■**文档属性**

|  |  |
| --- | --- |
| 标题 | Mssql登录协议分析 |
| 编号 |  |
| 编写日期 | 2018-1-8 |
| 修改日期 |  |
| 作者 |  |
| 产品名称 |  |
| 产品型号 |  |
| 产品版本号 |  |

文档修订记录

|  |  |  |  |
| --- | --- | --- | --- |
| 日期 | 版本号 | 修订说明 | 修订人 |
| 2018-1-8 | V1.0 | 创建 | 周擎阳 |
| 2018-3-6 | V1.0 | 增加流程图 | 周擎阳 |
|  |  |  |  |

**目 录**

TDS协议概述 2

TDS登陆协议分析 3

Pre-login阶段 3

Login阶段 7

## TDS协议概述

Mssql协议是基于TDS协议进行通讯的。所以sql-server进行登陆的操作就是双方利用TDS协议进行信息交换的过程。

TDS是闭源协议。微软只放出了部分有关文档。因此协议分析比较困难。

TDS协议格式如下：

|  |  |
| --- | --- |
| head | data |

对于head部分，我们可以细分为：

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| Type（1） | Status（1） | Length（2） | SPID（2） | PackID（1） | Window（1） |

Type：决定了这个包的作用。Dbshield中就是通过type类型判断这个包属于那种类型，进而推断这个属于那个过程。

Status：决定当前包的状态。

Length：包长

Spid：服务器端进程号，多线程时有用。

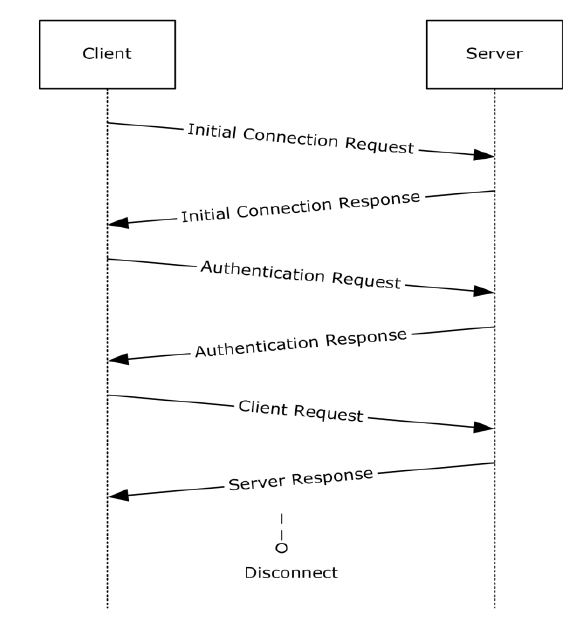
Packid：包编号，每当收到一个包，编号加一。

Window：保留位，恒为0x00。

对于data部分，一般有两种存储方式：

1. Token data： 数据存储在后面的data部分，前面token的类型，偏移值和数据长度等数据，用于定位。
2. Tokenless data：数据直接存储在当前位置。

## TDS整体流程分析



我们这个文档分析,着眼点在initial connection 和 authentication 两个部分.

## TDS登陆协议分析

TNS登陆分为两部分：

Pre-login阶段

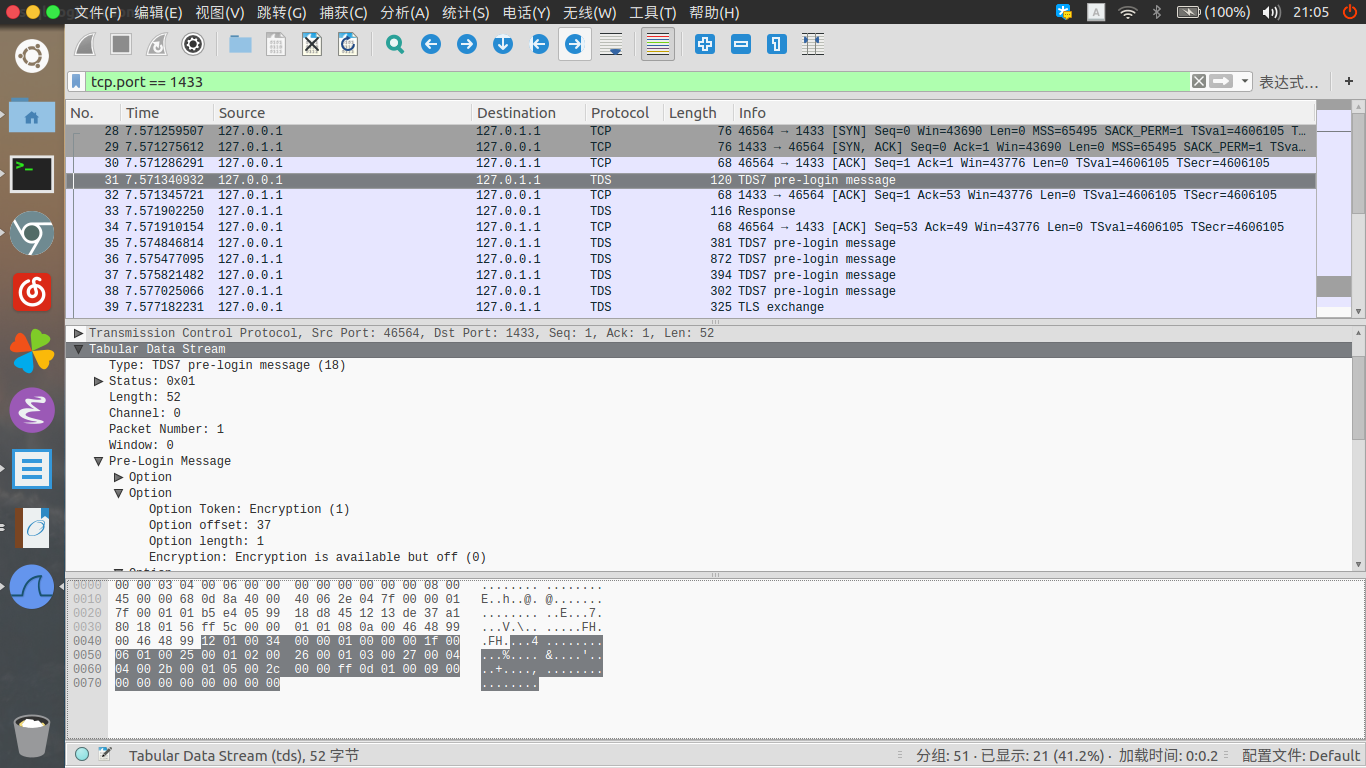
Pre-login阶段交换有关信息，如版本号。进程ID等。最重要的是其中商议了是否加密。如果加密了，接下来server会和client商议加密细节。此时数据包头仍然是pre-login的type。

**Pre-login特征**：

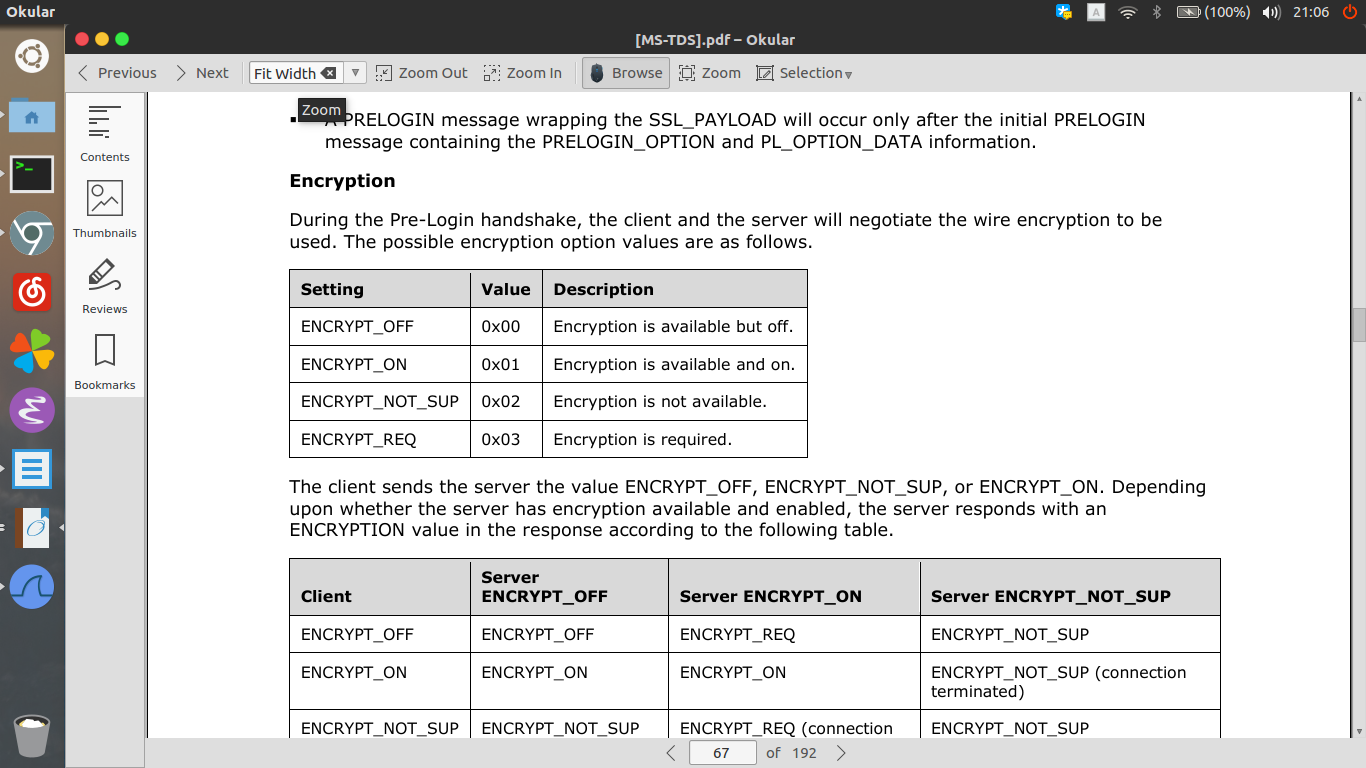
Head中Type值为0x12.  
**具体过程：**

1. Client 向服务器发送0x12的pre-login包
2. Server紧接着向client发送0x04的回应包。向client描述了自己的状态。
3. 如果启用ssl，则接下来进行ssl握手。

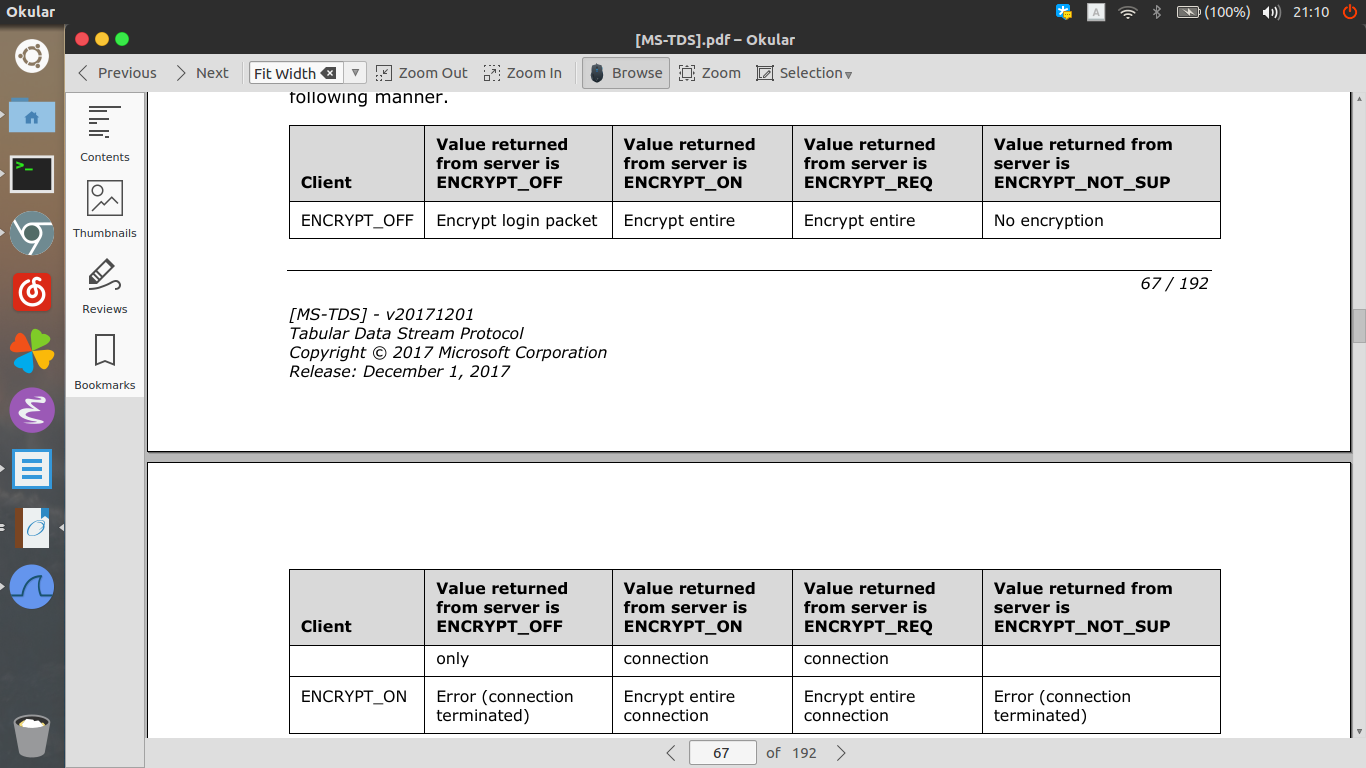
**截图说明**：



这里必须要说明一下pre-login商议的加密方式：



这四个值，是client发给server的。Server根据自身状态回复，然后client根据下表选择操作：



根据这个表，我们可以很清晰的看到，即使双方交流的都是encypt\_off，接下来动作也会是加密。就是说，在默认情况下，TDS一定会进行ssl握手进行加密。

**代码对应**：

我们必须看到，dbshield的github主页上并没有mssql服务器选项。但是在他们代码中我们找到了mssql相关分析。具体原因我认为是因为mssql代码还不够完善，导致他们还不能把支持mssql放到首页上：

//Receive PreLogin Request

buf, err := m.reader(m.client)

if err != nil {

return

}

if buf[0] != 0x12 {

err = errors.New("packet is not PRELOGIN")

return

}

//Send PreLogin to server

\_, err = m.server.Write(buf)

if err != nil {

return

}

//Receive PRELOGIN response

buf, err = m.reader(m.server)

if err != nil {

return

}

if buf[0] != 0x4 {

err = errors.New("packet is not PRELOGIN response")

return

}

//Send PreLogin to server

\_, err = m.client.Write(buf)

if err != nil {

return

}

//Set data to beginning of the prelogin message

data := buf[8:]

var encryption byte

//Lookup Encryption

for i := 0; i < len(data); i += 5 {

switch data[i] {

case 0x01: //Encryption

encryption = data[int(data[i+1])]

break

case 0xff: //Terminator

break

}

}

logger.Debugf("Encryption: %v", encryption)

可以看出，代码就是简单的接收pre login消息进行判断然后转发。并且根据上表，我们知道他提取的是否加密变量encryption也是错误的。是否加密不能单纯用回应包进行判断。

### Login阶段

在交换过信息后，login阶段分为了2种情况进行处理。

**Login包特点：**

客户端发送的Login7包头的type数值为0x10。

服务器端回应的loginack包头数值为0x04.

**具体过程：**

1. Client 向服务器发送0x10的pre-login包

2. Server紧接着向client发送0x04的回应包。描述了是否验证成功。

**具体过程：**

数据包中后很多选项，包括是否更改密码，client名称等。但是最主要的是用户名-密码。如果没有设置加密，根据相关文章，密码是通过移位方式进行加密。如果设置了，就会根据具体加密方法进行加密。

根据freetds，具体的函数：

unsigned char \*  
tds7\_crypt\_pass(const unsigned char \*clear\_pass, int len, unsigned char \*crypt\_pass)  
{  
 int i;

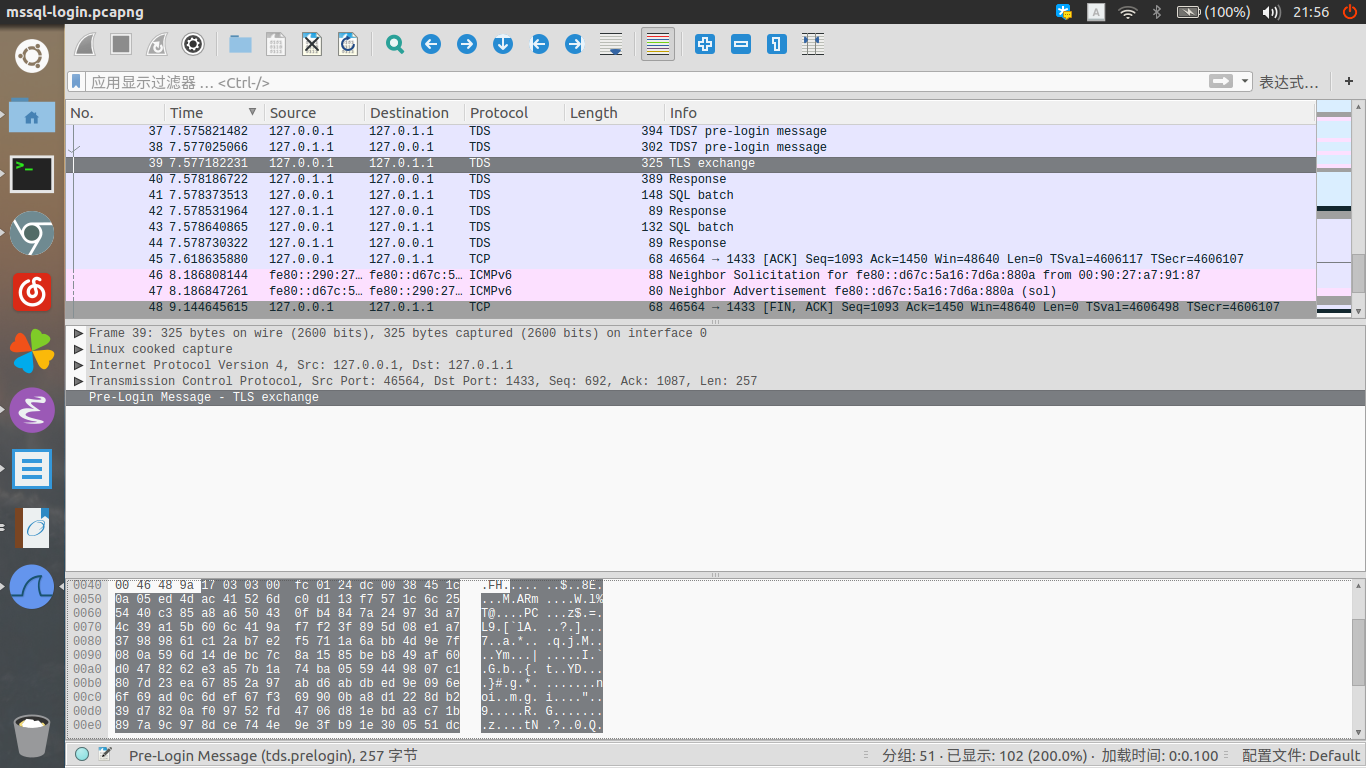
 for (i = 0; i < len; i++)  
  crypt\_pass[i] = ((clear\_pass[i] << 4) | (clear\_pass[i] >> 4)) ^ 0xA5;  
 return crypt\_pass;  
}

可以看出加密手段非常初级。

但是在实际操作中，情况又有所不同。

在wireshark抓包中，我们发现找不到0x10开头的包，但是能收到0x04关于登陆的回执包。因此我们怀疑，在新的tds协议中，登陆方式有所改变。

具体操作中



这个0x17开头的tls exchange包进入了我们的视线。他是在ssl握手刚刚结束发出的包，并且紧接着就是server回应登陆的0x04包。因此我们有理由怀疑这个就是0x10包。

在微软提供的文档中，找不到有关0x17包的内容，只是简单的提及了以0x17开头的包是sspi包。

在微软非官方（社区）中，我找到了有关解释：这其实是wireshark的问题。这个包并不是TDS包而是TLS包，0x17是tls标志。根据文档中有关过程：

 If **login-only encryption** was negotiated as described in section Message Syntax in the PreLogin  
  message description then **the first and only the first TDS packet of the Login message MUST be  
  encrypted using TLS and encapsulated in a TLS message**. All other TDS packets sent or received  
  MUST be in plaintext.  
  \* If **full encryption** was negotiated as described in section Message Syntax in the PreLogin message  
  description, then **all subsequent TDS packets sent or received from this point on MUST be  
  encrypted using TLS and encapsulated in a TLS message**.

可以看出，确实TDS是被TLS加密了。因此0x10找不到的问题也就被解决了。

**代码分析：**

Dbshield这一段代码有很大问题：

for {

//Receive PreLogin Request

err = readWrite(m.client, m.server, m.reader)

if err != nil {

return

}

//Receive PRELOGIN response

buf, err = m.reader(m.server)

if err != nil {

return

}

//Send PreLogin to server

\_, err = m.client.Write(buf)

if err != nil {

return

}

if buf[0] == 0x4 {

break

}

}

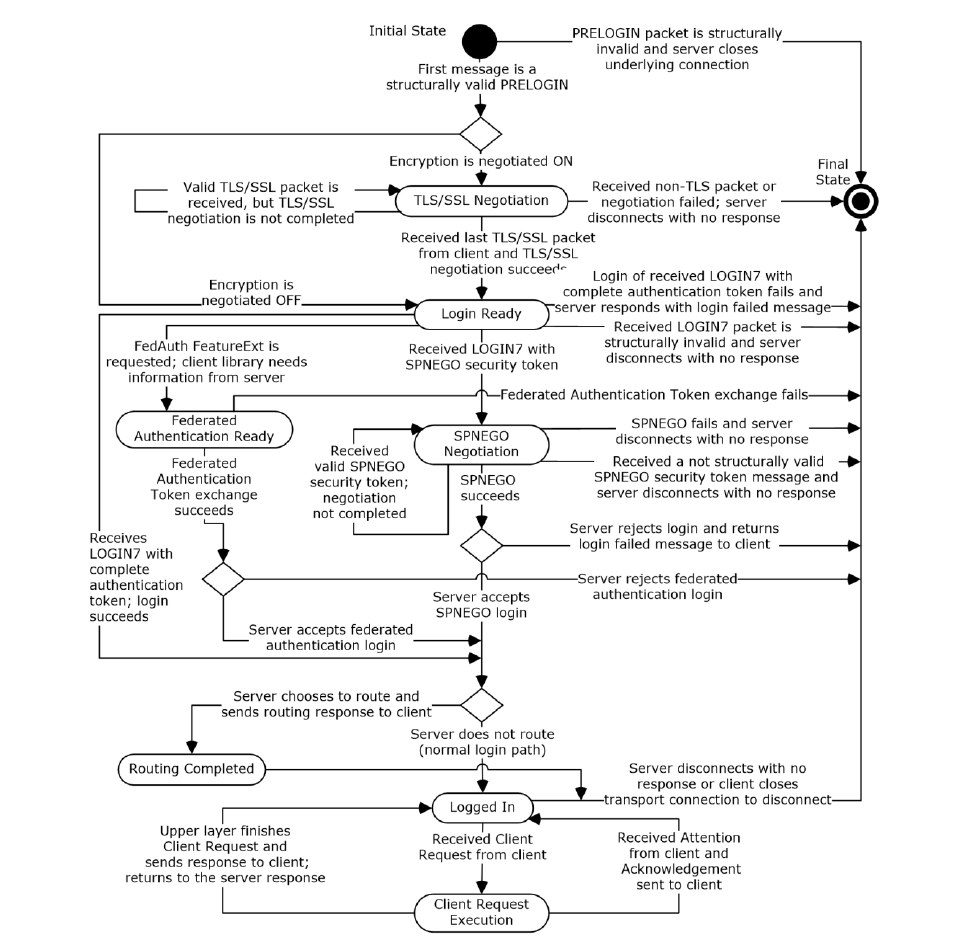
首先注释就有错误：这并不是pre-login阶段而是整个login阶段。很明显这是直接复制的上面的代码而且没有经过检查。此外这段代码就是做了一个转发，其他什么都没有做。最后收到一个0x04包就认为登陆完成。我认为这一段代码没有任何意义。将来可能需要很大的代码量去更正。

**流程图:**

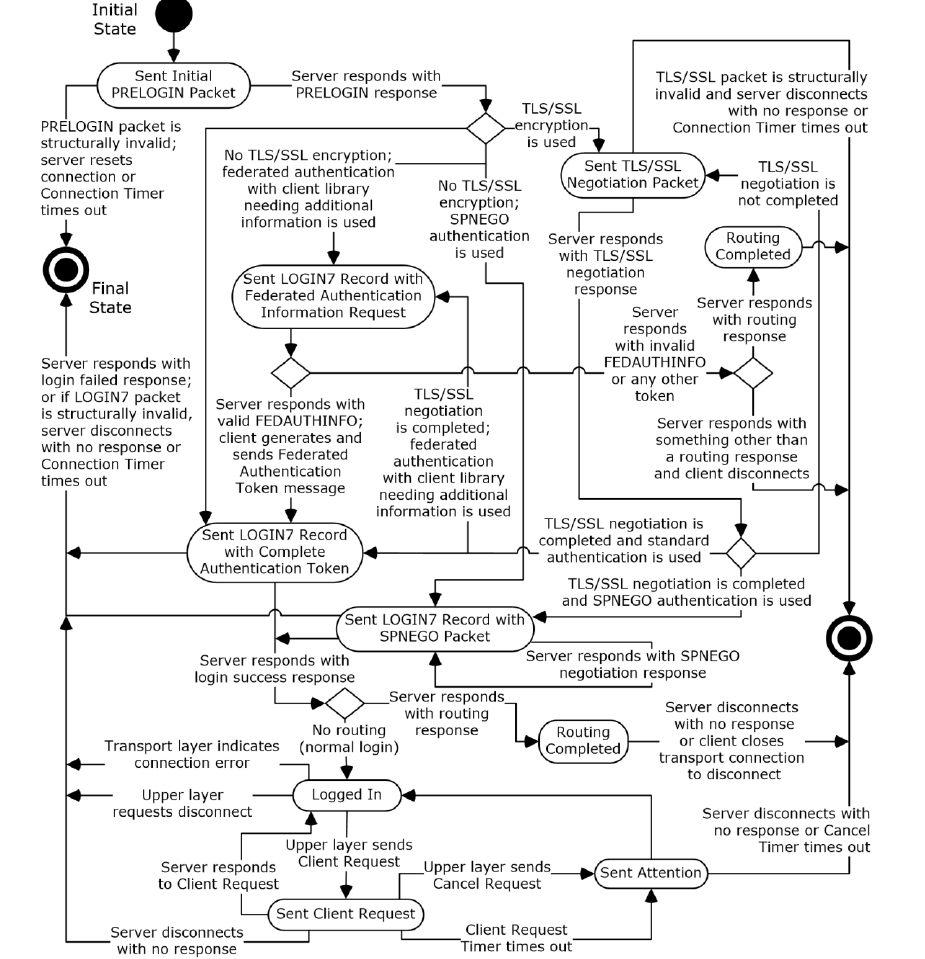
Pre-login结束后,login阶段分为2种情况进行处理:

1. 直接进行加密,利用sspi协议进行加密,用spengo进行验证.这个协议是微软以相关协议为基础写出的.
2. 利用 federated authentication进行联合认证.这需要传输附加信息,以及client端需要附加的库.

服务器端流程图:



客户端流程图:



参考文献:

官方文档:

https://msdn.microsoft.com/en-us/library/dd303260.aspx