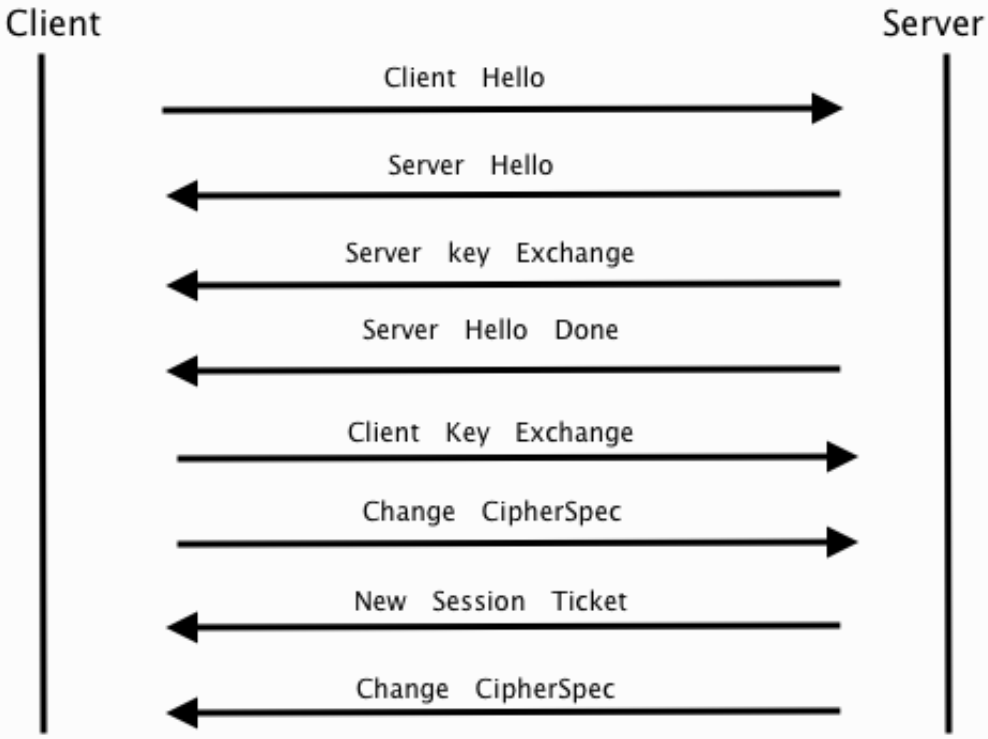
一、HTTPS协议

SSL握手交互的过程(通过wireshark分析):



（1）握手阶段暴露的侧信道信息(在tcp层面看包):

编号	意义	方向	大小(整个帧)
1	Client Hello	C—>S	400 ±
2	Server Hello+Certificate+Server key Exchange+Server Hello Done	C<—S	1494(tcp满载荷)
3	同上	C<—S	1494(tcp满载荷)
4	同上	C<—S	1494<
5	Client Key Exchange+Change CipherSpec	C—>S	192
6	New Session Ticket + Change CipherSpec	C<—S	300 ±

（2）侧信道暴露

direction	值
C—>S	0
C<—S	1

对应数字特征：

{ {"no":1,"driaction":0,"len":400 ± }

{"no":2,"driaction":1,"len":1494 }

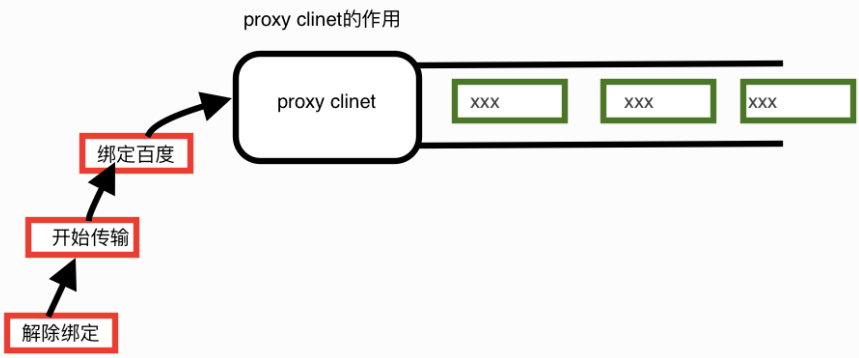
{"no":3,"driaction":1,"len":1494 }

{"no":4,"driaction":1,"len":1494< }

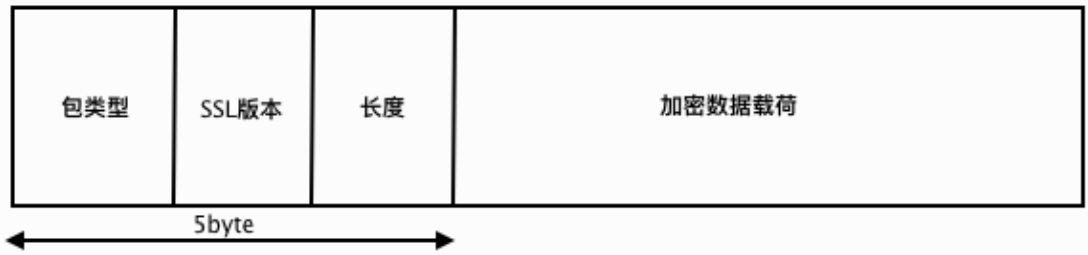
{"no":5,"driaction":0,"len":192 }

{"no":6,"driaction":1,"len":300 ± })

由于shadowsocks的流加密明文密文 1:1,所以同样的侧信道是可以在shadowsocks外被识别出来的：



包结构:



包类型:

16进制	10进制	类型
0x14	20	ChangeCipherSpec
0x15	21	Alert
0x16	22	Handshake
0x17	23	Application Date

其中0x17类型的是应用数据，其它的为控制型数据。

二、混淆算法与协议的设计

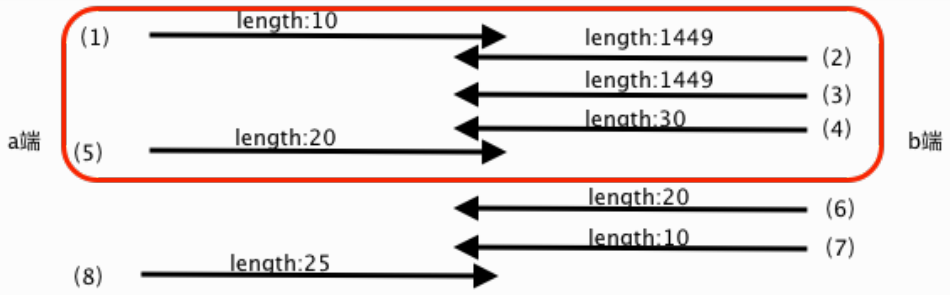
混淆算法

组建一个混淆流量的库，存储一些普通流量(即https进入数据交互时候的流量)，然后选取部分交互过程，把控制流量混淆进去。

选取混淆区域

假如client识别出client hello包，长度为27

且随机到流量库中有如下数据流量交互:



因为client hello的长度为27，选取a到b端多个请求长度大于27的(a至b 1、2号，10+20>27)

所以选取a至b方向的1、2,及b至a方向1、2、3、4方向的流量序列，把client hello 混入。

混淆流量库，应用流量的抽象：

(1)用wireshark收集大量的应用流量，且所收集的应用流量为https交互中，不含控制包的交互过程

(2)抽象为混淆需要的结构,每条数据三个参数

(3)代理服务器与代理客户端有相同的混淆流量库

例(上方选取的混淆区域的json格式)：

[{"no":1,"driaction":0,"len":10},

{"no":2,"driaction":1,"len":1449},

{"no":3,"driaction":1,"len":1449},

{"no":4,"driaction":1,"len":30},

{"no":5,"driaction":0,"len",20 },

{"no":6,"driaction":1,"len",20 })

(3)存为json文件，文件名是从1开始的整数（1.json、2.json...），供混淆算法随机选取文件

混淆协议的设计

```
control  header  7 bytes  =
        1 byte 0000  000[1/0] (first [1/0],1 => need response)
     +  2 byte mix byte start position
     +  2 byte mixpayload len (tcppayload + mixpayload)
     +  1 byte file num
     +  1 byte start  num
```

混淆协议的结构由控制位、真实要传输的数据、混淆数据组成的。

控制位：

值	长度(byte)	作用	对应位置
0/1	1	是否需要返回混淆包，0为不需要、1为需要	(0,1)
0~65535	2	混淆数据开始的位置	(1,3)
0~65535	2	包整体的长度	(3,5)
0~255	1	普通流量库的文件号	(5,6)
0~255	1	普通流量库的某文件号的第几号包	(6,7)

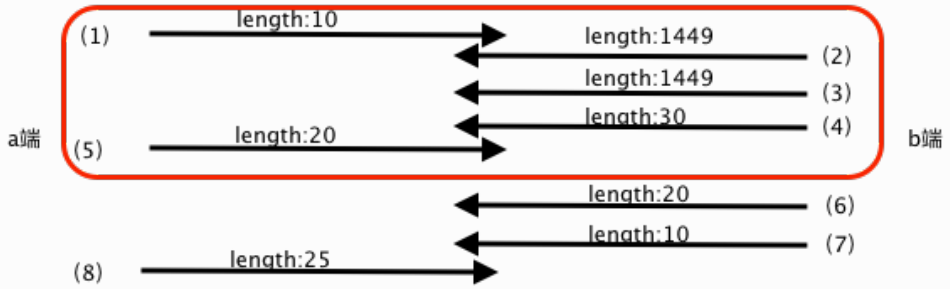
混淆协议与算法结合示例

通过三个部分：混淆算法、混淆流量库、混淆协议，三个的结合来实现对与https流量控制包侧信道流量的泄露的混淆

假如app向代理客户端发送了一个长度为27 byte的https控制包client hello

1.代理客户端接受client hello,通过密码学随机数从混淆流量数据库随机选取一个混淆特征文件4.json

2.通过计算选取以下部分(介绍混淆算法中有介绍如何计算)



3.代理客户端把client hello分为两部分，分别为长度为10、17的部分

长度为10的(1)号包对应头部表格

对应字段值(位置顺序从小到大)	作用
1	需要代理服务器返回纯混淆包
10	混淆填充位开始的位置
10	混淆包总长度
4	混淆库中对应的文件编号
1	该包模拟的为该文件中包序号

长度为17的(5)号包对应头部表格

对应字段值(位置顺序从小到大)	作用
0	不需要代理服务器返回纯混淆包
17	混淆填充位开始的位置
20	混淆包总长度
4	混淆库中对应的文件编号
5	该包模拟的为该文件中包序号

用以上头部构造包(1)、(5)号包

4.代理客户端发送(1)号包，且等待接受对端发出的纯混淆包 (2)、(3)、(4)

5.代理服务器收到(1)号包后

a.通过混淆协议头，确定需要混淆包的返回

b.定位对应4.json,(1)号包后对应的b端向a端发送的包(2)、(3)、(4)的长度依次为1449、1449、30

c.按照1449、1449、30构造混淆包

d.依次发送

e.向目标服务器发送（1）号包取出的真实数据

对应第一个1449纯混淆包的头部表格：

对应字段值(位置顺序从小到大)	作用
0	不需要代理客户端返回混淆包
0	混淆填充开始的位置
1449	混淆包的总长度
4	混淆库中对应的文件编号
2	该包模拟的为该文件中包序号

其它的构造形式类似

5.代理客户端依次接收纯混淆包(2)、(3)、(4)并向代理服务器发送(5)号包

6.代理客户端收到(5)号包后

a.通过混淆协议头，确定不需要混淆包的返回

b.向目标服务器发送（5）号包取出的真实数据

总结：

整个混淆长度为27的client hello的过程就完成了，整个过程中发送client hello的代理客户端扮演着导演的角色，他要求对端返回或不返回纯混淆的包，代理服务器收到代理客户端发送的包，通过混淆协议头，返回或不返回纯混淆的包。当https控制包是从代理服务器发向代理客户端的时候，导演的角色就给了代理代理服务器，混淆的过程相同。

三、代码实现技术点

在代理客户端、代理服务器实现接收buffer(缓冲区)，达到基于流协议的分包

（1）由于shadowsocks是基于tcp流协议，代理客户端要识别shadowsocks中https包的头来区分是否为控制包(除0x17数据包外)，所以要通过实现buffer在接收https，收集整个包后，通过头部的判断，将控制包通过混淆算法进行混淆。

（2）同样是因为基于tcp流协议，所以混淆协议也需要通过buffer来做到对混淆协议进行分包操作，进而对混淆协议进行 混淆协议的处理。

代理服务器是对称的作用，所以，在客户端与代理服务器都需要两个buffer(缓冲区)来达到对https的流量的分包以及对混淆协议的分包。

混淆库的生成

混淆库是由多个由不同编号(编号由1开始的整数)文件组成的应用流量特征json的数据库，生成特征库需要生成大量的应用流量来提取特征。

(1)收集https站点的列表

(2)通过python脚本实现对列表中https网站的访问，并且通过tshark软件抓区对应的https流量包(每个网站对应一个包)

(3)通过python脚本处理抓区的流量包，通过python的scapy模块解析tshark数据包，过滤掉tcp握手控制包(如ack不带数据)，对每一个数据包提取出对应的no（时序编号）、drication（方向）、len（长度）生成对应包的特征json文件。

收发协程并发控制

由于实验基于shadowsocks的go语言版本，且是通过go语言的协程来达到流量并行转发，在此基础上增加混淆协议后，需要协程之间的控制。

shadowsocks中每一个代理会话中有分别有两个tcp连接在代理客户端与代理服务器中。

代理客户端:(1)app与代理代理客户端tcp连接

(2)代理客户端与代理服务器的tcp连接

代理服务器:(1)代理客户端与代理服务器的tcp连接

(2)代理服务器与目标服务器的tcp连接

代理客户端中的转发伪代码:

```
connToApp = ReciveAppTcpConn(Local_ListenPort)
connToProxyServer = CreateConnToServer(ServerIP,ServerPort)
go PipeThenClose_to_SSpipe(connToApp,connToProxyServer)
go PipeThenClose_to_app(connToProxyServer,connToApp)


function PipeThenClose_to_SSpipe(src tcpconnect ,dst tcpconnect){
    for{
        receive = src.read( )
        dst.write(receive)
         }
}


function PipeThenClose_to_app(src tcpconnect ,dst tcpconnect){
    for{
        receive = src.read()
        dst.write(receive)
         }
}
```

代理服务器的实现与代理客户端的实现类似

对于代理客户端，通过这两个tcp连接，建立了两个流量通道，分别为从app到代理服务器，代理服务器到app的通道实现转发，为了不与go语言中的channl混淆，以下将app到代理服务器的通道、代理服务器到app的通道分别用用appToServer、serverToApp代替

对于上面转发与转发与混淆协议结合的示例中，当发出(1)号包后，appToServer需要知道的serverToApp收到的(2)、(3)、(4)号包才能发送(5)号包来达到模拟混淆的过程，所以协程之间需要通知机制。

做serverToApp 向 appToServer 的channel,当收到(2)、（3）、（4）号包后，serverToApp 的通道向channel内发送(2)、(3)、(4)号已经收到的信号量。当处于阻塞状态位于 appToServer中的channel接收到了(2)、（3）、（4）已经接收，然后发送(5)号包，保证时序、顺序、方向的模拟。