**Final Project Report**

Web Based Secure Purchase Order

圖資系 b97106021 謝育璘

1. **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：**

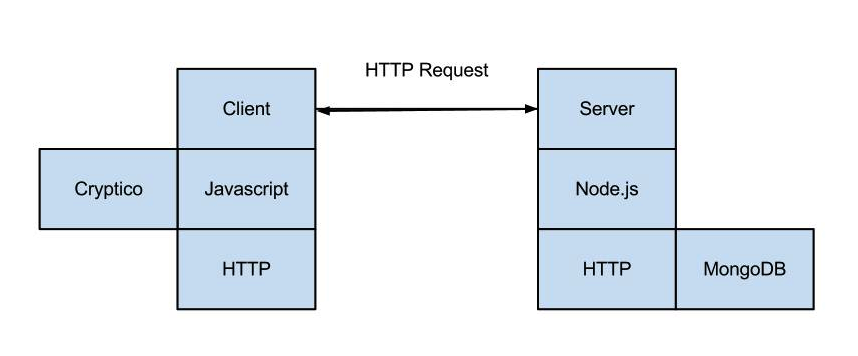
初步構想的系統架構主要是由一台Server負責Purchaser、Supervisor、Orders Department三方的溝通，而三種使用者都是透過瀏覽器與Server溝通，並且透過Server從Database存取任何一方給予的訊息，但是再這樣的架構下會有一些問題。

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。

1. **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。

1. **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：

另外還要宣告RSA Key的長度：

而產生Private Key的function如下：

產生Public Key的function如下：

* 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：

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

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：

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：

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。

1. **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：

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。

1. 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"),

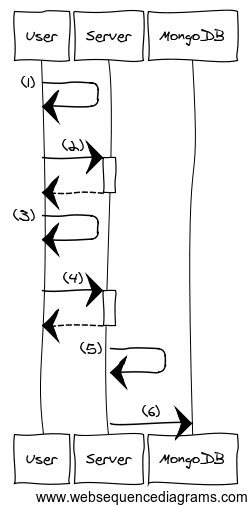
"purchasedOrders" : [EncryptedObject], "purchaserToOrdersDepartmentEncryptOrders" : [EncryptedObject], "supervisorToOrdersDepartmentEncryptOrders" : [EncryptedObject], "user" : "OrdersDepartment" }

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。

1. **系統運作流程**
2. **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

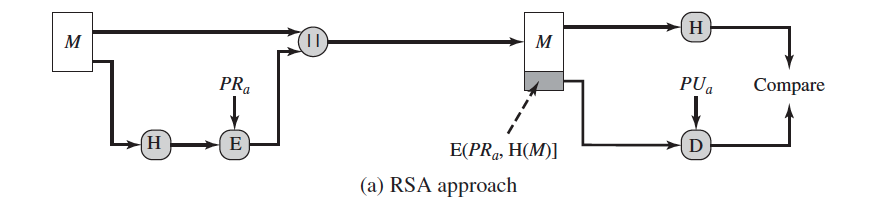
Server

(5). Server將Public/Private Key解密

(6). 將Public/Private存入Database中

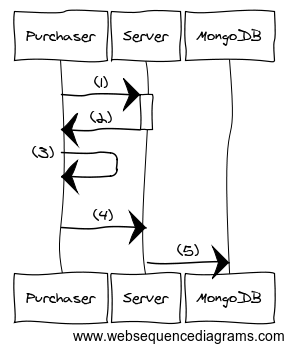
1. **Purchaser撰寫訂單**

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：



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

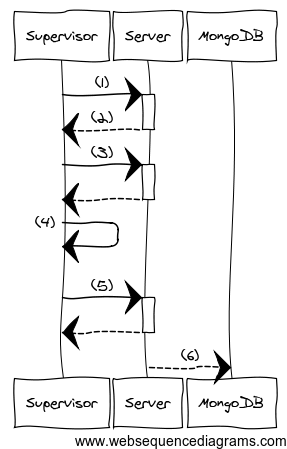
Department存取訂單的

Table中

1. **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。

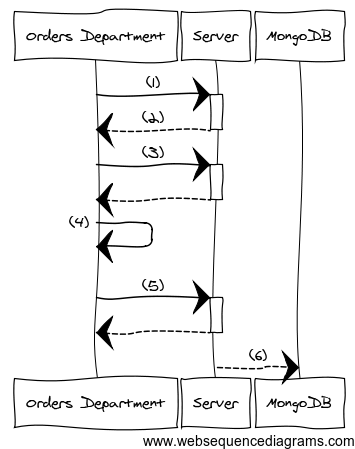
(5). 將審核並加密後的訂單傳送給 Server

(6). 將審核後並加密的訂單存到

Database

1. **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

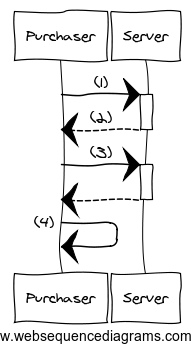
送給Server

(6). Server將訂單存到Database中

1. **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。

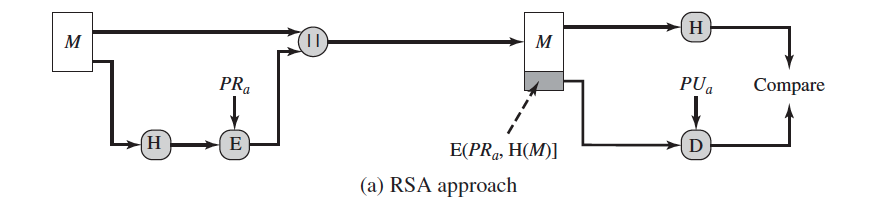
1. **採用的加密演算法**
2. **RSA Algorithm：**

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

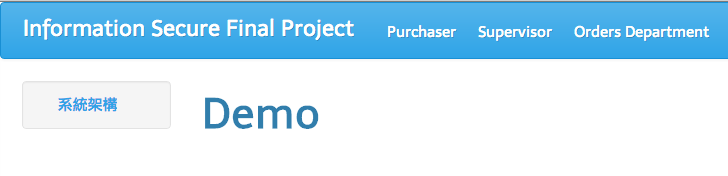
此演算法滿足題目的要求，Signature必須要是Public Key based，並且要使用Hash function，雖然RSA Digital Signature與DSA在產生相同金鑰長度的數位簽章時，DSA的速度較RSA快速，但是基於實作上的方便，在我所使用的Library Cryptico中已經有提供RSA的Signature，所以我直接選擇採用此演算法，在運算的速度上也在可以接受的範圍。

1. **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 |

1. **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



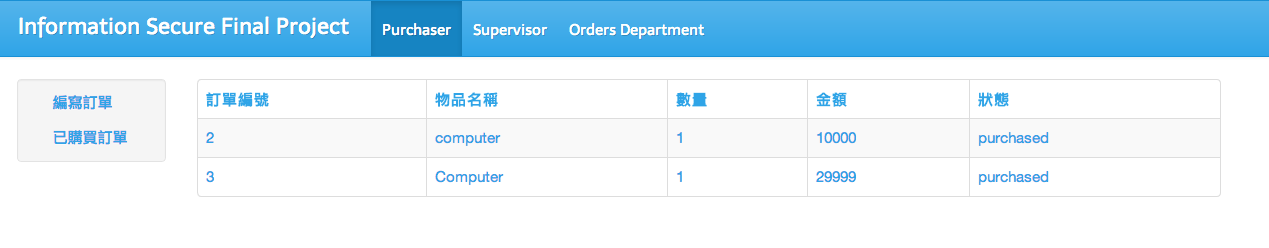
在編寫訂單的頁面中，上面可以讓使用者輸入購買的物品名稱、數量以及價格，確認後按下送出，新增的資料就會馬上出現在下列表格中：



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

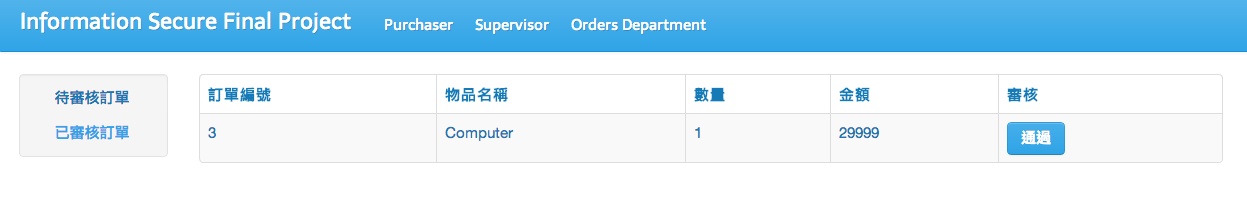
* 已購買訂單

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：



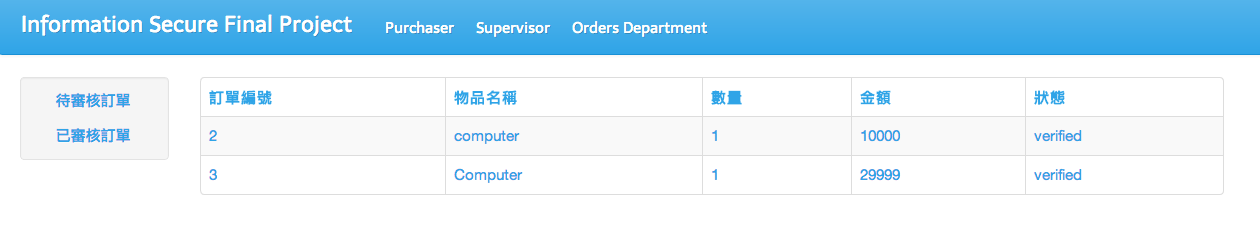
* Supervisor
* 待審核訂單

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：



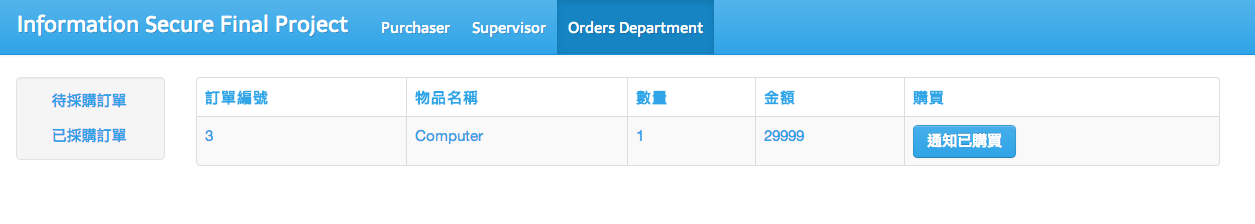
若是Supervisor審核訂單後，要將訂單送給Orders Department購買，則只要點下那筆訂單後面的”通過”按鈕，則此筆訂單會從待審核訂單中移除，並且會在已審核訂單中列出。

* 已審核訂單



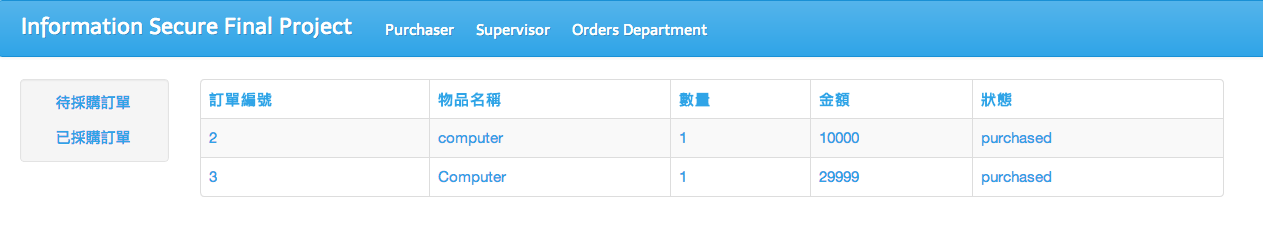
已經通過審核的訂單會在此列出，其狀態為Verified，而審核過的訂單也會送給Orders Department。

* 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。