Samir Farhat Dominguez - safarhat@bu.edu
Harshit Agrawal - harshit@bu.edu
Gianna Iafrate - giafrate@bu.edu
05/04/2022
EC544: Final Report

Secure Wireless Communication over RF for Data Acquisition

Github: https://github.com/SamirFarhat17/RF-data-secure-network-comms
YouTube Demo: https://youtu.be/HflZvpvrr3E

I. Introduction

This project aims to incorporate various layers for secure wireless communication over RF. The use of wireless radio links is advantageous when communication is required in areas where fixed bus networks cannot be used. In particular, free RF bands 314MHz, 433MHz, 866MHz, and 915MHz are often utilized by most commonly used home appliance devices such as various types of remote controls, sensors, and smart-home solutions. The data transmission is always one-way, from sensor probe with a transmitter to receiver. Such a data transfer is the free bandwidth, and it is not an issue to acquire and read sent data. We say a communication exchange is secure when two entities are communicating and a third party is not able to listen in. To accomplish this, they need to communicate in a way that is not susceptible to eavesdropping or interception. Possible attacks on such a data transfer are trivial, and data can be easily captured, modified, or resent. It is not possible to verify the origin of the data on the receiver node.

While standard secrecy methods, such as cryptography, protect the contents of the message from being accessed by unauthorized users, covert communication conceals the existence of the communication to prevent unauthorized users from detecting the communication. Secure communication includes means by which people can share information with varying degrees of certainty that third parties cannot intercept what was said and must be secure against replay attack.

This secure communication method is implemented in 4 main steps. First, the user interface connects with an API to receive a message and encrypt it. This encrypted message and ciphertext is then modulated and transmitted using bladeRF. The signal is then received and demodulated by the hackRF. Lastly, this ciphertext is decrypted to get the original message. In this process, the usage of authentication and timestamps are utilized to address the possibility of eavesdropping and ensure secure communication.

II. Project Overview

This project is about secure wireless communication over RF. The main advantage of this project is that the data cannot be received unless you have a receiver code that is compatible with the transmitter. The mode of communication that we have used in our project is a radio frequency channel. In our project, we have used **two layers of data encryption and decryption**. The first is PGP-key-based cryptographic encryption of the message and the second is securely modulating the signal. The RF modules that we have used for data transmission and reception work at 433MHZ.

To meet the present-day technology needs, data transfers at higher speeds are to be achieved which is possible by RF communication.

Before transmitting the data, the data was converted into an unreadable form (ciphertext) to be sent. At the receiving end, the reverse of encryption carries on to get back the original message. Thus the data will be protected in every way by following the encryption and decryption standard formats. Wireless makes this project more flexible. Standard algorithms require software to be installed into the system before actually using them and hardwired connections.

At the receiver end, the RF receiver receives the data from the air. The received data is sent to the processing module written in C to filter the repeated and ambiguous bits and filter ciphertext with delimiters, finally, cryptography modules decrypt the ciphertext and convert it into a form suitable for the user to read. This decrypted data can be seen on the receiver PC. Thus, the data is protected while it is transmitted and received between two PCs.

III. Resources

The below table is a comprehensive list that encompasses all the resources utilized for the final product. For the API and cryptography layer, personal laptops and software are the only resources. The transmission layer utilizes software as well as the HackRF and BladeRF for transmitting and receiving the signal.

Hardware	Software
Personal Laptops HackRF - Power Supply - ANT500 Antenna BladeRF - Power Supply - Tri-Band Antenna	GNU Radio 3.7+ - grc flowgraph - ZeroMQ Python - Python 3.9 - pip 22.03 - Libraries - Pseudo random function generators C++/C - gcc 9.3.0 - GNU Make 4.2.1 - Libraries - Pseudo random function generators OpenSSL GNUPG Github - Code repository and version control Windows subsystem for Linux Middleware: Twitter API - Credentials set - Account

IV. Risk Management

Below is the list of identified potential risk areas in the design. With each item is the mitigation that was implemented in order to avoid project setbacks.

Risk: Learning curve for eliminating noise from the transmission and reception may be steep
Mitigation: Dedicated one team member to learn and test simple transmission and reception techniques from the start. Implementation was left flexible enough to allow changes once the transmission abilities had been identified.

Risk: GNU-Radio not performing responsively.
Mitigation: To ensure proper performance, this was transitioned to running GNU-Radio in a well-resourced Linux environment machine on a personal laptop.

Risk: Connecting all components with the initial setup not working as expected
Mitigation: Tested this connectivity early on so any changes could be addressed quickly and easily.

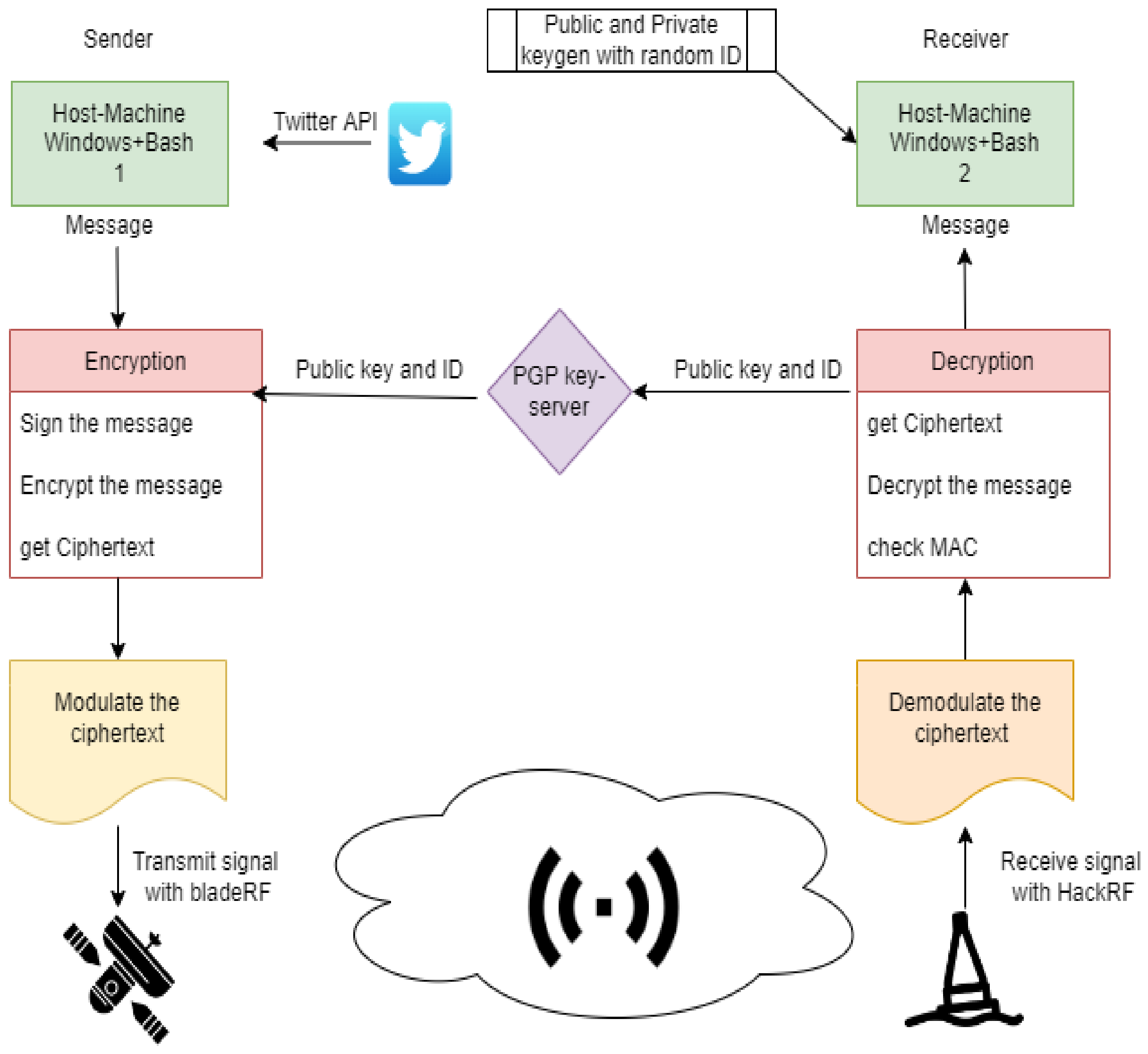
Risk: The learning curve for implementing secure cryptographic encryption with suitable signature and timestamping may be steep and we may underestimate the amount of work required to implement it.
Mitigation: Dedicated one team member to learn and investigate the Cryptographic algorithm and secure MAC from the start. This left the implementation flexible enough to switch to a different encryption algorithm if it proves more complicated than expected.

Risk: Hardware failure for either the BladeRF or HackRF.
Mitigation: While the RF board breaking down may have been somewhat catastrophic, we planned to be able to redirect the transport of encoded RF through traditional networking as a backup.

Risk: Testing and troubleshooting phase takes longer than anticipated, and project cannot come to fruition.
Mitigation: Ensured the team completed the weekly tasks through check-ins and updating the schedule realistically when applicable. Connected modules in a timely manner so there was enough time for troubleshooting.

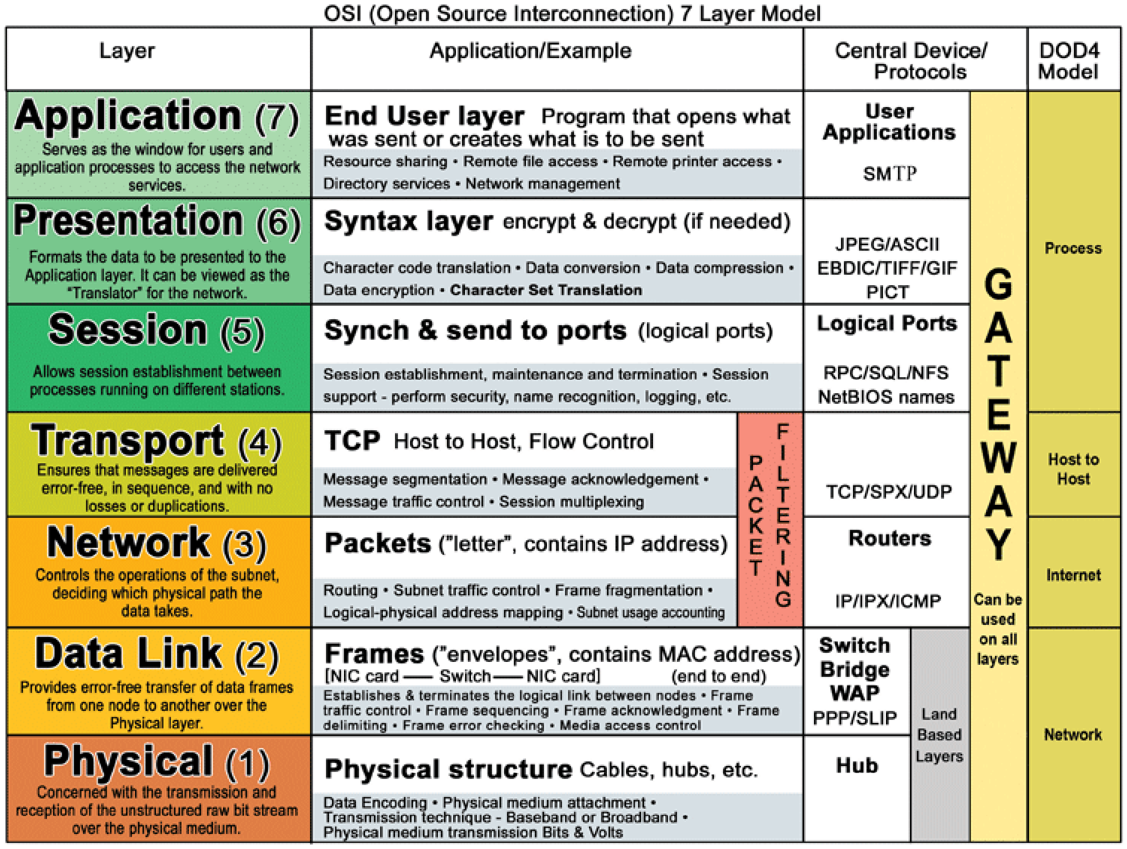
V. System Overview

As seen from the system diagram, this project incorporates many aspects learned throughout the modules of this course. The first module is API, in which the Twitter API is used to grab a certain message to encrypt. The second module layer is encryption/cryptography, which is the method this project uses to sign the message and get the ciphertext. The final layer is networking/connectivity, where the bladeRF connects and transmits a signal to be received. All of these layers also include an aspect of authentication, where the user must identify oneself in order to hinder an eavesdropper from getting the message.



The project can be divided into 3 major modules: The application layer where the user interacts, Cryptographic layer where encryption and signing take place, Communication layer where the ciphertext is modulated and transmitted via RF. These modules can come together to form a pseudo-OSI-7-layer model for networking.

OSI (Open Source Interconnection) 7 Layer Model



Layer	Application/Example		Central Device/ Protocols			DOD4 Model
Application (7) Serves as the window for users and application processes to access the network services.	**End User layer** Program that opens what was sent or creates what is to be sent Resource sharing • Remote file access • Remote printer access • Directory services • Network management		**User Applications** SMTP		**GATEWAY** Can be used on all layers	Process
Presentation (6) Formats the data to be presented to the Application layer. It can be viewed as the "Translator" for the network.	**Syntax layer** encrypt & decrypt (if needed) Character code translation • Data conversion • Data compression • Data encryption • **Character Set Translation**		JPEG/ASCII EBDIC/TIFF/GIF PICT			
Session (5) Allows session establishment between processes running on different stations.	**Synch & send to ports** (logical ports) Session establishment, maintenance and termination • Session support - perform security, name recognition, logging, etc.		**Logical Ports** RPC/SQL/NFS NetBIOS names			
Transport (4) Ensures that messages are delivered error-free, in sequence, and with no losses or duplications.	**TCP** Host to Host, Flow Control Message segmentation • Message acknowledgement • Message traffic control • Session multiplexing	PACKET FILTERING	TCP/SPX/UDP			Host to Host
Network (3) Controls the operations of the subnet, deciding which physical path the data takes.	**Packets** ("letter", contains IP address) Routing • Subnet traffic control • Frame fragmentation • Logical-physical address mapping • Subnet usage accounting		**Routers** IP/IPX/ICMP			Internet
Data Link (2) Provides error-free transfer of data frames from one node to another over the Physical layer.	**Frames** ("envelopes", contains MAC address) [NIC card —— Switch —— NIC card] (end to end) Establishes & terminates the logical link between nodes • Frame traffic control • Frame sequencing • Frame acknowledgment • Frame delimiting • Frame error checking • Media access control		**Switch** **Bridge** **WAP** PPP/SLIP	Land Based Layers		Network
Physical (1) Concerned with the transmission and reception of the unstructured raw bit stream over the physical medium.	**Physical structure** Cables, hubs, etc. Data Encoding • Physical medium attachment • Transmission technique - Baseband or Broadband • Physical medium transmission Bits & Volts		**Hub**			

VI. Technical Discussion

Application Layer: User Interface

The first step in the above process is to get the message that will be sent to the receiver. The user interface will utilize an Application Programming Interface, or API, to gather the message to send. An API acts as an interface between two parties in order to transfer information. In the case of this task, where the Twitter API is used, the two parties that the API acts as the interface between are the user and the Twitter application. APIs function through the use of keys and tokens. There are a few different tokens that one can use for an API, including the consumer key and token. The consumer key and token are private entities that allow identification of the user who is requesting access. This key and token pair is how authentication happens within the interface and allows for successful gathering of the messages. The key and token pair is necessary because, without it, anyone can access any information through this interface. An additional layer of authentication is taken through a bearer token or access token, in which another party has the ability to use the interface to gather information. With the bearer token, anyone that has this token can acquire access, without the need of a key. This type of key is beneficial for access that does not need to be extremely secure. Regardless of the key and token pair, a certain key is required to use an API and gather information.

Twitter API in particular has many methods and layers of authentication that one can implement. As stated above, there is the ability to create a consumer key and token pair, an access token, and bearer token. There is also another layer of authentication that can be implemented in Twitter API through the client key, which allows for reading and writing of tweets on the account of which the API application was created. Aside from reading and writing the application user's tweets, Twitter API has the functionality to search recent tweets over the last 7 days. The input into this query URL can be a range of parameters, from a keyword to search for, to the number of results to pull. Along with the tweet itself, one can pull other information such as the author ID of the user that wrote the tweet, the date the tweet was published, or any attachments to the tweets.

This application layer including API is key to ensuring the user interface is working properly and sending the correct message. The user interface utilizes both Python and the terminal window to pull data from the Twitter API. Upon opening the terminal window and running the python script, the user must indicate how many tweets are desired to be pulled. Then, the user is asked to input a keyword that they wish to grab a message about. Finally, the user is asked to input their bearer token.

The input keyword and input number of tweets are put into the parameter query to grab the messages from the most recent tweets. In addition to this, the query asks to output the tweet text, the author ID, and the time the tweet was written at. Once the URL is created, the API interface will attempt to connect to the end point, which is the Twitter link to the most recent tweets. The end point connection is generated through the search URL of tweets, the query parameters, and the bearer token. For this reason, the bearer token is the most beneficial token to use for this project. For the purpose of this project, one must ensure that they have appropriate Twitter API access and can securely grab tweets in that way. Therefore, there is no specific token or key associated with this task; as long as the user trying to run this project has a Twitter API application set up, they will be able to process the message.

With a successful connection to the endpoint, the tweets are pulled and stored into a list on Python. Each item in the list consists of the following parameters: tweet ID, author ID, text, and date created. The program will start a text file in the format necessary to encrypt the message. For each line item, the program will write the author ID, date, and text of the tweet on each line of the text file, indicating when you are viewing the next tweet. Once all the necessary tweets have been pulled, the edit file is closed and the text file is saved in the working directory where it can be used for encryption.

Presentation and Session Layer: Cryptography

Cryptography is the science that creates strategies for utilizing complex mathematical transformations to transmit data through conveyance channels in a frame that no one but authorized persons can get.
Cryptography is another main building block of our work. At every stage of development, we will be considering vulnerabilities, such as caching IP mappings and monitoring Ping requests. Taking the cryptographic component, our priorities are to achieve the following between the two machines.

1. Authentication: This entails that at the very least the sender and receiver have the means to establish that they are engaging with the entity they think they are communicating with. Our work will specifically target message signatures to establish authenticity and will be further elaborated on later.
2. Confidentiality in the presence of a 3rd party: This goal entails ensuring that under a set of conditions, principally not having private key access, a third party listener with oracle access is unable to establish any part of the plaintext from its corresponding ciphertext.
3. Integrity in the presence of a 3rd party: This goal relates to establishing a protocol where a user is unable to intercept or manipulate the transmission of data between Alice and Bob.

There are three generalized types of cryptographic algorithms; hash functions, symmetric key encryption, and asymmetric key encryption. While symmmetric key encryption can be useful, in the spirit secretive decentralized communication relevant to military and guerilla unit applications we decided to proceed with assymetric.

Asymmetric cryptography, also known as public-key cryptography, is a process that uses a pair of keys, one public and one private. The public key can be used to encrypt while the private can encrypt and decrypt. So a user can generate a pair and if they only expose their public key they will be able to receive the ciphertexts that only they can decrypt. Asymmetric key cryptography incurs a larger overhead due to the inevitability of having to have higher complexity and key lengths for public keys. This means that generation, encryption, and decryption algorithms occur more frequently and are slower to execute. However, since our system will only involve infrequent massive dumps of data between two parties over radio-frequency, nothing excludes asymmetric encryption as a valid method. Moreover, this is more relevant to our use-case, as it simulates an insurgent that is cut off from their handlers but not networking as a whole.

Asymmetric key encryption allows us to achieve the 3 goals we have outlined. We utilized openssl and gnupgp as well as there related C and Python frameworks to implement our system. At a detailed level, the cryptographic flow proceeds as follows

Receiver

The receiver wishes to receive some sort of intel from social media but these sites have been blocked, as such they want someone somewhere where that social media hasn't been banned to be able to send it to them without it being visible to the enemy.

1. The receiver begins by generating a random username, key id and passphrase, which the opposing party is aware of by passing it all to a shared Alice file.
2. The receiver produces a private 2048-bit modulus RSA key . From this private key they generate a corresponding public key this is tied to their corresponding username and fingerprint.
3. The public key and its public parameters are sent over to a public keyserver
4. The receiver then begins to monitor the RF channel as per our GNU radio specification, populating an RX file with the messages encoded at a pre-agreed frequency.
   Sender fullfils their role…
5. With the use of threading, the monitoring stops once the RX file has been populated at least once(likely much more than that)
6. We run a program that takes these received bytes into 1 clear ciphertext.
7. We then decrypt this message and voila, we have our securely transferred plaintext.

Sender

The sender wishes to send important intelligence gathered from social media to their allies in enemy territory. They need to send this as a single message since they are aware their ally may only be able to monitor for messages infrequently for short periods of time.

1. The sender retrieves intelligence from the API as described in that subsection populating a text file with that.

Receiver fullfils their role…

2. The sender retrieves the receiver's public key from the keyserver with their ID.
3. The sennder encrypts the message generating a signed and encrypted PGP ciphertext message
4. The sender runs the script which formats the encrypted message such that it can be transmitted over RF
5. The sender begins transferring the encrypted message hoping their ally can get a hold of it.

Communication Layer: RF Signals

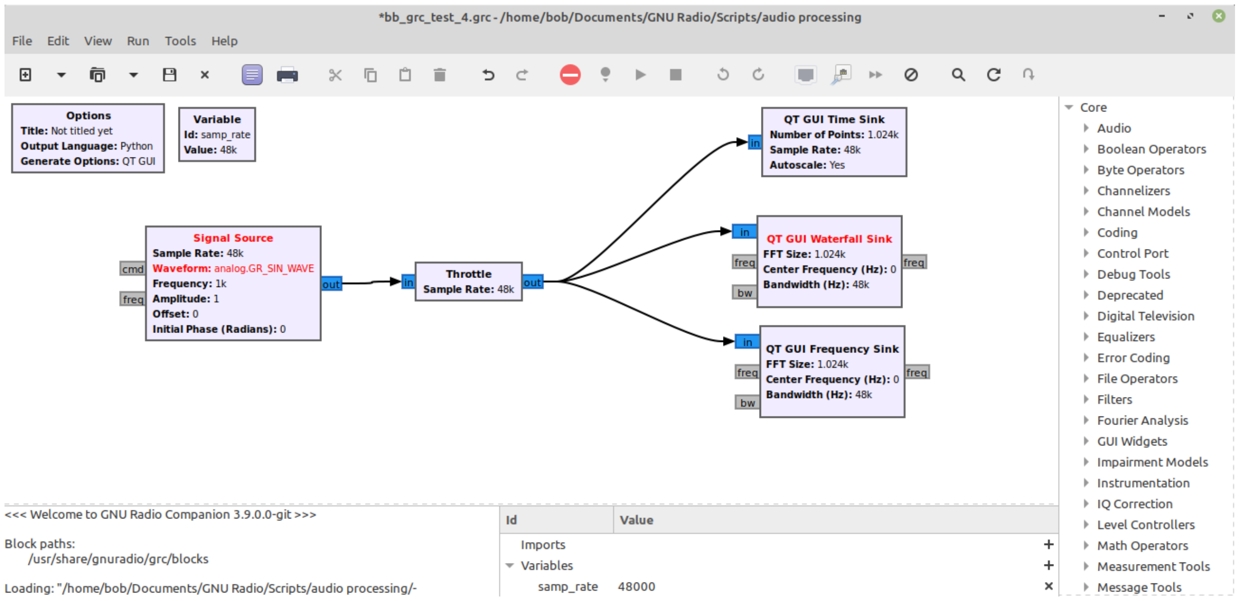
The secure communication layer is implemented into the RF communication system. Before the implementation itself, the requirements for the transmitter and receiver sides are as follows:

1. Transmitting side: When sending data, the ciphertext is different from the previous transmission for the same message to secure against replay attack.
2. Receiving side: Based on the known format of the received raw signal, it will be able to demodulate and decrypt with the correct key.

For transmission, we tested the system with HackRF and BladeRF 2.0 Micro, which is a wideband MIMO transceiver that covers 47MHz to 6GHz with up to 56 Mhz of bandwidth. It has 2x2 MIMO, the Host connection is via USB 3.0, and GNU Radio integration is provided via gr-osmosdr while another device we used as a receiver is RTL-sdr which is a receiver peripheral, operating from 1 MHz to 2.1 GHz, with a Hi-Speed USB 2.0 connection. It is bus-powered, portable, and has a maximum quadrature sample rate of 10 Msps. GNU Radio integration is provided via gr-osmosdr.

The fundamental issue with using SDR is that there's no uniform standard for transmitting frequency, modulation, and data protection, although some applications use signal encryption but aren't always getting a lot of interest.

We used GNU-radio, as it's an open-source software development toolkit and provides signal processing blocks to implement software radios. To process digital signals there are several stages (filtering, correction, analysis, detection...) as processing blocks, which we connected using simple flow-indicating arrows to different blocks in GNUradio, which is C++ and Python code in the backend.



Options
Title: Not titled yet
Output Language: Python
Generate Options: QT GUI

Variable
Id: samp_rate
Value: 48k

Signal Source
Sample Rate: 48k
Waveform: analog.GR_SIN_WAVE
Frequency: 1k
Amplitude: 1
Offset: 0
Initial Phase (Radians): 0

Throttle
Sample Rate: 48k

QT GUI Time Sink
Number of Points: 1.024k
Sample Rate: 48k
Autoscale: Yes

QT GUI Waterfall Sink
FFT Size: 1.024k
Center Frequency (Hz): 0
Bandwidth (Hz): 48k

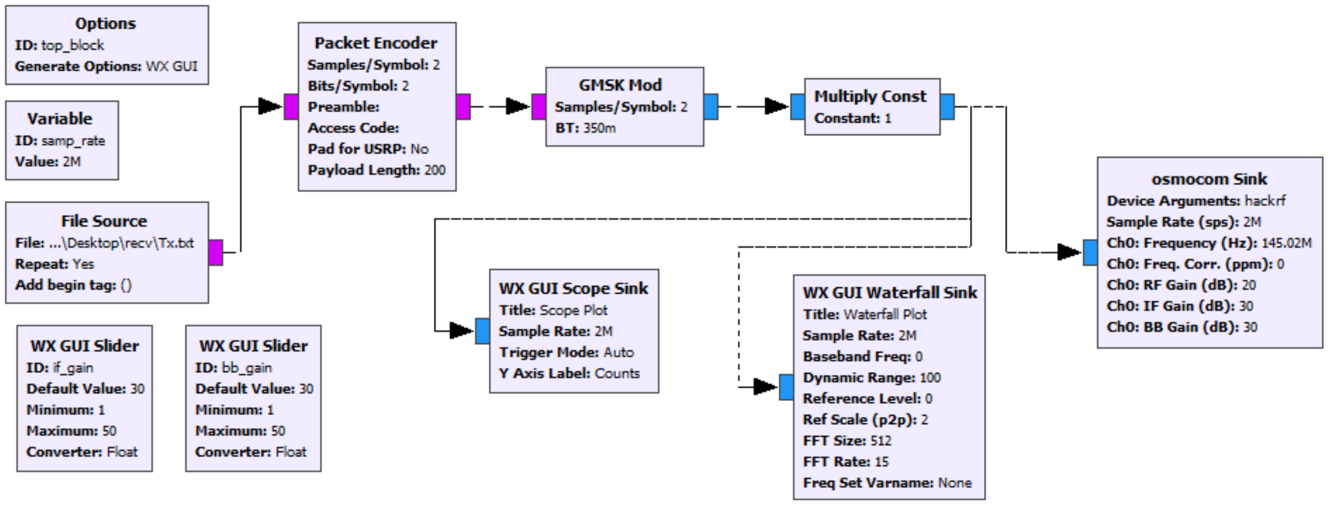
QT GUI Frequency Sink
FFT Size: 1.024k
Center Frequency (Hz): 0
Bandwidth (Hz): 48k

```
<<< Welcome to GNU Radio Companion 3.9.0.0-git >>>

Block paths:
    /usr/share/gnuradio/grc/blocks

Loading: "/home/bob/Documents/GNU Radio/Scripts/audio processing/-
```

Id	Value
Imports	
Variables	
samp_rate	48000

If the above method fails to transmit data with efficiency or there are any hardware failures (risk mitigation)... we had the backup to use (RF signals over IP), and not over the air with antennas. The use of IP-based transport eliminates the complexity of deploying actual radios and transmitting RF over the air. In particular, the ‘raw’ RF signals will be tunneled via ZeroMQ and can be received with traditional RF tools, as if they’re coming from an actual SDR. GNU-Radio supports ZeroMQ source blocks natively.

Transmission Flowgraph:



Options
ID: top_block
Generate Options: WX GUI

Variable
ID: samp_rate
Value: 2M

File Source
File: ...\Desktop\recv\Tx.txt
Repeat: Yes
Add begin tag: ()

WX GUI Slider
ID: if_gain
Default Value: 30
Minimum: 1
Maximum: 50
Converter: Float

WX GUI Slider
ID: bb_gain
Default Value: 30
Minimum: 1
Maximum: 50
Converter: Float

Packet Encoder
Samples/Symbol: 2
Bits/Symbol: 2
Preamble:
Access Code:
Pad for USRP: No
Payload Length: 200

GMSK Mod
Samples/Symbol: 2
BT: 350m

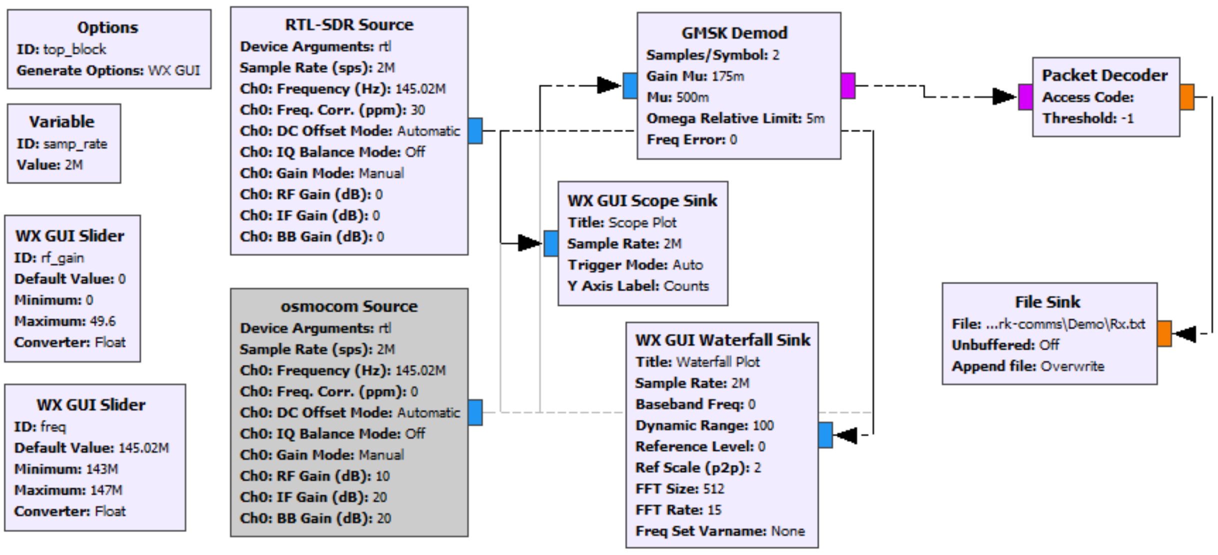
Multiply Const
Constant: 1

WX GUI Scope Sink
Title: Scope Plot
Sample Rate: 2M
Trigger Mode: Auto
Y Axis Label: Counts

WX GUI Waterfall Sink
Title: Waterfall Plot
Sample Rate: 2M
Baseband Freq: 0
Dynamic Range: 100
Reference Level: 0
Ref Scale (p2p): 2
FFT Size: 512
FFT Rate: 15
Freq Set Varname: None

osmocom Sink
Device Arguments: hackrf
Sample Rate (sps): 2M
Ch0: Frequency (Hz): 145.02M
Ch0: Freq. Corr. (ppm): 0
Ch0: RF Gain (dB): 20
Ch0: IF Gain (dB): 30
Ch0: BB Gain (dB): 30

Receiver Flowgraph:



Options
ID: top_block
Generate Options: WX GUI

Variable
ID: samp_rate
Value: 2M

WX GUI Slider
ID: rf_gain
Default Value: 0
Minimum: 0
Maximum: 49.6
Converter: Float

WX GUI Slider
ID: freq
Default Value: 145.02M
Minimum: 143M
Maximum: 147M
Converter: Float

RTL-SDR Source
Device Arguments: rtl
Sample Rate (sps): 2M
Ch0: Frequency (Hz): 145.02M
Ch0: Freq. Corr. (ppm): 30
Ch0: DC Offset Mode: Automatic
Ch0: IQ Balance Mode: Off
Ch0: Gain Mode: Manual
Ch0: RF Gain (dB): 0
Ch0: IF Gain (dB): 0
Ch0: BB Gain (dB): 0

osmocom Source
Device Arguments: rtl
Sample Rate (sps): 2M
Ch0: Frequency (Hz): 145.02M
Ch0: Freq. Corr. (ppm): 0
Ch0: DC Offset Mode: Automatic
Ch0: IQ Balance Mode: Off
Ch0: Gain Mode: Manual
Ch0: RF Gain (dB): 10
Ch0: IF Gain (dB): 20
Ch0: BB Gain (dB): 20

GMSK Demod
Samples/Symbol: 2
Gain Mu: 175m
Mu: 500m
Omega Relative Limit: 5m
Freq Error: 0

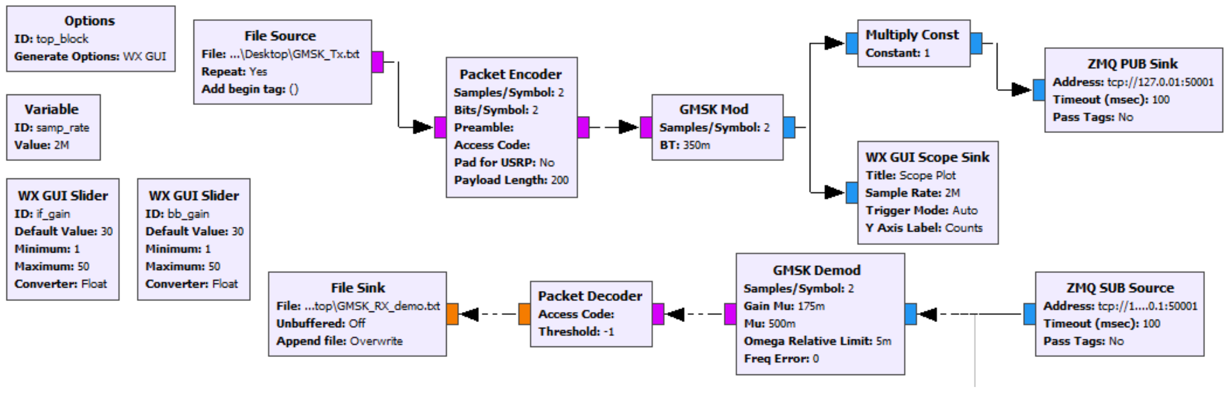
WX GUI Scope Sink
Title: Scope Plot
Sample Rate: 2M
Trigger Mode: Auto
Y Axis Label: Counts

WX GUI Waterfall Sink
Title: Waterfall Plot
Sample Rate: 2M
Baseband Freq: 0
Dynamic Range: 100
Reference Level: 0
Ref Scale (p2p): 2
FFT Size: 512
FFT Rate: 15
Freq Set Varname: None

Packet Decoder
Access Code:
Threshold: -1

File Sink
File: ...rk-comms\Demo\Rx.txt
Unbuffered: Off
Append file: Overwrite

Transmission-Receiver using ZeroMQ blocks (Radio-over-IP)



Options
ID: top_block
Generate Options: WX GUI

Variable
ID: samp_rate
Value: 2M

WX GUI Slider
ID: if_gain
Default Value: 30
Minimum: 1
Maximum: 50
Converter: Float

WX GUI Slider
ID: bb_gain
Default Value: 30
Minimum: 1
Maximum: 50
Converter: Float

File Source
File: ...\Desktop\GMSK_Tx.txt
Repeat: Yes
Add begin tag: ()

Packet Encoder
Samples/Symbol: 2
Bits/Symbol: 2
Preamble:
Access Code:
Pad for USRP: No
Payload Length: 200

GMSK Mod
Samples/Symbol: 2
BT: 350m

Multiply Const
Constant: 1

ZMQ PUB Sink
Address: tcp://127.0.01:50001
Timeout (msec): 100
Pass Tags: No

WX GUI Scope Sink
Title: Scope Plot
Sample Rate: 2M
Trigger Mode: Auto
Y Axis Label: Counts

ZMQ SUB Source
Address: tcp://1....0.1:50001
Timeout (msec): 100
Pass Tags: No

GMSK Demod
Samples/Symbol: 2
Gain Mu: 175m
Mu: 500m
Omega Relative Limit: 5m
Freq Error: 0

Packet Decoder
Access Code:
Threshold: -1

File Sink
File: ...top\GMSK_RX_demo.txt
Unbuffered: Off
Append file: Overwrite

VII. Milestones and Progression

Date	Harshit	Samir	Gianna
3/14	- Agree on project vision and direction - Research and decide on appropriate components: ○ GNUradio, ZeroMQ, Cryptography, DSP, Twitter API - Draft proposal & Distribute tasks		
3/21	- Research encoding techniques - Install GNUradio and verify working	- Look into AES specification - Design cryptographic model	- Generate Twitter API key and authentication
3/28	- Test transmit and receive with sdr - Run basic encoding techniques and check for packet loss while transmission	- Run basic code on both machines - Write key generation and exchange scripts	- Develop code to pull Twitter information - Work with user interface to draw on specific information to retrieve
4/4	- Working towards converting ciphertext to modulated carrier signal wave - Implement gnuradio flow graph	- Automate key generation/exchange protocol from TLS suite - Integrate encryption/decryption algorithm and run on Pi	- Run tests on sending the specific tweets to Samir's module - Work on user interface for receiving the message
4/11	- Connect the system to check integrity and response rate	- Automate full model on every Alice and Bob interaction - Perform sanity check by interconnecting Gianna's and Harshit's modules	- Connect with Samir's module and test connectivity of encrypting and decrypting the tweets
4/18	- Run several more general tests to make sure the entire system is robust and intact		
4/25	- A detailed manual shall be uploaded to Github, with all the procedures and steps to install our software. We will prepare for the final presentation (if any) and write the final report. - This part is not as hardcore, but time-consuming, so the last week was planned as such.		

Responsibilities

As for responsibilities, Samir has more experience with cryptography and networking, therefore he designed the system for secure encryption. Gianna is familiar with python and front-end development and led the user interface and integration of different project modules. Harshit is more oriented toward security and wireless communication and worked on the Physical layer part, writing code for RF-modulation schemes, and ensuring the overall security of the system. During the course of the project, we made sure everyone participated in everything so everyone learns something new.

VIII. Takeaways, Reflections and Future Work

Secure communication are when two entities are communicating and do not want a third party to listen in. For that, they need to communicate in a way not susceptible to eavesdropping or interception. While standard secrecy methods such as cryptography protect the contents of the message from being accessed by unauthorized users, covert communication conceals the existence of the communication to prevent unauthorized users to detect the communication. Secure communication includes means by which people can share information with varying degrees of certainty that third parties cannot intercept what was said. Other than spoken face-to-face communication with no possible eavesdropper, it is probably safe to say that no communication is guaranteed secure in this sense, although practical obstacles such as legislation, resources, technical issues (interception and encryption), and the sheer volume of communication serve to limit surveillance. With many communications taking place over long distances and mediated by technology, an increasing awareness of interception issues and technology are at the heart of this debate.

The proposed system is designed to be used for secret code transmissions needed in military, government, or other sensitive communications. In the future roadmap, we desire to update the system with the following functionalities:

1. Use of Open-source Signal protocol for End-to-End Encryption which is unique because of Perfect Forward Secrecy, which essentially describes the idea that if data is encrypted and secret now, it should also be encrypted and secret in the future.
2. Use of X3DH Protocol specification for private server replacing the existing Ubuntu keyserver.
3. Add dual-channel communication with more transceivers.
4. Add frequency hopping modules between a range of frequencies to avoid detection.
5. Use a Signal Power variation module which could change and hop between a range of RF gain, IB gain, and BB gain settings to change transmission power ensuring safety against triangulation and localization of source and destination.
6. Add option to add/integrate other messenger services API with existing system i.e Telegram.
7. Compact the relevant programs for senders and receivers to create a natural extension of the work.

On the matter of crypto, our original plan was to use symmetric key encryption and the Diffie-Hellman Key Exchange Protocol in combination with MACs and AES. Alice and Bob would establish a prime multiplicative group p and base g. They would then perform the necessary generation and message sends to produce a key as per the below diagram. We would combine AES and MAC to produce a modified CBC-MAC for variable-length message encryption/decryption. This modification would involve length-prepending and last-block encryption to ensure variable-length message indistinguishability. The variable-length consideration comes into play at the final block, where our final block, already signed with the MAC key, is encrypted with the 3rd final-block key.

IX References

1. Dudak, Juraj, Gabriel Gaspar, and Pavol Tanuska. "Implementation of Secure Communication via the RF Module for Data Acquisition." Journal of Sensors 2019 (2019).
2. Babak Kia. "CE In a Connected World Project Proposal Grading Criteria." Blackboard: EC544 (2022).
3. Toliupa, Serhii, et al. "RF Signals Encryption with AES in WDID." IT&I Workshops. 2020. (2020)
4. Cameron MacDonald Surman, Hairuo Sun, Yi-Wei Chen. “Satellite Project Final Report.” EC544 Networking the Physical World (2020)
5. Zimmermann, Hubert. "OSI reference model-the ISO model of architecture for open systems interconnection." IEEE Transactions on communications 28.4 (1980): 425-432.
6. Agrawal, Monika, and Pradeep Mishra. "A comparative survey on symmetric key encryption techniques." International Journal on Computer Science and Engineering 4.5 (2012): 877.
7. Yassein, Muneer Bani, et al. "Comprehensive study of symmetric key and asymmetric key encryption algorithms." 2017 international conference on engineering and technology (ICET). IEEE, 2017.
8. Blumenthal, Uri, Fabio Maino, and Keith McCloghrie. "The advanced encryption standard (AES) cipher algorithm in the SNMP user-based security model." Internet proposed standard RFC 3826 (2004).
9. Petrank, Erez, and Charles Rackoff. "CBC MAC for real-time data sources." Journal of Cryptology 13.3 (2000): 315-338.
10. Twitter Developer Platform. “Twitter API v2 Search tweets”. Twitter, https://developer.twitter.com/en/docs/twitter-api/tweets/search/api-reference/get-tweets-search-recent.
11. Twitter Developer Platform. “Twitter API v2 Fundamentals”. Twitter, https://developer.twitter.com/en/docs/twitter-api.
12. de Winter, Brenno, and Michael Fischer V. Mollard. "Gnu Privacy Guard (GnuPG) Mini Howto." retrieved from the Internet: http://www. gnupg. org/documentation/howtos. en. html, version 0.1 4 (2003).
13. Jallad, Kahil, Jonathan Katz, and Bruce Schneier. "Implementation of chosen-ciphertext attacks against PGP and GnuPG." International Conference on Information Security. Springer, Berlin, Heidelberg, 2002.
14. Leurent, Gaëtan, and Thomas Peyrin. "{SHA-1} is a Shambles: First {Chosen-Prefix} Collision on {SHA-1} and Application to the {PGP} Web of Trust." 29th USENIX Security Symposium (USENIX Security 20). 2020.
15. Ziegler, Jack L., Robert T. Arn, and William Chambers. "Modulation recognition with GNU radio, keras, and HackRF." 2017 IEEE International Symposium on Dynamic Spectrum Access Networks (DySPAN). IEEE, 2017.
16. Wang, Dawei, et al. "Secure transmission for mixed FSO-RF relay networks with physical-layer key encryption and wiretap coding." Optics Express 25.9 (2017): 10078-10089.
17. Barrie, Christopher, and Justin Chun-ting Ho. "academictwitteR: an R package to access the Twitter Academic Research Product Track v2 API endpoint." Journal of Open Source Software 6.62 (2021): 3272.
18. Raavi, Manohar, et al. "Security comparisons and performance analyses of post-quantum signature algorithms." International Conference on Applied Cryptography and Network Security. Springer, Cham, 2021.