# Final Project Report

Web Based Secure Purchase Order

圖資系 b97106021 謝育璘

## 一. 專題重點概述：

- 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

## 初步構想：

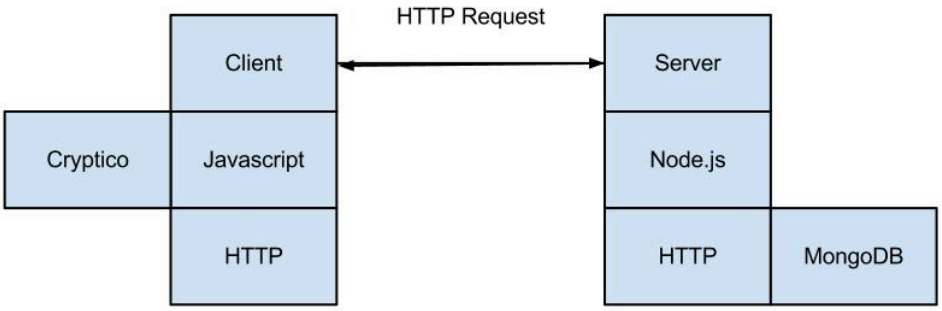
初步構想的系統架構主要是由一台 Server 負責 Purchaser、Supervisor、Orders Department 三方的溝通，而三種使用者都是透過瀏覽器與 Server 溝通，並且透過 Server 從 Database 存取任何一方給予的訊息，但是再這樣的架構下會有一些問題。

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。

## 二. 整體系統架構圖：




**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。

## 三. RSA Encrypt Library－Cryptico 介紹：

Cryptico 是一個 RSA 加密的 Library，是以 Javascript 撰寫，所以適合用在 Client 端，另外我自己將它移植到 Server 端，這樣 Server 端才能存 Client 端所產爭的 Private。

- Generating an RSA key pair & public key string：

  首先要先宣告一個 String，這個 String 是用來重複產生 RSA Key：

  ```
  var PassPhrase = "The Moon is a Harsh Mistree.";
  ```

  另外還要宣告 RSA Key 的長度：

  ```
  var Bits = 1024;
  ```

  而產生 Private Key 的 function 如下：

  ```
  var MaryRSAKey = crpytico.generateRSAKey(PassPhrase, Bits);
  ```

  產生 Public Key 的 function 如下：

  ```
  var MaryPublicKeyString = crpytico.publicKeyString(MaryRSAKey );
  ```

- Encrypting a message：

  在 Encrypt 的部份，必須將要加密的資料轉為 String，其加密的 Function 如下：

  var EncryptionResult

  = crpytico. encrypt(PlainText, MaryPublicKeyString);

  加密後的 EncryptionResult 是一個 JSON 物件，其結構如下：

  { cipher：String,

  status：Boolean}

  cipher 是 cipherText，而 Status 是指加密有沒有成功，若是加密失敗，則值會是 Failure，且不會有 cipherText。

  若是要加上 Signature 的 RSA Encrypt，則其 Function 如下

  var EncryptionResult

  = crpytico. encrypt(PlainText, MaryPublicKeyString, BobRSAKey);

  上面這個例子是 Bob 要傳訊息給 Mary，所以在 Encrypt function 中的參數分別是原文、Mary 的 Public Key、Bob 的 Private Key，其產生的結果EncryptionResult也是 JSON 物件，其結構與未加 Signature 的 EncryptionResult 相同。

- Decrypting a message：

  在未加 Signature 的時候，其解密的 Function 如下：

  DecryptionResult = cryptico. encrypt(ciphertext, MarryRSAKey);

  其解密後的結果也是 JSON 物件，其結構如下：

  { status: Boolean,

  plaintext: String}

  若是有 Signature，則其解密的 Function 如下：

  DecryptionResult = cryptico. encrypt(ciphertext, MarryRSAKey);

  但解密後的 JSON 物件結構如下：

  { status: Boolean,

  plaintext: String,

  signature: “unsigned” or “verified” or ”forged”,

  publicKeyString: publicKeyString of the signature}

  所以可以從 signature 值來確認此封包的 Signature 是否正確，若正確的話，publicKeyString 會是合法且正確的。

## 四. Database－MongoDB 簡介

MongoDB 是一種檔案導向資料庫，不同於傳統的關聯式資料庫，它是屬於 NoSQL 資料庫（也稱作 Not Only SQL）。

這種資料庫的特色在於它沒有固定的 Schema，如果以關聯式資料庫的角度來看，它可以自由增加 Column，也因為如此，它能夠避免使用 JOIN 的操作。另外，它的出現就是為了應付資訊量迅速暴增的情況，所以他的 Scalability 比傳統的資料庫要好，只要擴充硬碟數量，就能夠增加儲存空間，並不需要考慮 Schema 調整的問題。

在這個系統中，我採用 MongoDB 來儲存所有的訂單以及使用者的帳號密碼以及 Public/Private 都是儲存在資料庫中，MongoDB 儲存資料的資料結構是類似 BSON，它是一種類似 JSON 的結構，它的結構的樣式如下：

{ Index： Value}

以下使用系統的儲存資料格式做例子：

1. User

```
{ "_id" : ObjectId("50eeffb4327fae5d558a4874"),
 "type" : "Purchaser",
 "account" : "purchaser",
 "password" : "purchaser" }
```

在這個例子中，id 就是每一筆 Data 的 ID，然後記錄使用者的帳號密碼，這裡的帳號密碼是用明碼儲存，當然真正安全的系統應該是要經由 Hash Function 做處理，才能妥善保護使用者的帳號密碼。另外，type 是代表 User 的類型，總共有三種，分別是 Purchaser、Supervisor、Orders Department。

2. Order

Order 的部份分成三種，分別是 Purchaser、Supervisor、Orders Department，所以以下列出這三種使用者的 Orders 結構：

Purchaser：
{ "_id" : ObjectId("50f268c985cb381d5f000003"),
"orders" : [ EncryptedObject ],
"ordersDepartmentToPurchaserEncryptOrders" : [ EncryptedObject ],
"user" : "Purchaser" }

Supervisor：
{ "_id" : ObjectId("50f268c985cb381d5f000002"),
"purchaserToSupervisorEncryptOrders" : [EncryptedObject],
"user" : "Supervisor",
"verifiedOrders" : [EncryptedObject],

Department：
{ "_id" : ObjectId("50f268c985cb381d5f000004"),
"purchasedOrders" : [EncryptedObject],
"purchaserToOrdersDepartmentEncryptOrders" : [EncryptedObject],
"supervisorToOrdersDepartmentEncryptOrders" : [EncryptedObject],
"user" : "OrdersDepartment" }

在這個例子中，我們可以看到，MongoDB 中的 BSON 可以包含 Array，所以我將所有的 order 分為幾類，全部放進 Array 中，這裡要注意的一點是，在 Database 中的 Order 全部都是加密過的，所以看不到內容，要等到被取出來之後才能在 Client 端解密看到內容！

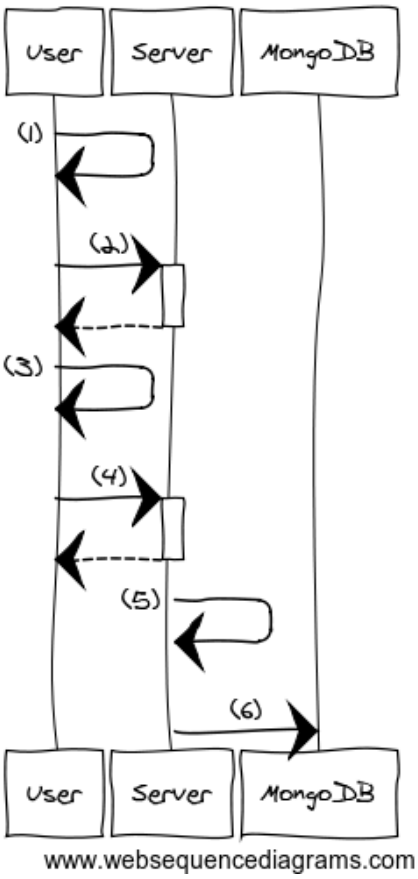
而 EncryptedObject 是經由 Cryptico Library 加密後的物件，其結構在前面介紹 Library 的部分有詳細的說明。

## 五. 系統運作流程

### 1. 取得各自的 Private Key

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

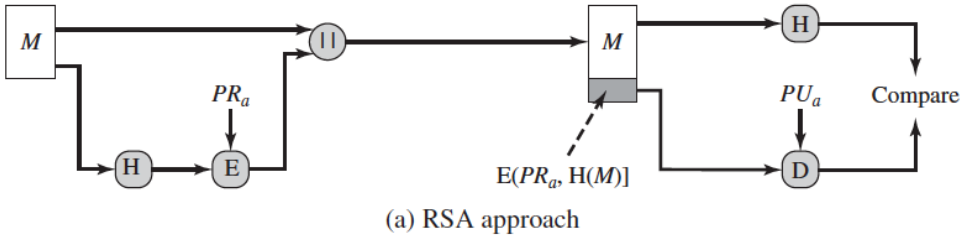
(4). 將加密後的 Public/Private Key 送給 Server

(5). Server 將 Public/Private Key 解密

(6). 將 Public/Private 存入 Database 中

### 2. Purchaser 撰寫訂單

Purchaser 完成訂單後，會在訂單加入 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：

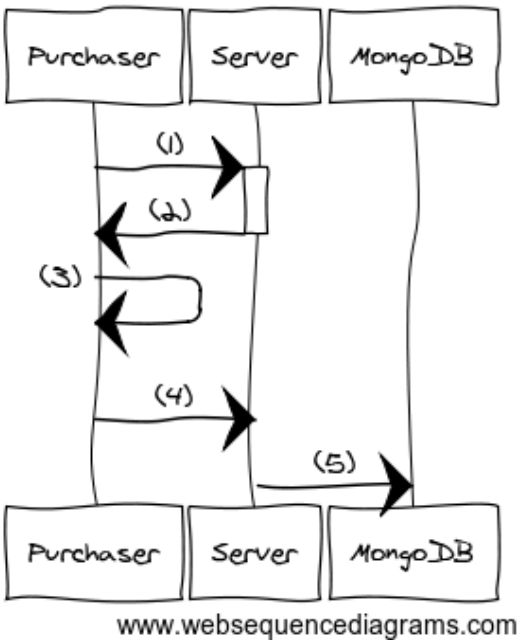



(a) RSA approach

其中$PR_a$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

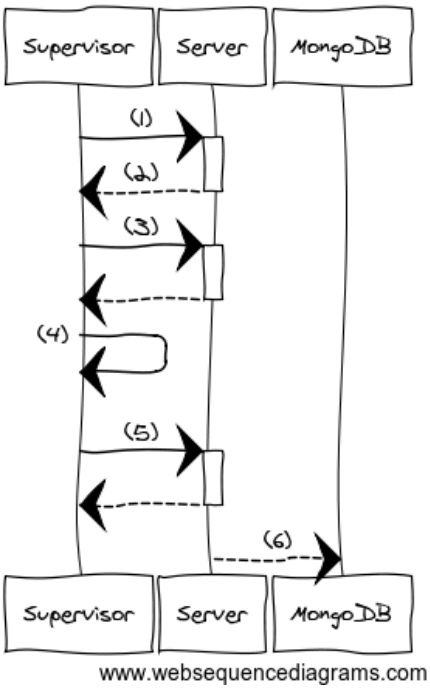
(4). 將這兩份訂單傳送給 Server

(5). Server 將這兩份訂單各自存到 Supervisor 以及 Orders Department 存取訂單的 Table 中

**3. Supervisor 取得要審核的訂單**

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。

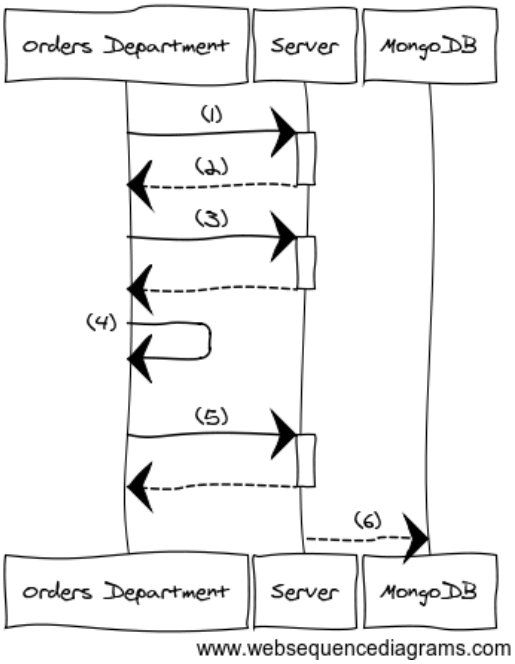
(5). 將審核並加密後的訂單傳送給 Server

(6). 將審核後並加密的訂單存到 Database

**4. Orders Department 取得已審核的訂單**

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。

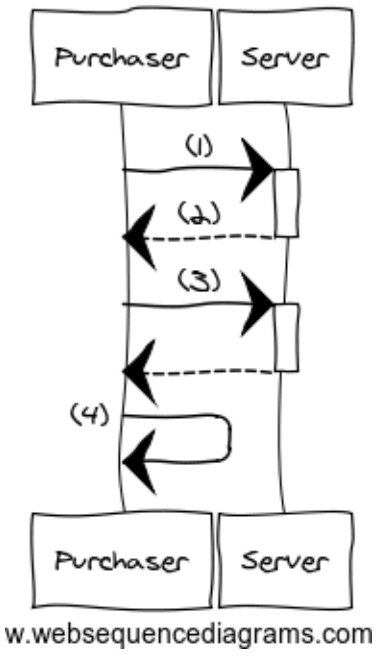
(5). 將確認已購買並加密後的訂單傳送給 Server

(6). Server 將訂單存到 Database 中

**5. Purchaser 取得已購買的訂單通知**

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。

## 六. 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。

### 2. RSA approach to Digital Signatures

其演算法的示意圖如下：


(a) RSA approach

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

$$S = H(M)^D mod\ N$$

(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

$$H(M) = S^E mod\ N$$

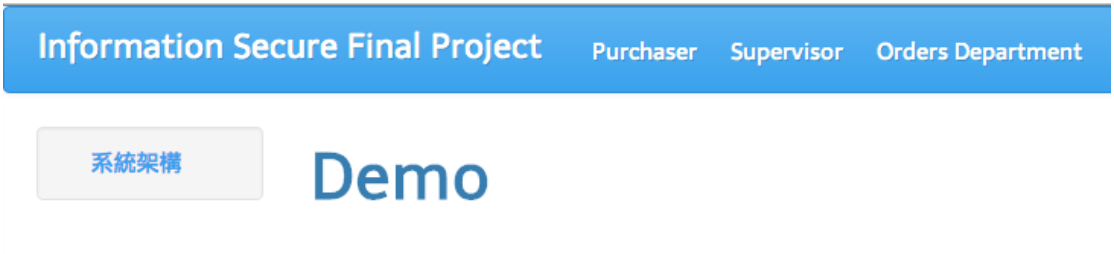
此演算法滿足題目的要求，Signature 必須要是 Public Key based，並且要使用 Hash function，雖然 RSA Digital Signature 與 DSA 在產生相同金鑰長度的數位簽章時，DSA 的速度較 RSA 快速，但是基於實作上的方便，在我所使用的 Library Cryptico 中已經有提供 RSA 的 Signature，所以我直接選擇採用此演算法，在運算的速度上也在可以接受的範圍。

## 七. 系統 API 介紹

實作的系統採用 Server-Client 架構，因為前端是由 Backbone.js Framework，前端可以自己維護一個 MVC 架構，然後由 Ajax 與 Server 做 HTTP Request 交換資料，所以以下列出此系統的 API:

| API | 說明 |
|---|---|
| /checkLoginStatus | 用來 Check 前端是否已經登入了，是由前端的 Cookie 來 check |
| /getOrderNumber | 取得目前 Order 編號 |
| /login | 使用帳號密碼做登入動作 |
| /Purchaser/PrivateKey | 取得 Purchaser 的 PrivateKey |
| /Supervisor/PrivateKey | 取得 Supervisor 的 PrivateKey |
| /OrdersDepartment /PrivateKey | 取得 OrdersDepartment 的 PrivateKey |
| /Purchaser/Orders | 取得 Purchaser 所編寫的訂單以及已經通過審核並購買的訂單，都是已經過加密的資料 |
| /Purchaser/SendOrder | Purchaser 將編寫好的訂單加密後送出，發送給 Orders Department 及 Supervisor |
| /Supervisor/getVerifiedOrder | 取得 Supervisor 的所有訂單，包含未審核以及審核完畢的訂單，其資料也是加密過的 |
| /Supervisor/removePurchaserToSupervisorOrder | 將已經審核過的訂單從 Database 刪除掉 |
| /Supervisor /sendVerifiedOrder | 將審核過的訂單加密後送出，發送給 Orders Department |
| /OrdersDepartment /getWaitedOrder | 取得 Purchaser 所編寫的訂單以及 Supervisor 審核過的訂單 |
| /OrdersDepartment/removeSupervisorToOrdersDepartmentOrder | 將 Purchaser 編寫的訂單以及 Supervisor 審核過的訂單做確認後刪除掉 |
| /OrdersDepartment /sendPurchasedOrder | 將已經購買的訂單加密後傳送給 Purchaser |
| /OrdersDepartment /PurchasedOrders | 取得已經購買的訂單資料，此資料也是加密過的 |

## 八. 系統使用流程




一開始進入系統時，會先看到這個畫面，在 Top Navigator 中有三個按鈕，分別是 Purchaser、Supervisor、Orders Department，使用者進入系統後，可以選擇要登入哪一個使用者，以下以 Purchaser 為例，若點選 Purchaser，則會進入下面頁面：


在尚未登入前，左邊的按鈕是無效的，要先輸入帳號密碼登入後，系統確認帳號密碼無誤，會取得 Purchaser 的 Private Key，並且進入 Purchaser 介面：

- Purchaser

順利登入Purchaser後，接著就能開始編寫訂單或者確認以購買訂單，其介面分別如下：

■ 編寫訂單


在編寫訂單的頁面中，上面可以讓使用者輸入購買的物品名稱、數量以及價格，確認後按下送出，新增的資料就會馬上出現在下列表格中：


且其狀態為”Not Yey Reviewed”，表示此訂單尚未被 Supervisor 審核通過。

■ 已購買訂單

已訂購訂單為 Supervisor 審核通過且 Orders Department 已經購買並通知 Purchaser 後，才會出現在已購買訂單的介面中：

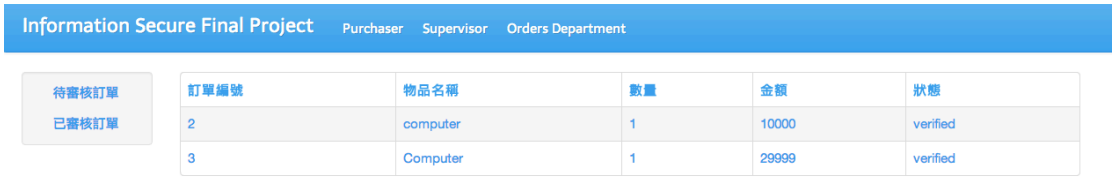

- Supervisor

■ 待審核訂單

Purchaser 發送訂單給 Supervisor 後，Supervisor 登入後點選待審核訂單，就可以看到尚未審核過的訂單：

若是 Supervisor 審核訂單後，要將訂單送給 Orders Department 購買，則只要點下那筆訂單後面的”通過”按鈕，則此筆訂單會從待審核訂單中移除，並且會在已審核訂單中列出。

■ 已審核訂單




已經通過審核的訂單會在此列出，其狀態為 Verified，而審核過的訂單也會送給 Orders Department。

● Orders Department

■ 待採購訂單


待採購訂單為通過 Supervisor 審核的訂單，所以在這裡列出的訂單都是必須要購買的，若是 Orders Department 已經購買此訂單的物品，則可按下”通知已購買”的按鈕，系統將會把此訂單傳送給 Purchaser。

■ 已採購訂單


已採購的訂單資料會列在這個表格中列出，其狀態為 Purchased，相同的資料也會傳送一份給 Purchaser。

上述是這個系統大致上的使用流程，所有的資料都是從 Database 取得，並且在前端透過各自的 Private Key 解密後才顯示在表格中。