# TODO

测试方案

<https://blog.csdn.net/u013136708/article/details/50610955>

<http://www.wolfcstech.com/2018/03/20/android_cert_mgr_and_verify/>

# 前言

**SSL**（Security Sockets Layer）和**TLS**（Transport Layer Security）[8-10]，都是建立在公钥密码的基础之上的。实际上，它们组合使用了对称密码、非对称密码、消息摘要和数字签名。这两种协议存在极大的相似。通常，会把SSL和TLS合起来简称为SSL。

SSL v3.0发布于1995年，TLS v1.0在1999年发布，一般被认为是SSL v3.0的后继者，因为它们确实非常相似。TLS现在最新的版本是2006年发布的1.1版本

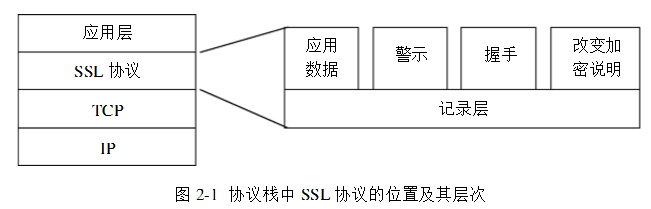
**OpenSSL**是一个功能非常强大的SSL实现。它实现了各种密码算法、密钥管理和证书管理功能，同时它支持SSL v2.0、SSL v3.0和TLS v1.0。OpenSSL被广泛地应用。它是开源的软件包，这为有兴趣学习各种密码算法实现和SSL实现的读者提供了非常宝贵的资源。OpenSSL里还提供很多有用的工具，使用它们可以很轻松地完成一些任务，例如签发证书、查看证书等等。它为密码技术和SSL的推广发挥了重要作用。

关于它的最新动态，可到www.openssl.com进行查阅

# SSL协议中握手过程

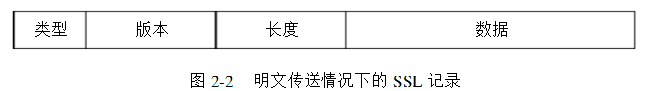
## SSL记录层协议的说明

SSL实际上是一个分层的协议。SSL记录是SSL协议发送和接收的基本单位。在记录层协议之上，有四种协议：改变加密说明（change\_cipher\_spec）、警示（alert）、握手（handshake）、应用数据（application\_data）。改变加密说明协议和握手协议将在下文详细讨论。应用数据协议是SSL为其上层协议服务所携带的应用数据，例如HTTP“请求”就是通过应用数据协议传送的。警示协议主要用于报告各种错误，另外一个重要的用途是指示连接将要关闭。



SSL有一个假定，就是假定其下层的数据包发送机制是可靠的。写入网络的数据将依顺序发送给另一端的程序，不会出现丢失数据包或者重复发送的情况。这种要求是必须的。从下文可以看到，SSL的握手消息是有时序的。在实际中，也有在UDP上运行的SSL变种，但SSL几乎只是在TCP上运行，它不能在UDP或者直接在IP上运行。其实这一点也令它的应用很广泛，因为很多应用层协议是基于TCP的，例如HTTP，FTP等等。SSL的这种特点也使其很自然地嵌入到应用层与TCP层之间。图2-1说明了这种关系。

在进一步对握手协议进行说明之前，先对SSL记录层协议进行说明。下文的分析都是以SSL记录作为基本单位的。记录层协议实际上是一个简单的封装或者说是“打包”协议。记录可能是明文传送的，也可能是被加密传送的，具体需要看通信实体所处的阶段。图2-2是明文传送情况的SSL记录的结构示意图。



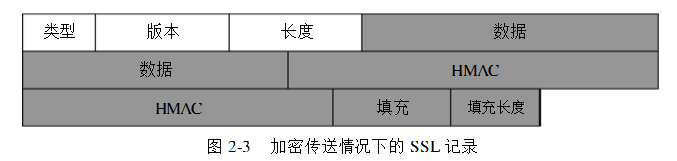
其中的类型、版本和长度合起来称为“记录头信息”，共长5个字节。各字段的大小和具体含义如下。

类型：1个字节，表示此记录携带的是何种数据。可能的取值有四种，change\_cipher\_spec=20，alert=21，handshake=22，application\_data=23。

版本：2个字节，表示此记录是什么协议版本的记录。分主版本和从版本，各占1个字节。对于SSL v3.0，主版本为3，从版本为0，那么值为0x0300；对于TLS v1.0，主版本为3，从版本为1，那么值为0x0301。

长度：2个字节，表示所携带数据的长度，并不包含记录头信息的5个字节。也就是说，整个记录的长度=长度+5。

从图2-3可以看出，经过加密后的SSL记录要比明文传送的记录多了一些内容。阴影的部分是经过加密的，而且是作为一个整体加密。然而记录头信息始终是明文传送。



HMAC：消息验证码，以确保消息的完整性。具体的长度需要根据协商的摘要算法而定。对于MD5是16字节，而SHA1则是20字节。至于如何计算HMAC，SSL v3.0与TLS v1.0所采用的具体方法并不相同。在下文将会看到这些差别。

填充，填充长度：只有在双方采用分组加密的情况下才会出现的字段。如果是采用序列加密，并不存在。“填充长度”是指“填充”字段的字节数，占1个字节。如果用Length(A)表示字段A的字节数，那么填充的目的就是使Length(数据)+ Length(HMAC)+ Length(填充)+ Length(填充长度)等于所使用分组加密算法的块大小的整数倍。至于“填充”字段的内容和长度，SSL v3.0和TLS v1.0有着不同的规定。

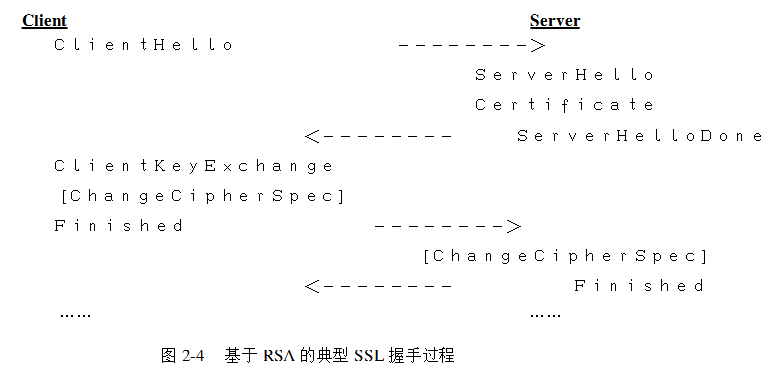
SSL v3.0的说明中规定，“填充长度”的值一定要小于所使用分组加密算法的块大小或者等于0，即填充最少的字节。而对“填充”字段的内容并没有作任何限制，是随意的。TLS v1.0的说明中则指出，“填充长度”的取值范围可以是从0到255（不含255）。而且“填充”字段中的每个字节的内容均为“填充长度”的值。

记录层协议的“数据”可以被压缩，只要双方都协商使用某一压缩算法。但SSL协议里只明确指出如何表示不使用压缩方法。至于如果要使用压缩算法，如何对其编码，则没有任何说明，这依赖于实现。

## SSL握手协议

### 握手协议简述

握手过程有多种形式。将要详细讨论的握手过程所涉及的消息如图2-4所示。这是一个非常常见而且典型的基于RSA的握手过程。通信双方通过这一系列信息的交换，从而使客户端可以对服务器的身份进行验证，确定服务器可信任，并且双方可以协商和计算接下来如何保护通信的方法和参数。这是握手过程的目的。下文将会对每条消息进行详细的说明。



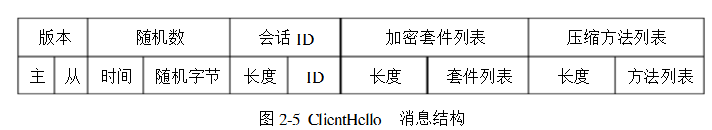
握手过程占据了整个SSL协议的绝大部分，内容相当丰富，而且需要证书及大量其他技术诸如密码学等的支持。握手过程的实现同样也占据了OpenSSL实现中的绝大部分，函数繁多。这足可见其重要。

对于SSL v3.0和TLS v1.0，握手协议所规定的消息格式基本上是相同的。不同之处来自两个方面：一个是说明上明确定义的消息结构的不同；一个则是消息结构定义相同，但由于某些历史原因所造成的实现上的不同。它们的差别也体现在一些计算的定义上。下文在适当的地方将会指出这些差别。

对于SSL记录层协议而言，整个握手消息就是记录中的“数据”部分。握手消息也有自己的格式。与SSL记录格式类似，握手消息也由头信息和内容两部分组成。头信息包含两个字段：握手消息类型，1个字节，用以指出内容是何种握手消息；长度，3个字节，用以指出内容的长度为多少字节。

### ClientHello消息

ClientHello是握手过程发送的第一条有效信息，并且总是从客户端发出。握手过程中，客户端必须先“说话”。该消息有两个目的：一个是传送一些连接的参数，为以后所用；一个是携带客户端给出的一些参数的选择列表，便于服务器从中选择。该消息的结构如图2-5所示



版本（client\_version）：与SSL记录层协议中“版本”字段的定义一样，2个字节，第一个表示主版本，第二个表示从版本。**此版本的值是客户端准备接受的“最高”的SSL版本号**。对于SSL v3.0，版本为0x3000；而对于TLS v1.0,版本为0x0301。值得注意的是，“版本”是客户端向服务器表明它希望使用的版本，但并没有办法告诉服务器它不使用什么版本。版本是一个非常重要的字段，它背后直接指出了消息是何种格式，参数如何计算等等，反映在实现上，就是一组不同的握手函数的集合。

随机数（random）：或者称为客户端随机数，32个字节。按照SSL说明，随机数由两部分组成：时间和随机字节。“时间”字段长4个字节，为消息产生时客户端的时间（自格林威治1970年1月1日午夜12点以来的秒数）；“随机字节”为28字节的随机数。尽管存在“时间”字段，但并不要求客户端时钟需要任何特殊的设置，也并不要求客户端和服务器时钟需要同步。整个随机数处理时并不要求作任何的检查。实际上，甚至头四个字节与时间并无关系，握手照样可以无差错地进行。协议说明中明确指出，高层协议可对此作一些额外要求。

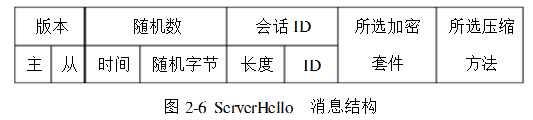
会话ID（session\_id）：这是目前为止第一个变长的字段。对于所有变长字段，在发送字段本身的内容前，必须先附加其长度。至于用多少个字节表示长度，原则是够用即可，根据各字段的最大长度不同而不同。例如，如果有一个变长字段的最大长度为216-1个字节，那么，用两个字节表示其长度就够了。SSL说明中对消息的定义并没有显式地标记出变长字段的“长度”，而是通过定义其为一变长字段，并给出最长长度来表示。在此特别标记出长度，是为了强调其存在和位置，也便于后面的分析。会话ID是一个从0字节到32字节的变长数组。所以“长度”只需1个字节。一个会话至少包括：会话ID，对方的证书，所协商好的加密套件，生成的MasterSecret，以及是否可重用的标识。当客户端想重复使用之前某次连接时的加密密钥资料时，它就提供这么一个会话ID。恢复一个会话在一些应用很重要，因为每次产生新的加密资料开销昂贵。如果想开始一个全新的握手过程，那么就只需提供长度为0的会话ID，即将“长度”字段设置为0，此时并不存在“ID”。

加密套件列表（cipher\_suites）：长度从2到216-1个字节的变长字段。“长度”占两个字节。一个加密套件用两个字节表示，如0x000A。加密套件定义了密钥交换算法、密钥交换中所使用的签名算法、加密算法和消息摘要算法，并标记了该套件是否是“可出口”的。由于历史上美国政府曾经对出口的加密算法的密钥长度有限制，对于被标记为“可出口”的加密套件，在握手过程中需要一些特别的步骤。客户端在列表中给出了其支持的加密套件的编码。至于客户端选择支持哪些加密套件，并且如何排列这些套件的顺序，协议说明没有作出任何规定，完全依赖于实现。

压缩方法列表（compression\_methods）：长度从2到28-1个字节的变长字段。“长度”占1个字节。每个压缩算法用一个字节表示。

### ServerHello消息

ServerHello是服务器对ClientHello的应答消息，结构如图2-6所示，主要携带服务器在客户端提供的列表中进行选择的结果以及交换服务器随机数



版本（server\_version）：格式与ClientHello中的“版本”格式和取值一样，表示服务器使用的协议版本。

随机数（random）：或者称为服务器随机数，以区别于客户端随机数。其格式与ClientHello中的“随机数”相同。服务器也提供随机数的目的是，即使客户端出现在两次握手中使用同一个随机数的情况，由于服务器随机数也参与计算，从而可以极大可能保证最终导出的密钥是不同的。有关的计算请见下文。

会话ID（session\_id）：其格式与ClientHello中的“会话ID”相同。如果客户端在ClientHello中请求恢复某一会话，服务器在其会话缓冲中进行查找，若找到而且会话是可恢复的，那么“会话ID”应来自所恢复的会话，并且与ClientHello中的“会话ID”应一致。会话的恢复是另一种握手方式，而且可以认为是“最精简”的握手，在一些情况下会使用。这里不讨论。通常情况下，服务器会建立一个新的会话，并提供一个客户端可以用来恢复此会话的ID。当然，如果服务器不准备恢复此会话，那么可以提供长度为0的ID。SSL说明中并没有指出如何来构造会话ID，完全依赖于具体的实现。

所选加密套件（cipher\_suite）： 2个字节。服务器从接收到的ClientHello的“加密套件列表”中，选择一个其支持的加密套件，并通过此字段告诉客户端。SSL说明文档中并没有指出服务器应该如何去选择加密套件。服务器有足够的自由。服务器既可以按照列表的中的先后顺序进行选择，也可以根据自己的规则来作出选择。客户端无法预知服务器的这个行为。如果是恢复的会话，此值应来自所恢复的会话。

所选压缩方法（compression\_method）：1个字节。服务器从接收到的ClientHello的“压缩方法列表”中，选择一个其支持的压缩方法。如果是恢复的会话，此值应来自所恢复的会话。

### Certificate消息

证书作为PKI（Public Key Infrastructure，公钥基础设施）的重要组成部分，在SSL协议中也扮演了重要的角色。在通常的情况下，服务器都给客户端发送其证书，以便客户端可以确认服务器的身份。而证书中所携带的关于服务器的公钥信息，是后面对有关消息内容进行加密或者进行验证的依据。

证书文件有多种格式。SSL协议中使用的是X.509 V3格式的证书文件。证书的结构、颁发以及验证等内容是一个专门而涉及广泛的话题，在此并不过多的进行讨论。通信双方都会有一系列可信任的根证书。所谓根证书，是指CA（Certificate Authority）自己给自己签发的自签名的证书。CA是分级的，具有层次结构。例如，CA给CA1签发证书Cert1，而CA1给CA2签发证书Cert2，CA2给用户Customer1签发证书CertCus，以证明Customer1的身份及其相关信息。当另外一个用户Customer2需要验证Customer1身份的时候，一般来说，Customer2需要CertCus、Cert2、Cert1以及CA的根证书，通过逐级验证，才能确定Customer1的身份。而CertCus-Cert2-Cert1-CA的根证书，就组成了一条证书链。Customer1并不需要发送整个证书链，特别是CA的根证书，只要Customer2拥有的可信任的证书能成功构造这条证书链就可以了。

Certificate消息正是服务器向客户端发送必要的证书，使客户端能对其身份进行验证的消息。该消息的结构很简单，如图2-7所示。



消息长度：3个字节，整个消息的剩下的长度。

第n份证书长度：3个字节，表示第n份证书的内容有多少个字节。 第n份证书内容：证书内容。

证书的顺序是，第一份是服务器自己的证书，如上文中的CertCus；第二份是给其颁发证书的CA的证书，如上文中的Cert2，如此类推

### ServerHelloDone消息

这是一条特别的有着重要作用的空消息。在一些特别的握手过程中，在证书之后可能还有其他的消息。ServerHelloDone表示服务器已发送完此阶段的全部信息。

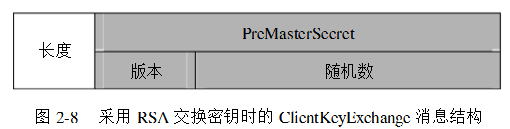
在SSL说明中对ServerHelloDone是这样定义的：

struct { } ServerHelloDone;

到此，客户端和服务器已经协商了加密套件，交换了各自的随机数，并且客户端能够对服务器的身份进行验证。

### ClientKeyExchange消息

这是一条非常重要的消息。而且其具体的格式与通过ClientHello与ServerHello所协商的加密套件密切相关。本文重点关心的是基于RSA作为密钥交换的握手，这是普遍使用的情况。在普通的RSA握手情况下，通信双方所传送的消息较其他密钥交换算法要少，比如DH或者Fortezza。此时ClientKeyExchange结构如图2-8所示。



长度：两个字节，表示消息剩下的字节数。

PreMasterSecret：客户端传送给服务器的一个重要数据，该数据将用于下文讨论的MasterSecret的计算。PreMasterSecret由两部分组成：客户端版本，两个字节；46字节的随机数。在此情况下，PreMasterSecret的明文长48字节，并且将使用服务器证书中所携带的RSA公钥对其进行加密，然后再传输。加密后的PreMasterSecret 也称为EncryptedPreMasterSecret。“长度”准确地说，应该说是此处的密文长度。例如，如果RSA的密钥是1024位，那么“长度”就应该是128字节，即0x0080。

### ChangeCipherSpec消息

ChangeCipherSpec消息是一条非常特殊的消息。无论何种握手形式，它均出现在握手过程当中。但从图2-1看到，它并不是握手协议里定义的消息，而是与握手协议并列的一种协议，被称为改变加密说明协议，有其自身的格式。

客户端和服务器都会发送ChangeCipherSpec。在会话恢复的握手情况下，是服务器首先向客户端发送，其余情况下则是客户端先发送。消息的内容非常简单，只有一个值为1的字节。

ChangeCipherSpec是告诉接收方，接下来的消息将采用新协商的加密套件和密钥进行通信。这将导致一系列的状态变化。这包括：所用的压缩算法、加密算法、摘要算法以及所使用的加密密钥、MAC密钥等等。通信双方独立地维护自己的状态，并且分读写两种状态，各有各的一套密钥资料。服务器的读状态与客户端的写状态相对应，写状态与客户端的读状态相对应。发送ChangeCipherSpec的一方，会改变自己的写状态，下条消息将使用新密钥资料进行发送；而收到ChangeCipherSpec的一方则改变自己的读状态，下条消息将使用新密钥资料进行读取。密钥资料不传送，双方独立计算，但结果应该是一致的。ChangeCipherSpec需要在双方协商完相关安全参数之后，在Finished消息之前发送。

### Finished消息

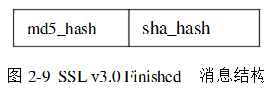
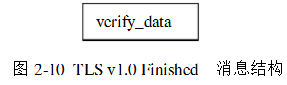
Finished消息是第一条使用新的加密参数进行加密传输的消息，它的作用是使双方能够确认没有任何握手消息被篡改过。通信双方均需要发送此消息。一方发送完ChangeCipherSpec之后，应该马上发送Finished消息。当双方都发送完Finished消息后，整个握手过程就正式完成。Finished消息也是需要分析的最后一条握手消息。

Finished是SSL v3.0与TLS v1.0中在结构定义上存在差别的唯一一条消息。

图2-9中是SSL v3.0的Finished结构。它

含有两个字段md5\_hash、sha\_hash。而图2-10则是TLS v1.0的Finished结构，它只含有一个字段sha\_hash。md5\_hash、sha\_hash和verify\_data都是某种运算法则下计算出来的消息摘要。其中md5\_hash长16字节，sha\_hash长20字节，verify\_data长12字节。

尽管具体内容不同，但思想是一致的。双方都是通过某种方法对所有握手消息和计算出来的MasterSecret进行消息摘要，并向另一方发送此摘要。而另一方在本地进行相同的运算，并将计算结果与接收结果比较，如若一致，则说明握手过程没有被篡改。

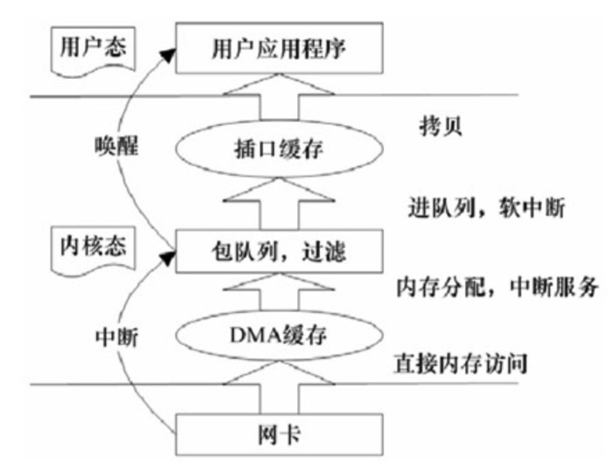
# 源码分析

\external\tcpdump\Android.mk

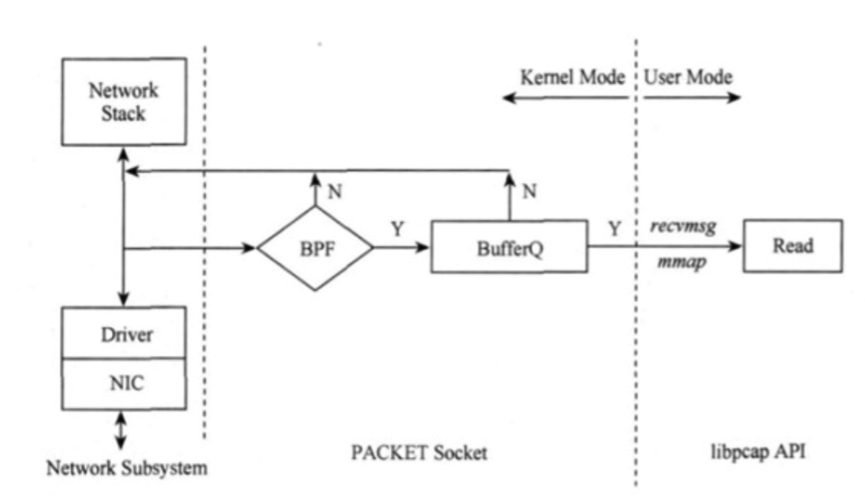
LOCAL\_MODULE\_TAGS := debug

Debug的系统才会有！！

先来看看包传递过来的流程，如下图。包从网卡到内存，到内核态，最后给用户程序使用。我们知道tcpdump程序运行在用户态，那如何实现从内核态的抓包呢？



是通过libpcap库来实现的，tcpdump调用libpcap的api函数，由libpcap进入到内核态到链路层来抓包,如下图。图中的BPF是过滤器，可以根据用户设置用于数据包过滤减少应用程序的数据包的包数和字节数从而提高性能。BufferQ是缓存供应用程序读取的数据包。我们可以说tcpdump底层原理其实就是libpcap的实现原理



而libpcap在linux系统链路层中抓包是通过PF\_PACKET套接字来实现的(不同的系统其实现机制是由差异的)，该方法在创建的时候，可以指定第二参数为SOCK\_DGRAM或者SOCK\_RAW，影响是否扣除链路层的首部。

            libpcap在内核收发包的接口处将skb\_clone()拿走的包.

关于内核中如何注册网络协议和钩子函数的过程，此处先不展开，后续专门讲解。我们接下去是看下libpcap的一些实现及其api.

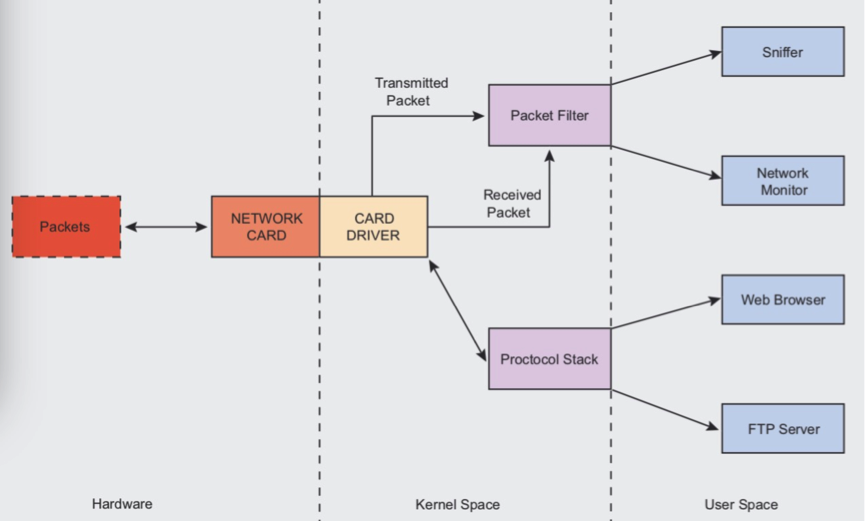
## Libpcap

当在系统中输入tcpdump –version的时候，输出的其实还有libpcap，足见其在tcpdump中的地位。

            其实最早的编译系统和过滤引擎是在tcpdump项目中的，后来为了编译其他抓包的应用，将其独立出来。现在libpcap提供独立于平台的库和API，来满足执行网络嗅探。

tcpdump.c正式使用libpcap里的函数完成两个最关键的动作：获取捕获报文的接口，和捕获报文并将报文交给callback。

libpcap支持“伯克利包过滤（BPF）”语法。BPF能够通过比较第2、3、4层协议中各个数据字段值的方法对流量进行过滤。Libpcap的使用逻辑如下图：



如果愿意，大家也可以基于libpcap开发一个类似tcpdump的抓包工具。需要注意的是如果使用分组捕获设备，只能在单个接口上接收到达的分组。

## 参考

tcpdump源码分析(2)——抓包原理

https://blog.csdn.net/notbaron/article/details/79735414

# 测试与分析的方法

利用tcpdump对android设备进行抓包，用wireshark进行抓包分析

tcpdump下载[http://www.strazzere.com/android/tcpdump](http://www.strazzere.com/android/tcpdump" \t "_blank)

把tcpdump放到/system/xbin下 chmod +x tcpdump

tcpdump -p -vv -s 0 -w /sdcard/capture.pcap

指定网口

tcpdump -p -vv -s 0 -i wlan0 -w /sdcard/and7.pcap

[https://help.aliyun.com/knowledge\_detail/43742.html](javascript:%20void%200)

可以用网上下载的

adb shell tcpdump -n -vvv -c 1000 -w /sdcard/tcpdump\_ok.cap

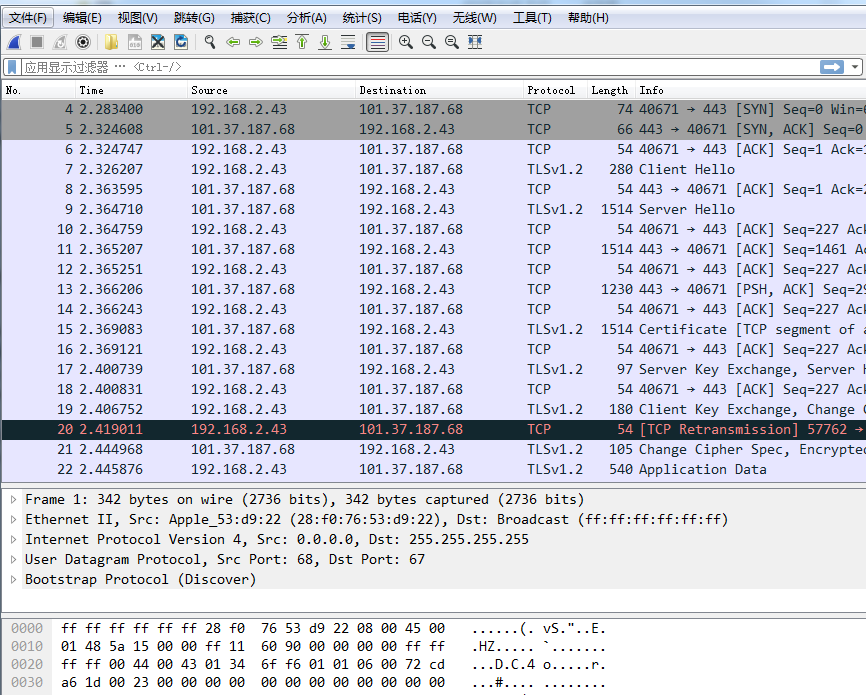
root后

把tcpdump放到/system/xbin下 chmod +x tcpdump

绑定ip

adb shell cat /system/etc/hosts

ping www.skypixel.com



## 其他

android设备抓包的方式还有不止这一种，假如你的电脑能开wifi热点的话，可以让android设备连接自己的wifi热点。可以直接用wireshark直接进行抓包，会简便很多。

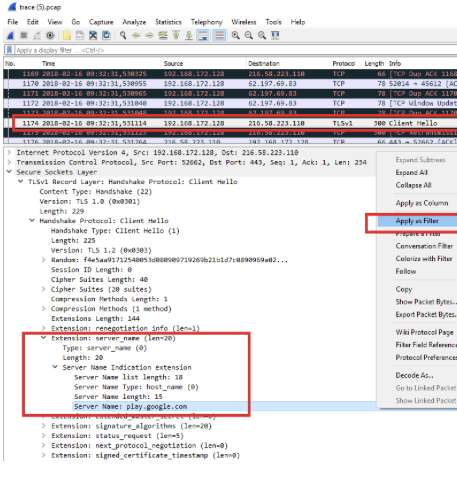
抓包以及分析技能做网络应用的童鞋们还是需要掌握的，比如访问服务器速度过慢的原因，也可以通过抓包来进行分析

## 实例分析

本文只讨论SSLv3.0和TLSv1.0

OpenSSL是一个庞大而且复杂的软件包，涉及的内容包括SSL实现、密码算法实现和证书管理等等，代码中包括了大量的数据结构和宏定义。本文重点关注的是以RSA为基础的典型的握手过程，这是讨论和分析其他形式握手的基础。由于TLS v1.0与SSL v3.0存在极大的相似，本文重点放在SSL v3.0。对于OpenSSL，重点也将放在SSL v3.0的实现代码。以下如无特别说明，均以SSL来指代SSL v3.0与TLS v1.0。如需分别给予说明，则会特别指出。

测试方法：tcpdump，埋点

你发起SSL请求的时候，在client hello阶段，没有带有server name的extension  
，如果支持SNI的包头，应该是类似这样的

# REF

[OpenSSL握手过程的测试与分析](https://wenku.baidu.com/view/5fee49d1240c844769eaee8b.html)