VoIPShark: Open Source VoIP Analysis Platform

Nishant Sharma
Jeswin Mathai
Ashish Bhangale
PentesterAcademy.com & AttackDefense.com

About Us

Me, **Nishant Sharma**

- R&D Manager and Lead Trainer, Pentester Academy
- Firmware developer, Enterprise WiFi APs and WIPS Sensors
- Masters degree in Infosec
- Published research at Blackhat US/Asia, DEF CON USA and other venues

Co-authors

- **Ashish Bhangale**, Sr. Security Researcher
- **Jeswin Mathai**, Security Researcher

PentesterAcademy.com

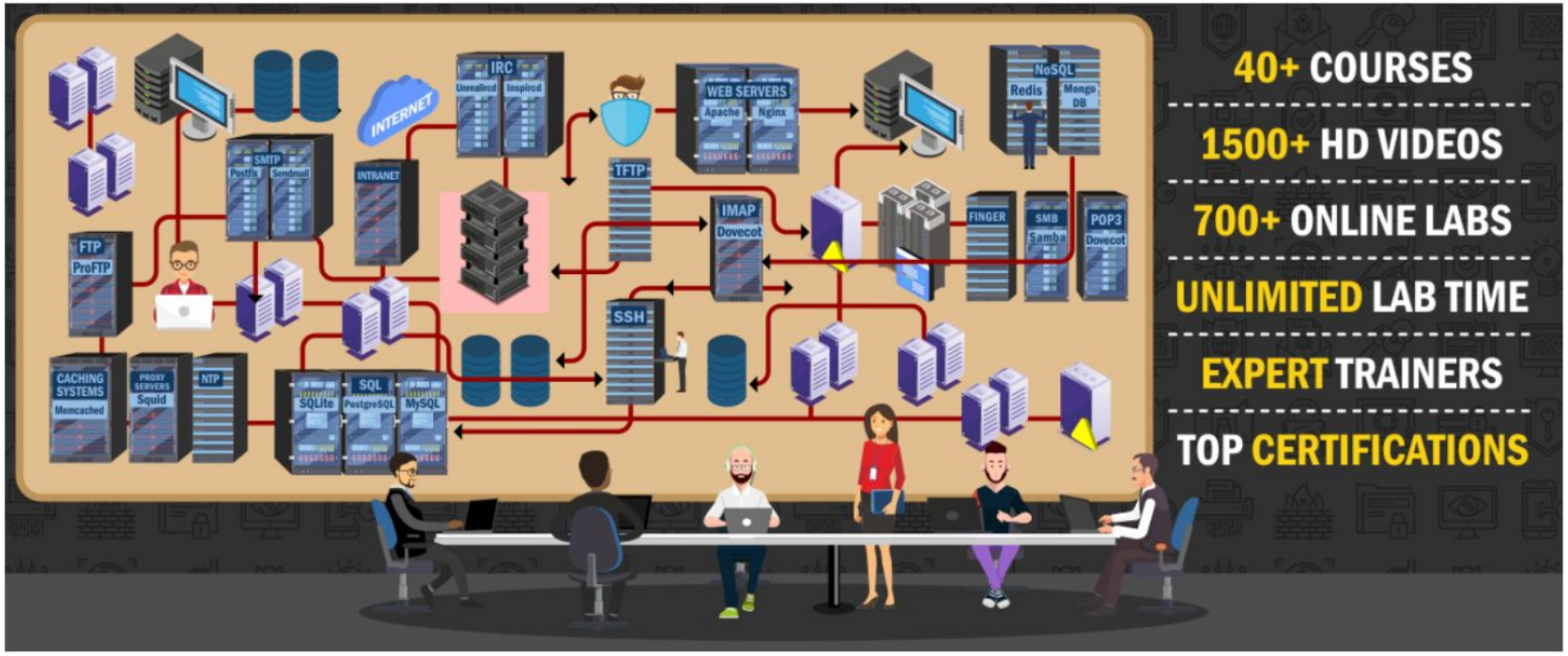
https://www.pentesteracademy.com

PentesterAcademy | Courses and Online Labs

Follow @SecurityTube 117K followers

Recommend 291K Share

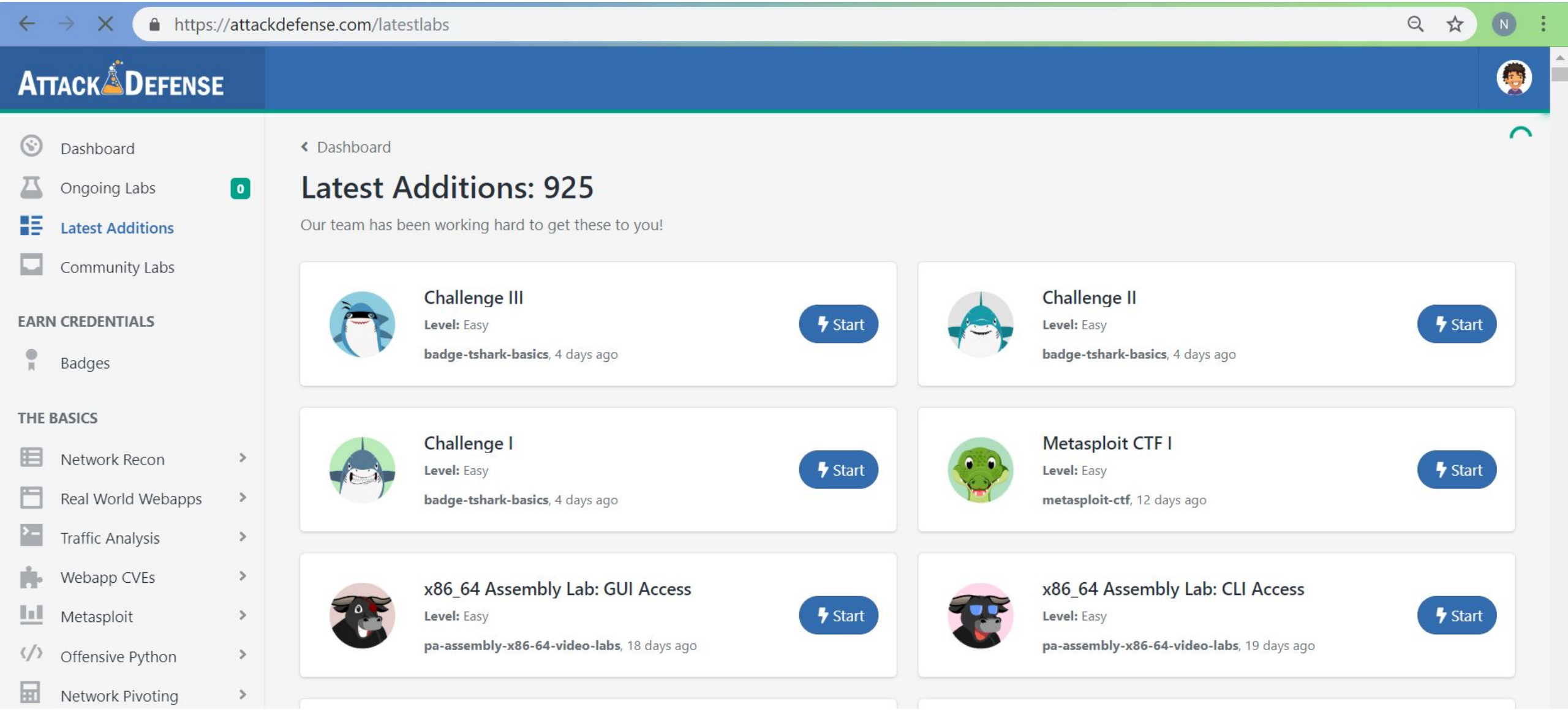
COURSES | ONLINE LABS | PRICING | WHY SUBSCRIBE | TESTIMONIALS | RED TEAM LABS | BLOG | MEMBER ACCESS



Training Professionals from

IBM vmware Symantec McAfee hp SONY

AttackDefense.com

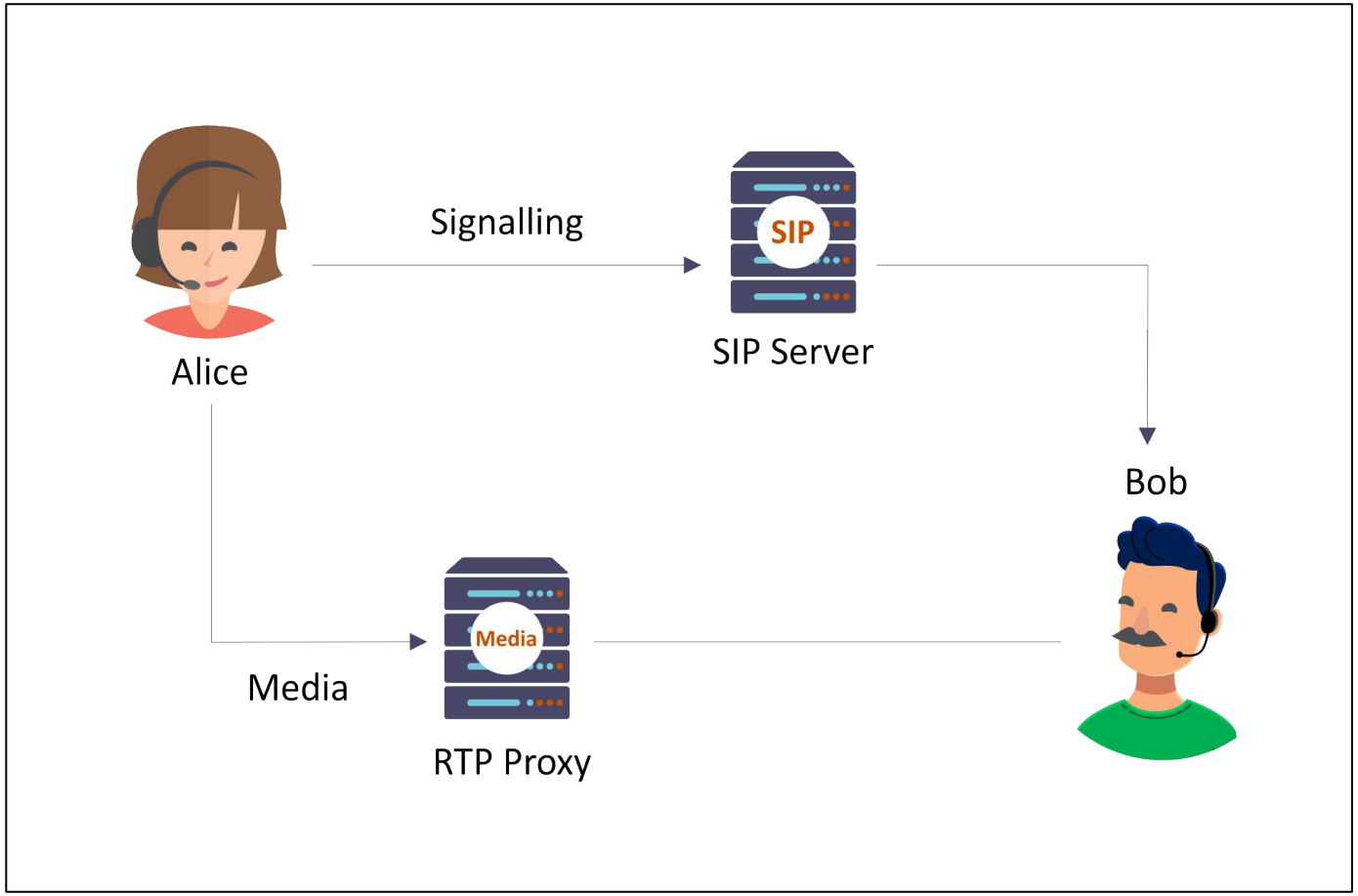


Talk Overview

- VoIP Basics
  - SIP, RTP
  - Secure: TLS, SRTP
- Recovering/Decrypting VoIP Calls
- Current open source tools and issues
- VoIPShark
  - Architecture and Internals
  - Analyzing VoIP Traffic
  - Recovering Calls
  - Detecting Attacks Passively
  - Demo

VoIP Telephony

- Signalling + Media



Signalling Protocols

SIP (Session Initiation Protocol)

- Developed by the IETF
- Replacement for the desk phones and PSTN (Public Switched Telephone Network)

H.323

- Created by the ITU-T
- Focused on videoconferencing but also used for voice calls

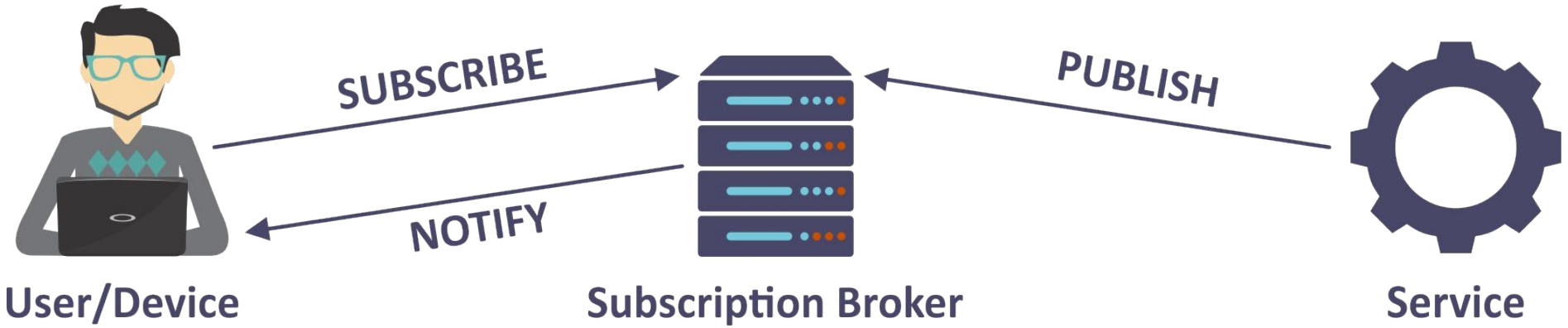
SCCP (Skinny)

- Cisco proprietary protocol used for line-side control of phones

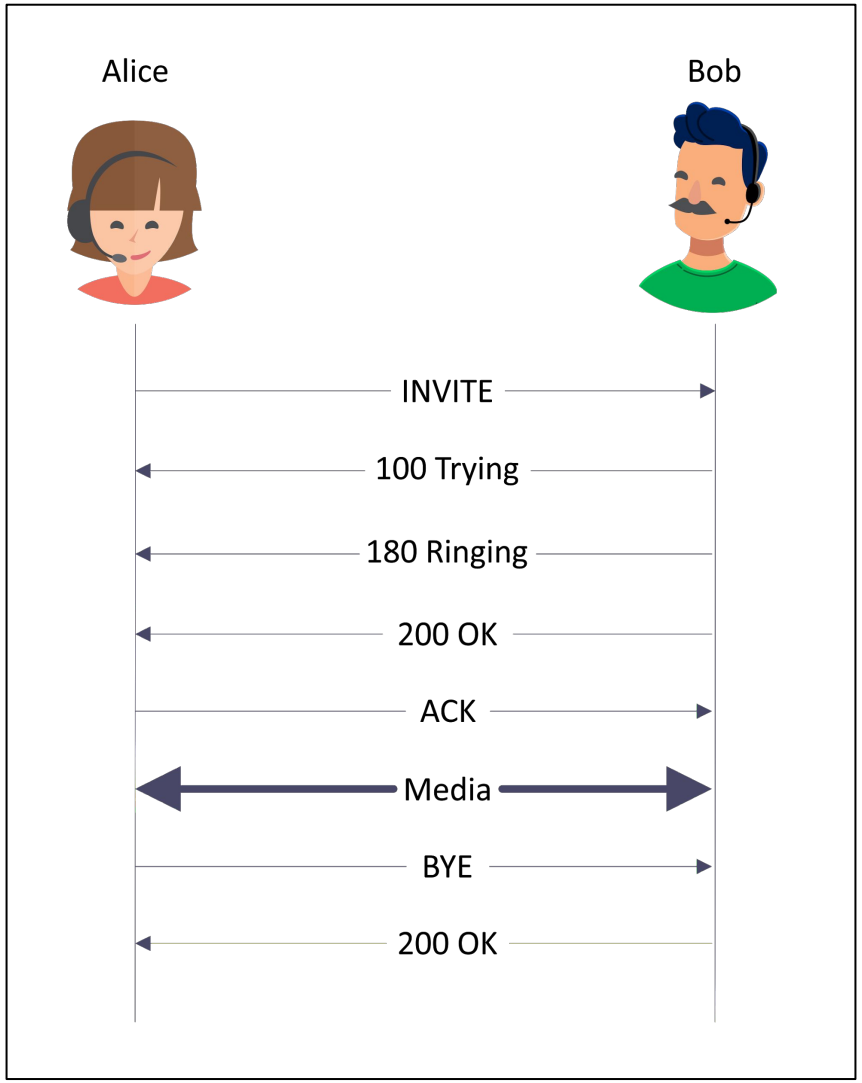
Session Initiation Protocol

- Text-based protocol
- Applications
  - Calls (audio, video) using other media steams like RTP
  - Text messages using SIP “Message” method
- Works with other protocols
- Session Description Protocol (SDP) to define with media negotiation and setup
- Can operate over TCP, UDP or SCTP (Stream Control Transmission Protocol)
- Security is provided by TLS (Transport Layer Security) i.e. SIP over TLS.

SUBSCRIBE, PUBLISH and NOTIFY



Session Initiation Protocol: Sample Call Flow



User Agent Server (UAS) Solutions

www.asterisk.org

www.sipfoundry.org

www.3cx.com

www.elastix.org

freeswitch.org

Softphone clients

- Program for making telephone calls over IP
- Some options
  - Zoiper
  - X Lite
  - LinPhone
  - MicroSIP

www.counterpath.com/x-lite-download

www.microsip.org

www.zoiper.com

Factors in choosing a good softphone client

- Check codec support
- Check encryption capabilities (Especially in free versions)
- Other functionalities (i.e. Text message option, hold, waiting etc.)

www.linphone.org

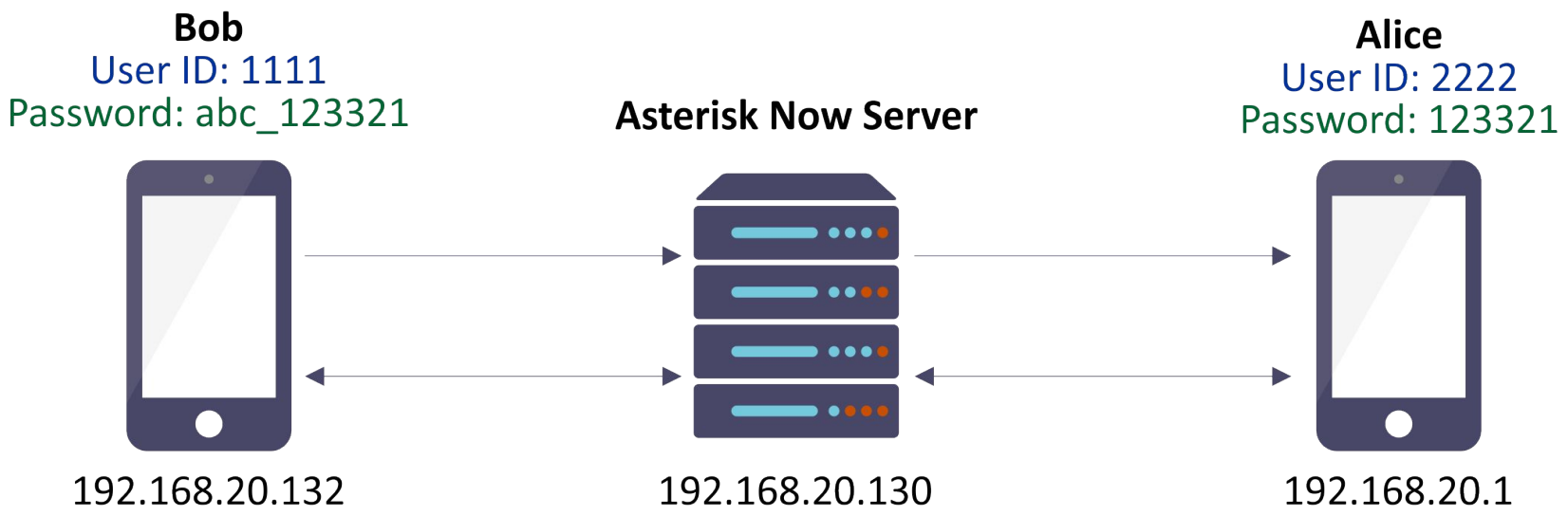
www.3cx.com

Asterisk Now

=

+

Scenario



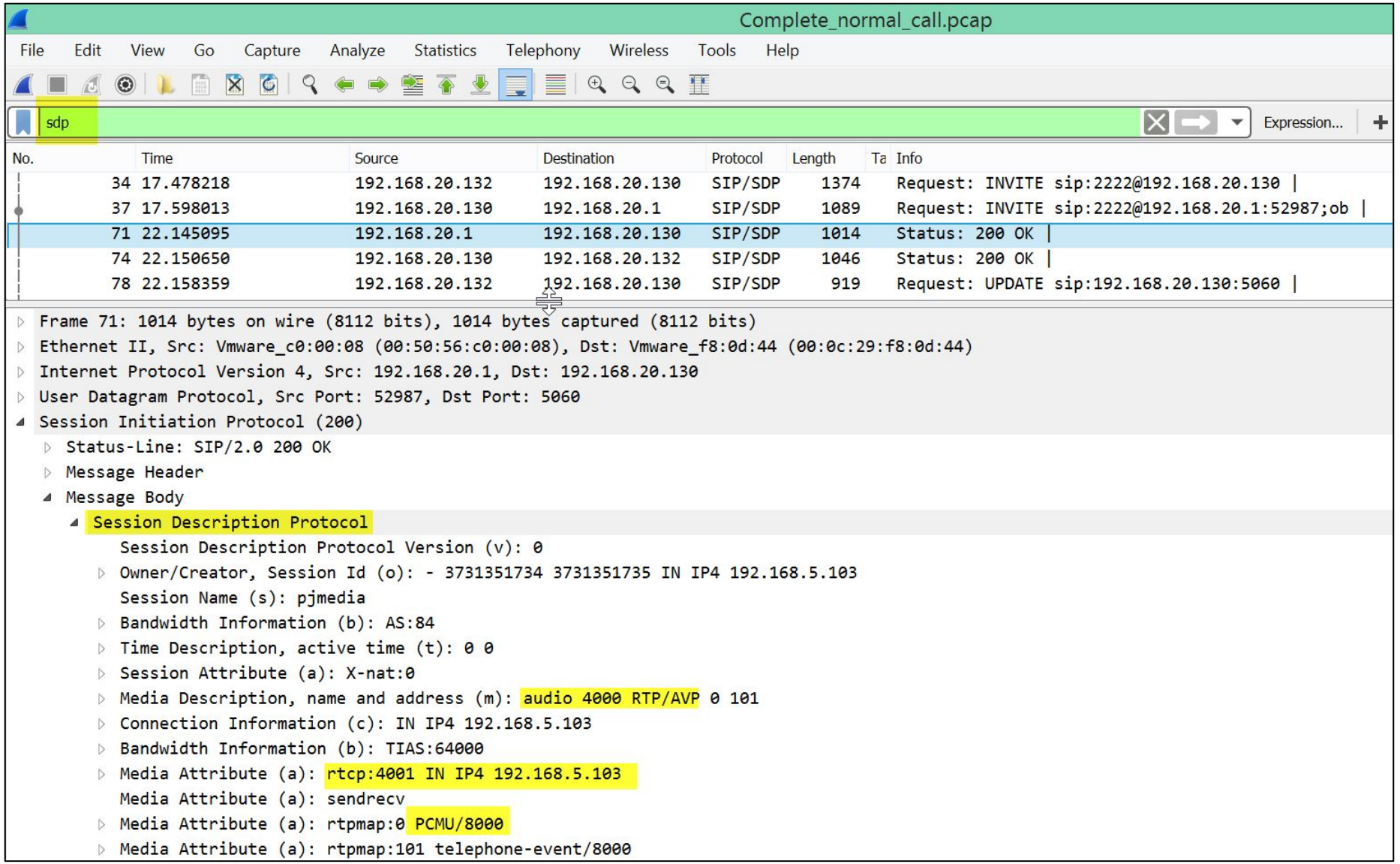
Possible Configurations

- SIP + RTP
- SIP over TLS + RTP
- SIP + SRTP
- SIP over TLS + SRTP

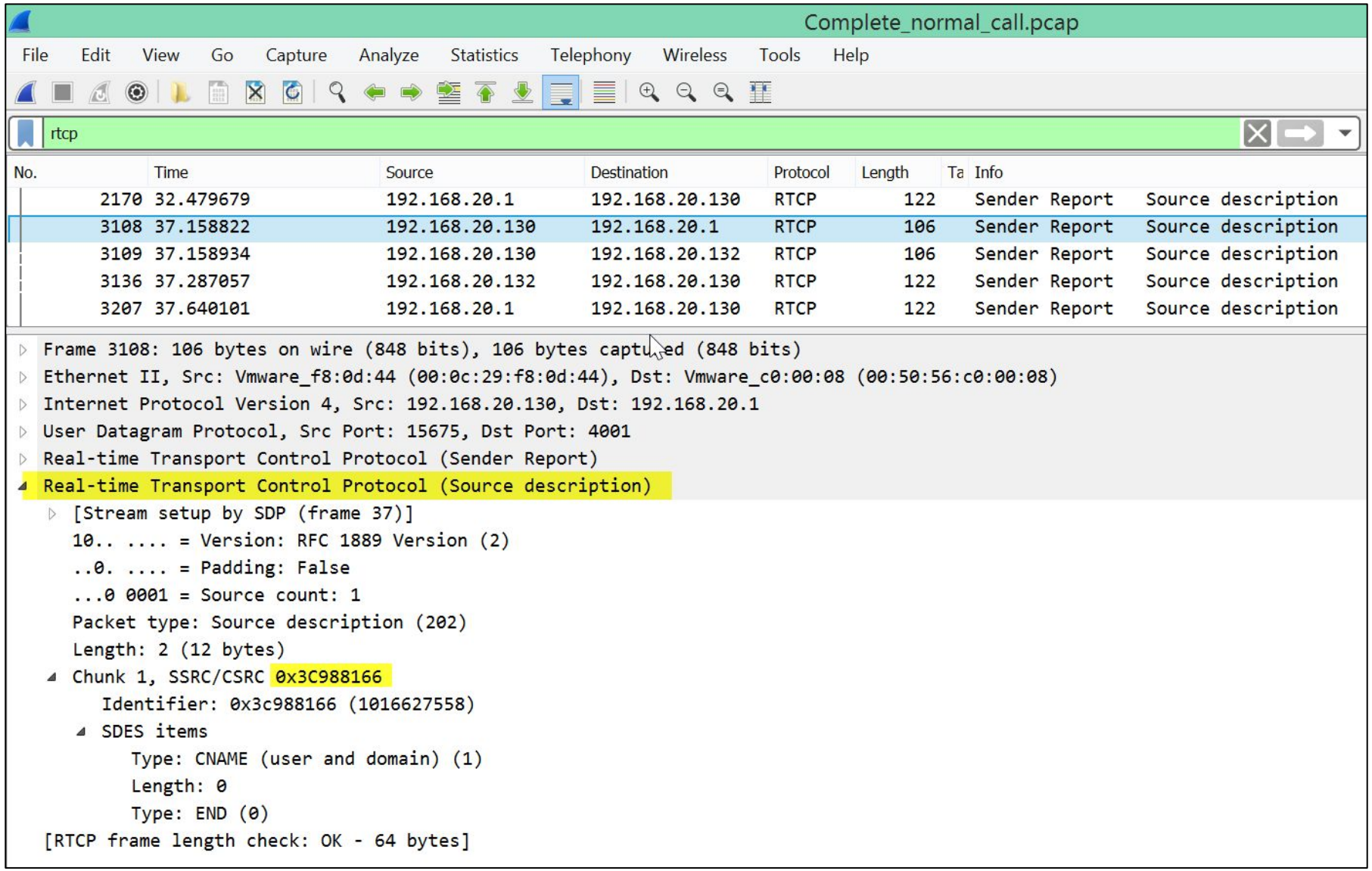
Possible Configurations

- **SIP + RTP**
- SIP over TLS + RTP
- SIP + SRTP
- SIP over TLS + SRTP

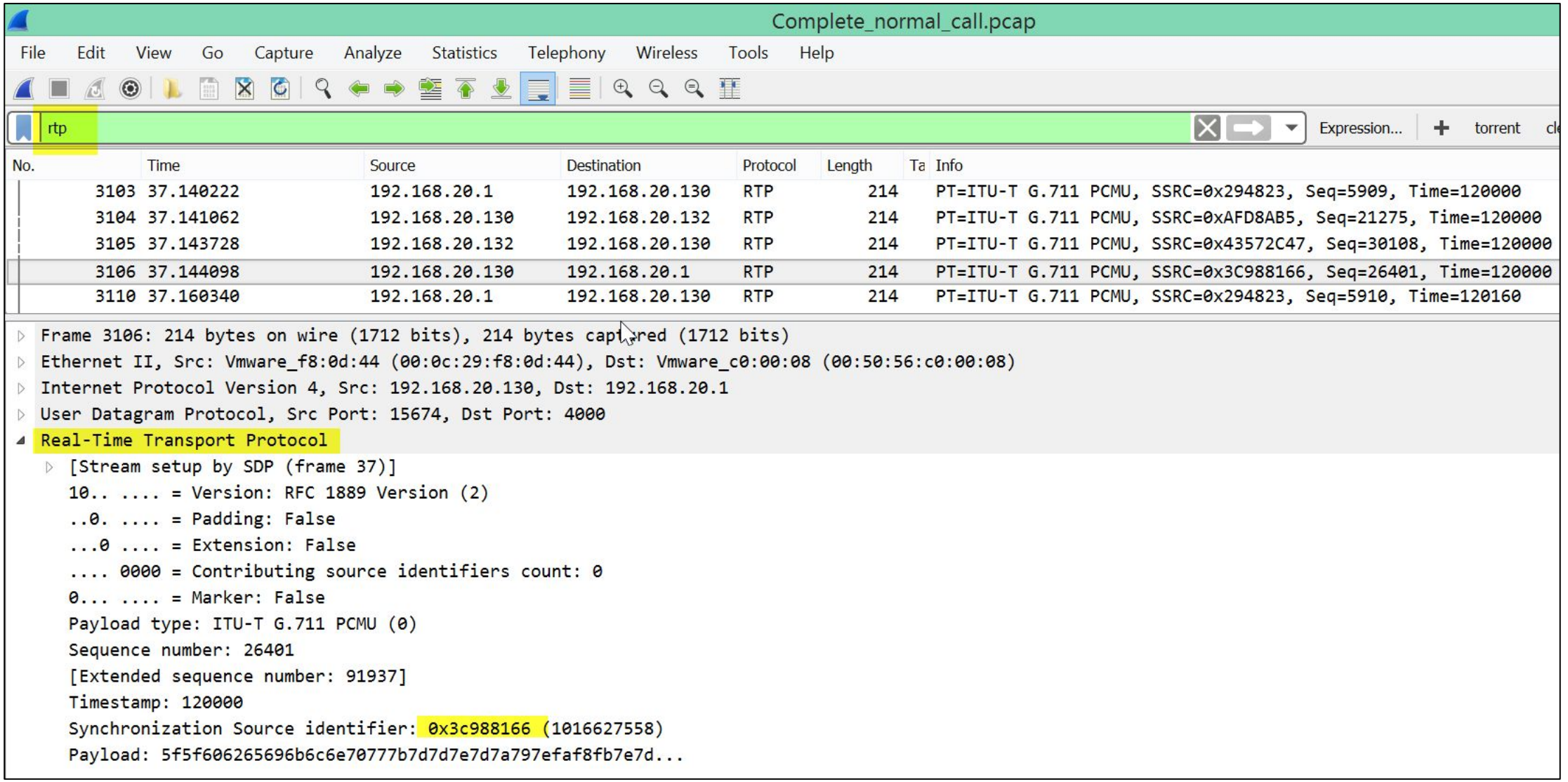
SIP/SDP Packets



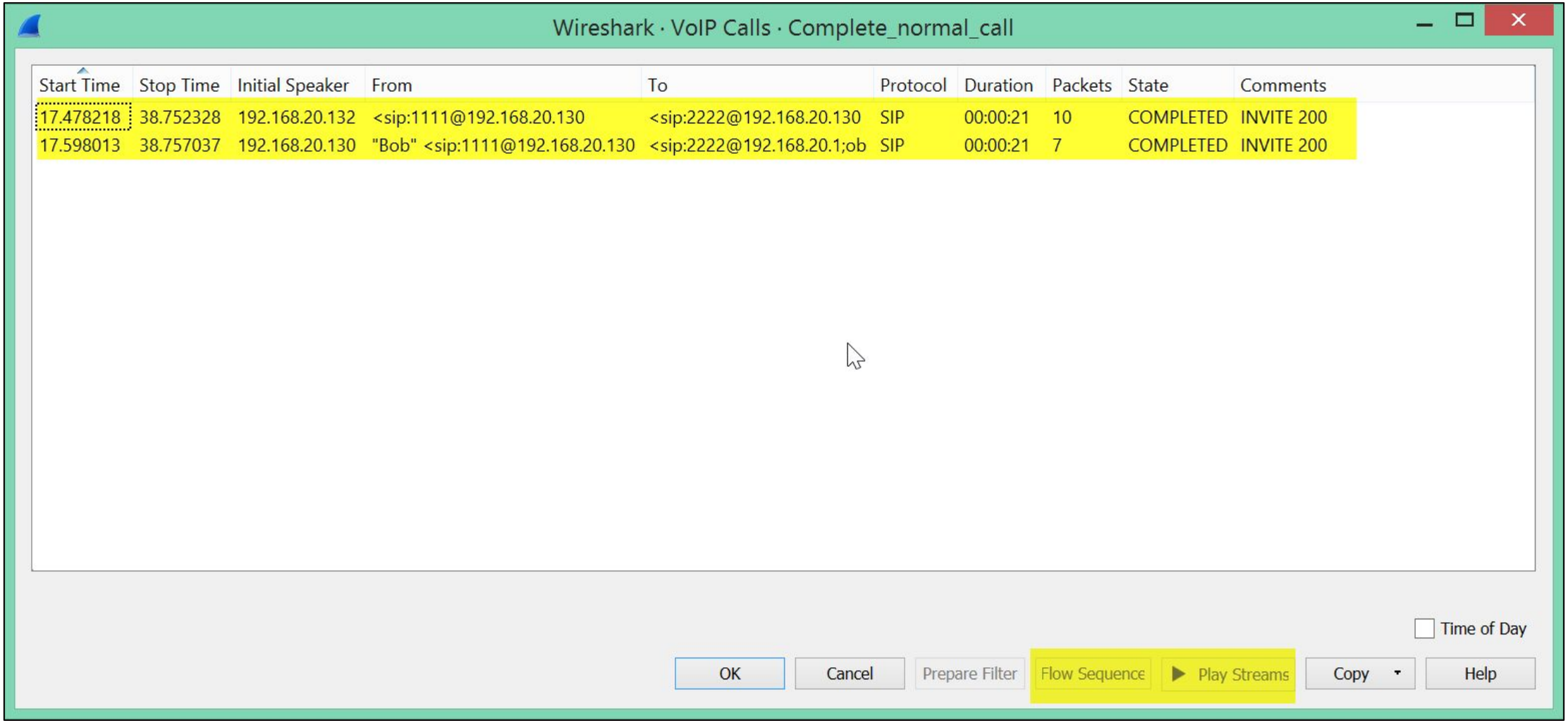
RTCP Packets



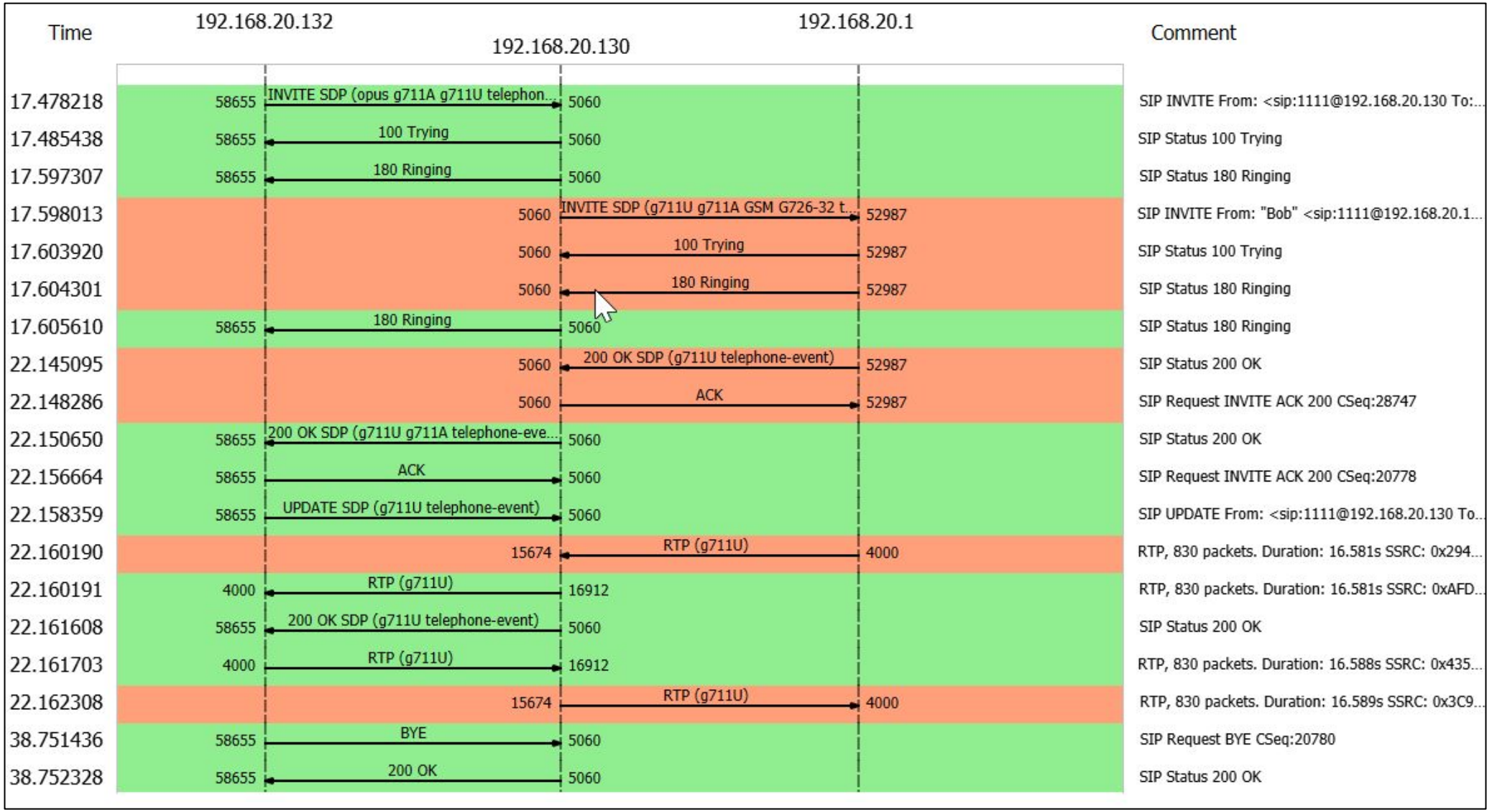
RTP Packets



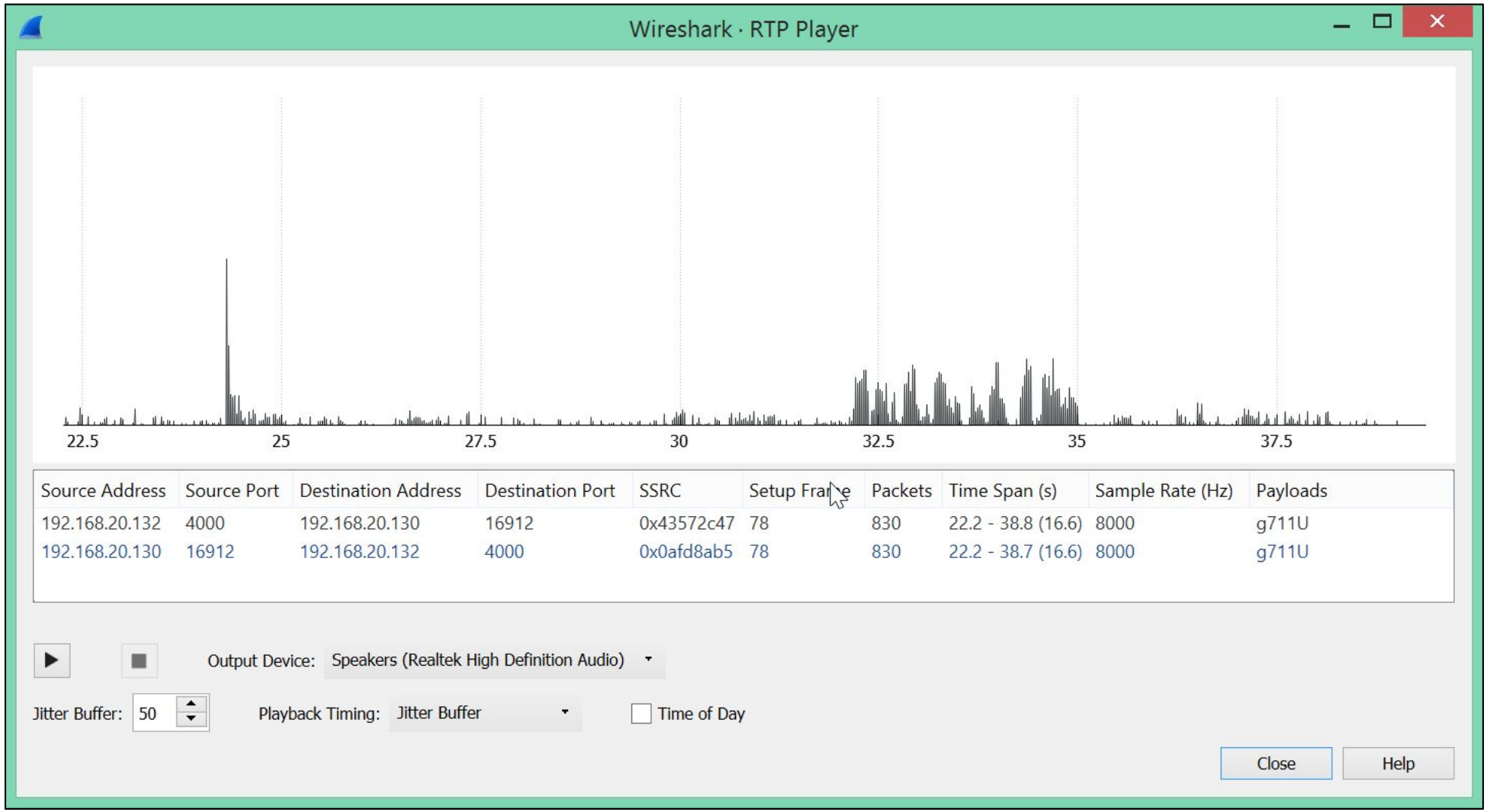
Recovered VoIP Calls



Flow Sequence



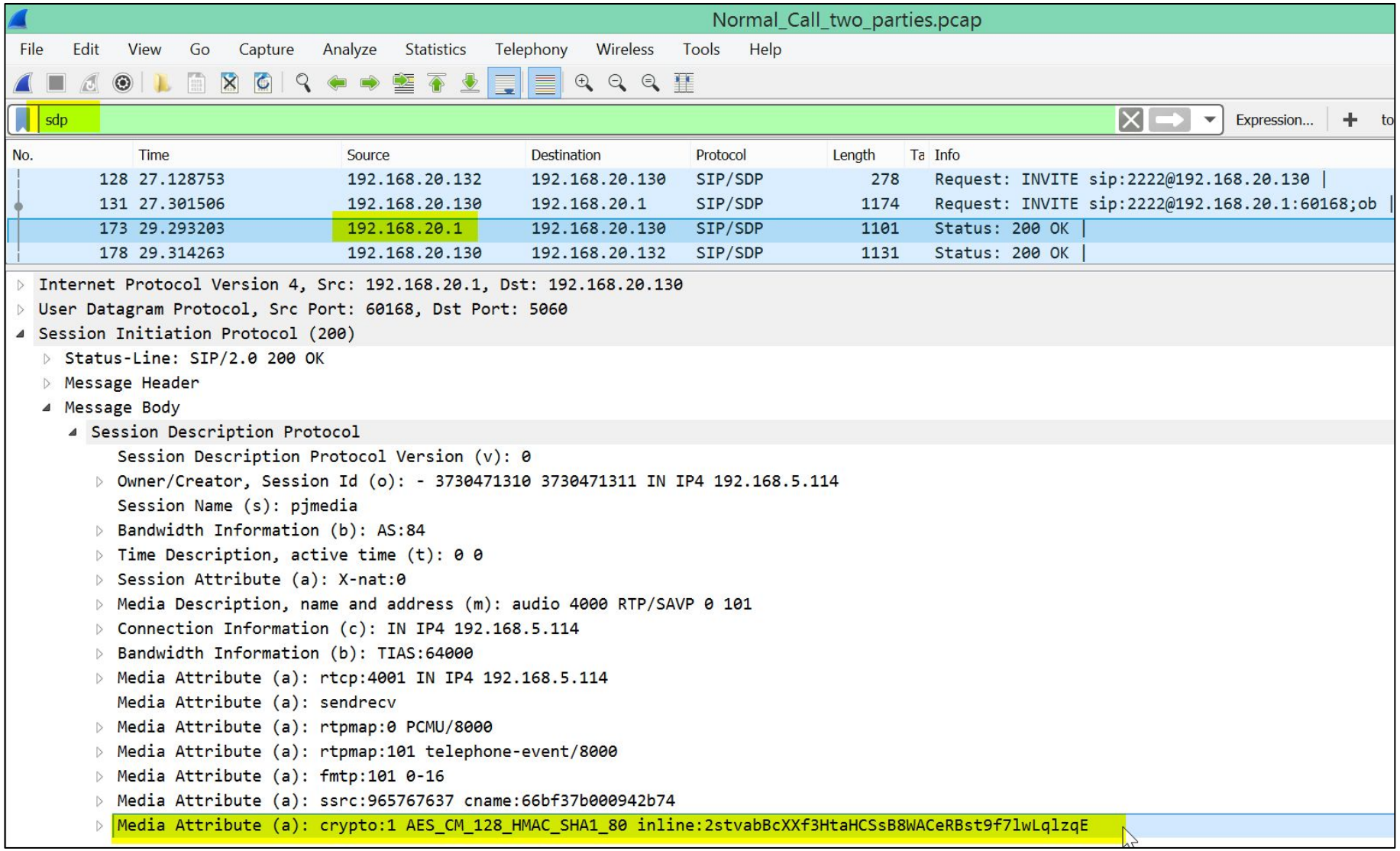
Reconstructed Call



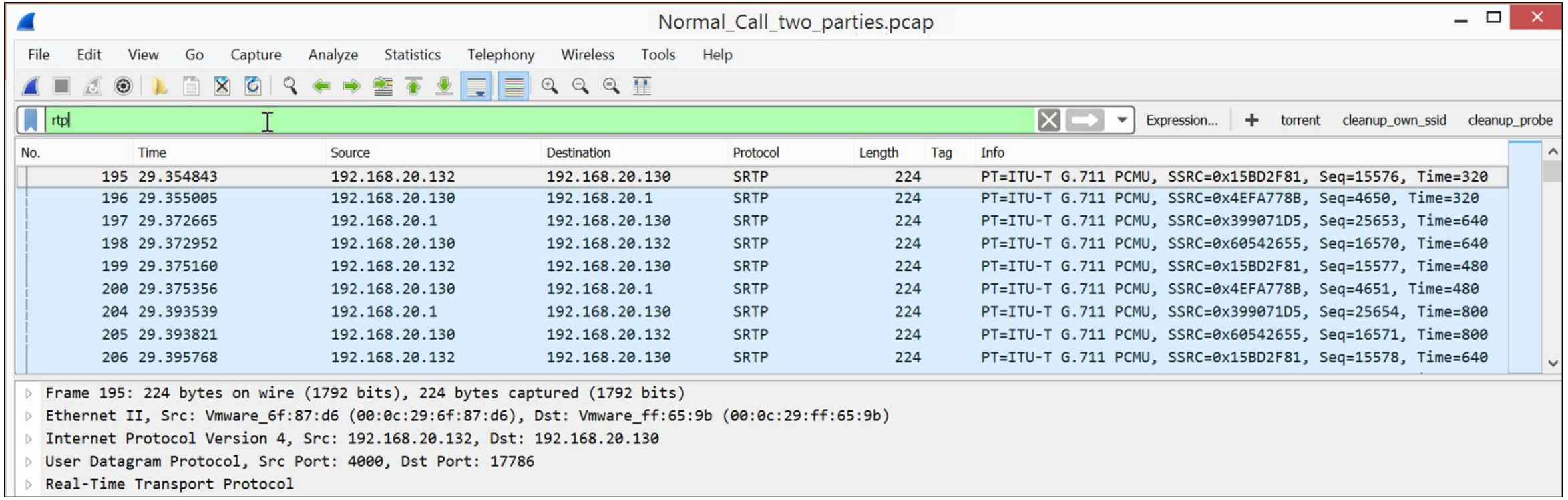
Possible Configurations

- SIP + RTP
- SIP over TLS + RTP
- **SIP + SRTP**
- SIP over TLS + SRTP

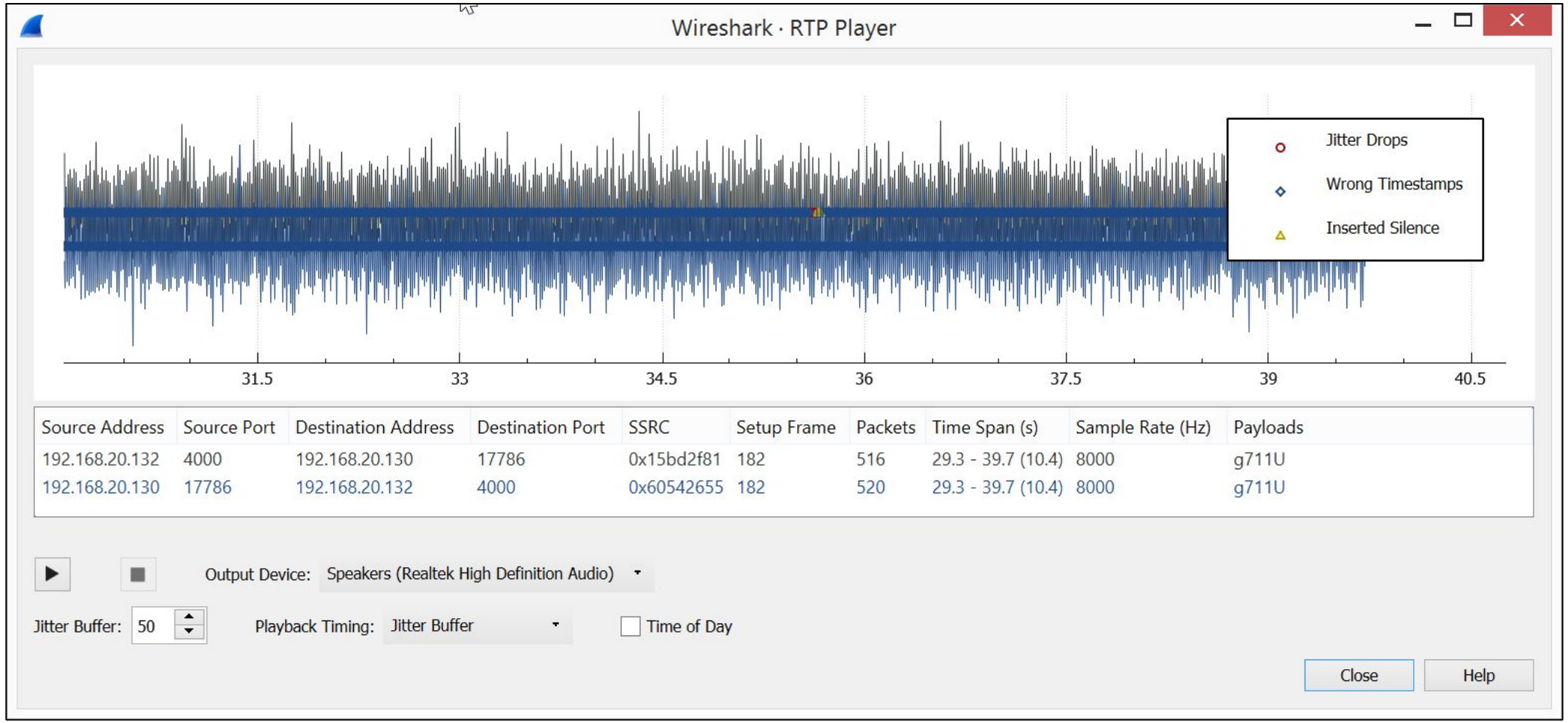
SRTP key in SDP packet



SRTP Traffic



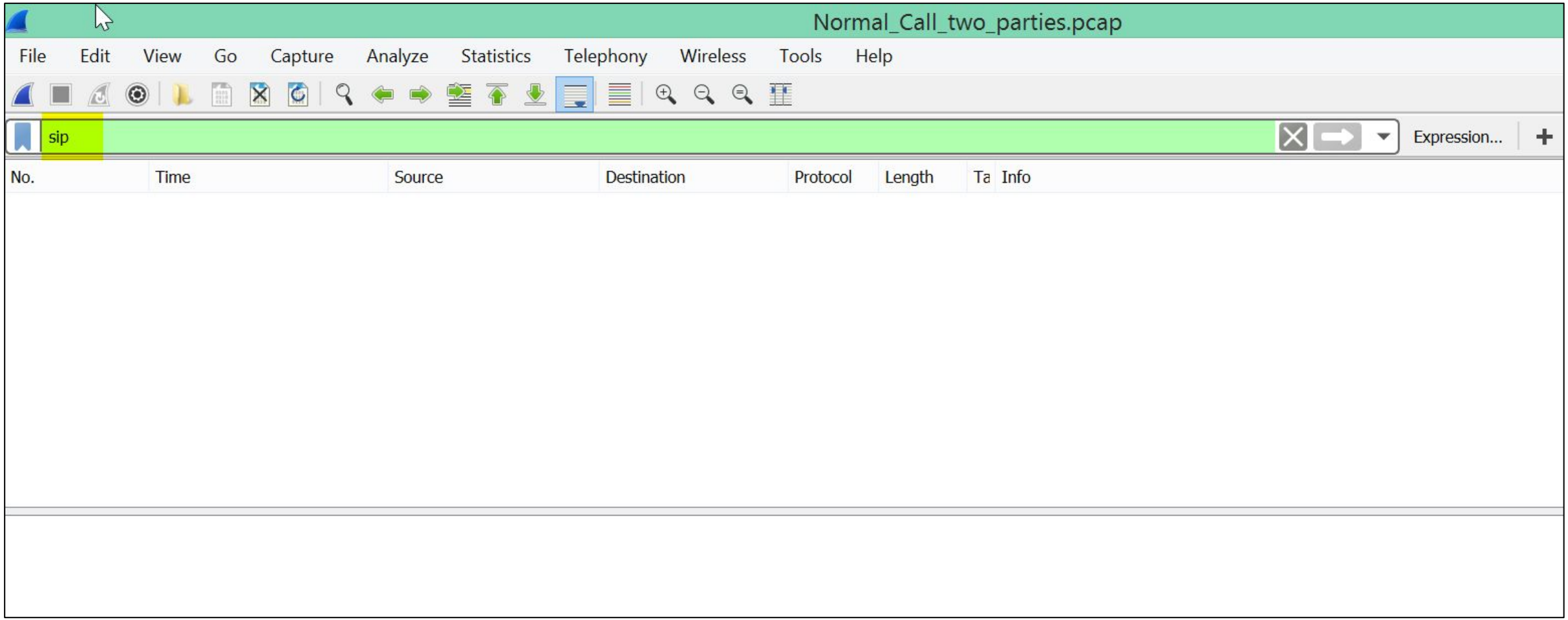
Encrypted Call



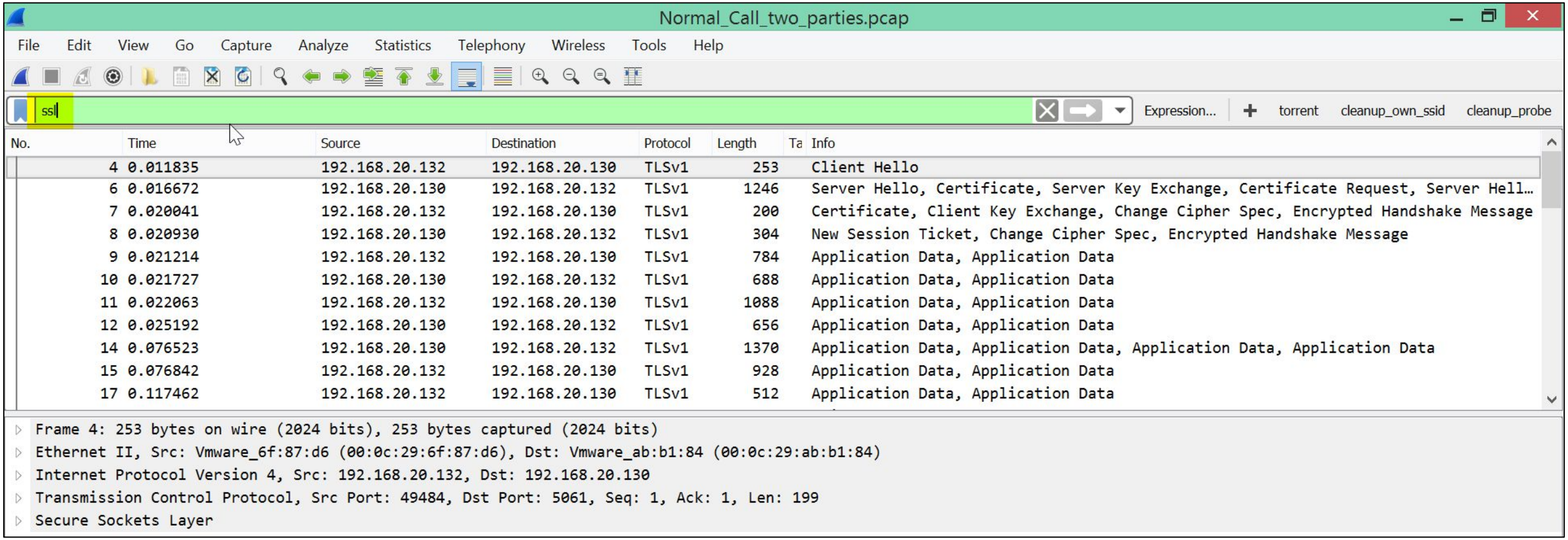
Possible Configurations

- SIP + RTP
- **SIP over TLS + RTP**
- SIP + SRTP
- SIP over TLS + SRTP

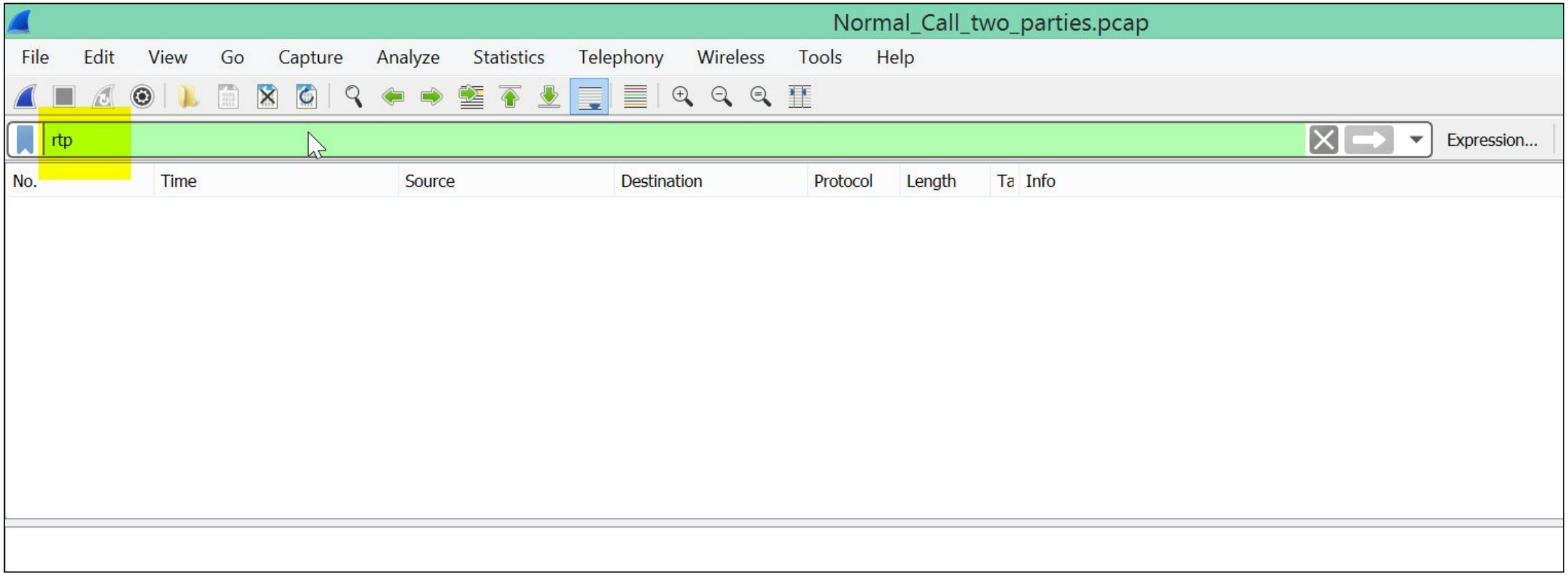
No SIP Traffic



TLS Traffic (SIP over TLS)

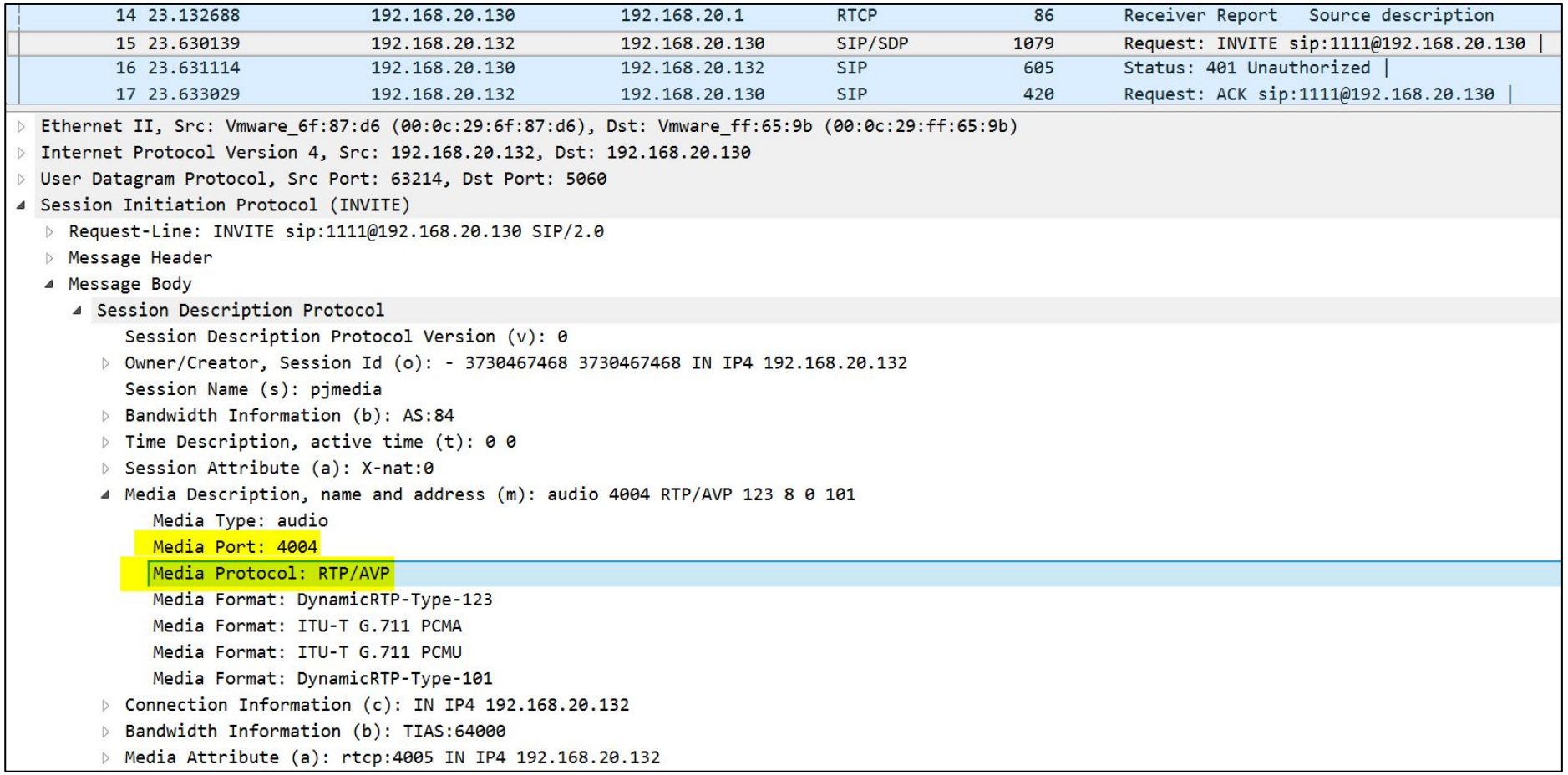


No RTP Traffic

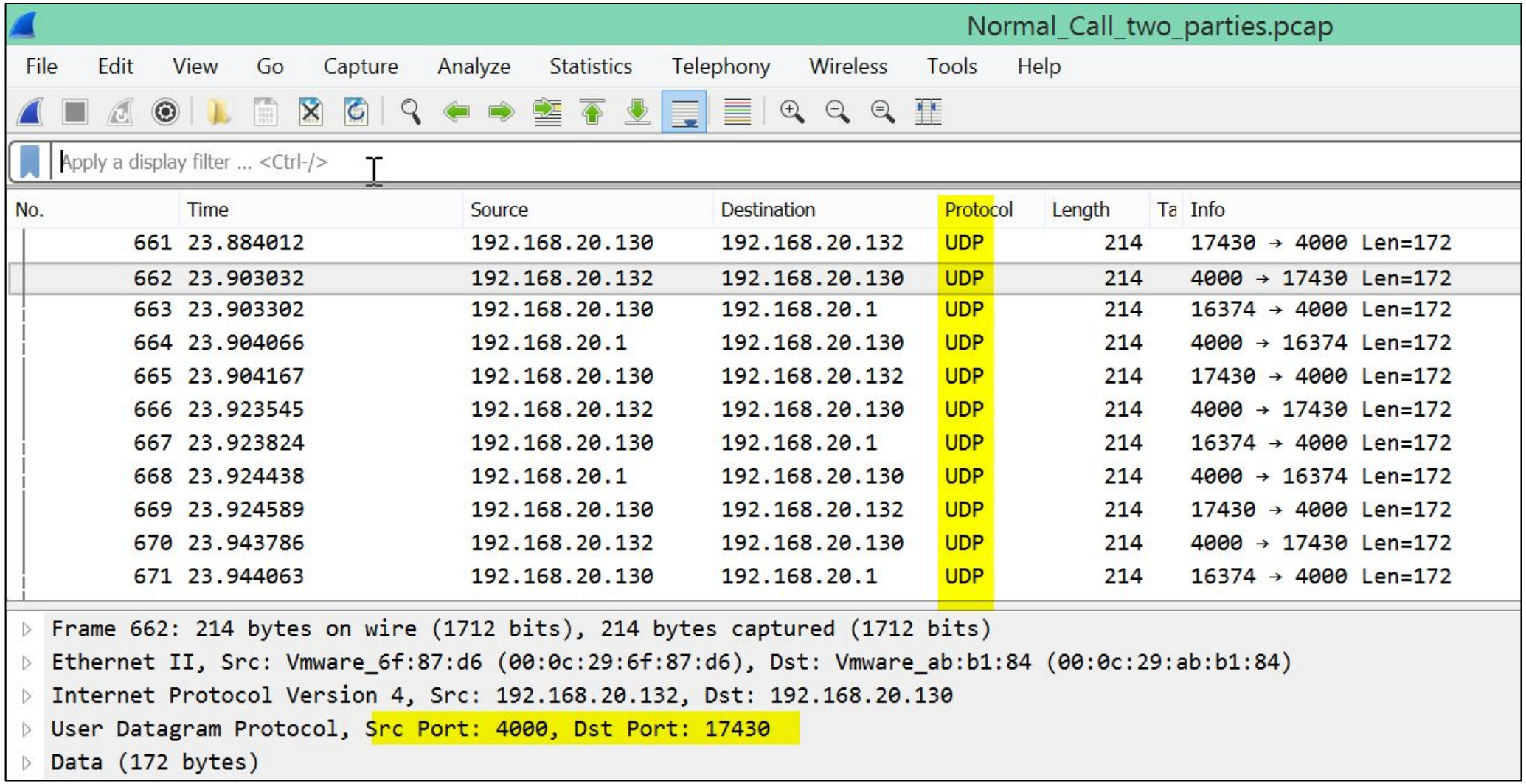


Why No RTP Traffic?

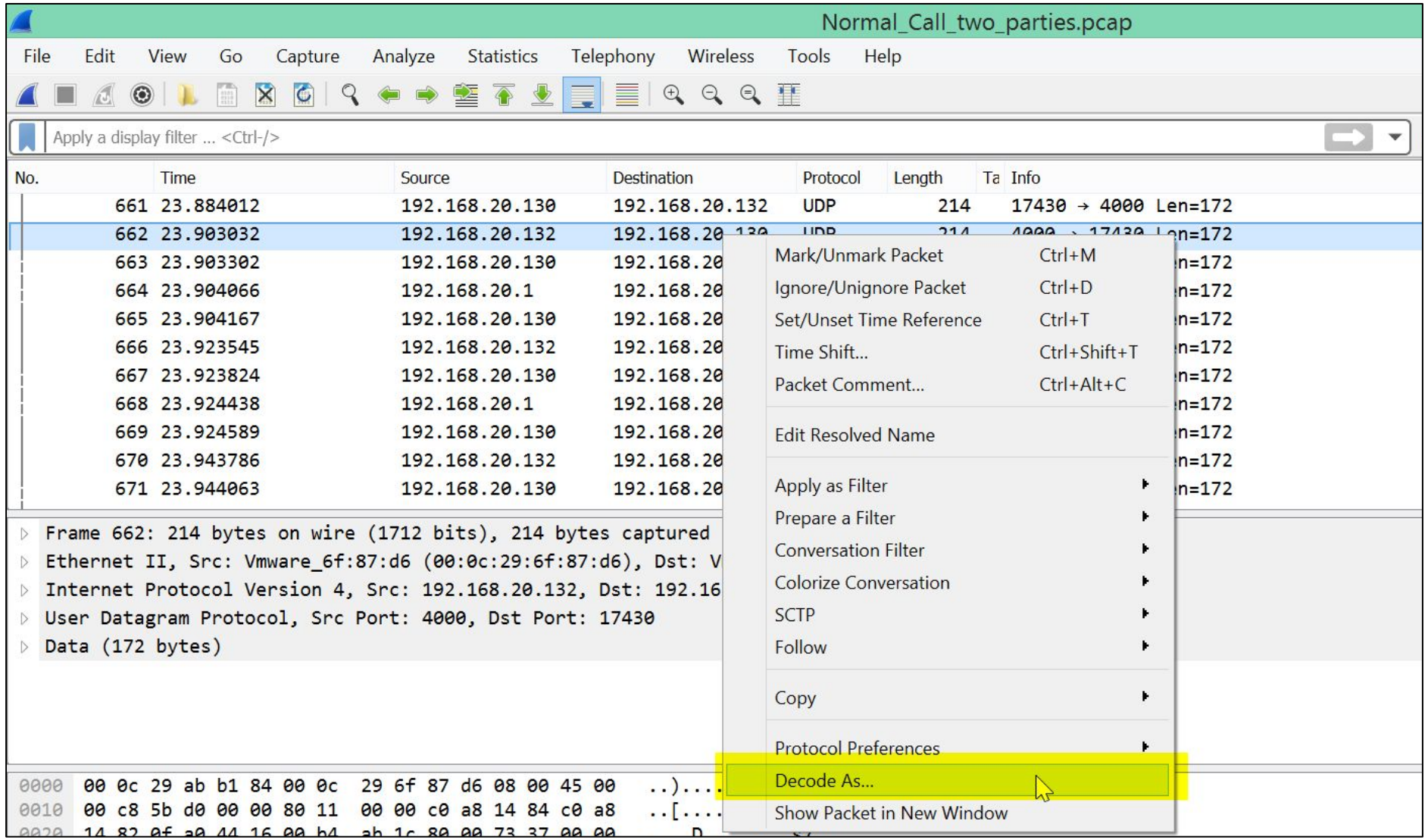
- Wireshark uses SDP packet to figure out the port RTP/SRTP stream will use.
- SIP and SDP are encrypted, so wireshark can't figure out.



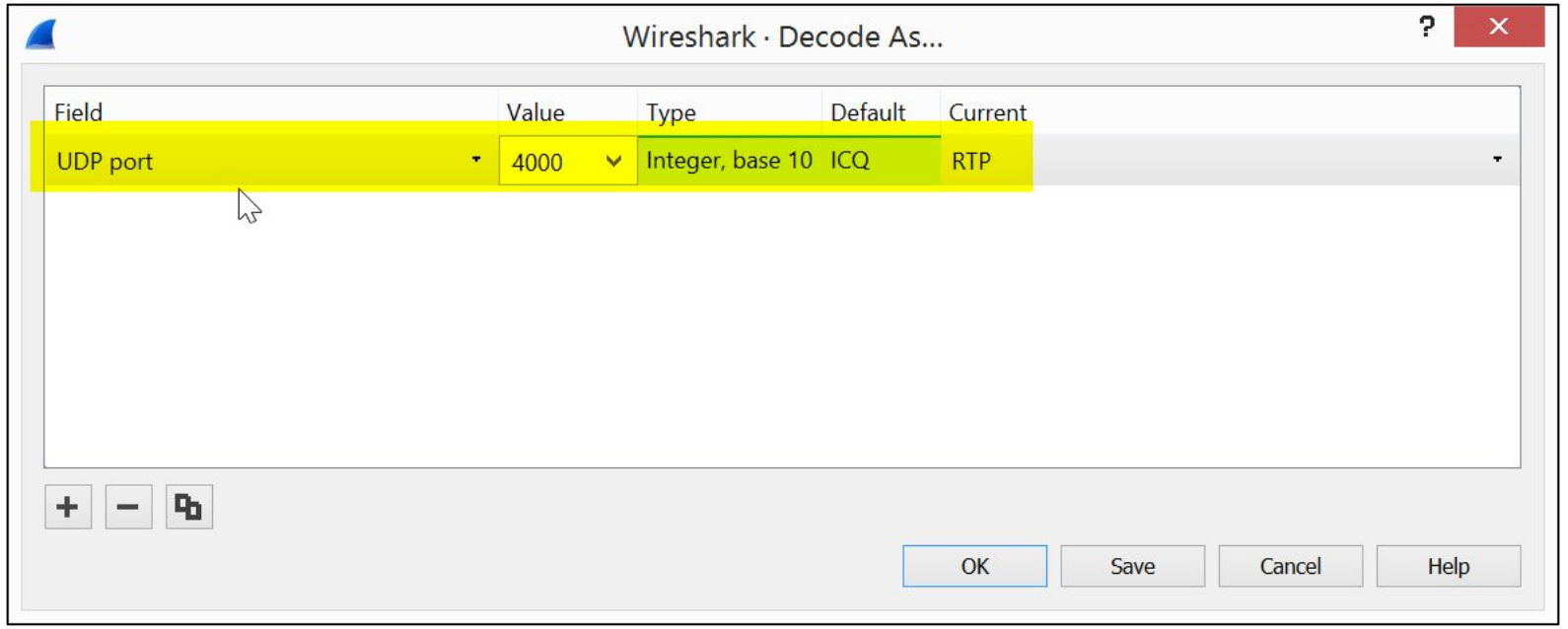
Undecoded RTP Traffic



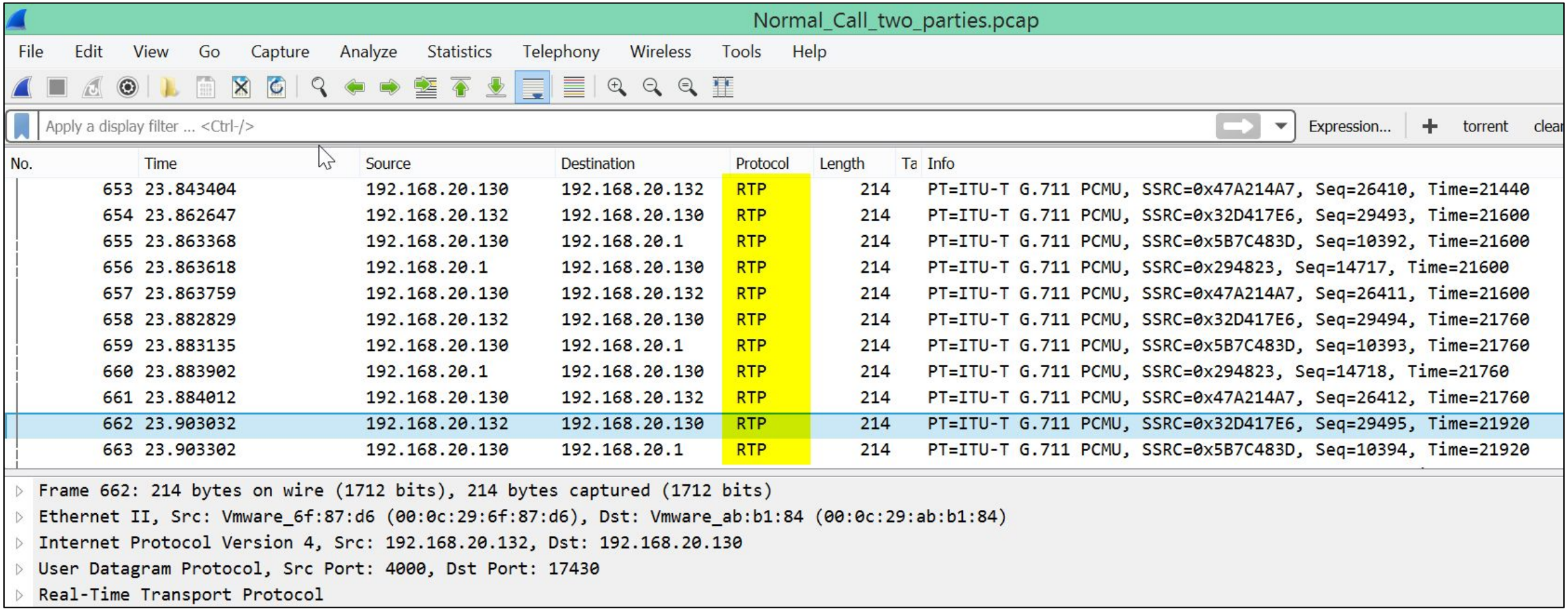
Decode As



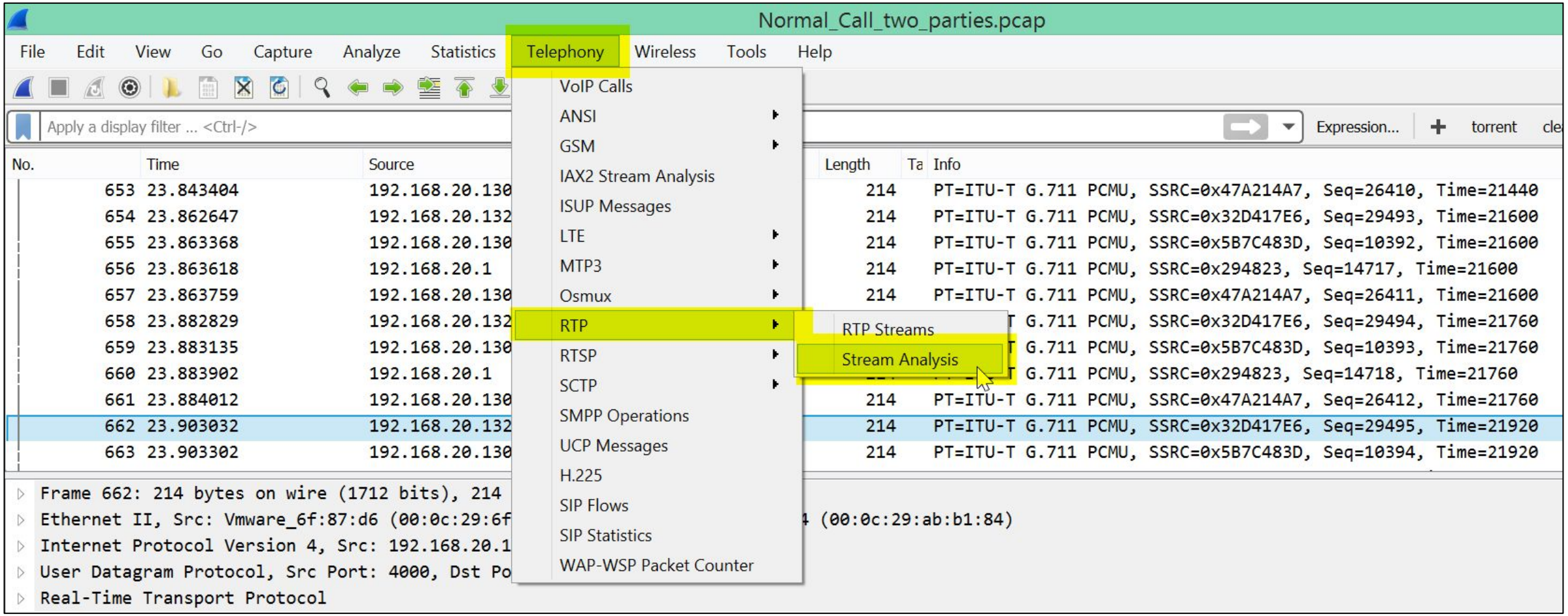
Decode As RTP



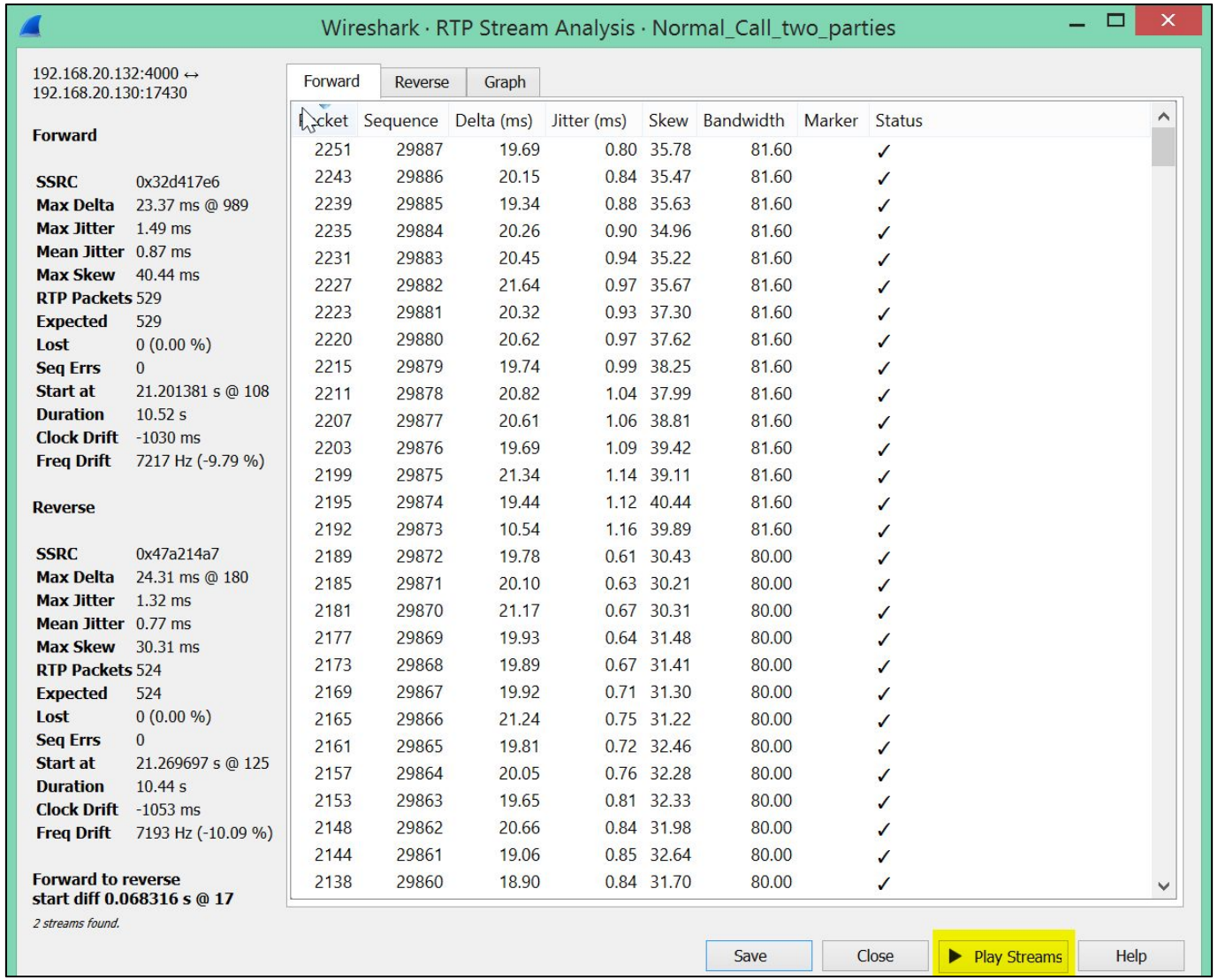
RTP Traffic



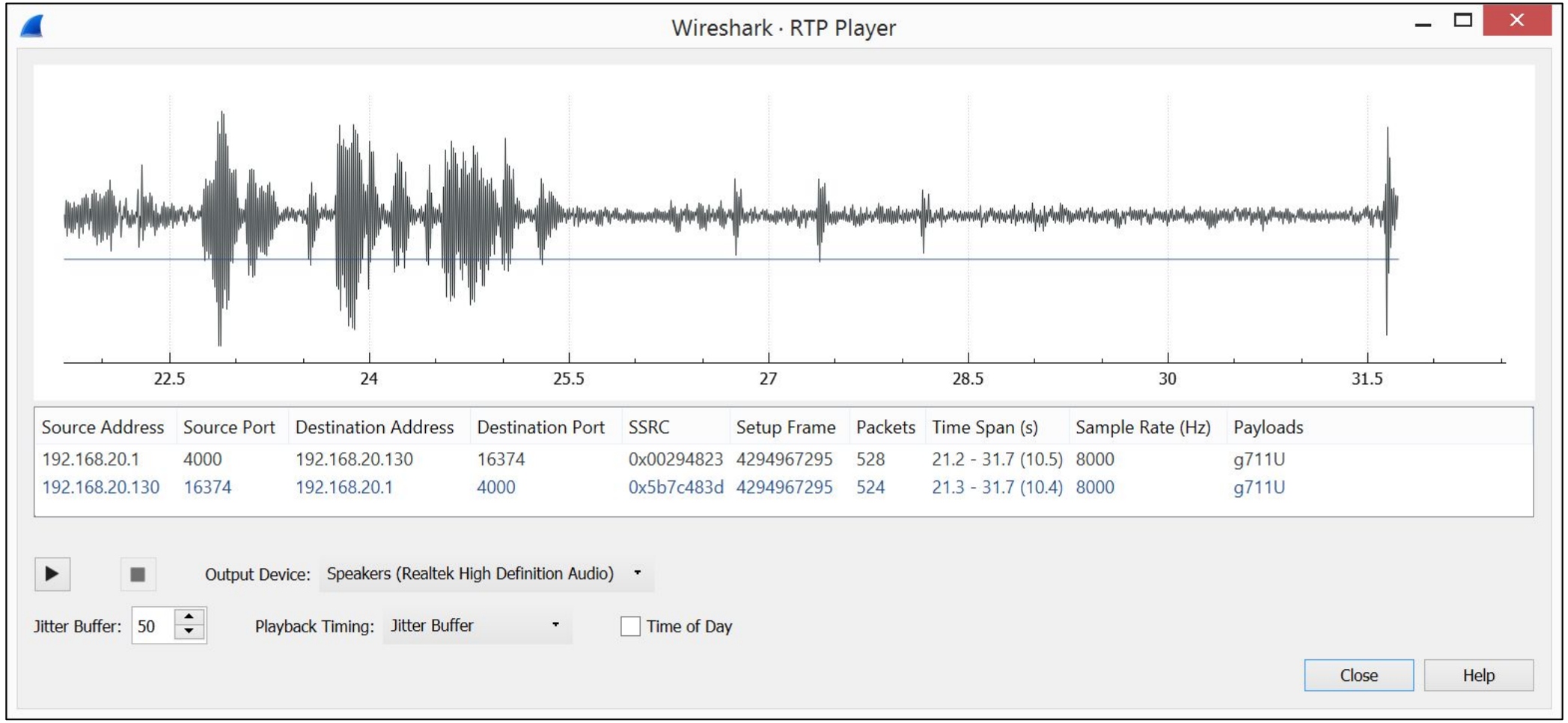
Checking RTP Streams



Analysing RTP Streams



Playing RTP Streams



Possible Configurations

- SIP + RTP
- SIP over TLS + RTP
- SIP + SRTP
- **SIP over TLS + SRTP**

TLS key exchange methods

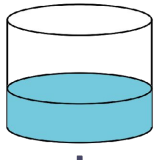
- TLS uses symmetric ciphers (i.e. AES, Blowfish) to encrypt the data

- **Two options under realistic approach**
  - DHE (Diffie Hellman Key Exchange)
  - RSA (Asymmetric encryption)

Diffie Hellman Exchange

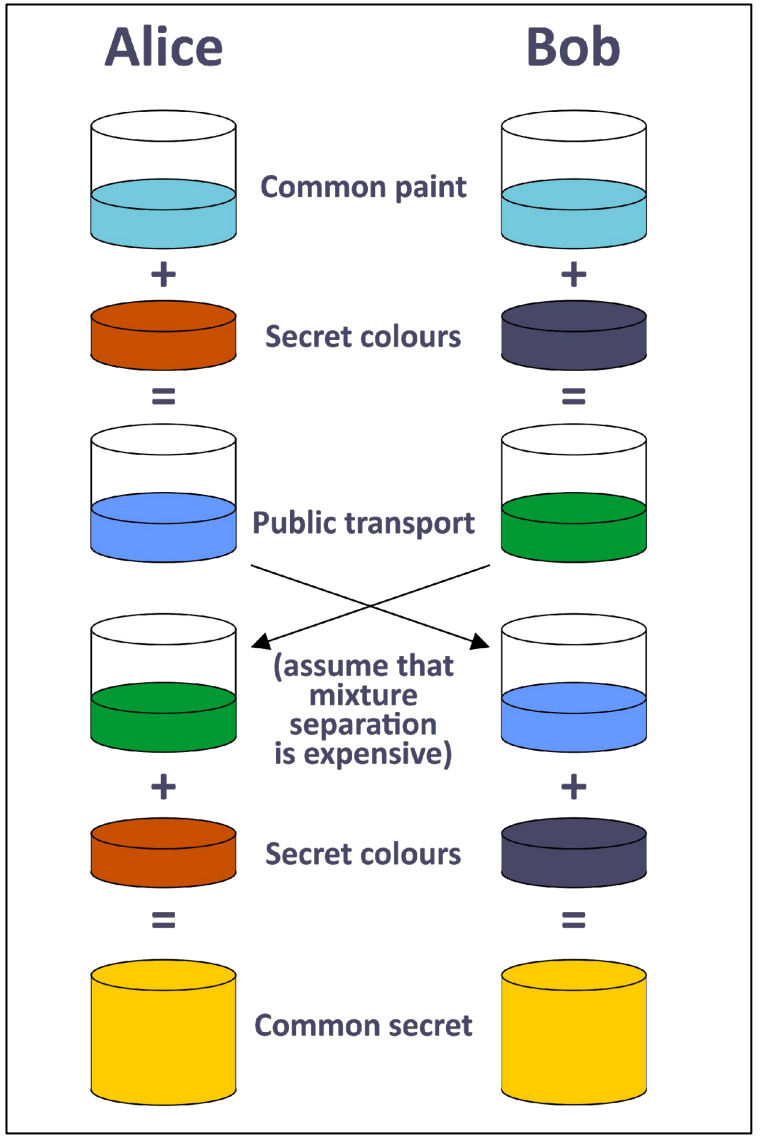
Assumption

- Attacker even after seeing the exchanged colours can't guess the secret colour.
- Attacker knows 

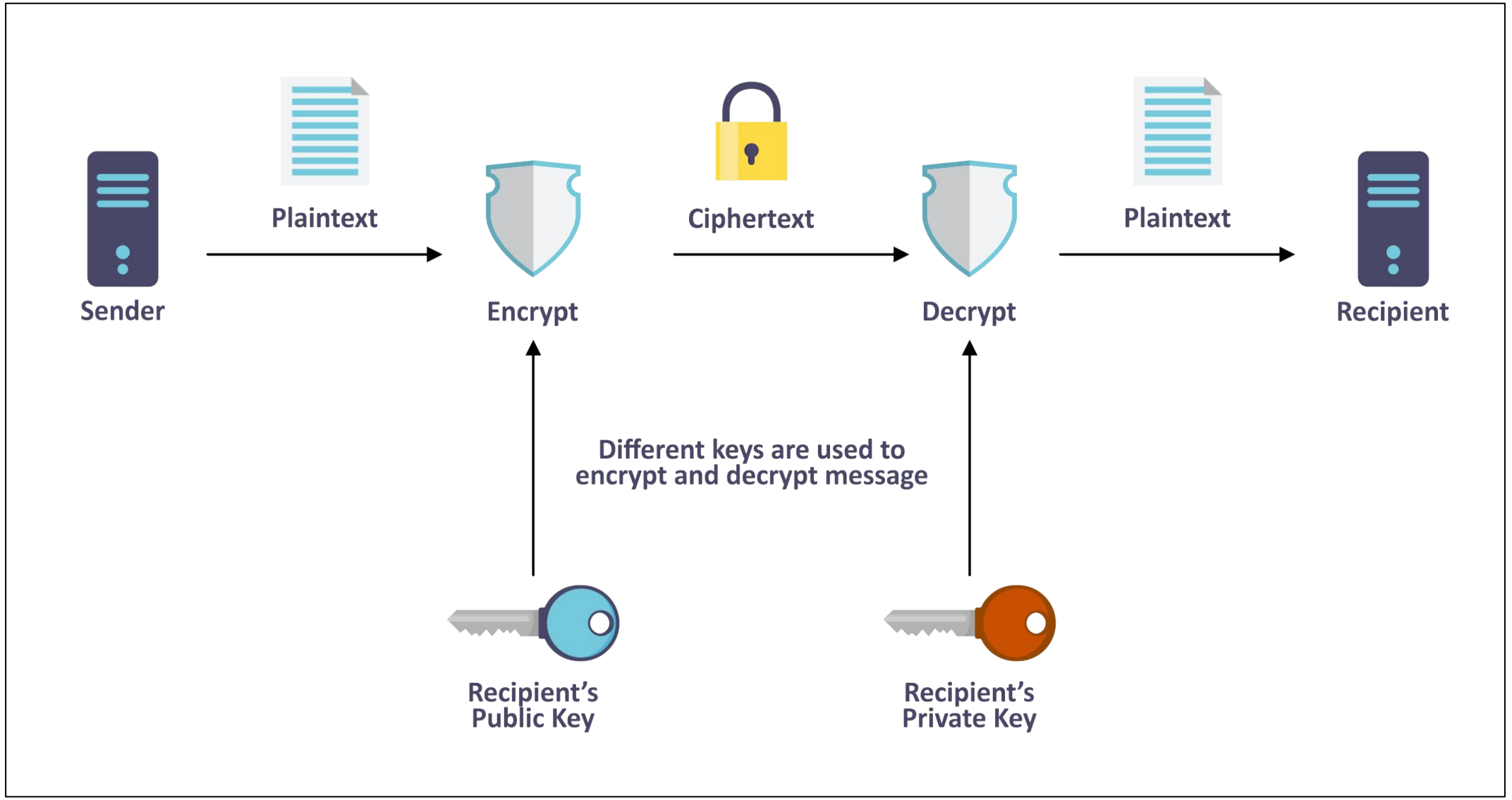
  and also 

  But can't know which colour is added.

More on: en.wikipedia.org/wiki/Diffie%E2%80%93Hellman_key_exchange



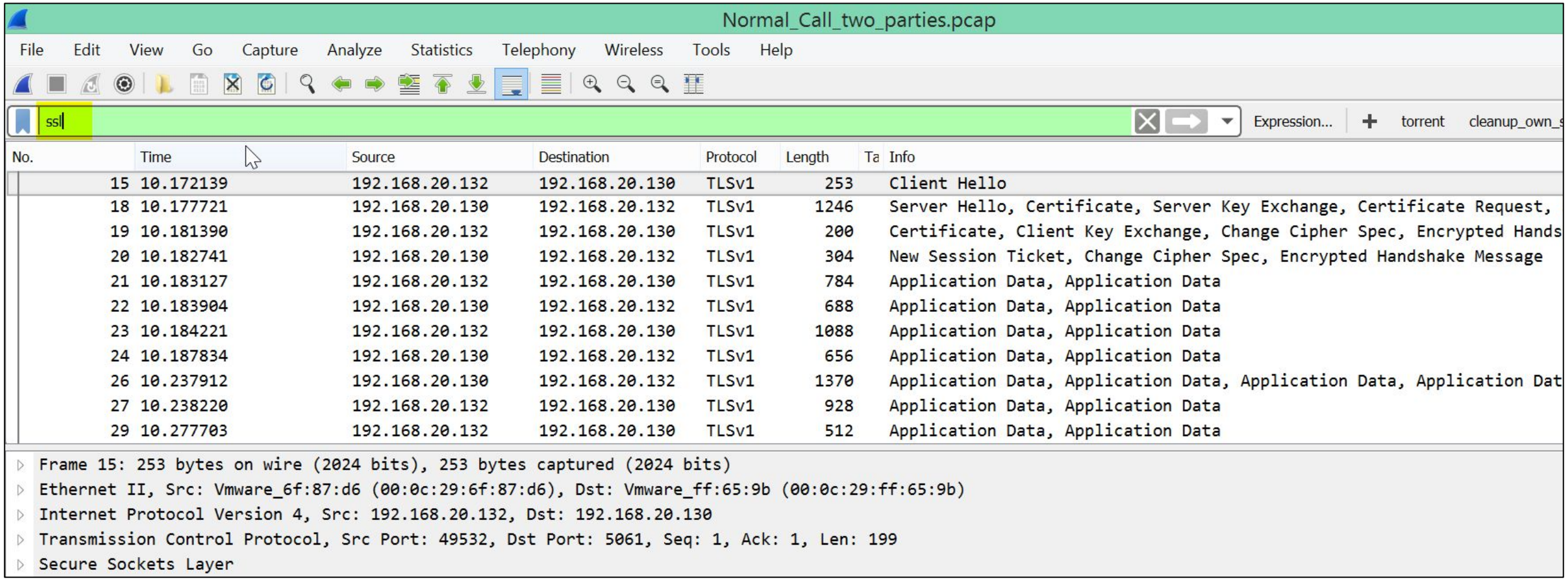
RSA (Asymmetric Encryption)



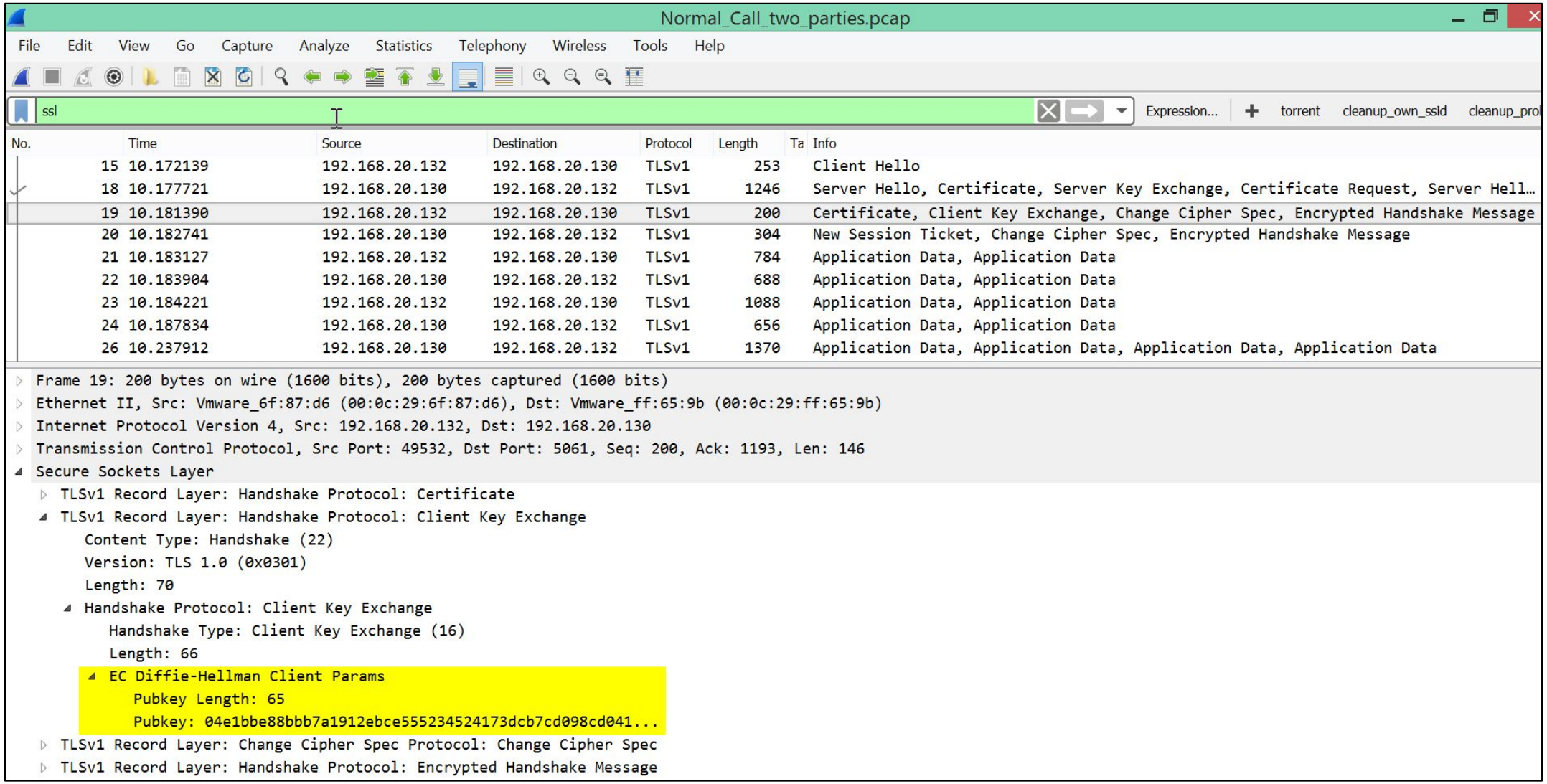
Observations?

- Can’t recover keys derived with **ECDHE/DHE** by listening to traffic
- For **RSA**, if we can get private key of server, we can decrypt traffic

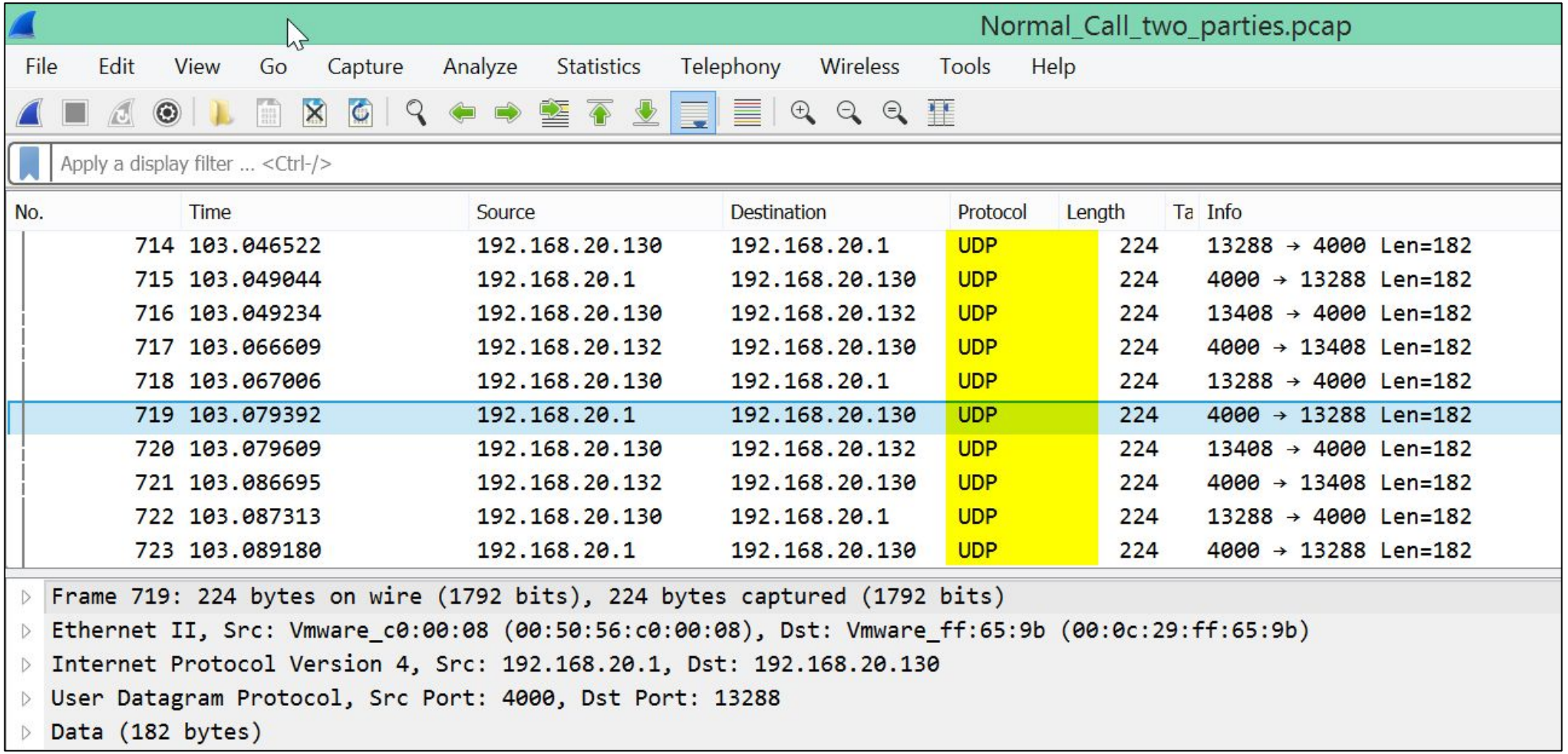
TLS Traffic (SIP over TLS)



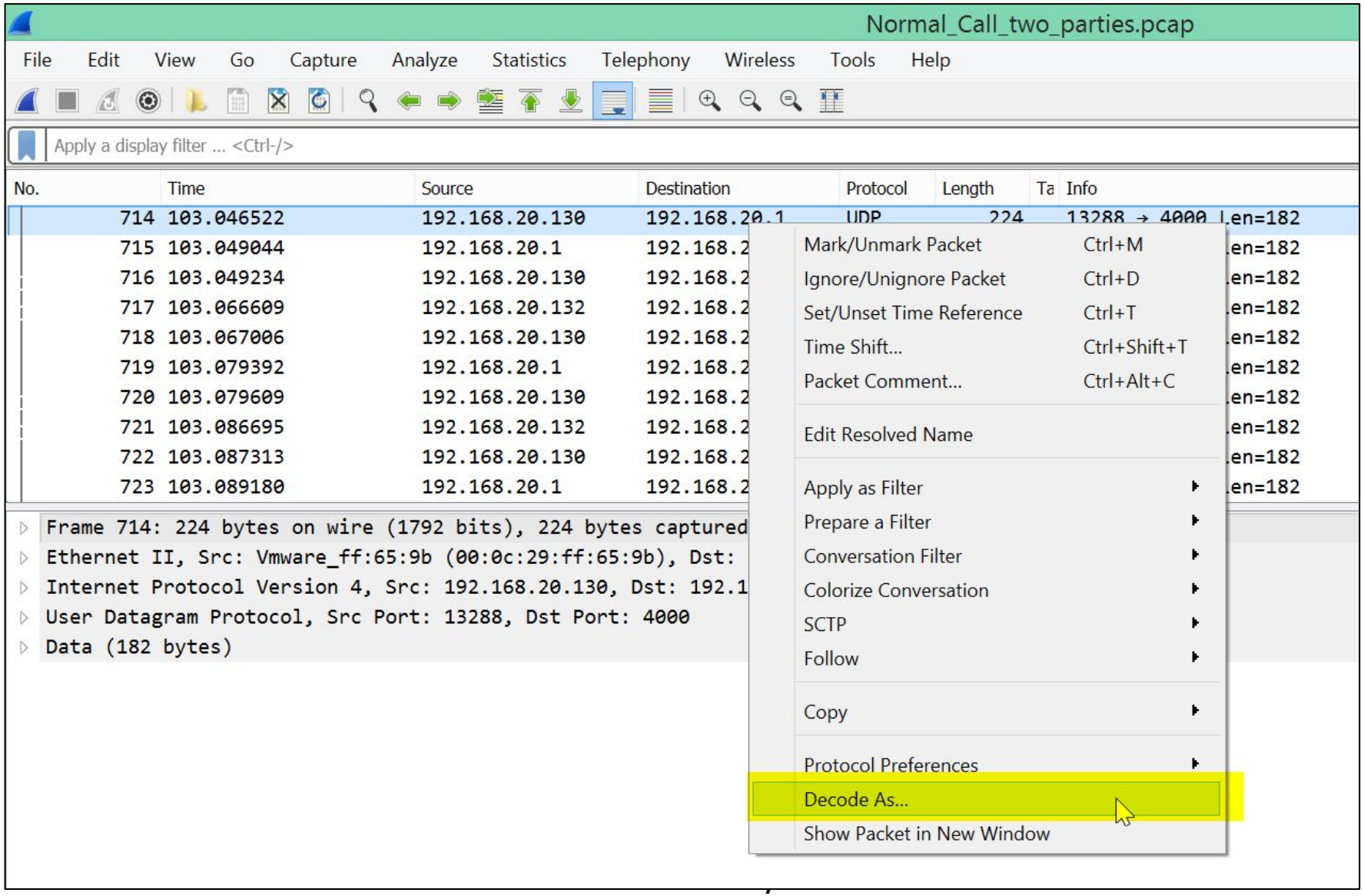
Diffie Hellman Exchange



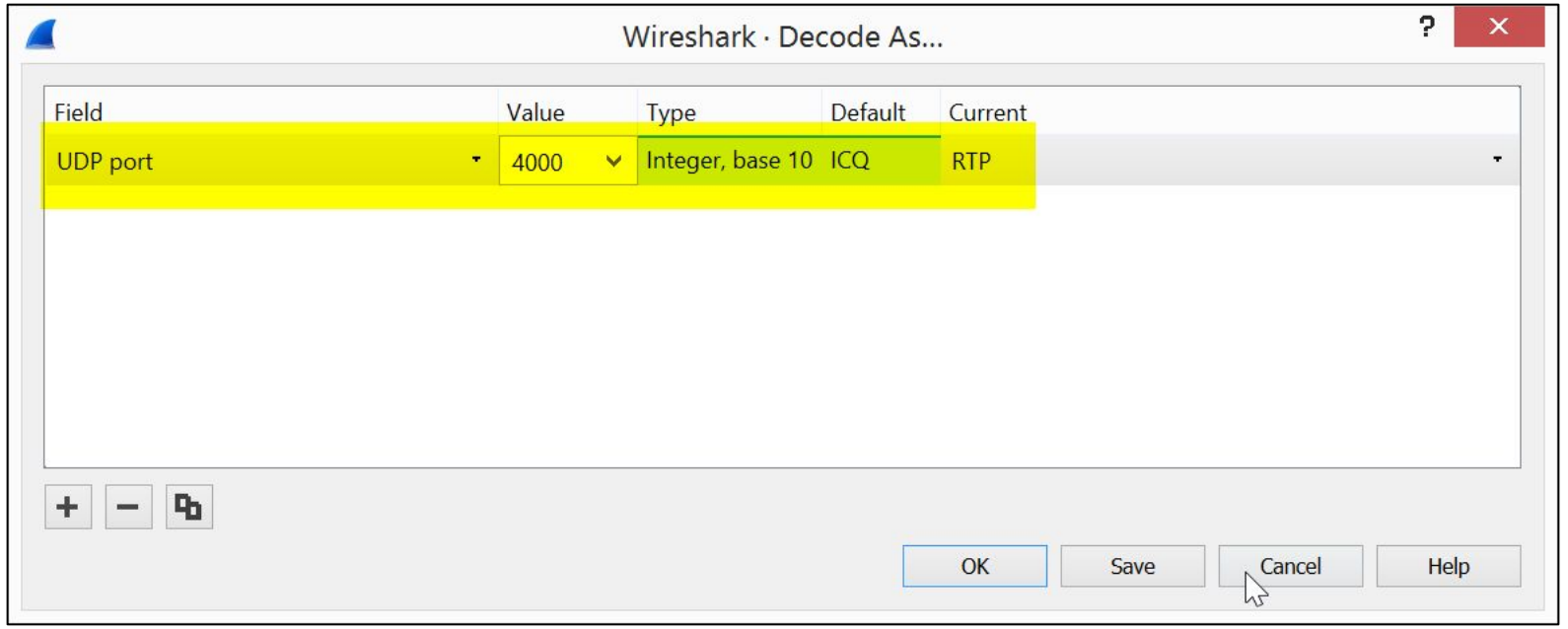
Undecoded SRTP Traffic



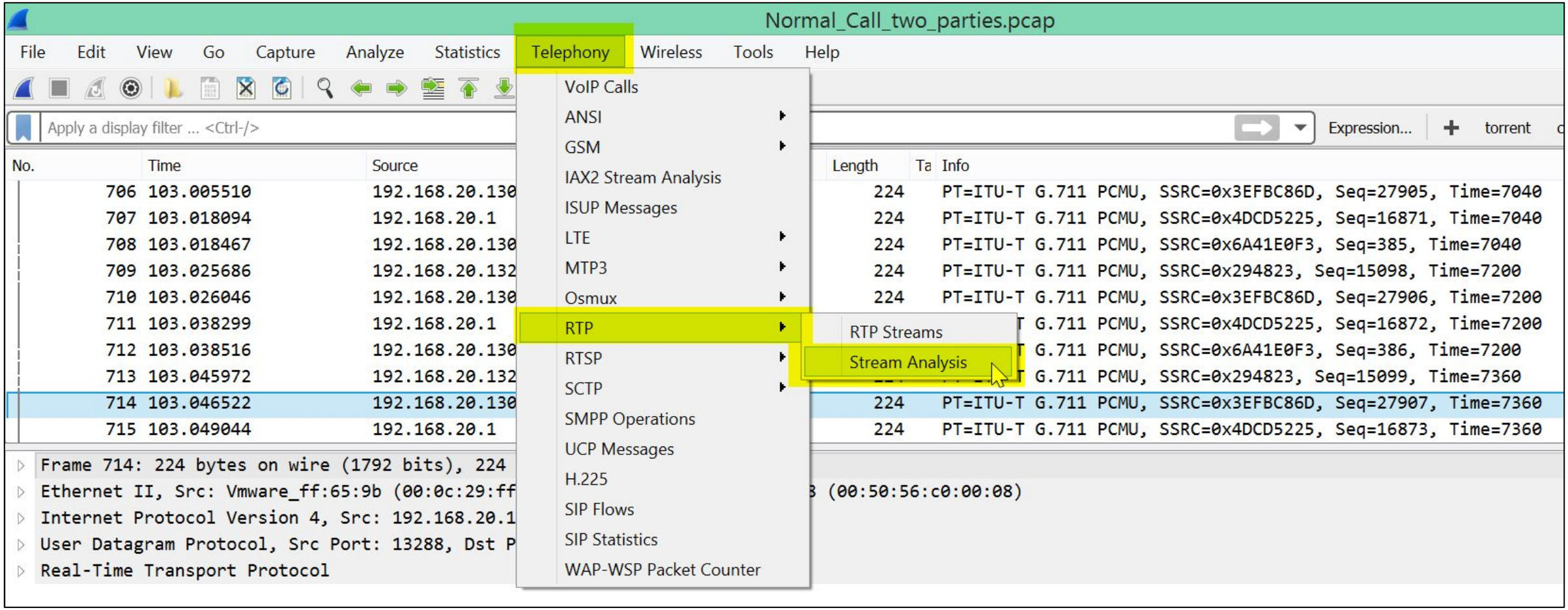
Decode As


Normal_Call_two_parties.pcap
File Edit View Go Capture Analyze Statistics Telephony Wireless Tools Help
Apply a display filter ... <Ctrl-/>
No. Time Source Destination Protocol Length Ta Info
714 103.046522 192.168.20.130 192.168.20.1 UDP 224 13288 → 4000 Len=182
715 103.049044 192.168.20.1 192.168.2 Len=182
716 103.049234 192.168.20.130 192.168.2 Len=182
717 103.066609 192.168.20.132 192.168.2 Len=182
718 103.067006 192.168.20.130 192.168.2 Len=182
719 103.079392 192.168.20.1 192.168.2 Len=182
720 103.079609 192.168.20.130 192.168.2 Len=182
721 103.086695 192.168.20.132 192.168.2 Len=182
722 103.087313 192.168.20.130 192.168.2 Len=182
723 103.089180 192.168.20.1 192.168.2 Len=182
Frame 714: 224 bytes on wire (1792 bits), 224 bytes captured
Ethernet II, Src: Vmware_ff:65:9b (00:0c:29:ff:65:9b), Dst:
Internet Protocol Version 4, Src: 192.168.20.130, Dst: 192.1
User Datagram Protocol, Src Port: 13288, Dst Port: 4000
Data (182 bytes)
Mark/Unmark Packet Ctrl+M
Ignore/Unignore Packet Ctrl+D
Set/Unset Time Reference Ctrl+T
Time Shift... Ctrl+Shift+T
Packet Comment... Ctrl+Alt+C
Edit Resolved Name
Apply as Filter
Prepare a Filter
Conversation Filter
Colorize Conversation
SCTP
Follow
Copy
Protocol Preferences
Decode As...
Show Packet in New Window

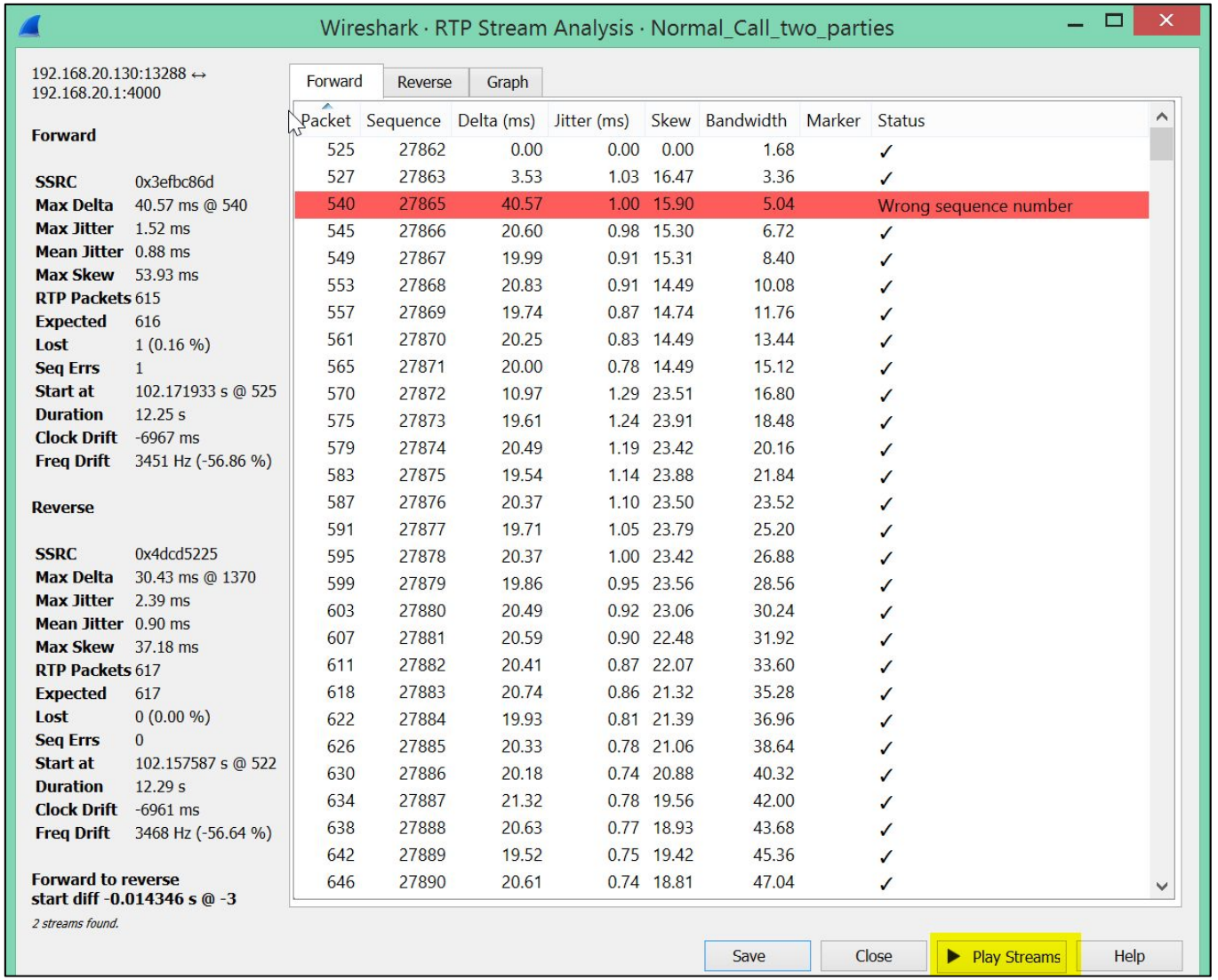
Decode As RTP



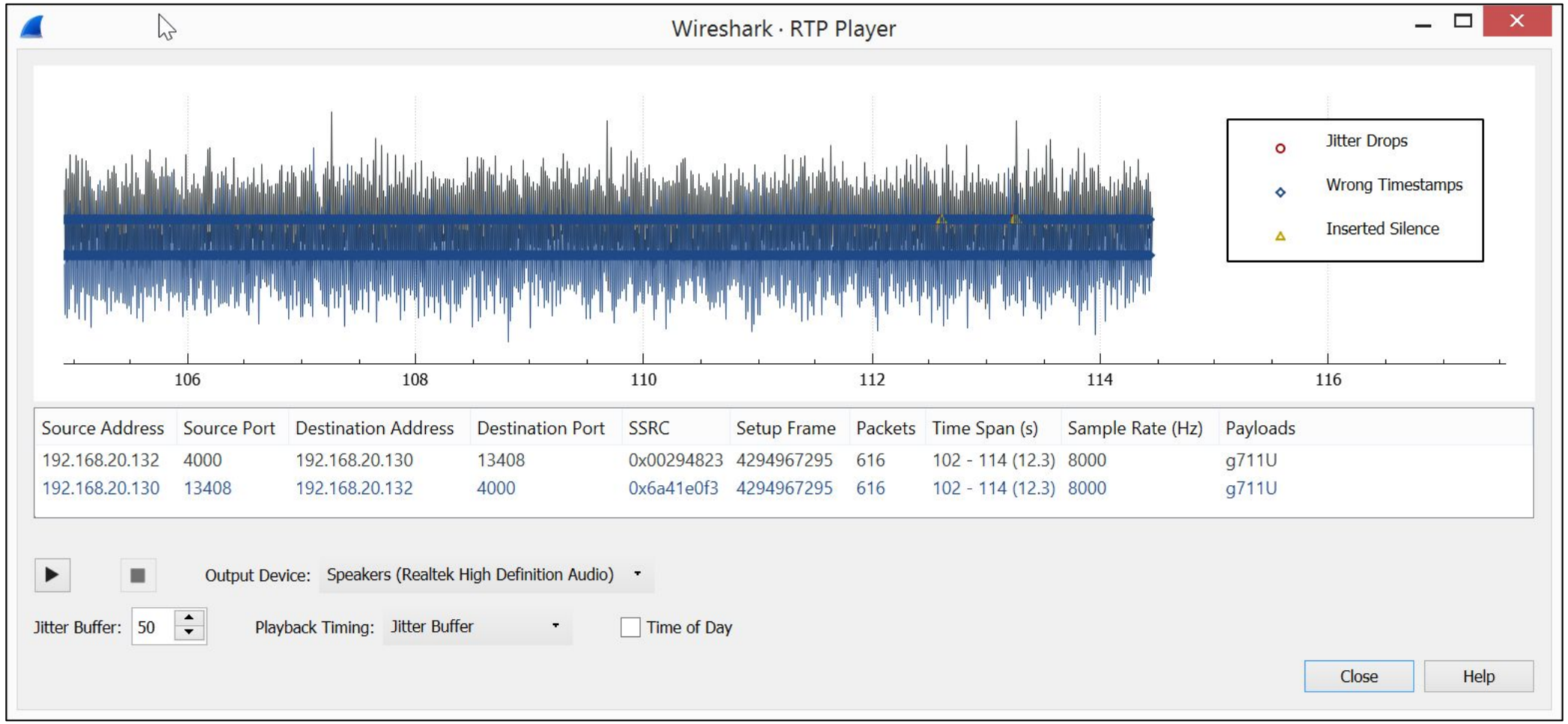
Checking RTP Streams



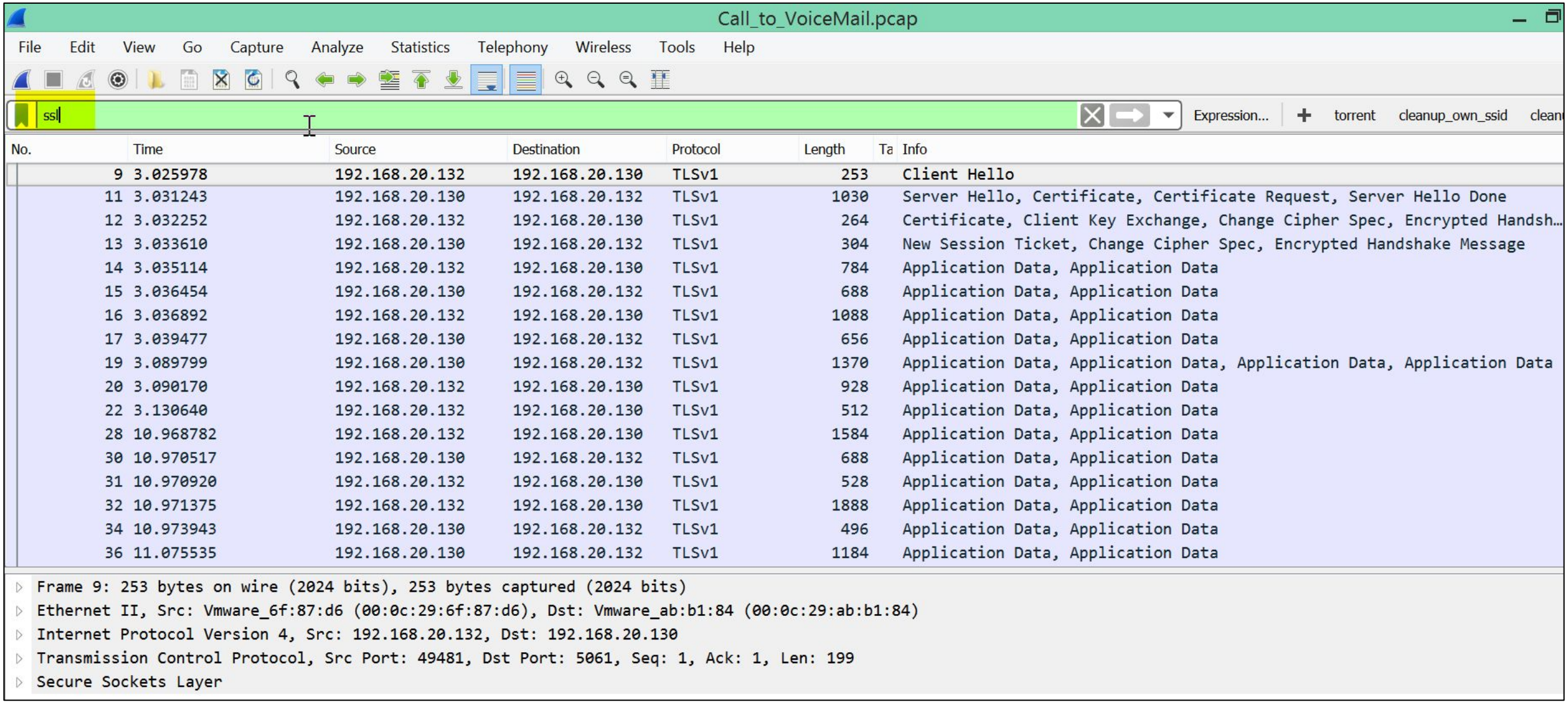
Analysing RTP Streams



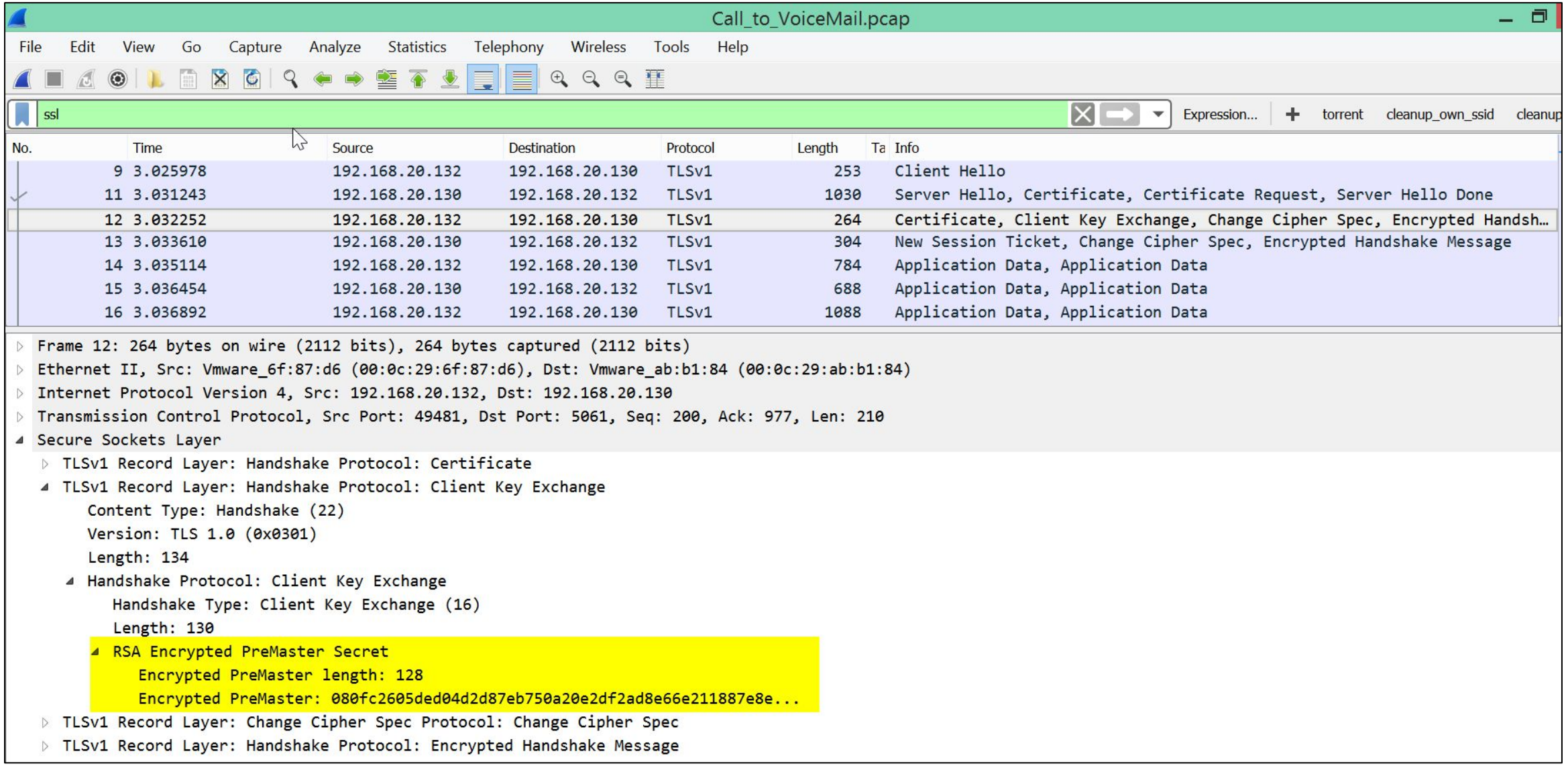
Playing RTP Steams



TLS Traffic (SIP over TLS)



RSA based key exchange

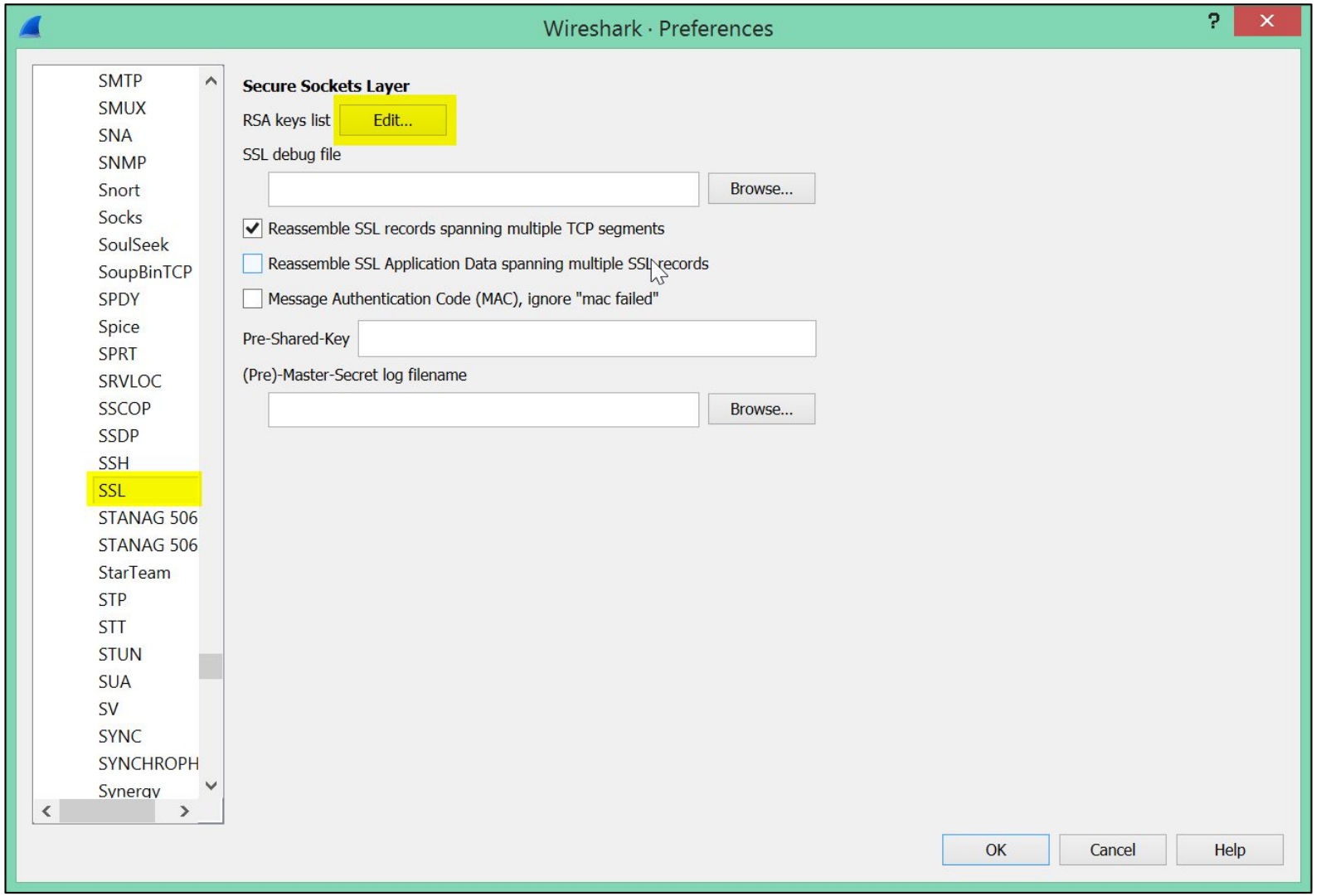


Decrypting TLS traffic

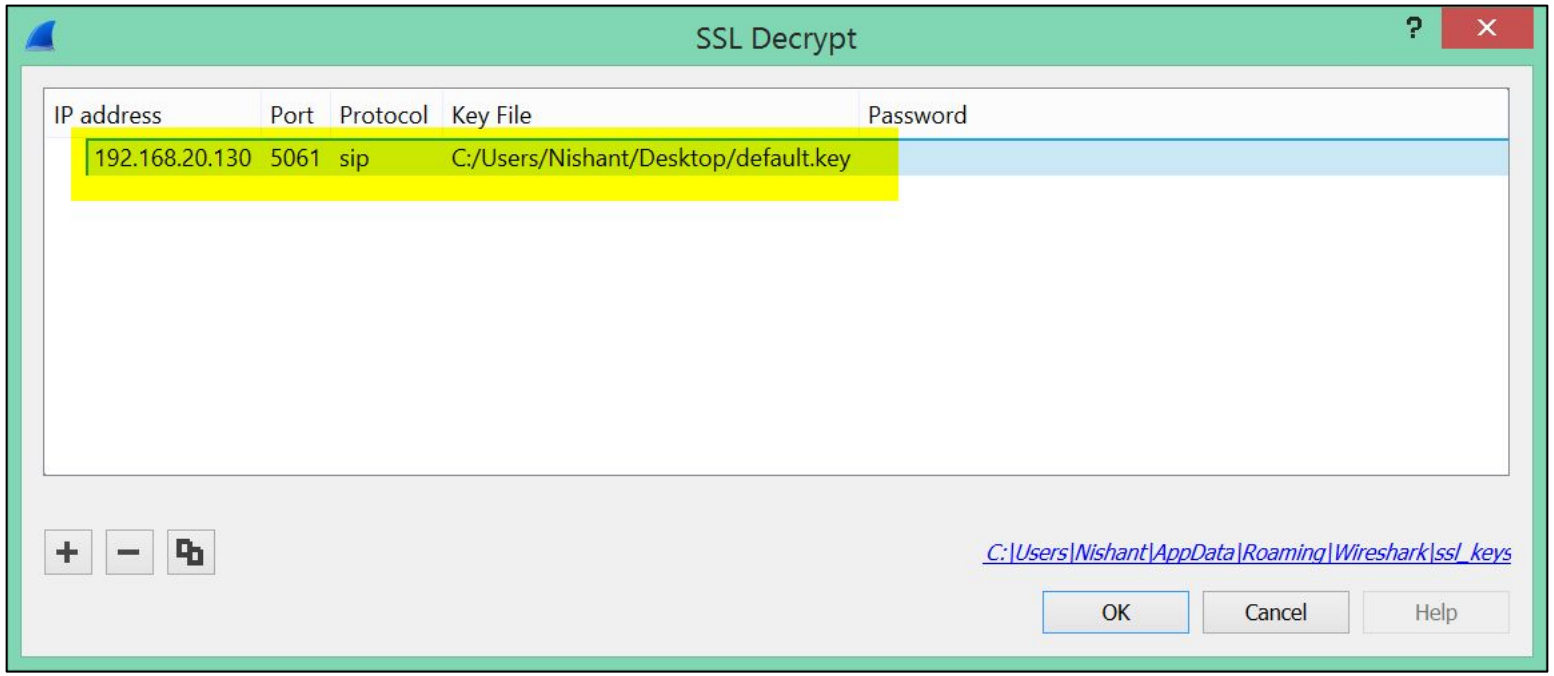
- RSA is used to exchange keys
- We can decrypt with private key installed on Asterisk One
- **Keys and certificate location on Asterisk One:** /etc/asterisk/keys
- We have to get the **default.key** from the server

```
[root@localhost ~]# cd /etc/asterisk/keys/
[root@localhost keys]# ls -l
total 32
-rw-rw-r--. 1 asterisk asterisk  215 Mar 19 03:59 ca.cfg
-rw-rw-r--. 1 asterisk asterisk 1789 Mar 19 03:59 ca.crt
-rw-rw-r--. 1 asterisk asterisk 3311 Mar 19 03:59 ca.key
-rw-------. 1 asterisk asterisk 1253 Mar 19 03:59 default.crt
-rw-------. 1 asterisk asterisk  595 Mar 19 03:59 default.csr
-rw-------. 1 asterisk asterisk  891 Mar 19 03:59 default.key
-rw-------. 1 asterisk asterisk 2144 Mar 19 03:59 default.pem
drwxrwxr-x. 2 asterisk asterisk 4096 Mar 19 03:59 integration
```

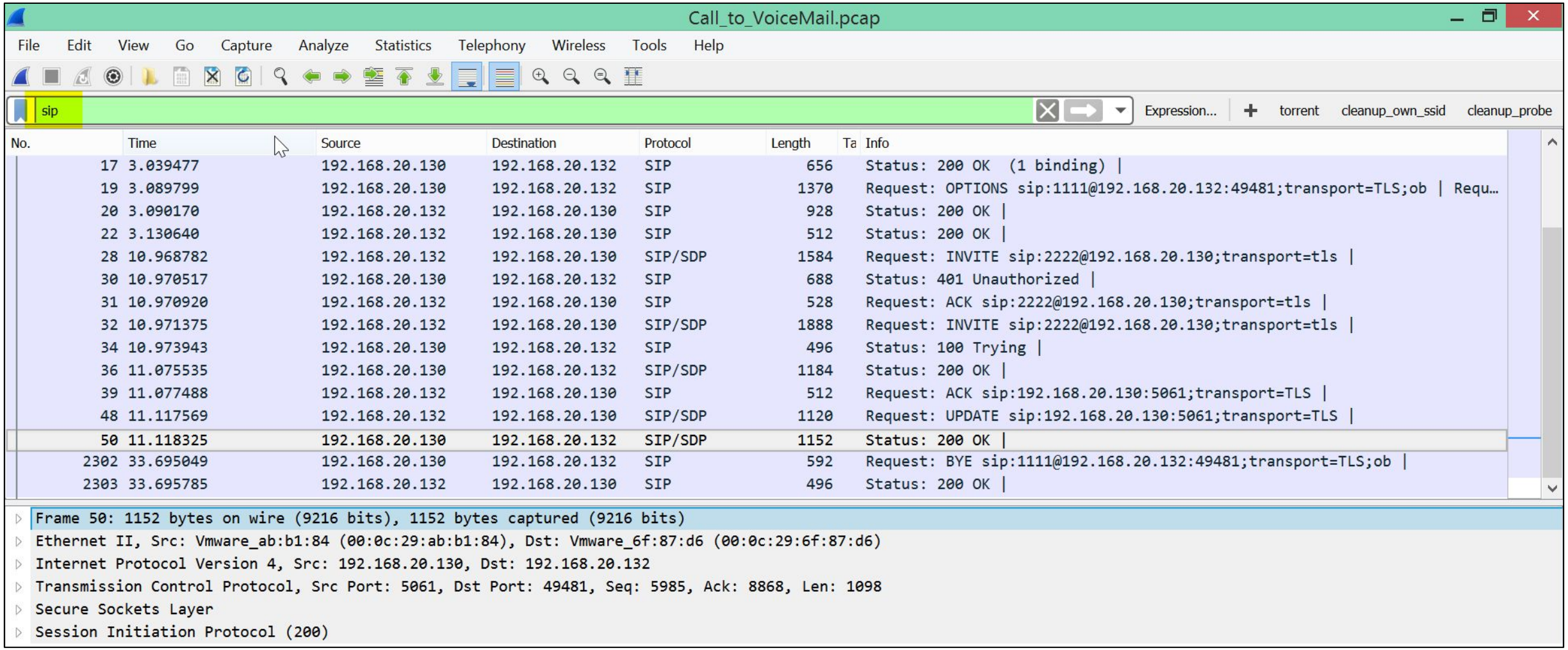
Edit > Preferences > Protocol > SSL



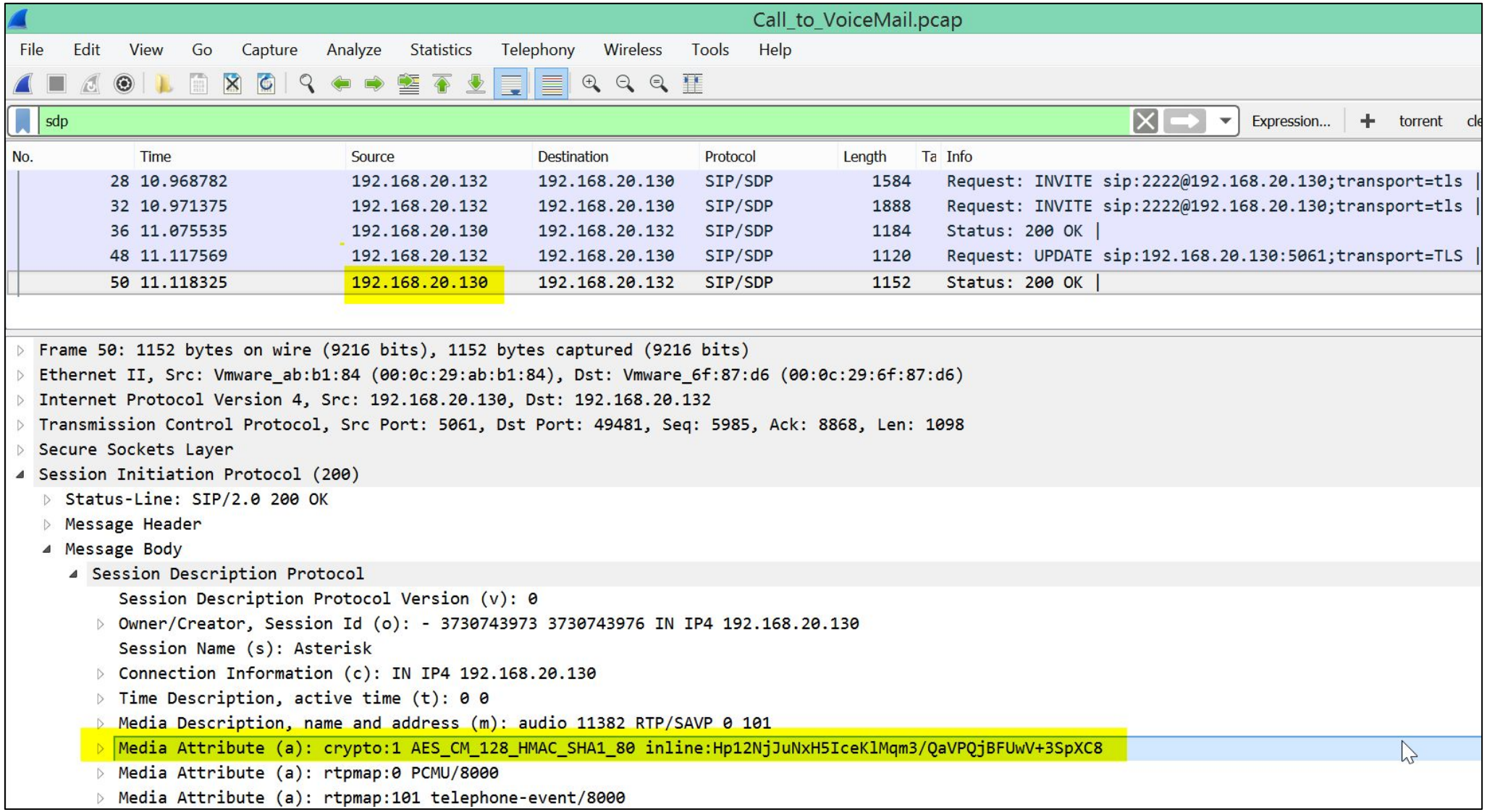
Adding Asterisk default private key



Decrypted SIP traffic



SRTP key in SIP/SDP decrypted packet



Open Source Tools for Decrypting SRTP

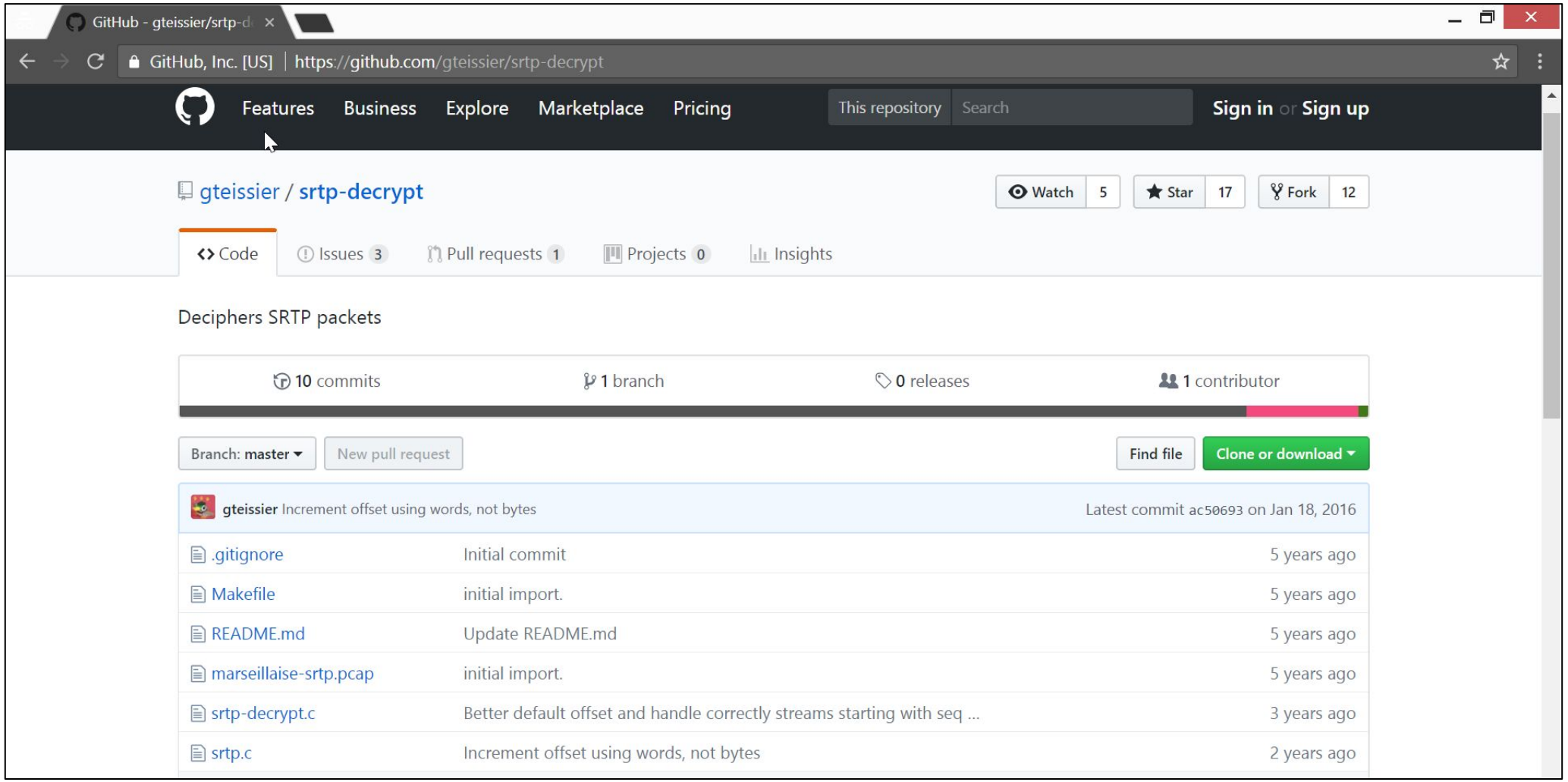
- SRTP Decrypt
- Libsrtp

SRTP Decrypt

- Tool to decipher SRTP packets
- Takes symmetric key to decrypt the SRTP traffic
- Output decrypted packets in form of hexdump
- Wireshark can reconstruct RTP packets from the hexdump

SRTP Decrypt

- GitHub: github.com/gteissier/srtp-decrypt



SRTP Decrypt: Pre-Installation

- Installing libgcrypt

```
pentester@PentesterAcademy:~/work/srtp-decrypt$ sudo apt-get install libgcrypt-dev
sudo: unable to resolve host PentesterAcademy
Reading package lists... Done
Building dependency tree
Reading state information... Done
Note, selecting 'libgcrypt20-dev' instead of 'libgcrypt-dev'
The following additional packages will be installed:
  libgcrypt20 libgpg-error-dev
Suggested packages:
```

- Installing libpcap

```
pentester@PentesterAcademy:~/work/srtp-decrypt$ sudo apt-get install libpcap-dev
sudo: unable to resolve host PentesterAcademy
Reading package lists... Done
Building dependency tree
Reading state information... Done
The following additional packages will be installed:
  libpcap0.8-dev
The following NEW packages will be installed:
  libpcap-dev libpcap0.8-dev
```

SRTP Decrypt: Installation

- Cloning

```
root@PentesterAcademy:/work# git clone https://github.com/gteissier/srtp-decrypt.git
Cloning into 'srtp-decrypt'...
remote: Counting objects: 35, done.
remote: Total 35 (delta 0), reused 0 (delta 0), pack-reused 35
Unpacking objects: 100% (35/35), done.
```

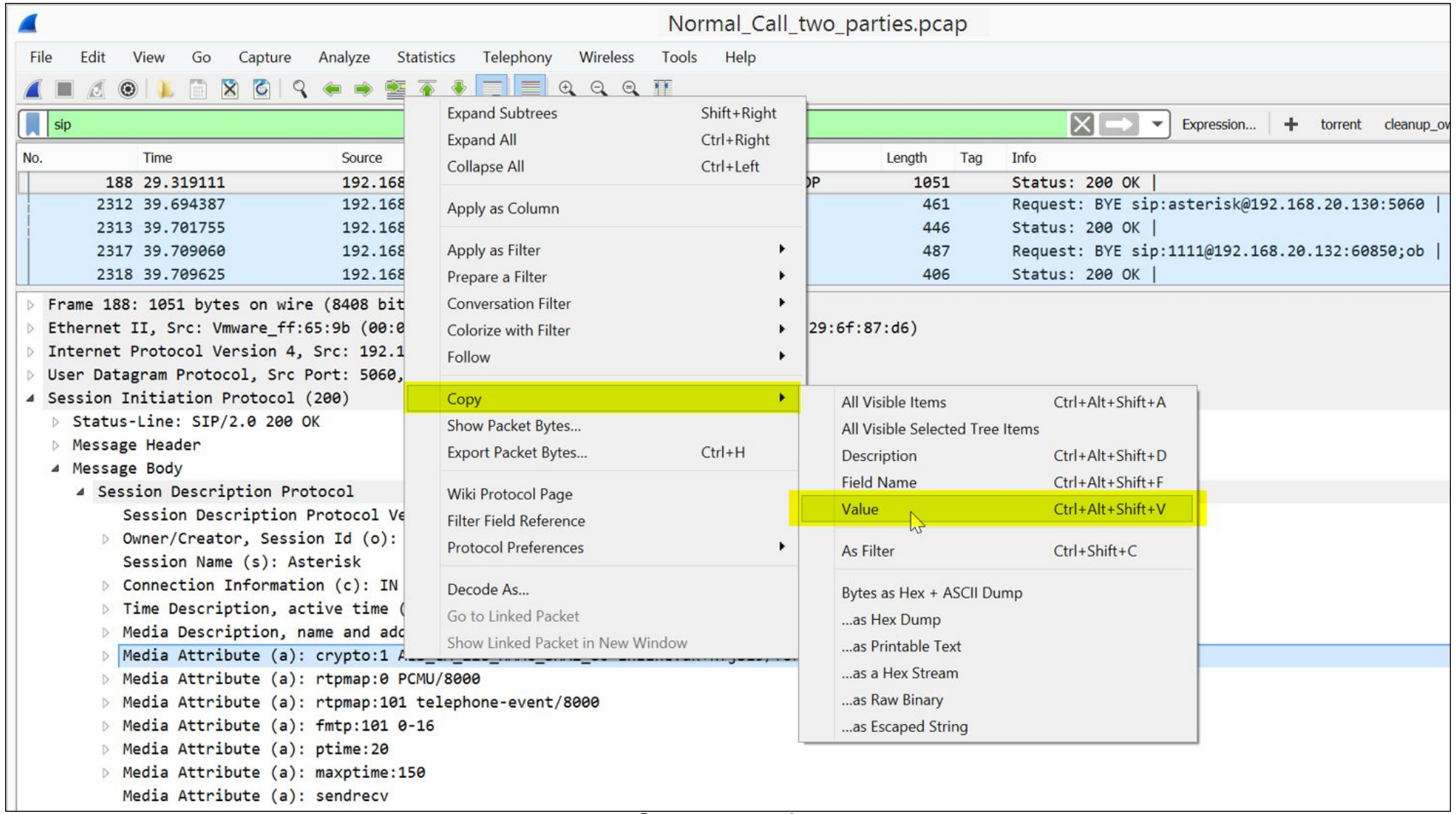
- Compiling

```
root@PentesterAcademy:/work/srtp-decrypt# make
cc -g -Os -Wall   -c -o srtp.o srtp.c
cc -g -Os -Wall   -c -o srtp-decrypt.o srtp-decrypt.c
cc -o srtp-decrypt srtp-decrypt.o srtp.o -lpcap -lgcrypt
```

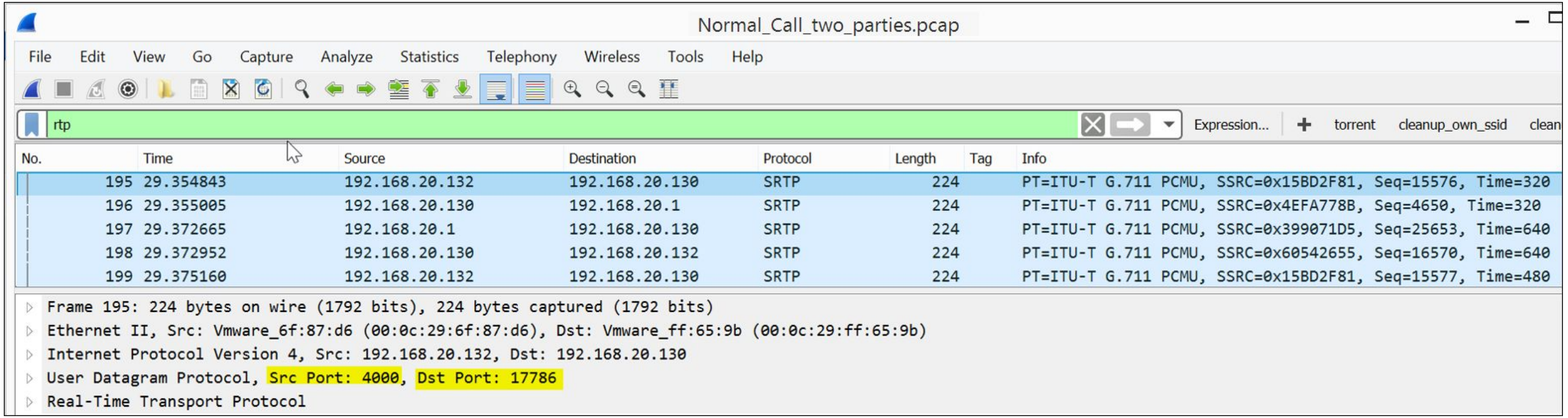
SRTP Decrypt: Ready

```
root@PentesterAcademy:/work/srtp-decrypt# ls -l
total 2964
-rw-r--r-- 1 root root     273 Mar 17 05:36 Makefile
-rw-r--r-- 1 root root 2853144 Mar 17 05:36 marseillaise-srtp.pcap
-rw-r--r-- 1 root root     945 Mar 17 05:36 README.md
-rw-r--r-- 1 root root   22057 Mar 17 05:36 srtp.c
-rwxr-xr-x 1 root root   54112 Mar 17 05:40 srtp-decrypt
-rw-r--r-- 1 root root    3917 Mar 17 05:36 srtp-decrypt.c
-rw-r--r-- 1 root root   26464 Mar 17 05:40 srtp-decrypt.o
-rw-r--r-- 1 root root    2720 Mar 17 05:36 srtp.h
-rw-r--r-- 1 root root   52096 Mar 17 05:40 srtp.o
```

SRTP Decrypt: Copying SRTP key



SRTP Decrypt: UDP Ports



SRTP Decrypt: Decrypting SRTP Traffic

Command: ./srtp-decrypt -k uK+RfjSi9/fUFr8zoJu6zdqPw6MGtONhgX4yqwRj < ../Normal_Call_two_parties.pcap > decoded.raw

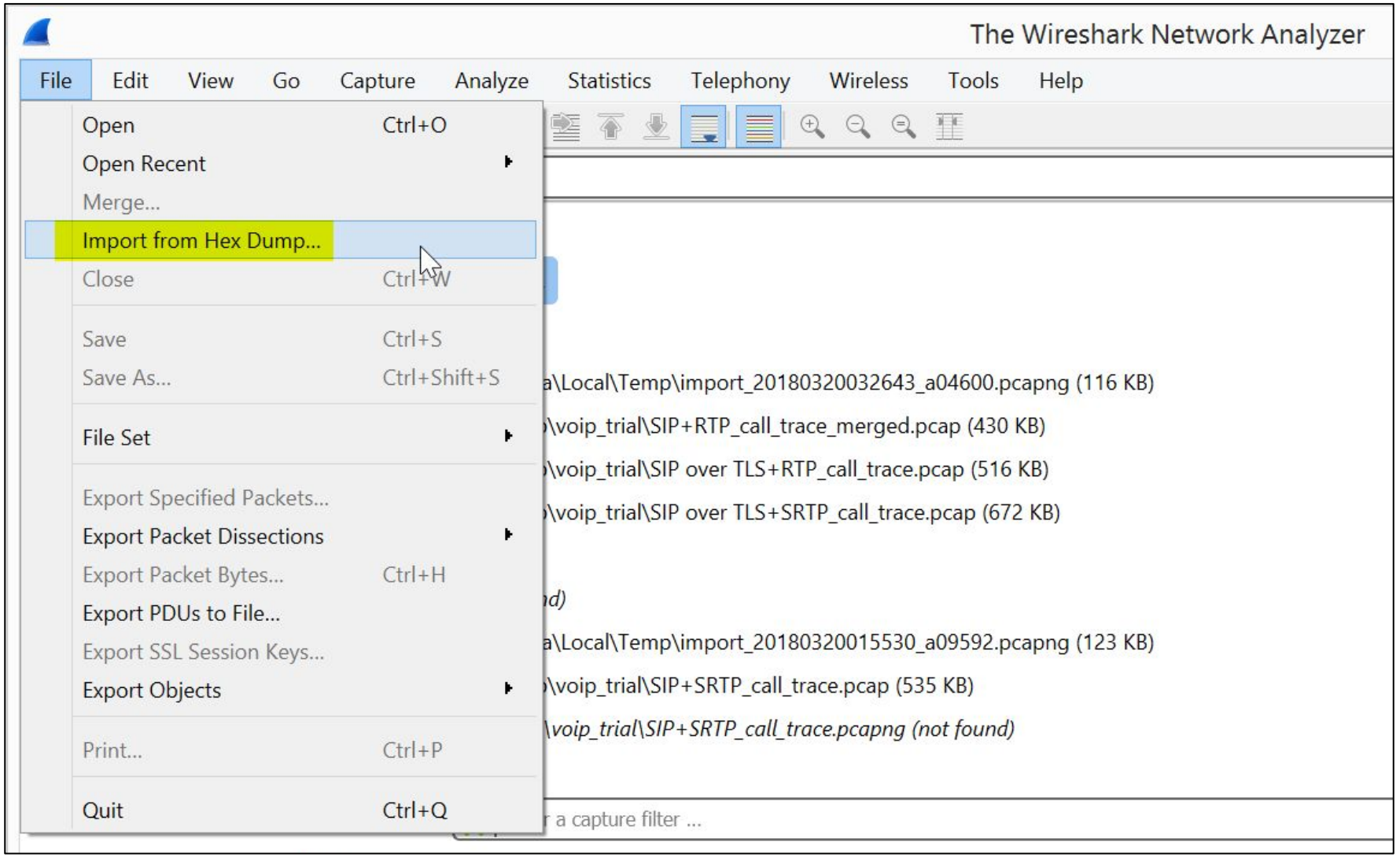
- -k : Defined SRTP key (uK+RfjSi9/fUFr8zoJu6zdqPw6MGtONhgX4yqwRj in this case)
- Normal_Call_two_parties.pcap Input file
- decoded.raw Output file

```
root@PentesterAcademy:/work/srtp-decrypt# ./srtp-decrypt -k uK+RfjSi9/fUFr8zoJu6zdqPw6M
GtONhgX4yqwRj < ../Normal_Call_two_parties.pcap > decoded.raw
frame 0 dropped: decoding failed 'Permission denied'
frame 1 dropped: decoding failed 'Permission denied'
frame 2 dropped: decoding failed 'Permission denied'
frame 3 dropped: decoding failed 'Permission denied'
frame 4 dropped: decoding failed 'Permission denied'
frame 5 dropped: decoding failed 'Permission denied'
frame 6 dropped: decoding failed 'Permission denied'
frame 7 dropped: decoding failed 'Permission denied'
frame 8 dropped: decoding failed 'Permission denied'
```

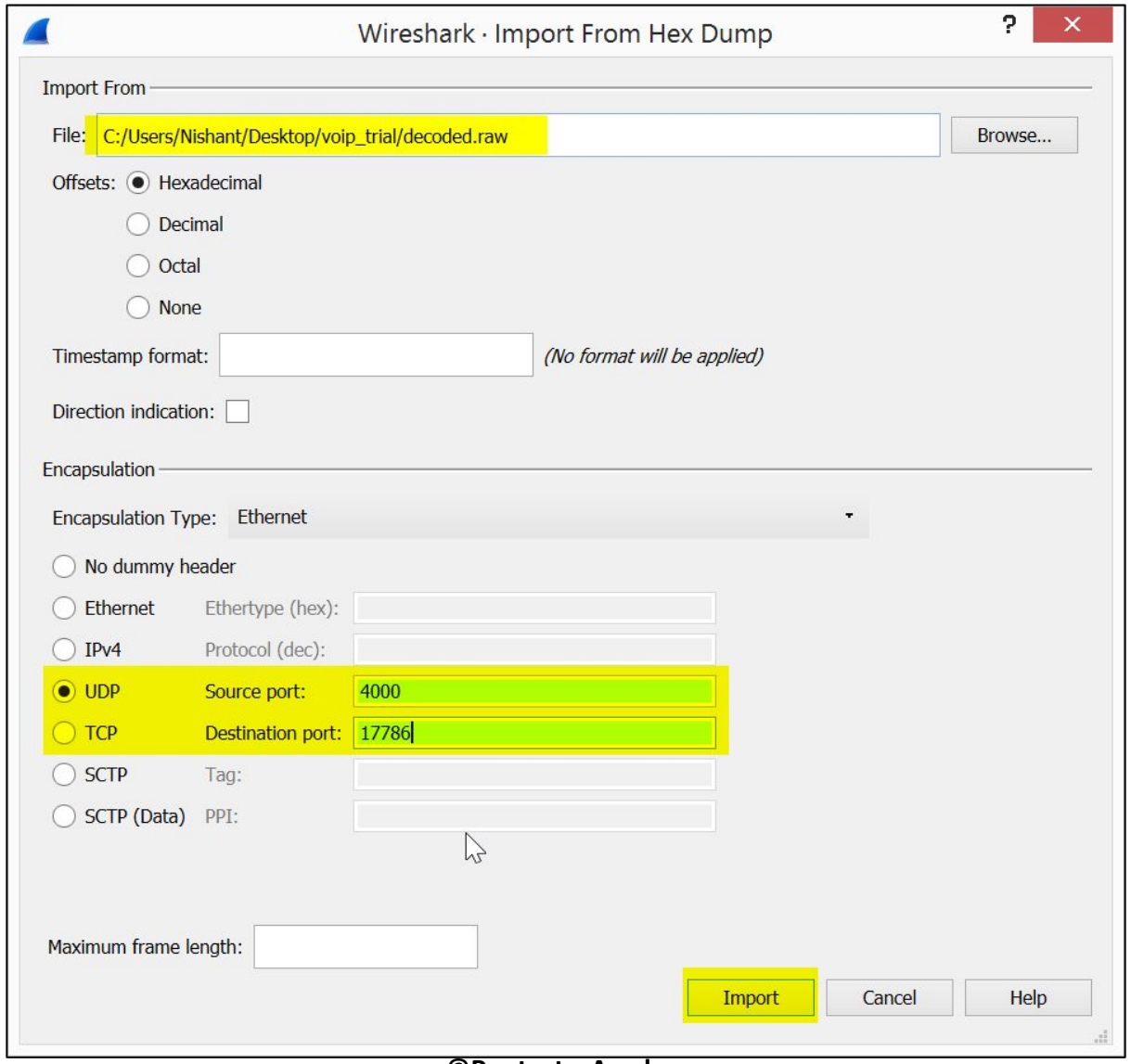
SRTP Decrypt: decoded.raw

```
 1 00:08.731764
 2 0000 80 00 64 2e 00 00 00 a0 58 2f 39 0c 7e 7e 7e 7e
 3 0010 7e 7e ff ff ff ff ff ff ff ff ff fe fe fe fe fe
 4 0020 fe ff ff ff 7e 7e 7e ff ff ff ff 7e 7e 7e 7e 7e
 5 0030 7e 7e 7e 7e 7e 7e 7e 7e 7e 7e 7e 7e 7e ff ff ff
 6 0040 ff 7e 7e 7e ff ff fe fe fe fe fe fe fe fe fe fe fe
 7 0050 fe fe fe fe ff ff 7e 7e 7e 7e 7e 7e 7e 7e 7e 7e
 8 0060 7e 7e 7e 7e 7e 7e 7e ff ff ff 7e 7e 7e 7e 7e 7e
 9 0070 7e 7e 7e ff ff ff ff fe fe fe fe fe fe fe fe fe ff
10 0080 7e 7e 7e 7e 7d 7d 7d 7d 7e 7e ff fe fe fe fe fe
11 0090 fe fe fe fe fe ff ff 7e 7e 7e 7d 7d 7d 7d 7d 7e
12 00a0 7e 7e ff ff ff fe fe fe fe fe fe fe
13 00:08.752171
14 0000 80 00 64 2f 00 00 01 40 58 2f 39 0c fe ff ff 7e
15 0010 7e 7e 7e 7d 7d 7d 7d 7d 7e 7e ff fe fe fe fd fd
16 0020 fd fe fe fe fe ff ff 7e 7e 7e 7e 7e 7e 7e 7e ff
17 0030 ff ff fe ff ff fe ff 7e 7e 7e 7e 7e 7e 7e 7e 7e
18 0040 7e ff ff ff fe fe fe ff ff 7e 7e 7e 7e 7d 7d 7d
19 0050 7d 7e 7e ff fe fe fd fd fd fd fe fe fe fe ff ff
20 0060 7e 7e 7e 7e 7e 7e 7e 7e ff ff ff ff ff ff fe fe
21 0070 fe ff ff 7e ff 7e 7e 7e 7e 7e 7e 7e 7e 7e ff ff
"decoded.raw" 12838 lines --0%--
```

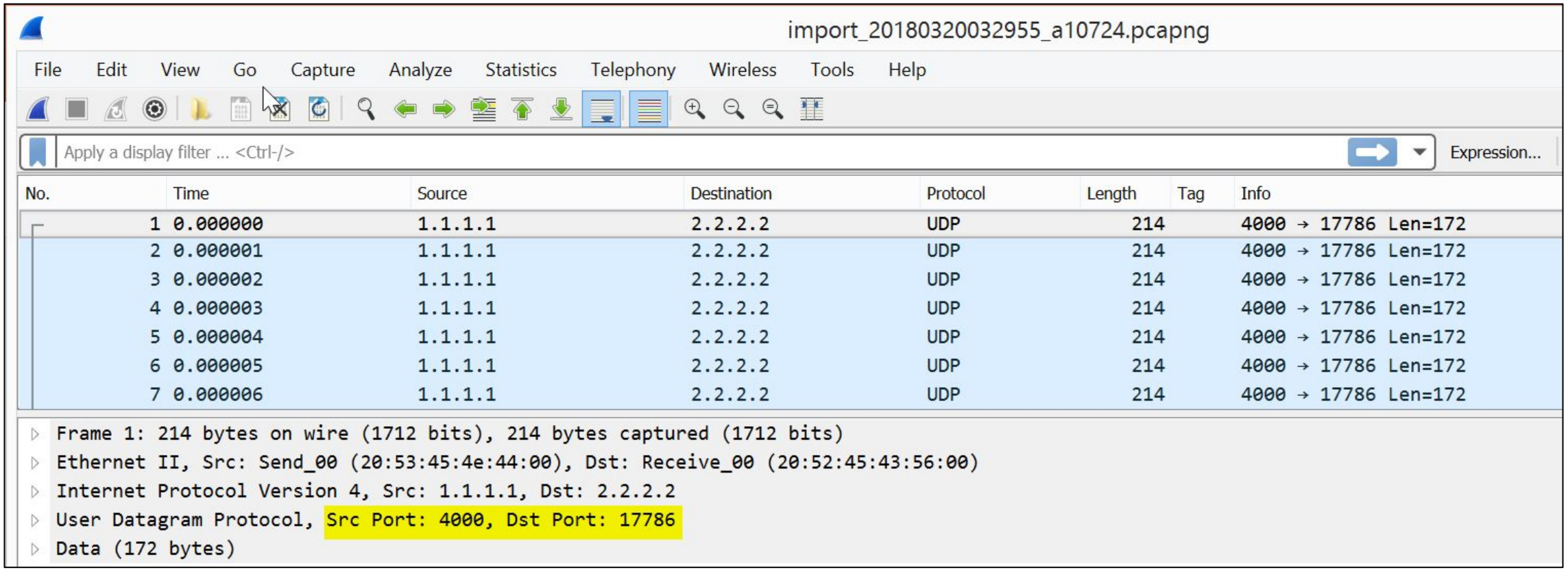
SRTP Decrypt: Importing Decrypted Content



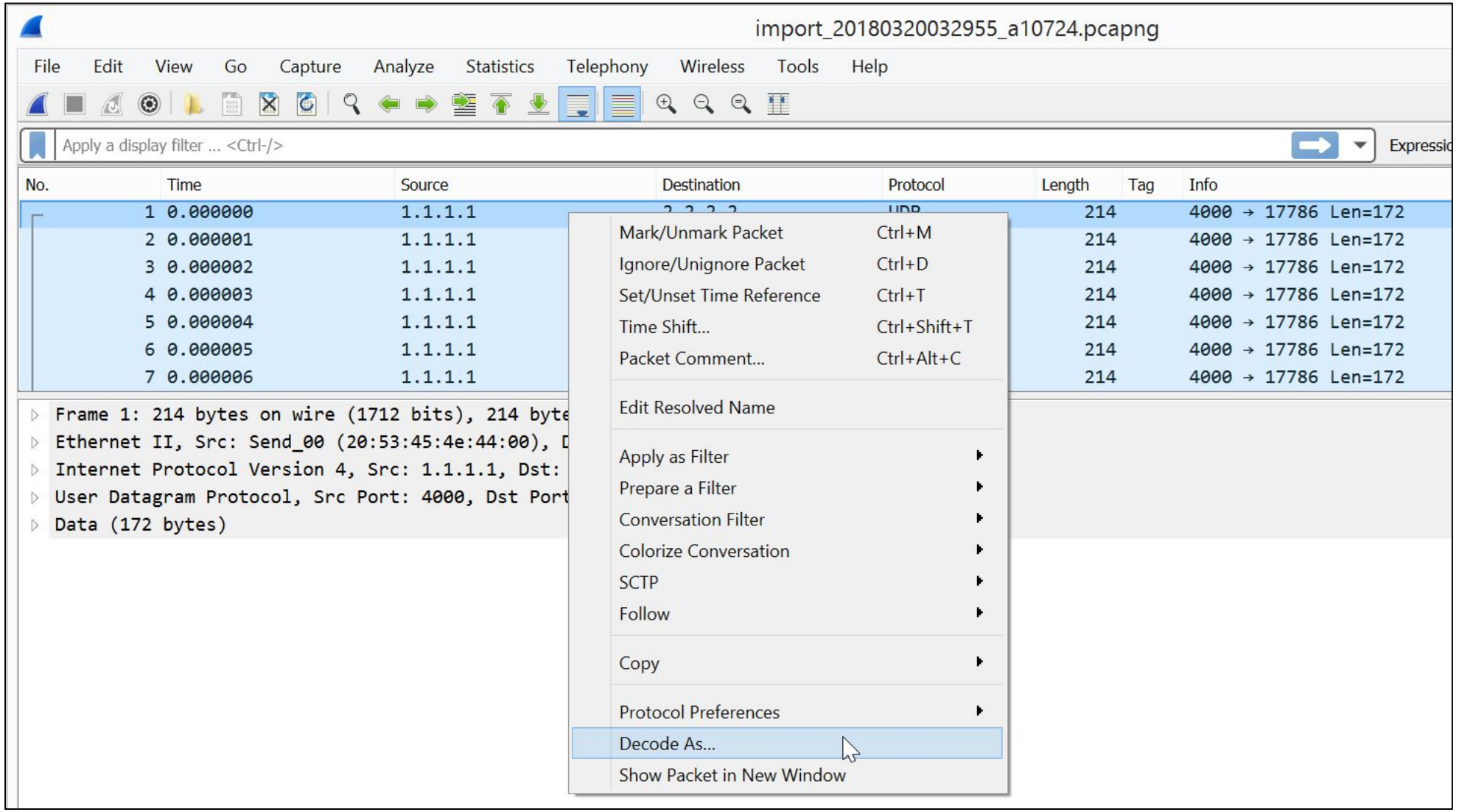
SRTP Decrypt: Importing Decrypted Content



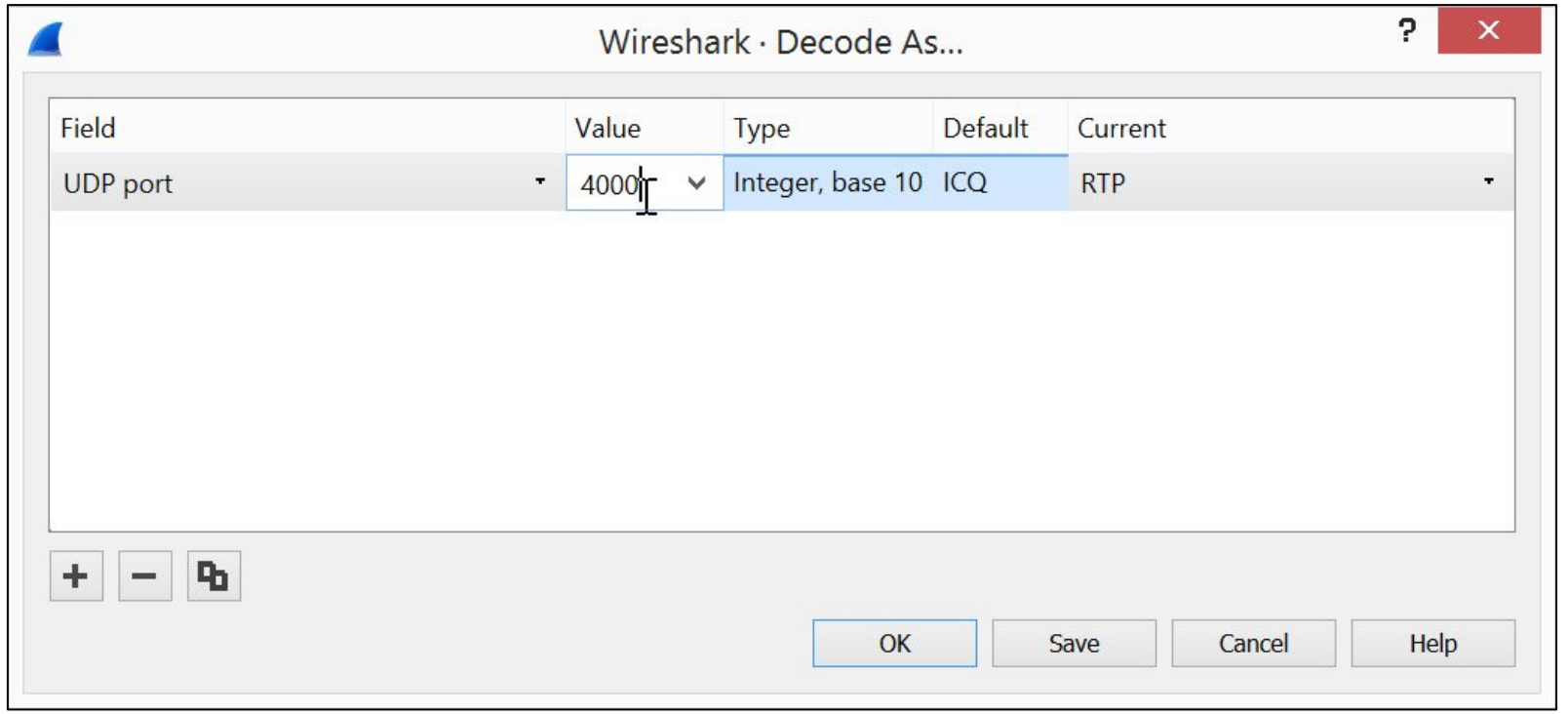
SRTP Decrypt: Imported Decrypted UDP Packets



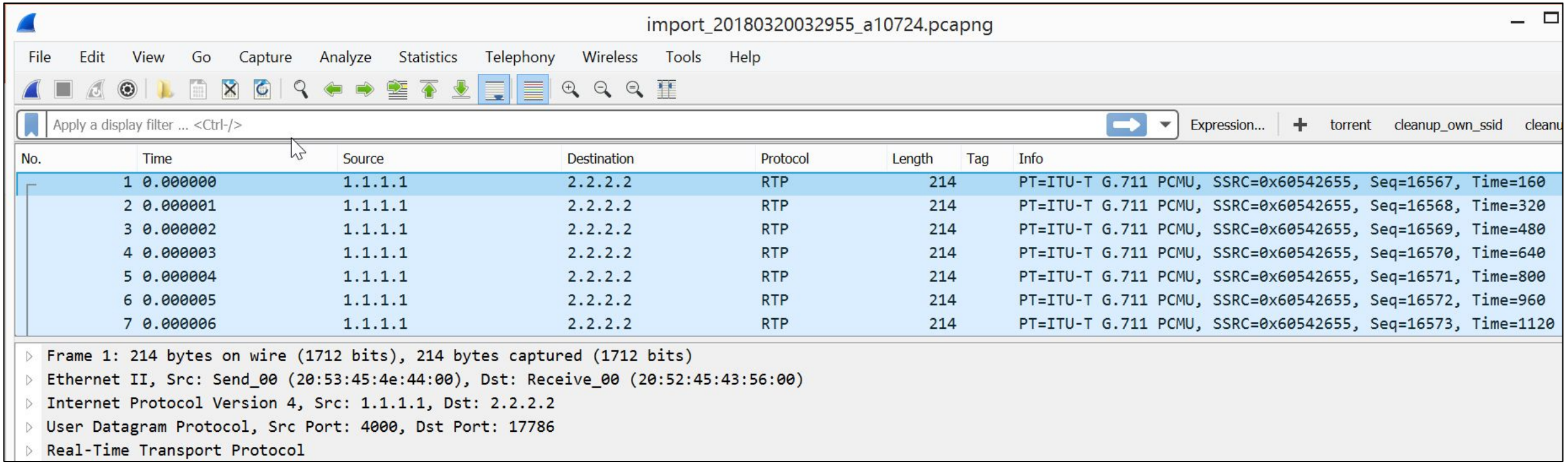
SRTP Decrypt: Decode As



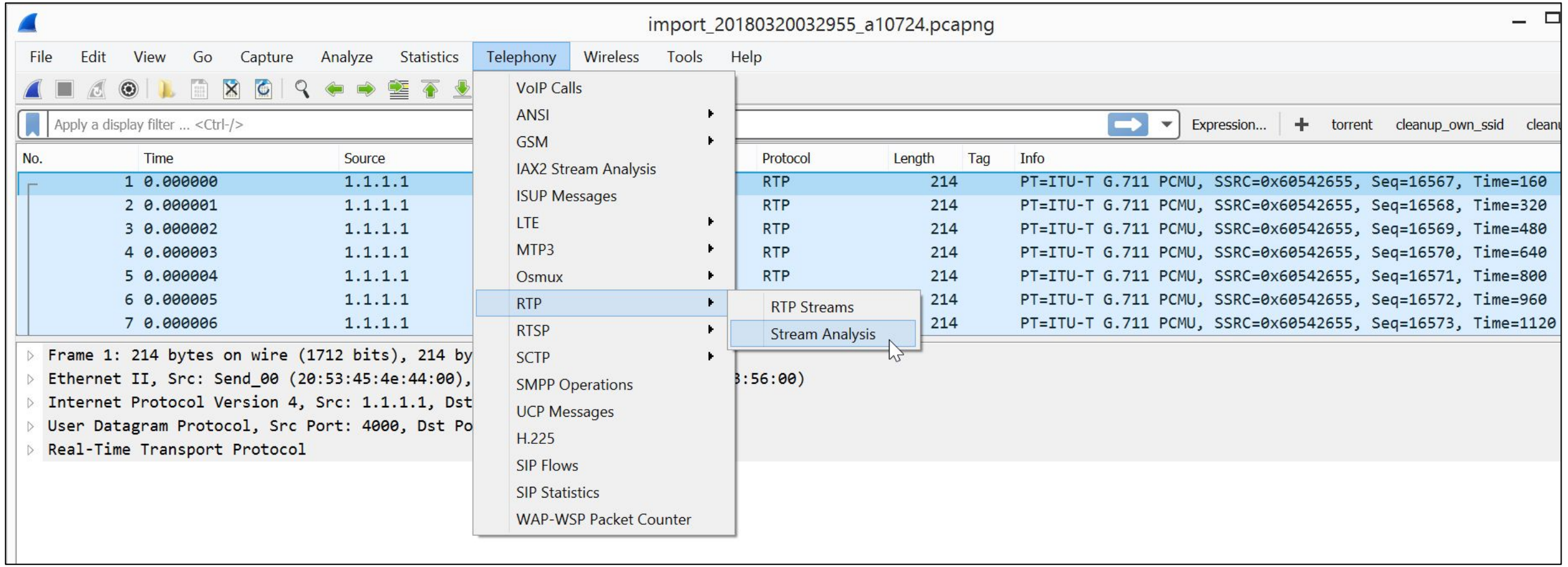
SRTP Decrypt: Decode As RTP



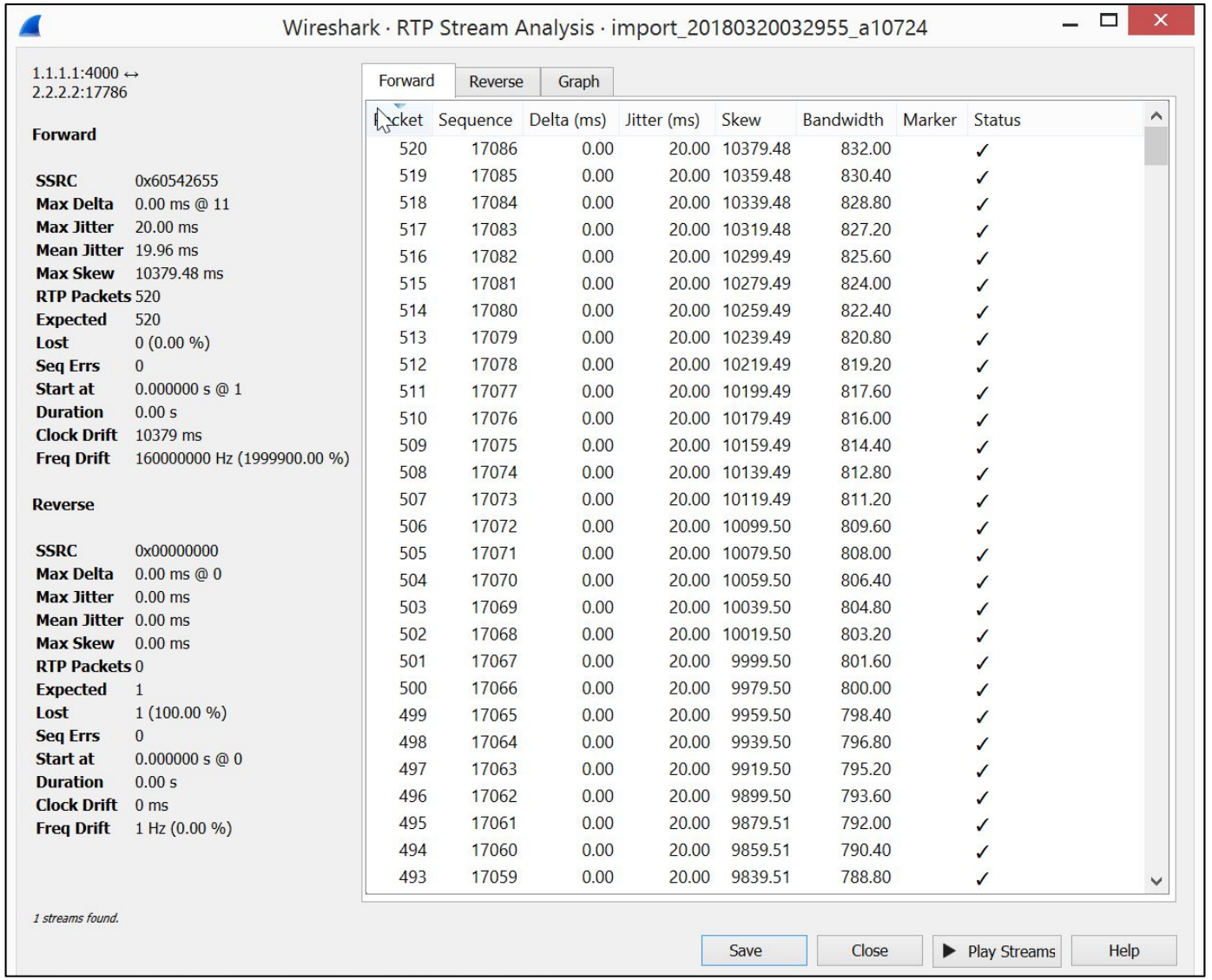
SRTP Decrypt: Decoded Packets



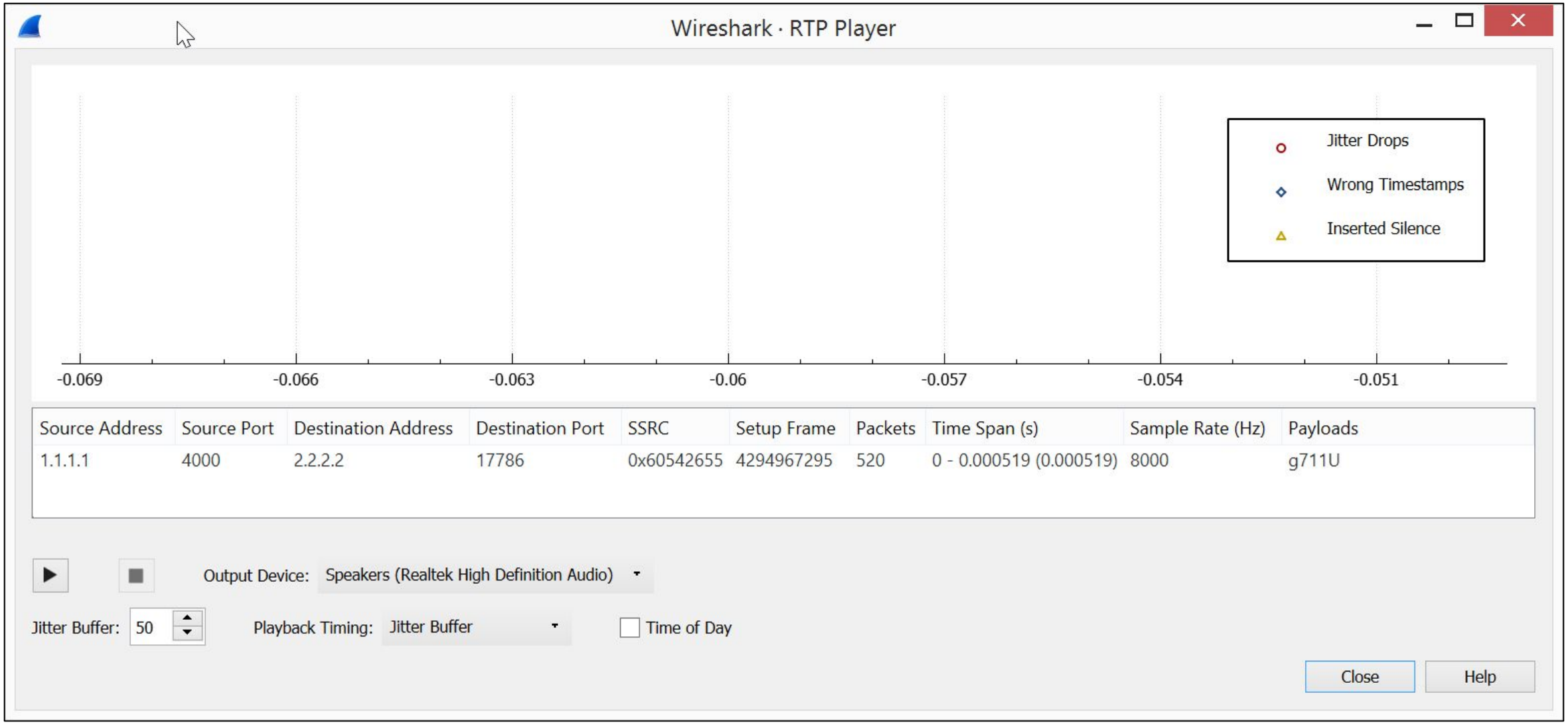
SRTP Decrypt: Checking RTP Streams



SRTP Decrypt: Analysing RTP Streams



SRTP Decrypt: Playing Decrypted Call

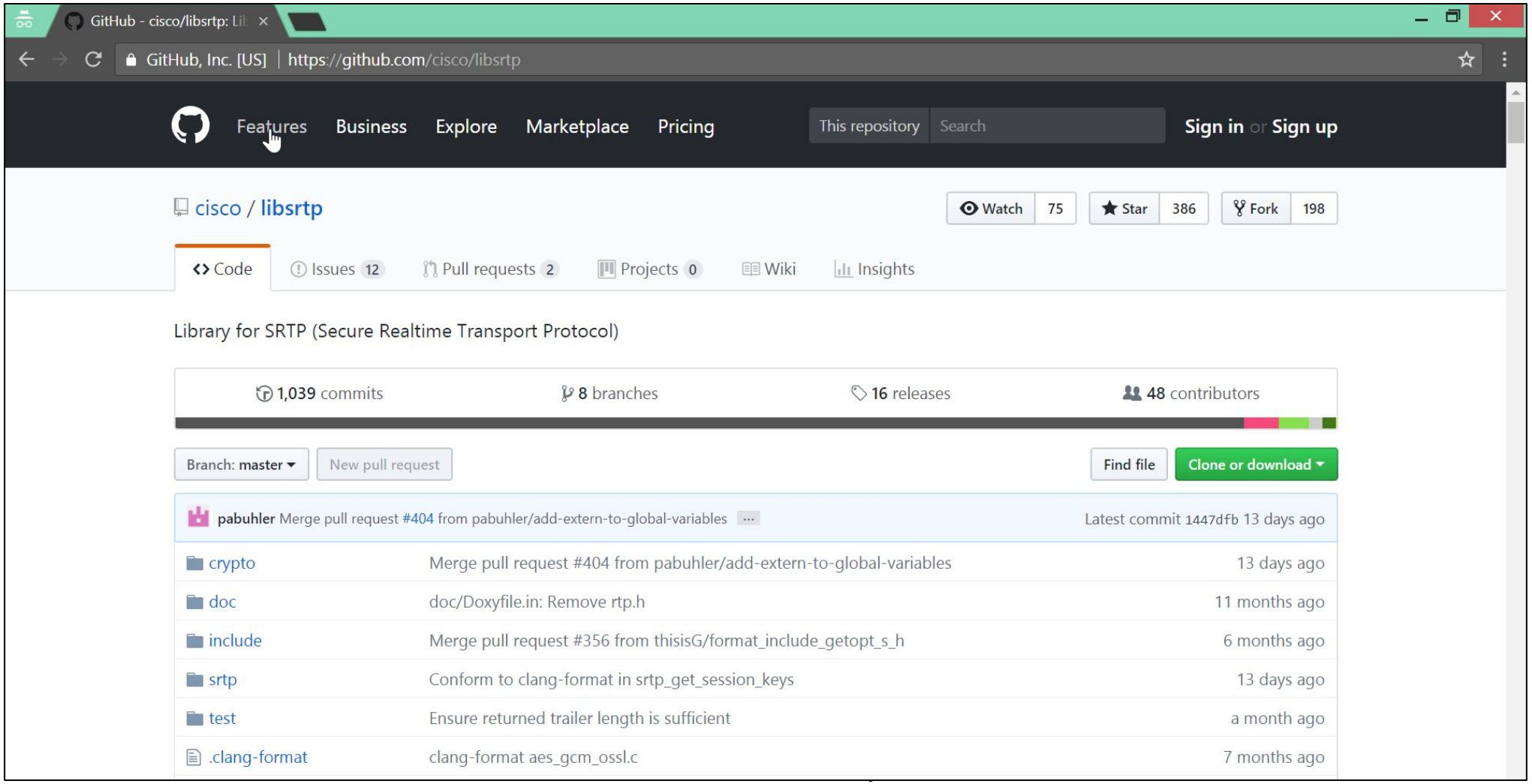


Libsrtp

- Implementation of the Secure Real-time Transport Protocol (SRTP)
- Can decipher SRTP packets

Libsrtp

- GitHub: github.com/cisco/libsrtp



Libsrtp: Installation

- Cloning

```
root@PentesterAcademy:/work# git clone https://github.com/cisco/libsrtp.git
Cloning into 'libsrtp'...
remote: Counting objects: 6495, done.
remote: Total 6495 (delta 0), reused 0 (delta 0), pack-reused 6495
Receiving objects: 100% (6495/6495), 5.28 MiB | 126.00 KiB/s, done.
Resolving deltas: 100% (4442/4442), done.
root@PentesterAcademy:/work# cd libsrtp/
```

Libsrtp: Installation

- Configure

```
root@PentesterAcademy:/work/libsrtp# ./configure
checking for gcc... gcc
checking whether the C compiler works... yes
checking for C compiler default output file name... a.out
checking for suffix of executables...
checking whether we are cross compiling... no
checking for suffix of object files... o
checking whether we are using the GNU C compiler... yes
checking whether gcc accepts -g... yes
checking for gcc option to accept ISO C89... none needed
checking how to run the C preprocessor... gcc -E
checking for ar... ar
checking the archiver (ar) interface... ar
checking for ranlib... ranlib
checking for a BSD-compatible install... /usr/bin/install -c
checking for a sed that does not truncate output... /bin/sed
checking for grep that handles long lines and -e... /bin/grep
```

Libsrtp: Installation

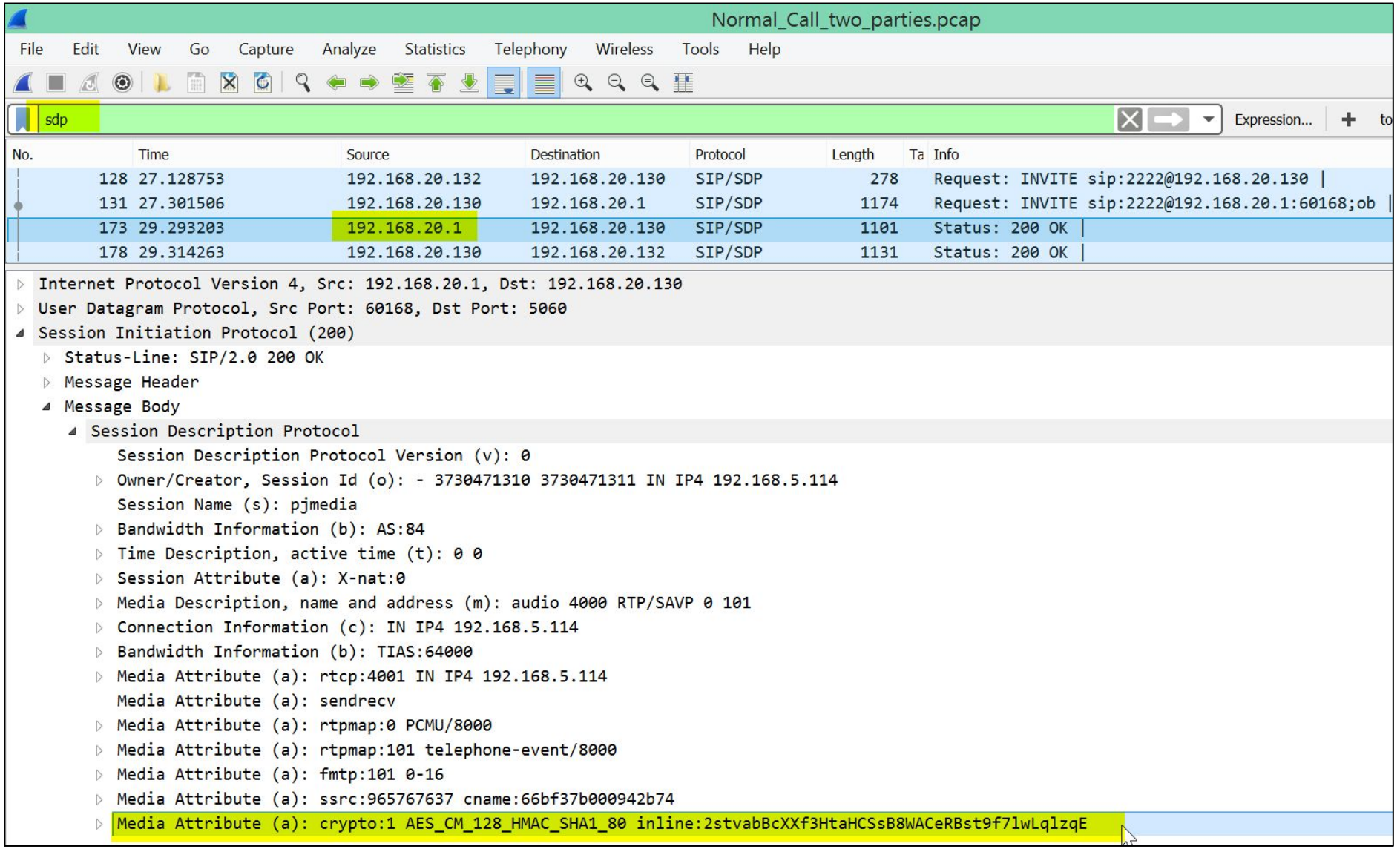
- Make

```
root@PentesterAcademy:/work/libsrtp# make
gcc -DHAVE_CONFIG_H -Icrypto/include -I./include -I./crypto/include  -fPIC -Wall
oll-loops -c srtp/srtp.c -o srtp/srtp.o
gcc -DHAVE_CONFIG_H -Icrypto/include -I./include -I./crypto/include  -fPIC -Wall
oll-loops -c srtp/ekt.c -o srtp/ekt.o
gcc -DHAVE_CONFIG_H -Icrypto/include -I./include -I./crypto/include  -fPIC -Wall
oll-loops -c crypto/cipher/cipher.c -o crypto/cipher/cipher.o
gcc -DHAVE_CONFIG_H -Icrypto/include -I./include -I./crypto/include  -fPIC -Wall
oll-loops -c crypto/cipher/null_cipher.c -o crypto/cipher/null_cipher.o
gcc -DHAVE_CONFIG_H -Icrypto/include -I./include -I./crypto/include  -fPIC -Wall
oll-loops -c crypto/cipher/aes_icm.c -o crypto/cipher/aes_icm.o
gcc -DHAVE_CONFIG_H -Icrypto/include -I./include -I./crypto/include  -fPIC -Wall
oll-loops -c crypto/cipher/aes.c -o crypto/cipher/aes.o
gcc -DHAVE_CONFIG_H -Icrypto/include -I./include -I./crypto/include  -fPIC -Wall
oll-loops -c crypto/hash/null_auth.c -o crypto/hash/null_auth.o
gcc -DHAVE_CONFIG_H -Icrypto/include -I./include -I./crypto/include  -fPIC -Wall
oll-loops -c crypto/hash/auth.c -o crypto/hash/auth.o
gcc -DHAVE_CONFIG_H -Icrypto/include -I./include -I./crypto/include  -fPIC -Wall
oll-loops -c crypto/hash/hmac.c -o crypto/hash/hmac.o
```

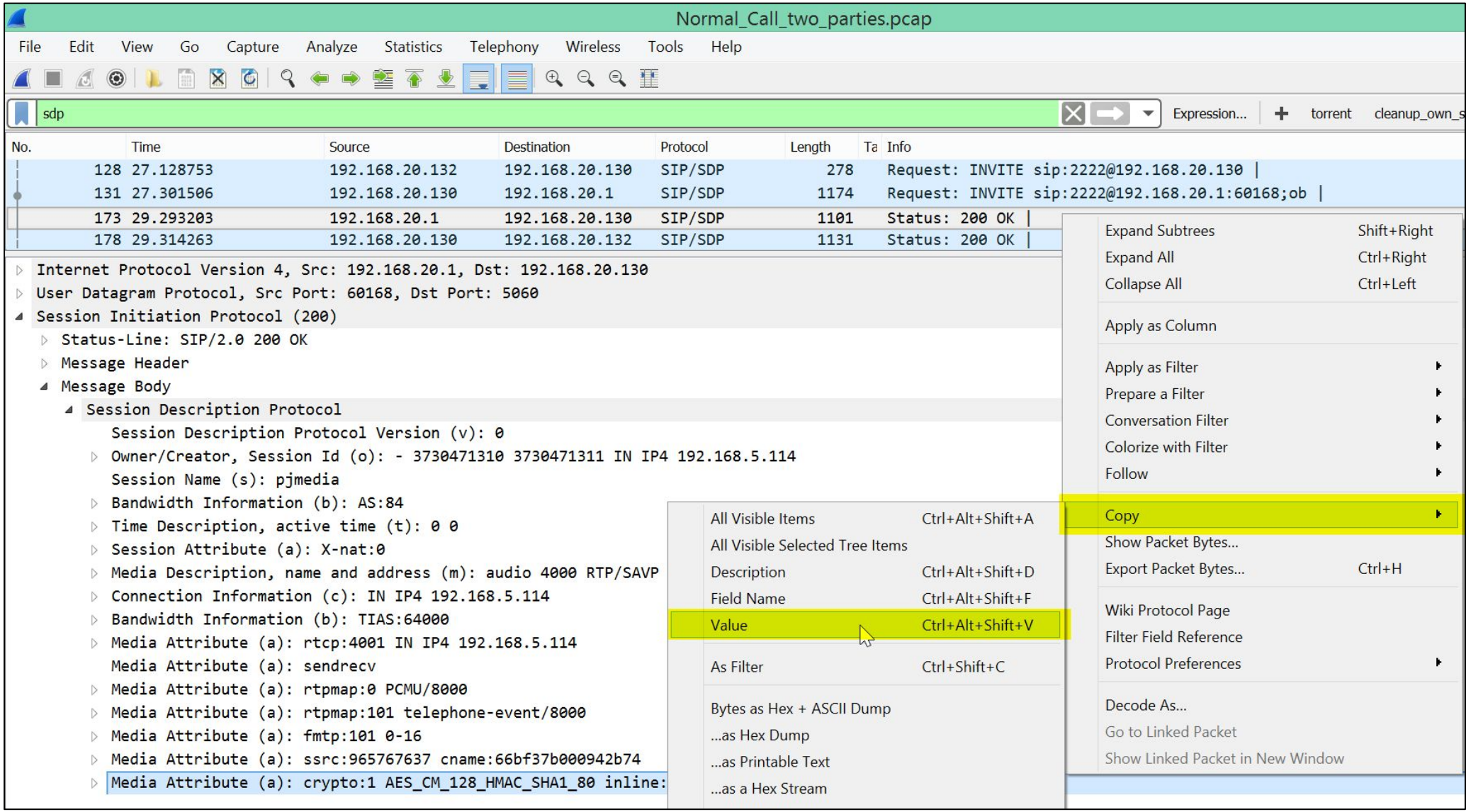
Libsrtp: Ready

```
root@PentesterAcademy:/work/libsrtp/test# ./rtp_decoder -h
Using libsrtp2 2.2.0-pre [0x2020000]
usage: ./rtp_decoder [-d <debug>]* [[-k][-b] <key> [-a][-e]]
or     ./rtp_decoder -l
where  -a use message authentication
       -e <key size> use encryption (use 128 or 256 for key size)
       -g Use AES-GCM mode (must be used with -e)
       -t <tag size> Tag size to use (in GCM mode use 8 or 16)
       -k <key>  sets the srtp master key given in hexadecimal
       -b <key>  sets the srtp master key given in base64
       -l list debug modules
       -f "<pcap filter>" to filter only the desired SRTP packets
       -d <debug> turn on debugging for module <debug>
       -s "<srtp-crypto-suite>" to set both key and tag size based
          on RFC4568-style crypto suite specification
```

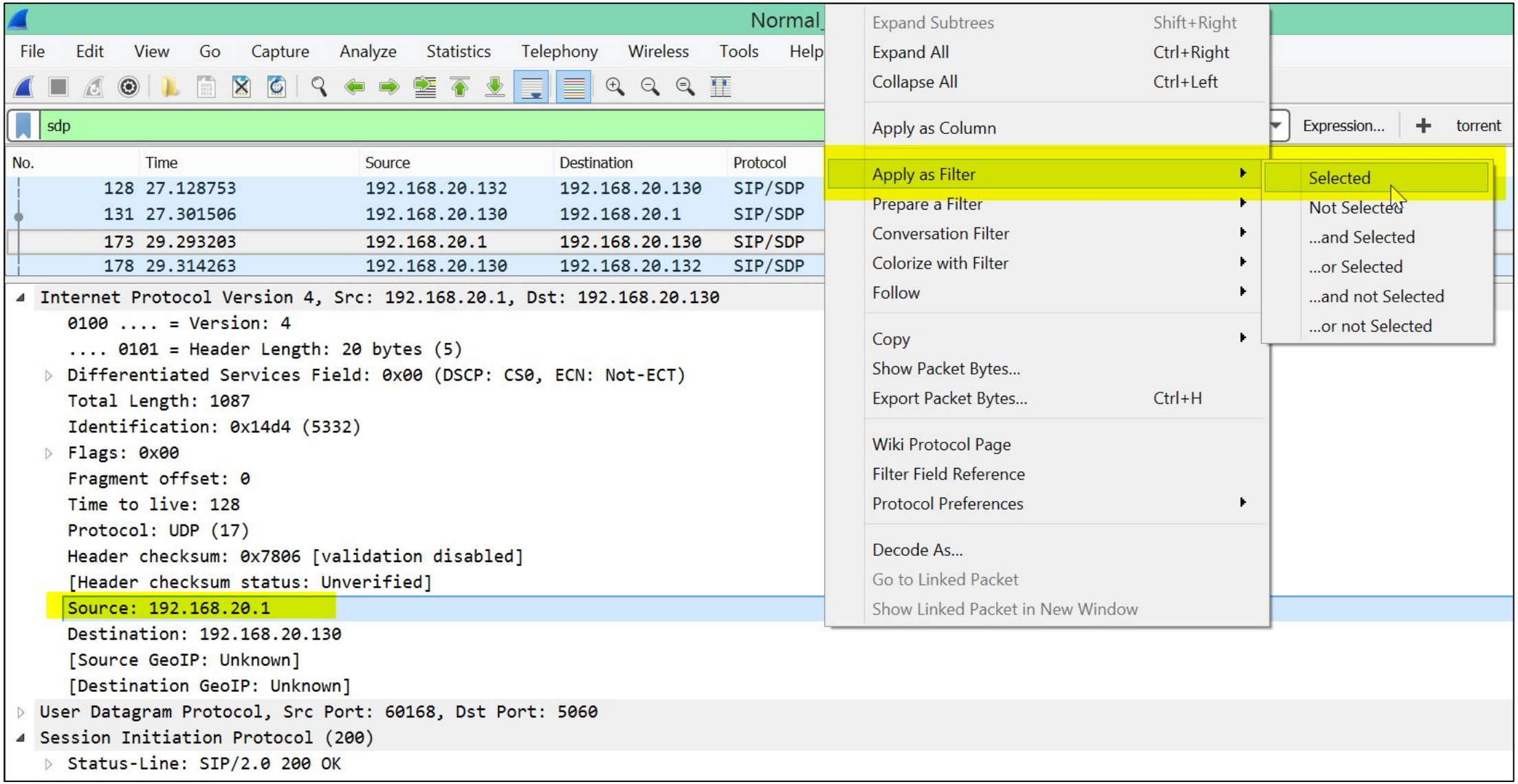
Libsrtp: SRTP key



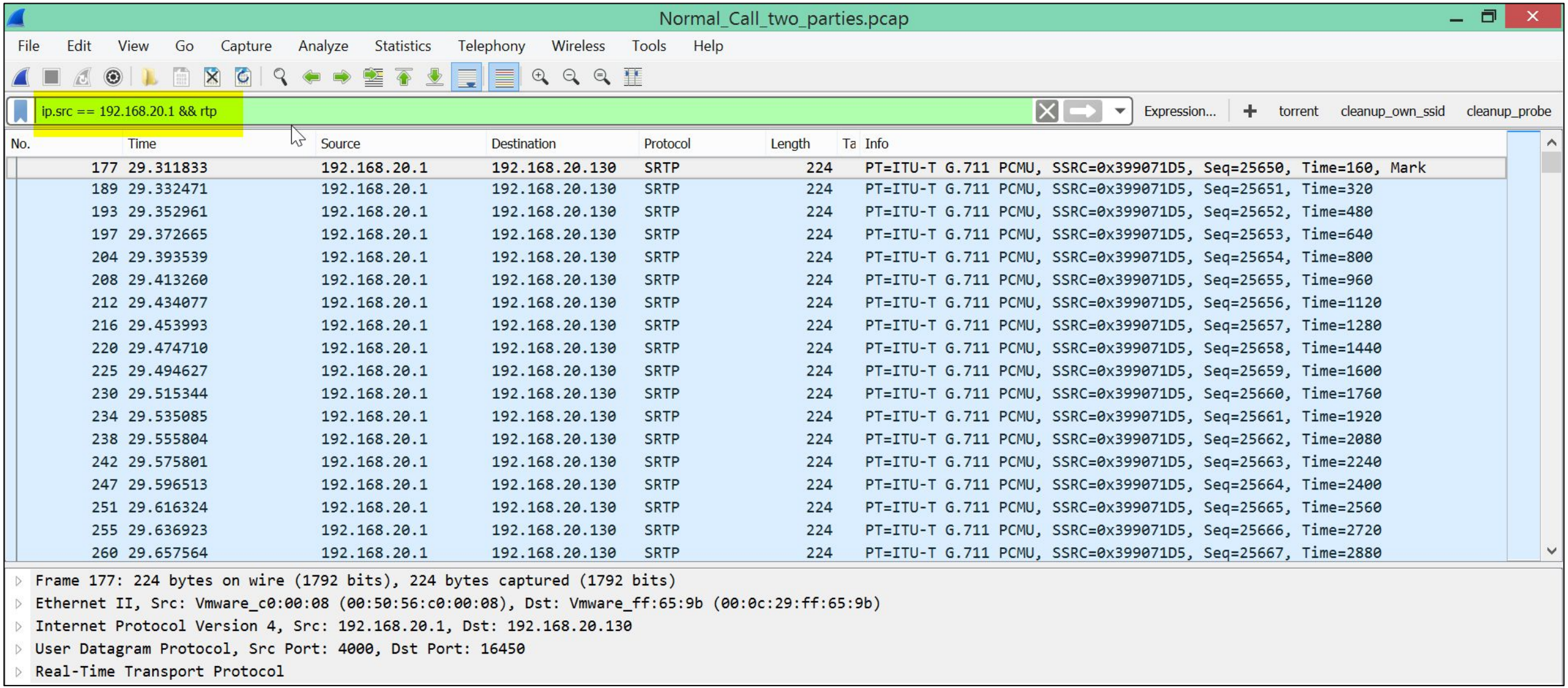
Libsrtp: Copying SRTP key



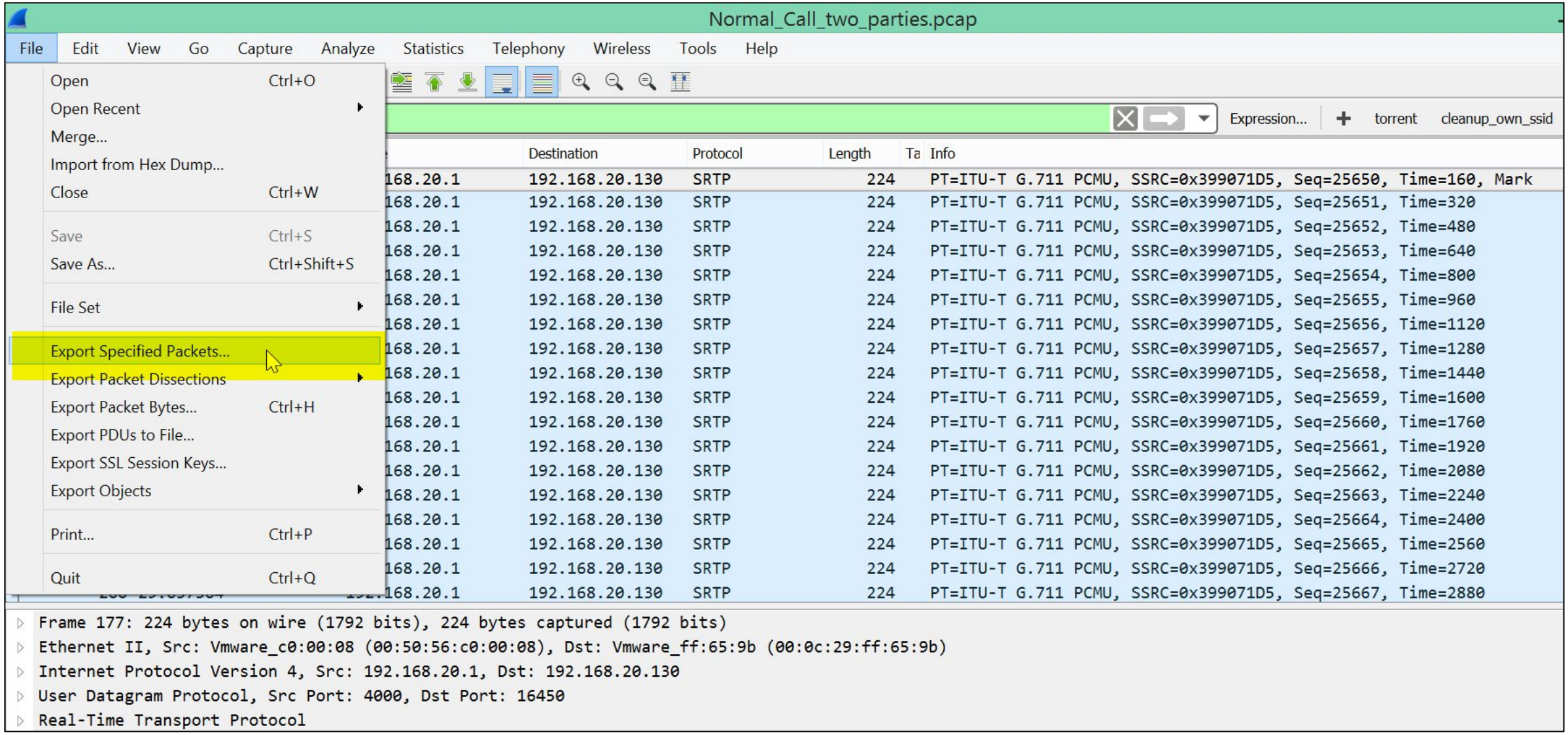
Libsrtp: Filtering for one sender



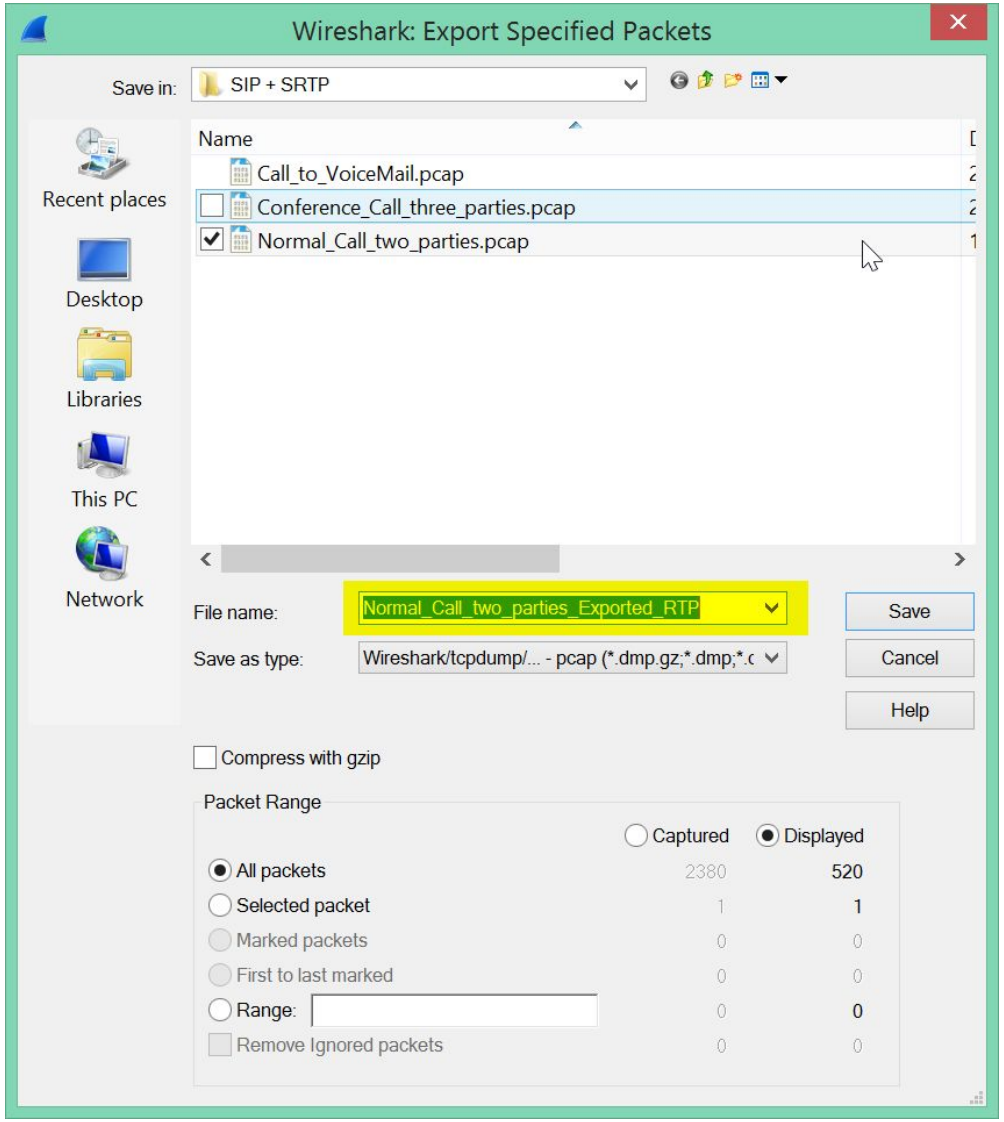
Libsrtp: Filtering single RTP stream



Libsrtp: Exporting filtered traffic



Libsrtp: Saving exported traffic



Libsrtp: Command

- ./rtp_decoder -a -t 10 -e 128 -b 2stvabBcXXf3HtaHCSsB8WACeRBst9f7lwLqlzqE * < ./Normal_Call_two_parties_Exported_RTP.pcap

- -a Use message authentication
- -t Authentication tag size (80 bits so 10 bytes)
- -e Length of encryption key. In our case, AES_CM_128_HMAC_SHA1_80 is cipher. Hence, 128 bit key is used.
- -b SRTP key in ASCII format

Libsrtp: Command output

```
root@PentesterAcademy:/work/libsrtp/test# ./rtp_decoder -a -t 10 -e 128 -b 2stvabBcXXf3HtaHCSsB8WACeRBst9f7lwLqlzqE * < ../../Normal_Call_two_part
ies_Exported_RTP.pcap
Using libsrtp2 2.2.0-pre [0x2020000]
security services: confidentiality message authentication
setting tag len 10
set master key/salt to dacb6f69b05c5d77f71ed6870 92b01f1/600279106cb7d7fb9702ea973a84
Starting decoder
00:00.000000
0000 80 80 64 32 00 00 00 a0 39 90 71 d5 ff ff ff ff
0010 7e ff 7e ff 7e 7e ff 7e fe 7d fe 7d fd 7d fe 7e
0020 7d fb 78 f6 73 f2 71 f2 76 fd f4 66 d5 29 1b 1e
0030 1e 22 24 27 29 2c 33 3a 3f 46 5a ef d5 c9 bb af
0040 ab a9 a8 a6 a3 9f 9e 9e 9d 9d 9c 9c 9b 9b 9a 99
0050 98 97 97 97 97 97 97 97 98 99 9a 9c 9d 9e 9e 9f
0060 a2 a6 a8 aa ac af b9 c2 ce df 60 4a 40 3b 37 35
0070 33 31 2e 2c 29 27 27 27 27 27 29 29 28 27 27 2a
0080 2c 2d 2d 2d 2e 2f 2e 2c 2b 2b 2c 2d 2e 2e 2e 2f
0090 2f 2f 2f 2f 32 36 3c 47 4e 55 56 56 67 ed e2 e4
00a0 d5 c3 bb b7 b6 b7 b6 b3 ae ad ac aa
00:00.020638
0000 80 00 64 33 00 00 01 40 39 90 71 d5 a9 aa aa aa
0010 a8 a7 a7 a6 a7 a8 a9 a8 a6 a5 a5 a4 a3 a2 a3 a3
0020 a3 a2 a2 a2 a4 a4 a5 a6 a7 a8 a8 a9 aa ab ac ad
0030 af b0 b4 b8 ba bd c0 c9 d7 ec 69 57 49 40 3d 3d
0040 3c 39 34 30 2e 2d 2d 2d 2c 2c 2c 2d 2c 2c 2b 2a
0050 2a 29 29 29 2a 29 28 27 27 27 27 27 27 2a 2c 2e
0060 2e 2f 32 36 3a 3c 3d 3d 40 42 42 3e 3a 35 2f 2b
0070 2a 29 29 2b 2f 3e fe c0 b2 a9 a1 9d 9b 9a 9b 9c
0080 9e a5 ae b9 d0 4c 37 2e 2b 2c 2d 2d 2a 28 25 25
0090 27 2d 3b 5d c4 ae a1 9a 96 94 93 93 95 99 9f aa
00a0 b9 d8 45 31 28 22 1e 1e 1d 1d 1e 1f
00:00.041128
```

Libsrtp: text2pcap help

```
root@PentesterAcademy:~# text2pcap
Must specify input and output filename

Usage: text2pcap [options] <infile> <outfile>

where  <infile> specifies input  filename (use - for standard input)
      <outfile> specifies output filename (use - for standard output)

Input:
  -o hex|oct|dec          parse offsets as (h)ex, (o)ctal or (d)ecimal;
                          default is hex.
  -t <timefmt>            treat the text before the packet as a date/time code;
                          the specified argument is a format string of the sort
                          supported by strptime.
                          Example: The time "10:15:14.5476" has the format code
                          "%H:%M:%S."
                          NOTE: The subsecond component delimiter, '.', must be
                          given, but no pattern is required; the remaining
                          number is assumed to be fractions of a second.
                          NOTE: Date/time fields from the current date/time are
                          used as the default for unspecified fields.
  -D                      the text before the packet starts with an I or an O,
                          indicating that the packet is inbound or outbound.
                          This is only stored if the output format is PCAP-NG.
  -a                      enable ASCII text dump identification.
                          The start of the ASCII text dump can be identified
                          and excluded from the packet data, even if it looks
                          like a HEX dump.
                          NOTE: Do not enable it if the input file does not
                          contain the ASCII text dump.
```

Libsrtp: text2pcap

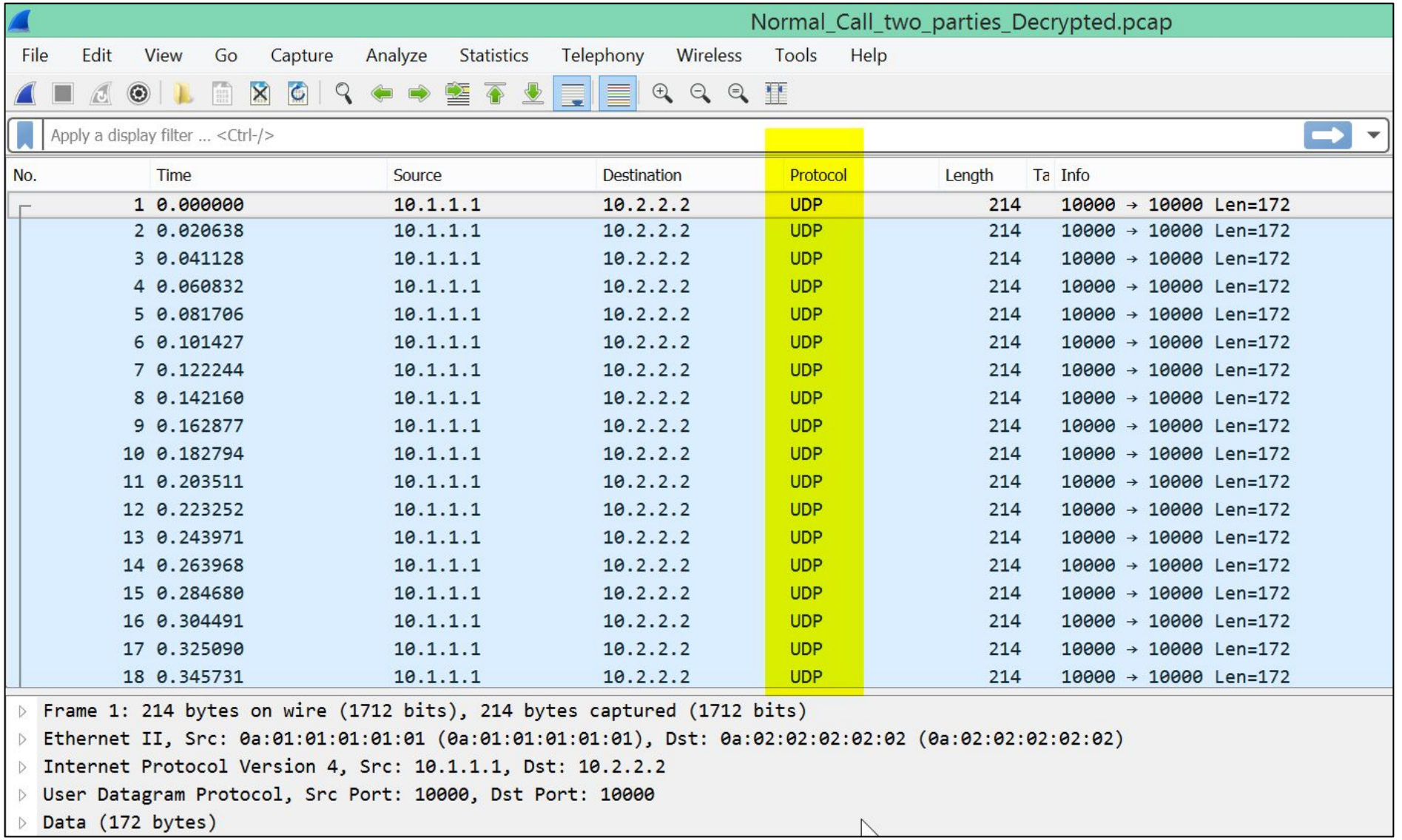
- text2pcap -t "%M:%S." -u 10000,10000 - - > ./Normal_Call_two_parties_Decrypted.pcap

- -t Treat the text before the packet as a date/time code
- %M:%S Time format
- -u Prepend dummy UDP header with specified source and destination ports

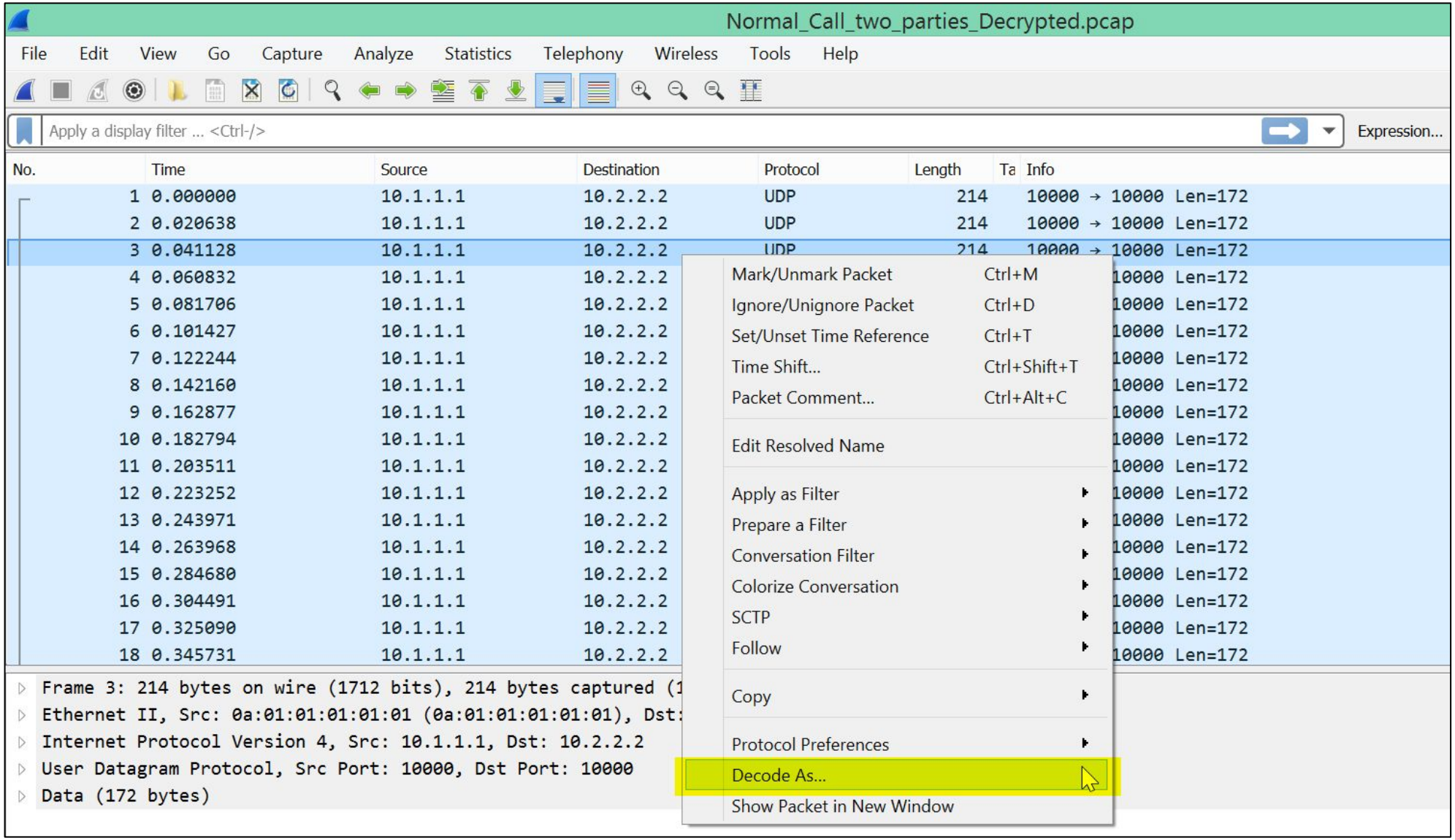
Libsrtp: Decrypting RTP traffic

```
root@PentesterAcademy:/work/libsrtp/test# ./rtp_decoder -a -t 10 -e 128 -b 2stvabBcXXf3Ht
aHCSsB8WACeRBst9f7lwLqlzqE  *  < ./Normal_Call_two_parties_Exported_RTP.pcap | text2pcap
-t "%M:%S." -u 10000,10000 - - > ./Normal_Call_two_parties_Decrypted.pcap
Input from: Standard input
Output to: Standard output
Output format: PCAP
Generate dummy Ethernet header: Protocol: 0x800
Generate dummy IP header: Protocol: 17
Generate dummy UDP header: Source port: 10000. Dest port: 10000
Using libsrtp2 2.2.0-pre [0x2020000]
security services: confidentiality message authentication
setting tag len 10
set master key/salt to dacb6f69b05c5d77f71ed687092b01f1/600279106cb7d7fb9702ea973a84
Starting decoder
Wrote packet of 214 bytes.
Wrote packet of 214 bytes.
Wrote packet of 214 bytes.
Wrote packet of 214 bytes.
Wrote packet of 214 bytes.
Wrote packet of 214 bytes.
Wrote packet of 214 bytes.
Wrote packet of 214 bytes.
Wrote packet of 214 bytes.
```

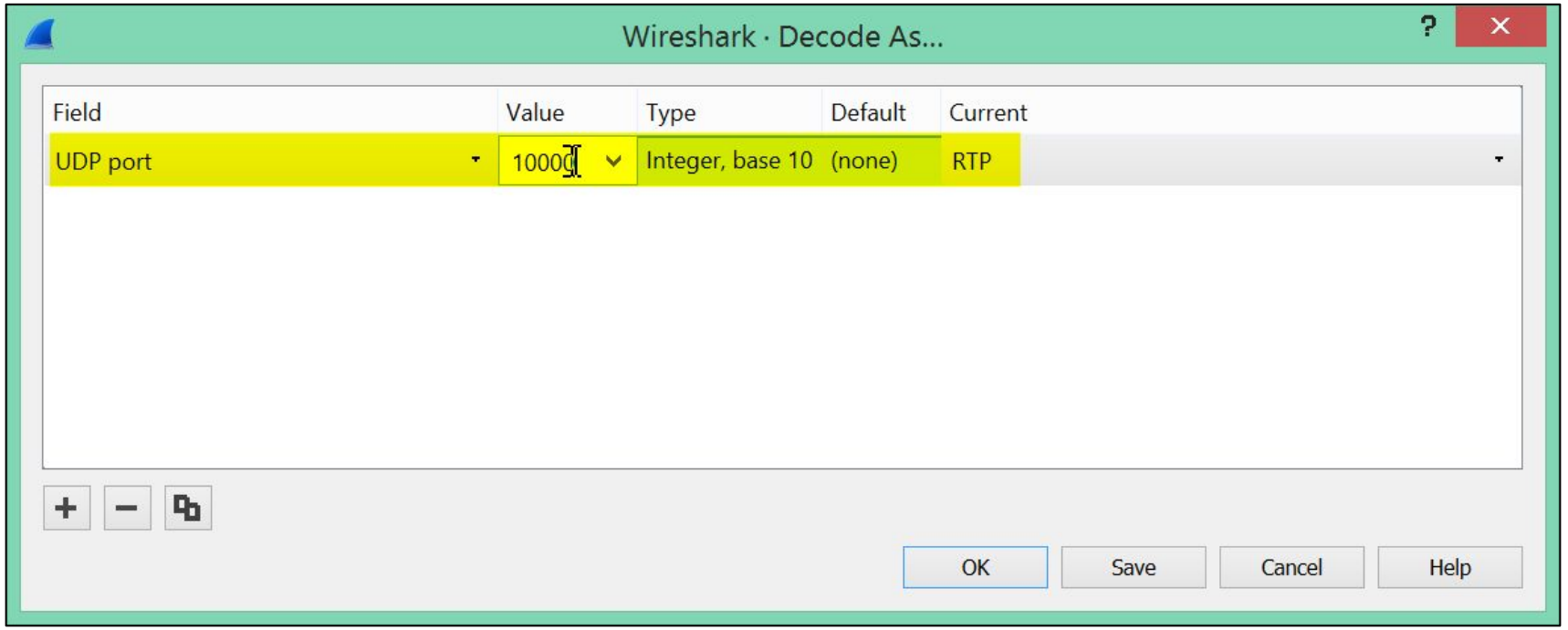
Libsrtp: Decrypted traffic



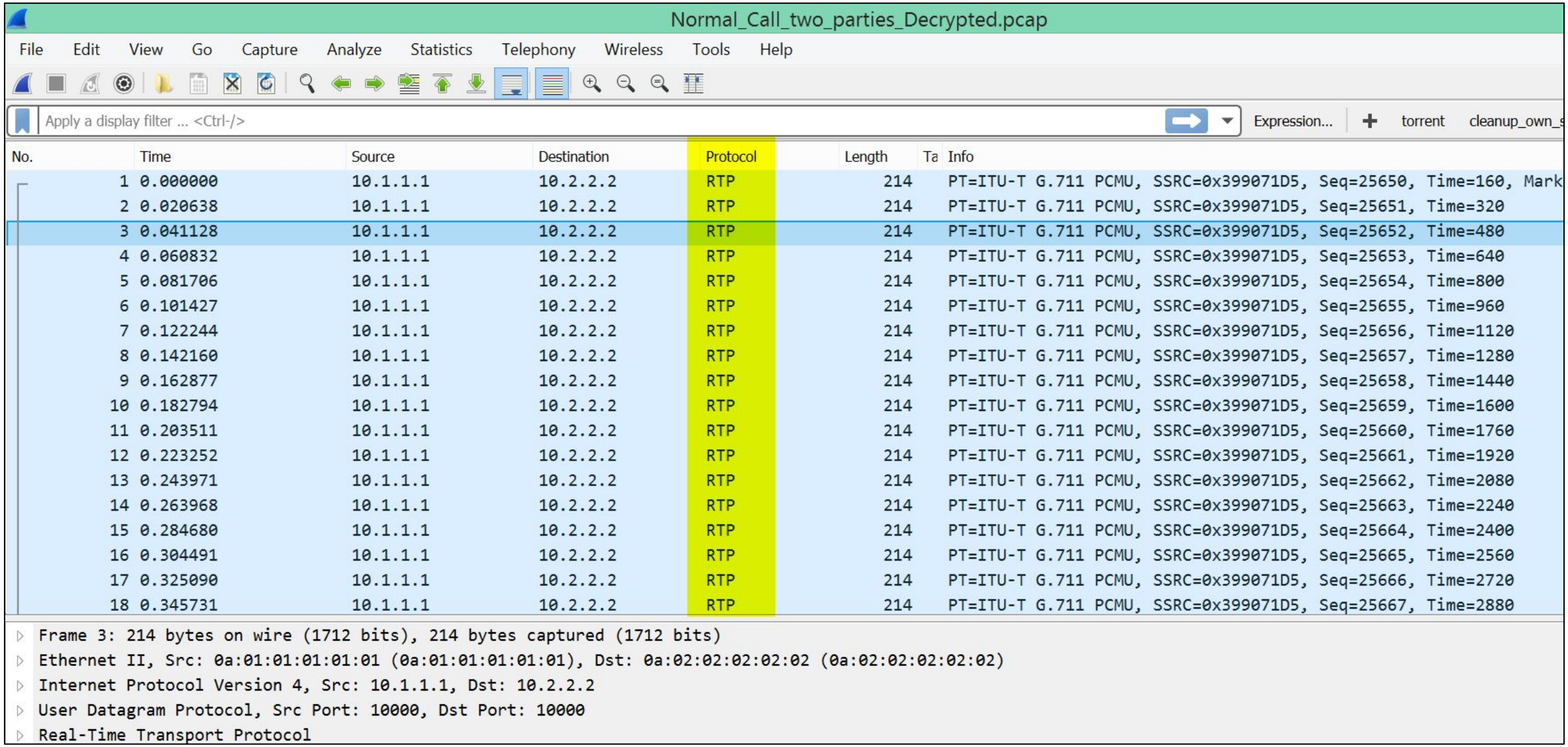
Libsrtp: Decode as



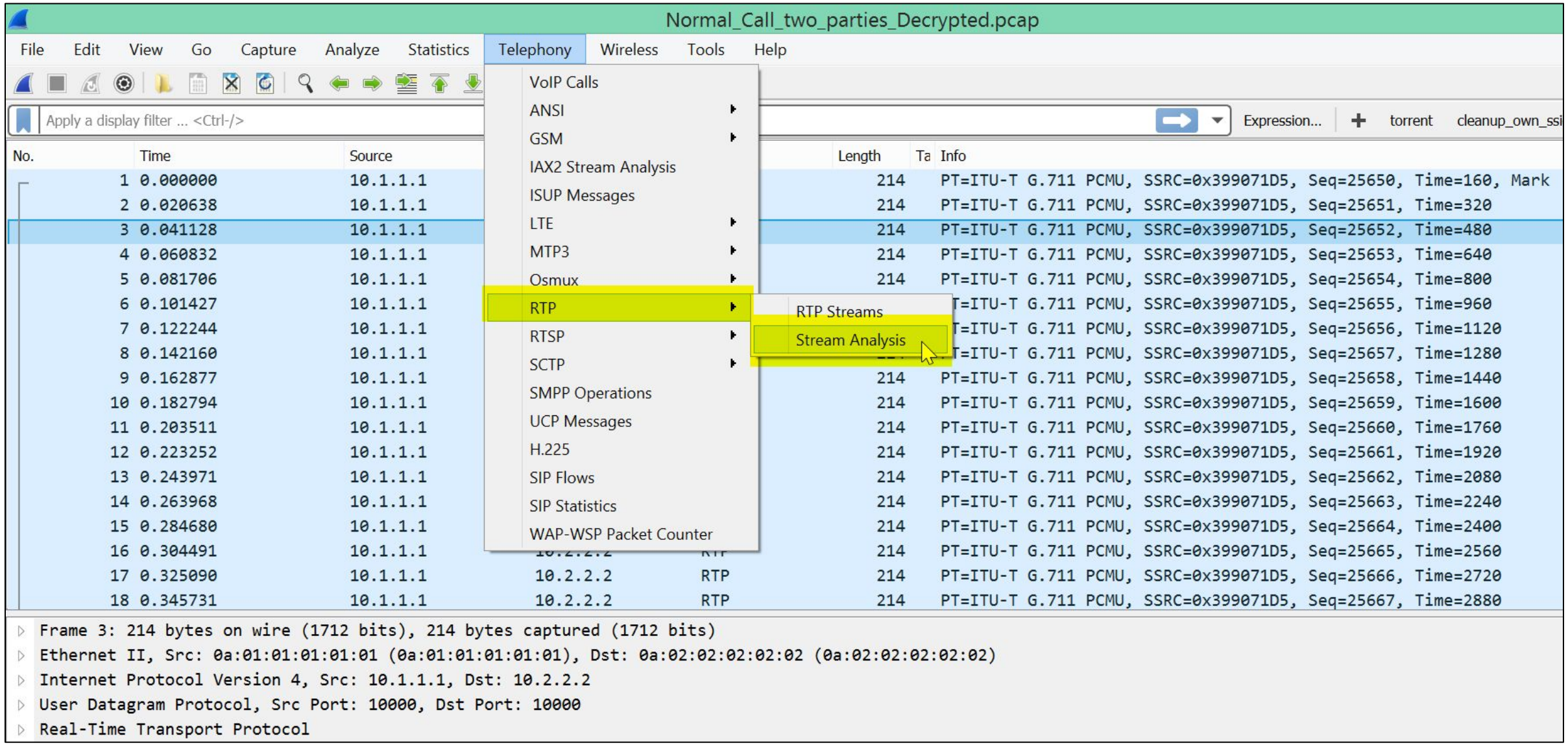
Libsrtp: Decode as RTP



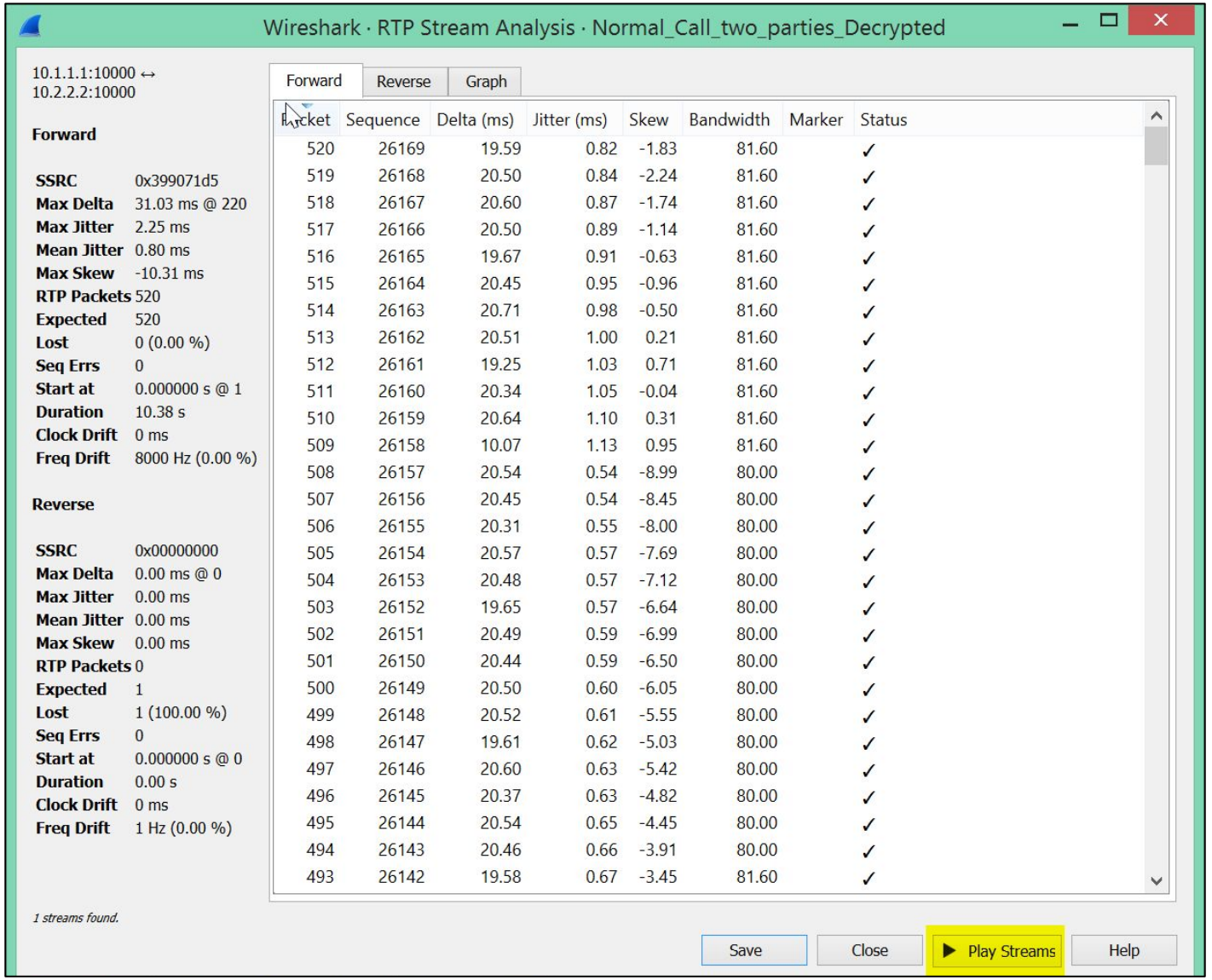
Libsrtp: Decrypted RTP traffic



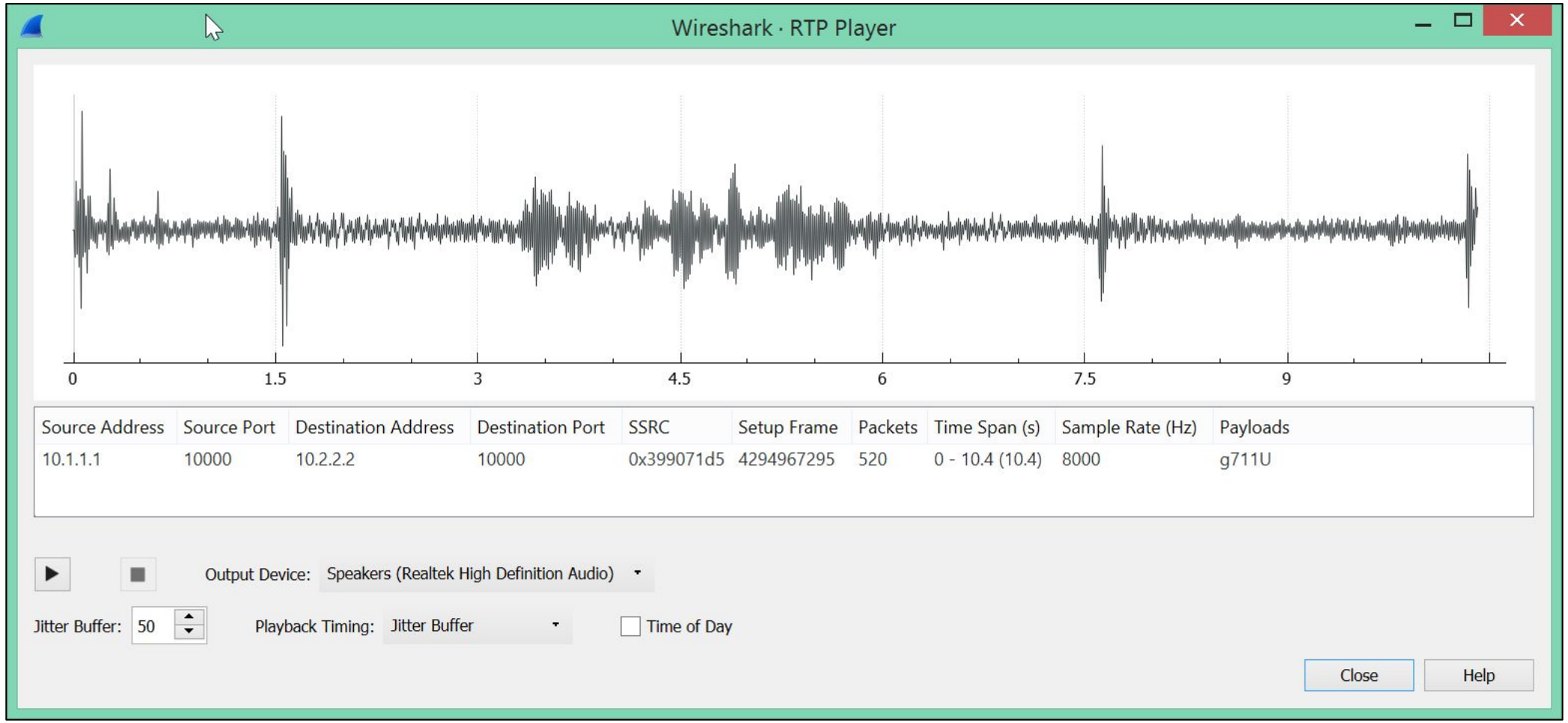
Libsrtp: Analysing RTP Streams



Libsrtp: Analysing RTP Streams



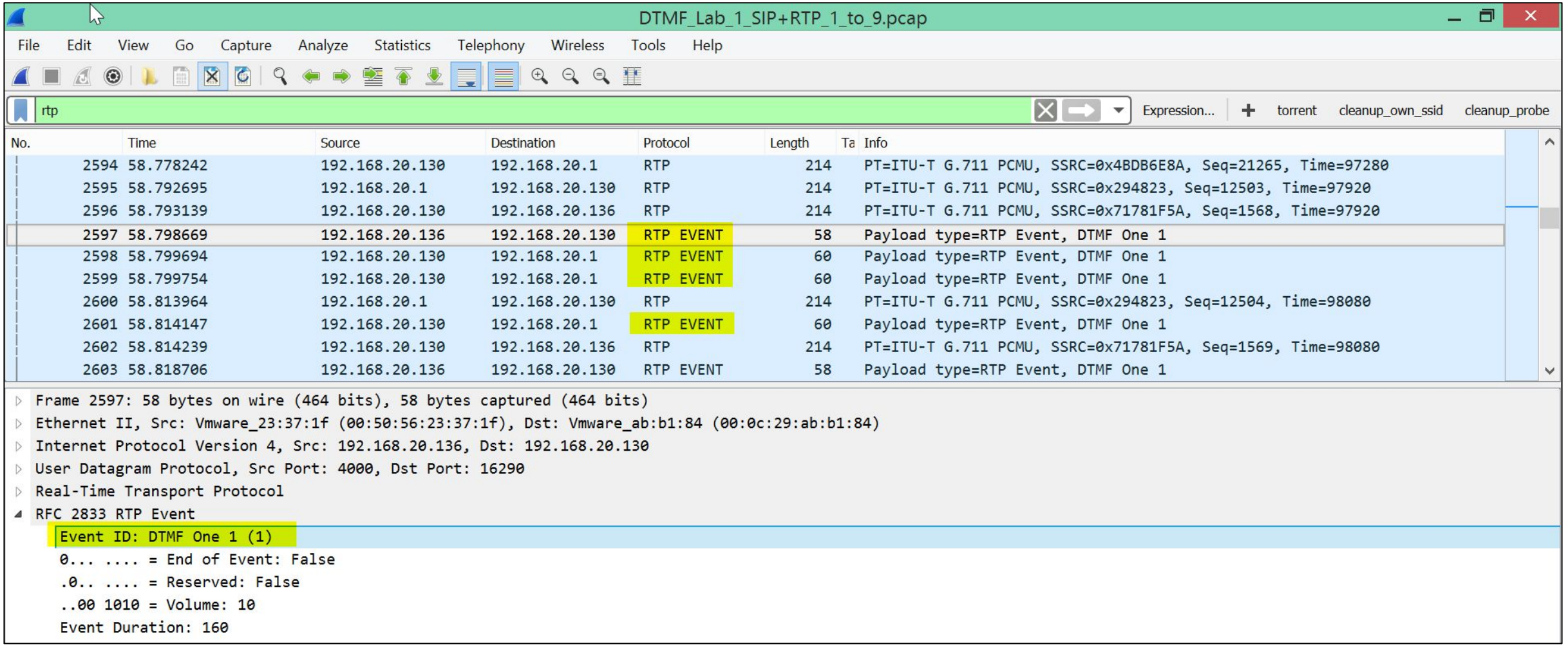
Libsrtp: Playing decrypted call



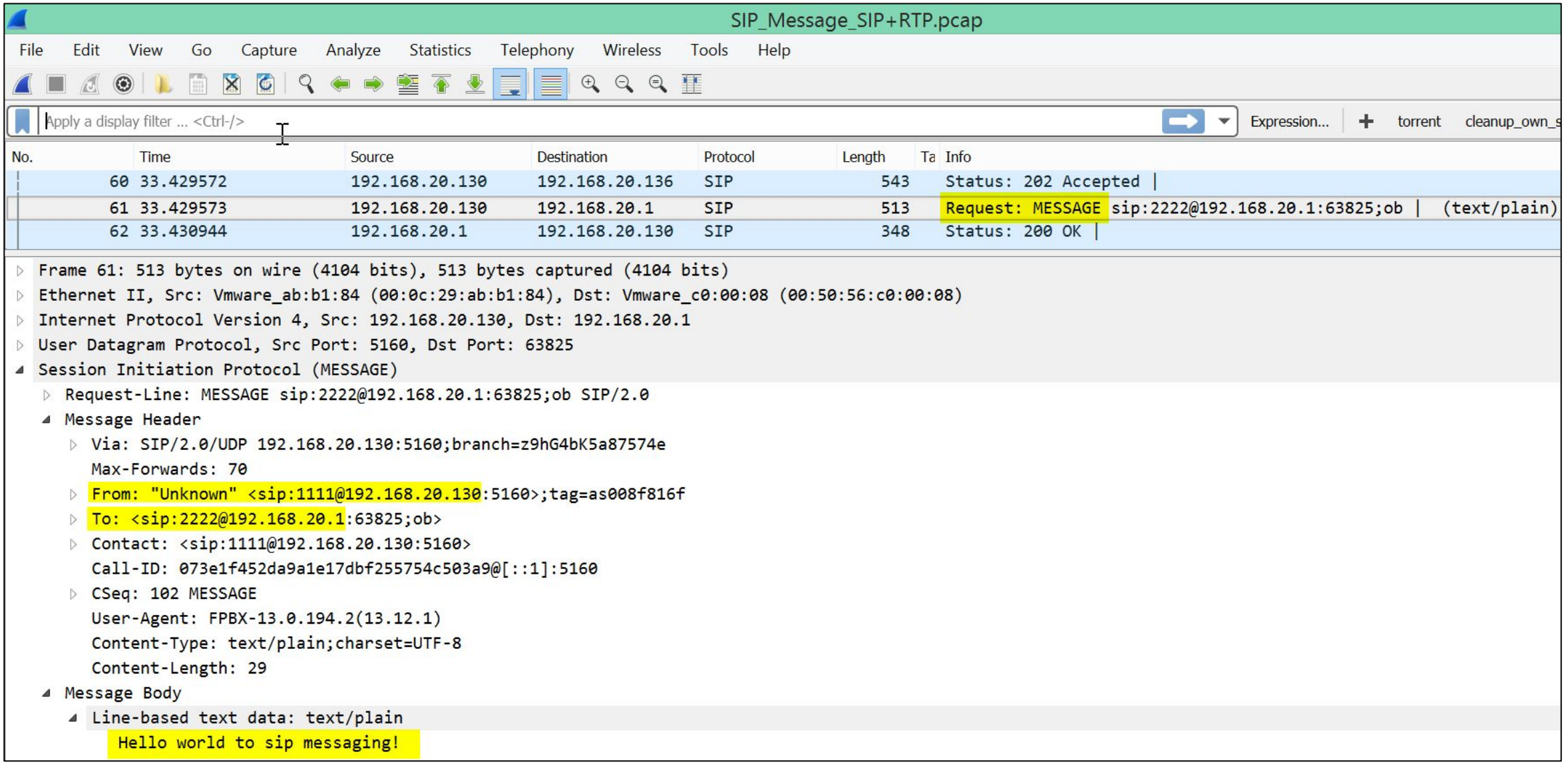
Other Important Parts?

- DTMF
- Messages (SMS)
- Exporting Call

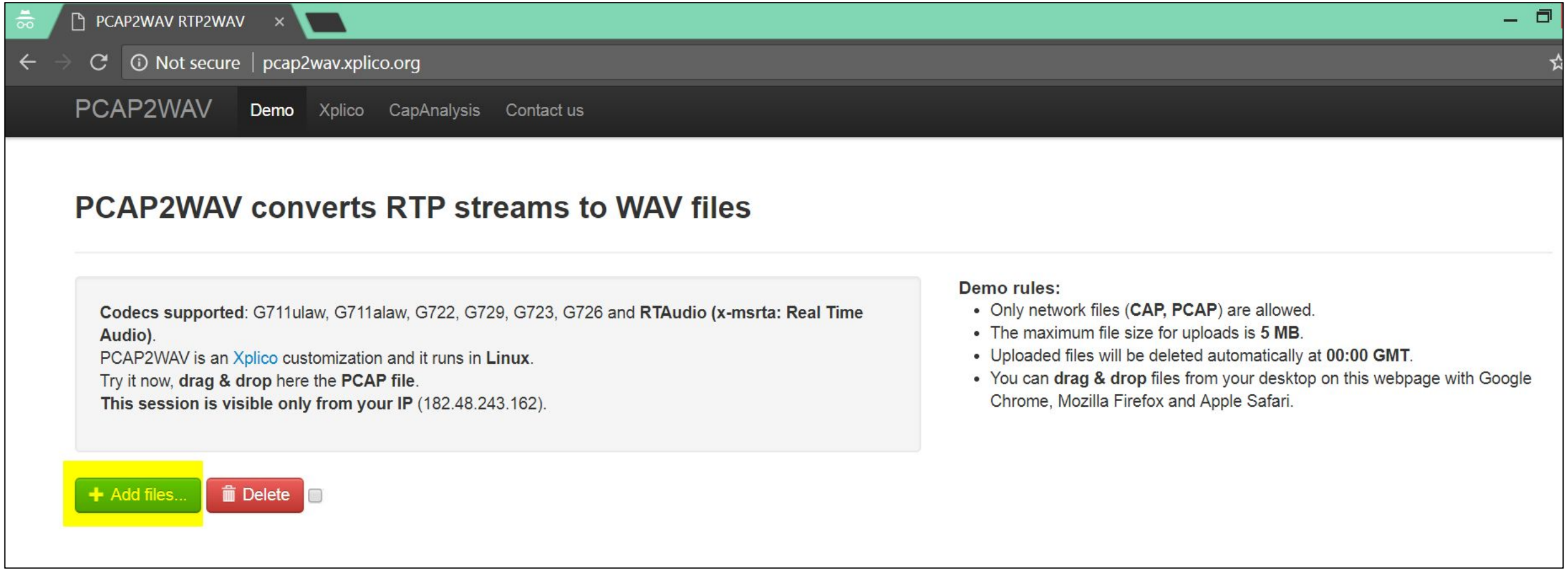
RTP DTMF



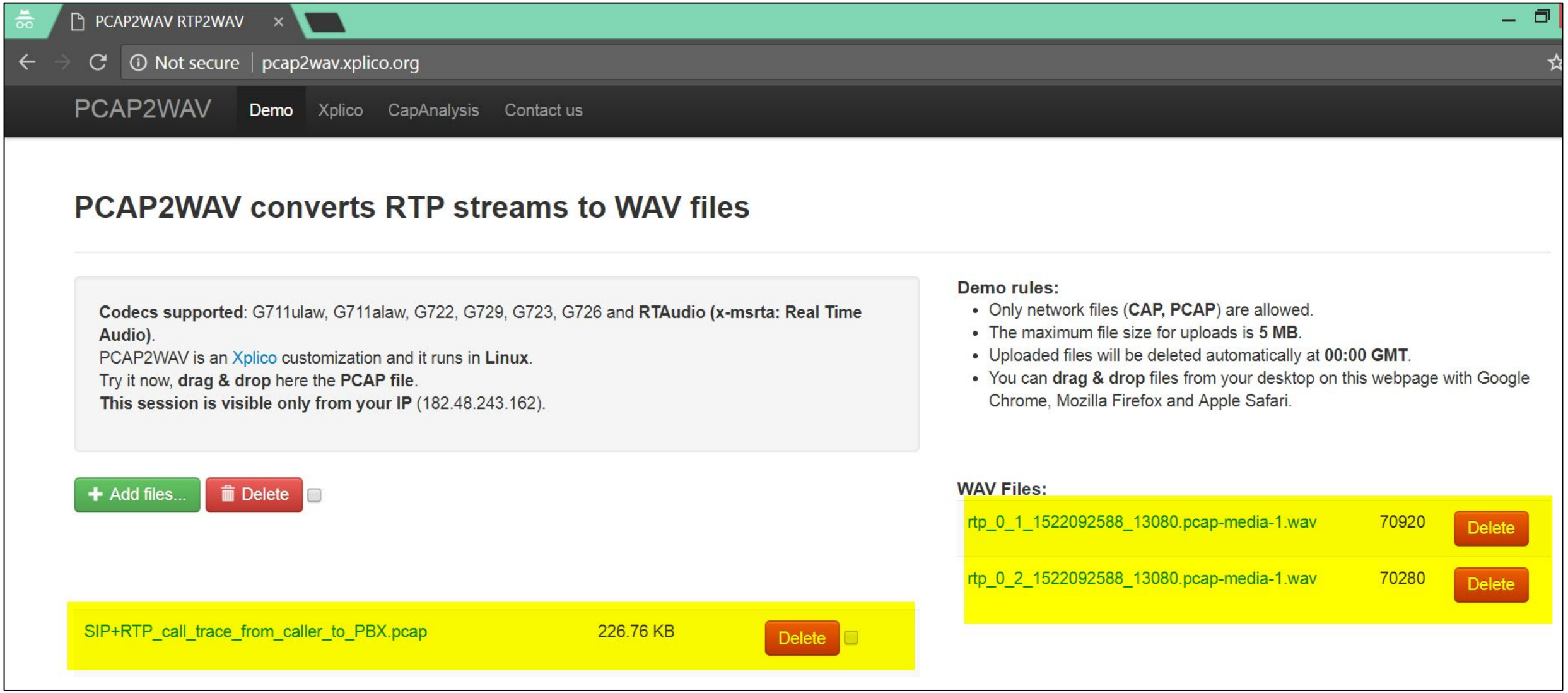
SIP Message



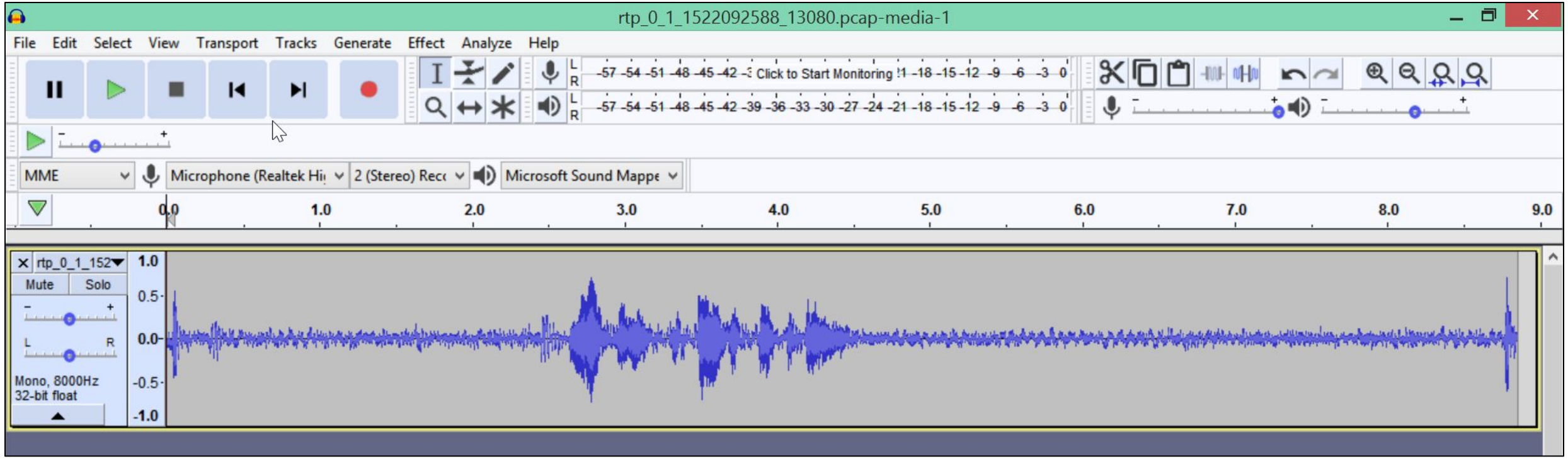
PCAP2WAV: Online service



PCAP2WAV: Uploading PCAP and Downloading Wav



PCAP2WAV: Wav in audacity



PCAP2WAV: Offline script

- Bash script to extract the audio from VoIP calls
- Outputs .wav file
- Uses tshark and sox
- GitHub: https://gist.github.com/avimar/d2e9d05e082ce273962d742eb9acac16

PCAP2WAV: Help

```
root@PentesterAcademy:/work/pcap2wav# ./pcap2wav.sh -h

pcap2wav is a simple utility to make it easier to extract the audio from a pcap

Dependencies:
   apt-get install -y tshark sox
   yum install wireshark sox

Usage:

  pcap2wav [opts] filename.pcap [target filename]

Script attempts to create a few files: a .<codec> file and a .wav file for each RTP stream

It requires Tshark to be installed on the system. If a codec other than PCMA or PCMU
 is used then the script will attempt to use fs_cli to decode and create a wav.

Supported codecs:
 PCMU (G711 ulaw)
 PCMA (G711 Alaw)
 GSM
 G722 (requires fs_encode)
 G729 (requres fs_encode with mod_com_g729)

Supported options:
 -z  Perform "clean and zip" - After converting to wav files the program will "clean up"
                               by putting the wav files into a .tgz file and then removing
                               the .wav and .<codec> files from the disk.
```

PCAP2WAV: Installing tshark and sox

```
root@PentesterAcademy:/work# apt-get install -y tshark sox
Reading package lists... Done
Building dependency tree
Reading state information... Done
tshark is already the newest version (2.4.4-1).
The following additional packages will be installed:
  libsox-fmt-alsa libsox-fmt-base libsox3
Suggested packages:
  libsox-fmt-all
The following NEW packages will be installed:
  libsox-fmt-alsa libsox-fmt-base libsox3 sox
0 upgraded, 4 newly installed, 0 to remove and 1826 not upgraded.
Need to get 530 kB of archives.
After this operation, 1,292 kB of additional disk space will be used.
Get:1 http://ftp.yzu.edu.tw/Linux/kali kali-rolling/main amd64 libsox3 amd64 14.4.2-3 [264 kB]
Get:2 http://ftp.yzu.edu.tw/Linux/kali kali-rolling/main amd64 libsox-fmt-alsa amd64 14.4.2-3 [51.3 kB]
Get:3 http://ftp.yzu.edu.tw/Linux/kali kali-rolling/main amd64 libsox-fmt-base amd64 14.4.2-3 [72.8 kB]
Get:4 http://ftp.yzu.edu.tw/Linux/kali kali-rolling/main amd64 sox amd64 14.4.2-3 [142 kB]
Fetched 530 kB in 6s (84.7 kB/s)
Selecting previously unselected package libsox3:amd64.
(Reading database ... 336924 files and directories currently installed.)
Preparing to unpack .../libsox3_14.4.2-3_amd64.deb ...
Unpacking libsox3:amd64 (14.4.2-3) ...
```

PCAP2WAV: Running the tool

```
root@PentesterAcademy:/work/pcap2wav# ./pcap2wav.sh SIP+RTP_call_trace_from_caller_to_PBX.pcap ./output_call.wav
Found SIP+RTP_call_trace_from_caller_to_PBX.pcap, working...
Using ./output_call.wav
Checking SIP+RTP_call_trace_from_caller_to_PBX.pcap for RTP streams...
Running as user "root" and group "root". This could be dangerous.
tshark: Lua: Error during loading:
 [string "/usr/share/wireshark/init.lua"]:44: dofile has been disabled due to running Wireshark as superuser. See https://wiki.wireshark.org/Captu
reSetup/CapturePrivileges for help in running Wireshark as an unprivileged user.
Running as user "root" and group "root". This could be dangerous.
tshark: Lua: Error during loading:
 [string "/usr/share/wireshark/init.lua"]:44: dofile has been disabled due to running Wireshark as superuser. See https://wiki.wireshark.org/Captu
reSetup/CapturePrivileges for help in running Wireshark as an unprivileged user.
Target files to create:
./output_call.wav_1.PCMU and ./output_call.wav_1.wav
./output_call.wav_2.PCMU and ./output_call.wav_2.wav

Stream 1 ssrc / port: 0x0fbb0c8d / 13080
Stream 2 ssrc / port: 0x4fcef51a / 4004

Extracting payloads 1 from 0x0fbb0c8d...
Extracting payloads 2 from 0x4fcef51a...
Combining 2 streams into a single wav file for convenience
No clean option specified - leaving .<codec> and .wav files on system.
```

PCAP2WAV: Directory contents

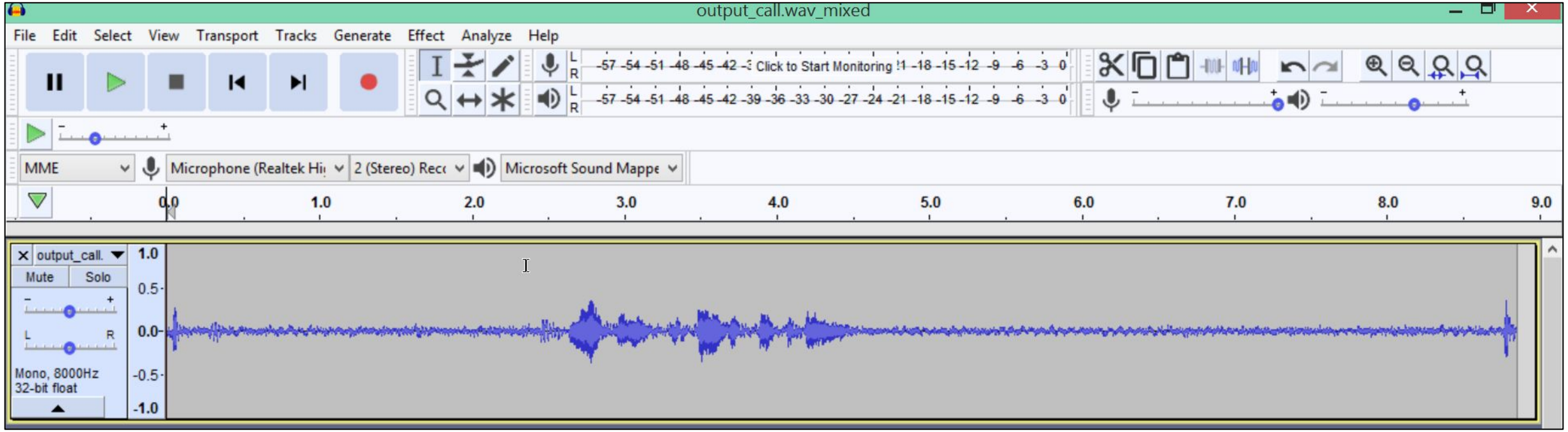
- Directory content before running the script

```
root@PentesterAcademy:/work/pcap2wav# ls -l
total 232
-rwxr-xr-x 1 root root   5927 Mar 27 01:18 pcap2wav.sh
-rw------- 1 root root 226760 Mar 19 17:29 SIP+RTP_call_trace_from_caller_to_PBX.pcap
```

- Directory content after running the script

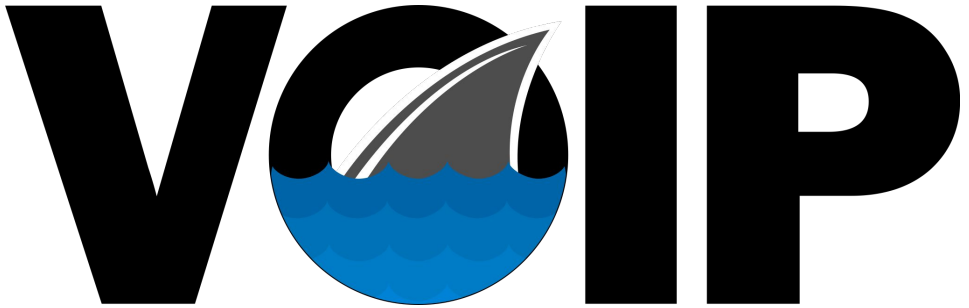
```
root@PentesterAcademy:/work/pcap2wav# ls -l
total 592
-rw-r--r-- 1 root root  70240 Mar 27 03:57 output_call.wav_1.PCMU
-rw-r--r-- 1 root root  70298 Mar 27 03:57 output_call.wav_1.wav
-rw-r--r-- 1 root root  70880 Mar 27 03:57 output_call.wav_2.PCMU
-rw-r--r-- 1 root root  70938 Mar 27 03:57 output_call.wav_2.wav
-rw-r--r-- 1 root root  70938 Mar 27 03:57 output_call.wav_mixed.wav
-rwxr-xr-x 1 root root   5927 Mar 27 01:18 pcap2wav.sh
-rw------- 1 root root 226760 Mar 19 17:29 SIP+RTP_call_trace_from_caller_to_PBX.pcap
```

PCAP2WAV: Wav in audacity



VoIPShark

- Collection of Wireshark plugins to
  - Decrypt VoIP calls
  - Export call audio
  - Overview of traffic (Extensions, SMS, DTMF)
  - Common VoIP attacks

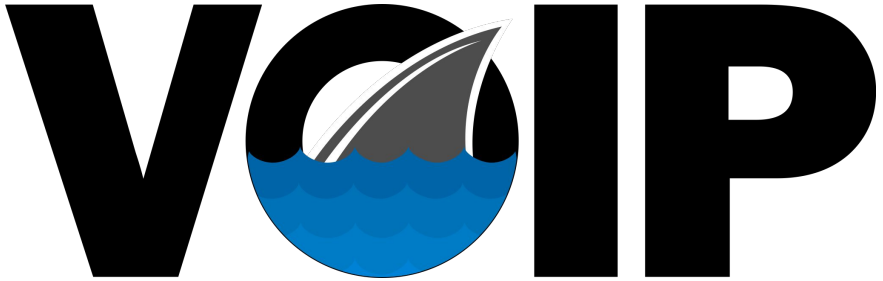


- GPL just like Wireshark

- Github: github.com/pentesteracademy/voipshark

VoIPShark: Need?

- Cumbersome and complex process

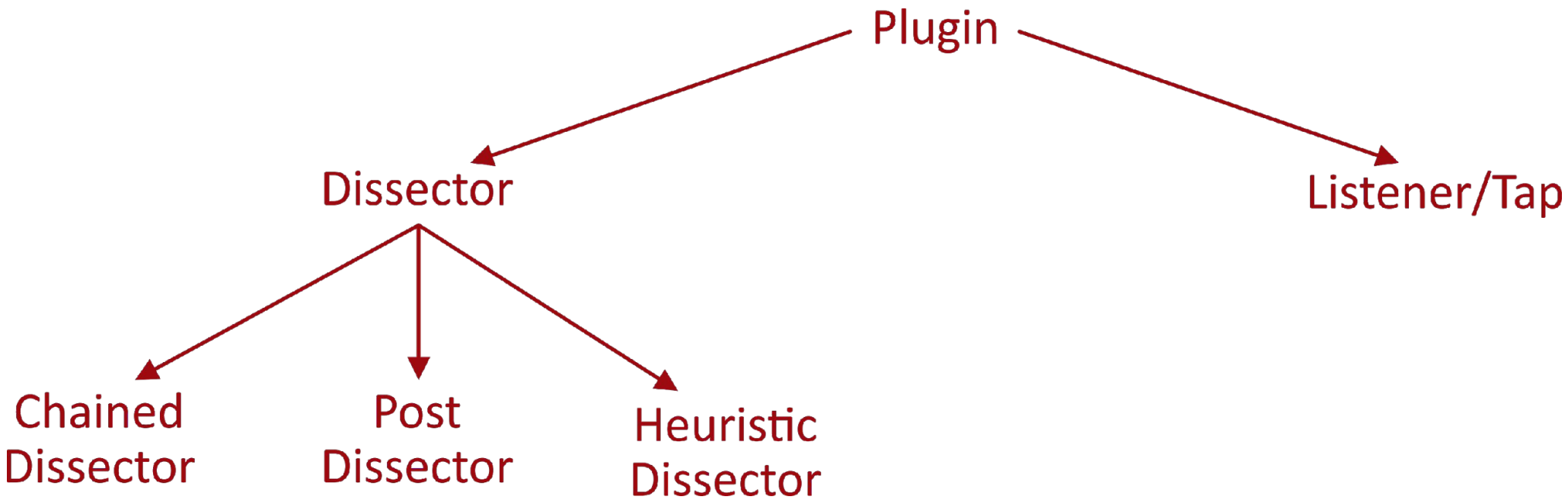


- Multiple tools
  - Need compilation, hence time consuming to set-up
  - Not easy to use
  - User dependent, prone to mistakes
- Inability to retain timestamp, IP addresses etc. during decryption
- Live traffic not supported

Why Wireshark Plugins?

- Plug and play
- Plugins can be
  - Lua scripts
  - Compiled C/C++ code
- Harnessing power of Wireshark
- OS independent
- Large user base

Wireshark Plugins Types



Dissector

- To interpret the payload data
- Decodes its part of the protocol and passes the payload to next

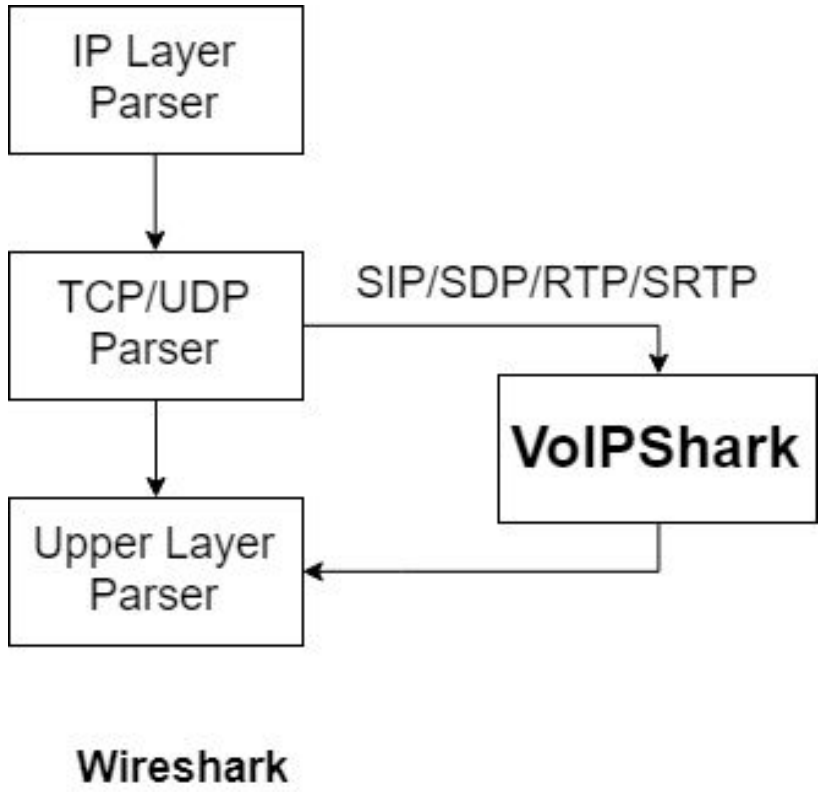
Example Dissection Flow

Chained Dissector

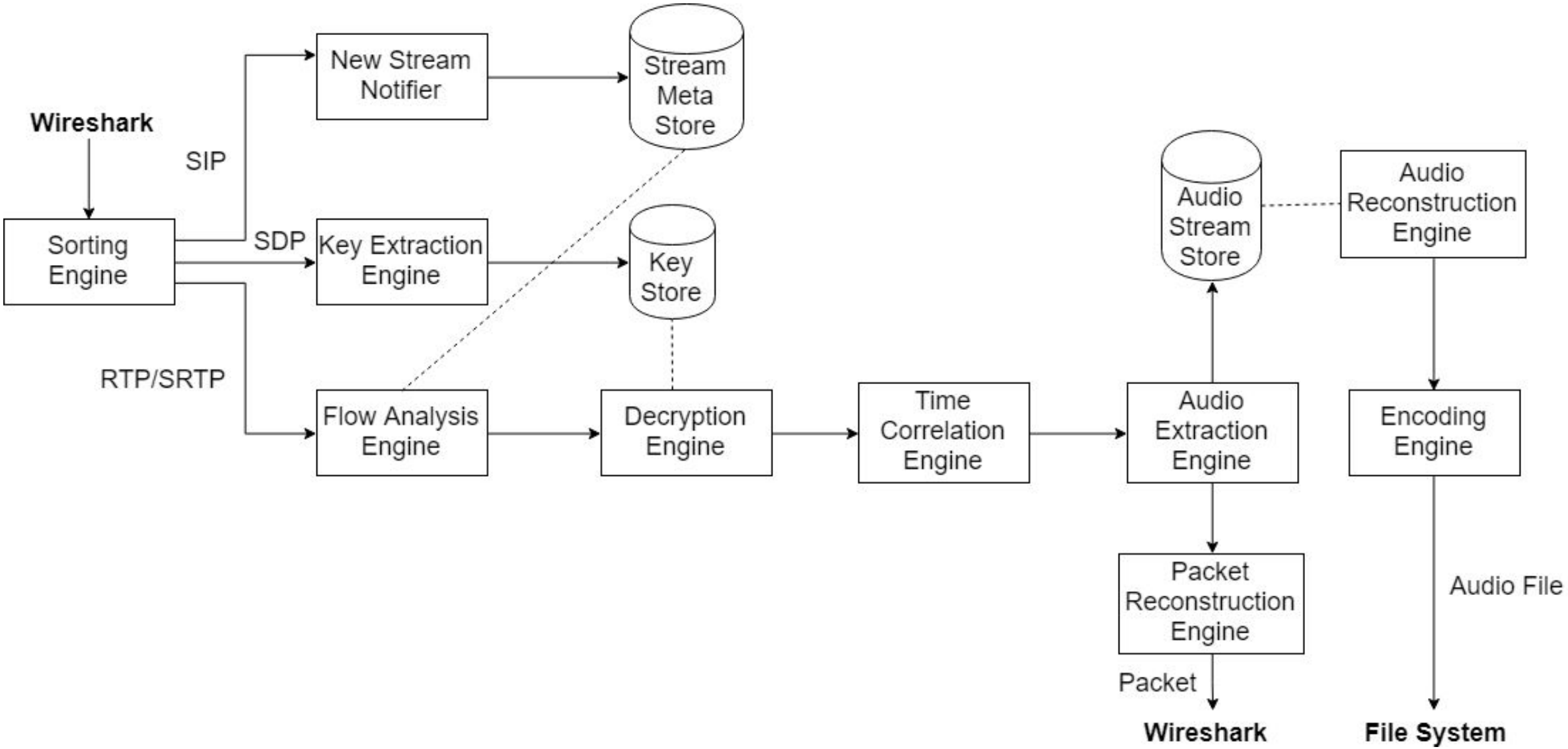
- Takes data from previous dissector, processes its part and pass the payload to next dissector

Example Dissection Flow

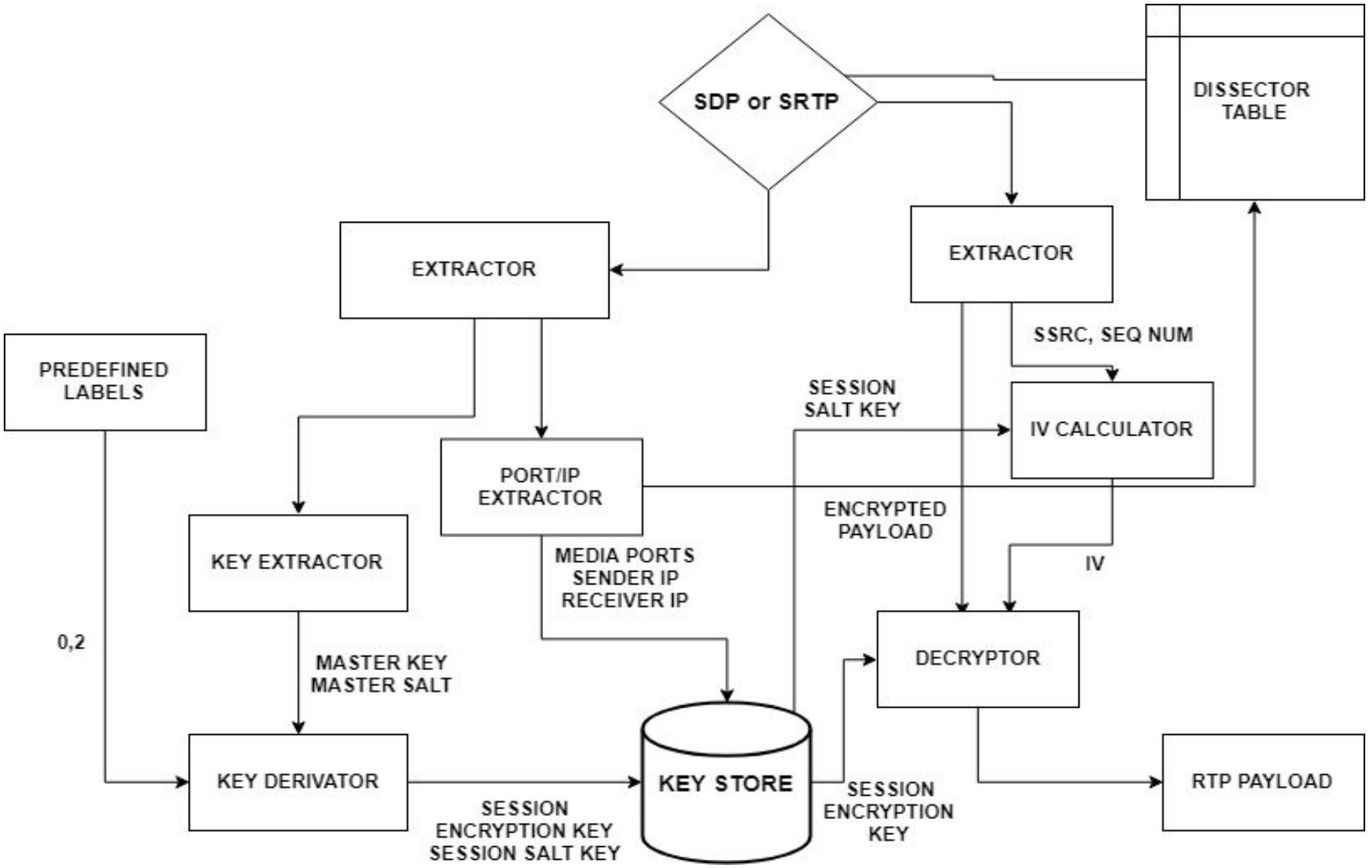
VoIPShark: Hook in Dissector Chain



VoIPShark: Overall Architecture



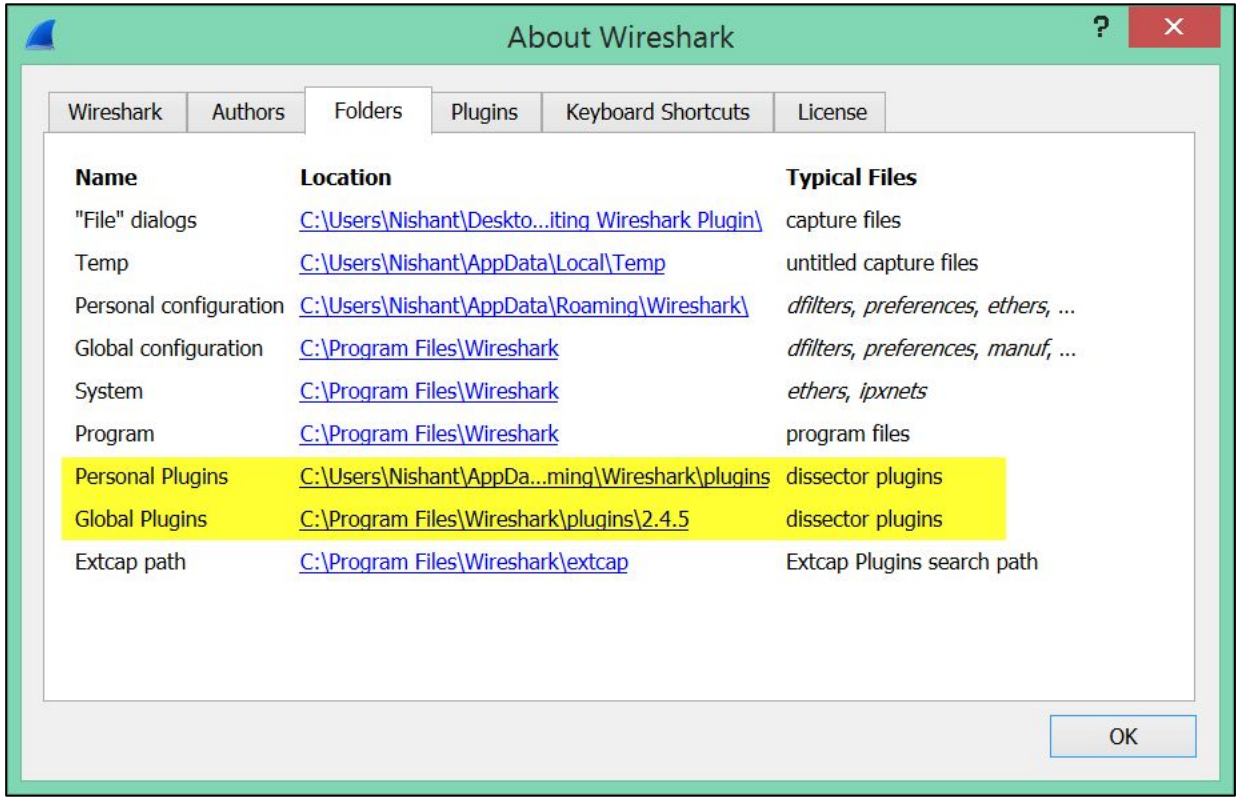
VoIPShark: Decryption Routines



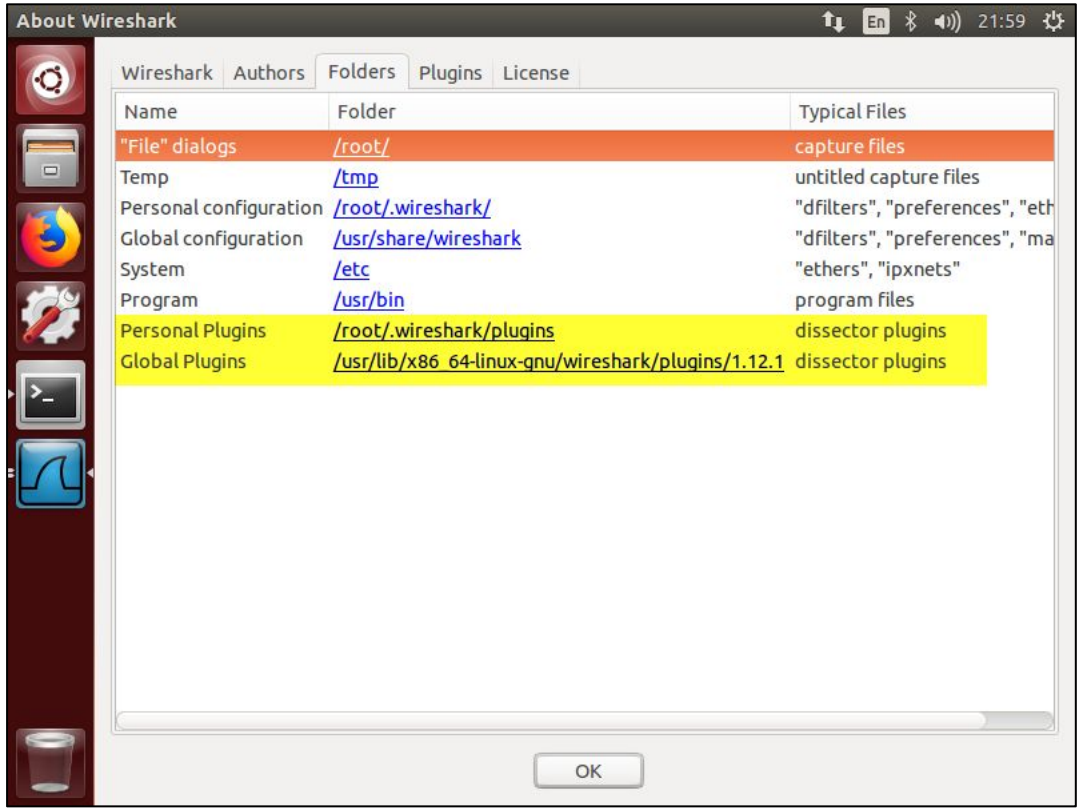
Plugins locations

- Check Help > About Wireshark > Folders

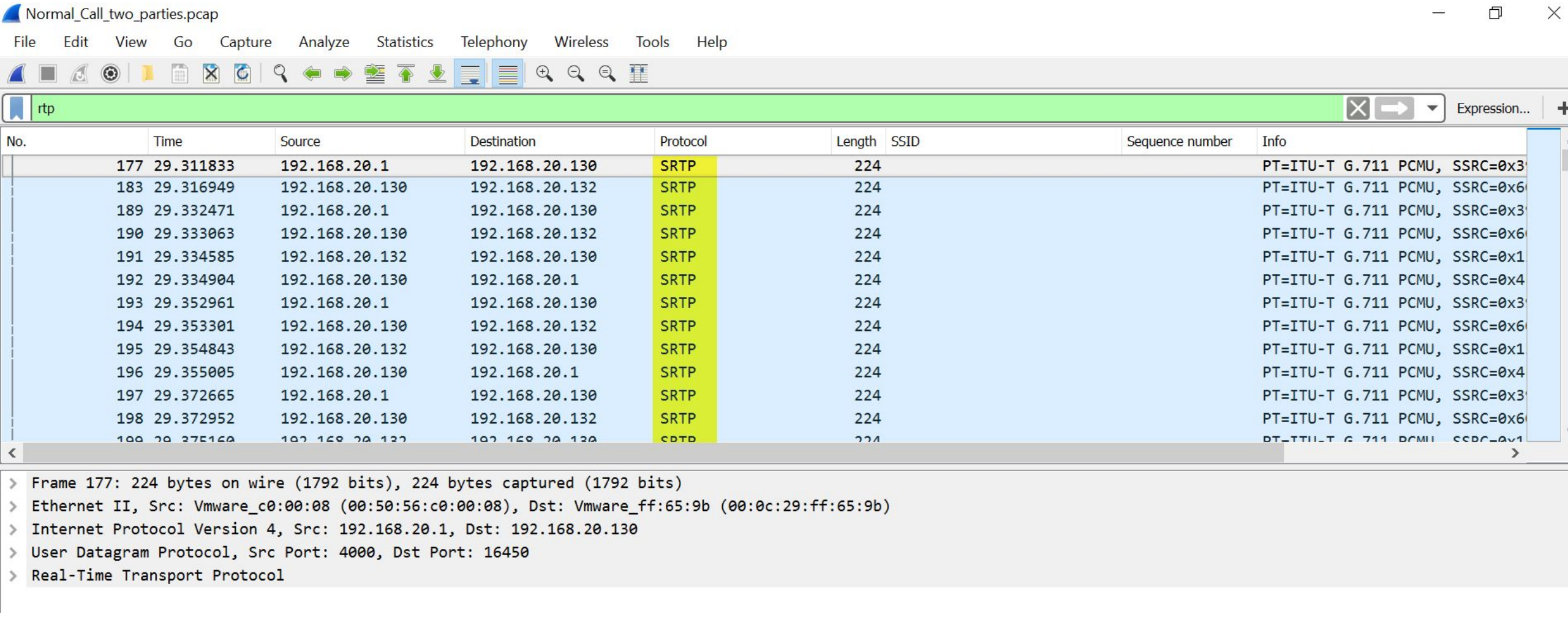
Windows



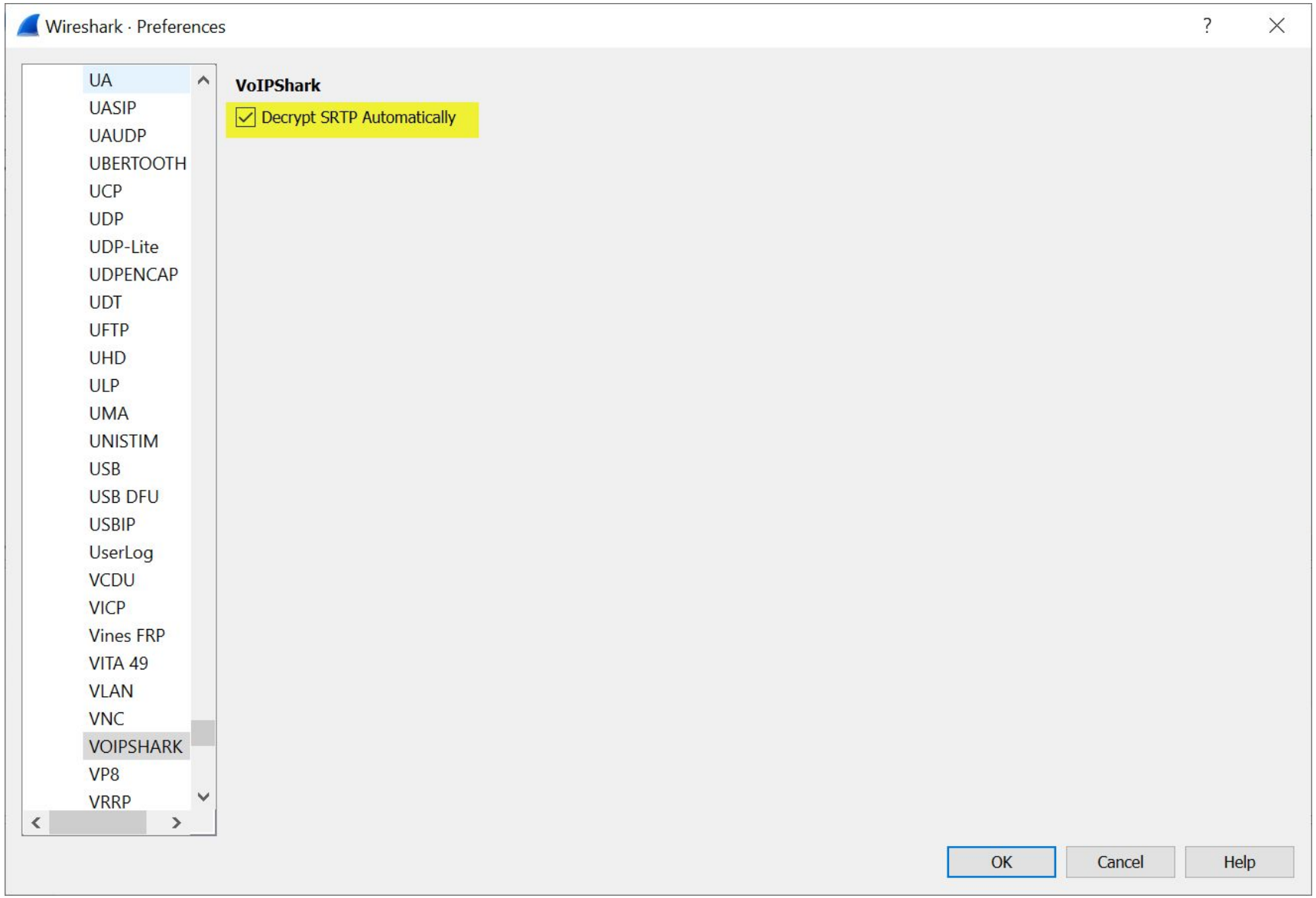
Ubuntu



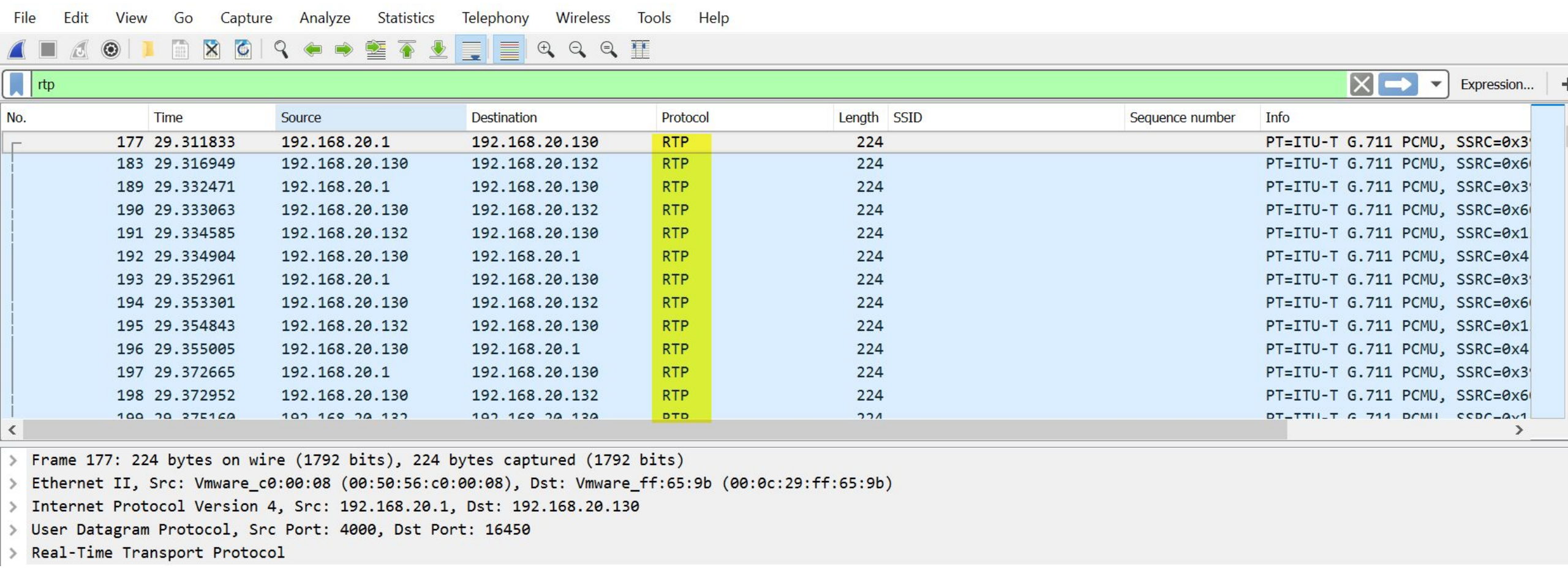
Decrypting SRTP: SRTP Packets



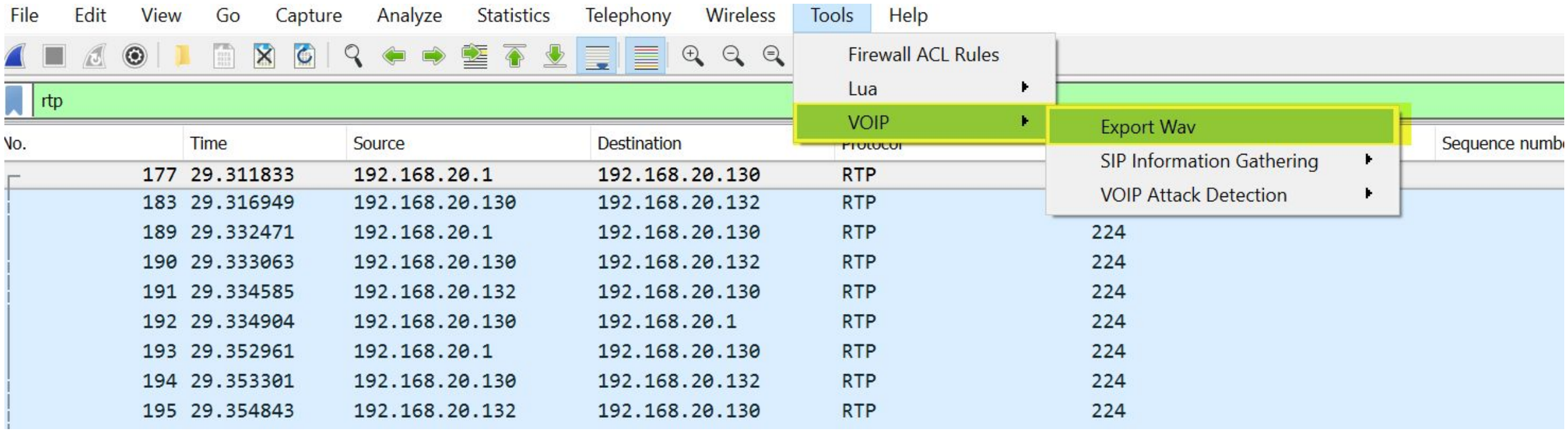
Decrypting SRTP: Enabling Auto Decryption



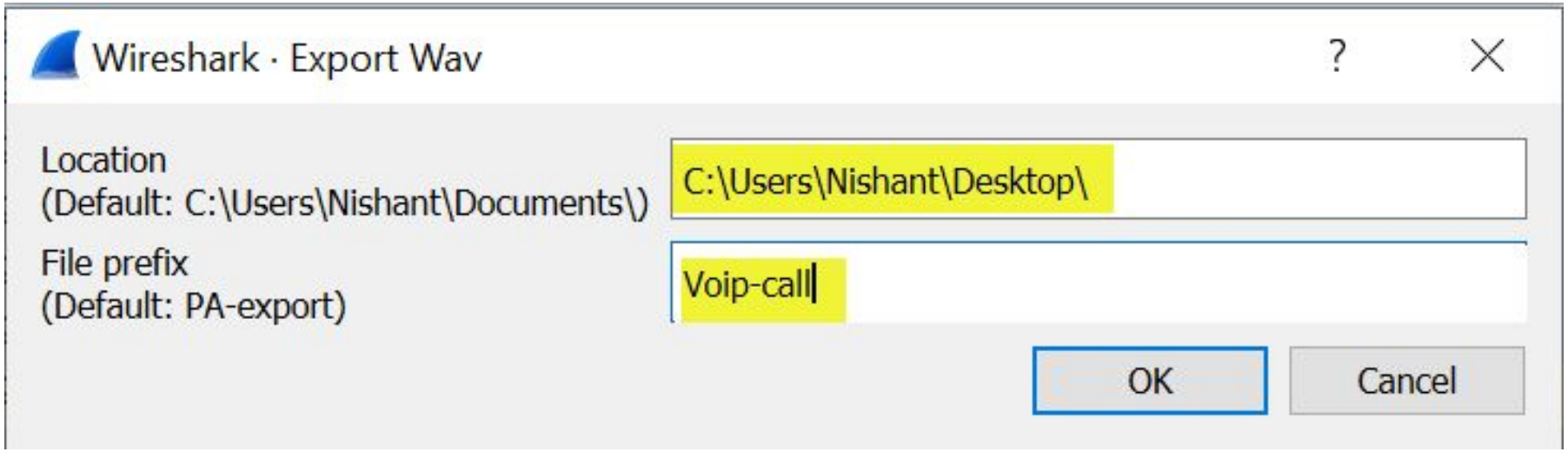
Decrypting SRTP: Decrypted SRTP (RTP)



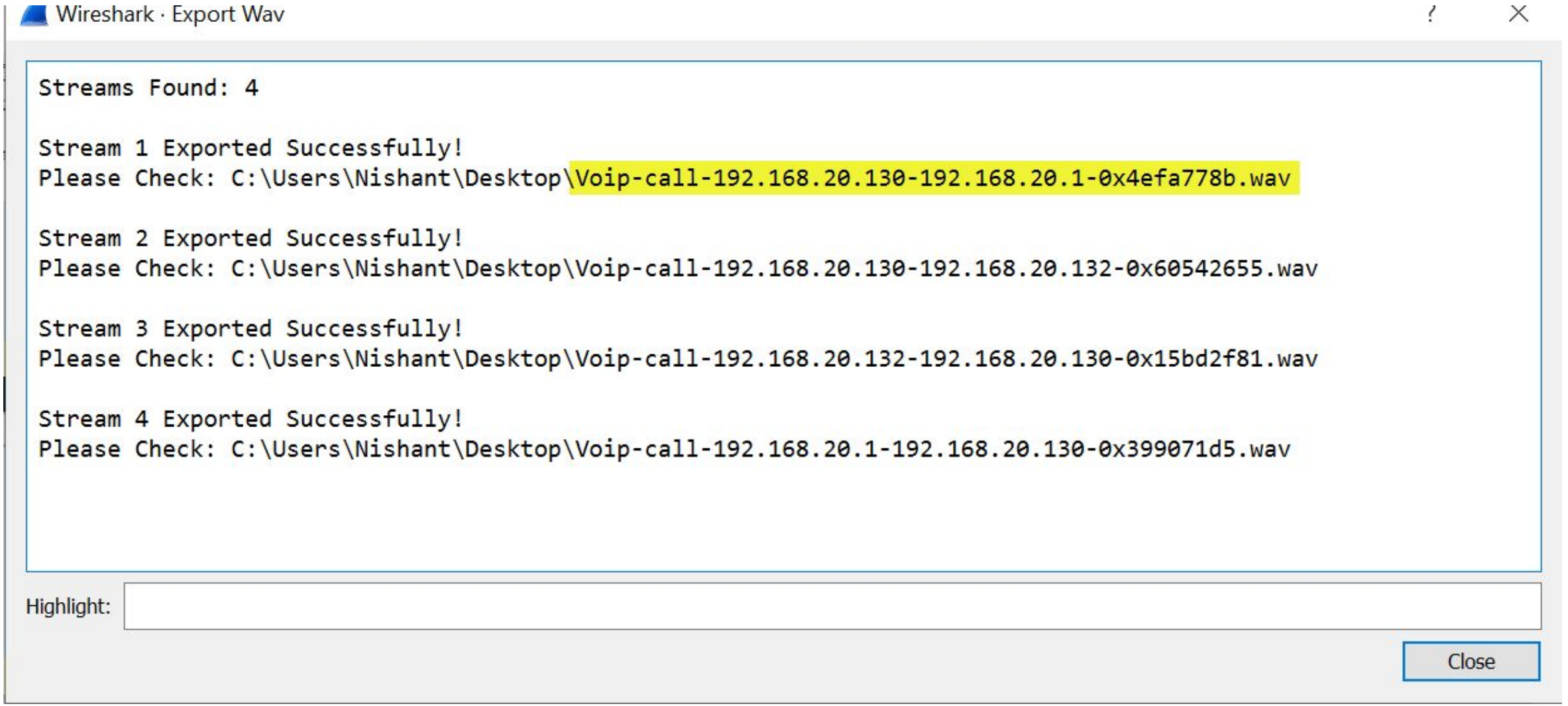
VoIPShark: Exporting Call Audio



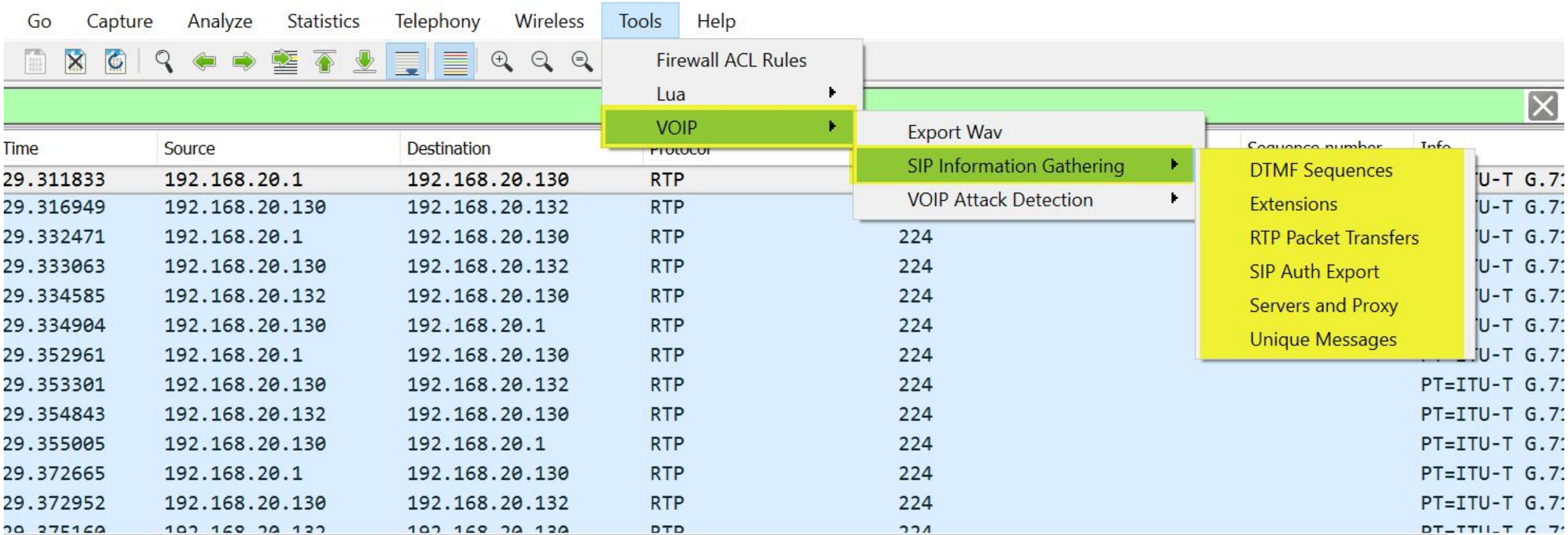
Exporting Call Audio: Specifying Location and File name



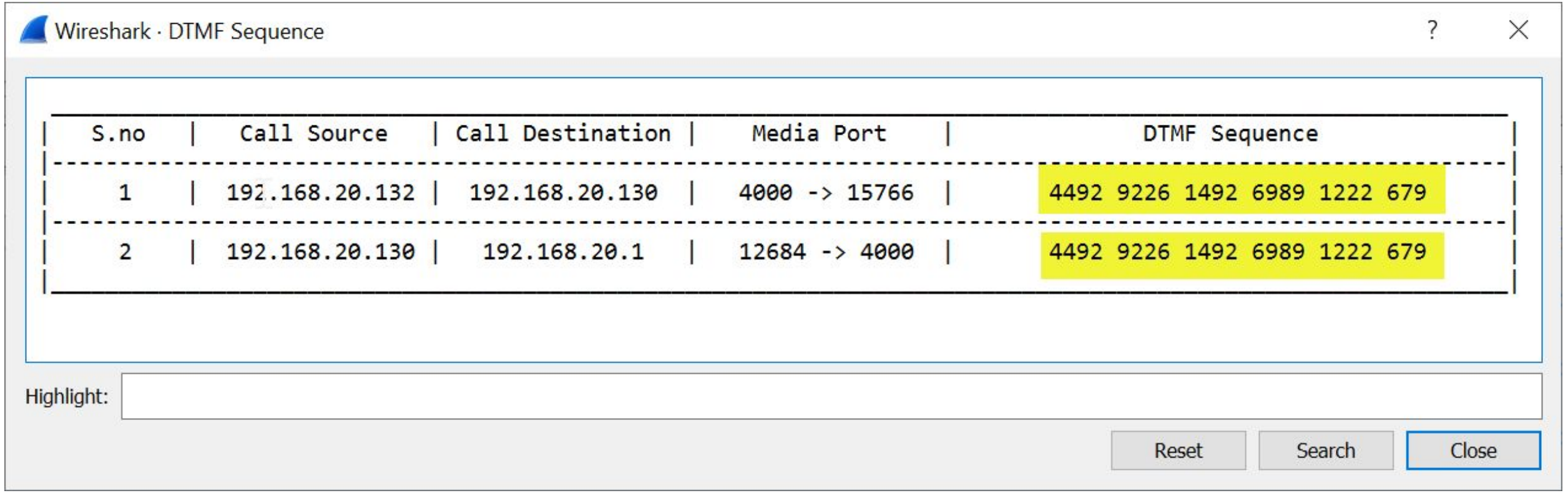
Exporting Call Audio: Exported Streams



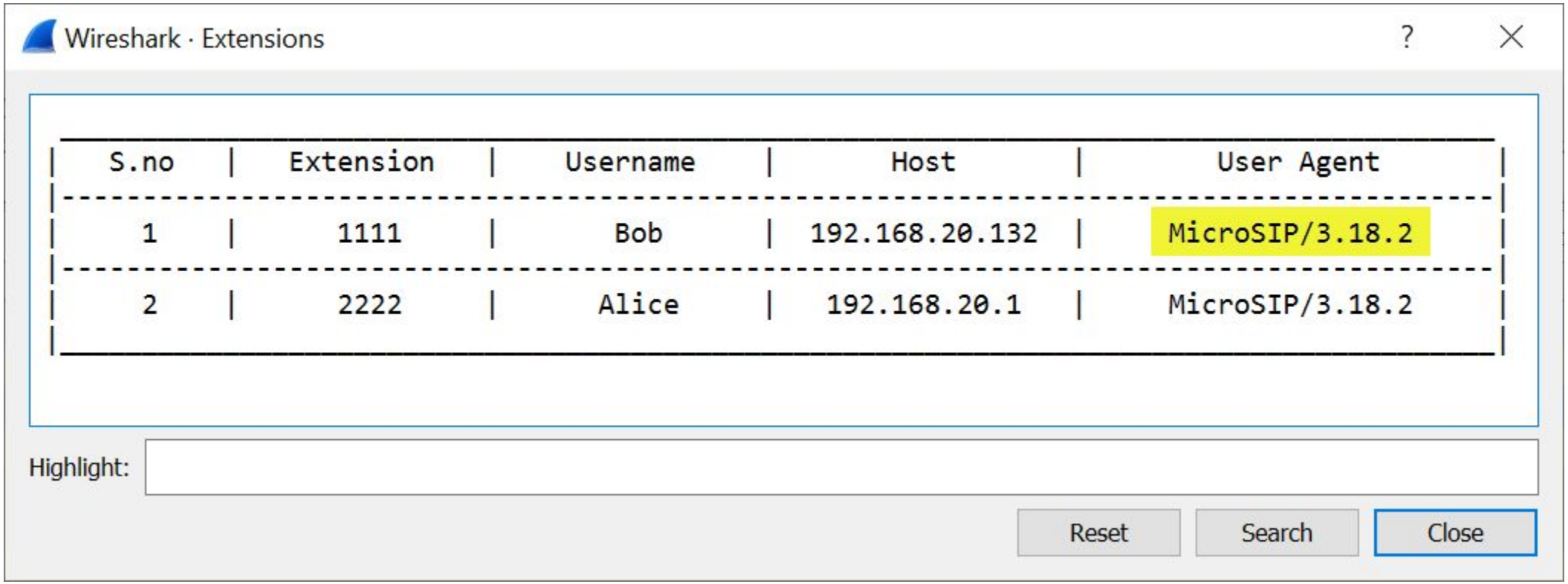
VoIPShark: SIP Information Gathering



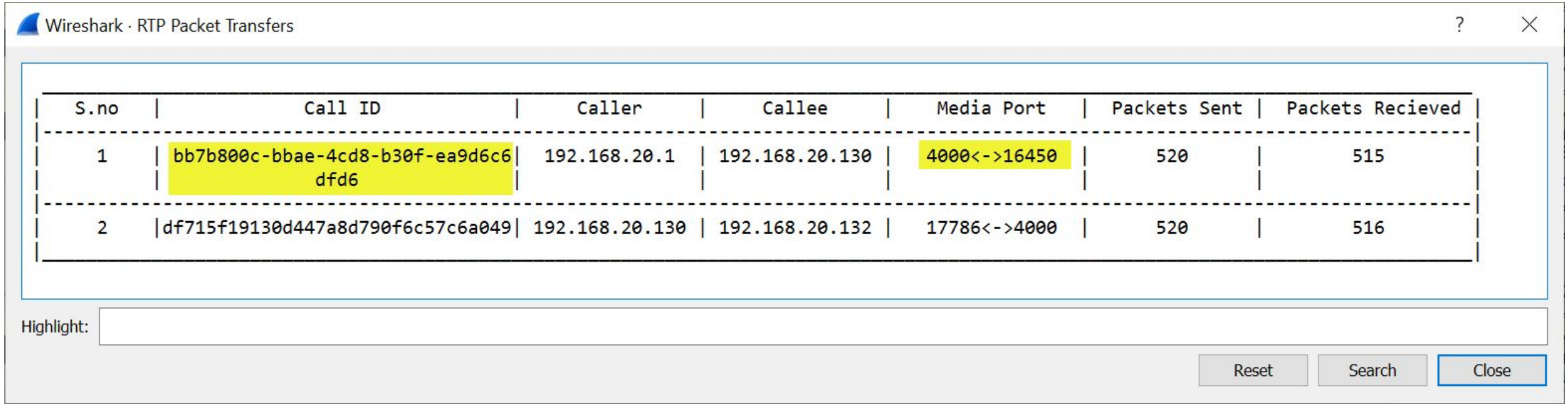
SIP Information Gathering : DTMF



SIP Information Gathering: Extensions



SIP Information Gathering: RTP Packet Transfers



SIP Information Gathering : SIP Auth Export

Wireshark · SIP Auth Export

S.no	username	Client IP	Server IP	Message Digest
1	1111	192.168.20.132	192.168.20.130	6a09af4b796d1b5ff376726fa9ae1ad9

```
$sip$***1111*asterisk*REGISTER*sip*192.168.20.130**1522268723/
f872129e9c735809884cb64de141967e*1c109c4b8a064ef5ae277c4d7d07c4d1*00000001*auth*MD5*6a09af4b796d1b5ff376726f
a9ae1ad9
```

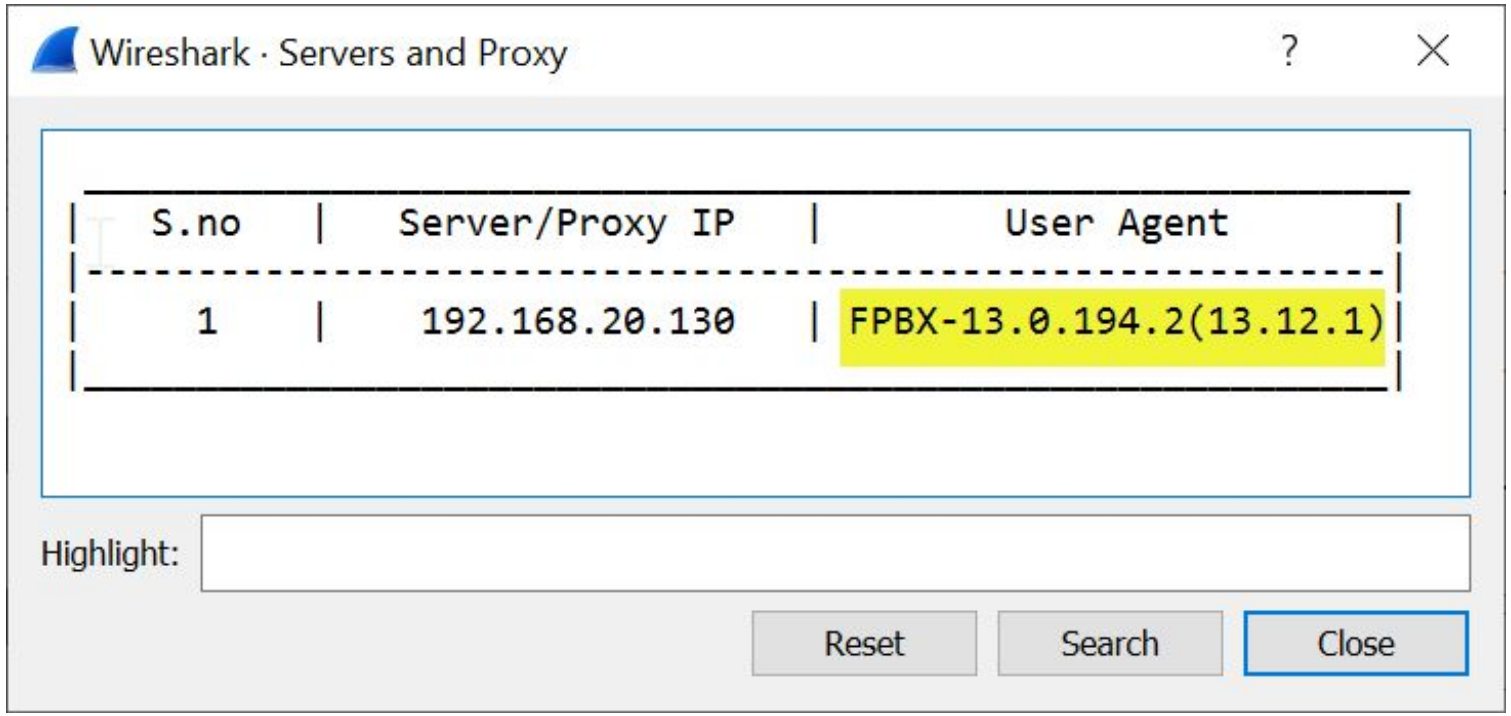
S.no	username	Client IP	Server IP	Message Digest
2	2222	192.168.20.1	192.168.20.130	f28aa9d6f10944e06f8693337fd3ba19

```
$sip$***2222*asterisk*REGISTER*sip*192.168.20.130**1522268729/
b27f0c3e27b25533a8ae9a41de712696*81aca7938c994d1d93d4abc8007095b5*00000001*auth*MD5*f28aa9d6f10944e06f869333
7fd3ba19
```

Highlight:

Reset Search Close

SIP Information Gathering : Servers and Proxy



SIP Information Gathering: Unique Messages

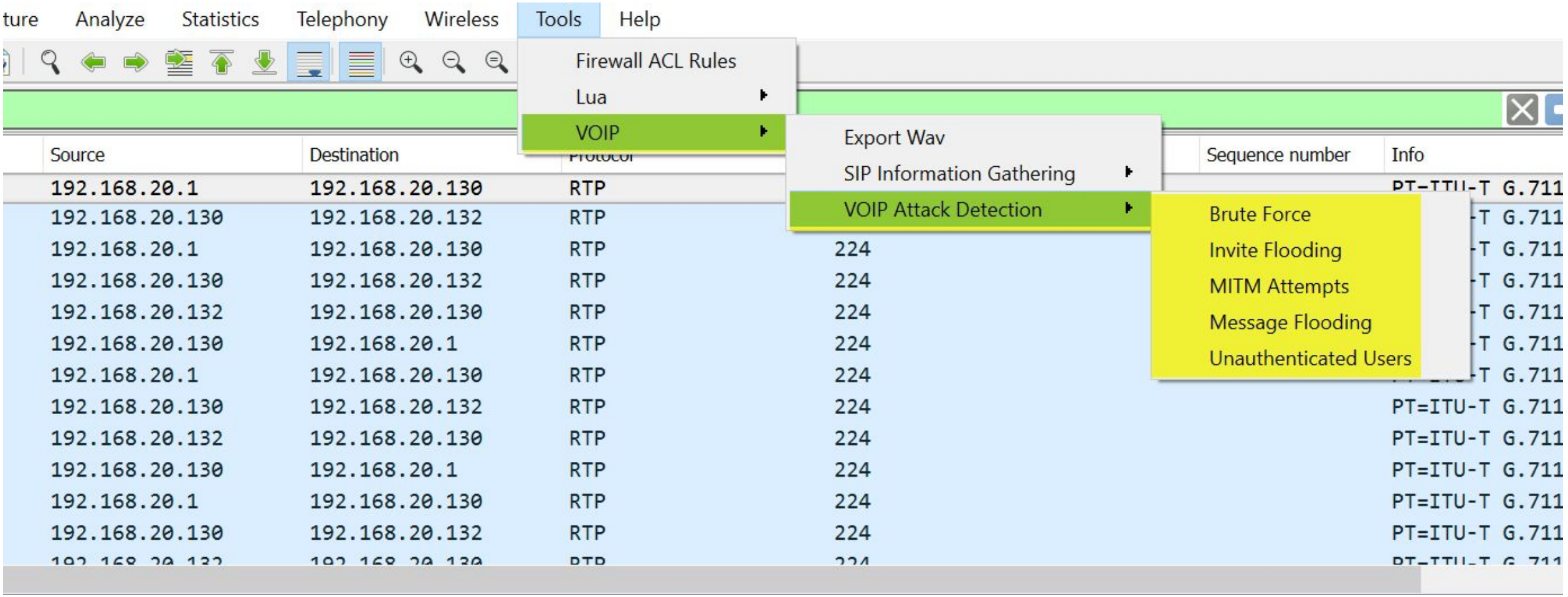
Wireshark · Unique Messages

S.no	Sender Username	Message Sender IP	Reciever Username	Message Reciever IP	Message
1	1111	192.168.20.136	2222	192.168.20.130	Hello world to sip messaging!
2	1111	192.168.20.130	2222	192.168.20.1	Hello world to sip messaging!

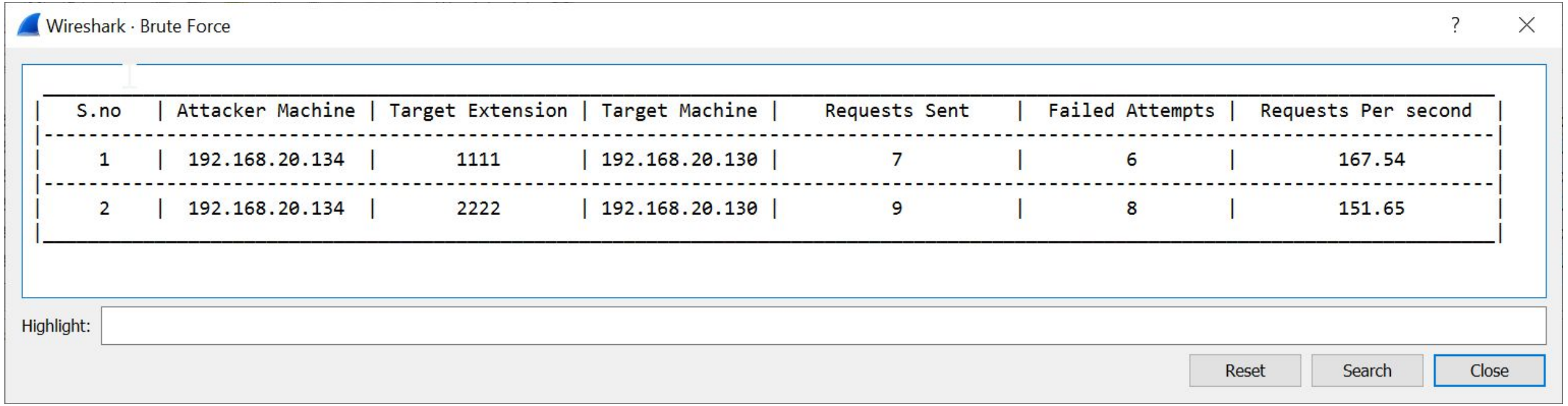
Highlight:

Reset Search Close

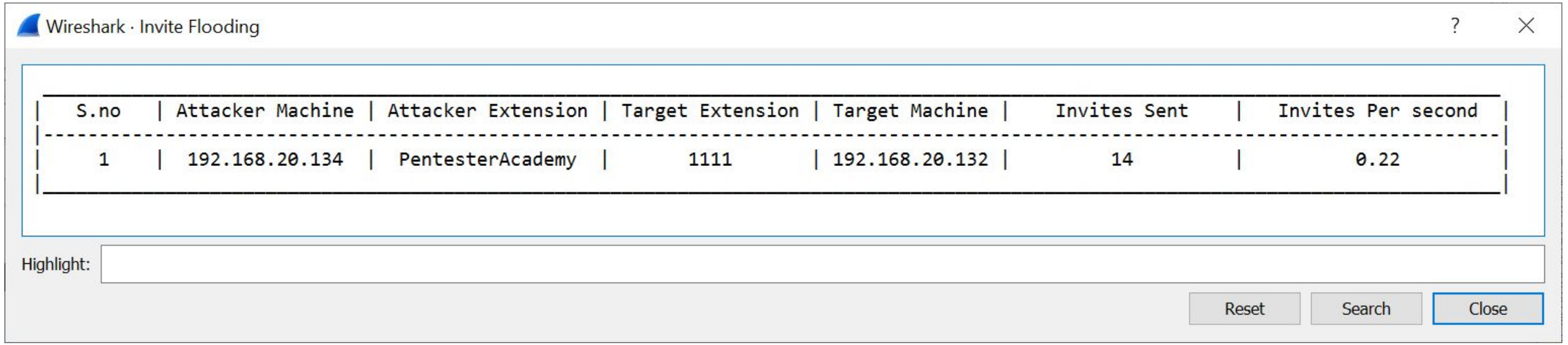
VoIPShark: VoIP Attack Detection



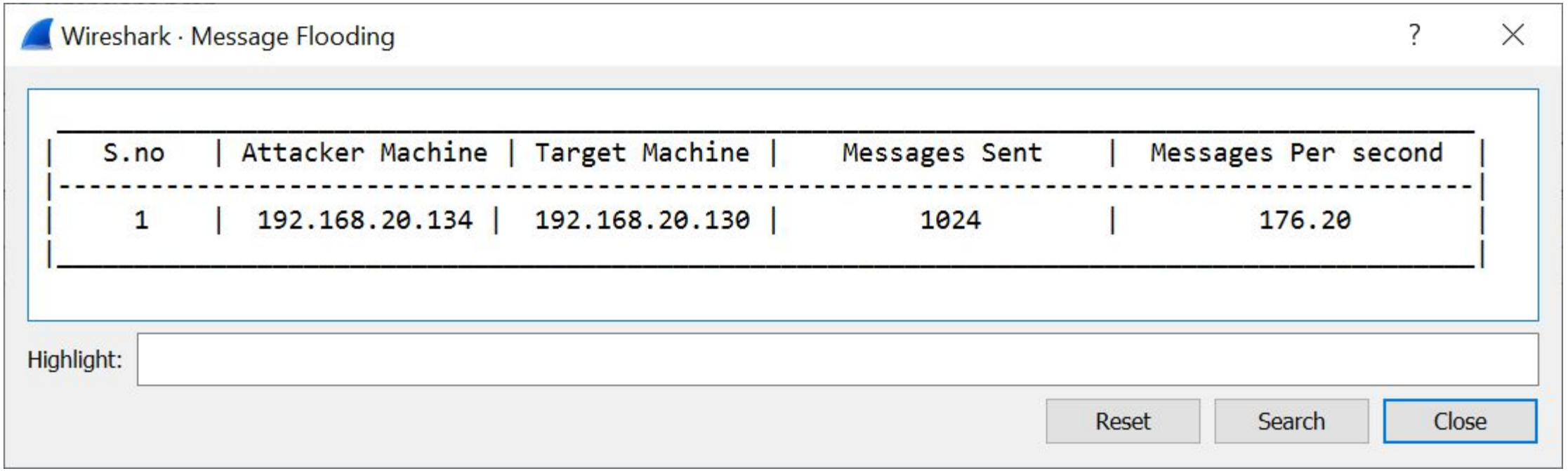
VoIP Attack Detection: Bruteforce



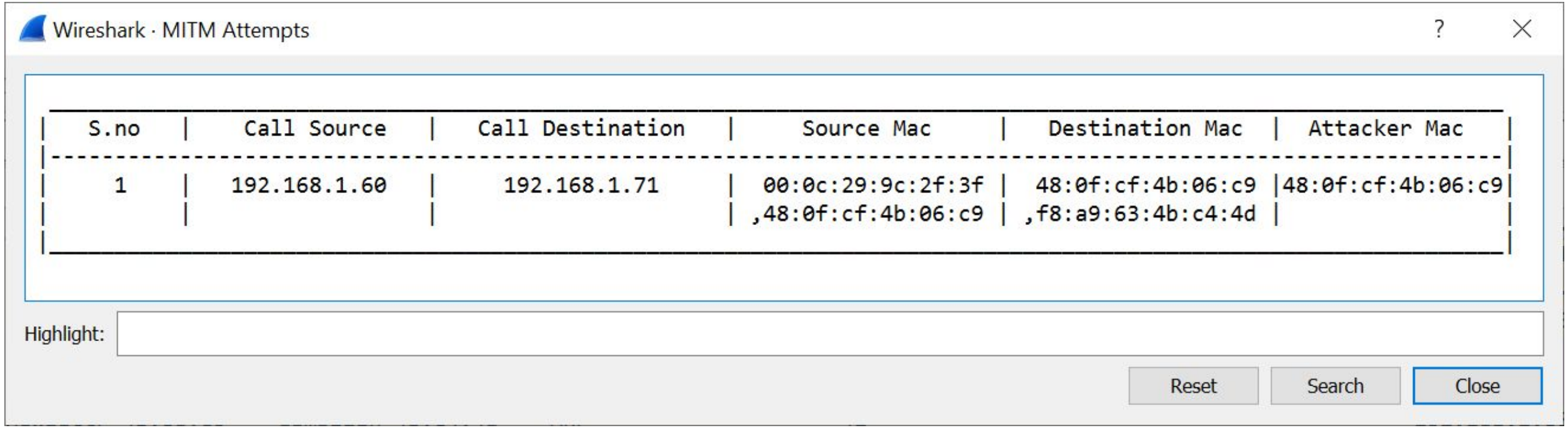
VoIP Attack Detection: Invite Flooding



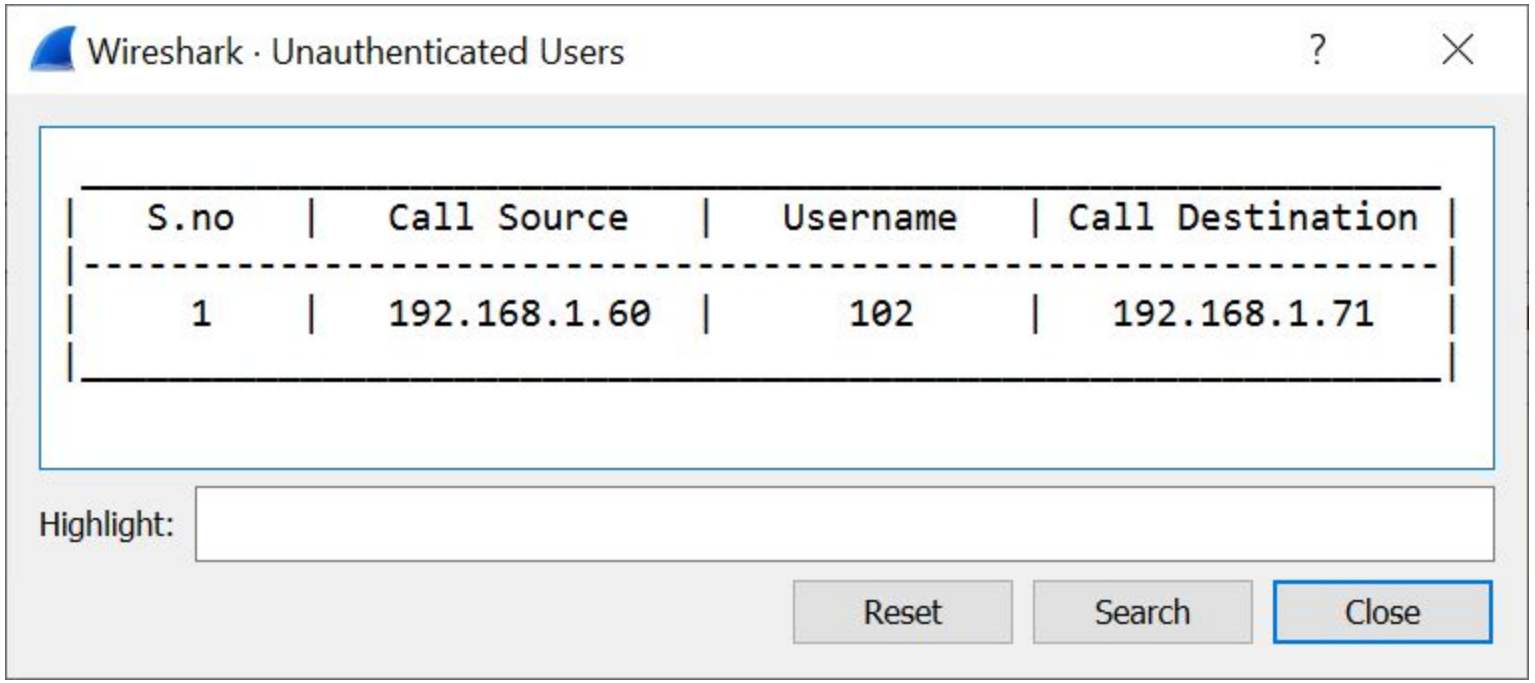
VoIP Attack Detection: Message Flooding



VoIP Attack Detection: MiTM Attempts



VoIP Attack Detection: Unauthenticated Users



Demo

Q & A

Github: github.com/pentesteracademy/voipshark
nishant@attackdefense.com