1

1.1

The file ag01598_6644818_1_1.spdl contains the base model of protocolΠ.

I choose the names of the roles based on their functions since it would make the file more readable, so R is Phone, S is Application, N is Network.

As the diagram shows, the first message is sent from the phone to the network to request the generation of a new session key.

The keys were modelled using a custom usertype called "SessionKey" and the time to live has modelled using a custom usertype called "Timestamp"

The network then answers to the Phone and the Application the keys and the time to live and the hashed value of that using a hash function named "Mac".

The Phone and the Application verify the Mac and then the phone sends a nonce to the phone and the phone answers back with a new nonce and the original nonce.

The Mac was modelled as a hashing function, then encryption. This was done this way because Scyther does not have a way of creating a mac function with keys, so the hashing is done first followed by the encrypted so that an attacker cannot modify it.

Scyther does not have a way to model the refresh/time to live parts, so that was not modelled.

1.2

The file ag01598_6644818_1_2.spdl contains the base model of protocolΠ and the claims.

I added non-injective synchronization(nisynch) to the Phone and Network, at least, some roles communicated as described by the protocol. I added a secret claim to SesK (Session key) to all roles, as the session key should be private. Furthermore, I added Commit and Running claims between some roles to check for agreement between some variables:

- Agreement between Phone and Network over the time to live and the session key
- Agreement between Application and Network over the time to live and the session key
- Agreement between Application and Phone over the message and the message m

These claims were chosen because they check agreement on message m between the Application Function and the Phone; The secrecy of SesK; And the synchronization and agreement between the Application Function, Phone, and Network.

There are 9 overall claims, where only three do not fail. The secrecy of SesK from the perspective of the Network. And agreement over the SesK and the time to live between the Phone and the Network, and the Application and the Network.

The protocol as it stands does not guarantee secrecy and agreement.

1.3

The file ag01598_6644818_1_3.spdl contains the fixed version of protocolΠ.

The first change was to require the refresh keys request was to require the application to send a nonce. This nonce is then sent back to the application to verify that the key was generated, was requested by the application and not by the attacker.

The second change was to make the network send the identity of the other party to the party that is receiving the message. i.e. Sending the identity of the Phone to the Application encrypted with the key Network, Application. This is done to guarantee that the Party receiving the communication is using a key that was intended for this communication.

1.4

The original protocolΠ is not an appropriate solution to the third-party problem as it cannot guarantee the secrecy of the session key, and since an attacker can obtain the session key, protocolΠ is not an appropriate solution to the presented problem. As Scyther, showed that there are attacks on the Dolev-Yao model. There are also some attacks outside the Dolev-Yao model.

For example:

With the assumptions, that:

- Dolev-Yao attacker.
- Strong cryptographic primitives
- The time to live is implemented as a counter, and not as an end date timestamp.
- One of the previous keys, where the encrypted version of the key and timestamp were recorded and leaked.

In this scenario, an attacker can record the messages where the key and the time to live were encrypted. When the key gets leaked, the attacker can perform a replay attack.

The attacker resends the previously recorded keys to the application server and the phone because the time to live is based on a counter and not on a timestamp, the phone, and the server accept the "new" key. And since the attacker already knows this key, the protocol failed to guarantee, the secrecy of the session key.

But it can be improved, as we saw in the answers for the question 1.3 by modifying the protocol slightly we can achieve secrecy of the session key in the Dolev-Yao model. Although it does not resolve issues related with the time to live if the protocol was implemented with counters for time to live instead of timestamp.

1.5

The file ag01598_6644818_1_5.spdl contains a more communication efficient version of the protocolΠ.

This version trades off computational power for communication efficiency. This version creates bigger encrypted messages, and as a trade-off reduces the number of messages that are sent.

The message that is removed is the message where the network sends the key to the phone. This data in this message is still sent, but it's sent when the application sends the m message.

And the data is sent to the Application inside the encrypted packed that is already sent to the Application when the phone receives the keys from the network.

It sends the session key that was sent by the network, to the phone, with the message m.

2

2.1

Using a system like GPG, you can generate keys by running the command

```
gpg --gen-key
```

.

2.2

There are multiple ways of securely exchanging the keys. For example, if meeting in person was a possibility, the keys cloud be put onto USB drives and the drives exchanged in person. And this would guarantee 100% authentication, but meeting in person could not be feasible.

If meeting in person is not feasible, an alterative method would be sending the public key via email, and then calling the colleague on the phone. Both parties would then hash key using for example

```
sha512sum key
```

and both parties would read out their public keys hash to each other. Since the parties know each other, and would recognize the voice, this would be a feasible way to exchange the public keys. With this, they could verify that the key came from the correct person.

2.3

Assuming that the peers were able to exchange public keys.

One of the Peer A would generate a random 256-bit key.

Peer A would then encrypt the key using Peer B's public key, and sign it using its own private key.

Peer A would then send the signed and encrypted symmetric key to Peer B in an insecure channel.

Peer B upon receiving the encrypted and signed symmetric key, would verify the signature, and decrypt the message and obtain the key.

After this, both A and B can communicate using the symmetric key.

Example:

Peer A:

```
head -c32 /dev/random > aes256key

gpg --output key.gpg --encrypt --recipient b@example.com aes256key

gpg --output key.gpg.sig --sign key.gpg
```

send key.gpg.sig to peer b

Peer B:

```
gpg --output key.gpg.sig --decrypt key.gpg

gpg --output key.gpg --decrypt key
```

Now both Peer A and Peer B have the same key and can start communicating between each other.

```
# Encrypting
openssl aes-256-cbc -in message.file -out message.file.enc -iter 10000 -kfile aes256key

# Decrypting
openssl aes-256-cbc -in message.file.enc -out message.file -d -iter 10000 -kfile aes256key
```

2.4

A secure communication channel for the purposes of this answer is a communication channel that can maintain the secrecy, and integrity.

A communication channel can be achieved by trading 2 symmetric keys.

One of the keys would be used to encrypt the messages and the other would be used to MAC the messages. This would guarantee that the messages are secret because of the encryption and that they have integrity because of the MAC.

The 2 different keys guarantee that the MAC that is generated is significantly different from the encrypted message.

In this system, messages should also contain the timestamp of when the messages were sent. This is useful to maintain freshness, it's also useful to provide different messages every time.

2.5

The system would work under a computational system, as the cryptographic primitives that were selected are computationally challenging to break.

The system would also work with a man in the middle attacker, as the attacker cannot change the messages without one of the people feeling suspicious and calling for a restart of the system.

This system would not work if the method for exchanging keys was via phone, and the attacker had a method of replicating the voice of one of the participants of the system, i.e. using an AI.

The system can maintain integrality, in the public key exchange phase with the phone call and the hash, in the symmetric key exchange phase with the signatures, and in the message exchange phase with the MAC.

The system can maintain secrecy throughout the entire process by using strong cryptographic functions, and having all messages that are sent that contain sensitive information encrypted.

2.6

An alterative solution to the problem is if both parties have a third party that they both know and trust. This system would have less communication and computational overhead, as the public key exchange could have been skipped and the symmetric key could have been exchanged from the begging.

This solution would have a less level of security as the system relies on the trust of a third party, this includes more levels of failures as the third party could get corrupted, making the system less secure.