Protocol Implementation

0.1 Application Instruction

To run this protocol in a practical scenario, at least 3 different devices are required - one be the Alice, one be the Bob and one be the Courier. This application implements all 3 entities within it, so once the device has this application, it can be any entity in the protocol. To achieve this, at the start of the application, user is asked to choose a particular role for the device to be. There are 4 options:

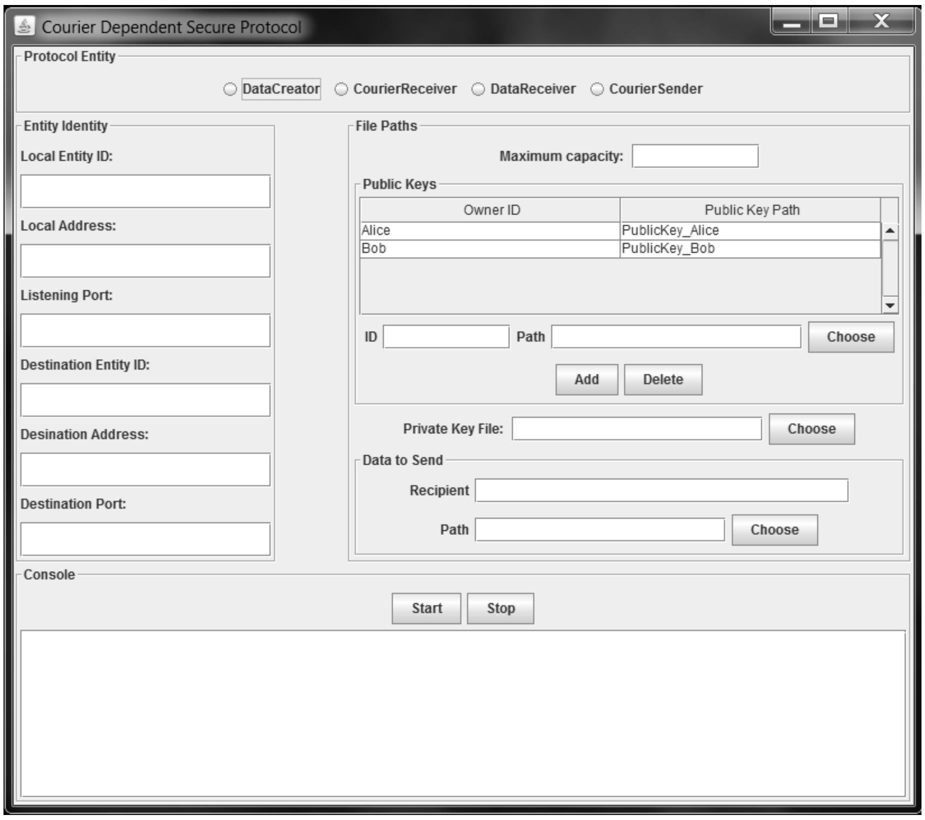
1. **DataCreator**, who creates message and wants to send it out.
2. **CourierReciever**, who connects to DataCreator and receives the message from it.
3. **DataReceiver**, who is the recipient of DataCreator.
4. **CourierSender**, who possesses the message from DataCreator and should transmit it to DataReceiver.

Once the choice has been made, the graphical user interface will adjust to the selected role.

To run Submit Protocol, Alice should choose to run as DataCreator and Courier should choose to run as CourierReceiver. Then Courier transports to Bob and chooses to run as CourierSender, meanwhile Bob should choose to run as DataReceiver to run Transmit Protocol with Courier. The detailed instruction of how to run those two sub-protocols will be explained in latter parts.

Below is a snapshot of the GUI of the application, it shows that the whole GUI consists of 4 main panels:

- **Protocol Entity**
  It is on the very top. It provides 4 roles (as mentioned above) to be chosen for the user. Because every device runs this protocol must be one of those four roles, it is essential that user specifies a role in this panel before start running the protocol.
- **Entity Identity**
  It locates on the left. Here user specifies the information about the host device and the destination device, such like IDs, and IP addresses.
- **File Paths**
  It locates on the right. Here user manages all the disk files that are related to the running of the protocol. Presumably, many important information like public keys, private keys and messages are all files in the device's disk, user has to specify those files' paths before running the protocol.
- **Console**
  It is at the bottom. Here provides buttons for user to start and stop the protocol, and it displays information about the running protocol in a information board.



Owner ID	Public Key Path
Alice	PublicKey_Alice
Bob	PublicKey_Bob

Figure 1: GUI Overview

0.1.1 Run Submit Protocol

It requires at least 2 devices to run this protocol, we call them D_0 and D_1. Assume D_0 is entity Alice, who possesses a message and wants to send it to Bob. Assume D_1 is entity Courier, who is able to carry the message and transmit it to Bob. To run the protocol successfully, it should be done by the following procedure:

1. D_0 starts the application.
2. D_0 user selects “DataCreator” in the Protocol Entity panel.
3. D_0 user specifies the entity ID, local address and listening port (9888 by default) by filling the corresponding textfields in Entity Identity panel.
4. D_0 user adds destination entity’s public key into its public key list.
   There are two ways to add an new entry to the public key repository:
   (a) Adding through application: input the ID of the public key holder and the path of public key file by filling the textfields in the Public Key panel. Then click Add button. The (ID, path) pair will then appear in the public key list. This effect is temporary, the pair will disappear in the next launch.

   (b) Adding through configuration file: by default, when application is launched, it will import all (ID, path) pairs from a file named PublicKeys in the current directory. User can appending a new line to PublicKeys file to add a new pair. The format is ID;Path, e.g. "Alice;/publickeys/alice.pk". This effect will take place since next launch of the application.

5. D_0 user specifies its private key by filling the textfield with the private key file.

6. D_0 user specifies recipient entity's ID by filling the textfield in the Data to Send panel.

7. D_0 user specifies the file that contains the message needs to be sent by filling the file path in the corresponding textfield.

8. D_0 user click Start button in the Console panel. The running progress of the protocol will be displayed in the information board in the Console panel.

9. D_1 starts the application.

10. D_1 user selects "CourierReceiver" in the Protocol Entity panel.

11. D_1 user specifies its own entity ID by filling the textfield in Entitiy Identity panel.

12. D_1 user specifies the ID of the entity he wants to contact (here D_0's ID), together with its address and port number by filling corresponding textfields in the rest of Entity Identity panel.

13. D_1 user add the public key of the entity he wants to contact into the public key list. The way of doing that has been introduced in step 4.

14. D_1 user click the Start button in the Console panel. The running progress of the protocol will be displayed in the information board in the Console panel.

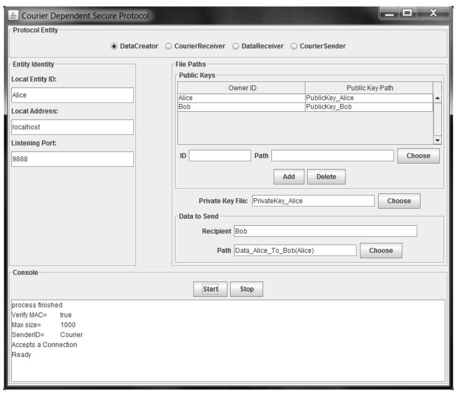
Below figures show snapshot of two applications successfully running Submit Protocol. After application consoles on both devices pops "process finish" successfully, the message of D_0 has been submitted to the D_1, and is stored in the file named by the message recipient's ID. If D_1 gathers messages from multiple devices, all messages to the same recipient will be accumulated in the same file.

0.1.2 Run Transmit Protocol

It requires at least 2 devices to run this protocol, we call them D_1 and D_2. Assume D_1 is entity Courier, who possesses an encrypted message from Alice and wants to send it to Bob. Assume D_2 is entity Bob, who waits to receive message from Alice. The running procedure of Transmit Protocol is:

1. D_2 starts the application.

2. D_2 user selects "DataReceiver" in the Protocol Entity panel.



Protocol Entity: [✓ DataCreator] CourierReceiver DataReceiver CourierSender

Entity Identity

Local Entity ID: Alice

Local Address: localhost

Listening Port: 9888

Owner ID	Public Key Path
Alice	PublicKey_Alice
Bob	PublicKey_Bob

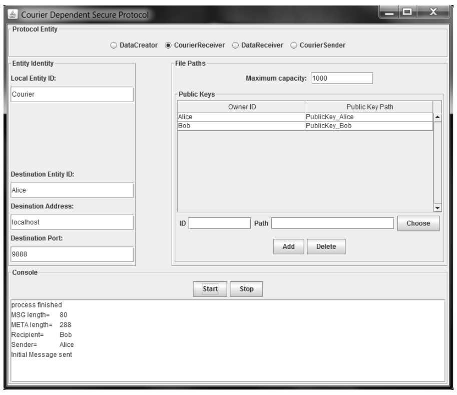
Private Key File: PrivateKey_Alice

Recipient: Bob

Path: Data_Alice_To_Bob(Alice)

```
process finished
Verify MAC=     true
Max size=       1000
SenderID=       Courier
Accepts a Connection
Ready
```

(a) DataCreator



Protocol Entity: DataCreator [✓ CourierReceiver] DataReceiver CourierSender

Entity Identity

Local Entity ID: Courier

Destination Entity ID: Alice

Desination Address: localhost

Destination Port: 9888

Maximum capacity: 1000

Owner ID	Public Key Path
Alice	PublicKey_Alice
Bob	PublicKey_Bob

```
process finished
MSG length=     80
META length=    288
Recipient=      Bob
Sender=         Alice
Initial Message sent
```

(b) CourierReceiver

Figure 2: Running Submit Protocol

3. D_2 user specifies the entity ID, local address and listening port (8888 by default) by filling the corresponding textfields in Entity Identity panel.

4. D_2 user adds destination entity's public key into its public key list. The way of doing that has been introduced in the step 4 of running Submit Protocol.

5. D_2 user specifies its private key by filling the textfield with the private key file.

6. D_2 user click Start button in the Console panel. The running progress of the protocol will be displayed in the information board in the Console panel.

7. D_1 starts the application.

8. D_1 user selects "CourierSender" in the Protocol Entity panel.

9. D_1 user specifies its own entity ID by filling the textfield in Entitiy Identity panel.

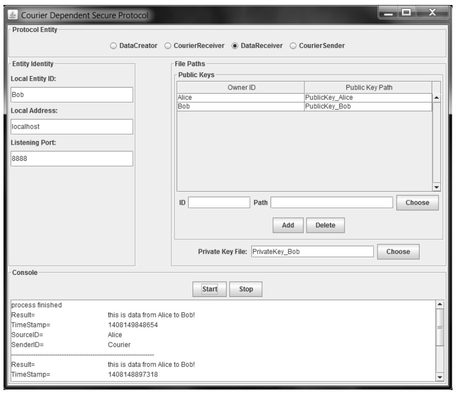
10. D_1 user specifies the ID of the entity he wants to contact (here D_2's ID), together with its address and port number by filling corresponding textfields in the rest of Entity Identity panel.

11. D_1 user add the public key of the entity he wants to contact into the public key list. The way of doing that has been introduced in step 4.

12. D_1 user specifies the file that contains the data needs to be sent, by filling the file path in the corresponding textfield in Data to Send panel.

13. D_1 user click the Start button in the Console panel. The running progress of the protocol will be displayed in the information board in the Console panel.

Below figures show snapshot of two applications successfully running Transmit Protocol. After application consoles on both devices pops “process finish” successfully, the data carried by D_1 has been transmitted to D_2. If the authenticity of the data is successfully verified, D_2 will print the message content out in the information board. If the data contains messages from multiple senders, they will be verified and printed one by one so that user knows which messages have been discarded.



Protocol Entity: ○ DataCreator ○ CourierReceiver [✓ DataReceiver] ○ CourierSender

Entity Identity
Local Entity ID: Bob
Local Address: localhost
Listening Port: 8888

File Paths
Public Keys

Owner ID	Public Key Path
Alice	PublicKey_Alice
Bob	PublicKey_Bob

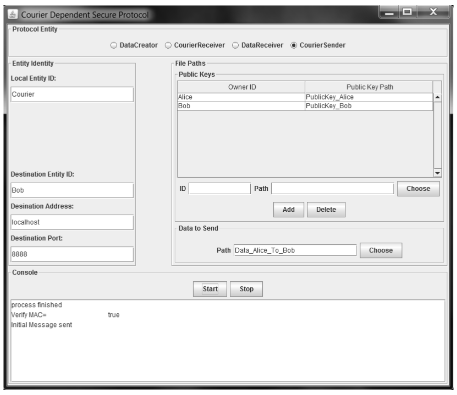
ID
Path

Private Key File: PrivateKey_Bob

Console

```
process finished
Result=                     this is data from Alice to Bob!
TimeStamp=                  1408149848654
SourceID=                   Alice
SenderID=                   Courier
---------------------------------------------------
Result=                     this is data from Alice to Bob!
TimeStamp=                  1408148897318
```

(a) DataReceiver



Protocol Entity: ○ DataCreator ○ CourierReceiver ○ DataReceiver [✓ CourierSender]

Entity Identity
Local Entity ID: Courier
Destination Entity ID: Bob
Desination Address: localhost
Destination Port: 8888

File Paths
Public Keys

Owner ID	Public Key Path
Alice	PublicKey_Alice
Bob	PublicKey_Bob

ID
Path

Data to Send
Path Data_Alice_To_Bob

Console

```
process finished
Verify MAC=                 true
Initial Message sent
```

(b) CourierSender

Figure 3: Running Submit Protocol

0.1.3 Use Case

0.2 Software Design

The implementation of this protocol mainly consists of two parts - a core library and an user interface. The core library defines the framework of the program and provides all essential components for running the protocol. While the user interface takes user’s input, configures components provided by the core library, runs the protocol and outputs running results if necessary. This design separates the implementation of user interfaces from the actual running of the protocol, the major benefit is that when this protocol is running on different devices or platforms, various user interfaces can be created and easily plugged to the protocol library without modifying the core library.

0.2.1 Program Architecture

Framework

Entity Model The $\mathcal{S}$ denotes its set of internal states which defines how $\prod$ responds to a certain input $\mathcal{M}$. Taking $\mathcal{A}$ as example,the figure below shows a 3-internal-state oracle which abstracts the behaviour of $\mathcal{A}$: initially $\prod_S$ is in Wait state, waiting for incoming message . Once it receives and verifies $\mathcal{M}_0$, it sends the $\mathcal{M}_0'$ out and enters Send state, waiting for next message. After that it receives and verifies $\mathcal{M}_1$ and enters the final state Check. If any error occurs when $\prod_S$ in Wait or Send state, it enters Error state and session is aborted.

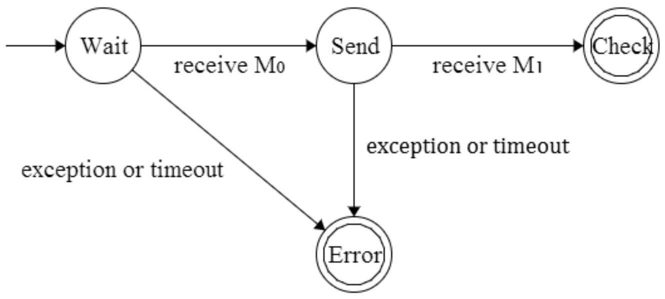


Figure 4: State Machine Model

0.2.2 Message Structure

0.3 Implementation Details

UML Class Diagram