

# 安全的Sock5 Proxy

2016010752 姚振翮

2016011258 贾明麟

## 一、项目说明

本项目是一个提供sock5代理的server端和local端代理程序

### 1.1 代码说明

- 文件说明
  - utils中为测试的 client脚本、通用基类和生成公私钥代码
  - server文件夹下为服务器端代码
  - client文件夹下为本地端代码

项目源码在[我的个人仓库](#)

### 1.2 使用说明

- 配置
  - 安装相关依赖
  - server端配置好 `server_config.json`
    - 自身的启动地址
    - 自身的启动端口
    - whitelist (针对local所在地址)
    - local的公钥文件地址
    - 自身的私钥文件地址
  - local端配置好 `local_config.json`
    - 自身的启动地址
    - 自身的启动端口
    - 用户账号 (针对sock5协议)
    - 用户密码 (针对sock5协议)
    - 是否使用账号密码校验 (针对sock5协议)
    - whitelist (针对浏览器所在地址)
    - server地址
    - server端口
    - server的公钥文件地址
    - 自身的私钥文件地址
- 启动
  - python3 分别在服务器和本地启动 `server.py` 与 `local.py`
  - 配置浏览器使用sock5协议，目的端口和地址为local服务器的端口和地址

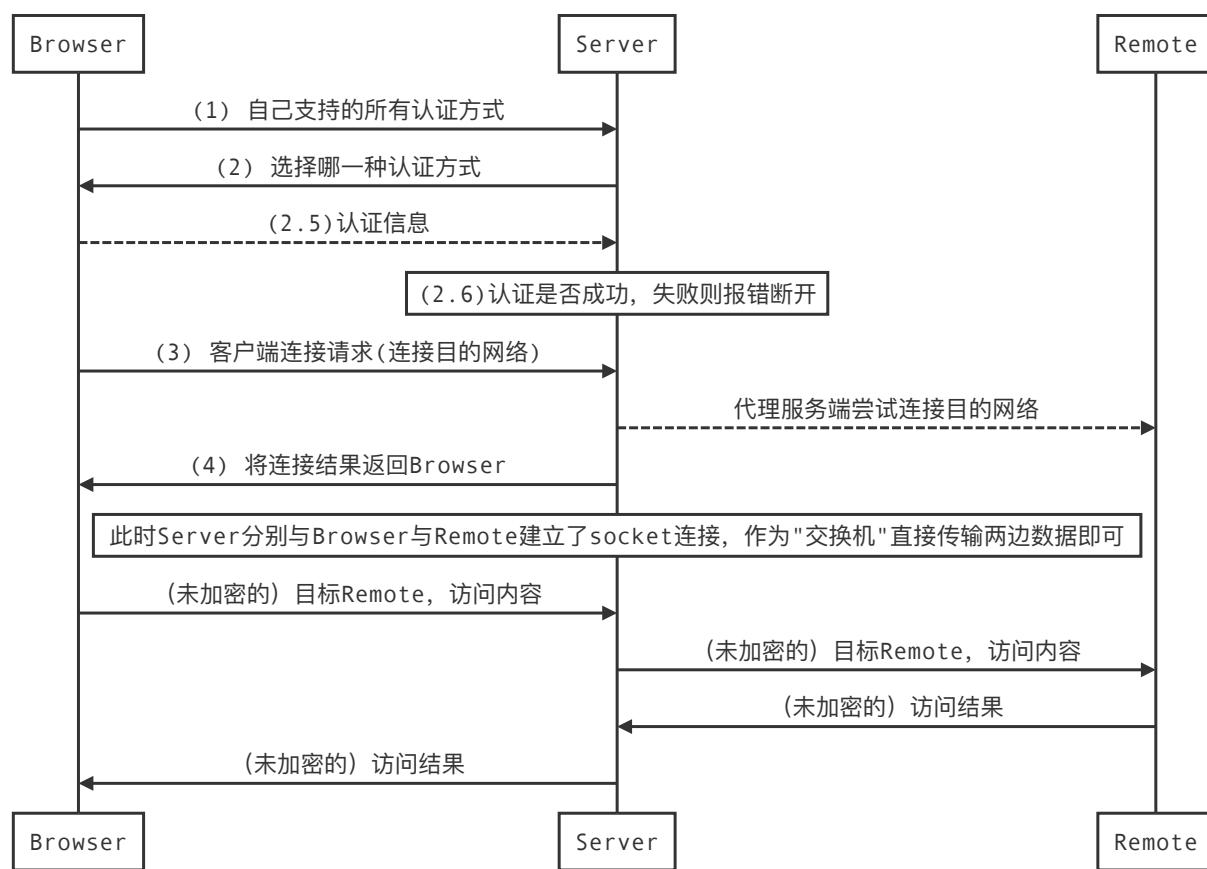
- 此后就可以正常代理上网了

## 二、设计说明

### 2.1 通信流程

#### 2.1.1 sock5 通信建立流程

一般sock5通信建立流程如下



#### 2.1.2 本项目整体设计

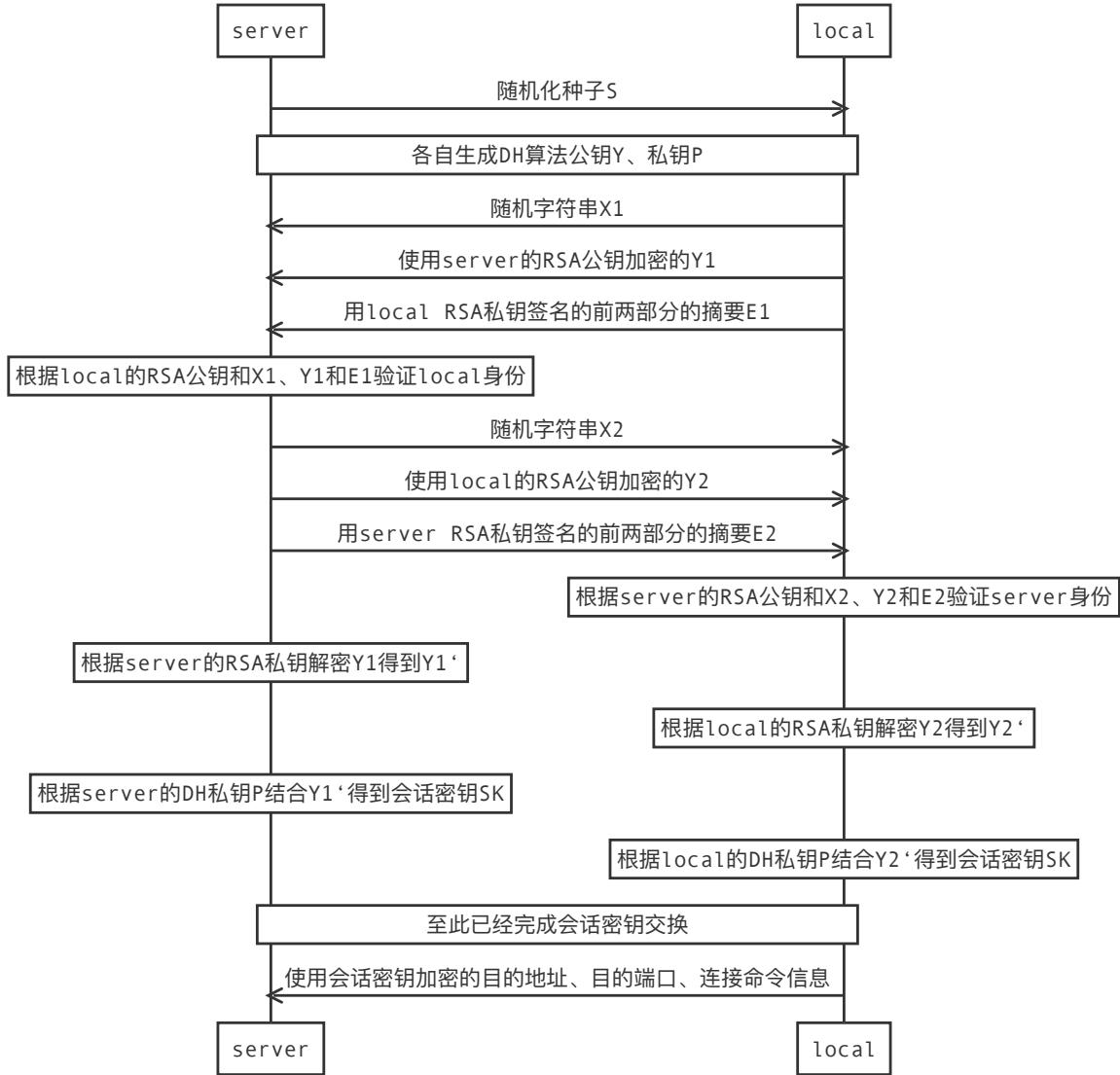
本项目将上述sock5通信中的Server功能分为远程服务器（下称server）和本地服务器（下称local），并在local与server之间设立安全通信信道



## 2.2 细节说明

## 2.2.1 local与server间通信握手

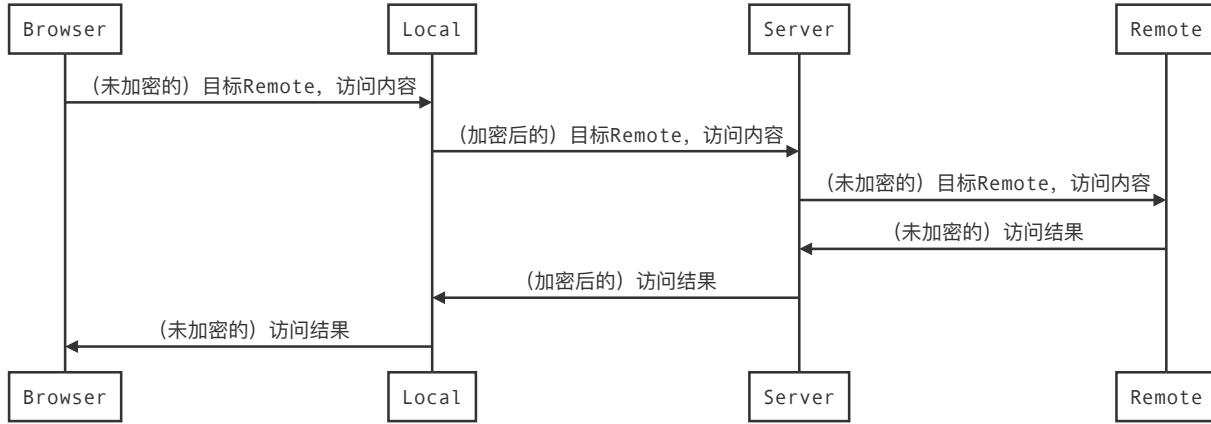
- 在local与server通信建立的初始阶段，local与server仅有各自的公钥，由于公钥算法速度不如预期，一般仅使用公钥用于会话密钥传输，本项目也是如此设计。
  - 握手阶段，local与server主要完成以下任务：
    - 使用RSA算法和DH算法实现密钥交换
    - 传输local从sock5协议交互中从浏览器接收的目的地址、目的端口、命令标识符（此时使用会话密钥加密）
  - 具体的握手交互流程如下



- 完成握手后，server与local间一切传输均使用会话密钥加密，此后不再重复说明"使用会话密钥加密/解密"
- 完成握手后，server将根据最后传输的目的地址、目的端口、连接命令等控制信息，尝试对目的地址端口进行连接
  - 注：此处如果连接命令不是0x01 (connect)，则将报错包返回给local端，由local端解密后返回给浏览器

## 2.2.2 local与server间的数据组织和交换

其主要的数据流如下



### 2.2.2.1 数据包格式协议

其中发生在local和server之间的数据传输遵循以下的协议

- 数据字节流按照64字节填充重组为固定包后使用加密算法后进行传输
- 数据传输以64字节为一个单元
  - 含有4字节包头
  - 含有10字节随机字符
  - 含50字节payload实际数据
- 64字节数据传输包按照如下方式组织

字节数	1	1	1	1	offset-4	50或更少	64-length-offset
单元含义	offset	length	version	res	Padding1	payload	Padding2
说明	payload起始位置	payload长度	版本号, 固定为1	保留	随机填充字符	实际负载数据	随机填充字符

### 2.2.2.2 数据包传输流程描述

- server从Remote收到数据
  - 将数据按照50字节分组
  - 每个分组随机生成位于[ 4 , 64 - 分组长度 ]之间的整数offset, 按照如上方法生成64字节包
  - 每个64字节包使用AES算法加密
  - 将加密后数据传送给local
- local从server收到数据
  - 等待收齐64字节
  - 将每个64字节分组使用AES算法解密
  - 根据解密后各个包信息offset和length和提取出payload
  - 将payload字节传送给浏览器 (Browser)
- local从Browser收到数据
  - 将数据按照50字节分组
  - 每个分组随机生成位于[4,64-length]之间的整数offset, 按照如上方法生成64字节包
  - 每个64字节包使用AES算法加密

- 将加密后数据传输给server
- server从local收到数据
  - 等待收齐64字节
  - 将每个64字节分组使用AES算法解密
  - 根据解密后各个包信息offset和length和提取出payload
  - 将payload字节传输给目的主机 (Remote)

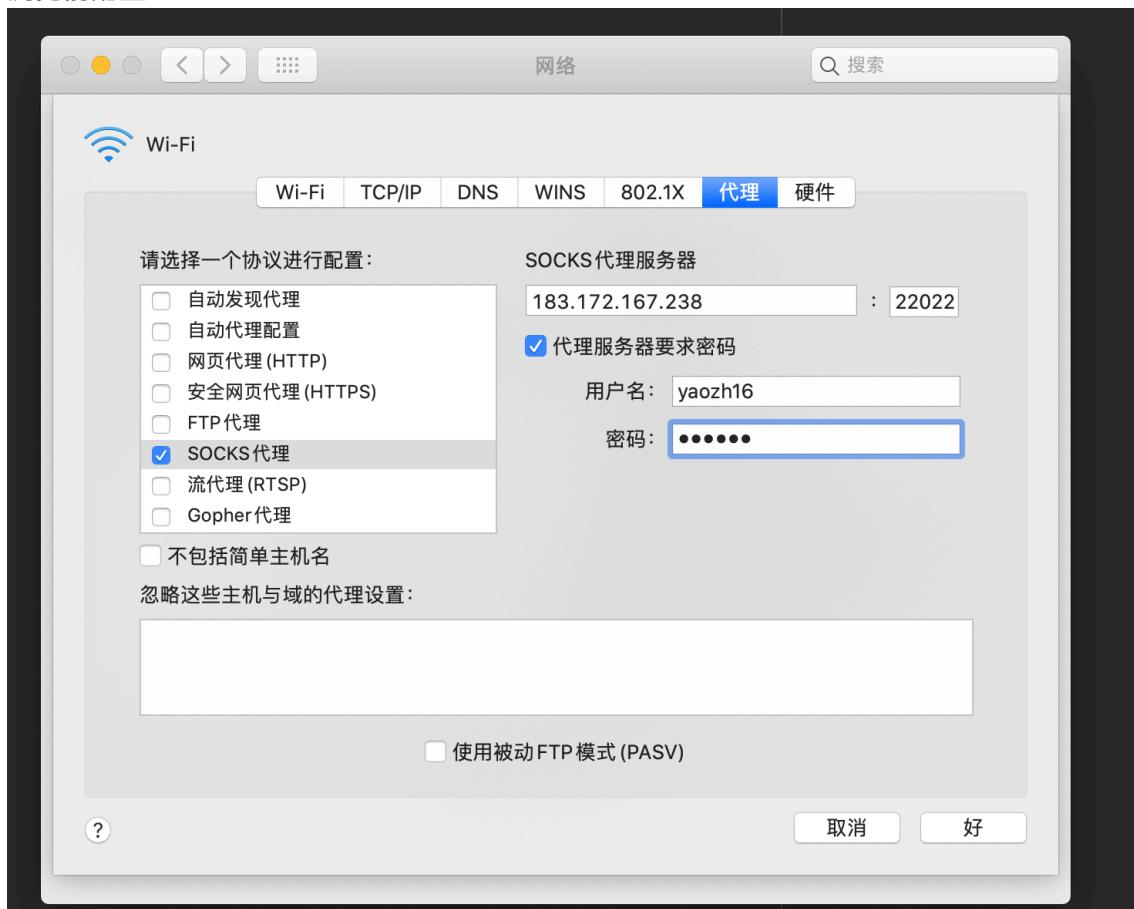
### 2.2.2.3 对数据进行重组的原因

- 在网络传输中，由于不知道传输数据的具体长度，而AES加密算法又要求输入数据为某些固定长度倍数的数据，我们只能对不足长度数据进行填充，需要有一些原始长度信息与补充信息需要保存在数据包中
- 另一方面为了保证加密解密阶段的数据一致性，只能使用本次通信中之前local与server握手获得的会话密钥。为了保障即使同样的payload也有不同的加密结果，防止被通过某些统计分析方式破解，因此才设计了在payload前和payload后的两段随机字符

## 三、项目使用示例

- 配置完成后，按照本说明“一”中配置启动

- 浏览器前配置



- 浏览器访问代理正常上网

- 原始IP

- 代理后