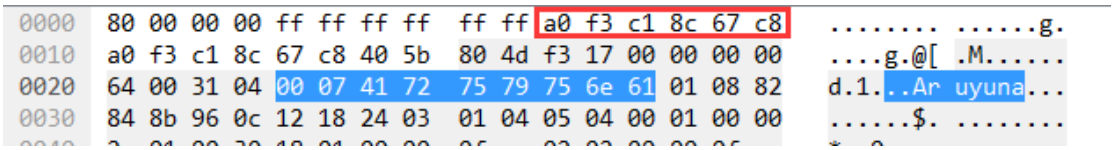
WPA/WPA2 加密全流程（原理）

李大燊

中山大学

2018.1.4

WiFi 无线接入点（以下简称 AP）在启动后会向周围发送 Beacon 无线信号，表示本大爷在这一带，你们知道密码的可以连过来。此时发送的 Beacon 帧中包含有自己的 WiFi 名字（以下简称 SSID），Beacon 帧的前几位如下。



问题 1：怎么知道这个是 Beacon 帧

答案：前面 4 字节是 8 后面 7 个 0 的就是

问题 2：SSID 在哪里

答案：上图中深蓝底部分，前 2 字节表示 SSID 的长度，这里是 7，表示接下来的 7 字节就是 SSID，从 41……到 61 这部分就是 SSID 的二进制编码（全英文的话就是 ascii 码），可以看右边的深蓝底部分得到明文形式。SSID 长度值的位置固定在 0x0024-25 这个位置，在这里提取长度（假设为 x）后，从 0x0026 开始往后推到 x 的字节的这部分就是 SSID 的值

问题 3：那个红框框着的是啥

答案：该 AP 的 Mac 地址，之后解包时要根据这个值确定这个握手包对应的 SSID，因为握手包里没有 SSID

之后，AP 会根据自身设定的 SSID 和密码，计算出认证时使用的 PMK（PMK 的计算会在下面介绍）。

手机（或者平板电脑等其他可以无线上网的设备，以下简称 STA）开启 WiFi 后，会搜索周围的 WiFi 信号，接收到 AP 发送的 Beacon 帧，就会将该 AP 的信息打在手机界面上，包括 SSID，加密方式（开放/WEP/WPA/WPA2/802.11）。

一般的 AP（包括无线路由器，手机热点）加密方式都是 WPA/WPA2 方式（当然也有可能是开放，这种如果不是需要浏览器登录方式的话简直是小天使ヾ（´▽｀）ノ）

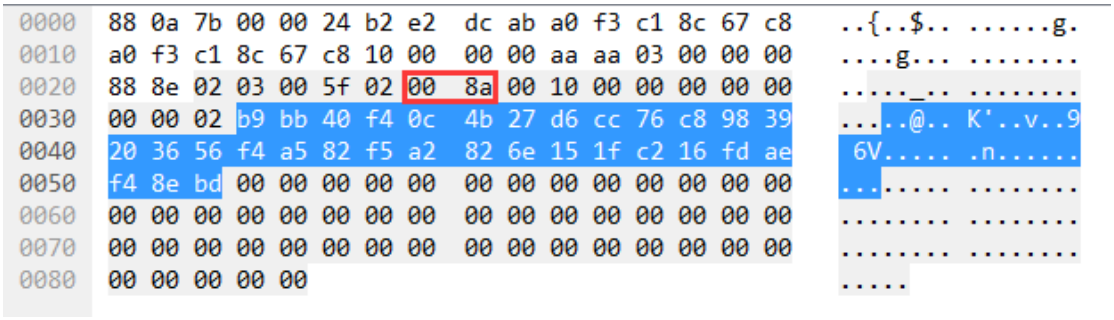
WEP 作为 10 年前就公认的 SB 加密方式就不去讨论了。

如果是 WPA/WPA2 方式，那么需要验证 WiFi 密码才能登陆。

验证过程包括 4 次握手：

Round 1. AP→STA

AP（路由器）产生了一个 256 位（32 字节）的随机数（下面简称 ANonce）用于这次验证，同时 EAPOL 数据包里还包含了本 WiFi 的加密方式（体现在 Key Information 的最后 3 位，如果 WiFi 是 WPA 加密，这 3 位为 001，使用 HMAC_MD5 加密算法计算 MIC；如果是 WPA2 加密，这 3 位为 010，使用 HMAC_SHA1 加密算法计算 MIC。当然 WPA 和 WPA2 就只有计算 MIC 算法不同这一个差别，其他部分是一致的。哦还有，WPA 数据传输时使用 TKIP 加密，这是有可能确解的，而 WPA2 数据传输使用 AES-CCMP 加密，这个无法反推得到）。其他资料完全后，AP 把这个包发给 STA。



问题 1：我怎么知道这是 EAPOL 数据包

答案：开头第 1 字节是 88 的，中间有一段 Logical-Link Control 字段（001a~0021 那里为”aa aa 03 00 00 00 88 8e”）。

问题 2：我怎么知道是 WPA 还是 WPA2

答案：看红框（固定在 0x0027-28 位置），为 Key Information 字段，最后字符是 a，写成二进制就是 1010，后 3 位是 010，写成十六进制是 2，从前面的说法，知道这是 WPA2 加密方式

问题 3：ANonce 在哪？

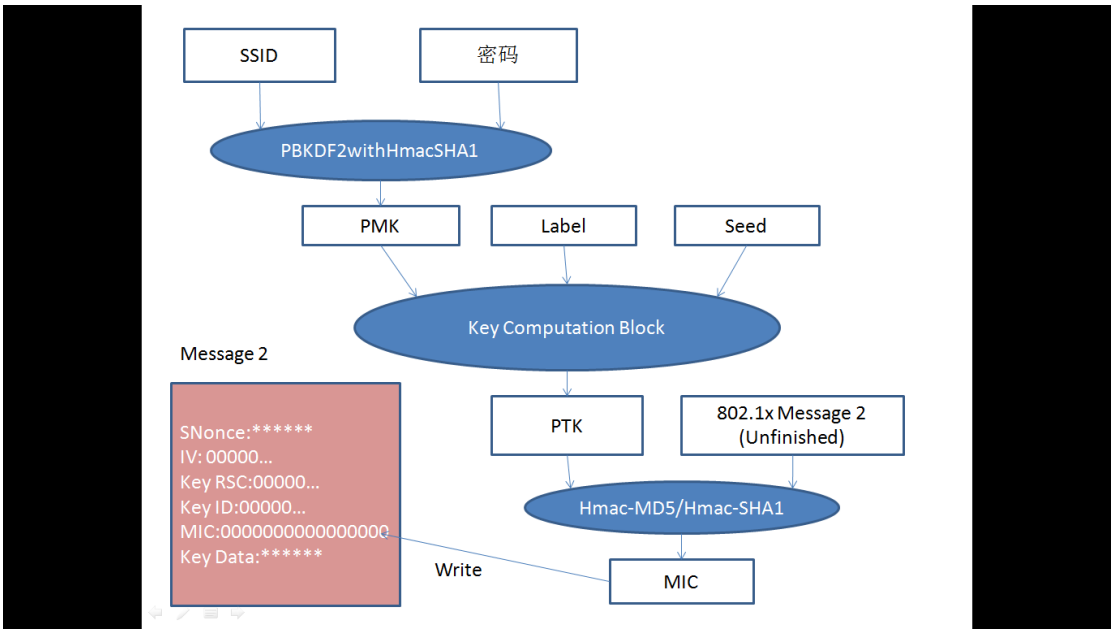
答案：上图中深蓝底部分（固定 0x0033-52 位置）

Round 2. AP→STA

关键部分来了， 破解的关键全在这个包（除了没有 ANonce）。

STA（你的手机）接到 AP 发来的第一个包后开始了一系列的计算：

一图以蔽之：



下面将详细介绍这个图。

你输入了这个 AP 的密码（可能是对的也可能是错的，但 STA 并不知道这是不是对的，它只管拿到你输入的密码，然后开始计算）

STA 使用这个 AP 的 SSID 和你刚刚输入的密码（简称 psw）使用 PBKDF2_SHA1 算法加密得到临时主密钥（PMK）：

PMK=PBKDF2（HMAC_SHA1(), psw, SSID, 4096, 256）;

参数列表：HMAC_SHA1()是加密用的迭代算法，WiFi 加密全部用 HMAC_SHA1。psw 为密码，SSID 字面意思，4096 表示迭代 4096 次，256 是最后输出的位数（256 位，32 字节）

对 PBKDF2 算法理解不够深，细节方面现在说不了。以后补。

由于 4096 这个参数，计算 PMK 时将会执行 4096 次 HMAC_SHA1 加密函数，还有一些其他的乱七八糟的字符串拼接等等的操作，因此计算一个 PMK 是及其耗时的，也是破解的难点所在。这也是为什么手机连一个新的 WiFi 时连上去要好几秒，而连一个之前连过的旧 WiFi 都是秒连，因为你连一个旧 WiFi 不用再输密码，也就不用修改 PMK（PMK 的结果只与 WiFi 名字和密码有关）。当然如果旧 WiFi 路由器偷偷改了密码，你连上去会给你说密码错误，因为 PMK 已经失效了。

STA 在计算完 PMK 后，它会从 Beacon 帧里抓出 AP 的 Mac 地址（这个在第 1 次握手里也能抓到；下称 AMac），从第 1 次握手包里抓出 ANonce，然后 STA 自己也生成一个随机数（也是 256 位，下称 SNonce），再加上它自己的 Mac 地址（下称 SMac），这 5 个东西作为参数生成成对扩展密钥（PTK）……

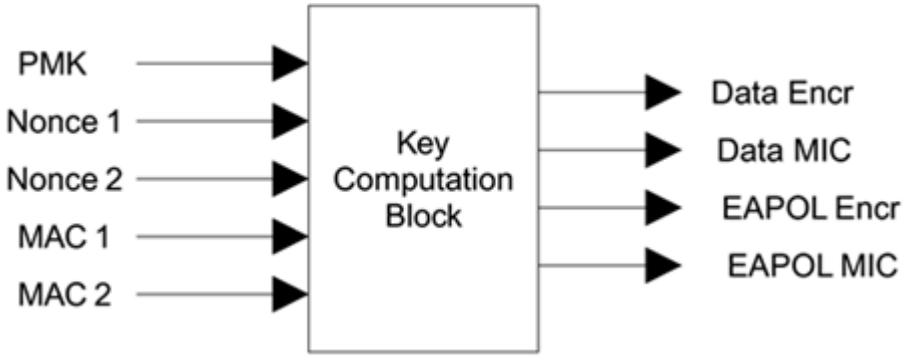
————————————————————起始分割线————————————————————

在这里先打住，讲点非常好玩的东西！

之前翻阅了很多讲述 WPA 加密的书，对于 PTK 的计算，他们的说法是：利用 PMK，AMac，SMac，ANonce，SNonce 通过一系列加密生成 512 位（64 字节）的 PTK，PTK 分为 4 部分：KCK（0-127 位），KEK

（128-255 位），TK1（256-383 位），TK2（384-511 位）。【注：WPA2 貌似没有 TK2，不过这都不影响了，因为破解密码只需要 KCK，剩下的是连上 WiFi 之后的事了。】

然后贴上这个图

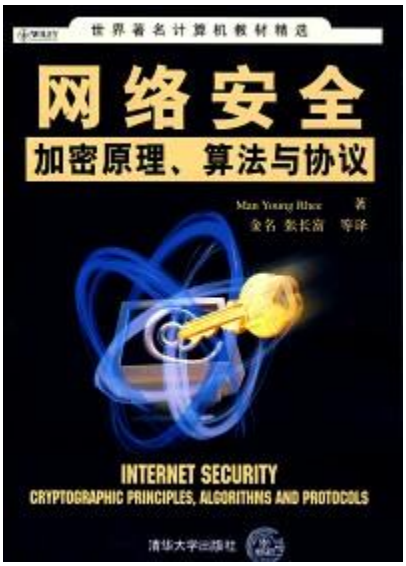


这个图我相信已经烂大街了，部分介绍 WPA 四次握手的技术博客也用了。但问题是这个 Key Computation Block 里面到底是什么？

书上没找到，去 Internet 上翻了一圈，所有讲述了 WPA 加密的文章这里的说法全是“通过一系列计算生成 PTK”，然后摆出上面那张图。兄弟们啊，这个 Block 里面有啥？？

当然好一点的是：PTK=SHA1_PRF(…)。好歹还把算法讲出来了，但是关于这个算法，几乎找不到任何资料支撑（天上掉下来的？），而提到这个计算公式的文章，就把这个公式摆出来，然后提一下“前 16 字节用于生成 MIC”然后就到了下一步……

后来在图书馆里找到一本关于网络安全算法介绍的书，如下图（《网络安全：加密原理、算法与协议》，ISBN 978-7-302-15259-0）



里面提到了 SHA1，HMAC 和 PRF 算法（希望的曙光？），但没有提到前面的 PBKDF2。

算了，见一步走一步就行，于是开始琢磨这个传说中的 PRF，发现还要人为去定义这个 PRF 的规则（emmmmm）。

去撸代码实现了书上的例子，发现把这 5 个玩意扔进去最后的结果一点都不一样（怎么会一样，书上的例子规则和 WPA 规则完全不一样）

因此在一脸懵逼之际，突然想起来 Aircrack-ng 这个软件是开源的！于是去瞄了一眼源代码，找到了计算 PTK 的方法，这才终于解决了这个 PRF 算法是什么的问题（哈哈哈哈哈）……

故事完结，回到正文。

————————————————结束分割线————————————————

PTK=PRF(HMAC_SHA1(), secret, label, seed);

参数说明：

HMAC_SHA1 是加密算法（又是你）。

secret 是 PMK。

label 是标签（一段扩展密钥用的字符串，在 WPA 加密的 PTK 扩展中是固定字符串[Pairwise key expansion]，区分大小写，有空格，22 个字节）。

seed 是一段东西的拼接，这些东西共 76 字节，前 12 字节是 AMac 与 SMac 的值（较小在前，较大在后），后 64 字节是 ANonce 与 SNonce（较小在前，较大在后），然后这 4 个东西直接拼接形成 76 字节作为 seed。

那么生成 PTK 的 PRF 算法的规则是啥？下面告诉你：

第①步：先把 label 和 seed 通过特殊规则拼起来得到 Labelseed（共 100 字节）……

————————————————————起始分割线————————————————————

好玩的东西又来了。

之前提到的那本书提到了 PRF 算法，他对 PRF 是这样描述的：

A(0)=HMAC_SHA1(psw, Label || seed);（这里 psw 是一般加密密钥）

B(0)=HMAC_SHA1(psw, A(0));

A(n)=HMAC_SHA1(psw, A(n-1) || Label || seed);

B(n)=HMAC_SHA1(psw, A(n)); (n=1,2,……)

当时发现了 WPA 的 PRF 算法后我直接把 Label 和 seed 拼起来作为 Text 输入，但结果永远与 Aircrack-ng 的不一样。

WTF？？

Aircrack-ng 源码上的确也是这个参数格式啊

```
/* compute the pairwise transient key and the frame MIC */

for (i = 0; i < 4; i++)
{
        pke[99] = i;
        HMAC(EVP_sha1(), pmk[j], 32, pke, 100, ptk[j] + i * 20, NULL);
}
```

我对了一遍 label 也没错，seed 也没错……

```
/* pre-compute the key expansion buffer */
memcpy( pke, "Pairwise key expansion", 23 );
if( memcmp( ap->wpa.stmac, ap->bssid, 6 ) < 0 ) {
        memcpy( pke + 23, ap->wpa.stmac, 6 );
        memcpy( pke + 29, ap->bssid, 6 );
} else {
        memcpy( pke + 23, ap->bssid, 6 );
        memcpy( pke + 29, ap->wpa.stmac, 6 );
}
if( memcmp( ap->wpa.snonce, ap->wpa.anonce, 32 ) < 0 ) {
        memcpy( pke + 35, ap->wpa.snonce, 32 );
        memcpy( pke + 67, ap->wpa.anonce, 32 );
} else {
        memcpy( pke + 35, ap->wpa.anonce, 32 );
        memcpy( pke + 67, ap->wpa.snonce, 32 );
}
```

咦不对，从 ptk 内存的第 23 个位置开始填，那 label 是多长来着？

8+3+9+2=22

嗯？再一个一个数……还是 22

原来第 23 个位置空出来的？

回去把计算模块更新到下标 22 的位置空出来，从 23 到 98 为那 4 段文字的拼接，输进去，发现还是错的

……

又回去 Aircrack-ng 源码那里，这次发现了一行短到被忽视的代码

```
/* compute the pairwise transient key and the

for (i = 0; i < 4; i++)
{
        pke[99] = i;
        HMAC(EVP_sha1(), pmk[j], 32, pke, 100,
}
```

原来 PRF 是这么个意思！！！

故事完结，回到正文

————————————————————结束分割线————————————————————

Labelseed 的拼接规则是：0-21 字节为 Label 字段（22 字节），第 22 字节为 0（全 0 二进制位，不是 ascii 为 48 的那个 0），第 23-98 字节为 seed 字段（76 字节），第 99 字节为扩展次数（初始为 0）。

第②步：把上面生成的 Labelseed 作为 Text 与 PMK 作为 key 通过 HMAC_SHA1 算法得到第 1 个结果（共 20 字节）。扩展次数加 1。

第③步：重复①②，后续每次的 HMAC_SHA1 得到的结果直接拼到前一次结果的后面，一共 4 次最终得到 80 字节的扩展码。最后取前 64（WPA）/48（WPA2）字节作为 PTK。

PTK 的前 16 字节作为 KCK 与一个 802.1x 数据帧作为参数根据 WPA 或者 WPA2 的加密方式选择对应加密算法计算 MIC……

————————————————————起始分割线————————————————————

好了又要讲故事了，正文真短。

从 Aircrack-ng 源码里，刚刚 PTK 的代码后面就是 MIC 计算代码

```
if (ap->wpa.keyver == 1)
        HMAC(EVP_md5(), ptk[j], 16, ap->wpa.eapol, ap->wpa.eapol_size, mic[j], NULL);
else
        HMAC(EVP_sha1(), ptk[j], 16, ap->wpa.eapol, ap->wpa.eapol_size, mic[j], NULL);
```

参数是 PTK 前 16 字节，以及 ap 结构里的 wpa 成员的 eapol 值，这个值我翻了一下结构描述的头文件，一个 buffer 为 256 的 char 型，记录 EAPOL 帧内容。

但从代码里没看出来这指的是哪个帧，从数据完整性判断很大可能性是第 2 次握手的数据。

但第 2 次握手中 MIC 字段已经是结果，是要抽出来对比的，这一整个数据段带 MIC 写进去肯定最后计算出的 MIC 不一样。

我试过把最后 22 字节（标示为 Key Data）的部分作为参数，但结果不对。

然后想翻翻网上有没有什么介绍，一翻把我笑尿了。

在那些介绍 WPA 加密原理的文章里，对于 MIC 的计算描述大概全是这个样子

第三次握手：AP 发送 ANonce 给 STATION，STATION 端接受到 ANonce 和以前的 PMK，SNonce，AP_MAC(AA)，STATION_MAC（SA），用同样的算法生成 PTK，提取前 16 字节组成一个 MIC_KEY,用 MIC_KEY 和 802.1x data 数据帧算出 MIC（MIC=HMAC_MD5(MIC_KEY, 16, 802.1x data)）

后面 MD5 是 WPA 模式使用的算法，WPA2 用的 SHA1 这个从前面的代码里看出来了，所以不管那个问题。问题是，这个“802.1x data 数据帧”又是啥？

所有文章的该段文字前后都没有提及这个 MIC 是什么（葛优躺

兄弟们啊，你们写篇技术类博客不要整篇照抄的好伐？搞事呢？

代码都看不出来是什么，正当束手无策之际，突然想起来了一个绝招——

Debug！

于是我加了这么两行（前面是计算 PTK 的，后面是计算 MIC 的，无视%d 后面那个光标）

```
for (i = 0; i < 4; i++)
{
        pke[99] = i;
        HMAC(EVP_sha1(), pmk[j], 32, pke, 100, ptk[j] + i * 20, NULL);
}
uint32_t abcde=0;
for(abcde=0; abcde < ap->wpa.eapol_size; abcde++)printf("%d ",ap->wpa.eapol[abcde]);

if (ap->wpa.keyver == 1)
        HMAC(EVP_md5(), ptk[j], 16, ap->wpa.eapol, ap->wpa.eapol_size, mic[j], NULL);
else
        HMAC(EVP_sha1(), ptk[j], 16, ap->wpa.eapol, ap->wpa.eapol_size, mic[j], NULL);
```

重新 make 了一次，运行跑包程序，界面被我搞成了这个样子

```
                                Aircrack-ng 1.2 rc41 3 0 117 2 1 10 0 0 0 0 0 0
 0 0 0 2 253 239 162 39 59 10 215 106 85 20 111 220 194 46 101 118 148 4 239 143
 53 25[00:00:00] 1/0 keys tested (27.92 k/s) 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 22 48 20 1 0 0 15 1
72 2 1Time left: 0 seconds                                      inf%

                          KEY FOUND! [ 00100001 ]


      Master Key     : 4E 19 22 AC 94 8C C3 5D 0C B3 B2 A5 12 AD D0 CB
                       31 85 DF CA CB 3A 1B 46 79 F5 EF 9D 0D 93 DE A1

      Transient Key  : F7 9B C0 C4 38 59 77 A6 11 21 A5 CB BA 0A 3F 52
                       63 46 7E CB 76 A1 28 C3 C8 8F B0 1C 3F EF FE FB
                       9C 21 F3 FE CD 53 EE AF 92 5E 56 3C 16 C2 10 39
                       AA 7A 02 8B 1A 57 C0 2C B8 7B 0D 13 70 3F 9A 1D

      EAPOL HMAC     : 34 0F 7F CD 5D F2 2E A9 94 F1 B4 D9 3F 26 F3 C5
```

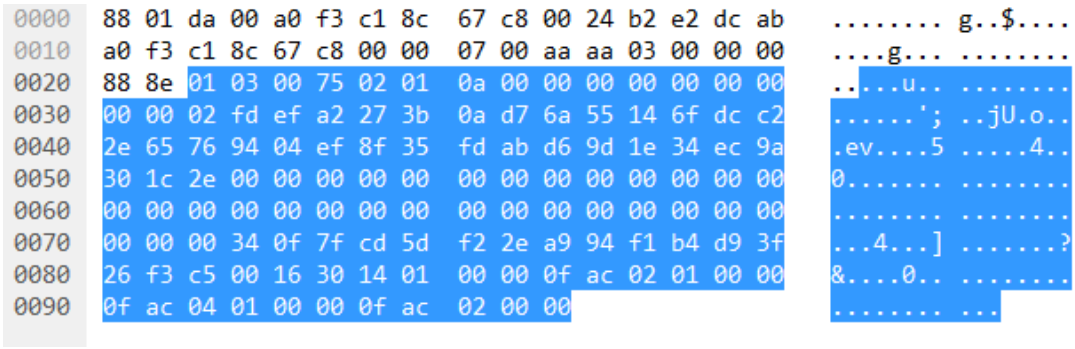
（原来的第一行到”rc4”就结束了）

好了，这个 EAPOL data 帧到底是谁很明了了，对比前面几位是 1（0x01），3（0x03），0（0x00），117（0x75），2（0x02），1（0x01）……

那么在十六进制模式下看到的应该是

01 03 00 75 02 01……

回去 Wireshark 一比对，真的似李，第 2 次！



中间和最后有部分文字被 Aircrack 的说明文覆盖了，很明显这个 eapol 帧指的是这一整段深蓝色部分。但问题来了，MIC 字段那里原来应该是什么？

MIC 位置在倒数第 2,3 行那个位置，后面接的是 00 16 代表后面 0x16 个字节是 Key Data，所以倒回到 00 16 前面的部分（在命令行里的输出应该是 0 22 48 20……）

很容易就找到了，发现这里原来全是零（震惊！困扰了这么久的问题居然是……）

故事结束，回到正文。

————————————————————结束分割线————————————————————

计算 MIC 的这两个参数分别是 PTK 的前 16 位作为 Key，自身的 802.1x Auth 数据帧部分（固定 0x0022-最后，0x0024-25 存了后续数据总长，这个数加上 4 就是数据帧总长）把其他该填的都填好（MIC 暂时不填，全 0），保存下来作为 Text，使用 HMAC_MD5（对 WPA）或者 HMAC_SHA1（对 WPA2）算法计算结果，前 16 位填充到这次握手包的 MIC 字段中。最后，STA 把这个 EAPOL 帧发给 AP。

Round 3：AP→STA

终于来到了 Round 3 了，前面 Round 2 真是充满了怨念 hhhhhhhh

AP 之前第 1 次握手时已经指定了 STA 的 MAC，也保存了自己的 Nonce 和 MAC，以及早已计算好的 PMK（见第 1 页）。收到了第 2 次握手包后，从里面抓出来 SNonce 和将要对比的 MIC。这些参数用上面同样的方法生成 PTK，MIC，并把这个 MIC 结果和第 2 次握手包里抓出来的 MIC 进行匹配，完全一致则原来 STA 输入了正确密码，验证通过，发一个数据包给 AP 表示密钥已装好，可以开始通信了【通信数据加密使用 TK（256/128 位对于 WPA/WPA2）作为加密 Key】。

如果 MIC 不一致，（滑稽）则 AP 丢掉这个包，认证失败。你想连接就重来，从第 1 次握手重新开始。之后 ANonce，SNonce 都会不一样。

Round 4：STA→AP

STA 收到 AP 的通过帧后同样安装密钥进行通信，并给 AP 发送一个 ACK 帧表示已经准备好。同时 STA 的屏幕界面显示已连接。

如果想破解这个密码的话，按照 Round 2 的流程把程序弄出来就行了。当然，重点在优化 PMK 的生成上面。还有需要注意的是，这必须要是一个完整的 4 次握手包才能确解，因为出现第 3，4 次才表明验证成功，如果只有 2 次的话结果是不正确的，因为密码错误了算出来的 MIC 也是错误的（虽然 Aircrack-ng 查到 2 次握手照样跑而且如果字典里有的话会跑出一个你输入的错误密码 2333333）。