SSL VPN 技术规范

SSL VPN Technology Specification

国 家 密 码 管 理 局

2010 年 8 月

目　次

前　言

本规范的协议内容参照传输层安全协议（RFC4346 TLS1.1）。按照我国相关密码政策和法规，结合我国实际应用需求及产品生产厂商的实践经验，在 TLS1.1 的握手协议中增加了 ECC、IBC 的认证模式和密钥交换模式，取消了 DH 密钥协商方式，修改了密码套件的定义。另外，在本规范中还增加了网关-网关协议。

本规范由国家密码管理局提出并归口。

本规范主要起草单位：上海格尔软件股份有限公司、华为技术有限公司、深圳市深信服电子科技有限公司、深圳市奥联科技有限公司、网御神州科技（北京）有限公司、成都卫士通信息产业股份有限公司、北京东方华盾信息技术有限公司、中国国际电子商务有限公司、联想网御科技（北京）有限公司、上海安达通信息安全有限公司、无锡江南信息安全工程技术中心。

本规范主要起草人：谭武征、黄敏、曾建发、但波、刘建锋、罗俊、李志超、李飞伯、何致宇、陈凯、朱正超、倪永年。

本规范的责任专家：刘平。

引　　言

本规范主要由密码算法和密钥种类、协议、产品要求、产品检测及合格判定等章节组成。
本规范中未明确指明为可选要素的部分均为必备要素。
基于本规范研制的SSL VPN产品所用的密码算法和密码部件须经国家密码管理局审批。

1 范围

本规范对 SSL VPN 的技术协议、产品的功能、性能和管理以及检测进行了规定。

本规范适用于 SSL VPN 产品的研制，也可用于指导 SSL VPN 产品的检测和使用。

2 规范性引用文件

下列文件中的条款通过本规范的引用而成为本规范的条款。凡是注明日期的引用文件，其随后所有的修改单（不包括勘误的内容）或修订版均不适用于本规范，然而，鼓励根据本规范达成协议的各方研究是否可使用这些文件的最新版本。凡是不注明日期的引用文件，其最新版本适用于本规范。

GB/T 20518-2006 信息安全技术　公钥基础设施　数字证书格式

GB/T XXXX-200X 信息技术 安全技术 密码术语

RFC4346 The Transport Layer Security (TLS) Protocol Version 1.1

随机数检测规范

3 术语及定义

下列术语仅适用于本规范。

3.1

数字证书 digital certificate

又称为公钥证书，是由证书认证机构签名的包含公开密钥拥有者的信息、公开密钥、签发者信息、有效期以及一些扩展信息的数字文件。本规范中使用的是签名证书或者具有签名能力的证书。

3.2

IBC 算法 Identity Based Cryptography Algorithm

IBC算法又称为标识密码算法，是一种能以任意标识作为公钥，不需要使用数字证书证明公钥的非对称密码算法。

3.3

IBC 标识 IBC Identity

IBC标识是表示实体身份或属性的字符串。

3.4

IBC 公共参数 IBC Public Parameter

IBC公共参数包含了IBC密钥管理中心的名称、运算曲线、标识编码方式和密钥生成算法等公开参数信息，这些信息用于将实体标识转换为公开密钥。

3.5

初始化向量 initialization vector/initialization value（IV）

在密码变换中，为增加安全性或使密码设备同步而引入的用作数据变换的起始数据。

3.6

SSL 协议 secure sockets layer protocol

一种应用于传输层的安全协议，用于构建客户端和服务端之间的安全通道。

3.7

会话 Session

SSL会话是通过握手协议创建的客户端和服务端的关联，一个会话可供多个连接共享。

4 符号和缩略语

下列符号适用于本规范：

+	串联
⊕	异或运算
^	指数运算
HMAC(X,Y)	以 X 为密钥对 Y 进行密码杂凑运算
ceil(X)	对 X 向上取整

下列缩略语适用于本规范：

CBC	密码分组链接工作模式
HMAC	采用杂凑算法计算的消息验证码
VPN	虚拟专用网络
SSL	安全套接层
IBC	标识密码算法

5 密码算法和密钥种类

5.1 密码算法

5.1.1 非对称密码算法

非对称密码算法包括 256 位群阶 ECC 椭圆曲线密码算法 SM2、IBC 标识密码算法 SM9 和 1024 位以上 RSA 算法，用于身份鉴别、数字签名、密钥协商等。

5.1.2 分组密码算法

分组密码算法为 SM1 算法，用于密钥协商数据的加密保护和报文数据的加密保护。该算法的工作模式为 CBC 模式。

5.1.3 密码杂凑算法

密码杂凑算法包括 SM3 算法和 SHA-1 算法，用于密钥生成和完整性校验。

5.1.4 数据扩展函数 P_hash

P_hash 函数定义如下：

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

其中：

secret 是进行计算所需要的密钥。

seed 是进行计算所需要的数据。

A(0) = seed

A(i) = HMAC (secret, A(i-1))

P_hash 能够反复迭代直至产生要求长度的数据。

5.1.5 伪随机函数 PRF

定义 S1 和 S2 分别为 secret 的前半部分和后半部分。

则 S1 的长度 L_S1 = S2 的长度 L_S2 =ceil(secret 的长度/ 2);

S1=secret 的前 L_S1 个字节;

S2=secret 的后 L_S2 个字节

计算方法如下:

PRF(secret, label, seed) = P_SHA1(S1, label + seed) ⊕ P_SM3(S2, label + seed);

5.2 密钥种类

5.2.1 概述

在本规范中采用非对称密码算法进行身份鉴别和密钥协商，身份鉴别通过后协商预主密钥，双方各自计算主密钥，进而推导出工作密钥。使用工作密钥进行加解密和完整性校验。

5.2.2 服务端密钥

服务端密钥为非对称密码算法的密钥对。用于握手过程中服务端身份鉴别和预主密钥的协商。

5.2.3 客户端密钥

客户端密钥为非对称密码算法的密钥对。用于握手过程中客户端身份鉴别和预主密钥的协商。

5.2.4 预主密钥

预主密钥（pre_master_secret）是双方协商生成的 48 字节的密钥素材，用于生成主密钥。

5.2.5 主密钥

主密钥（master_secret）由预主密钥、客户端随机数、服务端随机数、常量字符串，经计算生成的 48 字节密钥素材，用于生成工作密钥。

5.2.6 工作密钥

工作密钥包括数据加密密钥和校验密钥。其中数据加密密钥用于数据的加密和解密，校验密钥用于数据的完整性计算和校验。在本规范中，发送方使用的工作密钥称为写密钥，接收方使用的工作密钥称为读密钥。

6 协议

6.1 概述

SSL VPN 协议包括握手协议、密码规格变更协议、报警协议、网关到网关协议和记录层协议。握手协议用于身份鉴别和安全参数协商；密码规格变更协议用于通知安全参数的变更；报警协议用于关闭通知和对错误进行报警；网关到网关协议用于建立网关到网关的传输层隧道；记录层协议用于传输数据的分段、压缩及解压缩、加密及解密、完整性校验等。

6.2 数据类型定义

6.2.1 基本数据类型

本规范定义了六种基本数据类型，分别为 opaque、uint8、uint16、uint24、uint32、uint64。所有类型都以网络字节顺序表示，最小数据的大小是一个 8 位字节。

opaque ： 任意类型数据，1 个字节
uint8 ： 无符号的 8 位整数，1 个字节
uint16 ： 无符号的 16 位整数，2 个字节
uint24 ： 无符号的 24 位整数，3 个字节
uint32 ： 无符号的 32 位整数，4 个字节
uint64 ： 无符号的 64 位整数，8 个字节

6.2.2 向量

向量（Vectors）是给定类型的数据序列。向量分为两种：定长和变长。定长向量以[x]来表示；变长向量以< x..y>来表示，其中 x 代表下限，y 代表上限，如果只需表达上限时，用<y>表示。所有向量的长度都以字节为单位。变长向量首部表示向量实际长度，其首部大小为能够容纳变长向量最大长度的最小字节数。

6.2.3 枚举类型

枚举（Enumerateds）是一系列特定值的字段集合，通常每个字段都包括一个名称和值。如果包含一个未命名的值，这个值表示指定的最大值。如果只包含了名称而不定义值，只能用来指代状态值，不能在实际编码中使用。例如，enum{red(0), green(1), (255)}color。枚举变量大小为能够容纳最大枚举值的最小字节数。

6.2.4 结构类型

结构类型（Constructed Types）用 struct 来定义，与 C 语言的 struct 语法类似。struct 中的字段按照先后顺序串连起来进行编码。如果一个 struct 包含于另一个 struct 中，则可以省略该 struct 的名字。

6.2.5 变体类型

变体（Variants）类型用 select、case 来定义，用于定义依赖外部信息而变化的结构，类似于 C 语言中的 union 或 ASN.1 的 CHOICE。

6.3 记录层协议

记录层协议是分层次的，每一层都包括长度字段、描述字段和内容字段。记录层协议接收将要被传输的消息，将数据分块、压缩（可选）、计算 HMAC、加密，然后传输。接收到的数据经过解密、验证、解压缩（可选）、重新封装然后传送给高层应用。

记录层协议包括：握手，报警，密码规格变更和网关到网关等类型。为了支持协议的扩展，记录层协议可支持其它的记录类型。任何新的记录类型都必须在针对上述类型分配的内容类型值之外去分配。如果接收到一个不能识别的记录类型应忽略。

6.3.1 连接状态

连接状态是记录层协议的操作环境。包括四种典型的连接状态：当前读与写状态，未决的读与写状态。其中，读表示接收数据，写表示发送数据。所有记录都是在当前读写状态下处理的。未决状态的

安全参数可以由握手协议设定，而密码规格变更消息可使未决状态变为当前状态。除了最初的当前状态外，当前状态必须包含经过协商的安全参数。

连接状态的安全参数结构如下：

```
struct {
          ConnectionEnd          entity;
          BulkCipherAlgorithm    bulk_cipher_algorithm;
          CipherType             cipher_type;
          uint8                  key_size;
          uint8                  key_material_length;
          MACAlgorithm           mac_algorithm;
          uint8                  hash_size;
          CompressionMethod      compression_algorithm;
          opaque                 master_secret[48];
          opaque                 client_random[32];
          opaque                 server_random[32];
      } SecurityParameters;
```

其中：

1) connection end

表示本端在连接中的角色，为客户端或服务端。定义为：

```
enum { server, client } ConnectionEnd
```

2) bulk encryption algorithm

用于数据加解密的密码算法。定义为：

```
enum { sm1}BulkCipherAlgorithm
```

3) CipherType

表示密码算法的类型。定义为：

```
enum { block } CipherType;
```

4) MAC algorithm

用于计算和校验消息完整性的密码杂凑算法。定义为：

```
enum { sha ,sm3} MACAlgorithm
```

5) compression algorithm

用于数据压缩的算法。定义为：

```
enum { null(0), (255) } CompressionMethod
```

6) master secret

在协商过程中由预主密钥、客户端随机数、服务端随机数计算出的 48 字节密钥。

7) client random

由客户端产生的 32 字节随机数据。

8) server random

由服务端产生的 32 字节随机数据。

记录层将使用上述安全参数来生成下列内容：

客户端写校验密钥 client write MAC secret

服务端写校验密钥 server write MAC secret

客户端写密钥 client write key

服务端写密钥 server write key

服务端接收和处理记录时使用客户端写参数，客户端接收和处理记录时使用服务端写参数。用安全参数生成这些密钥的算法，详见 6.5。密钥生成后，连接状态就可以改变为当前状态。

6.3.2 记录层

记录层接收从高层来的任意大小的非空连续数据，将数据分段、压缩、计算校验码、加密，然后传输。接收到的数据经过解密、验证、解压缩、重新封装然后传送给高层应用。

6.3.2.1 分段

记录层将数据分成 2^{14} 字节或者更小的片段。

每个片段结构如下：

```
struct {
        ContentType type;
        ProtocolVersion version;
        uint16 length;
        opaque fragment[TLSPlaintext.length];
      } TLSPlaintext;
```

其中：

1) type

片段的记录层协议类型。定义为：

```
enum {
      change_cipher_spec(20),alert(21),handshake(22),
       application_data(23), site2site(80),(255)
   } ContentType;
```

2) version

所用协议的版本号。本规范的版本号为 1.0。定义为：

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

3) length

以字节为单位的片段长度，小于等于 2^{14}。

4) fragment

将传输的数据。记录层协议不关心具体数据内容。

6.3.2.2 压缩和解压缩

所有的记录都使用当前会话状态指定的压缩算法进行压缩。当前会话状态指定的压缩算法被初始化为空算法。

压缩算法将一个 TLSPlaintext 结构的数据转换成一个 TLSCompressed 结构的数据。

压缩后的数据长度最多只能增加 1024 个字节。如果解压缩后的数据长度超过了 2^{14} 个字节，则报告一个 decompression failure 致命错误。

压缩后数据结构如下：

```
struct {
        ContentType type;
        ProtocolVersion version;
        uint16 length;
        opaque fragment[TLSCompressed.length];
    } TLSCompressed;
```

其中：

1) type、 version 的定义同 6.3.2.1。

2) length

以字节为单位的 TLSCompressed. fragment 长度，小于等于 2^{14} + 1024。

3) fragment

TLSPlaintext.fragment 的压缩形式。

6.3.2.3 加密和校验

加密运算和校验运算把一个 TLSCompressed 结构的数据转化为一个 TLSCiphertext 结构的数据。解密运算则是执行相反的操作。

加密后数据结构如下：

```
struct {
        ContentType type;
        ProtocolVersion version;
        uint16 length;
        select (CipherSpec.cipher_type) {
             case block: GenericBlockCipher;
        } fragment;
    } TLSCiphertext;
```

其中：

1) type、 version 的定义同 6.3.2.1 分段。

2) length

以字节为单位的 TLSCiphertext.fragment 长度，小于等于 2^{14} + 2048。

3) fragment

带有校验码 的 TLSCompressed.fragment 加密形式。

6.3.2.3.1 校验算法的数据处理

校验码的计算是在加密之前进行，运算方法如下：

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

1) seq_num

序列号。每一个读写状态都分别维持一个单调递增序列号。序列号的类型为 uint64, 序列号初始值为零，最大不能超出 2^{64}-1。序列号不能回绕。如果序列号溢出，那就必须重新开始握手。

2) hash

计算校验码时使用的杂凑算法。

6.3.2.3.2 分组密码算法的数据处理

使用分组密码算法加解密数据时，加密运算和校验运算用于将一个 TLSCompressed.fragment 结构的数据转换成一个 TLSCiphertext.fragment 结构的数据。被加密的数据包括了校验运算的结果。

加密处理前数据结构如下：

```
block-ciphered struct {
        opaque IV[CipherSpec.block_length];
        opaque content[TLSCompressed.length];
        opaque MAC[CipherSpec.hash_size];
        uint8 padding[GenericBlockCipher.padding_length];
```

```
        uint8 padding_length;
    } GenericBlockCipher;
```

其中：

1) IV

在 GenericBlockCipher 中传输的初始化向量。该向量必须随机产生。

2) content

加密前的明文数据。

3) MAC

Content 的校验值。

4) padding

填充的数据。在数据加密前需要将数据填充为密码算法分组长度的整数倍，填充的长度不能超过 255 个字节。填充的每个字节的内容是填充的字节数。接收者应检查这个填充，如果出错，发送 bad_record_mac 报警消息。

6.4 握手协议族

握手协议族由密码规格变更协议、握手协议和报警协议三个子协议组成，用于双方协商出供记录层使用的安全参数，进行身份验证以及向对方报告错误等。

握手协议族负责协商出一个会话，这个会话包含：

会话标识：由服务端选取的随意的字节序列，用于识别活跃或可恢复的会话。

证书： X509 v3 格式的数字证书。

压缩方法：压缩数据的算法。

密码规格：指定的密码算法。

主密钥：客户端和服务端共享的 48 个字节的密钥。

重用标识：标明能否用该会话发起一个新连接的标识。

利用以上数据可以产生安全参数。利用握手协议的重用特性，可以使用相同的会话建立多个连接。

6.4.1 密码规格变更协议

密码规格变更协议用于通知密码规格的改变,即通知对方使用刚协商好的安全参数来保护接下来的数据。该协议由一条消息组成，该消息用当前的压缩算法压缩并用当前的密码规格加密，如果是首次协商，该消息为明文。

该消息的长度为一个字节，其值为 1。客户端和服务端都要在安全参数协商完毕之后、握手结束消息之前发送此消息。

对于刚协商好的密钥，写密钥在此消息发送之后立即启用；读密钥在收到此消息之后立即启用。

密码规格变更消息结构定义如下：

```
struct {
    enum { change_cipher_spec(1), (255) } type;
} ChangeCipherSpec;
```

6.4.2 报警协议

报警协议用于关闭连接的通知以及对整个连接过程中出现的错误进行报警,其中关闭通知由发起者发送，错误报警由错误的发现者发送。该协议由一条消息组成，该消息用当前的压缩算法压缩并用当前的密码规格加密，如果是首次协商，该消息为明文。

报警消息的长度为两个字节，分别为报警级别和报警内容。

报警消息结构定义如下：

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

```
enum {
        close_notify(0),
        unexpected_message(10),
        bad_record_mac(20),
        decryption_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),
        protocol_version(70),
        insufficient_security(71),
        internal_error(80),
        user_canceled(90),
        no_renegotiation(100),
        unsupported_site2site (200) ,
        no_area(201),
        unsupported_areatype(202),
        bad_ibcparam(203),
        unsupported_ibcparam(204)
        identity_need(205),
        (255)
} AlertDescription;

struct {
    AlertLevel level;
    AlertDescription description;
} Alert;
```

6.4.2.1 关闭通知

除非出现致命报警，客户端和服务端任何一方在结束连接之前都应发送关闭通知消息。对于该消息的发送方，该消息通知对方不再发送任何数据，可以等待接收方回应的关闭通知消息后关闭本次连接，也可以立即关闭本次连接。对于接收方，收到该消息后应回应一个关闭通知消息，然后关闭本次连接，不再接收和发送数据。

6.4.2.2 错误报警

当发送或接收到一个致命级别报警之后，双方都应立即关闭连接，废弃出错连接的会话标识和密钥，被致命报警关闭的连接是不能复用的。本规范定义的报警如表 1 所示：

表 1　错误报警表

错误报警名称	值	级别	描述
unexpected_message	10	致命	接收到一个不符合上下文关系的消息
bad_record_mac	20	致命	MAC 校验错误或解密错误
decryption_failed	21	致命	解密失败
Record overflow	22	致命	报文过长
Decompression failure	30	致命	解压缩失败
Handshake failure	40	致命	协商失败
Bad certificate	42		证书被破坏
Unsupported certificate	43		不支持证书类型
Certificate revoked	44		证书被撤销
Certificate expired	45		证书过期或未生效
Certificate unknown	46		未知证书错误
Illegal parameter	47	致命	非法参数
Unknown ca	48	致命	根证书不可信
Access denied	49	致命	拒绝访问
Decode error	50	致命	消息解码失败
Decrypt error	51		消息解密失败
Protocol version	70	致命	版本不匹配
Insufficient security	71	致命	安全性不足
Internal error	80	致命	内部错误
User canceled	90	警告	用户取消操作
No renegotiation	100	警告	拒绝重新协商
unsupported_site2site	200	致命	不支持 site2site
No_area	201		没有保护域
unsupported_areatype	202		不支持的保护域类型
bad_ibcparam	203	致命	接收到一个无效的 ibc 公共参数
unsupported_ibcparam	204	致命	不支持 ibc 公共参数中定义的信息
identity_need	205	致命	缺少对方的 ibc 标识

对于没有明确指出级别的错误报警，发送者可以自行决定是否致命，如果发送者认为是致命的，应该向接收者发出关闭通知，最后关闭本次连接；如果接收到一个警告级别的报警，接收者可以自行决定是否致命，如果接收者认为是致命的，应该向发起者发出关闭通知，最后关闭本次连接。

6.4.3　握手协议总览

握手协议涉及以下过程：

- 交换 hello 消息来协商密码套件，交换随机数，决定是否会话重用。
- 交换必要的参数，协商预主密钥。
- 交换证书或 IBC 信息，用于验证对方。
- 使用预主密钥和交换的随机数生成主密钥。
- 向记录层提供安全参数。

- 验证双方计算的安全参数的一致性、握手过程的真实性和完整性。

握手过程如下：客户端发送客户端 hello 消息给服务端，服务端应回应服务端 hello 消息，否则产生一个致命错误并且断开连接。客户端 hello 和服务端 hello 用于在客户端和服务端进行基于 RSA、ECC 或 IBC 的密码算法协商，以及确定安全传输能力，包括协议版本，会话标识，密码套件等属性，并且产生和交换随机数。

在客户端 hello 和 服务端 hello 消息之后是身份验证和密钥交换过程。包括服务端证书、服务端密钥交换，客户端证书、客户端密钥交换。

在服务端发送完 hello 消息之后，接着发送自己的证书消息，服务端密钥交换消息（可选）。如果服务端需要验证客户端的身份，则向客户端发送证书请求消息。然后发送服务端 hello 完成消息，表示 hello 消息阶段已经结束，服务端等待客户端的返回消息。如果服务端发送了一个证书请求消息，客户端必须返回一个证书消息。然后客户端发送密钥交换消息，消息内容取决于客户端 hello 消息和服务端 hello 消息协商出的密钥交换算法。如果客户端发送了证书消息，那么也应发送一个带数字签名的证书验证消息供服务端验证客户端的身份。

接着客户端发送密码规格变更消息，然后客户端立即使用刚协商的算法和密钥，加密并发送握手结束消息。服务端则回应密码规格变更消息，使用刚协商的算法和密钥，加密并发送握手结束消息。至此握手过程结束，服务端和客户端可以开始数据安全传输。

握手消息流程如图 1 握手消息流程所示：

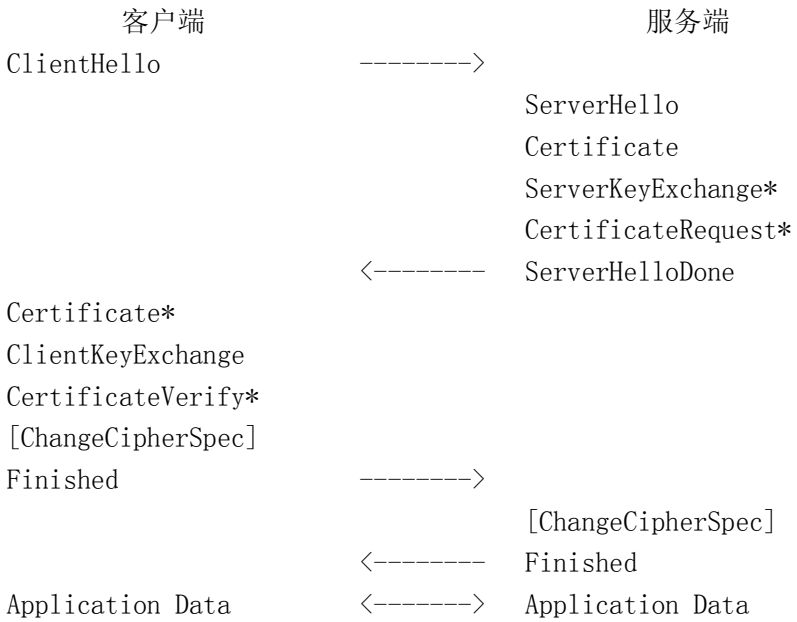

图 1 握手消息流程

图 1 中：

*表示可选或依赖于上下文关系的消息，不是每次都发送。

[]不属于握手协议消息。

如果客户端和服务端决定重用之前的会话，可不必重新协商安全参数。客户端发送客户端 hello 消息，并且带上要重用的会话标识。如果服务端有匹配的会话存在，服务端则使用相应的会话状态接受连接，发送一个具有相同会话标识的服务端 hello 消息。然后客户端和服务端各自发送密码规格变更消息和握手结束消息。至此握手过程结束，服务端和客户端可以开始数据安全传输。如果服务端没有匹配

的会话标识，服务端会生成一个新的会话标识进行一个完整的握手过程。

会话重用的握手消息流程如图 2 重用的握手消息流程所示：

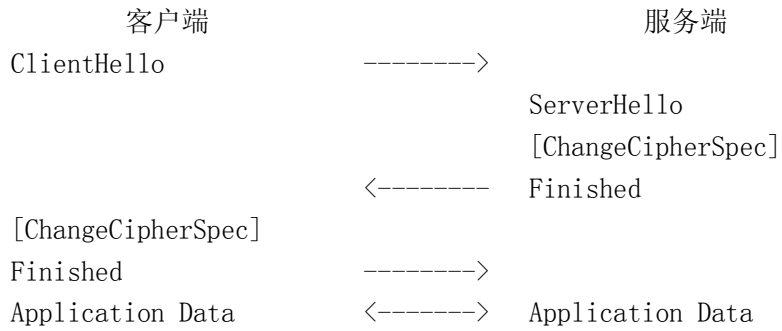

图 2 重用的握手消息流程

6.4.4 握手协议

握手协议是在记录层协议之上的协议，用于协商安全参数。握手协议的消息通过记录层协议传输。

握手消息结构定义如下：

```
struct {
          HandshakeType msg_type;
          uint24 length;
          select (HandshakeType) {
              case hello_request:       HelloRequest;
              case client_hello:        ClientHello;
              case server_hello:        ServerHello;
              case certificate:         Certificate;
              case server_key_exchange: ServerKeyExchange;
              case certificate_request: CertificateRequest;
              case server_hello_done:   ServerHelloDone;
              case certificate_verify:  CertificateVerify;
              case client_key_exchange: ClientKeyExchange;
              case finished:            Finished;
          } body;
      } Handshake;
```

握手消息类型定义如下：

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

握手协议消息应按照规定的流程顺序进行发送，否则将会导致致命的错误。不需要的握手消息可以被接收方忽略。Hello 请求消息不受流程顺序的约束，可以在任何时刻被发送，但在握手过程中应被忽略。

6.4.4.1 Hello消息

hello消息包括hello请求消息、客户端hello消息和服务端hello消息。

客户端 hello消息和服务端 hello消息用于确定双方的安全传输能力，包括协议版本，会话标识，密码套件等属性，并且产生和交换随机数。当新会话开始时，记录层连接状态中的密码算法，杂凑算法，压缩算法都初始化为空。

6.4.4.1.1 Hello Request消息

该消息为hello请求消息，服务端可以在任何时候发送该消息。

hello请求消息通知客户端可以发送客户端hello消息以进行重新协商，如果服务端发送该消息后没有接收到客户端hello消息回应，可发送致命报警消息并关闭这个连接。

如果客户端处在握手过程中，hello请求消息将被忽略。如果客户端不希望进行重新协商，也将会被忽略，并且客户端可以回应no_renegotiation报警消息。

hello请求消息结构定义如下：

```
struct { } HelloRequest;
```

6.4.4.1.2 Client Hello消息

该消息为客户端hello消息。

客户端hello消息作为握手协议的第一条消息。客户端也可以发送客户端hello消息回应服务端的hello请求消息以开始一个新的握手过程，或者在已有的连接里发送该消息以重新协商安全参数。

客户端在发送客户端hello消息之后，等待服务端回应服务端hello消息，此外除了hello请求消息，服务端发送过来的所有其它消息都被视为致命错误。

客户端hello消息结构定义如下：

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

其中：

1) client_version

客户端在这个会话中使用的协议版本。在本规范里，协议号是1.0。

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

2) random

客户端产生的随机信息，其内容包括时钟和随机数，结构定义如下：

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

gmt_unix_time 为标准UNIX 32位格式表示的发送者时钟，其值为从1970年1月1日零点到当前时间的秒数。

random_bytes 为 28 个字节的随机数。

3) session_id

客户端在连接中使用的会话标识，定义为：

opaque SessionID<0..32>

session_id 是一个可变长字段，其值由服务端决定。如果没有可重用的会话标识或希望协商安全参数，该字段应为空，否则表示客户端希望重用该会话。这个会话标识可能是之前的连接标识、当前连接标识、或其它处于连接状态的连接标识。会话标识生成后应一直保持到被超时删除或与这个会话相关的连接遇到致命错误被关闭。一个会话失效或被关闭时则与其相关的连接都应被强制关闭。

4) cipher_suites

客户端所支持的密码套件列表，客户端应按照密码套件使用的优先级顺序排列，优先级最高的密码套件应排在首位。如果会话标识字段不为空，本字段应至少包含将重用的会话所使用的密码套件。

密码套件定义如下：

uint8 CipherSuite[2];

每个密码套件包括一个密钥交换算法，一个加密算法及密钥长度，和一个校验算法。服务端将在密码套件列表中选择一个与之匹配的密码套件，如果没有可匹配的密码套件，应返回握手失败报警消息 handshake_failure 并且关闭连接。

本规范支持的密码套件列表如表 2：

表 2 密码套件列表

序号	名称	值
1	ECDHE_SM1_SM3	{0xe0,0x01}
2	ECDHE_SM1_SHA1	{0xe0,0x02}
3	ECC_SM1_SM3	{0xe0,0x03}
4	ECC_SM1_SHA1	{0xe0,0x04}
5	IBSDH_SM1_SM3	{0xe0,0x05}
6	IBSDH_SM1_SHA1	{0xe0,0x06}
7	IBC_SM1_SM3	{0xe0,0x07}
8	IBC_SM1_SHA1	{0xe0,0x08}
9	RSA_SM1_SM3	{0xe0,0x09}
10	RSA_SM1_SHA1	{0xe0,0x0a}

在本规范中实现 ECC 和 ECDHE 的算法为 SM2；实现 IBC 和 IBSDH 的算法为 SM9。

5) compression_methods

客户端所支持的压缩算法列表，客户端应按照压缩算法使用的优先级顺序排列，优先级最高的压缩算法应排在首位。

定义如下：

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

服务端将在压缩算法列表中选择一个与之匹配的压缩算法。列表中必须包含空压缩算法，这样客户端和服务端总能协商出一致的压缩算法。

6.4.4.1.3 Server Hello 消息

该消息为服务端 hello 消息。

如果能从客户端 hello 消息中找到匹配的密码套件，服务端发送这个消息作为对客户端 hello 消息的回复。如果找不到匹配的密码套件，服务端将回应 handshake failure 报警消息。

服务端 hello 消息结构定义如下:

```
struct {
    ProtocolVersion server_version;
    Random random;
    SessionID session_id;
    CipherSuite cipher_suite;
    CompressionMethod compression_method;
} ServerHello;
```

其中：

1) server_version

该字段表示服务端支持的协议版本。本规范的版本号是 1.0。

2) random

服务端产生的随机数。

3) session_id

服务端使用的会话标识，如果客户端 hello 消息中的会话标识不为空，且服务端存在匹配的会话标识，则服务端重用与该标识对应的会话建立新连接，并在回应的服务端 hello 消息中带上与客户端一致的会话标识，否则服务端产生一个新的会话标识，用来建立一个新的会话。

4) cipher_suite

服务端从客户端 hello 消息中选取的一个密码套件。对于重用的会话，本字段存放重用会话使用的密码套件。

5) compression_method

服务端从客户端 hello 消息中选取的一个压缩算法，对于重用的会话，本字段存放重用会话使用的压缩算法。

6.4.4.2 Server Certificate 消息

该消息为服务端证书消息。

服务端必须发送一个服务端证书消息给客户端，该消息总是紧跟在服务端 hello 消息之后。当选中的密码套件使用 RSA 或 ECC 或 ECDHE 算法时，本消息的内容为服务端的数字证书；当选中的密码套件使用 IBC 或 IBSDH 算法时，本消息的内容为服务端标识和 IBC 公共参数，用于客户端与服务端协商 IBC 公开参数。

证书格式为 X.509 v3，证书类型必须能适用于已经确定的密钥交换算法。密钥交换算法与证书密钥类型的关系如表 3：

表 3 密钥交换算法与证书密钥类型关系表

密钥交换算法	证书密钥类型
RSA	RSA 公钥；证书必须允许该公钥用于加密。
IBC	服务端标识和 IBC 公共参数
IBSDH	服务端标识和 IBC 公共参数
ECC	ECC 公钥
ECDHE	ECC 公钥

对于 RSA 证书消息结构如下：

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

certificate_list

证书链。在证书链中，发送方的证书放在第一个，其后的每一个证书都应能直接认证它前面的一个证书。

IBC 标识及公共参数结构：

```
opaque ASN.1IBCParam<1..2^24-1>;
struct {
     opaque ibc_id<1..2^16-1>;
  ASN.1IBCParam ibc_parameter;
} Certificate;
```

其中：

1) ibc_id

服务端标识

2) ibc_parameter

IBC 公共参数，遵循 ASN.1 编码。

6.4.4.3 Server Key Exchange 消息

本消息为服务端密钥交换消息。

如果选择的密码套件使用了 ECDHE 或 IBSDH 进行密钥交换，服务端应在服务端证书消息之后发送此消息。

本消息传送的信息用于客户端计算预主密钥。

服务端密钥交换消息结构定义如下：

```
enum { ECDHE, IBSDH} KeyExchangeAlgorithm;
  struct {
      select (KeyExchangeAlgorithm) {
            case ECDHE:
               Opaque ServerECDHEParams<1..2^16-1>;
             case IBSDH:
                Opaque ServerIBSDHParams<1..2^16-1>;
      };
   } ServerKeyExchange;
```

其中：

1) ServerECDHEParams

使用 ECDHE 算法时，服务端的密钥交换参数

2) ServerIBSDHParams

使用 IBSDH 算法时，服务端的密钥交换参数

6.4.4.4 Certificate Request 消息

本消息为证书请求消息。

如果服务端要求认证客户端，则应发送此消息，要求客户端发送自己的证书。

如果有服务端密钥交换消息的话，该消息紧跟在服务端密钥交换消息之后；否则，该消息紧跟在服务端证书消息之后。

证书请求消息的结构定义如下：

```
struct {
```

```
    ClientCertificateType certificate_types<1..2^8-1>;
    DistinguishedName certificate_authorities<0..2^16-1>;
  } CertificateRequest;
```

其中：

1) certificate_types

要求客户端提供的证书类型的列表。

```
enum {
   rsa_sign(1), ecdsa_sign(64), ibc_params(80), (255)
   } ClientCertificateType;
```

2) certificate_authorities

如果 ClientCertificateType 是 ibc_params，本字段的内容是 IBC 密钥管理中心的信任域名列表。否则是服务端信任的 CA 的证书 DN 列表，包括根 CA 或者二级 CA 的 DN。

定义如下：

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

6.4.4.5 Server Hello Done 消息

本消息为服务端 hello 完成消息。

表示握手过程的 hello 消息阶段完成。发送完该消息后服务端会等待客户端的响应消息。

客户端接收到服务端的 hello 完成消息之后，应验证服务端证书是否有效，并检验服务端的 hello 消息参数是否可以接受。如果可以接受，客户端继续握手过程。否则发送一个 Handshake failure 致命报警。

服务端 hello 完成消息结构如下：

```
    struct { } ServerHelloDone;
```

6.4.4.6 Client Certificate 消息

本消息为客户端证书消息。

客户端接收到服务端 hello 完成消息后，如果服务端请求客户端证书，客户端要随后发送本消息。如果没有服务端要求的证书，则本消息中不包含任何证书，在这种情况下，如果服务端要求认证客户端，将导致一个 Handshake failure 致命报警。

如果协商的密码套件使用 IBC 或 IBSDH 算法，此消息的内容为客户端标识和 IBC 公共参数，用于客户端与服务端协商 IBC 公开参数。

客户端证书消息的结构同 6.4.4.2 定义的结构。

6.4.4.7 Client Key Exchange 消息

本消息为客户端密钥交换消息。

如果服务端请求客户端证书，本消息紧跟于客户端证书消息之后，否则本消息是客户端接收到服务端 hello 完成消息后所发送的第一条消息。

如果密钥交换算法使用 RSA 算法、ECC 算法和 IBC 算法，本消息中包含预主密钥，该预主密钥由客户端产生，采用服务端的公钥进行加密。当服务端收到加密后的预主密钥后，利用相应的私钥进行解密，获取所述预主密钥的明文。如果是 IBC 算法，客户端利用获取的服务端标识和 IBC 公开参数，产生服务端公钥。如果是 RSA 算法，建议使用 PKCS#1 版本 1.5 对 RSA 加密后的密文进行编码。如果密钥交换算法使用 ECDHE 算法或 IBSDH 算法，本消息中包含计算预主密钥的客户端密钥交换参数。

客户端密钥交换消息结构定义如下：

```
struct {
 select (KeyExchangeAlgorithm) {
      case ec_key_agreement:
            Opaque ClientECDHEParams<1..2^16-1>;
```

```
        case ibs_key_agreement:
                Opaque ClientIBSDHParams<1..2^16-1>;
        case ecc_encryption:
                opaque ECCEncryptedPreMasterSecret<0..2^16-1>;
        case ibe:
                opaque IBEEncryptedPreMasterSecret<0..2^16-1>;
        case rsa:
                opaque RSAEncryptedPreMasterSecret<0..2^16-1>;

    } exchange_keys;
  } ClientKeyExchange;
```

其中：

1) ClientECDHEParams：
   使用 ECDHE 算法时，客户端的密钥交换参数
2) ClientIBSDHParams：
   使用 IBSDH 算法时，客户端的密钥交换参数
3) ECCEncryptedPreMasterSecret
   使用 ECC 加密算法时，用服务端公钥加密的预主密钥。
4) IBEEncryptedPreMasterSecret
   使用 IBC 加密算法时，用服务端公钥加密的预主密钥。
5) RSAEncryptedPreMasterSecret
   使用 RSA 加密算法时，用服务端公钥加密的预主密钥。

预主密钥的数据结构如下：

```
struct {
       ProtocolVersion client_version;
       opaque random[46];
} PreMasterSecret;
```

其中：

1) client_version
   客户端所支持的版本号。服务端要检查这个值是否跟客户端 hello 消息中所发送的值相匹配。
2) random
   46 字节的随机数。

6.4.4.8 Certificate Verify 消息

本消息为证书校验消息。

该消息用于鉴别客户端是否为证书的合法持有者，紧跟于客户端密钥交换消息之后。

证书校验消息的数据结构如下：

```
struct {
     Signature signature;
} CertificateVerify;
```

其中：

Signature 的结构如下：

```
enum { rsa_sha1,rsa_sm3,
       ecc_sha1,ecc_sm3,
       ibs_sha1,ibs_sm3 } SignatureAlgorithm;
select (SignatureAlgorithm)
```

```
{
    case rsa_sha1:
        digitally-signed struct {
            opaque sha1_hash[20];
        };
    case rsa_sm3:
        digitally-signed struct {
            opaque sm3_hash[20];
        };
  case ecc_sha1:
        digitally-signed struct {
            opaque sha1_hash[20];
        };
  case ecc_sm3:
        digitally-signed struct {
            opaque sm3_hash[20];
        };
  case ibs_sha1:
        digitally-signed struct {
            opaque sha1_hash[20];
        };
  case ibs_sm3:
        digitally-signed struct {
            opaque sm3_hash[20];
        };
} Signature;
```

sm3_hash 和 sha1_hash 是指 hash 运算的结果，运算的内容是自客户端 hello 消息开始直到本消息为止（不包括本消息）的所有与握手有关的消息，包括握手消息的类型和长度域。

6.4.4.9 Finished 消息

本消息为握手结束消息。

服务端和客户端各自在密码规格变更消息之后发送本消息，用于验证密钥交换过程是否成功，并校验握手过程的完整性。

本消息用本次握手过程协商出的算法和密钥保护。

本消息的接收方必须检验消息内容的正确性。一旦一方发送了握手结束消息，并且接收到了对方的握手结束消息并通过校验，就可以使用该连接进行数据安全传输。

握手结束消息数据结构如下：

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

其中：

verify_data 为校验数据，该数据产生方法如下：

PRF(master_secret, finished_label, SHA1 (handshake_messages) + SM3(handshake_messages)) [0..11];

表达式中：

1) finished_label

对于由客户端发送的结束消息，该标签是字符串“client finished”。对于服务端，该标签是字符串“server finished”。

2) handshake_messages

指自客户端 hello 消息开始直到本消息为止(不包括本消息、密码规格变更消息和 hello 请求消息)的所有与握手有关的消息，包括握手消息的类型和长度域。

6.5 密钥计算

6.5.1 主密钥计算

主密钥由 48 个字节组成，由预主密钥、客户端随机数、服务端随机数、常量字符串，经 PRF 计算生成。

计算方法如下：

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

6.5.2 工作密钥

工作密钥包括校验密钥和加密密钥，具体密钥长度由选用的密码算法决定。由主密钥、客户端随机数、服务端随机数、常量字符串，经 PRF 计算生成。

计算方法如下：

```
key_block = PRF(SecurityParameters.master_secret, "key expansion",
            SecurityParameters.server_random +SecurityParameters.client_random);
```

直到生成所需长度的输出，然后按顺序分割得到所需的密钥：

```
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]
```

6.6 网关到网关协议

6.6.1 概述

网关到网关协议定义了SSL VPN之间建立网关到网关的传输层隧道，对IP数据报文进行安全传输时所采用的报文格式（包括控制报文与数据报文），以及控制报文交换过程和数据报文封装过程。网关到网关协议采用记录层协议作为底层的承载协议，和握手协议在同一层次，使用80作为协议类型编号，详见6.3.2.1。网关到网关协议在当前的连接状态之下进行传输数据的处理。

6.6.2 协议报文格式

网关到网关协议包括两种协议报文，控制报文和数据报文。定义如下：

```
enum { command (0) , data (1) , (255) } Site2SiteContentType;
struct {
        Site2SiteContentType type;
        uint16 length;
        select (type) {
            case command: Site2SiteCmd;
            case data: Site2SiteData;
        } Site2Sitefragment;
} Site2Site;
```

其中：

1) type

网关到网关载荷数据类型，0为控制报文，1为数据报文。

2) length

网关到网关载荷的长度，不包括类型域和长度域。

3) Site2SiteCmd

网关到网关控制报文，定义了保护域的信息交换报文。定义如下：

```
enum { cmd_area (0) , (255) } CmdType;
struct {
    CmdType type;
    select (type) {
       case  cmdarea:
                    Arealist arealist<0.. 2^14>;
    }CmdFragment;
} Site2SiteCmd;
```

其中：

AreaList

SSL VPN 保护域列表，包括网络层保护域和传输层保护域。网络层保护域可以采用子网与子网掩码、网络地址范围这两种方式表示。定义如下：

```
enum {subnet_ipv4(0),subnet_ipv6(1),range_ipv4(2),range_ipv6(3),(255)} AreaType;
```

传输层保护域由传输层协议和传输层端口来表示。传输层端口为 0 代表任意端口（TCP/UDP）或任意类型（ICMP）。定义如下：

```
struct {
    uint8  Protocol;
    uint16 port;
} TlArea;

struct {
AreaType type;
    select (type){
        case  subnet_ipv4: ipv4_addr_subnet;
        case  subnet_ipv6: ipv6_addr_subnet;
        case  range_ipv4：ipv4_addr_range;
        case  range_ipv6：ipv6_addr_range;
    }NlAreaFragment;
        TlArea  TlAreaFragment; }AreaList;
```

其中:

TlArea

传输层保护域

ipv4_addr_subnet

表示 ipv4 的子网地址。由两个 32 位的值组成，第一个值表示子网的 IP 地址，第二个值表示子网的网络掩码。网络掩码中的比特 1 表示该子网的 IP 地址中对应位是固定值，比特 0 表示对应位是通配值。

ipv4_addr_range

表示 ipv4 的地址范围。由两个 32 位的值组成，第一个值表示地址范围的起始地址，第二个值表示地址范围的结束地址。落在这两个地址区间的所有 IP 地址都被认为属于该地址范围之内。

ipv6_addr_subnet

表示 ipv6 的子网地址。由一个 128 位和一个 8 位的值构成。第一个值表示 ipv6 子网地址，第二个值表示子网地址的前缀长度。该长度表示 128 位的子网地址的前若干位为固定值，剩下的后若干位为通配值。

ipv6_addr_range

表示 ipv6 的地址范围。由两个 128 位的值组成，第一个值表示地址范围的起始地址，第二个值表示地址范围的结束地址。落在这两个地址区间的所有 IP 地址都被认为属于该地址范围之内。

4) Site2SiteData

网关到网关数据报文，内容为原始的完整IP报文。

6.6.3 控制报文交换过程

控制报文的交换过程由通讯双方的任何一方发起，通知对方自己所保护的网络信息。控制报文在握手协议完成之后的任何时间都可以发送，使用握手协议协商好的安全参数保护。

在网关到网关的数据报文传输过程中，如果接收到新的传递保护域的控制报文，将触发该连接的重用过程，详见 6.4.3。

6.6.4 数据报文出入站过程

SSL VPN网关之间通过握手协议建立SSL连接，并将该连接与本地和对端的保护域进行绑定，出入站的IP报文需要与对端和本地的保护域进行匹配。本地和对端保护域可以通过控制报文获取，也可以人工配置。

6.6.4.1 出站报文处理

本地保护域内的 IP 报文在通过 SSL VPN 转发时，SSL VPN 根据该报文的目的 IP 地址和源 IP 地址等信息查找匹配的对端保护域及本地保护域，获取相应的有效 SSL 连接。然后将整个 IP 报文作为记录层的数据内容封装在该连接的记录层之上，通过记录层传送，封装格式详见 6.3.2。如果没有相应的有效连接，则触发新的握手过程或者该连接的重用过程，详见 6.4.3，协商建立或者更新相应的连接。如果查找不到相应的保护域，根据本地安全策略丢弃该报文或者进行其他处理。

6.6.4.2 入站报文处理

对入站的数据报文进行解密、校验、解压缩操作，然后对于恢复出的 IP 报文，首先根据目的 IP 地址和源 IP 地址等信息查找匹配的本地保护域及对端保护域，并检查和该 SSL 连接的绑定关系是否一致，对于查找到对应保护域并符合绑定关系的 IP 报文，进行路由和转发。如果查找不到相应的保护域或者不符合绑定关系的 IP 报文，根据本地安全策略丢弃该报文或者进行其他处理。

7 产品要求

7.1 产品功能要求

7.1.1 工作模式

SSL VPN 产品工作模式分为客户端-服务端模式和网关-网关模式两种。其中客户端-服务端模式是必备模式，网关-网关模式是可选模式，两种模式都应按照本规范第 6 章的要求正确实现。

7.1.2 随机数生成

SSL VPN 产品应具有随机数生成功能，在使用随机数前能对生成的随机数进行偏“0”、偏“1”、“0、1”平衡等常规检测，并提供检测接口，能通过检测接口对 SSL VPN 产品所生成的随机数进行样本采集。

7.1.3 密钥协商

SSL VPN产品应具有密钥协商功能，通过协商产生工作密钥。
密钥协商应按照本规范6.4的要求进行。

7.1.4 安全报文传输

SSL VPN产品具有安全报文传输功能，保证数据的安全传输。
安全报文传输应按照本规范6.3的要求进行。

7.1.5 身份鉴别

SSL VPN产品应具有实体鉴别的功能，服务端的鉴别是必备功能，客户端的鉴别是可选功能，应支持基于数字证书（RSA或ECC）或者基于标识算法的鉴别机制。任何一种鉴别方式都需要保证鉴别的完整性和有效性。

7.1.6 访问控制

SSL VPN产品应具有细粒度的访问控制功能，基于用户或用户组对资源进行有效控制。其中对网络访问至少应控制到IP地址、端口，对Web资源的访问至少应控制到URL，并能根据访问时间进行控制。

7.1.7 密钥更新

SSL VPN产品应具有根据时间周期或报文流量进行工作密钥更新的功能。其中，根据时间周期进行更新为必备功能，根据报文流量进行更新为可选功能。根据时间周期进行更新的情况下，客户端-服务端模式最长时间不超过8小时，网关-网关模式最长时间不超过1小时。密钥更新后SessionID不变。

7.1.8 客户端主机安全检查

SSL VPN产品应具有客户端主机安全检查功能。客户端在连接服务端时，根据服务端下发的客户端安全策略检查用户操作系统的安全性。不符合安全策略的用户将无法使用SSL VPN。
客户端安全策略应至少包括以下条件之一：
--是否已安装并启用反病毒软件
--是否已安装并启用个人防火墙
--是否已安装最新的操作系统安全补丁
--是否已为系统设置了登录口令

7.2 产品性能要求

7.2.1 最大并发用户数

同时在线用户的最大数目，此指标反映产品能够同时提供服务的最大用户数量。

7.2.2 最大并发连接数

同时在线SSL连接的最大数目，此指标反映产品能够同时处理的最大SSL连接数量。

7.2.3 每秒新建连接数

每秒钟可以新建的最大SSL连接数目，此指标反映产品每秒能够接入新SSL连接的能力。

7.2.4 吞吐率

在丢包率为0的条件下，服务端产品在内网口上达到的双向数据最大流量。

7.3 安全管理要求

7.3.1 安全要求

7.3.1.1 密钥安全

7.3.1.1.1 服务端密钥

服务端密钥由产品自身产生时，其公钥应能被导出。由外部产生时，应有安全措施保证私钥在产生、存储、分发和导入时的安全。密钥存储时要求保证机密性和完整性，并保证服务端私钥不在服务端外使用。密钥分发过程不应泄露私钥的任何一个部分，要确保服务端在此之前没有遭受到可能导致密钥或敏感数据泄露的破坏时，才能将密钥导入到服务端中。

服务端密钥应保存在非易失性存储器中，并且能够按设定的安全策略进行更新，密钥更新的时间应小于成功实施攻击的时间。服务端被更换的密钥不允许再被激活使用并要求销毁。

服务端密钥应能以安全形式进行备份，并在需要时能够恢复。

7.3.1.1.2 工作密钥

工作密钥产生后应保存在易失性存储器中，达到其更新条件后应立即更换，在连接断开、设备断电时应销毁。

7.3.1.2 配置数据安全

所有的配置数据应保证其在设备中的完整性、可靠性。应有管理界面对配置数据进行配置和管理，管理员进入管理界面应通过身份鉴别。

7.3.1.2.1 硬件安全

SSL VPN产品应提供安全措施，保证密码算法、密钥、关键数据的存储安全。

所有密码运算应在独立的密码部件中进行。

除必需的通信接口和管理接口以外，不提供任何可供调试、跟踪的外部接口。内部的调试、检测接口应在产品定型后封闭。

7.3.1.2.2 软件安全

所有的安全协议及管理软件应自主实现。

操作系统应进行安全加固，关闭所有不需要的端口和服务。

任何操作指令及其任意组合，不能泄露密钥和敏感信息。

7.3.1.2.3 客户端安全

SSL VPN客户端产品应具有完整性的自校验功能，包括厂商对客户端软件的签名，以保护完整性。

7.3.2 管理要求

7.3.2.1 日志管理

SSL VPN产品应提供日志记录、查看和导出功能。

日志内容包括：

——管理员操作行为，包括登录认证、系统配置、密钥管理等操作。

——用户访问行为，包括用户、时间、访问资源、结果等。

——异常事件，包括认证失败、非法访问等异常事件的记录。

7.3.2.2 管理员管理

SSL VPN服务端产品应设置管理员，进行系统配置、密钥生成、导入、备份和恢复等操作。管理员应持有表征用户身份信息的硬件装置，与登录口令相结合登录系统，进行管理操作前应通过身份鉴别。

登录口令长度应不小于8个字符。

使用错误口令或非法身份登录的次数限制应小于或等于8。

7.3.2.3 设备管理

7.3.2.3.1 设备初始化

SSL VPN产品的初始化，除必须由厂商进行的操作外，系统配置、密钥的生成和管理、管理员的产生等均应由用户完成。

7.3.2.3.2 设备自检

应对密码运算部件等关键部件进行正确性检查。

应对存储的密钥等敏感信息进行完整性检查。

在检查不通过时应报警并停止工作。

8 产品检测

8.1 产品功能检测

8.1.1 工作模式

在客户端-服务端工作模式下，客户端应能通过服务端访问到受保护内网服务器。在网关-网关工作模式下，一个网关保护的客户主机应能访问到另一个网关保护的内网服务器。检测结果应符合本规范7.1.1的要求。

8.1.2 随机数功能

按照《随机数检测规范》的要求提取样本，并按照该规范的相关要求进行检测，检测结果应合格。

8.1.3 密钥协商

密钥协商协议应按照本规范6.4的要求进行。检测结果应符合本规范7.1.3的要求。

8.1.4 安全报文传输

安全报文封装协议应按照本规范6.3的要求进行。检测结果应符合本规范7.1.4的要求。

8.1.5 身份鉴别

身份鉴别应按照本规范6.4的要求进行。检测结果应符合本规范7.1.5的要求。

8.1.6 访问控制

从客户端访问服务端保护的内网服务器，应只能访问到授权的资源。检测结果应符合本规范7.1.6的要求。

8.1.7 密钥更新

密钥更新应按照本规范6.4的要求进行。检测结果应符合本规范7.1.7的要求。

8.2 产品性能检测

8.2.1 最大并发用户数

在检测平台模拟多个客户端行为，与服务端建立SSL会话，在这个会话上，从内网服务器下载512字节页面的数据，并在内网服务器上设置页面延迟，以保证在整个负载增加的过程中每一个会话均被保持且有数据通过。然后，不断增加客户端，并重复此过程，取负载稳定期的平均并发会话数作为测试结果。检测结果应符合本规范7.2.1的要求。

8.2.2 最大并发连接数

在检测平台模拟多个客户端行为，与服务端进行SSL连接并保持，然后不断增加客户端，并重复此过程，直到无法建立并保持连接为止。取已经接入的SSL连接数目为测试结果。检测结果应符合本规范7.2.2的要求。

8.2.3 每秒新建连接数

在检测平台模拟多个客户端行为，并发与服务端建立SSL会话。重复此过程一段时间，取每秒建立SSL会话数目的平均值作为测试结果。检测结果应符合本规范7.2.3的要求。

8.2.4 吞吐率

在检测平台模拟多个客户端行为，与服务端建立SSL会话。在这个会话上，从内网服务器下载1MB数据，重复以上步骤，直到每个用户成功下载20MB大小的数据。然后向内网服务器上传1MB数据，重复以上步骤，直到每个用户成功上传20MB大小的数据。取内网服务器收发数据的平均速率作为测试结果。检测结果应符合本规范7.2.5的要求。

8.3 安全管理检测

8.3.1 安全检测

8.3.1.1 密钥安全

8.3.1.1.1 服务端密钥

检测结果应符合本规范7.3.1.1.1的要求。

8.3.1.1.2 工作密钥

检测结果应符合本规范7.3.1.1.2的要求。

8.3.1.2 配置数据安全

检测结果应符合本规范7.3.1.2的要求。

8.3.1.2.1 硬件安全

检测结果应符合本规范7.3.1.2.1的要求。

8.3.1.2.2 软件安全

检测结果应符合本规范7.3.1.2.2的要求。

8.3.1.2.3 客户端安全

检测结果应符合本规范7.3.1.2.3的要求。

8.3.2 管理检测

8.3.2.1 日志管理

检测结果应符合本规范7.3.2.1的要求。

8.3.2.2 管理员管理

检测结果应符合本规范7.3.2.2的要求。

8.3.2.3 设备管理

8.3.2.3.1 设备初始化

检测结果应符合本规范7.3.2.3.1的要求。

8.3.2.3.2 设备自检

检测结果应符合本规范7.3.2.3.2的要求。

9 合格判定

本规范中，7.1 以及 7.3 中除 7.3.1.2（配置数据安全）以外的各项要求中，其任意一项要求不合格，判定为产品不合格。