**Final Project Proposal**

Web Based Secure Purchase Order

圖資系 b97106021 謝育璘

**專題要求概述：**

* 共有三種使用者：Purchaser、Supervisor、Orders Department
* 所有使用者的互動必須是透過網路
* 在每一方溝通的過程中，必須先經過Public-key mutual authentication
* 任何一個使用者所傳送的訊息都必須要經過RSA加密
* signature使用public key based，並且在傳送的message中必須加上timestamp

**初步想法：**

我初步構想的系統架構主要是由一台Server負責Purchaser、Supervisor、Orders Department三方的溝通，而三種使用者都是透過瀏覽器與Server溝通，並且透過Server從Database存取任何一方給予的訊息，但是再這樣的架構下會有一些問題。

例如三種使用者在第一次使用系統時，會由前端產生一組Public/Private Key，但因為是使用瀏覽器操作，前端無法使用Javascript語言做讀檔寫檔來存放任何變數，所以必須將自己的Public/Private Key傳送到Server，並存放在Database中，每次登入系統後會依照使用者的身分來與Server取得自己的Public/Private Key；當然，在這樣的架構下就必須假設Server是值得信任的，所以才能將Private Key存放在Server。

另外一個問題是在選擇Library時，因為Server和Client需要加密運作方式相同的Library，所以在尋找適合的Library需要花很多時間，或者可以選擇先找到Client端適合的Encrypt/Decrypt Library，在將它移植到Server端。

以上是系統大致上的架構以及初步會遇到的問題，下面是系統大概的架構圖，以及Client與Server的架構。

**整體系統架構圖：**

Client

Javascript

HTTP

Cryptico

Service

Node.js

HTTP

MacOS

MongoDB

HTTP Request

**Client：**

在Client端主要使用的語言是Javascript，使用的Encrypt/Decrypt Library是Cryptico，另外還使用Backbone.js framework來建置前端的MVC架構，並且透過HTTP發Request與Server溝通。

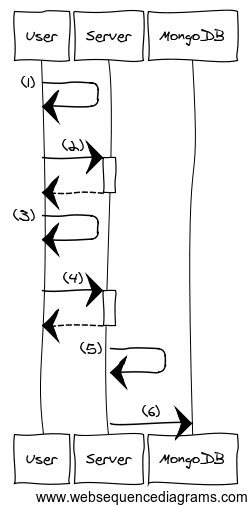
**Server：**

在Server端是使用Node.js來實作，所以不需要額外架Apache Server，與Client透過HTTP protocol來溝通，在實作上我是將Server直接建置在Macbook上，另外Server會將接收到的Message先存到Database已供其他使用者存取，Database是採用No-SQL的Database－MongoDB。

**整個系統的流程如下：**

1. Purchaser、Supervisor、Orders Department各自產生各自的Private/Public Key，並且存放到Server，在傳送的過程會先要求Server提供Public加密這組Key，Server接收到Key後解密再存到MongoDB中。

下圖呈現User與Server在存入Private/Public Key的互動過程：

(1). User先產生一組Public/Private Key

(2). 取得Server的Public Key，這組Public

Key只能使用一次

(3). 用Server的Public Key將自己的

Public/Private Key加密

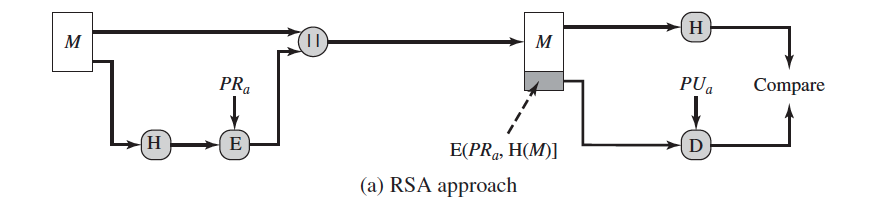
(4). 將加密後的Public/Private Key送給

Server

(5). Server將Public/Private Key解密

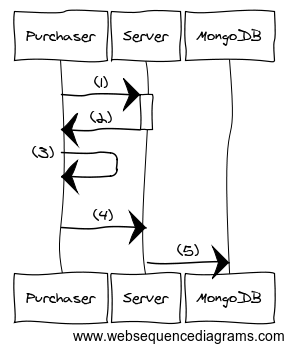
(6). 將Public/Private存入Database中

1. Purchaser撰寫訂單，在訂單內加入timestamp，再使用RSA加密後，並且加上Signature，Sign的方式是採用RSA的方法，其加密方式如下圖：



其中是自己的Private Key，對方可以用你的Public Key來解密，並且比較其值是不是相同來確定對方的身分。

下圖為此一階段使用者與Server的互動流程：

(1). 身分認證（使用帳號密碼）

(2). 向Server取得自己的Private

Key以及Supervisor、Orders

Department的Public Key

(3). 將訂單複製兩份，加上

timestamp後，分別用兩個

Public Key加密，並且附上用

自己的Private Key所產生的

Signature

(4). 將這兩份訂單傳送給Server

(5). Server將這兩份訂單各自存

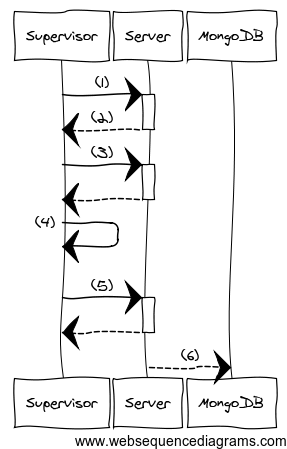
到Supervisor以及Orders

Department存取訂單的

Table中

1. Purchaser送出訂單後，Supervisor可以向Server取得未審核的訂單，用Supervisor的Private Key解密後，確認其signature，在經審核後，將審核後的訂單以及signature傳送給Server。

下圖為此一階段使用者與Server的互動流程：

(1). 身分認證（使用帳號密碼）

(2). 取得自己的Private Key以及其他

使用者的Public Key

(3). 取得未審核的訂單以及Purchaser

的Public Key

(4). 用自己的Private Key解密訂單，

由Signature確認Purchaser

身分後，審核訂單，將審核後的訂

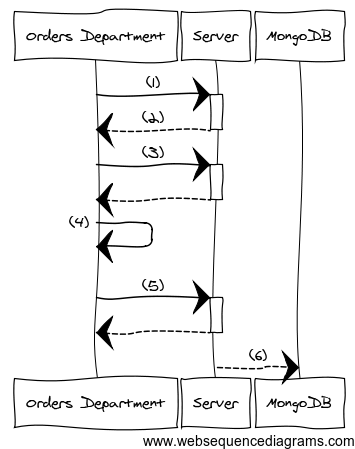
單用Orders Department的

Public Key加密，並且附上用自己 的Private Key所產生Signature。

(5). 將審核並加密後的訂單傳送給 Server

(6). 將審核後並加密的訂單存到 Database

1. Orders Department從Server取得Purchaser以及Supervisor審核後的訂單，解密後確認他們的signature、timestamp，經過比對後，將確認會購買的訂單送給Server，供Purchaser使用。

下圖為此一階段使用者與Server的互動流程：

(1). 身分認證（使用帳號密碼）

(2). 取得自己的Private Key以及其 他使用者的Public Key

(3). 取得Purchaser的訂單以及 Supervisor已審核過的訂單

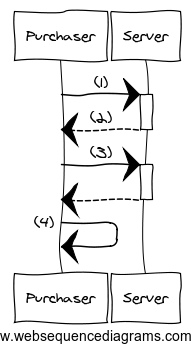
(4). 用自己的Private Key解密訂單， 由Signature確認Purchaser、 Supervisor身分以及timestamp， 確認無誤後將確認已購買的訂單 用Purchaser的Public Key加密， 並且附上用自己的Private Key 所產生的Signature。

(5). 將確認已購買並加密後的訂單傳送給Server

(6). Server將訂單存到Database中

1. Purchaser從Server取得”確認已購買”的訂單。

下圖為此一階段使用者與Server的互動流

(1). 身分認證（使用帳號密碼）

(2). 取得自己的Private Key以及其他使用者

的Public Key

(3). 從Server取得”確認已購買”的訂單

(4). 使用自己的Private Key解密，並且用

Orders Department的Public Key確認

Signature。

**採用的加密演算法：**

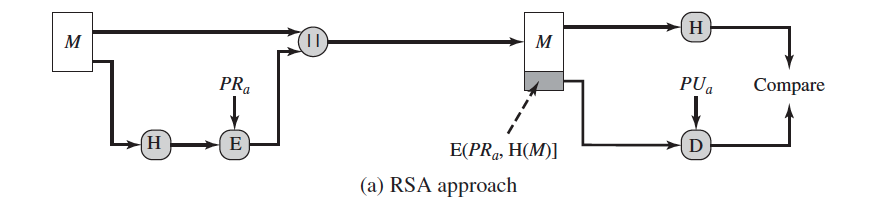
1. RSA Algorithm：

在User與Server之間傳送的所有Message都是由RSA加密後才能傳送，會選擇RSA演算法一方面是因為題目的要求，另一方面是因為RSA演算法的運算速度還在可以接受的範圍，並且採用512位元也具有相當的安全性，且在訊息傳送的過程中，若是採用一次傳送產生一組新的RSA Key，安全性則可以在提昇。

採用RSA 演算法在Key的保管上也比較容易，如果你要跟N的對象交換訊息也只要保管好自己的Private Key，將Public Key公開給對方知道就可以了，但你若是採用其他對稱式加密演算法的話，跟N個對象交換訊息就要保管N把Key，保管Key的成本會變得像當高。

1. RSA approach to Digital Signatures

其演算法的示意圖如下：



其演算法的操作如下：

假設H(M)代表訊息M的訊息指紋值，那麼RSA數位簽章的產生與檢驗可簡單描述如下

1. 產生RSA簽章: 用私密金鑰D與公開金鑰N，以下列公式計算簽章
2. 檢驗RSA簽章: 將收到的訊息M重新計算訊息指紋值H(M)。取得公開金鑰E, N。若且唯若下列公式成立，則接受簽章，否則拒絕簽章

此演算法滿足題目的要求，Signature必須要是Public Key based，並且要使用Hash function，雖然RSA Digital Signature與DSA在產生相同金鑰長度的數位簽章時，DSA的速度較RSA快速，但是基於實作上的方便，在我所使用的Library Cryptico中已經有提供RSA的Signature，所以我直接選擇採用此演算法，在運算的速度上也在可以接受的範圍。