### 电表

#### Smi\_register.py 电表请求注册

电表注册过程：生成高熵随机数r1，发送{IDi, r1}给服务提供商，接收服务提供商返回的Mi。

存{IDi, r1, Mi}到防篡改设备（本地文件meter\_data.json）

#### smi\_sksm.py 电表身份认证-密钥协商过程

生成会话共享密钥SKsm。

run\_authentication(sock, SMi\_ip, SPj\_ip)：

调用main(sock, SMi\_ip, SPj\_ip)执行认证-密钥协商过程，生成会话共享密钥SK。

返回SK，以及认证过程总时间。

#### Test\_SK\_derive\_time.py（SM目录下） 测试认证-密钥派生过程时间

电表执行n次认证-密钥协商-密钥派生。计算总时间，输出每次认证-密钥派生的平均执行时间

#### SMI.py 电表与服务提供商建立连接，发送接收消息。

调用smi\_sksm.run\_authentication(sock, SMi\_ip ,SPj\_ip)执行身份认证和密钥派生过程。

调用derive\_key.HMAC\_keys(SK, 'SM') 执行密钥派生，生成发送和接收密钥。

调用test\_send.test\_send(sock, sport, SPj\_ip, ks, ksend, SK)向服务提供商发送消息接收响应。

启动线程threading.Thread(target=receive.listen\_for\_messages, args=(sockrev, rport, SK, kr, krev))，监听来自服务提供商的消息并发送响应。

电表发送消息到服务提供商的50001端口，并在50001端口监听响应。

电表在端口50002监听来自服务提供商的消息，并发送响应到服务提供商的50002端口。

### 服务提供商

#### spj\_register.py 服务提供商处理注册请求

服务提供商处理注册请求。计算Mi，Qi。发送Mi给电表。存储Qi到动态验证表（本地文件dynamic\_validation\_table.csv） 。

#### spj\_sksp.py 服务提供商的认证-密钥协商过程

生成会话共享密钥SKsp。

run\_authentication(sock, IDj, s, SPj\_ip):

调用main(sock, IDj, s, SPj\_ip)执行认证-密钥协商过程。

返回共享会话密钥SKsp、认证-密钥协商过程总时间、请求通信的电表ip(SMi\_ip)

#### Test\_SK\_derive\_time.py （SP目录下）测试认证-密钥派生过程时间

服务提供商执行n次认证-密钥协商-密钥派生。计算总时间，输出每次认证-密钥派生的平均执行时间。

#### SPJ.py 服务提供商与电表建立连接，发送接收消息

调用spj\_sksp.run\_authentication(sockrev, IDj, s, SPj\_ip)执行身份认证和密钥派生过程。

调用derive\_key.HMAC\_keys(SK, 'SP') 执行密钥派生，生成发送和接收密钥。

调用test\_send.test\_send(socksend, sport, SMi\_ip, ks, ksend, SK)向电表发送消息接收响应。

启动线程threading.Thread(target=receive.listen\_for\_messages, args=(sockrev, rport, SK, kr, krev))，监听来自电表的消息并发送响应。

服务提供商发送消息到电表的50002端口，并在50002端口监听响应。

服务提供商在端口50001监听来自服务提供商的消息，并发送响应到电表的50001端口。

### Derive\_key.py 密钥派生

1. HKDF\_keys(SK , W)

2. HMAC\_keys(SK, W)

3. KDF\_keys(SK, W)

接受参数SK,W

W=‘SP’表示服务提供商（ks=HMAC(SK, “receive”)、 kr = derive\_key\_hkdf(SK, "send")）

W=‘SM’表示电表（ks=HMAC(SK, “send”)、 kr = derive\_key\_hkdf(SK, " receive ")）

### 同步通信：

#### Test\_Send.py 生成测试消息并发送

生成n条测试消息。

生成测试消息。调用send\_message.communication\_flow (sock, port, rev\_ip, ks, k, SK, msg)依次发送消息。每次发送消息后需要等待响应，响应成功才继续发送下一消息。

所有消息发送并收到响应后输出本次发送消息数以及平均每条消息发送时间。

#### Send\_message.py 发送消息等待接收响应

**communication\_flow(sock, port, rev\_ip, ks: bytes, k: bytes, SK: bytes, message: str)：**

调用send\_data(sock, port, rev\_ip, ks, k, SK, message) 发送消息、更新密钥。

等待响应。响应成功则返回新的发送密钥ks和辅助密钥k。

超过10s没有收到响应则调用retransmit\_message(sock, port, rev\_ip, SK, Ci)重传消息。

重传超过5次尝试重置密钥reset\_keys(sock, port, rev\_ip, SK)。

密钥重置成功则用新密钥重新加密并发送消息

密钥重置失败返回0，0，结束通信

**send\_data(sock, port, rev\_ip, ks, k, SK, message)：**

负责计算C,Tag,ti，并发送消息。更新ks, k。

**receive\_response(sock, rev\_ip)：**

负责监听普通消息的响应，有效相应返回True，无效继续监听直到超时。

**reset\_keys(sock, port, rev\_ip, SK)重置密钥：**

随机生成32字节高熵随机数x，发送给接收方。

新的发送密钥：ks = HMAC(SK, x)、辅助密钥：k = H(ks || SK)。

等待接收密钥重置的响应（H(kr)）。收到响应后验证哈希值，哈希值一致则密钥重置成功。返回新的ks,k。

超时未收到有效响应则重传密钥重置消息，重传超过10次认为通信异常返回0，0。（结束通信）

**receive\_resetkey(sock, rev\_ip, ks\_reset)：**

监听密钥重置的响应，收到有效响应后验证发送密钥与对方接收密钥哈希值是否一致，一致返回True，否则继续监听直到超时。

#### receive.py 接收消息，返回响应

**listen\_for\_messages(sock, port, SK, kr, k)：**

负责接收消息并发送响应。

如果收到{“reset\_key\_x”: … }表示密钥重置请求，提取x并重置接收密钥kr和辅助密钥k。

kr = HMAC(SK, x)。k = H(kr || SK)。

如果收到{“Ci”: …, “Tag”: … , “ti”: …} 表示一般消息。

验证时间戳有效性-验证标签tag-判断是否重传-存数据包标识符-解密消息-更新密钥kr和辅助密钥k-发送响应{"ack": … }

超过两分钟没有收到消息则输出本次处理的消息数以及平均时间

### 异步通信：

#### Test\_Send.py 生成测试消息并发送

**key\_reset\_event：**密钥重置完成的信号。

**stop\_sending\_event ：**密钥开始重置的信号（暂停发送消息接收响应）。

**stop\_event ：**终止通信、退出线程的信号。

**管理器MessageManager中：**

**unconfirmed\_messages = {}**记录未发送以及发送未收到响应的消息。

记录发送的第j（发送序号）条消息对应的密文、发送时间、重传次数。如果消息j超时没有收到响应，重传消息。重传次数超过5次，重置密钥。

**pending\_messages = set()**记录已发送未响应的消息的原始序号i。

在密钥重置后，这些消息需要重新加密发送。发送消息时的时候使用发送序号j，j从1开始重新编码。

**映射表 message\_map**：建立发送消息的发送序号j和消息的原始序号i之间的映射。

**主线程：test\_send(sock, port, rev\_ip, ks, k, SK)：**

生成n条测试消息 "Test message {i}: {os.urandom(16).hex()}

创建线程listener\_thread负责监听响应。

创建线程checker\_thread负责重发消息、重置密钥。

依次发送测试消息Test message {i}，调用send(sock, port, rev\_ip, ks, k, SK, msg, message\_id, message\_manager)发送{message\_id; Ci；Tag； ti}，message\_id是发送序号j。（不需要等待响应再发下一条）。Send发送消息后，在unconfirmed\_messages添加相关记录。

每发送一条，记录发送序号message\_id和原始序号i的映射。

消息发送过程中，如果信号stop\_sending\_event被设置，说明密钥正在重置，退出循环，停止发送消息。

如果密钥重置失败，则设置信号stop\_event，通知其它线程退出，终止通信。

如果密钥重置成功，则清除映射表message\_map，重新发送集合pending\_messages中的消息。

所有消息发送完成后，如果集合pending\_messages为空，说明所有消息发送完成且收到响应。输出发送n条消息的总时间以及平均每条消息发送时间，设置信号stop\_event，通知其它线程退出，结束通信。

**负责监听响应的线程listen\_for\_responses(sock, rev\_ip, message\_manager)：**

调用send\_message.receive\_response(sock, rev\_ip)监听响应，收到响应后，提取对应的消息发送序号j，在字典unconfirmed\_messages中删除该消息对应的数据。通过映射表message\_map找到消息的原始序号，在集合pending\_messages中删除该原始序号，表示这条消息已经发送并收到响应。

如果信号stop\_sending\_event被设置，表示需要重置密钥，暂停监听。

如果信号stop\_event被设置，表示通信停止，关闭线程。

**负责重发消息，密钥重置的线程check\_unconfirmed\_messages(sock, port, rev\_ip, message\_manager, SK)：**

每5秒检查一次字典unconfirmed\_messages，如果有消息超过5秒没有收到响应，则调用send\_message.retransmit\_message重传消息。

一旦出现某条消息重传次数超过5次，设置信号stop\_sending\_event，停止发送消息，监听响应暂停。调用send\_message.reset\_keys(sock, port, rev\_ip, SK)重置密钥。

密钥重置结束后，清空字典unconfirmed\_messages; 设置信号key\_reset\_event，表示密钥重置完成。

密钥重置失败则结束该线程。

密钥重置成功，清除信号stop\_sending\_event。

#### Send\_message.py 负责发送各类消息

**send\_data(sock, port, rev\_ip, ks, k, SK, message)：**

负责计算C,Tag,ti，并发送消息{'j':message\_id ,'Ci': Ci.decode('utf-8'), 'Tag': Tag.hex(), 'ti': ti}。更新ks, k

**retransmit\_message(sock, port, rev\_ip, SK: bytes, Ci: bytes, message\_id: int)：**

负责重新计算tag, ti，重发消息

**reset\_keys(sock, port, rev\_ip, SK)重置密钥：**

随机生成32字节高熵随机数x，发送给接收方。

新的发送密钥：ks = HMAC(SK, x)、辅助密钥：k = H(ks || SK)。

等待接收密钥重置的响应（H(kr)）。收到响应后验证哈希值，哈希值一致则密钥重置成功。返回新的ks,k。

超时未收到响应则重传密钥重置消息，重传超过10次认为通信异常返回0，0。（结束通信）

**receive\_resetkey(sock, rev\_ip, ks\_reset)：**

负责接收密钥重置的响应。收到响应{ "reset\_key": H(kr) }后，验证ks的哈希值是否等于收到的H(kr)。如果不相等则忽略该响应，继续监听。收到有效响应返回True，超时返回false

**receive\_response(sock, rev\_ip)：**

负责接收一般消息的响应，如果响应为{“acki”: …}，返回响应。

#### receive.py 接收消息，返回响应

集合**received\_identifiers ：**负责记录接收消息的发送序号j

字典**key\_store：**负责记录消息j对应的接收密钥kr\_j

**j\_max：**记录用当前的接收密钥kr解密的消息j

**listen\_for\_messages(sock, port, SK, kr, k)：**

负责接收消息并发送响应。

如果收到{reset\_key\_x: … }表示密钥重置请求，提取x并重置接收密钥kr和辅助密钥k。

kr = HMAC(SK, x)。k = H(kr || SK)。清空存储的密钥key\_store，重置消息编号j\_max，清空接收消息的记录received\_identifiers。

如果收到{“j”: …“Ci”: …, “Tag”: … , “ti”: …} 表示一般消息。

验证时间戳有效性-验证标签tag-判断是否为重复数据包。

重复则直接发送响应{“ackj”: j}。

不是重传数据包：

如果id<j\_max，说明密钥已经存在，直接从key\_store取出，解密，发送响应。

如果id=j\_max，说明该消息的接收密钥为当前密钥kr，解密，发送响应，更新密钥。

如果id>j\_max，说明后面的消息先到，计算解密该消息需要更新密钥的次数。

如果存储密钥的空间不足（key\_store），不更新密钥，忽略该消息。

如果空间足够，则更新密钥，存储更新过程生成的接收密钥kr，直到更新到当前消息对应的解密密钥kr。用该密钥解密当前消息j。发送响应。更新密钥kr,k。

超过两分钟没有收到消息则输出本次处理的消息数以及处理每条消息的平均时间