Alpaca Security Solutions

completely invisible

BLACKCHAT
INCOGNITO MESSAGING SYSTEM

Preliminary Design Review

Russell W. Crowe
Patrick D. Lloyd

Abstract

The BlackChat Incognito Messaging System consists of a small USB device with a built in wireless transceiver that can transfer encrypted text messages between computers. The user will interface with the device through a portable executable applet on their computer where they can enter their note, generate secret keys, and select various options that control the functionality of the device. A USB to UART bridge will be used to translate messages between the user's computer and the internal microcontroller which will be responsible for packet formation as well as encryption and decryption of the messages. A 433 MHz wireless transceiver will modulate and broadcast outgoing data as well as read and demodulate incoming messages

Table of Contents

Introduction

In many situations it is crucial to communicate information between several parties confidentially. With the recent exposure of warrantless government wiretapping and the wide availability of simple tools to monitor and capture information, protecting one's privacy is now more difficult than ever. The purpose of BlackChat is to create a secure, discreet, and plausibly deniable point-to-point communication system for text between two or more computers.

BlackChat uses two common practices to guarantee secure communication: AES-128 encryption using a pre-shared key (PSK) as well as “security through obscurity” by broadcasting through an uncommon frequency. When sending data through the Internet, even on secure channels like SLL and TLS, transaction logs and data monitoring is almost guaranteed. Software solutions like Wickr and Whispr offer very secure point-to-point messaging services but still rely on these traditional standardized infrastructure channels.

Figure 1: The Advanced Encryption Standard with 128-bit cryptographic block size (AES-128) is a proven, highly secure method used all over the world to keep your data safe [Credit: Wired Magazine]

System Overview

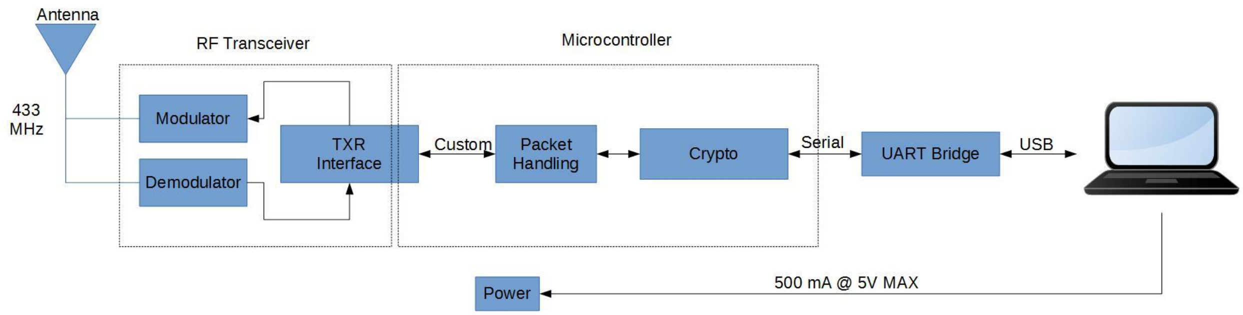


Figure 2: System overview

The BlackChat messaging system will consist of three main blocks: digital, RF, and power. A TI MSP430 microcontroller will handle communication between the laptop and the transceiver as well as encrypt and decrypt packets. The RF block modulates and demodulates digital information to and from a 433MHz carrier frequency. The entire system will glean power from the host laptop's USB 5V rail. 3.3 V components will use power from a buck regulator.

Microcontroller Selection

BlackChat will use an MSP430 from the F2x/4x family, due to program space and peripheral requirements.

AES-128 Encryption

BlackChat features AES-128 encryption to lessen the possibility of sniffing and deciphering the transmitted data. Users will select a pre-shared key to be used with each session and will have the ability to generate new ones for different conversations. The encryption will be implemented entirely in the microcontroller firmware.

Portable Linux Interface

The user will interact with BlackChat through a small Linux program. The user will be able to set a pre-shared key for the device to operate on, send and receive messages, and customize some options of the transmission.

Discrete RF Transceiver

Wireless data transmission will be implemented using a form of amplitude modulation (AM) called on-off keying (OOK). The high bits will be transmitted as a 433MHz sine wave and the low bits will be represented simply as a lack of a carrier signal. An example of OOK modulation can be seen below.

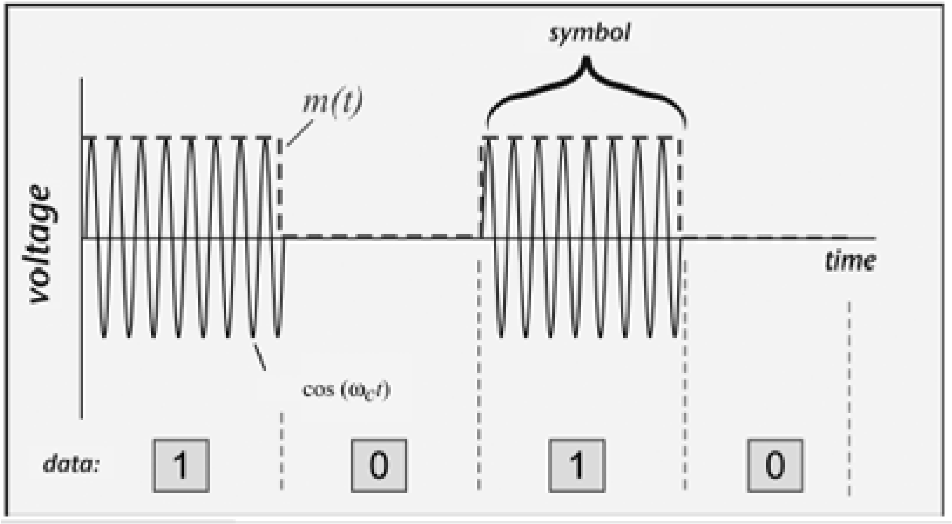


Figure 3: An example of ON-OFF keying amplitude modulation [Credit: EE Times]

Power Consumption

Since BlackChat runs completely off of USB power, it has a hard limit of 500 mA at 5 V operating power. This requirement has driven the power regulation choices, as well as the microcontroller selection. From Table 1, it can be seen that the operating power of the device is projected to be around 200 mA at maximum operating power.

Table 1: Approximate Power Consumption by Part

Device	Maximum Power Consumption
MSP430F2x/4x @ 16MHz	6.5 mA @ 3.3 V
TUSB3410	100 mA @ 5 V
RF Transmitter	50 mA @ 5 V
RF Receiver	50 mA @ 5V

Hardware

BlackChat has two primary hardware blocks to perform its functions: the combined digital hardware and power regulation block and the RF communication block. The digital block is used to communicate between the laptop and the microcontroller as well as form packets to be sent to the transceiver.

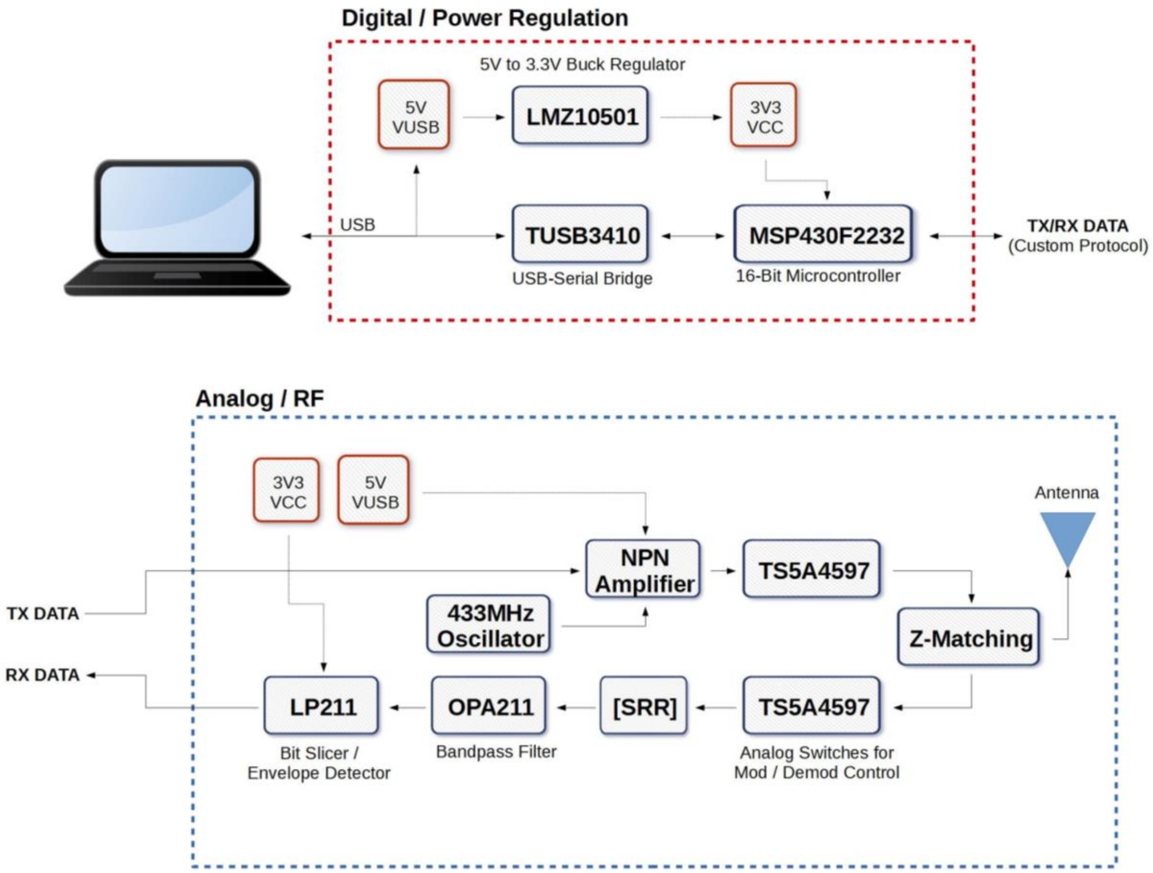
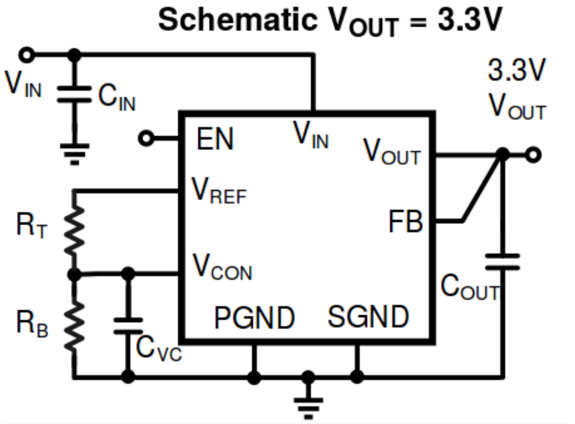


Figure 4: Flow chart of hardware system blocks

Power Regulation

Power regulation will be handled by the highly efficient (up to 95%) LMZ10501 switch mode buck regulator. This will provide a 3.3 V rail for the microcontroller and analog components that will interface with it. The module is mostly self-contained and only requires three external capacitors and two resistors as seen in the figure below.



C_{IN}	10 µF	≥ 6.3V	0805	X7R or X5R
C_{OUT}	10 µF	≥ 6.3V	0805	X7R or X5R
C_{VC}	470 pF	≥ 6.3V	0603	X7R or X5R
R_T	118 kΩ	1%	0603	
R_B	150 kΩ	1%	0603	

Figure 5: Application schematic from LMZ10501 at 3.3 V operation

Digital

There are two hard requirements for the selection of a microcontroller. One is that it must have two internal UARTs, since one will be used to communicate with the TUSB3410, and another will be used to serially communicate with the transceiver. The other requirement is flash memory size, or program space. In addition to firmware operations for handling data protocol, the device will use AES-128 encryption, which requires 4300 bytes of flash. Therefore, it is critical that the microcontroller have well over 4300 bytes of flash memory available to successfully implement data transmission protocols.

Another criteria for microcontroller selection is power consumption, since this device will run on USB power, which has a hard limit of 500 mA at 5 V. This made a microcontroller from the MSP430 line an obvious choice. Due to size constraints, minimal GPIO pins is optimal. There are several devices from the MSP430F2x/4x series fit these criteria, so the microcontroller will likely be chosen from that group.

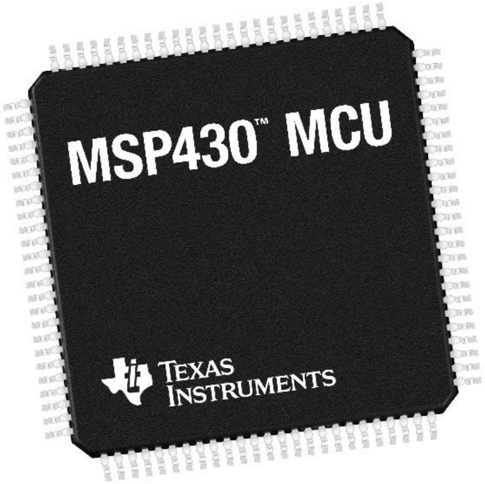


Figure 6: The MSP430 balances low-power operation, GPIO access, and processing performance [Credit: Texas Instruments]

RF Transceiver

The 433 MHz transceiver consists of the modulator and demodulator circuits. Both will exist on the same board and two 8-ohm SPST switches will be used to switch the antenna between each module.

Modulator

The modulator interprets digitally encoded data as 3.3V square waves from the microcontroller and mixes the signal with the high frequency sine wave from a 433 MHz oscillator. That signal is then used as a control signal for the base pin of an NPN-based RF amplifier. The antenna is a simple 5cm loop design that can exist either as a wire attached to the board or as a PCB microstrip. An example schematic can be seen below.

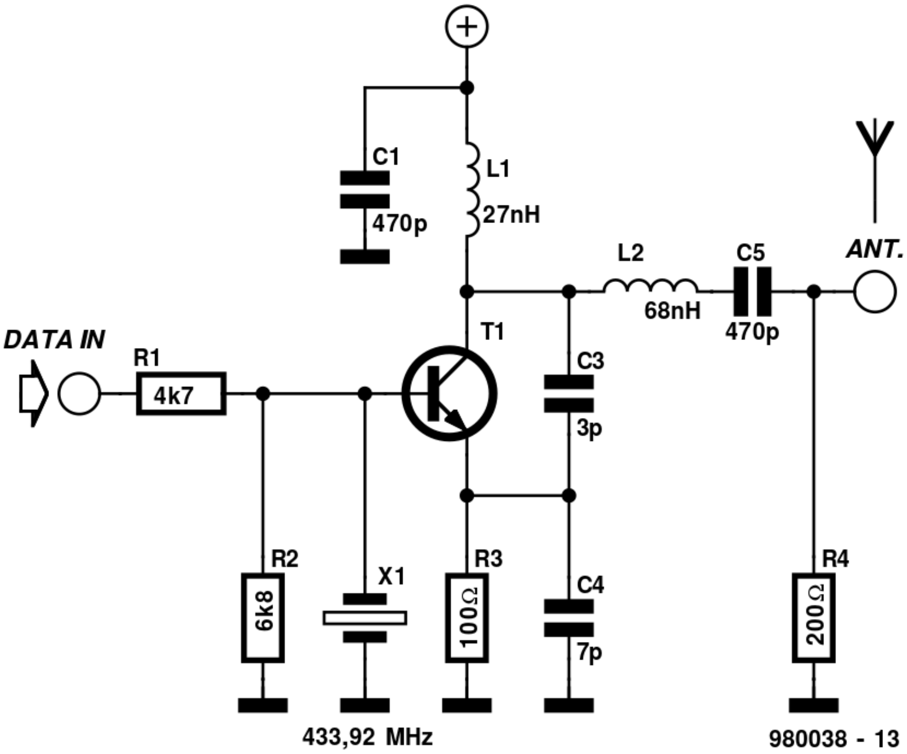


Figure 7: Example modulator circuit implementing OOK with 433.92 MHz oscillator [Credit: Elektor Electronics]

Demodulator

The demodulator performs all the steps necessary to interpret the incoming sinusoid and recover the data sent by the transmitter. An example schematic of the receiver can be seen below.

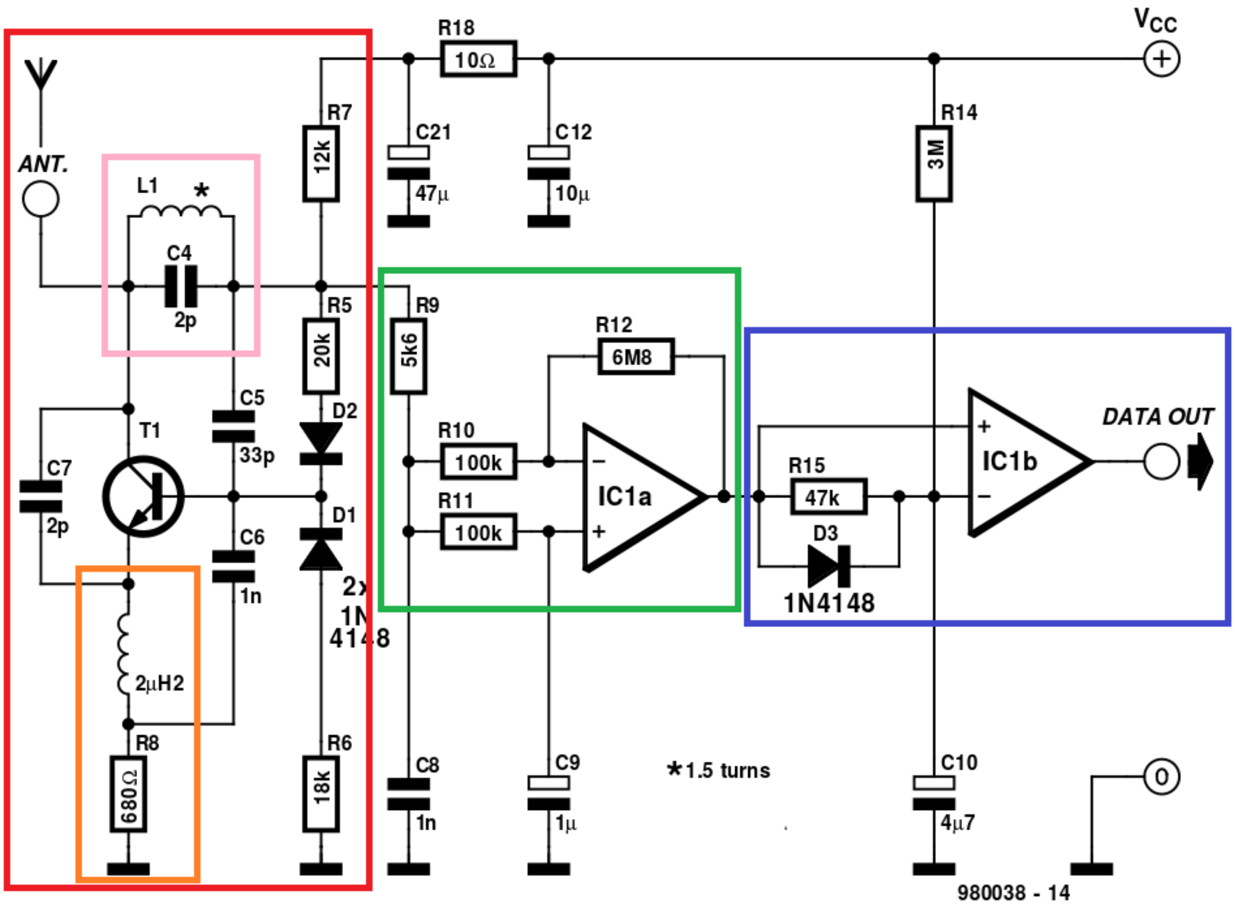


Figure 8: Demodulator schematic with highlighted sub-circuits

BlackChat will use a receiver topology known as a super-regenerative receiver (SRR) which is highlighted in red above. The SRR is a self-oscillating circuit that acts as an automatic gain controller (AGC) to recover an incoming signal. L1 and C2 (pink box), also known as a tank circuit, determine the center frequency of the receiver. Noise energy is picked up on the antenna and used to force the tank into oscillation. If only noise is being received from the antenna, the AGC can produce gains of up to 1,000,000, but immediately drops when legitimate signals are measured. The NPN transistor configuration functionally adds a negative resistance term to the system's transfer function in order to provide positive feedback and force instability. The orange box is known as a "quench" circuit and will shut down the SRR after each sample. The interval at which the SRR is quenched is synonymous with the sample rate of the input data.

The green box above is an active band-pass filter. It removes high frequency noise and harmonics picked up on the antenna as well as the spectral components added by the quench circuit. The filtered signal is then sent into a special comparator setup known as a bit-slicer, seen above in blue. The comparator's threshold voltage is frequency dependent so if amplified high-frequency noise is presented to it, C10 charges and raises the threshold. If a relatively low-speed data message is sent to it, the threshold is lowered and can more accurately produce digital waveforms to be read by the microcontroller.

Software

At the top level, this device has two software systems that will communicate with each other: the microcontroller code, from here on referred to as the firmware, and the code that will exist on the user's computer, from here on referred to as the user client. The operation of these two units will be described below.

Transmitted data is broken up on three levels. A 140 character unit of text sent by the user is referred to as a Wark, after the sound an alpaca makes in the wild. Warks are split up into 14-character, or 112-bit, chunks of information. An 8-bit header and an 8-bit cyclic redundancy check is added, bringing the total to 128 bits. This 128 bit chunk is then set for AES-128 encryption. These 128-bit crypto blocks are called messages. Messages are further broken up into 8 bit packets, to fit the standard UART data transfer size.

Data routing is as follows: user enters a 140 character Wark in the user client. This Wark is sent to the firmware, where it is used to generate ten 128-bit messages. Each of these messages is then split up into sixteen 8-bit packets, and then through the UART, is sent to the transceiver to be transmitted and then received by another device. On the receiving end, the packets are reassembled into 128-bit messages, which are in turn reassembled into the 140 character Wark and sent to the user client.

Firmware

MSP430 firmware is the link between the Linux terminal and the transceiver. Data transfers between the Linux serial port and the MSP430 will be carried out with the TUSB3410 UART to USB converter. This interface will use one of the UART peripherals within the MSP430. The MSP430 will also handle communication with the transceiver using the second UART peripheral within the MSP430. Data moving through the TX path will go through header formation, encryption, packetization, and serial transmission to the transceiver. Data moving in the RX path will undergo serial reception from the transceiver, decryption, parsing, error checking, and serial transmission to the Linux serial port.

The firmware consists of three main routines: an initialization routine, a receive routine, and a transmit routine. The initialization routine sets up the UARTs in the configuration described above and establishes the AES-128 bit encryption key that all messages sent and received during the current session will be based on. When the microcontroller boots up, it initializes the UARTs and then waits for the user client to send the encryption key. After receiving the key, the firmware sends an acknowledgement to the user client, signifying that the firmware is ready to receive messages, either from the transceiver side, or from the user client side.

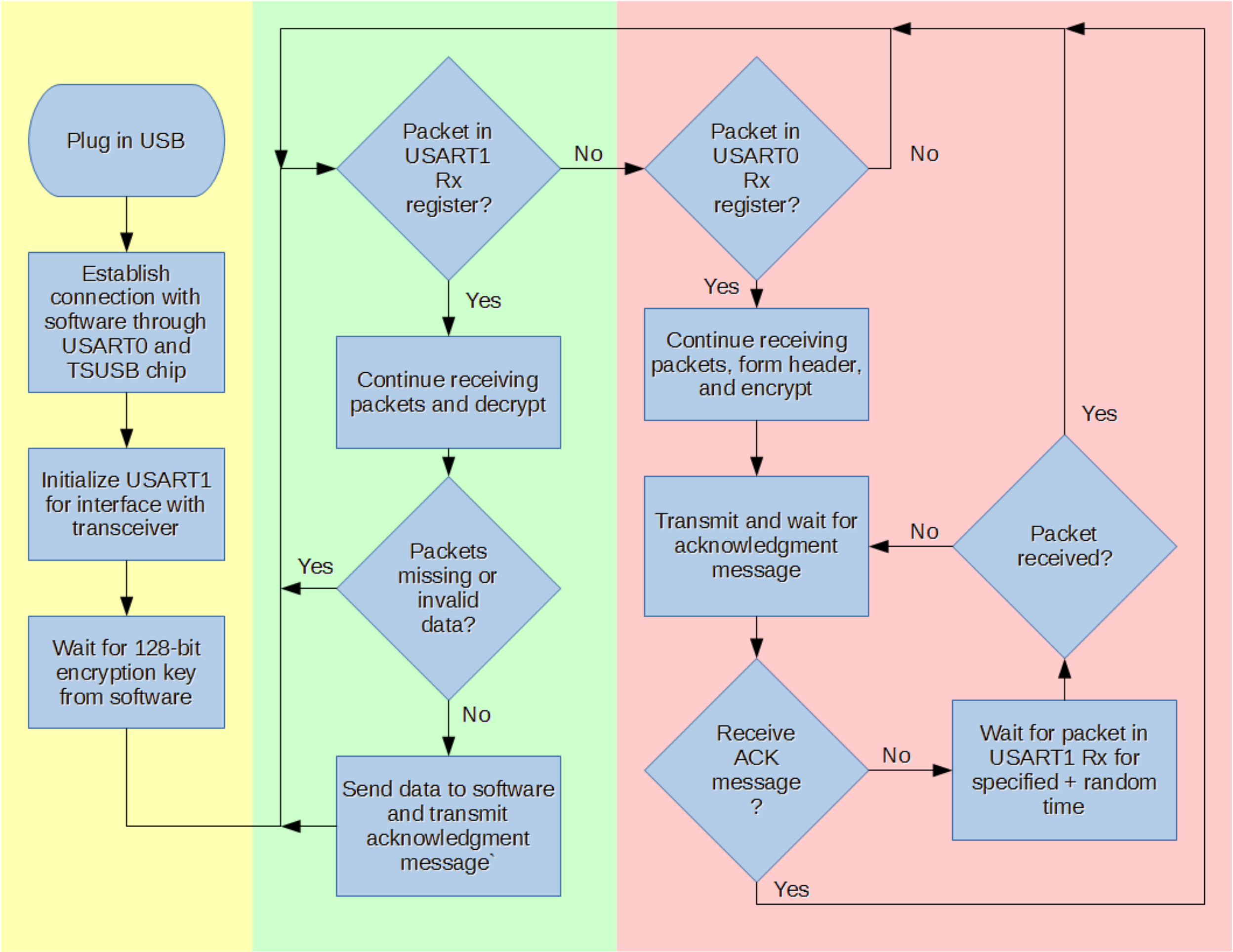

Plug in USB
Establish connection with software through USART0 and TSUSB chip
Initialize USART1 for interface with transceiver
Wait for 128-bit encryption key from software
Packet in USART1 Rx register?
No
Yes
Continue receiving packets and decrypt
Packets missing or invalid data?
Yes
No
Send data to software and transmit acknowledgment message`
Packet in USART0 Rx register?
No
Yes
Continue receiving packets, form header, and encrypt
Transmit and wait for acknowledgment message
No
Packet received?
Yes
Receive ACK message ?
No
Yes
Wait for packet in USART1 Rx for specified + random time

Figure 9: Firmware flow chart highlighting initialization (yellow), receive (green), and transmit (red) routines

After the encryption key has been established, the firmware drops into the receive routine. This routine is responsible for receiving and interpreting messages from the transceiver. This is the default state of the firmware. The firmware will poll the packet received flag for the UART, and store packets as they are received until a full message has been reached. When packets are being received, a busy code is sent to the user client by the firmware so that no Warks are sent to the firmware while it is busy receiving a message.

A full message requires 16 packets. If any less than 16 packets are received, the message is considered corrupted, and the firmware will wait for the sender to resend the message. If all 16 packets are properly received, the 128-bit piece of data will be decrypted, and the data will be checked against the 8-bit cyclic redundancy check. If this check fails, the message is thrown out as invalid, and the firmware will again wait for the sender to resend the message. Upon the firmware receiving a full message which properly checks out with the cyclic redundancy check, an acknowledgement message will be sent, so that the sender knows the message has been properly received and interpreted. Then the sender can send the next message in the queue, or move into receive mode. Once the firmware receives and generates a full

Wark, it is sent to the software to be read by the user, and a ready code is sent to the user client so that the user can then send a Wark.

The third firmware routine is the transmit routine. This is an interrupt driven routine that is triggered when the user sends a Wark. Once the Wark has been transferred to the firmware through the TUSB UART, it is split up into messages, encrypted, and then split again into packets to be transmitted through the UART to the transceiver. Once a 16-packet, or 128-bit message has been sent, the firmware waits for an acknowledgement message from the receiver. If this acknowledgement is not received within a set amount of time, the firmware will attempt to send the message again until an acknowledgement is received. Once the message is successfully sent (determined by a successful acknowledgement), the next message in the Wark is sent, until the entire Wark transmission is complete. The firmware will then move back into its default state of listening for incoming messages, the receive state.

In the case where users send messages at different times, there is no issue with communication since the firmware defaults to a listen state, only leaving that state when a message is to be sent. One user sends a message, and the other is already waiting for the message, so there is no collision. However, if both users send a message at the same time, neither will be in a receive state and they will both try to resend their message until the device is reset. The method that will be used to resolve this conflict is to listen for messages while waiting for acknowledgements, and have different wait times for each device.

For example, client 1 and client 2 send packets at time t = 0. Client 1 is set to wait 100 ms before trying to resend, and client 2 is set to wait 200 ms before trying to resend. The initial packets will be dropped, but while client 2 is still waiting, client 1 will resend its packet, and client 2 will drop into receive mode to complete the message transfer, storing its message in a buffer to be sent after completion.

User Client

The user client will be the link between the user and the firmware. Since all communications happen through the terminal, and not much processing of data happens on the user client side, the user client is essentially a serial terminal interpreter. In other words, it takes message data and formatting/control codes and converts it into something more readable and useable.

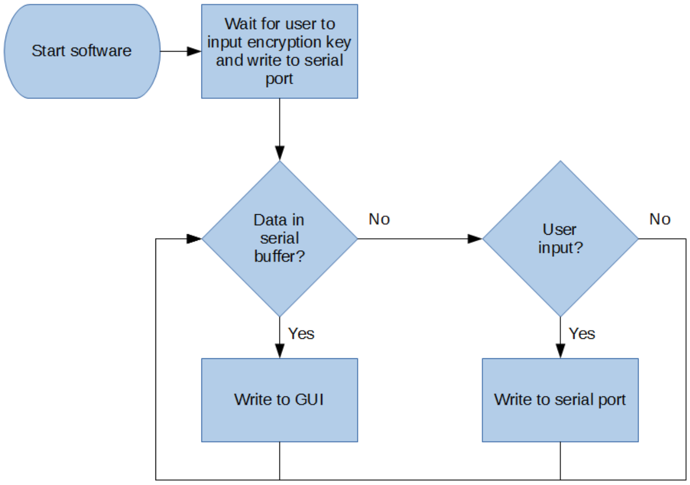


Figure 10: User client flowchart

The user client is organized in a similar manner to the firmware. At run-time, there is a setup routine and the user must input an encryption key and a client id number. These two pieces of information must be agreed upon beforehand by the two users that will be communicating with each other. The encryption key will be user supplied or generated from the user client. The client id number will be unique for each user and will relate to message priority.

After the encryption key and client id numbers have been established and communicated to the firmware, the user client will then listen to the serial port for incoming Warks. When a Wark is received, it will be displayed to the user. The user client will also interpret control codes sent by the firmware to signify when data is about to be sent, or when the firmware is ready to receive data.

When the user sends a Wark, it will be loaded into an outgoing Wark buffer and sent to the firmware as soon as the firmware indicates the “ready to send” state. Only one Wark may be handled by the firmware at once, so each Wark will be unloaded as the processor becomes available.

Simulation and Testing

Several parts of the transceiver are being simulated in order to define and better tune system parameters. Below is the fully simulated modulator where an equivalent LCR model is substituted for the 433 MHz crystal oscillator.

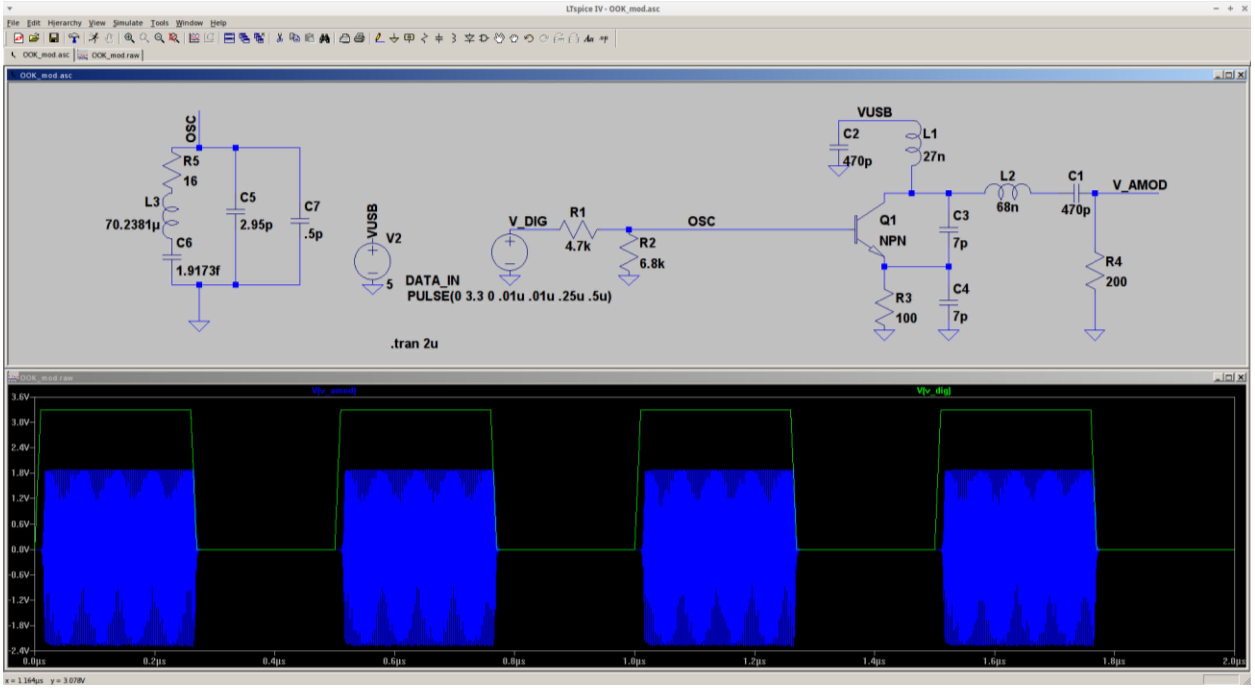
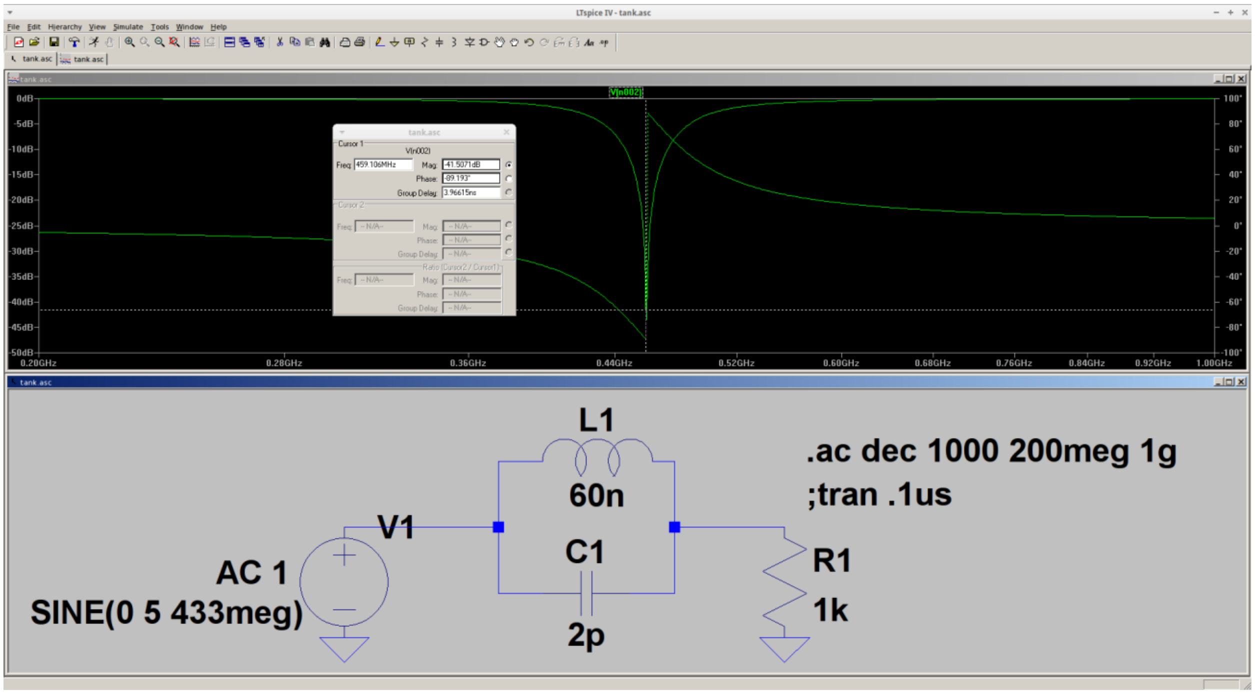


Figure 11: Circuit simulation demonstrating OOK at 433MHz with a crystal oscillator equivalent

The LTSpice solver has a difficult time simulating highly resonant (a.k.a. high-Q) circuits and if left to its own devices, would take days to reach a steady state oscillation. With this in mind, the receiver is much more difficult to simulate. The super-regenerative receiver is, by design, highly resonant and so parts were simulated individually instead of as a whole system. Below is the LC tank circuit for the SRR that determines the resonant frequency.



tank.asc

Cursor 1

V(n002)

Freq: 459.106MHz Mag: -41.5071dB ●

Phase: -89.193° ○

Group Delay: 3.96615ns ○

Cursor 2

Freq: -- N/A-- Mag: -- N/A-- ○

Phase: -- N/A-- ○

Group Delay: -- N/A-- ○

Ratio (Cursor2 / Cursor1)

Freq: -- N/A-- Mag: -- N/A--

Phase: -- N/A--

Group Delay: -- N/A--

Figure 12: AC Analysis of tuned LC tank circuit

In addition to simulation, the system will need to be tested and validated. Since many of the oscilloscopes available for use are bandwidth limited to 200 MHz or less, RF transmission will be measured using an RTL2832U USB dongle and GNU Radio. Below is a test of a GNU Radio test bench measuring the spectrum of a 315 MHz wireless doorbell transmitter.

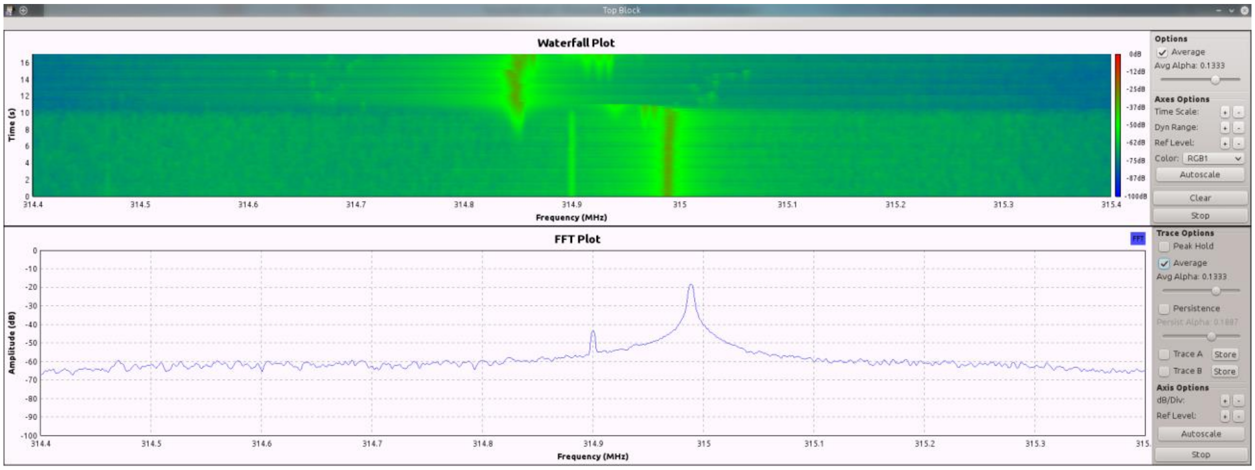


Figure 13: GNU Radio waterfall plot (top) and FFT (bottom) of 315 MHz wireless doorbell transmitter

Project Management

In order to manage the BlackChat project effectively, steps need to be taken to distribute tasks to team members, budget time productively, and prioritize subsystem development and features.

Distribution of Labor

Tasks will primarily be accomplished in tandem but a distribution of labor was developed to guarantee project accountability. Individual subsystem responsibilities can be seen in the table below.

Table 2: BlackChat distribution of labor

Task	Russell Crowe	Patrick Lloyd
PDR	•	•
Transmitter	•	
Antenna		•
Receiver		•
Firmware	•	
uC-TRX Interface		•
Client Interface	•	
uC-Laptop Interface	•	
Final Board Design	•	•
Validation and Testing	•	•
Stretch Goals	•	•

Timeline

In order to stay on track, an aggressive timeline was developed to encourage development. Ideally the project designs will be finalized and built near early November in order to complete final validation and testing. The hardware designs can be finalized and manufactured fairly early and once proven functional, the rest of the semester can be spent developing software features. The Gantt chart in Table 3 below shows the projected timeline for BlackChat development.

Table 3: Gantt chart timeline

Week	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16
Date	1-Sep-14	8-Sep-14	15-Sep-14	22-Sep-14	29-Sep-14	6-Oct-14	13-Oct-14	20-Oct-14	27-Oct-14	3-Nov-14	10-Nov-14	17-Nov-14	24-Nov-14	1-Dec-14	8-Dec-14	15-Dec-14
PDR	🟦	🟦	🟦	🟦												
Transmitter				🟦	🟦	🟦										
Antenna				🟦	🟦	🟦										
Receiver					🟦	🟦	🟦	🟦								
Firmware							🟦	🟦	🟦							
uC TRX Interface							🟦	🟦	🟦							
Laptop Software								🟦	🟦	🟦						
uC Laptop Interface								🟦	🟦	🟦						
Final Board Design								🟦	🟦							
Testing & Validation									🟦	🟦	🟦	🟦				
Stretch Goals													🟦	🟦	🟦	🟦

House of Quality

During development it is crucial to pick the most important features to implement in the event of unforeseen complications. If, for instance, the final hardware design fails catastrophically and time becomes short, certain features will have to be cut while still maintaining core functionality.

Table 4: House of Quality

Feature	Ranking(3 highest - 0 lowest)
Wireless transmission of digital data	3
Linux user client	3
Encryption	2
Data transfer rate	1
Small form factor	1
Two-way communication	3
More than two-way communication	0
Windows user client	0
Mixed signal board	0
Error checking and mitigation	2
Communication range	1

Features that are absolutely crucial to what is considered a functional product are defined by a ranking of '3'. Rankings of '2' are high priority but in worst case scenarios will be cut. '1' is a non-crucial feature that the board will be designed with consideration to and '0' is a stretch goal if all other features are built and functioning optimally.

Conclusion

BlackChat will allow wireless transmission of encrypted data over the 433MHz ISM band. It incorporates an easy to use Linux interface and a USB dongle. Two users can plug in their USB dongles, start up the software on their computers, enter a previously agreed upon encryption key, and start communicating. Users may send a maximum of 140 characters per message, or Wark. BlackChat allows seamless communication between two clients while protecting data from prying eyes.