Codimension-Two Free Boundary Problems

Keith Gillow

St Catherine's College

University of Oxford

A thesis submitted for the degree of

Doctor of Philosophy

Trinity 1998

Contents

List of Figures

Chapter 1

Protocol Implementation

1.1 Application Overview

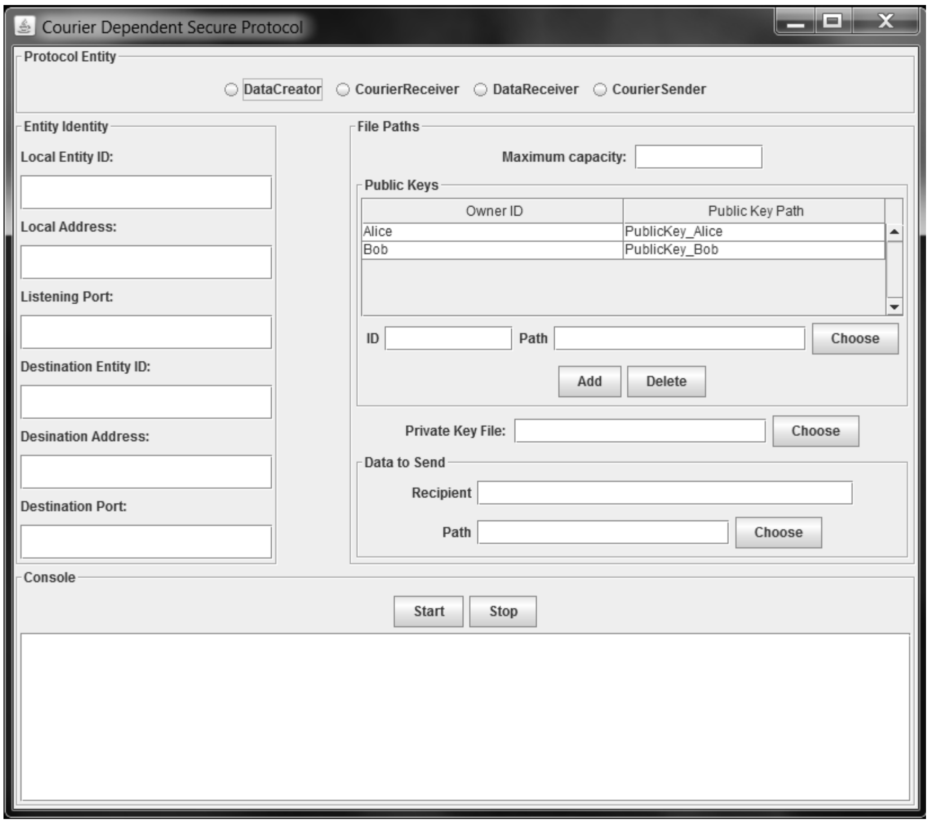
To run this protocol in a practical scenario, at least 3 different devices are required - one be Alice, one be Bob and one be Courier. This application implements all 3 entities within it, so once the device has this application, it can be any entity in the protocol. To achieve this, at the start of 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, and user will be requested to input some information related to the selected role.

Below is a snapshot of the GUI of the application, it shows all the selections and textfields that will be used when the application interacts with user.

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 is given in the appendix.



Owner ID	Public Key Path
Alice	PublicKey_Alice
Bob	PublicKey_Bob

Figure 1.1: GUI Overview

To further illustrate how user can interact with the system, a use case diagram is given and every use cases appear in the diagram will be explained. For disambiguating, the “Alice”, “Bob” and “Courier” appeared in the use case diagram do not mean the protocol entity as introduced in the system model, but are the names of the actors who control the device. The name of the actor simply reflects which entity the actor wants to be in the protocol.

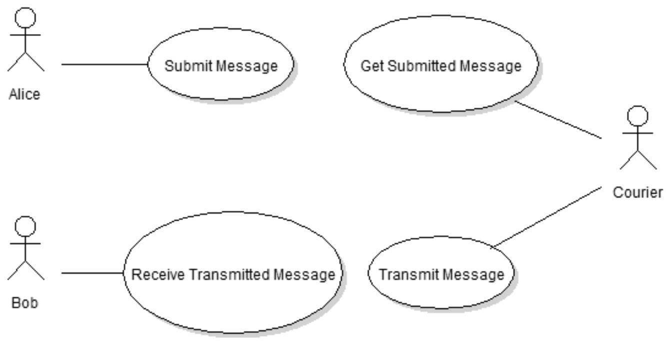


Figure 1.2: Use Case Diagram

Use Case 1: Submit Message

Title	Submit Message
Primary Actor	Alice
Goal in Context	Alice tries to submit the message to a Courier
Scope	System (Black Box)
Level	
Stakeholders	Alice and Courier
Success Guarantees	Message is received by Courier and Alice confirms the fact
Trigger	Alice starts the Submit Protocol
Main Success Scenario	1. Alice starts the application. 2. Alice selects “DataCreator” as the protocol entity in the application. 3. Alice specifies all related information as instructed in the “Run Submit Protocol” section. 4. Alice starts running the protocol.

Use Case 2: Get Submitted Message

Title	Get Submitted Message
Primary Actor	Courier
Goal in Context	Courier tries to obtain the message of Alice
Scope	System (Black Box)
Level	
Stakeholders	Courier and Alice
Success Guarantees	Courier gets a message from Alice
Trigger	Courier starts the Submit Protocol
Main Success Scenario	1. Courier starts the application. 2. Courier selects “CourierReceiver” as the protocol entity in the application. 3. Courier specifies all related information as instructed in the “Run Submit Protocol” section. 4. Courier starts running the protocol.

Use Case 3: Receive Transmitted Message

Title	Receive Transmitted Message
Primary Actor	Bob
Goal in Context	Bob tries to receive message of Alice carried by Courier
Scope	System (Black Box)
Level	
Stakeholders	Bob and Courier
Success Guarantees	Bob receives the message
Trigger	Bob starts the Transmit Protocol
Main Success Scenario	1. Bob starts the application. 2. Bob selects “DataReceiver” as the protocol entity in the application. 3. Bob specifies all related information as instructed in the “Run Transmit Protocol” section. 4. Bob starts running the protocol.

Use Case 4: Transmit Message

Title	Transmit Message
Primary Actor	Courier
Goal in Context	Courier tries to transmit Alice's message to Bob
Scope	System (Black Box)
Level	
Stakeholders	Courier and Bob
Success Guarantees	Message is received by Bob and Courier confirms the fact
Trigger	Courier starts the Transmit Protocol
Main Success Scenario	1. Courier starts the application. 2. Courier selects "CourierSender" as the protocol entity in the application. 3. Courier specifies all related information as instructed in the "Run Transmit Protocol" section. 4. Courier starts running the protocol.

1.2 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, its user interfaces can be customized and easily plugged to the core library without modifying it.

1.2.1 Program Architecture

As introduced above, the whole program contains 4 roles - DataCreator, DataReceiver, CourierSender and CourierReceiver. Classfied by their main structure, those 4 different roles fall into 2 categories - (1) Accepter, who continuously listens to the port and waits for incoming connection. (2) Initiator, who actively connects to a

Accepter. Although these two categories sounds similar to Client-Server pattern, it is not quite the case because Accepter does not actually provide any service. Based on their behaviour definition in protocol specification, DataCreator and DataReceiver belong to Accepter, while CourierSender and CourierReceiver belong to Initiator. Following paragraphs will introduce the internal architecture for Accepter and Initiator respectively.

Accecpter Accepter contains 5 main parts:

- Dispatch Thread
  The Dispatch Thread listens to the program port, waits for incoming connections. Once it receives a connection, it will create a new session, then handover the connection to the newly created session. After that, it will continue listening to the port, wait for next incoming connection.

- Sessions
  Sessions are created by the Dispatch Thread, each session corresponds to a single connection and sessions are independent between each other. Once a session takeovers a connection, it is fully responsible for it. Inside a session there is a Delegate and a Sub-thread.

  The **Delegate** defines all communication content (e.g. a Delegate of Alice defines the content of messages to send out, and it also defines what to do when receives a certain message), it behaves like a state machine which takes message as input, processes the message and output a new message for response. During the processing of input messages, it may interact with the 3 global objects - User Interface, Cryptographic Kit and Data Manager. The detail of Delegate will be explained in Delegate Model.

  The **Sub-thread** listens to the connection which is handover by Dispatch Thread, capture any message from the connection. It delivers the captured message to Delegate, and receives a new message from Delegate. If the output message from Delegate does not indicate an exception or termination, the Sub-thread sends the message through the connection, otherwise it reports an error or terminates.

- User Interface
  The User Interface is shared between all sessions in Accepter. It displays all

relative information to show the progress of the protocol running and reports errors to the user.

- Cryptographic Kit
  The Cryptographic Kit is shared between all sessions in Accepter. It provides functionalities of all cryptographic operations that is needed in the protocol, such as encryption, decryption, key generation, MAC generation and verification, and digital signature generation and verification. When any session need to do cryptographic operations, it just call from Cryptographic Kit and doesn't care the internal implementation. For the consistency concern, it is required that any two communicating device should use same Cryptographic Kit.

- Data Manager
  The Data Manager is shared between all sessions in Accepter. It keeps record of all data files in the device's disk which is related to the protocol, such as files of public keys, secret keys, and the message Courier carries. Data Manager is configured at the start of the application, when any session need data in disk, it just request from Data Manager.

The relation between those 5 components is illustrated in the figure "Accepter Architecture".

Initiator Similar to Accepter, Initiator consists of 4 main parts: (1) Session, (2) User Interface, (3) Cryptographic Kit and (4) Data Manager. The major difference between Initiator and Accepter is that Initiator does not have a Dispatch Thread, every Initiator only has one Session. The Session is created once the Initiator is started, and it will ask its Delegate for a "initial message" - which will the first message of a protocol, and send it to the specified Accepter. Its architecture design is a simpler version of Accepter's, and it is illustrated in the figure "Initiator Architecture".

1.2.2 Delegate Model

The object Delegate is the essence of the application, it directly refers to the specification of the protocol, defines how an entity processes a message - here "process a message" may involve doing cryptographic operations, checking message content validity and giving a response message.

Basically, every Delegate acts like a finite-state machine, who possesses an internal state, takes messages as inputs and outputs new messages as response. The internal

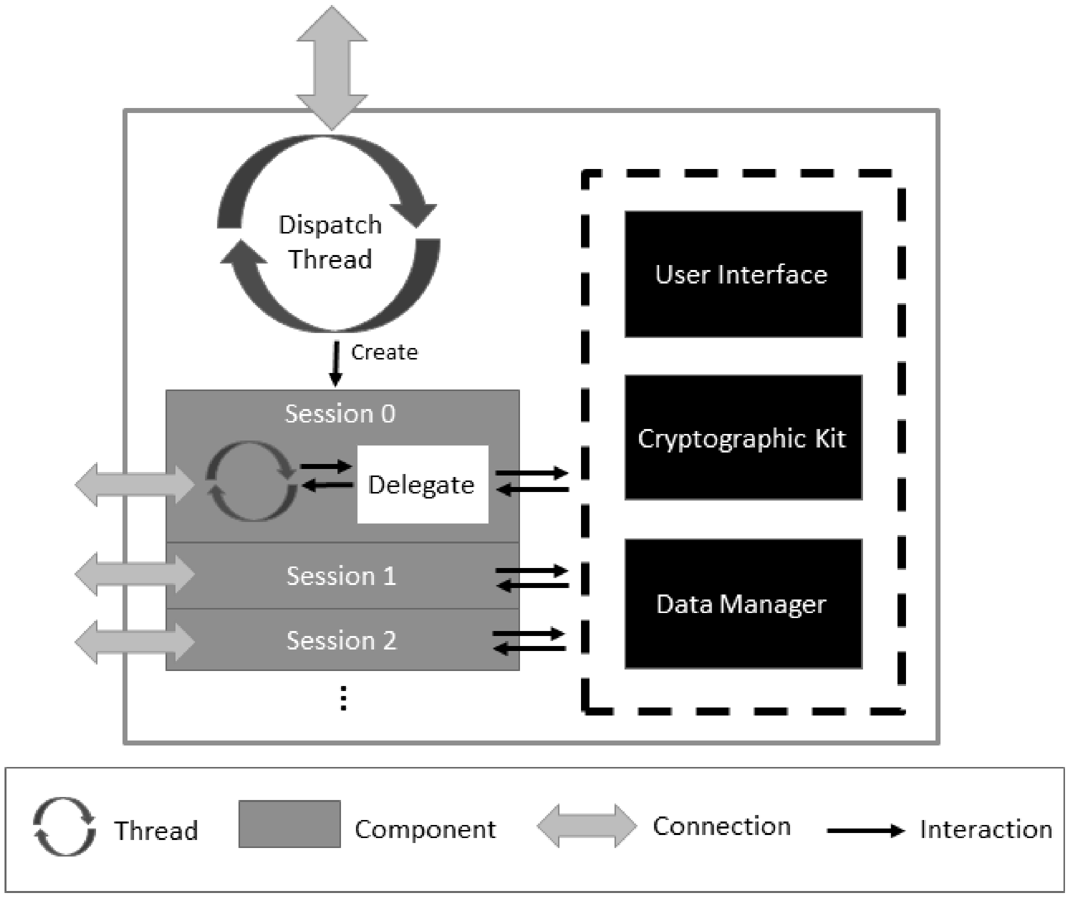


Figure 1.3: Accepter Architecture

state controls the behaviour of the Delegate - Delegates in different states will respond differently to a same input. So when an input message comes, Delegate will process the message according to its current state, if the message is successfully processed, it will change its current state and wait for next message, until it reaches the final state.

Taking entity $\mathcal{A}$ as example,the figure below shows a 4-internal-state machine which abstracts the behaviour of $\mathcal{A}$: initially $\mathcal{A}$ is in Wait state, waiting for the first message. Once it receives the first message, it will process the message (according to the Submit Protocol, it will check the first message and submit its data to Courier). If $\mathcal{A}$ successfully processed M_0, it enters Submitted state, waiting for the second message. Then it will receive and process M_1 (according to the Submit Protocol, it will check the validity of the MAC). If it is successful again, it enters the final state Checked and stops. If any error occurs in processing the input messages in early

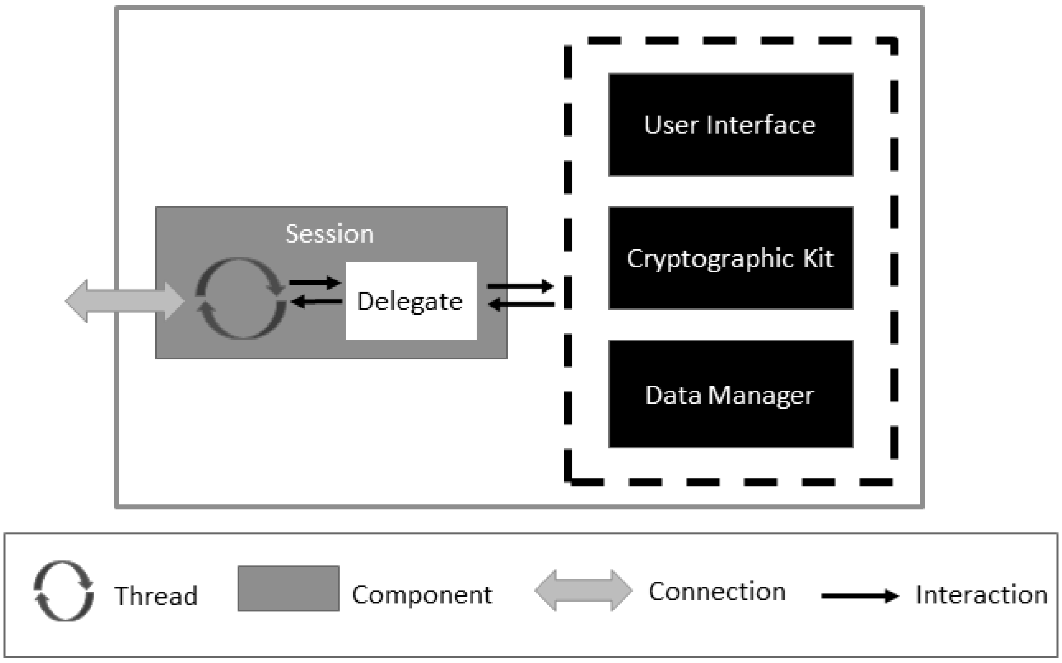


Figure 1.4: Initiator Architecture

states, $\mathcal{A}$ directly enters final state Error and stops.

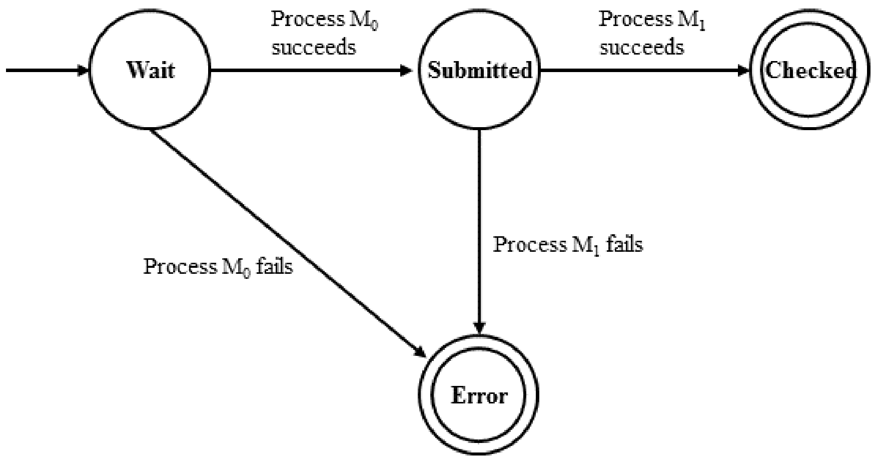


Figure 1.5: State Machine of $\mathcal{A}$

1.2.3 Message Structure

1.3 Implementation Details

UML Class Diagram how key, data is managed time out timestamp not implemented

Appendix A

Sample Title

Lorem ipsum dolor sit amet, consectetur adipiscing elit, sed do eiusmod tempor incididunt ut labore et dolore magna aliqua. Ut enim ad minim veniam, quis nostrud exercitation ullamco laboris nisi ut aliquip ex ea commodo consequat. Duis aute irure dolor in reprehenderit in voluptate velit esse cillum dolore eu fugiat nulla pariatur. Excepteur sint occaecat cupidatat non proident, sunt in culpa qui officia deserunt mollit anim id est laborum.

Appendix B

Sample Title

Lorem ipsum dolor sit amet, consectetur adipiscing elit, sed do eiusmod tempor incididunt ut labore et dolore magna aliqua. Ut enim ad minim veniam, quis nostrud exercitation ullamco laboris nisi ut aliquip ex ea commodo consequat. Duis aute irure dolor in reprehenderit in voluptate velit esse cillum dolore eu fugiat nulla pariatur. Excepteur sint occaecat cupidatat non proident, sunt in culpa qui officia deserunt mollit anim id est laborum.

Bibliography

[1] Albert Einstein. Zur Elektrodynamik bewegter Körper. (German) [On the electrodynamics of moving bodies]. *Annalen der Physik*, 322(10):891–921, 1905.