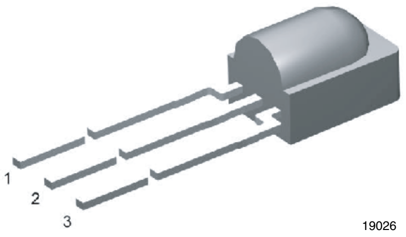
2. A critical component is any component of a life support device or system whose failure to perform can be reasonably expected to cause the failure of the life support device or system, or to affect its safety or effectiveness.

PRODUCT STATUS DEFINITIONS

Definition of Terms

Datasheet Identification	Product Status	Definition
Advance Information	Formative or In Design	This datasheet contains the design specifications for product development. Specifications may change in any manner without notice.
Preliminary	First Production	This datasheet contains preliminary data, and supplementary data will be published at a later date. Fairchild Semiconductor reserves the right to make changes at any time without notice in order to improve design.
No Identification Needed	Full Production	This datasheet contains final specifications. Fairchild Semiconductor reserves the right to make changes at any time without notice in order to improve design.
Obsolete	Not In Production	This datasheet contains specifications on a product that has been discontinued by Fairchild semiconductor. The datasheet is printed for reference information only.

IR Receiver Modules for Remote Control Systems



FEATURES

- Very low supply current
- Photo detector and preamplifier in one package
- Internal filter for PCM frequency
- Supply voltage: 2.5 V to 5.5 V
- Improved immunity against ambient light
- Insensitive to supply voltage ripple and noise
- Material categorization: for definitions of compliance please see www.vishay.com/doc?99912

Pb-free
e3
RoHS
COMPLIANT
HALOGEN
FREE
GREEN
(5-2008)

MECHANICAL DATA

Pinning for TSOP382.., TSOP384..:

1 = OUT, 2 = GND, 3 = V_S

DESCRIPTION

These products are miniaturized IR receiver modules for infrared remote control systems. A PIN diode and a preamplifier are assembled on a leadframe, the epoxy package contains an IR filter.

The demodulated output signal can be directly connected to a microprocessor for decoding.

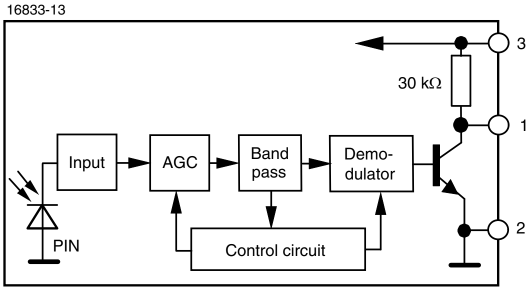
The TSOP384.. series devices are optimized to suppress almost all spurious pulses from energy saving lamps like CFLs. The AGC4 used in the TSOP384.. may suppress some data signals. The TSOP382.. series are provided primarily for compatibility with old AGC2 designs. New designs should prefer the TSOP384.. series containing the newer AGC4.

These components have not been qualified according to automotive specifications.

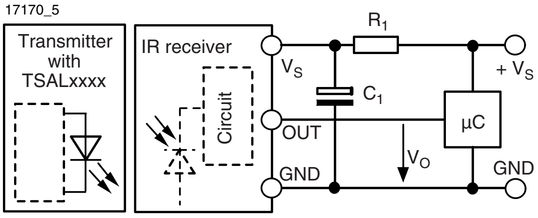
PARTS TABLE

AGC		LEGACY, FOR LONG BURST REMOTE CONTROLS (AGC2)	RECOMMENDED FOR LONG BURST CODES (AGC4)
Carrier frequency	30 kHz	TSOP38230	TSOP38430
	33 kHz	TSOP38233	TSOP38433
	36 kHz	TSOP38236	TSOP38436 (1)(2)(3)
	38 kHz	TSOP38238	TSOP38438 (4)(5)
	40 kHz	TSOP38240	TSOP38440
	56 kHz	TSOP38256	TSOP38456 (6)(7)
Package		Minicast	
Pinning		1 = OUT, 2 = GND, 3 = V_S	
Dimensions (mm)		5.0 W x 6.95 H x 4.8 D	
Mounting		Leaded	
Application		Remote control	
Best remote control code		(1) RC-5 (2) RC-6 (3) Panasonic (4) NEC (5) Sharp (6) r-step (7) Thomson RCA	

BLOCK DIAGRAM



APPLICATION CIRCUIT



R_1 and C_1 are recommended for protection against EOS. Components should be in the range of 33 Ω < R_1 < 1 kΩ, C_1 > 0.1 µF.

ABSOLUTE MAXIMUM RATINGS

PARAMETER	TEST CONDITION	SYMBOL	VALUE	UNIT
Supply voltage		V_S	-0.3 to +6	V
Supply current		I_S	3	mA
Output voltage		V_O	-0.3 to (V_S + 0.3)	V
Output current		I_O	5	mA
Junction temperature		T_j	100	°C
Storage temperature range		T_{stg}	-25 to +85	°C
Operating temperature range		T_{amb}	-25 to +85	°C
Power consumption	$T_{amb} \leq 85$ °C	P_{tot}	10	mW
Soldering temperature	t ≤ 10 s, 1 mm from case	T_{sd}	260	°C

Note

- Stresses beyond those listed under “Absolute Maximum Ratings” may cause permanent damage to the device. This is a stress rating only and functional operation of the device at these or any other conditions beyond those indicated in the operational sections of this specification is not implied. Exposure to absolute maximum rating conditions for extended periods may affect the device reliability.

ELECTRICAL AND OPTICAL CHARACTERISTICS (T_{amb} = 25 °C, unless otherwise specified)

PARAMETER	TEST CONDITION	SYMBOL	MIN.	TYP.	MAX.	UNIT
Supply current	E_v = 0, V_S = 3.3 V	I_{SD}	0.27	0.35	0.45	mA
	E_v = 40 klx, sunlight	I_{SH}	-	0.45	-	mA
Supply voltage		V_S	2.5	-	5.5	V
Transmission distance	E_v = 0, test signal see fig. 1, IR diode TSAL6200, I_F = 200 mA	d	-	45	-	m
Output voltage low	I_{OSL} = 0.5 mA, E_e = 0.7 mW/m^2, test signal see fig. 1	V_{OSL}	-	-	100	mV
Minimum irradiance	Pulse width tolerance: $t_{pi} - 5/f_o < t_{po} < t_{pi} + 6/f_o$, test signal see fig. 1	$E_{e\ min.}$	-	0.12	0.25	mW/m^2
Maximum irradiance	$t_{pi} - 5/f_o < t_{po} < t_{pi} + 6/f_o$, test signal see fig. 1	$E_{e\ max.}$	30	-	-	W/m^2
Directivity	Angle of half transmission distance	$\varphi_{1/2}$	-	± 45	-	deg

TYPICAL CHARACTERISTICS (T_{amb} = 25 °C, unless otherwise specified)

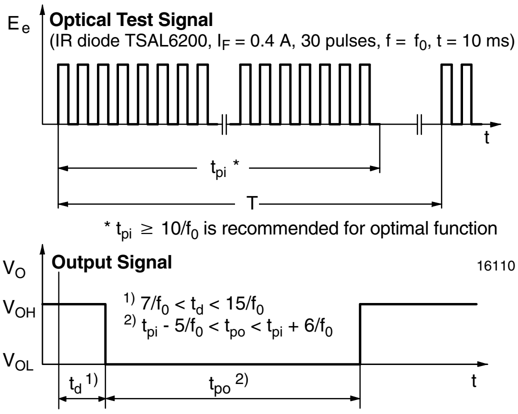

Ee
Optical Test Signal
(IR diode TSAL6200, IF = 0.4 A, 30 pulses, f = f0, t = 10 ms)
t
tpi *
T
* tpi ≥ 10/f0 is recommended for optimal function
VO
Output Signal
16110
VOH
1) 7/f0 < td < 15/f0
2) tpi - 5/f0 < tpo < tpi + 6/f0
VOL
td 1)
tpo 2)
t

Fig. 1 - Output Active Low

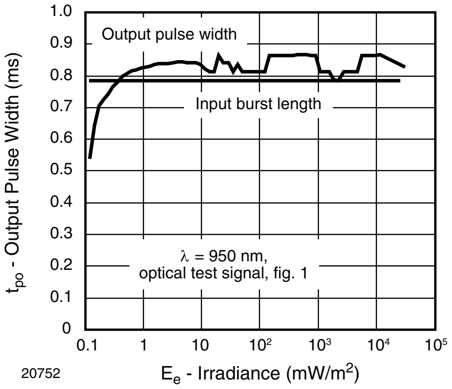


Fig. 2 - Pulse Length and Sensitivity in Dark Ambient

THIS DOCUMENT IS SUBJECT TO CHANGE WITHOUT NOTICE. THE PRODUCTS DESCRIBED HEREIN AND THIS DOCUMENT ARE SUBJECT TO SPECIFIC DISCLAIMERS, SET FORTH AT www.vishay.com/doc?91000

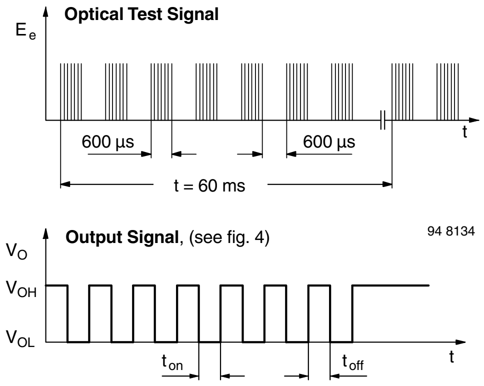


Fig. 3 - Output Function

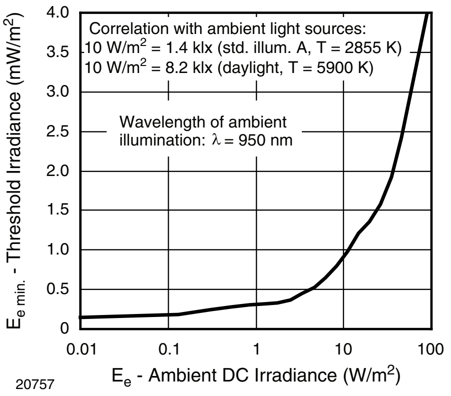


Fig. 6 - Sensitivity in Bright Ambient

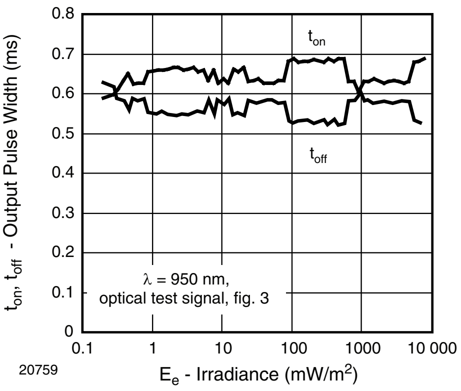


Fig. 4 - Output Pulse Diagram

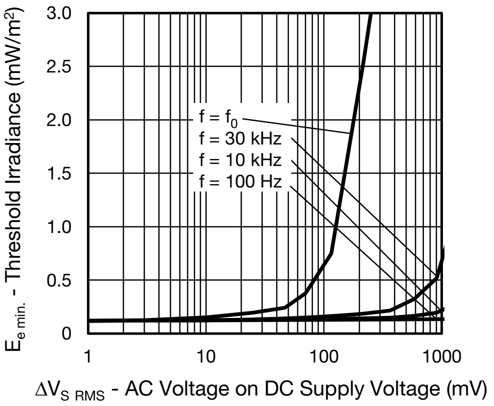


Fig. 7 - Sensitivity vs. Supply Voltage Disturbances

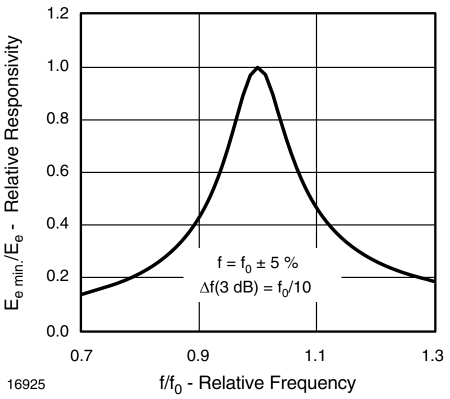


Fig. 5 - Frequency Dependence of Responsivity

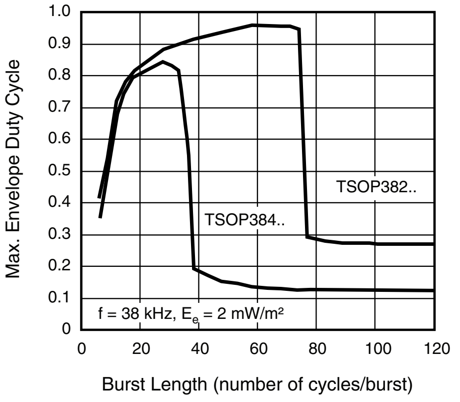


Fig. 8 - Max. Envelope Duty Cycle vs. Burst Length

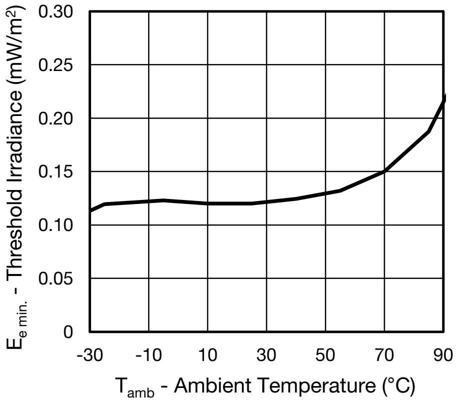


Fig. 9 - Sensitivity vs. Ambient Temperature

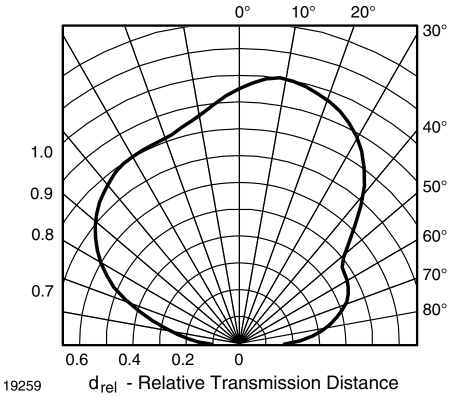


Fig. 12 - Vertical Directivity

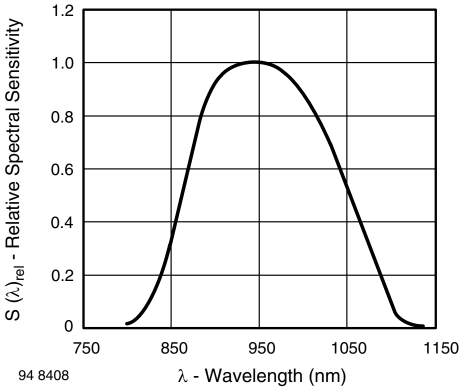


Fig. 10 - Relative Spectral Sensitivity vs. Wavelength

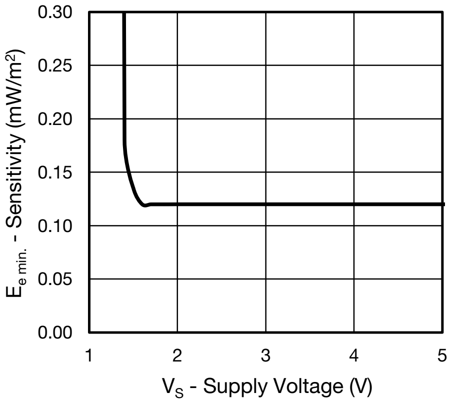


Fig. 13 - Sensitivity vs. Supply Voltage

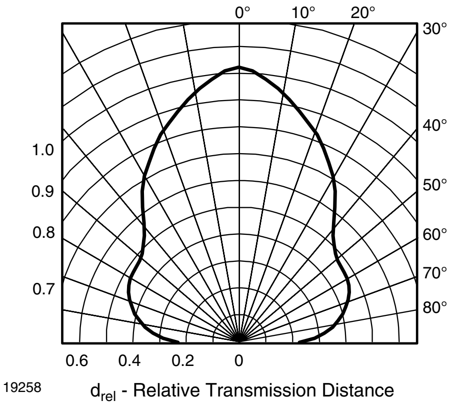


Fig. 11 - Horizontal Directivity

THIS DOCUMENT IS SUBJECT TO CHANGE WITHOUT NOTICE. THE PRODUCTS DESCRIBED HEREIN AND THIS DOCUMENT ARE SUBJECT TO SPECIFIC DISCLAIMERS, SET FORTH AT www.vishay.com/doc?91000

SUITABLE DATA FORMAT

This series is designed to suppress spurious output pulses due to noise or disturbance signals. The devices can distinguish data signals from noise due to differences in frequency, burst length, and envelope duty cycle. The data signal should be close to the device's band-pass center frequency (e.g. 38 kHz) and fulfill the conditions in the table below.

When a data signal is applied to the product in the presence of a disturbance, the sensitivity of the receiver is automatically reduced by the AGC to insure that no spurious pulses are present at the receiver's output.

Some examples which are suppressed are:

- DC light (e.g. from tungsten bulbs sunlight)
- Continuous signals at any frequency
- Strongly or weakly modulated patterns from fluorescent lamps with electronic ballasts (see fig. 14 or fig. 15).

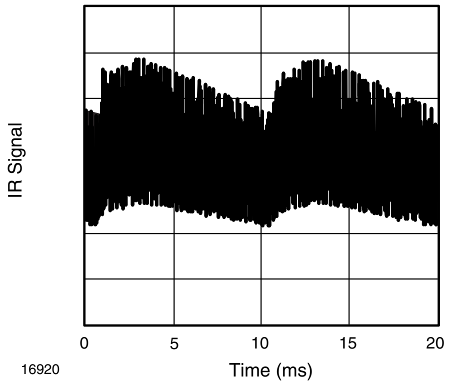


Fig. 14 - IR Disturbance from Fluorescent Lamp with Low Modulation

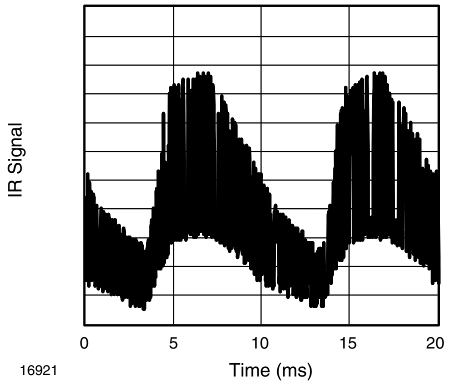


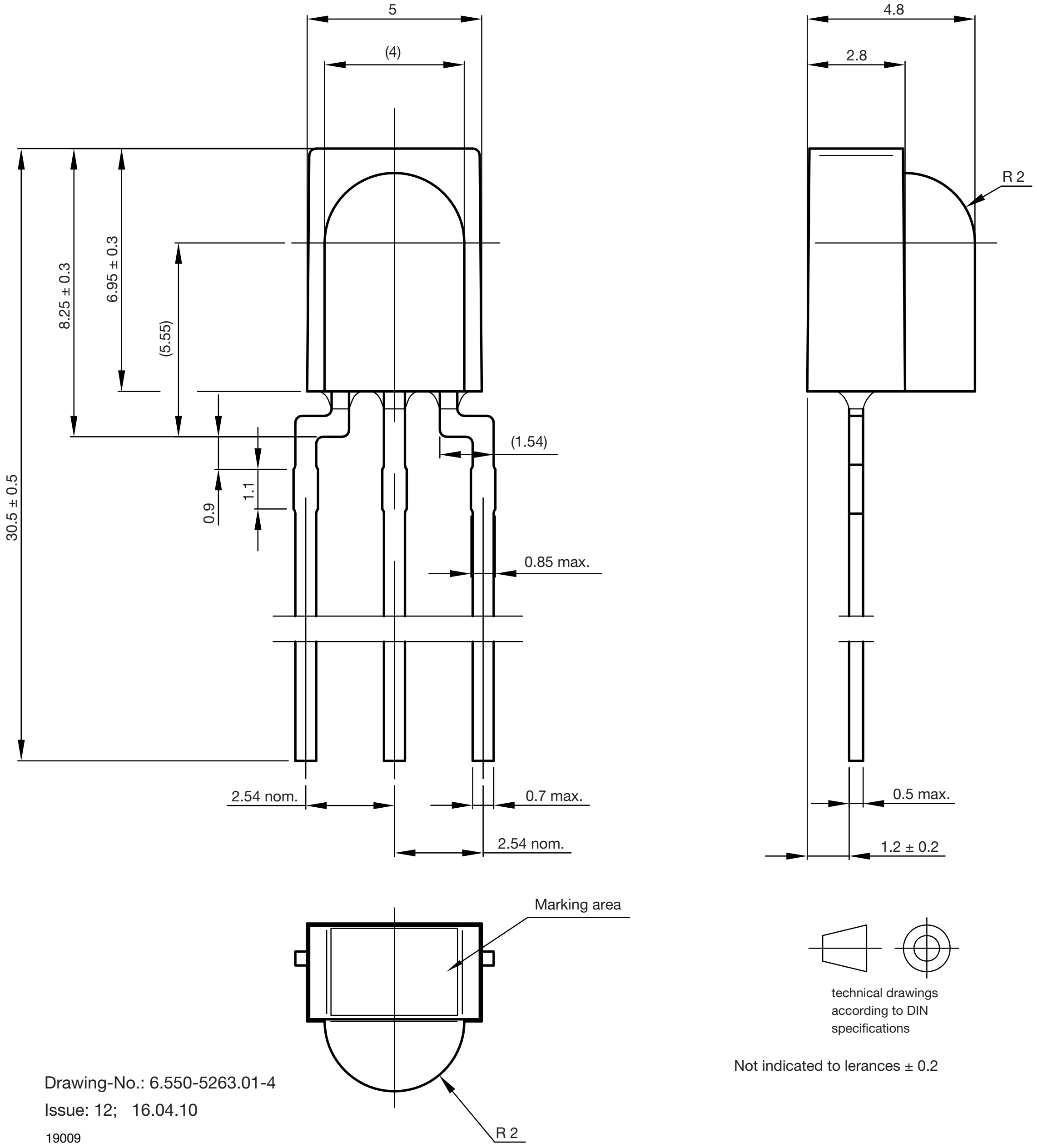
Fig. 15 - IR Disturbance from Fluorescent Lamp with High Modulation

	TSOP382..	TSOP384..
Minimum burst length	10 cycles/burst	10 cycles/burst
After each burst of length a minimum gap time is required of	10 to 70 cycles ≥ 10 cycles	10 to 35 cycles ≥ 10 cycles
For bursts greater than a minimum gap time in the data stream is needed of	70 cycles > 4 x burst length	35 cycles > 10 x burst length
Maximum number of continuous short bursts/second	1800	1500
NEC code	Yes	Preferred
RC5/RC6 code	Yes	Preferred
Thomson 56 kHz code	Yes	Preferred
Sharp code	Yes	Preferred
Suppression of interference from fluorescent lamps	Mild disturbance patterns are suppressed (example: signal pattern of fig. 14)	Complex and critical disturbance patterns are suppressed (example: signal pattern of fig. 15 or highly dimmed LCDs)

Notes

- For data formats with short bursts please see the datasheet for TSOP383.., TSOP385..
- For Sony 12, 15, and 20 bit IR codes please see the datasheet of TSOP38S40

PACKAGE DIMENSIONS in millimeters



Drawing-No.: 6.550-5263.01-4

Issue: 12; 16.04.10

19009

Not indicated to lerances ± 0.2

Disclaimer

ALL PRODUCT, PRODUCT SPECIFICATIONS AND DATA ARE SUBJECT TO CHANGE WITHOUT NOTICE TO IMPROVE RELIABILITY, FUNCTION OR DESIGN OR OTHERWISE.

Vishay Intertechnology, Inc., its affiliates, agents, and employees, and all persons acting on its or their behalf (collectively, “Vishay”), disclaim any and all liability for any errors, inaccuracies or incompleteness contained in any datasheet or in any other disclosure relating to any product.

Vishay makes no warranty, representation or guarantee regarding the suitability of the products for any particular purpose or the continuing production of any product. To the maximum extent permitted by applicable law, Vishay disclaims (i) any and all liability arising out of the application or use of any product, (ii) any and all liability, including without limitation special, consequential or incidental damages, and (iii) any and all implied warranties, including warranties of fitness for particular purpose, non-infringement and merchantability.

Statements regarding the suitability of products for certain types of applications are based on Vishay’s knowledge of typical requirements that are often placed on Vishay products in generic applications. Such statements are not binding statements about the suitability of products for a particular application. It is the customer’s responsibility to validate that a particular product with the properties described in the product specification is suitable for use in a particular application. Parameters provided in datasheets and/or specifications may vary in different applications and performance may vary over time. All operating parameters, including typical parameters, must be validated for each customer application by the customer’s technical experts. Product specifications do not expand or otherwise modify Vishay’s terms and conditions of purchase, including but not limited to the warranty expressed therein.

Except as expressly indicated in writing, Vishay products are not designed for use in medical, life-saving, or life-sustaining applications or for any other application in which the failure of the Vishay product could result in personal injury or death. Customers using or selling Vishay products not expressly indicated for use in such applications do so at their own risk. Please contact authorized Vishay personnel to obtain written terms and conditions regarding products designed for such applications.

No license, express or implied, by estoppel or otherwise, to any intellectual property rights is granted by this document or by any conduct of Vishay. Product names and markings noted herein may be trademarks of their respective owners.

Material Category Policy

Vishay Intertechnology, Inc. hereby certifies that all its products that are identified as RoHS-Compliant fulfill the definitions and restrictions defined under Directive 2011/65/EU of The European Parliament and of the Council of June 8, 2011 on the restriction of the use of certain hazardous substances in electrical and electronic equipment (EEE) - recast, unless otherwise specified as non-compliant.

Please note that some Vishay documentation may still make reference to RoHS Directive 2002/95/EC. We confirm that all the products identified as being compliant to Directive 2002/95/EC conform to Directive 2011/65/EU.

Vishay Intertechnology, Inc. hereby certifies that all its products that are identified as Halogen-Free follow Halogen-Free requirements as per JEDEC JS709A standards. Please note that some Vishay documentation may still make reference to the IEC 61249-2-21 definition. We confirm that all the products identified as being compliant to IEC 61249-2-21 conform to JEDEC JS709A standards.

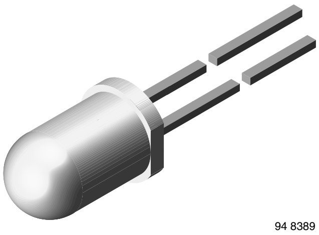
High Power Infrared Emitting Diode, 950 nm, GaAlAs/GaAs

Description

TSAL7400 is a high efficiency infrared emitting diode in GaAlAs on GaAs technology, molded in clear plastic packages.

In comparison with the standard GaAs on GaAs technology these emitters achieve more than 100 % radiant power improvement at a similar wavelength.

The forward voltages at low current and at high pulse current roughly correspond to the low values of the standard technology. Therefore these emitters are ideally suitable as high performance replacements of standard emitters.



Features

- Extra high radiant power and radiant intensity
- High reliability
- Low forward voltage
- Suitable for high pulse current operation
- Standard T-1¾ (∅ 5 mm) package
- Angle of half intensity $\varphi = \pm 25°$
- Peak wavelength $\lambda_p = 940$ nm
- Good spectral matching to Si photodetectors
- Lead (Pb)-free component
- Component in accordance to RoHS 2002/95/EC and WEEE 2002/96/EC

Pb
Pb-free

e2

Applications

- Infrared remote control units with high power requirements
- Free air transmission systems
- Infrared source for optical counters and card readers
- IR source for smoke detectors

Absolute Maximum Ratings

T_{amb} = 25 °C, unless otherwise specified

Parameter	Test condition	Symbol	Value	Unit
Reverse voltage		V_R	5	V
Forward current		I_F	100	mA
Peak forward current	$t_p/T = 0.5$, $t_p = 100$ µs	I_{FM}	200	mA
Surge forward current	$t_p = 100$ µs	I_{FSM}	1.5	A
Power dissipation		P_V	210	mW
Junction temperature		T_j	100	°C
Operating temperature range		T_{amb}	- 55 to + 100	°C
Storage temperature range		T_{stg}	- 55 to + 100	°C
Soldering temperature	$t \leq 5$ sec, 2 mm from case	T_{sd}	260	°C
Thermal resistance junction/ ambient		R_{thJA}	350	K/W

Electrical Characteristics

T_{amb} = 25 °C, unless otherwise specified

Parameter	Test condition	Symbol	Min	Typ.	Max	Unit
Forward voltage	I_F = 100 mA, t_p = 20 ms	V_F		1.35	1.6	V
	I_F = 1 A, t_p = 100 µs	V_F		2.6	3	V
Temp. coefficient of V_F	I_F = 100 mA	TK_{VF}		- 1.3		mV/K
Reverse current	V_R = 5 V	I_R			10	µA
Junction capacitance	V_R = 0 V, f = 1 MHz, E = 0	C_j		25		pF

Optical Characteristics

T_{amb} = 25 °C, unless otherwise specified

Parameter	Test condition	Symbol	Min	Typ.	Max	Unit
Radiant intensity	I_F = 100 mA, t_p = 20 ms	I_e	25	40	125	mW/sr
	I_F = 1.0 A, t_p = 100 µs	I_e	220	310		mW/sr
Radiant power	I_F = 100 mA, t_p = 20 ms	ϕ_e		35		mW
Temp. coefficient of ϕ_e	I_F = 20 mA	$TK\phi_e$		- 0.6		%/K
Angle of half intensity		φ		± 25		deg
Peak wavelength	I_F = 100 mA	λ_p		940		nm
Spectral bandwidth	I_F = 100 mA	$\Delta\lambda$		50		nm
Temp. coefficient of λ_p	I_F = 100 mA	$TK\lambda_p$		0.2		nm/K
Rise time	I_F = 100 mA	t_r		800		ns
Fall time	I_F = 100 mA	t_f		800		ns
Virtual source diameter	method: 63 % encircled energy	∅		2.2		mm

Typical Characteristics

T_{amb} = 25 °C, unless otherwise specified

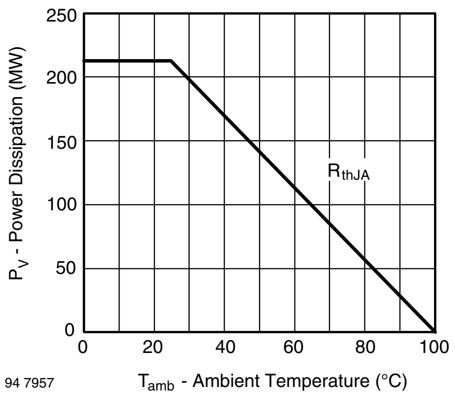


Figure 1. Power Dissipation vs. Ambient Temperature

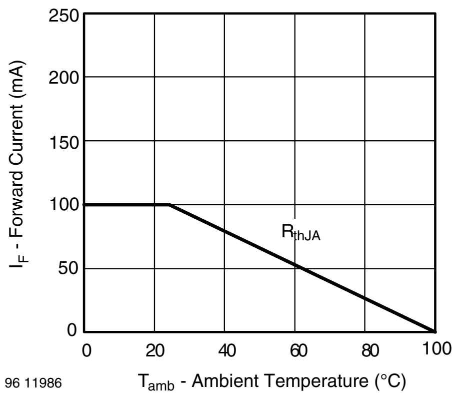


Figure 2. Forward Current vs. Ambient Temperature

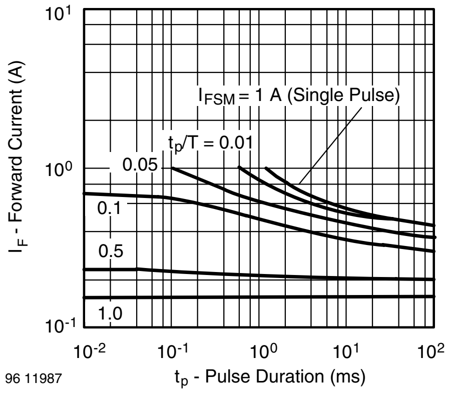


Figure 3. Pulse Forward Current vs. Pulse Duration

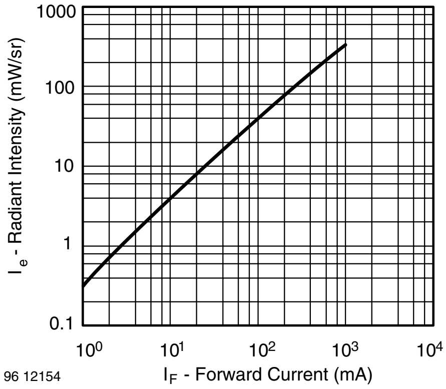


Figure 6. Radiant Intensity vs. Forward Current

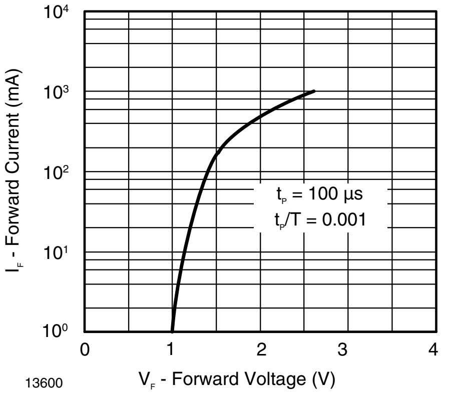


Figure 4. Forward Current vs. Forward Voltage

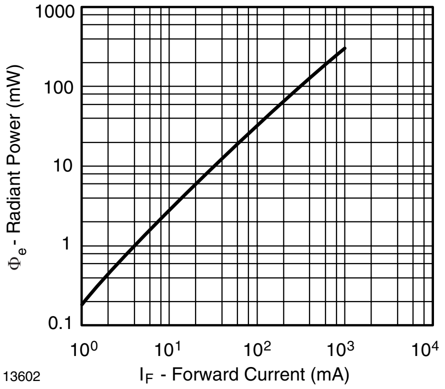


Figure 7. Radiant Power vs. Forward Current

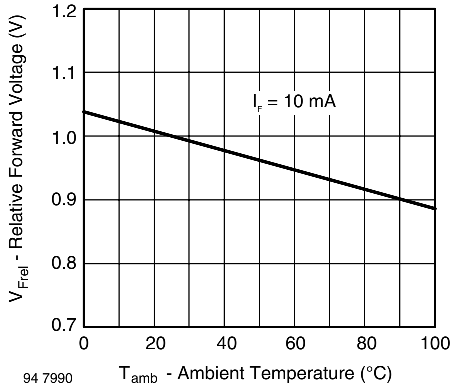


Figure 5. Relative Forward Voltage vs. Ambient Temperature

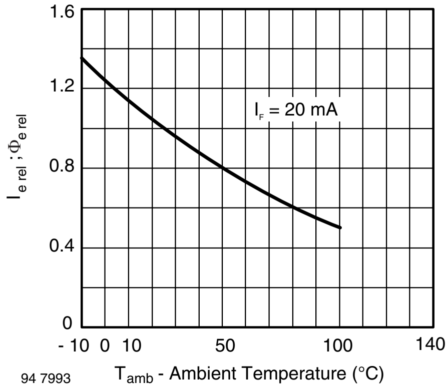


Figure 8. Rel. Radiant Intensity/Power vs. Ambient Temperature

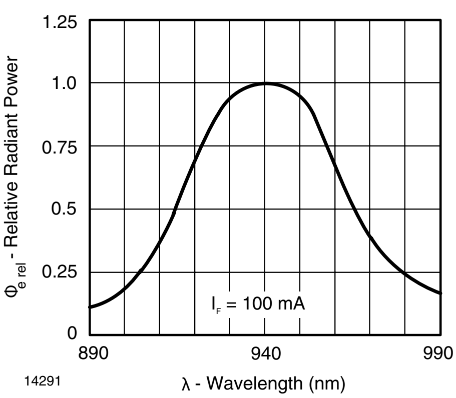


Figure 9. Relative Radiant Power vs. Wavelength

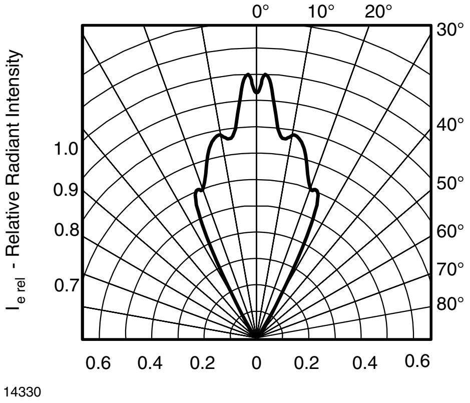
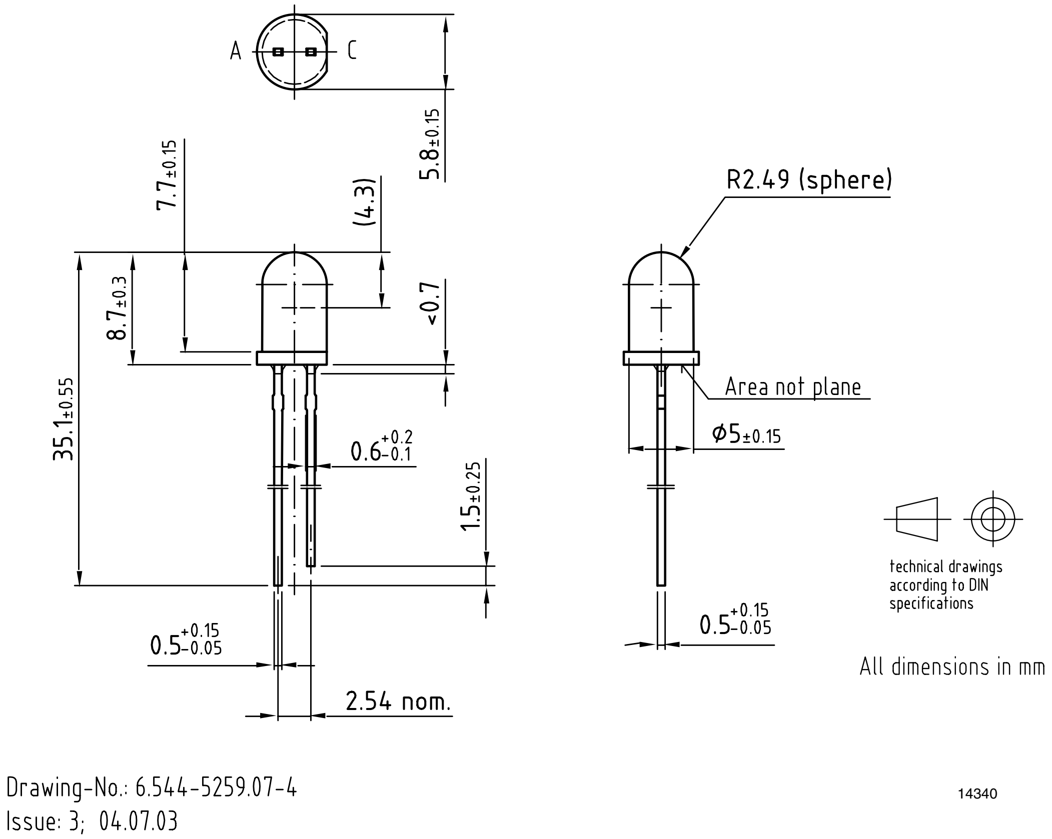


Figure 10. Relative Radiant Intensity vs. Angular Displacement

Package Dimensions in mm



Ozone Depleting Substances Policy Statement

It is the policy of Vishay Semiconductor GmbH to

1. Meet all present and future national and international statutory requirements.
2. Regularly and continuously improve the performance of our products, processes, distribution and operating systems with respect to their impact on the health and safety of our employees and the public, as well as their impact on the environment.

It is particular concern to control or eliminate releases of those substances into the atmosphere which are known as ozone depleting substances (ODSs).

The Montreal Protocol (1987) and its London Amendments (1990) intend to severely restrict the use of ODSs and forbid their use within the next ten years. Various national and international initiatives are pressing for an earlier ban on these substances.

Vishay Semiconductor GmbH has been able to use its policy of continuous improvements to eliminate the use of ODSs listed in the following documents.

1. Annex A, B and list of transitional substances of the Montreal Protocol and the London Amendments respectively
2. Class I and II ozone depleting substances in the Clean Air Act Amendments of 1990 by the Environmental Protection Agency (EPA) in the USA
3. Council Decision 88/540/EEC and 91/690/EEC Annex A, B and C (transitional substances) respectively.

Vishay Semiconductor GmbH can certify that our semiconductors are not manufactured with ozone depleting substances and do not contain such substances.

We reserve the right to make changes to improve technical design
and may do so without further notice.
Parameters can vary in different applications. All operating parameters must be validated for each customer application by the customer. Should the buyer use Vishay Semiconductors products for any unintended or unauthorized application, the buyer shall indemnify Vishay Semiconductors against all claims, costs, damages, and expenses, arising out of, directly or indirectly, any claim of personal damage, injury or death associated with such unintended or unauthorized use.

Vishay Semiconductor GmbH, P.O.B. 3535, D-74025 Heilbronn, Germany

Disclaimer

All product specifications and data are subject to change without notice.

Vishay Intertechnology, Inc., its affiliates, agents, and employees, and all persons acting on its or their behalf (collectively, “Vishay”), disclaim any and all liability for any errors, inaccuracies or incompleteness contained herein or in any other disclosure relating to any product.

Vishay disclaims any and all liability arising out of the use or application of any product described herein or of any information provided herein to the maximum extent permitted by law. The product specifications do not expand or otherwise modify Vishay’s terms and conditions of purchase, including but not limited to the warranty expressed therein, which apply to these products.

No license, express or implied, by estoppel or otherwise, to any intellectual property rights is granted by this document or by any conduct of Vishay.

The products shown herein are not designed for use in medical, life-saving, or life-sustaining applications unless otherwise expressly indicated. Customers using or selling Vishay products not expressly indicated for use in such applications do so entirely at their own risk and agree to fully indemnify Vishay for any damages arising or resulting from such use or sale. Please contact authorized Vishay personnel to obtain written terms and conditions regarding products designed for such applications.

Product names and markings noted herein may be trademarks of their respective owners.