Bifröst : Covert Data Exfiltration from Air-gapped Network via Smart Bulbs

Muhammad Arsath KF, Sourav Das, Keerthi K, and Sugandha Tiwari
(Mentored by Chester Rebeiro)

Indian Institute of Technology Madras

Motivation

With the rapid advancements in the Internet of Things (IoT), smart homes and offices are going to be very popular in the near future. Houses would be equipped with various electronics, which promise to make our lives much more comfortable. Smart bulbs would be an integral part of smart homes. These bulbs, would permit automatic control of lighting in a room. For example, an intelligent agent may control the color and intensity of the light emitted by the smart bulb, thus changing the ambience of the room based on the context. A far more important advantage is that smart bulbs can reduce energy consumption drastically by turning bulbs off and controlling the intensity.

Smart homes, however are a double edged sword. There are growing security and privacy concerns with the increasing 'smartness' of the electronic equipment used at homes. Many of the equipment control critical operations that are related to safety and security of homes. Further, many of the equipment monitor sensitive activities, and protecting this information is going to be far more difficult than what the current security mechanisms can achieve.

This project demonstrates one such security concern, where smart bulbs can be used to steal sensitive data by means of a covert channel. The channel, which we call ***Bifröst***[1], demonstrates how a low cost electronic equipment, such as a smart bulb, can cause major security concerns. ***Bifröst*** permits data to be leaked over the air using various controllable parameters of a typical smart bulb. For demonstration, we have implemented our attacks using the Magic Bluetooth bulb[2]. This smart bulb permits control of various parameters such as turning on/off, color and intensity of the light, over Bluetooth. We demonstrate multiple attacks of how a malicious application in the control computer can create a covert channel using the tune able knobs present in the bulb thus leaking sensitive data over an air gap. We have generalized the implementations in such a way that, this will work for other smart bulbs with minor changes. We also evaluated trade offs between efficiency and error rate of the covert channel.

Background

Before going into the attack plan of ***Bifröst***, we need to have an understanding about smart homes infrastructure. A Smart Home is an infrastructure where a home automation system controls home appliances, temperature, light and all the other devices. A smart home automation system is typically controlled by a central hub and can be a completely

[1] The rainbow bridge of the gods from their realm Asgard to earth

[2] https://www.amazon.in/Generic-Bluetooth-lighting-dimmable-AC85-265V/dp/B06XC6DQL4

air-gapped network. A Smart Bulb is an LED bulb that allows the lightning to be customized and controlled remotely. These devices are also connected to the internal network and need a remote controller (or controller application) to change the intensity, color and the brightness of the bulb. Some smart bulbs also provide advanced features such as the change in the color based on the change in environmental conditions or with the arrival of new emails etc.

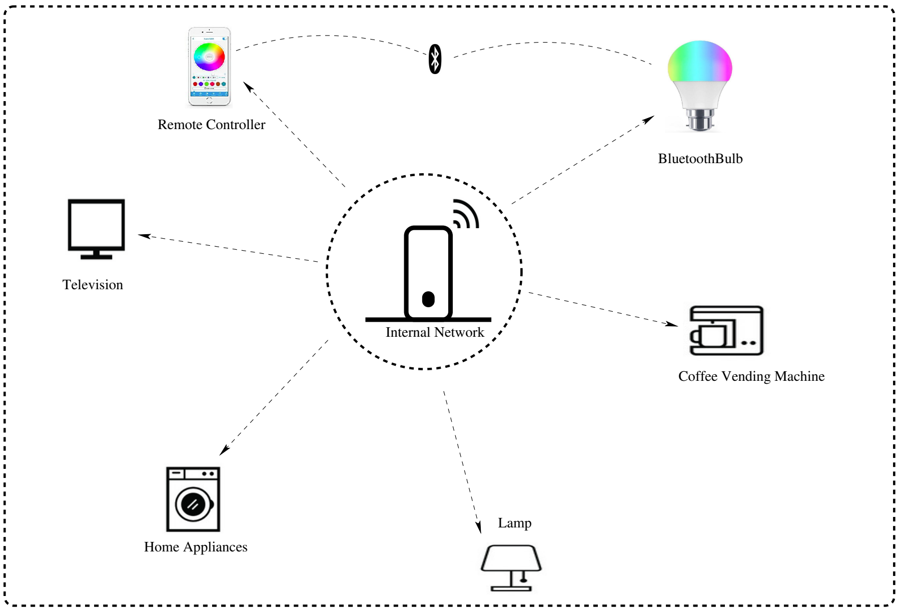


Figure 1: Smart Home Set-up of the Lean Enterprise Advanced Knowledge Solutions (LEAKS), Employs a complete air-gap separation for all its internal networks, ensuring no data is leaked through public connections. The smart bulb color is changed using the remote controller using a Bluetooth connection.

In Smart Homes, the internal communication is managed by a hub that is connected to all the devices as shown in Figure 1. The routers can be connected to the devices using electrical lines **X10**[3], radio waves **ZigBee**[4] and **Z-Wave**[5], or **Insteons**[6]. Smart bulb use an RF receiver which enables the bulb to connect to the controller. Most bulbs can directly connect to the Bluetooth or Wi-Fi network. The smart bulb will receive the command to perform some action either from a controller or directly from a mobile application. The commands received by the smart bulb are sent for execution to a processing unit. The processing unit controls the LED inside the bulb by using PWM (Pulse Width Modulation). Each smart bulb will contain a set of LEDs and drivers to controls the LEDs. Different colors and brightness level in the bulb are obtained by sending different signals to the driver.

[3]https://www.x10.com/
[4]https://www.digi.com/resources/standards-and-technologies/zigbee-wireless-standard
[5]https://www.z-wave.com/
[6]https://www.insteon.com/

Reverse Engineering the Bulb

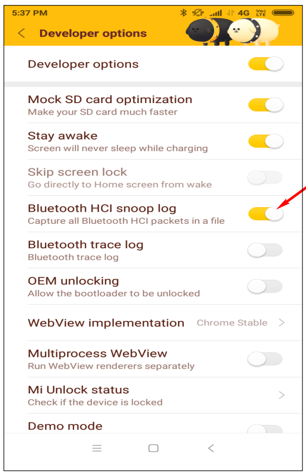
This was one of the challenging jobs for us, as none of us is dealing with Bluetooth applications in our research. We have found many research articles, but most of them are incomplete about the reverse engineering aspects of smart-bulb. After spending many sleepless days, we found the method for reverse engineering the bulb. This reverse engineering is done also with the help other android application. As the development of those applications was out of the scope of our work, we just decided to trace the patterns that are needed to be sent for different applications. After spending some days, we conquered the challenge!. We could ON/OFF the bulb, without using the official app for the bulb. That was one of the exciting moment when we realized that we can have entire control on BLE devices.

All smart devices including the smart bulb are distinguished by their unique device ID namely MACID or device address. Each of these Bluetooth Low Energy devices provide certain services, which can be determined by their unique UUID's (Universally unique identifier). Each service has certain characteristics along with certain descriptions. The major aim of reverse engineering is to capture the information such as which of these services and the characteristics are writable, which helps us to alter the data and change the functionality. The steps for reverse engineering the bulb for exploitation as follows :

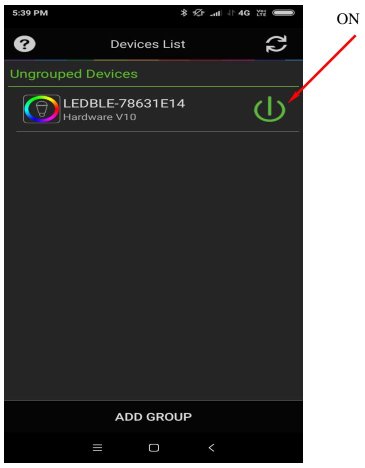
- *Monitor Bluetooth packet :* To capture the pattern of Bluetooth packet transferred from the smart bulb controller or the mobile application, we need to switch to developer mode in the device, and also need to enable the **"Bluetooth HCI snoop log"**.

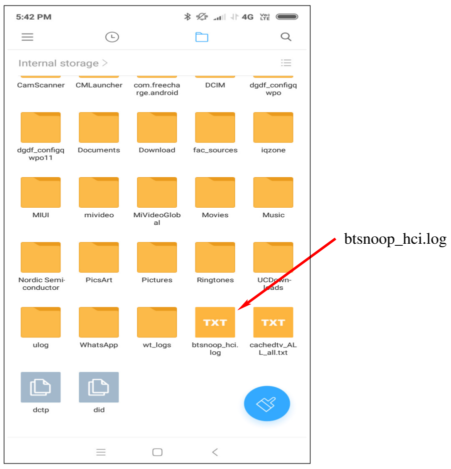
- *Activate mobile application:* The second step is to activate the standard smartphone application, which helps to control the bulb. The application can then be used to transfer some data by changing the color, intensity etc.

- *Monitor the log files:*All the packet that we send through Bluetooth is logged in the log file namely **btsnoop_hci.log**, which is internally stored in the smartphone.



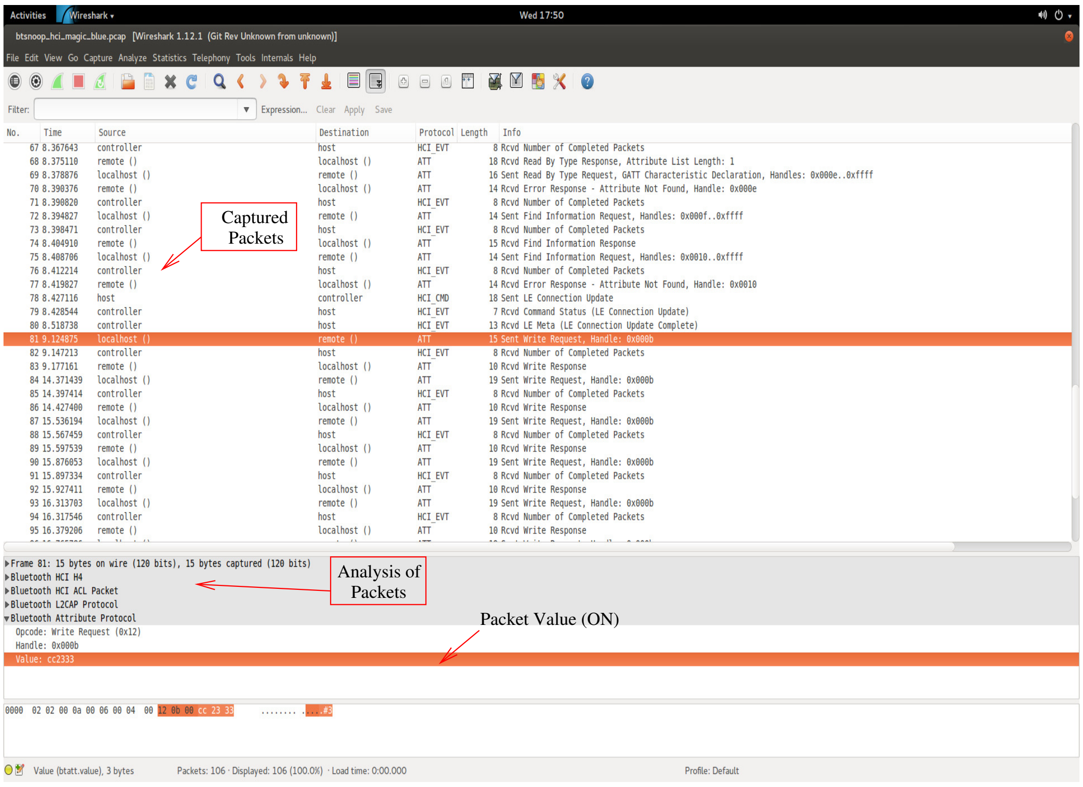
Developer options [✓ on]
Mock SD card optimization — Make your SD card much faster [✓ on]
Stay awake — Screen will never sleep while charging [✓ on]
Skip screen lock — Go directly to Home screen from wake [off]
Bluetooth HCI snoop log — Capture all Bluetooth HCI packets in a file [✓ on]
Bluetooth trace log — Bluetooth trace log [off]
OEM unlocking — Allow the bootloader to be unlocked [off]
WebView implementation — Chrome Stable
Multiprocess WebView — Run WebView renderers separately [off]
Mi Unlock status — Check if the device is locked
Demo mode [off]





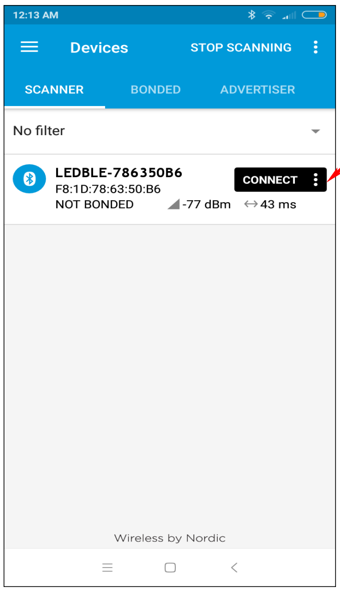
- *Tracking the packets:* Convert the log file to **btsnoop_hci.pcap** and capture the packet information from these files using Wireshark.

- *Monitoring Attributes*: Monitor the attributes such as Attribute protocol (ATT), which are responsible for device discovery, reading and writing data etc. Also monitor the value fields for different Bluetooth packets, to determine the information that is transmitted.

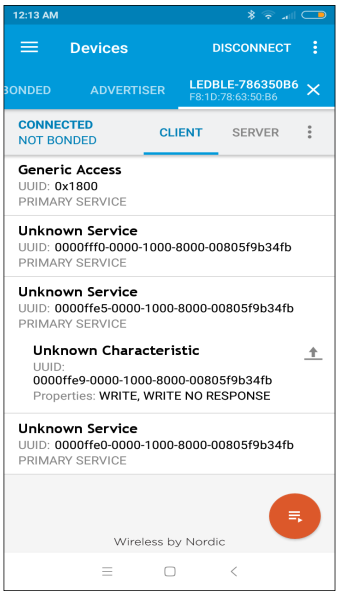

Activities
Wireshark
Wed 17:50
btsnoop_hci_magic_blue.pcap [Wireshark 1.12.1 (Git Rev Unknown from unknown)]
File Edit View Go Capture Analyze Statistics Telephony Tools Internals Help
Filter:
Expression... Clear Apply Save
Captured Packets
Analysis of Packets
Packet Value (ON)

```
No.  Time       Source        Destination   Protocol Length Info
 67 8.367643    controller    host          HCI_EVT     8 Rcvd Number of Completed Packets
 68 8.375110    remote ()     localhost ()  ATT        18 Rcvd Read By Type Response, Attribute List Length: 1
 69 8.378876    localhost ()  remote ()     ATT        16 Sent Read By Type Request, GATT Characteristic Declaration, Handles: 0x000e..0xffff
 70 8.390376    remote ()     localhost ()  ATT        14 Rcvd Error Response - Attribute Not Found, Handle: 0x000e
 71 8.390820    controller    host          HCI_EVT     8 Rcvd Number of Completed Packets
 72 8.394827    localhost ()  remote ()     ATT        14 Sent Find Information Request, Handles: 0x000f..0xffff
 73 8.398471    controller    host          HCI_EVT     8 Rcvd Number of Completed Packets
 74 8.404910    remote ()     localhost ()  ATT        15 Rcvd Find Information Response
 75 8.408706    localhost ()  remote ()     ATT        14 Sent Find Information Request, Handles: 0x0010..0xffff
 76 8.412214    controller    host          HCI_EVT     8 Rcvd Number of Completed Packets
 77 8.419827    remote ()     localhost ()  ATT        14 Rcvd Error Response - Attribute Not Found, Handle: 0x0010
 78 8.427116    host          controller    HCI_CMD    18 Sent LE Connection Update
 79 8.428544    controller    host          HCI_EVT     7 Rcvd Command Status (LE Connection Update)
 80 8.518738    controller    host          HCI_EVT    13 Rcvd LE Meta (LE Connection Update Complete)
 81 9.124875    localhost ()  remote ()     ATT        15 Sent Write Request, Handle: 0x000b
 82 9.147213    controller    host          HCI_EVT     8 Rcvd Number of Completed Packets
 83 9.177161    remote ()     localhost ()  ATT        10 Rcvd Write Response
 84 14.371439   localhost ()  remote ()     ATT        19 Sent Write Request, Handle: 0x000b
 85 14.397414   controller    host          HCI_EVT     8 Rcvd Number of Completed Packets
 86 14.427400   remote ()     localhost ()  ATT        10 Rcvd Write Response
 87 15.536194   localhost ()  remote ()     ATT        19 Sent Write Request, Handle: 0x000b
 88 15.567459   controller    host          HCI_EVT     8 Rcvd Number of Completed Packets
 89 15.597539   remote ()     localhost ()  ATT        10 Rcvd Write Response
 90 15.876053   localhost ()  remote ()     ATT        19 Sent Write Request, Handle: 0x000b
 91 15.897334   controller    host          HCI_EVT     8 Rcvd Number of Completed Packets
 92 15.927411   remote ()     localhost ()  ATT        10 Rcvd Write Response
 93 16.313703   localhost ()  remote ()     ATT        19 Sent Write Request, Handle: 0x000b
 94 16.317546   controller    host          HCI_EVT     8 Rcvd Number of Completed Packets
 95 16.379206   remote ()     localhost ()  ATT        10 Rcvd Write Response

▸Frame 81: 15 bytes on wire (120 bits), 15 bytes captured (120 bits)
▸Bluetooth HCI H4
▸Bluetooth HCI ACL Packet
▸Bluetooth L2CAP Protocol
▾Bluetooth Attribute Protocol
   Opcode: Write Request (0x12)
   Handle: 0x000b
   Value: cc2333

0000  02 02 00 0a 00 06 00 04  00 12 0b 00 cc 23 33     ........ .....#3

Value (btatt.value), 3 bytes     Packets: 106 · Displayed: 106 (100.0%) · Load time: 0:00.000     Profile: Default
```

- *nRF connect for sending the packets:* We have used nRF Connect app to send this value to characteristic **ffe9** of service **ffe5** - which is the writable service.



LEDBLE-786350B6
F8:1D:78:63:50:B6
NOT BONDED -77 dBm ↔43 ms



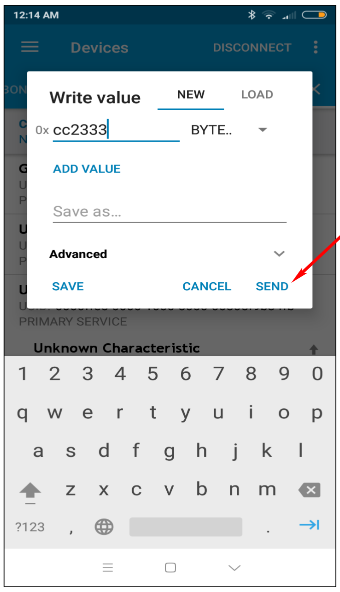
Generic Access
UUID: 0x1800
PRIMARY SERVICE

Unknown Service
UUID: 0000fff0-0000-1000-8000-00805f9b34fb
PRIMARY SERVICE

Unknown Service
UUID: 0000ffe5-0000-1000-8000-00805f9b34fb
PRIMARY SERVICE

Unknown Characteristic
UUID:
0000ffe9-0000-1000-8000-00805f9b34fb
Properties: WRITE, WRITE NO RESPONSE

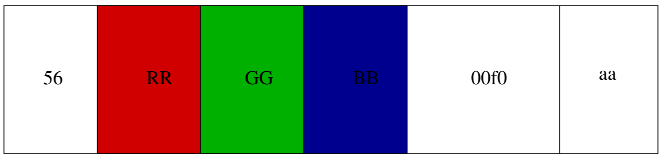
Unknown Service
UUID: 0000ffe0-0000-1000-8000-00805f9b34fb
PRIMARY SERVICE



0x cc2333

we've hacked the bulbs -now for (more) fun!

So now we've hacked the bulb by reverse engineering and now we could easily change the colors of smart bulb without the help of official app. The **UUID** of the Magic Bluetooth bulb is as follows: The first byte is always **56**, then three bytes for the **RED**, **GREEN** and **BLUE** values, and then another three bytes with values **00f0 aa**. We can change different colors by varying these values.



As next step we need to have a floor plan for the covert data exfiltration. The plan need an android mobile application for the sender side to control the bulb.

Bifröst : The Proposed Covert Channel Method

Smart bulb hacking is the first step for covert channel data exfiltration. The second part of the covert channel comprises of the protocol for sending and retrieving the data from smart bulb. The task of data exfiltration is not so simple and needs more time for getting an accurate transmission. Also the speed of transmission and error rate is a trade off for the covert channel. For successful data exfiltration, we would need to make certain assumptions for the implementation of the covert channel.

Assumptions

The assumptions that we adopted place a major role in the covert channel data exfiltration, which we enumerate here.

1. A major assumption is that one of the internal nodes in the air-gapped network is compromised, and which helps to send data to the outside world with the help of the smart bulb.

2. The compromised device uses a malicious application that helps to change the color of the smart bulb, based on the encoding technique the sender and the receiver decide apriori.

3. The maximum distance at which the attack can be possible is based on the strength of the receiver. We evaluated two low cost receivers: a webcam and light sensor. The attack strength can be increased with the help of a high-resolution camera and a telescope to focus the smart-light.

4. We make use of two receiver setup in two different scenario. If the smart bulb is fixed to a single color, then the message encoding and decoding are done using different intensity levels of the particular color. Also if the bulb is switching between multiple colors the encoding and decoding scheme depends on the number of colors that can be detected using the webcam.

5. For long distance attack, we assume that there are no environmental disturbances (such as heavy rain and storms), which can affect the correctness.

Overview

Our covert channel data exfiltration mainly includes two parts : a transmitter (sender) and a receiver.

- The transmission setup comprises of a smart bulb (which is visible to the outside world) that is connected to an air-gapped network. The smart bulb can be controlled by the official application with in the network. We have used our malicious application to control the smart bulb based on the requirements.
- The receiver setup is based on the attack plan we choose. i.e, we assume two attack plans as follows:
  - The receiver setup includes a webcam to detect color, light, and intensity of the emitted light.
  - the setup includes an android light sensor application that can detect different light intensity (when the bulb is fixed to a single color).

The bulb we consider has two important features. It can provide different shades of a single color and can also switch between multiple colors (can call it as a disco mode!). We use these two features for constructing two different covert channels and demonstrate that each of these features result in different properties for the covert channel. The first plan (plan 1) uses different color shades of the same color, while the second plan (plan 2) uses different colors for transmission

In the figure below, we provide a pictorial representation of the entire attack plan. For demonstration we include the receiver setup for attack plan 2 (when bulb switches between multiple colors).

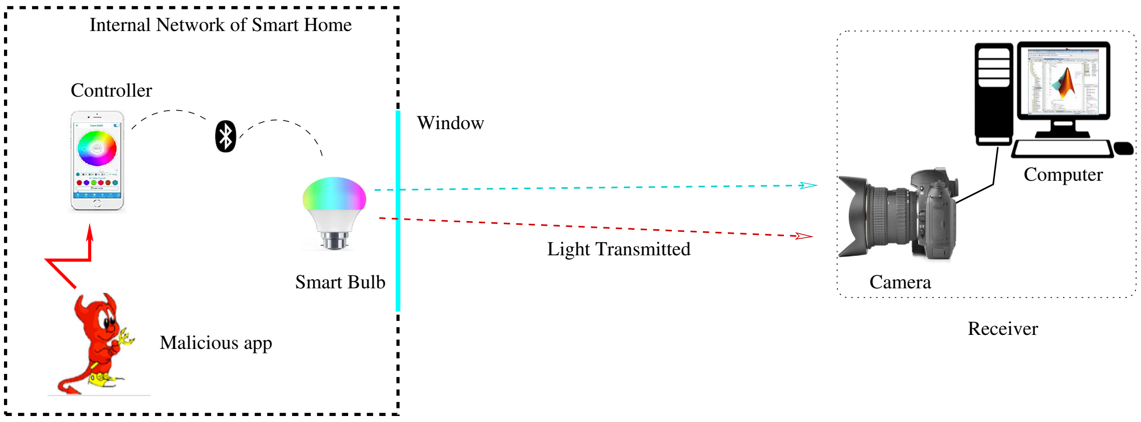


Figure 2: Setup for attack Plan 2 of **BIFRÖST**. Within the internal network, the smart bulb controller contains a malicious application that manages the changes in color and intensity of the smart bulb. Receiver setup for **ATTACK PLAN 2** includes a Camera and the computer, and for **ATTACK PLAN 1** includes a light sensor application

Figure 3 shows the overall implementation plan of **Bifröst**. The block diagram includes the step by step implementation of the sender and the receiver setup, for encoding and decoding the message. The block diagram includes the algorithms that we used for sender and the receiver implementation. We have also included the steps where we failed in the implementation.

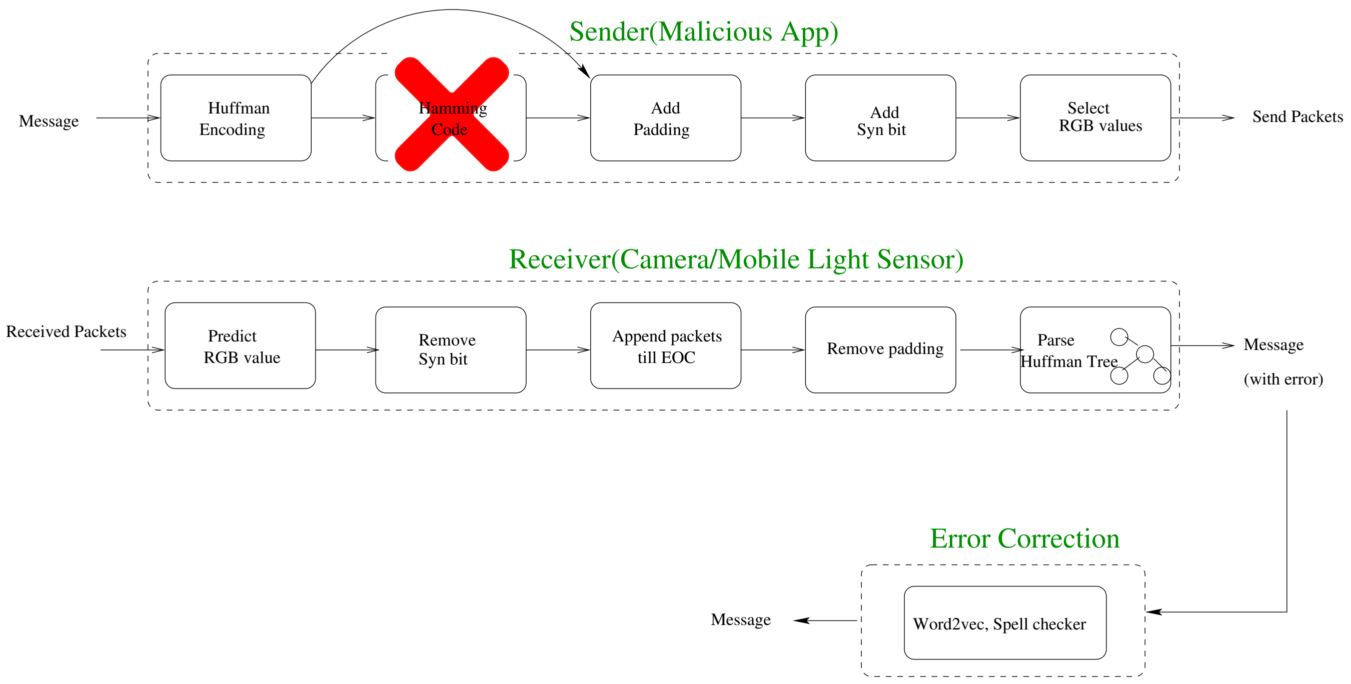


Figure 3: Overall implementation plan of **BIFRÖST**. The sender(malicious application) - procedure for converting the message to packets. The decoding method is same in both the attack plans except the platform for implementation.

Encoding Technique (some encoding results!)

The sender in Figure 3, gives the overall idea about the encoding technique. We have used huffman encoding for encoding the message. We also have added padding bits and synchronization bits wherever necessary for encoding the message. We would like to demonstrate some important details about the algorithms with some examples in the following part of the section.

Huffman encoding

We have used Huffman encoding[7] for converting the message to bit strings. The major advantage of Huffman encoding is that the average number of bits is less compared to ASCII encoding. For example consider the message in the text box below:

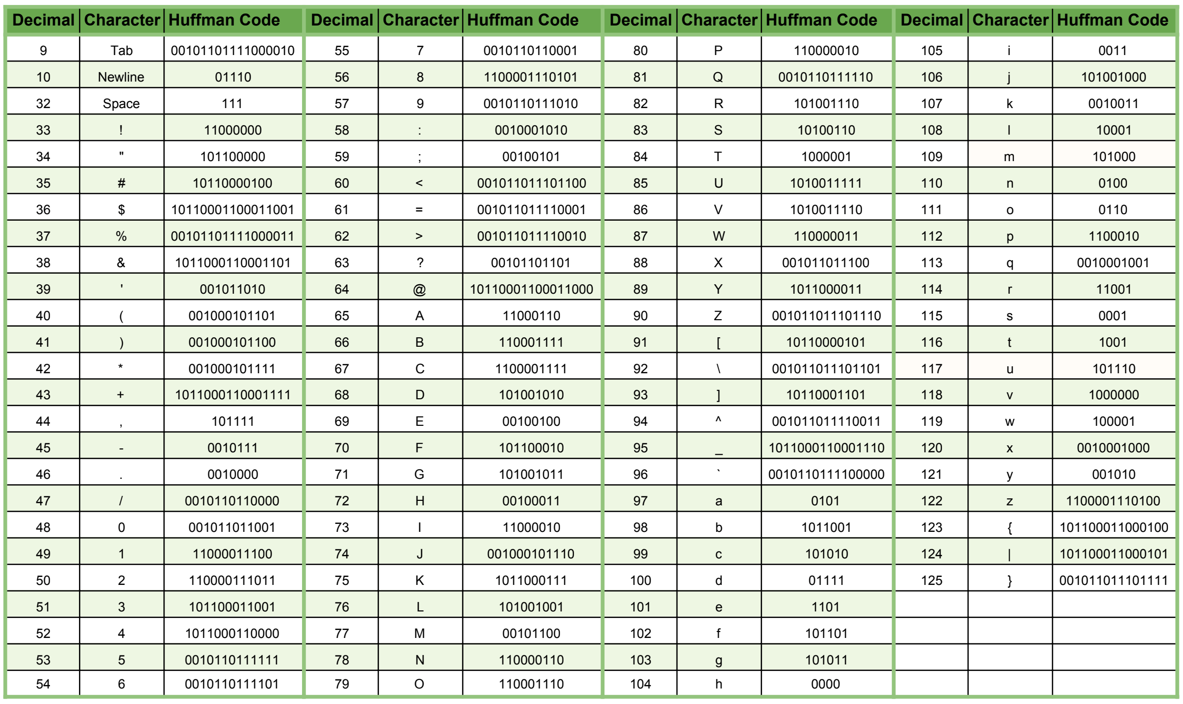
"The quick brown fox jumps over the lazy dog.Its the possibility of having a dream come true that makes life interesting.You never really understand a person until you consider things from his point of view.We accept the love we think we deserve.There is no greater agony than bearing an untold story inside you.Nothing in life is as important as you think it is, while you are thinking about it.All the Light We Cannot See.But What if We are Wrong.Its only after we have lost everything that we are free to do anything.Jesuschrist@12345"

These set of messages were used for our analysis. The total number of characters is 536. The performance gain in using Huffman encoding is as follows:

[7] https://ieeexplore.ieee.org/document/4051119

```
Total chars = 536 ==> Total bits (ASCII)   = 536*7 = 3752
                      Total bits (Huffman) = 2496
                      Avg.bits per char(Huffman)  = 4.66
                      Performance gain   = 1256 bits = 33.4 %
```

We have created the Huffman tree based on the *"Bhagvadgita.txt"* file, which includes almost all the ASCII characters. Following table includes the final bit pattern for each character for our encoding scheme.



Decimal	Character	Huffman Code	Decimal	Character	Huffman Code	Decimal	Character	Huffman Code	Decimal	Character	Huffman Code
9	Tab	00101101111000010	55	7	0010110110001	80	P	110000010	105	i	0011
10	Newline	01110	56	8	1100001110101	81	Q	0010110111110	106	j	101001000
32	Space	111	57	9	0010110111010	82	R	101001110	107	k	0010011
33	!	11000000	58	:	0010001010	83	S	10100110	108	l	10001
34	"	101100000	59	;	00100101	84	T	1000001	109	m	101000
35	#	10110000100	60	<	001011011101100	85	U	1010011111	110	n	0100
36	$	10110001100011001	61	=	001011011110001	86	V	1010011110	111	o	0110
37	%	00101101111000011	62	>	001011011110010	87	W	110000011	112	p	1100010
38	&	1011000110001101	63	?	00101101101	88	X	001011011100	113	q	0010001001
39	'	001011010	64	@	10110001100011000	89	Y	1011000011	114	r	11001
40	(	001000101101	65	A	11000110	90	Z	001011011101110	115	s	0001
41	)	001000101100	66	B	110001111	91	[	10110000101	116	t	1001
42	*	001000101111	67	C	1100001111	92	\	001011011101101	117	u	101110
43	+	1011000110001111	68	D	101001010	93	]	10110001101	118	v	1000000
44	,	101111	69	E	00100100	94	^	001011011110011	119	w	100001
45	-	0010111	70	F	101100010	95	_	1011000110001110	120	x	0010001000
46	.	0010000	71	G	101001011	96	`	0010110111100000	121	y	001010
47	/	0010110110000	72	H	00100011	97	a	0101	122	z	1100001110100
48	0	001011011001	73	I	11000010	98	b	1011001	123	{	101100011000100
49	1	11000011100	74	J	001000101110	99	c	101010	124	|	101100011000101
50	2	110000111011	75	K	1011000111	100	d	01111	125	}	001011011101111
51	3	101100011001	76	L	101001001	101	e	1101			
52	4	1011000110000	77	M	00101100	102	f	101101			
53	5	0010110111111	78	N	110000110	103	g	101011			
54	6	0010110111101	79	O	110001110	104	h	0000			

Add Padding Bits and Synchronization bit

The number of bits per packet depends on the number of colors or intensity that we can detect using the receiver setup. In order to have equal number of bits per packet we add *padding bits* wherever required . The addition of *synchronization bit (syn bit)* is to distinguish between two incoming packets and to discard duplicate packets.

We have used character by character encoding scheme for better efficiency. Hence at the end of each character we need to send *EOC (End of Character)*, which can be an out of range intensity in plan1 and different color in plan2.

Demonstrating an example of encoding in attack plan1 - with varying intensity. Consider that the sender want to send message **"Eat"**. The figure below shows the various levels of intensity for bulb fixed to red color. Assume that we have fixed the intensity range from 100 -228, where below 100 and above 228 can be used as EOC.

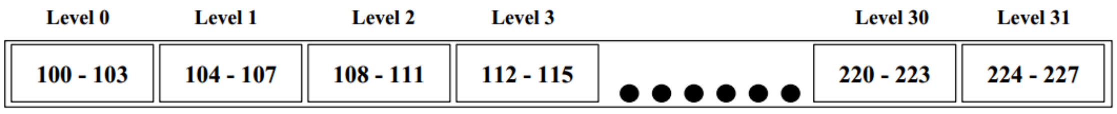


Figure 4: Different intensity levels for red light, where the width of each levels is 4

For encoding the message **Eat**, where the binary values of each character is as follows:

E : **00100100** **a** : **0101** **t** : **1001**

Since we have the usable range from 100 - 228 and the width of each level is 4, therefore we can have 32 different levels(resembling 5 bits per packet). As each packet contains 4 bit information and a syn bit so a single character can be splitted into multiple packets. The end of character can be detected using EOC. The implementation is shown in figure below. After adding the syn bit and padding bits the hexadecimal value of message bit is taken and appended with the RGB values of each packets.

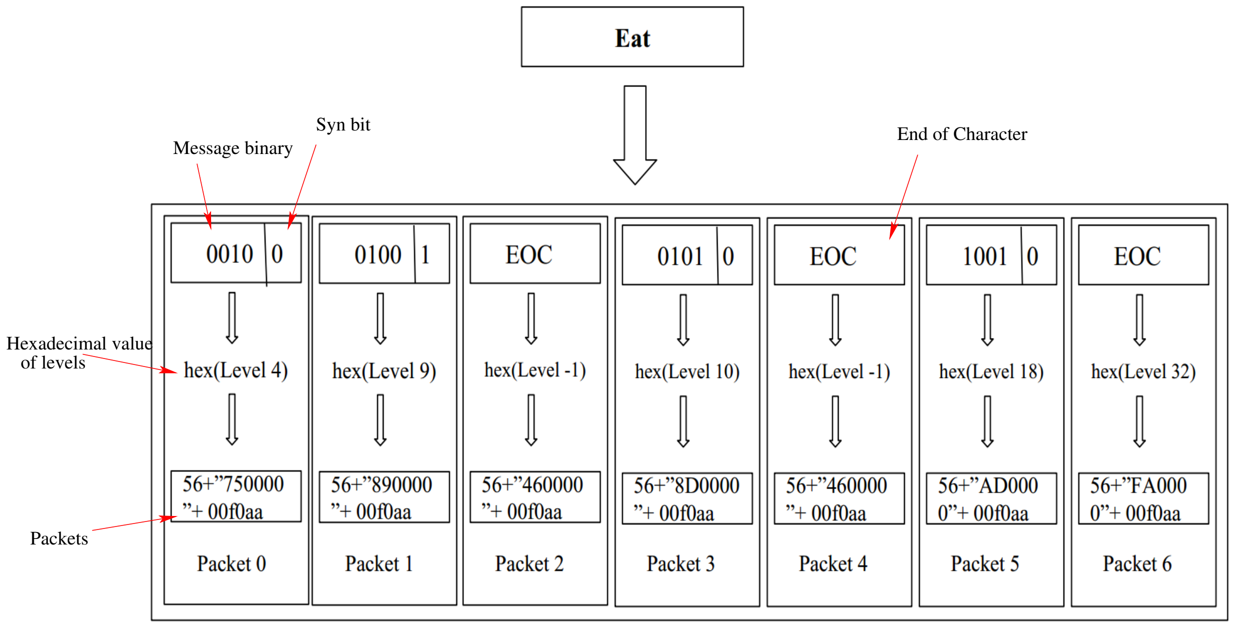


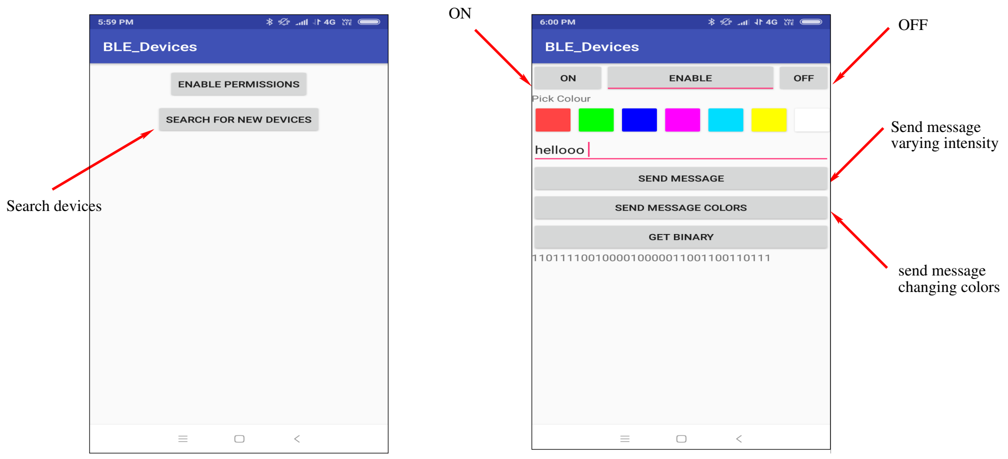
Figure 5: The conversion of the packets to different intensity levels. Conversion uses the intensity levels given in Figure 4.

Malicious Application

The malicious application that we developed is the back bone of ***Bifröst***. Without this application, our attack plan is practically impossible. The two important feature on the application is as follows:

(1) *Bulb switching between multiple colors* : Assume that smart bulb is switching between multiple colors, then we require a malicious application to change the color of the bulb, depending on the data that has to be transmitted.

(2) *Bulb fixed to a single color*: Assuming that the bulb is fixed to a single color, then covert channel data exfiltration needs to be restricted to that specific color (assume bulb is fixed to red color). In order to transmit the message effectively, we need to play around with different intensity levels of red.



hellooo

1101111001000010000011001100110111

The android app have different features, such as *send message* and *send message colors*, which can be used for attack plan 1 and plan 2. The different color buttons in the app is for sending that particular color to the bulb and *get binary* is for getting the binary of the string that we have sent. The *enable* button is for enabling the multiple color mode.

Receiver Setup with Webcam:

The aim of this attack is sending data when the bulb is in multi-color mode and the attacker is taking advantage of this mode for data transmission. But the challenging task here is to send the data as stealthy as possible in such a way that the change in the color should not be perceptible to the outside world. In this setup, we used a webcam for reading/capturing colors from the bulb. All the processing steps are done using MATLAB setup, with the help of the webcam connected to the system. The following images depict the setup of webcam that captures color values at 30fps. Figure 6 shows the overall view of the receiver setup.

Each data transmission is enclosed by start and end packets as shown in the encoding part. The packets send to the bulb are captured using Webcam, which is connected to a MATLAB setup. As soon as the start of packet is received, all the upcoming packets are stored till the End of packet is received. Based on the stored packets we will calculate the RGB values of each frame. Considering the example of **"Skynet is alive!"**, the RGB value obtained is as shown in Figure 7.

MATLAB decoding script will scan the received RGB stream and develop color stream on each frame. These color stream will be processed further for extracting the information. We have used 8 basic colors such as Cyan, White, Blue, Pink, Green, Orange, Red and Yellow for date exfiltration. As we could send 3 bit at time, we make use of 2 bit for information and 1 bit for synchronization bit. An example color stream, that we processed for the above message is as follows:

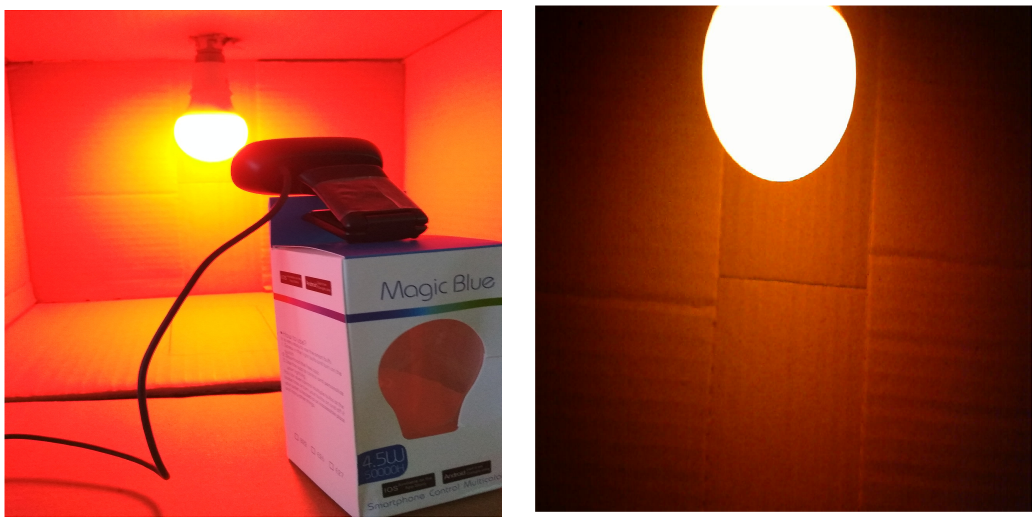


Figure 6: The receiver setup when bulb is switching between multiple colors. The right side shows the webcam view in MATLAB for processing the data.

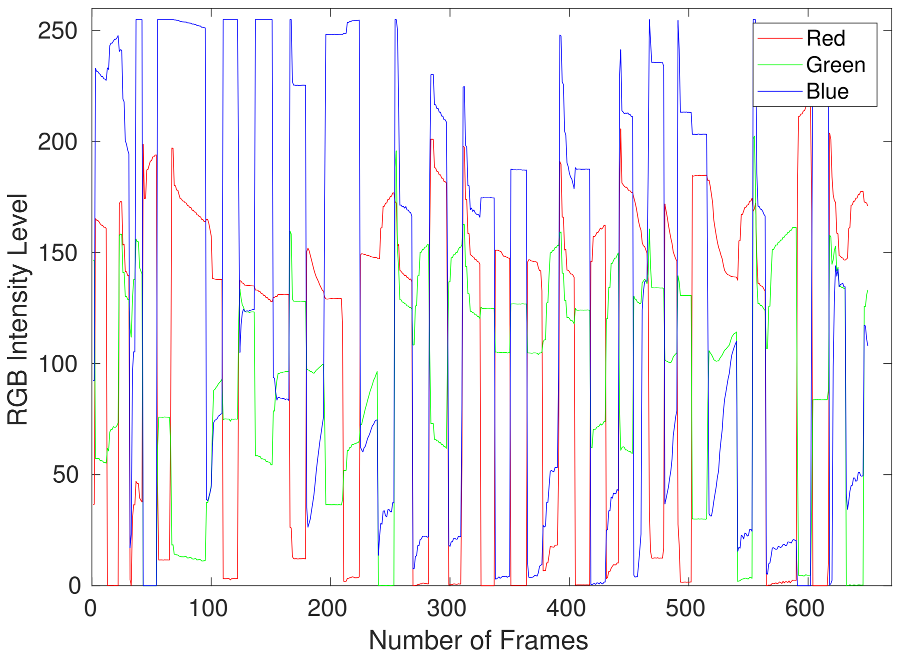


Figure 7: RGB values of the message in terms of number of frames vs intensity levels

All the packets are appended together after removing the syn bit, until we receive the end of character. The step of decoding is straight forward as encoding, the bit pattern is matched with the Huffman tree. The advantage of decoding character by character is that a bit error in one character will not affect the upcoming characters and provide better efficiency.

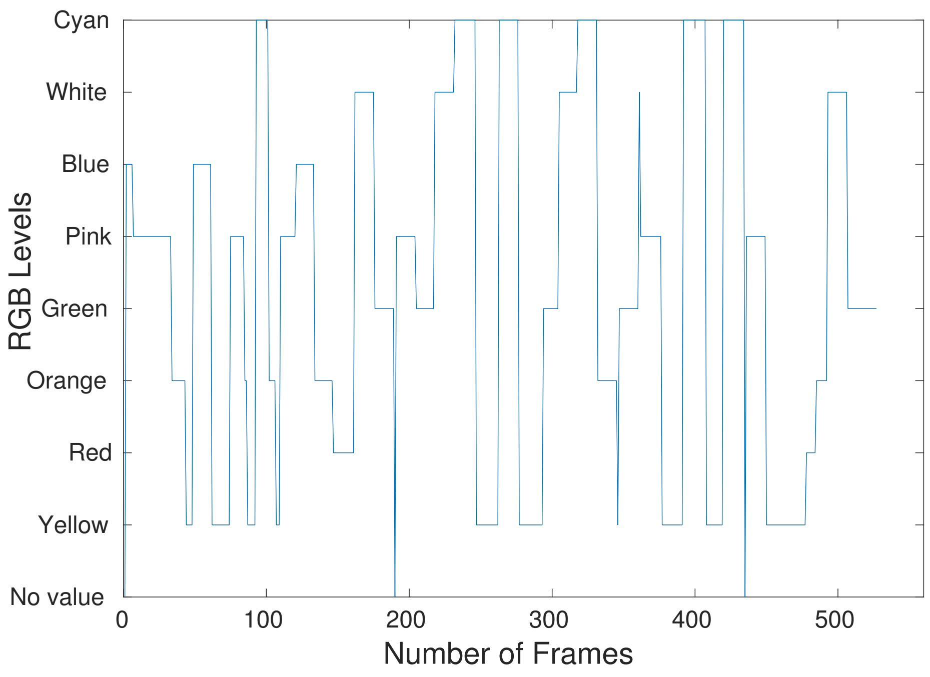
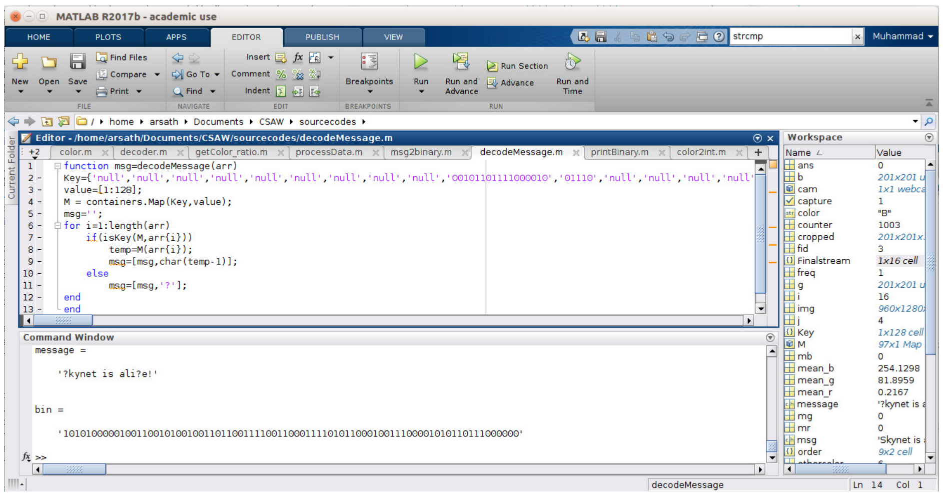


Figure 8: Prediction of colors using RGB values



```
function msg=decodeMessage(arr)
Key={'null','null','null','null','null','null','null','null','null','0010110111100001O','01110','null','null','null','null'
value=[1:128];
M = containers.Map(Key,value);
msg='';
for i=1:length(arr)
    if(isKey(M,arr{i}))
        temp=M(arr{i});
        msg=[msg,char(temp-1)];
    else
        msg=[msg,'?'];
end
end
```

```
message =

    '?kynet is ali?e!'


bin =

    '10101000000100110010100100110110011110011000111101011000100111000010101101110000000'

>>
```

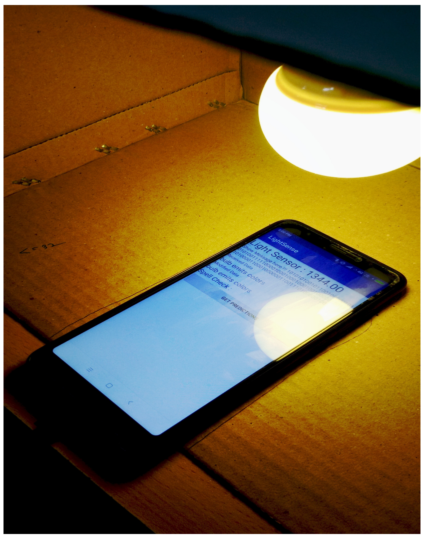
Name	Value
ans	0
b	201x201 u
cam	1x1 webca
capture	1
color	"B"
counter	1003
cropped	201x201x
fid	3
Finalstream	1x16 cell
freq	1
g	201x201 u
i	16
img	960x1280
j	4
Key	1x128 cell
M	97x1 Map
mb	0
mean_b	254.1298
mean_g	81.8959
mean_r	0.2167
message	'?kynet is a
mg	0
mr	0
msg	'Skynet is
order	9x2 cell

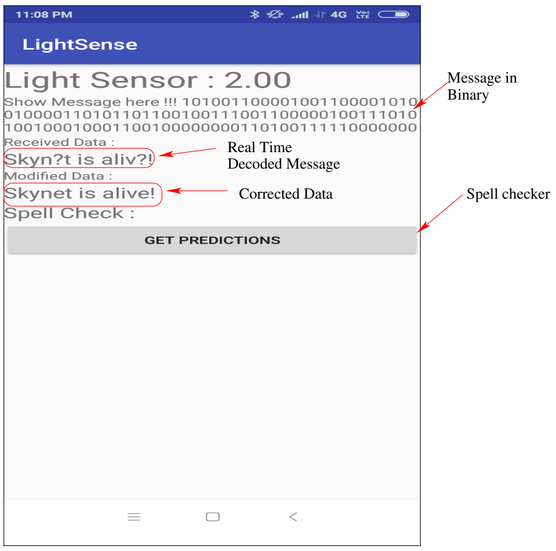
The above figure shows that the MATLAB script decodes the information with few errors.In the later part, we will see how these errors can be corrected using error correction schemes.

Receiver Setup with Light Sensor

The implementation of the attack plan is same as that the overall plan given in Figure 3. We have developed the android application that makes use of the intensity values from the light sensor and decodes the message based on the obtained values.

The figure below shows the receiver setup that we have used to exfiltrate the data from the smart bulb. The left one shows how it detects the variation in light intensity and right one shows the actual view of the android application, which displays the real time decoded message.

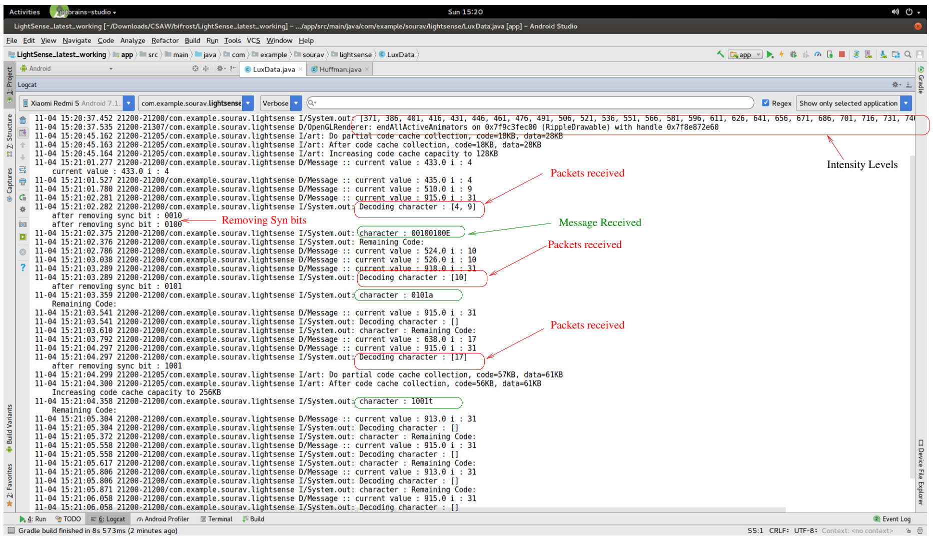




```
11:08 PM                                   4G
LightSense
Light Sensor : 2.00
Show Message here !!! 1010011000010011000010100100001101011011001001110011000001001110101001000100011001000000001101001111110000000
Received Data :
Skyn?t is aliv?!
Modified Data :
Skynet is alive!
Spell Check :
GET PREDICTIONS
```

We have used the reverse of the encoding scheme as shown in Figure 5. The decoding process should generate the intensity levels based on the current values obtained from the light sensor. From the intensity values obtained, we need to predict the decimal value, then remove the syn bits from the binary and append the packets until we get an EOC. The obtained binary is parsed through the Huffman tree to decode the character.

Figure 9 shows the decoding of the message at the receiver end. The *intensity levels* shows the matrix created at the receiver for the red light. After the range is fixed, the variation in the intensity levels is measured to obtain the packets(marked in red - Packets received). Append the packets after removing the syn bits. The final characters obtained is marked in green.



```
11-04 15:20:37.452 21200-21200/com.example.sourav.lightsense I/System.out: [371, 386, 401, 416, 431, 446, 461, 476, 491, 506, 521, 536, 551, 566, 581, 596, 611, 626, 641, 656, 671, 686, 701, 716, 731, 74
11-04 15:20:37.535 21200-21307/com.example.sourav.lightsense D/OpenGLRenderer: endAllActiveAnimators on 0x7f9c3fec00 (RippleDrawable) with handle 0x7f8e872e60
11-04 15:20:45.162 21200-21205/com.example.sourav.lightsense I/art: Do partial code cache collection, code=18KB, data=28KB
11-04 15:20:45.163 21200-21205/com.example.sourav.lightsense I/art: After code cache collection, code=18KB, data=28KB
11-04 15:20:45.164 21200-21205/com.example.sourav.lightsense I/art: Increasing code cache capacity to 128KB
11-04 15:21:01.277 21200-21200/com.example.sourav.lightsense D/Message :: current value : 433.0 i : 4
    current value : 433.0 i : 4
11-04 15:21:01.527 21200-21200/com.example.sourav.lightsense D/Message :: current value : 435.0 i : 4
11-04 15:21:01.780 21200-21200/com.example.sourav.lightsense D/Message :: current value : 510.0 i : 9
11-04 15:21:02.281 21200-21200/com.example.sourav.lightsense D/Message :: current value : 915.0 i : 31
11-04 15:21:02.282 21200-21200/com.example.sourav.lightsense I/System.out: Decoding character : [4, 9]
    after removing sync bit : 0010
    after removing sync bit : 0100
11-04 15:21:02.375 21200-21200/com.example.sourav.lightsense I/System.out: character : 00100100E
11-04 15:21:02.376 21200-21200/com.example.sourav.lightsense I/System.out: Remaining Code:
11-04 15:21:02.786 21200-21200/com.example.sourav.lightsense D/Message :: current value : 524.0 i : 10
11-04 15:21:03.038 21200-21200/com.example.sourav.lightsense D/Message :: current value : 526.0 i : 10
11-04 15:21:03.289 21200-21200/com.example.sourav.lightsense D/Message :: current value : 918.0 i : 31
11-04 15:21:03.289 21200-21200/com.example.sourav.lightsense I/System.out: Decoding character : [10]
    after removing sync bit : 0101
11-04 15:21:03.359 21200-21200/com.example.sourav.lightsense I/System.out: character : 0101a
    Remaining Code:
11-04 15:21:03.541 21200-21200/com.example.sourav.lightsense D/Message :: current value : 915.0 i : 31
11-04 15:21:03.541 21200-21200/com.example.sourav.lightsense I/System.out: Decoding character : []
11-04 15:21:03.610 21200-21200/com.example.sourav.lightsense I/System.out: character : Remaining Code:
11-04 15:21:03.792 21200-21200/com.example.sourav.lightsense D/Message :: current value : 638.0 i : 17
11-04 15:21:04.297 21200-21200/com.example.sourav.lightsense D/Message :: current value : 915.0 i : 31
11-04 15:21:04.297 21200-21200/com.example.sourav.lightsense I/System.out: Decoding character : [17]
    after removing sync bit : 1001
11-04 15:21:04.299 21200-21205/com.example.sourav.lightsense I/art: Do partial code cache collection, code=57KB, data=61KB
11-04 15:21:04.300 21200-21205/com.example.sourav.lightsense I/art: After code cache collection, code=56KB, data=61KB
    Increasing code cache capacity to 256KB
11-04 15:21:04.358 21200-21200/com.example.sourav.lightsense I/System.out: character : 1001t
    Remaining Code:
11-04 15:21:05.304 21200-21200/com.example.sourav.lightsense D/Message :: current value : 913.0 i : 31
11-04 15:21:05.304 21200-21200/com.example.sourav.lightsense I/System.out: Decoding character : []
11-04 15:21:05.372 21200-21200/com.example.sourav.lightsense I/System.out: character : Remaining Code:
11-04 15:21:05.558 21200-21200/com.example.sourav.lightsense D/Message :: current value : 915.0 i : 31
11-04 15:21:05.558 21200-21200/com.example.sourav.lightsense I/System.out: Decoding character : []
11-04 15:21:05.617 21200-21200/com.example.sourav.lightsense I/System.out: character : Remaining Code:
11-04 15:21:05.806 21200-21200/com.example.sourav.lightsense D/Message :: current value : 913.0 i : 31
11-04 15:21:05.806 21200-21200/com.example.sourav.lightsense I/System.out: Decoding character : []
11-04 15:21:05.871 21200-21200/com.example.sourav.lightsense I/System.out: character : Remaining Code:
11-04 15:21:06.058 21200-21200/com.example.sourav.lightsense D/Message :: current value : 915.0 i : 31
11-04 15:21:06.058 21200-21200/com.example.sourav.lightsense I/System.out: Decoding character : []
```

Activities
jetbrains-studio
Sun 15:20
LightSense_latest_working [~/Downloads/CSAW/bifrost/LightSense_latest_working] – .../app/src/main/java/com/example/sourav/lightsense/LuxData.java [app] – Android Studio
File Edit View Navigate Code Analyze Refactor Build Run Tools VCS Window Help
LightSense_latest_working app src main java com example sourav lightsense LuxData
LuxData.java
Huffman.java
Logcat
Xiaomi Redmi 5 Android 7.1.
com.example.sourav.lightsense
Verbose
Regex
Show only selected application
Intensity Levels
Packets received
Removing Syn bits
Message Received
Packets received
Packets received
4: Run
TODO
6: Logcat
Android Profiler
Terminal
Build
Event Log
Gradle build finished in 8s 573ms (2 minutes ago)
55:1 CRLF UTF-8

Figure 9: Decoding the Message using the receiver setup

Error Correction and Prediction using NLP

So far we have seen, Bifröst is capable of receiving messages through covert channel (with few errors in case of higher bandwidth and stealthy conditions). The rate of error will vary depending on the environmental conditions and the strength of receiver setup. The error rate can be effectively reduced with the help of a word predictor using spell checker as well as NLP based tools such as Word2vec(if both parties are agreed upon transmitting the domain specific messages).

In Bifröst, we have two level of error correction scheme. In the first level, most of the erroneous words can be easily corrected(one or two vowel character missing) using a python spell checker. Moreover if we know the domain/context of the data being transmitted, prediction can be done with the help of NLP tool Word2vec. It is a 2-layer neural network which stores distance between each word with other words that are present in the training data.

In the example given below, words with missing vowels are corrected with the help of spell checker, but which cannot be effective for multiple character errors. But word2vec is able to predict such errors with more accuracy.

Actual Message : **I really want to go to work, but I am too sick to drive.**

Text Received : **I rexlly waxt to go to w??k, b?t I am tao s?ck to dr?ve.**

After Spell Correction: **I really want to go to work but I am tao sick to drove**

After Error Prediction: **I really want to go to work but I am too sick to drive**

Evaluation

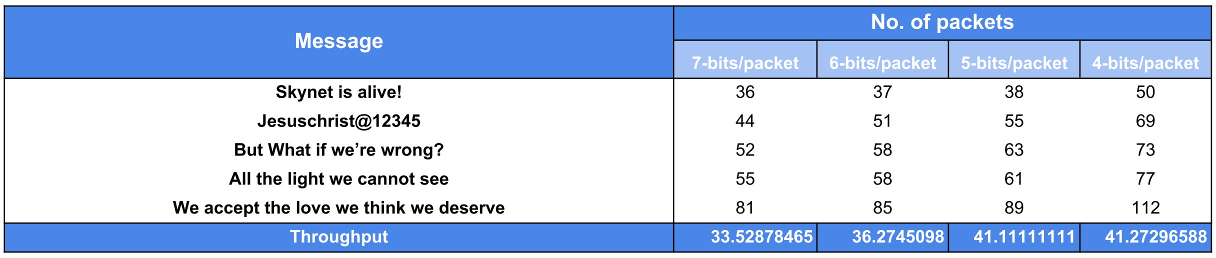
We would like to evaluate the effectiveness of Bifröst for the covert channel data exfiltration using smart bulb. Even though the demonstration of the covert channel is done on Magic Bluetooth Bulb, Bifröst attack plan can be efficiently used for other IoT devices also. We analyzed the Smart Bulb case study in several environmental conditions. We would like to demonstrate some of these result of the data exfiltration based on the factors of stealthiness, bit rate etc.

Throughput and Effectiveness :

Throughput and Effectiveness of the implemented attack includes bit rate at which the proposed method can exfiltrate data, and the distance at which these data can be meaningfully received and processed by an appropriate receiver.

- **When the receiver setup is mobile light sensor:** Figure 10 shows throughput that we obtained when we send different number of bits per packets. The table below shows the how the throughput value changes when the bits per packets decreases from 7 to 4.

  Figure 11 shows the error rate for various packet sizes. X axis represents transmitted bits per second and Y axis represents the error rate. i.e, for sending 4 bit per packets at a delay of 500 ms the transmitted number of bits per second is 7.9 bits. The graph shows the change in error rate when the packet information increases from 4 to 7.



Message	No. of packets			
	7-bits/packet	6-bits/packet	5-bits/packet	4-bits/packet
Skynet is alive!	36	37	38	50
Jesuschrist@12345	44	51	55	69
But What if we're wrong?	52	58	63	73
All the light we cannot see	55	58	61	77
We accept the love we think we deserve	81	85	89	112
Throughput	33.52878465	36.2745098	41.11111111	41.27296588

Figure 10: Throughput for different number of bits per packet

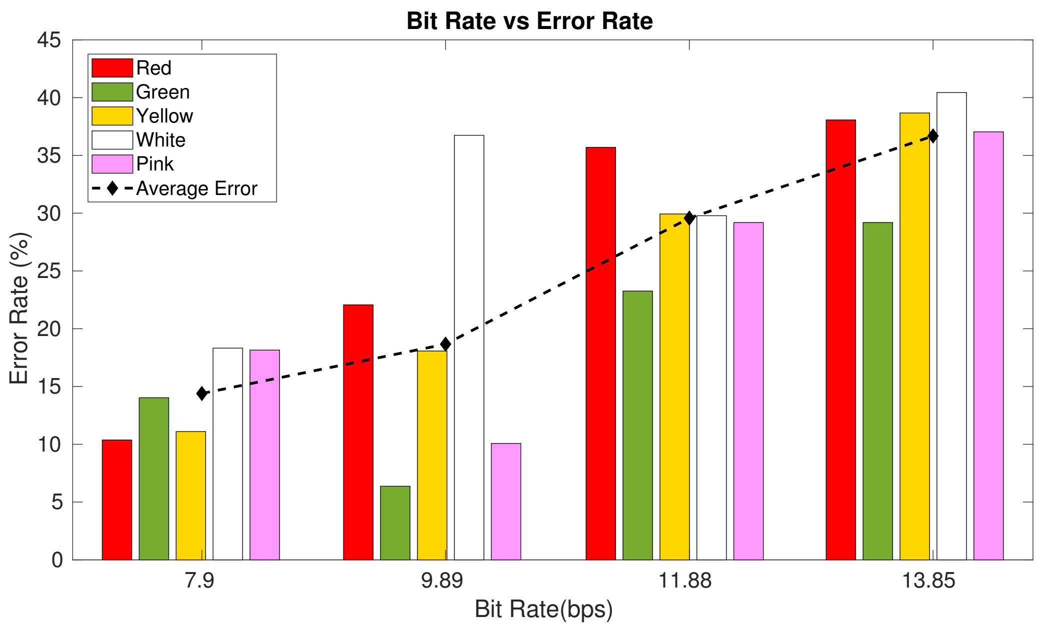


Figure 11: Graph for Bit rate vs error rate when the receiver setup is mobile light sensor

- **When the receiver setup is Webcam:** Figure 12 shows the bit rate vs success rate, for varying speed for sending the packets. The analysis is done for varying speed from 500 ms to 100 ms.

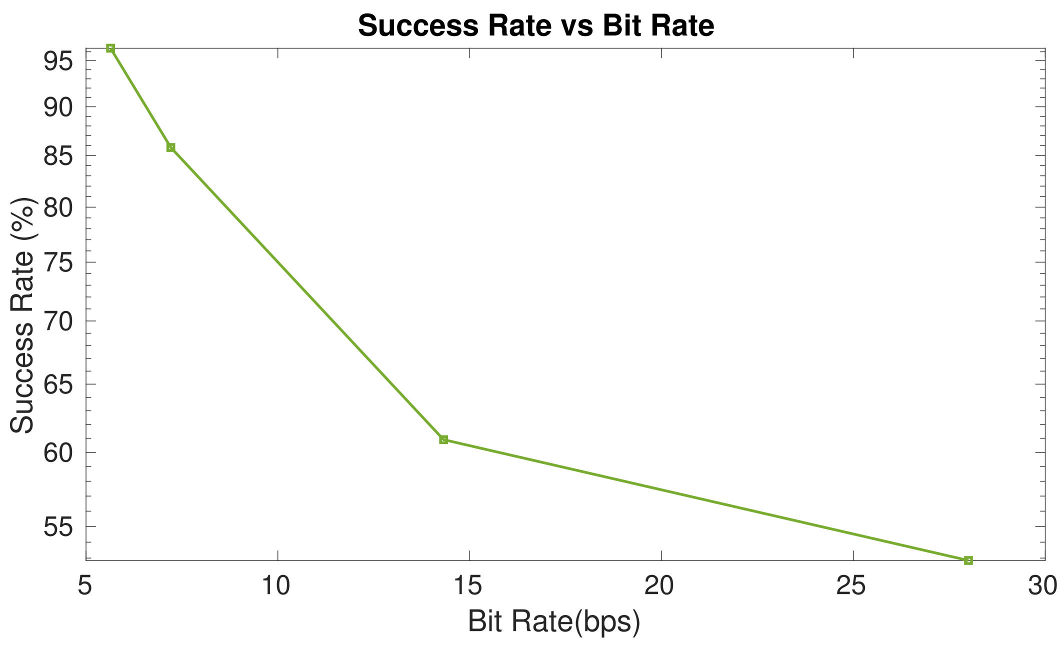


Figure 12: Graph for Bit rate vs error rate when the receiver setup is webcam

Stealthiness :

We define **Stealthiness** as the width of each level of intensity.

- **The receiver setup is light sensor**: We could have different intensity range that make the attack stealthier. We have done different analysis based on different width such as 8, 4, 2, 1. The stealthiness increases with the decrease in width from 8 to 1 providing that the bits per packet is constant. Graph 13 demonstrate the stealthiness vs error rate analysis done for different colors.

  From the graph it is clear that as the stealthiness levels changes from 8 to 1, the error rate increases. From the graph we could have a trade-off between error rate and stealthiness. So for better efficiency we need to increase the width for each levels.

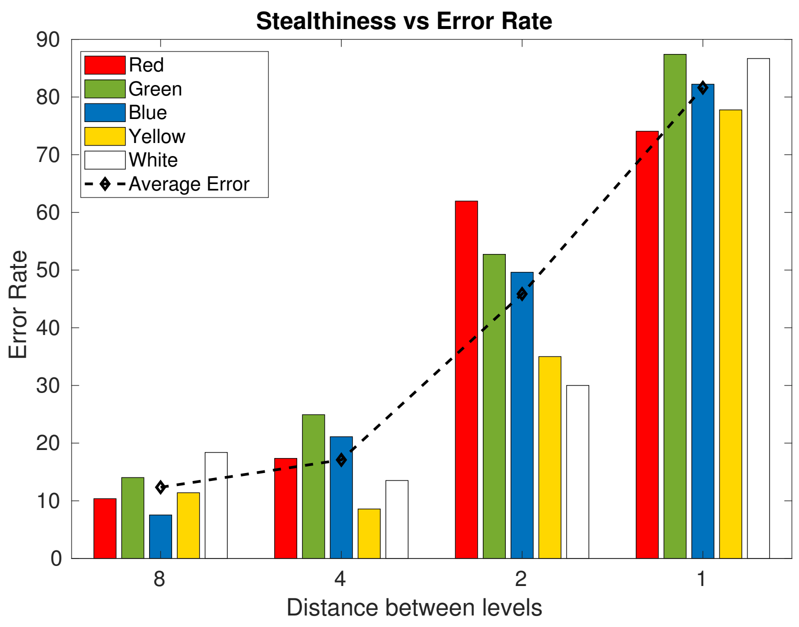


Figure 13: Graph for Bit rate vs error rate when the receiver setup is mobile light sensor

Generality :

Using the same reverse engineering technique we can hack any IoT device. The data exfiltration part varies depending on the IoT devices used. For example in smart bulb we used light as the medium where as in bluetooth speakers we may need some other parameters such as audio visualization. Hence these attack plans can be easily adopted for other IoT devices.

Analysis Results

We would like to demonstrate some more results for the various analysis that we have done for the both the receiver setup.

Noise Analysis :

For the receiver setup with mobile light sensor, we have added noise using the other bulb placed beside the bulb used for data exfiltration and by varying the intensity of the white light in the second bulb. Figure 14 shows that the increase in error rate when the surrounding white light intensity increases from 0 to 100 %. From the results it is clear that our covert channel model gives better results with less noise or interference.

Distance Analysis :

For the receiver setup with webcam, we have varied distance between the bulb and the webcam for covert data data exfiltration. Figure 15 shows the error rate vs distance. The figure signifies that error rate is directly proportional to the distance.

Noise Intensity	Message Received with Error
0	Skynet is alive!
10	Sk;net ih alive!
25	Sk;net ih Kk#ae!
50	_e'noef a igaLep
75	_rnno f a _gaL__
100	_rnnrt_____gnL__

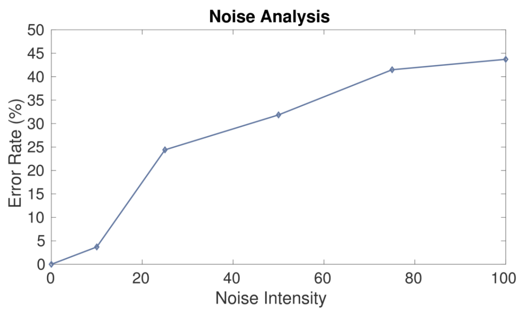


Figure 14: Graph for noise rate vs error rate

Distance Analysis (Bulb is placed at 45cm height and camera is at 25 cm height)			
Distance (cm)	20	30	40
Error Rate (%)	3.6	20.6	33.7

Figure 15: Graph for distance rate vs error rate

Some conclusion:

The table below shows the 3-D view of error-rate for a given bit rate and a stealth value. From the table it is clear that to obtain around 90 % accuracy we need to fix the bit rate to 7.9 bits per second (4 bits per packet) with the width between levels as 8.

From the table it is also clear that there is a trade off between speed and error rate. For better speed of 14 bits per second we could have an accuracy of 63%

Stealthiness	Bandwidth			
	7.9	9.89	11.88	13.85
8	12.344	-	-	-
4	17.098	18.66	-	-
2	45.862	-	29.57	-
1	81.626	-	-	36.68

Learning's and Inferences

Attack-I: Bulb switching between multiple color

- Initially we have tested our receiver(webcam) by placing around 5 meters from the bulb. Due to the environmental conditions and no option for focusing the bulb using the webcam, we failed to detect the the colors from the bulb.
- In order to exfiltrate the data from the received color stream, the initial plan was to sample a particular frame rather than parsing the entire frames based on the frequency at which the encoder is sending the data. But as we are using colors with different wavelengths, it was very challenging to go with the frequency.
- We tried online processing of data at decoder, which ended up in very less success rate. As there is a processing delay in decoder, due to which synchronization between the encoder and the decoder have lost.
- We were conservative on choosing RGB ranges to predict the colors from received data stream. But for transition to a particular colour from different other colours it behaves differently. In order to handle the RGB boundary on each levels, we used ratio between red, green and blue bands which gave good results in most of the scenarios.

Attack-II: Bulb switching between different intensity levels

- Hamming Code : We were expecting to have a better result after adding hamming code for error correction. But our plan failed because we were receiving multiple bit errors.
- Our initial implementation did not had end of character packet. We were taking the Huffman binary equivalent of the whole sentence and dividing into multiple packets. Hence one packet can have bits from multiple characters. As we were decoding using the Huffman tree a single bit error in a character can affect the upcoming characters.