网络工作组T. Dierks
请求评论：2246 Certicom
类别：标准轨道C.艾伦

Certicom
1999年1月

TLS协议
版本1.0

此备忘录的状态

本文件规定了一个互联网标准协议
互联网社区，并要求讨论和建议
改进。请参见当前版本的“互联网”
官方协议标准“（STD 1）为标准化状态
和状态。此备忘录的分发不受限制。

版权声明

抽象

本文档指定传输层安全性的1.0版本
（TLS）协议。TLS协议提供通信隐私
互联网。协议允许客户端/服务器应用程序
以设计成防止窃听的方式通信，
篡改或信息伪造。

目录

1.介绍

TLS协议的主要目标是提供隐私和数据
两个通信应用程序之间的完整性。协议是
由两层组成：TLS记录协议和TLS握手
协议。在最底层，分层的一些可靠
传输协议（例如，TCP [TCP]）是TLS记录协议。的
TLS记录协议提供了两个基本的连接安全性
属性：

- 连接是私人的。使用对称加密
  数据加密（例如，DES [DES]，RC4 [RC4]等）
  这种对称加密是为每个都唯一地生成的
  连接，并基于由另一方协商的秘密
  协议（例如TLS握手协议）。记录
  协议也可以不加密使用。

- 连接可靠。消息传输包括消息
  使用密钥MAC的完整性检查。安全散列函数（例如，
  SHA，MD5等）用于MAC计算。记录
  协议可以在没有MAC的情况下操作，但通常仅用于

此模式，而另一个协议使用记录协议
用于协商安全参数的传输。

TLS记录协议用于封装各种较高的
级协议。一种这样的封装协议，TLS握手
协议，允许服务器和客户端相互验证和
以先协商加密算法和密钥

应用协议发送或接收其第一个字节
数据。TLS握手协议提供了连接安全性
有三个基本属性：

- 对等体的身份可以使用非对称认证，或
  公共密钥，加密（例如，RSA [RSA]，DSS [DSS]等）。这个
  认证可以是可选的，但通常是需要的
  对于至少一个对等体。

- 共享秘密的协商是安全的：协商
  秘密不可用于窃听者，并且对于任何验证
  连接的秘密不能获得，即使是攻击者谁
  可以将自己置于连接的中间。

- 协商是可靠的：没有攻击者可以修改
  协商通信，而不被双方检测到
  到通信。

TLS的一个优点是它是应用协议无关的。
更高级别的协议可以层叠在TLS协议之上
透明。然而，TLS标准不指定如何
协议使用TLS添加安全性；关于如何启动TLS的决定
握手以及如何解释身份验证证书
交换是留给设计师的判断和
在TLS之上运行的协议的实现者。

2.目标

TLS协议的目标，按照它们的优先顺序，是：

1. 加密安全：TLS应该用于建立安全
   双方之间的连接。

2. 互操作性：独立的程序员应该能够
   使用TLS开发应用程序，然后才能使用
   成功地交换加密参数没有知识
   的对方的代码。

3. 可扩展性：TLS旨在提供一个新的框架
   公钥和批量加密方法可以并入
   必要。这也将实现两个子目标：防止

   需要创建一个新的协议（和风险的介绍
   的可能的新弱点），并避免需要实施
   整个新的安全库。

4. 相对效率：密码操作往往很高
   CPU密集型，特别是公钥操作。为了这
   原因，TLS协议已经并入了可选会话
   缓存方案来减少需要的连接数
   从头开始建立。此外，已经小心
   减少网络活动。

3.本文档的目标

本文档和TLS协议本身基于SSL 3.0
协议规范由Netscape公布。差异
这个协议和SSL 3.0之间不是戏剧性的，但他们是
足够重要，TLS 1.0和SSL 3.0不能互操作
（虽然TLS 1.0确实包含了TLS的机制
实现可以退回到SSL 3.0）。本文档旨在
主要是为那些将要实施协议的读者
做加密分析。规格已经
书面记住这一点，它的目的是反映的需要
这两组。为此，许多算法依赖
数据结构和规则包括在文本的正文中（如
反对在附录中），从而更容易访问它们。

本文档不提供任何服务的详细信息
定义也不是接口定义，尽管它覆盖了选择
维持固体所需的政策领域
安全。

4.演讲语言

本文档涉及外部数据的格式化
表示。以下非常基本和有点随便
将使用定义的表示语法。语法从
几个来源在其结构。虽然它类似
编程语言“C”在其语法和XDR [XDR]中
语法和意图，绘制太多的并行是有风险的。的
这个表示语言的目的是仅记录TLS，而不是
具有超出该特定目标的一般应用。

4.1。基本块大小

显式指定所有数据项的表示。的
基本数据块大小为一个字节（即8位）。多字节数据
项目是字节的连接，从左到右，从顶部到
底部。从字节流中的一个多字节项（一个数字在
示例）（使用C符号）：

```
值=（字节[0] << 8 *（n-1））|  （byte [1] << 8 *（n-2））|
        ... | byte [n-1];
```

多字节值的字节排序是常见的网络
字节顺序或大端格式。

4.2。杂

注释以“/ *”开头，以“* /”结尾。

可选组件通过将它们包围在“[[]]”double中来表示
括号。

包含未解释数据的单字节实体是类型
不透明。

4.3。载体

向量（单维数组）是均匀数据流
元素。向量的大小可以在文档中指定
时间或未指定，直到运行时。在任一情况下的长度
声明字节数，而不是元素的数量
向量。用于指定新类型T'的语法，其是固定的
T类型的长度向量

```
T T'[n];
```

这里T'在数据流中占据n个字节，其中n是数据流的倍数
T的大小。向量的长度不包括在中
编码流。

在下面的示例中，Datum定义为三个连续
协议不解释的字节，而数据为三
连续的Datum，共消耗9个字节。

```
opaque Datum [3]; / *三个未解释字节* /
基准数据[9]; / * 3个连续的3字节向量* /
```


可变长度向量通过指定法律的子范围来定义
长度，包括使用符号<floor..ceiling>。什么时候
编码，实际长度在字节中的向量的内容之前
流。长度将以消耗尽可能多的数字的形式
字节，用于保存向量的指定最大值（ceiling）
长度。具有实际长度字段为零的可变长度向量
被称为空向量。

```
T T'<floor ...>
```

在以下示例中，mandatory是必须包含的向量
在300和400字节之间类型不透明。它永远不能为空。的
实际长度字段消耗两个字节，一个uint16，足够
表示值400（参见第4.4节）。另一方面，更长
可以表示高达800字节的数据，或400个uint16元素，并且它
可能为空。其编码将包括两个字节的实际长度
字段附加到向量。编码向量的长度必须
是单个元素的长度的偶数倍（例如，a
17字节的uint16向量将是非法的）。

```
不透明强制<300..400>;
      / * length字段为2个字节，不能为空* /
uint16 longer <0..800>;
      / *零到400个16位无符号整数* /
```

4.4。数字

基本数值数据类型是无符号字节（uint8）。所有更大
数字数据类型由固定长度的字节系列形成
如第4.1节所述连接，也是无符号的。的
以下数字类型是预定义的。

```
uint8 uint16 [2];
uint8 uint24 [3];
uint8 uint32 [4];
uint8 uint64 [8];
```

存储在此处和说明书中的其它地方的所有值
“网络”或“大端”顺序; uint32由hex表示
字节01 02 03 04相当于十进制值16909060。

4.5。枚举

额外的稀疏数据类型称为枚举。字段
类型枚举只能假定在定义中声明的值。
每个定义是不同的类型。只有枚举的相同
类型可以被分配或比较。枚举的每个元素必须

被分配一个值，如下面的例子所示。以来
枚举的元素不是有序的，它们可以分配
任何唯一值，以任何顺序。

```
枚举{e1（v1），e2（v2），...，en（vn）[[，（n）]]
```

枚举在字节流中占据与其相同的空间
最大定义的序数值。以下定义将导致
一个字节用于携带颜色类型的字段。

```
枚举{红（3），蓝（5），白（7）}颜色;
```

可以可选地指定没有其相关联的标签的值
强制定义宽度而不定义多余的元素。
在以下示例中，Taste将在数据中占用两个字节
流，但只能取值1,2或4。

```
enum {sweet（1），sour（2），bitter（4），（32000）}味道;
```

枚举元素的名称在范围内
定义类型。在第一个例子中，一个完全限定的引用
枚举的第二个元素将是Color.blue。这样
如果作业的目标良好，则不需要资格
指定。

```
颜色= Color.blue; / * overspecified，legal * /
颜色=蓝色; / *正确，类型隐式* /
```

对于从未转换为外部表示的枚举，
可以省略数值信息。

```
枚举{low, medium, high}金额;
```

4.6。构造类型

结构类型可以从基本类型构造
方便。每个规范声明一个新的，唯一的类型。的
定义的语法与C语言非常相似。

```
struct {
  T1 f1;
  T2 f2;
  ... ...
  Tn fn;
} [[T]];
```


结构中的字段可以使用类型的名称限定
使用类似于可用于枚举的语法。例如，
T.f2引用前一个声明的第二个字段。
可以嵌入结构定义。

4.6.1。变体

定义的结构可以具有基于一些知识的变体
在环境中可用。选择器必须是枚举的
类型，定义结构定义的可能变量。那里
必须是枚举中的每个元素的case arm
选择。变体结构的主体可以被给予标签
以供参考。选择变体的机制
运行时不由呈现语言规定。

```
struct {
    T1 f1;
    T2 f2;
    ....
    Tn fn;
    选择（E）{
        情况e1：Te1;
        情况e2：Te2;
        ....
        case en：Ten;
    } [[fv]];
} [[Tv]];
```

例如：

```
枚举{apple, orange} VariantTag;
struct {
    uint16 number;
    不透明线<0.10>; / *可变长度* /
} V1;
struct {
    uint32 number;
    不透明串[10]; / *固定长度* /
} V2;
struct {
```

```
            select (VariantTag) {/ *选择器的值是隐含的* /
                case apple: V1; / * VariantBody, tag = apple * /
                案例橙色: V2; / * VariantBody, tag = orange * /
            } variant_body; / *变量上的可选标签* /
        } VariantRecord;
```

可以通过指定值来限定（缩小）变体结构
对于类型之前的选择器。例如，a


```
        橙色VariantRecord
```

是包含variant_body的VariantRecord的缩小类型
类型V2。

4.7。加密属性

四种密码操作数字签名，流密码
加密，分组密码加密和公钥加密
指定的数字签名，流加密，块加密和
公共密钥加密。字段的加密
通过在前面添加适当的关键字来指定处理
指定字段类型规范之前。加密密钥
由当前会话状态隐含（参见第6.1节）。

在数字签名中，单向散列函数用作a的输入
签名算法。数字签名的元素被编码为不透明的
向量<0..2 ^ 16-1>，其中长度由签名指定
算法和密钥。

在RSA签名中，一个36字节的两个散列结构（一个SHA和一个
MD5）签名（使用私钥加密）。它用编码
PKCS＃1块类型0或类型1，如[PKCS1]中所述。

在DSS中，20个字节的SHA哈希直接通过
没有额外散列的数字签名算法。这产生
两个值，r和s。DSS签名是不透明向量，如上所述，
其内容是DER编码的：

```
    Dss-Sig-Value :: = SEQUENCE {
         r INTEGER,
         s INTEGER
    }}
```

在流密码加密中，明文与a进行异或运算
从密码安全产生的相同数量的输出
键控伪随机数发生器。

在块密码加密中，每个明文块加密到a
密文块。所有块密码加密在CBC中完成
（密码块链接）模式，以及块加密的所有项目
将是密码块长度的精确倍数。

在公钥加密中，使用公钥算法进行加密
数据，使得其可以仅用匹配来解密
私钥。公钥加密的元素被编码为不透明的

向量<0..2 ^ 16-1>，其中长度由签名指定
算法和密钥。

RSA加密值用PKCS＃1块类型2编码
描述在[PKCS1]中。

在以下示例中：

```
流加密struct {
    uint8 field1;
    uint8 field2;
    数字签名的不透明散列[20];
} UserType;
```

哈希的内容用作签名算法的输入，
那么整个结构用流密码加密。的
此结构的长度（以字节为单位）将等于2个字节
field1和field2，加上两个字节作为签名的长度，
加上签名算法的输出长度。这是已知的
由于用于签名的算法和密钥的事实
在编码或解码该结构之前已知。

4.8。常量

类型化常数可以定义为
声明所需类型的符号并为其分配值。
低于指定类型（不透明，可变长度向量和
包含opaque的结构）不能赋值。没有字段
可以省略多元件结构或矢量。

例如，

```
struct {
    uint8 f1;
    uint8 f2;
} Example1;

example1 ex1 = {1, 4}; / *赋值f1 = 1, f2 = 4 * /
```

5. HMAC和伪随机函数

需要在TLS记录和握手层中执行多个操作
键控MAC；这是由a保护的一些数据的安全摘要
秘密。伪造MAC是不可行的，没有MAC的知识
秘密。我们用于这个操作的结构被称为HMAC，
描述在[HMAC]中。

HMAC可以与各种不同的哈希算法一起使用。TLS
使用它在握手与两种不同的算法：MD5和SHA-
1，将它们表示为HMAC_MD5（秘密，数据）和HMAC_SHA（秘密，

数据）。额外的散列算法可以由密码套件和定义
用于保护记录数据，但MD5和SHA-1是硬编码的
该版本协议的握手的描述。

此外，需要一个结构来做扩展的秘密
成为用于密钥生成或验证目的的数据块。
该伪随机函数（PRF）将秘密，种子，
和识别标签，并产生任意长度的输出。

为了使PRF尽可能安全，它使用两个散列
算法，如果其中任何一个应该保证其安全性的方式
算法保持安全。

首先，我们定义一个数据扩展函数P_hash（secret，data）
其使用单个散列函数来将秘密和种子扩展到
任意输出量：

```
P_hash (secret, seed) = HMAC_hash (secret, A (1) + seed) +
                        HMAC_hash (secret, A (2) + seed) +
                        HMAC_hash (secret, A (3) + seed) + ...
```

其中+表示连接。

A（）定义为：

```
A (0) =种子
A (i) = HMAC_hash (secret, A (i-1) )
```

P_hash可以迭代多次，以产生必要的
所需数量的数据。例如，如果使用P_SHA-1
创建64字节的数据，它将被重复4次
（通过A（4）），创建80字节的输出数据；最后16个字节
的最后迭代将被丢弃，留下64字节
输出数据。

TLS的PRF是通过将秘密分成两半和来创建的
使用一半用P_MD5生成数据，另一半用
使用P_SHA-1生成数据，然后对输出进行异或运算
这两个扩展函数在一起。

S1和S2是秘密的两半，每个都是相同的
长度。S1取自秘密的前半部分，S2来自
下半场。它们的长度通过向上舍入的长度而产生
整体秘密除以二；因此，如果原始秘密是奇数
字节长度，S1的最后一个字节将与
S2的第一个字节。

```
L_S =以字节为单位的长度;
L_S1 = L_S2 = ceil (L_S / 2) ;
```


秘密被分成两半（有可能的
一个共享字节），如上所述，S1取第一个L_S1字节

S2为最后的L_S2字节。

然后将PRF定义为混合两个伪随机的结果
流"。

```
PRF (secret, label, seed) = P_MD5 (S1, label + seed) XOR
                            P_SHA-1 (S2, 标记+种子) ;
```

标签是ASCII字符串。它应该包括在确切的形式
它给出没有长度字节或尾随空字符。对于
例如，标签"slithy toves"将通过散列处理
以下字节：

```
73 6C 69 74 68 79 20 74 6F 76 65 73
```

注意，因为MD5产生16字节输出，SHA-1产生20
字节输出，它们的内部迭代的边界将不是
对齐; 生成一个80字节的输出将涉及P_MD5
通过A（5）迭代，而P_SHA-1将仅迭代通过A（4）。

6. TLS记录协议

TLS记录协议是分层协议。在每层，
消息可以包括用于长度，描述和内容的字段。
记录协议需要传输的消息，片段
数据转换为可管理块，可选地压缩数据
MAC，加密和发送结果。接收的数据为
解密，验证，解压缩和重组，然后传递到
更高级别的客户端。

本文档中描述了四个记录协议客户端：
握手协议，警报协议，更改密码规范
协议和应用数据协议。为了允许
扩展TLS协议，可以附加记录类型
支持记录协议。任何新的记录类型都应该
立即分配类型值超过ContentType值
这里描述的四种记录类型（见附录A.2）。如果是TLS
实现接收它不能理解的记录类型，它
应该忽略它。任何设计用于TLS的协议必须是
精心设计以处理所有可能的攻击。
请注意，因为记录的类型和长度不受保护
通过加密，应注意尽量减少流量的价值
分析这些值。

6.1。连接状态

TLS连接状态是TLS记录的操作环境
协议。它指定了压缩算法，加密算法，
和MAC算法。此外，这些算法的参数
是已知的：MAC秘密和批量加密密钥和IV
在读取和写入方向上的连接。逻辑上，
总有四个连接状态突出：当前读取

和写状态，以及挂起的读和写状态。所有记录
在当前读写状态下进行处理。安全
待处理状态的参数可以由TLS握手设置
协议和握手协议可以选择性地进行
待决状态电流，在这种情况下适当的电流
状态被处理并被替换为挂起状态； 待处理
状态然后被重新初始化为空状态。这是非法的
未使用安全参数a初始化的状态
当前状态。初始当前状态总是指定no
将使用加密，压缩或MAC。

TLS连接读取和写入状态的安全参数为
通过提供以下值设置：

连接端
这个实体是否被认为是“客户端”或“服务器”
这个连接。

批量加密算法
用于批量加密的算法。本规范
包括此算法的密钥大小，该密钥的多少
秘密，无论是块还是流密码，块大小
密码（如果适用），以及是否被认为是
“export”密码。

MAC算法
用于消息认证的算法。这个
规范包括返回的散列的大小
MAC算法。

压缩算法
用于数据压缩的算法。本规范
必须包括算法需要做的所有信息
压缩。

主秘密
在连接中的两个对等体之间共享的48字节秘密。

客户端随机
客户端提供的32字节值。

服务器随机
服务器提供的32字节值。

这些参数在呈现语言中定义为：

```
枚举{server，client} ConnectionEnd;

enum {null，rc4，rc2，des，3des，des40} BulkCipherAlgorithm;

枚举{流，块}密码类型;

枚举{true，false} IsExportable;

enum {null，md5，sha} MACAlgorithm;
```

```
        枚举{null（0），（255）} CompressionMethod;

        / * CompressionMethod中指定的算法，
           BulkCipherAlgorithm和MACAlgorithm。* /

        struct {
            连接实体;
            BulkCipherAlgorithm bulk_cipher_algorithm;
            CipherType cipher_type;
            uint8 key_size;
            uint8 key_material_length;
            isExportable is_exportable;
            MACAlgorithm mac_algorithm;
            uint8 hash_size;
            CompressionMethod compression_algorithm;
            opaque master_secret [48];
            opaque client_random [32];
            opaque server_random [32];
        } SecurityParameters;
```

记录层将使用安全参数来生成
以下六项：

客户端写MAC地址
服务器写MAC密钥
客户端写密钥
服务器写密钥
客户端写入IV（仅用于分组密码）
服务器写入IV（仅用于分组密码）

客户机写入参数由服务器在接收和使用时使用
处理记录，反之亦然。用于生成的算法
这些项目从安全参数描述在第6.3节。

一旦安全参数已设置并且密钥已经设置
生成，连接状态可以通过使它们实例化
当前状态。必须为每个状态更新这些当前状态
记录处理。每个连接状态包括以下
元素：

压缩状态
压缩算法的当前状态。

密码状态
加密算法的当前状态。这将包括
的连接的计划密钥。另外，对于块
以CBC模式运行的密码（为TLS指定的唯一模式），
这将首先包含用于该连接状态的IV
更新为包含最后一个块的密文加密
或者作为记录被解密。对于流密码，这
将包含任何必要的状态信息是允许的
该流继续加密或解密数据。

MAC秘密
以上生成的此连接的MAC密钥。

序列号
每个连接状态包含一个序列号，即
分别保持读和写状态。序列
每当建立连接状态时，数字必须设置为零
活动状态。序列号的类型为uint64，可能不是
超过2 ^ 64-1。序列号在每个之后递增
record：具体来说，第一条记录是根据传输的
特定连接状态应使用序列号0。

6.2。记录层

TLS记录层从更高层接收未解释的数据
在任意大小的非空块中。

6.2.1。碎片

记录层将信息块分割成TLSPlaintext
记录以2 ^ 14字节或更少的块存储数据。客户端消息
边界不被保留在记录层中（即，多个
相同ContentType的客户端消息可以合并为a
单个TLSP明文记录，或者单个消息可以被分段
跨越几个记录）。

```
struct {
    uint8 major, minor;
} ProtocolVersion;
```



```
枚举{
    change_cipher_spec（20），alert（21），handshake（22），
    application_data（23），（255）
} 内容类型;

struct {
    ContentType类型;
    协议版本;
    uint16 length;
    不透明片段[TLSPlaintext.length];
} TLSPlaintext;
```

类型
用于处理封闭片段的较高级协议。

版
正在使用的协议版本。这个文件
描述了使用版本{3,1}的TLS版本1.0。的
版本值3.1是历史：TLS版本1.0是次要的
修改承载版本的SSL 3.0协议
值3.0。（见附录A.1）。

长度
以下TLSPlaintext.fragment的长度（以字节为单位）。
长度不应超过2 ^ 14。

分段

应用程序数据。此数据是透明的，并被视为
独立块由更高级协议处理
由类型字段指定。

注意：不同TLS记录层内容类型的数据可能不同
交织。应用程序数据通常具有较低的优先级
用于传输比其他内容类型。

6.2.2。记录压缩和解压缩

所有记录都使用中定义的压缩算法进行压缩
当前会话状态。总是有一个主动压缩
算法；然而，最初它被定义为
CompressionMethod.null。压缩算法翻译a
TLSPlaintext结构转换为TLSCompressed结构。压缩
函数使用默认状态信息初始化
连接状态被激活。

压缩必须是无损的，并且不能增加内容长度
超过1024字节。如果解压缩函数遇到a
TLSCompressed.fragment将解压缩到超过的长度
2 ^ 14字节，它应该报告致命的解压缩失败错误。

```
struct {
    ContentType类型; / *与TLSPlaintext.type相同* /
    ProtocolVersion版本; / *与TLSPlaintext.version * /
    uint16 length;
    opaque fragment [TLSCompressed.length];
} TLSCompressed;
```

长度
以下TLSCompressed.fragment的长度（以字节为单位）。
长度不应超过2 ^ 14 + 1024。

分段
压缩形式的TLSPlaintext.fragment。

注意：CompressionMethod.null操作是身份操作；没有
字段被改变。

实施说明：
减压功能负责确保
消息不能导致内部缓冲区溢出。

6.2.3。记录负载保护

加密和MAC功能翻译TLSCompressed结构
转换为TLSCiphertext。解密函数逆过程。
记录的MAC还包括序列号
丢失，额外或重复的消息是可检测的。

```
struct {
```

```
            ContentType类型;
            协议版本;
            uint16 length;
            选择（CipherSpec.cipher_type）{
                case流：GenericStreamCipher;
                case块：GenericBlockCipher;
            } fragment;
        } TLSCiphertext;
```

类型
    类型字段与TLSCompressed.type相同。

版
    版本字段与TLSCompressed.version相同。

长度
    以下TLSCiphertext.fragment的长度（以字节为单位）。
    长度不能超过2 ^ 14 + 2048。

分段
    TLSCompressed.fragment的加密形式，带有MAC。

6.2.3.1。空或标准流密码

流密码（包括BulkCipherAlgorithm.null - 见附录
A.6）将TLSCompressed.fragment结构转换成流和从流中转换
TLSCiphertext.fragment结构。

```
    流加密struct {
        opaque content [TLSCompressed.length];
        opaque MAC [CipherSpec.hash_size];
    } GenericStreamCipher;
```

MAC生成为：

```
    HMAC_hash（MAC_write_secret，seq_num + TLSCompressed.type +
                  TLSCompressed.version + TLSCompressed.length +
                  TLSCompressed.fragment））；
```

其中“+”表示连接。

seq_num
    此记录的序列号。

哈希
    指定的散列算法
    SecurityParameters.mac_algorithm。

注意，在加密之前计算MAC。流密码
加密整个块，包括MAC。对于流密码
不要使用同步向量（如RC4），流密码
状态从一个记录的结尾简单地在后续使用
包。如果CipherSuite是TLS_NULL_WITH_NULL_NULL，加密
包括身份操作（即，数据未加密
并且MAC大小为0，意味着不使用MAC）。
TLSCiphertext.length是TLSCompressed.length加

CipherSpec.hash_size。

6.2.3.2。CBC块密码

对于分组密码（例如RC2或DES），加密和MAC
函数将TLSCompressed.fragment结构转换为块
TLSCiphertext.fragment结构。


```
        块加密的struct {
            opaque content [TLSCompressed.length];
            opaque MAC [CipherSpec.hash_size];
            uint8 padding [GenericBlockCipher.padding_length];
            uint8 padding_length;
        } GenericBlockCipher;
```

MAC按第6.2.3.1节中所述生成。

填充
添加的填充，以强制明文的长度
是分组密码的块长度的整数倍。的
填充可以是长达255字节的任何长度，只要i t
导致TLSCiphertext.length是的整数倍
块长度。长度超过必要长度可能
希望阻止对基于分析的协议的攻击
交换消息的长度。每个uint8在填充数据
向量必须填充填充长度值。

padding_length
填充长度应该使得总大小
GenericBlockCipher结构是密码块的倍数
长度。合法值的范围从0到255（含）。这个
length指定填充字段的长度，不包括
padding_length字段本身。

加密数据长度（TLSCiphertext.length）是一个
sum of TLSCompressed.length，CipherSpec.hash_size和
padding_length。

示例：如果块长度为8字节，则为内容长度
（TLSCompressed.length）为61字节，MAC长度为20
字节，填充前的长度为82字节。就这样
填充长度模8必须等于6才能生成
总长度为8个字节的偶数倍（块
长度）。填充长度可以是6，14，22，等等，
如果填充长度是必要的最小长度，
6，填充将是6字节，每个包含值6。
因此，在Block之前的GenericBlockCipher的最后8个字节
加密将是xx 06 06 06 06 06 06 06，其中xx是
MAC的最后一个八位字节。

注意：使用CBC模式（密码块链接）的分组密码
生成第一记录的初始化向量（IV）
其他密钥和秘密，当安全参数设置。
后续记录的IV是最后一个密文块
上一条记录。

6.3。键计算

记录协议需要一个算法来生成密钥，IV和
MAC秘密从握手提供的安全参数
协议。

主秘密被散列成安全字节序列，其中
分配给所需的MAC密钥，密钥和非导出IV
当前连接状态（见附录A.6）。CipherSpecs要求
客户端写MAC密钥，服务器写MAC密钥，客户端写
密钥，服务器写入密钥，客户端写入IV和服务器写入IV，
它们是以该顺序从主秘密生成的。没用过
值为空。

当生成密钥和MAC秘密时，主秘密用作
熵源，并且随机值提供未加密的盐
材料和IVs出口密码。

要生成密钥材料，请进行计算

```
key_block = PRF (SecurityParameters.master_secret,
                 “密钥扩展”,
                 SecurityParameters.server_random +
                 SecurityParameters.client_random) ;
```

直到产生足够的输出。然后key_block是
分区如下：

```
client_write_MAC_secret [SecurityParameters.hash_size]
server_write_MAC_secret [SecurityParameters.hash_size]
client_write_key [SecurityParameters.key_material_length]
server_write_key [SecurityParameters.key_material_length]
client_write_IV [SecurityParameters.IV_size]
server_write_IV [SecurityParameters.IV_size]
```

client_write_IV和server_write_IV仅针对非易失性存储器
导出块密码。对于可导出的块密码，
稍后生成初始化向量，如下所述。任何
额外的key_block资料被丢弃。

实施说明：
在本文档中定义的密码规范要求
最多的材料是3DES_EDE_CBC_SHA：它需要2 x 24字节
键，2×20字节的MAC秘密和2×8字节的IV，总共
104字节的密钥材料。

可导出的加密算法（为其提供CipherSpec.is_exportable
是真的）需要额外的处理如下以导出它们
最后写密钥：

```
final_client_write_key =
PRF (SecurityParameters.client_write_key,
                            “客户端写密钥”,
                            SecurityParameters.client_random +
                            SecurityParameters.server_random) ;
final_server_write_key =
PRF (SecurityParameters.server_write_key,
                            “server write key”,
                            SecurityParameters.client_random +
                            SecurityParameters.server_random) ;
```

可导出加密算法仅从其中导出其IV
来自hello消息的随机值：

```
iv_block = PRF (“”, “IV block”, SecurityParameters.client_random +
               SecurityParameters.server_random) ;
```

iv_block被划分为两个初始化向量
key_block在上面：

```
client_write_IV [SecurityParameters.IV_size]
server_write_IV [SecurityParameters.IV_size]
```

注意，在这种情况下使用PRF而没有秘密：这只是
意味着秘密具有零字节的长度并且贡献
没有什么到PRF中的散列。

6.3.1。导出密钥生成示例

TLS_RSA_EXPORT_WITH_RC2_CBC_40_MD5需要五个随机字节
每个MAC都有两个加密密钥和16个字节
键，共42字节的密钥材料。PRF输出为
存储在key_block中。key_block是分区的，并且写
键是盐化的，因为这是可导出的加密算法。

```
key_block = PRF (master_secret,
                                “密钥扩展”,
                                server_random +
                                client_random) [0..41]
client_write_MAC_secret = key_block [0..15]
server_write_MAC_secret = key_block [16..31]
client_write_key = key_block [32..36]
server_write_key = key_block [37..41]
```



```
final_client_write_key = PRF (client_write_key,
                                “客户端写密钥”,
                                client_random +
                                server_random) [0..15]
final_server_write_key = PRF (server_write_key,
                                “server write key”,
```

```
                                    client_random +
                                    server_random) [0..15]

        iv_block = PRF ("", "IV block", client_random +
                                    server_random) [0..15]
        client_write_IV = iv_block [0..7]
        server_write_IV = iv_block [8..15]
```

7. TLS握手协议

TLS握手协议由三套子协议组成
其用于允许对等体同意安全参数
记录层，验证自身，实例化协商
安全参数和报告错误条件。

握手协议负责协商会话，
其中包括以下项目：

会话标识符
由服务器选择的任意字节序列以标识
活动或恢复会话状态。

对等证书
X509v3 [X509]证书。这个元素的状态
可以为null。

压缩方法
用于在加密之前压缩数据的算法。

密码规范
指定批量数据加密算法（如null，DES，
等等）和MAC算法（例如MD5或SHA）。它也定义
加密属性，如hash_size。（见附录A.6
用于形式定义）

主秘密
客户端和服务器之间共享的48字节秘密。

是可恢复的
指示会话是否可用于发起新的标志
连接。

这些项目然后用于创建供使用的安全参数
保护应用程序数据时的记录层。许多连接
可以使用相同的会话通过恢复来实例化
TLS握手协议的特性。

7.1。更改密码规范协议

改变密码规范协议存在以用于信号转换
加密策略。协议由单个消息组成，
其在当前（不是待决）下被加密和压缩，
连接状态。消息由值为1的单个字节组成。

```
        struct {
            枚举{change_cipher_spec (1) , (255) }类型;
        } ChangeCipherSpec;
```

更改密码规范消息由客户端和服务器发送
以通知接收方随后的记录将是
受新协商的CipherSpec和密钥的保护。接待
该消息使接收器指示记录层
立即将读取暂挂状态复制到读取当前状态。
发送此消息后，发送方应立即指示
使记录层使写暂挂状态为写激活
州。（参见6.1节）发送change cipher spec消息
在安全参数同意后的握手期间
，但在发送验证完成消息之前（见第
7.4.9）。

7.2。警报协议

TLS记录层支持的内容类型之一是
警报类型。警报消息传达消息的严重性和a
警报的描述。具有致命结果级别的警报消息
在立即终止连接。在这种情况下，其他
对应于会话的连接可以继续，但是
会话标识符必须无效，防止失败的会话
从被用于建立新的连接。像其他消息，
警报消息被加密和压缩，如由
当前连接状态。

```
        enum {warning (1) , fatal (2) , (255) } AlertLevel;

        枚举{
            close_notify (0) ,
            unexpected_message (10) ,
            bad_record_mac (20) ,
            decrypt_failed (21) ,
            record_overflow (22) ,
```



```
            decompression_failure (30) ,
            handshake_failure (40) ,
            bad_certificate (42) ,
            unsupported_certificate (43) ,
            certificate_revoked (44) ,
            certificate_expired (45) ,
            certificate_unknown (46) ,
            illegal_parameter (47) ,
            unknown_ca (48) ,
            access_denied (49) ,
            decode_error (50) ,
            decrypt_error (51) ,
            export_restriction (60) ,
            protocol_version (70) ,
            insufficient_security (71) ,
            internal_error (80) ,
            user_canceled (90) ,
```

```
            no_renegotiation (100),
             (255)
        } AlertDescription;

        struct {
            AlertLevel级别;
            AlertDescription描述;
        }警报;
```

7.2.1。关闭警报

客户端和服务器必须共享连接的知识
结束以避免截断攻击。任何一方都可以
发起关闭消息的交换。

close_notify
此消息通知收件人发件人不会发送
此连接上的任何其他消息。会话成为
不可恢复如果任何连接终止没有适当
close_notify级别等于warning的消息。

任一方可以通过发送close_notify警报来发起关闭。
忽略关闭警报后接收的任何数据。

每个参与方必须在关闭前发送close_notify警报
写端的连接。它要求另一方
响应自己的close_notify警报，关闭
连接，丢弃任何挂起的写入。不是这样
需要发起者的close来等待响应
close_notify在关闭连接的读取端之前的警报。

如果使用TLS的应用协议规定任何数据可能
在TLS连接之后承载底层传输
关闭，TLS实现必须接收响应
close_notify在向应用层指示之前的警报
TLS连接已结束。如果应用协议不会
传输任何额外的数据，但只会关闭底层
传输连接，那么实现可以选择关闭
传输无需等待响应close_notify。没有部分
本标准应被视为规定使用的方式
配置文件为TLS管理其数据传输，包括何时
连接打开或关闭。

注意：假设关闭连接可靠地传递
等待数据，然后破坏传输。

7.2.2。错误警报

TLS握手协议中的错误处理非常简单。当一个
检测到错误，检测方向另一方发送消息
派对。在发送或接收致命警报消息时，两者
各方立即关闭连接。服务器和客户端
需要忘记任何会话标识符，密钥和秘密
与失败的连接相关联。以下错误警报

定义：

unexpected_message
收到不适当的讯息。此警报总是致命的
并且不应该在适当之间的沟通中观察到
实现。

bad_record_mac
如果接收到的记录不正确，将返回此警报
苹果电脑。这个消息总是致命的。

decrypt_failed
以无效方式解密的TLSCiphertext：或者它不是
偶数倍的块长度或其填充值
检查，不正确。这个消息总是致命的。

record_overflow
接收到一个长度大于的TLSCiphertext记录
2 ^ 14 + 2048字节，或者解密为TLSCompressed记录的记录
与超过2 ^ 14 + 1024字节。这个消息总是致命的。

decompression_failure
解压缩函数接收到不正确的输入（例如数据
将扩大到过长的长度）。此消息始终是
致命。

handshake_failure
handshake_failure警报消息的接收指示
发件人无法协商可接受的一组安全性
参数给出了可用的选项。这是一个致命错误。

bad_certificate
证书已损坏，包含未签名的签名
验证正确等。

unsupported_certificate
证书类型不受支持。

certificate_revoked
证书已被其签名者撤销。

certificate_expired
证书已过期或当前无效。

certificate_unknown
一些其他（未指定）问题出现在处理
证书，使其不可接受。

illegal_parameter
握手中的字段超出范围或不一致
其他领域。这总是致命的。

unknown_ca
接收到有效的证书链或部分链，但是
证书不被接受，因为CA证书不能
被定位或不能与已知的，可信的CA匹配。这个
消息总是致命的。

拒绝访问
已接收到有效的证书，但在访问控制时
应用，发送方决定不继续协商。
这个消息总是致命的。

decode_error
无法解码邮件，因为某些字段已超出
指定的范围或消息的长度不正确。这个
消息总是致命的。

decrypt_error
握手加密操作失败，包括失败
无法正确验证签名，解密密钥交换，
或验证完成的消息。

export_restriction
不符合出口限制的谈判是
检测；例如，尝试传输1024位
RSA_EXPORT握手方法的临时RSA密钥。这个
消息总是致命的。

protocol_version
客户端尝试协商的协议版本是
认可，但不支持。（例如，旧协议
版本可能由于安全原因而被避免）。此消息是
总是致命的。

insufficient_security
当协商已经返回而不是handshake_failure
失败，因为服务器需要更多的密码
安全比客户端支持的那些。此消息始终是
致命。

内部错误
与对等体无关的内部错误或正确性
协议使得它不可能继续（如内存
分配失败）。这个消息总是致命的。

user_canceled
此握手正在被取消由于某种原因与a不相关
协议故障。如果用户取消后的操作
握手是完成的，只是通过发送一个关闭连接
close_notify更为合适。应遵循此警报
通过close_notify。此消息通常是一个警告。

no_renegotiation
由客户端发送以响应hello请求或由
服务器响应在初始握手之后的客户端问候。
这两种情况通常都会导致重新谈判；当时
是不适当的，收件人应该用此警报响应；
在这一点上，原始请求者可以决定是否
继续进行连接。一种情况是这将是

适当的将是一个服务器已经产生了一个进程
满足请求；该进程可能会收到安全参数
（密钥长度，身份验证等），它可能是
难以沟通这些参数后的变化
点。此消息始终是警告。

对于未显式指定警报级别的所有错误，
发送方可以自行决定这是否是致命的
错误或不；如果接收到具有警告级别的警报，则

接收方可自行决定是否将其视为
一个致命错误或不。但是，所有发送的消息
具有致命的水平必须被视为致命的消息。

7.3。握手协议概述

会话状态的密码参数由
TLS握手协议，它在TLS记录之上操作
层。当TLS客户端和服务器首先开始通信时，它们
同意协议版本，选择加密算法，
可选地彼此认证，并使用公钥加密
技术来生成共享秘密。

TLS握手协议包括以下步骤：

- 交换hello消息以同意算法，交换随机
  值，并检查会话恢复。

- 交换必要的加密参数以允许
  客户端和服务器来商定一个预制的秘密。

- 交换证书和加密信息以允许
  客户端和服务器进行身份验证。

- 从主机秘密生成主秘密并交换
  随机值。

- 向记录层提供安全参数。

- 允许客户端和服务器验证他们的对等体
  计算相同的安全参数和握手
  发生时没有被攻击者篡改。

请注意，更高层不应该总是过分依赖TLS
协商两个对等体之间的最强可能的连接：
有一个中间人攻击者可以尝试的方式
使两个实体下拉到最不安全的方法
支持。该协议已被设计为最小化这种风险，但是
仍然有可用的攻击：例如，攻击者可以
阻止对运行安全服务的端口的访问，或尝试获取
对等体协商未认证的连接。基本
规则是更高的水平必须认识到他们的安全
要求是并且从不通过信道发送信息

安全比他们所需要的。TLS协议是安全的，在其中
任何加密套件提供它承诺的安全级别：如果你
协商3DES与一个主机的1024位RSA密钥交换
证书你已经验证，你可以期望是安全的。

但是，你不应该通过用40位加密的链接发送数据
安全，除非你觉得数据不值得努力
需要打破加密。

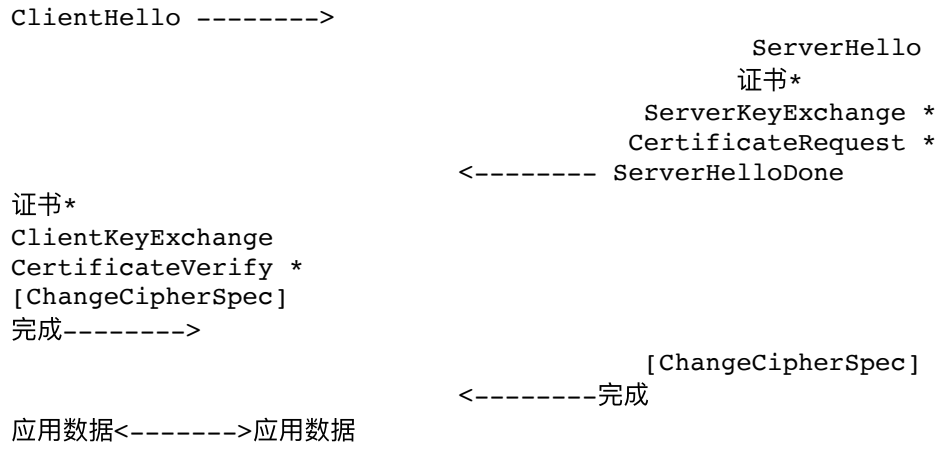
这些目标通过握手协议来实现，这可以是
总结如下：客户端发送客户端hello消息
服务器必须使用服务器hello消息响应，否则a
将发生致命错误，并且连接将失败。客户端hello
和服务器问候用于建立安全增强
客户端和服务器之间的功能。客户端hello和服务器
hello建立以下属性：协议版本，会话
ID，密码套件和压缩方法。另外，两个随机
生成并交换值：ClientHello.random和
ServerHello.random。

实际的密钥交换最多使用四条消息：服务器
证书，服务器密钥交换，客户端证书和
客户端密钥交换。新的密钥交换方法可以通过创建
指定这些消息的格式并定义使用
消息以允许客户端和服务器在共享上达成一致
秘密。这个秘密应该很长；当前定义的键
交换方法交换从48到128字节的秘密
长度。

按照hello消息，服务器将发送其证书，
如果它要被认证。此外，服务器密钥交换
消息可以被发送，如果它是必需的（例如，如果他们的服务器没有
证书，或者其证书仅用于签名）。如果
服务器进行身份验证，它可以从请求证书
客户端，如果这适合于选择的密码套件。现在
服务器会发送服务器hello done消息，指示
握手的hello消息阶段完成。服务器将
然后等待客户端响应。如果服务器已发送证书
请求消息，客户端必须发送证书消息。的
客户端密钥交换消息现在发送，并且其内容
消息将取决于在之间选择的公钥算法
客户端hello和服务器hello。如果客户端发送了一个
具有签名能力的证书，数字签名的证书
发送验证消息以显式验证证书。

此时，客户端发送改变密码规范消息，
并且客户端将待决密码规范拷贝到当前密码中
规格。然后客户端立即发送完成的消息
新的算法，密钥和秘密。作为响应，服务器将
发送自己的更改密码规范消息，将挂起的传输到
当前Cipher Spec，并将其完成的消息发送给新的

密码规范。此时，握手完成并且客户端
并且服务器可以开始交换应用层数据。（见流程
以下图表。）



图。1 - 完全握手的消息流

*表示不是可选的或情境相关的消息
总是发送。

注意：为了帮助避免流水线停顿，ChangeCipherSpec是一个
独立的TLS协议内容类型，并且实际上不是TLS
握手消息。

当客户端和服务器决定恢复先前的会话或
复制现有会话（而不是协商新的安全性
参数）消息流如下：

客户端使用会话的会话ID发送ClientHello
恢复。服务器然后检查其会话缓存的匹配。如果
找到匹配，并且服务器愿意重新建立
连接在指定的会话状态下，它会发送一个
ServerHello具有相同的会话ID值。在这一点上，两者
客户端和服务器必须发送更改密码规范消息并继续
直接完成消息。一旦重新建立完成，
客户端和服务器可以开始交换应用层数据。
（见下面的流程图。）如果没有找到会话ID匹配，则
服务器生成新的会话ID和TLS客户端和服务器
执行完全握手。

客户端服务器

```
ClientHello -------->
                                                    ServerHello
                                                [ChangeCipherSpec]
                                    <--------完成
[ChangeCipherSpec]
完成-------->
应用数据<------->应用数据
```

图。2 – 缩写握手的消息流

每个消息的内容和意义将在
详细信息在以下部分。

7.4。握手协议

TLS握手协议是定义的更高级别客户端之一
的TLS记录协议。这个协议用于协商
安全的会话属性。提供握手消息
TLS记录层，它们被封装在一个或多个内
TLSPlaintext结构，其被处理和传输
由当前活动会话状态指定。

```
枚举{
    hello_request（0），client_hello（1），server_hello（2），
    证书（11），server_key_exchange（12），
    certificate_request（13），server_hello_done（14），
    certificate_verify（15），client_key_exchange（16），
    完成（20），（255）
} HandshakeType;

struct {
    HandshakeType msg_type; / *握手类型* /
    uint24 length; / *消息中的字节* /
    select（HandshakeType）{
        case hello_request：HelloRequest;
        case client_hello：ClientHello;
        case server_hello：ServerHello;
        案例证书：证书;
        case server_key_exchange：ServerKeyExchange;
        case certificate_request：CertificateRequest;
        case server_hello_done：ServerHelloDone;
        case certificate_verify：CertificateVerify;
        case client_key_exchange：ClientKeyExchange;
        箱完成：完成;
    } 身体;
}握手;
```


握手协议消息按照它们的顺序在下面呈现
必须发送；以意外的顺序发送握手消息
导致致命错误。不需要的握手消息可以省略，
然而。注意排序的一个例外：证书消息
在握手中使用两次（从服务器到客户端，然后从
客户端到服务器），但只描述在其第一个位置。唯一的那个

消息，这些消息不受Hello中的这些排序规则的约束
请求消息，可以随时发送，但应该是
如果客户端在握手过程中到达，则被客户端忽略。

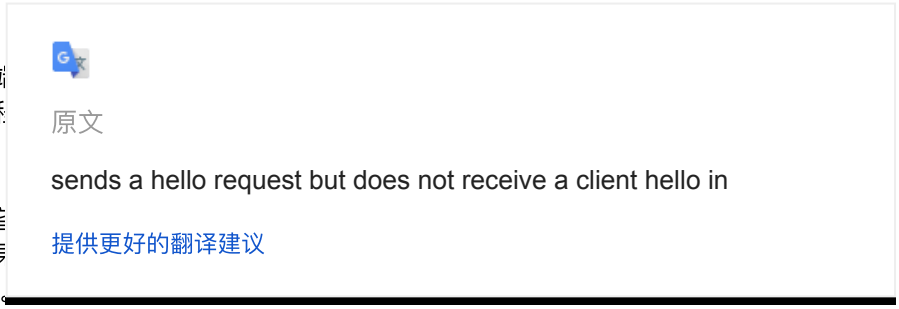
7.4.1。你好消息

hello phase消息用于交换安全增强
客户端和服务器之间的功能。当一个新的会话
开始，记录层的连接状态加密，哈希和
压缩算法初始化为null。电流
连接状态用于重新协商消息。

7.4.1.1。您好请求

当此消息将被发送时：
hello请求消息可以由服务器在任何时间发送。

此消息的含义：
Hello请求是一个简单的通知，客户端
通过发送客户端问候重新开始协商过程
消息方便。此消息将被忽略
如果客户端当前正在协商会话。这个
消息可以被客户端忽略，如果它不希望
重新协商会话，或者客户端可以，如果
with no_renegotiation alert。
旨在具有高于应用数据的传输优先级，
预计谈判将在不再开始之前开始
比从客户端接收几个记录。如果服务器
发送hello请求，但不接收客户端hello
响应，它可能会关闭与致命警报的连接。



发送hello请求后，服务器不应重复请求
直到后续握手协商完成。

此消息的结构：

```
struct {} HelloRequest;
```

注意：此消息不应包含在消息散列中
在整个握手中被保持并在完成中使用
消息和证书验证消息。

7.4.1.2。客户端问候

当此消息将被发送时：
当客户端首次连接到服务器时，需要发送
客户端hello作为其第一条消息。客户端也可以发送
客户端hello响应hello请求或自己
主动为了重新协商安全参数
现有连接。

此消息的结构：
客户端问候消息包括随机结构
稍后在协议中使用。

```
            struct {
               uint32 gmt_unix_time;
               opaque random_bytes [28];
            }随机;
```

gmt_unix_time
标准UNIX 32位格式的当前时间和日期（秒
自从1970年1月1日起，午夜起），根据
发送方的内部时钟。不需要设置时钟
由基本TLS协议正确；更高水平或应用
协议可以定义附加要求。

random_bytes
28字节由安全随机数生成器生成。

客户端问候消息包括可变长度会话
标识符。如果不为空，该值标识了之间的会话
同一客户端和服务器的客户端希望的安全参数
重用。会话标识符可以来自较早的连接，这是
连接或另一个当前活动的连接。第二个选项
如果客户端仅希望更新随机结构，那么它是有用的
和连接的派生值，而第三个选项
可能建立几个独立的安全连接
重复完整的握手协议。这些独立的连接
可以顺序或同时发生；SessionID变为有效
当握手谈判完成与交换
完成的邮件并持续存在，直到由于老化或因为被删除
在与之关联的连接上遇到致命错误
会话。SessionID的实际内容由定义
服务器。

```
opaque SessionID <0..32>;
```


警告：
因为SessionID是在没有加密的情况下传输的
立即MAC保护，服务器不得保密
信息在会话标识符或让假的内容
会话标识符导致任何安全漏洞。（注意
握手的内容作为一个整体，包括SessionID在内
受到在结束时交换的完成消息的保护
握手。）

CipherSuite列表，从客户端传递到服务器中的
客户端hello消息，包含加密的组合
客户端按照客户端的顺序支持的算法
偏好（首选最喜爱的选择）。每个CipherSuite定义一个键
交换算法，批量加密算法（包括密钥）
长度）和MAC算法。服务器将选择一个加密套件
或者，如果没有可接受的选择，则返回握手
故障警报并关闭连接。

```
        uint8 CipherSuite [2]; / *加密套件选择器* /
```

客户端问候包括支持的压缩算法的列表
由客户端根据客户端的偏好排序。

```
        枚举{null（0），（255）} CompressionMethod;

        struct {
            ProtocolVersion client_version;
            随机随机;
            SessionID session_id;
            CipherSuite cipher_suites <2..2 ^ 16-1>;
            CompressionMethod compression_methods <1..2 ^ 8-1>;
        } ClientHello;
```

client_version
客户端希望的TLS协议版本
在此会话期间进行通信。这应该是最新的
（最高价值）版本。为了这
版本的规范，版本将是3.1（见
有关向后兼容性的详细信息，请参见附录E.

随机
客户端生成的随机结构。

session_id
客户端希望用于此连接的会话的ID。
如果没有session_id可用，则此字段应为空
客户端希望生成新的安全参数。

密码
这是由支持的加密选项的列表
客户端，具有客户端的第一偏好。如果
session_id字段不为空（意味着会话恢复
请求）这个向量必须至少包括cipher_suite从
那个会话。值在附录A.5中定义。

compression_methods
这是由支持的压缩方法的列表
客户端，按客户端首选项排序。如果session_id字段是
不为空（意味着会话恢复请求），它必须包括
来自该会话的compression_method。这个向量必须
包含，并且所有实现必须支持，
CompressionMethod.null。因此，客户端和服务器将始终是
能够同意压缩方法。

发送客户端问候消息后，客户端等待服务器
你好消息。服务器返回的任何其他握手消息
除了hello请求被视为致命错误。

前向兼容性说明：
为了向前兼容性，允许a
客户端hello消息以在压缩之后包括额外的数据
方法。此数据必须包含在握手哈希中，但是
否则必须忽略。这是唯一的握手消息

这是合法的；对于所有其他消息，数据量
在消息中必须匹配消息的描述
恰恰。

7.4.1.3。服务器问候

当此消息将被发送时：
服务器将发送此消息以响应客户端问候
消息，当它能够找到一组可接受的算法。
如果它不能找到这样的匹配，它将响应与握手
故障警报。

此消息的结构：

```
struct {
    ProtocolVersion server_version;
    随机随机;
    SessionID session_id;
    CipherSuite cipher_suite;
    CompressionMethod compression_method;
} ServerHello;
```


server_version
此字段将包含客户端建议的较低值
在客户端hello和最高的由服务器支持。对于
这个版本的规范，版本是3.1（见
有关向后兼容性的详细信息，请参见附录E.

随机
此结构由服务器生成，并且必须不同
从（和独立）ClientHello.random。

session_id
这是与此相对应的会话的标识
连接。如果ClientHello.session_id不为空，则
服务器将在其会话缓存中查找匹配。如果匹配
发现并且服务器愿意建立新的连接
使用指定的会话状态，服务器将响应
与客户端提供的值相同。这表示a
恢复会议，并要求各方必须进行
直接到完成的消息。否则此字段将
包含标识新会话的不同值。服务器
可以返回空的session_id来指示会话将会
不被缓存，因此不能被恢复。如果会话是
恢复，它必须使用相同的密码套件恢复
最初协商。

cipher_suite
单个加密套件由服务器从中的列表中选择
ClientHello.cipher_suites。对于恢复会话，此字段为
值从会话的状态恢复。

compression_method
单个压缩算法由服务器选择

ClientHello.compression_methods中的列表。续会
此字段是来自恢复的会话状态的值。

7.4.2。服务器证书

当此消息将被发送时：
服务器必须在约定的密钥时发送证书
交换方法不是匿名的。此消息将始终
立即跟随服务器hello消息。

此消息的含义：
证书类型必须适合所选的密码
套件的密钥交换算法，一般是X.509v3
证书。它必须包含与密钥交换匹配的密钥
法，如下。除非另有规定，签字

证书的算法必须与算法相同
为证书密钥。除非另有规定，公众
键可以具有任何长度。

密钥交换算法证书密钥类型

RSA RSA公钥；证书必须
允许该密钥用于加密。

RSA_EXPORT RSA公钥的长度大于
512位，可用于签名，
或512位或更短的密钥
可用于加密或
签署。

DHE_DSS DSS公钥。

DHE_DSS_EXPORT DSS公钥。

DHE_RSA RSA公钥可用于
签署。

DHE_RSA_EXPORT可用于的RSA公钥
签署。

DH_DSS Diffie-Hellman密钥。使用的算法
签署证书应该是DSS。

DH_RSA Diffie-Hellman密钥。使用的算法
签署证书应该是RSA。

定义所有证书配置文件，密钥和加密格式
由IETF PKIX工作组[PKIX]。当密钥使用扩展名为
当前，必须设置digitalSignature位为该键
有资格进行签名，如上所述，以及keyEncipherment位
必须存在以允许加密，如上所述。的
必须在Diffie-Hellman证书上设置keyAgreement位。

指定指定新的密钥交换方法的密文套件
对于TLS协议，它们将暗示证书格式和
需要的编码键控信息。

此消息的结构：

```
不透明ASN.1Cert <1..2 ^ 24-1>;

struct {
    ASN.1Cert certificate_list <0..2 ^ 24-1>;
} 证书;
```


certificate_list
这是X.509v3证书的序列（链）。发件人
证书必须在列表中排在第一位。每个以下
证书必须直接证明其前面的证书。因为
证书验证需要分发根密钥
独立的，自签名证书指定
根证书颁发机构可以可选地从中省略
链，假设远端必须已经
拥有它，以便在任何情况下验证它。

相同的消息类型和结构将用于客户端
响应证书请求消息。注意客户端可以
如果没有适当的证书，则不发送证书
以响应服务器的认证请求而发送。

注意：PKCS＃7 [PKCS7]不用作证书的格式
向量，因为PKCS＃6 [PKCS6]扩展证书不是
用过的。此外，PKCS＃7定义了一个SET，而不是SEQUENCE
解析列表的任务更加困难。

7.4.3。服务器密钥交换消息

当此消息将被发送时：
此消息将在服务器之后立即发送
证书消息（或服务器hello消息，如果这是一个
匿名协商）。

服务器密钥交换消息仅在服务器发送
服务器证书消息（如果发送）不够包含
数据，以允许客户端交换premaster秘密。这是
适用于以下密钥交换方法：

- RSA_EXPORT（如果服务器证书中的公钥是
  长于512位）
- DHE_DSS
- DHE_DSS_EXPORT
- DHE_RSA
- DHE_RSA_EXPORT
- DH_anon

发送服务器密钥交换消息是不合法的
以下密钥交换方法：

- RSA
- RSA_EXPORT（当服务器证书中的公钥为

小于或等于512位长度）
DH_DSS
DH_RSA

此消息的含义：
此消息传达加密信息以允许
客户端沟通的前兆秘密：要么是RSA公共
密钥，用Diffie-Hellman加密预置密钥
公钥，客户端可以使用该密钥完成密钥交换
（结果是主人的秘密）。

由于为包括新密钥的TLS定义了附加的CipherSuites
交换算法，服务器密钥交换消息将被发送如果
并且只有当与密钥交换相关联的证书类型
算法不能为客户端提供足够的信息
交换一个预备秘密。

注：根据美国现行出口法，RSA模量大于512
位不能用于从软件导出的密钥交换
美国。使用此消息，较大的RSA密钥编码
证书可以用于签署临时较短的RSA密钥
RSA_EXPORT密钥交换方法。

此消息的结构：

```
枚举{rsa, diffie_hellman} KeyExchangeAlgorithm;

struct {
    opaque rsa_modulus <1..2 ^ 16-1>;
    opaque rsa_exponent <1..2 ^ 16-1>;
} ServerRSAParams;
```

rsa_modulus
服务器临时RSA密钥的模数。

rsa_exponent
服务器的临时RSA密钥的公共指数。

```
struct {
    不透明dh_p <1..2 ^ 16-1>;
    不透明dh_g <1..2 ^ 16-1>;
    不透明dh_Ys <1..2 ^ 16-1>;
} ServerDHParams; / *短暂DH参数* /
```

dh_p
用于Diffie-Hellman操作的素数模量。

dh_g
用于Diffie-Hellman操作的发生器。

dh_Ys
服务器的Diffie-Hellman公共值（g ^ X模p）。

```
struct {
    select (KeyExchangeAlgorithm) {
        case diffie_hellman:
            ServerDHParams params;
            签名signed_params;
        case rsa:
            ServerRSAParams params;
            签名signed_params;
    };
} ServerKeyExchange;
```

参数
服务器的密钥交换参数。

signed_params
对于非匿名密钥交换，对应的哈希值
params值，其中签名适合于该哈希
应用。

md5_hash
MD5 (ClientHello.random + ServerHello.random + ServerParams) ;

sha_hash
SHA (ClientHello.random + ServerHello.random + ServerParams) ;

enum {anonymous, rsa, dsa}签名算法;

选择 (SignatureAlgorithm)
{case anonymous: struct {};
case rsa:
数字签名struct {
opaque md5_hash [16];
opaque sha_hash [20];
};
case dsa:
数字签名struct {
opaque sha_hash [20];
};
}签名;

7.4.4。证书请求

当此消息将被发送时:
非匿名服务器可以选择请求证书
客户端，如果适用于所选的密码套件。这个
消息，如果发送，将立即跟随服务器密钥交换
消息（如果它被发送;否则，服务器证书
信息）。

此消息的结构:
枚举{
rsa_sign (1) , dss_sign (2) , rsa_fixed_dh (3) , dss_fixed_dh (4) ,
(255)

```
        } ClientCertificateType;

        opaque DistinguishedName <1..2 ^ 16-1>;

        struct {
            ClientCertificateType certificate_types <1..2 ^ 8-1>;
            DistinguishedName certificate_authorities <3..2 ^ 16-1>;
        } CertificateRequest;
```

certificate_types
此字段是请求的证书类型的列表，
按服务器的首选项顺序排序。

certificate_authorities
可接受证书的可分辨名称的列表
当局。这些可分辨名称可以指定期望的名称
根CA或下级CA的可分辨名称；
因此，该消息可以用于描述已知的根
和期望的授权空间。

注意：DistinguishedName派生自[X509]。

注意：这是一个致命的handshake_failure警报，匿名服务器
请求客户端标识。

7.4.5。服务器hello完成

当此消息将被发送时：
服务器hello done消息由服务器发送以指示
服务器端的hello和关联的消息。后
发送此消息，服务器将等待客户端响应。

此消息的含义：
此消息表示服务器完成发送消息
支持密钥交换，并且客户端可以继续其
阶段的密钥交换。

在收到服务器问候消息后，客户端应该
请验证服务器是否提供了有效的证书（如果需要）
并检查服务器问候参数是否可接受。

此消息的结构：

```
struct {} ServerHelloDone;
```


7.4.6。客户端证书

当此消息将被发送时：
这是客户端在接收到之后可以发送的第一条消息
服务器问候消息。此消息仅在
服务器请求证书。如果没有合适的证书
可用，客户端应发送证书消息
不包含证书。如果需要客户端认证
由服务器为握手继续，它可能响应
一个致命的握手失败警报。将发送客户端证书
使用第7.4.2节中定义的证书结构。

注意：当使用基于Diffie-Hellman的静态密钥交换方法时
（DH_DSS或DH_RSA），如果请求客户端认证，
Diffie-Hellman组和生成器编码在客户端
证书必须匹配服务器指定的Diffie-Hellman
参数，如果客户端的参数要用于密钥
交换。

7.4.7。客户端密钥交换消息

当此消息将被发送时：
此消息始终由客户端发送。它会立即
按照客户端证书消息，如果它被发送。除此以外
它将是客户端收到后发送的第一条消息
服务器hello完成消息。

此消息的含义：
有了这个消息，预置秘密被设置，不管
直接传输RSA加密的秘密，或由
传输Diffie-Hellman参数，这将允许每个
侧面同意同一个前辈的秘密。当钥匙
交换方法是DH_RSA或DH_DSS，客户认证有
被请求，并且客户端能够响应
证书，其包含Diffie-Hellman公钥
参数（组和生成器）匹配那些指定的
服务器在其证书中，此消息将不包含任何内容
数据。

此消息的结构：
消息的选择取决于哪个密钥交换方法具有
已选择。KeyExchangeAlgorithm参见7.4.3节
定义。

```
struct {
    select (KeyExchangeAlgorithm) {
        case rsa: EncryptedPreMasterSecret;
        case diffie_hellman: ClientDiffieHellmanPublic;
```



```
    } exchange_keys;
} ClientKeyExchange;
```

7.4.7.1。RSA加密预先保密消息

此消息的含义：
如果RSA用于密钥协商和认证，
客户端生成一个48字节的premaster秘密，使用加密它
从服务器的证书或临时RSA的公钥
密钥提供在服务器密钥交换消息中，并发送
导致加密的前置机密消息。这种结构
是客户端密钥交换消息的一种变体，而不是消息
本身。

此消息的结构：

```
struct {
    ProtocolVersion client_version;
```

不透明随机[46];
} PreMasterSecret;

client_version
客户端支持的最新（最新）版本。这是
用于检测版本回滚攻击。收到
premaster秘密，服务器应该检查这个值
匹配客户端中客户端发送的值
你好消息。

随机
46个安全生成的随机字节。

struct {
public-key-encrypted PreMasterSecret pre_master_secret;
} EncryptedPreMasterSecret;

注意：可以使用由Daniel Bleichenbacher [BLEI]发现的攻击
攻击使用PKCS＃1编码的RSA的TLS服务器。的
攻击利用了不同的失败的事实
方式，TLS服务器可以强制显示是否
特定消息，当解密时，正确地PKCS＃1格式化
或不。

避免这种攻击的最佳方法是治疗
格式不正确的邮件无法区分
正确格式化的RSA块。因此，当它接收到
格式不正确的RSA块，服务器应该生成一个
随机的48字节值并继续使用它作为预制
秘密。因此，服务器将作用相同
所接收的RSA块是否被正确编码。

pre_master_secret
这个随机值由客户端生成并用于
生成主秘密，如第8.1节中所述。

7.4.7.2。客户端Diffie-Hellman公共值

此消息的含义：
这种结构传达客户的Diffie-Hellman公共价值
（Yc）（如果它尚未包括在客户端的证书中）。
用于Yc的编码由枚举确定
PublicValueEncoding。这个结构是客户端的一个变体
密钥交换消息，而不是消息本身。

此消息的结构：
枚举{implicit，explicit} PublicValueEncoding;

隐含
如果客户端证书已经包含合适的
Diffie-Hellman密钥，那么Yc是隐式的，不需要
再次发送。在这种情况下，客户端密钥交换消息
将被发送，但将为空。

显式

Yc需要发送。

```
struct {
    select (PublicValueEncoding) {
        case implicit: struct {};
        case explicit: opaque dh_Yc <1..2 ^ 16-1>;
    } dh_public;
} ClientDiffieHellmanPublic;

dh_Yc
    客户端的Diffie-Hellman公共值（Yc）。
```

7.4.8。证书验证

当此消息将被发送时：
此消息用于提供客户端的显式验证
证书。此消息仅在客户端后发送
具有签名能力的证书（即所有证书
除了包含固定Diffie-Hellman参数的那些）。什么时候
发送，它会立即跟随客户端密钥交换消息。

此消息的结构：

```
struct {
     签名签名;
} CertificateVerify;
```


签名类型在7.4.3中定义。

```
CertificateVerify.signature.md5_hash
    MD5 (handshake_messages) ;

Certificate.signature.sha_hash
    SHA (handshake_messages) ;
```

这里handshake_messages是指发送的所有握手消息
收到从客户端开始直到但不包括这个
消息，包括握手的类型和长度字段
消息。这是所有握手结构的级联
如7.4中所定义的。

7.4.9。完成

当此消息将被发送时：
完成的消息总是在更改后立即发送
密码规范消息来验证密钥交换和
认证过程成功。至关重要的是
在另一方之间接收改变密码规范消息
握手消息和完成消息。

此消息的含义：
完成的消息是第一个受保护的，
协商的算法，密钥和秘密。收件人完成
消息必须验证内容是否正确。一旦一面
已发送其完成消息并接收并验证
来自其对等体的完成消息，它可以开始发送和接收
应用程序数据。

```
        struct {
            opaque verify_data [12];
        }完成;

        verify_data
            PRF (master_secret, finished_label, MD5 (handshake_messages) +
            SHA-1 (handshake_messages) ) [0..11];

        finished_label
            对于由客户端发送的完成消息，字符串“client
            完成“。对于由服务器发送的完成消息
            字符串“服务器完成”。

        handshake_messages
            来自所有握手消息的所有数据，直到但不是
            包括此消息。这只是数据可见
            握手层，并且不包括记录层标头。
```



```
            这是所有握手结构的连接
            定义在7.4交换到目前为止。
```

如果完成的消息前面没有更改，那么这是一个致命错误
密码规范消息在握手中的适当点。

服务器发送的完成消息中包含的散列
合并Sender.server; 那些由客户端发送的
Sender.client。值handshake_messages包括所有握手
从客户端hello开始，但不包括此消息的消息
完成消息。这可能不同于handshake_messages中的
第7.4.8节，因为它将包括证书验证消息
（如果发送）。此外，完成的消息的handshake_messages发送
由客户端将不同于完成的消息
由服务器发送，因为第二个发送的包括
前一个。

注意：更改密码规范消息，警报和任何其他记录类型
不是握手消息，并且不包括在哈希中
计算。此外，Hello请求消息从中省略
握手哈希。

8.加密计算

为了开始连接保护，TLS记录协议
需要规范一套算法，一个主秘密，和
客户端和服务器随机值。认证，加密，
并且MAC算法由所选择的cipher_suite确定
服务器并在服务器hello消息中显示。压缩
算法 协商的Hello报文，随机值
在hello消息中交换。所有剩下的就是计算
主秘密。

8.1。计算主密钥

对于所有密钥交换方法，使用相同的算法进行转换

将pre_master_secret转换成master_secret。pre_master_secret
应该从内存中删除一次master_secret已经
计算。

```
master_secret = PRF (pre_master_secret, “master secret”,
                     ClientHello.random + ServerHello.random)
[0..47];
```

主秘密的长度总是正好为48个字节。长度
根据密钥交换方法，预置密钥将有所不同。

8.1.1。RSA

当RSA用于服务器认证和密钥交换时，
byte pre_master_secret由客户端生成，加密下
服务器的公钥，并发送到服务器。服务器使用它
私钥解密pre_master_secret。双方然后
将pre_master_secret转换为master_secret，如指定的
以上。

RSA数字签名使用PKCS＃1 [PKCS1]块类型执行
1.使用PKCS＃1块类型2执行RSA公钥加密。

8.1.2。Diffie-Hellman

执行常规的Diffie-Hellman计算。的
协商密钥（Z）用作pre_master_secret，并且被转换
进入master_secret，如上所述。

注意：Diffie-Hellman参数由服务器指定，并且可能
可以是短暂的，也可以包含在服务器的证书中。

9.强制密码套件

在没有应用配置文件标准规定的情况下
否则，TLS兼容应用程序必须实现密码
套件TLS_DHE_DSS_WITH_3DES_EDE_CBC_SHA。

10.应用数据协议

应用程序数据消息由记录层承载
碎片化，压缩和基于当前连接加密
州。消息被视为记录的透明数据
层。

协议常数值

本节介绍协议类型和常量。

A.1。记录层

```
struct {
    uint8 major, minor;
} ProtocolVersion;

ProtocolVersion version = {3,1}; / * TLS v1.0 * /

枚举{
    change_cipher_spec (20) , alert (21) , handshake (22) ,
    application_data (23) ,  (255)
} 内容类型;

struct {
    ContentType类型;
    协议版本;
    uint16 length;
    不透明片段[TLSPlaintext.length];
} TLSPlaintext;

struct {
    ContentType类型;
    协议版本;
    uint16 length;
    opaque fragment [TLSCompressed.length];
} TLSCompressed;

struct {
    ContentType类型;
    协议版本;
    uint16 length;
    选择 (CipherSpec.cipher_type) {
        case流：GenericStreamCipher;
        case块：GenericBlockCipher;
    } fragment;
} TLSCiphertext;

流加密struct {
    opaque content [TLSCompressed.length];
    opaque MAC [CipherSpec.hash_size];
} GenericStreamCipher;

块加密的struct {
    opaque content [TLSCompressed.length];
```

```
        opaque MAC [CipherSpec.hash_size];
        uint8 padding [GenericBlockCipher.padding_length];
        uint8 padding_length;
    } GenericBlockCipher;
```

A2。更改密码规范消息

```
    struct {
        枚举{change_cipher_spec (1) , (255) }类型;
    } ChangeCipherSpec;
```

A.3。警报消息

```
    enum {warning (1) , fatal (2) , (255) } AlertLevel;

        枚举{
            close_notify (0) ,
            unexpected_message (10) ,
            bad_record_mac (20) ,
            decrypt_failed (21) ,
            record_overflow (22) ,
            decompression_failure (30) ,
            handshake_failure (40) ,
            bad_certificate (42) ,
            unsupported_certificate (43) ,
            certificate_revoked (44) ,
            certificate_expired (45) ,
            certificate_unknown (46) ,
            illegal_parameter (47) ,
            unknown_ca (48) ,
            access_denied (49) ,
            decode_error (50) ,
            decrypt_error (51) ,
            export_restriction (60) ,
            protocol_version (70) ,
            insufficient_security (71) ,
            internal_error (80) ,
            user_canceled (90) ,
            no_renegotiation (100) ,
             (255)
        } AlertDescription;

    struct {
        AlertLevel级别;
        AlertDescription描述;
    }警报;
```


A.4。握手协议

```
    枚举{
        hello_request (0) , client_hello (1) , server_hello (2) ,
```

```
        证书 (11), server_key_exchange (12),
        certificate_request (13), server_hello_done (14),
        certificate_verify (15), client_key_exchange (16),
        完成 (20), (255)
    } HandshakeType;

    struct {
        HandshakeType msg_type;
        uint24 length;
        select (HandshakeType) {
            case hello_request: HelloRequest;
            case client_hello: ClientHello;
            case server_hello: ServerHello;
            案例证书: 证书;
            case server_key_exchange: ServerKeyExchange;
            case certificate_request: CertificateRequest;
            case server_hello_done: ServerHelloDone;
            case certificate_verify: CertificateVerify;
            case client_key_exchange: ClientKeyExchange;
            箱完成: 完成;
        } 身体;
    }握手;
```

A.4.1。你好消息

```
    struct {} HelloRequest;

    struct {
        uint32 gmt_unix_time;
        opaque random_bytes [28];
    }随机;

    opaque SessionID <0..32>;

    uint8 CipherSuite [2];

    枚举{null (0), (255) } CompressionMethod;

    struct {
        ProtocolVersion client_version;
        随机随机;
        SessionID session_id;
        CipherSuite cipher_suites <2..2 ^ 16-1>;
        CompressionMethod compression_methods <1..2 ^ 8-1>;
```



```
    } ClientHello;

    struct {
        ProtocolVersion server_version;
        随机随机;
        SessionID session_id;
        CipherSuite cipher_suite;
        CompressionMethod compression_method;
    } ServerHello;
```

A.4.2。服务器认证和密钥交换消息

```
    不透明ASN.1Cert <2 ^ 24-1>;
```

```
    struct {
        ASN.1Cert certificate_list <1..2 ^ 24-1>;
    } 证书;

    枚举{rsa, diffie_hellman} KeyExchangeAlgorithm;

    struct {
        opaque RSA_modulus <1..2 ^ 16-1>;
        opaque RSA_exponent <1..2 ^ 16-1>;
    } ServerRSAParams;

    struct {
        不透明DH_p <1..2 ^ 16-1>;
        不透明DH_g <1..2 ^ 16-1>;
        不透明DH_Ys <1..2 ^ 16-1>;
    } ServerDHParams;

    struct {
        select (KeyExchangeAlgorithm) {
            case diffie_hellman:
                ServerDHParams params;
                签名signed_params;
            case rsa:
                ServerRSAParams params;
                签名signed_params;
        };
    } ServerKeyExchange;

    enum {anonymous, rsa, dsa}签名算法;

    选择 (SignatureAlgorithm)
    {case anonymous: struct {};
        case rsa:
            数字签名struct {
```



```
                opaque md5_hash [16];
                opaque sha_hash [20];
            };
        case dsa:
            数字签名struct {
                opaque sha_hash [20];
            };
    }签名;

    枚举{
        rsa_sign (1) , dss_sign (2) , rsa_fixed_dh (3) , dss_fixed_dh (4) ,
         (255)
    } ClientCertificateType;

    opaque DistinguishedName <1..2 ^ 16-1>;

    struct {
        ClientCertificateType certificate_types <1..2 ^ 8-1>;
        DistinguishedName certificate_authorities <3..2 ^ 16-1>;
    } CertificateRequest;

    struct {} ServerHelloDone;
```

A.4.3。客户端认证和密钥交换消息

```
    struct {
        select (KeyExchangeAlgorithm) {
            case rsa: EncryptedPreMasterSecret;
            case diffie_hellman: DiffieHellmanClientPublicValue;
        } exchange_keys;
    } ClientKeyExchange;

    struct {
        ProtocolVersion client_version;
        不透明随机[46];

    } PreMasterSecret;

    struct {
        public-key-encrypted PreMasterSecret pre_master_secret;
    } EncryptedPreMasterSecret;

    枚举{implicit, explicit} PublicValueEncoding;

    struct {
        select (PublicValueEncoding) {
            case implicit: struct {};
            case explicit: opaque DH_Yc <1..2 ^ 16-1>;
```



```
        } dh_public;
    } ClientDiffieHellmanPublic;

    struct {
        签名签名;
    } CertificateVerify;
```

A.4.4。握手完成消息

```
    struct {
        opaque verify_data [12];
    }完成;
```

A.5。密码套件

以下值定义客户端中使用的CipherSuite代码
hello和服务器hello消息。

CipherSuite定义TLS版本中支持的密码规范
1.0。

指定了TLS_NULL_WITH_NULL_NULL，它是a的初始状态
TLS连接在该通道上的第一次握手，但必须
不能谈判，因为它不提供更多的保护
不安全的连接。

```
 CipherSuite TLS_NULL_WITH_NULL_NULL = {0x00,0x00};
```

以下CipherSuite定义要求服务器提供
可用于密钥交换的RSA证书。服务器可以
请求RSA或具有DSS签名功能的证书
证书请求消息。

```
    CipherSuite TLS_RSA_WITH_NULL_MD5 = {0x00,0x01};
    CipherSuite TLS_RSA_WITH_NULL_SHA = {0x00,0x02};
    CipherSuite TLS_RSA_EXPORT_WITH_RC4_40_MD5 = {0x00,0x03};
    CipherSuite TLS_RSA_WITH_RC4_128_MD5 = {0x00,0x04};
    CipherSuite TLS_RSA_WITH_RC4_128_SHA = {0x00,0x05};
    CipherSuite TLS_RSA_EXPORT_WITH_RC2_CBC_40_MD5 = {0x00,0x06};
    CipherSuite TLS_RSA_WITH_IDEA_CBC_SHA = {0x00,0x07};
    CipherSuite TLS_RSA_EXPORT_WITH_DES40_CBC_SHA = {0x00,0x08};
    CipherSuite TLS_RSA_WITH_DES_CBC_SHA = {0x00,0x09};
    CipherSuite TLS_RSA_WITH_3DES_EDE_CBC_SHA = {0x00,0x0A};
```

以下CipherSuite定义用于服务器 -
认证（并且可选地客户端认证）Diffie-Hellman。
DH表示服务器证书包含的加密套件
由认证中心签名的Diffie-Hellman参数

（CA）。DHE表示短暂的Diffie-Hellman，其中Diffie-Hellman
参数由DSS或RSA证书签名，这已经过
由CA签名。使用的签名算法在后面指定
DH或DHE参数。服务器可以请求RSA或DSS签名 -
从客户端进行客户端认证或它
可以请求Diffie-Hellman证书。任何Diffie-Hellman
证书由客户端提供必须使用参数（组和
发电机）。

```
    CipherSuite TLS_DH_DSS_EXPORT_WITH_DES40_CBC_SHA = {0x00,0x0B};
    CipherSuite TLS_DH_DSS_WITH_DES_CBC_SHA = {0x00,0x0C};
    CipherSuite TLS_DH_DSS_WITH_3DES_EDE_CBC_SHA = {0x00,0x0D};
    CipherSuite TLS_DH_RSA_EXPORT_WITH_DES40_CBC_SHA = {0x00,0x0E};
    CipherSuite TLS_DH_RSA_WITH_DES_CBC_SHA = {0x00,0x0F};
    CipherSuite TLS_DH_RSA_WITH_3DES_EDE_CBC_SHA = {0x00,0x10};
    CipherSuite TLS_DHE_DSS_EXPORT_WITH_DES40_CBC_SHA = {0x00,0x11};
    CipherSuite TLS_DHE_DSS_WITH_DES_CBC_SHA = {0x00,0x12};
    CipherSuite TLS_DHE_DSS_WITH_3DES_EDE_CBC_SHA = {0x00,0x13};
    CipherSuite TLS_DHE_RSA_EXPORT_WITH_DES40_CBC_SHA = {0x00,0x14};
    CipherSuite TLS_DHE_RSA_WITH_DES_CBC_SHA = {0x00,0x15};
    CipherSuite TLS_DHE_RSA_WITH_3DES_EDE_CBC_SHA = {0x00,0x16};
```

以下密码套件用于完全匿名
Diffie-Hellman通信，其中任一方都不是
验证。请注意，此模式容易受到中间人攻击
攻击，因此被弃用。

```
    CipherSuite TLS_DH_anon_EXPORT_WITH_RC4_40_MD5 = {0x00,0x17};
    CipherSuite TLS_DH_anon_WITH_RC4_128_MD5 = {0x00,0x18};
    CipherSuite TLS_DH_anon_EXPORT_WITH_DES40_CBC_SHA = {0x00,0x19};
    CipherSuite TLS_DH_anon_WITH_DES_CBC_SHA = {0x00,0x1A};
    CipherSuite TLS_DH_anon_WITH_3DES_EDE_CBC_SHA = {0x00,0x1B};
```

注意：考虑第一个字节为0xFF的所有密码套件
私有并且可以用于定义本地/实验
算法。这种类型的互操作性是一个局部问题。

注意：可以通过发布RFC注册其他密码套件
其指定密码套件，包括必要的TLS
协议信息，包括消息编码，预制
秘密推导，对称加密和MAC计算

适当的参考信息。
RFC编辑的办公室可以自行决定发布
规格的密码套件，不完全
描述（例如，对于分类算法）
规范具有技术兴趣和完全
指定。

注意：密码套件值{0x00，0x1C}和{0x00，0x1D}是
保留以避免与基于Fortezza的密码套件冲突
SSL 3。

A.6。安全参数

这些安全参数由TLS握手确定
协议，并作为参数提供给TLS记录层
以初始化连接状态。SecurityParameters包括：

```
枚举{null（0），（255）} CompressionMethod;

枚举{server，client} ConnectionEnd;

enum {null，rc4，rc2，des，3des，des40，idea}
BulkCipherAlgorithm;

枚举{流，块}密码类型;

枚举{true，false} IsExportable;

enum {null，md5，sha} MACAlgorithm;

/ * CompressionMethod中指定的算法，
BulkCipherAlgorithm和MACAlgorithm。* /

struct {
    连接实体;
    BulkCipherAlgorithm bulk_cipher_algorithm;
    CipherType cipher_type;
    uint8 key_size;
    uint8 key_material_length;
    isExportable is_exportable;
    MACAlgorithm mac_algorithm;
    uint8 hash_size;
    CompressionMethod compression_algorithm;
    opaque master_secret [48];
    opaque client_random [32];
    opaque server_random [32];
} SecurityParameters;
```

B.词汇表

应用协议
应用协议是通常分层的协议
直接在传输层（例如，TCP / IP）之上。例子
包括HTTP，TELNET，FTP和SMTP。

非对称密码
请参阅公共密钥加密。

认证
认证是一个实体确定的能力
另一个实体的身份。

块密码
块密码是对明文进行操作的算法
称为块的位组。64位是公共块大小。

批量密码
用于加密大量数据的对称加密算法
数据的。

密码块链接（CBC）
CBC是其中用a加密的每个明文块的模式
块密码首先与前一密文进行异或运算
块（或，在第一块的情况下，与
初始化向量）。对于解密，每个块是第一个
解密，然后与前一密文块进行异或运算
（或IV）。

证书
作为X.509协议（也称为ISO认证）的一部分
框架），证书由受信任的证书分配
权威和提供一个强大的约束力的一方的身份
或一些其他属性及其公钥。

客户
启动与a的TLS连接的应用程序实体
服务器。这可能或可能不意味着客户端发起了
底层传输连接。主要操作
服务器和客户端之间的区别是服务器是
一般认证，而客户端只是可选的
验证。

客户端写密钥
用于加密客户端写入的数据的密钥。

客户端写MAC地址
用于验证客户端写入的数据的秘密数据。

连接
连接是传输（在OSI分层模型中
定义），提供合适类型的服务。对于TLS，
这种连接是对等关系。连接
是暂时的。每个连接都与一个会话相关联。

数据加密标准
DES是一种非常广泛使用的对称加密算法。DES是
具有56位密钥和8字节块大小的块密码。注意
在TLS中，为了密钥生成的目的，DES被视为
具有8字节的密钥长度（64位），但它仍然只提供
56位保护。（假设每个关键字节的低位
设置为在该关键字节中产生奇数奇偶校验。）DES也可以
以三个独立键和三个键的模式操作
加密用于每个数据块；这使用168位
的密钥（24个字节的TLS密钥生成方法）并提供
相当于112位的安全性。[DES]，[3DES]

数字签名标准（DSS）
数字签名的标准，包括数字签名
算法，由国家标准研究所和
技术，定义在NIST FIPS PUB 186，“数字签名
标准”，1994年5月由美国商务部出版。
[DSS]

数字签名
数字签名使用公钥加密和单向
散列函数产生可以是数据的签名
认证，并且难以伪造或拒绝。

握手
客户端和服务器之间的初始协商建立
他们的事务的参数。

初始化向量（IV）
当在CBC模式下使用分组密码时，初始化
向量与之前的第一明文块进行异或运算
加密。

理念
由Xuejia Lai和James Massey设计的64位块密码。
[理念]

消息认证码（MAC）
消息认证码是从a计算的单向散列
消息和一些秘密数据。这是很难伪造没有
知道秘密数据。其目的是检测消息
已更改。

主秘密
用于生成加密密钥（MAC）的安全机密数据
秘密和IV。

MD5
MD5是一种可以任意转换的安全散列函数
长数据流转换为固定大小（16字节）的摘要。[MD5]

公钥密码术
一种使用双密钥密码的密码技术。
使用公钥加密的邮件只能使用解密
相关联的私钥。相反，用
私钥可以用公钥进行验证。

单向散列函数
一种单向变换，可以转换任意数量的
数据转换成固定长度的哈希值。它在计算上很难
逆转换或找到碰撞。MD5和SHA是
单向散列函数的示例。

RC2
由Ron Rivest在RSA Data Security，Inc。开发的块密码
[RC2]中描述的[RSADSI]。

RC4
由RSA数据安全[RSADSI]许可的流加密。一个
兼容密码在[RC4]中描述。

RSA
一种非常广泛使用的公钥算法可用于
无论是加密还是数字签名。[RSA]

盐
用于使导出加密密钥抵抗的非秘密随机数据
预计算攻击。

服务器
服务器是响应请求的应用程序实体
用于来自客户端的连接。另见客户端下。

会话
TLS会话是客户端和服务器之间的关联。
会话由握手协议创建。会话定义a
一组加密安全参数，可以共享
在多个连接中。会话用于避免
昂贵的谈判新的安全参数为每个
连接。

会话标识符
会话标识符是由服务器生成的值
标识特定会话。

服务器写密钥
用于加密由服务器写入的数据的密钥。

服务器写MAC密钥
用于验证服务器写入的数据的秘密数据。

SHA
安全散列算法在FIPS PUB 180-1中定义。它
产生20字节输出。注意，所有引用SHA
实际上使用修改后的SHA-1算法。[SHA]

SSL
Netscape的安全套接字层协议[SSL3]。TLS是基于
SSL版本3.0

流密码
一种将密钥转换为a的加密算法
密码强的密钥流，然后进行异或运算
与明文。

对称密码
请参阅批量密码。

传输层安全（TLS）
这个协议；还有，传输层安全工作组
的互联网工程任务组（IETF）。请参阅“评论”
本文结束。

C. CipherSuite定义

CipherSuite是密钥哈希

	可导出交易所
TLS_NULL_WITH_NULL_NULL	* NULL NULL
TLS_RSA_WITH_NULL_MD5	* RSA NULL MD5
TLS_RSA_WITH_NULL_SHA	* RSA NULL SHA
TLS_RSA_EXPORT_WITH_RC4_40_MD5	* RSA_EXPORT RC4_40 MD5
TLS_RSA_WITH_RC4_128_MD5	RSA RC4_128 MD5
TLS_RSA_WITH_RC4_128_SHA	RSA RC4_128 SHA
TLS_RSA_EXPORT_WITH_RC2_CBC_40_MD5	* RSA_EXPORT RC2_CBC_40 MD5
TLS_RSA_WITH_IDEA_CBC_SHA	RSA IDEA_CBC SHA
TLS_RSA_EXPORT_WITH_DES40_CBC_SHA	* RSA_EXPORT DES40_CBC SHA
TLS_RSA_WITH_DES_CBC_SHA	RSA DES_CBC SHA
TLS_RSA_WITH_3DES_EDE_CBC_SHA	RSA 3DES_EDE_CBC SHA
TLS_DH_DSS_EXPORT_WITH_DES40_CBC_SHA	* DH_DSS_EXPORT DES40_CBC SHA
TLS_DH_DSS_WITH_DES_CBC_SHA	DH_DSS DES_CBC SHA
TLS_DH_DSS_WITH_3DES_EDE_CBC_SHA	DH_DSS 3DES_EDE_CBC SHA
TLS_DH_RSA_EXPORT_WITH_DES40_CBC_SHA	* DH_RSA_EXPORT DES40_CBC SHA

```
TLS_DH_RSA_WITH_DES_CBC_SHA DH_RSA DES_CBC SHA
TLS_DH_RSA_WITH_3DES_EDE_CBC_SHA DH_RSA 3DES_EDE_CBC SHA
TLS_DHE_DSS_EXPORT_WITH_DES40_CBC_SHA * DHE_DSS_EXPORT DES40_CBC SHA
TLS_DHE_DSS_WITH_DES_CBC_SHA DHE_DSS DES_CBC SHA
TLS_DHE_DSS_WITH_3DES_EDE_CBC_SHA DHE_DSS 3DES_EDE_CBC SHA
TLS_DHE_RSA_EXPORT_WITH_DES40_CBC_SHA * DHE_RSA_EXPORT DES40_CBC SHA
TLS_DHE_RSA_WITH_DES_CBC_SHA DHE_RSA DES_CBC SHA
TLS_DHE_RSA_WITH_3DES_EDE_CBC_SHA DHE_RSA 3DES_EDE_CBC SHA
TLS_DH_anon_EXPORT_WITH_RC4_40_MD5 * DH_anon_EXPORT RC4_40 MD5
TLS_DH_anon_WITH_RC4_128_MD5 DH_anon RC4_128 MD5
TLS_DH_anon_EXPORT_WITH_DES40_CBC_SHA DH_anon DES40_CBC SHA
TLS_DH_anon_WITH_DES_CBC_SHA DH_anon DES_CBC SHA
TLS_DH_anon_WITH_3DES_EDE_CBC_SHA DH_anon 3DES_EDE_CBC SHA
```

*表示IsExportable为True

键
交换
算法描述密钥大小限制

DHE_DSS具有DSS签名的临时DH无
DHE_DSS_EXPORT具有DSS签名DH = 512位的临时DH
DHE_RSA临时DH与RSA签名无
DHE_RSA_EXPORT具有RSA签名的临时DH DH = 512位,
RSA =无
DH_anon匿名DH，无签名无
DH_anon_EXPORT匿名DH，无签名DH = 512位

DH_DSS DH具有基于DSS的证书无
DH_DSS_EXPORT DH与基于DSS的证书DH = 512位
DH_RSA DH具有基于RSA的证书无
DH_RSA_EXPORT DH具有基于RSA的证书DH = 512位,
RSA =无
NULL无密钥交换N / A
RSA RSA密钥交换无
RSA_EXPORT RSA密钥交换RSA = 512位

键大小限制
密钥大小限制给出了最大公钥的大小
可以合法地用于加密在密码套件中
可出口。

扩展有效IV块
密码类型材料密钥材料密钥位大小大小

NULL *流0 0 0 0 N / A
IDEA_CBC块16 16 128 8 8
RC2_CBC_40 *块5 16 40 8 8
RC4_40 *流5 16 40 0 N / A
RC4_128流16 16 128 0 N / A
DES40_CBC * Block 5 8 40 8 8
DES_CBC Block 8 8 56 8 8
3DES_EDE_CBC Block 24 24 168 8 8

*表示IsExportable为true。

类型
指示这是流密码还是块密码
运行在CBC模式。

密钥材料
来自用于的key_block的字节数
生成写密钥。

扩展密钥材料
实际馈入加密算法的字节数

有效密钥位
在进料的关键材料中有多少熵材料
加密程序。

IV尺寸
需要为初始化生成多少数据
向量。零为流密码; 等于块的大小
块密码。

块大小
块密码在一个块中加密的数据量; 一个
块密码运行在CBC模式下只能加密一个偶数
其块大小的倍数。

哈希哈希填充 功能尺寸大小
NULL 0 0
MD5 16 48
SHA 20 40

D.实施说明

TLS协议不能防止许多常见的安全错误。这个
一节提供了几个建议来帮助实现者。

D.1。临时RSA密钥

美国出口限制将用于加密的RSA密钥限制为512
位，但不要对用于的RSA密钥的长度设置任何限制
签名操作。证书通常需要大于512
位，因为512位RSA密钥对高值不够安全
交易或需要长期安全的应用程序。一些
证书也被指定为仅签署，在这种情况下
不能用于密钥交换。

当证书中的公钥不能用于加密时，
服务器签署临时RSA密钥，然后进行交换。在
可导出应用程序，临时RSA密钥应为最大值
允许长度（即512位）。因为512位RSA密钥
相对不安全，应该经常更换。典型
电子商务应用，建议密钥
每天或每500个交易更改，如果可能，更频繁。
注意，虽然可以使用相同的临时密钥
多个事务，它必须在每次使用时签名。

RSA密钥生成是一个耗时的过程。在许多情况下，a
低优先级进程可以分配任务的密钥生成。

每当新密钥完成时，现有的临时密钥可以是
替换为新的。

D.2。随机数生成和种子

TLS需要一个密码安全的伪随机数生成器
（PRNG）。在设计和播种PRNG时必须小心。PRNG
基于安全散列操作，最值得注意的是MD5和/或SHA
可接受，但不能提供比尺寸更大的安全性
随机数发生器状态。（例如，通常基于MD5的PRNG
提供128位的状态。）

要估计生产的种子材料的量，请添加
每个种子字节中不可预测的信息的比特数。对于
示例，击键定时值取自PC兼容的18.2 Hz
定时器每个提供1或2个安全位，即使总大小
计数器值为16位或更多。种子128位PRNG，一个

将因此需要大约100个这样的定时器值。

警告：RSAREF中的播种功能和BSAFE之前的版本
3.0是顺序无关的。例如，如果1000个种子位是
提供，一次一个，在1000个分开的调用种子
函数，PRNG将最终处于仅依赖的状态
对种子数据中的0或1个种子位的数量（即，
有1001种可能的最终状态）。应用程序使用
BSAFE或RSAREF必须格外小心，以确保正确播种。
这可以通过将种子比特累积到a中来实现
缓冲区并且同时处理它们或通过处理
每个种子位递增计数器；任何一种方法都会
重新引入顺序依赖到种子过程。

D.3。证书和身份验证

实现负责验证的完整性
证书并且通常应支持证书吊销
消息。证书应始终验证以确保正确
由受信任的证书颁发机构（CA）签名。选择和
添加受信任的CA应该非常仔细。用户应该
能够查看有关证书和根CA的信息。

D.4。密码套件

TLS支持一系列密钥大小和安全级别，包括一些
其不提供或最小的安全性。正确的实现将
可能不支持许多密码套件。例如，40位
加密容易崩溃，因此实现需要强大
安全性不应允许40位密钥。类似地，匿名Diffie-
Hellman强烈反对，因为它不能防止人 -
中间人攻击。应用程序还应强制执行最小和
最大键大小。例如，包含512位的证书链
RSA密钥或签名不适合高安全性
应用程序。

E.与SSL的向后兼容性

出于历史原因，以避免浪费消费
保留端口号，受保护的应用协议
TLS 1.0，SSL 3.0和SSL 2.0通常共享相同
连接端口：例如，https协议（由SSL保护的HTTP）
或TLS）使用端口443，而不管它是哪个安全协议
使用。因此，必须确定一些机制来区分和
在各种协议之间进行协商。

TLS版本1.0和SSL 3.0非常相似；从而，支持两者
简单。希望与SSL 3.0服务器协商的TLS客户端
应使用SSL 3.0记录格式发送客户端hello消息
客户端hello结构，发送{3，1}为版本字段注释
他们支持TLS 1.0。如果服务器仅支持SSL 3.0，它
将使用SSL 3.0服务器hello响应；如果它支持TLS，用a
TLS服务器问候。然后，协商进行适当的
协商协议。

类似地，希望与SSL 3.0互操作的TLS服务器
客户端应该接受SSL 3.0客户端hello消息并进行响应
SSL 3.0服务器问候（如果接收到SSL 3.0客户端问候）
有版本字段{3,0}，表示此客户端没有
支持TLS。

每当客户端已经知道a的最高协议
服务器（例如，当恢复会话时），它应该启动
连接。

支持SSL 2.0版服务器的TLS 1.0客户端必须发送SSL
版本2.0客户端hello消息[SSL2]。TLS服务器应该接受
如果客户端希望支持SSL 2.0客户端，则为客户端hello格式
相同的连接端口。唯一的偏离版本2.0
规范是指定具有值的版本的能力
三和支持更多的加密类型在CipherSpec。

警告：发送2.0版客户端hello消息的能力将是
逐渐退出与所有急切。实现者应该使每一个
努力尽快前进。版本3.0
提供更好的机制移动到更新的版本。

以下密码规范是来自SSL版本的转发
2.0。这些假定使用RSA进行密钥交换
认证。

```
V2CipherSpec TLS_RC4_128_WITH_MD5 = {0x01,0x00,0x80};
V2CipherSpec TLS_RC4_128_EXPORT40_WITH_MD5 = {0x02,0x00,0x80};
V2CipherSpec TLS_RC2_CBC_128_CBC_WITH_MD5 = {0x03,0x00,0x80};
```



```
V2CipherSpec TLS_RC2_CBC_128_CBC_EXPORT40_WITH_MD5
                                          = {0x04,0x00,0x80};
V2CipherSpec TLS_IDEA_128_CBC_WITH_MD5 = {0x05,0x00,0x80};
```

```
        V2CipherSpec TLS_DES_64_CBC_WITH_MD5 = {0x06,0x00,0x40};
        V2CipherSpec TLS_DES_192_EDE3_CBC_WITH_MD5 = {0x07,0x00,0xC0};
```

TLS本地的密码规范可以包括在版本2.0中
客户端hello消息使用以下语法。任何V2CipherSpec
元素，其第一个字节等于零将被版本忽略
2.0服务器。发送任何上述V2CipherSpecs的客户端应该
也包括TLS等同物（见附录A.5）：

```
        V2CipherSpec（参见TLS名称）= {0x00，CipherSuite};
```

E.1。版本2客户端问候

下面使用这个版本2.0客户端问候消息
文档的演示模型。仍然假设真正的定义
成为SSL 2.0版规范。

```
        uint8 V2CipherSpec [3];

        struct {
            uint8 msg_type;
            版本版本;
            uint16 cipher_spec_length;
            uint16 session_id_length;
            uint16 challenge_length;
            V2CipherSpec cipher_specs [V2ClientHello.cipher_spec_length];
            opaque session_id [V2ClientHello.session_id_length];
            随机挑战;
        } V2ClientHello;
```

msg_type
此字段与版本字段相结合，标识a
版本2客户端hello消息。该值应为一（1）。

版
客户端支持的协议的最高版本
（等于ProtocolVersion.version，见附录A.1）。

cipher_spec_length
此字段是字段cipher_specs的总长度。它
不能为零，并且必须是V2CipherSpec长度的倍数
（3）。

session_id_length
此字段必须具有零或16的值。如果为零，则
客户端正在创建新会话。如果16，则为session_id字段
将包含16字节的会话标识。

challenge_length
客户端对服务器的挑战的长度（以字节为单位）
验证自身。此值必须为32。

cipher_specs
这是客户端愿意并能够执行的所有CipherSpecs的列表

使用。必须至少有一个CipherSpec可以接受
服务器。

session_id
如果此字段的长度不为零，它将包含
对客户端希望恢复的会话的标识。

挑战
客户端对服务器的挑战要让服务器识别
本身是（几乎）任意长度的随机。TLS服务器将
右对齐挑战数据成为ClientHello.random
数据（如果需要，用前导零填充），如第
本协议规范。如果挑战的长度
大于32字节，则仅使用最后32个字节。它是
合法（但不是必需），V3服务器拒绝V2
ClientHello具有少于16个字节的挑战数据。

注意：恢复TLS会话的请求应使用TLS客户端问候。

E.2。避免中间人版本回滚

当TLS客户端回到2.0版兼容模式时，它们
应使用特殊的PKCS＃1块格式化。这样做使TLS
服务器将拒绝具有TLS功能的客户端的2.0版会话。

当TLS客户端处于2.0版兼容模式时，它们设置
右端（最低有效）8个随机字节的PKCS填充
（不包括填充的终端零）
加密CLIENT-MASTER-KEY的ENCRYPTED-KEY-DATA字段
到0x03（其他填充字节是随机的）。解密后
ENCRYPTED-KEY-DATA字段，支持TLS的服务器应发出
错误，如果这八个填充字节是0x03。版本2.0服务器
以这种方式填充的接收块将正常进行。

F.安全分析

TLS协议旨在建立之间的安全连接
客户端和服务器通过不安全的通道进行通信。这个
文档做了几个传统的假设，包括那些
攻击者拥有大量的计算资源，无法获得
来自协议之外的源的秘密信息。攻击者
假设有能力捕获，修改，删除，重放和
否则篡改通过通信信道发送的消息。
本附录概述了TLS是如何设计来抵御各种各样的
的攻击。

F.1。握手协议

握手协议负责选择CipherSpec和
生成主密钥，它们一起构成主密钥
与安全会话相关联的密码参数。的

握手协议还可以可选地认证具有的各方
由受信任的证书颁发机构签名的证书。

F.1.1。认证和密钥交换

TLS支持三种认证模式：两者的认证
与未认证的客户端的服务器认证，以及
完全匿名。每当服务器通过身份验证时，通道为
安全防止中间人攻击，但完全匿名
会话本身容易受到这种攻击。匿名
服务器无法验证客户端。如果服务器已通过身份验证，
其证书消息必须提供有效的证书链
导致可接受的证书颁发机构。类似地，
经过身份验证的客户端必须向其提供可接受的证书
服务器。每一方都有责任验证对方的
证书有效，并且未过期或被撤销。

密钥交换过程的总体目标是创建一个
pre_master_secret已知通信方，而不是
攻击者。pre_master_secret将用于生成
master_secret（见第8.1节）。需要master_secret
生成证书验证和完成消息，加密
密钥和MAC秘密（参见第7.4.8,7.4.9和6.3节）。发送
正确完成的消息，各方因此证明他们知道
正确pre_master_secret。

F.1.1.1。匿名密钥交换

完全匿名会话可以使用RSA或
密钥交换的Diffie-Hellman。用匿名RSA，客户端
使用服务器的未认证公钥加密pre_master_secret

从服务器提取的密钥交换消息。结果发送进去
客户端密钥交换消息。因为窃听者不知道
服务器的私钥，它将不可行，他们解码
pre_master_secret。（注意，没有匿名RSA密码套件
在本文档中定义）。

使用Diffie-Hellman，服务器的公共参数包含在
服务器密钥交换消息和客户端在中发送
客户端密钥交换消息。窃听者不知道
私有值应该不能找到Diffie-Hellman结果
（即pre_master_secret）。

警告：完全匿名连接仅提供保护
反对被动窃听。除非独立的窜改 -
防伪通道用于验证完成的消息
没有被攻击者替代，服务器认证是
在活跃中间人的环境中需要
攻击是一个关注。

F.1.1.2。RSA密钥交换和身份验证

使用RSA，组合密钥交换和服务器认证。的

公钥可以包含在服务器的证书中，也可以包含在服务器的证书中
是在服务器密钥交换消息中发送的临时RSA密钥。什么时候
使用临时RSA密钥，它们由服务器的RSA或签名
DSS证书。签名包括电流
ClientHello.random，所以旧的签名和临时密钥不能
重播。服务器可以使用单个临时RSA密钥用于多个
谈判会话。

注意：如果服务器需要较大，则临时RSA密钥选项很有用
证书，但必须遵守政府规定的大小限制
用于密钥交换的密钥。

验证服务器的证书后，客户端加密a
pre_master_secret与服务器的公钥。成功
解码pre_master_secret并产生正确完成
消息，则服务器会显示它知道私钥
对应于服务器证书。

当RSA用于密钥交换时，客户端使用认证
证书验证消息（参见第7.4.8节）。客户签字
从master_secret派生的值和所有在前的握手
消息。这些握手消息包括服务器证书，
其将签名绑定到服务器，以及ServerHello.random，
其将签名绑定到当前握手过程。

F.1.1.3。Diffie-Hellman密钥交换与认证

当使用Diffie-Hellman密钥交换时，服务器可以
提供包含固定Diffie-Hellman参数的证书
可以使用服务器密钥交换消息来发送一组临时的
使用DSS或RSA证书签名的Diffie-Hellman参数。
临时参数使用之前的hello.random值进行哈希
签名以确保攻击者不重播旧参数。在
无论哪种情况，客户端都可以验证证书或签名
确保参数属于服务器。

如果客户端具有包含固定Diffie-Hellman的证书
参数，其证书包含所需的信息
完成密钥交换。注意在这种情况下客户端和
服务器将生成相同的Diffie-Hellman结果（即，
pre_master_secret）每次他们沟通。防止
pre_master_secret从留在内存不再需要，
它应该尽快转换成master_secret。
客户端Diffie-Hellman参数必须与这些参数兼容
由服务器提供的密钥交换工作。

如果客户端有标准DSS或RSA证书或是
未认证，它会向服务器发送一组临时参数
在客户端密钥交换消息中，然后可选地使用a
证书验证消息来验证自身。

F.1.2。版本回滚攻击

因为TLS包括对SSL版本2.0的实质性改进，
攻击者可能会尝试使支持TLS的客户端和服务器回退
到版本2.0。这种攻击可能发生如果（并且只有）两个TLS-
有能力的各方使用SSL 2.0握手。

虽然解决方案使用非随机PKCS＃1块类型2消息
填充是不起眼的，它为Version提供了一个相当安全的方法
3.0服务器来检测攻击。这个解决方案不安全
攻击者谁可以暴力强制关键和替换一个新的
ENCRYPTED-KEY-DATA消息包含相同的键（但具有正常
填充）应用程序指定的等待阈值已过期。
不应使用关注这一规模的攻击的缔约方
40位加密密钥。改变最小二乘法的填充，
PKCS填充的有效8字节不影响安全性
有符号哈希的大小和RSA中使用的密钥长度
协议，因为这基本上等于增加
输入块大小为8字节。

F.1.3。检测对握手协议的攻击

攻击者可能会试图影响握手交换
各方选择不同的加密算法
通常选择。因为许多实现将支持40位
可导出的加密，有些甚至可以支持空加密
MAC算法，这种攻击是特别关注的。

对于此攻击，攻击者必须主动更改一个或多个
握手消息。如果发生这种情况，客户端和服务器将
为握手消息哈希计算不同的值。作为一个
结果，各方将不会接受彼此完成的消息。
没有master_secret，攻击者无法修复完成
消息，所以攻击将被发现。

F.1.4。恢复会话

当通过恢复会话建立连接时，新
ClientHello.random和ServerHello.random值与
session的master_secret。只要master_secret没有
妥协和用于产生的安全哈希操作
加密密钥和MAC秘密是安全的，连接应该是
安全和有效地独立于以前的连接。
攻击者不能使用已知的加密密钥或MAC密钥
妥协master_secret而不破坏安全散列
操作（使用SHA和MD5）。

除非客户端和服务器同意，否则无法恢复会话。
如果任何一方怀疑会话可能已经泄密，
或证书可能已过期或已被撤销，它应该
强制完全握手。建议上限为24小时
会话ID生存期，因为攻击者获得了master_secret

可能能够假冒被侵害方，直到
相应的会话ID退休。可运行的应用程序
相对不安全的环境不应该写入会话ID
稳定存储。

F.1.5。MD5和SHA

TLS非常保守地使用散列函数。在可能的情况下，MD5
和SHA串联使用以确保非灾难性缺陷
一个算法不会破坏整个协议。

F.2。保护应用程序数据

master_secret使用ClientHello.random和进行散列
ServerHello.random生成唯一的数据加密密钥和MAC
每个连接的秘密。

在发送之前，用MAC保护输出数据。阻止
消息重放或修改攻击，MAC从
MAC秘密，序列号，消息长度，消息
内容，和两个固定字符串。消息类型字段为
以确保发送给一个TLS记录层的消息
客户端不会重定向到另一个。序列号确保
将尝试删除或重新排序邮件将被检测。以来
序列号为64位长，它们不应该溢出。
来自一方的消息不能插入对方的输出，
因为它们使用独立的MAC秘密。同样，服务器写
和客户端写密钥是独立的，因此使用流密码密钥
只有一次。

如果攻击者破坏了加密密钥，则所有消息都加密
可以读取。类似地，MAC密钥的妥协可以做到
消息修改攻击。因为MAC也是
加密，消息更改攻击通常需要打破
加密算法以及MAC。

注意：MAC秘密可能大于加密密钥，因此消息可以
即使加密密钥被破坏也保持防篡改。

F.3。最后的笔记

为了使TLS能够提供安全的连接，客户端
并且服务器系统，密钥和应用程序必须是安全的。在
另外，实现必须没有安全错误。

系统只有最弱的密钥交换和强大
认证算法支持，只有可信赖
应使用加密函数。短公钥，40位
批量加密密钥和匿名服务器应该很好地使用
警告。实施和用户在决定时必须小心
哪些证书和证书当局是可以接受的； 一个
不诚实的认证机构可以做到巨大的损害。

G.专利声明

一些建议在这里使用的密码算法
协议有专利要求。另外Netscape
通信公司在安全套接字上有专利声明
本标准所基于的图层（SSL）工作。互联网
RFC 2026中定义的标准过程要求语句为
从专利持有人获得，表明将会发出许可证
在合理的条款和条件下向申请人提供。

麻省理工学院授予RSA数据
安全，公司，以下的独家次级许可权
在美国发布的专利：

密码通信系统和方法（“RSA”），No.
4,405,829

Netscape Communications Corporation已发布以下内容
专利在美国：

安全套接字层应用程序装置和方法
（“SSL”），No.5,657,390

Netscape Communications发布了以下声明：

知识产权

安全链路层

美国专利商标局（“PTO”）
最近发布的美国专利号5,657,390（“SSL专利”）
描述为安全套接字层的发明的Netscape
（“SSL”）。IETF目前正在考虑采用SSL
传输协议与安全特性。Netscape鼓励
免版税的采纳和使用SSL协议
以下条款及条件：

*如果你已经有一个有效的SSL Ref许可证
包括Netscape的源代码，一个额外的专利
根据SSL专利的许可证不是必需的。

*如果您没有SSL Ref许可证，您可能有版税
免费许可证构建由SSL覆盖的实现
专利声明或IETF TLS规范提供您

不要对Netscape或其他方面声明任何专利权
公司实施SSL或IETF TLS
建议。

什么是“专利权利要求”：

专利权利要求是发布的外国或国内专利中的权利要求
那：

1）必须被侵犯以实施方法或构建
产品根据IETF TLS规范；要么

2）要求SSL专利的要素的专利权利要求
权利要求和/或其等同物被侵犯。

互联网协会，互联网架构委员会，互联网
工程指导小组和国家研究公司
倡议不对专利的有效性或范围持有立场
和专利申请，也不关于条款的适用性
保证。互联网协会和上述其他团体
没有对任何其他知识产权作出任何决定
权利可能适用于本标准的实践。任何进一步
考虑这些事项是用户自己的责任。

安全注意事项

安全问题在本备忘录中讨论。

参考文献

[3DES] W. Tuchman，“Hellman Presents No Shortcut Solutions To DES”，
IEEE Spectrum，v.16，n。7，July 1979，pp40-41。

[BLEI] Bleichenbacher D.，“Chosen Ciphertext Attacks against
基于RSA加密标准PKCS＃1的协议“
Advances in Cryptology - CRYPTO'98，LNCS vol。1462，pages：
1--12，1998。

[DES] ANSI X3.106，“美国国家信息标准
系统 - 数据链路加密“，美国国家标准
研究所，1983年。

[DH1] W.Diffie和MEHellman，“New Directions in
Cryptography，“IEEE Transactions on Information Theory，V.
IT-22，n。6，1977年6月，第74-84页。

[DSS] NIST FIPS PUB 186，“数字签名标准”，国家
美国标准技术研究所
商务部，1994年5月18日。

[FTP] Postel J.，and J.Reynolds，“File Transfer Protocol”，STD 9，
RFC 959，1985年10月。

[HTTP] Berners-Lee，T.，Fielding，R.，and H.Frystyk，“Hypertext
传输协议 - HTTP / 1.0“，RFC 1945，1996年5月。

[HMAC] Krawczyk，H.，Bellare，M.，and R.Canetti，“HMAC：Keyed-
Hashing for Message Authentication“，RFC 2104，February
1997年。

[IDEA] X.Lai，“On the Design and Security of Block Ciphers”，ETH
Series in Information Processing，v.1，Konstanz：Hartung-
Gorre Verlag，1992。

[MD2] Kaliski，B.，“The MD2 Message Digest Algorithm”，RFC 1319，
1992年4月。

[MD5] Rivest，R。，“The MD5 Message Digest Algorithm”，RFC 1321，
1992年4月。

[PKCS1] RSA实验室，“PKCS＃1：RSA加密标准”
版本1.5，1993年11月。

[PKCS6] RSA实验室，“PKCS＃6：RSA扩展证书语法
Standard，“version 1.5，November 1993。

[PKCS7] RSA Laboratories，“PKCS＃7：RSA加密消息语法
Standard，“version 1.5，November 1993。

[PKIX] Housley，R.，Ford，W.，Polk，W.and D. Solo，“Internet
公钥基础设施：第一部分：X.509证书和CRL
Profile“，RFC 2459，1999年1月。

[RC2] Rivest，R。，“A Description of the RC2（r）Encryption
算法“，RFC 2268，1998年1月。

[RC4] Thayer，R。和K.Kaukonen，A Stream Cipher Encryption
算法，工作进行中。

[RSA] R.Rweest，A.Shamir和LMAdleman，“A Method for
获取数字签名和公钥密码系统“
ACM通讯，v。21，n。2，Feb 1978，pp。120-
126。

[RSADSI]联系RSA Data Security，Inc.，电话：415-595-8782

[SCH] B.Schneier。应用密码学：协议，算法，
和源代码在C中，由John Wiley＆Sons，Inc。出版
1994年。

[SHA] NIST FIPS PUB 180-1，“Secure Hash Standard，”National

美国标准技术研究所
商业，正在进行中，1994年5月31日。

[SSL2] Hickman，Kipp，“The SSL Protocol”，Netscape Communications
Corp.，Feb 9，1995。

[SSL3] A.Frier，P.Karlton和P.Kocher，“The SSL 3.0 Protocol”，
Netscape Communications Corp.，Nov 18，1996。

[TCP] Postel，J.，“Transmission Control Protocol”，STD 7，RFC 793，
1981年9月。

[TEL] Postel J.，and J.Reynolds，“Telnet Protocol
Specifications“，STD 8，RFC 854，May 1993。

[TEL] Postel J.，and J.Reynolds，“Telnet Option Specifications”，
STD 8，RFC 855，May 1993。

[X509] CCITT。建议X.509：“号码簿 - 认证
框架“。

[XDR] R. Srinivansan，Sun Microsystems，RFC-1832：XDR：External
数据表示标准，1995年8月。

积分

赢得Treese
公开市场

电子邮件：treese@openmarket.com

编辑

Christopher Allen Tim Dierks
Certicom Certicom

电子邮件：callen@certicom.com电子邮件：tdierks@certicom.com

作者地址

蒂姆·迪克斯菲利普·卡尔顿
Certicom Netscape通信

电子邮件：tdierks@certicom.com

Alan O. Freier Paul C. Kocher
Netscape通信独立顾问

电子邮件：freier@netscape.com电子邮件：pck@netcom.com

其他贡献者

马丁·阿巴迪罗伯特Relyea

数字设备公司Netscape通信

电子邮件：ma@pa.dec.com电子邮件：relyea@netscape.com

Ran Canetti吉姆罗斯金德
IBM沃森研究中心Netscape Communications

电子邮件：canetti@watson.ibm.com电子邮件：jar@netscape.com

Taher Elgamal Micheal J. Sabin博士
Securify咨询工程师

电子邮件：elgamal@securify.com电子邮件：msabin@netcom.com

Anil R. Gangolli丹西蒙
结构化艺术计算公司

电子邮件：gangolli@structuredarts.com电子邮件：dansimon@microsoft.com

Kipp EB Hickman汤姆·温斯坦
Netscape通信Netscape通信

电子邮件：kipp@netscape.com电子邮件：tomw@netscape.com

Hugo Krawczyk
IBM沃森研究中心

电子邮件：hugo@watson.ibm.com

注释

IETF TLS工作组的讨论列表位于
电子邮件地址<ietf-tls@lists.consensus.com>。关于的信息
组和关于如何订阅列表的信息
<http://lists.consensus.com/>。

名单的档案可在以下网址找到：
<http://www.imc.org/ietf-tls/mail-archive/>

完整的版权声明