Security Shell（SSH）协议中文版

版本：1.0

广东金赋信息科技有限公司
Guangdong Kamfu Information & Technology Co., Ltd.

地址：广东省佛山市南海区桂城东平路瀚天科技城综合楼三楼　邮编：528200
电话：(0757)88023456　传真：(0757)88351111
网址：www.kamfu.com.cn　电子邮件：kamfu@kamfu.com.cn

文档信息

标题	Security Shell（SSH）协议中文版
描述	本文档是 Security Shell（SSH）协议系列文档（RFC 4251～4254）的中文翻译。
创建日期	2011 年 6 月 10 日
翻译作者	薛立徽

修订记录

日期	版本	描述	作者

目　录

SSH 协议架构

摘要

SSH 协议是在不安全的网络上进行安全远程登录和其他安全网络服务的协议。本文描述 SSH 协议的架构，以及 SSH 协议文档中所用的惯用约定和术语。本文也讨论了允许本地扩展的 SSH 算法命名体系。SSH 协议由三个主要部分组成：传输层协议（Transport Layer Protocol）提供服务器验证、保密性和完整性，具完全前向安全性（perfect forward secrecy）。验证协议（Authentication Protocol）向服务器验证客户端。连接协议（Connection Protocol）将加密隧道复用为若干逻辑信道。

1．简介

SSH（Security Shell）是在不安全的网络上进行安全远程登录和其他安全网络服务的协议。它由三个主要部分组成：

- 传输层协议[SSH-TRANS]提供服务器验证、保密性和完整性。它也可选的提供压缩。传输层一般运行在一个 TCP/IP 连接上，但也可能被用在任何其他可靠的数据流上。
- 验证协议[SSH-USERAUTH]向服务器验证客户端用户。它运行在传输层协议上。
- 连接协议[SSH-CONNECT]将加密隧道复用为若干逻辑信道。它运行在验证协议上。

在一个安全的传输层连接被建立后，客户端发送一个服务请求。在用户验证完成后，第二个服务请求被发出。这允许新的协议被定义并与上述协议共存。

连接协议提供的信道可被用于广泛的目的。提供了标准方法用于建立安全的交互式 shell 进程以及转发（隧道）任意 TCP/IP 端口和 X11 连接。

2．撰稿者

（略）

3．本文中的惯用约定

所有与 SSH 协议相关的文档都应使用关键词“必须”、“禁止（绝不能）”、“必须的”、“应”、“不应”、“推荐”、“可（能）”、“可选的”来描述要求。这些关键词的含义符合[RFC2119]的描述。

本文中用于描述命名空间分配的关键词"PRIVATE USE"、"HIERARCHICAL ALLOCATION"、"FIRST COME FIRST SERVED"、 "EXPERT REVIEW"、"SPECIFICATION REQUIRED"、"IESG APPROVAL"、"IETF CONSENSUS"以及"STANDARDS ACTION"的含义符合[RFC2434]的描述。

协议文档中定义了协议域和可能的取值。协议域将在消息定义中定义。例如，SSH_MSG_CHANNEL_DATA 的定义如下：

```
byte        SSH_MSG_CHANNEL_DATA
uint32      recipient channel
string      data
```

在整套协议文档中，当引用域时，域的名称出现在单引号中。当引用域的值时，它们出现在双引号中。例如，'data'的可能取值是"foo"和"bar"。

4．架构

4.1．主机密钥

每一个服务器主机应有一个主机密钥。主机可有多个使用不同算法的主机密钥。多个主机可共享相同的主机密钥。如果主机具有密钥，那么对每一种必须的公钥算法（DSS）必须有至少一个密钥。

服务器主机密钥在密钥交换中用于检验客户端真的是在和正确的服务器对话。为使之可能，客户端必须事先知道服务器主机密钥的公钥。

两个不同的信任模型可被使用：

- 客户端有一个本地的数据库，将每个主机名（与用户输入相同）与对应的主机密钥公钥关联起来。这种方法不需要集中管理的基础架构或第三方的配合。缺点是主机名-密钥关联数据库的维护可能成为一个负担。

- 主机名-密钥关联由一个可信的认证机构（CA）证明。客户端只知道 CA 根密钥，并且能够检验所有由 CA 认证的主机密钥的有效性。第二种方式消除了维护问题，因为理想的只有一个 CA 密钥需要被安全的保存在客户端。但另一方面，在进行验证前，每个主机密钥必须被一个中心机构以适当的方式认证。同时，中心基础架构被寄予了极大的信任。

协议提供提供了这种选项：在第一次连接到主机时不检查服务器名-主机密钥的关联。这允许在事先不知道主机密钥或证书的情况下进行通信。连接仍然提供对被动侦听（passive listening）的保护；但是，它容易受到主动的中间人攻击（active man-in-the-middle attacks）。系统实现正常情况不应默认的允许这种连接，因其造成潜在的安全问题。但是，由于在撰写本文时互联网上没有一个被广泛部署的密钥基础架构，这一选项使协议在这种基础架构出现前的过渡时期内的可用性大幅提高，同时仍然提供了比旧的解决方案（如 telnet 和 rlogin）高得多的安全性。

系统实现应尽量检查主机密钥。一个可能的策略是：只有当第一次连接到一个主机时才不经检查的接受主机密钥，把密钥保存到本地数据库，在将来所有的到该主机的连接中都以此为基准进行对比。

系统实现可提供附加的方法来检验主机密钥的正确性，例如，通过 SHA-1 哈希从公钥产生一个十六进制的数字指纹。这种数字指纹能够很容易的用电话或其他外部通信通道来检验。

所有系统实现应提供一个选项：不接受不能被检验的主机密钥。

本工作组的成员相信“易用”是终端用户接受安全解决方案的关键，如果新的解决方案没有并采用，就不会带来任何安全性的提高。因此，提供不检查服务器主机密钥的选项提高了 Internet 整体的安全性，尽管在允许该选项的设置中它降低了协议的安全性。

4.2. 扩展性

我们相信协议将会随时间演化，一些组织将希望使用它们自己的加密、验证及/或密钥交互方法。对所有扩展进行集中注册是繁琐的，特别对于试验性或保密的特性。另一方面，没有集中注册会导致方法标识符冲突，影响互操作性。

我们选择通过特定格式的名称来标识算法、方法、格式和扩展协议。DNS 名称被用于创建本地命名空间，在其中可定义试验性或保密的扩展而不用担心与其他系统实现相冲突。

一个设计目标是保持基础协议尽可能简单，并且要求尽可能少的算法。但是，所有系统实现必须支持一个最小的算法集合以保证互操作性（这不表示所有主机上的本地策略要允许这些算法）。强制性的算法在相关协议文档中规定。

附加的算法、方法、格式和扩展协议可在单独的文档中定义。参见第 6 节，算法命名。

4.3. 策略问题

协议允许对加密、完整性、密钥交换、压缩以及公钥算法和格式进行完整的协商。两个传输方向的加密、完整性、公钥和压缩算法可以不同。

下列策略问题应在系统实现的配置机制中被解决：

- 每个传输方向的加密、完整性和压缩算法。策略必须规定哪一个是优选算法（例如，在每个类别下列出的第一个算法）。

- 主机验证使用的公钥算法和密钥交换方法。已存在的使用不同算法的受信任的主机密钥也影响这一选择。

- 服务器对每个用户要求的验证方法。服务器的策略可对一些或所有用户要求多重认证。要求的算法可依赖于用户试图获得授权时所在的位置。

- 用户被允许使用连接协议进行的操作。一些问题与安全性有关；例如，策略不应允许服务器在客户机上启动进程或运行命令，并且必须不允许连接到验证代理，除非被要求转发这样的连接。另一些问题，例如谁能够转发那个 TCP/IP 端口，非常明显是本地策略。很多这样的问题可能涉及穿越或绕过防火墙，并且与本地安全策略互相联系。

4.4. 安全特性

SSH 协议的主要目的是改善 Internet 的安全性。它尝试通过一种容易部署的方式实现该目的，为此，甚至不惜牺牲绝对安全。

- 所使用的所有加密、完整性，以及公钥算法都是广为人知的、成熟的算法。

- 所有算法都使用从密码学看来合理的密钥长度，即密钥长度被认为能在几十年内提供对最强的密钥解析攻击的保护。

- 所有算法都透过协商，在某些算法失效的情况下，可以容易的切换到其他算法而不需要修改基础协议。

为了使大范围、快速的部署容易实现，协议做了特定的妥协。具体而言，在验证服务器主机密钥确实属于目标主机方面，协议允许不进行验证，但这是不推荐的。这被认为在短期内能够显著的改善协议的可用性，直到出现普及的 Internet 公钥基础架构。

4.5．本地化和字符集支持

在大多数情况下，SSH 协议不直接传递会显示给用户的文本。但是，有些情况下这种数据可能被传递。当可应用时，数据的字符集必须被显式的规定。在绝大多数情况下，使用的是 ISO-10646 UTF-8 编码[RFC3629]。当可应用时，也供了一个字段用于语言标志[RFC3066]。

一个重要的问题是交互式进程的字符集。这个问题没有清晰的解决方案，因为不同的应用软件可能用不同的格式显示数据。客户端也可能使用不同类型的终端仿真，所要使用的字符集实际上是由终端仿真决定的。其结果是，没有为直接规定终端进程数据的字符集或编码提供位置。但是，终端仿真类型（例如，"vt100"）被传送到远端，它隐式的规定了字符集和编码。应用软件通常使用终端类型来决定使用什么字符集，或者字符集由一些外部方法来决定。终端仿真也可能允许设置缺省字符集。在任何情况下，终端进程的字符集被认为主要是一个客户端本地问题。

用于识别算法或协议的内部名称一般不会显示给用户，而且必须采用 US-ASCII。

客户端和服务器用户名固有的受到服务器可接受名称的约束。它们可能有时出现在日志、报告等位置。它们必须使用 ISO-10646 UTF-8 编码，但在一些情况下可能要求其他编码。由服务器决定如何将用户名称与已接受的用户名称对应起来。推荐使用直接的比特式、二进制比较。

为了本地化，协议尝试尽量减少传送的文本消息的数量。当出现时，这些消息通常与错误、除错信息或一些外部配置数据相关。对一般情况下需显示出来的数据，应使用一个数字编码使得用一个本地化的消息代替被传送的消息成为可能。文本消息应可配置。

5．SSH 协议使用的数据类型表示法

byte

byte 标识任意一个 8 位值（8 位字节）。固定长度的数据有时被表示为一个字节数组，写作 byte[n]，其中 n 是数组中字节的数量。

boolean

一个布尔值作为一个字节存储。0 表示 FALSE，1 表示 TRUE。所有非零的值必须被解释为 TRUE；但是，应用软件禁止储存除 0 和 1 以外的值。

uint32

表示一个 32 位无符号整数。按重要性降序（网络字节顺序）储存为 4 个字节。例如，

699921578（0x29b7f4aa）被存储为 29 b7 f4 aa。

uint64

表示一个 64 位无符号整数。按重要性降序（网络字节顺序）储存为 8 个字节。

string

任意长度二进制字符串。字符串用于装载任意二进制数据，包括空字符和 8 位字符。字符串被储存为 1 个包含其长度（后续字节数量）的 uint32 以及 0（=空字符串）或作为字符串的值的更多的字节。不使用终结符（空字符）。

字符串也被用来存储文本。在这种情况下，内部名称使用 US-ASCII，可能显示给用户的文本使用 ISO-10646 UTF-8。终结符（空字符）一般不应被保存在字符串中。例如，US-ASCII 字符串"testing"被表示为 00 00 00 07 t e s t i n g。UTF-8 映射不改变 US-ASCII 字符的编码。

mpint

表示二进制补码（two's complement）格式的多精度整数，存储为一个字符串，每字节 8 位，从高位到低位（MSB first）。负数的数据区的首字节的最高位（the most significant bit）的值为 1。对于正数，如果最高位将被置为 1，则必须在前面加一个值为 0 的字节。禁止包含值为 0 或 255 的非必要的前导字节（leading bytes）。零必须被存储为具有 0 个字节的数据的字符串。

例：

value (hex)	representation (hex)
0	00 00 00 00
9a378f9b2e332a7	00 00 00 08 09 a3 78 f9 b2 e3 32 a7
80	00 00 00 02 00 80
-1234	00 00 00 02 ed cc
-deadbeef	00 00 00 05 ff 21 52 41 11

name-list

一个包含逗号分隔的名称列表的字符串。名称列表表示为一个含有其长度（后续字节数量）的 uint32，加上一个包含 0 或多个逗号分隔的名称的列表。名称的长度禁止为 0，并且禁止包含逗号(",")。由于这是一个名称列表，所有被包含的元素都是名称并且必须使用 US-ASCII。上下文可能对名称有附加的限制。例如，名称列表中的名称可能必须是一系列有效的算法标识，或一系列[RFC3066]语言标识。名称列表中名称的顺序可能有也可能没有意义。这取决于使用列表时的上下文。对单个名称或整个列表都禁止使用终结字符（空字符）。

例：

value	representation (hex)
()，空名称列表	00 00 00 00

("zlib")	00 00 00 04 7a 6c 69 62
("zlib,none")	00 00 00 09 7a 6c 69 62 2c 6e 6f 6e 65

6．算法和方法命名

SSH 协议使用名称引用特定的哈希、加密、完整性、压缩和密钥交换算法或方法。有一些标准算法和方法是所有系统实现都必须支持的。也有一些算法和方法在协议规定中定义了，但是是可选的。此外，预期有一些组织将希望使用它们自有的算法或方法。

在本协议中，所有算法和方法的标识必须是由可打印的 US-ASCII 字符组成的非空字符串，长度不大于 64 字符。名称必须是大小写敏感的。

算法和方法名称有两种格式：

- 不含 at 符号("@")的名称为了等待 IETF CONSENSUS（Internet 工程任务组指派）而保留。例如"3des-cbc"、 "sha-1"、"hmac-sha1"和"zlib"（双引号不是名称的一部分）。这种格式的名称必须在 IANA （Internet 号码分配局）注册后才有效。注册的名称禁止包含 at 符号("@")、逗号(",")、空格、控制字符（ASCII 码 32 及以下）或 ASCII 码 127（DEL）。名称是大小写敏感的，并且长度禁止超过 64 字符。

- 任何人都可以使用符合 name@domainname 格式的名称定义附加的算法或方法，例如，"ourcipher-cbc@example.com"。at 符号前的部分的格式没有规定，但必须是可打印的 ASCII 字符串，而且禁止包含逗号(",")、空格、控制字符（ASCII 码 32 及以下）或 ASCII 码 127（DEL）。它必须仅有一个 at 符号。at 符号后的部分必须是一个由定义该名称的人或组织控制的有效的、完全合格的域名[RFC1034]。名称是大小写敏感的，并且长度禁止超过 64 字符。如何管理本地命名空间取决于各个域。应说明的是，这里的名称看起来与 email 地址 STD 11 [RFC0822]一样。这纯属巧合，并且与 STD 11 [RFC0822]没有任何关系。

7．消息编号

SSH 数据包的消息编号从 1 到 255。这些编号的分配如下：

传输层协议：

1 到 19	传输层通用（例如，断开连接、忽略、除错等）
20 到 29	算法协商
30 到 49	密钥交换方法规定（编号可被重用于不同的交换方法）

验证协议：

50 到 59	用户验证通用
60 到 79	用户验证方法规定（编号可被重用于不同的验证方法）

连接协议：

80 到 89　　连接协议通用
90 到 127　　信道相关消息

为客户端协议保留：

128 到 191　保留

本地扩展：

192 到 255　本地扩展

8. IANA 考虑

对[SSH-ARCH]、[SSH-TRANS]、[SSH-USERAUTH]和[SSH-CONNECT]定义的SSH协议的IANA指引详见[SSH-NUMBERS]。以下是一个简单的总结，但注意[SSH-NUMBERS]包含对IANA的实际指引，该指引将来可能被新的内容取代。

在SSH协议中，对下列类型的名称的分配由IETF CONSENSUS指定：

- 服务名称
  - 验证方法
  - 连接协议信道名称
  - 连接协议全局请求名称
  - 连接协议信道请求名称
- 密钥交换方法名称
- 指定的算法名称
  - 加密算法名称
  - MAC算法名称
  - 公钥算法名称
  - 压缩算法名称

这些名称必须为可打印的US-ASCII字串，而且禁止包含at符号（"@"）、逗号（","）、空格、控制字符（ASCII码32及以下）或ASCII码127（DEL）。名称是大小写敏感的，而且长度禁止超过64字符。

含有at符号（"@"）的名称是本地定义的扩展，不由IANA控制。

上面列出的每一个名称类别具有一个独立的命名空间。但是，应避免在多个类别中使用相同的名称以尽量避免混淆。

0至191范围内的消息编号（见第7节）是通过IETF CONSENSUS分配的，见[RFC2434]。191至255范围内的消息编号（本地扩展）为PRIVATE USE预留，同样见[RFC2434]。

9．安全性考虑

为了让全部的安全性考虑更加易懂，在这里集中了传输、验证和连接文档的安全性考虑。

传输层协议[SSH-TRANS]提供了一个在不安全的网络上的机密的信道。它执行服务器主机验证、密钥交换、加密，以及完整性保护。它也形成一个唯一的会话 id，可被更上层协议使用。

认证协议[SSH-USERAUTH]提供了一套可用于向服务器验证客户端用户的机制。验证协议规定的各个机制使用传输层协议提供的会话 id，并/或依赖传输层协议的安全性和完整性保障。

连接协议[SSH-CONNECT]规定了一种多路复用机制，用于在机密的、已验证的传输上实现多个数据流（信道）。它也规定了使用交互式命令解释程序（shell）的信道、通过端口转发在安全传输上实现各种外部协议（包括任意 TCP/IP 协议）的信道，以及使用服务器主机上的安全子系统的信道。

9.1．伪随机数生成

本协议在用于产生会话密钥的哈希值中包含随机的、对应特定会话的数据，从而将每个会话密钥与会话绑定。应特别注意保证所有随机数具有良好的质量。如果这里的随机数据（例如，DiffieHellman 参数）是伪随机的，那么伪随机数发生器应是密码学上安全的（例如，它的下一个输出不能被轻易的猜测出来，即使知道之前所有的输出），而且，适当的熵需要被加入伪随机数发生器。[RFC4086]提供了随机数和熵来源的建议。实现者（Implementers）应注意到对熵的重要性和对正确实现伪随机数生成公式的困难程度的善意的警告。

能够提供给客户端或服务器的熵可能有时达不到需要的量。在这种情况下，必须在熵不足的情况下继续使用伪随机数发生器，或者拒绝运行协议。后者是更值得推荐的做法。

9.2．控制字符筛选

当向用户显示文本时，例如错误或除错消息，客户端软件应将任何控制字符（除制表符、回车、换行以外）替换为安全的序列以避免通过发送终端控制字符的攻击。

9.3．传输

9.3.1．机密性

分析或推荐除已经成熟并被业界接受的加密器之外的特定的加密器，超出了本文和本工作组的范围。在撰写本文时，普遍使用的加密器包括 3DES、ARCFOUR、twofish、serpent 和 blowfish。AES 已由美国联邦信息处理标准发布为[FIPS-197]，并且密码学界也已接受 AES。像通常一样，实现者和用户应查阅当前的文献以保证产品使用的加密器中没有被发现新的脆弱点。实现者也应检查哪些加密器被认为相对较强，并应向用户推荐使用这些较强的加密器。当选择了一个较弱的加密器时，产品可以礼貌的、不突兀的告知用户一个更强的算法是有效的并应被使用，这应被视为一种良好的特性。

“无”这种加密器被提供用于除错并且不应被用于其他用途。它的密码学特性在[RFC2410]中得到了充分的描述，显示出它不符合本协议的目的。

在当前的文献中也可以找到这些和其他加密器的优缺点。[SCHNEIER]和[KAUFMAN]这两个参考文献可以提供该主题的信息。这两个参考文献都描述了特定加密器的加密块链接模式（CBC

mode）以及该模式的弱点。实质上，由于数据序列的开头的高可预测性，这种模式理论上对选择密文攻击（chosen cipher-text attack）是脆弱的。但是，这种攻击被认为是困难和并非完全可预测的，特别是当使用的数据块大小相对较大时。

此外，另一种 CBC 模式攻击可以使用插入包含 SSH_MSG_IGNORE 的数据包的方法来缓和。不使用这种技巧，一种特定的攻击可能成功。这种攻击（通常称为 Rogaway 攻击[ROGAWAY]、[DAI]、[BELLARE]）要生效，攻击者需要知道将被加密的下一个数据块的初始向量（IV）。在 CBC 模式下，它就是对上一个数据块加密的输出。如果攻击者还没有任何办法查看数据包（即，它是在 SSH 系统实现的内部缓冲区甚至内核中），那么这种攻击不会生效。如果上一个数据包已经被发送到网络（即，攻击者可获取它），那么他能够使用这种攻击。

在理想的情况下，实现者仅当数据包被发送到网络并且没有其他数据包等待传送时，才需要添加一个额外的数据包。实现者可能希望检查还有没有等待发送的数据包；不幸的是，从内核或缓冲区中获取该信息一般并不容易。如果没有未发送的数据包，那么一个包含 SSH_MSG_IGNORE 的数据包应被发送。如果每次攻击者知道下一个数据包假定使用的 IV 时，都向数据流中添加一个新的数据包，那么攻击者将不能猜到正确的 IV，这样攻击永远不会成功。

作为一个例子，考虑以下情况：

```
    Client                                          Server
    ------                                          ------
    TCP(seq=x, len=500)          ---->
    contains Record 1

                        [500 ms passes, no ACK]

    TCP(seq=x, len=1000)         ---->
    contains Records 1,2
                                                      ACK
```

1．发包优化算法（Nagle algorithm）+TCP 重发意味着两个记录被合并为一个 TCP 片段。
2．记录 2 不在 TCP 片段的开头，而且永远也不会，因为它获得了应答（ACK）。
3．然而，攻击是可能的，因为记录 1 已经被发出了。

这个例子指出，用 TCP 缓冲区中是否存在未清除的数据作为是否需要一个空数据包的条件是不安全的，因为当第二个 write()被执行时，缓冲区将包含没有获得应答的记录 1。

另一方面，如下情况是完全安全的：

```
    Client                                          Server
    ------                                          ------
    TCP(seq=x, len=500)              ---->
        contains SSH_MSG_IGNORE

    TCP(seq=y, len=500)              ---->
        contains Data
```

假定第二个 SSH 记录的 IV 在数据包的数据确定下来时就固定了，那么应执行以下步骤：

读取用户输入（read from user）
加密空数据包（encrypt null packet）
加密数据包（encrypt data packet）

9.3.2. 数据完整性

本协议确实允许数据完整性机制被禁用。实现者为除错之外的目的使用这个特性时应特别谨慎。当"none"类型的 MAC（数据校验码）被启用时，应向用户和管理员显式的发出警告。

只要"none"类型的 MAC 没有被使用，本协议提供数据完整性。

由于 MAC 使用一个 32 位的序号（sequence number），它在发送了 2**32 个数据包后可能开始泄漏信息。但是，根据推荐来更新密钥应可防止这种攻击。传输协议[SSH-TRANS]推荐在 1G 字节的数据后更新密钥，可能的最小的数据包是 16 字节。因此，最多在 2**28 个数据包后就应更新密钥。

9.3.3. 重放

使用除"none"之外的 MAC 提供了完整性和验证。此外，传输协议提供一个唯一的会话标识（部分与参与算法和密钥交换过程的伪随机数据绑定），它可被高层协议用于将数据与特定的会话绑定，防止之前会话数据的重放。例如，验证协议[SSH-USERAUTH]使用它来防止重放之前会话的签名。由于公钥验证交换是密码学上与会话绑定的（即，绑定到初始密钥交换），它不能在其他会话中被成功的重放。注意，会话 id 可以被公开而不会损害协议的安全性。

如果两个会话有同样的会话 id（密钥交换的哈希值），那么其中一个会话的数据包能被用于对另一个会话的重放攻击。需要强调，发生这种情况的几率，毋庸置疑的，在使用现代密码学方法时是极小的。在规定较大的哈希函数输出和 DH 参数时，更是如此。

在[RFC2085]、[RFC2246]、[RFC2743]、[RFC1964]、[RFC2025]和[RFC4120]中，对在一些情况下使用单增的序号作为 MAC 或 HMAC 的输入来检测重放进行了描述。[RFC2104]讨论了底层的构造。基本上，每个数据包中的一个不同的序号保证了 MAC 函数的输入中至少该序号是不相同的，从而提供了攻击者无法预测的不重现的 MAC 输出。但是，如果会话保持活动的时间足够长，这个序号将在到达最大值后从头开始。这可能给攻击者提供了重放一个之前记录的序号相同的数据包的机会，但仅当通讯各方在传输使用该序号的上一个数据包后没有更新密钥的情况下才成立。如果通讯各方更新了密钥，那么重放将由于 MAC 校验失败而被检测出来。为此，必须强调，通讯各方必须在序号从头开始前更新密钥。自然的，如果攻击者确实尝试在通讯各方更新密钥之前重放一个捕获的数据包，那么这个重复数据包的接收者将不能验证 MAC 的合法性并丢弃该数据包。MAC 失败的原因是，接收者将根据数据包的内容、共享秘密（shared secret）和预期的序号计算一个 MAC。由于重放的数据包使用的不是预期的序号（重放的数据包的序号在接受者一方已经过去了），计算得到的 MAC 将与从数据包接收到的 MAC 不符。

9.3.4. 中间人

本协议对分配主机公钥的基础架构或方法未做假设或规定。预期在某些时候使用本协议时，将不对服务器主机密钥和服务器主机名称的关联进行检验。这种用法对中间人攻击是脆弱的。本节对此进行描述，并劝告管理员和用户理解在初始化任何会话前检验这种关联的重要性。

有三种中间人攻击的情况要考虑。第一种是攻击者在会话初始化之前已将一个设备放置在客户端和服务器之间。在这种情况下，攻击设备试图模仿合法的服务器并会在客户端初始化会话时向客户端提供它的公钥。如果它提供的是服务器的公钥，那么它将无法对合法的服务器和客户端之间的传输进行解密或签名，除非它也获取了主机的私钥。同时，攻击设备也将冒充客户端初始化一个与合法的服务器的会话。如果服务器的公钥在会话初始化之前已被安全的分配给客户端，攻击设备提供给客户端的密钥将与客户端保存的密钥不符。在这种情况下，应警告用户提供的主机密钥与客户端暂存的主机密钥不一致。像 4.1 中描述的一样，用户可被允许接受新的密钥并继续会话。推荐上述警告包含有关客户端设备的足够的信息，以使用户能够做出有根据的决定。如果用户选择使用存储的服务器公钥（而不是在会话开始时提供的公钥）来继续会话，那么由于上面讨论的随机性，客户端-攻击者的会话与攻击者-服务器的会话中会话特有数据将不同。从这一点出发，由于攻击者没有服务器私钥，攻击者将无法对服务器发出的包含会话特有数据的数据包进行正确的签名，因此攻击无法成功。

要考虑的第二种情况与第一种情况相似，也发生在连接时，但这种情况指出需要安全的分配服务器公钥。如果服务器公钥没有被安全的分配，那么客户端无法知道自己是否在与目标服务器对话。攻击者可能使用社会工程（social engineering）技巧将让不知情的用户接受冒充的服务器密钥，然后在合法的服务器和客户端之间放置中间人攻击设备。如果发生这种情况，那么客户端将产生客户端-攻击者会话，攻击者将产生攻击者-服务器会话，攻击者将能够监控和操纵客户端和合法服务器之间的所有传输。服务器管理员最好提供检查主机密钥指纹的方法，这种方法的安全性不依赖于实际主机密钥的完整性。可能的机制除 4.1 中讨论的之外，还可能包括安全的网页、物理的纸张等。实现者应对如何以最佳的方式在其系统实现中提供这种机制给出建议。由于本协议是可扩展的，将来对本协议的扩展可能提供更好的机制来解决需要在连接前知道服务器的主机密钥的问题。例如，可能在安全的 DNS 查找（DNS lookup）中提供主机密钥指纹，或在密钥交换中使用 GSS-API（[RFC1964]）上的 Kerberos（[RFC4120]）来验证服务器。

在第三种中间人攻击的情况中，攻击者可能试图在会话建立之后操纵在通讯各方之间传输的数据包。像 9.3.3 描述的那样，以这种方式成功攻击是不大可能的。9.3.3 的推理确实假定 MAC 是安全的，并且不可能构造 MAC 算法的输入以获得已知的特定的输出。在[RFC2104]的第 6 节对此进行了详细得多的讨论。如果 MAC 算法有脆弱性或足够弱，那么攻击者可能指定特定的输入以产生已知的 MAC。通过这样，攻击者可能修改传输中的数据包的内容。或者，攻击者可能检阅捕获的数据包的 MAC，利用算法的脆弱性或弱点找到共享秘密。在上面的任何一种情况下，攻击者能够构造一个或多个数据包并插入一个 SSH 数据流。为防止这种情况，实现者最好使用普遍接受的 MAC 算法，管理员最好关注当前的密码学文献和讨论，以保证没有使用最近被发现脆弱性或弱点的 MAC 算法。

总而言之，在没有对主机和主机密钥建立可靠的绑定的情况下使用本协议在本质上是不安全的，而且是不推荐的。但是，在安全性不是非常关键的环境下，本协议的这种使用可能是必要的，并且仍将提供对被动攻击的保护。本协议和运行在本协议之上的应用程序的实现者应记住这种可能性。

9.3.5. 拒绝服务

本协议设计为基于可靠的传输。如果发生传输错误或消息操纵（message manipulation），连接将被断开。如果发生这种情况，应重新建立连接。类似断线器（wire cutter）这种拒绝服务攻击几乎无法避免。

此外，本协议对拒绝服务攻击是脆弱的，因为攻击者能够在不通过验证的情况下迫使服务器执行消耗大量 CPU 和内存的连接建立和密钥交换任务。实现者应提供使这种攻击更加困难的特性，例如，只允许有合法用户的那一部分客户端子集的连接。

9.3.6. 隐蔽信道（Covert Channels）

本协议未被设计用于消除隐蔽信道。例如，填充（padding）、SSH_MSG_IGNORE 消息，以及协议中的几个其他位置能被用于传递隐蔽信息，而且接受方没有可靠的方法来检验是否有这种信息正被发送。

9.3.7. 前向安全性

需要注意的是，Diffie-Hellman 密钥交换能提供完全前向安全性（PFS）。本质上，PFS 是一个密钥建立协议的密码学属性，表示会话密钥或长期私钥在一个特定的会话之后泄密，不会造成更早的会话的泄密[ANSI-T1.523-2001]。使用在[SSH-TRANS]中 Diffie-Hellman 密钥交换一节描述的 Diffie-Hellman 方法（包括"diffie-hellman-group1-sha1"和"diffie-hellman-group14-sha1"）创建的 SSH 会话，即使在私钥/验证资料后来被透露的情况下仍然是安全的，但如果会话密钥被透露则不再安全。因此，对于上述 PFS 的定义，SSH 确实具备 PFS。但是，这个属性不会传递给（commuted to）任何使用 SSH 作为传输的应用程序或协议。SSH 的传输层提供依赖于秘密数据的密码验证和其他方法。

当然，如果客户端和服务器的 DH 私密参数被透露，那么会话密钥被透露了，但是这些内容在密钥交换完成后可被抛弃。值得指出，不应使这些内容最后处于交换空间，它们应在密钥交换完成后立即被从内存中擦除。

9.3.8. 密钥交换方法的顺序

像[SSH-TRANS]中的算法协商一节中陈述的一样，每一个设备将发送一个优先密钥交换方法的列表。首选的方法是列表中的第一个。推荐按密码学强度对算法进行排序，最强的第一。在[RFC3766]中给出了对此问题的一些额外的指引。

9.3.9. 通信量分析

对任何协议的被动侦听可能给攻击者有关会话、用户或协议指定信息的一些通过其他途径无法收集的信息。例如，已显示对 SSH 会话的通信量分析能产生有关密码长度的信息——[Openwall]和 [USENIX]。实现者应使用 SSH_MSG_IGNORE 数据包和包含随机长度的填充，以阻止通信量分析的企图。其他方法也可能被发现和实现。

9.4. 验证协议

本协议的目的是执行客户端用户验证。它假定运行在一个安全的传输层协议上，传输层协议已经验证了服务器、建立了加密的通信信道，并且为会话计算了一个唯一的会话标识。

允许几种具有不同安全特征的验证方法。对每个用户，服务器能接受哪种方法（或方法组合），由服务器的本地策略决定。验证强度不会高于所允许的最弱的组合。

服务器可在重复多次的不成功的验证尝试后进入一个睡眠期，使关键字检索对攻击者更加困难。应小心使这种机制不会导向自我拒绝服务（selfdenial of service）。

9.4.1. 弱连接（Weak Transport）

如果传输层不提供机密性，依赖秘密数据的验证方法应被禁用。如果传输层不提供强完整性保护，修改验证数据（例如，修改密码）的请求应被禁用，以防止攻击者在不被注意的情况下修改密文，或使新的验证数据不可用（拒绝服务）。

上述假定——验证协议仅在事先验证了服务器的安全的传输上运行——非常重要。提醒部署 SSH 的人员，如果客户端对于服务器和服务器主机密钥之间没有一个非常强的先验的（priori）关联，后果将导致中间人攻击。特别的，对验证协议的情况，客户端可能生成一个到中间人攻击设备的会话并泄漏用户凭证，如用户名和密码。即使在没有用户凭证被泄露的验证中，攻击者还可能通过捕获击键来获取不应有的信息，与蜜罐（honeypot）程序工作的方法基本相同。

9.4.2. 除错消息

在设计除错消息时应特别小心。如果设计得不合理，这些消息可能透露的有关主机的信息多得惊人。如果需要高安全性，可在用户验证阶段禁用除错消息。主机管理员应尝试各种方法来划分所有事件通知消息并保护它们不受未经授权的查看。开发人员应对一些正常事件和除错消息本质上的敏感性保持警惕，也可能希望为管理员提供如何防止未授权人员接触这些信息的指引。开发人员应考虑使用户在验证阶段能获得的敏感信息最小化，与本地策略一致。由于这个原因，推荐在部署时禁用除错消息，需要管理员主动进行启用。同时，在管理员启用除错信息时，推荐向其显示一条有关上述内容的消息。

9.4.3. 本地安全策略

实现者必须保证提供的凭证验证用户有效，而且必须保证服务器的本地策略允许该用户的访问请求。特别的，由于 SSH 连接协议的灵活性，在验证时可能无法决定应当应用哪些（如果有的话）本地安全策略，因为在这个时候用户所请求的服务还不确定。例如，本地策略可能允许用户访问服务器上的文件，但不允许启动交互式命令解释程序。但是，在验证协议中，并不知道用户将要访问文件、尝试使用交互式命令解释程序，或两者。在任何事件中，当服务器主机的本地安全策略存在时，它必须被正确的应用和强制实施。

实现者最好提供一个默认的本地策略，并将其参数告知管理员和用户。实现者可自行决定，默认的策略是顺着“什么都行（anything-goes）”路线，对用户不加任何限制，还是顺着“极端限制（excessively-restrictive）”路线，让管理员不得不主动对初始默认参数进行修改以满足他们的需要。或者，也可以尝试为管理员提供实用和立即可用的策略，使他们不必太费力就能让 SSH 运转起来。无论做出哪种选择，必须按上面的要求应用和强制实施。

9.4.4 公钥验证

使用公钥验证假定客户端主机没有失密。它也假定服务器主机的私钥没有失密。

该风险可以通过在私钥上使用口令句（passphrases）来降低；但是，这不是一个能强制执行的策略。建议使用智能卡或其他技术使口令句成为一个可强制执行的策略。

服务器可同时要求密码和公钥验证；但是，这要求客户端向服务器暴露它的密码（见下一节的密码验证）。

9.4.5. 密码验证

密码机制，像验证协议规定的那样，假定服务器没有失密。如果服务器失密，使用密码验证将会把有效的用户名/密码组合显示给攻击者，可能导致进一步的泄密。

该脆弱性可通过使用其他验证方式来减轻。例如，公钥验证并不假定服务器的安全性。

9.4.6. 基于主机的验证

基于主机的验证假定客户端没有失密。除了组合使用另一种验证方法外，没有降低风险的策略。

9.5. 连接协议

9.5.1. 末端安全性

连接协议假定了末端安全性。如果服务器失密，在该主机上的任何终端会话、端口转发或系统访问都会失密。对此没有降低风险的因素。

如果客户端已经失密，而且服务器在验证协议中未能阻止攻击者，所有暴露的服务（无论是子系统还是通过映射）对攻击都是脆弱的。实现者应提供机制让管理员可控制暴露哪些服务，以限制其他服务的脆弱性。上述控制可能包括控制哪个机器和端口能在端口转发中作为目标，哪些用户被允许使用交互式命令解释程序，或哪些用户被允许使用暴露的子系统。

9.5.2. 代理转发（Proxy Forwarding ）

SSH 连接协议允许对其他协议的代理转发，如 SMTP、POP3 和 HTTP。对于希望控制物理上外部的用户访问特定应用程序的网络管理员来所，这可能是一个需要考虑的问题。本质上，对这些协议的转发可能违反属于特定位置的（site-specific）安全策略，因为它们可能以探测不到的方式贯通防火墙。实现者应提供一种管理机制来控制代理转发功能，使属于特定位置的安全策略可能维持。

此外，可以使用反向代理转发功能，它也可以被用于绕开防火墙控制。

像上面指出的一样，在代理转发操作中假定了末端安全性。末端安全性失效将使所有经代理转发传递的数据失密。

9.5.3. X11 转发

SSH 连接协议提供的另一种形式的代理转发是对 X11 协议的转发。如果末端安全性失效，X11 转发可能允许对 X11 服务器的攻击。用户和管理员应，理所当然的，使用合理的 X11 安全机制来防止未经授权的使用 X11 服务器。希望进一步探索 X11 的安全机制的实现者、管理员和用户，可参阅 [SCHEIFLER] 和分析 CERT 之前报告的 SSH 转发与 X11 之间交互的问题 VU#363181 和 VU#118892 [CERT]。

使用 SSH 的 X11 远程显示（display forwarding），就其自身，不足以解决众所周知的 X11 安全性问题 [VENEMA]。但是，SSH（或其他安全协议）的 X11 远程显示，与只接受由许可或访问控制列表（ACL）授权的通过本地进程间通信（IPC）机制连接的实际和虚拟显示（actual and pseudo-display）组合，确实解决了很多 X11 安全性问题，只要没有使用"none"这种 MAC。推荐 X11 显示的系统实现默认只允许在本地 IPC 上开启显示。推荐支持 X11 转发的 SSH 服务器的系统实现默认只允许在本地 IPC 上开启显示。在单用户系统上，可能默认允许在 TCP/IP

上开启本地显示是合理的。

X11 转发协议的实现者应实现 [SSH-CONNECT] 中描述的防 cookie 欺骗机制，作为一种附加机制用于防止在未经授权的情况下使用代理。

10. 参考文献

（略）

SSH 传输层协议

摘要

SSH 是在不安全的网络上进行安全远程登录和其他安全网络服务的协议。本文讨论 SSH 传输层协议，它典型的运行在 TCP/IP 之上。协议可被用作多个安全网络服务的基础。它提供强有力的加密、服务器验证和完整性保护。它也可以提供压缩。密钥交互方法、公钥算法、对称加密算法、消息验证算法和哈希算法都通过协商。本文还描述了 Diffie-Hellman 密钥交换方法和实现 SSH 传输层协议所需的最小的算法集。

1．简介

SSH 传输层是一个安全的、底层的传输协议。它提供强有力的加密、服务器验证和完整性保护。

本协议的验证是基于主机的；本协议不执行用户验证。可以在本协议之上设计更高层的用于用户验证的协议。

协议被设计为简单和灵活的，允许参数协商，以及尽可能减少往返的次数。密钥交互方法、公钥算法、对称加密算法、消息验证算法和哈希算法都通过协商。预期在大多数情况下，完成整个密钥交换、服务器验证、服务请求和服务请求受理通知（acceptance notification）只需要 2 次往返。最坏的情况是 3 次往返。

2．撰稿者

（略）

3．本文中的惯用约定

所有与 SSH 协议相关的文档都应使用关键词“必须”、“禁止（绝不能）”、“必须的”、“应”、“不应”、“推荐”、“可（能）”、“可选的”来描述要求。对这些关键词的解释符合[RFC2119]的描述。

本文中用于描述命名空间分配的关键词"PRIVATE USE"、"HIERARCHICAL ALLOCATION"、"FIRST COME FIRST SERVED"、 "EXPERT REVIEW"、"SPECIFICATION REQUIRED"、"IESG APPROVAL"、"IETF CONSENSUS"以及"STANDARDS ACTION"的含义符合[RFC2434]的描述。

协议文档中定义了协议域和可能的取值。协议域将在消息定义中定义。例如，SSH_MSG_CHANNEL_DATA 的定义如下：

```
byte      SSH_MSG_CHANNEL_DATA
uint32    recipient channel
string    data
```

在整套协议文档中，当引用域时，域的名称出现在单引号中。当引用域的值时，它们出现在双引号中。例如，'data'的可能取值是"foo"和"bar"。

4．连接建立

SSH 在任何 8 位干净（8-bit clean）、二进制透明（binary-transparent）的传输。底层的传输应防止传输错误，因为这种错误将导致 SSH 连接断开。

由客户端初始化连接。

4.1．在 TCP/IP 上使用

当在 TCP/IP 上使用时，服务器一般在端口 22 监听连接。该端口号已经在 IANA 注册，已经被正式授予 SSH。

4.2．协议版本交换

当连接被建立后，双方都必须发送一个标识字串。该标识字串必须是

```
SSH-protoversion-softwareversion SP comments CR LF
```

由于在本文档中定义的协议是 2.0 版，'protoversion'必须是"2.0"。'comments'字符串时可选的。如果包含'comments'字串，必须有一个'space'字符（记作 SP，ASCII 32）分隔'softwareversion'字串和'comments'字串。标识字串必须以一个回车（CR，ASCII 13）和一个换行（LF，ASCII 10）终结。希望维持对本协议更旧的、无正式文件的版本的兼容性的实现者，可能出于本文第 5 节描述的原因，希望在对标识字串进行处理时不要求存在回车字符。禁止发送空字符。字符串的最大长度是 255 字符，包括回车和换行。

标识字串在回车和换行之前的部分被用于 Diffie-Hellman 密钥交换（见第 8 节）。

服务器可在发送版本字串前发送其他数据行。每个行应使用回车和换行终结。这样的行禁止以"SSH-"开头，而且应使用 ISO-10646 UTF-8 编码[RFC3629]（未指定语言）。客户端必须能够处理这样的行。这样的行可被静默的忽略，或可被显示给客户端用户。如果它们被显示，应使用在[SSH-ARCH]中讨论的控制字符筛选。本功能的主要作用是允许 TCP 包装器（TCPwrappers）在断开前显示错误消息。

'protoversion'和'softwareversion'字串必须包含可打印的 US-ASCII 字符，不包括空格字符和减号（-）。'softwareversion'字串主要用于触发兼容性扩展和标示系统实现的能力。'comments'字串应包含对解决用户的问题可能有帮助的附加信息。像这样，一个有效的标识字串的例子是

```
SSH-2.0-billsSSH_3.6.3q3<CR><LF>
```

这个标识字串不包含可选的'comments'字串，因此在'softwareversion'字串后立即用 CR 和 LF 终结。

密钥交换将在发送标识字串后立即开始。所有在标识字串之后的数据包应使用第 6 节所描述的二进制数据包协议。

5．与 SSH 旧版本的兼容性

（略）

6．二进制数据包协议

每个数据包为以下格式：

```
uint32    packet_length
byte      padding_length
byte[n1]  payload; n1 = packet_length - padding_length - 1
byte[n2]  random padding; n2 = padding_length
byte[m]   mac (Message Authentication Code - MAC); m = mac_length
```

packet_length

以字节为单位的数据包长度，不包括'mac'或'packet_length'域自身。

padding_length

'random padding'的长度（字节）。

payload

数据包中有用的内容。如果已经协商了压缩，该域是压缩的。初始时，压缩必须为"none"。

random padding

任意长度的填充，使(packet_length || padding_length || payload || random padding)的总长度是加密分组长度或 8 中较大者的倍数。最少必须有 4 字节的填充。填充应包含随机字节。填充的最大长度为 255 字节。

mac

消息验证码。如果已经协商了消息验证，该域包含 MAC。初始时，MAC 算法必须是"none"。

注意，(packet_length || padding_length || payload || random padding)的长度必须是加密分块长度或 8 中较大者的倍数。这一约束必须被强制实施，即使当使用流式加密时也是如此。注意'packet_length'域也是加密的，在发送和接收数据包时，对它的处理要特别当心。同时指出，插入长度变化的'random padding'可帮助阻止通信流量分析攻击。

一个数据包最小的长度是 16（或加密分块长度，两者中较大的一个）字节（加'mac'）。系统实现应在接收数据包的前 8（或加密分块长度，两者中较大的一个）字节后解密数据包的长度。

6.1．最大数据包长度

所有系统实现必须能够处理带有 32768 字节或以下的非压缩有效载荷（payload）、总长度 35000 字节或以下（包括'packet_length'、'padding_length'、'payload'、'random padding'和'mac'）的数据包。最大长度 35000 字节是一个任意选定的值，大于上述非压缩有效载荷的长度。系统实现当需要时应支持更大的数据包。例如，如果一个系统实现希望发送数

量很多的证书，在标识字串表明对方能处理时，可以向对方发送更大的数据包。但是，系统实现应检查数据包长度是否合理，以避免拒绝服务及/或缓冲溢出攻击。

6.2. 压缩

如果协商了压缩，有效载荷（并且仅有效载荷）将使用协商的算法进行压缩。'packet_length'域和'mac'将根据压缩后的有效载荷计算产生。加密将在压缩之后进行。

根据压缩方法，压缩可能是有状态的（stateful）。两个方向的压缩必须是独立的，而且系统实现必须允许独立选择两个方向上的压缩算法。但是在实际使用中，推荐在两个方向上使用相同的压缩方法。

当前定义了下列压缩算法：

none	REQUIRED	无压缩
zlib	OPTIONAL	ZLIB（LZ77）压缩

"zlib"压缩在[RFC1950]和[RFC1951]中描述。压缩的上下文在每次密钥交换后初始化，并且从一个数据包传递到下一个数据包，在每个数据包的结尾只执行一个部分刷新（partial flush）。部分刷新的含义是结束当前压缩块并且输出所有数据。如果当前块不是一个存储块（stored block），将在当前块之后加上一个或多个空块，以保证从当前块的块结束代码到数据包的有效载荷结束至少有 8 比特。

可按[SSH-ARCH]和[SSH-NUMBERS]的规定定义更多的方法。

6.3. 加密

在密钥交互中将协商出一种加密算法和一个密钥。当加密生效时，每个数据包的数据包长度、填充长度、有效载荷和填充域必须使用给定的算法加密。

所有从一个方向发送的数据包中的加密数据应被认为是一个数据流。例如，初始向量应从一个数据包的结束传递到下一个数据包的开始。所有加密器应使用有效密钥长度为 128 位或以上的密钥。

两个方向上的加密器必须独立运行。如果本地策略允许多种算法，系统实现必须允许独立选择每个方向上的算法。但是，在实际使用中，推荐在两个方向上使用相同的算法。

当前定义了下列加密器：

3des-cbc	REQUIRED	三密钥 3DES，CBC 模式
blowfish-cbc	OPTIONAL	Blowfish，CBC 模式
twofish256-cbc	OPTIONAL	Twofish，CBC 模式，256 位密钥
twofish-cbc	OPTIONAL	"twofish256-cbc"的别名（历史原因保留）
twofish192-cbc	OPTIONAL	Twofish，192 位密钥
twofish128-cbc	OPTIONAL	Twofish，128 位密钥
aes256-cbc	OPTIONAL	AES，CBC 模式，256 位密钥

aes192-cbc	OPTIONAL	AES，192 位密钥
aes128-cbc	RECOMMENDED	AES，128 位密钥
serpent256-cbc	OPTIONAL	Serpent，CBC 模式，256 位密钥
serpent192-cbc	OPTIONAL	Serpent，192 位密钥
serpent128-cbc	OPTIONAL	Serpent，128 位密钥
arcfour	OPTIONAL	ARCFOUR 流式加密器，128 位密钥
idea-cbc	OPTIONAL	IDEA，CBC 模式
cast128-cbc	OPTIONAL	CAST-128，CBC 模式
none	OPTIONAL	无加密；不推荐

"3des-cbc"是三密钥的三重 DES（加密-解密-加密）加密器，密钥的前 8 字节用于第一次加密，接下来的 8 字节用于解密，再接下来的 8 字节用于最终加密。这要求 24 字节的密钥数据（实际使用了 168 比特）。为了实现 CBC 模式，必须使用外链（outer chaining）（即，只有一个初始向量）。这是一个使用 8 字节分块大小的分块加密器。算法的定义见[FIPS-46-3]。要注意由于该算法的有效密钥长度只有 112 位（[SCHNEIER]），它不满足 SSH 加密算法应使用 128 位或以上密钥的要求。但是，由于历史原因该算法仍然是必须的；基本上，在撰写本文时，所有已知的系统实现都支持这种算法，而且由于它是一种基本的、可互操作的算法，因此被广泛使用。预期将来另一种更强的算法会变得极为普及以至于"3des-cbc"将最终在另一次 STANDARDS ACTION 被弃用。

"arcfour"是使用 128 位密钥的 Arcfour 流式加密器。Arcfour 加密器被认为与 RC4 加密器兼容[SCHNEIER]。Arcfour（以及 RC4）对弱密钥有问题，在使用时应谨慎。

"none"算法规定不进行加密。注意这种方法不提供机密性保护，是不推荐的。当选择了这种加密器时，由于安全性原因，一些功能（例如，密码验证）可能被禁用。

可按[SSH-ARCH]和[SSH-NUMBERS]的规定定义更多的方法。

6.4. 数据完整性

每个数据包包含一个从共享秘密、数据包序号和数据包内容计算出来的 MAC 以对数据完整性进行保护。

在密钥交互时协商消息验证算法和密钥。初始时，没有 MAC 生效，它的长度必须是零。在密钥交换后，所选 MAC 算法的'mac'将在加密前从拼接的数据包数据中计算得到：

```
mac = MAC(key, sequence_number || unencrypted_packet)
```

unencrypted_packet 是除'mac'之外的完整的数据包（长度、'payload'和'random padding'），sequence_number 是一个 unit32 型的隐式的数据包序号。对第一个数据包，sequence_number 被初始化为零，并且在每个数据包（无论是否使用了加密或 MAC）之后增加。它不会被重置，即使后来重新协商了密钥/算法。在每 2^32 个数据包后，它到达最大之后又从零开始。数据包的 sequence_number 自身不包含在通过线路发送的数据包中。

每个方向上的 MAC 算法必须独立运行，而且系统实现必须允许独立选择每个方向上的算法。但

是，在实际使用中，推荐两个方向使用相同的算法。

MAC 算法生成的'mac'值必须作为数据包的最后一部分不加密的进行传输。'mac'的字节数由所选的算法决定。

当前定义了下列 MAC 算法：

hmac-sha1	REQUIRED	HMAC-SHA1（摘要长度=密钥长度=20）
hmac-sha1-96	RECOMMENDED	HMAC-SHA1 的前 96 比特（摘要长度=12，密钥长度=20）
hmac-md5	OPTIONAL	HMAC-MD5（摘要长度=密钥长度=16）
hmac-md5-96	OPTIONAL	HMAC-MD5 的前 96 比特（摘要长度=12，密钥长度=16）
none	OPTIONAL	无 MAC；不推荐

[RFC2104]描述了这些"hmac-*"算法。"*-n"的 MAC 只使用生成值的前 n 个比特。

[FIPS-180-2]描述了 SHA-1，[RFC1321]描述了 MD5。

可按[SSH-ARCH]和[SSH-NUMBERS]的规定定义更多的方法。

6.5. 密钥交换方法

密钥交换方法规定如何生成用于加密和验证的一次性会话密钥，以及如何进行服务器验证。

定义了两个必须的密钥交换方法：

diffie-hellman-group1-sha1	REQUIRED
diffie-hellman-group14-sha1	REQUIRED

这些方法在第 8 节中描述。

可按[SSH-NUMBERS]的规定定义更多的方法。名称"diffie-hellman-group1-sha1"代表一种使用 Oakley 组的密钥交换方法，见[RFC2409]中的定义。SSH 维持自己的组标识空间，逻辑上与 Oakley[RFC2412]和 IKE 是完全分开的；但是，工作组沿用了[RFC3526]分配的编号，使用"diffiehellman-group14-sha1"作为第二个组的名称。系统实现应把这些名称仅仅当作标识，而不应假定 SSH 使用的组和 IKE 定义的组之间存在关联。

6.6. 公钥算法

本协议设计为可使用几乎任何公钥格式、编码和算法（签名及/或加密）。

几个方面定义了一个公钥的类型：

- 密钥格式。密钥是如何编码的，以及证书是如何表现的。本协议的密钥块（key blob）除密钥外还可包含证书。

- 签名及/或加密算法。一些密钥类型可能不能同时支持签名和加密。密钥使用也可能受到策略声明的限制（例如，在证书中）。在这种情况下，对不同的策略选择应定义不同的密钥类型。

- 签名及/或加密数据的编码。包括但不限于填充、字节顺序和数据格式。

当前定义了下列公钥及/或证书格式：

ssh-dss	REQUIRED	对 Raw DSS Key 签名
ssh-rsa	RECOMMENDED	对 Raw RSA Key 签名
pgp-sign-rsa	OPTIONAL	对 OpenPGP 证书签名（RSA 密钥）
pgp-sign-dss	OPTIONAL	对 OpenPGP 证书签名（DSS 密钥）

可按[SSH-ARCH]和[SSH-NUMBERS]的规定定义更多的密钥类型。

密钥类型必须总是被显式的知悉（从算法协商或其它来源）。它一般不包含在密钥块中。

证书和公钥的编码如下：

```
string     certificate or public key format identifier
byte[n]    key/certificate data
```

证书部分可能是一个长度为零的字符串，但公钥是必须的。这是将被用于验证的公钥。证书块（certificate blob）中的证书序列（certificate sequence）可被用于提供授权。

没有显式的指定一种签名格式标识符的公钥/证书格式必须使用公钥/证书格式标识符作为签名标识符。

签名的编码如下：

```
string     signature format identifier （由公钥/证书格式规定）
byte[n]    signature blob in format specific encoding
```

"ssh-dss"密钥格式的编码规定如下：

```
string     "ssh-dss"
mpint      p
mpint      q
mpint      g
mpint      y
```

这里，参数'p'、'q'、'g'和'y'组成签名密钥块。

使用这种密钥格式进行签名和验证符合数字签名标准[FIPS-186-2]，使用 SHA-1 哈希[FIPS-180-2]。

生成的签名的编码如下：

```
string    "ssh-dss"
string    dss_signature_blob
```

'dss_signature_blob'的值被编码为包含 r 和其后的 s（是 160 比特、无长度或填充、无符号、按网络字节顺序的整数）的字符串。

"ssh-rsa"密钥格式的编码规定如下：

```
string    "ssh-rsa"
mpint     e
mpint     n
```

这里，参数'e'和'n'组成签名密钥块。

使用这种密钥格式进行签名和验证符合 RSASSA-PKCS1-v1_5 模式[RFC3447]，使用 SHA-1 哈希。

生成的签名的编码如下：

```
string    "ssh-rsa"
string    rsa_signature_blob
```

'rsa_signature_blob'的值被编码为包含 s（是 160 比特、无长度或填充、无符号、按网络字节顺序的整数）的字符串。

"pgp-sign-rsa"方法表明证书、公钥和签名采用与 OpenPGP 兼容的二进制格式（[RFC2440]）。这种方法表明密钥是 RSA 密钥。

"pgp-sign-dss"同上，但表明密钥是 DSS 密钥。

7．密钥交换

密钥交换（kex）从每一方发送支持的算法名称列表开始。每一方对每个类型有一个首选算法，并且假定大多数系统实现在任何时候都会使用相同的首选算法。每一方可猜测另一方正在使用哪种算法，如果与首选方法相符，也可按照该算法来发送初始密钥交换数据包。

上述猜测被认为不正确，如果：

- kex 算法及/或主机密钥算法猜测错误（服务器和客户端首选的算法不同），或

o 如果任何其他算法未达成一致（该过程在 7.1 节中定义）。

否则，猜测被认为正确，而且上述根据猜测发送的数据包必须被作为首个密钥交换数据包处理。

但是，如果猜测不正确，而且一方或双发已经根据猜测发送了数据包，这些数据包必须被忽略（即使猜测中的错误将不影响初始数据包的内容），并且有关方必须发送正确的初始数据包。

如果密钥交换消息包含一个签名或其他对服务器真实性的证明，密钥交换方法将使用显式的服务器验证。如果为了证明其真实性，服务器也必须通过发送一条消息和对应的、客户端能验证的 MAC 以证明它知道共享秘密 K，那么密钥交换方法将使用隐式服务器验证。

本文定义的密钥交换方法使用显式服务器验证。但是，本协议可使用隐式服务器验证的密钥交换方法。在使用隐式服务器验证的密钥交换之后，客户端在发送任何数据前，必须等待对其服务请求的响应。

7.1. 算法协商

密钥交换从每一方发送如下数据包开始：

```
byte          SSH_MSG_KEXINIT
byte[16]      cookie (random bytes)
name-list     kex_algorithms
name-list     server_host_key_algorithms
name-list     encryption_algorithms_client_to_server
name-list     encryption_algorithms_server_to_client
name-list     mac_algorithms_client_to_server
name-list     mac_algorithms_server_to_client
name-list     compression_algorithms_client_to_server
name-list     compression_algorithms_server_to_client
name-list     languages_client_to_server
name-list     languages_server_to_client
boolean       first_kex_packet_follows
uint32        0 （为将来扩展预留）
```

每一个算法名称列表必须是一个逗号分隔的算法名称的列表（见[SSH-ARCH]的算法命名和[SSH-NUMBERS]中的附加信息）。每一种支持的（允许的）算法必须按优先顺序排列，从优先级最高到优先级最低。

每个名称列表中的第一个算法必须是首选的（猜测的）算法。每个名称列表必须包含至少一个算法名称。

cookie

'cookie'必须是一个由发送方生成的随机值。它的作用是使任何一方都不可能对密钥和会话标识符拥有完全决定权。

kex_algorithms

密钥交换算法在之前定义。第一个算法必须是首选的（及猜测的）算法。如果两方作出同样的猜测，这种算法必须被使用。否则，必须使用以下算法来选择一个密钥交换方法：逐一遍历客户端的 kex 算法，选择满足下列条件的第一种算法：

- 服务器也支持该算法，
- 如果该算法需要一个能用于加密的主机密钥，服务器的 server_host_key_algorithms 中存在一种能用于加密的算法，且客户端也支持该算法，并且
- 如果该算法需要一个能用于签名的主机密钥，服务器的 server_host_key_algorithms 中存在一种能用于加密的算法，且客户端也支持该算法。

如果找不到一种满足上述所有条件的算法，连接即失败，双方必须断开连接。

server_host_key_algorithms

受支持的为服务器主机密钥服务的算法的名称列表。服务器列出其拥有主机密钥的那些算法；客户端列出其原意接受的算法。一个主机可有多个主机密钥，可能对应于不同的算法。

一些主机密钥可能不同时支持签名和加密（可由算法判定），因此，不是所有主机密钥都能使用所有密钥交换方法。

算法选择依赖于选定的密钥交换算法是否需要一个能用于签名或能用于加密的主机密钥。必须能够从公钥算法名称上对此进行判断。必须选择客户端的名称列表中的第一个满足要求且受服务器支持的算法。如果不存在满足条件的算法，双方必须断开连接。

encryption_algorithms

可接受的对称加密算法（也称为加密器）的名称列表，按优先级排序。每个方向上选定的加密算法必须是客户端的名称列表中的第一个同时也在服务器名称列表中的算法。如果不存在满足条件的算法，双方必须断开连接。

注意，如果希望接受"none"，则它必须被显式的列出。已定义的算法名称在 6.3 节中列出。

mac_algorithms

可接受的 MAC 算法的名称列表，按优先级排序。选定的 MAC 算法必须是客户端名称列表中的第一个同时也在服务器名称列表中的算法。如果不存在满足条件的算法，双方必须断开连接。

注意，如果希望接受"none"，则它必须被显式的列出。MAC 算法名称在 6.4 节中列出。

compression_algorithms

可接受的压缩算法的名称列表，按优先级排序。选定的压缩算法必须是客户端名称列表中

的第一个同时也在服务器名称列表中的算法。如果不存在满足条件的算法，双方必须断开连接。

注意，如果希望接受"none"，则它必须被显式的列出。压缩算法名称在 6.2 节中列出。

languages

语言标志的名称列表，按优先级排序[RFC3066]。双方均可忽略该名称列表。如果没有语言优先级，该名称列表应为空，像在[SSH-ARCH]的第 5 节中定义的那样。发送方除非确信需要语言标志，否则不应提供。

first_kex_packet_follows

表明是否有一个猜测的密钥交换数据包跟随。如果将会发送一个猜测的数据包，该域必须是 TRUE。如果不会发送猜测的数据包，该域必须是 FALSE。

在从另一方接收 SSH_MSG_KEXINIT 数据包后，每一方将知道另一方的猜测是否正确。如果另一方的猜测不正确，且该域为 TRUE，下一个数据包必须被静默的忽略，并且接下来双方必须根据协商的密钥交换方法来行动。如果猜测正确，必须使用猜测的数据包继续进行密钥交换。

在 SSH_MSG_KEXINIT 消息交换后，密钥交换算法开始运行。根据密钥交换算法的规定，它可能包含几个数据包交换。

当一方为密钥交换或重新交换发送了一个 SSH_MSG_KEXINIT 消息，直到其完成发送一个 SSH_MSG_NEWKEYS 消息（见 7.3 节），该方禁止发送除下列以外的任何消息：

- 传输层通用消息（1 到 19）（但禁止发送 SSH_MSG_SERVICE_REQUEST 和 SSH_MSG_SERVICE_ACCEPT）；
- 算法协商消息（20 到 29）（但禁止发送更多的 SSH_MSG_KEXINIT 消息）；
- 规定密钥交换方法消息（30 到 49）。

第 11 节的规定适用于未被识别的消息。

但是要注意，在一个密钥交换过程中，在发送一个 SSH_MSG_KEXINIT 消息后，至收到另一方的 SSH_MSG_KEXINIT 之前，每一方必须准备处理任意数量的可能正在传输中的消息。

7.2. 密钥交换的输出

密钥交换产生两个值：一个共享秘密 K，以及一个交换哈希 H。加密和验证密钥来自它们。第一次密钥交换的交换哈希 H 也被用作会话标识，它是一个对该连接的唯一标识。它是验证方法中被签名（以证明拥有私钥）的数据的一部分。会话标识被计算出来后，即使后来重新交换了密钥，也不会改变。

每种密钥交换方法规定了一个用于密钥交换的哈希函数。生成密钥必须使用同样的哈希算法。在这里，我们将其称为 HASH。

加密密钥必须是对一个已知值和 K 的 HASH 结果，方法如下：

- 客户端到服务器的初始 IV：HASH(K || H || "A" || session_id)（这里 K 为 mpint 格式，"A"为 byte 格式，session_id 为原始数据（raw data）。"A"是单个字母 A，ASCII 65）。
- 服务器到客户端的初始 IV：HASH(K || H || "B" || session_id)
- 客户端到服务器的加密密钥：HASH(K || H || "C" || session_id)
- 服务器到客户端的加密密钥：HASH(K || H || "D" || session_id)
- 客户端到服务器的完整性密钥：HASH(K || H || "E" || session_id)
- 服务器到客户端的完整性密钥：HASH(K || H || "F" || session_id)

密钥数据必须从哈希输出的开头开始取。即从哈希值的开头开始，取所需数量的字节。如果需要的密钥长度超过 HASH 输出，则拼接 K、H 和当前的整个密钥并计算其 HASH，然后将 HASH 产生的字节附加到密钥尾部。重复该过程，直到获得了足够的密钥材料；密钥从该值的开头开始取。换句话说：

```
K1 = HASH(K || H || X || session_id)（X 表示"A"等）
K2 = HASH(K || H || K1)
K3 = HASH(K || H || K1 || K2)
...
key = K1 || K2 || K3 || ...
```

如果 K 的熵比 HASH 的内状态（internal state）大小要大，则该过程将造成熵的丢失。

7.3. 密钥的交付使用

密钥交换在每一方发送一个 SSH_MSG_NEWKEYS 消息后结束。该消息使用旧的密钥和算法发送。所有在该消息之后发送的消息必须使用新的密钥和算法。

当接收完该消息后，必须使用新密钥和算法进行接收。

该消息的作用是保证当密钥交换出错时，一方能够以另一方可理解的 SSH_MSG_DISCONNECT 消息进行响应。

```
byte      SSH_MSG_NEWKEYS
```

8. Diffie-Hellman 密钥交换

Diffie-Hellman（DH）密钥交换提供了一个不能由任何一方单独决定的共享秘密。密钥交换与一个使用主机密钥的签名结合，以提供主机验证。该密钥交换方法提供第 7 节定义的显式服

务器验证。

交换一个密钥使用下列步骤。在这里，C 是客户端；S 是服务器；p 是一个大的安全素数（safe prime）；g 是素域 GF(p)的一个子群（subgroup）的发生器；q 是子群的阶（order）；V_S 是 S 的识别字串；V_C 是 C 的识别字串；K_S 是 S 的公钥；I_C 是在这部分开始之前交换的 C 的 SSH_MSG_KEXINIT 消息；I_S 是在这部分开始之前交换的 S 的 SSH_MSG_KEXINIT 消息。

1. C 生成一个随机数 x（1 < x < q）并计算 e = g^x mod p。C 把 e 发送给 S。

2. S 生成一个随机数 y（0 < y < q）并计算 f = g^y mod p。S 接收到 e。S 计算 K、H 和 s。K = e^y mod p，H = hash(V_C || V_S || I_C || I_S || K_S || e || f || K)（这些元素根据他们的类型编码；见下），s 是 S 使用它的私钥对 H 的签名。S 把(K_S || f || s)发送给 C。签名操作可能包含第二个哈希运算。

3. C 验证 K_S 确实是 S 的主机密钥（例如，使用证书或本地数据库）。C 也可以不加验证的接受 K_S；但是，这样做将使协议对主动攻击不安全（但在很多环境下短期内可能因为实际的考量是可取的）。C 然后计算 K = f^x mod p，H = hash(V_C || V_S || I_C || I_S || K_S || e || f || K)，并验证对 H 的签名 s。任何一方都禁止发送或接受不在[1, p-1]范围之内的'e'或'f'值。如果违反了该条件，密钥交换失败。

这些通过下列消息实现。计算交换哈希使用的哈希算法由方法名称定义，称为 HASH。用于签名的公钥算法通过 SSH_MSG_KEXINIT 消息协商。

首先，客户端发送：

```
byte      SSH_MSG_KEXDH_INIT
mpint     e
```

服务器响应如下：

```
byte      SSH_MSG_KEXDH_REPLY
string    K_S，服务器公钥和证书 (
mpint     f
string    s，对 H 的签名
```

哈希值 H 是对下列内容拼接进行 HASH 的结果：

```
string    V_C，客户端识别字串（不包括 CR 和 LF）
string    V_S，服务器识别字串（不包括 CR 和 LF）
string    I_C，客户端 SSH_MSG_KEXINIT 的有效载荷
string    I_S，服务器 SSH_MSG_KEXINIT 的有效载荷
string    K_S，主机密钥
mpint     e，客户端发送的交换值
```

mpint f，服务器发送的交换值
mpint K，共享秘密

该值称为交换哈希（exchange hash），它用于验证密钥交换。交换哈希应保密。

签名算法必须应用于 H，而不是原始数据。大多数签名算法包含哈希运算和附加填充（例如，"ssh-dss"规定 SHA-1 哈希运算）。在这种情况下，数据首先经过 HASH 计算得到 H，然后在签名过程中再对 H 使用 SHA-1 进行了哈希运算。

8.1. diffie-hellman-group1-sha1

"diffie-hellman-group1-sha1"方法规定使用 SHA-1 为 HASH 和 Oakley Group 2 [RFC2409]（1024 bit MODP Group）的 Diffie-Hellman 密钥交换。所有已知的系统实现当前都支持该方法，因此必须支持该方法以保证互操作性。注意该方法使用"group1"为名称，尽管它规定使用 Oakley Group 2。

8.2. diffie-hellman-group14-sha1

"diffie-hellman-group14-sha1"方法规定使用 SHA-1 为 HASH 和 Oakley Group 14 [RFC3526]（2048 bit MODP Group）的 Diffie-Hellman 密钥交换，它也必须被支持。

9．密钥重新交换

除正在进行密钥交换以外的时刻（见 7.1 节的描述），发送一个 SSH_MSG_KEXINIT 数据包将开始密钥重新交换。当一方接收到该消息时，必须用自己的 SSH_MSG_KEXINIT 消息进行响应，除非接收到的 SSH_MSG_KEXINIT 消息已经是一个响应。任何一方均可发起重新交换，但禁止改变角色（即，服务器仍是服务器，客户端仍是客户端）。

密钥重新交换使用交换开始时正在使用的加密。加密、压缩和 MAC 方法在密钥交换结束发送新的 SSH_MSG_NEWKEYS 前不改变（与最初的密钥交换相同）。除会话标识保留不变外，重新交换的处理与最初的密钥交换相同。可允许在重新交换中改变一些或全部算法。也可改变主机密钥。所有密钥和初始向量在交换后重新计算。压缩和加密的上下文被重置。

推荐在传输 1G 字节数据或连接一小时（两者中较早到达的一个）后改变密钥。但是，由于重新交换是一个需要较大的处理能力的公钥操作，它不应被过于频繁执行。

在 SSH_MSG_NEWKEYS 数据包之后可以继续发送应用数据；密钥交换不影响位于 SSH 传输层之上的协议。

10．服务请求

在密钥交换之后，客户端请求一个服务。服务由其名称标识。[SSH-ARCH]和[SSH-NUMBERS]中定义了名称的格式和定义新名称的过程。

当前，以下名称被保留：

ssh-userauth
ssh-connection

对服务名称的本地命名策略与对算法的命名策略相似。本地服务应使用 PRIVATE USE 语法，即"servicename@domain"。

```
byte       SSH_MSG_SERVICE_REQUEST
string     service name
```

如果服务器拒绝服务请求，它应发送一个适当的 SSH_MSG_DISCONNECT 消息且必须断开连接。

当服务开始，它可访问在密钥交换过程中生成的会话标识。

如果服务器支持该服务（并且允许客户端使用它），它必须发送如下响应：

```
byte       SSH_MSG_SERVICE_ACCEPT
string     service name
```

服务使用的消息编号应在为其保留的范围内（见[SSH-ARCH]和[SSH-NUMBERS]）。传输层将继续处理它自己的消息。

注意，在使用隐式服务其验证的密钥交换后，客户端在发送其他数据前，必须等待一个对其服务请求消息的响应。

11. 附加消息

任何一方可在任何时候发送下列任何消息。

11.1. 断开连接消息

```
byte       SSH_MSG_DISCONNECT
uint32     reason code
string     description，采用 ISO-10646 UTF-8 编码[RFC3629]
string     language tag，[RFC3066]
```

该消息导致连接的立即终止。所有系统实现必须能够处理该消息；并且应能够发送该消息。

发送方在该消息之后禁止发送或接收任何数据，而且接收方在接收该消息后禁止接受任何数据。断开连接消息的'description'字串以人可读的形式给出一个更加具体的解释。断开连接消息的'reason code'以更加机器可读的形式给出原因（适合本地化），其可能的取值见下表。注意表中使用十进制表示法以提高可读性，实际上值为 uint32 格式。

Symbolic name	reason code
SSH_DISCONNECT_HOST_NOT_ALLOWED_TO_CONNECT	1
SSH_DISCONNECT_PROTOCOL_ERROR	2
SSH_DISCONNECT_KEY_EXCHANGE_FAILED	3
SSH_DISCONNECT_RESERVED	4

SSH_DISCONNECT_MAC_ERROR	5
SSH_DISCONNECT_COMPRESSION_ERROR	6
SSH_DISCONNECT_SERVICE_NOT_AVAILABLE	7
SSH_DISCONNECT_PROTOCOL_VERSION_NOT_SUPPORTED	8
SSH_DISCONNECT_HOST_KEY_NOT_VERIFIABLE	9
SSH_DISCONNECT_CONNECTION_LOST	10
SSH_DISCONNECT_BY_APPLICATION	11
SSH_DISCONNECT_TOO_MANY_CONNECTIONS	12
SSH_DISCONNECT_AUTH_CANCELLED_BY_USER	13
SSH_DISCONNECT_NO_MORE_AUTH_METHODS_AVAILABLE	14
SSH_DISCONNECT_ILLEGAL_USER_NAME	15

如果显示'description'字串，应使用[SSH-ARCH]中讨论的控制字符筛选来避免发送终端控制字符的攻击。

分配断开连接消息的新的在 0x00000010 至 0xFDFFFFFF 范围内的'reason code'值（以及相关的'description'文本）必须通过[RFC2434]描述的 IETF CONSENSUS 方法。在 0xFE000000 至 0xFFFFFFFF 范围内的断开连接'reason code'为 PRIVATE USE 保留，对 IANA 的实际指引见[SSH-NUMBERS]。

11.2. 被忽略的数据消息

```
byte      SSH_MSG_IGNORE
string    data
```

所有系统实现在任何时间（在接收识别字串后）必须理解（并忽略）该消息。不要求系统实现发送这些消息。该消息可用作对高级流量分析技巧的一种附加保护措施。

11.3. 除错消息

```
byte      SSH_MSG_DEBUG
boolean   always_display
string    message，采用 ISO-10646 UTF-8 编码[RFC3629]
string    language tag，[RFC3066]
```

所有系统实现必须理解该消息，但允许忽略它。该消息用于传输可能帮助除错的信息。如果'always_display'为 TRUE，消息应被显示。否则，它不应被显示，除非用户显式的请求除错信息。

'message'不需要包括一个新行。它允许包含多个用 CRLF（回车换行）分隔的行。

如果显示'message'字串，应使用[SSH-ARCH]中讨论的控制字符筛选来避免发送终端控制字符的攻击。

11.4. 保留消息

系统实现必须使用 SSH_MSG_UNIMPLEMENTED 消息按接收顺序对所有未识别的消息进行相应。

除此以外，这些消息必须被忽略。今后的协议版本可能为这些消息类型定义其他意义。

```
byte      SSH_MSG_UNIMPLEMENTED
uint32    packet sequence number，被拒绝的消息的数据包序号
```

12．消息编号总结

以下是对消息及其编号的总结。

```
SSH_MSG_DISCONNECT          1
SSH_MSG_IGNORE              2
SSH_MSG_UNIMPLEMENTED       3
SSH_MSG_DEBUG               4
SSH_MSG_SERVICE_REQUEST     5
SSH_MSG_SERVICE_ACCEPT      6
SSH_MSG_KEXINIT             20
SSH_MSG_NEWKEYS             21
```

注意，编号 30-49 被用于 kex 数据包。不同的 kex 方法可重用这一范围内的消息编号。

13．IANA 考虑

[SSH-ARCH]、[SSH-TRANS]、[SSH-USERAUTH]和[SSH-CONNECT]定义的 SSH 协议的 IANA 考虑详见[SSH-NUMBERS]。

14．安全性考虑

本协议在一个不安全的网络上提供一个安全的加密信道。它执行服务器主机验证、密钥交换、加密和完整性保护。它也产生一个可由高层协议使用的唯一的会话 ID。

对本协议的全面的安全性考虑见[SSH-ARCH]。

15．参考文献

（略）

SSH 验证协议

摘要

SSH 是在一个不安全的网络上实现安全的远程登录和其他安全的网络服务的协议。本文描述 SSH 验证协议框架以及公钥、密码和基于主机的客户端验证方法。附加的验证方法在独立的文档中描述。SSH 验证协议运行在 SSH 传输层协议之上，而且为 SSH 连接协议提供了一个单一的已验证的隧道。

1．简介

SSH 验证协议是一个通用的用户验证协议。它意图运行在 SSH 传输层协议[SSH-TRANS]之上。本协议假定下层协议提供了完整性和机密性保护。

读者应首先阅读 SSH 架构文档[SSH-ARCH]之后再阅读本文。本文在使用架构文档中的术语和标记法时未加引用或解释。

本协议的'service name'为"ssh-userauth"。

当本协议启动时，它从下层协议接收会话标识（它是首次密钥交换的交换哈希 H）。会话标识唯一的标识会话，而且适合在证明拥有私钥的过程中作为签名的对象。本协议也需要知道下层协议是否提供了机密性保护。

2．撰写者

（略）

3．本文中的惯用约定

所有与 SSH 协议相关的文档都应使用关键词“必须”、“禁止（绝不能）”、“必须的”、“应”、“不应”、“推荐”、“可（能）”、“可选的”来描述要求。这些关键词的含义符合[RFC2119]的描述。

本文中用于描述命名空间分配的关键词"PRIVATE USE"、"HIERARCHICAL ALLOCATION"、"FIRST COME FIRST SERVED"、 "EXPERT REVIEW"、"SPECIFICATION REQUIRED"、"IESG APPROVAL"、"IETF CONSENSUS"以及"STANDARDS ACTION"的含义符合[RFC2434]的描述。

协议文档中定义了协议域和可能的取值。协议域将在消息定义中定义。例如，SSH_MSG_CHANNEL_DATA 的定义如下：

```
byte      SSH_MSG_CHANNEL_DATA
uint32    recipient channel
string    data
```

在整套协议文档中，当引用域时，域的名称出现在单引号中。当引用域的值时，它们出现在双引

号中。例如，'data'的可能取值是"foo"和"bar"。

4．验证协议框架

服务器通过在任意时间告知客户端为继续交换信息可使用哪些验证方法来驱动验证过程。客户端可按任意顺序尝试服务器列出的方法。这使得服务器当希望时能够完全控制验证过程，但当服务器提供了多种方法时也给了客户端提供了足够的灵活性，让客户端能使用自己支持的或者对用户而言最方便的方法。

验证方法由它们的名称标识，见[SSH-ARCH]。"none"方法被保留，而且禁止被列为受支持的方法。尽管如此，客户端可发送"none"方法。如果要在未经验证的情况下授权客户端访问，服务器必须接受该请求，否则服务器必须总是拒绝该请求。发送该请求的主要作用是获取服务器支持的方法的列表。

服务器应对验证设置超时，而且如果验证请求没有在超时时间内通过应断开连接。推荐的超时期间是 10 分钟。此外，系统实现应限制一个客户端在一个会话中尝试验证并失败的次数（推荐的限制是 20 次）。如果超过了阀值，服务器应断开连接。

更多的关于验证超时和重试的考虑可参见[ssh-1.2.30]。

5．验证请求

所有验证请求必须使用以下消息格式。只定义了前几个域；其余的域依赖于验证方法。

```
byte      SSH_MSG_USERAUTH_REQUEST
string    user name，采用 ISO-10646 UTF-8 编码[RFC3629]
string    service name，采用 US-ASCII
string    method name，采用 US-ASCII
....      方法特有的域
```

'user name'和'service name'在每一次新的验证尝试中重复出现，而且它们的值可能改变。服务器系统实现必须在每个消息中对它们进行仔细的检查，当它们改变时必须刷新任何累积的验证状态。如果当'user name'或'service name'改变时，服务器系统实现无法刷新验证状态，则必须断开连接。

'service name'规定了验证完成后启动的服务。可能有多个不同的已验证的服务。如果所请求的服务不可用，服务器可立即或在其后的任何时间断开连接。推荐发送一个合适的断开连接消息。无论如何，如果服务不存在，验证绝不能通过。

如果请求的'user name'不存在，服务器可断开连接或发送一个假的可接受的'method name'值列表，但不能让验证通过。这使得服务器能避免泄漏哪些帐户存在的信息。无论如何，如果'user name'不存在，验证绝不能通过。

虽然发送未被服务器列为可用的请求通常对客户端来说没有什么意义，但发送这种请求并违法，服务器应简单的拒绝其不能识别的请求。

验证请求可能引发更多消息的交换，这些消息依赖于验证方法。客户端可在任何时间发送一个新的 SSH_MSG_USERAUTH_REQUEST 消息，在这种情况下，服务器必须丢弃之前的验证尝试并继续处理新的验证。

定义了下列'method name'。

```
"publickey"       REQUIRED
"password"        OPTIONAL
"hostbased"       OPTIONAL
"none"            NOT RECOMMENDED
```

可根据[SSH-ARCH]和[SSH-NUMBERS]的规定定义更多的'method name'。

5.1. 对验证请求的响应

如果服务器拒绝验证请求，它必须进行如下响应：

```
byte       SSH_MSG_USERAUTH_FAILURE
name-list  authentications that can continue
boolean    partial success
```

'authentications that can continue'是一个逗号分隔的名称列表，包含服务器认为可用的'method name'值。

推荐服务器只把确实有用的'method name'值放在名称列表中。但是，名称列表中包含不可用的'method name'值并不违法。

已经成功完成的验证方法不应被包含在名称列表中，除非出于某些原因应再次执行该验证方法。

如果验证请求被成功的处理了，则'partial success'的值必须为 TRUE。如果请求没有被成功处理，该值必须为 FALSE。

当服务器认为验证通过时，它必须进行如下响应：

```
byte       SSH_MSG_USERAUTH_SUCCESS
```

注意，在一个包含多个方法的验证序列中，该消息不是在每一步之后发送，而是只在验证完成后发送。

客户端可连续发送多个验证请求而不必等待前一个请求的响应。服务器必须完整的处理每个请求，对失败的验证请求给予 SSH_MSG_USERAUTH_FAILURE 消息应答，然后再处理下一个请求。

新的验证请求将导致之前的需要更多的消息交换的验证请求被放弃。如果客户端还没有从服务器接收到对之前的验证请求的响应，它绝不能发送一个新的验证请求。当一个验证方法被放弃时，禁止发送 SSH_MSG_USERAUTH_FAILURE 消息。

SSH_MSG_USERAUTH_SUCCESS 必须只发送一次。发送 SSH_MSG_USERAUTH_SUCCESS 之后接收到的任何其他的验证请求应被静默的忽略。

如果验证请求的结果是 SSH_MSG_USERAUTH_SUCCESS，客户端在发送该请求之后发送的任何非验证消息必须被传递给运行在本协议之上的服务。这些消息可通过它们的消息编号来识别（见第 6 节）。

5.2. "none"验证请求

客户端可使用"none"作为验证的'method name'来请求可用的'method name'值列表。

如果对该用户不需要验证，服务器必须返回 SSH_MSG_USERAUTH_SUCCESS。否则，服务器必须返回 SSH_MSG_USERAUTH_FAILURE，并可在'authentications that can continue'的值中包含可用的方法列表。

服务器绝不能把"none"列为可用的'method name'。

5.3. 完成用户验证

验证当服务器已经以 SSH_MSG_USERAUTH_SUCCESS 响应时完成。所有在发送该消息后收到的验证相关消息应被静默的忽略。

在发送 SSH_MSG_USERAUTH_SUCCESS 后，服务器启动所请求的服务。

5.4. 警告消息（Banner Message）

在一些司法管辖区，在验证前发送一个警告消息可能对获得法律保护有作用。例如，很多 UNIX 主机一般使用 TCP wrapper 或类似的软件在提示登录前显示一个警告信息，该信息的文本一般在/etc/issue 目录下。

SSH 服务器可在本验证协议开始到验证成功期间的任意时间发送一个 SSH_MSG_USERAUTH_BANNER 消息。该消息包含要在尝试验证前显示给客户端用户的文本。格式如下：

```
byte        SSH_MSG_USERAUTH_BANNER
string      message，采用 ISO-10646 UTF-8 编码[RFC3629]
string      language tag，[RFC3066]
```

默认情况下，客户端应在屏幕上显示'message'。但是，由于'message'很有可能对每一个登录尝试都会发送，而且有些客户端软件要打开一个单独的窗口来显示该警告，因此客户端软件可以允许用户显式的禁用显示警告的功能。'message'可包含多个行，用 CRLF 表示换行。

如果显示了'message'字串，应使用[SSH-ARCH]中讨论的控制字符筛选以避免受到发送终端控制字符的攻击。

6．验证协议消息编号

本验证协议使用的所有消息编号在 50 至 79 范围内，该范围属于为运行在 SSH 传输层协议之上的协议保留的范围。

80 及其后的消息编号为在本验证协议之后运行的协议保留，因此在验证完成前收到一个这种消息属于错误，服务器必须断开连接，为便于故障诊断最好发送适当的断开连接消息。

在验证通过后，这种消息被传递给更高层的服务。

以下是通用的验证消息编号：

```
SSH_MSG_USERAUTH_REQUEST        50
SSH_MSG_USERAUTH_FAILURE        51
SSH_MSG_USERAUTH_SUCCESS        52
SSH_MSG_USERAUTH_BANNER         53
```

除上述消息编号之外，60 至 79 这一段消息编号是为方法特有的消息保留的。只有服务器发送这些消息（客户端只发送 SSH_MSG_USERAUTH_REQUEST 消息）。不同的验证方法可重用相同的消息编号。

7．公钥验证方法："publickey"

唯一 REQUIRED 的验证'method name'是"publickey"验证。所有系统实现必须支持这种方法；但是，并非所有用户都需要拥有公钥，而且在不远的将来大多数本地策略不大可能对所有用户都要求公钥验证。

对这种方法，拥有一个私钥相当于可通过验证。该方法的工作方式是：发送一个使用用户私钥产生的签名，服务器必须检查该密钥是不是该用户的一个有效的证明，并且必须检查签名是不是合法。如果两项都成立，验证请求必须被通过；否则，必须被拒绝。注意，在成功通过验证后，服务器可要求进一步的验证。

私钥经常以加密的形式保存在客户端主机，而且在生成签名前用户必须提供密码。即使私钥没有加密，签名运算也包含一些复杂的计算。为了避免没有必要的处理和用户交互，可使用以下消息来询问使用"publickey"方法验证是否可被接受。

```
byte      SSH_MSG_USERAUTH_REQUEST
string    user name，采用 ISO-10646 UTF-8 编码[RFC3629]
string    service name，US-ASCII 编码
string    "publickey"
boolean   FALSE
string    public key algorithm name
string    public key blob
```

公钥算法在传输层协议[SSH-TRANS]中规定。'public key blob'可包含证书。

任何公钥算法都可供用于验证。特别的，可用的算法不受密钥交换过程中已协商的算法的限制。如果服务器不支持一些算法，它必须简单的拒绝请求。

服务器必须以 SSH_MSG_USERAUTH_FAILURE 或下列消息对以上消息进行应答：

```
byte      SSH_MSG_USERAUTH_PK_OK
string    public key algorithm name from the request
string    public key blob from the request
```

要执行实际的验证，客户端可在其后发送一个用私钥生成的签名。客户端可直接发送签名，而不首先检验该密钥是否可接受。发送签名的数据包如下：

```
byte      SSH_MSG_USERAUTH_REQUEST
string    user name
string    service name
string    "publickey"
boolean   TRUE
string    public key algorithm name
string    public key to be used for authentication
string    signature
```

'signature'的值是使用私钥对下列数据（按下列顺序）的签名：

```
string    session identifier
byte      SSH_MSG_USERAUTH_REQUEST
string    user name
string    service name
string    "publickey"
boolean   TRUE
string    public key algorithm name
string    public key to be used for authentication
```

当服务器接收到该消息，它必须检查提供的密钥是否可接受，如果是，还必须检查签名是否正确。

如果两个检查都成功，则该方法成功。注意服务器可要求进一步验证。如果不需要更多的验证，服务器必须以 SSH_MSG_USERAUTH_SUCCESS 响应，如果请求失败或需要更多的验证，服务器必须以 SSH_MSG_USERAUTH_FAILURE 响应。

"publickey"验证方法使用了以下方法特定的消息编号：

```
SSH_MSG_USERAUTH_PK_OK          60
```

8．密码验证方法："password"

密钥验证使用以下的数据包。注意，服务器可要求用户修改密码。所有系统实现应支持密码验证。

```
byte        SSH_MSG_USERAUTH_REQUEST
string      user name
string      service name
string      "password"
boolean     FALSE
string      plaintext password，采用 ISO-10646 UTF-8 编码[RFC3629]
```

注意，'plaintext password'值采用 ISO-10646 UTF-8 编码。服务器负责解释密码以及根据密码数据库对密码进行验证。但是，如果客户端读取密码时使用的是其他编码（例如，ISO 8859-1 – ISO Latin1），它必须在传输前将密码转换为 ISO-10646 UTF-8，而服务器必须将密码再转换为原系统使用的编码。

从国际化的观点出发，希望只要用户输入了密码，验证过程就能工作，而不论用户使用的是什么操作系统和客户端软件。这需要规格化（normalization）。支持非 ASCII 密码的系统应在把密码和用户名添加到数据库或将其与数据库中已有的项目进行比较（做或不做哈希）时对密码和用户名进行规格化。含有储存密码或比较密码的 SSH 系统实现应使用[RFC4013]进行规格化。

注意，尽管数据包传输的是明文密码，整个数据包是由传输层加密的。服务器和客户端都应检查底层的传输层是否提供了机密性（即，是否使用了加密）。如果没有提供机密性（"none"加密器），应禁用密码验证。如果没有机密性或没有 MAC，应禁用密码修改。

通常，服务器以成功或失败对该消息进行响应。但是，如果密码过期，服务器应使用 SH_MSG_USERAUTH_PASSWD_CHANGEREQ 作为响应，以表明该情况。在任何情况下，服务器绝不能允许使用一个过期的密码进行验证。

```
byte        SSH_MSG_USERAUTH_PASSWD_CHANGEREQ
string      prompt，采用 ISO-10646 UTF-8 编码[RFC3629]
string      language tag，[RFC3066]
```

在这种情况下，客户端可使用一种其他的验证方式，或向用户要求一个新的密码并使用以下消息重试密码验证。在服务器没有要求修改密码时，客户端也可用该消息代替正常的密码验证请求。

```
byte        SSH_MSG_USERAUTH_REQUEST
string      user name
string      service name
string      "password"
boolean     TRUE
string      plaintext old password，采用 ISO-10646 UTF-8 编码[RFC3629]
string      plaintext new password，采用 ISO-10646 UTF-8 编码[RFC3629]
```

对每一个这种请求，服务器必须以 SSH_MSG_USERAUTH_SUCCESS 、SSH_MSG_USERAUTH_FAILURE 或另一个 SSH_MSG_USERAUTH_PASSWD_CHANGEREQ 应答。这些应答的含义如下：

- SSH_MSG_USERAUTH_SUCCESS – 密码已修改，验证成功完成。

- SSH_MSG_USERAUTH_FAILURE with partial success – 密码已修改，但需要更多的验证。

- SSH_MSG_USERAUTH_FAILURE without partial success – 密码未被修改。原因为服务器不支持修改密码或旧密码不正确。注意，如果服务器已经发送过SSH_MSG_USERAUTH_PASSWD_CHANGEREQ，可知服务器支持修改密码。

- SSH_MSG_USERAUTH_CHANGEREQ – 密码未被修改。原因是新密码不可接受（例如，太容易猜测）。

密码验证方法使用了以下方法特定的消息编号：

```
SSH_MSG_USERAUTH_PASSWD_CHANGEREQ      60
```

9．基于主机的验证："hostbased"

一些站点希望基于用户的主机和远程主机上的用户名进行验证。虽然这种验证方式不适合高安全性站点，但在很多环境下是非常方便的。这种验证方式是可选的。当使用时，应特别注意防止普通用户获得主机的私钥。

客户端发送以下消息来请求基于主机的验证。它类似于 UNIX 的"rhosts"和"hosts.equiv"风格的验证，但对客户端主机身份的检查更加严格。

该方法的工作方式如下：客户端发送一个用客户端主机私钥产生的签名，服务器用该主机的公钥对签名进行检查。一旦客户端主机的身份被确认，将根据服务器和客户端的用户名以及客户端主机名称进行授权（但不用进行进一步的身份验证）

```
byte      SSH_MSG_USERAUTH_REQUEST
string    user name
string    service name
string    "hostbased"
string    public key algorithm for host key
string    public host key and certificates for client host
string    client host name，表示为 FQDN（全域名），US-ASCII 编码
string    user name on the client host，ISO-10646 UTF-8 编码 [RFC3629]
string    signature
```

传输层协议[SSH-TRANS]定义了'public key algorithm for host key'中可使用的公钥算法名称。'public host key and certificates for client host'可包含证书。

'signature'的值是使用主机私钥对下列数据（按下列顺序）的签名：

```
string    session identifier
byte      SSH_MSG_USERAUTH_REQUEST
string    user name
string    service name
string    "hostbased"
string    public key algorithm for host key
string    public host key and certificates for client host
string    client host name，表示为 FQDN（全域名），US-ASCII 编码
string    user name on the client host，ISO-10646 UTF-8 编码[RFC3629]
```

服务器必须验证主机密钥确实属于消息中给出的客户端主机、所述主机上的所述用户被允许登录，以及'signature'的值是使用所述主机的私钥对恰当的值的合法的签名。如果服务器希望只验证客户端主机，它可忽略客户端的'user name'。

在任何可能的情况下，推荐服务器另行对从（不可信的）网络上获得的网络地址与所述的客户端主机名称是否相符机型检查。这增加了利用失密的主机密钥的难度。注意，对穿过防火墙的连接可能需要进行特殊处理。

10. IANA 考虑

[SSH-ARCH]、[SSH-TRANS]、[SSH-USERAUTH]和[SSH-CONNECT]定义的 SSH 协议的 IANA 考虑详见[SSH-NUMBERS]。

11. 安全性考虑

本协议的目的是执行客户端用户验证。本协议假定其运行在一个安全的传输层协议之上，已经验证了服务器，建立了一个加密的通信信道，并对该会话产生了一个唯一的会话标识。传输层为密码验证和其他依赖于秘密数据的方法提供了前向安全性（forward secrecy）。

对本协议的全面的安全性考虑见[SSH-ARCH]。

12. 参考文献

（略）

SSH 连接协议

摘要

SSH 是在一个不安全的网络上实现安全的远程登录和其他安全的网络服务的协议。本文描述 SSH 连接协议。它提供交互式登录会话、远程执行命令、TCP/IP 连接转发，以及 X11 连接转发。所有这些信道是对一个单一的加密隧道的多路复用。SSH 连接协议设计为运行于 SSH 传输层协议和验证协议之上。

1．简介

SSH 连接协议设计为运行于 SSH 传输层协议和验证协议（[SSH-TRANS]和[SSH-USERAUTH]）之上。它提供交互式登录会话、远程执行命令、TCP/IP 连接转发，以及 X11 连接转发。

本协议的'service name'为"ssh-connection"。

读者应首先阅读 SSH 架构文档[SSH-ARCH]之后再阅读本文。本文在使用架构文档中的术语和标记法时未加引用或解释。

2．撰写者

（略）

3．本文中的惯用约定

所有与 SSH 协议相关的文档都应使用关键词“必须”、“禁止（绝不能）”、“必须的”、“应”、“不应”、“推荐”、“可（能）”、“可选的”来描述要求。这些关键词的含义符合[RFC2119]的描述。

本文中用于描述命名空间分配的关键词"PRIVATE USE"、"HIERARCHICAL ALLOCATION"、"FIRST COME FIRST SERVED"、 "EXPERT REVIEW"、"SPECIFICATION REQUIRED"、"IESG APPROVAL"、"IETF CONSENSUS"以及"STANDARDS ACTION"的含义符合[RFC2434]的描述。

协议文档中定义了协议域和可能的取值。协议域将在消息定义中定义。例如，SSH_MSG_CHANNEL_DATA 的定义如下：

```
byte      SSH_MSG_CHANNEL_DATA
uint32    recipient channel
string    data
```

在整套协议文档中，当引用域时，域的名称出现在单引号中。当引用域的值时，它们出现在双引号中。例如，'data'的可能取值是"foo"和"bar"。

4．全局请求

有几种请求会全局的影响远端的状态，与信道无关。例如，在指定的端口启动 TCP/IP 转发的

请求。注意，客户端和服务器均可在任何时候发送全局请求，接收方必须适当的响应。所有这种请求使用以下格式。

```
byte        SSH_MSG_GLOBAL_REQUEST
string      request name，US-ASCII 编码
boolean     want reply
....        请求特定的数据
```

'request name'的值遵循 DNS 可扩展性命名约定，其要点见[SSH-ARCH]。

如果'want reply'的值为 TRUE，接收方将以 SSH_MSG_REQUEST_SUCCESS 或 SSH_MSG_REQUEST_FAILURE 对请求进行响应。

```
byte        SSH_MSG_REQUEST_SUCCESS
....        响应特定的数据
```

通常，'response specific data'并不存在。

如果接收方未识别或不支持该请求，它简单的以 SSH_MSG_REQUEST_FAILURE 进行响应。

```
byte        SSH_MSG_REQUEST_FAILURE
```

一般来说，应答消息不包含请求类型标识。为了让请求发起人能够识别每个应答涉及哪个请求，发送对 SSH_MSG_GLOBAL_REQUESTS 的应答的顺序必须与对应的请求消息的顺序一致。对信道请求，与同一个信道相关的应答同样必须按正确的顺序进行。但是，对不同信道的信道请求的应答可不按顺序。

5．信道机制

所有终端会话、转发连接等等，都是信道。任何一方都可打开一个信道。多个信道多路复用同一个连接。

双方通过编号对信道进行识别。各方指称同一个信道的编号可能不同。打开一个信道的请求包含发送方的信道编号。任何其他信道相关消息包含接收方对该信道的信道编号。

信道是流控制的（flow-controlled）。在接收到一个消息表明信息窗口（window space）可用之前，不能向信道发送数据。

5.1．打开一个信道

当任何一方希望打开一个新的信道，它为该信道分配一个本地编号。然后，它向另一方发送以下消息，并在消息中包含本地信道编号和初始窗口大小（initial window size）。

```
byte        SSH_MSG_CHANNEL_OPEN
string      channel type，US-ASCII 编码
uint32      sender channel
```

```
uint32     initial window size
uint32     maximum packet size
....       信道类型特定数据
```

'channel type'是一个带有相似的扩展机制的名称，见[SSH-ARCH]和[SSH-NUMBERS]的描述。'sender channel'是本消息的发送方使用的对信道的一个本地标识。'initial window size'规定在无需调整窗口的情况下，能够通过信道发送给本消息的发送方的数据的字节数。'maximum packet size'规定能发送给本消息发送方的单个数据包的最大大小。例如，可能希望对交互式连接使用较小的数据包，以在慢速连接上获得较好的交互响应。

另一方然后决定能否打开该信道，并且以 SSH_MSG_CHANNEL_OPEN_CONFIRMATION 或 SSH_MSG_CHANNEL_OPEN_FAILURE 进行应答。

```
byte       SSH_MSG_CHANNEL_OPEN_CONFIRMATION
uint32     recipient channel
uint32     sender channel
uint32     initial window size
uint32     maximum packet size
....       信道类型特定数据
```

'recipient channel'是原打开信道请求中给出的信道编号，'sender channel'是另一方分配的信道编号。

```
byte       SSH_MSG_CHANNEL_OPEN_FAILURE
uint32     recipient channel
uint32     reason code
string     description，ISO-10646 UTF-8 编码[RFC3629]
string     language tag，[RFC3066]
```

如果 SSH_MSG_CHANNEL_OPEN 消息的接收方不支持指定的'channel type'，它简单的以 SSH_MSG_CHANNEL_OPEN_FAILURE 进行应答。客户端可将'description'显示给用户。如果这样做了，客户端软件应采取[SSH-ARCH]中讨论的预防措施。

SSH_MSG_CHANNEL_OPEN_FAILURE 中的'reason code'值在下表中定义。注意，为了可读性，'reason code'的值是以十进制格式给出的，但它们实际是 uint32 值。

Symbolic name	reason code
SSH_OPEN_ADMINISTRATIVELY_PROHIBITED	1
SSH_OPEN_CONNECT_FAILED	2
SSH_OPEN_UNKNOWN_CHANNEL_TYPE	3
SSH_OPEN_RESOURCE_SHORTAGE	4

要请求为 SSH_MSG_CHANNEL_OPEN 指派新的范围在 0x00000005 至 0xFDFFFFFF 之间

的'reason code'值（以及相关'description'文本），必须使用 IETF CONSENSUS 方法，见[RFC2434]。IANA 不会指派在 0xFE000000 至 0xFFFFFFFF 范围内的信道连接失败的'reason code'值。该范围内的信道连接失败的'reason code'值留给 PRIVATE USE，见[RFC2434]。

虽然 IANA 将不会控制 0xFE000000 至 0xFFFFFFFF 范围，但该范围将划分为两部分，并依以下约定管理。

- 0xFE000000 至 0xFEFFFFFF 将与本地指派的信道共同使用。例如，如果提议打开一个'channel type'是"example_session@example.com"的信道，但失败，则响应将包含一个由 IANA 指派的'reason code'（在 0x00000001 至 0xFDFFFFFF 范围内）或一个本地指派的在 0xFE000000 至 0xFEFFFFFF 范围内的值。自然的，如果服务器不理解提议的'channel type'，那么如果发送'reason code'，它必须是 0x00000003。如果服务器确实理解该'channel type'，但打开信道仍然失败，那么服务器应以一个本地指派的、与提议的本地'channel type'相一致的'reason code'值进行响应。这里假定系统实现人员将首先尝试使用 IANA 指派的'reason code'值，然后再使用他们本地指派的'reason code'。

- 对从 0xFF 开始的范围没有限制或建议。该范围内的任何值没有互操作性是正常的，它们主要是用于试验。

5.2. 数据传输

窗口大小规定了对方在必须等待窗口调整之前能够发送多少字节。双方使用以下消息调整窗口。

```
byte       SSH_MSG_CHANNEL_WINDOW_ADJUST
uint32     recipient channel
uint32     bytes to add
```

在接收该消息后，接收方可在之前被允许发送的字节数之后继续发送指定的字节数；窗口空间是递增的。系统实现必须正确的处理最大 2^32 - 1 字节的窗口。窗口禁止超过 2^32 - 1 字节。

数据传输通过以下类型的消息实现。

```
byte       SSH_MSG_CHANNEL_DATA
uint32     recipient channel
string     data
```

数据的最大长度由信道的数据包最大长度或当前窗口空间两者中较小者决定。当前窗口空间根据已发送的数据量递减。双方均可忽略在窗口空间耗尽后发送的所有额外的数据。

系统实现预计将对 SSH 传输层数据包大小有一些限制（对接收的数据包的任何限制必须大于等于 32768 字节，见[SSH-TRANS]）。SSH 连接层的系统实现

- 绝不能公布一个最大的数据包大小，使可能产生比传输层可接收的数据包更大的数据包。

- 绝不能生成比其传输层可发送的数据包更大的数据包，即使远端可能可以接受非常大的数据包。

此外，一些信道能传输几种类型的数据。例如，交互式会话的 stderr 数据。这种数据能够通过 SSH_MSG_CHANNEL_EXTENDED_DATA 消息传递，其中一个单独的整数规定数据的类型。可用的类型和它们的解释依赖于信道类型。

```
byte       SSH_MSG_CHANNEL_EXTENDED_DATA
uint32     recipient channel
uint32     data_type_code
string     data
```

通过该消息发送的数据，与普通数据消耗的窗口空间相同。

目前，只定义了以下类型。注意，为了可读性'data_type_code'的值以十进制格式给出，但该值实际是 uint32 类型。

Symbolic name	data_type_code
SSH_EXTENDED_DATA_STDERR	1

扩展信道数据传输的'data_type_code'值必须按顺序指派。要请求为扩展信道数据传输的'data_type_code'指派新的范围在 0x00000002 至 0xFDFFFFFF 之间的值和相关的'data'字串，必须使用 IETF CONSENSUS 方法，见[RFC2434]。IANA 将不会为扩展信道数据传输的'data_type_code'指派 0xFE000000 至 0xFFFFFFFF 范围内的值。该范围内的'data_type_code'值留作 PRIVATE USE，见[RFC2434]。如上所述，对 IANA 的实际指引见[SSH-NUMBERS]。

5.3. 关闭一个信道

当一方将不再向一个信道发送数据，它应发送 SSH_MSG_CHANNEL_EOF。

```
byte       SSH_MSG_CHANNEL_EOF
uint32     recipient channel
```

对本消息不发送显式的响应，但应用软件也可向信道的另一端发送 EOF。注意，信道在发送本消息之后仍保持打开，对方仍可发送更多数据。本消息不消耗窗口空间，在窗口空间耗尽的情况下仍可发送本消息。

当任何一方希望终止信道时，它发送 SSH_MSG_CHANNEL_CLOSE。一方在收到该消息时，必须回发一条 SSH_MSG_CHANNEL_CLOSE 消息，除非已经对该信道发送了该消息。对一方来说，信道在该方已发送并且已接收 SSH_MSG_CHANNEL_CLOSE 时视为关闭，这时该方可重用该信道编号。任何一方可在没有发送或接收 SSH_MSG_CHANNEL_EOF 的情况下发送 SSH_MSG_CHANNEL_CLOSE。

```
byte        SSH_MSG_CHANNEL_CLOSE
uint32      recipient channel
```

本消息不消耗窗口空间，在窗口空间耗尽的情况下仍可发送本消息。

推荐如果可能应把在该消息之前发送的所有数据都递送到实际目的地。

5.4．信道特定的请求

很多'channel type'有特定于该'channel type'的扩展。例如，为一个交互式会话请求一个 pty（pseudo terminal，伪终端）。

所有信道特定请求使用下列格式。

```
byte        SSH_MSG_CHANNEL_REQUEST
uint32      recipient channel
string      request type，US-ASCII 编码
boolean     want reply
....        类型特定数据
```

如果'want reply'为 FALSE，将不会对请求发送响应。否则，接收方以 SSH_MSG_CHANNEL_SUCCESS、SSH_MSG_CHANNEL_FAILURE 或请求特定的延续消息。如果未识别请求或所用信道不支持该请求，返回 SSH_MSG_CHANNEL_FAILURE。

本消息不消耗窗口空间，在窗口空间耗尽的情况下仍可发送本消息。'request type'值对于每一个信道类型是局部的。

客户端可继续发送消息而不等待对该请求的响应。

'request type'名称遵循 DNS 可扩展性命名约定，见[SSH-ARCH]和[SSH-NUMBERS]。

```
byte        SSH_MSG_CHANNEL_SUCCESS
uint32      recipient channel

byte        SSH_MSG_CHANNEL_FAILURE
uint32      recipient channel
```

本消息不消耗窗口空间，在窗口空间耗尽的情况下仍可发送本消息。

6．交互式会话

会话是远程执行一个程序。所述程序可以是一个命令解释程序、一个应用程序、一个系统命令或一些内置的子系统。它可以有也可以没有 tty，而且可以涉及也可以不涉及 X11 转发。多个会话可同时活动。

6.1. 打开一个会话

通过发送以下消息启动一个会话。

```
byte      SSH_MSG_CHANNEL_OPEN
string    "session"
uint32    sender channel
uint32    initial window size
uint32    maximum packet size
```

客户端系统实现应拒绝任何打开会话信道的请求，以加大通过已被攻破的服务器攻击客户端的难度。

6.2. 请求一个伪终端

使用以下消息可为会话分配一个伪终端。

```
byte      SSH_MSG_CHANNEL_REQUEST
uint32    recipient channel
string    "pty-req"
boolean   want_reply
string    TERM environment variable value（如 vt100）
uint32    terminal width，单位为字符（如 80）
uint32    terminal height，单位为行（如 24)）
uint32    terminal width，单位为像素（如 640）
uint32    terminal height，单位为像素（如 480）
string    encoded terminal modes
```

'encoded terminal modes'在第 8 节描述。必须忽略为零的尺寸参数。以字符/行为单位的尺寸（当不为零时）覆盖以像素为单位的尺寸。像素尺寸指窗口的可绘制区域。

尺寸参数仅为提示性的。

客户端应忽略 pty 请求。

6.3. X11 转发

6.3.1. 请求 X11 转发

会话可发送一个 SSH_MSG_CHANNEL_REQUEST 消息来请求 X11 转发。

```
byte      SSH_MSG_CHANNEL_REQUEST
uint32    recipient channel
string    "x11-req"
boolean   want reply
boolean   single connection
string    x11 authentication protocol
string    x11 authentication cookie
```

```
uint32     x11 screen number
```

推荐发送的'x11 authentication cookie'是一个假的、随机的 cookie，当连接请求被接收时，检查该 cookie 并替换为真正的 cookie。

当会话信道被关闭时，X11 连接转发应停止。但是，当会话信道被关闭时，已经打开的转发不应被自动关闭。

如'single connection'为 TRUE，只应转发单个连接。在第一个连接之后，或在会话信道关闭之后，均不会转发更多的连接。

'x11 authentication protocol'是所使用的 X11 验证方法的名称，例如，"MIT-MAGIC-COOKIE-1"。

'x11 authentication cookie'必须为十六进制编码。X 协议文档见[SCHEIFLER]。

6.3.2. X11 信道

X11 信道通过一个打开信道请求打开。所打开的信道独立于会话，关闭会话信道不会关闭转发的 X11 信道。

```
byte       SSH_MSG_CHANNEL_OPEN
string     "x11"
uint32     sender channel
uint32     initial window size
uint32     maximum packet size
string     originator address（如"192.168.7.38"）
uint32     originator port
```

接收方应以 SH_MSG_CHANNEL_OPEN_CONFIRMATION 或 SSH_MSG_CHANNEL_OPEN_FAILURE 响应。

如果系统实现没有请求 X11 转发，它必须拒绝任何打开 X11 信道的请求。

6.4. 环境变量传递

环境变量可被传递给后续将启动的命令解释程序/命令。在一个享有特权的进程中不受控的环境变量设置可称为一场安全性灾难。推荐系统实现维持一个允许的变量名称列表或只在服务器进程提供了足够的特权时才设置环境变量。

```
byte       SSH_MSG_CHANNEL_REQUEST
uint32     recipient channel
string     "env"
boolean    want reply
string     variable name
string     variable value
```

6.5. 启动一个命令解释程序或一个命令

一旦会话被设立，远端将启动一个程序。该程序可能是一个命令解释程序、一个应用程序或一个具有与主机无关的名称的子系统。每个信道只允许一个请求。

```
byte       SSH_MSG_CHANNEL_REQUEST
uint32     recipient channel
string     "shell"
boolean    want reply
```

该消息将请求在远端启动用户的默认命令解释程序（典型的 UNIX 系统下在/etc/passwd 中定义）。

```
byte       SSH_MSG_CHANNEL_REQUEST
uint32     recipient channel
string     "exec"
boolean    want reply
string     command
```

该消息将请求服务器开始执行指定的命令。'command'字串可包括一个路径。必须采取一般性预防措施以防止执行未授权的命令。

```
byte       SSH_MSG_CHANNEL_REQUEST
uint32     recipient channel
string     "subsystem"
boolean    want reply
string     subsystem name
```

最后这种形式执行一个预定义的子系统。可预计该行为将包含一个通用的文件传输机制，以及可能的其他功能。系统实现也可允许设置更多的机制。由于通常使用用户的命令解释程序来执行子系统，建议子系统协议在协议传输的开始包含一个"magic cookie"以区分命令解释程序初始化脚本等生成的任意输出。这些命令解释程序的伪输出可在服务器或客户端过滤掉。

当启动命令解释程序或其他程序时，服务器不应暂停协议堆栈的执行。这些程序的所有输入和输出应被重定向到信道或加密隧道。

推荐要求对这些消息进行回复并对回复进行检查。客户端应忽略这些消息。

子系统名称遵循 DNS 可扩展性命名约定，见[SSH-NUMBERS]。

6.6. 会话数据传输

会话数据的传输使用 SSH_MSG_CHANNEL_DATA 和 SSH_MSG_CHANNEL_EXTENDED_DATA 数据包和窗口机制完成。为 stderr 数据定义了扩展数据类型 SSH_EXTENDED_DATA_STDERR。

6.7. 窗口大小变化消息

当客户端窗口（终端）大小变化时，客户端可向另一方发送一条消息，向另一方通知新的大小。

```
byte      SSH_MSG_CHANNEL_REQUEST
uint32    recipient channel
string    "window-change"
boolean   FALSE
uint32    terminal width，列
uint32    terminal height，行
uint32    terminal width，像素
uint32    terminal height，像素
```

对该消息不应发送响应。

6.8. 本地流程控制

在很多系统下，可确定一个伪终端是否使用 control-S/control-Q 流程控制。当允许流程控制时，经常希望在客户端进行流程控制来加速对用户请求的响应。这通过以下通知实现。初始时，服务器负责流程控制。（这里，再一次的，客户端指创建会话的一方，服务器指另一方。）

服务器用以下消息将其能或不能执行流程控制（control-S/control-Q 处理）通知给客户端。如果'client can do'为 TRUE，允许客户端使用 control-S 和 control-Q 执行流程控制。客户端可忽略该消息。

```
byte      SSH_MSG_CHANNEL_REQUEST
uint32    recipient channel
string    "xon-xoff"
boolean   FALSE
boolean   client can do
```

对该消息不发送响应。

6.9. 信号 （Signals）

可使用以下消息向远端进程/服务投递信号。一些系统可能没有实现信号，这种情况下它们应忽略该消息。

```
byte      SSH_MSG_CHANNEL_REQUEST
uint32    recipient channel
string    "signal"
boolean   FALSE
string    signal name（不含"SIG" 前缀）
```

对'signal name'值的编码应符合本节有关在 SSH_MSG_CHANNEL_REQUEST 中使用"exit-signal"的讨论。

6.10. 返回终止状态（Exit Status）

当在另一方运行的命令终止时，可通过发送以下消息返回命令的终止状态。推荐返回该状态。对该消息不发送应答。在该消息之后，需要使用 SSH_MSG_CHANNEL_CLOSE 关闭信道。

客户端可忽略该消息。

```
byte       SSH_MSG_CHANNEL_REQUEST
uint32     recipient channel
string     "exit-status"
boolean    FALSE
uint32     exit_status
```

远端的命令也可根据信号强制终止。该情况可使用以下消息。'exit_status'的值为零通常表示命令成功完成。

```
byte       SSH_MSG_CHANNEL_REQUEST
uint32     recipient channel
string     "exit-signal"
boolean    FALSE
string     signal name（不含"SIG" 前缀）
boolean    core dumped
string     error message，ISO-10646 UTF-8 编码
string     language tag，[RFC3066]
```

'signal name'是以下名称之一（来自[POSIX]）。

```
ABRT
ALRM
FPE
HUP
ILL
INT
KILL
PIPE
QUIT
SEGV
TERM
USR1
USR2
```

可使用"sig-name@xyz"格式发送更多的'signal name'值，其中"sig-name"和"xyz"可以是系统实现想要的任何名称（但不能包含"@"符号）。尽管如此，如果使用了'configure'脚本，建议所有非标准的'signal name'值采用"SIG@xyz.config.guess"格式，其中"SIG"是不包含"SIG"前缀的'signal name'，"xyz"是"config.guess"确定的主机类型。

'error message'包含对错误消息的附加的文本解释。可包含多个 CRLF（回车-换行）分隔的行。客户端软件可向用户显示该消息。如果这样做，客户端软件应采取[SSH-ARCH]中讨论的预防措施。

7. TCP/IP 端口转发

7.1. 请求端口转发

对于己方向另一方的端口转发不需要显式的进行请求。但是，如果希望将到另一方的端口的连接转发到本地，则必须显式的请求。

```
byte        SSH_MSG_GLOBAL_REQUEST
string      "tcpip-forward"
boolean     want reply
string      address to bind（例如，"0.0.0.0"）
uint32      port number to bind
```

'address to bind'和'port number to bind'规定接受转发连接的 IP 地址（或域名）和端口。一些'address to bind'字串有特殊语义：

- ""表示在 SSH 系统实现支持的所有协议族上接受连接。
- "0.0.0.0"表示监听所有 IPv4 地址。
- "::"表示监听所有 IPv6 地址。
- "localhost"表示在 SSH 系统实现支持的所有协议族上监听回送地址（loopback address），见[RFC3330]和[RFC3513]。
- "127.0.0.1"和"::1"分别表示监听 IPv4 和 IPv6 的回送接口。

注意，客户端仍可基于打开请求中传递的信息对连接进行过滤。

如果用户验证为享有特权的用户，系统实现应只允许转发享有特权的端口。

正常情况下只有客户端发送这些消息，因此客户端系统实现应拒绝这些消息。

如果客户端要求绑定的端口号为 0 而且'want reply'为 TRUE，则服务器指派下一个可用的非享有特权的端口号并回复以下消息；否则，无需响应。

```
byte        SSH_MSG_REQUEST_SUCCESS
uint32      port that was bound on the server
```

可使用以下消息取消端口转发。注意，可能在接收到该消息的回复后才接收到打开信道请求。

```
byte      SSH_MSG_GLOBAL_REQUEST
string    "cancel-tcpip-forward"
boolean   want reply
string    address_to_bind（例如，"127.0.0.1"）
uint32    port number to bind
```

正常情况下只有客户端发送这些消息，因此客户端系统实现应拒绝这些消息。

7.2. TCP/IP 转发信道

当连接到一个被请求远程转发的端口时，会打开一个信道以将该端口转发到另一方。

```
byte      SSH_MSG_CHANNEL_OPEN
string    "forwarded-tcpip"
uint32    sender channel
uint32    initial window size
uint32    maximum packet size
string    address that was connected
uint32    port that was connected
string    originator IP address
uint32    originator port
```

除非之前请求了指定端口的 TCP/IP 转发，否则系统实现必须拒绝该消息。

当连接到本地被转发的 TCP/IP 端口时，要向另一方发送以下数据包。注意，即使没有显式的请求对端口进行转发，也可发送该消息。该消息的接收方必须决定是否允许转发。

```
byte      SSH_MSG_CHANNEL_OPEN
string    "direct-tcpip"
uint32    sender channel
uint32    initial window size
uint32    maximum packet size
string    host to connect
uint32    port to connect
string    originator IP address
uint32    originator port
```

'host to connect'和'port to connect'规定了接收方建立信道连接的目的 TCP/IP 主机和端口。'host to connect'可以是域名或 IP 地址。

'originator IP address'是创建连接请求的机器的 IP 地址，'originator port'是创建连接的主机上的端口。

转发的 TCP/IP 信道独立于任何会话，关闭会话信道不表示转发的连接应被关闭。

出于安全性原因，客户端系统实现应拒绝直接打开 TCP/IP 请求。

8．终端模式编码

所有'encoded terminal modes'（像在 pty 请求中传递的）被编码成一个字节数据流。该设计意欲让编码在不同的环境下可移植。数据流包含操作码引数对（opcode argument pairs），其中操作码是一个字节值。操作码 1 至 159 有一个 unit32 引数。操作码 160 至 255 还没有定义，而且导致语法分析停止（它们只应在任何其他数据后使用）。数据流以操作码 TTY_OP_END（0x00）终止。

客户端应把它所知道的所有模式放入数据流，服务器可忽略任何它不知道的模式。这允许一定程度的机器无关，至少是在使用类似 POSIX 的 tty 界面的系统之间。协议能支持其他系统，但客户端可能需要为一些参数填写合理的值，使服务器的 pty 被设置为一个合理的模式（服务器对所有未指定的模式位（mode bits）取其默认值，并且只有一些组合有意义）。

对操作码值的命名基本上遵从 POSIX 终端模式标识。定义了下列操作码值。注意，为了可读性下面给出的值采用十进制格式，但它们实际上是 byte 类型。

opcode	mnemonic	description
0	TTY_OP_END	表示选项的结束。
1	VINTR	中断字符；如无为 255。对其他字符类似。不是所有系统都支持所有这些字符。
2	VQUIT	退出字符（在 POSIX 系统上发送 SIGQUIT 信号）。
3	VERASE	擦除光标左侧的字符。
4	VKILL	废除（Kill）当前输入行。
5	VEOF	文件尾字符（从终端发送 EOF） 。
6	VEOL	除回车及/或换行之外的行尾字符。
7	VEOL2	附加的行尾字符。
8	VSTART	继续暂停的输出（一般为 control-Q ）。
9	VSTOP	暂停输出（一般为 control-S）。
10	VSUSP	挂起（suspend）当前程序。
11	VDSUSP	另一个挂起字符。
12	VREPRINT	重印（reprint）当前输入行。
13	VWERASE	擦除光标左侧的一个单词。
14	VLNEXT	照字面上输入键入的下一个字符，即使它是一个特殊字符。
15	VFLUSH	刷新（flush）输出字符。
16	VSWTCH	切换到一个不同的命名解释程序层。
17	VSTATUS	打印系统状态行（load、command，pid 等等）
18	VDISCARD	触发战斗输出的刷新。
30	IGNPAR	忽略奇偶性标识。0 表示该标识为 FALSE，1 表示该标识为 TRUE。
31	PARMRK	屏蔽（mark）奇偶性和帧（framing）错误。
32	INPCK	允许检查奇偶性错误，
33	ISTRIP	去除字符的第八位（8th bit）。

34	INLCR	在输入中将 NL 映射为 CR。
35	IGNCR	忽略输入的 CR。
36	ICRNL	在输入中将 CR 映射为 NL。
37	IUCLC	将大写字符转换为小写。
38	IXON	允许输出的流程控制。
39	IXANY	在停止后，任意字符重新开始。
40	IXOFF	允许输入的流程控制。
41	IMAXBEL	在输入队列满时响铃。
50	ISIG	允许 INTR、QUIT、[D]SUSP 信号。
51	ICANON	规范化（canonicalize）输入行。
52	XCASE	允许使用带"\"的小写字符来输入和输出大写字符。
53	ECHO	允许回送（echoing）.
54	ECHOE	视觉上擦除字符。
55	ECHOK	Kill 字符废除当前行。
56	ECHONL	即使 ECHO 设为关闭，依然回送 NL。
57	NOFLSH	在终端后不更新。
58	TOSTOP	停止输出中的后台工作。
59	IEXTEN	允许扩展。
60	ECHOCTL	允许使用^(Char)格式的控制字符。
61	ECHOKE	废除行视觉上擦除行。
62	PENDIN	重新键入挂起的输入。
70	OPOST	允许输出处理。
71	OLCUC	将小写转换为大写。
72	ONLCR	将 NL 映射为 CR-NL。
73	OCRNL	将回车转换为 newline（输出）。
74	ONOCR	将 newline 转换为回车-newline（输出）。
75	ONLRET	Newline 执行一个回车（输出）。
90	CS7	7 位模式。
91	CS8	8 位模式。
92	PARENB	允许奇偶校验。
93	PARODD	奇偶性为偶，否则为奇。
128	TTY_OP_ISPEED	规定输入的波特率，单位是比特每秒。
129	TTY_OP_OSPEED	规定输出的波特率，单位是比特每秒。

9．消息编码总结

以下是对消息和它们的编码的总结。

SSH_MSG_GLOBAL_REQUEST	80
SSH_MSG_REQUEST_SUCCESS	81
SSH_MSG_REQUEST_FAILURE	82
SSH_MSG_CHANNEL_OPEN	90
SSH_MSG_CHANNEL_OPEN_CONFIRMATION	91
SSH_MSG_CHANNEL_OPEN_FAILURE	92
SSH_MSG_CHANNEL_WINDOW_ADJUST	93

SSH_MSG_CHANNEL_DATA	94
SSH_MSG_CHANNEL_EXTENDED_DATA	95
SSH_MSG_CHANNEL_EOF	96
SSH_MSG_CHANNEL_CLOSE	97
SSH_MSG_CHANNEL_REQUEST	98
SSH_MSG_CHANNEL_SUCCESS	99
SSH_MSG_CHANNEL_FAILURE	100

10. IANA 考虑

[SSH-ARCH]、[SSH-TRANS]、[SSH-USERAUTH]和[SSH-CONNECT]定义的 SSH 协议的 IANA 考虑详见[SSH-NUMBERS]。

11. 安全性考虑

本协议假定运行在安全的、已验证的传输之上。假定底层协议提供了用户验证和对网络级别的攻击的防护。

对本协议的完整的安全性考虑见[SSH-ARCH]。针对本文，如果主机密钥在没有通知或解释的情况下发生改变，推荐系统实现禁用所有具有潜在危险的功能（例如，代理转发（agent forwarding）、X11 转发和 TCP/IP 转发）。

12. 参考文献

（略）