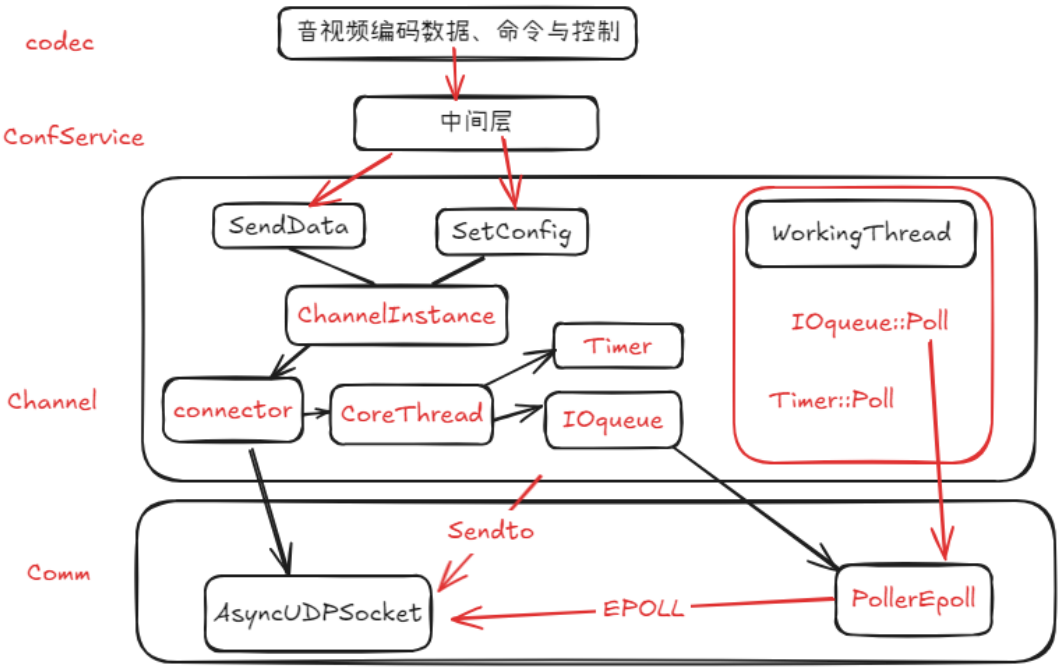
1.大致流程

VOIP发送框架示意图，其中各种回调和定时器未画出。黑色箭头表示持有关系，对象之间主要通过hashtable或固定偏移持有。通过socket发送数据，sockfd、epollfd、connid等通过hashtable、红黑树、固定偏移相互关联。红色箭头表示函数调用关系。



和音视频有关的数据，java层产生并在Codec中进行编码，同时Codec中还会发送许多控制信息，音视频数据和音视频控制信息由Codec交付到中间层。

中间层通过Channel中的全局对象channel_Instance负责直接和Channel交互。通过调用channel_Instance的虚函数SendData把音视频编码或控制数据的原始buf传递给Channel，Channel中根据各种参数信息来对数据进行加密。协议封装完成后，通过调用IOqueue的Sendto方法，把待发送数据加入到comm中AsyncUDPSocket的消息队列中，并通过epoll_ctl把对应sockfd修改为EPOLLIN | EPOLLOUT使得该AsyncUDPSocket可以分发写事件。

同时Channel中有一个独立线程WorkingThread，通过循环不断调用IOqueue的Poll方法来控制PollerEpoll执行epoll_wait来检测有哪些AsyncUDPSocket有读写事件需要处理，并协助AsyncUDPSocket进行数据发生和接收，通过回调往上层上报执行情况。数据发送后通过epoll_ctl禁用对应sockfd的EPOLLOUT。

2.音视频协议类型

微信VOIP中报文种类较多，主要分为Relay经过服务器中转的类型，和Direct客户端通过nat穿透直接传输的类型。比如调用HandleRelayPackets解析中转类型时，是根据报文第一个字节来解析报文类型。根据中转报文首字节，又有数十种报文类型。

D0、D1、D5主要是服务器的命令与控制报文

```
if ( (sub_721A8(v11) & 1) != 0 )
{
  v19 = *(unsigned __int16 *)(RelayRoom + 42);
  if ( v19 <= 0xA && ((1 << v19) & 0x44A) != 0 )
    *(_WORD *)(RelayRoom + 1314) = 0;
  HandleRelaySvrCmdPacket(RelayMgr, v13, *v9, RelayRoom, addr, conn_id, zero & 1);
  if ( ptr )
    free(ptr);
}
else
{
  if ( (sub_72108(v11) & 1) == 0 )
  {
    switch ( v11 )
    {
      case 0x75u:
        ParseRelayDataPktChaChaV3(&v42, v13, v9, ConnInfo, &v37, &v40, &v39, (char *)&v37 + 4, (char *)&v38 + 4);
        goto LABEL_43;
```

其他首字节对应不同的加密方式，其中和音视频相关的主要是96和98，采用chacha20_poly1305加密方法

```
case 0x96u:
case 0x98u:
  ParseRelayDataPktChaChaLive(
    &v42,
    v13,
    v9,
    ConnInfo,
    &v37,
    &v35,
    &v40,
    &v39,
    (char *)&v37 + 4,
    (char *)&v38 + 4,
    &v36);
  goto LABEL_43;
case 0x97u:
  ParseRelayDataPktChaChaNew(&v42, v13, v9, ConnInfo,
  goto LABEL_43;
default:
  if ( v11 == 215 )                    // d7
  {
    ParseRelayDataPkt64bit(
      &v42,
      v13,
      v9,
      ConnInfo,
      &v37,
      &v40,
      &v39,
      (char *)&v37 + 4,
      &v38,
      (char *)&v38 + 4);
```

3.音视频相关的报文

3.1 数据处理流程

着重分析一下协议头和加密算法是否存在弱点。我们前面提起过VOIP架构，主要是通过ChannelInstance的senddata交付数据，senddata中调用SendConn

```
v20 = SendConn(
        channelInstance + 440,
        connid,
        (__int64)v42,
        v41,
        HIDWORD(v39),          // 音频流0x76
                               // 控制流0x83
        a6,
        v37 & 1,
        0,
        v36,
        a10);                  //
```

SendConn中再次分流，中转类型的音视频数据主要是RelaySendData

```
    v23 = RelaySendTcpData(
            *(_QWORD *)(RelayRoom + 816),
            RelayRoom,
            (__int64)data,
            data_len,
            cloud_data_type,
            a6,
            a7 & 1,
            connector,
            a9,
            a10);
    goto LABEL_20;
  }
  return 0;
}
if ( cloud_data_type - 80 <= 4 && ((1 << (cloud_data_type - 80)) & 0x15) != 0 )
  v23 = RelaySvrSendARQ_FEC(
          *(_QWORD *)(RelayRoom + 816),
          RelayRoom,
          (__int64)data,
          data_len,
          cloud_data_type,
          connector,
          (__int64)OnSent);
else
  v23 = RelaySendData(
          *(_QWORD *)(RelayRoom + 816),
          RelayRoom,
          data,
          data_len,
          cloud_data_type,
          a6,
          a7 & 1,
          connector,
          (__int64)OnSent,
          a8,
          a9,
```

RelaySendData中调用PackRelayPktChaChaLive，PackRelayPktChaChaLive根据业务类型再次分流，同时报文头部也在PackRelayPktChaChaLive这里确定

```
if ( business_type == 8
  || business_type == 10
  || business_type == 200
  || cloud_data_type == 118 && business_type == 6 && v20 == 5
  || (v36 = business_type == 6, cloud_data_type == 131)
  || (unsigned int)(cloud_data_type - 119) < 3 && v36 && *(_WORD *)(RelayRoom + 138) == 5 )
{
  PackRelayPktChaChaLive(
    RelayRoom,
    *(__int64 **)(RelayMgr + 16),
    cloud_data_type,
    data,
    data_len,
    a6,
    a7 & 1,
    a11,
    &buf0,
    a12,
    &buf_size);                          // chacah20加密
```

3.2 中转协议分析

3.2.1 PackLiveCloud和中转控制协议

PackRelayPktChaChaLive中首先判断business_type，当business_type是10时

```
{
  if ( business_type == 10 )
  {
    v25 = *(_DWORD *)(RelayRoom + 1064);
    room_id = *(_QWORD *)(RelayRoom + 16);
    v27 = *(_WORD *)(RelayRoom + 40);
    *(_WORD *)v47 = 5014;
    *(_WORD *)&v47[12] = v25;
    *(_QWORD *)&v47[2] = room_id;
    *(_WORD *)&v47[10] = v27;
    v28 = 16;
    switch ( cloud_data_type )
    {
      case 0x76u:
        break;
      case 0x77u:
      case 0x78u:
        v28 = 32;
        break;
      case 0x79u:
        v28 = 96;
        break;
```

会调用PackLiveCloud，其中RelayRoom + 1904开始就是32字节的chachakey，然后是Data和datasize，后面两个是接收加密后数据的outarray和arraysize，v47就是和协议头部有关一些信息，这里重命名为ConfigArray

```
conn_id = *(_DWORD *)(RelayRoom + 780);
BYTE2(v48) = a7 & 1;
v47[14] = v28 | a6 & 0xF;
HIBYTE(v48) = conn_id;
v39 = RelayRoom + 2LL * (unsigned int)TypeMaptoArrayIndex(cloud_data_type);
LOWORD(v48) = *(_WORD *)(v39 + 1084);
*(_WORD *)(v39 + 1084) = v48 + 1;
ret_1 = PackLiveCloud(RelayRoom + 1904, data, v11, (__int64)v19, buf_size, (__int128 *)v47);
if ( ret_1 )
```

在PackLiveCloud中又存在调用链：sub_A3BA8-->sub_A3A70-->sub_A4888-->sub_A49F0，这里sub_A3BA8第一个参数用来接收加密数据，且从第20字节开始接收，因为前19字节是协议头部，协议头部的来源就是configArray的前19字节

```
v22[1] = *(_QWORD *)(_ReadStatusReg(TPIDR_EL0) + 40);
if ( a2 && a3 >= 1 && outarray )
{
  v7 = *((unsigned __int8 *)configArray + 14);
  v11 = v7 & 0xF0;
  if ( v11 == 96 )
  {
    v12 = v7 & 0xF | 0x20;
    v7 = v12;
    *((_BYTE *)configArray + 14) = v12;
  }
  v13 = *((_WORD *)configArray + 5);
  v22[0] = *(_QWORD *)(a1 + 32) + (v7 << 32) + *(unsigned __int16 *)((char *)configArray + 15);
  v20 = v13;
  if ( v11 == 96 )
    *((_BYTE *)configArray + 14) = v7 & 0xF | 0x60;
  v14 = (unsigned int)(a3 + 2);
  v15 = *configArray;
  v16 = (unsigned int)a3;
  *(_DWORD *)(outarray + 15) = *(_DWORD *)((char *)configArray + 15);
  *(_OWORD *)outarray = v15;
  v17 = (_WORD *)operator new[](v14);
  *v17 = v20;
  memcpy(v17 + 1, a2, v16);
  v21 = 0;
  v18 = sub_A3BA8(outarray + 19, &v21, (__int64)v17, v14, 0, 0, 0, (__int64)v22, a1);
  *a5 = v21 + 19;
  operator delete[](v17);
}
```

调用链到sub_A49F0，这里只是chacha20加密初始状态字的生成，然后调用subA4AC0真正加密数据，初始状态字生成这段代码没法反编译，只能看汇编

```
loc_A49F0                               ; CODE XREF: sub_A4888+C↑j
                                        ; sub_A4898+24↑j
                                        ; DATA XREF: ...
; __unwind {
                MOV             X8, X2
                CBZ             X2, loc_A4A50
                SUB             SP, SP, #0x50 ; 'P'
                STP             X29, X30, [SP,#0x50+var_10]
                ADD             X29, SP, #0x50+var_10
                ADRP            X9, #xmmword_21B70@PAGE
                MOV             X2, X0
                LDR             Q3, [X5,#0x10]
                LSR             X10, X4, #0x20 ; ' '
                LDR             Q0, [X5]
                MOV             X0, SP
                LDR             Q1, [X9,#xmmword_21B70@PAGEOFF]
                LDR             D2, [X3]
                MOV             X3, X8
                STP             W4, W10, [SP,#0x50+var_20]
                STR             Q1, [SP,#0x50+var_50]
                STP             Q0, Q3, [SP,#0x50+var_40]
                STR             D2, [SP,#0x50+var_18]
                BL              sub_A4AC0
                MOV             X0, SP  ; int
                MOV             W1, #0x40 ; '@' ; n
                BL              sub_AF308
                LDP             X29, X30, [SP,#0x50+var_10]
                ADD             SP, SP, #0x50 ; 'P'

loc_A4A50                               ; CODE XREF: sub_A4898+15C↑j
                MOV             W0, WZR
                RET
; } // starts at A49F0
```

sub_A4AC0中实现了加密密钥流生成

```
do
{
  v28 = v12 + v15;
  v29 = v11 + v16;
  v30 = v13 + v17;
  v31 = v14 + v18;
  v32 = v28 ^ v26;
  v33 = v29 ^ v25;
  v34 = v30 ^ v24;
  v35 = v31 ^ v23;
  v21 += 2;
  HIDWORD(v37) = v32;
  LODWORD(v37) = v32;
  v36 = v37 >> 16;
  HIDWORD(v37) = v33;
  LODWORD(v37) = v33;
  v38 = v37 >> 16;
  HIDWORD(v37) = v34;
  LODWORD(v37) = v34;
  v39 = v37 >> 16;
  HIDWORD(v37) = v35;
  LODWORD(v37) = v35;
  v40 = v36 + v19;
  v41 = v38 + v20;
  v42 = v39 + v22;
  v43 = (v37 >> 16) + v27;
  v44 = v40 ^ v15;
  v45 = v41 ^ v16;
  v46 = v42 ^ v17;
```

用密钥流异或就是chacha20加密。根据这些信息可以推导出sub_A4AC0的参数：a1是64字节状态数组经过变换生成加密密钥，密钥和明文数组a2逐字节异或，结果存储在a3中，a4是长度，

```
  v105 = v12 + v136;
  v106 = a2[1] ^ (v11 + v135);
  v107 = a2[2] ^ (v13 + v134);
  v108 = a2[3] ^ (v14 + v133);
  v109 = a2[4] ^ (v15 + v132);
  v110 = a2[5] ^ (v16 + v131);
  v111 = a2[6] ^ (v17 + v130);
  v112 = a2[7] ^ (v18 + v129);
  v113 = a2[11];
  result = a2[8] ^ (unsigned int)(v19 + v128);
  v115 = a2[9] ^ (v20 + v127);
  v116 = a2[12];
  v117 = a2[13];
  v118 = a2[10] ^ (v22 + v126);
  v119 = a2[14];
  v120 = a2[15];
  *v10 = *a2 ^ v105;
  v10[1] = v106;
  v10[2] = v107;
  v10[3] = v108;
  v121 = v116 ^ (v26 + v4);
  v10[4] = v109;
  v10[5] = v110;
  v122 = __CFADD__(v4++, 1);
  v10[12] = v121;
  v10[13] = v117 ^ (v25 + v5);
  if ( v122 )
    ++v5;
  v10[6] = v111;
  v10[7] = v112;
  v10[8] = result;
  v10[9] = v115;
  v10[10] = v118;
```

A4A38处的 `BL ub_A4AC0` 这里跳转时x0-x3这四个寄存器的内容：跳转后x3是长度，x3是x8给的，x8又是x2给的，x2在arm64架构表示第三个参数，那么三个参数是data_len；同样x2是密文数组，以此类推还原出函数的参数

```
.text:0000000000A49F0 ; __unwind {
.text:0000000000A49F0                 MOV             X8, X2
.text:0000000000A49F4                 CBZ             X2, loc_A4A50
.text:0000000000A49F8                 SUB             SP, SP, #0x50 ; 'P'
.text:0000000000A49FC                 STP             X29, X30, [SP,#0x50+var_10]
.text:0000000000A4A00                 ADD             X29, SP, #0x50+var_10
.text:0000000000A4A04                 ADRP            X9, #xmmword_21B70@PAGE
.text:0000000000A4A08                 MOV             X2, X0
.text:0000000000A4A0C                 LDR             Q3, [X5,#0x10]
.text:0000000000A4A10                 LSR             X10, X4, #0x20 ; ' '
.text:0000000000A4A14                 LDR             Q0, [X5]
.text:0000000000A4A18                 MOV             X0, SP
.text:0000000000A4A1C                 LDR             Q1, [X9,#xmmword_21B70@PAGEOFF]
.text:0000000000A4A20                 LDR             D2, [X3]
.text:0000000000A4A24                 MOV             X3, X8
.text:0000000000A4A28                 STP             W4, W10, [SP,#0x50+var_20]
.text:0000000000A4A2C                 STR             Q1, [SP,#0x50+var_50]
.text:0000000000A4A30                 STP             Q0, Q3, [SP,#0x50+var_40]
.text:0000000000A4A34                 STR             D2, [SP,#0x50+var_18]
.text:0000000000A4A38                 BL              sub_A4AC0
.text:0000000000A4A3C                 MOV             X0, SP  ; int
.text:0000000000A4A40                 MOV             W1, #0x40 ; '@' ; n
.text:0000000000A4A44                 BL              sub_AF308
.text:0000000000A4A48                 LDP             X29, X30, [SP,#0x50+var_10]
.text:0000000000A4A4C                 ADD             SP, SP, #0x50 ; 'P'
.text:0000000000A4A50
```

加密状态字还原：x0是chacha20加密状态字，指向栈sp，前16字节在栈上0x10开始，总共十六字节，然后偏移0x20是一个32字节的秘钥，因为Q0和Q3都是十六字节，而且Q0、Q3来源于[x5]，那么可知x5是一个密钥指针；偏移0x40是一个字长表示计时器，这个计数器来自于w4和w10，而w4是1，w10是空的说明计数器从1开始，符合要求；然后0x48开始是D2寄存器，而D2寄存器又来源于[x3]说明x3是一个8字节noise，再加上一开始x29被抬高0x10刚好满足0x40也就是64字节的初始状态字大小

```
00000A49F0                 MOV             X8, X2
00000A49F4                 CBZ             X2, loc_A4A50
00000A49F8                 SUB             SP, SP, #0x50 ; 'P'
00000A49FC                 STP             X29, X30, [SP,#0x50+var_10]
00000A4A00                 ADD             X29, SP, #0x50+var_10
00000A4A04                 ADRP            X9, #xmmword_21B70@PAGE
00000A4A08                 MOV             X2, X0
00000A4A0C                 LDR             Q3, [X5,#0x10]
00000A4A10                 LSR             X10, X4, #0x20 ; ' '
00000A4A14                 LDR             Q0, [X5]
00000A4A18                 MOV             X0, SP
00000A4A1C                 LDR             Q1, [X9,#xmmword_21B70@PAGEOFF]
00000A4A20                 LDR             D2, [X3]
00000A4A24                 MOV             X3, X8
00000A4A28                 STP             W4, W10, [SP,#0x50+var_20]
00000A4A2C                 STR             Q1, [SP,#0x50+var_50]
00000A4A30                 STP             Q0, Q3, [SP,#0x50+var_40]
00000A4A34                 STR             D2, [SP,#0x50+var_18]
00000A4A38                 BL              sub_A4AC0
00000A4A3C                 MOV             X0, SP  ; int
00000A4A40                 MOV             W1, #0x40 ; '@' ; n
00000A4A44                 BL              sub_AF308
00000A4A48                 LDP             X29, X30, [SP,#0x50+var_10]
00000A4A4C                 ADD             SP, SP, #0x50 ; 'P'
```

计数器　固定字　密钥　noise

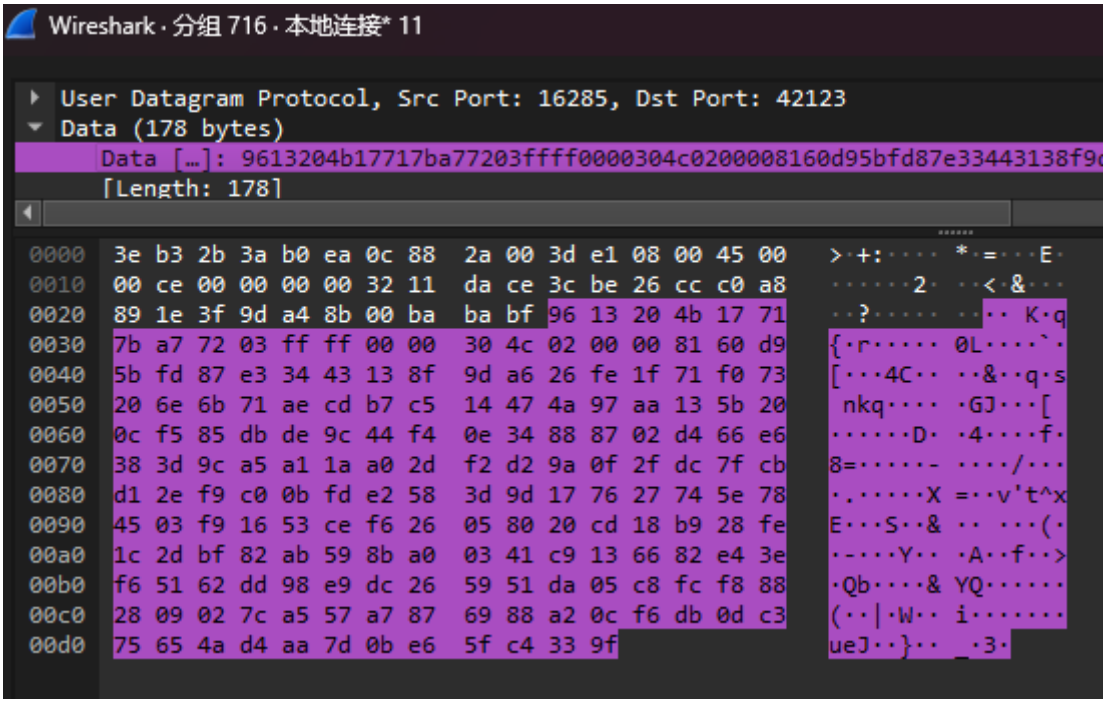
最终还原出了PackLiveCloud参数的意义。第20字节开始是chacha20加密数据。而chacha20加密的安全性和noise、key都有关系，这个chachakey是最上面架构图中中间层通过setConfig给ChannelInstance，ChannelInstance在创建Conn时InitRelayRoom给到RelayRoom，RelayRoom是Conn中一个关键对象。而中间层又是通过解析java层的protobuf数据获取到chachakey，chachakey交换采用的是mmtls，mmtls安全性约等于tls1.3，noise是变化的，因此19字节开始是安全的。

```
noise[1] = *(_QWORD *)(_ReadStatusReg(TPIDR_EL0) + 40);
if ( a2 && a3 >= 1 && outarray )
{
  v7 = *((unsigned __int8 *)configArray + 14);
  v11 = v7 & 0xF0;
  if ( v11 == 96 )
  {
    v12 = v7 & 0xF | 0x20;
    v7 = v12;
    *((_BYTE *)configArray + 14) = v12;
  }
  v13 = *((_WORD *)configArray + 5);
  noise[0] = *(_QWORD *)(key + 32) + (v7 << 32) + *(unsigned __int16 *)((char *)configArray + 15);
  v20 = v13;
  if ( v11 == 96 )
    *((_BYTE *)configArray + 14) = v7 & 0xF | 0x60;
  v14 = (unsigned int)(a3 + 2);
  v15 = *configArray;
  v16 = (unsigned int)a3;
  *(_DWORD *)(outarray + 15) = *(_DWORD *)((char *)configArray + 15);
  *(_QWORD *)outarray = v15;
  len = (_WORD *)operator new[](v14);
  *len = v20;
  memcpy(len + 1, a2, v16);
  dataArray = 0;
  v18 = sub_A3BA8(outarray + 19, &dataArray, (__int64)len, v14, 0, 0, 0, (__int64)noise, key);
  *a5 = dataArray + 19;
  operator delete[](len);
```

然后看协议头部19个字节，全部来源于confArray，confArray头部两字节是5014，小端序就是96 13，接着八个字节是room_id，然后是sn报文序列号，然后是memberid

```
v25 = *(_DWORD *)(RelayRoom + 1064);
room_id = *(_QWORD *)(RelayRoom + 16);
v27 = *(_WORD *)(RelayRoom + 40);
*(_WORD *)v47 = 5014;
*(_WORD *)&v47[12] = v25;
*(_QWORD *)&v47[2] = room_id;
*(_WORD *)&v47[10] = v27;
```

比如下面这个报文96 13就是固定头部特征，20 4b 17 71 7b a7 72 03就是小端序roomid，ff ff是memberid，00 00 是报文序列号，表示当前是该RelayRoom中第几个报文。



3.2.2 中转数据协议

几乎和上面96 13原理一样，区别就是协议头开始两字节是98 15

```
v21 = *(_DWORD *)(RelayRoom + 1064);
v22 = *(_QWORD *)(RelayRoom + 16);
v23 = *(_WORD *)(RelayRoom + 40);
*(_WORD *)v47 = 5528;
*(_WORD *)&v47[12] = v21;
*(_QWORD *)&v47[2] = v22;
*(_WORD *)&v47[10] = v23;
v24 = 16;
```

也是chacha20加密

```
v34 = *(_DWORD *)(RelayRoom + 780);
v49 = a8;
v47[14] = v24 | a6 & 0xF;
v50 = v34;
v35 = RelayRoom + 2LL * (unsigned int)TypeMaptoArrayIndex(cloud_data_type);
v36 = *(_WORD *)(v35 + 1084);
v48 = a10;
*(_WORD *)(v35 + 1084) = v36 + 1;
ret = PackLiveCloud_0(RelayRoom + 1904, data, v11, (__int64)v19, buf_size, (__int128 *)v47);
```

不过第22字节才是加密数据

```
v22[1] = *(_QWORD *)(_ReadStatusReg(TPIDR_EL0) + 40);
if ( data && data_len >= 1 && outarray )
{
  v7 = *((unsigned __int8 *)configArray + 14);
  v11 = v7 & 0xF0;
  if ( v11 == 96 )
  {
    v12 = v7 & 0xF | 0x20;
    v7 = v12;
    *((_BYTE *)configArray + 14) = v12;
  }
  v13 = *((_WORD *)configArray + 5);
  v22[0] = *(_QWORD *)(a1 + 32) + (v7 << 32) + *(unsigned int *)((char *)configArray + 15);
  v20 = v13;
  if ( v11 == 96 )
    *((_BYTE *)configArray + 14) = v7 & 0xF | 0x60;
  data_array_len = (unsigned int)(data_len + 2);
  v15 = *configArray;
  v16 = (unsigned int)data_len;
  *(_QWORD *)(outarray + 13) = *(_QWORD *)((char *)configArray + 13);
  *(_QWORD *)outarray = v15;
  data_array = (_WORD *)operator new[](data_array_len);
  *data_array = v20;
  memcpy(data_array + 1, data, v16);
  v21 = 0;
  v18 = sub_A3BA8(outarray + 21, &v21, (__int64)data_array, data_array_len, 0, 0, 0, (__int64)v22, a1);
  *array_size = v21 + 21;
  operator delete[](data_array);
}
else
```

示例

```
0c 88 2a 00 3d e1 3e b3  2b 3a b0 ea 08 00 45 00   ··*·=·>· +:····E·
00 7f 94 e7 40 00 40 11  f8 35 c0 a8 89 1e 3c be   ····@·@· ·5····<·
26 cc a4 8b 3f 9d 00 6b  29 06 98 15 20 4b 17 71   &···?··k )··· K·q
7b a7 72 03 00 00 98 05  10 00 00 00 00 0b 00 4c   {·r····· ·······L
ac 4f c0 0f e2 d1 4b d5  38 61 1c f3 72 d6 a3 04   ·O····K· 8a··r···
53 e9 32 7d 17 8c dd fc  01 e0 b9 ab d8 f8 27 b6   S·2}···· ······'·
4e 73 50 ea 8b f8 a1 ff  a1 65 b5 d3 a4 6e b3 08   NsP····· ·e···n··
2e 5e 65 1b f7 e7 d7 bc  22 cb 1e fb 9c 1e 8a 95   .^e····· "·······
88 aa 7c f9 a4 f2 b1 b2  18 bf 7e b6 65            ··|····· ··~·e
```

并且加密后的代码可以发现96 13和98 15采用的是同一个偏移的sn序列号

```
{
  PackRelayPktChaChaLive(
    RelayRoom,
    *(__int64 **)(RelayMgr + 16),
    cloud_data_type,
    data,
    data_len,
    a6,
    a7 & 1,
    a11,
    &buf0,
    a12,
    &buf_size);                          // chacah20加密
}
else if ( (unsigned int)(v19 - 4) >= 3 )
{
  PackRelayLegacyDataPkt(
    (__int64 *)&buf0,
    RelayRoom,
    *(_QWORD *)(RelayMgr + 16),
    cloud_data_type,
    (__int64)data,
    data_len,
    (unsigned __int64 *)&buf_size);
}
else
{
  PackRelayPktChaCha(
    RelayRoom,
    *(__int64 **)(RelayMgr + 16),
    cloud_data_type,
    (__int64)data,
    data_len,
    &buf_size,
    &buf0);
}
v22 = buf0;
if ( !buf0 )
  return 0;
v23 = *(_DWORD *)(RelayRoom + 1128) + 1;
v24 = *(_QWORD *)(RelayRoom + 1136) + buf_size;
++*(_DWORD *)(RelayRoom + 1064);
*(_DWORD *)(RelayRoom + 1128) = v23;
```

3.3.点对点数据协议

在SendConn时还有一种Direct的情况，这是服务器为了节省带宽，在客户端尝试nat穿透，如果成功了就直接点对点传输一部分数据

```
conn_type = *(_DWORD *)ConnInfo;              // Relay-6,Direct-5
if ( *(_DWORD *)ConnInfo != 6 )
{
  if ( conn_type == 5 )
  {
    v20 = ConnInfo[1];
    if ( v20 )
      return DirectSendData(
               *(_QWORD *)(v20 + 6768),
               v20,
               data,
               data_len,
               connector,
               (__int64)OnSent,
               0,
               cloud_data_type);
  }
```

看到DirectSendData，原理仍然类似，n是一个8字节数组，sub_7C880进行chacha20加密，DirectClient + 6376是chachakey

```
v18 = *(_DWORD *)(DirectClient + 6712);
v19 = *(_DWORD *)(DirectClient + 6312);
v20 = *(_WORD *)(DirectClient + 6680);
LOBYTE(n) = -92;
outarray_len = 0;
WORD2(n) = v18;
BYTE1(n) = v19;
WORD1(n) = v20;
HIWORD(n) = v15;
v21 = sub_7C880(DirectClient + 6376, (__int64)data, data_len, outarray, &outarray_len, (__int64 *)&n);
```

在sub_7C880可以发现这里只有前八字节是协议头部

```
noise[1] = *(_QWORD *)(_ReadStatusReg(TPIDR_EL0) + 40);
v7 = *((unsigned __int16 *)a6 + 1);
v8 = bswap32(*((unsigned __int16 *)a6 + 3));
v9 = __rev16(v7);
v10 = *(_QWORD *)(key + 32) + v7;
*((_WORD *)a6 + 2) = bswap32(*((unsigned __int16 *)a6 + 2)) >> 16;
*((_WORD *)a6 + 1) = v9;
*((_WORD *)a6 + 3) = HIWORD(v8);
v11 = *a6;
v13 = 0;
noise[0] = v10;
*outarray = v11;
result = sub_A3BA8((__int64)(outarray + 1), &v13, data_array, data_array_len, 0, 0, 0, (__int64)noise, key);
*a5 = v13 + 8;
return result;
```

再分析一下这个八字节数据，第一个字节是-92，补码就是a4，v19暂且不知，然后是两个字节的sn，两个字节的memberid，两个字节headsign，这个headsign要么是0x77，要么a8的值

```
memberid = *(_DWORD *)(DirectClient + 6712);
v19 = *(_DWORD *)(DirectClient + 6312);
sn = *(_WORD *)(DirectClient + 6680);
LOBYTE(n) = -92;
outarray_len = 0;
WORD2(n) = memberid;
BYTE1(n) = v19;
WORD1(n) = sn;
HIWORD(n) = headsign;
v21 = sub_7C880(DirectClient + 6376, (__int64)data, data_len, outarray, &outarray_len, (__int64 *)&n);
if ( v21 )

if ( (_DWORD)a8 == 121 )
  headsign = 119;
else
  headsign = a8;
```

抓包示例a4表示点对点音视频数据报文，08 b3是报文序列号sn，00 00是memberid，0x76是一个headsign

```
Data: a40108b300000076764fa1b3514c1713114fcf5f08338ee2a7971518ee65a1a1d08b
[Length: 84]

0c 88 2a 00 3d e1 3e b3  2b 3a b0 ea 08 00 45 00   ··*·=·>· +:····E·
00 70 90 0f 40 00 40 11  9e 9a c0 a8 89 1e c0 a8   ·p··@·@· ········
01 64 82 23 7b 2d 00 5c  2c eb a4 01 08 b3 00 00   ·d·#{-·\ ,·······
00 76 76 4f a1 b3 51 4c  17 13 11 4f cf 5f 08 33   ·vvO··QL ···O·_·3
8e e2 a7 97 15 18 ee 65  a1 a1 d0 8b ab cb 25 69   ·······e ······%i
c4 a0 0e 90 dc 36 0f d6  41 40 a9 ff 1d 5d 5e e5   ·····6·· A@···]^·
5c dc 0a da d7 c1 14 bd  4d 20 e4 f9 3a 87 30 eb   \······ M ··:·0·
b8 3b dd 65 92 8c e6 79  78 cf ae 6c e6 f6         ·;·e···y x··l··
```

4.总结

96 13/98 15/A4是经过Codec的三种协议报文，报文都采用chacha20加密，安全交换密钥。并不存在可解密的wxid等重要数据，但是8字节的room_id可以关联音视频通话发送者和接受者。报文头部有两个字节的小端序列号，9815和9613共用sn，通过sn可以简单判断数据包起始和结束，但是只有两字节存在覆盖的情况。