第16章 无线传输层安全规范

16.1 范围

无线应用协议（Wireless Application Protocol，WAP）是WAP论坛经过不断努力得到的成果，它提供了一个业界技术规范，以便开发出适用于各种无线通信网络的应用和业务。 WAP论坛的工作范围就是为各种业务和应用制定一系列的技术规范。无线市场正在快速增长，新的用户不断增多，新的业务不断涌现。为了给运营商和生产者提供一个面对先进业务、多种类业务和快速灵活的业务生成的商机， WAP制定了一系列传输层、会话层和应用层协议。有关WAP体系结构更多的信息，请参阅“ 无线应用协议体系结构规范 ” (Wireless Application Protocol Architecture Specification) [WAP]。

WAP中的安全层协议被称为无线传输层安全协议 (Wireless Transport Layer Security, WTLS)。WTLS层运行于传输协议层之上，它是模块化的，其是否使用取决于给定应用所要求的安全层次。 WTLS为WAP的上层提供了一个安全的传送的服务接口，这一接口在它下面保留了传送服务接口。另外， WTLS提供了管理安全连接的一个接口（比如产生和终止安全连接）。

WTLS的主要目的是在两个进行通信的应用间提供保密性、数据整合以及鉴权。 WTLS提供与TLS1.0类似的功能并且包括了新的特点，诸如：数据报支持，优化的握手，动态密匙刷新。WTLS对窄带的，有相对较长反应时间的承载网络是最优的。

16.2 WTLS结构概述

参考模型

WAP的分层模型如图16-1所示，WAP协议和它的各种功能是按照[ISO7498]的样式进行分层的，其中[ISO7498]是国际标准化组织的OSI参考模型。分层管理实体负责处理协议的初始化、配置和错误状态（例如，由于移动台漫游出服务区而导致的连接失败）等协议自身无法控制的操作。

WTLS是工作在面向连接的和/或数据报传送协议上。安全层被认为是在传送层上的可选层，它保留了传送服务的接口。应用管理或会话管理实体为安全连接管理（如建立和终止）的需求提供了附加的支持。

16.3 用于层到层通信的WTLS元素

16.3.1 使用的注解

1. 服务原语和参数的定义

层间通信通过服务原语的方式完成的。服务原语这个说法简单地表示了安全层和相邻层

之间信息的逻辑交换和控制，它由命令及与另一层所需服务有关的响应共同组成。通常一个原语的语法是：

```
X-Service.type (Parameters)
```

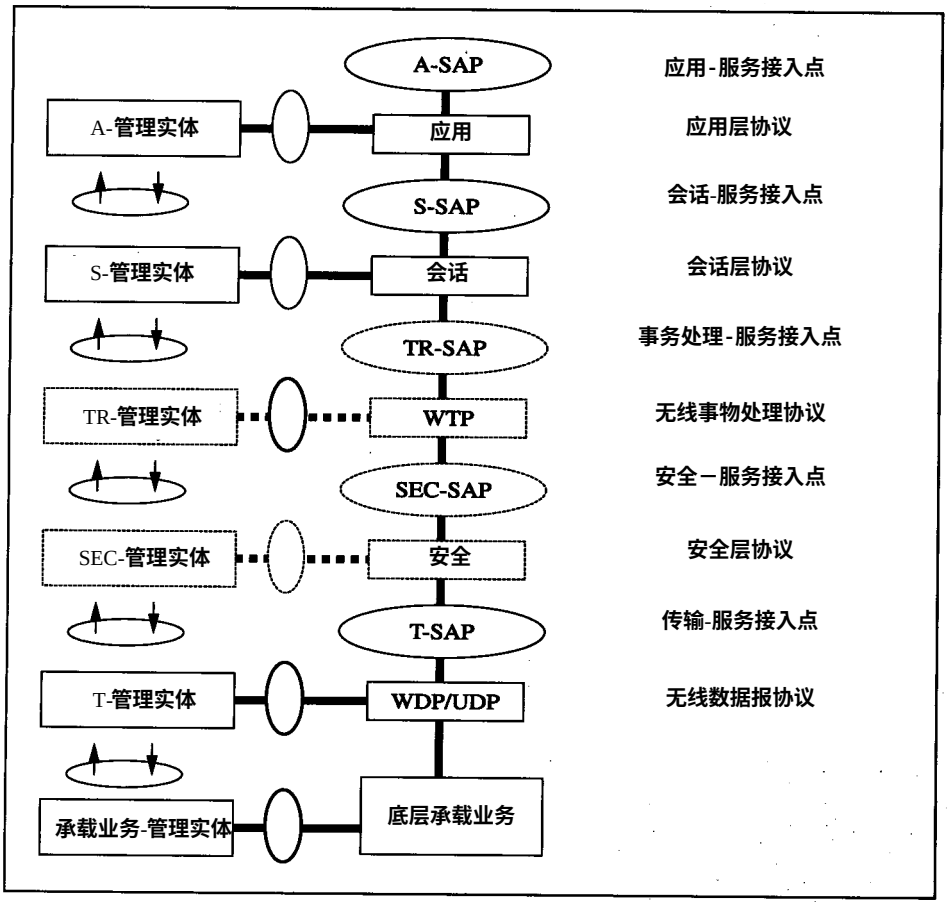

图16-1 无线应用协议参考模型

X-服务.类型 (参数)

X代表提供服务的层。在本规范中，对安全层而言， X是“ SEC ”。

服务原语不同于一个应用程序接口（API），它并不意味着任何特定的API实现方法，它是一个抽象的方法，描述由协议层提供给上一层的服务。如何把这些概念映射到一个实际的 API及其相关的语法是一个实现问题，这个问题不属于本规范的范畴。

2. 时序图

服务原语的行为用时间序列表来表示，具体描述见 [ISO10731]。图16-2给出了一个简单的无验证服务，这个服务由一个请求原语发起，并在对等端产生了一个指示原语。

虚线表示经过提供者一段时间的传播，这种传播由表示原语的两个箭头之间的纵向间隔代表。

3. 原语类型

原语类型定义如表16-1所示。

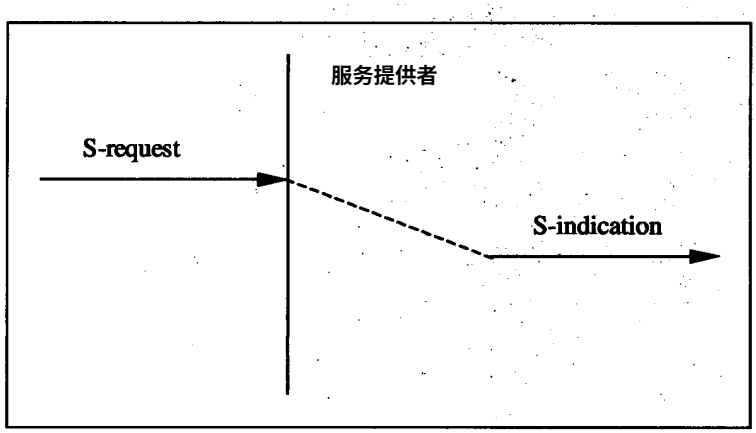

图16-2 未证实的服务

表16-1 原语类型

类 型	缩 写	描 述
请求（request）	req	当一个较高层向其下层请求服务时使用
指示（indication）	ind	提供服务的层使用此原语类型通知其上一层与对等层（例如请求原语的调用）或是服务提供者（例如一个由协议产生的事件）相联的操作
响应（response）	res	某一层使用响应原语类型以确认从下一层收到了指示原语类型
确认（confirm）	cnf	提供请求服务的层使用确定原语类型来报告操作已成功完成

4. 服务参数表

服务原语使用表格来定义，这些表格定义了哪些参数是可能的，以及他们是如何与不同原语一起使用的。例如，一个简单的证实原语可以如表 16-2中所示定义。

表16-2 原语 S-PRIMITIVE

参 数	REQ	IND	RES	CNF
参数1	M	M(=)		
参数2			O	C(=)

如果某个原语类型是不可用的，那一列会被省略。原语类型栏中的符号含义在表 16-3中定义。

表16-3 参数使用符号的说明

M	参数的存在是强制性的，即它必须存在
C	参数的存在与否依条件而定，依赖于其他参数的值
O	参数的存在由用户选择，它可被省略
P	参数的存在与否由服务提供者选择，某一个实现可能不提供它
й	参数是默认的
*	参数的存在由底层协议决定
(=)	参数值等于上一个服务原语的对应参数值

在前面的表中，参数1在S-primitive.request和对应的S-primitive.indication总是存在的。参数2可以在S-primitive.response中规定，在这种情况下，它必须出现，并且在相应的S-primitive.cofirm中有同样的值，否则，不能出现。

16.3.2 WTLS传送服务

1. 服务原语

(1) SEC-Unitdata

SEC-Unitdata是用于在同层体间交换用户数据，它只能在同层体的传送地址间存在一个安全连接的情况下才能被激活（见表16-4）。

表16-4 原语SEC-Unitdata

参 数	REQ	IND
源地址①	M	M(=)
源端口②	M	M(=)
目标地址③	M	O(=)
目标端口④	M	O(=)
用户数据⑤	M	M(=)

① 标识发起者。
② 标识发送消息的端口。
③ 标识接用户数据的对等端。
④ 标识接收消息的端口。
⑤ 被传送的数据。

16.3.3 WTLS连接管理

1. 概述

WTLS连接管理允许一个客户端连接到一个服务器上，并就要使用的协议选项达成一致。

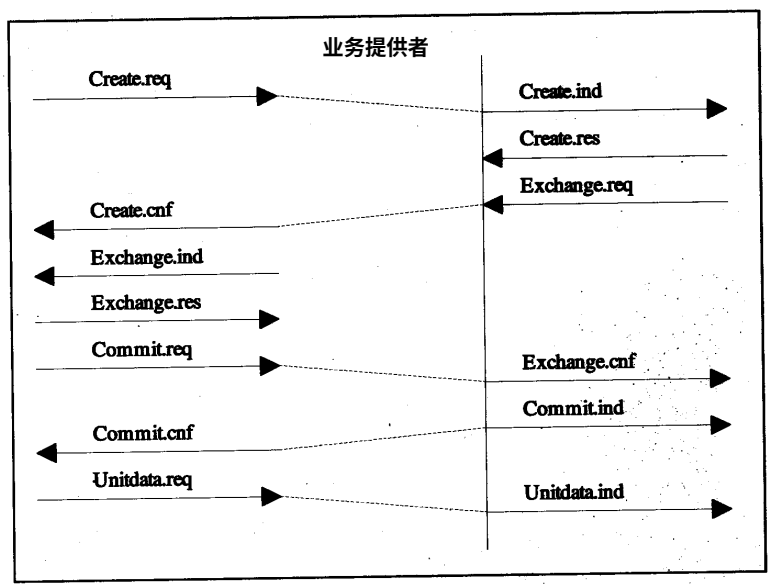

图16-3 完全握手

一个安全连接的建立过程包括几个步骤，客户端或服务器端都可以根据需要中断这一协商过程（例如：如果由对等端提供的参数不可接受）。协商的内容包括：安全参数（如加密算法、密匙长度）、密匙交换及授权。服务器或客户端在任何时间都可以终止连接。

图16-3给出了建立一个安全会话（完全握手）的原语序列。

图16-4给出了以优化或简单的方式建立一个安全会话的原语序列。

2. 服务原语

(1) SEC-Create

这个原语是用于发起一个安全连接的建立过程（见表 16-5）。

表16-5 原语SEC-Create

参 数	REQ	IND	RES	CNF
源地址①	M	M(=)		
源端口②	M	M(=)	-	-
目标地址③	M	O(=)	-	-
目标端口④	M	O(=)		
客户端标识⑤	O	C(=)	-	-
建议的密匙交换簇⑥	M	M(=)	-	-
建议的加密簇⑦	M	M(=)	-	-
建议的压缩方法⑧	M	M(=)	-	-
序列号模式⑨	O	C(=)	M	M(=)
密匙刷新⑩	O	C(=)	M	M(=)
会话Id⑪	O	C(=)	M	M(=)
选择的密匙交换簇⑫	-	-	M	M(=)
选择的加密簇⑬	-	-	M	M(=)
选择的压缩方法⑭	-	-	M	M(=)
服务器验证字⑮	-	-	O	C(=)

① 标识发起者。

② 标识发送消息的端口。

③ 标识接用户数据的对等端。

④ 标识接收消息的端口。

⑤ 用一个与传输无关的方式标识发起者。服务器可以用这个参数来查找对应的客户端验证字。客户端可以发送多个客户端标识以对应不同的密匙或验证字。

⑥ 包括由客户端提出的密匙交换簇。

⑦ 包括由客户端提出的加密簇。

⑧ 包括由客户端建议的压缩方法。

⑨ 定义了序列号在这次全连接中是如何使用的。

⑩ 定义了在一个安全连接内加密和保护密匙的刷新频率。

⑪ 标识了这个安全会话。这个ID对每一服务器是唯一的。

⑫ 标识了服务器选择的密匙交换簇。

⑬ 标识了服务器选择的加密簇。

⑭ 标识了服务器选择的压缩方法。

⑮ 服务器公共的密匙验证字。

(2) SEC-Exchange

当服务器希望与客户端进行公开密匙授权或密匙交换时，本原语用于一个安全连接的建立过程（见表16-6）。

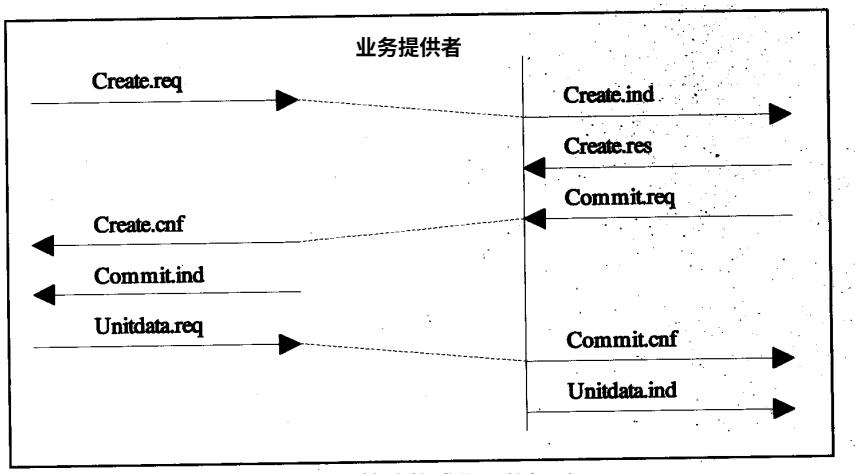

图16-4 简略的或优化的握手

表16-6 原语SEC-Exchange

参 数	REQ	IND	RES	CNF
客户端验证字①	-	-	M	M(=)

① 客户端公共的密匙验证字。

(3) SEC-提交

当握手结束，而对等端的任一方要求切换到新的协商好的连接状态时，该原语被激活(见表16-7)。

表16-7 原语SEC-提交

参 数	REQ	IND	RES	CNF
-	-	-	-	-

(4) SEC-Terminate

该原语用于终止连接（见表16-8)。

表16-8 原语SEC-Terminate

参 数	REQ	IND
告警描述①	M	M(=)
告警级别②	M	M(=)

① 标识了导致中断的原因。
② 指出了是一个会话（致命的）还是仅仅是一个连接（严重的）被终止。

(5) SEC-Exception

该原语用于通知另一端警告的级别（见表16-9)。

表16-9 原语SEC-Exception

参 数	REQ	IND
告警描述①	M	M(=)

① 标志了导致告警的原因。

(6) SEC-Create-Request

服务器使用该原语来要求客户端初始化一个新的握手过程见表 16-10。

表16-10 原语SEC-Create-Request

	REQ	IND
源地址①	O	C(=)
源端口②	O	C(=)
目标地址③	O	C(=)
目标端口④	O	C(=)

① 标识发起者。
② 标识发送消息的端口
③ 标识接收数据的客户端。当在无（无）会话状态中使用原语时需要这个参数。
④ 标识接收数据的端口

3. 使用服务原语的限制

表16-11至表16-13定义了在服务接口上允许使用的原语序列。由于服务是非对称的，客户端和服务器使用的原语分别在不同的表上。

行中是允许使用的原语。为了简单起见，省略了层的前缀。表中的各项在表 16-11中给出了解释。

表16-11 表中符号的说明

记 录	描 述
	指出或确认原语不能出现
N/A	调用这个原语是一个错误，这是本地的具体实现应该采取的行动
STATE_NAME	原语被允许，它将问题从服务接口转移到了被命名的状态

表16-12 可以使用的客户端安全层原语

客户端	会话状态							
SEC-原语	NULL	CREATING	CREATED	EXCHANGE	COMMIT1	COMMIT2	OPENING	OPEN
Create.req	CREATING	N/A	N/A	N/A	N/A	N/A	N/A	CREATING
Commit.req	N/A	N/A	N/A	N/A	COMMIT2	N/A	N/A	N/A
Terminate.req	N/A	NULL	NULL	NULL	NULL	NULL	NULL	NULL
Exception.req	N/A	CREATING	CREATED	EXCHANGE	COMMIT1	COMMIT2	OPENING	OPEN
Unitdata.req	N/A	N/A	N/A	N/A	N/A	N/A	OPENING	OPEN
Exchange.res	N/A	N/A	N/A	COMMIT1	N/A	N/A	N/A	N/A
Exchange.ind			EXCHANGE					
Commit.ind			OPENING					
Terminate.ind		NULL				NULL	NULL	NULL
Exception.ind		CREATING				COMMIT 2	OPEN	OPEN
Create-Request.ind	NULL						OPEN	OPEN
Unitdata.ind							OPEN	OPEN
Create.cnf		CREATED						
Commit.cnf						OPEN		

表16-13 允许的服务器安全层原语

服务器	会话状态						
SEC-原语	NULL	CREATING	CREATED	EXCHANGE	COMMIT	OPENING	OPEN
Exchange.req	N/A	N/A	EXCHANGE	N/A	N/A	N/A	N/A
Commit.req	N/A	N/A	COMMIT	N/A	N/A	N/A	N/A
Create-Request.req	NULL	N/A	N/A	N/A	N/A	N/A	OPEN
Terminate.req	N/A	NULL	NULL	NULL	NULL	NULL	NULL
Exception.req	N/A	CREATING	CREATED	EXCHANGE	COMMIT	OPENING	OPEN
Unitdata.req	N/A	N/A	N/A	N/A	N/A	N/A	OPEN
Create.res	N/A	CREATED	N/A	N/A	N/A	N/A	N/A
Commit.ind						OPEN	
Create.ind	CREATING			CREATING	CREATING		CREATING
Terminate.ind				NULL	NULL		NULL
Exception.ind	NULL			EXCHANGE		COMMIT	OPEN
Unitdata.ind							OPEN
Exchange.cnf				OPENING			
Commit.cnf						OPEN	

16.4 WTLS状态表

下面的状态表定义了WLTS在一个数据报传输服务提供者上进行的动作。

WTLS PUD以斜体标识。

在默认状态下，所有的WTLS PDU都将在当前使用的状态下被处理。

如果收到未被列在“条件”栏中的PDU，接收者可以根据情况的严重程度发出一个告警。

尽管提供的状态表对理解WTLS协议很有帮助，但它们并不是正式和完整的定义，对这些表的描述尽量的简洁，以增加可读，因此某些层次的细节没有反映出来。因此，这个规范的文本描述才是WTLS协议的唯一和完整的定义。

16.4.1 客户端状态表

表16-14～表16-21描述了协议状态和在客户端需要处理的事件。

表16-14 客户端安全会话无效

事 件	条 件	动 作	下一状态
SEC-Create.req		T-Unitdata.req *(ClientHello)*在握手中给出序列号	正在创建
T-Unitdata.ind	*HelloRequest*	SEC-Create-Request.ind客户端可以通过一个SEC-Create.req.初始化一个握手，或初始化一个*alert (no_renegotiation)*，或忽视这个请求	无

表16-15 客户端安全会话生成

事 件	条 件	动 作	下一状态
SEC-Terminate.req		T-Unitdata.req (*Alert (fatal or critical)*)	无
SEC-Exception.req		T-Unitdata.req (*Alert (warning)*)	正在生成
T-Unitdata.ind	*ServerHello Certificate*① *ServerKeyExchange*② *CertificateRequest*③ *ServerHelloDone*	SEC-Create.cnf SEC-Exchange.ind	交换
	ServerHello *Certificate*④ *ChangeCipherSpec* *Finished*	SEC-Create.cnf SEC-提交.ind 由*ChangeChipherSpec*挂起状态，以便在新的状态下可以处理 *Finished*，并将序列号设置成0	已经生成
	Alert (critical or fatal)	SEC-Terminate.ind	无
	Alert (warning)	SEC-Exception.ind	正在生成
重发计时器超时		T-Unitdata.req (*ClientHello*) 不增加序列号重发上一个缓冲区 重发计时器清零 重发计数器加一	正在生成
传输计数器几乎取到最大值		SEC-Terminate.ind	无

①②③④ 表示这些消息是否出现取决于采用的密匙交换方法

表16-16 客户端安全会话交换

事 件	条 件	动 作	下一状态
SEC-Exchange.res		产生一个有下面消息的缓冲区：*ClientKeyExchange*① *CertificateVerify*②	提交1
SEC-Terminate.req		T-Unitdata *(Alert) (critical or fatal))*	无
SEC-Exception.req		T-Unitdata (*Alert* (*warning*))	交换

①② 表示这些消息是否出现取决于采用的密匙交换方法

表16-17 客户端安全会话提交1

事 件	条 件	动 作	下一状态
SEC-Commit.req		列将下列消息添加到缓冲区中：*ChangeCipher-Spec Finished*用*ChangeCipherSpec*产生一个未决状态，以便在新协商的状态下处理 *Finished*，并把序列号清零 发送带有T-Unitdata.req.的缓冲区	提交2

(续)

事 件	条 件	动 作	下一状态
SEC-Terminate.req		T-Unitdata Alert*(critical or fatal)*	无
SEC-Exception.req		T-Unitdata *(Alert(warning))*	提交1

表16-18 客户端安全会话提交2

事 件	条 件	动 作	下一状态
SEC-Terminate.req		T-Unitdata.req *(Alert (critical or fatal))*	无
SEC-Exception.req		T-Unitdata *(Alert(warning))*	提交2
T-Unitdata.ind	Alert (critical or fatal)	SEC-Terminate.ind	无
	Alert (warning)	SEC-Exception.ind	提交2
	Finished	SEC-commit.cnf	OPEN
重发计时器失效（重发超时）	服务器段没有响应	个T-Unitdata.req不增加序列号重发上一个缓冲区 重发计时器清零 重发计数器加一	提交2
重发计数器超过最大值		SEC-Terminate.ind	无

表16-19 客户端安全会话已生成

事 件	条 件	动 作	下一状态
			正在打开
	实现可以立即发送一个不带有用户数据的Finished消息	产生一个带有 *Finished* 的缓冲区，将它随同 T-Unitdata.req发送出去	正在打开
	实现可以延缓发送 *Finished*并将它加在 用户数据前（如果有的话）	产生一个带有 *Finished* 的缓冲区，建立一个加前缀计时器	
SEC-Terminate.req		T-Unitdata.req (*Alert (critical or fatal)*)	无
SEC-Exception.req		T-Unitdata.req *(Alert (warning)*	已生成

表16-20 客户端安全会话正在打开

事 件	条 件	动 作	下一状态
SEC-Unitdata.req		在用户数据前加上缓冲区，并且呼叫T-Unitdata.req	正在打开
SEC-Terminate.req		T-Unitdata.req (*Alert (critical or fatal)*)	无
SEC-Exception.req		T-Unitdata.req (*Alert (warning)*)	正在打开
	设置结束的预定义计时器	在用户数据前加上缓冲区，并且呼叫T-Unitdata.req 删除结束的预定义计时器	正在打开
结束的预定义计时器超时		发送带有T-Unitdata.req.的缓冲区	正在打开

（续）

事 件	条 件	动 作	下一状态
T-Unitdata.ind	接收到用户数据	SEC-Unitdata.ind	打开
	Alert (duplicate_finished_received)		打开
	Alert (critical or fatal)	SEC-Terminate.ind	无
	Alert (warning)	SEC-Exception.ind	打开
	HelloRequest	SEC-Create-Request.ind 客户端可以用 SEC-Create.req初始化一个握手，初始化一个Alert（*no_regeneration*）或忽略这个请求	打开

表16-21 客户端安全会话打开

事 件	条 件	动 作	下一状态
SEC-Create.req		T-Unitdata.req (*ClientHello*)	正在生成
SEC-Terminate.req		T-Unitdata.req (*Alert (critical or fatal)*)	无
SEC-Exception.req		T-Unitdata.req (*Alert (warning)*)	打开
SEC-Unitdata.req		T-Unitdata.req	打开
T-Unitdata.ind	接收到用户数据	SEC-Unitdata.ind	打开
	Alert (*critical or fatal*)	SEC-Terminate.ind	无
	Alert (*warning*)	SEC-Exception.ind	打开
	HelloRequest	SEC-Create-Request.ind 客户端可以初始化一个带有SEC-Create- Request.req，初始化一个 *Alert*（*no_renegotiation*）或忽略这个请求	打开

16.4.2 服务器状态表

表16-22～表16-28给出协议状态和在服务器上处理的事件。

表16-22 服务器安全会话无效

事 件	条 件	动 作	下一状态
SEC-Create-Request.req		T-Unitdata.req *(HelloRequest)* 发送HelloRequests的速率必须有所限制	无
T-Unitdata.ind	*ClientHello*	SEC-Create.ind	正在生成
	Alert (no_renegotiation)	SEC-Exception.ind	无

表16-23 服务器安全会话正在生成

事 件	条 件	动 作	下一状态
SEC-Terminate.req		T-Unitdata.req (*Alert (critical or fatal)*)	无
SEC-Exception.req		T-Unitdata.req (*Alert (warning)*)	正在生成
SEC-Create.res		产生一个缓冲区，这个缓冲区带有： *ServerHello* *Certificate*①	已生成

① 表示这些消息是否出现取决于采用的密匙交换方法

表16-24 服务器安全会话已生成

事 件	条 件	动 作	下一状态
SEC-Exchange.req	完整的握手	将*ServerKeyExchange*① *CertificateRequest*② *ServerHelloDone* 添加到缓冲区末尾，并将它随同T-Unitdata.req发送出去	交换
SEC-Commit.req	优化或简单的握手	将*ChangeCipherSpec Finished*添加到缓冲区末尾 用*ChangeCipherSpec*产生一个未决状态，以便在新协商的状态下处理 *Finished*，并把序列号清零 发送带有T-Unitdata.req.的缓冲区	提交
SEC-Terminate.req		T-Unitdata.req (*Alert (criti-calor fatal)*)	无
SEC-Exception.req		T-Unitdata.req (*Alert (warning)*)	已生成

①② 表示这些消息是否出现取决于采用的密匙交换方法

表16-25 服务器安全会话交换

事 件	条 件	动 作	下一状态
SEC-Terminate.req		T-Unitdata.req (*Alert (critical or fatal)*)	无
SEC-Exception.req		T-Unitdata.req (*Alert (warning)*)	EXCHANGE
T-Unitdata.ind	ClientHello接收到与前一记录相同的记录	重发带有T-Unitdata.req.的上一个缓冲区	EXCHANGE
	ClientHello接收到与前一记录不相同的记录	SEC-Create.ind	正在生成
	Alert (critical or fatal)	SEC-Terminate.ind	无
	Alert (warning)	SEC-Exception.ind	EXCHANGE
	Certificate*	SEC-Exchange.cnf	OPENING
	ClientKeyExchange①	SEC-Commit.ind	

（续）

事 件	条 件	动 作	下一状态
	CertificateVerify②	用*ChangeCipherSpec*	
	ChangeCipherSpec	产生一个未决状态，以便在新协商的状态下处理*Finished*，并把序列号清零	
	Finished	产生一个带有*Finished*新的缓冲区	
		发送带有T-Unitdata.req.的缓冲区	

①② 表示这些消息是否出现取决于采用的密匙交换方法

表16-26 服务器安全会话提交

事 件	条 件	动 作	下一状态
SEC-Terminate.req		T-Unitdata.req (*Alert (critical or fatal)*)	无
SEC-Exception.req		T-Unitdata.req (*Alert (warning)*)	提交
T-Unitdata.ind	*ClientHello*收到一个同前一样的记录	重发带有T-Unitdata.req.的最后一个缓冲区	提交
	*ClientHello*收到同前一个不一样的记录	SEC-Create.ind	正在生成
	Alert (critical or fatal)	SEC-Terminate.ind	无
	Alert (warning)	SEC-Exception.ind	提交
	Finished	SEC-Commit.cnf	打开
	Finished 和用户数据	SEC- Commit.cnf	打开
		SEC-Unitdata.ind	

表16-27 服务器安全会话正在打开

事 件	条 件	动 作	下一状态
SEC-Create-Request.req		T-Unitdata (*HelloRequest*)	正在打开
SEC-Terminate.req		T-Unitdata (*Alert (critical or fatal)*)	无
SEC-Exception.req		T-Unitdata.req(*Alert (warning)*)	正在打开
SEC-Unitdata.req		T-Unitdata.req	正在打开
T-Unitdata.ind	ClientHello	SEC-Create.ind	正在生成
	Alert (critical or fatal)	SEC-Terminate.ind	无
	Alert (warning)	SEC-Exception.ind	正在打开
	收到用户数据	SEC-Unitdata.ind	打开
	Certificate	重发带有T-Unitdata.req.的最后一	
	ClientKeyExchange	个缓冲区	正在打开
	CertificateVerify		
	ChangeCipherSpec Finished		
	收到一组与前一记录相同的记录		

表16-28 服务器安全会话打开

事 件	条 件	动 作	下一状态
SEC-Create-Request.req		T-Unitdata (*HelloRequest*)	打开
SEC-Terminate.req		T-Unitdata (*Alert (critical or fatal)*)	无
SEC-Exception.req		T-Unitdata.req (*Alert (warning)*)	打开
SEC-Unitdata.req		T-Unitdata.req	打开
T-Unitdata.ind	*ClientHello*	SEC-Create.ind	正在生成
	Alert (critical or fatal)	SEC-Terminate.ind	无
	Alert (warning)	SEC-Exception.ind	打开
	收到用户数据	SEC-Unitdata.ind	打开
	Finished	T-Unitdata (*Alert (duplicate_finished_received)*)	打开
	收到与前一个*Finished*相同的记录		
	Finished 和用户数据	SEC-Unitdata.ind	打开
	收到与前一个*Finished*相同的记录	T-Unitdata (*Alert (duplicate_finished_received)*)	

16.5 表示语言

本文档讨论了一个与TLS类似的外部表示的数据格式，我们将会使用下面这种非常基本和有点随意的定义表示语言语法。该语法的结构是从其他几种语言发展过来的，尽管在语法上类似于编程语言C，在语法和意图上类似于XDR[XDR]，然而过多的类比是不恰当的，这个表示语言的意图仅仅是用于记录WTLS，而不是超越这个特定点的通常的应用。

16.5.1 基本块的大小

所有数据项的表示都定义得非常清楚，基本块的大小是1个字节（即8比特）。多字节数据项是从左到右，从顶部到底部的一连串字节。借助于字节流，一个多字节项（在这个例子中是一个数字）通过以下方式构成（使用C的注解语法）：

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

这种有序的多字节值使用的字节次序是网络字节常用的顺序，或称为big endian格式。

16.5.2 其他

注释使用“ /* ”作为开始，使用“ */ ”作为结束。

可选的部分用双括号“ [[ ]] ”括起来。

包含未解释数据的单字节实体的类型是不透明的。

16.5.3 矢量

一个矢量（一维数组）是一串同型的数据元素。矢量的大小可在记录时指定，也可直到运行时才指定。在两种情况下，长度代表了矢量中的字节数，而不是元素的数目。如果要定

义一个新的类型T'，而T'是固定长度类型T的矢量，定义的语法如下：

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

这里T'在数据流里占据了n个字节，n是T的大小的倍数。矢量的长度没有包括在编码的流中。

在下面这个例子中， Datum定义为连续3个协议没有解释的字节，而 Data是3个连续的Datum，共占据9个字节。

```
opaque Datum[3];    /*三个未解释的字节*/
Datum Data[9];     /*9个连续的3字节矢量*/
```

可变长矢量通过指定一个合法子范围来定义，包括使用注解 <floor.，ceiling>。当被编码后，真实的长度在字节流中位于矢量内容的前面，长度将会是能够容纳矢量所指定的最大长度字节数的开销。一个实际长度域为0的可变长矢量被称作空矢量。

```
T T'<floor..ceiling>;
```

在下面的例子中，mandatory是一个必须包含300至400字节不透明类型的矢量，它不能为空，真实长度域占2个字节，即一个uint16，足够表示值400。另一方面，longer可以表示多达800字节的数据或400个uint16元素，可以为空。在编码时在矢量前面要加上 2字节的真实长度域，一个编码后的矢量必须是单个元素长度的偶数倍（例如，一个 17字节的uint16矢量是非法的）。

```
opaque mandatory<300..400>;  长度域为2字节，不可为空*/
uint16 longer<0..800>;  /*到400个16-bit （16比特）无符号整数*/
```

注释

```
A = B[first..last];
```

表示矢量A的元素被赋值为矢量B的第first到第last个元素。

16.5.4 数字

基本的数字数据类型是无符号字节（uint8）。所有整型数字数据类型都是由固定长度的字节序列构成，这些序列中的字节相互连接，并且都是无符号的。

预定义了下面的数字类型：

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

所有在规范中出现的值，都是以“网络”或称为“ big endian ”顺序存储的。一个以十六进制表示的uint32 01 02 03 04等于十进制数16909060。

16.5.5 枚举类型

一个附加的稀疏数据类型称为枚举类型（enum），一个枚举类型的域只能取定义中声明的值，每一个定义是一个不同的类型。只有同一类型的枚举可以相互赋值或比较。如下例所示，每一个枚举必须赋一个值，因为在枚举类型中的元素是无序的，他们可以以任意顺序，被赋以任意单一值。

```
enum { e1(v1), e2(v2), ... , en(vn),  [[(n)]]  } Te;
```

枚举值在字节流中占据它被定义为最大有序值所需要的空间。下面的定义使得类型 Color 的域占据1个字节。

```
enum { red(3), blue(5), white(7) } Color;
```

也可以指定一个不赋予任何标志的值，用来强迫定义需要的宽度，而不必定义一个无用的元素。在下面的例子中，Taste将会在数据流中占2个字节，但仅仅有1，2，4三个值。

```
Enum { sweet(1),sour(2) bitter(4), (32000) } Taste;
```

一个枚举类型的元素名必须在定义类型的范围中。在第一个例子中，一个完全合格的对枚举类型第二个元素的引用应该是 Color.blue。如果被赋值的目标类型指定的非常清楚，则无需如此严格的限制。

```
Color color = Color.blue;  指定的过分严格了，但是合法*/
Color color = blue;  /正确，类型隐含*/
```

对不会被转换成外部表示的枚举型，数字信息可以省略。

```
enum { low, medium, high } Amount;
```

16.5.6 构造的类型

为了方便，可以从原始类型构造出结构类型。每一个说明定义一个新的、唯一的类型。定义的语法非常类似于C语言：

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

一个结构类型内的域可以使用类似于枚举类型的语法来引用。例如， T.f2指前面说明的类型的第二个域。结构定义可以嵌套。

变元

定义的类型可以有基于某些在环境中可知条件的变元。选择器必须是枚举类型，这个枚举类型定义了结构本身定义的可能的变元。对每一个在选择中定义的枚举类型的元素，都必须有一个条件分支，或者是对所有其他元素有一个缺省分支。可以给予变元结构体一个标签，以便引用。在运行时选择变元的机制在演示语言中没有规定。

```
struct {
   T1 f1;
   T2 f2;
   ...
   Tn fn;
   Td fd;
   select (E) {
     case e1: Te1;
     case e2: Te2;
     ...
```

```
      case en: Ten;
      default: TeDefault;
   } [[fv]];
}  [[Tv]];
```

例如：

```
enum { apple, orange } VariantTag;
struct {
   uint16 number;
   opaque string<0..10>;  /*变量长度*/
}  V1;
struct {
  uint32 number;
  opaque string<10>;  /*固定长度*/
}  V2;
struct {
  select (VariantTag) {    /*选择器的值是隐含的*/
     case apple: V1;      /*变元体，tag = apple  */
     case orange: V2;     /*变元体，tag = orange  */
   } variant_body;
} VariantRecord;
```

变元结构可以通过在类型前说明选择器的值来使之合法（窄化）。例如：一个 orange VariantRecord是一个VariantRecord的窄化变元，它包含一个类型为 V2的变元体。

16.5.7 密码属性

有四个密码操作：数字签名、流加密、块加密和公共密匙加密，它们是分别完成的。一个域的加密过程通过在域的类型说明前加上适当的关键字指定而说明，加密的密匙通过当前会话状态来给出。

在数字签名中，使用单向哈希函数作为签名算法。一个进行数字签名的元素被加密为一个不透明矢量<2^{16}-1>，其长度由签字算法和密匙指定。

在流加密中，明文是唯一的，它带有一个标识的输出量，这些输出隐秘地从安全密匙的伪随机码发生器中产生。

在块加密中，每一块明文被加密为一块暗文，所有的块加密都是在 CBC（Cipher Block Chaining密码块连接）模式下进行的，并且所有被块加密的项都是加密块长度的倍数。

在公共密匙加密中，使用了一个公共密匙，以这样的方式加密：它只能被相符的私有密匙解密。一个被公共密匙加密的元素被编码成一个不透明的矢量 <2^{16}-1>，其长度由签字算法和密匙决定。

在下面的例子中，使用哈希的内容作为签名算法的输入，然后，这个结构被块加密器加密。以字节计，这个结构的长度是加密块长度的倍数。

```
block-ciphered struct {
   uint8 field1;
   uint8 field2;
   digitally-signed opaque hash[20];
}UserType;
```

16.5.8 常量

为了规范的原因，可以通过声明一个需要类型的符号，并且对其赋值，来定义一个有类型的常量。用户定义的类型（不透明的，变长的变量以及包含不透明内容的结构）不可被赋值，多元素的结构或矢量的域不能被省略。

例如：

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

Example1 ex1={1, 4};    /赋值 f1=1, f2=4 */
```

16.5.9 串常量

一个串常量必须被解释为具有固定长度字节的矢量。串被包含在标注符中，不像 C语言那样，不使用用于终止的空字符。必须使用 ASCII编码。

```
block = H(parameter, *key expansion * );
```

/*串长度是13字节（没有终止字符） */

16.6 记录协议规范

WTLS记录协议是一个分层协议。记录协议获取需要传送的消息，压缩这些数据或不压缩，使用一个MAC加密，然后传送结果。接收到的数据被解密，验证，然后解压缩，最后分发到高层的客户端那里。

在本文中描述了四个记录协议客户端：改变密码设定协议、握手协议、告警协议和应用数据协议。如果一个WTLS的实现接收到一个它无法理解的记录类型，应该忽略掉。

多个记录可以被连接成一个传送 SDU。例如，几个握手消息可以在一个传送 SDU中发送，这在袖珍设备的传送（如GSM短消息）中尤其有用。

16.6.1 连接状态

一个WTLS连接状态是指WTLS记录协议的运行环境，它规定压缩算法、加密算法和 MAC算法。另外，这些算法的参数也是已知的： MAC的密码、体加密密匙以及读写双向安全连接的IV。

逻辑上，通常有两个连接状态很重要：当前状态和未决状态。所有的记录都在当前状态下进行处理，未决状态的安全参数由WTLS握手协议进行设置，握手协议必须将未决状态转变为当前状态，然后未决状态被重新初始化为空状态。最初的当前状态通常都指明不使用加密、压缩或MAC（见表16-29）。

通过提供下列值来设置WTLS连接状态的安全参数。注意在一个安全会话的协商期间下面的值在客户端和服务器之间通过握手过程中达成一致。

表示语言定义的这些参数如下：

```
enum { server(1), client(2) } ConnectionEnd;
```

```
uint8 BulkCipherAlgorithm;
enum { stream(1), block(2), (255) } CipherType;
enum { true, false } IsExportable;
uint8 MACAlgorithm;
enum { off(0), implicit(1), explicit(2), (255) } SequenceNumberMode;
uint CompressionMethod;
struct {
   ConnectionEnd       entity;
   BulkCipherAlgorithm bulk_cipher_algorithm;
   CipherType          cipher_type;
   uint8               key_size; /字节 */
   uint8               iv_size; /字节 */
   uint8               key_material_length; 字节 */
   IsExportable        is_exportable;
   MACAlgorithm        mac_algorithm;
   uint8               mac_key_size; 字节 */
   uint8               mac_size; /字节 */
   opaque              master_secret[20];
   opaque              client_random[16];
   opaque              server_random[16];
   SequenceNumberMode  sequence_number_mode;
   uint8               key_refresh;
   CompressionMethod   compression_algorithm;
} SecurityParameters;
```

表16-29 连接状态

项 目	描 述
连接端	在本安全会话中这个实体被看作是客户端还是服务器
体密码算法	用于体加密的算法。这里的定义包括了加密算法密匙的大小，这个密匙中有多少是秘密的，是块加密还是流加密，加密的块大小（如果是适当的）以及它是否被看作是“出口加密”。在本章的16.14节中给出了体加密的算法
MAC 算法	用于消息验证的算法。这里定义了 MAC计算的密匙的大小，由 MAC算法返回的哈希的大小。在本章的附录 A给出了MAC算法
压缩算法	在加密前压缩数据的算法。这里必须指明所有进行压缩需要的信息
主密码	在安全连接中两个同层体间共享的一个 20字节的密码
客户端随机数	客户端提供的一个16字节的值
服务器随机数	服务器提供的一个16字节的值
序列号模式	在本安全连接中使用下面何种方案来交流序列号： 1. 隐含编号法（Implicit sequence numbering）： 序列号被作为MAC计算的输入，这要求使用可靠的传送协议 2. 显式编号法（Explicit sequence numbering）： 序列号将会同记录层消息一起以明文发送，并作为 MAC计算的输入。当运行于数据报传送协议时，必须使用这个选择。注意在这种情况下，序列号无需是完全连续的序列，但是发送它们的方式必须无变化（在每个记录中的序列号比前一个大）。 如果验证失败，通常发送*bad_record_moc* 告警 3. 关（Off）： 不使用序列号。不推荐使用这一方案，并且选择这种方案使得系统面对回放攻击时非常脆弱，在这种情况下，更高层必须提供防止这种攻击的保护

（续）

项 目	描 述
密匙刷新	定义了某些连接状态参数（加密匙，MAC密码，和IV）更新的频率。新密匙的计算频率为每 $n=2^{key\_refresh}$ 个消息，就是说，当序列号是 0，n，2n，3n等等的时候 例如，如果选择 3作为key_refresh的值，每8（2^8）个消息（即序列号是 0，8，16等等的消息）生成一个新的密匙集。如果选择 0，则为每个消息（2^0）一个新的密匙集

一旦已经设定安全参数，生成了密匙，目前的连接状态可以通过使它们成为当前状态来示例，这些当前状态必须根据每一个已处理的记录更新。每一个连接状态包括表16-30所列的元素。

表16-30 连接状态包括的元素

项 目	描 述
压缩状态	压缩算法的当前状态。注意当运行于数据报协议之上时，不能使用有状态压缩。如果使用了有状态压缩，在两个方向上有不同的状态
客户端写的MAC密码	由客户端发送的，用于对记录的 MAC 计算和验证的密码，密码必须根据密匙刷新参数更新
客户端写的加密密匙	用于客户端发送的记录的加密，解密的密匙。密匙必须根据密匙刷新参数更新
客户端写的IV	用于计算记录层次IV的基IV，客户端发送的记录使用这个记录层次的 IV在CBC模式下进行块加密
客户端写的序列号	客户端发送的记录所使用的序列号，序列号的类型为 uint16，并且不能超过 2^{16}-1。当建立一个新的连接状态后，第一个记录的序列号为 0
服务器写的MAC密码	本密码用于服务器发送记录的 MAC计算和验证。密码必须根据密匙刷新参数更新
服务器写的加密密匙	用于服务器发送的记录的加密、解密的密匙。密匙必须根据密匙刷新参数更新
服务器写的 IV	用于计算记录层次IV的基IV，服务器发送的记录使用这个记录层次的 IV在CBC模式下进行块加密
服务器写的序列号	服务器发送的记录所使用的序列号。序列号数据类型为 uint16，且不能超过 2^{16}-1。当建立一个新的连接状态后，第一个记录的序列号为 0

16.6.2 记录层

WTLS记录层从更高层接收在非空块中的未解释数据，这些块最大为 2^{16}-1。

1. 分段

不像在TLS中，记录层不对信息块分段。假设传送层已经进行了必要的分段和重组。

```
enum {
   change_cipher_spec(1), alert(2), handshake(3),
    application_data(4), (15)
} ContentType;

enum { without(0), with(1) } SequenceNumberIndication;

enum { without(0), with(1) } FragmentLengthIndication;

struct {
   opaque record_type[1];
   select (SequenceNumberIndication) {
```

```
        case without: struct {};
        case with: uint16 sequence_number;
    }
    select (FragmentLengthIndication) {
        case without: struct {};
        case with: uint16 length;
    }   opaque fragment[WTLSPlaintext.length];
} WTLSPlaintext;
```

对WTLS明文域的描述参见16-31。

表16-31 对WTLS明文域的描述

项 目	描 述
Record_type	定义了处理包括的分段的高层协议。也包括了有关可选域是否存在的信息和加密状态的标志 比特 0-3 长度 4 bits 描述 内容类型 比特 4 长度 1 bit 描述 保留，以备以后使用 比特 5 长度 1 bit 描述 密码设定标识定义了这个记录是否以一个不同于密码设定的方式传送 0＝使用空密码设定 1＝使用当前的，不同于空的密码设定 空密码设定（无 cipher spec）意味着不使用压缩，MAC保护或加密。他的使用限制在开始一个新的会话的握手消息或某些明文发送的告警中 比特 6 长度 1 bit 描述 序列号域标识指出了这个记录的下一个字节是否包括一个序列号域 0＝没有序列号域 1＝包括序列号域 在数据报传送中必须使用序列号域（详细的编号方法请参见16.6.1节） 比特 7 长度 1 bit 描述 记录长度域标识指出了记录是否包含一个长度域： 0＝没有记录长度域 1＝包含记录长度域 在某些情况下，在记录层不发送记录长度是可能的。这减小了每记录2字节的系统开销。对省略该域的要求： 1. 接收机必须能够决定传送SDU的大小。 2. 这是传送SDU中的最后一个记录。 如果两个条件都能达到，每一个同层体都可以决定在每个消息中它们是否使用记录长度域。如果可能，记录长度域应该省略
序列号（sequence_number）	记录的可选序列号。注意在数据报传送中该域必须使用
长度（length）	下面WTLSPlaintext.fragment的可选长度（以字节为单位）。如果几个记录连接成了一个传送SDU，这个域必须使用
分段数据（fragment）	应用数据。这个数据是透明的，并被作为独立的块对待，这个块有类型域指定的更高层协议来处理

注意 不同WTLS记录层内容类型的数据可以相互交织，应用数据在传输中通常位于其他内容类型之后。

2. 记录压缩和解压

所有记录都以在当前状态中定义的压缩算法进行压缩。通常，有一个活动算法，但是，初始时它被定义为无（空）。如果WTLS运行在数据报传送之上，不能使用有状态的压缩算法。

压缩算法把 WTLSPlaintext结构翻译为 WTLSCompressed结构，这意味着 WTLS-Plaintext.fragment被压缩了和拷贝了，其他域（如分段长度）则根据需要更新了。

```
struct {
   opaque record_type[1];
   select (SequenceNumberIndication) {
       case without: struct {};
       case with: uint16 sequence_number;
   }
   select (FragmentLengthIndication) {
       case without: struct {};
       case with: uint16 length;
   }
   opaque fragment[WTLSCompressed.length];
} WTLSCompressed;
```

对WTLSCompressed 域的描述参见表16-32。

表16-32 对WTLSCompressed 域的描述

项 目	描 述
record_type	同章节16.6.2中“分段”
sequence_number	同章节16.6.2中“分段”
Length	下面WTLSCompressed.fragment的可选长度（以字节为单位）参见 16.6.2节中“分段”
Fragment	WTLSPlaintext.fragment的压缩形式

3. 记录有效载荷保护

加密和MAC功能把一个WTLSCompressed结构翻译成一个WTLSCiphertext。解密功能的过程相反见表16-33。

```
struct {
   opaque record_type[1];
   select (SequenceNumberIndication) {
       case without: struct {};
       case with: uint16 sequence_number;
   }
   select (FragmentLengthIndication) {
       case without: struct {};
       case with: uint16 length;
   }
   select (SecurityParameters.cipher_type) {
       case stream: GenericStreamCipher;
       case block: GenericBlockCipher;
   } fragment;
} WTLSCiphertext;
```

表16-33 记录有效载荷保护

项	描 述
record_type	同章节16.6.2中“分段”
sequence_number	同章节16.6.2中“分段”
length	下面WTLSCiphertext.fragment的可选长度（以字节为单位）参见16.6.2节中“分段”
fragment	WTLSCompressed.fragment的压缩形式

(1) 显式序列编号法

当使用显式序列编号法，要求使用特殊的方法来进行记录验证和解密。数据报传送协议中必须使用显式序列编号，即记录可以被丢失、复制或不按顺序接收。接收者必须登记接收到的记录，以便丢弃重复的记录，这可以用滑动窗实现。例如，可以使用大小为 32的滑动窗。利用这一滑动窗，接收者可以在登记序列号 n-32 ... n之间接收到消息。这里n是当前（希望的）序列号。具有序列号n-32的记录必须丢弃。

当以明文消息交换开始一次握手时，序列号从 0开始，每次握手消息增加 1。当在安全连接上开始一次握手时，安全连接的当前序列号被用作握手消息的当前序列号，每次握手消息加1。在上面两种情况中，在 ChangeCipherSpec消息后，它们都被置 0。在重发时，序列号保持和原先消息中的一样。当序列号超过 2^{16}-1时，安全连接必须关闭。

在握手消息中，必须使用序列号（甚至是在面向连接的传送中）。协商后，序列号可用可不用。注意在数据报传送协议中，必须使用序列号。

(2) 零或标准流加密

流加密（包括 BulkCipherAlgorithm NULL）把WTLSCompressed.fragment结构转换为 WTLSCiphertext.fragment 结构，或反过来，把 WTLSCiphertext.fragment结构转换为WTLSCompressed.fragment结构。

```
stream-ciphered struct {
   opaque content[WTLSCompressed.length];
   opaque MAC[SecurityParameters.mac_size];
} GenericStreamCipher;
```

MAC被生成为：

```
HMAC_hash ( MAC_secret, seq_number + WTLSCompressed.record_type +
           WTLSCompressed.length + WTLSCompressed.fragment )
```

其中“+”标识连锁（连接）。如果WTLSCompressed.length不可得，则应使用压缩分段的实际长度来代替。如果根本没有使用序列号，则应该认为假设是 0。

注意除了BulkCipherAlgorithm NULL，目前的WTLS规范中没有定义其他流加密。

(3) CBC块加密（见表16-34）

对块加密（如RC5和DES）而言，加密和MAC功能将结构WTLSCompressed.fragment转换为块结构WTLSciphertext.fragment，或由WTLSciphertext.fragment转换为WTLSCompressed.fragment。

```
block-ciphered struct {
   opaque content (WTLScompressed.length);
   opaque MAC(SecurityParameters.hash_size);
   uint8 padding(padding_length);
   unit8 padding_length;
```

```
} GenericBlockCipher;
```

MAC的生成见16.6.1节。

表16-34 CBC块加密

参 数	定 义
Padding	加入填充位，使普通文本长度增加到块密码长度的倍数。填充数据矢量的每个uint8必须按填充长度值进行填充
Padding_length	填充长度必须符合：结构GenericBlockCipher的长度是密码块长度的倍数。合法的长度值为0-255，包含0，255

加密数据长度（WTLSCiphertext.length）比 WTLSCompressed.length,CipherSpec.hash_size及padding_length三者之和多一位。

例：如果块长度为 8bytes,内容长度（WTLSCompressed.length）为59bytes,且使用10bytesMAC，填充前长度为70bytes。因为70模8运算为6，所以需要2bytes填充长度。

16.7 握手协议规范

WTLS握手协议包括三个子协议，允许对等实体基于记录层采用一致的安全参数，互相验证，初始化协议安全参数，并互相报告出错信息。

握手协议负责进行安全对话的协商，包括的内容参见表 16-35

表16-35 安全对话协商的内容

参 数	定 义
Session Identifier	服务器选择的随机字节序列，用于识别激活或重起的安全对话
Protocol Version	WTLS协议版本号
Peer Certificate	对等实体确认。该参数状态可能为无
Compression Method	加密前用于压缩数据的算法
Cipher Spec	定义批数据加密算法（如无,RC5,DES等）和MAC算法（如SHA-1）。也用于定义加密属性，如mac_size等
Master Secret	客户端与服务器共享的20byte私有数据
Sequence Number Mode	安全连接中所采用的序列编号方法
Key Refresh	定义以多快频率执行某些连接状态值(密钥，MAC私有数据及IV)的计算
Is Resumable	标志位，指示某一安全对话是否可以用来初始化一个新的安全连接

记录层在保护应用数据时，将使用这些数据产生安全参数。采用 WTLS握手协议的重起特征，许多连接就可以使用同样的安全对话进行初始化。

16.7.1 改变密码规范协议

改变密码规范协议应用在加密算法中，用于信号传输。该协议仅包含一个简单信息，用于在当前连接状态下（非待处理状态）进行加密与压缩。信息中包含 1个字节，值为1。

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

参数change cipher spec由客户端或服务器发送，用于通知另一方：以后的数据记录将采用新协商的密码规范和密钥。在握手时，经过双方同意安全参数后并在最后校验信息发送前，

改变后的密码规范信息才被传送。运行中必须检查改变密码规范信息的发送或接收是否在最后校验信息发送或接收之前进行，这样，已结束和接下来的信息将受新的密码规范和密钥的保护。

16.7.2 告警协议

告警类型是WTLS记录层支持的内容类型之一，告警信息传送的内容为信息错误的严重程度及告警描述。

致命性的告警信息将导致立即终止安全连接。在这种情况下，可以继续其他安全连接对话，但对话表示符必须标志为无效，以防止用已经终止的安全对话建立新的安全连接。

严重性达到严重级别的告警信息将导致安全连接的立即终止。这时，其他安全连接对话可以继续，对话表示符也可以保存，用于建立新的安全连接。

告警信息的传送可以由当前连接状态（如压缩与加密）指定或采用无密码（如不进行压缩与加密）。

告警中采用4byte校验和，该值可以通过接收其他对话方的上次记录计算得出，方法如下：

(1) 以0填充记录，使其长度为4的模。

(2) 将结果分割为4byte的长度块。

(3) 对这些块进行XOR运算。

告警接收方应该检查校验和是否与以前所发送的信息相匹配。

```
enum { warning(1), critial(2), fatal(3), (255) } AlertLevel;

enum {
    connection_close_notify(0),
    session_close_notify(1),
    no_connection(5),
    unexpected_message(10),
    bad_record_mac(20),
    decryption_failed(21),
    record_overflow(22),
    decompression_failure(30),
    handshake_failure(40),
    bad_certificate(43),
    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(52),
    unknown_key_id(52),
    disabled_key_id(52),
    key_exchange_disabled(54),
    session_not_ready(55),
```

```
    unknown_parameter_index(56),
    duplicate_finished_received(57),
    export_restriction(60),
    protocol_version(70),
    insufficient_security(71),
    internal_error(80),
    user_canceled(90),
    no_renegotiation(100), (255)
} AlertDescription;

struct {
    AlertLevel level;
    AlertDescription description;
    Opaque checksum[4]
} Alert;
```

1. 结束告警

客户端和服务器必须共享安全连接结束信息，双方可以相互交换结束信息见表 16-36。

表16-36 告警信息及其定义

参 数	定 义
Connection_close_notify	用于通知接收端：发送端不会在该连接状态下再发送任何信息
Session_close_notify	用于通知接收端：发送端不会在该连接状态或该安全对话模式下再发送任何信息

对话双方通过发送告警信息 connection_close_notify或session_close_notify启动结束操作，结束告警后接收的任何信息都将被忽略。该操作需要通信的一方独立发送自身的告警信息 connection_close_notify或session_close_notiry，立即关闭安全连接，并放弃任何未处理的写操作。有告警信息session_close_notify时，接收端必须使对话标志符无效。启动器结束安全连接的读操作时无须等待响应的告警信息 connection_close_notify或session_close_notify。

2. 错误告警

WTLS握手协议中的错误处理非常简单。当检测到某个错误时，检测方发送一个告警信息给另一方，在发送或接收到一个致命错误告警信息后，双方立即结束安全连接，客户端和服务器必须丢弃失败安全连接中的任何对话标志符、密钥和密码。在发送或接收到一个严重告警信息后，双方立即停止安全连接，但可以保留对话标志符以便建立新的安全连接。错误告警定义见表16-37。

表16-37 错误告警的定义

告 警	定 义
no_connection	未与发送端建立安全连接时接收到的信息。该信息为致命或严重错误，以cleartext形式发送
unexpected_message	接收到不恰当信息。该告警为致命或严重错误
bad_record_mac	当接收某个记录时存在错误的 MAC，返回该告警。该信息一般为警告，以cleartext形式发送
decryption_failed	某个WTLS加密文本以无效方式解密，或者其校验长度不是块长或填充值长度的倍数均为出错。该信息一般为警告，以 cleartext形式发送

（续）

告 警	定 义
record_overflow	接收的WTLS加密文本记录长度大于允许的字节长度，或者将某一记录解密为WTLS压缩记录时，其长度大于允许的字节。该信息一般为警告，以 cleartext形式发送
decompression_failure	解压缩时接收到不恰当的输入（如数据解压缩后超过规定长度）。该信息一般为警告，以cleartext形式发送
handshake_failure	接收的握手失败告警信息表示：发送端在给定的选择中不能与接收端协商出可接受的安全参数集。该信息标志致命错误
bad_certificate	鉴权证书为假，包含签名被验证为假等
unsupported_certificate	鉴权证书的类型不被支持
certificate_revoked	签名者撤回鉴权证书。注意，证书撤回只能由服务器检测
certificate_expired	鉴权证书已过期或处于当前无效状态
certificate_unknown	处理鉴权证书时出现了其他事件（非指定），使证书不能被接受
illegal_parameter	握手中某字段超出范围，或与其他字段不一致，该信息通常为致命错误
unknown_ca	收到有效的鉴权序列或其中一部分，但由于不能定位鉴权证书或与已知可信的鉴权书不匹配，因而该鉴权证书不能接受。该信息一般为致命错误
access_denied	接收到了有效的鉴权证书，但在应用接入控制时，发送端对对话协议不予处理。该信息一般为致命错误
decode_error	由于接受信息某些字段长度超出指定范围或信息长度不正确，导致信息不能解码。该信息一般为致命或严重错误
decrypt_error	握手时密码操作失败，包括不能正确校验签名、解密密钥交换信息及检查解密后信息。该信息应以致命错误发送
unknown_key_id	服务器不能识别ClientHello.client_key_ids列表中该客户端标识key_ids，或当服务器需要client_key_ids识别客户端时，客户端不能提供任何参数。该信息为致命错误告警
disabled_key_id	强制使ClientHello.client_key_ids中的client_key_ids失效。该信息通常为严重告警
key_exchange_disabled	密钥交换信息被强制为无效，以防止匿名密钥交换信息被后续不合要求的匿名密钥交换信息所覆盖
session_not_ready	由于监控原因，安全对话未准备好重起新的安全连接，比如因为服务器维护导致对话暂时不能进行。该信息一般为严重告警
unknown_parameter_index	客户端提供一套服务器支持的密钥交换信息，但服务器不知道密钥交换信息的参数索引。接收到此告警信息时，客户端可以启动一个新的握手过程，建议另一个参数索引并清楚地提供给服务器，或者让服务器提供参数
duplicate_finished_received	在简化或经过优化的握手过程中，客户端发送了第二个（重发）结束信息。该信息一般为警告
export_restriction	检测到了一个与输出限制不一致的协议。该信息一般为致命错误
protocol_version	已识别出客户端（或服务器）协商的协议版本号，但服务器（或客户端）不支持（例如，出于安全原因应避免出现老的协议版本号）。该信息一般为致命错误
insufficient_security	对话协商失败时代替hand_failure信息返回的信息。原因是服务器需要的密码安全性比客户端支持的密码安全性高。该信息一般为致命错误
internal_error	内部出错信息由于与对话实体或协议正确性无关，使对话无法继续（如存储器分配出错）。该信息一般为致命或严重告警

（续）

告 警	定 义
user_canceled	由于某些与协议错误无关的原因导致握手过程被取消。如果用户在握手过程结束后要取消某个操作，通过发送 connection_close_notify信息来结束安全连接会更适宜。此告警信息后应跟随信息 xonnection_close_notify。该信息一般为警告
no_renegotiation	在初始握手过程后由客户端发送，以响应服务器的问候请求，或者由服务器发送以响应客户端的问候请求，两者都将导致重新进行协商。当接收不适当，接收端应发送此告警响应。这时，请求发起者可以决定是否继续安全连接

对告警级别没有清晰的定义，发送端可以自行选择其级别为致命或严重告警。当接收端收到一般告警或严重告警时可以自行按致命或非致命告警处理，但在告警级别为致命错误时，所有信息都必须按致命错误信息处理。

在对话执行过程中可能会收到许多告警信息，级别为一般或严重告警，当超过某个配置界限时，接收端将按照致命告警处理它们。

在现有安全对话的基础上进行握手时，致命错误告警只结束即将开始的新对话过程，而对已有对话过程没有影响，但是在某些情况下则必须结束正在进行的对话过程。在建立新对话的握手过程中，对话的一方发送或接收到致命错误告警后，如果决定立即结束正在进行的对话，则必须向对话另一方发送通知信息 session_close_notify。其他情况下，致命错误告警都按本节开始所述处理。

16.7.3 握手协议概述

安全对话的加密参数由 WTLS记录层上的WTLS握手协议产生。当 WTLS客户端和服务器建立通信后，双方就协议版本达成一致，选择加密算法，互相鉴权，并且使用公共密钥加密技术产生双方共享的密码。

WTLS握手协议包含以下一些步骤：

1) 交换问候信息（hello）以取得算法一致性，同时交换随机数值。

2) 交换必要的加密参数供客户端和服务器使用，以达成预加密一致。

3) 交换鉴权证书和加密信息，供客户端和服务器互相鉴权使用。

4) 由预加密和已交换的随机数值生成主控加密。

5) 向记录层提供加密参数。

6) 允许客户端和服务器检查对方是否以相同的加密参数进行计算，以及确认握手时没有受到攻击者的干扰。

上述目标可以由握手协议完成，总结如下：客户端发送问候信息，服务器必须以问候信息响应，否则将被视为致命错误，终止安全连接对话。交换问候信息用以增强客户端和服务器的安全性能。双方的问候信息将建立以下属性：协议版本、密钥交换套件、密码套件、压缩方法、密钥更新以及序列码模式，而且将产生和交换两个随机数值： ClientHello.random和ServerHello.random。

实际密钥交换中有四种信息：服务器鉴权、服务器密钥交换、客户端鉴权和客户端密钥交换。可以通过新方法建立新的密钥交换方法，包括为这些信息指定格式、定义其使用方法，从而使客户端和服务器在公共密码方面达成一致。在无线环境中以 20bytes长度比较适宜。

问候信息发送后，如果要对服务器鉴权，服务器就会发送鉴权证书，而且，在需要时还应发送服务器密钥交换信息（例如，服务器没有鉴权证书或其鉴权证书仅用来签名用）。如果所选择的密钥交换信息适合，服务器可以要求客户端发送鉴权证书（或从其他鉴权分布式设备获得鉴权证书），然后，服务器发送服务端问候结束信息（hello done），表示握手过程中的问候信息阶段已经结束（以前的握手信息与低层信息相结合），并开始等待客户端响应。如果服务器端发送了鉴权请求信息，客户端则必须发送相应的鉴权证书信息。此时，如果客户端鉴权证书没有包含足够的密钥交换信息或证书根本没被发送，就必须发送客户端密钥交换信息，内容根据客户端和服务器问候时所选择的公共密钥加密算法而定。如果采用有签名能力的鉴权证书对客户端进行鉴权（如RSA），则应该发送一个数字签名鉴权校验信息用于校验。

这时，客户端发送一个改变密码规范信息，并将待处理的密码规范（CipherSpec）信息改变为新密码规范信息。客户端随即按新算法、密钥和密码发送结束信息，并将密码规范指示位置1。服务器接收到改变密码规范信息后，也将待处理的密码规范信息改变为新密码规范信息，并相应的按新密码规范发送已结束信息。此时握手过程结束，客户端和服务器可以开始交换应用层数据信息(见图16-5)。

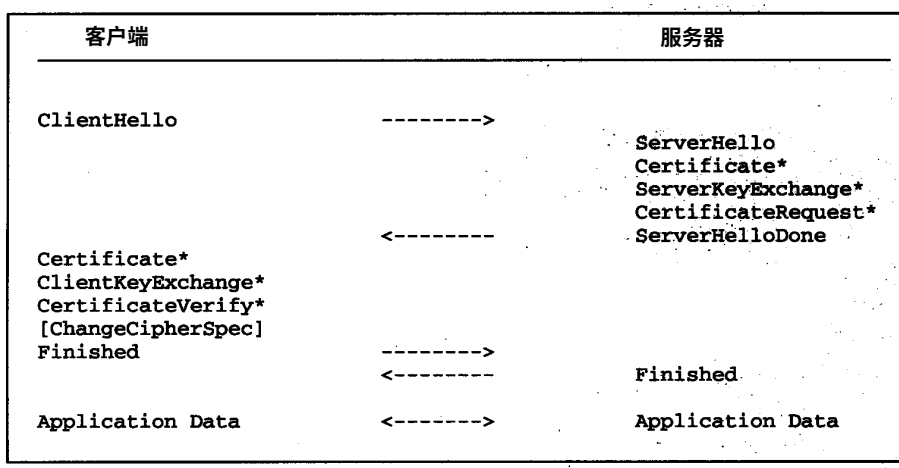

图16-5 完全握手信号的信息流

当客户端和服务器决定不采用协议产生新的安全参数，而是重起一个先前已有的安全对话时，信息流动如下：客户端以重起安全对话的信息标识号发送客户端问候信息(ClientHello)，服务器检查匹配的安全对话缓冲区，如果找到，服务器将按指定的安全对话重新建立安全连接，并发送服务器问候信息（ServerHello），对话标识号值与客户端相同。这时，服务器必须发送改变密码的规范信息，并且直接处理结束信息，这些信息与客户端结束信息相对应。一旦完成对话过程的重新建立，客户端和服务器就可以开始交换应用层数据信息了（见图16-6）。如果没有找到匹配的对话ID标识号，服务器端将产生新的对话标识号，并且TLS客户端和服务器要完成完整的握手过程。

注意，许多并发的安全连接可以由同一个安全对话产生，基于同一安全对话的安全连接共享某些系统参数（如主密码）。

共享密码的握手过程表示新的安全对话过程是基于双方共享的密码（如物理共享），这种情况下，客户端需要共享的密钥交换套件。该信息流类似于图 16-6的握手过程略图。

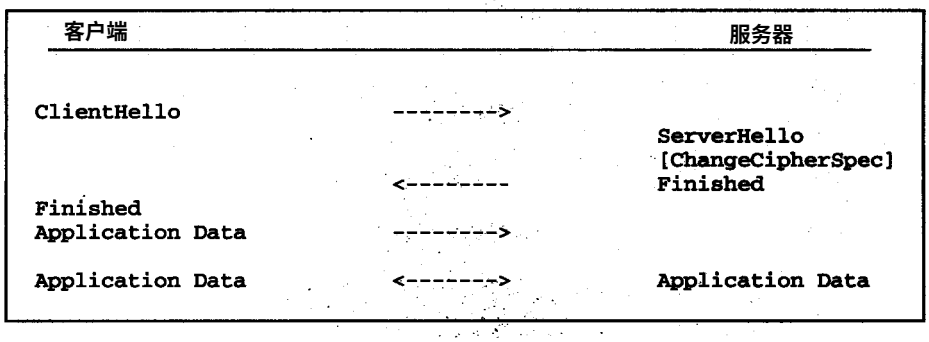

图16-6 简化的握手信号的信息流

另外一个变化是当服务器在收到客户端问候信息后，能够运用认证分发机制或从认证信息的来源处恢复对客户端的认证。在迪福 -海尔曼（Diffie-Hellman）类型的密钥交换方法中，假设迪福-海尔曼参数在认证信息中已被提供，服务器便可以在这时计算出前主密钥和主密钥。在这种情况下，服务器便可以发送认证应答信息，包括一个交换密码特征消息和一个结束消息（见图16-7）。

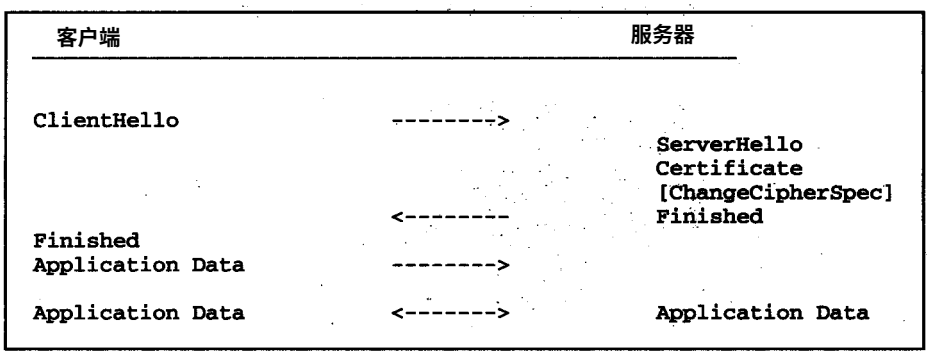

图16-7 优化的完全握手信号的信息流

16.7.4 基于数据报的握手过程的可靠性

在只能使用数据报的情况下，握手过程中的消息可能丢失，也可能次序混乱，甚至会有重复。为了使基于数据报的握手过程可靠， WTLS要求在同一方向上的握手消息必须被连接在一个传输层服务数据单元(SDU)中传输，并且WTLS要求客户端在必要时重传握手消息，而服务器端必须对来自客户端的重传消息进行适当的回应。

在所要求的对端回应到来之前，握手过程可以连续在同一方向上发送多条消息。为了保证在同一传输层服务数据单元中的所有消息按顺序到达，连续发送的多条消息必须被连续放置在一个传输层服务数据单元中来传送或重传。例如，服务器问候消息、交换密码特征消息和结束消息(Finished)可以在同一传输层服务数据单元中传送，这样，握手的过程就被缩短了。在传输层下面一层中的服务数据单元必须足够大以包含上述的那些消息。

对于完整的握手过程来说，如果客户端没有在预先定义的超时时限内从服务器端收到希望的响应，则必须重传客户端问候消息和结束消息。注意包含结束消息的传输层服务数据单

元必须全部重传。当重传次数超过了预先定义的最大重传次数时，客户端会终止握手过程。如果WTP栈已在WTLS栈之上，那么预先定义的时间门限值和计数器最大值就可以通过管理实体从WTP栈获得。

对于已优化和缩短的握手过程来说，如同完整的握手过程，客户端在必要时仍会重传客户端问候消息。另外，当客户端从服务器端收到应用数据消息（Application Data)并成功解密后，客户端应根据应用数据消息来发送结束消息，当客户端从服务器端收到重复结束已收到(duplicated_finished_received)警告后，客户端则应根据该警告来发送结束消息。当然，第一个结束消息既可以单独发送，也可以根据应用数据消息来发送。

对于完整的握手过程，服务器端必须在收到重复的客户端问候消息后，重传包含服务器问候消息的传输层服务数据单元。如果客户端问候消息是新发的，服务器端就必须开始一个新的握手过程，并且生成SEC-Create.ind服务原语。服务器端必须在收到重复的结束消息后，重传包含结束消息的传输层服务数据单元。

对于已优化和缩短的握手过程来说，服务器端处理重复或新的客户端问候消息的行为和完整的握手过程一样。另外，服务器端必须忽略重复的结束消息并保持已承担任务的安全连接完好无损。如果服务器端没有应用数据消息送给客户端，它应该发送重复结束已收到的警告“duplicated_finished_received”。

16.7.5 握手协议

WTLS握手协议是WTLS记录协议中已定义的较高层客户端之一，该协议被用来在安全会话中就安全属性进行协商。握手过程中的消息被提供给WTLS记录层，在该层各消息被密封在一个或多个WTLSPlaintext结构中，该结构在当前特定的活动连接中被处理和传输。

```
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;
struct  {
     handshakeType  msg_type;/*handshake  type*/
     uint16  length;        /*bytes  in  message*/
     select  (msg_type){
            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;
```

握手协议消息必须按顺序发送，不按顺序的发送将会导致严重的错误。不必要的握手消

息可以被省略。注意按序发送中的特殊情况：认证消息（Certificate）在握手过程中被用了两次（从客户端到服务器端，从服务器端到客户端），但仅仅在它的第一个位置上被描述。不被顺序规则所限制的是问候请求消息(Hello Request)，它可以在任何时间被发送，但当它在一个握手过程中到达时，应被客户端忽略。

1. 问候消息

问候消息被用来表示同意客户端和服务器端之间的安全参数。当一个新的安全会话开始时，连接状态（解密，检查，压缩规则）都被初始化为0，在记录中，密码特征指数被设置为0。

(1) 问候请求

该消息发送的时间：问候请求（Hello Request）消息可以被服务器端在任何时候发送。

该消息的含义：问候请求消息只是一个简单的提示，提醒客户端应该在方便的情况下通过发送一个客户端问候消息来重新开始一个协商过程。如果现在正在协商一个安全会话，这个消息将被客户端忽略；如果客户端不想进行一个新的握手过程，或者客户端希望响应一个拒绝再协商(no_renegotiation)警告，那么该消息可以被客户端忽略。因为握手过程中的消息比应用的数据有较高的优先级，所以握手过程将在收到客户端的几个记录后就开始进行协商。如果服务器端发送了一个问候请求，但没有收到客户端的问候响应，它则可以发出致命错误的警告，并关掉安全连接。

在发出问候请求后，服务器端不应该重复该请求，直到后来的握手过程完成。然而，如果客户端不在一个合理的时间内响应，该消息可以被再次发送。

该消息的结构：

```
struct {  }  HelloRequest;
```

注意：该消息不可以包含在持续于握手过程中的各种消息中，同样不可用于结束消息和认证确定消息。

(2) 客户端的问候消息

该消息发送的时间：当一个客户端第一次和一个服务器端相连，它被要求发送客户端问候消息作为第一个消息。客户端也可以发送一个客户端问候消息、响应问候请求或者发送它自己的初始化参数，以便在一个已存在的安全连接中就安全参数进行重新协商。

该消息的结构：密钥交换列表包含了客户端所支持的密码交换规则，这些规则是以递减的顺序排列的。另外，每一个入口定义了客户端希望使用的认证或公共密钥。服务器端将从中选择一个，或在选无可选的情况下，返回一个握手失败（handshake_failure）警告，并关闭安全连接。具有类似格式的可信判断列表确定了客户端所知的可信的认证。

```
struct  {
   uint32  gmt_unix_time;
   opaque   random_bytes[12];
}   Random;
uint8  KeyExchangeSuite;/*Key exchange suite selector*/

struct  {
    uint8  dh_e;
    opaque  dh_p<1..2^16-1>;
    opaque  dh_g<1..2^16-1>;
} DHParameters;
```

表16-38 问候消息及其定义㊀

参 数	定 义
gmt_unix_time	根据发送者的内部时钟所确定的标准的Unix 32位格式的当前时间和日期（秒数从格林尼制时间1970年1月1日午夜开始）。在基本的WTLS协议中，时钟不必正确的设置（如果客户端无法获得时间和日期，它可以在这里设为0），较高级别或应用协议可以定义附加的要求
random_bytes	一个安全随机数产生器产生的12个字节的参数，这个值将在协议的后面被用到

表16-39 参数及其定义㊁

参 数	定 义
dh_e	以字节为单位的说明长度。值为0表明使用的是默认的长度
dh_p	迪福-海尔曼操作所使用的最基本的模数
dh_g	迪福-海尔曼操作所使用的发生器

```
enum  {ec_prime_p(1),ec_characteristic_two(2),(255) }ECFieldID;

enum  {ec_basis_onb,ec_basis_trinomial,ec_basis_pentanomial  }
ECBasisType;

struct {
    opaque  a<1..2^8-1>;
    opaque  b<1..2^8-1>;
    opaque  seed<0..2^8-1>;
} ECCurve;
```

表16-40 参数及其定义㊂

参 数	定 义
a,b	这些参数详细说明了椭圆曲线的系数。每个值为八位字串，代表一个域的值，可根据X9.26协议4.3.1部分转换惯例确定
Seed	这是一个可选的参数，用于获得随机产生的椭圆曲线的系数

```
struct  {
   opaque  point<1..2^8-1>;
} ECPoint;
```

表16-41 参数及其定义㊃

参 数	定 义
point	这是一个八位字串变量，代表一个椭圆曲线上的点，该点根据X9.26协议4.4.2.a部分转换惯例确定。八位字串变量格式根据X9.26协议4.4部分定义

```
struct  {
    ECFieldID   field;
    select   (field)  {
       case  ec_prime_p:opaque  prime_p<1..2^8-1>;
       case  ec_characteristic_two:
          uint16  m;
          ECBasisType  basis;
          select  (basis) {
              case  ec_basis_onb:
                 struct{  }
             case ec_trinomial:
                    opaque  k  <1..2^8-1>;
```

㊀㊁㊂㊃ 表16-38～表16-41所列为程序对应的参数及其定义。

```
            case ec_pentanomial:
                  opaque  k1  <1..2^8-1>;
                   opaque  k2  <1..2^8-1>;
                   opaque  k3  <1..2^8-1>;
        };
    };
    ECCurve   curve;
    ECPoint    base;
    Opaque     order<1..2^8-1>;
    Opaque     cofactor<1..2^8-1>;
}ECParameters;
```

表16-42 参数及其定义㊀

参 数	定 义
Field	该参数确定了限定的域，该域定义了椭圆曲线
prime_p	该参数是对Fp域的附加基本定义
m	该参数定义了对特征二域F2m的度数
k	N次三项式的指数
k1,k2,k3	N次五项式的指数
curve	确定了椭圆曲线E的系数a和b
base	椭圆曲线上的基点P
order	基点的顺序n，基点P的顺序号应取最小的可能整数n，以使nP=0（基点在无穷远处）
cofactor	整数h=#E(Fq)/n,#E(Fq)代表基于Fq域所定义的椭圆曲线E的基点数

```
uint8  ParameterIndex;
enum {rea,diffie_hellman,elliptic_curve} PublicKeyAlgorithm;

struct {

   select  (PublicKeyAlgorithm){
     case  rea:struct{};
     case  diffie_hellman:DHParameters params;
     case   elliptic_curve:ECParameter params;
     }
}ParameterSet;
struct {
   ParameterIndex  parameter_index;
   select (parameter_index){
       case  255:ParameterSet parameter_set;
       default:struct{};
   }
}ParameterSpecifier;
```

表16-43 参数及其定义㊁

参 数	定 义
parameter_index	该参数指示了与密码交换有关的一组参数。0表示无适用参数或在别处指定。1-254表示指定的一组参数号，详细定义见16.14节。255表明参数在下一个域中提供
parameter_set	明确参数形式。例如，迪福-海尔曼或ECDH参数。具体应用时应该用参数索引代替明确的参数

㊀㊁ 表16-42和表16-43所列为程序对应的参数及其定义。

```
Enum{ 无(0),text(1),binary(2),key_hash_sha(254),x509_name(255)}
    IdentifierType;

uint16  CharacterSet;

struct{
   IdentifierType  identifier_type;
   select(identifier_type){
       case 无:struct{};
       case  text:
          CharacterSet  character_set;
          opaque  name<1..2^8-1>;
       case  binary:opaque  identifier<1..2^8-1>;
       case   key_hash_sha:opaque  key_hash[20];
       case   x509_name:opaque  distinguished_name<1..2^8-1>;
}Identifier;
```

表16-44 参数及其定义㊀

参 数	定 义
idenrifier_type	所用标识符的类型 0=无标识符 1=字符形式的文本名字 2=二进制标识 254=公共密钥的SHA-1哈希串 255=X.509协议中的名字
character_set	至IANA的图定义了字符组
name	文本名字
identifier	二进制标识符
key_hash	客户端欲在握手过程中证明自己身份而使用的公共密钥对的哈希值
distinguished_name	X.509协议中的名字

```
struct {
   KeyExchangeSuite  key_exchange_suite;
   ParameterSpecifier  parameter_specifier;
   Identifier  identifier;
} KeyExchangeID;
```

表16-45 参数及其定义㊁

参 数	定 义
key_exchange_suite	指定的密钥交换组数，于16.14节中定义
parameter_specifier	详细说明了密钥交换组的相关参数。密钥交换组所用参数的参数索引值为 0，表明服务器端必须提供参数
identifier	以与密钥交换组有关的方式标识了客户端。服务器端可以用该信息从数据库中获得客户端凭证

在从客户端传送到服务器端的客户端问候消息中，包含了密码套件列表，该列表包括了以客户端所支持的对称解密规则组及其优先顺序。每一个密码套件定义了一组解码规则（包括密码长度）和MAC规则。服务器端将选择一个密码套件，如果没有选择可提供，服务器端

㊀㊁ 表16-44和表16-45所列为程序对应的参数及其定义。

将返回一个握手失败警告和关闭安全连接。

```
struct {
  BulkCipherAlgorithm  bulk_cipher_algorithm;
  MACAlgorithm           mac_algorithm;
}CipherSuite
```

表16-46 参数及其定义[⊖]

参 数	定 义
bulk_cipher_algorithm	指定的解码规则号，详见16.14节
mac_algorithm	指定的MAC规则号，详见16.14节

客户端的问候信息包括客户端支持的压缩规则列表，以客户端的优先权为序。

```
uint8 CompressionMethod;

struct{
   uint8 client_version;
   Random  random;
   SessionID session_id;
   KeyExchangeId client_key_ids<3..2^16-1>;
   KeyExchangeId trusted_key_ids<0..2^16-1>;
   CipherSuite  cipher_suites<2..2^8-1>;
   CompressionMethod  compression_methods<1..2^8-1>;
   SequenceNumberMode  sequence_number_mode;
   uint8  key_refresh;
}ClientHello;
```

表16-47 参数及其定义[⊖]

参 数	定 义
client_version	在安全会话的过程中，客户端希望使用的WTLS协议版本，该版本应是客户端所能支持的最近的版本，对于该版本的规格说明来说，版本号应为1
random	一个客户端生成的随机结构
session_id	客户端使用的安全连接所希望使用的安全会话标识号。当会话的标识号不可得或客户端希望生成新的安全参数时，该域应该为空
client_key_ids	客户端所支持的解码密钥交换选项和标识号列表，以客户端的第一优先顺序为主
trusted_key_ids	客户端所知的可信的凭证列表，以客户端的第一优先顺序为主
cipher_suites	客户端所支持的解码选项列表，以客户端的第一优先顺序为主
compression_methods	客户端支持的压缩方法列表，按客户端的要求排序。这一矢量必须包含并且所有的执行必须支持压缩方法为空。所以，一个客户端和服务器端应能够就某一压缩方法达成一致
sequence_number_mode	该参数值表明了序列号应如何应用于记录层消息中
key_refresh	该参数定义了一些连接状态参数的刷新频率

在发送了客户端问候消息后，客户端等待服务器端的问候消息。除了问候请求外，服务器端发送的任何其他握手消息被认为是关键的或致命的错误。

当客户端有一个已有的会话标识符，并且正在初始化一个缩短的握手过程，它可以在客户端问候消息中省略与密钥交换相关的参数（client_key_ids，trusted_key_ids）。在这种情况

⊖⊖ 表16-46和表16-47所列为程序对应的参数及其定义。

下，如果服务器端不希望继续会话和不能够继续完整的握手过程，它必须返回一个不知密钥标识号（unknown_key_id）警告。

(3) 服务器端问候消息

该消息发送的时间：当服务器端找到一个可接受的规则组后，服务器端将发送该消息以响应客户端的问候消息。如果它不能找到，它必须返回一个握手失败 (handshake_failure)的警告。

该消息的结构（参数及其定义见表 16-48）：

```
struct {
    uint8  server_version;
    Random  random;
    SessionID  session_id;
    uint8  client_key_id;
    CipherSuite  cipher_suite;
    CompressionMethod   compression_method;
    SequenceNumberMode   sequence_number_mode;
    uint8  key_refresh;
} ServerHello;
```

表16-48 参数及其定义

参 数	定 义
server_version	该域将包含客户端在客户端问候消息中建议使用的较低的协议版本和服务器端支持的最高的协议版本。对于该版本的规格说明来说，版本号应为 1
random	该结构由服务器端生成，必须与客户端问候消息中的随机结构不同
session_id	与安全连接一致的安全会话标识号。当客户端问候消息中的会话标识号不空时，服务器端将在它的安全会话缓存中找寻匹配值。如果匹配值可以找到并且服务器端愿意用特定的安全会话建立新的安全连接，则服务器端将响应一个与客户端所提供的一致的标识值。这就表明了一个继续的安全会话，并默认双方必须直接进行到结束消息。否则，该域将包含一个不同的值以标识一个新的安全会话。服务器端可以返回一个空的会话标识以指示安全会话将不被缓存，因此不能继续。如果一个安全会话继续，它必须使用原先协商的同一解码套件
client_key_ids	服务器端从客户端问候消息中的 client_key_ids列表中所选出的密钥交换套件号。例如，值1表示第一个入口被选择了
cipher_suite	服务器端从客户端的问候消息中的 cipher_suites列表中选出的单一解码套件
compression_method	服务器端从客户端的问候消息中的 compression_method列表中选出的单一压缩规则
sequence_number_mode	如果客户端建议使用序列号，那么服务器端必须确认该值。如果客户端没有建议，那么服务器端可以确认或指示应该使用序列号。所以，如果任何一方想用序列号，双方都应该使用
key_refresh	该参数指示了服务器端希望用序列号中的几位去触发密钥的刷新，该值可以小于等于客户端所建议的。因此，较少选择的使用会导致更频繁的刷新和更高的安全性

2. 服务器端的授权消息

该消息的发送时间：该消息必须跟随在服务器端的问候消息后面发送。

该消息的含义：授权类型必须适应于所选的密钥交换套件的规则。可根据 X.509v3协议或WTLS协议的授权消息进行大小的优化，另外的授权消息的类型在将来添加，它必须包含一个

匹配与密钥交换方法的密钥。除非特别指出，授权消息中的签名规则必须与授权消息中的密钥规则一致，公共密钥可以为任意长度。

当指定了新的密钥交换方法的密钥交换套件(KeyExchangeSuite)已在WTLS协议中被确定，则该套件将包含授权格式和所要求的密钥信息。

该消息的结构（参数及其定义见表16-49～表16-51）：

```
enum  {WTLSCert(1),X509Cert(2),(255) }CertificateFormat;

opaque  ASN1Cert<1..2^16-1>;

enum {anonymous(0),ecdsa_sha(1),rsa_sha(1),(255)}
SignatureAlgorithm;

enum {rsa(2),ecdh(3),ecdsa(4),(255)}PublicKeyType;
ECPoint ECPublicKey;
```

表16-49 参数及其定义

参 数	定 义
ECPublicKey	EC公共密钥W=sG

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

表16-50 参数及其定义

参 数	定 义
rsa_exponent	服务器端的RSA密钥指数
rsa_modulus	服务器端的RSA密钥模数

```
Struct  {
   Select(PublicKeyType){
        case  ecdh:ECPublicKey;
        case  ecdsa:ECPublicKey;
        case  rsa:RSAPublicKey;
}PublicKey;

struct  {
  uint8 certificate_version;
  SignatureAlgorithm   signature_algorithm;
  Identifier  issuer;
  uint32  valid_not_before;
  uint32  valid_not_after;
  Identifier  subject;
  PublicKeyType  public_key_type;
  ParameterSpecifier  parameter_specifier;
  PublicKey  public_key;
```

```
}ToBeSignedCertificate;
```

表16-51 参数及其定义

参 数	定 义
certificate_version	授权消息的版本，在本说明中，版本号为 1

表16-52 参数及其定义

signature_algorithm	签署授权消息的规则
issuer	授权消息的发出者，定义了授权消息的签署人，授权消息通常由授权机构(CA)签署
valid_not_before	授权有效期的开始，以标准的 32位UNIX格式表示（秒数自格林尼制时间1970年1月1日午夜开始计算）
valid_not_after	授权有效期的终止，以标准的 32位UNIX格式表示（秒数自格林尼制时间1970年1月1日午夜开始计算）
subject	与被授权的公共密钥有关的密钥的所有者
public_key_type	公共密钥的类型
parameter_specifier	确定了与公共密钥有关的参数
public_key	被授权的公共密钥

哈希值和签名是根据授权签名规则(CertificateSignatureAlgorithm)**所定义的规则从将签署的授权**(ToBeSignedCertificate)**消息中计算所得。**

```
select (SignatureAlgorithm)
{
   case  anonymous:{ };
   case   ecdsa_sha:
     digitally-signed  struct{
          opaque sha_hash[20];/*SHA-1 hash of data to be signed */
     }
case   rsa_sha:
     digitally-signed  struct{
          opaque  sha_hash[20];/*SHA-1 hash of data to be signed */
     }
}Signature;

struct {
    ToBeSignedCertificate  to_be_signed_certificate;
    Signature  signature;
    }WTLSCertificate;
struct  {

  CertificateFormat  certificate_format;
   select(certificate_format){
      case  X.509:  ASN1Cert;
      case  WTLSCert:  WTLSCertificate;
   }
}Certificate;

struct {
    Certificate  certificate_list<0..2^16-1>;
}Certificates;
```

表16-53 参数及其定义

参 数	定 义
certificate_list	这是一系列的授权消息。发送者的授权消息必须在列表中首先出现，随之出现的每一个授权消息必须直接确定它前面的一条消息。因为授权的有效性要求根密钥必须独立的分发，所以，假设远端为了在任何情况下都使授权有效而已经拥有了它，那么为了确定根授权机构而自己签署的授权消息可以在列表中省略

在客户端对授权请求消息的响应中，使用了相同的消息类型和结构。注意，如果客户端没有适当的授权响应给服务器端的鉴权请求，那么客户端可以发送不授权消息。

为了优化通信和客户端的处理，授权链应有最小的长度。对于服务器端的授权来说，可能只有一个授权消息：由单独分发的CA公共密钥确定的服务器端的授权消息。

客户端的授权链可以包含几个授权消息，而由于该链是由服务器处理的，所以这是可以接受的，并且服务器端也可以从一个授权分发服务中获得客户端授权消息。

在一个授权链中，所有的授权消息必须用适合于所选的密钥交换套件的规则。例如：

- 对于RSA，所有的授权消息已经带了用RSA签署的RSA密钥。
- 对于ECDH_ECDSA，第一个授权消息包含了一个由ECDSA签署的ECDH密钥，随后的授权消息则带了由ECDSA签署的ECDSA密钥。

3. 服务器端的密钥交换消息

该消息的发送时间：该消息将在服务器端的授权消息发送后立即发送（如果在一个匿名的协商中，也可在服务器端问候消息后发送。）

只有当服务器端的授权消息无法包含足够的数据去允许客户端交换前主密码时，服务器端的密钥交换消息才被服务器端发送。如在下面的密钥交换方法中：

- ECDH_anon
- RSA_anon
- DH_anon

服务器端密钥交换消息不可以在以下的密钥交换方法中发送：

- ECDH_ECDSA（固定的参数）
- RSA

该消息的含义：该消息传送了解码信息以便客户端传送前主密码或者是一个用于解密的RSA密钥，或者是客户端用来完成密钥交换的迪福 -海尔曼参数。因为WTLS协议中定义的附加密钥交换套件包含了新的密钥交换规则，所以如果与密钥交换规则有关的授权消息类型没有给客户端提供足够的信息以便交换前主密钥，则服务器端发送密钥交换消息。

该消息的结构（参数及其定义见16-54和16-55）：

```
enum {rsa,rsa_anon,dh_anon,ecdh_anon}KeyExchangeAlgorithm;

struct{
    opaque  dh_Y<1..2^16-1>;
}DHPublicKey;
```

表16-54 参数及其定义

参 数	定 义
dh_Y	迪福-海尔曼公共值(Y)

```
struct{
  ParameterSpecifier  parameter_specifier;
  select (KeyExchangeAlgorithm){
     case  rsa_anon:
              RSAPublicKey  params;
     case  diffie_hellman_anon:
              DHPublickey params;
     case  ec_diffie_hellman_anon;
              ECPublicKey  params;
   };
}ServerKeyExchange;
```

表16-55 参数及其定义

参 数	定 义
parameter_specifier	明确说明了与密钥交换套件相关的参数。密钥交换套件所用的参数索引值为0，指示了服务器端将用那些客户端指定的参数。如果客户端没有指定参数，则服务器端必须指定它们
params	服务器端的密钥交换参数(RSA,ECDH或DH公共密钥)

4. 授权请求消息

该消息发送的时间：如果适合于所选的解密套件，服务器端可以向客户端请求一个授权消息，这是可选的。如发送该消息，必须跟随在服务器端授权消息和服务器端密钥交换消息之后。

```
struct   {
  KeyExchangeId  trusted_authorities<0..2^16-1>;
}   CertificateRequest;
```

表16-56 参数及其定义[⊖]

参 数	定 义
Trusted_authorities	可接受的授权机构的名字和类型的列表。这些名字可以确定希望的根CA的标识号或次级CA的标识号，因此，该消息可用于描述已知的根CA和希望的机构空间。如果没有授权机构被发送，则客户端可以送任何授权消息

5. 服务器端的问候已做消息

该消息发送的时间：服务器端的问候已做消息在指示服务器端问候消息结束和相关消息时发送。在发送该消息后，服务器端等候客户端的响应。

该消息的含义：该消息表明服务器端已经发送了消息去支持密钥交换，客户端可以处理密钥交换过程。

在收到服务器端的问候已做消息的基础上，客户端应该确认服务器端提供了所要的有效的授权消息，并检验服务器端的问候消息参数是可接受的。

该消息的结构：

```
struct {   } ServerHelloDone;
```

6. 客户端授权消息

该消息发送的时间：在客户端收到服务器端问候已做消息后，该消息被发送。如服务器

⊖ 表16-56所列为程序对应的参数及其定义。

端请求一个授权消息，该消息被发送。如无合适的授权，客户端必须发送一个包含了无授权的授权消息。如服务器端为了继续握手过程而要求对客户端鉴权，则客户端可响应一个严重的握手失败(handshake_failure)的警告。客户端的授权消息应用了原先为服务器端的授权消息所定义的结构。

7. 客户端密钥交换消息

该消息发送的时间：该消息必须在客户端授权消息发送后发送。否则，客户端只能在收到服务器端问候已做消息后发送该消息。

该消息的含义：通过这个消息，前主密码或通过对 RSA解密密码的直接传送来设置，或通过传送能使双方同意同一前主密码的 EC迪福-海尔曼公共密钥来设置。当密钥交换的方法是ECDH时，客户端授权消息是需要的，并且客户端能响应一个包含匹配于服务器端在它的授权消息中所包含的EC迪福-海尔曼参数，该消息是可以省略的。

该消息的结构：

该消息的结构取决于使用的是何种密钥交换方法。

```
struct {
   select(KeyExchangeAlgorithm){
     case  rsa:RSAEncryptedSecret  param;
     case  rsa_anon:RSAEncryptedSecret  param;
     case  dh_anon:DHPublicKey  param;/*client public value*/
     case  ecdh_anon:ECPublicKey  param;/*client public value*/
   }exchange_keys;
}ClientKeyExchange;
```

(1) RSA加密密文消息

该消息的含义：当RSA被用来形成一致的密钥和鉴权时，客户端生成一个 20字节的密文，用从服务器端的授权消息中的公共密钥加密后，放入加密密文消息中传送。

该消息的结构（参数及其定义见表 16-57和表16-58）：

```
struct{
   uint8  client_version;
   opaque  random[19];
}Secret;
```

表16-57 参数及其定义

参 数	定 义
client_version	客户端所支持的最新版本，它被用来检测旧版本的存在。在收到的密文的基础上，服务器端应该检查该值是否匹配客户端在客户端问候消息中的传输值
random	随机生成的19个安全字节

```
struct  {
    public-key-encrypted Secret  secret;
}EncryptedSecret;
```

表16-58 参数及其定义

参 数	定 义
secret	该值由客户端随机产生。该值附加于公共密钥，被用于前主密钥，该密钥被用来生成主密钥

(2) 客户端EC迪福-海尔曼公共密钥值消息

该消息的含义：该消息在客户端的授权消息不包含 EC迪福-海尔曼公共密钥的情况下，传递客户端的EC迪福-海尔曼公共密钥。该结构是客户端密钥交换消息的一种变化，不包含在该消息中。

(3) 客户端迪福-海尔曼公共密钥值消息

该消息的含义：该消息在客户端的授权消息不包含迪福-海尔曼公共密钥的情况下，传递客户端的迪福-海尔曼公共密钥。该结构是客户端密钥交换消息的一种变化，不包含在该消息中。

8. 授权确定消息

该消息发送的时间：该消息被用来提供对客户端授权消息的明确的确定，它在客户端发出具有签署能力的授权消息后发送。当该消息被发送后，应立即发送客户端密钥交换消息。

该消息的结构（参数及其定义见表 16-59）：

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

表16-59 参数及其定义

参 数	定 义
signature	将签署的哈希值按如下方法计算： H(handshake_message)指的是所有的握手消息。从客户端问候消息开始按序发送和接收的消息，但不包括本消息，同时，它还包括握手层可见的数据和握手消息的类型和长度域。该参数与所有握手结构的交换是相关的 所用的哈希规则是经过握手协商并取得一致的

9. 结束消息

该消息发送的时间：结束消息通常在握手过程的终点发送，表示确认了密钥交换和鉴权过程是成功的。所有的终点必须在一个变化加密投机 (ChangeCipherSpec)消息后改变结束消息。

该消息的含义：结束消息是第一个用刚刚协商好的规则、密钥和密文保护的消息，它的接收者必须确认内容是正确的。一旦一端已经发送了它的结束消息并从对端收到了有效的结束消息，它就可以在安全连接上开始发送和接收应用数据。

该消息的结构（参数及其定义见表 16-60）：

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

表16-60 参数及其定义

参 数	定 义
verify_data	该值按以下方法计算： PRF(master_secret,finished_lable,H(handshake_messages))[0..11];finished_|abe| 客户端发送的结束消息，字符串为 "client finished"; 服务器端发送的结束消息，字符串为 "server finished"; handshake_messages 所有的按客户端或服务器端发送顺序出现的握手消息，但不包括该条消息中的所有的数据。在握手层但不包括记录层消息头中的可见数据。该参数与所有握手结构的交换是相关的

如果在一个握手过程的相应点上一个变化加密投机消息后没有结束消息，将导致一个严重或致命的错误。

握手消息的值包括了从开始的客户端问候消息的所有的握手消息，但不包括结束消息。客户端发送的结束消息中的握手消息值不同于服务器端发送的结束消息中的握手消息值，因为第二个发送的将包含前面一个。

注意 变化加密投机消息、警告及任何其他的记录类型不是握手消息并且不包含在哈希计算中。问候请求消息在握手过程的哈希计算中是省略的。

16.8 加密计算

16.8.1 计算主密文

为了进行消息保护，WTLS记录协议要求一套确定的规则，一个主密文和客户端、服务器端的随机值。加密规则和MAC规则由服务器端所选择的加密套件决定并在服务器端问候消息中显示，密钥交换和鉴权规则由密钥交换套件决定并在服务器端问候消息中显示，压缩规则在问候消息中协商，随机值在问候消息中交换。所有剩余的用来计算主密文。

对于所有的密钥交换方法，同样的规则用来将前主密文转化为主密文。当主密文已计算出时，前主密文应从内存中删除。

```
master_secret=PRF(pre_master_secret,"master secret",
                  ClientHello.random+ServerHello.random) [0..19];
```

主密文通常为20个字节的长度。前主密文将根据密钥交换方法来变化。

1. RSA加密方案

当RSA被用来进行服务器鉴权和密钥交换时，一个 20字节的密文值由客户端生成，由服务器端的公共密钥加密，并发送到服务器端，服务器端用自己私有的密钥对密文解密。前主密文已附加了服务器端的公共密钥，如前所述，双方将前主密文转化为主密文。

在RSA签名中，一个20字节的哈希结构SHA-1被签署（以私有的密钥加密），用PKCS#1块类型1。

RSA公共密钥加密用PKCS#1块类型2执行。

2. 迪福-海尔曼

常规的迪福-海尔曼计算方法被执行。协商后的密钥被用来最前主密文，并被转变为主密文，如前所述。

3. EC 迪福-海尔曼

EC迪福-海尔曼计算方法被执行。协商后的密钥被用来最前主密文，并被转变为主密文，如前所述。

椭圆曲线计算根据[P1363]所述的方法执行。

EC参数可以被明确的传送，或使用一个详细说明了预定义参数的规则定义。（见16.14节）

EC点根据椭圆曲线到八位字符串原语代表，执行时应使用点压缩。

ECDSA签名和确认根据椭圆曲线签名方案执行：

- EMSA用SHA-1进行哈希计算。计算将签署数据的哈希值。
- 椭圆曲线签名原语， DSA版本(ECSP-DSA)的签名，椭圆曲线确认原语， DSA版本

(ECVP-DSA)的确认。

- ECSSA的输出格式，签名以八位字符串格式输出。
- 密钥Z的ECDH计算根据[P1363]进行：
- 用椭圆曲线密文值获得原语，迪福 -海尔曼版本(ECSVDP-DH)去产生一个共享的密文值Z作为一个域的元素。
- 用域元素到八位字符串转化原语(FE2OSP)将共享的密文值Z转化为八位字符串Z。

4. 会话继续消息

在会话继续中，主密文不再计算，这表明一个继续的会话使用了与前一个会话相同的主密文。

注意 虽然使用同样的主密文，新的客户端问候随机值和服务器端问候随机值再缩短的握手过程中被交换。在密钥块的生成中使用这些随机值，这意味着每一个安全连接以不同的密钥初始值开始。

16.8.2 密钥计算

一个连接的状态是记录协议的操作环境。从握手协议所提供的安全会话参数中产生连接状态（压缩密钥，IVs,MAC密文）需要一个规则。

一个连接的状态以下述方法计算：

主密文被哈希放置在一系列已指定为压缩密钥， IVs,MAC密文的字节中。为了生成密钥源，计算：

```
key_block=PRF (SecurityParameters.master_secret,
               expansion_label,seq_num+
               SecurityParameters.server_random+
               SecurityParameters.client_random);
```

直到需要的输出数量生成。

一个新的密钥块的生成发生在序列号的间隔处，与密钥刷新频率一致。计算中使用的序列号是要求密钥刷新的第一个号。

不同的扩展标志值用于客户端写密钥和服务器端写密钥，所以，密钥块生成“ client expansion ”作为扩展标志值，分割为以下各项：

```
client_write_MAC_secret[SecurityParameters.hash_size]
client_write  encryption_key[SecurityParameters.key_material]
client_write  IV[SecurityParameters.IV_size]
```

密钥块生成“ server expansion ”作为扩展标志值，分割为以下各项：

```
server_write_MAC_secret[SecurityParameters.hash_size]
server_write  encryption_key[SecurityParameters.key_material]
server_write  IV[SecurityParameters.IV_size]
```

在WTLS协议中，许多连接状态参数可以在安全连接期间被重新计算，这一特点被称为密钥刷新，它是为了减小新的握手过程而被执行。在密钥刷新中， MAC密文值、加密密钥和IV将根据序列号而变化。刷新频率根据密钥刷新参数。例如，密钥刷新每四条记录执行一次。上述计算中所用的序列号参数是触发密钥刷新的记录序号。如果序列号根本不用，则在所有的计算中序列号为0。

可输出的加密规则（SecurityParameter.is_exportable值为真）要求如下的处理以获得它们的最终的写密钥：

```
final_client_write_encryption_key=
   PRF(SecurityParameter.client_write_encryption_key,"client write key",
   SecurityParameters.client_random+SecurityParameter.server_random);

final_server_write_encryption_key=
   PRF(SecurityParameter.server_write_encryption_key,"server write key",
   SecurityParameters.client_random+SecurityParameter.server_random);
```

可输出的加密规则只从问候消息的随机值中获得它们的 Ivs。

```
iv_block=PRF(" ","IV block",seq_num+
   SecurityParameters.client_random+SecurityParameters.server_random);
```

如同密钥块，iv块也被分割为两个矢量：

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

注意在这种情况下PRF没有用密文，这意味着密文长度为0字节并对PRF的哈希值无贡献。

对CBC状态块密码，每一记录的IV以下述方法计算：

```
record_IV=IV XOR  S
```

IV值是原先的IV值(客户端写IV值或服务器端写IV值)，S是将两个字节的记录序列号连接起来而获得的，因此需要从 IV中多次获得足够多的字节。也有可能加密规则支持将序列号用做输入，那么，记录序列号被用做规则序列号。

16.8.3 HMAC和伪随机函数

在WTLS记录和握手层中的一系列操作要求一个密钥 MAC，这是由密文保护的安全摘要。

另外，将密文扩展成数据块以便密钥生成和生效需要一个构造过程。伪随机函数 (PRF)以密文、密根和标识符为输入，生成任意长的输出。

1. MAC计算

HMAC[HMAC]可以用不同的哈希规则。例如， SHA-1[SHA]或MD5[MD5]可以用。用密码写的哈希函数以H表示。另外，密文密钥K是需要的。我们假设H表示用密码写的哈希函数，对数据块的重复的基本压缩过程生成了哈希数据。 B(B=64表示哈希函数前的所有例子)表示该数据块的字节长度，L表示哈希输出的字节长度 (L=16 MD5,L=20 SHA-1)。鉴权密钥K取决于B，使用密钥长度超过B的应用应该首先用H计算密钥的哈希值，然后取L个字节作为HMAC的实际密钥。在任何情况下，推荐给 K的最小长度是L个字节（哈希输出长度）。

我们定义了两个固定的、不同的字符串 ipad和opad（“ i ”和“ o ”代表内部和外部）如下所见：

```
ipad=the byte 0x36 repeated B times
opad=the byte 0x5c repeated B times
```

在我们执行的数据上计算HMAC值：

```
H(K  XOR  opad + H (K XOR ipad+data) )
```

加代表连接。也就是说：

1) 在K的末尾附加0以生成B个字节的字符串（例，K长度为20字节，B=64，则K后附加44个0字节0x00）。

2) 将ipad和步骤 1)中算出的B字符串取异或(XOR)。

3) 在 2)中的B字符串后附加数据。

4) 将H用于步骤 3)的数据中。

5) 将opad和步骤 1)中算出的B字符串取异或(XOR)。

6) 将步骤4)中的H附加在步骤五中的B字符串后。

7) 将H用于步骤 6)的数据中，输出结果。

2. 伪随机函数

在TLS标准中，为了使PRF尽可能的安全，使用了两个哈希规则，为节省资源， WTLS可以只用一个哈希规则。实际已经使用的哈希规则在握手过程中作为加密投机消息的一部分取得一致。

首先，我们定义一个数据扩展函数， P_hash（密文，数据）用一个单一的哈希函数将密文和密根扩展成大量的重复输出。

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

“+”代表连接。

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

P_hash可以重复多次以产生所要求的数据量。例如，如果 P_SHA用来生成64字节的数据，它将重复5次，生成80个字节输出数据，最后的16个数据将被丢弃，剩下64个字节的输出数据。那么，

```
PRF(secret,label,seed)=P_hash (secret,label+seed)
```

16.9 术语定义

本规范采用了下列术语：

简化的握手过程（Abbreviated handshake） 在已有的安全会话基础上的连接状态，见“会话恢复”。

连接状态（Connection State） 记录协议的操作环境。连接状态包含了加密操作的所需要的参数（加密/解密和MAC计算/确认），每个安全连接都有安全状态。

数据报传输（Datagram Transport） 一种不能保证传输层的SDU不被丢失、复制、乱序的传输层服务。

握手过程（Handshake） 在客户端和服务器端之间就协议选项达成一致的过程。它包括安全参数协商（例如，运算法则和密钥长度）、密钥交换和鉴权。握手过程在每一个安全连接的开始时进行。

握手过程协议（Handshake Protocol） 进行握手的协议。

完整的握手过程（Full Handshake） 对等端之间生成一个新的安全会话。完整的握手过程包括客户端和服务器端的参数协商和公共密钥信息交换。

优化的握手过程（Optimized Handshake） 对等端之间生成一个新的安全会话。与完整的握手过程不同的是，服务器端从自己的资源中寻找客户端授权信息，而不要求从客户端发送。

记录（Record） 记录协议层的协议数据单元。

记录协议（Record Protocol） 记录协议传送消息，可以选择压缩数据、应用 MAC、加密和传送结果。接收到的数据被解密、确认、解压缩，传递给高层客户端。在本文档中有四种记录协议：握手协议、警告协议、改变加密投机协议 (ChangeCipherSpec)和应用数据协议。

安全连接（Secure Connection） 有连接状态的WTLS连接。每一个安全连接由通信的对等端传输地址确定。

安全会话（Secure Session） 安全会话在握手协商过程中进行。协商的参数（会话标识，运算规则，主密文）被用来生成安全连接。每一个安全会话由服务器端分配的会话标识确定。

会话恢复（Session Resume） 一个新的安全连接可以在事先协商好的安全会话的基础上建立，所以，如果存在安全会话，无须执行完整的握手过程和加密计算过程。例如，一个安全连接可以被终止并在以后恢复。许多安全连接可以使用同一安全会话通过 WTLS握手协议的可恢复特性来建立。

共享密文鉴权（Shared Secret Authentication） 一种基于共享密文的鉴权方法。该方法无需公共密钥规则即可进行，但要求前主密文嵌入或人工输入到客户端和服务器端。共享密文是敏感信息，所以需要安全的信道发送。

16.10 缩略语

本规范采用了下列缩略语：

API	Application Programming Interface	应用程序接口
CA	Certification Authority	鉴权
CBC	Cipher Block Chaining	密码块链接
DH	Diffie-Hellman	迪福-海尔曼
EC	Elliptic Curve	椭圆曲线
ECC	Elliptic Curve Cryptography	椭圆曲线密码学
ECDH	Elliptic Curve Diffie-Hellman	迪福-海尔曼椭圆曲线
ECDSA	Elliptic Curve Digital Signature Algorithm	椭圆曲线数字签名规则
IV	Initialization Vector	初始矢量
MAC	Message Authentication Code	消息认证代码
ME	Management Entity	管理实体
OSI	Open System Interconnection	开放系统互连
PDU	Protocol Data Unit	协议数据单元
PRF	Pseudo-Random Function	伪随机函数
SAP	Service Access Point	业务接入点
SDU	Service Data Unit	业务数据单元
SHA-1	Secure Hash Algorithm	安全杂凑算法
SMS	Short Message Service	短消息业务

SSL	Secure Sockets Layer	安全套接层
TLS	Transport Layer Security	传输层安全
WAP	Wireless Application Protocol	无线应用协议
WDP	Wireless Datagram Protocol	无线数据报协议
WSP	Wireless Session Protocol	无线会话协议
WTLS	Wireless Transport Layer Security	无线传输层安全
WTP	Wireless Transaction Protocol	无线事务协议

16.11 参考标准

[WAPARCH] "WAP Architecture Specification",WAP Forum,30-April-1998
URL:http://www.wapforum.org/

[WAPWDP] "Wireless Datagram Protocol Specification", WAP Forum,30-April-1998
URL:http://www.wapforum.org/

[WAPWTP] "Wireless Transaction Protocol Specification", WAP Forum, 30-April-1998
URL:http://www.wapforum.org/

[RFC2119] "Key Words for Use in RFCs to Indicate Requirement Levels", Bradner,S.,March 1997
URL:ftp://ftp.isi.edu/in-notes/rfc2119

[TLS] "The TLS Protocol",Dierks,T.and Allen,C.,November 1997
URL:ftp://ftp.ietf.org/internet-drafts/draft-ietf-tls-protocol-05.txt

[RFC2068] "Hypertext Transfer Protocol-HTTP1.1",Fielding,R.,et.al.,January 1997
URL:ftp://ftp.isi.edu/in-notes/rfc2068

[HMAC] "HMAC:Keyed-Hashing for Message Authentication",Krawczyk , H. , Bellare , M.,and Canetti,R.,RFC 2104,February 1997
URL:ftp://ftp.isi.edu/in-notes/rfc2104.txt

[SHA] "Secure Hash Standard",NIST FIPS PUB 180-1,National Institute of Standards and Technology,U.S. Department of Commerce, DRAFT， May 1994

[X509] "The Directory-Authentication Framework", CCITT , Recommendation X.509 , 1998

[3DES] "Hellman Presents no Shortcut Solution To DES",Tuchman,W., IEEE Spectrum,v.16 ,n.7,July 1979, pp.40-41

[DES] "American National Standard for Information System-Data Link Encryption"，ANSI X3.106 , American National Standards Institute,1983

[DH1] "New Directions in Cryptography", Diffie, W. And Hellman M.E.,IEEE Transactions on Information Theory,V.IT-22, n.6, Jun 1977,pp.74-84

[DSS] "Digital Signature Standard",NIST FIPS PUB 186 , National Institute of Standards and Technology,U.S. Department of Commerce， May 1994

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

[MD5] "The MD5 Message Digest Algorithm",Rivest,R., RFC 2104,April 1992. URL:ftp://ftp.isi.edu/in-notes/rfc2104.txt

[PKCS1] "PKCS#1:RSA Encryption Standard",version 1.5, RSA Laboratories, November 1993

[RSA] "A Method for Obtaining Digital Signature and Public-Key Cryptosystems",Rivest, R., Shamir, A., and Adleman L. M., Communications of the ACM, v.21, n.2, Feb 1978,pp. 120-126

[RC5] "The RC5, RC5-CBC, RC5-CBC-Pad, and RC5-CTS Algorithms",Baldwin, R.and Rivest R., RFC 2040, October 1996 URL:ftp://ftp.isi.edu/in-notes/rfc2040.txt

[P1363] "Standard Specification For Public Key Cryptography", IEEE P1363/D1a (Draft Version 1a), February 1998 URL:http://grouper.ieee.org/groups/1363/

[X9.62] "The Elliptic Curve Digital Signature Algorithm (ECDSA)",ANSI X9.62 Working Draft , November 1997

16.12 参考资料

[WAPWSP] "Wireless Session Protocol Specification", WAP Forum,30-April-1998 URL:http://www.wapforum.org/

[GSM03.40] "European Digital Celluar Telecommunication System(phase 2+):Technical Realization of Short Message Service (SMS) Point-to-Point(P)",ETSI

[XDR] "XDR：External Data Representation Standard",Srinivansan,R.,RFC-1832:,August 1995 URL:ftp://ftp.isi.edu/in-notes/rfc1832.txt

[ISO7498] "Information Technology-Open System Interconnection-Basic Reference Model:The Basic Model", ISO/IEC 7498-1:1994

[ISO10731] "Information Technology-Open System Interconnection-Basic Reference Model-Conventions for the Definition of OSI Services", ISO/IEC 10731:1994

16.13 确认

WTLS来自[TLS]。TLS 基于SSL3.0规范。

16.14 算法定义

算法定义可参见表16-61～表16-63。

表16-61 可获得的密钥交换套件

密钥交换套件	指定数字	描 述	密钥长度限制(位)
NULL	0	无密钥交换可做。使用了一个长度为0的前主密文。主密文和结束消息仅用来检错	N/A
SHARED_SECRET	1	基于对称密钥的握手过程。各方共用一个如前主密钥的密文密钥	None

(续)

密钥交换套件	指定数字	描 述	密钥长度限制(位)
DH_anon	2	无鉴权的迪福-海尔曼密钥交换。各方彼此发送DH公共密钥，各方基于自己的私有密钥和对方的公共密钥计算前主密文	None
DH_anon_512	3	同DH_anon，但DH密钥长度有限	512
DH_anon_768	4	同DH_anon，但DH密钥长度有限	768
RSA_anon	5	无鉴权的RSA密钥交换。服务器端发送它的RSA公共密钥，客户端生成一个密文值，用服务器端的公共密钥加密并发送给服务器。前主密文是密文值附加服务器端的公共密钥组成	None
RSA_anon_512	6	同RSA_anon，服务器端公共密钥长度有限	512
RSA_anon_768	7	同RSA_anon，服务器端公共密钥长度有限	768
RSA	8	基于RSA授权的RSA密钥交换。服务器端发送一个授权消息，包含了它的 RSA公共密钥。服务器端的授权消息是由客户端信任的第三方用 RSA签署的，客户端从收到的授权消息中提取服务器端的公共密钥，生成密文值，用服务器端的公共密钥加密，发送给服务器端。前主密文是密文值加服务器端的公共密钥。如客户端要认证，它将用它的RSA私有密钥签署一些数据（握手过程中发送的消息）并发送它的授权消息和签署过的数据	None
RSA_512	9	同RSA，但授权的服务器端公共密钥长度有限	512
RSA_768	10	同RSA，但授权的服务器端公共密钥长度有限	768
ECDH_anon	11	无鉴权的EC迪福-海尔曼密钥交换。各方彼此发送 ECDH公共密钥，以它们的私有密钥和对方的公共密钥计算前主密文	None
ECDH_anon_113	12	同ECDH_anon，但ECDH密钥长度有限	113
ECDH_anon_131	13	同ECDH_anon，但ECDH密钥长度有限	131
ECDH_ECDSA	14	基于ECDSA授权的EC迪福-海尔曼密钥交换。服务器端发送一个授权消息，包含了它的ECDH公共密钥。服务器端的授权消息是由客户端信任的第三方用 ECDSA签署的。根据客户端是否被认证，它发送一个授权消息，包含了它的 ECDH公共密钥。该授权消息是由服务器端信任的第三方用 ECDSA签署的，或仅仅是它的 ECDH公共密钥。各方基于自己的私有密钥和对方的公共密钥计算前主密文，前主密钥或独自接收或包含在授权信息	None

表16-62 可获得的块加密规则

密 码	指定值	可输出	类 型	密钥源（字节）	扩展密钥源（字节）	有效密钥位（位）	IV大小（字节）	块大小（字节）
NULL	0	TRUE	STREAM	0	0	0	0	N/A
RC5_CBC_40	1	TRUE	BLOCK	5	16	40	8	8
RC5_CBC_56	2	TRUE	BLOCK	7	16	56	8	8
RC5_CBC	3	FALSE	BLOCK	16	16	128	8	8
DES_CBC_40	4	TRUE	BLOCK	5	8	40	8	8
DES_CBC	5	FALSE	BLOCK	8	8	56	8	8
3DES_CBC_EDE	6	FALSE	BLOCK	24	24	168	8	8
IDEA_CBC_40	7	TRUE	BLOCK	5	16	40	8	8
IDEA_CBC_56	8	TRUE	BLOCK	7	16	56	8	8
IDEA_CBC	9	FALSE	BLOCK	16	16	128	8	8

表16-63 可获得的块加密规则

域	定 义
可输出	可输出域值为真的加密规则为了适应一定的出口规则有长度有限的有效密钥。因此,一个特殊的密钥扩展被执行,初始化向量以特定的方法获得。该定义并不意味着是否从一个国家向另一个国家输出这些规则确实合法（或可输出域值为非时,输出这些规则为非法）
类型	指示在CBC状态下执行的是流密码还是块密码
密钥源	为了生成写密钥而使用的密钥块中的字节数
扩展密钥源	写密钥中的字节数
有效密钥位	加密规则中的密钥源中的信息量
IV大小	初始向量所需要的数据,流密码为0,等于块密码中块的大小
块大小	在一个大块中的块密码的数据量,CBC状态下执行的块密码仅仅能加密一个块的大小

RC5是块密码规则族。RC5的执行可特指为RC5-w/r/b,w是词的位数(块大小的一半)，r是回合的数量，b是密钥的字节长度。使用该表示法，RC5_CBC就是RC5-32/16/16。RC5_CBC_40作为输出密码执行。使用5个字节作为密钥源并扩展为16个字节，再应用RC5_32/12/16；RC5_CBC_56作为输出密码执行,使用7个字节作为密钥源并扩展为16个字节,再应用RC5_32/12/16。

数据加密标准(DES)是一个可广泛应用的对称加密规则，它有56位密钥和8字节块的块密码。注意,在WTLS中,为了生成密钥,DES被认为8字节的长度,但仍然提供56位的保护。DES也可以在每一数据块使用3个独立的密钥和3个加密法的状态下操作，这使用了168位(WTLS密钥生成法中24个字节)的密钥，等于112位的安全性。

IDEA是由Xuejia Lai 和James Massey设计的64位块密码（见表16-64～表16-68）。

表16-64 64位块密码IDEA

域	定 义
密钥大小	HMAC密钥使用的字节数
哈希大小	MAC使用的字节数

表16-65 MAC密钥规则

哈希函数	指定值	描述	密钥大小（字节）	哈希大小(字节)
SHA_0	0	无MAC密钥可计算。注意在其他的非MAC操作中，全长的SHA-1被使用	0	0
SHA_40	1	输出的SHA-1中的前5个字节被使用来计算MAC密钥。注意在其他的非MAC操作中，全长的SHA-1被使用	20	5
SHA_80	2	输出的SHA-1中的前10个字节被使用来计算MAC密钥。注意在其他的非MAC操作中，全长的SHA-1被使用	20	10
SHA	3	SHA-1被使用来计算MAC密钥	20	20
SHA_XOR_40	4	5个字节的校验和。输入的数据被分为5字节大小的块，所有块前后取异或，如最后的一个块少于5字节，以0填充。SHA比XOR在生成MACs要强大，虽然没有关于XOR MACs的不利报告，但它必须被加密并使用于CBC状态块密码中。XOR仅使用于CPU资源有限的设备中 警告：可输出级别的加密法中，XOR不能提供与SHA一样强的消息综合保护性。在这些环境中采用XOR MAC法之前，建议安全后果应被仔细的评估。在其他的对消息综合性的MAC操作中，全长的SHA-1被使用	0	5
MD5_40	5	输出的MD5中的前5个字节被使用来计算MAC密钥。注意在其他的非MAC操作中，全长的MD5被使用	16	5
MD5_80	6	输出的MD5中的前10个字节被使用来计算MAC密钥。注意在其他的非MAC操作中，全长的MD5被使用	16	10
MD5	7	MD5被使用来计算MAC密钥	16	16

表16-66 压缩规则

压缩规则	指定值	描述
无	0	无压缩

表16-67 所选曲线的椭圆曲线参数

参数	值
Assigned number	1
Field size	113

（续）

参 数	值
Elliptic curve E	$y^2+xy=x^3+ax^2+b$ over F (2^{113})
Curve parameter a	1
Curve parameter b	1
Generating point G	01667979A40BA497E5D5C270780617， 00F44B4AF1ECC2630E08785CEBCC15
The order of G	00FFFFFFFFFFFFFFFDBF91AF6DEA73
The cofactor k	2
Assigned number	2
Field size	131
Elliptic curve E	$y^2+xy=x^3+ax^2+b$ over F (2^{131})
Curve parameter a	0
Curve parameter b	1
Generating point G	043A891E4FD64F01E60F8831C3D7E195B22FF19BEE, 04035AB7114A900F460549987F48C3B1F00B5A1D58
The order of G	020000000000000004D4FDD5703A3F269
The cofactor k	4
Assigned number	3
Field size	163
Elliptic curve E	$y^2+xy=x^3+ax^2+b$ over F (2^{163})
Curve parameter a	1
Curve parameter b	1
Generating point G	02FE13C0537BBC11ACAA07D793DE4E6D5E5C94EEE8， 0289070FB05D38FF58321F2E800536D538CCDAA3D9
The order of G	040000000000000000000020108A2E0CC0D99F8A5EF
The cofactor k	2

表16-68 预定义的迪福-海尔曼参数

参 数	值
Assigned number	1
Exponent bits	160
Prime modulus(512 bits)	FAF30C63D171E54A8131CD331D7C8D6C 8AED41B0354E1A29D8DAD03E2E67FF8E 00053A07FD28A1EE6AF199FD70330EA8 C4C602B86EDFBF47FD1D7BFB6456BD57
Generator(512 bits)	E7734EBBCF50893C760181B2AA2DB0AC F2D5B6E775EE88BAFC7AA5A6BB20A64E B9F54301141F90291B7B375135394504 81C9F9CB2BA3E67B4580E2153FD22B80
Assigned number	2
Exponent bits	160
Prime modulus(768 bits)	85DB5DB185090AED3BDB3BABFCB46669F9563E681EDB4359 9241FEF6AA9B5DF9EFE39C0CB7994A04F2BD8F57B5B22AF7 5E360526216420BCA08FCDF98FF6417DCFDD1C40E4FFB183 260E3B28EF0B31A3633788C988B1BC6734A81B31A28CD6FB
Generator(760 bits)	1B15C3C57263B0DD1A9D996768B88370ED458D7B0081A220 054EFDD23B9CD8298B719FD3B67CB093817332D033642D21 130F83D9CB2CC5ACDD36E6E6DDB2410AB30311CDBEE9222C CFE644443B0C7204F2D12F7A3719C8866A20A0E778EBBA

16.15 执行注意点

以下的执行注意点用来标识实现选择可能影响 WTLS协议的安全性、性能和有效性的地方，它给协议的执行者提供指导。

16.15.1 协商空特定密码

在会话中可以协商使用空特定密码。空密钥交换套件可以由于该原因而被使用 ,所以没有密钥交换工程发生。主密文用 0长度的前主密文计算。消息流类似缩短的握手过程。

当收到一个空特定密码时,执行时必须仔细,因为它不提供安全性。

16.15.2 匿名握手过程

完全匿名会话可以使用 RSA或迪福-海尔曼建立以便密钥交换。用匿名 RSA,客户端生成一个密文值，并且用从服务器端的密钥交换消息中提取的未授权的公共密钥来加密，结果放在客户端的密钥交换消息中发送。因为偷听者不知道服务器端的私有密钥，所以他们无法解开密文值(前主密文是由该值附加服务器端的公共密钥而成的)。

用迪福-海尔曼法,服务器端的公共密钥值包含在服务器端的密钥交换消息中，在客户端的密钥交换消息中发送。偷听者不知道私有密钥值的，无法发现迪福 -海尔曼结果。

警告：完全匿名握手过程 (例如,客户端和服务器端都未被认证)仅提供了对被动偷听者的防范措施，主动偷听者或主动的第三方攻击人可以在创建会话的握手过程中以他们的消息代替结束消息。然而,现在已有方法能在可能遭受攻击的环境中有效地击败那些主动的攻击。例如,服务器端认证或使用一个独立的防干扰信道去证实结束消息没有被攻击者替换。当握手过程完毕并被认证或证实 ,已建立的会话应是安全的并可防范第三方攻击或偷听者的被动的和主动的攻击。

16.15.3 密钥刷新

WTLS协议的被动密钥刷新机制使在安全连接中无需握手过程就可以更新密钥成为可能。密钥刷新使密码分析对攻击者的吸引力大为减小 ,因为密钥经常无效,可获得的源码很有限。在输出受限的加密过程和握手过程代价昂贵的环境中 ,该方法特别有效(例如,希望连接长期有效)，密钥的刷新频率在握手过程中取得一致，该参数定义了在密钥刷新被触发前有多少消息被发送。例如, 每4条消息被发送后，密钥刷新被触发。

在密钥刷新中,使用主密文作为信息源,伪随机函数中的消息序列号作为附加参数 ,来生成一个新的密钥块。生成的密钥块被用来作为消息保护密钥： MAC密钥、加密密钥和初始向量。

注意：密钥刷新经常由执行 /迫使一个新的握手过程来做。

16.15.4 拒绝服务的攻击

因为WTLS在数据报的基础上操作 ,执行应小心以防拒绝服务的攻击。应考虑在一些网络传输地址中，假造是相对容易的，为了使拒绝服务的攻击难以完成 ,攻击者不能从一个假造的地址仅凭一个明文消息就破坏一个存在的连接 /会话。另外,服务器端应在一个存在的安全连接

中接收一个明文连接请求时必须仔细，需注意的是服务器端不能忽略它们。例如，明文客户端问候消息可能由连接状态丢失的客户端发送。在服务器端响应了客户端问候消息后立刻转换到待处理状态的判优和优化握手过程中尤其要注意。在此情况下，旧主动状态应保持直到新的握手过程完成，换句话说，服务器端应不丢弃旧的主动状态直到客户端响应了结束消息，并且握手过程成功完成。如果握手过程在开始时无效是明显的，则旧的主动状态应恢复到当前状态。同样的原因，当一个客户端在它的安全连接上收到一个明文服务器端问候消息时，它不能因为意外的消息而打断现存的安全连接，它应保持现存的安全连接并发送意外消息作为警告。

16.16 执行类型

WTLS协议执行可以支持多种特征。本节定义了指导执行者选择这些特征的各种类型，一种类型可以有主要的和可选的对一个特定的特征的支持，特定的特征在当前的说明版本中没有定义。

该版本的WTLS协议的详细说明涵盖了类型1的所有特征（见表16-69）。

表16-69 执行类型

特 征	类 型 1	类 型 2	类 型 3
Public-key exchange	M	M	M
Server certificates	O	M	M
Client certificates	O	O	M
Shared-secret handshake	O	O	O
Compression	-	O	O
Encryption	M	M	M
MAC	M	M	M
Smart card interface	-	O	O

16.17 WTLS协议的要求

表16-70所列为无线移动网络的共同要求。

表16-70 WTLS协议的要求

参 数	描 述
Datagram transport protocol	必须支持数据报和面向连接的传输层协议。必须能处理丢失的、复制的、乱序的数据报，而不必破坏连接状态
Slow interactions	协议必须考虑有载体（如：SMS[GSM03.40]）的往返时间是较长的。例如，发送一个查询和收到一个响应要求10秒。在协议设计中必须考虑
Low transfer rate	一些载体速度较慢是主要的局限。所以，开销必须保持最小。例如，用SMS有效传输率将减少100bit/s
Limited processing power	许多移动终端的处理效率是有限的，选择加密规则时必须考虑
Limited memory capacity	许多移动终端的内存容量较有限，所以，加密规则数应最小，并选择小规模的规则。特别对RAM要求应尽可能的低
Restrictions on exporting and using cryptography	必须考虑使用、输出、输入加密法的国际限制和规则。这意味着一定可以根据每个地区的法律来获得最佳的安全性。例如，在许多情况下，强加密被禁止而强鉴权是可用的