DTLS 协议调研

调研人：许敏超
调研日期：2020.6

1. 摘要

通过调研发现，DTLS 是基于 UDP，位于传输层与应用层之间的协议，作用是为给 UDP 提供端到端的安全通道。DTLS 协议是一个两层结构，位于底层的是 DTLS 记录协议，用于封装和传输上层的协议和应用数据。位于记录协议之上的是握手协议、警告协议以及修改密码规范协议。其中最重要的是握手协议，它负责 DTLS 连接建立以及双方通信过程中安全参数的协商。告警协议和修改密码规范协议主要用来对 DTLS 连接交换进行管理。

DTLS 协议能够保证基于数据报协议的连接安全，这主要体现在几个方面：第一，通过对通信过程中传输的数据进行加密来保证数据的机密性；第二，通过计算数据的 MAC 值以及验证过程来保证数据的完整性；第三，通过身份认证技术来验证通信双方的身份以保证两个正确的通信实体进行通信；第四，通过握手时的 cookie 机制来抗 Dos 攻击等。

2. 调研目的

调研分析 DTLS 协议的原理、分层模型、握手协议以及安全性，以便更好的分析和利用网络中的 DTLS 协议包。

3. 调研方法

通过查询相关论文、论坛以及 RFC 标准等资料来获取相关内容。

4. 调研内容

本文主要调研的内容包括：DTLS 协议的由来、分层结构、安全性以及应用。

其中着重分析的是握手过程以及协议安全性。

4.1. DTLS 协议概述

互联网先驱们最开始在设计互联网协议时主要考虑的是可用性，安全性是较少考虑在其中的，所以传输层的 TCP、UDP 协议本身安全性较差。SSL/TLS 协议是基于 TCP socket，在传输层和应用层之间构建了一个端到端的安全通道，保证了传输数据的加密性。由于基于 UDP 协议的应用逐渐普遍，伴随着这些应用的安全问题也越来越受瞩目。TLS 协议可以保证基于 TCP 的应用的安全，但是 TLS 协议不能用来保证基于 UDP 协议的应用安全。为了弥补 TLS 协议的这种缺点，保证基于 UDP 协议之上的应用安全，对 TLS 协议进行扩展，使之支持 UDP 协议，这时提出了 DTLS（Datagram Transport Layer Security）协议。

4.1.1. DTLS 协议发展历史

1994 年年末，网景公司设计了 SSL 1.0 版（并未对外发布），并更新为 2.0 版应用在自家开发的浏览器上。但因为网景公司在设计协议时闭门造车，使得该版本很快爆出不少漏洞，例如：

（1） 身份认证存在漏洞，易受到中间人攻击，导致客户端与服务器之间约定的密码套件被恶意篡改；

（2） 消息认证码（MAC）算法和 MD5 算法不够强力安全；

（3） 客户端不能要求修改加密算法及更换密钥；

（4） 单一服务，且证书被绑定了固定域名，无法满足当时服务器开发流行的虚拟主机技术。

诸多问题使得网景公司不得不重新设计协议，并在 1995 年发布了 SSL 3.0 版。1996 年，SSL 的设计加入了国际互联网工程任务组，并由专门成立的 TLS 工作组负责后续版本工作。

1999 年，基于 SSL 3.0 的 TLS 1.0 正式出炉，两者差别不显著，区别在于：

（1） 密钥导出算法不同；

（2） 消息认证码算法不同；

（3） 握手中的 Finished 消息内容有变化；

（4） 较之 SSL 3.0 有着更多的警告消息；

（5） TLS 1.0 需要 DSS 和 Diffie-Hellman（DH）支持。

2006 年，TLS 1.1 问世，修复了 TLS 1.0 存在的安全漏洞，总体的变化如下：

（1） 使用显式而非隐式的初始向量 IV 来防止 Cipher Block Chaining（CBC）攻击；

（2） 对填充错误的处理方式有变化；

（3） 提供参数以支持 IANA 注册。

同年，DTLS 1.0 版本问世，其规范基于 TLS 1.1。之前的 SSL 协议不能为 UDP 提供加密传输服务，而 DTLS 的出现填补了这一项空白。

2008 年，TLS 1.2 发布，相比旧版本做出了更多修改，例如：

（1） 伪随机数产生器和握手 Finished 消息中使用的算法组合从 MD5-SHA-1 变成了 SHA-256；

（2） 数字签名中使用的算法组合从 MD5-SHA-1 变成了 SHA-1；

（3） 服务器和客户端可更自主地选择可接受的 MAC 算法和加密算法；

（4） 支持对额外数据模型进行认证加密。

2012 年，基于 TLS 1.2 的 DTLS 1.2 发布（没有 DTLS 1.1 这个版本）。

TLS 最新的 1.3 版的草案在 2018 年 1 月发布，相比 TLS 1.2 做出了不少改动：

（1） 修改了握手协议，支持网络往返更少的 1-RTT 及 0-RTT 密钥协商。

（2） 完全禁止了压缩行为和 RC4 算法。

（3） 完全禁止了包括 RSA 在内的不安全的密钥协商算法。

（4） 只使用 AEAD 作为消息验证码算法，并取消了其显式 IV 及长度域。

（5） 去除了 ChangeCipherSpec 消息。

（6） 不允许重新协商。

TLS 1.3 这些举措不仅修正了旧版本的潜在漏洞，还为需要低延迟的移动互联网提供了更多支持。而 DTLS 1.0 和 1.2 版分别基于 TLS 1.1 和 1.2 版，

因此本文不讨论 TLS 1.3 的内容。

DTLS 协议是基于 UDP 协议，位于传输层与应用层之间，DTLS 协议在传输数据模型中的位置如图 4.1 所示。DTLS 的作用为给 UDP 提供端到端的安全通道，就像 SSL/TLS 协议对 TCP 的作用一样。

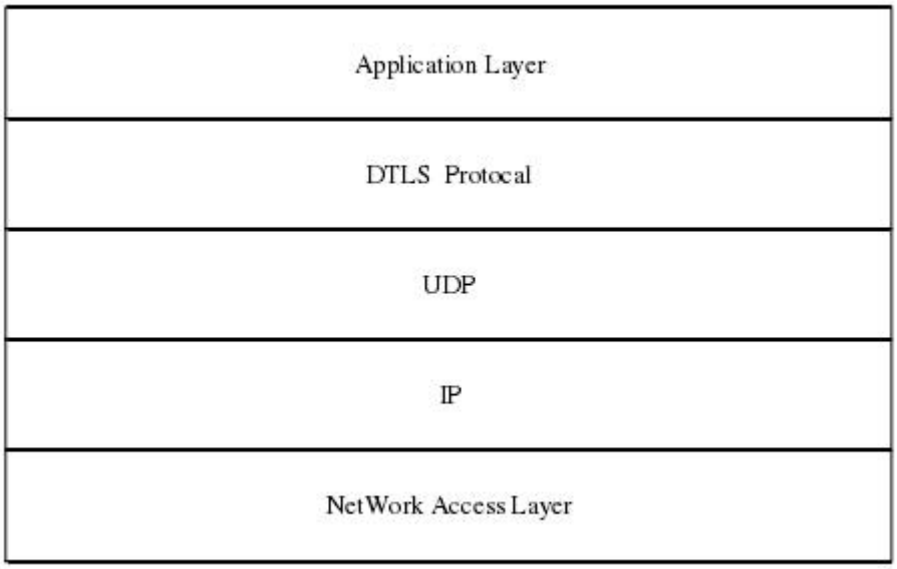

图 4.1 DTLS 层次结构图

4.1.2. 协议结构

DTLS 是一个两层协议，位于底层的是 DTLS 记录协议（Record Protocol），它是一个封装协议，用于封装和传输上层的协议和应用数据，以保证 DTLS 连接的安全性。位于记录协议之上的是握手协议(Handshake Protocol)、警告协议(Alert Protocol)以及修改密码规范协议(Change Cipher Spec Protocol)。其中最重要的是握手协议，它负责 DTLS 连接建立以及双方通信过程中安全参数的协商。告警协议和修改密码规范协议主要用来对 DTLS 连接交换进行管理。其组成如图 4.2 所示。

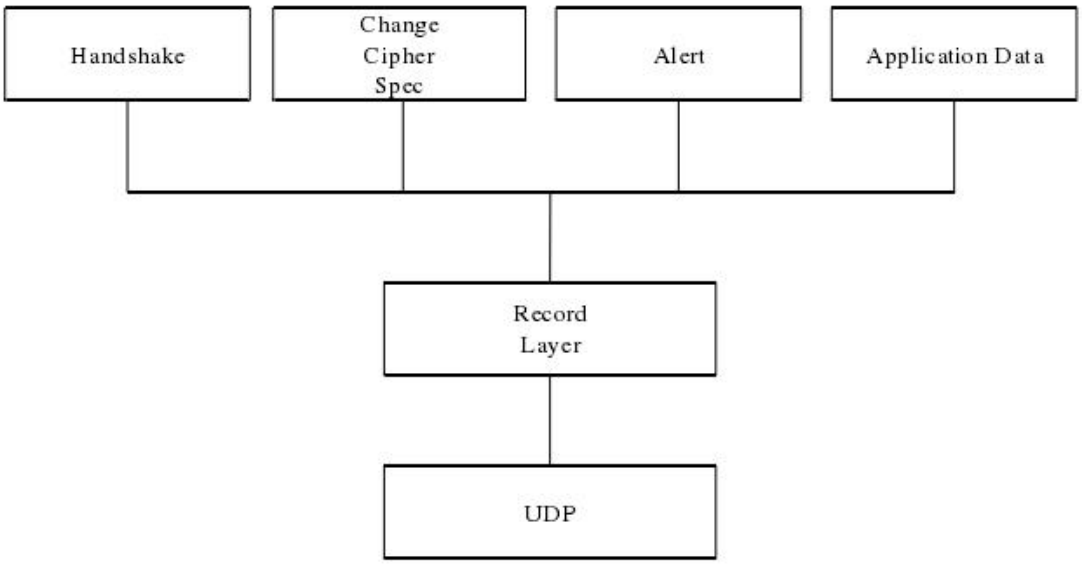

图 4.2 DTLS 协议分层结构

4.2. DTLS 协议分层结构

4.2.1. 记录层协议

DTLS 记录层协议首先将记录进行分段，段长由程序本身来指定，不考虑数据的类型，只负责对数据进行加工、发送和接收。在发送的时候，记录层接收应用层传来的数据后，对所要发送的数据包进行分段、压缩、添加 MAC、加密、添加 DTLS 包头处理，最后将 DTLS 数据包传给最底层 UDP 层，并加上 UDP 包头。DTLS 记录层接收数据包过程与发送过程正好相反，即解密、对 MAC 验证、解压缩、组装分段，然后发送给应用层用户。DTLS 记录层数据处理过程如图 4.3 所示。

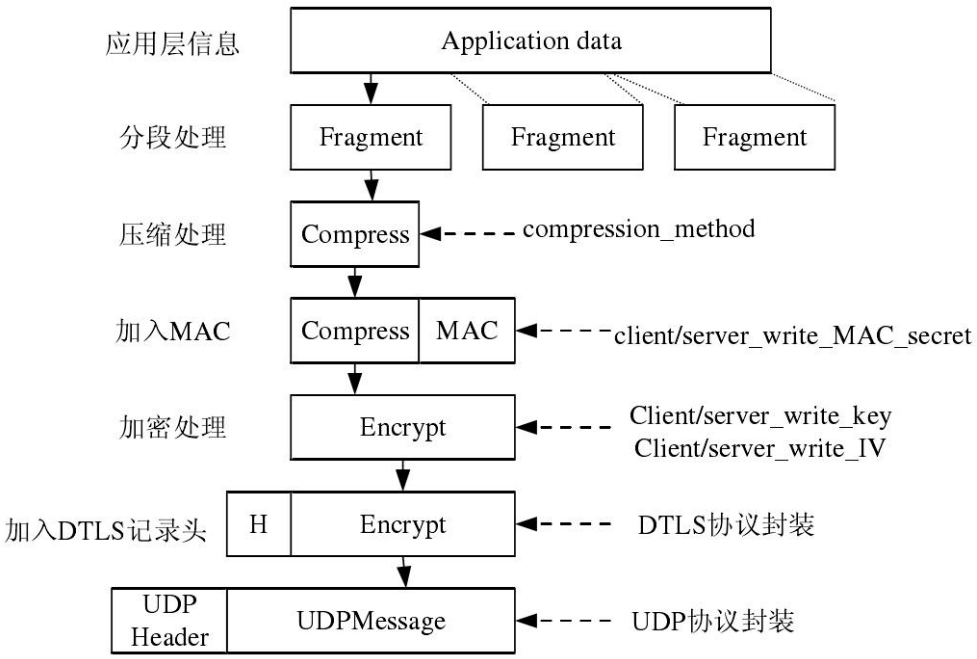

图 4.3 DTLS 协议记录层数据处理过程

以下对该过程进行详细介绍。

（1）分段处理

TLS 握手消息比较小，可能多个相同类型的消息封装在同一块记录中。但是不同类型的消息内容还是会被分别处理到不同的记录块中。而 DTLS 握手消息太大，它不同于 TLS 协议可能将几个消息封装到一个记录中，DTLS 会把一个消息封装到若干个记录中。因此 DTLS 记录层把上层传输的数据包以 16KB 为单位分成 DTLS 明文记录块，当分到最后一块时可能不足 16K，这时还以一个记录块来计算。

（2）压缩与解压缩

为了减少网络流量，需要对记录层数据进行压缩操作，这时使用握手协议中协商好的压缩算法进行压缩，当传输握手协议数据时压缩算法为空。只有当握手协议执行完后，才使用压缩算法进行压缩，且压缩后数据不能超过 1024B。当对方接收到传送的记录时，对数据进行解压缩。

（3）编码与解码

在握手刚开始时，这时还没有协商好加密算法和消息摘要算法，此时在记录层传输数据时是明文传输，没有任何安全保护，但是一旦握手成功，此时便协商好加密算法、加密密钥以及 MAC 算法，此时可以在记录层对消息进行计算消息

验证码(MAC)、加密操作和添加 MAC 操作，可以将 DTLS 明文记录块转化成 DTLS 密文记录块。解密处理则与以上过程相反，解密、对消息进行验证。

(4)添加 DTLS 记录头部

在最后一个步骤，对 DTLS 记录层协议添加一个头部，该头部由下面一些字段组成：

① 内容类型 type (8 比特)：有 4 种类型，分别是改变密码规格消息、握手消息、警告消息和应用程序数据包。

② 版本信息(16 比特)：表示所使用版本信息分为高低两个字节。DTLS 协议版本为{254，255}

③ epoche 是改变加密套件计数器，其默认值为 0，在每次执行改变密码套件操作时加 1。

④ sequence_number 记录的序列号，用来重组记录时使用，当每次发送 ChangeCiperSpec 消息后该序列号清 0。

记录层数据包格式如图 4.4 所示。

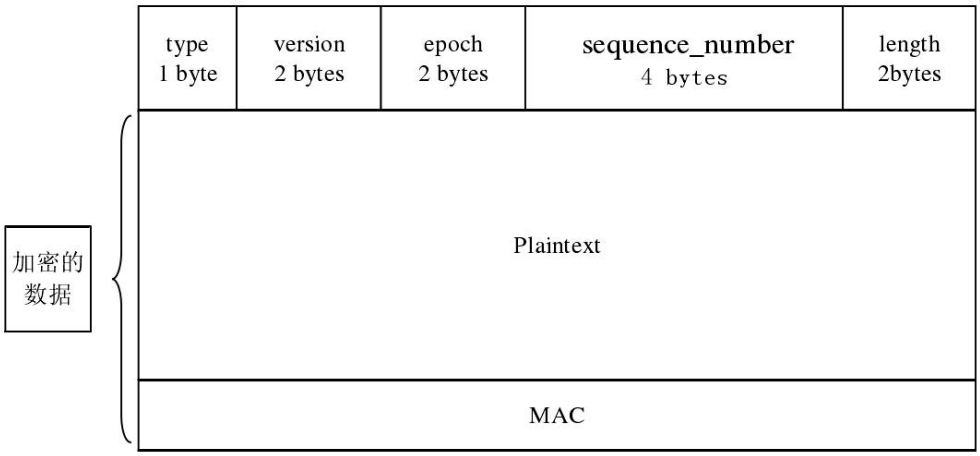

图 4.4 记录层数据包格式

4.2.2. 握手协议

相比较于 TLS 的握手协议，DTLS 的握手协议主要有以下三点改变：

(1)为处理消息丢失、乱序、分片和重组，DTLS 修改了 TLS 握手消息头格式（序号机制）；

(2)DTLS 握手协议采用定时器技术重传丢失或超时的握手消息；

(3)为了避免 DoS 攻击，DTLS 握手协议增添了无状态 cookie 交换技术。

除了以上三点不同之外，DTLS 握手协议的消息格式和实现流程均与 TLS 握手协议相同。以下将对这三种新机制展开介绍。

4.2.2.1.握手协议新机制

(1) DTLS 握手协议头格式

不同于 TLS 协议，DTLS 协议的握手消息长度太大，不像 TLS 协议可能将几个消息封装到一个记录中，因此要把一个消息封装到若干个记录中。由此说明要对 DTLS 握手消息进行分片和重组。为了分片和重组，相比较于 TLS 握手协议，DTLS 握手消息头部增添了 3 个新的字段。DTLS 握手消息头格式如图 4.5 所示。

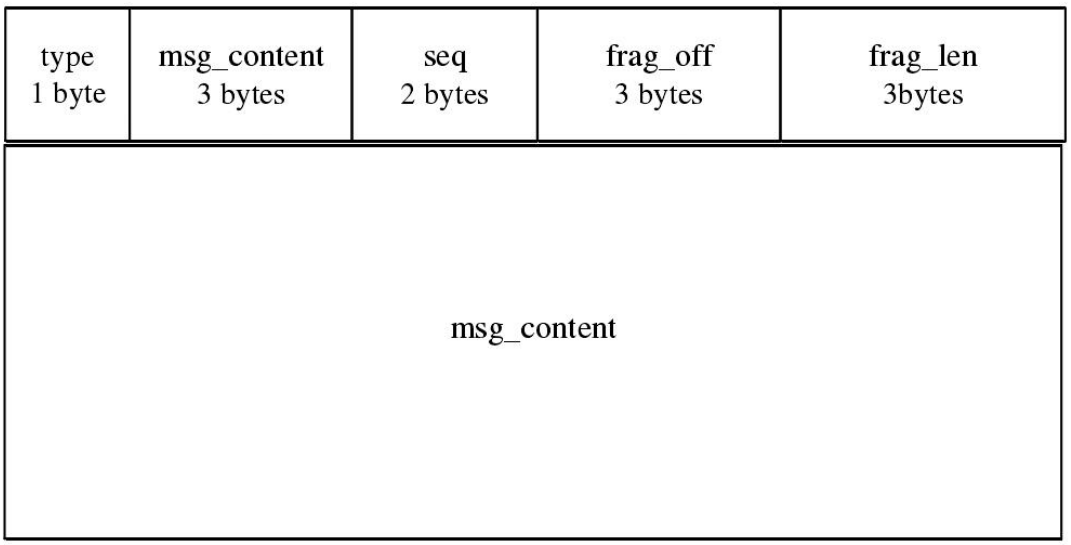

图 4.5 DTLS 握手消息头格式

其中：

① type 包含的类型如下所示，分别代表握手过程中的消息类型，具体含义见 4.2.2.2.节。

MASTER_SECRET_SIZE

CLIENT_HELLO _COOKIES

CLIENT_HELLO

SERVER_HELLO

HELLO _VERIY_REQUEST

CERTIFICATE

SERVER_KEY_EXCHANGE

CERTIFICATE_REQUEST

SERVER_HELLO_DONE

CERTIFICATE_VERIFY

CLIENT_KEY_EXCHANGE

FINISHED

WAIT_FOR_CHANGE_CIPHER_SPECS

WAIT_FOR_FINISHED

HANDSHAKE_FINISHED

② message_seq 字段长度为 2byte，表示所发送的握手消息的序列号。当握手消息丢失时便可以采用相同的序列号进行重传。

③ frag_offset 字段长度为 3byte 表示分片偏移量

④ frag_length 字段长度为 3byte 表示分片大小，即所包含的字节数。

(2)消息重传

由于 DTLS 协议是基于 UDP 之上的，消息在传输过程中可能会丢失，所以是不可靠的。为了解决这种情况，DTLS 协议采用接收确认这种传统的处理消息丢失的方法进行处理。比如 Cilent 发送一条含 cookie 状态 ClientHello 消息到 Server 的同时，Client 启动一个定时器用来计时，定时时间由自己在程序中定义，并希望在定时器超时前接收到服务器的 ServerHello 消息。如果在定时器超时之前未接收到 Server 的 ServerHello 消息，Client 则认为 ClientHello 消息丢失或者服务器端发送的 ServerHello 消息丢失，此时 Cilent 便重新发送 ClientHello 消息。服务器端处理情况与此类似。

(3)cookie 机制

DTLS 协议实现了无状态 cookie 技术。客户端首先发送未添加 cookie 技术的 ClientHello 消息给服务器端，当服务器接收后，便为客户端产生一个 cookie，并添加到 HelloVerifyReuqest 消息中，然后把添加了 cookie 的 HelloVerifyReuqest 消息发送到客户端。客户端从获得的消息中取出 cookie，并添加到 ClientHello 消息中，然后重新发送添加了 cookie 的 ClientHello

消息给服务器。服务器验证发送来的 cookie，如果验证的 cookie 是有效的，则继续握手过程。

4.2.2.2.握手过程

握手处理过程如图 4.6 所示。握手处理过程分析如下：

(1) ClientHello

客户端发送第一个 ClientHello 消息里面没有 Cookie 值，这个消息当前作用是判断服务器是否有响应。由于 DTLS 是基于 UDP 不可靠连接，所以开始必须询问对方是否在线。

(2) HelloVerifyRequest

当服务器端接收到一个没有 Cookie 值的消息时，会给当前的客户端返回一个 Cookie 值。

(3) ClientHello

当客户端收到服务器发送来的 HelloVerifyRequest 消息时，会提取 Cookie 值，然后在 ClientHello 消息中加入 Cookie 并发送。这个消息作用是将客户端所有的加密套件和压缩算法以及产生的随机数发送给服务器，让服务器选择合适的加密套件和压缩算法。

(4) ServerHello

当服务器端收到一个含有 Cookie 的消息时，服务器会选择合适的加密套件和压缩算法，创建一个会话 ID，并产生一个随机数返回给客户端。

(5) ServerCertificate

这个消息紧跟着上面消息发送。这个消息里面包含一个证书链，作用是给客户端表明身份。

(6) ServerKeyExchange

这个消息紧跟着上面消息发送。这个消息里面包含了一个临时产生公钥(密钥交换算法在 ServerHello 中已经协商完毕)，这个公钥的作用是加密对称密钥(对称密钥并不是直接生成的，是客户端产生一个预主密码 premaster，然后用密钥交换算法交给服务器端，两端根据这个预主密码计算出主密码，再用主密码生成对称密钥）。

(7) CertificateRequest

这个消息紧跟着上面消息发送，此消息包含两部分内容：一个是 server 端支持的证书类型(RSA, DSA 等)，另一部分是 server 端所信任的所有证书发行机构列表，客户端会用这些信息来筛选证书。

(8) ServerHelloDone

这个消息没有什么内容，表示服务器已经发送完毕，等着客户端回应了。

(9) ClientCertificates

当客户端收齐上面 5 个消息时，才会发送这个消息。这时候要用到 server 端之前发的 CertificateRequest 消息中的支持的证书类型列表和信任的根 CA 列表，满足这个两个条件的第一个证书链就会被选中作为客户端证书，然后将选择好的证书链发送。

(10) ClientKeyExchange

这个消息紧跟着上面消息发送，客户端会产生一个预主密码，通过传来的公钥进行加密，然后发送给服务器。

(11) CertificateVerify

这个消息紧跟着上面消息发送，发这个消息的目的是让服务器验证发消息的客户端和客户端证书的真实所有者。这个消息中要包含一个签名，签名里头内容就是从 ClientHello 开始到目前为止所有握手消息(不包括本消息)的摘要，把摘要信息用客户端私钥进行签名。到时候服务器端会用收到的证书中的公钥来验证签名。

(12) ChangeCipherSpec

发送这个消息，然后把 Session 的密钥设置成计算得到的对称密钥，从此消息之后再发送消息就会用这个写密钥来加密消息。

(13) ClientFinished

Client 端的 Finished 消息一般都是紧随 ChangeCipherSpec 消息发送出去，标志着本方的协商成功结束。

(14) ChangeCipherSpec

Server 收到 Client 的证书链后验证证书，并验证 CertificateVerify 中的签名，验证通过了也就确认了 client 的身份。收到 ClientKeyExchange 消息从中拿出 Pre-master 计算出 master，生成对称密钥。收到 Client 的

ChangeCipherSpec 后将会话的读密钥设置为刚产生的对称密钥。

(15) ServerFinished

Server 端的 Finished 消息紧随 ChangeCipherSpec 消息发送出去，标志着协商成功结束。

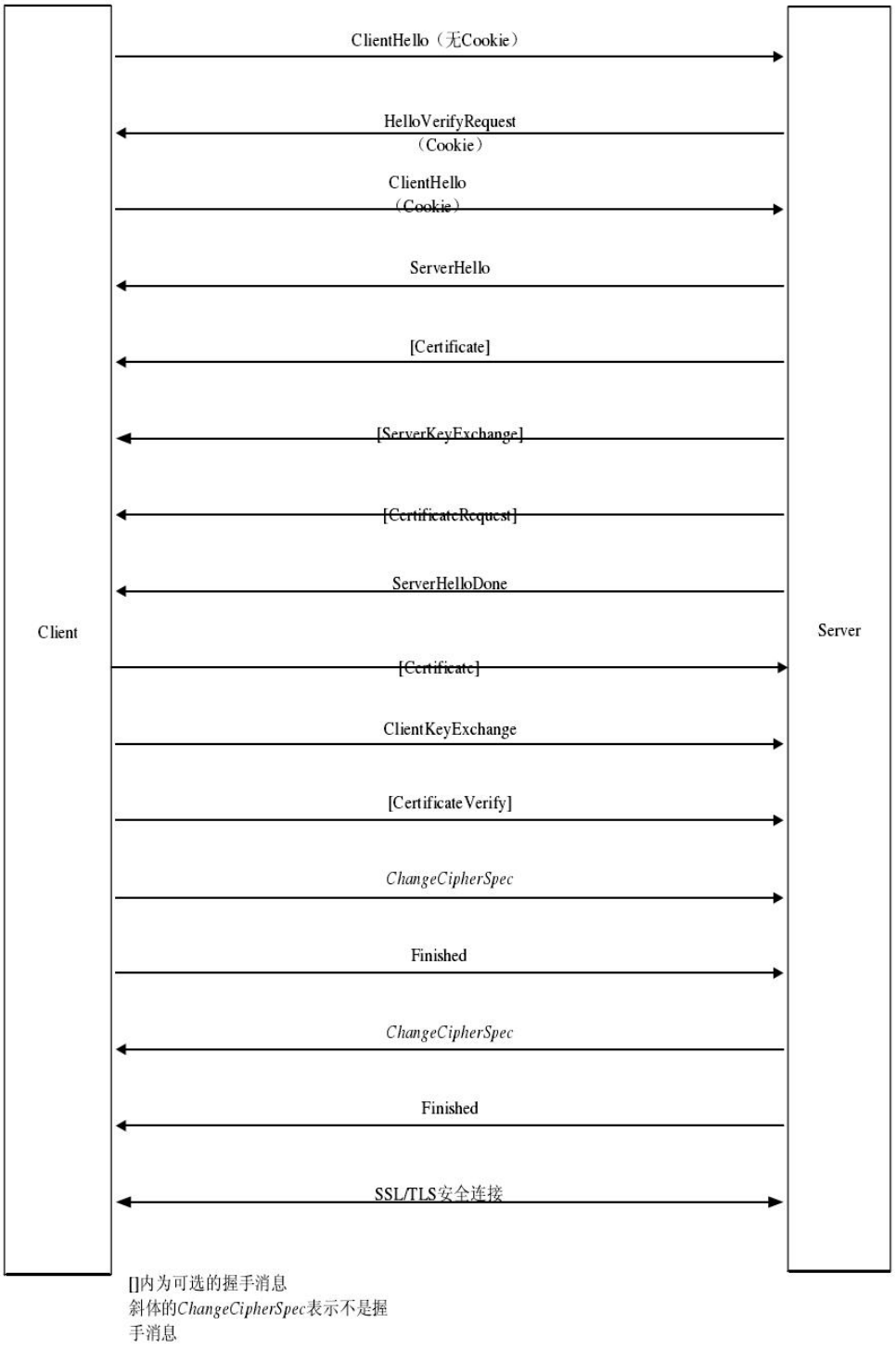

图 4.6 DTLS 握手过程

4.2.3. 改变密码规格协议

改变密码规格协议(change cipher spec protocol)是在分层模型中最简单的协议，该协议只有一个消息(是一个字节的数值)，它使数据传输在记录层按照密码规范中所指定的方式进行压缩和加密，而不按照是改变后的加密规格。它的存在是为了使密码策略改变可以及时通知到对方。在客户端和服务器端为了通知接收方发送的记录使用刚刚协商好的加密规格来保护将要发出的信息都会给对方发出改变密码规格消息。客户端发送改变加密规格消息是在发送完握客户端密钥交换消息和证书检验消息(如果需要)之后;服务器端则在成功接收并处理从客户端发送的改变密码规格消息后发送服务器端改变密码规格消息。当产生一个意外的改变加密规格消息时将导致一个 unExpectedMessgae 警告。当时用恢复会话功能时，改变加密规格消息将在问候消息之后便发送。

4.2.4. 警告协议

警告 alert 类型是 DTLS 记录层所支持的数据传输内容类型之一，同样 DTLS 记录层支持的协议之一是警告协议。警告协议把警报消息以及它们的严重程度发送给参加 DTLS 协议的主体。DTLS 握手协议中处理错误警告消息的方式：当任何一方检测到存在一个错误时，检测到的一方就向另一方发送一个错误警告消息。如果警告消息存在一个致命的行为，则通信的双方应立即关闭连接。任何与该失败的连接相关联的会话标识符、密钥和秘密都需要被双方忘记。然而对于所有的非致命错误，通信双方可以在会话中缓存信息以备以后恢复该连接。警告消息与其它消息一样，都需要经过记录层处理，DTLS 警告消息结构与 TLS 相同，可选择的类型以及对应的数值如下：

WARNING=1;

FATAL=2;

CLOSE_NOTIFY=0;

UNEXPECTED_MESSAGE=10;

BAD_RECORD_MAC=20;

DECRYPTION_FAILED=21;

```
RECORD_OVERFLOW=22;
DECOMPRESSION_FAILURE=30;
HANDSHAKE_FAILURE=40;
NO_CERTIFICATE=41;
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_CANCELLED=90;
NO_RENEGOTIATION=100;
```

4.3. DTLS 协议安全性

DTLS 协议能够保证基于数据报协议的连接安全，这主要体现在三个方面：第一，通过对通信过程中传输的数据进行加密来保证数据的机密性；第二，通过计算数据的 MAC 值以及验证过程来保证数据的完整性；第三，通过身份认证技术来验证通信双方的身份以保证两个正确的通信实体进行通信，而且在此基础上服务器能够按照客户的身份为其分配相应的资源，方便用户的管理。除此之外，DTLS 的告警协议能够在检测到连接错误时通知对方，也能够在一定程度上提供连接的安全性。由前面对 DTLS 协议的分析可知，因为需要支持不可靠的传输，

在其握手协议和记录协议本身也增加了一些安全机制来保证握手阶段的可靠性以及防止重放攻击，拒绝服务攻击等。

根据 DTLS 需要提供的通信安全，在 DTLS 握手的过程中涉及的加密算法主要有三种：密钥交换算法、数据加密算法和散列算法。其中密钥交换算法采用非对称密钥加密算法如 RSA 或 DH，用于通信双方密钥协商过程；数据加密算法采用对称密钥加密算法如 DES、RC4 等，使用密钥交换算法协商的密钥对数据进行加密；散列算法用于数据的完整性检查。这些加密算法相互组合构成若干密码套件，每一个密码套件对应一个固定的整数值，且所有密码套件都采用统一的格式。

4.3.1. 身份认证

在 DTLS 握手过程中，Hello 阶段结束后会需要通信双方进行身份验证，这时可以选择只验证服务器，也可以对客户端和服务器都进行验证。DTLS 采用与 TLS 相同的证书机制实现身份认证，这种身份认证方式基于 PKI（公钥基础设施，Public Key Infrastructure）架构，通过验证主体的数字证书实现对主体身份的认证。这种验证不仅可以确认主体的身份，而且可以确认证书中包含的公开密钥与主体的绑定。证书由可信的发证权威机构签发。被验证方只需把它的证书传给验证方即可，验证方实施对证书的各种检查。

CA 模式下“信任性”的基础是：证书持有者是经过 CA 审查并信任的实体，原本相互之间不存在信任关系的两个实体，只要他们被同一 CA 信任，则这两个实体间就可以相互信任。这种信任模式可以扩展到层次 CA 认证模式和 CA 交叉认证模式。

4.3.2 密钥管理

DTLS 协议中客户端与服务端之间通信所使用的密钥信息是通过握手过程产生和交换的，通信双方首先协商生成一个预主密钥，然后通过这个预主密钥来计算产生一个主密钥，最后是通过主密钥计算出数据加密密钥和 MAC 计算密钥的。产生主密钥时使用到了 Hello 信息交换阶段双方生成的随机数，而且双方交换的是预主密钥而不是主密钥，这样能够最大限度的保证主密钥的安全，也就能够

保证最终产生的通信密钥的安全。

由主密钥产生的密钥包括服务端加密数据的密钥，客户端加密数据的密钥，服务端计算 MAC 的密钥以及客户端计算 MAC 的密钥，所有使用的密钥都是客户端和服务端根据预主密钥和相同的密钥导出算法一步步通过计算产生的，整个计算过程并不需要交换任何信息，因此攻击者无法得到密钥导出的过程。此外，在计算会话密钥时用到了双方生成的随机数，也就是说密钥的产生依赖了双方的信息，并不是通信的某一方独立产生的，这也给攻击者获取密钥信息带来了更大的难度，能够保证最终产生的密钥是安全可信的。

密钥的生成过程随密钥交换算法的不同而不同，密钥交换算法是在握手过程中选择确定的，DTLS 协议通常提供 RSA 和 Diffie-Hellman 两种密钥交换算法（虽然协议规范中还包括 FORTEZZA 算法，但该算法一般用于军事领域，在商业实现中没有提供）。在 DTLS 协议中，密钥交换算法交换的并不是对称加密主密钥，而是预共享主密钥 pre-master secret。在双方商定预共享主密钥 pre-master secret 后，再由 pre-master secret 生成主密钥 master secret，然后由 master secret 生成最终的用于连接的对称加密密钥、MAC 密钥。

下面以 RSA 为例说明密钥的生成过程。密钥的生成从握手过程可以看到大致可以分为以下几个阶段：

（1）在握手过程中，当客户方对服务器方完成身份认证以后, 客户方就可以从服务器方的证书中得到服务器的公开密钥；

（2）客户方产生一个随机数，作为准密钥（pre-master secret）；

（3）客户方用对方的公开密钥加密准密钥；

（4）客户方通过网络把该密文传给服务器方；

（5）服务器用自己的私钥解密得到准密钥

（6）至此，双方都具有一个完全相同的准密钥，在此基础上，根据前面的握手过程协商确定的应用数据加密算法和散列算法，通信双方将各自使用相同的算法导出主密钥 master secret。为了安全起见，一旦生成了 master secret，应立即将 pre-master secret 删除。

（7）双方得到 master secret 后，再由 master secret 经一定的计算生成 key block，然后对 key block 进行切分，从而得到加密所需的各个密钥和散列

所需的散列密钥。如果切分完成后 key block 还有剩余字节，则直接将其抛弃。

4.3.3 MAC 验证

DTLS 协议记录层在封装数据包时，会首先对数据计算 MAC 值，将其附加在实际数据片段之后，再对包括数据和其 MAC 值在内的整个信息进行加密，当对方收到经过加密的数据包后，对其解密并且验证 MAC 值的正确性，以此来保证传输数据的机密性和完整性要求。值得一提的是，DTLS 记录层添加了一个消息的序列号，在计算数据的 MAC 值时使用到了这个序列号。这样不仅能够在接收方处理消息的丢失和乱序问题，还能有效地抵御重放攻击。当接收方收到一个消息后，首先根据序列号判断是否已经收到过此消息，然后通过验证数据的 MAC 值来确定数据有没有被篡改。如果在传输过程中序列号被篡改，则同样不能通过 MAC 验证，这也就在保证数据完整性的同时进一步防止了重放攻击，增加了数据的安全性。

4.3.4. 抗 Dos 机制

DTLS 基于 UDP。UDP 中最常见便是 UDP 洪水攻击，该攻击利用 TCP / IP 协议规范，向启用 Echo（回响）服务的 UDP 服务器发送大量无用数据。这些数据包的目的端口在目标主机并没有应用程序使用，所以目标主机就会朝攻击者给出的源 IP 地址回复 ICMP 包。但攻击者的源 IP 是伪造的，所以目标主机发送这些 ICMP 包不仅没有了下文，而且会消耗自己的流量，最后瘫痪。

UDP 可以一对多通信，因此 DTLS 多少会受到低层协议的影响，这一点比 TLS 更严重。在 DTLS 和 TLS 握手中，只要服务端一收到客户端发来的 ClientHello，就必须回复。TLS 服务端直接就回复了 ServerHello 消息，开始向客户端发送证书了。但如果 DTLS 也这样设计，那么攻击者就可以通过大量发送 ClientHello 消息来迫使服务不断回复证书等数据，一定会导致 DTLS 服务器瘫痪。所以在 DTLS 握手过程中，服务器发送一个 HelloVerifyRequest 消息，该消息带有随机的 Cookie，客户端需要复制该 Cookie 并在第二个 ClientHello 消息中回传给服务端，后者通过零知识证明（该 Cookie 并不在服

务端保存)的方式验证回传的 Cookie 是否是自己在 HelloVerifyRequest 消息放置的 Cookie。如果是，双方就成功进入下一回合，如果不是，则双方握手失败。这样的方式迫使攻击者必须发送额外数据，类似于网站注册时采取的验证码，服务端也不用立即发送证书，可以有效减少 DoS 攻击的影响。此外，该方式使得攻击者不能使用伪造的 IP，否则它将无法收到 HelloVerifyRequest 消息，结果便是握手失败，通信结束。

4.4. DTLS 协议的应用

此节主要调研了 Alexa 艺术排名前 20 的视频网站以及国内各类日常应用所使用的传输协议，通过 wireshark 抓包分析它们传输数据时所使用的协议。并具体分析包的内容。

Alexa 艺术排名前 20 的视频网站传输视频流时所使用的协议如表 4.1 所示。从表中很明显能看到，视频传输协议都是采用 TLS。

表 4.1 Alexa 艺术排名前 20 的视频网站

排名	URL	协议
1	https://www.youtube.com	TLS1.2
2	https://www.Netflix.com	TLS1.2
3	https://www.Imdb.com	TLS1.2 TCP
4	https://www.Cnn.com	TLS1.2
5	https://www.Spotify.com	TLS1.2
6	https://www.Bbc.co.uk	TLS1.3
7	https://www.Espn.com	TLS1.3
8	https://www.Cnbc.com	TLS1.2
9	https://www.Ndtv.com	TLS1.2 TLS1.3
10	https://www.Npr.org	TLS1.3
11	https://www.Ign.com	TLS1.2
12	https://www.Ultimate-guitar.com	TLS1.2
13	https://www.Rottentomatoes.com	TLS1.2 TLS1.3

14	https://www.Abcnews.go.com	TLS1.2 TCP
15	https://www.Cbc.ca	TLS1.2 TCP
16	https://www.Cbsnews.com	TLS1.2
17	https://www.wimp.com	TLS1.3
18	https://www.avid.com	TLS1.2
19	https://www.borisfx.com	TLS1.3
20	https://www.pinnaclesys.com	TLS1.3

国内各类网站传输数据对应的协议如表 4.2 所示，从表中数据可知 DTLS 协议主要运用在直播网站和国内视频网站的视频数据传输过程。

表 4.2 国内各类应用传输音视频的协议

编号	应用	操作	传输协议
1	微信	视频通话	UDP
2	QQ	视频通话	UDP
3	爱奇艺	观看视频	DTLS TLS
4	腾讯视频	观看视频	DTLS TLS
5	哔哩哔哩	观看视频、直播	TLS1.2 TLS1.3
6	优酷	观看视频	DTLS TLS1.2
7	芒果 TV	观看视频	TCP TLS
8	斗鱼	观看直播	DTLS
9	虎牙	观看直播	DTLS
10	淘宝	观看直播	TCP TLS
11	企鹅电竞	观看直播	DTLS
12	龙珠直播	观看直播	TCP

通过 wireshark 抓包分析 DTLS 协议结果如下：

（1）Client Hello 消息

Client Hello 消息如图 4.7 中所示，从中可得到握手类型（Handshake Type）为 Client Hello、握手包的序号值（Message Sequence）为 0、字段偏移值（Fragment Offset）为 0、字段长度（Fragment Length）为 130、版本类型

（Version）为 DTLS 1.2 以及随机数（Random）的值。紧随 Random 之后的字节表示 Session Id 的长度。此时 Session Id 长度为 0，表示客户端要生成一套新的加密参数；如果 session_id 长度不为 0，则 Session Id 表示客户端希望重用的会话。Cookie 长度（Cookie Length）之后的 Cipher Suites Length 表示客户端支持的加密套件的个数，每一个加密套件 2 字节，此时长度为 24，表示支持的加密套件个数为 12。紧接着的 Cipher Suites 为具体的加密套件。加密套件之后是客户端支持的压缩算法的个数（Compression Methods Length），每一个压缩算法占 1 字节，此时表示客户端支持 1 个压缩算法。而随后的 Compression Methods 表示具体支持的压缩算法。

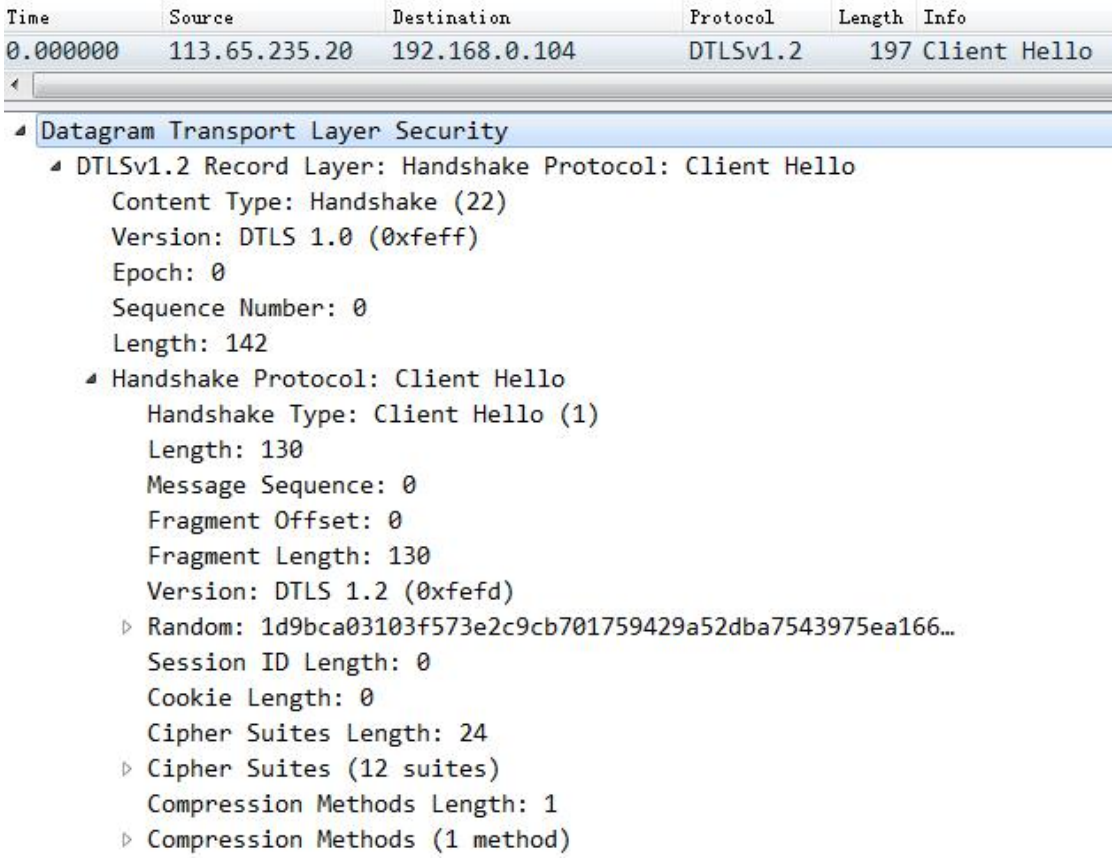

图 4.7 Client Hello 消息

（2）Server Hello 消息

Server Hello 消息如图 4.7 所示，其中 Handshake Type、Message Sequence、Fragment Offset、Fragment Length、Version 字段分析与上节类似，此处不再展开分析。其后的 32 字节是服务器产生的随机数（Random），结构与 ClientHello 消息中的 Random 相同。Random 之后的 1 字节表示服务器的 Session Id 的长度。如果 Session Id 长度不为 0，则表示服务器允许客户端

在以后重用该会话。此时值为 0，表示服务器不希望客户端重用该会话。Ciper Suite 占用 2 字节，表示服务器所选择的加密套件。ServerHello 消息的最后 1 字节 Compression Methods 表示服务器选择的压缩算法。

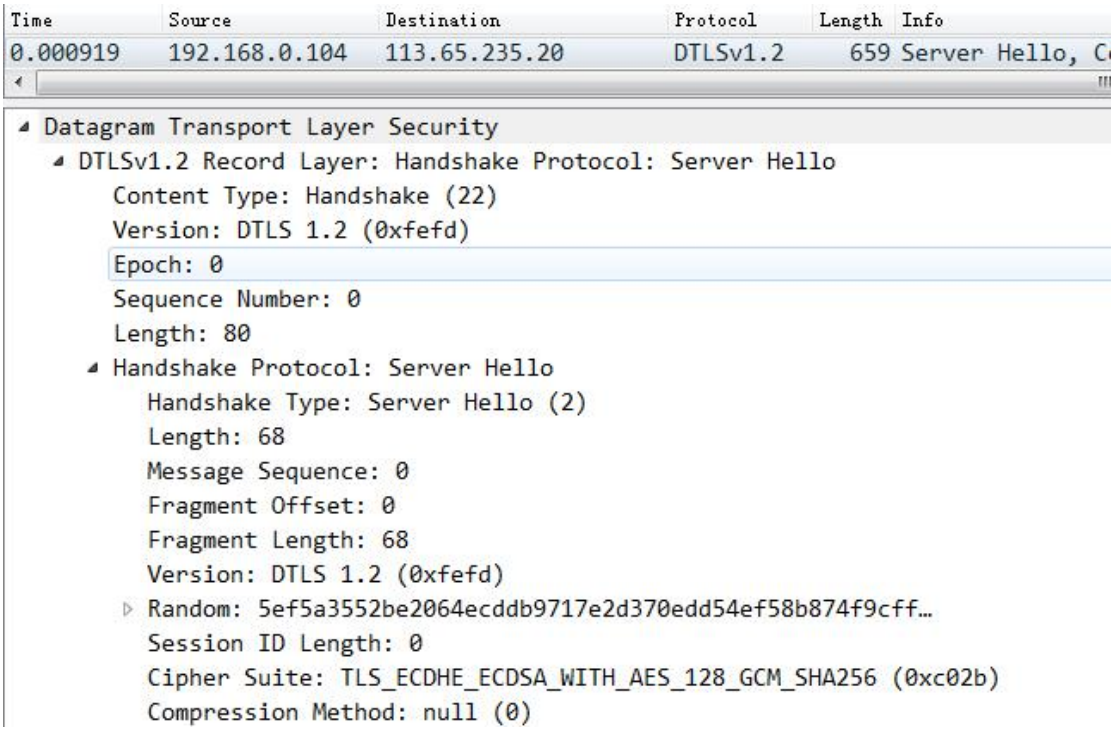

图 4.7 Server Hello 消息

（3）Certificate 消息

Certificate 消息如图 4.8 所示，其中 Certificate 部分包含有服务器证书的公钥；客户机接收到该报文段后，按照协议规定，从报文段的对应位置中读取出服务器证书的公钥存入相关变量中。

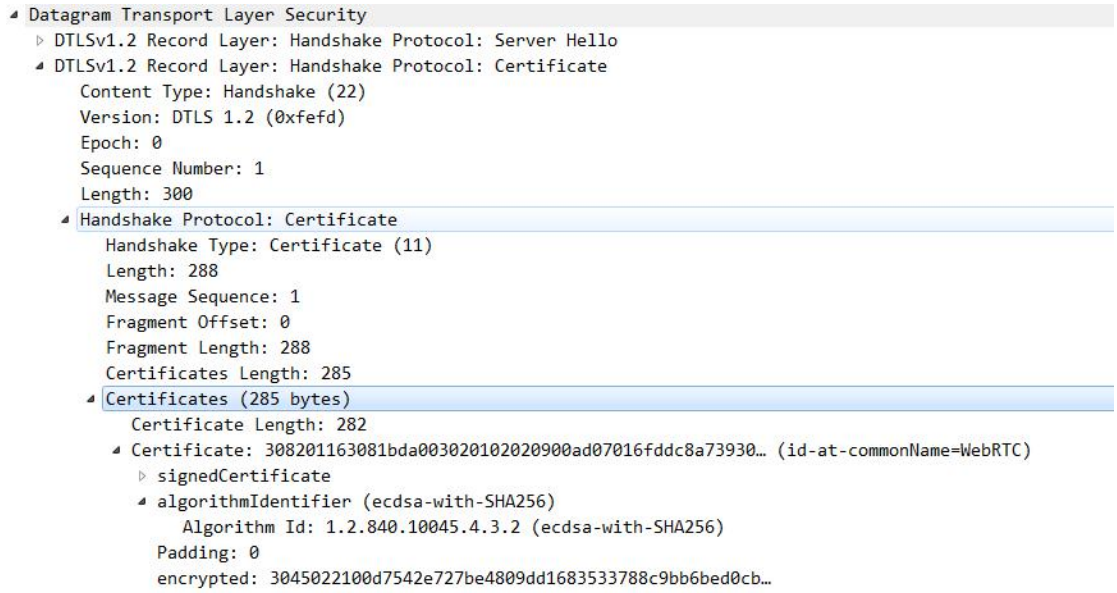

图 4.8 certificate 消息

（4）Server Key Exchange 消息

Server Key Exchange 消息如图 4.9 所示，从 ED Diffie-Hellman Server Params 字段可知协议所选的是 ECC（椭圆曲线加密），使用 ECDH（椭圆曲线秘钥交换协议）和 ECDSA（椭圆曲线数字签名算法）。

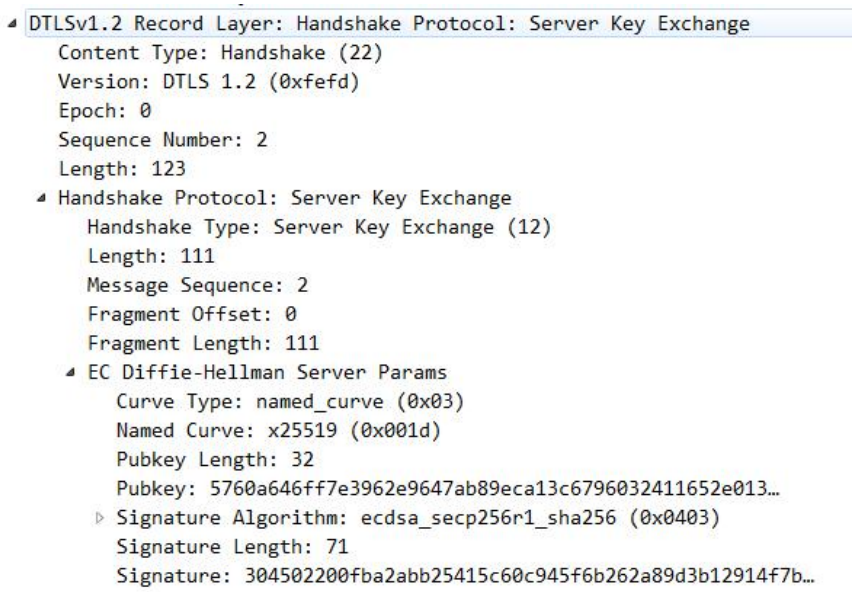

图 4.9 Server Key Exchange 消息

（5）Certificate Request 消息

Certificate Request 消息如图 4.10 所示，其中 Certificate Types 字段是由单字节表示的签名算法组成，指定了服务器可接受的认证类型为 RSA 认证与 ECDSA 认证。Signature Hash Algorithms 为支持的认证散列算法。

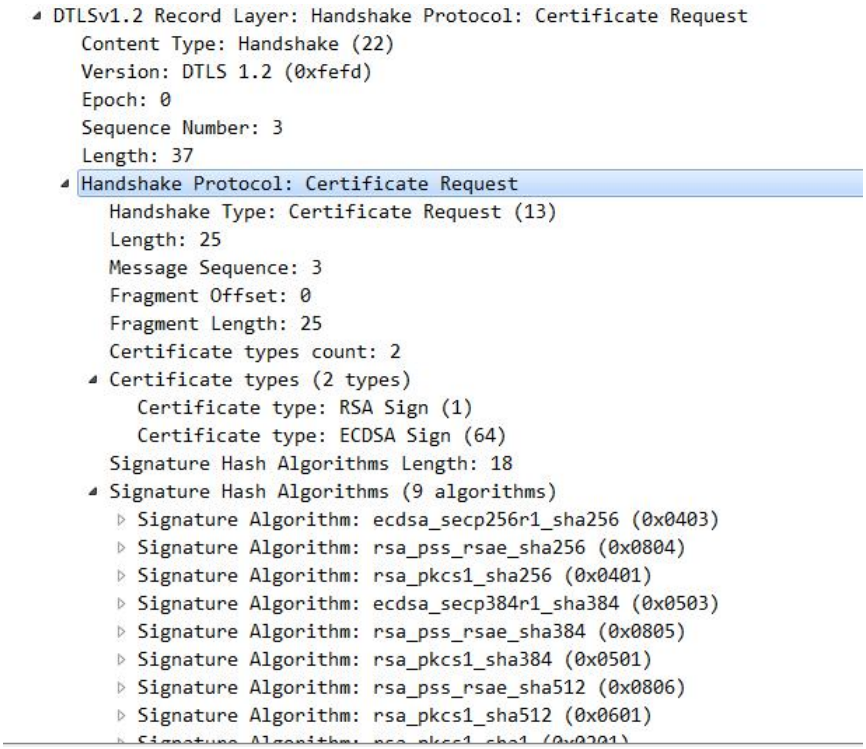

图 4.10 certificate request 消息

（6）Server Hello Done 消息

Server Hello Done 消息如图 4.11 所示，Server Hello Done 消息是一条空消息，表示服务器已经发送完了本阶段所有的消息。该消息是必需的，因为在 Certificate 消息后服务器还可以发送一些可选消息。

且从图中可看出，上述的（2）-（6）消息都是封装在一个包中由服务器发给客户端。

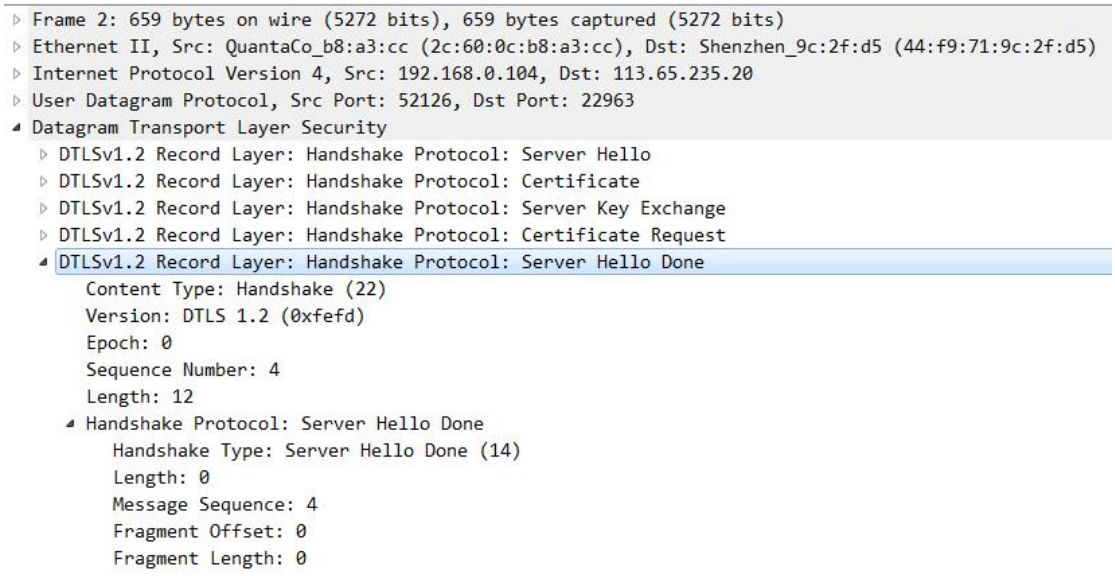

图 4.11 Server Hello Done 消息

最后，整个过程的数据包如图 4.11 所示，上述主要分析了第一二个数据包，其中第三个客户端到服务器的 Certificate，Client Key Exchange，Certificate Verify，Change Cipher Spec 与第二个数据包的分析类似，此处不展开分析。

Time	Source	Destination	Protocol	Length	Info
0.000000	113.65.235.20	192.168.0.104	DTLSv1.2	197	Client Hello
0.000919	192.168.0.104	113.65.235.20	DTLSv1.2	659	Server Hello, Certificate, Server Key Exchange, Certificate Request, Server Hello Done
0.009628	113.65.235.20	192.168.0.104	DTLSv1.2	587	Certificate, Client Key Exchange, Certificate Verify, Change Cipher Spec, Encrypted Handshake Message
0.011122	192.168.0.104	113.65.235.20	DTLSv1.2	612	New Session Ticket, Change Cipher Spec, Encrypted Handshake Message
0.016268	192.168.0.104	113.65.235.20	DTLSv1.2	179	Application Data

图 4.11 总体数据包

5. 前瞻

虽然目前并没有专门针对 DTLS 协议本身的有效攻击，但是 DTLS 在实际实现中的具体特定方法却让攻击者有很多有机可乘之处，尤其是以下问题，在使用 DTLS 时需要特别加以注意。

5.1. 密钥的安全和随机数强度

在 DTLS 握手阶段，除了协商密码算法和身份认证以外，最重要的就是密钥的产生和交换，密钥的安全是通信安全的基础和关键。

DTLS 的密钥交换主要使用两个协议：RSA 协议和 DH 协议。对于 RSA 算法，由客户端随机生成一个预主密钥，并使用服务器证书中的公钥或者服务器发送过来的 ServerKeyExchange 消息中的临时 RSA 密钥进行加密，然后发送给服务端；对于 DH 算法，客户端向服务端发送的是其 DH 公钥，然后双方根据共享的参数和对方的公钥计算出相同的共享密钥，以这个共享密钥作为预主密钥。由此可见，两种协议虽然提供了不同的密钥协商方法，但是最终的目的都是要产生预主密钥（premaster Secret）。预主密钥的作用是计算出主密钥，从而得到加密通信数据需要使用的所有密钥。

由次可见，要保证密钥的安全性，首先需要保证预主密钥的安全。因为在 DTLS 的密钥交换过程中实际交换的是预主密钥，一旦预主密钥泄露，攻击者可能会很快根据它推算出会话密钥，那么连接将不再有安全性可言，因此保护预主密钥是保证密钥安全的关键，也是保证通信安全的关键。

其次，要保证使用强度较高的随机数。因为无论使用哪种密钥交换协议，归根到底都是以生成随机数为基础的（若使用 DH 算法，通信双方使用的私钥都是随机产生的，这是计算预主密钥的基础；而对于 RSA 算法，预主密钥实际上就是客户端产生的随机数），而且会话密钥的计算也需要使用到通信双方产生的随机数。由于在交换 Hello 消息时还没有协商好加密算法和密钥，双方是以明文的形式进行传输的，也就是说此时生成的随机数是没有经过加密处理的，因此若使用了弱随机数则攻击者很可能推算出预主密钥，进而得到主密钥，一旦主密钥丧失了安全性，造成的后果将不堪设想。

5.2. 选择合适的密码算法

DTLS 支持很多种的密码算法，各种算法的强度也参差不齐，为了有效确保数据的安全性，应该根据数据本身的重要程度来选择合适的加密算法。这并不是说对所有的数据都要选择强度最高的算法，因为数据加密过程本身也很耗费资源。只要攻击者破译密文所花费的代价大于数据本身的价值或者花费的时间超出了数据的有效期就能够有效的保证数据的安全性。因此，客户端可以根据数据的价值对提供给服务端的相应密码套件进行一定的限制，使算法的强度不低于数据价值本身的要求。

DTLS 在协商密码算法时，服务端需要从客户端提供的算法集合中选择出自己也支持的密码套件，否则就不能进行连接。所以为了最大限度的满足这个条件，客户端应该向服务端提供满足其数据安全性最低要求的所有算法，这样不仅能够保证提供的都是有效的密码算法，也能最大限度的保证提供的加密套件服务端也同样支持。

5.3. 使用强身份认证方式

虽然 DTLS 协议可以支持不同的身份认证方式，包括匿名的连接（即对客户端和服务器均不进行有效身份验证），这种方式虽然免去了验证环节而使得通信非常方便，但是却对安全造成了很大的威胁，很容易遭受中间人攻击，攻击者甚至很容易伪装成服务器和客户端进行通信，从而窃取一些机密的信息。因此为了

确保连接双方的真实可靠性，还是应该使用对通信双方的身份均进行验证的强身份认证方式。

为了保证用户持有的证书是可信的，首先要确保签发证书的 CA 是可信的。如果使用了不可信的 CA，即便服务器在证书中声明了自己的身份，也不能保证它提供的信息是可靠的，若此时继续进行握手连接，很可能遭受中间人或者其他类型的攻击。此外，还需要确保证书的安全性。

6. 参考文献

[1]魏阳. DTLS 协议的缺陷分析和解决方案[D].重庆邮电大学,2018.

[2]冀云刚. 传输层安全协议研究及应用[D].西安电子科技大学,2011.

[3]吴祥业. 基于 Java 的 DTLS 协议实现与安全性分析[D].西安电子科技大学,2012.

[4]邓荭. 基于 DTLS 协议 VPN 的研究与实现[D].电子科技大学,2011.

[5]刘洪强. 基于 SSL 协议的 VPN 技术研究与实现[D].山东大学,2008.

[6]曾强. 网络安全协议 SSL 原理及应用[D].天津大学,2005.

[7]张兴隆,程庆丰,马建峰.TLS 1.3 协议研究进展[J].武汉大学学报(理学版),2018,64(06):471-484.

[8]https://tools.ietf.org/html/rfc6347

[9]https://blog.csdn.net/weixin_33924770/article/details/94279627?ops_request_misc=&request_id=&biz_id=102&utm_term=DTLS&utm_medium=distribute.pc_search_result.none-task-blog-2~all~sobaiduweb~default-2-94279627

[10]https://blog.csdn.net/weixin_33749131/article/details/91538090?ops_request_misc=%257B%2522request%255Fid%2522%253A%25221592120448197252476174 03%2522%252C%2522scm%2522%253A%252220140713.130102334..%2522%257D&request_id=159212044819725247617403&biz_id=0&utm_medium=distribute.pc_search_result.none-task-blog-2~all~sobaiduend~default-3-91538090.ecpm_v1_rank_ctr_v4&utm_term=DTLS