SSH 简介

1. 对称加密和非对称加密

对称加密又称私钥加密。是说加密方和解密方用的都是同一个 key，这个 key 对于加密方和解密方来说是保密的，第三方是不能知道的。在第三方不知道私钥的情况下，是很难将加密的数据解密的。一般来说是加密方先产生私钥，然后通过一个安全的途径来告知解密方这个私钥。

非对称加密又称公钥加密.是说解密的一方首先生成一对密钥，一个私钥一个公钥，私钥不会泄漏出去，而公钥则是可以任意的对外发布的。用公钥进行加密的数据，只能用私钥才能解密。加密方首先从解密方获取公钥，然后利用这个公钥进行加密，把数据发送给解密方。解密方利用私钥进行解密。如果解密的数据在传输的过程中被第三方截获，也不用担心，因为第三方没有私钥，没有办法进行解密。

2. 数据一致性

数据一致性是说一段数据在传输的过程中没有遗漏、破坏或者修改过。一般来说，目前流行的做法是对数据进行 hash，得到的 hash 值和数据一起传输，然后在收到数据的时候也对数据进行 hash，将得到的 hash 值和传输过来的 hash 值进行比对，如果是不一样的，说明数据已经被修改过；如果是一样的，则说明极有可能是完整的。

3. 身份验证

身份验证说的是，判断一个人或者机器是不是就是你想要联系的。也就是说如果 A 想要和 B 通信，一般来说开始的时候会交换一些数据，A 怎么可以判断发送回来的数据就真的是 B 发送的呢？现实中有很多方法可以假冒一个机器。

在 SSH 里面，这主要是通过公钥来完成的。首先客户端会有一个公钥列表，保存的是它信任的机器上面的公钥。在开始 SSH 连接之后，服务器会发送过来一个公钥，然后客户端就会进行查找，如果这个公钥在这个列表里面，就说明这个机器是真的服务器。

4. SSH 协议

SSH2 协议分为 3 个子协议，分别是 SSH-TRANS, SSH-AUTH 和 SSH-CONN。SSH2 协议的基本框架如下图所示。

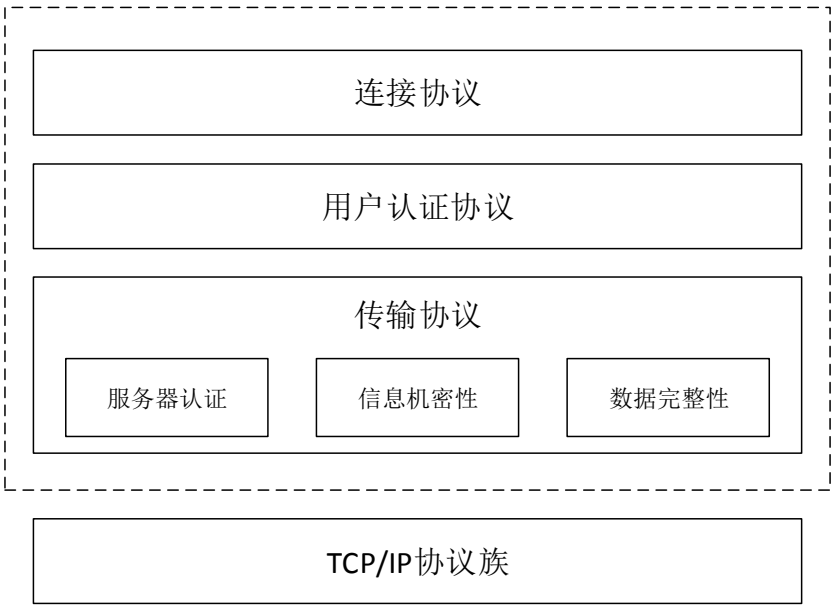


图 4.1 SSH2 协议基本框架

(1) SSH-TRANS

传输层协议 [SSH-TRANS] 提供了服务器认证，保密性及完整性。此外它有时还提供压缩功能。 SSH-TRANS 通常运行在 TCP/IP 连接上，也可能用于其它可靠数据流上。 SSH-TRANS 提供了强力的加密技术、密码主机认证及完整性保护。该协议中的认证基于主机，并且该协议不执行用户认证。更高层的用户认证协议可以设计为在此协议之上。该层涉及两个阶段：SSH 版本协商阶段、算法协商阶段。

(2) SSH-AUTH

用户认证协议（The User Authentication Protocol） 则为服务器提供客户端的身份鉴别

(3) SSH-CONN

连接协议 [SSH-CONNECT] 将多个加密隧道分成逻辑通道。它运行在用户认证协议上。它提供了交互式登录话路、远程命令执行、转发 TCP/IP 连接和转发 X11 连接。

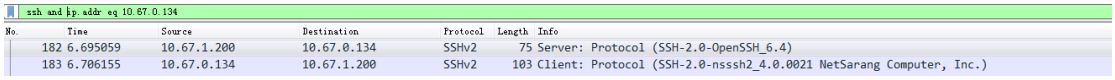
一旦建立一个安全传输层连接，客户机就发送一个服务请求。当用户认证完成之后，会发送第二个服务请求。这样就允许新定义的协议可以与上述协议共存。连接协议提供了用途广泛的各种通道，有标准的方法用于建立安全交互式会话外壳和转发（“隧道技术”）专有 TCP/IP 端口和 X11 连接。

5. SSH 工作过程

(1) 版本协商阶段

服务器等待客户端连接，客户端发起 TCP 连接请求后，服务器返回客户端一个版本号；客户端收到报文后，解析该数据包，如果能使用服务器端的协议版本号，则使用，否则使用自己的协议版本号，然后将协议版本号发送至服务器；服务器比较客户端发来的版本号，决

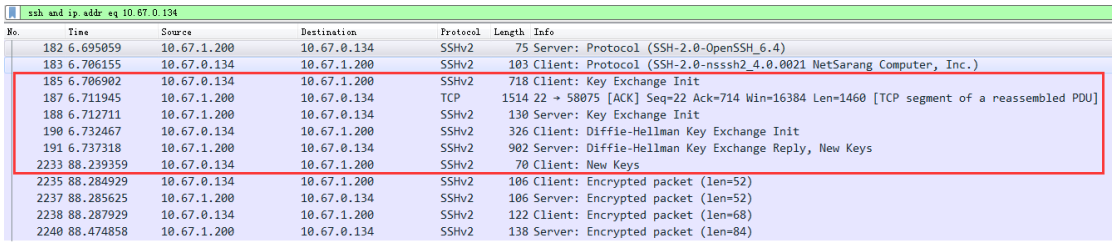
定是否能同客户端一起工作。如果能一起工作，进入下一阶段，如不能服务器断开 TCP 连接，会话结束。 第一阶段均为明文传输。



No.	Time	Source	Destination	Protocol	Length	Info
182	6.695059	10.67.1.200	10.67.0.134	SSHv2	75	Server: Protocol (SSH-2.0-OpenSSH_6.4)
183	6.706155	10.67.0.134	10.67.1.200	SSHv2	103	Client: Protocol (SSH-2.0-nsssh2_4.0.0021 NetSarang Computer, Inc.)

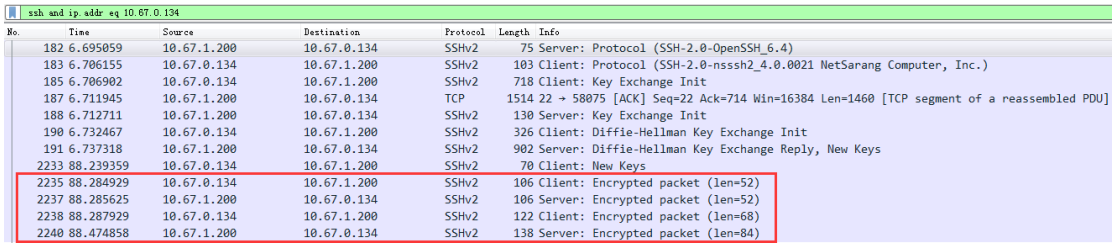
(2) 算法协商阶段

第一阶段完成后，服务器和客户端向对方发送一个自己支持的公钥算法列表、加密算法列表、消息验证码算法列表、压缩算法列表等。 确定最终使用的算法，然后使用 DH 交换算法，利用主机密钥对等参数，生成会话密钥和会话 ID，DH 交换算法仅用于客户端和服务器会话密钥和会话 ID 的交换，并不用于消息的加密。经过这个 DH 交换算法后，客户端和服务端会有相同的会话秘钥 K 和会话 ID，需要说明的是，这两个参数并不是哪一方直接生成传给对方的，而是通过 DH 交换算法各自经过复杂计算得到，其中会话 ID 是会话秘钥 K 与现有信息通过求 hash 得到。如果双方得到的 K 不一致，这个阶段就会失败，断开连接，如果成功，客户端最后会返回 New Keys。



No.	Time	Source	Destination	Protocol	Length	Info
182	6.695059	10.67.1.200	10.67.0.134	SSHv2	75	Server: Protocol (SSH-2.0-OpenSSH_6.4)
183	6.706155	10.67.0.134	10.67.1.200	SSHv2	103	Client: Protocol (SSH-2.0-nsssh2_4.0.0021 NetSarang Computer, Inc.)
185	6.706902	10.67.0.134	10.67.1.200	SSHv2	718	Client: Key Exchange Init
187	6.711945	10.67.1.200	10.67.0.134	TCP	1514	22 → 58075 [ACK] Seq=22 Ack=714 Win=16384 Len=1460 [TCP segment of a reassembled PDU]
188	6.712711	10.67.1.200	10.67.0.134	SSHv2	130	Server: Key Exchange Init
190	6.732467	10.67.0.134	10.67.1.200	SSHv2	326	Client: Diffie-Hellman Key Exchange Init
191	6.737318	10.67.1.200	10.67.0.134	SSHv2	902	Server: Diffie-Hellman Key Exchange Reply, New Keys
2233	88.239359	10.67.0.134	10.67.1.200	SSHv2	70	Client: New Keys
2235	88.284929	10.67.0.134	10.67.1.200	SSHv2	106	Client: Encrypted packet (len=52)
2237	88.285625	10.67.1.200	10.67.0.134	SSHv2	106	Server: Encrypted packet (len=52)
2238	88.287929	10.67.0.134	10.67.1.200	SSHv2	122	Client: Encrypted packet (len=68)
2240	88.474858	10.67.1.200	10.67.0.134	SSHv2	138	Server: Encrypted packet (len=84)

后续所有的会话，客户端和服务端都会使用同一加密密钥加密和解密，这里的加密秘钥由会话秘钥 K 会话 ID 加上其他一些信息 hash 计算得到，由于客户端和服务端有相同的会话秘钥 K 会话 ID，因此计算的加密秘钥是相同的，这里用到了对称加密算法，通常采用 aes 算法。



No.	Time	Source	Destination	Protocol	Length	Info
182	6.695059	10.67.1.200	10.67.0.134	SSHv2	75	Server: Protocol (SSH-2.0-OpenSSH_6.4)
183	6.706155	10.67.0.134	10.67.1.200	SSHv2	103	Client: Protocol (SSH-2.0-nsssh2_4.0.0021 NetSarang Computer, Inc.)
185	6.706902	10.67.0.134	10.67.1.200	SSHv2	718	Client: Key Exchange Init
187	6.711945	10.67.1.200	10.67.0.134	TCP	1514	22 → 58075 [ACK] Seq=22 Ack=714 Win=16384 Len=1460 [TCP segment of a reassembled PDU]
188	6.712711	10.67.1.200	10.67.0.134	SSHv2	130	Server: Key Exchange Init
190	6.732467	10.67.0.134	10.67.1.200	SSHv2	326	Client: Diffie-Hellman Key Exchange Init
191	6.737318	10.67.1.200	10.67.0.134	SSHv2	902	Server: Diffie-Hellman Key Exchange Reply, New Keys
2233	88.239359	10.67.0.134	10.67.1.200	SSHv2	70	Client: New Keys
2235	88.284929	10.67.0.134	10.67.1.200	SSHv2	106	Client: Encrypted packet (len=52)
2237	88.285625	10.67.1.200	10.67.0.134	SSHv2	106	Server: Encrypted packet (len=52)
2238	88.287929	10.67.0.134	10.67.1.200	SSHv2	122	Client: Encrypted packet (len=68)
2240	88.474858	10.67.1.200	10.67.0.134	SSHv2	138	Server: Encrypted packet (len=84)

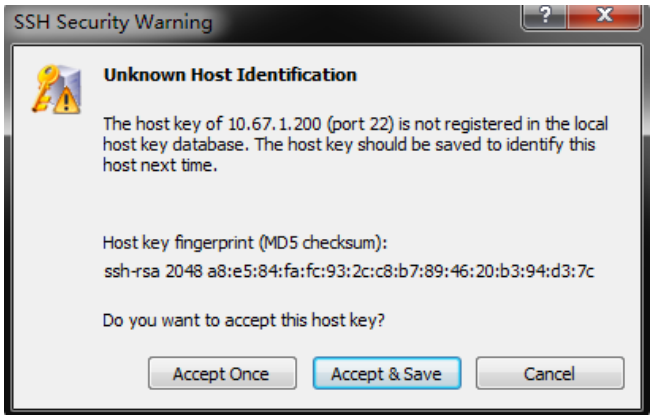
特别说明：在协商阶段之前，服务器端已经生成 RSA 或 DSA 密钥对，他们主要用于参与会话密钥的生成。

(3) 认证阶段

认证阶段包括两种方式：用户名密码、公钥认证。

◆ 用户名密码

当我们第一次登陆时，会弹出如下提示框



这里表达的意思是只知道此 host 的公钥指纹，还继续连接吗？ 我们输入 yes 之后，服务器将其公钥发送给客户端，客户端将公钥保存在 .ssh/known_hosts 里，后续再也不会看到输入 yes/no 的这一步。

到这里，服务器请求客户端输入密码，客户端获取用户输入的密码后使用服务器的公钥进行加密并发送给服务器，服务器使用自己的私钥进行解密，对比密码是否是该用户的密码，如是则允许登陆，接收客户端用户发送的指令，如否则返回验证失败，拒绝登陆。当然指令也是经过公钥加密的，所有的会话都会再经过第二阶段产生的会话密钥进行加密，所以传输是相当安全的。

- 公钥认证

使用公钥认证（免密码登陆）。这功能首先需要我们手动将客户端的公钥复制到服务器的 authorized_keys 文件中后才能实现，相当于服务器获取了客户端的公钥。

在公钥认证阶段时，服务端用客户端的公钥加密一个 256 位的随机字符串，客户端接收后使用自己的私钥解密，然后将这个字符串和会话 id 合并在一起，对结果应用 MD5 散列函数并把散列值返回给服务器，服务器进行相同的 MD5 散列函数处理，如果客户端和该值可以匹配，那么认证成功，允许登陆，达到免密登陆的效果。 至此认证阶段结束。当我们第一次使用 git 时，通常会让你配置 ssh key 到 github 上，这里就是为了以后公钥认证。

(4) 会话请求阶段

- 服务器等待客户端的请求
- 认证通过后，客户端向服务器发送会话请求
- 服务器处理客户端的请求。
- 请求被成功处理后， 服务器会向客户端回应 SSH_SMSG_SUCCESS 包，SSH 进入交

互会话阶段

- 否则回应 SSH_SMSG_FAILURE 包，表示服务器处理请求失败或者不能识别请求

(5) 交互会话阶段

- 客户端将要执行的命令加密后传给服务器
- 服务器接收到报文，解密后执行该命令,将执行的结果加密发还给客户端
- 客户端将接收到的结果解密后显示到终端上