國立政治大學資訊科學系

Department of Computer Science

National Chengchi University

碩士論文

Master's Thesis

區塊鏈與物聯網整合架構下資料匯集與使用者自主管理存取機制的設計樣式

Design Patterns for Accountable Data Collection and User-Managed Access Mechanism in Blockchain-driven IoT Services

研 究 生：林俊安

指導教授：廖峻鋒 博士

中華民國一零九年六月

June 2020

DOI:10.6814/NCCU202101738

區塊鏈與物聯網整合架構下資料匯集與使用者自主管理存取機制的設計樣式

Design Patterns for Accountable Data Collection and User-Managed Access Mechanism in Blockchain-driven IoT Services

研 究 生：林俊安　Student：Chun-An Lin

指導教授：廖峻鋒　Advisor：Chun-Feng Liao

國立政治大學
資訊科學系

碩士論文

A Thesis
submitted to Department of Computer Science
National Chengchi University
in partial fulfillment of the Requirements
for the degree of
Master
in
Computer Science

中華民國一〇九年六月

June 2020

DOI:10.6814/NCCU202101738

摘要

近年來，越來越多開發人員開始將區塊鏈與異質技術結合。其中，因為區塊鏈具有高度去中心化與擴展性，故區塊鏈與物聯網整合服務，又稱 B-IoT (Blockchain-driven IoT services)，受到很大的矚目，也有許多實作原型被提出。由於區塊鏈技術仍在發展階段，因此，建置高品質的 B-IoT 系統困難度較高。在匯集物聯網裝置中的資料時，裝置間互動機制的設計會對資料的安全性、系統的效能與成本造成極大的影響。此外，由於近年來資安攻擊事件頻繁，許多物聯網開發人員選擇依賴於中心化的存取控制服務來確保系統的安全。然而，若越依賴中心化的存取控制機制，系統的可用性與可維護性則越低。基於上述原因，本論文聚焦於在 B-IoT 中部署區塊鏈節點的邊界伺服器與物聯網裝置的架構，討論三種可行的資料匯集設計樣式。另一方面，也針對物聯網的存取控制議題進行研究，並提出基於區塊鏈的使用者自主管理存取機制與其設計樣式。最後，本論文以「智慧海運」系統為案例，實作實證系統並進行可行性分析，以引導開發人員縮短開發時間，並建置出具高品質與安全性的系統。

關鍵字：區塊鏈、物聯網、邊界運算、設計樣式、使用者自主管理存取

DOI:10.6814/NCCU202101738

Abstract

There is an increasing number of software developers that take advantage of blockchain technology in their projects. Meanwhile, IoT (Internet of Things) is recognized as one of the most promising application domains for blockchain technology due to the highly distributed and extensible nature of blockchain. When collecting data in the blockchain-driven IoT services (B-IoT), the security, throughput, and cost of the data are highly affected by the underlying design strategies of the communication and interaction mechanisms. Besides, traditionally IoT systems rely on centralized access control services. However, the more reliance on the centralized access control mechanism, the lower availability, and scalability of the system can be. On these grounds, the objective of this thesis is two folds. First, the design issues of data collection among the edge server and IoT devices in the B-IoT system are investigated. Then, this research also suggests a decentralized access control approach for B-IoT based on UMA (User-Managed Access). Finally, the findings are presented following the design pattern format to make them reusable by other developers. To explain how these patterns work, this thesis also introduces an "Intelligent Refrigerated Shipping Containers" scenario. Moreover, the prototype is implemented based on the proposed patterns to demonstrate the feasibility. Also, several experiments are conducted to evaluate the performance of the system. The results show that the proposed patterns are feasible and are able to realize a decentralized access control within a reasonable cost of response time.

Keywords: Blockchain, Internet-of-Things, Edge computing, Design pattern, User-Managed Access

DOI:10.6814/NCCU202101738

誌謝

光陰似箭，兩年的政大生活就要接近尾聲了，而不久的將來就要出國攻讀雙聯學位，回憶一年多前，我還是一個跨系來唸資科系的門外漢，沒想到在短短不到的兩年時間，可以順利完成畢業論文，並獲得被理學院推薦為斐陶斐會員的殊榮。

首先，非常感謝我的指導老師 廖峻鋒 教授。老師總是像個大學長一樣，細心的指導。無論是研究上、生活上，甚至是做人處事的道理，無不受啟發與影響。更在我的研究所生涯中，指導我發表國內，甚至國際研討會的論文，啟發自己在學術研究上的能力。在此對廖老師至上最深的謝意。

再者，承蒙各位口試委員 台灣海洋大學資工系 馬尚彬 教授，臺灣科技大學電機系 陸敬互 教授，在口試期間提出諸多寶貴的意見，幫助本論文更加完備，學生由衷感謝。

在加入 SE-Lab 的期間，特別要感謝實驗室的學長姐建哲、子翔與稜惠，讓我剛加入實驗室時，很快就能融入實驗室的氛圍，與大家打成一塊，並給我很多關於研究與技術上的指導。也要感謝心茹默默的給我打氣，讓我不管再大的困難都不會輕易的被擊倒。另外感謝昀臻、緯政和宇凡一起為論文打拼、勉勵和相互關懷，在我失落的時候獲得不少溫暖。

最後，要感謝我的家人，對我不求回饋的付出以及資助，在我成長過程中給我無比的關愛，讓我能無後顧之憂的在異地打拼。

在此向所有幫助我的朋友們致上最深的謝意。

DOI:10.6814/NCCU202101738

目錄

DOI:10.6814/NCCU202101738

DOI:10.6814/NCCU202101738

DOI:10.6814/NCCU202101738

圖目錄

DOI:10.6814/NCCU202101738

DOI:10.6814/NCCU202101738

表目錄

DOI:10.6814/NCCU202101738

第1章 緒論

1.1 研究背景

區塊鏈是數位加密貨幣系統中最核心的技術，也是比特幣[1]的基礎。它被視為擁有不可篡改性、透明性與可追溯性的去中心化平台。除了分散式帳本外，在特定區塊鏈平台，如以太坊 (Ethereum)，開發人員亦能在鏈上運行能夠用來驗證與執行交易的程式邏輯，稱為智能合約 (Smart Contract)[2]。近年來，區塊鏈技術發展出許多面相之應用。其中，物聯網 (Internet of Things, IoT)與區塊鏈整合服務，又稱 B-IoT (Blockchain-driven IoT services)受到相當大的矚目。物聯網為大量具有計算能力與連網能力的裝置建構而成的系統[3]，裝置之間的資料傳輸不需要經過人 (human)，就能達成資源共享。然而，物聯網在實際運作上面臨有許多挑戰，如缺乏隱私性、高成熟度的商業模式以及高維護與傳輸成本等，使得大多數的物聯網技術僅應用於高價值的案例 (如：醫療、國防與太空產業)。IBM 的 Brody 等人[4]提出「裝置民主化」的概念。具體來說，在未來，裝置可以直接透過去中心化系統提供的去信任點對點訊息傳輸 (Trustless p2p messaging)，來達成安全資料共享。而區塊鏈技術的導入，不僅能提供物聯網系統一個健全與高擴充性(scalability)的底層共識技術，更能提供裝置間安全的協作與交易處理環境，有利於物聯網實現高流通與透明的資料共享平台，降低信任成本，並可促成體的資產(如：物聯網裝置提供的服務)數位化。

然而，由於區塊鏈技術還在快速發展的階段，許多設計的議題還尚未成熟。到目前為止，要建置一個高品質與高去中心化的 B-IoT 系統有相當大的困難度。

DOI:10.6814/NCCU202101738

在開發 B-IoT 時，因為系統的特性、開發的預算或與既有系統的相容性等設計限制，會導致開發人員在引入區塊鏈技術時，對區塊鏈節點部署的策略有所不同，進而導致系統的去中心化程度也隨之不同。從設計觀點來看，當直接參與區塊鏈網路的 B-IoT 物件越多，則去中心化的程度越高。在理想上，為了充分利用區塊鏈的優勢，所有性能低的物聯網裝置 (Device)理應要成為一個區塊鏈節點。但由於區塊鏈的運行需要消耗大量計算 (區塊驗證)與儲存 (區塊資料)的資源，因此，在實作上，Device 通常不會部署區塊鏈節點或透過轉接至邊界伺服器 (Edge)、雲端伺服器 (Cloud)的方式與區塊鏈整合，導致間接降低了系統使用區塊鏈的優勢。不過，近年來 Device 直接參與區塊鏈網路的可行性提升，主要由於以下因素：(1)隨著 5G 技術的成熟，網路的覆蓋率與傳輸速度將大幅增加[5]；(2)硬體隨著晶片製程技術的提升，增加了效能與降低了資料儲存成本；(3)區塊鏈輕節點 (Light Client) [6]技術的提出，性能不足的裝置可不必同步所有區塊資料與參與交易驗證即可部署區塊鏈節點。基於上述理由，開發人員設計去中心化程度較高的 B-IoT 系統的條件越來越成熟。

即便如此，開發人員在實現 B-IoT 時，仍會面臨不少架構與設計決策需要審慎選擇。例如，不同的區塊鏈節點部署策略或裝置之間傳輸方法都會對 B-IoT 系統的效能造成相當大的影響[7]。因此，樣式 (Pattern)的提出有助於幫助開發人員以更有效率的方式[8]，解決軟體架構層次常面臨到的問題，並可以在面臨到類似或重複的問題時，減少開發時間並提升系統的品質。到目前為止，學術上已有一些針對區塊鏈樣式的研究[9-12]。但目前文獻較少對「物聯網與區塊鏈整合時的裝置之間區塊鏈節點的部署策略與傳輸方式的選擇」進行有系統且深入的解析討論而產出的樣式。其中，Liao 等人[13]針對區塊鏈節點在 B-IoT 中各個元件部署的可行性，整理出四種 B-IoT 的架構風格 (Architectural style)，包含 *Fully*

DOI:10.6814/NCCU202101738

Centralized、*Pseudo Distributed Things*、*Distributed Things* 與 *Fully Distribute*。在四種架構風格中，*Distributed Things* (圖 1)為平衡 B-IoT 中的資源限制與區塊鏈節點的覆蓋率中最可行的風格，且在輕節點技術提出後，未來將十分可能成為建置 B-IoT 系統時，最合適的架構風格。基於上述理由，本論文將聚焦在 *Distributed Things* 的 B-IoT 環境下，討論設計 B-IoT 時相關的議題。

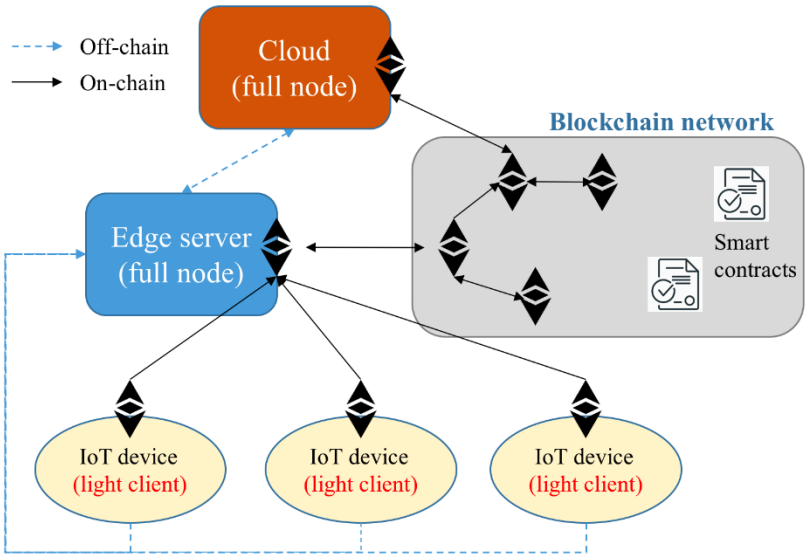

圖 1：Distributed things 架構圖

1.2 研究動機

在 B-IoT 的議題中，本論文聚焦於兩個重點，分別為「資料匯集 (data collection)」與「存取控制 (access control)」。在物聯網系統中，裝置與裝置之間的資料匯集樣式是一個開發人員需要深入考量的議題。當資料價值高或因為法律、其他規定等因素而不能遺失，需確保傳輸過程中與傳輸後的高度安全性時，根據過往的物聯網系統開發經驗指出，需要透過高度複雜的軟體安全機制來達成，但現有物聯網平台欠缺確保資料不受竄改或永久保存的機制，若要自行實作，開發人員需在開發初期花費相當多的時間在解決系統的設計議題與選擇合適的樣式[14]。

DOI:10.6814/NCCU202101738

另一方面，由於未來物聯網裝置的數量大幅增加，根據 CISCO 的預估[15]，2020 年將會有 5000 億個裝置連網，加上近來資安攻擊事件頻傳。因此，安全性議題也是在設計物聯網系統時一個重要的考量層面。其中，存取控制在資訊安全領域中是一個非常重要的概念[16]。在 IoT 系統中，由於 IoT 元件 (component)數量龐大、溝通頻繁以及考量到硬體與網路的限制，獨立的存取控制機制可能造成系統的可維護性 (maintainability)降低、耗費相當的資源並減低系統的安全性[17]。

本研究希望透過整合區塊鏈與物聯網，利用區塊鏈的去中心化特性所帶來的可靠性、不可篡改性與可追蹤性的優勢，來解決上述之物聯網相關設計議題，以下將分別論述之。

(一)、 B-IoT 資料匯集

在 Liao 等人[13]所提出的四種架構風格中，相較於 *Fully Centralized*、*Pseudo Distributed Things*，*Distributed Things* 在與其他兩個架構風格在設計策略上的最大的不同是，在 Device 上部署區塊鏈節點。然而，基於可行性因素，目前的 B-IoT 資料匯集議題大多針對去中心化程度較低的架構風格 (*Fully Centralized* 與 *Pseudo Distributed Things*)進行討論。正因如此，現行有關的 B-IoT 研究與應用的設計中，Edge 與 Device 的資料傳輸方法大多利用應用層協議 (Application layer protocol)，而缺乏直接透過區塊鏈機制來達成資料匯集目的的樣式。而本研究則聚焦於 *Distributed Things* 中，部署區塊鏈節點的 Edge 與部署輕節點的 Device 之間，資料匯集議題進行深入分析與討論。相較於去中心化程度較低的架構風格，Edge 與 Device 之間的資料匯集可以直接透過區塊鏈，以充分利用去中心化所帶來的優勢。

然而，在設計階段仍有很多不同的因素會影響 Edge 與 Device 實現資料匯集

DOI:10.6814/NCCU202101738

的過程中的複雜度與成本 (例如：時間、硬體消耗或交易手續費)。為了設計最合適的樣式，開發人員需要考慮以下三點：

- 資料請求端 (Edge)的主被動角色：當 Edge 需要調用 Device 的特定遠端方法以匯集資料的過程中，Edge 扮演主動角色。具體來說，由於 Edge 需事先擁有 Device 的位址、參數格式傳輸資訊，並確保 Device 中服務的可用性，才能正確地調用該遠端方法。在這種情況下 Edge 與 Device 之間的關係為緊密耦合，只要系統中有任一調用過程失效，很可能造成整個系統停擺。故就系統觀點來看，其擴展性與可靠性下降。從另一個角度來看，若 Edge 的角色為被動時 (Device 主動向 Edge 傳送資料)，單一連線失效對整個系統影響較小。

- 資料的傳輸媒介：B-IoT 的資料傳輸主要分為兩種形式。第一，使用者可以透過鏈上 (on-chain)，向區塊鏈中智能合約發送交易的方式，將資料透過區塊鏈永久儲存，並推播至所有節點；第二，透過鏈下 (off-chain) 方式傳輸，即可減輕或克服使用鏈上傳輸所帶來的限制 (例如：龐大儲存成本與冗長驗證時間)，並將資料透過非區塊鏈的方法來傳輸與儲存。理想上，*Distributed Things* 中裝置元件之間傳輸資料時，利用鏈上傳輸是比較理想的做法。然而，由於不同的傳輸方式可能影響資料不可篡改性、私密性與可追蹤性等。因此，開發人員仍需權衡不同情境會面臨的問題，來決定合適資料傳輸方式。

- 存取控制：為了在匯集資料時確保 Device 所提供資源的機密性或確認 Edge 所訂閱的資料由特定的裝置或個體所發出，必須實作存取控制機制。值得一提的是，目前大多數的區塊鏈平台提供公私鑰機制，可以對發送交易的節點進行紀錄與驗證，開發人員可選擇此種技術在區塊鏈實作安全可靠的的存取

DOI:10.6814/NCCU202101738

控制機制。另一方面，在設計資料匯集樣式時，必須透過評估資料的特性來決定是否需要對特定裝置進行認證與授權。

(二)、 B-IoT 存取控制

如上所指出，在 IoT 領域中，存取控制是一個很重要的議題。在 IoT 系統的設計層面，考慮 Device 的硬體與網路限制，現有的解決方案大多數依賴於第三方的授權機制，如開放授權協定 (OAuth2) (如圖 2)，透過第三方的代理授權，可以減輕 Device 在實作存取控制機制上的負擔。然而，OAuth2 的中心化授權機制可能遭遇到單一節點失敗、駭客攻擊與隱私資料外洩等風險[18]。另一方面，在 OAuth2 中，使用者需要相信中心化組織與其不透明的授權架構，任其代理使用者管理其資源 (或個人資訊)，甚至需要被動接受中心化組織所定義的授權規則 (policy)，再者，中心化組織也可能會利用使用者儲存於資源伺服器的資源加以牟利。如此一來，使用者就失去了自主管理資源與定義授權方式的權利。

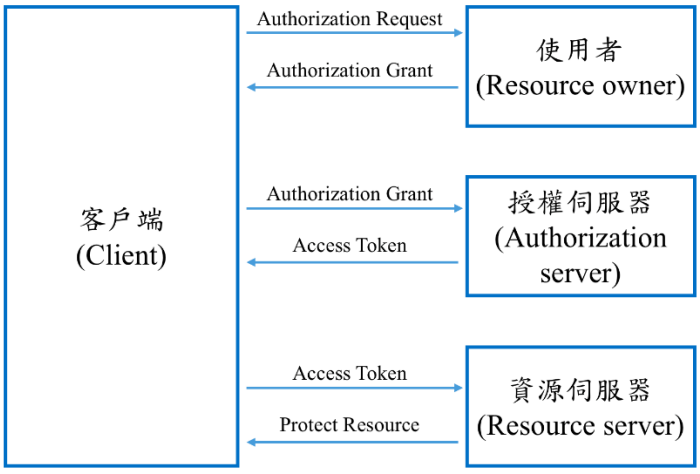

圖 2：OAuth2 - Authorization Code grant type

使用者自主管理存取 (User-Managed Access) [19]，簡稱 UMA，是由 Kantara

DOI:10.6814/NCCU202101738

Initiative 基於 OAuth2 所開發的機制。UMA 的特色是由使用者驅動授權規則(user-driven policies)，並自主管理存取控制 (user management of access control)[20]，以彌補 OAuth2 未定義 Party-to-Party 授權機制之缺口。然而，不管在利用 OAuth2 或 UMA 建置中心化的授權系統時，都可能面臨下列的風險： (1)可用性：OAuth2 或 UMA 授權機制依賴於中心化授權伺服器而面臨單點失敗的風險； (2)公開透明性與可追蹤性：OAuth2 或 UMA 機制缺乏授權流程的公開透明與授權歷史的可追蹤性； (3)可維護性：中心化授權系統一經上線之後，版本更新需要耗費相當大的時間與成本。

區塊鏈被視為一個未來可以管理、控制與維護 IoT 安全性的重要角色[21]。為了解決上述問題，區塊鏈技術將為 UMA 帶來以下幾點優勢。第一，區塊鏈的去中心化特性。UMA 所提出的核心概念為使用者自主管理存取，區塊鏈能協助使用者不需要依賴第三方中心化系統，透過去中心化的機制確保授權資料與規則的不可篡改性與可追蹤性；第二，區塊鏈導入能提升系統的可維護性。使用者能使用區塊鏈之原生機制，例如：以智能合約實作驗證授權機制，參與區塊鏈的節點皆能透過自主管理的智能合約進行代理驗證與授權請求，而不需要透過中心化的伺服器制定授權規則。具體來說，當中心化系統中已授權規則或是使用者資訊需要更改時，複雜度與成本相當高；第三，區塊鏈的原生身分與分散式帳本機制。所有區塊鏈的參與者能夠在區塊鏈中擁有獨立的私鑰，如此一來，就能在區塊鏈中利用私鑰認證以取得授權。另一方面，區塊鏈系統中提供的安全金流層(security billing layer)能夠透過區塊鏈達成點對點的交易，促使資源或是資料共享能夠在區塊鏈生態系中形成市場機制；最後，區塊鏈的公開透明性。每個區塊鏈參與者皆能追蹤比對已部署的智能合約內容與真實性，並追蹤授權歷史，智能合約的擁有者不能隱瞞其透過區塊鏈實作的授權規則或服務內容。然而，目前在學

DOI:10.6814/NCCU202101738

術上缺乏針對 UMA 與 B-IoT 結合的使用者自主管理存取機制的深入討論。因此，針對以上的議題進行可行性探討與設計將會成為本論文的一大貢獻。

1.3 研究目標

基於上述缺口，本論文將深入討論基於 B-IoT 系統中 *Distributed Things* 架構風格下，資料匯集以及使用者自主管理存取的兩項議題。在「資料匯集」層面，本研究在具備區塊鏈輕節點的物聯網裝置與全節點的邊界伺服器之物聯網環境前提下，基於區塊鏈以太坊平台為實作範例，針對物聯網裝置與邊界伺服器之間資料匯集機制的實現，整理三種可行的設計樣式；在「使用者自主管理存取」層面，本論文基於 UMA 流程的規格[19]，發展出結合區塊鏈與 UMA 的「區塊鏈支援下使用者自主管理存取」 (Blockchain-assisted User-Managed Access，簡稱 B-UMA)機制的規格與最佳實踐。此機制透過智能合約，代理使用者進行第三方所發出的資源請求之驗證與授權。另一方面，本論文針對 B-UMA 中的智能合約設計時常見的問題，整理三種設計樣式。

為驗證設計的可行性，針對本論文的資料匯集設計樣式與 B-UMA 機制，以「智慧海運」為案例，實作虛實整合 (Cyber-Physical)的原型實證系統。本系統以 Node.js 為軟體開發平台，配合 JavaScript 語言進行開發；區塊鏈部分使用以太坊，搭配以太坊客戶端軟體 geth (Go-ethereum) [22]；後台系統與 API 伺服器採用 Koa2 框架與區塊鏈標準 API (web3.js 函式庫)等技術實作。最後，將本論文所探討的兩個設計議題，針對實證原型系統分成資料匯集樣式的實驗測試與使用者自主管理存取機制的案例研討，並依據實驗結果對實證原型系統進行可行性驗證與效能及安全性評估。本研究的成果可做為未來系統開發人員設計此類系統時，在不同的情境與條件下的設計決策與評估指南。

DOI:10.6814/NCCU202101738

第2章　技術背景與相關研究

2.1　技術背景

2.1.1　區塊鏈與以太坊

以太坊 (Ethereum) [23]是一個以區塊鏈為基礎的去中心化平台，使用者可以在平台上編譯程式碼，更改帳戶 (Account)狀態或與其他帳戶互動。以太坊將比特幣的區塊鏈技術加以擴充，提供以太坊的虛擬機器 (Ethereum Virtual Machine, EVM)來編譯與運行圖靈完備 (Turing-complete)的區塊鏈原生語言 (如：Solidity、Vyper與 Bamboo)，又稱「智能合約」。因此，開發者能以智能合約的形式，在區塊鏈中運行去中心化的應用程序。相較於比特幣，以太坊具備的可擴充性與彈性更適合軟體應用的發展。

以太坊中主要提供兩種節點間的資料傳輸方法。第一，鏈上傳輸機制主要為智能合約原生語言所提供的事件 (Event)機制，支持區塊鏈與鏈下之間的資料傳遞。裝置上所運行的程序 (如：node.js)能夠透過區塊鏈標準 API (如：Web3.js 函式庫)向智能合約發出狀態更新的交易，調用與訂閱智能合約中的 Event。智能合約開發人員可以透過智能合約中提供的 Event 機制，將變化的數值儲存於每筆交易所產生的交易收據 (Transaction Receipt)的 Event log 中，並隨著交易收據透過區塊鏈去中心化帳本儲存。每當 Event 被調用，則立即通知透過區塊鏈客戶端 Event 訂閱的程序。第二，以太坊也提供鏈下的點對點傳輸協議，如：Whisper [24]與 w3f (Messaging for Web3) [25]，由於 w3f 的目前開發還尚未成熟，本研究中以

DOI:10.6814/NCCU202101738

討論 Whisper 為主。Whisper 是一種運行於以太坊通訊協議 (ÐΞVP2P)框架之上的 P2P (peer-to-peer)訊息傳遞服務。相較於 Event 機制，Whisper 協議提供了具安全性與私密性的鏈下傳輸方式。開發人員可以利用 Web3.js 函式庫所提供的 Whisper API，在以太坊節點之間建立 Whisper 私密通道來傳輸資料。

2.1.2 區塊鏈客戶端與同步模式

在區塊鏈網路中，區塊鏈客戶端 (Blockchain client)為用來實現區塊鏈對等網路與相關規範的應用程式[26]，區塊鏈參與者透過區塊鏈客戶端與其他的客戶端通信。在以太坊網路中，常見的客戶端如表 1 所示。區塊鏈開發人員可以透過 Geth 或 Party 運行以太坊客戶端來參與以太坊公有鏈或測試鏈，另外也可以透過上述兩種以太坊客戶端來架設私有鏈;若系統開發處於本地端建置或智能合約測試階段，則可透過名為 Ganache [27]的私有以太坊客戶端來進行去中心化應用程式 (DApp)的測試與開發。

在以太坊客戶端中，基於以太坊的共識協議 (Proof-of-work, PoW) [28]，主要提供兩種節點模式，分別為全節點 (full node)與輕節點 (light client)。全節點提供完整的區塊鏈功能(如區塊驗證、同步完整區塊資料、離線查詢區塊狀態與發送交易等)。另外，在區塊鏈中的區塊驗證，又稱挖礦 (mining)的全節點數量決定區塊鏈網路的安全程度。然而，運行全節點需要消耗大量的計算能力與儲存資源(完整的節點與公有鏈同步時，需要下載超過 80GB 的資料[26])。因此，在現實生活中，為了解決越來越多裝置欲使用區塊鏈，卻因為性能上的限制而無法部署全節點的狀況，以太坊提供了輕節點同步協議 (Light Ethereum Subprotocol, LES)，客戶端只需要下載包含交易 hash 的區塊標頭 (Block Header)，而不需要參與區塊驗證，就能參與區塊鏈網路與實現部分全節點的功能 (如：發送交易、管理區塊

DOI:10.6814/NCCU202101738

鏈私鑰與線上同步區塊資料等)。另一方面，若輕節點想要訂閱特定的 Event，可透過區塊標頭所包含的 Bloom Filters 查看是否有對應合約地址與 Topic 的 Event。若配對成功，則下載該 Event 之詳細資料。總結來說，由於輕節點的提出，計算與儲存能力小的裝置直接參與區塊鏈的可行性大幅提升。

表 1：以太坊客戶端列舉

客戶端名稱	開發語言	所支援的區塊鏈種類
Go-Ethereum (Geth)	Go	公有鏈、測試鏈與私有鏈
Parity	Rust	公有鏈、測試鏈與私有鏈
Ganache	JavaScript	測試用，僅能單機運行

2.1.3 設計樣式

設計樣式 (Design Pattern)一詞與概念源自於建築界[29]，被應用於描述房屋的建造。爾後，Erich Gamma 等人[30]在 1994 年將設計樣式應用於軟體工程領域。設計樣式在軟體工程領域被認為是描述在特定情境 (Context)下，解決設計問題(Problem)的最佳實踐。而設計樣式中所提出的解決方案 (Solution)經過多次的驗證，能使軟體開發者在類似情境下，遭遇相關問題時反覆使用並幫助其釐清問題的脈絡。另外，也可以讓軟體開發者在軟體開發的初期透過選擇合適的設計樣式來引導其開發合適的系統架構，並降低花在軟體設計階段的時間。另一方面，在軟體樣式 (Software Pattern)語言中除了設計樣式外，也有架構樣式 (Architecture Pattern)能夠幫助解決軟體在架構層次的議題。

本研究提出或整理的設計樣式參考 POSA [8]中所使用的樣式描述格式，不同設計樣式會依照其需求，挑選需要的元素來組成樣式描述格式。表 2 列出了本論文所使用的 POSA 樣式元素。值得一提的是，在不同樣式中，不同的樣式元

DOI:10.6814/NCCU202101738

素可能被使用，以充分說明樣式的內容。

表 2：POSA 中樣式元素介紹

元素名稱	介紹
Name	Pattern 以 name 來識別。通常融合 problem 與 solution 或取自 solution 中的特性，可以幫助讀者更易於理解
Context	利用整理 problem 可能發生的情境來界定 problem 可能發生的範圍。雖然 Context 不能列出所有可能的情境，但至少能提供重要的指引
Problem	**Problem** 會在提出的 context 中重複發生。另外，problem 也為 pattern 的核心元素，說明主要的設計議題 **Force** 幫助 pattern 具體形塑出 solution 的邊界，並列出幾個需要解決的層面：(1)solution 需要滿足的要求；(2) solution 需要考慮的限制；(3)solution 需要包含的特性
Solution	Solution 提供解決重複發生的 problem 的方法，並盡可能平衡相關的 force。另一方面，solution 以 **structure** 與 **dynamics** 層面來描述 pattern 中不同元件之間的靜態關聯與動態協作
Implementation	引導讀者實作 pattern。此元素可以依照需求採用。並適當提供實作範例 (如：程式碼)
Example	透過舉例來輔助在 solution、structure、dynamics 與 implementation 中沒有被提及的解決方案層面
Known uses	列舉與 pattern 相關的現存的系統
Consequences	整理 pattern 的優劣勢
Related patterns	用來解決相關問題或者能與提出的 pattern 整合運作的 patterns

2.1.4 使用者自主管理存取 (User-Managed Access, UMA)

開放授權 (OAuth2)為目前較受到廣泛應用的第三方授權協議，但此協議並不能涵蓋所有場景[31]。OAuth2 主要是針對 Person-to-Self 的授權，更精確的說，Client

DOI:10.6814/NCCU202101738

與 Resource server 的使用者為同一人。例如：Alice 要使用修圖軟體 (扮演 Client 角色)的服務，故透過雲端伺服器 (扮演 Resource server 角色)將自己存在雲端的相片授權給修圖軟體使用。然而，OAuth2 並未定義 Party-to-Party 的授權機制。換句話說，OAuth2 只支援 Alice 授予某些資源的存取權給特定軟體，但沒辦法授權給另一使用者 Bob 使用。這個問題在物聯網的應用場合中會更加突顯，例如機車的分享租用，很明顯是將資源 (機車)分享給另一位使用者。

UMA 為 Kantara Initiative [32]所開始開發的 Party-to-Party 授權機制，與 OAuth2 的最大差別在於，UMA 規範中考慮了不同使用者之間的相互授權機制，而其規範 (draft)則公開於 Internet Engineering Task Force (IETF) [19]。UMA 由幾種實體 (entities)所組成，其定義如表 3 所示。

表 3：UMA entities 定義表

UMA entities	description
Resource Owner (RO)	資源擁有者。定義其需受保護的資源的授權規則
Requesting Party (RqP)	資源請求方。需要請求 RO 受保護的資源
Client	第三方應用。代理 RqP 以其身分存取受保護的資源
Resource Server (RS)	資源伺服器。此伺服器儲存受保護的資源，並有能力處理對第三方所發出的資源請求
Authorization Server (AS)	授權伺服器。此伺服器代理 RO 保護儲存於 RS 的資源，並以非同步的方式授權 RqP 所發出的資源請求

UMA 為基於 OAuth2 擴充而來，但有別於一般以 OAuth2 實作之授權機制，UMA 的資源授權不再是 RqP 被動同意授權條款。在 UMA 中 RqP 可以選擇性提供有價值的資料，在網路中的所有節點皆能平等地共享資訊。例如，在機車分享租用的例子中，欲租用機車的使用者皆能選擇是否提供其駕照資料。另外，UMA

DOI:10.6814/NCCU202101738

中也標準化了 RS 與 AS 的互動方法，但在 OAuth2 中並未明確定義。Eve Maler[20] 提出 UMA 的幾個關鍵核心理念，以下列舉其中較重要的項目：

- 使用者驅動的授權規則 (User-driven policy)：使用者 (RO)可以自訂受保護資源的存取規則。在網路中，任何人都能透過該規則請求 AS 的授權。
- 支援 claim-base 的資料存取：RqP 在請求受保護資料的過程中，授權方可以要求 RqP 提供更多的證明 (claim)，以驗證 RqP 請求之合法性。其中，claim 可以想像為對於 RqP 資訊的封裝，如：姓名、電話或 email 等。
- 使用者管理的存取控制 (User management of access control)：RO 不需要親自參與驗證與授權過程，而是將授權的規則定義於 AS。另一方面，RO 也能動態的更改規則，並能隨時終止資源的授權。

UMA 主要分為兩個 domain，如圖 3 所示，分為 Protection domain 與 Authorization domain。在兩個 domain 中包含三個 UMA 階段，其中，資源保護 (Protecting a Resources)階段屬於 Protection domain，主要以 RO 為中心，透過 RS 向 AS 註冊被保護資源；取得授權 (Getting Authorization)與資源存取 (Accessing a Resource)階段屬於 Authorization domain，主要以 RqP 為中心，向 AS 請求授權與向 RS 存取被保護資源。而各階段所使用之 token 如表 4，各個階段詳述如下：

(一)、 資源保護

RO 將 RS 註冊至 AS，註冊後 RS 會從 AS 得到 Protection API Token (PAT)。註冊之後，RO 指示 RS 將儲存於 RS 的資源集 (Resource set)註冊至 AS，同時 RO 會在 AS 設置資源集的相應授權規則。

DOI:10.6814/NCCU202101738

(二)、 取得授權

RqP 向 RS 請求資源。RqP 先透過 Client 代理向 RS 發出資源請求，RS 會轉向 AS 取得 permission ticket 並連同 AS 之 reference (如：Url)回傳給 Client，Client 再利用取得的資訊向 AS 取得資源的授權，最後取得 Requesting Party Token (RPT)。

(三)、 資源存取

取得 RPT 後的 RqP 可以透過出示 RPT 向 RS 存取資源。相較於 OAuth2 未明確規範 RS 與 AS 之溝通方法，在 UMA 中，由於 RS 不直接參與授權於 RPT 的驗證，所以明確定義了 RS 收到 RqP 出示之 RPT 後，透過 AS 檢查 RPT 與授權 RqP 存取受保護資源的流程，稱之為 RPT 的內部檢查 (RPT Introspect)。

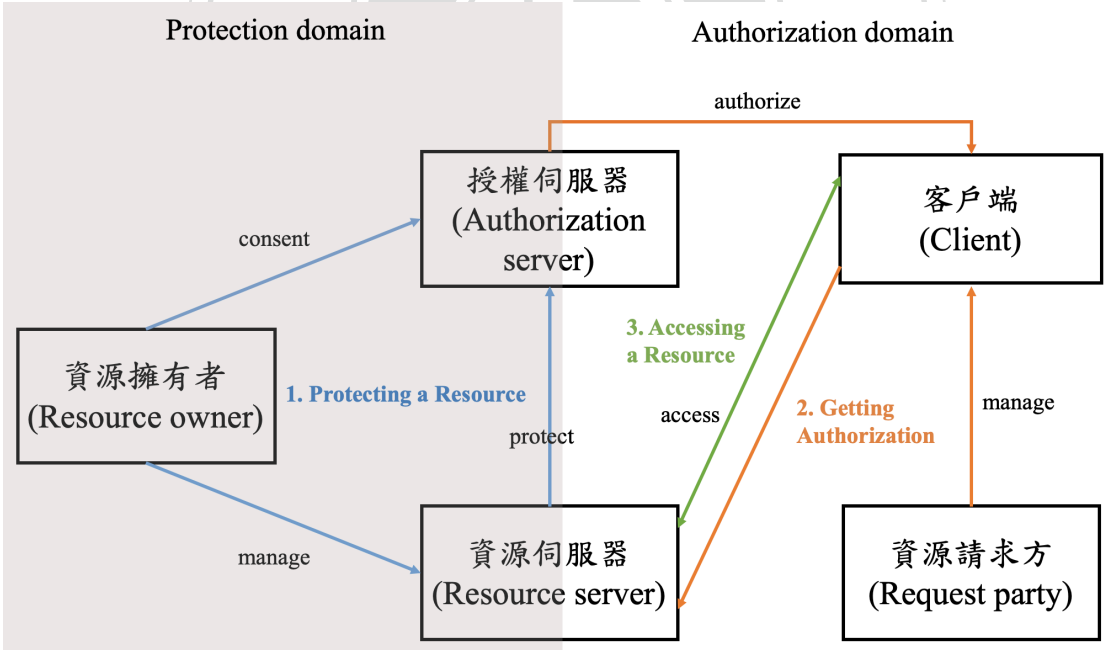

圖 3：UMA 中各階段示意圖

DOI:10.6814/NCCU202101738

表 4：UMA 中所使用各 Token 表

Token	Description
Protection API Token (PAT)	RS 以 PAT 來呼叫 Protection API，並使用 AS 之功能
Persisted Claims Token (PCT)	代表 RqP 在授權過程所提供的 claims。client 可以將其封裝成 PCT，當未來需要再出示 claims 時，client 可以出示 PCT 以取代 claims
Authorization Access Token (AAT)	用以綁定 (binding)client 與 RqP。client 在請求 AS 授權時，可以出示 AAT 用以代表 RqP
Requesting Party Token (RPT)	與 OAuth2 access token 性質相似，為 AS 發布給 client，client 能用以向 RS 存取受保護資源。為了與 UMA 中的其他 token 做區別，故稱之為 RPT

2.2 相關研究

2.2.1 區塊鏈物聯網整合服務 (B-IoT)與樣式

近年來出現不少 B-IoT 相關研究。其中，Sun 等人[7]以電動車電池交換系統為例，提出了 Rich-thin-clients 的物聯網解決方案。研究中比較在物聯網裝置 (thin-clients)中透過部署節點但不執行挖礦與使用 API 轉接至遠端伺服器(rich-client)節點以整合區塊鏈的兩種方式。結果顯示後者在效能上表現較好，但安全性相對較前者低，而反之亦然；Özyılmaz 等人[33]提出將資料儲存於以太坊去中心化資料庫 (Swarm)並將其 hash 記錄儲存於區塊鏈中，以利未來做資料完整性驗證，來解決區塊鏈資料儲存成本高的問題。此外，其研究也建議將物聯網裝置部署輕節點，使物聯網裝置能夠直接參與區塊鏈，以增加安全性；Liao 等人[13]針對物聯網系統中不同的節點部署策略，提出四種架構風格 (architectural styles)，並基於「節點的分佈」、「商業邏輯與資料的區分」與「虛實整合的機制」檢核研究中提出的架構風格。其中，當節點與商業邏輯越分散，去中心化程度則越高。上述

DOI:10.6814/NCCU202101738

研究皆針對 B-IoT 中節點部署策略進行分析探討，但並未針對 B-IoT 運行的細節進行深入討論。

到目前為止，在區塊鏈領域中針對 B-IoT 樣式的整理仍非常有限。為了幫助區塊鏈開發人員解決不同情境下開發區塊鏈應用程式時常面臨到的問題，Wöhrer 等人[34]針對 Solidity 提出數個智能合約設計樣式；Eberhardt 等人[12]則聚焦於整理將區塊鏈應用中邏輯運算與資料儲存從鏈上移至鏈下處理之架構樣式。值得注意的是，其研究提出了將鏈下與鏈上整合而不失區塊鏈核心價值的方法。其中，鏈下簽名樣式 (Off-Chain Signatures Pattern)利用資料傳輸雙方建立鏈下 P2P 傳輸通道，私下進行交易訊息的交換，直到其中一方將最後的交易狀態更新至智能合約為止。此樣式既降低了鏈上傳輸的成本，也增加了傳輸的私密性；XU 等人[11]整理了 15 個區塊鏈應用相關的樣式，包含鏈上和鏈下的互動樣式，資料管理樣式與安全樣式等，並針對每個樣式提供相關的案例與使用場景，分析其優勢與限制。本研究則有別於上述研究的樣式，針對 B-IoT 中邊界伺服器與物聯網裝置間的資料匯集進行設計樣式的提出與整理。

2.2.2 區塊鏈與存取控制

隨著物聯網裝置的數量大幅增加，在網際網路中，提供服務的物聯網裝置的安全性也成為重要的議題。另一方面，近年來，第三方驗證授權機制，如 OAuth2、UMA 與 OpenID [35]等，在商業應用上蓬勃發展，其安全性與可靠性也隨之提升。Cirani 等人[36]提出整合 IoT 場景並以 OAuth2 協議為基礎的授權服務架構「IoT-OAS」，該研究中指出，由於物聯網裝置有網路、計算能力與電量等限制，加上儲存私密資料的隱私性問題，因此在物聯網裝置中實作安全與授權機制的可行性與效益較低；Cruz-Piris 等人[31]則以 UMA 為基礎來實作雲端與物聯網整合系統

DOI:10.6814/NCCU202101738

中存取控制機制，並選擇 Message Queuing Telemetry Transport (MQTT)作為資源存取過程中，資源伺服器與提供服務的物聯網裝置間訊息交換的傳輸協議。在該研究中，將 MQTT 中的 Topic (特定種類傳輸的主題)，整合至 UMA 中受保護資源內容。值得一提的是，在該研究的資料匯集階段，資源伺服器藉由已註冊的 Topic，向物聯網裝置進行資料匯集。基於上述研究，透過與第三方的授權機制整合，不僅能使得物聯網開發人員更專注於服務的開發，也增加了整個物聯網系統的彈性與可維護性。

近年來，隨著區塊鏈技術的發展，利用區塊鏈系統去中心化的特性，解決中心化授權系統所面臨的風險，成為區塊鏈與異質性技術結合的研究領域中，受矚目的一環；Ourad 等人[37] 認為若物聯網裝置依賴 OAuth2 來實作第三方驗證與授權機制，雖然降低了在物聯網裝置實作存取控制機制的成本與提升安全性，但仍會有面臨單一節點失敗等風險。因此，該研究設計了一個能夠透過智能合約驗證資源請求方並提供授權的機制，增加了存取控制系統去中心化的程度。Almadhoun 等人[18]則考慮到物聯網裝置的效能限制，將物聯網裝置基於區塊鏈的驗證與授權服務轉接到霧 (fog)節點；Siris 等人[38]基於 OAuth2 協議建構出一個以區塊鏈為基礎建設的去中心化授權平台。值得注意的是，該平台結合區塊鏈自身的交易清算功能，將區塊鏈中金流設計為授權機制的一部分，資源擁有者可以透過分享其私人的資源而獲得相應的報酬；Tapas 等人[39]則整合區塊鏈與開放式雲端計算平台 Openstack 中 Keystone 身分認證機制，物聯網裝置擁有者能夠透過智能合約紀錄資源授權流程，並藉由智能合約判斷相關角色是否能存取特定資源下的資源子集。另一方面，該研究提出了多層級委派的概念 (multi-level delegation system)，高層級的使用者可以藉由智能合約將自身的權限委派(delegating)給低層級的使用者。在該研究中，區塊鏈的導入增加了原先 Keystone

DOI:10.6814/NCCU202101738

授權的彈性，並提升授權驗證系統的可靠度。

由上面的探討可知，雖然 UMA 可彌補 OAuth2 欠缺 Party-to-Party 授權機制的問題，且已有研究透過區塊鏈的導入，提升驗證機制的去中心化程度。但現在基於區塊鏈設計的存取控制解決方案仍缺乏對使用者自主定義授權規則與資源請求方自主選擇提供相關授權資料的彈性。再者，由於區塊鏈中儲存或發布的資訊皆為公開透明，區塊鏈的參與者皆能取得智能合約中的重要資訊，故提供安全的授權資料之驗證機制會是基於區塊鏈設計的存取控制系統中重要的議題。然而，目前尚缺乏基於區塊鏈結合 UMA 的相關研究或實作的提出。本研究著眼於此一缺口，設計了一個結合區塊鏈與 UMA 的使用者自主管理存取機制，並專注於設計相關規格、流程與智能合約的設計樣式，促使往後的相關研究能夠參照本論文提出的方法做更進一步的發展與探討。

2.2.3 智慧海運案例

本論文參考「智慧冷藏運輸集裝箱 (Intelligent Refrigerated Shipping Containers)」的應用場景[40]，以此案例作為本論文所提出的不同設計樣式之特定情境的背景，同時以此案例為基礎建置實證系統。在該案例中，海運的貨物以智慧冷藏箱 (ship hauling intelligent “reefers”)保存，並追蹤其環境參數，確保運送過程環境參數合規。然而，當配送「高價物品」時，需要確保資料傳輸過程中的安全性。以下將會詳細說明在本研究案例中，所使用的情境：

在一個海運配送作業中，有兩個合作但缺乏信任關係的組織，分別為貨代公司 (freight forwarder)與船運公司 (shipping company)。當海運配送高價商品 (如：珍貴藝術品)過程中，需要確保環境參數 (如：溫濕度)是否合規與配送過程中的環境狀況 (如：照片紀錄)。配送時，商品將以配有物聯網裝置 (Device)之 reefer

DOI:10.6814/NCCU202101738

保存。此時，貨代公司需要透過 Device 資料來匯集 reefer 中的環境相關資料。

然而，在船離岸時，由於網路訊號較弱，貨代公司需要授權船運公司的船員(crew)透過船上的邊界伺服器 (Edge)來操作 reefer 所配置的 Device 以匯集感測資料，並確保感測資料在匯集的過程中，包含傳輸過程與傳輸完成後，感測資料的安全性 (機密性、完整性與可用性)。在圖 4 中，船員透過 Edge 向 reefer 中的 Device 發送感測資料請求，並將資料回傳 Edge，待 Edge 將資料處理過後，再與 Cloud 同步。

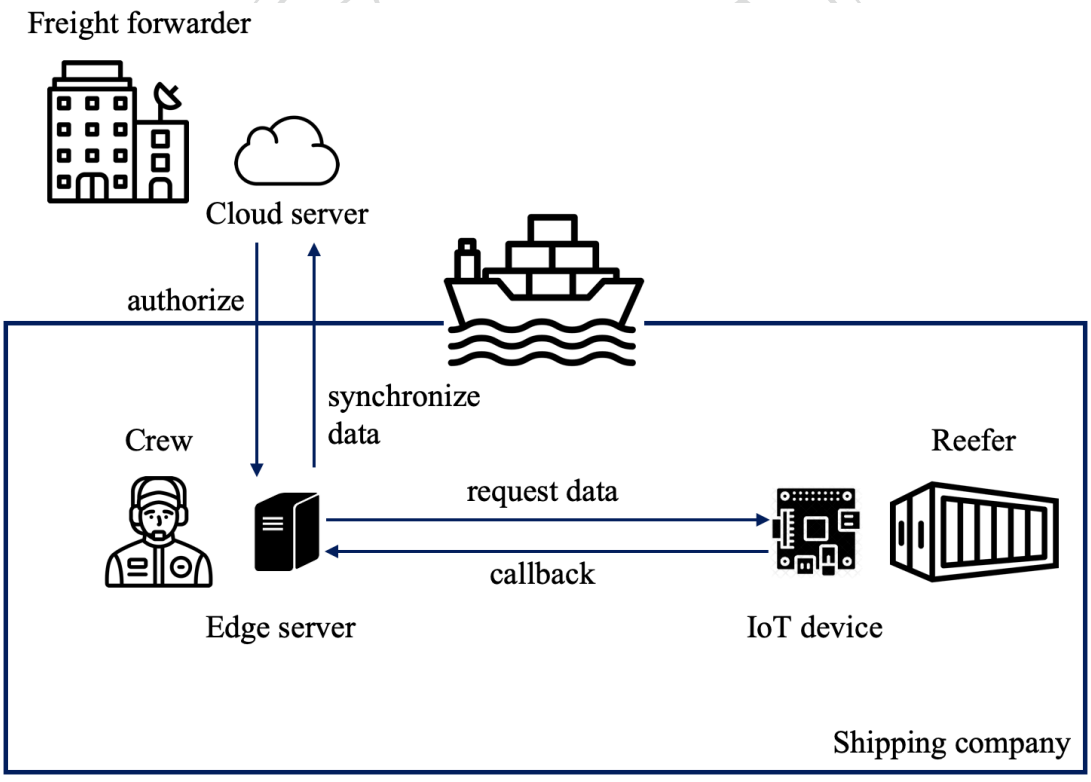

圖 4：智慧海運的角色與系統架構圖

DOI:10.6814/NCCU202101738

第3章 B-IoT 的資料匯集設計樣式

本章將針對在 *Distributed Things*[13]的架構風格下，整理適用於 Edge 匯集 Device 資料傳輸時的設計樣式，並以 POSA 中的樣式描述格式說明問題 (problem)與情境 (context)。之後，以 force 形塑出解決方案 (solution)之範圍，並在每個解決方案中說明樣式內函，提供樣式的示意圖與範例合約。另一方面，在本章節中所討論的資料指的是由於價值高、不能被篡改或因為法律或其他規定等因素而必須確保安全性與權威的資料，英文稱之為 *Accountable Data*。

樣式命名部分，本研究考慮到由哪個物聯網物件開始資料的匯集流程來命名。例如：由邊界伺服器發起的資料匯集流程，該樣式名稱就以「邊界伺服器發起」開頭。而命名也考慮到資料的傳輸方法，若 Device 將資料由鏈上傳輸至 Edge，則以「鏈上資料」方式命名。本章節所提出的三個樣式的命名將依照上述樣式規則命名。本章共提出三種設計樣式，分別為「邊界伺服器發起調用鏈上資料」(On-chain Edge-initiated Invocation，簡稱 OEI)、「物聯網裝置發起推播鏈上資料」(On-chain Device-initiated Provision，簡稱 ODP)與「邊界伺服器發起調用鏈下資料」(OFF-chain Edge-initiated Invocation，簡稱 OFEI)，以下將針對三種設計樣式分別進行整理，各個樣式所提供的範例情境 (Example)將以第 2 章中所提供的「智慧海運」作為情境背景。

DOI:10.6814/NCCU202101738

3.1 On-chain Edge-initiated Invocation (OEI)

- **Context**:

在典型物聯網架構中，當 Edge 需要取得資料時，需要向 Device 發出請求 (request)。例如：客戶端需要透過 Edge Service 所提供之 API，利用應用層協議驅動 Device，使其回傳資料。然而，資料在傳輸的過程中，可能有遭受惡意第三方攔截或竄改的風險。因此，安全的資料傳輸將成為樣式設計的主要議題。

- **Problem**:

在 B-IoT 中，當 Edge 向 Device 匯集資料時，如何確保傳輸的安全性？

Forces:

- 不可篡改性：相較於鏈下傳輸方法，鏈上傳輸確保資料不受篡改。然而，資料透過鏈上傳輸可能由於區塊鏈的共識機制 (如：Proof-of-Work)，造成需消耗更高的儲存成本與更低資料傳輸量。

- Edge 為主動之角色：在此樣式中，Edge 需要先向 Device 發送資料請求，Device 將請求資訊處理過後，再回傳至對應的合約。

- 請求合法性：為了避免惡意的請求或攻擊，傳輸過程必須要驗證向 Device 所發出之請求來自合法的裝置，並確保只有 Device 有權限可以更新相關的資料。如果沒有完善的存取控制機制，基於區塊鏈的公開透明性，所有區塊鏈中的節點皆可調用智能合約的功能。

- 擴展性：在透過鏈上傳輸過程中，Edge 與 Device 會以智能合約為媒介，透過區塊鏈傳遞資料。在 B-IoT 資料傳輸的情境中，Edge 要先透過智

DOI:10.6814/NCCU202101738

能合約向匯集資料的目標 Device 發出請求，之後該 Device 再透過智能合約向 Edge 回傳資料。該情境很明顯地在 Edge 與 Device 之間存在與智能合約的耦合關係。Edge 與 Device 在資料匯集前需要知道與儲存特定智能合約的相關資料。再者，由於 Device 中儲存空間與可維護性低的限制，當有新的智能合約部署時，在 Device 更新智能合約資料的難度較高。因此，上述情況導致了 B-IoT 系統的擴張性低。

- **Solution:**

為了確保資料在傳輸過程中的完整性與不可篡改性，本研究皆以 on-chain 方式傳輸，以充分利用區塊鏈去中心化優勢。在區塊鏈中，缺乏節點對節點之間的資料傳輸機制，鏈上資料傳輸需要透過調用智能合約的功能。因此，如圖 5 所示，節點與節點之間的資料傳輸可以善用智能合約的 Event 機制。

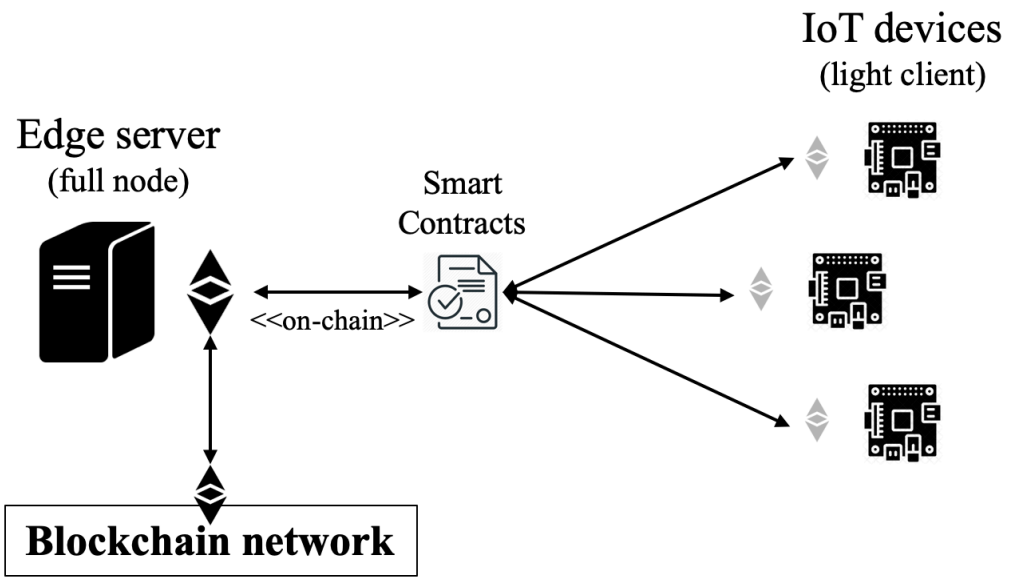

圖 5：OEI 架構圖

DOI:10.6814/NCCU202101738

Structure:

在此樣式中，需要透過智能合約來實現資料請求的註冊、資料存儲與 Event 觸發的機制。如圖 6 所示，本樣式中設計了兩個合約，分別為 RequestRegistry 合約與 Consumer 合約。RequestRegistry 合約有以下功能： (1)提升擴展性：為解決系統擴展性與請求的管理問題，透過 RequestRegistry 合約設計一個集中化的註冊機制 (邏輯上，集中化的註冊機制是中心化的設計。然而，由於該機制透過智能合約實作，需要在每一個區塊鏈節點複製一份，因此，就全域來說仍保有去中心化特性)。如此一來，Device 只要儲存一個智能合約 (即 RequestRegistry 合約)的相關資料； (2) 確保請求的合法性：負責註冊來自其他智能合約的資料請求，並授權只有特定的區塊鏈帳戶 (如圖 6 中的 Device 與 Consumer 合約的帳戶)才可以存取； (3)請求排序：透過標籤 (labeling)的方式，將所有資料請求的排序，以便未來追蹤； (4)記錄資料：透過智能合約提供的 Event 機制，記錄 Device 所回傳的資料，有利未來的追蹤與查詢。

Consumer 合約則代表代理 Edge 發出資料請求的合約。考慮到存取 Device 資料的目的可能不盡相同，例如：物流系統中，代表不同合約的貨物會存放在同一個櫃體中，但每一個櫃體只有一個 Device。因此，本研究以 Consumer 合約代表需要請求同一個 Device 中資料的所有合約，負責接收 Edge 的資料請求。

DOI:10.6814/NCCU202101738

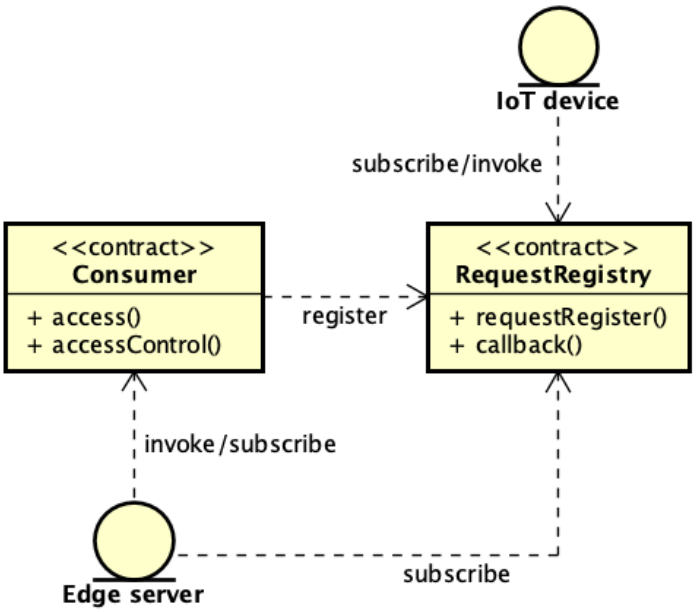

圖 6：OEI 類別圖

Dynamics:

OEI 流程中分為「請求註冊」與「傳遞與記錄資料」兩個階段。請求註冊階段流程如圖 7 所示，此階段的主要目的為 Edge 透過在 RequestRegistry 合約註冊資料請求的方式通知 Device 存取資料的意圖。首先，Edge 會主動向 Consumer 合約發送資料請求交易。Consumer 合約在驗證 Edge 之身分 (區塊鏈帳戶)後，若驗證失敗則會停止交易的進行；若驗證成功，Consumer 合約會向 RequestRegistry 合約註冊資料請求，同時 RequestRegistry 合約也會檢查 Consumer 合約地址是否合法。待交易驗證過後，RequestRegistry 合約會發送 AccessRequest Event (包含欲請求的資料型態與請求的 identifier)。最後 Event 將會由 Event log 記錄到區塊鏈。

在傳遞與記錄資料階段中 (如圖 8)，當 Device 監聽到相關 Event 後，會執行相關資料收集的程序，並將結果透過 Device 上的區塊鏈節點回傳至 RequestRegistry 合約中儲存。其中，當調用 RequestRegistry 合約中 callback fucntion

DOI:10.6814/NCCU202101738

的時候，需要出示 identifier，以使智能合約將資料儲存至相對應請求的資料結構中。最後，當交易驗證過後，RequestRegistry 合約藉由 Event 機制將資料推播回 Edge。

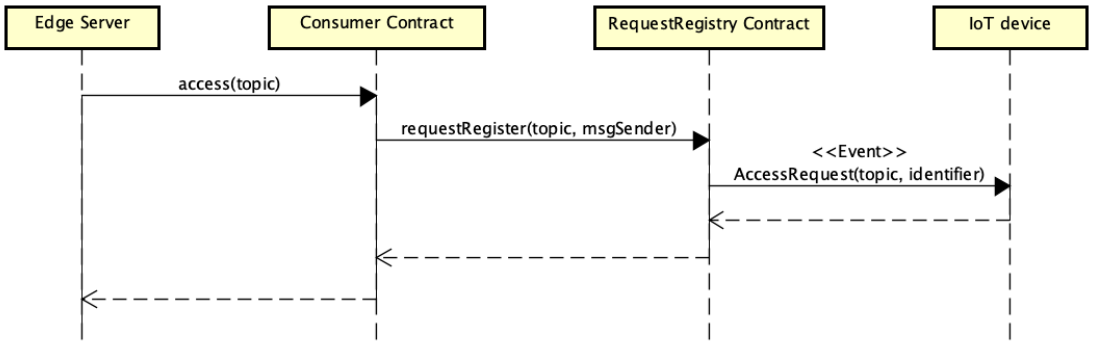

圖 7：OEI 中請求註冊階段循序圖

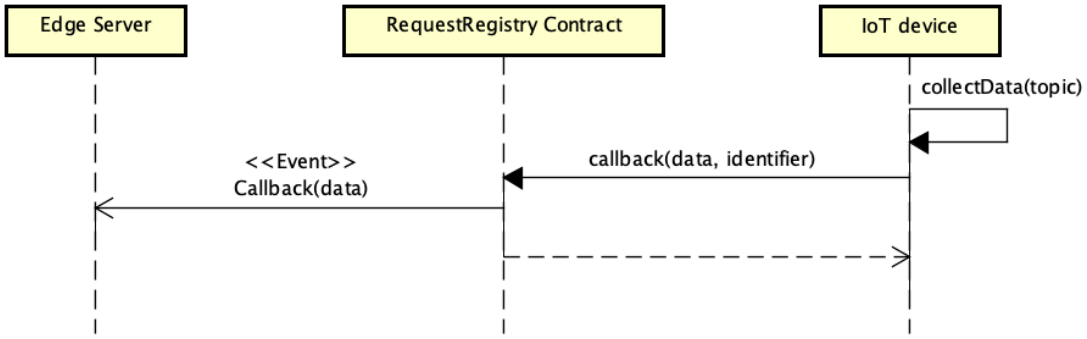

圖 8：OEI 中傳遞與記錄資料階段循序圖

- **Implementation:**

  - 在 Device 與 Edge 上部署節點：在此樣式中，由於雙向傳輸皆經由鏈上傳輸，首先，必須要在 Edge 與 Device 上部署區塊鏈節點。由於 Device 上部署之區塊鏈節點為輕節點，必須要依賴 Edge 中的全節點來同步資料，若 Edge 與 Device 中缺乏信任關係，必須確認 Edge 之身分已經過 Device 之擁有者驗證，才能開始進行資料同步。

  - 部署 RequestRegistry 合約 (圖 9)與 Consumer 合約 (圖 10)：若 Edge 與

DOI:10.6814/NCCU202101738

Device 為互相不信任方，則智能合約由中立第三方部署。例如，在海運系統中，貨櫃中的感測裝置與船上的操作人員為不同組織的情況下，合約必須要有中立的第三方，例如物流公司來部署與管理。另外，由於 Edge 透過 Consumer 合約請求資料後，要透過 RequestRegistry 註冊請求與推播 Event 通知 Device。因此，在部署 Consumer 同時，必須要將已部署 RequestRegistry 合約的 address 註冊至其中。

- 定義智能合約存取權限：為了防止惡意使用者存取智能合約，例如：DDoS 攻擊，必須確保存取智能合約之使用者身分。在區塊鏈中，每個參與者都能以其專屬私鑰代表其身分。因此，在智能合約中，可以事先定義能夠存取特定合約的有效區塊鏈帳戶。在此樣式中，Consumer 合約能由有效的 Edge 帳戶存取；而 RequestRegistry 則能由有效的 Device 帳戶存取。

- Edge 與 Device 雙方事先約定被請求的服務內容：Edge 需要先與 Device 確認 Device 所提供服務的調用方法，以正確的對特定的資源發送存取請求。另外，Edge 也需要知道發送存取請求時需要提供內容 (如資料主題或識別碼)，以避免 Device 回傳多餘的資料。

- 由資料量的大小與內容決定傳輸方法：在此樣式中，Edge 與 Device 的雙向傳輸皆透過鏈上傳輸。然而，若 Device 需要回傳的頻率高且的資料量大時，則 Device 所回傳的原始資料可以利用鏈下傳輸，再將原始資料透過單向雜湊函數 (hash) 雜湊之後所得到的雜湊碼，更新至智能合約，以確保鏈下傳送資料的完整性。

- 預先監聽智能合約所發出的 Event：Device 與 Edge 在資料請求流程開

DOI:10.6814/NCCU202101738

始之前，分別預先開始監聽智能合約所發出的 Event，以即時接收與處理資料的請求。

```
contract RequestRegistry {
    bytes32 identifier;
    mapping(bytes32 => bool) validateQueries;
    event AccessRequest(string topic, bytes32 identifier);
    event CallbackEvent(string data);

    function requestRegister(string memory _topic) public returns(bytes32){
        //implement the access control mechanism and provide an identifier for the
request
        emit AccessRequest(_topic,identifier);
        return identifier;
    }

    function callback(string memory _data, bytes32 _identifier) public {
        //implement the access control that allow specific device to access
        delete validateQueries[_identifier];
        emit CallbackEvent(_data);
    }
}
```

圖 9：RequestRegistry 範例合約

```
contract Consumer{
    address public requestRegistryAddress;

    constructor(address _requestRegistryAddress) public{
        requestRegistryAddress = _requestRegistryAddress;
    }

    //implement the access control function

    function access(string memory _topic) public returns(bytes32){
        //implement the access control mechanism
        RequestRegistry requestRegistry = RequestRegistry(requestRegistryAddress);
        bytes32 identifer = requestRegistry.requestRegister(_topic);
        return identifer;
    }
}
```

圖 10：Consumer 範例合約

- **Example**

在商品配送過程中，由於船公司必須向貨代公司證明物品配送前後的完整度，所

DOI:10.6814/NCCU202101738

以船員必須要在貨物配送開始前或途中，透過 Edge 向 reefer 中的 Device 請求拍攝貨物照片來確保貨物運送中的完整性。在此情境中，除了紀錄每次資料請求的過程與驗證請求船員的身分，也需要確認照片在傳輸過程中與 Edge 收到後的完整性。

上述的情境正好符合 *OEI* 描述之 context 下，所面臨問題。但由於照片資料量過大，不適合透過區塊鏈傳輸。因此，需要以 hash 的方式代替原始照片資料儲存於區塊鏈中。具體來說，Device 將照片雜湊後取得 hash，並利用區塊鏈中資料不可篡改的特性，將 hash 值安全的保存與區塊鏈中並回傳至 Edge；另一方面，原始照片則以安全的鏈外傳輸方式，回傳至 Edge。最後，船公司或貨代公司，能夠透過 Edge 所取得照片的原始資料與區塊鏈上所紀錄(log)的 hash 值，驗證在傳輸過程中，照片是否受到篡改。

- **Known uses:**
  - Electric Vehicle Battery Refueling[7]：以區塊鏈驗證電池交換的過程作為案例。在研究中，Edge 與 Device 雙方，直接透過區塊鏈節點交換相關資料。
  - Chainlink [41]：作為區塊鏈系統中，鏈上與鏈下資料的傳輸與驗證的中介軟體。Chainlink 之鏈下機制將客戶所需要的資料匯集並驗證之後，透過鏈上傳至 Chainlink 合約，再利用 Chainlink 合約將資料提供給客戶的合約。
- **Consequences：**

Benefits:

DOI:10.6814/NCCU202101738

- 資料完整性與不可篡改性：透過鏈上的傳輸，區塊鏈傳輸與儲存機制能夠有效確保傳輸過程中與傳輸過程後的完整與不可篡改性。

- 公開透明性：由於在傳輸過程中所上鏈的資料皆以公開透明的形式保存，所有參與區塊鏈的節點皆能追蹤歷史資料。

Drawbacks:

- 成本：由於區塊鏈上的每筆交易都需要消耗驗證費用，若 B-IoT 中的使用的區塊鏈為公有鏈，則需要消耗現金。當傳輸頻率高且資料量大時，則會導致交易花費的提升。另一方面，由於區塊鏈的共識機制 (如：PoW 或 PoS)，所有鏈上的交易資料將會被複製成多份，故硬體的儲存成本也將會上升。

- 私密性：由於區塊鏈上資料具有公開透明性，所有參與區塊鏈的使用者皆可以追蹤存取。因此，透過鏈上傳輸的資料需要透過其他加密機制來確保不洩漏給第三方。

- 擴充性：由於目前主流的共識演算法 (如：PoW)，礦工 (miner)在驗證交易的時間固定，單位時間的交易量 (transaction per second)不會因為礦工的增加而明顯提升，故資料收集的吞吐量與響應時間相對鏈下傳輸機制將會明顯較低。

- **Related patterns:**

  - *Oracle*[26]：鏈下的 Oracle 機制透過訂閱智能合約中 Event 機制，每當 Oracle 訂閱到特定 Event 並將資料匯集過後，再上傳至智能合約保存。

DOI:10.6814/NCCU202101738

- *Ownership*[10]：智能合約藉由設定使用者的權限，授權特定使用者異動智能合約中資料。

- *Adapter*[30]：將一個客戶端無法匹配的介面轉成可以相容的介面，以消除不同軟體或物件之間兼容性問題。

3.2 On-chain Device-initiated Provision (ODP)

- **Context:**

在特定的情況下，如資料不具私密性或不需儲存或驗證 Edge 身份就能取得，且 Device 會在未來持續匯集資料，並儲存至區塊鏈中。此時若透過 *OEI*， 使 Edge 持續向智能合約發出資料請求會顯得浪費資源。

- **Problem:**

當不需要儲存或驗證 Edge 的資料請求時，如何降低 *OEI* 所造成的資源浪費？

Forces:

- 不可否認與不可篡改性：本研究需要確保資料不被惡意第三方在傳輸過程中與過程後異動，也需要確保資料的傳送紀錄被永久保存。

- Device 為主動之角色：由於 Device 會持續匯集資料，故由 Device 主動向 Edge 推送，以減少 Edge 資料請求之雙向傳輸交易成本。

- 資料主題分類：由於 Device 可能匯集不同類型的資料，若所有資料透過智能合約傳送時，未將資料分類，此時 Edge 必須從節點下載所有資料，造成資源浪費。因此，在資料傳輸前，能將不同主題將資料分類，以減少冗餘資料下載。

DOI:10.6814/NCCU202101738

- 擴展性：在一般 IoT 架構中，Device 在向 Edge 推送資料時，必須要先取得想要訂閱資料的 Edge 或 Edge 中應用程序的相關通訊資料 (例如：IP address)，才能正確將資料推送。然而。若需要訂閱資料的應用程序增加，這種方式會使得系統的擴展性降低。

- **Solution:**

由於 Device 不需要驗證資料的請求者，故可以以推播的方式直接將資料傳送給想要訂閱的 Edge。另一方面，如圖 11 所示，欲確保資料在傳送過程中的不可篡改性，Device 需將資料透過區塊鏈節點傳輸，並透過區塊鏈機制以去中心化方式保存。最後，Device 作為資料擁有者可以利用密碼學中簽章的機制將資料簽名，以確保不可否認性。

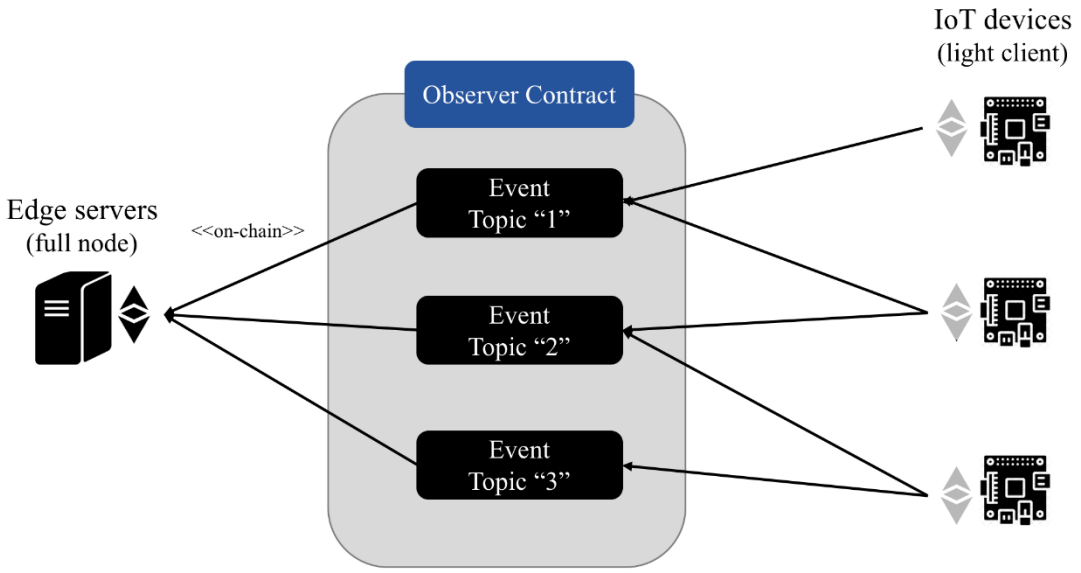

圖 11：ODP 架構圖

Structure:

在此樣式中，以 Observer 合約作為資料傳輸的媒介，該合約用以匯集 Device 持

DOI:10.6814/NCCU202101738

續更新的資料並推播至所有節點。如圖 12 所示，將 Device 視為資料的發佈者(Publisher)，負責主動向區塊鏈發送資料，Observer 合約收到後，透過 Event 機制將 Event log 儲存至區塊鏈並推播至所有節點；Edge 則作為資料的訂閱者(Subscriber)，利用其伺服器所部署的區塊鏈節點訂閱新區塊中的 Event，並將其下載儲存。

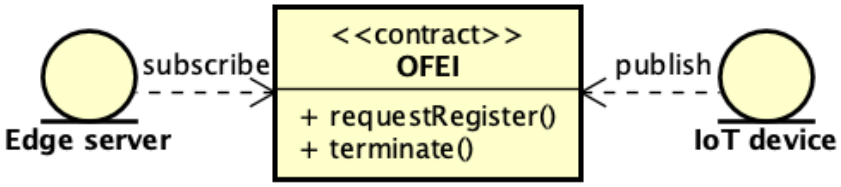

圖 12：ODP 類別圖

Dynamics:

如圖 13 所示，首先 Deivce 匯集完資料後，可以透過專屬的私鑰將欲推播給 Edge 的資料簽名。再透過調用 *Observer* 合約的 function，傳送資料簽章與代表資料的 Topic。另一方面，Edge 則會向區塊鏈訂閱相關 Topic 的 Event。當 *Observer* 合約發送相關 Event 時，Edge 能夠透過 Event 中的簽章，檢查收到資料的完整性與發佈者。

DOI:10.6814/NCCU202101738

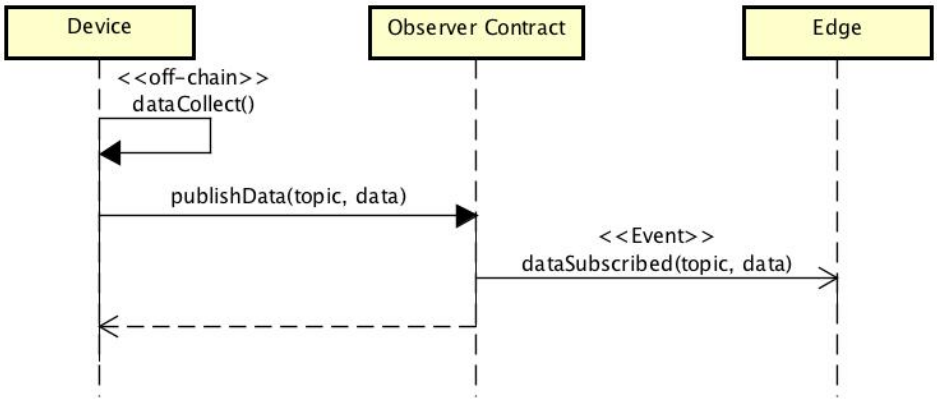

圖 13：ODP 循序圖

- **Implementation:**

  - 部署 Observer 合約：Device 或中立第三方必須將用以接收合約部署至區塊鏈，如圖 14 的 Observer 合約。為使 Edge 可以藉由區塊鏈節點訂閱特定主題的資料，可以利用智能合約 Event 機制將資料分類。其中，Solidity 的 Event 機制提供將 Event 依 Topic 分類之方法。在 *dataSubscribed* 的 Event 宣告式中，可以利用在屬性前加入 indexed 的關鍵字，智能合約在交易驗證後會在 Event log 中存放於 Topic 的區塊，Event 訂閱者可以只從區塊鏈節點下載特定 Topic 的 Event。

  - 事先約定資料傳送的 Topic 與簽章機制：為了確保資料的不可否認性，Device 在傳送資料前需將資料簽名。因此，傳輸雙方需要事先約定實作資料的簽章機制。另一方面，Edge 也需要向 Device 取得欲訂閱 Event 的方法 (如：Event 的 Topic)。

  - Edge 訂閱合約中的 Event：Edge 為了訂閱 Observer 合約中的 Event，需透過區塊鏈標準 API 向區塊鏈客戶端訂閱包含特定 Topic 的 Event。

DOI:10.6814/NCCU202101738

```
contract Observer {

    event dataSubscribed (address indexed senderEoA, string indexed topic, string
          signature, uint timestamp);

    function publishData (string memory _topicName, string memory _data) public {
        //...access control code
        emit dataSubscribed(msg.sender,_topicName, _data,now);
    }
}
```

圖 14：ODP 範例合約

- **Example*:***

Device 在取得的溫濕度與 GPS 資料時，可以同時透過 Device 中的應用程式驗證資料是否異常。例如：溫濕度資料是否超過配送合約規定。由於在此案例中，Device 為主動發送資料之角色，因此，不適合以 *OEI* 或 *OFEI* 實作。另一方面，由於異常資料關係到是否履約，必須確保資料在傳輸過程中的安全性與透明性，因此 *ODP* 為較合適的選擇。

- **Known uses:**

  - MedRec [42]：為一個區塊鏈醫療保健資料管理研究所提出系統原型，該系統透過智能合約發送 Event。來觸發 HER (Electronic Health Records) Manager 通知使用者資料更新，並同步資料庫中健康資料的紀錄。

- **Consequences:**

Benefits:

  - 成本：相較於 *OEI*，*ODP* 不需要 Edge 透過鏈上主動請求 Device 匯集資料，故區塊驗證成本相對降低。

DOI:10.6814/NCCU202101738

- 資料安全性：資料上傳至區塊鏈並透過區塊鏈保存，如此一來能夠有效利用區塊鏈的去中心化之優勢。

- 資料分類：在鏈上資料的傳輸過程中，透過智能合約提供的 Event 分類機制，Edge 中的程序可以訂閱到特定有興趣的資料。

Drawbacks:

- 私密性：基於區塊鏈的公開透明性，當資料會儲存於區塊鏈中，所有參與區塊鏈網路的節點皆可查看資料更動的歷史。

- **Related patterns:**

- *Broker* [30]：在分散式系統中，針對多個元件或異質性平台，透過實作中介軟體 (broker)將訊息發佈方與訂閱方的模組以及時間解耦，增加系統的擴展性。

3.3 OFF-chain Edge-initiated Invocation (OFEI)

- **Context:**

當 Edge 向 Device 發出存取請求的頻率不高，且在需要持續匯集價值低 (零碎)或具私密性的資料之情況下，若資料透過鏈上匯集，效率會降低，且每筆鏈上所發生的交易將會造成很大的金錢與時間成本。

- **Problem:**

在 B-IoT 中，如何以低成本且兼顧安全性的方式，在 Edge 與 Device 之間傳輸資料？

DOI:10.6814/NCCU202101738

Forces:

- Edge 為主動之角色：為了驅動 Device 回傳感測資料並通知資料請求的內容，Edge 需主動向 Device 發出請求。
- 成本：由於區塊鏈的特性，資料若透過鏈上匯集 (如 *OEI*)，將會消耗大量成本，包含資源消耗與區塊驗證花費等。然而，若資料透過鏈上傳輸，則可充分利用區塊鏈中不可篡改性與可追蹤性等優勢。
- 安全與私密性：在私密資料傳輸的過程中，同時需要確保資料安全送達並不洩漏給 Edge 與 Device 以外的第三者。但在實作安全的資料傳輸通道時會增加系統實作的複雜度。
- 擴充性：由區塊鏈的共識演算法的特性 (如：Proof-of-Work)，挖礦節點的增加並不會導致每筆交易所需要驗證處理的時間 (transaction per second)減少。
- 加密金鑰的交換：在 B-IoT 中節點之間若需使用鏈下方法傳輸時，必須使用安全的傳輸機制傳遞資料。在此之前，傳輸雙方必須要事先約定好鏈下傳輸機制 (如：HTTPS)並交換用來加密資料的密鑰。普遍的加密方法有兩種，非對稱式加密方法比較適合在公開的網路上做金鑰的交換，但較對稱式加密方法的效率低。

- **Solution:**

當 Edge 在向 Device 匯集資料時，可以透過鏈上傳輸方式，以確認每一筆資料請求不被竄改，如圖 15 所示；另外，在 Device 之資料回傳的部分，考慮到資料量的大小與鏈上交易成本，透過鏈下的安全傳輸機制也有助於提升系統的擴充性，

DOI:10.6814/NCCU202101738

降低系統開發的成本。

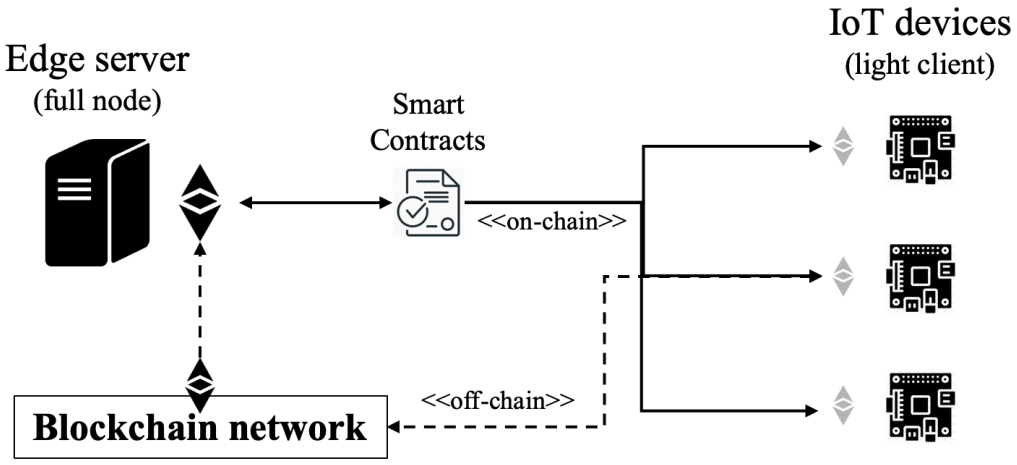

圖 15：OFEI 架構圖

Structure:

為了充分利用區塊鏈的優勢，此樣式以智能合約來註冊並推播 Edge 所發出的資料請求 (fig. 9)。另外，為了限制資料請求的權限，亦可透過合約授權特定區塊鏈帳戶請求的權限。如此一來，將能有效管理存取控制，確保資料匯集的機密性。

鏈下傳輸部分，為了確保資料傳輸過程中的安全性與私密性，傳輸必須使用安全的加密與點對點 (peer-to-peer)傳輸機制。另外，為了建立安全的鏈下傳輸通道，Edge 需要在發出鏈上請求的同時，通知 Device 在鏈下回傳資料時的傳輸方式與格式。同時，在傳輸的過程中，Device 必須確保只有 Edge 能夠識別資料的內容。基於區塊鏈的公開透明性與上述理由，使用公開金鑰密碼系統 (Public-Key Cryptosystem)作為傳輸過程的加密方式，會是有效的解決方法。

DOI:10.6814/NCCU202101738

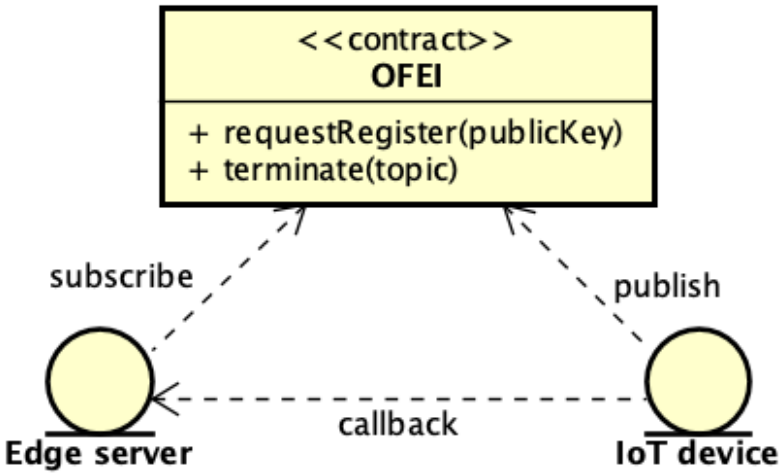

圖 16：OFEI 類別圖

Dynamics:

如圖 17 所示，首先，Edge 向 Device 透過智能合約中發出存取請求，並同時將 Edge 之 public key 作為參數傳送至鏈上儲存。當 Device 訂閱到 Event 後，會開始匯集資料，並持續使用 Event 中的 public key 加密後，透過 Edge 與 Device 之間鏈下預先建立的私密通道回傳至 Edge。

DOI:10.6814/NCCU202101738

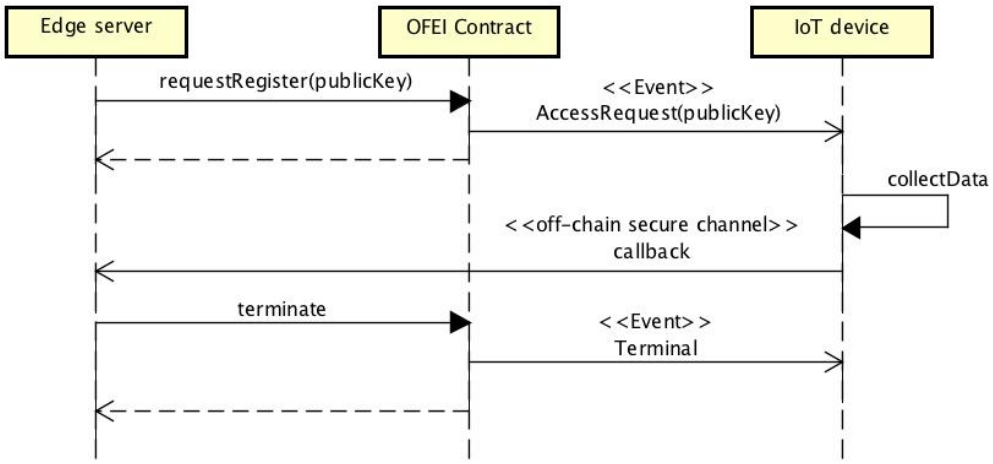

圖 17：OFEI 循序圖

- **Implementation:**

  - Edge 與 Device 雙方事先約定安全的鏈下傳輸機制：在實作與部署智能合約之前，需要選擇能夠以鏈下傳輸但又能確保資料傳輸完整性與私密性的傳輸機制。值得一提的是，區塊鏈生態系所提供的 P2P Messaging protocol (如：Ethereum Whisper, Hyperledger fabric Gossip Protocol[43])，保有區塊鏈去中心化特點並能確保上述之需求。

  - 部署智能合約：智能合約 (圖 18)必須實作存取控制機制，並由智能合約將請求 Event 推播給處理請求之 Device。在此樣式中的註冊 Event 機制，可實作密鑰交換機制 (如公鑰)，使 Device 得以傳送加密訊息。另外，資料匯集完成後，可以刪除合約中已註冊的資料請求。

  - 建立鏈下資料傳輸通道：為了將資料回傳至 Edge，負責匯集資料的 Edge 節點需要和 Device 建立安全的傳輸通道。Device 可以透過解析 Edge 透過鏈上所發出的請求 Event 內容，來匯集並回傳對應的資料。

DOI:10.6814/NCCU202101738

```
contract OFEI{

    event AccessRequest(uint indexed deviceID,string topic,string publicKey);
    event Terminal(uint deviceID,string topic);

    function requestRegister(uint _deviceID,string memory _topic,string memory
       _publicKey) public{
        //implement the access control and  the access request registering mechanism
        emit AccessRequest(_deviceID,_topic, _publicKey);
    }

    function terminate(uint _deviceID,string memory _topic) public{
        //implement the access control and the registred request deleting mechanism
        emit Terminal(_deviceID, _topic);
    }
}
```

圖 18：OFEI 範例合約

- **Example**

在商品配送過程中，為了監控溫濕度與 GPS 等環境參數，reefer 中的 Device 必須隨時回傳感測資料。由於資料回傳頻率高且資料量大，此時，若選擇 *OEI* 樣式，當資料價值低時，透過鏈上傳輸將會顯得浪費。因此，以安全的鏈外傳輸來回傳 Device 所收集的資料，是較為合適的選擇。

- **Known uses:**

  - Status [44]：透過以太坊生態系中的 Whisper Protocol，信息傳輸的節點雙方建立一個私密通道，透過鏈下相互傳輸訊息 (message)。

  - Raiden [45]：在以太坊生態系中的鏈下解決方案。透過節點之間在鏈下建立通道，進行私密通信。當通道關閉之後，再將節點雙方確認的最終結果上鏈。

DOI:10.6814/NCCU202101738

- **Consequences:**

Benefits：

- 傳輸私密性與安全性：資料透過鏈下去中心化點對點傳輸通道傳輸，則可以確保傳輸內容不在網路上被公開。另一方面，可以確保訊息不被惡意第三方攔截。

- 成本：Device 透過鏈下私密傳輸通道，回傳資料，可以減省透過鏈上傳輸所消耗的交易費用。

- 擴充性：由於鏈下傳輸不需要透過區塊鏈共識演算法驗證，故資料傳輸之吞吐量會相對提升。

Drawbacks:

- Device 之身分驗證：在此樣式中，Device 利用鏈下回傳資料至 Edge。因此，Edge 無法有效確認 Device 之身分。若需確認 Device 之身分，可將此樣式與數位簽章或其他驗證機制混合使用。

- 不可篡改性：以鏈下傳輸的資料無法確保在接收方收到資料後，不擅自更改資料內容。

- 可追蹤性：透過鏈下傳輸的資料，由於資料沒有透過區塊鏈去中心化保存，參與區塊鏈的節點無法有效追蹤透過鏈下傳輸的資料內容。

- **Related patterns:**

- *Off-Chain Signatures*[12]：在鏈上最終交易確認前，雙方通信透過鏈下機制建立私密交易通道達成。

DOI:10.6814/NCCU202101738

- *Off-Chain Data Storage*[11]：裝置將無法上鏈 (如：資料量大)的資料，透過鏈下傳輸，再將原始資料透過 reference (如：hash)的方式上鏈儲存，以驗證鏈下資料的正確性。

國立政治大學 National Chengchi University

DOI:10.6814/NCCU202101738

第4章 使用者自主管理存取機制與設計樣式

4.1 設計考量

如同上一章所述，在 B-IoT 中 *Distributed things* 的架構風格環境下，資源請求方(Request party)透過 Edge 匯集 Device 之資料過程中，資源請求方在請求時是否需要經過授權會是選擇設計樣式時的一個重要因素。然而在上一章中尚未針對存取控制進行深入的解析與探討。

由於在 IoT 系統中，資源擁有者 (Resource owner)要將 IoT 的資源(Resource)授權給資源請求方時，資源擁有者需要花費相當大的時間與成本設計安全的授權驗證機制，加上 IoT 設備的運算與儲存資源的限制，因此，透過可信任的第三方來實作安全的授權機制較為合適。UMA[20] 為一個能完整協助資源擁有者進行授權，並基於 OAuth2 的 Party-to-Party 機制。且在授權過程中，能保有 OAuth2 協議中的安全與私密等特性。在授權過程中，授權伺服器(Authorization server)能夠代理資源擁有者，將IoT設備的資源授權給資源請求方。更精確的說，授權伺服器能夠以非同步的方式授權資源請求。此時，資源擁有者不需要隨時在線授權資源的請求。因此，資源請求方所送出的請求也能即時透過授權伺服器取得授權。

在本章中，將會設計應用於 *Distributed things* 的物聯網環境之 UMA 機制。之後整理設計 Blockchain UMA (B-UMA)之使用者自主管理機制時常面臨的問題，提出可行的智能合約設計樣式。

DOI:10.6814/NCCU202101738

4.1.1 B-UMA 面臨的挑戰

以下幾點整理在設計 B-UMA 時，會面臨的挑戰：

首先，由於區塊鏈具有的公開透明性，隱密資訊容易暴露。若 UMA 中所需要使用到的隱密資料 (如：token、claim)，皆經由區塊鏈傳輸的話，在傳輸上需要更注重其可用性與私密性。

第二，由於區塊鏈具有不可篡改性，智能合約部署後，程式邏輯無法再修改。智能合約一經部署過後，就無法再更改其內容 (如：已定義的數值與函式名稱等)。若授權規則要再調整的話，就必須重新部署合約。因此，在設計智能合約時，一方面需要確保合約內容具有高度彈性，例如，將往後需要動態更改的值，以可透過向區塊鏈發送交易來動態更改的變數取代。另一方面，根據 Data Segregation Pattern [34] ，將儲存「邏輯」與「資料」之合約分開實作，使儲存 「資料」的合約不會因為 「邏輯」之變動而需要重新部署，而造成原有的 「資料」需要重新上傳導致浪費資源；相對的，也能有效增加合約之彈性。

第三，鏈上與鏈下整合的複雜度高。在整個 B-UMA 的驗證流程，資料在元件間的傳輸方式會有所不同，需要去權衡哪些過程需要透過鏈上傳輸。區塊鏈僅能對每筆鏈上的交易做驗證，若驗證發生於鏈下，則區塊鏈中的機制無法確保其安全性與正確性，需要透過其他機制來確保。

最後，區塊鏈以太坊 (ÐΞVp2p)與 UMA(HTTP)的通信協定不相容。區塊鏈的傳輸協定有別於傳統 UMA 或 OAuth2 ，並非透過 HTTP 協議傳輸。因此，在裝置之間，為了確保驗證授權過程資料之完整保存而透過區塊鏈機制傳輸時，需要將傳統 HTTP 格式內容轉換為區塊鏈的傳輸協定格式。

DOI:10.6814/NCCU202101738

4.2 授權機制角色

B-UMA 的角色與角色之間的關係如圖 19 所示，整個授權機制以區塊鏈底層技術建構而成。在此機制中，本研究預設資源擁有者 (RO)、資源請求方 (RqP)與 Device (可視為 UMA 中的資源伺服器)皆擁有專屬的區塊鏈私鑰，以作為其身分識別。表 5 中說明了本論文所提出的授權機制與 UMA 的角色差異，主要的設計在於將 UMA 中的授權伺服器 (AS)的授權機制，在區塊鏈中以智能合約方式實作。另一方面，由於此機制中 Device 為 RqP 要存取的資源的對象，因此視為 UMA 中的資源伺服器 (RS)。在傳輸機制方面，則分為鏈上與鏈下，鏈下傳輸為了確保資料在傳輸過程中的安全性，則依照 UMA 之傳輸協議規格，透過應用層與傳輸層的安全加密協議 (例如：SSL)機制傳輸。

在 UMA 中，AS 處理的任務主要有兩大面向：管理 Resource set 與基於 RO 所制定之 policy 代理其授權 RqP 所發出的資源存取請求。但由於 policy 內容可能會動態的變更，且基於區塊鏈的不可篡改性，合約因而需要重新部署。在重新部署時，Resource set 可能需要重新註冊，造成資源的浪費與開發上的困難。因此，在 B-UMA 中本研究將註冊資源與授權之合約解耦，分別設計成 Resource management (RM)與 Authorization (Authz)合約。Edge 能透過 RM 合約來註冊與管理 Device 的資源，並取得對應的識別碼 (identifier)。另外 policy 的制定與授權驗證機制則由 Authz 合約來處理。如此一來，當 Authz 合約重新部署時，就不會影響到已註冊的 Resource set。

DOI:10.6814/NCCU202101738

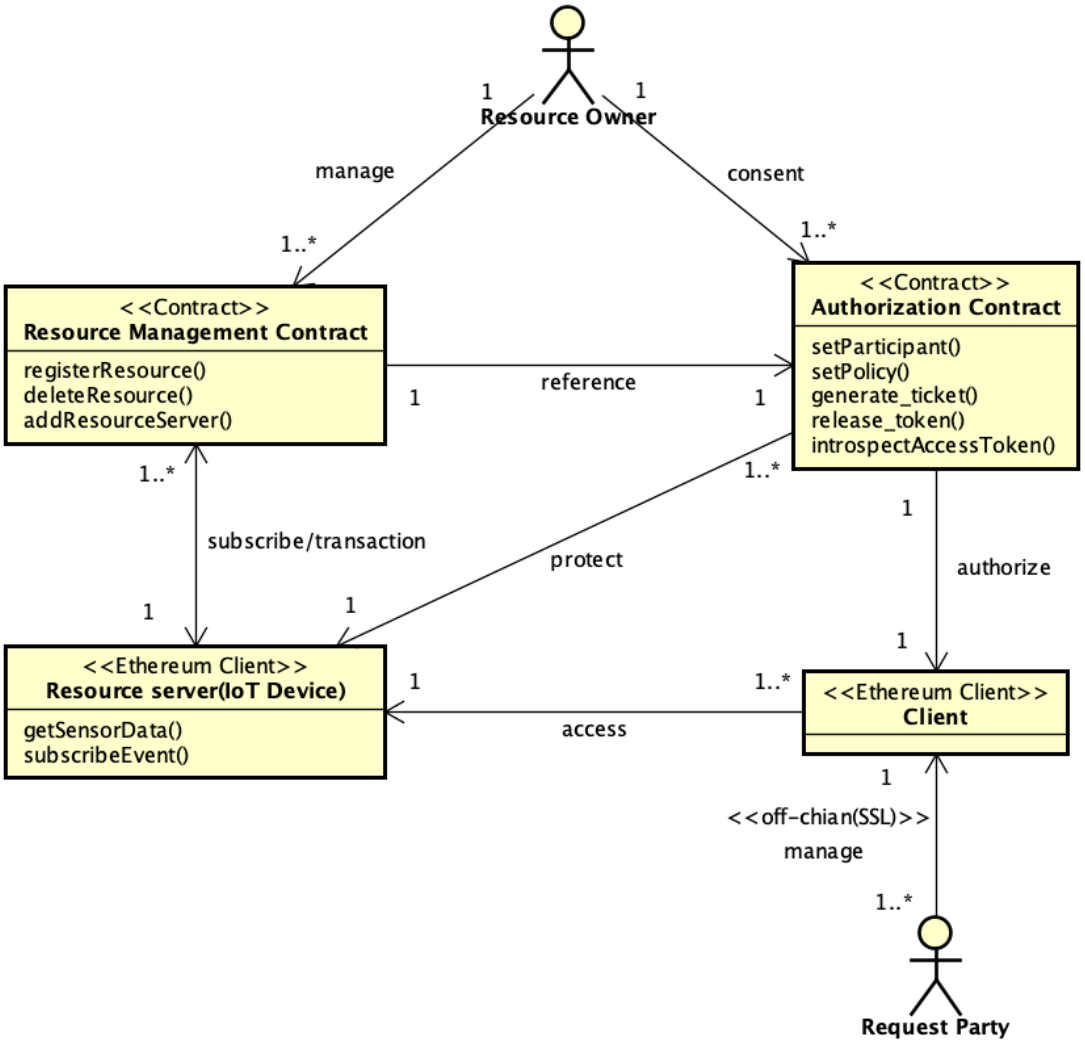

圖 19：B-UMA 系統架構與角色關係圖

表 5：UMA 與 B-UMA 的角色對照

B-UMA 中的角色	UMA 中的角色
Resource Owner	Resource Owner (RO)
Requesting Party	Requesting Party (RqP)
Client (Edge service)	Client
Resource Server (IoT device)	Resource Server (RS)
Resource Management contract	Authorization Server (AS)
Authorization contract	Authorization Server (AS)

DOI:10.6814/NCCU202101738

為了更清楚說明 B-UMA 中之角色內容與其預設之功能，其詳述如下：

- **Resource Owner (RO)：**

RO 為 Device 之擁有與管理者，同時也為透過智能合約來代理其授權第三方請求。RO 同時也為智能合約的擁有者，能夠制定、部署與動態管理智能合約中的 policy。

- **Resource Server (IoT device)：**

Device 可視為 UMA 中的 RS。Device 之存取機制交由區塊鏈中智能合約管理。另一方面，Device 透過區塊鏈輕節點的客戶端與區塊鏈網路互動。由於效能限制，Device 不參與區塊鏈之區塊驗證 (mining)，必須依賴其信任或授權的配置全節點之伺服器 (例如：Edge server)同步區塊資料。

- **Requesting Party (RqP)：**

RqP 透過 Client 所提供服務參與區塊鏈並請求 Device 的資源。另外，RqP 在區塊鏈中以自身擁有的私鑰作為身分以請求智能合約之授權。

- **Client：**

與 UMA 中的 Client 角色相同，RqP 透過 Client 來代理其身分，向 Device 請求資源。另外，由於 Client 並需要與透過鏈上傳輸來獲得智能合約之授權，因此在 Client 的伺服器中，需要透過配置以太坊客戶端來與區塊鏈溝通。

- **Resource management contract (RM 合約)：**

RM 合約負責儲存 Device 資源的參照。RM 合約中可以註冊 Device 中 Resource set 的相關訊息，如：資源名稱，可被存取範圍 (scope)等。一張合約可以註冊多

DOI:10.6814/NCCU202101738

個受保護資源，且每個資源有相對應的識別碼 (identifier)。在後續的流程中，可以直接透過註冊後所取得的識別碼來操作資源。

- **Authorization contract (Authz 合約)：**

Authz 合約可視為 UMA 中的授權伺服器。Authz 合約可以代理 RO，並透過預先設定的 policy 與參照 RM 合約中的識別碼來驗證與授權 Requesting Party 所發出的資源請求。

4.3 B-UMA 授權流程

整個系統設計主要分為三個部分，分別為資源保護 (Protecting a Resource)、取得授權 (Getting Authorization)與資源存取 (Accessing a Resource)。在本節中，本研究將會詳述本研究所提出的 B-UMA 授權流程的細節與考量。

4.3.1 第一階段 - 資源保護

根據之前所論述，Device 可被視為 UMA 中的 RS，RO 不僅為 Device 的擁有者，也能制定智能合約中的授權代理機制。在本研究所提出的 B-UMA 機制中，RO 主要有以下權限：

- 部署 RM 與 Authz 合約：RO 即為合約的 creator 與 owner，擁有修改合約之最高權限。

- 在 RM 合約中增減註冊 Device：Device 在修改或新增 RM 合約中的 Resource set 的資訊時，合約會先識別 Device 之身分 (利用驗證 Device 在發出交易時，用來簽名的私鑰的帳戶)作為此次發送交易是否有效之必要條件。因此，RO 必須先在部署 RM 合約同時，建立一個資料結構存放有效的代表 Device

DOI:10.6814/NCCU202101738

的區塊鏈帳戶。

- 設定與修改 policy：RO 在部署完 Authz 合約後，可以在合約中動態新增或更改 policy，甚至可以在修改合約之架構或授權規則後，部署一份新的合約。

以下將介紹第一階段，B-UMA 系統建置的流程，圖 20 為第一階段流程的循序圖，其詳細流程說明如下：

- 部署 RM 合約：在 UMA 中，RO 將 RS 之需受保護資源註冊於 AS 中。之後 RS 需要透過帶有 Protection API Token (PAT)的 Protection API 來註冊其資源。在此步驟中，RS 如同 OAuth2 中的 Client 角色，AS 必須透過 PAT 來驗證資源註冊請求的合法性。由於在 B-UMA 的環境下，Device 在區塊鏈中已經有足以代表其身分的私鑰，因此，RO 在部署 RM 合約後，能將已知的 Device 在區塊鏈中專屬的帳戶儲存於智能合約中。如此一來，未來在授權驗證的過程中，智能合約能以 Device 的帳戶代替 PAT 作驗證呼叫智能合約的合法性。

- 部署 Authz 合約：在部署 RM 合約之後，RO 會得到 RM 合約的地址。由於 Authz 合約必須參照儲存於 RM 合約中之資源識別碼，故必須要引用 RM 合約。因此，在部署 Authz 合約時，必須儲存預先部署之 RM 合約地址。

- RO 透過 Device 註冊受保護的資源：在本研究所提出的機制中，Device 負責管理智能合約中的 Resource set。當 RO 要向 RM 合約註冊 Device 所擁有的 Resource set 時，需要觸發 Device 並透過 Device 的帳戶向 RM 合約調用 *registerResource* 的功能。發送交易時需要帶入之參數範例如下：

DOI:10.6814/NCCU202101738

- ***resourceName***：Device 中 Resource set 的名稱。

- ***scope***：scope 代表 RqP 所能對 resource 請求之範圍。在 UMA 中，並未明確規範 scope 之規格。在 B-UMA 中，scope 的內容如：'scopes': ['view', 'update']。

在區塊鏈交易成功後，RM 合約會隨機生成一組獨立 identifier，回傳並儲存於合約與 Device 中，未來在驗證過程中均以該 identifier 來代表已註冊的 Resource set。

- 註冊 policy 至 Authz 合約：在 RM 合約註冊完 Resource set 後，Resource owner 可以向 Authz 合約調用 *setPolicy* 的功能，以註冊對應 identifier 之 policy。在 UMA 的授權流程中，AS 可以透過 policy 做出授權決策，或是當驗證資訊不足時，向 RqP 要求提供更完整 claim 以予以授權。在 B-UMA 中，本研究針對 RqP 的驗證提出兩種 policy 的可行方法：

  - 透過區塊鏈使用者帳戶驗證：由於區塊鏈中，區塊鏈參與者能以私鑰代表其獨立的身分。因此，RO 可以直接於 Authz 合約中註冊特定的區塊鏈帳戶。發出存取請求之帳戶經過智能合約驗證過後，能夠不需要提供 claim 就能直接通過授權驗證。更精確的說，RO 能夠授權擁有特定私鑰的使用者直接請求特定資源的權限。

  - 向 RqP 要求提供更多的 claim：當無法確認 RqP 之身分時，或是需要 RqP 提供更多有利驗證的相關資訊時，在收到授權請求後，Authz 合約會請求 RqP 主動提供 claim。例如：要求 RqP 提供其 email 帳號等。在圖 20 中，*hint* 則為 Authz 向 RqP 請求提供 claim 時所出示的提示，

DOI:10.6814/NCCU202101738

以利 RqP 判斷需要出示哪些內容的 claim。

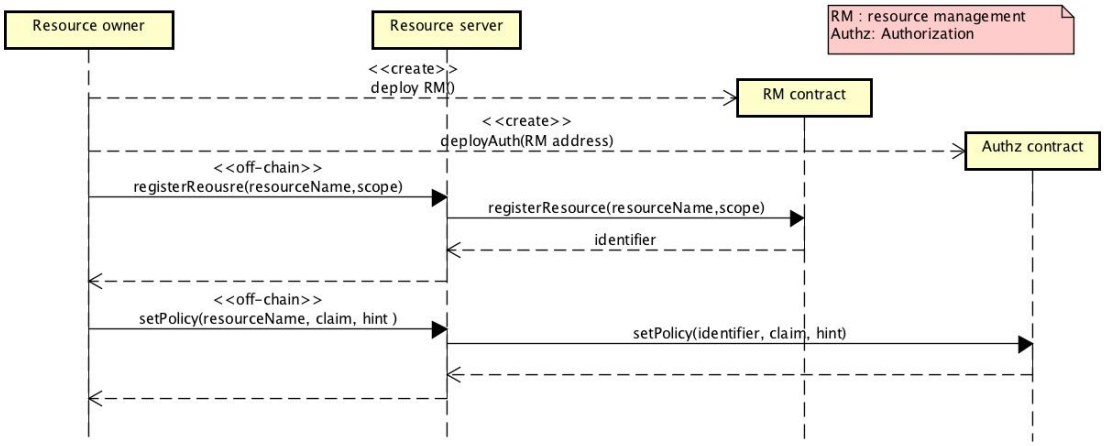

圖 20：B-UMA 的系統建置流程循序圖

4.3.2 第二階段 - 取得授權

第二階段分為兩個步驟以完成 B-UMA 中 Request party (RqP)的請求授權流程，分別為「取得 permission ticket」與「取得 access token」。

(一)、 取得 permission ticket

圖 21 說明了 RqP 向 Authz 合約取得 permission ticket 的流程。由於需要確保每次請求是經由 Device 發送，並透過 Device 中儲存的 identifier 來取得智能合約的授權。因此，在取得 access token 前，Client 需要先取得 Authz 合約所核准請求的 ticket。另一方面。在流程開始之 Client 必須取得向 resource server 請求 resource 之 reference，例如：使用服務發現機制，取得呼叫 Device 的 API。

首先，RO 透過 Client 向 Device 發出資源請求 API，此時請求的中不需要提供 access token。HTTP request 如下：

```
GET photo/example HTTP/1.1
Host: B_UMA.com
```


DOI:10.6814/NCCU202101738

Device 會再檢查是否存在被請求的資源，之後 Device 會利用對應資源的 identifier 向 Authz 合約請求 permission ticket。為了防止惡意的攻擊，在 Authz 合約核發 permission ticket 前，合約內部會先檢查對應的 resource set 是否已經註冊，並確認交易是由合法的 Device 區塊鏈帳戶所發出。確認過後，Authz 合約會透過 Event 機制推播 permission ticket 與先前 RO 所設定的 *hint* 至 Device。為了讓 Client 在下一步能夠向 Authz 合約請求 access token，Device 需要另外將 Authz 合約地址回傳至 Client。最後，Client 向 RqP 出示已收到的 *hint*，並要求其提供相關的 claim。

在此步驟中，Device 的 HTTP response 如下：

```
HTTP/1.1 401 Unauthorized WWW-Authenticate: B-UMA
             realm="example", authz_contract="0xec…",
             ticket="0x26…", hint="email",
             idenitifier="0x54…", ...
```

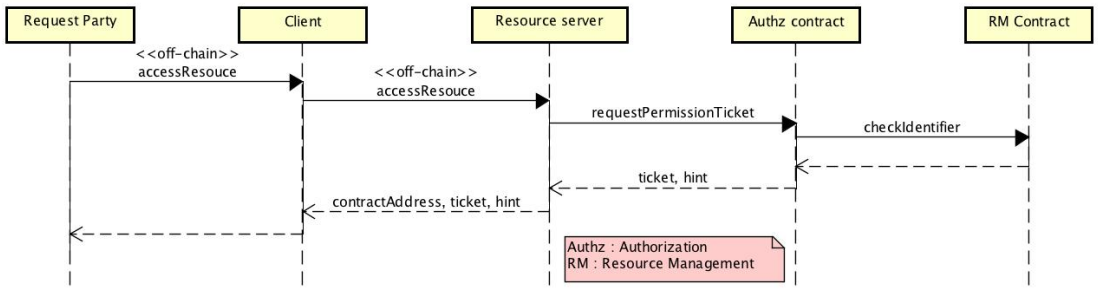

圖 21：取得 permission ticket 循序圖

(二)、 取得 access token

此步驟用意為利用將上一個步驟所取得的 permission ticket 與 RqP 所提供的 claim，向 Authz 合約交換能有效請求 Resource server 中資源的 access token (圖 22)。當 RqP 蒐集完相關的 claim 後，則再次透過 Client 上所配置的區塊鏈節點，向 Authz

DOI:10.6814/NCCU202101738

合約發出交易以使用 permission ticket 換取 access token。值得一提的是，由於 claim 與 permission ticket 在上鏈的過程，是透過本地端 (local side)的區塊鏈節點。因此，相對於一般 UMA 機制，較不會有 claim 在網路上的外洩問題。但在智能合約中，若需紀錄 RqP 所傳送的 claim 內容，為了防止區塊鏈網路中的參與者存取 claim 中的隱私資料，則需在智能合約中以安全的加密機制保存。

Client 調用智能合約中 *requestToken* function 時，傳送的參數範例以 JSON 格式表示如下：

```
{
  "ticket": "0x26…",
  "claim": "...@email.com"
}
```

Authz 合約驗證 claim 的內容後，若驗證成功將會隨機產生一組 access token。在 UMA 的規格中，access token 使用的是 Bearer token [46]的格式，但有鑑於目前智能合約關於編碼解碼等相關技術尚未成熟，加上區塊鏈傳輸機制不相容於 HTTP，因此，Bearer token 的格式較不適合應用於以區塊鏈為底層的傳輸機制。在 B-UMA 中，本研究提出了 token 的產生範例，但未限制 token 的使用格式。此 token 具有唯一性，能夠對應授權的請求，並透過智能合約的機制將其對照(mapping)到發送交易的 RqP 的帳戶地址，其目的為在之後的內部檢查流程時，可以比對智能合約中 RqP 的帳戶地址所對照的 access token，是否與 Client 請求資源時所出示的 access token 一致。最後，Authz 合約會透過智能合約的 Event 機制，將 access token 推播至 Client，並同時帶有相關授權參數，記錄於區塊鏈之交易訊息中，相關授權參數範例如表 6 之內容。另一方面，若在此步驟中，access token 交換失敗時，兩種常見的錯誤訊息將如以下所示：

DOI:10.6814/NCCU202101738

- 當 permission ticket 格式錯誤或不存在時，智能合約會停止交易 (rollback)並發送以下錯誤訊息：

```
"invalid_grant"
```

- 當 RqP 所提供之 claim 或身分不足以符合授權需求時，智能合約會停止交易 (rollback)並發送以下錯誤訊息：

```
"request_denied"
```

表 6：Authz 合約所發出的授權 Event 範例內容

log 屬性	內容
access token	用來存取 Device 中 Resource 之 token
identifier	表示成功授權 resource set 之識別碼
scope	請求授權之範圍
address	授權的帳戶
iat (issued at)	access token 的簽發時間 (Unix time)
exp (expires)	access token 的到期時間 (Unix time)

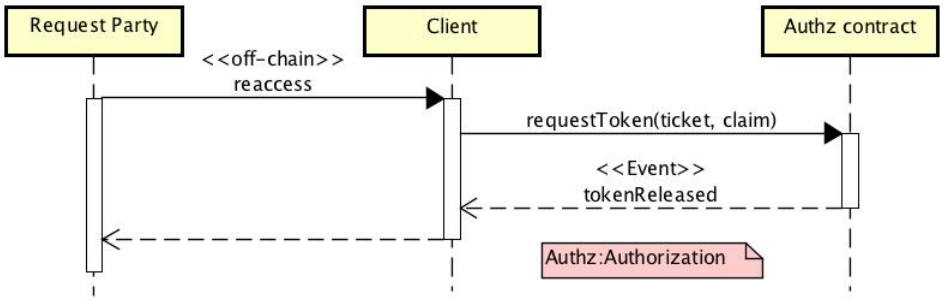

圖 22：取得 access token 循序圖

4.3.3 第三階段 - 資源存取

在此階段中，基於 Client 已獲得 access token 的情況下，RqP 可以在此透過 Client 向 Resource server 請求資源。在此階段中，Device 的資料的匯集可以參考第 3 章

DOI:10.6814/NCCU202101738

所提出的 B-IoT 資料匯集設計樣式中的 *OEI* 與 *OFEI* 模式設計。

RqP 在重新請求 Device 資源的過程中，由於 Client 所發出的 request 中，會在 HTTP 的 header 中，帶有 access token。Device 在收到 request 後，將以 UMA 中所提供的「內部檢查 token (introspect token)」機制，驗證所收到的 access token 之正確性，以下將詳述在 B-UMA 中針對 「內部檢查 token (introspect token)」機制的設計：

在 UMA 中，當 Client 成功取得 Authz 合約所發配之 access token 後，可以向 Device 出示 access token 以存取 Device 之資源。由於 Device 不參與 UMA 中驗證機制的關係，access token 必須要轉交由 Authz 合約進行檢驗。但 access token 在產生過程會紀錄於區塊鏈，並基於區塊鏈之公開透明性的因素，參與區塊鏈之節點很容易就取得 access token 之內容。考慮到惡意節點可能透過 access token 冒充 RqP 的身分。因此，在 B-UMA 中檢查 access token 時，RqP 必須透過自身的私鑰將 token 簽名，再透過 Client 呼叫 Authz 合約中檢查 access token 的功能，之後利用智能合約中簽章的地址還原機制[47]，確認 access token 是否由 RqP 的私鑰簽名。若驗證成功，Authz 合約會回傳 true。若驗證失敗，智能合約將會回傳 false 或終止交易 (rollback)並發送驗證失敗訊息。

DOI:10.6814/NCCU202101738

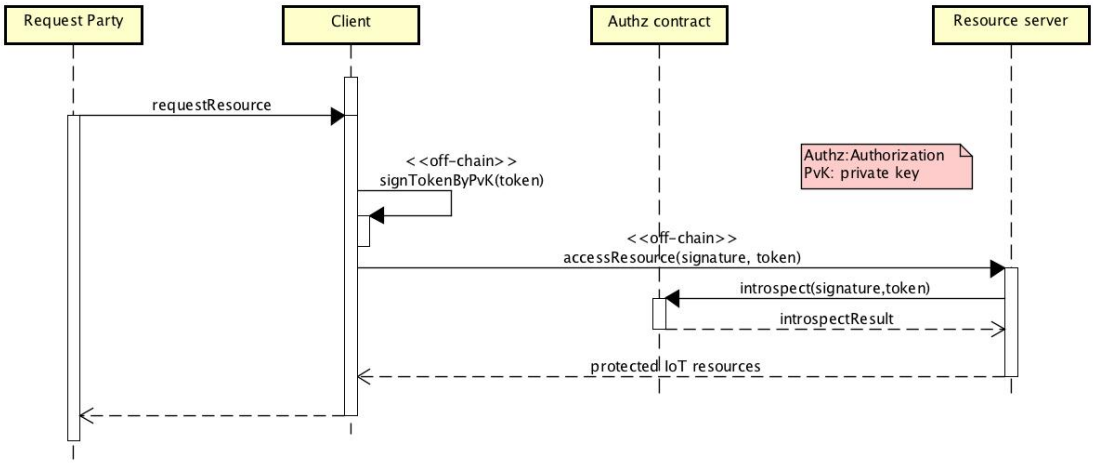

圖 23：B-UMA introspect token 循序圖

在 introspect token 步驟中 HTTP request 與智能合約 function 的調用範例如下：

- Client 在取得 access token 後重新向 Device 請求資源的 HTTP request 格式 (Authorization: access token, tokenSignature: access token 之簽章)：

```
GET /photo/example HTTP/1.1
Host: B_UMA.com
tokenSignature: 0x22b...
Authorization: 0x52...
```

- Resource server 向智能合約調用 *introspect* function 時，所帶入的參數 (JSON 格式)：

```
{
   "signature":"0x22b...",
   "token":"0x52..."
}
```

- introspect function 驗證 access token 失敗時，發出的錯誤訊息內容：

```
"invalid_token"
```


DOI:10.6814/NCCU202101738

最後，在內部檢查 token 步驟完成後，Device 即可授權 Edge 所發出之資源請求，並向 Edge 提供相關資源。RqP 可以在 access token 之授權期限內，多次透過 Client 向 Device 存取資源。當 access token 過期時，則可再次透過 B-UMA 機制取得新的 access token。

4.4 B-UMA 的智能合約設計樣式

此節針對本論文所提出的 B-UMA，在設計相關智能合約時可能面臨的問題深入解析與討論。並根據智能合約的開發經驗，提供基於 B-UMA 機制下的解決方案。最後，將情境、問題與解決方案以設計樣式形式呈現。開發人員在設計或實作 B-UMA 或其他基於區塊鏈的驗證機制時，可參考本節所整理之設計樣式，分別為「授權與資源管理機制分離 (Authorization and Resource Management Segregation)」、「被授權用戶註冊 (Authorized User Registry)」與「Token 的內部檢查 (Token introspection)」。

4.4.1 授權與資源管理機制分離

- **Context:**

在設計 B-UMA 或基於區塊鏈的授權機制時，資源擁有者若要同時透過智能合約註冊與管理其資源，也需要透過智能合約中設置的 policy 進行 access request 之驗證與授權的情況下，若部署合約後發現設計漏洞、或是合約內容需要動態調整，造成已部署之合約不堪使用時，則需要重新部署合約。然而，原儲存於智能合約中的既有靜態資料 (如：用戶資訊，被存取資源等等)，必須遷移到新版本的合約。如此一來將耗費額外的資源與成本。

DOI:10.6814/NCCU202101738

- **Problem:**

在 B-UMA 或是基於區塊鏈的授權機制下，如何解決當智能合約有更新需求時，需要遷移或重複將大量靜態資料上鏈的限制？

Forces:

- 軟體分層設計：在軟體工程領域中，分層 (layers)設計是一個重要的概念。軟體系統的分層設計不僅有助於隔離不同的專注點 (Concern Point)，減輕不同層次之間的相依性，更有助於降低系統的複雜度，以提升系統品質。
- 不同類型資料的修改頻率不同：在 UMA 流程中，主要需要部署的資料主要為兩大類，分為受 Protected resource 與 policy。兩種資料的更新頻率因應用場景而異。
- 歷史授權紀錄的管理：若合約需要重新部署時，需要有一套管理舊有合約的方法，使得過去的授權得以永久追蹤。

- **Solution:**

在軟體分層設計樣式 (Software Layered Architecture Pattern)中，將商業邏輯與資料庫的服務分為業務層 (Business Layer)與資料層 (Data Layer)。將商業邏輯與資料存儲服務分層處理。在本樣式中，將上述 UMA 中的兩類資料分為兩層管理。如圖 24，本研究將智能合約分為下層「資源管理合約」 (Resource Management contract, RM 合約)與上層「授權合約」 (Authorization contract, Authz 合約)。RM 合約負責管理資源伺服器中受保護的資源，Authz 合約則引用 RM 合約中的資訊，專注於驗證 (authentication)與授權 (authorization)流程。

DOI:10.6814/NCCU202101738

圖 24：B-UMA 智能合約分層類別圖

在將合約分別部署之後，若某一合約因為版本更新需要重新部署，舊的合約中的資料必須要被保存時，則必須要記錄下合約的區塊鏈地址。如圖 25 之範例程式碼，將參考 RM 合約之變數 (*RM_address*)指向最新版合約 address，並將舊版本合約位址存進儲存歷史合約地址的陣列 (*previous_RM_addres*)。另一方面，由於 Authz 合約與 RM 合約之更新頻率不同，更精確的說，某部分的資料可能是靜態資料 (不會有版本更新的問題)。因此，開發人員可以考量是否要實作合約版本管理的機制。

DOI:10.6814/NCCU202101738

```
contract AuthorizationContract {

    address public resourceOwner;
    address public RM_Address;
    address[] previous_RM_address;

    constructor(address _RM_Address) public{
        resourceOwner=msg.sender;
        RM_Address = _RM_Address;
    }

    function changeRM(address _RM_Address, address _new_RM_address) public {
        require(resourceOwner == msg.sender, "Access deny, not resourceOwner");
        require(_RM_Address == RM_Address,"invaild RM_Address");
        RM_Address = _new_RM_address;
        previous_RM_address.push(_RM_Address);
        //... emit update event including array index
    }
}
```

圖 25：Authorization 合約的版本管理機制範例程式碼

- **Consequence:**

Benefits:

- 減少資料遷移的成本：將資源管理與授權機制分為兩個合約設計與部署。有助於減少當只有部分的合約邏輯需要更改時，不需要重新部署所有合約邏輯與遷移已上鏈的靜態資料。因此，能大幅減少區塊鏈機制使用之成本。

- 可更新性 (Upgradability)：將負責儲存靜態資料 (用戶、被授權資源的資料)與動態授權邏輯的合約分開設計，可以有助於資源擁有者在不需要影響到靜態資料的前提下，部署新版本的授權合約。

Drawbacks:

- 擴展性：由於在 B-UMA 授權過程中，一方面要 Authz 合約紀錄 RM 合約之參考資訊，如：RM 合約的地址與 RM 合約的 ABI (Application Binary

DOI:10.6814/NCCU202101738

Interface)等。另一方面，RqP 也要取得相關調用合約的參考資訊。基於上述理由，授權機制中系統的擴展性將會有所限制。

- 合約之間的調用成本：由於本設計樣式將 Authz 合約與 RM 合約分層設計，因此在授權流程中，需要透過兩個合約的相互存取來完成。如此一來，將增加透過智能合約間調用的成本 (如：ethereum 中的交易消耗的 gas、時間與儲存空間)。

- **Related patterns:**

  - *Contract Register* [34]：管理在智能合約中註冊的調用合約版本，使智能合約能確保調用到的合約為最新版本的合約。

  - *Layers* [8]：將系統以階層化的方式定義，抽象成不同層次的群組。每個群組會依照所界定的界線區分其負責的工作，而上層群組將依賴於下層群組所提供的服務。

  - *Data Segregation*[34]：為了降低智能合約中，「邏輯」與「資料」的耦合度，避免未來所有靜態資料將隨著版本更新重新部署。因此，將智能合約分成「資料」與「邏輯」兩種面向設計，並分別部署至區塊鏈中。

4.4.2 被授權用戶註冊

- **Context:**

在普遍的授權驗證機制中，藉由認證資源請求方之身分來予以授權是很常見的方式。在 UMA 中，某些情況下資源擁有者可以針對特定使用者進行授權。在 UMA 的系統部署階段，資源擁有者可以在 policy 中註冊被允許存取特定受保護資源的資源請求方之參考資料 (reference)。當該資源請求方向授權伺服器發出 access

DOI:10.6814/NCCU202101738

request 時，授權伺服器可在驗證資源請求方之身分後，授權相關的 access request。

- **Problem:**

在 UMA 中，資源擁有者都會預先在 policy 中設定可以存取資源的第三方。然而，由於被授權的第三方可能會動態增減，加上在 B-UMA 中，若智能合約已經部署就不能再被更改。因此，如何在智能合約中建立動態管理被授權第三方參考資料的機制，會是一大議題。

Forces:

- 機密性：由於區塊鏈公開透明的特性，任何人都可以調用特定智能合約。但在智能合約中，若某些功能只限定該合約擁有者或任務可以存取，此時必須對特定功能設計存取管理機制。
- 授權存取特定資源：在 UMA 中，受保護的資源可能有很多個，資源擁有者在特定情境下希望授權不同的用戶存取特定的資源。
- 動態管理：由於智能合約之程式碼與邏輯一經部署後，就無法再進行修改。因此若能設計動態管理存取控制的智能合約樣式，將能有效節省重新部署智能合約的成本。

- **Solution:**

由於區塊鏈系統中具有其公私鑰機制，因此將有助於作為在智能合約中設計存取控制機制時的驗證方法。資源擁有者在設計 policy 時，可以利用檢查區塊鏈帳戶的方式來認證資源請求方的身分。如上述所述之安全性問題，在動態增加授權使用者時，必須確保交易發送者的身分。在圖 26 中，本研究設計授權第三方的存

DOI:10.6814/NCCU202101738

取機制，並只有合約的擁有者可以調用該機制。

為了將特定資源授權給特定第三方，必須將特定使用者與已註冊資源做耦合，資源擁有者能夠透過智能合約，確認某資源能夠授權予特定第三方。在圖 26 的 *setParticipantOfResource* function 中，本研究先將受保護資源的識別碼 (identifier) 與授權第三方區塊鏈帳戶雜湊，取得雜湊值，並透過智能合約中 *mapping* 機制將該雜湊值對應為 true (代表授權許可)。在 B-UMA 的授權流程中，一方面可以利用向智能合約動態新增被授權用戶 (調用 *setParticipantOfResource* function)可以利用智能合約中的 modifier 的預先檢查機制 (*checkParticipantOfResource* modifier)，檢查特定存取資源是否授權予特定第三方。

```
contract AuthorizationContract {

    address public resourceOwner;
    mapping(bytes32 => bool) isParticipantOfIdentifier;

    modifier checkParticipantOfResource(bytes32 _identifier, address _rqpAddress) {
        bytes32 hash = keccak256(abi.encodePacked(_identifier, _rqpAddress));
        require(isParticipantOfIdentifier[hash] == true, "not vaild account");
        _;
    }

    constructor() public{
        resourceOwner=msg.sender;
    }

    function setParticipantOfResource(bytes32 _identifier,address _address) public{
        require(msg.sender == resourceOwner,"Access deny, not resourceOwner");
        bytes32 _hash= (keccak256(abi.encodePacked(_identifier, _address))) ;
        isParticipantOfIdentifier[_hash]= true;
    }

    function deleteParticipantOfResource(bytes32 _identifier,address _address) public{
        require(msg.sender == resourceOwner,"Access deny, not resourceOwner");
        bytes32 _hash= (keccak256(abi.encodePacked(_identifier, _address))) ;
        delete isParticipantOfIdentifier[_hash];
    }
}
```

圖 26：資源請求方身分認證的設計樣式範例程式碼

DOI:10.6814/NCCU202101738

- **Consequence**:

Benefits:

- 機密性：透過在智能合約中設計被授權用戶帳戶的註冊機制，來驗證調用智能合約的區塊鏈參與者之身分，並以此來確保智能合約中，特定 function 只能被特定用戶所存取。

- 動態管理：在圖 26 的範例程式碼中，利用智能合約中的資料結構 (mapping)，合約擁有者可以授予特定區塊鏈帳戶存取資源的權限，也可以移除該帳戶的權限。相關存取權限新增與刪除過程可以通過調用智能合約的 function 動態修改。

Drawbacks:

- 成本：若頻繁的調用智能合約來增減註冊的被授權帳戶，則需要付出相對應的交易成本。

- **Related patterns:**

- *Ownership* [34]：藉由設定智能合約的擁有者，以確保特定的 fucntion 只有智能合約的擁有者可以存取，以此來增強智能合約存取的安全性。

- *Multiple Authorization* [11]：在智能合約中事先定義授權用戶之集合 (set)，持有授權用戶之私鑰的使用者，可以透過 *M-of-N multi-signature* 的簽章機制，授權特定交易是否能合法進行。

DOI:10.6814/NCCU202101738

4.4.3 Token 內部檢查

- **Context:**

在 UMA 機制中，資源伺服器在收到資源請求方所發出的帶有 access token 之請求後，會交由授權伺服器檢查 access token 之合法性。此機制在本論文所提出的 B-UMA 中，access token 是由智能合約所發布。由於區塊鏈機制有公開透明之特性，此外，access token 在鏈下傳輸時可能被惡意第三方取得，因此，RPT 可能會面臨暴露的風險。

- **Problem:**

智能合約在接收機密資訊 (如 B-UMA 中之 access token)時，如何確保機密資訊之完整性與發送來源？

Forces:

- 完整性：由於 access token 足以表示一個請求已通過授權機制的授權，所以 access token 持有者可以向資源伺服器存取資源。但由於在 B-UMA 中，access token 會儲存於區塊鏈，所有區塊鏈參與者將能存取該合約內容。故 access token 可能受到惡意第三方所控制，進而非法存取被授權的資源，如攻擊者利用重放攻擊 (replay attack)，非法修改或存取受保護的資源。

- 機密性：基於區塊鏈的公開透明性，智能合約需要確保請求所出示之 access token 為被授權的資源請求方所發出。以防止攻擊者冒用被授權的資源請求方名義取得資源。

DOI:10.6814/NCCU202101738

- **Solution:**

如圖 27，為了確保授權驗證過程的完整性，在發布 access token 時，將 access token 中加入時間戳與隨機數，以確保 access token 的唯一性。另外，能藉由設定 access token 之時效，防止 access token 暴露所導致的重送攻擊或重複請求。

另一方面，為了防止 access token 由非被授權方發出，可以用以太坊中的簽章機制，確認 access token 是由被授權的資源請求方所出示。如圖 27 與圖 28，首先在取得 access token (*releaseToken*)的階段，將 access token 對應 (mapping)到發出交易之資源請求方之區塊鏈帳戶。之後在內部檢查 token (*introspectAccessToken*)的階段，資源請求方需要將 token 透過其專屬私鑰簽名 access token 後，將簽章和 access token 出示給資源伺服器。最後，資源伺服器透過智能合約檢查收到的 access token。當檢查成功後，透過智能合約之簽章驗證機制 (*ecrecover*)還原出簽章之簽名者[48]。如此一來可以有效確認 access token 出示者之身分，確保資源存取的機密性。

```
mapping(address => bytes32) access_token;
mapping(bytes32 => uint)token_vaildTime;
event tokenReleased(address msg_sender,bytes32 access_token,uint iat, uint exp);

function releaseToken() public {
    // ...implement the code of validate the ticket and the claim
    uint random_number = uint(keccak256(abi.encodePacked(block.timestamp)))%100 +1;

    access_token[msg.sender] = (keccak256(abi.encodePacked(
        now, msg.sender, random_number))) ;

    token_vaildTime[access_token[msg.sender]] = now+ 1 days;

    emit tokenReleased(
        msg.sender,
        access_token[msg.sender],
        now,  //issued at
        token_vaildTime[access_token[msg.sender]]);  //expire time
}
```

圖 27：Authorization 合約的變數定義與發佈 access token 範例程式碼

DOI:10.6814/NCCU202101738

```
function introspectAccessToken(
bytes32 _token,
uint8 _v,
bytes32 _r,
bytes32 _s
) public view returns(bool) {
    require(_token != 0, "invaildToken");
    if(now > token_vaildTime[_token]){
        require(1 == 0, "exceeded the expiration date");
        return false;
    }

    bytes memory prefix = "\x19Ethereum Signed Message:\n32";
    address signer = ecrecover(
        (keccak256(abi.encodePacked(prefix,_token))),
        _v, _r, _s
    );

    if(access_token[signer] != 0 && access_token[signer] == _token){
        return true;
    }else{
        require(1 == 0, "invaild signer");
        return false;
    }
}
```

圖 28：Authorization 合約 introspect token 範例程式碼

- **Consequence:**

Benefits:

- 可驗證性 (Authenticity)：在 B-UMA 中，資源伺服器所收到的密文已由資源請求方的私鑰加密與簽名。因此，資源伺服器在 introspect token 階段可以透過智能合約機制將簽名還原為簽章的帳戶以驗證資源請求方的身分。

- 機密性：Token 明文在資源請求方本地端 (local)加密過後，再透過 off-chain 傳輸至資源伺服器。因此，資源請求方不需要擔心 token 在傳輸的過程洩露至網路中。

- 成本：因為透過鏈上機制驗證簽章不需要更動變數或於區塊鏈記錄資料，

DOI:10.6814/NCCU202101738

因此不必消耗區塊鏈交易的手續費與等待區塊驗證時間。

Drawbacks:

- 明文加密的成本：在此樣式中，所有欲加密 access token 的使用者都必須擁有相關區塊鏈的私鑰並且實作加密機制。因此，會造成加密使用者的計算與開發成本提高。

- 智能合約之彈性：若欲使用此設計樣式，開發人員必須在設計智能合約時，將相關的鏈上 access token 檢查與 access token 的儲存機制實作於智能合約中。若合約已經部署，相關設計則無法再更改。如此一來，合約的彈性也會相對降低。

- **Related patterns:**

  - *Delegated Computation*[12]：使用者將明文加密後的密文，透過將密文提供給代理人 (如第三方的伺服器)，該代理人透過區塊鏈智能合約中的機制驗證密文內容的正確性。另一方面，使用者可以向代理人證明該密文確實由使用者所擁有 (密文已經過使用者的私鑰簽名)。

DOI:10.6814/NCCU202101738

第5章 系統實作

本章節為了說明本論文所整理與提出 B-IoT 情境中的資料匯集樣式與使用者自主管理存取機制如何運作，將以第 2 章中所介紹的「智慧海運」案例，建置一個基於 B-IoT 情境下，實證「使用者自主管理存取機制 (B-UMA)」與「邊界伺服器和物聯網裝置之間資料匯集」兩類設計樣式的系統。本章也詳述該實證系統的架構、硬體、網路設施與伺服器軟體所使用之框架和模組與區塊鏈相關實作工具。

5.1 區塊鏈與智能合約

在本實證系統中，區塊鏈使用以太坊 (Ethereum)實作。以太坊生態系提供完整的智能合約開發與部署平台，也有提供使用不同程式語言開發 DApp 時，對接的 API (Application Programming Interface)，例如：支援於 JavaScript 的 Web3.js 或支援於 Java 的 Web3J。在不同的 B-IoT 裝置上，本研究皆透過以太坊官方所提供，以 Go 語言開發的以太坊客戶端 (go-ethereum)，簡稱 geth，來建立私有鏈。為了建立私有鏈，必須先設定創世區塊檔案 (genesis.json)，其詳細設定如圖 29 所示。其中值得一提的是，為了減少私有鏈中區塊驗證的難度，本研究將 *difficulty* 設為較低的值；*chainId* 代表區塊鏈的網路識別碼，所有私有鏈的節點需要使用相同的 *chainId*；*gasLimit* 則是在每筆交易中，所允許的交易手續費 (gas)的上限。

DOI:10.6814/NCCU202101738

```
{
  "config": {
    "chainId": 33,
    "homesteadBlock": 0,
    "eip155Block": 0,
    "eip158Block": 0,
    "ByzantiumBlock": 0
  },
  "nonce": "0x000000000000033",
  "timestamp": "0x0",
  "parentHash": "0x0000000000000000000000000000000000000000000000000000000000000000",
  "gasLimit": "0x8000000",
  "difficulty": "0x1",
  "mixhash": "0x0000000000000000000000000000000000000000000000000000000000000000",
  "coinbase": "0x3333333333333333333333333333333333333333",
}
```

圖 29：以太坊創世區塊設定檔案 (genesis.json)

智能合約方面，本研究採用以太坊官方所提供的圖靈完備的智能合約語言 - Solidity 撰寫。由於 Solidity 的語法與 JavaScript 較為相似，在以太坊生態系中，也提供較多用於開發與測試的工具與模組，維護與版本的更新頻率也較高，因此，對於以太坊開發者來說，相較於其他智能合約語言 (如：Vyper, Bamboo)容易上手與實作。另一方面，在開發的過程中，本研究透過針對開發 Solidity 所提供的線上 IDE - Remix [49]，進行智能合約的編譯、除錯與單元測試。也利用開發測試用的私有區塊鏈客戶端 - Ganache [27]，協助開發中的應用程式能夠正確調用部署於區塊鏈中的智能合約功能。最後，再把測試完成的智能合約部署至私有鏈中。

5.2 實證系統

5.2.1 系統架構

本實證系統將依照 B-IoT 架構中，雲端 (Cloud side)與邊界端 (Edge side)分別進行實作。如圖 30 所示，雲端主要為貨代公司 (資源擁有者)在雲端架設之伺服器

DOI:10.6814/NCCU202101738

與區塊鏈網路中之節點。區塊鏈網路中部署與實證系統相關的智能合約，並透過貨代公司部署；邊界端根據本論文中所使用之案例情境，設定為貨櫃船上的環境。船上配置包含邊界伺服器與架設於 reefer 內的 Device (B-UMA 中的 resource server)。船員 (資源請求方)可透過船上的 Edge (B-UMA 中的 client)存取 reefer 內的 Device 之服務。

區塊鏈節點部署於 Device、Edge 與 Cloud 中。由於船上的網路訊號較弱與 Device 之硬體限制的緣故，Device 上部署的區塊鏈輕節點需要依賴於 Edge 上部署的全節點同步區塊資料。船上的 Edge 則透過區塊鏈網路與其他區塊鏈節點同步資料，並參與區塊鏈中的區塊驗證工作。實證系統中船員、貨代公司與 Device 皆擁有在區塊鏈網路中，獨立的以太坊私鑰以代表其身分。

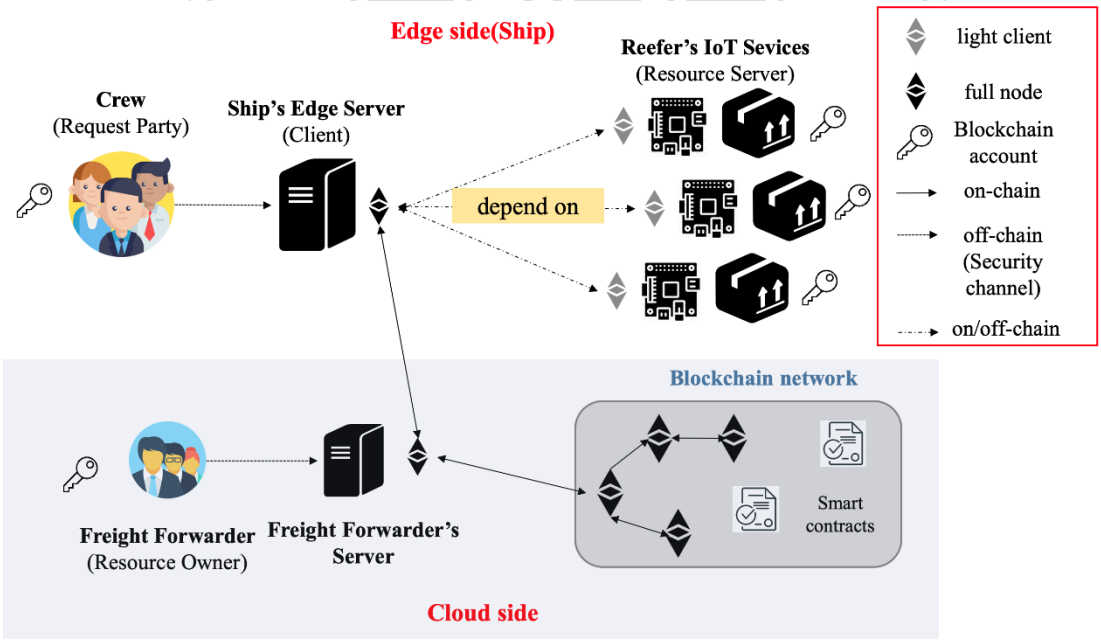

圖 30：B-IoT 實證系統架構圖

5.2.2 軟硬體架構

在實證系統中，本研究利用兩台伺服器 (表 7 中的 PC1 與 PC2)作為區塊鏈節點，

DOI:10.6814/NCCU202101738

各架設一個以太坊的全節點。而針對 B-IoT 資料匯集與 B-UMA 的不同層面，為了有利於系統評估與實驗的進行，本研究將分開實作與討論，以下將分別介紹兩個層面的軟硬體配置。

B-IoT 資料匯集方面，實證系統透過表 7 中的樹莓派 (Raspberry Pi)與 PC1 組成。其中，PC1 扮演 Edge 角色，提供 API 供呼叫 Edge 的資料請求等服務；另外，再利用樹莓派扮演 Device 角色，主要負責提供感測資料 (如：溫濕度、定位資訊與照片等)的資料匯集服務，樹莓派也作為區塊鏈節點並部署以太坊輕節點。

B-UMA 方面，則透過表 7 中的 PC2 實作兩個 process，來模擬圖 30 中的 fregiht forwarder's server 與 ship edge server。而圖 30 中的 reefer's IoT devices 則透過表 7 中的樹莓派 (Raspberry Pi)來模擬。

表 7：實證系統軟硬體規格

設備名稱	作業系統	CPUs	RAM	geth 同步模式
Raspberry Pi B3+	Raspbian GNU/Linux 9.4	ARMv7 1.2GHz CPUs	1GB RAM	light client
PC1	Ubuntu 18.04.2	Intel 4GHz 64bit	16GB RAM	full node
PC2	macOS 10.15.2	Intel 2.5GHz 64bit	16GB RAM	full node

5.3 實證系統介面設計

實證系統採用 JavaScript 語言進行開發，並以 Node.js 作為 JavaScript 之的執行環

DOI:10.6814/NCCU202101738

境，伺服器架設則使用 Koa2 後端框架，將後台系統服務模組透過 HTTP 中 Restful 樣式的 API 封裝。前端應用程式或其他程序可以透過 Koa2 所提供之 Restful API，藉由 Koa2 模組所提供的路由中介軟體，調用相關服務。

本實證系統針對不同應用種類與種類中的介面設計了相關的 Restful API，種類主要分別為「B-IoT 的資料匯集樣式」與「B-UMA」，本節將針對各個種類分別整理與介紹相關的 API 與使用的模組。

5.3.1 B-IoT 的資料匯集設計樣式介面設計

針對 Edge 與 Device 之間的資料匯集樣式的種類，為了有利實驗測試，本系統針對 *OEI*、*ODP* 與 *OFEI* 三個設計樣式的介面，設計了 Restful API。API 封裝之後端服務透過 web3.js 調用 geth 的服務。表 8、表 9 與表 10 有針對相關 API 之介紹。其中，*ofei/whisperSubscribe* 與 *ofei/dataCallbackByWhisper* 透過 web.js 模組提供之 API 調用 geth 中之 whisper 功能，透過 whisper 協議建立鏈下訊息傳輸通道傳遞訊息。*oei/deployRequestRegistry*，*odp/deployObserver* 與 *ofei/deployRequestRegistry* API 則參考第 3 章設計樣式提供的範例合約實作。

表 8：針對 OEI 介面的 API 設計

HTTP method	API 路徑	功能簡介
POST	oei/deployRequestRegistry	資源擁有者或者中立第三方部署智能合約
POST	oei/deployConsumer	資源擁有者或者中立第三方部署智能合約
GET	oei/listenCallbackEvent	Edge 向智能合約監聽是否 Device 有回傳 callback 資料
GET	oei/listenAccessRequestEvent	Device 向智能合約監聽 Edge 是否有發出資料請求

DOI:10.6814/NCCU202101738

POST	oei/callback	Device 將匯集完成的資料透過智能合約回傳至 Edge

表 9：針對 ODP 介面的 API 設計

HTTP method	API 路徑	功能簡介
POST	odp/deployObserver	資源擁有者或者中立第三方部署智能合約
GET	odp/dataSubscribed	Edge 向區塊鏈訂閱資料發佈的 Event
POST	odp/publishData	Device 向智能合約發佈資料，並透過 Event 機制通知 Device

表 10：針對 OFEI 介面的 API 設計

HTTP method	API 路徑	功能簡介
POST	ofei/deployRequestRegistry	資源擁有者或者中立第三方部署智能合約
GET	ofei/whisperSubscribe	透過 web3.js 產生 whisper keypair，並開始監聽 whisper 信息
POST	ofei/requestRegister	Edge 向 Device 發出資料呼叫請求，並將 whisper 公鑰同時送上合約儲存
GET	ofei/listenAccessRequestEvent	監聽智能合約 AccessRequest Event，若有監聽到 Event 則開始匯集資料，並呼叫 dataCallbackByWhisper API 透過 whisper 回傳資料
GET	ofei/listenTerminalEvent	Device 監聽智能合約 Terminal Event，若有監聽到 Event 則停止匯集與回傳資料

DOI:10.6814/NCCU202101738

POST	ofei/dataCallbackByWhisper	透過 web3.js 呼叫 whisperAPI 回傳資料
POST	ofei/terminate	Edge 調用智能合約 teminate function，向 Device 發送停止匯集資料 Event

5.3.2 B-UMA 的介面設計

在 B-UMA 種類中，介面分別針對第 4 章中提出的 B-UMA 驗證流程中所使用到的 RM 合約、Authz 合約以及鏈下服務做設計，表 11、表 12 與表 13 針對相關 API 做介紹。API 中以 rs 開頭的代表 B-UMA 中 Resource server 提供的服務，rqp 開頭的則代表 Client 所提供的服務。其中，*rqp/encryptToken* API 所封裝的服務會調用 npm (Node Package Manager)提供之 eth-crypto [50]模組，將 access token 明文透過資源請求方所提供的以太坊私鑰簽名。另外 *auth/introspectAccessToken* API 所封裝之服務則也調用 eth-crypto 模組，將資源伺服器所收到之 access token 之簽章拆解，以產生 Authz 合約在 introspect token 階段需要之參數 (圖 28 中之_v, _r, _s 參數)。在實證系統中，針對智能合約介面所設計之服務，皆透過 web.js 模組之 API 與智能合約溝通。*blockchain/deployAuth* 與 *resourceManage/deployRM* API 則負責 Authz 與 RM 合約的部署，相關合約參考 4.4 所提供的範例程式碼實作，並透過 npm 所提供的 solc 模組編譯後，再利用 web3.js 模組提供之 API 部署至區塊鏈。

表 11：針對 Authz 合約介面的 API 設計

HTTP method	API 路徑	功能簡介
POST	rs/blockchain/deployAuth	透過資源擁有者或中立第三方部署 Authz 合約

DOI:10.6814/NCCU202101738

POST	rs/auth/setPolicy	透過對應 identifier 設定 UMA 之 policy，如 claim、hint。
POST	rs/auth/setParticipantOfIdentifier	設定被授權能夠存取特定資源的使用者帳戶
POST	rs/auth/generateTicket	資源伺服器驗證 access request 後，向智能合約請求 permission ticket
POST	rs/auth/releaseToken	RqP 向智能合約以 permission ticket 交換 access token
GET	rs/auth/introspectAccessToken	在 B-UMA 授權流程中，資源伺服器透過智能合約驗證收到的 token 之簽章

表 12：針對 RM 合約介面的 API 設計

HTTP method	API 路徑	功能簡介
POST	rs/resourceManage/deployRM	透過資源擁有者或中立第三方部署 Authz 合約
POST	rs/resourceManage/registerResourceSet	資源擁有者將受保護註冊資源至 RM 合約
GET	rs/resourceManage/checkIdentifier	向智能合約查看已註冊資源的識別碼
GET	rs/resourceManage/checkScope	向智能合約查看已註冊資源的 Scope

DOI:10.6814/NCCU202101738

表 13：鏈下服務介面的 API 設計

HTTP method	API 路徑	功能簡介
GET	rs/db/getResourceSet rqp/db/getResourceSet	Resource owner 向本地端所建置的資料庫查詢 Reource set 的相關資訊
GET	rs/db/getPolicy	Resource owner 向本地端所建置的資料庫查詢已註冊的特定 identifier 的 policy 的相關資訊
GET	rs/offchain/requestResource	RqP 透過 client 向 resource server 請求受保護的資源
POST	rqp/encryptToken	RqP 透過 client 本地端的服務將 access token 做加密簽章

國立政治大學 National Chengchi University

DOI:10.6814/NCCU202101738

第6章 系統評估

本章節將基於第 5 章的實證系統，藉以評估本論文提出的設計是否能夠解決本論文所整理關於 B-IoT 之問題與挑戰，驗證第 3 章與第 4 章所整理與提出的樣式與機制之可行性，並重複檢驗系統之缺失與疏漏。首先，針對資料匯集設計樣式，為了證明每個樣式的 Solution 能否解決特定 Context 下所面臨的 Problem，將針對該樣式所考慮之 Force 中可量化的指標 (如：資源消耗、響應時間與成本)進行評估與實驗測試。

第二，針對本論文所提出的 B-UMA 與其智能合約設計樣式，根據 B-UMA 不同階段中的應用場景，進行案例研討，包含功能性測試與安全性分析。其中，功能性測試將針對實證系統中所設計的 API 介面，進行單元測試，驗證結果是否與預期一致。安全性分析方面，將針對智能合約進行安全性測試與靜態測試。安全性測試以資安三要素中 (機密性、完整性與可用性)，各個要素常見的幾種攻擊手段，以質性方式論證 B-UMA 能夠有效防禦攻擊；靜態測試則利用測試工具，檢查 B-UMA 中的智能合約程式原始碼是否有安全性漏洞。

6.1 B-IoT 的資料匯集設計樣式

6.1.1 實驗設計

基於第 5 章所介紹的實證系統，本論文將針對 5.3.1 所提出的設計進行一系列的實驗，以驗證與評估在第 3 章中所整理的設計樣式。本章節實驗針對 *OEI*、*ODP* 與 *OFEI* 進行「N 筆資料匯集完成時間」、「平均記憶體使用量」、「平均 CPU 使

DOI:10.6814/NCCU202101738

用率」、以及「Gas 消耗量」進行實驗測試，並針對實驗數據進行之交叉比對，詳細實驗設計如表 14 所示。

值得一提的是，在實驗過程中，回傳的數據將以 Device 所產生的隨機資料模擬，所以實驗數據不需要考慮感測器或 I/O 資料匯集過程所消耗的時間。因此，本實驗量測的主要是 Edge 與 Device 在鏈上與鏈下的資料匯集過程所消耗的時間。另外，本實驗以第 5 章中所介紹的 Edge 伺服器，利用 geth 啟動兩個執行緒進行挖礦，並透過區塊鏈網路與 Device 中的輕節點同步區塊資料。

表 14：B-IoT 資料匯集樣式實驗設計

實驗名稱	實驗設計
N 筆資料匯集完成時間	• *OEI*、*OFEI*：Edge 發出一筆資料請求，收到 n 筆資料的總回應時間 (n = 200, 400, 600, 800) • 每種實驗的資料皆測試 5 次並求其平均值與標準差
平均記憶體使用量	• 一次匯集 10 筆資料 (發送請求與收到回傳的資料)之記憶體使用量，半秒取樣一次，求使用量的平均
平均 CPU 使用率	• 一次匯集 10 筆資料 (發送請求與收到回傳的資料)之 CPU 使用率平均，半秒取樣一次
gas 消耗量	• 各個樣式的智能合約中，不同操作 (operation)的 gas 消耗量

6.1.2 N 筆資料匯集完成時間比較

在第 3 章中，回應時間是開發人員評估選擇何種樣式來匯集資料的重要指標之一。在相同網路與硬體的環境下，B-IoT 系統中裝置之間的資料傳輸時間會因為區塊鏈共識演算法與創世區塊中設定的挖礦難度而有顯著的不同。在本實驗中，本研究測試兩種樣式 (*OEI* 與 *OFEI*)的不同資料數量匯集時間，主要是針對鏈上

DOI:10.6814/NCCU202101738

與鏈下 (透過以太坊的 whisper 機制)傳輸方法的差異做比較。由於 *OEI* 與 *OFEI* 為 Edge「主動」向 Device 發出資料請求的樣式，所以在資料匯集之前，會先向智能合約發送一筆資料請求的交易，並透過智能合約中的 Event 機制通知 Device 需要匯集的資料內容與數量，以觸發 Device 回傳感測資料。另一方面，由於在 *ODP* 中 Edge 為「被動」向 Device 訂閱資料，且資料傳輸方法與 *OEI* 一致 (皆經由鏈上傳輸)。因此，*ODP* 中的 Solution 在本實驗中較不適合與其他兩者做比較。

實驗將 n 筆資料匯集分為一組，每組分別測試 5 次，求其平均時間與標準差。透過實驗結果資料，以 x 軸 (number of data responses) 50 單位區間做片段式三次內插 (Matlab 中 *pchip* 的內插方法)後，繪製成圖 31 的實驗結果。另外，表 15 資料顯示每組實驗平均結果的標準差。從圖 31 的實驗結果可推斷，*OFEI* 因為使用鏈下傳輸機制，在資料匯集的速度表現上最快。原則上，由於 *OFEI* 中，Device 傳輸資料需要事先透過 *whisper* 協議進行加密與簽章機制，故無法做到及時鏈下資料傳輸；*OEI* 資料由於透過鏈上傳輸，因此需要等待區塊驗證的時間。因此，相較於 *OFEI* 鏈下傳輸機制，當 *OEI* 交易的筆數越多，資料匯集完成所需的時間差越大；圖 31 中起始時間表示 Edge 向 Device 發出請求的 API 至 Edge 訂閱到第一筆回傳的資料的時間，可以推斷 *OFEI* 由於在智能合約中，請求的註冊與 Event 的推送邏輯較 *OEI* 複雜，因此，請求的驗證的時間可能較 *OEI* 久。

表 15 可以看出透過鏈上傳輸資料的樣式 (*OEI*)的標準差普遍大於使用鏈下(*OFEI*)傳輸的方式。以 B-IoT 開發經驗來推斷，區塊鏈每個區塊產生的時間會因共識機制的難度而有所差異。故相較於鏈下傳輸，鏈上傳輸的資料匯集時間偏差較大，且此偏差可能會因為需要匯集的資料量而增加。

DOI:10.6814/NCCU202101738

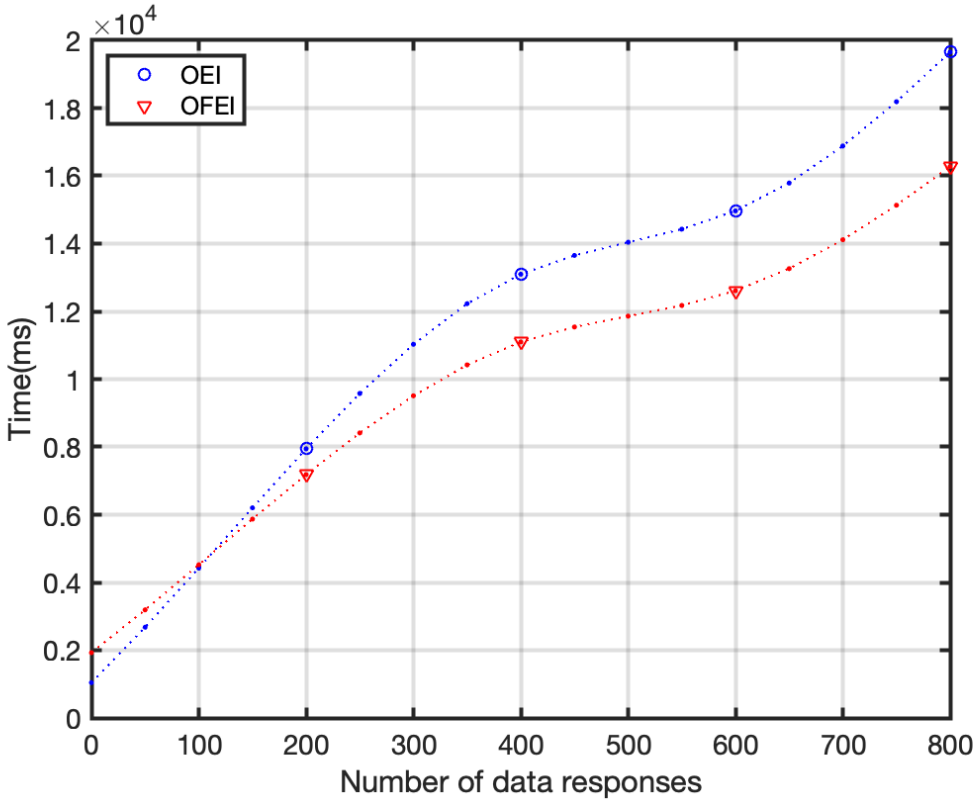

圖 31：物聯網裝置的資料匯集完成時間比較

表 15：資料匯集時間的標準差 (單位：MS)

樣式/資料數量	200	400	600	800
OEI	1638.31	2387.26	2848.16	2786.17
OFEI	**1377.85**	**716.01**	**1076.39**	**1697.14**

6.1.3 記憶體使用量比較

記憶體部分透過 npm 所提供的 pidusage [51]模組進行測量。由表 16 可以看出區塊鏈客戶端相較於後端 process 所使用的記憶體量多出許多。另一方面，Edge 與 Device 的區塊鏈客戶端所佔據的記憶體量差距極大，可以推斷是由於輕節點僅需同步區塊的標頭，導致 Edge 上所部署的區塊鏈全節點所使用的記憶體遠大於 Device 上的輕節點。另外也可以觀察到 Device 在運行不同樣式時，區塊鏈客戶

DOI:10.6814/NCCU202101738

端的記憶體使用量有明顯差別，本研究可以做出幾個推測： (1)區塊鏈輕節點需要下載後端 process 所訂閱的 Event 資料； (2)區塊的資料量會因為區塊數量變多而增加，因此，每個不同時間所測量到的記憶體使用量可能有所不同。基於以上兩點，可以推測 Device 運行 *OFEI* 時，區塊鏈客戶端所使用記憶體量較小的原因應是區塊鏈客戶端不必訂閱智能合約中的資料更新的 Event，所以記憶體使用量相對較小。綜觀記憶體的實驗結果，*OEI* 所使用的記憶體明顯相對其他樣式來的大，而 *OFEI* 則相對使用較少記憶體。

表 16：各資料匯集樣式記憶體使用量比較 (單位：MB)

樣式/測試標地名稱	Edge Backend Process	Edge Blockchain Endpoint	Device Backend Process	Device Blockchain Endpoint (light)
OEI	56.62	2063.30	44.80	380.07
ODP	55.08	1637.91	43.40	157.66
OFEI	55.01	**1597.66**	41.37	**79.54**

6.1.4 CPU 使用率比較

CPU 同樣使用上述所提到的 pidusage 模組進行測量。在表 17 中可以觀察到較顯著的部分為區塊鏈客戶端的 CPU 使用率，相較於 Device 上所部署的輕節點，Edge 上部署的全節點由於在運行挖礦機制，因此佔掉的大部分 CPU 的使用量。然而，輕節點由於不具挖礦功能，故只佔 Device 中 CPU 的 2 個百分比左右。而綜觀 CPU 的實驗結果，各樣式的差別則除了區塊鏈客戶端所表現的極大差異外，其餘差異則較不明顯。

DOI:10.6814/NCCU202101738

表 17：各資料匯集樣式 CPU 使用率比較 (單位：100%*CPU 核心個數)

樣式/測試標地名稱	Edge Backend Process (4 core)	Edge Blockchain Endpoint (4 core)	Device Backend Process (1 core)	Device Blockchain Endpoint (1 core)
OEI	0.31	**206.65**	2.83	2.06
ODP	0.31	**206.47**	4.01	2.71
OFEI	0.26	**206.69**	1.99	2.93

6.1.5 Gas 消耗量比較

區塊鏈開發人員在開發 B-IoT 同時，若所使用的區塊鏈為公有鏈，將會消耗真實的金錢。故在第 3 章中將區塊鏈的交易手續費 (gas)視為開發人員需要考量的重要 Force 之一。為了測量 gas 消耗量，本研究透過智能合約線上開發 IDE – Remix 量測在各個樣式中不同操作所消耗的 gas 量，各個樣式的測量結果如圖 32、圖 33 與圖 34 所示。以下將針對測試結果針對不同操作層面進行論述： (1)在合約部署 (*deploy*)方面，可以看出在所有樣式中，部署合約所消耗的 gas 最多。而其中 *OEI* 由於需要部署兩個不同的合約，故需要消耗最多 gas； (2)在 Device 的資料傳輸方面，由於 *OFEI* 透過鏈下傳輸，故不需要額外消耗 gas，花費因此最低。*OEI* 與 *ODP* 則在資料傳輸 (*callback*/*updateData*)部分表現相近，可以推斷透過鏈上傳輸資料所消耗的 gas 相近； (3)在資料請求發送方面，由於 *ODP* 為 Device 主動推播資料，因此不需要消耗 gas。*OFEI* (*offchainQueryRegistry)*由於需要利用智能合約儲存資料量較大的鏈下傳輸所使用的公鑰 (whisper public key)與進行較複雜的計算，所以需要消耗相對 *OEI* (*queryData*)多的 gas。

DOI:10.6814/NCCU202101738

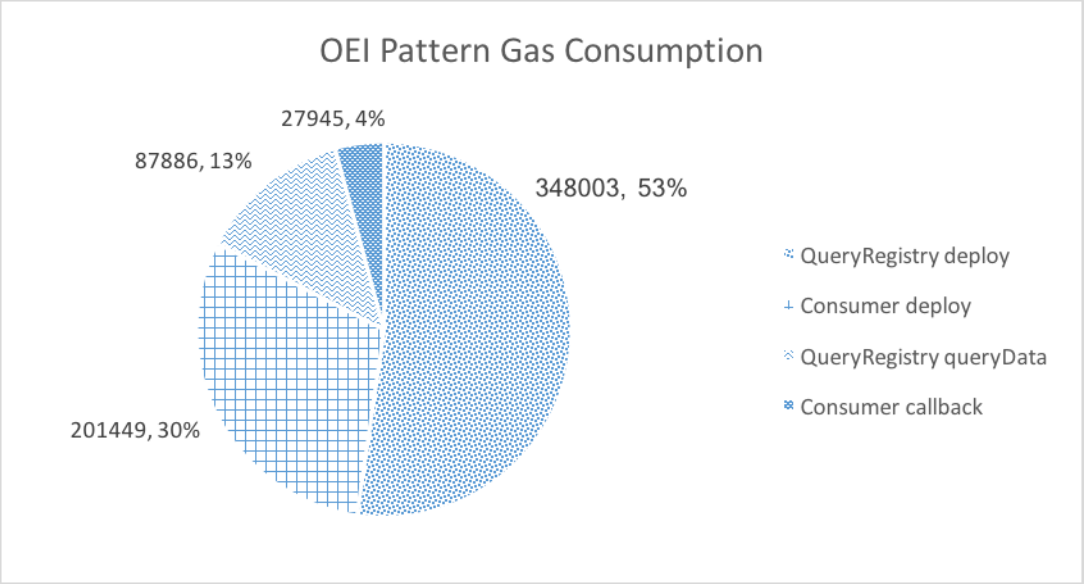

圖 32：OEI 的智能合約操作 gas 消耗量

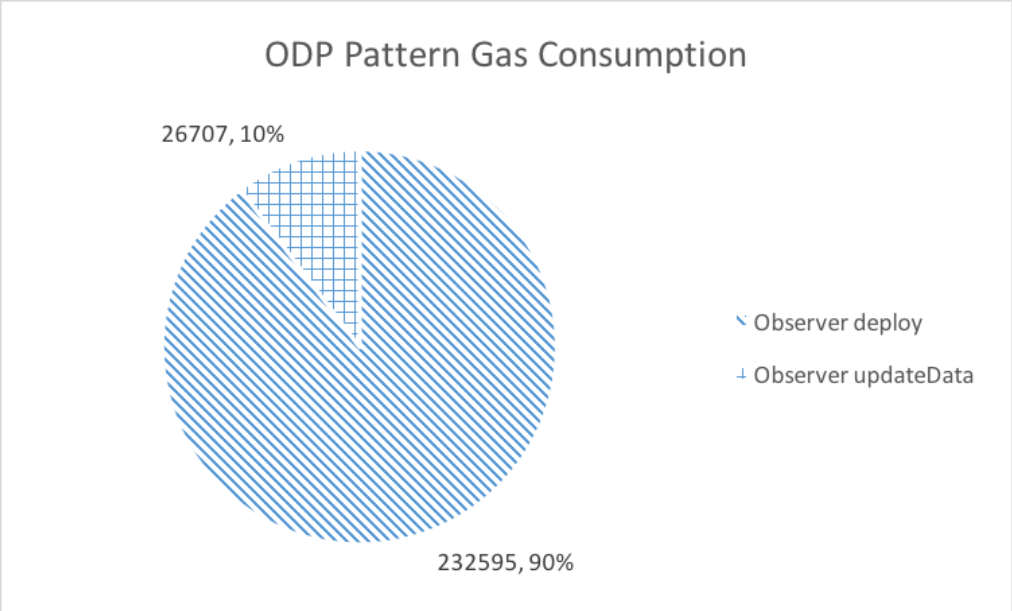

圖 33：ODP 的智能合約操作 gas 消耗量

DOI:10.6814/NCCU202101738

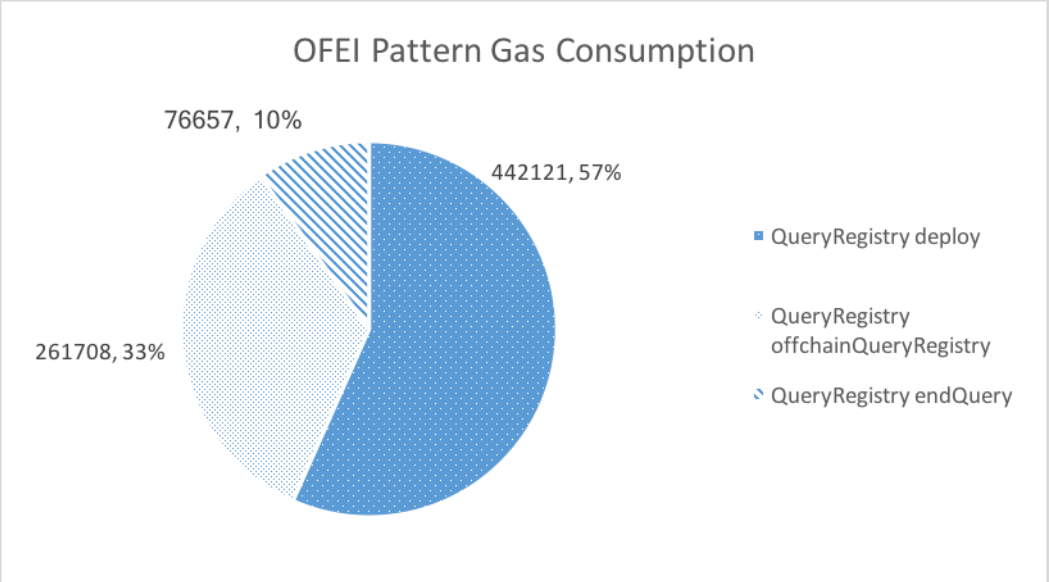

圖 34：OFEI 的智能合約操作 gas 消耗量

6.1.6 實驗結果

本節的實驗依據第 3 章所提出的 Force 設計實驗的標的，並透過實驗結果驗證各個樣式所提出的 Solution 的可行性。其中，實驗包含資料匯集的時間、記憶體與 CPU 的使用以及區塊鏈手續費的消耗，其結果為開發人員在開發設計 B-IoT 是重要的考量依據。實驗結果也顯示了不同設計樣式適用於不同的情境，*OEI* 適用於驗證與紀錄資料請求，且 Device 服務與所回傳的資料需具可追蹤性與不可篡改性；*ODP* 適用於資料可以為公開透明、不可被竄改與匯集過程不需要發出驗證資料請求節點的情境；*OFEI* 的適用情境中，在資料請求性質同 *OEI*，而匯集過程中需要確保資料的私密性、傳輸的即時性與成本的節省。

DOI:10.6814/NCCU202101738

6.2 使用者自主管理存取機制案例分析

本節從 B-UMA 中資源擁有者和資源請求方的角度，以第 5 章所提出的案例為情境，說明使用者如何使用基於 B-UMA 機制所實作的實證系統。

6.2.1 案例說明

在「智慧海運」實證系統的中，假設所有 B-IoT 中的元件，已連線網路。當船靠岸時，貨代公司負責部署 reefer 並配置其中的 Device。同時，Device 需要將其區塊鏈輕節點同步於船上的 Edge 全節點。在此之前，為了確保船上的 Edge 非惡意伺服器，貨代公司需要事先對 Edge 進行授權 (依照貨代公司所使用的鏈下授權機制授權，不在本論文討論的範圍)。授權完成後，再讓 Device 上的輕節點與船上的 Edge 之全節點同步。另一方面，實證系統必須確保所有船員、貨代公司，擁有其專屬的以太坊私鑰。待所有裝置連上區塊鏈網路，將開始進行 B-UMA 的授權流程，其詳述如下：

- **資源保護階段**

本實證系統針對使用者 (在此階段代表貨代公司)的雲端伺服器實作了 resource owner 系統，並提供了部署智能合約、註冊查詢受保護資源與相關 policy 的頁面，其使用流程如圖 37 所示。首先當使用者尚未部署 RM 合約與 Authz 合約時，可以透過選擇其以太坊帳戶來部署合約，而用來部署合約的帳戶將擁有操作智能合約的最高權限。部署完成後，使用者可以切換至註冊需受保護的資源的頁面 (如圖 35)。

第二，在註冊受保護資源時，使用者必須為受保護資源命名與設定可存取範圍 (scope)。值得一提的是，由於名稱作為資源的識別碼之一，故受保護資源名

DOI:10.6814/NCCU202101738

稱不能與別的資源名稱重複。頁面下方提供表格供使用者查看資料庫中已註冊的資訊。當使用者按下頁面中的 *add resource* 按鈕，後台系統將會把資料註冊至區塊鏈中的 RM 合約，並同時在雲端資料庫中，將註冊與 RM 合約回傳的資訊儲存。另一方面，使用者也可以重複使用此頁面註冊需受保護的資源。

第三，當完成上一步後，使用者可以更新下方表格來查詢已註冊的資訊。在特定的受保護資源欄中按下 set policy 按鈕後，會切換至設定 policy 之頁面 (如圖 36)。使用者可以透過該頁面設定受保護資源的 policy，其中包含註冊資源請求方需在請求授權階段提供的 claim，並可以在 hint 欄位輸入相關提示。其中，claim 可以視使用者的需求而定義，例如：經過編碼或雜湊後的加密訊息或是以明文顯示的使用者資訊。另外，claim 也可以利用在智能合約中實作 JWT (JSON Web Token) [52]以與鏈下的授權機制整合。再者，使用者也可以透過 *Add request party* 欄位，向智能合約設定可以存取受保護資源的使用者的以太坊帳戶。以上兩個功能執行後，後台系統皆會調用 Authz 合約中的相關功能並備份相同的輸入的資訊於雲端資料庫中。

DOI:10.6814/NCCU202101738

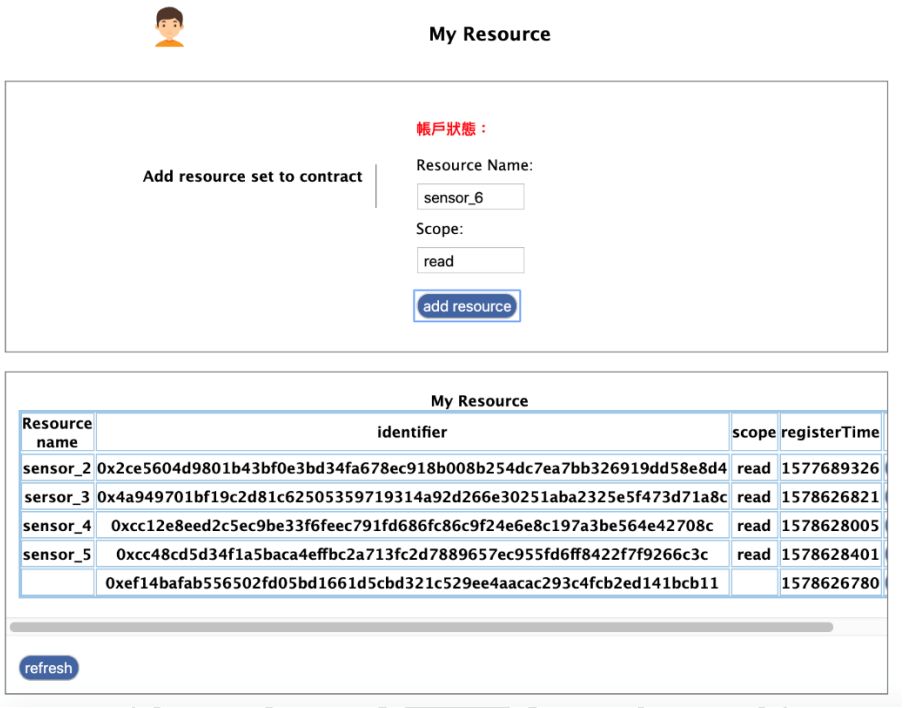

圖 35：資源擁有者註冊需受保護資源頁面

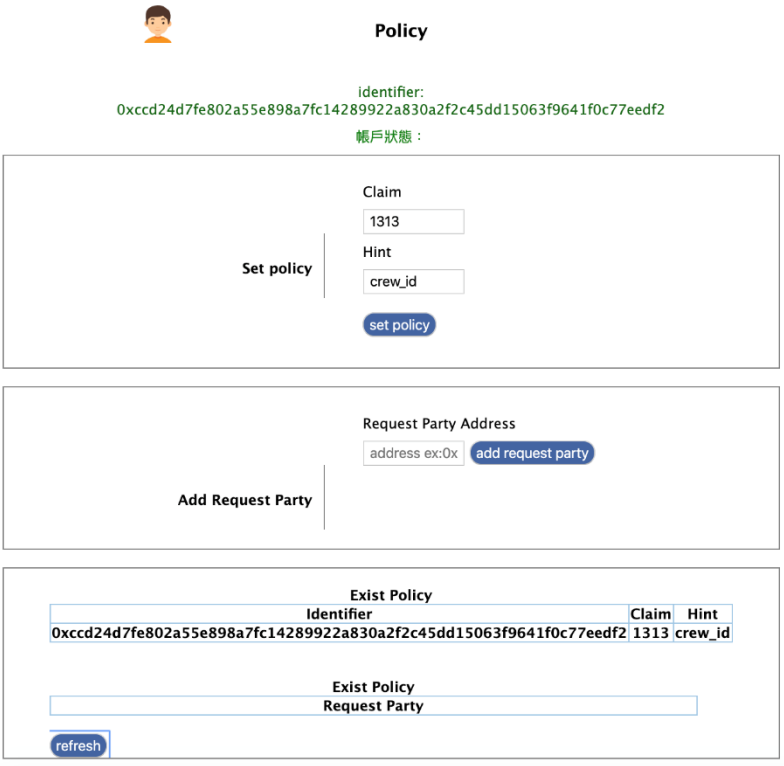

圖 36：資源擁有者設定 policy 與被授權的第三方帳戶頁面

DOI:10.6814/NCCU202101738

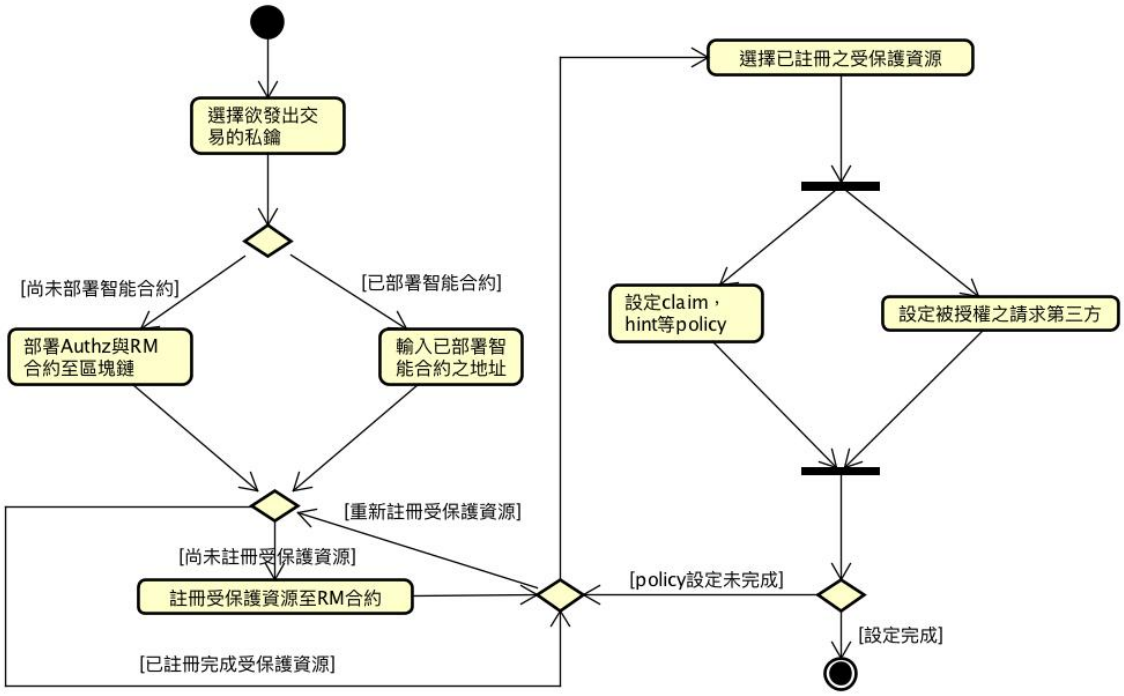

圖 37：resource owner 系統使用流程圖

- **請求授權與資源存取階段**

有別於貨代公司在雲端伺服器所架設的 resource owener 系統，本實證系統針對船上的 Edge 實作了 client 系統。使用者 (在這個階段指船員)可以透過該系統向智能合約請求授權並存取 reefer 中 Device 的受保護資源，其流程如圖 41 所示。

首先，client 系統提供了使用者存取 Device 中受保護資源的頁面 (如圖 38)。在該頁面中，使用者需要先選取其儲存於區塊鏈客戶端的專屬帳戶。之後，選取貨代公司已註冊在智能合約中的受保護資源。最後，使用者透過該頁面，驅動系統後台向 Device 發出資源請求，並開始 B-UMA 機制中的請求授權階段流程。

待 Device 成功向 Authz 合約取得 permission ticket 後，會將 ticket、Authz 合

DOI:10.6814/NCCU202101738

約地址與要求使用者提供 claim 的訊息 (hint)回傳至 client 後台系統。之後，前端頁面會要求使用者輸入 claim (如圖 39)。當使用者送出 claim 後，client 後台系統會透過區塊鏈客戶端調用 Authz 合約中取得 access token 的 fucntion。

最後，當請求授權成功時，智能合約會發送帶有 access token 的授權成功 Event。後台系統收到相關 Event 後，會透過前端頁面通知使用者授權成功訊息。在 resource owener 系統中，使用者可以在 access token 的有效期間內，多次向 Device 存取資源；若 access token 過期，前端頁面會出現 token 過期提醒，此時使用者需要重新開始此階段的授權流程 (如圖 40)。

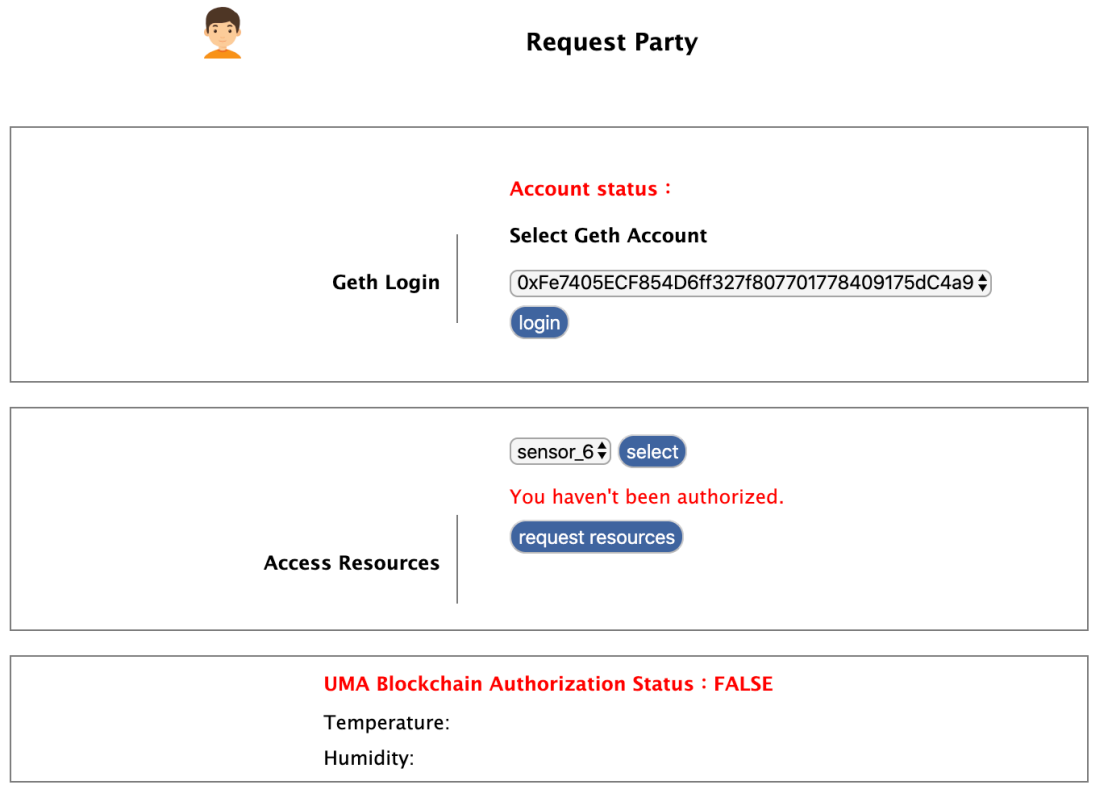

圖 38：資源請求方請求受保護資源的頁面

DOI:10.6814/NCCU202101738

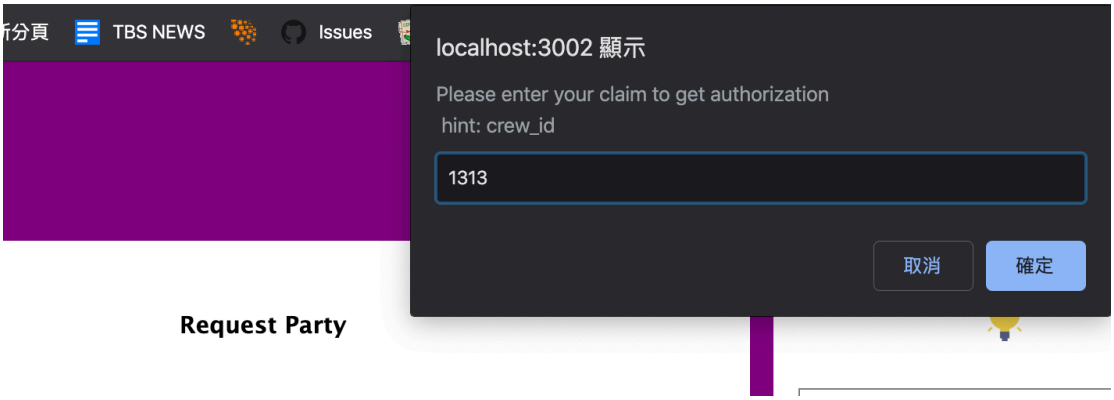

圖 39：資源請求方向 Authz 合約提供相關 claim 的提示

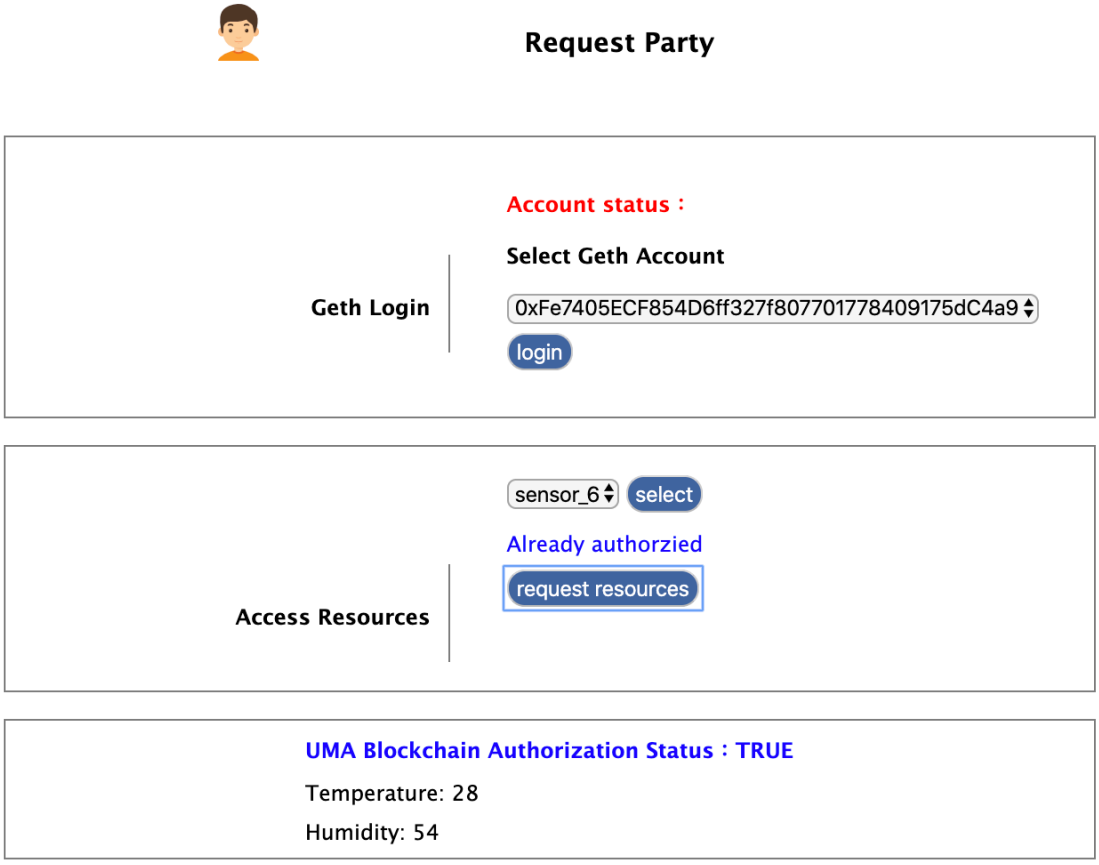

圖 40：驗證請求成功並獲取受保護資料頁面

DOI:10.6814/NCCU202101738

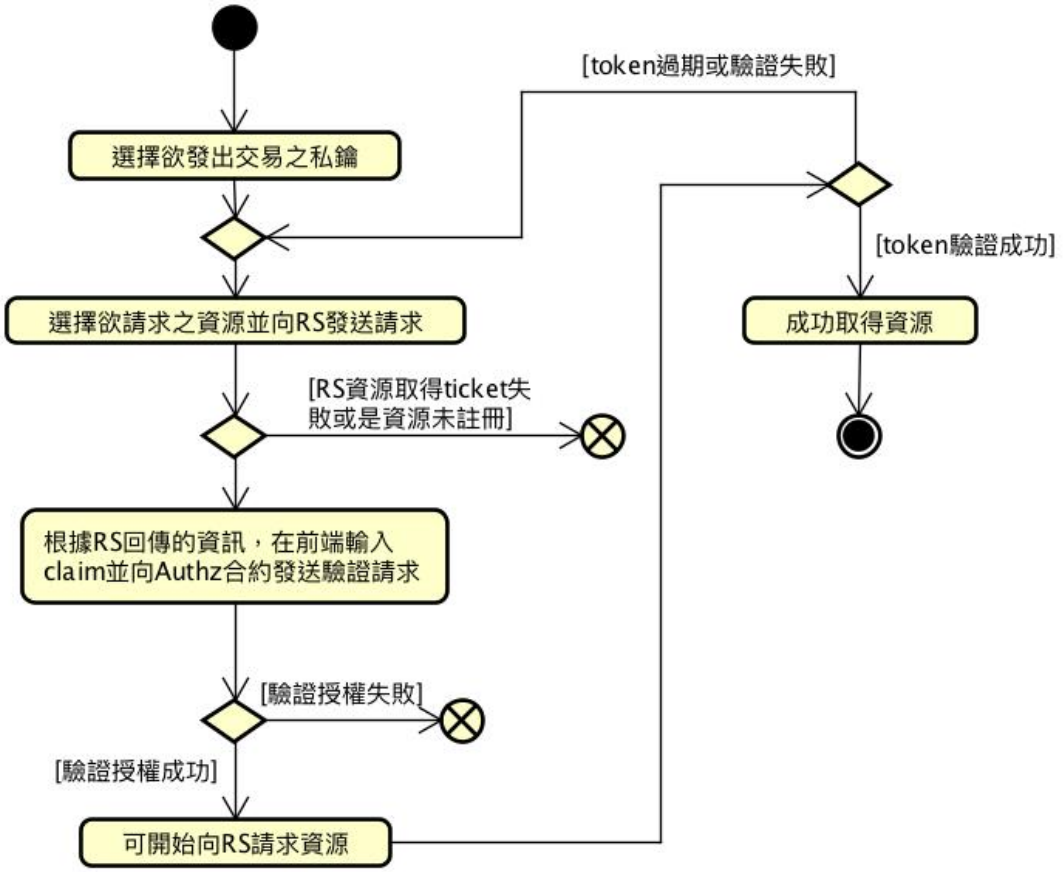

圖 41：client 系統使用流程圖

6.2.2 安全性分析

首先，在安全性測試中，本研究將區分成機密性、完整性與可用性三種類型的攻擊，並說明本論文所提出的 B-UMA 機制如何抵禦相關攻擊。

- **機密性：**
  - 惡意的第三方請求：在系統未設有安全的存取控制機制的情況下，惡意使用者可以很容易的存取 Device 服務。但透過 B-UMA 保護的資源，可以防止未經過授權的第三方存取。
  - 竊聽明文：在一般的 UMA 或 OAuth 授權機制中，資料在傳送過程透過 TLS 加密技術加密，以防止敏感訊息外洩。在 B-UMA 中，參與的節點

DOI:10.6814/NCCU202101738

可以透過區塊鏈原生公私鑰機制加密資料，以防止資料被惡意攻擊者識別。

- **完整性：**
  - 重放攻擊 (Replay attack)：在發布 access token 時，設定 token 的期限，並當 token 一過期，就將其註銷，以防止惡意的重新請求。另外，將智能合約中，permission ticket 在 access token 發布後就將其刪除 (限定只能使用一次)，也能有效預防惡意的重放攻擊。
  - 篡改授權訊息：依照第 4 章中所提出的 token 內部檢查的設計樣式，當 access token 經過簽名過後，即使篡改 token 明文，也能夠透過 token 簽名與區塊鏈的授權紀錄來驗證 token 的完整性。
  - 中間人攻擊 (Man-In-The-Middle)：第一，Client 直接從本地端向智能合約發送交易訊息，可以有效防止資料在網路傳輸過程被截取。第二，即便攻擊中利用中間人攻擊取得相關授權資料，由於在區塊鏈中，發送交易皆需要透過被授權者簽名，因此，只要被授權者的區塊鏈私鑰不外洩，攻擊者就無法以簽名者身分存取受保護的資源。
  - 偽裝成 client 以存取服務：在 OAuth2 與 UMA 中，可能面臨惡意使用者偽裝成 client，向資源伺服器存取服務。在 B-UMA 中，由於可以藉由區塊鏈帳戶來識別 client 的身分，就算偽裝者取得授權資訊，向區塊鏈發送資源請求交易，也無法通過智能合約的帳戶認證機制。
- **可用性**
  - DDoS 攻擊：受限於區塊鏈的去中心化機制，Denial-of-Service (DoS)攻擊所造成的單一節點失敗，當區塊鏈網路中挖礦節點數量越多，則對 DDoS 的抵抗則越有效。另外，由於對區塊鏈發送有效的交易需要付出

DOI:10.6814/NCCU202101738

相當高的代價，所以攻擊者向區塊鏈發送 DDoS 攻擊的意願相對降低。

第二，在靜態測試中，為了確認智能合約中不存在潛在的安全性疑慮，本研究利用 MythX [53]提供的安全性分析工具輔助分析智能合約中的安全性議題。MythX 有提供靜態與動態分析，但由於本章節主要測試目標是針對智能合約程式碼做安全性的分析，因此，本研究使用 MythX 提供的 Remix 線上合約靜態分析工具，分析實證系統中所使用的智能合約。而安全性的議題經過實作的修正過後，並未再回報安全性的議題。

6.2.3 結果與討論

本實證系統在 B-UMA 的部分模擬了第 5 章中所描述的案例，透過案例研討展示在案例中貨代公司如何透過 B-UMA 機制代理實作存取控制機制，也展示了船員如何透過B-UMA 系統請求受保護資源。本研究可以從案例研討中觀察到B-UMA 的幾點優勢：

- 授權系統的去中心化：基於實證系統中的 B-UMA 機制，B-IoT 物件能夠直接透過在本地端的區塊鏈客戶端將授權資料上鏈，避免授權資料受到第三方攔截或竄改的風險。

- 以智能合約實作的非同步授權：相較於 UMA 機制中使用授權伺服器進行請求的驗證與授權，實證系統直接透過區塊鏈中的智能合約進行去中心化且非同步的驗證與授權。如此一來，整個實證系統可以避免授權伺服器面臨單一節點失敗風險，若參與區塊鏈網路的全節點增加，系統的可用性 (availability) 與可靠性提升。

- 使用區塊鏈原生公私鑰機制：在實務上，UMA 參與者需要透過授權伺服器

DOI:10.6814/NCCU202101738

核發的 token (PAT、AAT)來證明其身分。但在 B-UMA 中，身分可以直接透過區塊鏈既有的公私鑰機制驗證。如此一來，減輕了授權系統實作 token 與管理 token 的安全性之困難度和複雜度。

- 需要穩定的硬體與網路：在 B-UMA 系統中，所有角色皆需參與區塊鏈網路，這需要建立在網路穩定與裝置足夠應付部署區塊鏈節點的所需的軟硬體規格 (例如：使用樹莓派與 4G 以上網路)。

比較與評估 B-UMA 與其他存取控制機制，是衡量本論文所提出的 B-UMA 機制的貢獻的重要指標。表 18 列出了本論文所提出的 B-UMA 機制與第 2 章中所提到的其他存取控制機制，並以第 1 章與第 4 章在論述存取控制機制時，考量到的各個面向做綜合性比較。Availability 表示能克服系統在運行時發生單點失敗，導致無法使用的程度。B-UMA 相較於其他存取控制機制，基於區塊鏈去中心化的特性，單點失敗對區塊鏈系統的影響微乎其微； Maintainability 表示當特定授權資料或流程需要異動時，系統對更新的靈活度與彈性。其中，由於 OAuth2 與 UMA 服務在普遍的應用上，皆透過中心化組織提供實作，當資源擁有者需要更改中心化系統的授權規則或授權流程時，需要考量到其他被保護資源或資源伺服器對該授權服務的依賴。而 OAuth2 相較於 UMA，缺乏對特定被保護資源的授權規則修改的彈性。另一方面，B-UMA 則可以直接透過重新部署智能合約，來達成修改授權規則與流程的目的，因此，其可維護性更勝於中心化的 UMA 服務；Transparency 表示系統的授權政策與授權過程的透明與可追蹤的程度。UMA 相較於 OAuth2，使用者較能掌握授權規則。而 B-UMA 中區塊鏈的導入，更使授權流程的透明度提升；Scalability 表示在一定時間內系統處理大量授權請求的能力。B-UMA 受限於區塊鏈共識機制的效率，在處理大量授權請求能力上的表現

DOI:10.6814/NCCU202101738

較差；複雜度 Complexity 表示在設計或開發系統時的難度、可能會消耗的時間與空間成本。其中，B-UMA 由於需要考慮區塊鏈機制與權衡其特性 (與一般授權規格、傳輸機制的差異)，因此開發難度相較於其他存取控制機制高。

表 18：存取控制機制各面向的表現比較表

	Basic	OAuth2	UMA	B-UMA
Availability	+	++	++	++++
Maintainability	+	++	+++	++++
Transparency	+	++	+++	++++
Scalability	++++	+++	++	+
Complexity	++++	+++	++	+

++++=最好，+++=好，++=差，+=最差

DOI:10.6814/NCCU202101738

第7章 結論

有鑑於區塊鏈的去中心化機制導入有助於解決物聯網所面臨安全性與裝置之間的可信任性等挑戰，B-IoT 技術日益受重視。另一方面，隨著 5G 網絡的成熟與硬體成本的降低，物聯網裝置加入區塊鏈網路的可行性提高。為了實現高品質的 B-IoT，開發人員必須謹慎的評估設計決策。然而，目前為止仍缺乏有系統的針對 B-IoT 樣式深入解析討論的文獻。本論文針對 B-IoT 中邊界伺服器與部署區塊鏈輕節點的物聯網裝置架構，並基於鏈上與鏈下的傳輸方式與邊界伺服器的主被動角色等考量，討論三種可行的資料匯集設計樣式。開發人員可以針對資料匯集過程中的安全性、成本、私密性與資源消耗等不同構面來評估合適的設計樣式。

另一方面，本論文所提出的基於區塊鏈使用者自主管理存取機制 (B-UMA) 設計樣式的主要貢獻在於： (1)增加了典型 UMA 的去中心化的特性，減少依賴中心化機制所面臨之困境； (2)由於裝置加入區塊鏈系統與操縱智能合約的可行性提升，B-UMA 系統的可維護性也隨之提升； (3)資源授權過程的公開透明與安全性增加。最後，本研究提供的「智慧海運」案例與實證系統的說明以及評估，可以提供給未來研究人員在開發類似的系統或機制時，更明確的指引。

最後，本研究中所討論的設計議題皆針對區塊鏈現有的機制與技術做討論，未來希望針對更多類型的 B-IoT 元件 (Cloud, Edge, Device)做更全面的解析討論，並期望能有更適合 B-IoT 的區塊鏈技術提出以解決時間與空間的設計議題。

DOI:10.6814/NCCU202101738

參考文獻

[1] S. Nakamoto, "Bitcoin: A peer-to-peer electronic cash system," 2008.

[2] X. Xu, I. Weber, M. Staples, L. Zhu, J. Bosch, L. Bass, C. Pautasso, and P. Rimba, "A taxonomy of blockchain-based systems for architecture design," in *2017 IEEE International Conference on Software Architecture (ICSA)*, 2017: IEEE, pp. 243-252.

[3] L. Atzori, A. Iera, and G. Morabito, "The internet of things: A survey," *Computer networks,* vol. 54, no. 15, pp. 2787-2805, 2010.

[4] P. Brody and V. Pureswaran, "Device democracy: Saving the future of the internet of things," *IBM, September,* 2014.

[5] J. G. Andrews, S. Buzzi, W. Choi, S. V. Hanly, A. Lozano, A. C. Soong, and J. C. J. I. J. o. s. a. i. c. Zhang, "What will 5G be?," vol. 32, no. 6, pp. 1065-1082, 2014.

[6] Ethereum Foundation. "Light Ethereum Subprotocol (LES)." https://github.com/zsfelfoldi/go-ethereum/wiki/Light-Ethereum-Subprotocol-%28LES%29 (accessed February 4, 2020).

[7] H. Sun, S. Hua, E. Zhou, B. Pi, J. Sun, and K. Yamashita, "Using ethereum blockchain in Internet of Things: A solution for electric vehicle battery refueling," in *International Conference on Blockchain*, 2018: Springer, pp. 3-17.

[8] F. Buschmann, R. Meunier, H. Rohnert, P. Sommerlad, and M. Stal, *Pattern-Oriented Software Architecture - Volume 1: A System of Patterns*. Wiley Publishing, 1996, p. 476.

[9] M. Wöhrer and U. Zdun, "Design patterns for smart contracts in the ethereum ecosystem," in *2018 IEEE International Conference on Internet of Things (iThings) and IEEE Green Computing and Communications (GreenCom) and IEEE Cyber, Physical and Social Computing (CPSCom) and IEEE Smart Data (SmartData)*, 2018: IEEE, pp. 1513-1520.

[10] M. Wohrer and U. Zdun, "Smart contracts: security patterns in the ethereum ecosystem and solidity," in *2018 International Workshop on Blockchain*

DOI:10.6814/NCCU202101738

Oriented Software Engineering (IWBOSE), 2018: IEEE, pp. 2-8.

[11] X. Xu, C. Pautasso, L. Zhu, Q. Lu, and I. Weber, "A Pattern Collection for Blockchain-based Applications," presented at the Proceedings of the 23rd European Conference on Pattern Languages of Programs, Irsee, Germany, 2018.

[12] J. Eberhardt and S. Tai, "On or off the blockchain? Insights on off-chaining computation and data," in *European Conference on Service-Oriented and Cloud Computing*, 2017: Springer, pp. 3-15.

[13] C.-F. Liao, C.-C. Hung, and K. Chen, "Blockchain and the Internet of Things: A Software Architecture Perspective," in *Business Transformation through Blockchain*: Springer, 2019, pp. 53-75.

[14] D. C. Schmidt, M. Stal, H. Rohnert, and F. Buschmann, *Pattern-Oriented Software Architecture, Patterns for Concurrent and Networked Objects*. John Wiley & Sons, 2013.

[15] D. Evans, "The internet of things: How the next evolution of the internet is changing everything," *CISCO white paper,* vol. 1, no. 2011, pp. 1-11, 2011.

[16] S. Z. S. Idrus, E. Cherrier, C. Rosenberger, and J.-J. Schwartzmann, "A review on authentication methods," 2013.

[17] A. Z. Ourad, B. Belgacem, and K. Salah, "Using blockchain for IOT access control and authentication management," in *International Conference on Internet of Things*, 2018: Springer, pp. 150-164.

[18] R. Almadhoun, M. Kadadha, M. Alhemeiri, M. Alshehhi, and K. Salah, "A user authentication scheme of iot devices using blockchain-enabled fog nodes," in *2018 IEEE/ACS 15th International Conference on Computer Systems and Applications (AICCSA)*, 2018: IEEE, pp. 1-8.

[19] Kantara Initiative. "User-Managed Access (UMA) Core Protocol draft-hardjono-oauth-umacore-00." https://tools.ietf.org/html/draft-maler-oauth-umagrant-00 (accessed February 4, 2020).

[20] E. Maler, "Controlling Data Usage with User-Managed Access (UMA)," in *W3C Privacy and Data Usage Control Workshop, Cambridge*, 2010.

[21] M. A. Khan and K. Salah, "IoT security: Review, blockchain solutions, and open challenges," *Future Generation Computer Systems,* vol. 82, pp. 395-411, 2018.

[22] F. L. Viktor Trón. "Go-ethereum." https://github.com/ethereum/go-ethereum (accessed February 4, 2020).

[23] V.Buterin. "A Next-Generation Smart Contract and Decentralized Application Platform." https://github.com/ethereum/wiki/wiki/White-Paper (accessed

DOI:10.6814/NCCU202101738

February 4, 2020).

[24] Ethereum Foundation. "Whisper." https://github.com/ethereum/wiki/wiki/Whisper (accessed February 4, 2020).

[25] Ethereum Foundation. "w3f." https://github.com/w3f/messaging/ (accessed February 4, 2020).

[26] A. M. Antonopoulos and G. Wood, *Mastering ethereum: building smart contracts and dapps*. O'Reilly Media, 2018.

[27] Truffle Blockchain Group. "Ganache." https://www.trufflesuite.com/ganache (accessed February 4, 2020).

[28] D. Puthal and S. P. J. I. P. Mohanty, "Proof of Authentication: IoT-Friendly Blockchains," vol. 38, no. 1, pp. 26-29, 2019.

[29] C. Alexander, *The timeless way of building*. New York: Oxford University Press, 1979.

[30] E. Gamma, *Design patterns: elements of reusable object-oriented software*. Pearson Education India, 1995.

[31] L. Cruz-Piris, D. Rivera, I. Marsa-Maestre, E. De La Hoz, and J. Velasco, "Access control mechanism for IoT environments based on modelling communication procedures as resources," *Sensors,* vol. 18, no. 3, p. 917, 2018.

[32] Kantara Initiative, "Kantara Initiative." [Online]. Available: https://kantarainitiative.org/.

[33] K. R. Özyılmaz and A. J. a. p. a. Yurdakul, "Designing a blockchain-based IoT infrastructure with Ethereum, Swarm and LoRa," 2018.

[34] M. Wöhrer and U. Zdun, "Design patterns for smart contracts in the ethereum ecosystem," 2018.

[35] OpenID Foundation. "OpenID." https://openid.net/ (accessed November 1, 2019).

[36] S. Cirani, M. Picone, P. Gonizzi, L. Veltri, and G. Ferrari, "Iot-oas: An oauth-based authorization service architecture for secure services in iot scenarios," *IEEE sensors journal,* vol. 15, no. 2, pp. 1224-1234, 2014.

[37] A. Z. Ourad, B. Belgacem, and K. Salah, "IOT Access control and Authentication Management via blockchain."

[38] V. A. Siris, D. Dimopoulos, N. Fotiou, S. Voulgaris, and G. C. Polyzos, "OAuth 2.0 meets Blockchain for Authorization in Constrained IoT Environments," *arXiv preprint arXiv:1905.01665,* 2019.

[39] N. Tapas, G. Merlino, and F. Longo, "Blockchain-based IoT-cloud authorization

DOI:10.6814/NCCU202101738

and delegation," in *2018 IEEE International Conference on Smart Computing (SMARTCOMP)*, 2018: IEEE, pp. 411-416.

[40] P. Dittmer, M. Veigt, B. Scholz-Reiter, N. Heidmann, and S. Paul, "The intelligent container as a part of the Internet of Things," in *2012 IEEE International Conference on Cyber Technology in Automation, Control, and Intelligent Systems (CYBER)*, 2012: IEEE, pp. 209-214.

[41] Chainlink Ltd SEZC. "Chainlink." https://chain.link/ (accessed February 4, 2020).

[42] A. Ekblaw, A. Azaria, J. D. Halamka, and A. Lippman, "A Case Study for Blockchain in Healthcare:"MedRec" prototype for electronic health records and medical research data," in *Proceedings of IEEE open & big data conference*, 2016, vol. 13, p. 13.

[43] E. Androulaki, A. Barger, V. Bortnikov, C. Cachin, K. Christidis, A. De Caro, D. Enyeart, C. Ferris, G. Laventman, and Y. Manevich, "Hyperledger fabric: a distributed operating system for permissioned blockchains," in *Proceedings of the Thirteenth EuroSys Conference*, 2018: ACM, p. 30.

[44] Status Research & Development GmbH. "Status." https://status.im/ (accessed February 4, 2020).

[45] brainbot labs Est. "Raiden network." https://raiden.network/ (accessed February 4, 2020).

[46] M. Jones and D. Hardt, "The oauth 2.0 authorization framework: Bearer token usage," RFC 6750, October, 2012.

[47] D. Meyer. "Sign and validate data with solidity." https://github.com/pubkey/eth-crypto/blob/master/tutorials/signed-data.md (accessed February 4, 2020).

[48] Ethereum Foundation. "Security Considerations in Solidity." https://solidity.readthedocs.io/en/v0.5.11/security-considerations.html#abstraction-and-false-positives (accessed February 4, 2020).

[49] Ethereum Foundation. "Remix." https://remix.ethereum.org/ (accessed February 4, 2020).

[50] D. Meyer. "eth-crypto." https://github.com/pubkey/eth-crypto?fbclid=IwAR0Qka4PQAAeWK95c-EQwxakZCJlpQgkac-IU-GAsQ5GUrdYq6WJsSWIqhY#txdatabycompiled (accessed February 4, 2020).

[51] A. Bluchet. "pidusage." https://www.npmjs.com/package/pidusage (accessed February 4, 2020).

DOI:10.6814/NCCU202101738

[52] OpenZeppelin. "solidity-jwt." https://github.com/OpenZeppelin/solidity-jwt (accessed February 4, 2020).

[53] MythX. "MythX." https://mythx.io/ (accessed February 4, 2020).

國立政治大學 National Chengchi University

DOI:10.6814/NCCU202101738

附錄

附錄一 相關發表著作

1. 林俊安, 廖峻鋒, "區塊鏈輕節點物聯網裝置與邊界伺服器的感測資料匯集設計樣式," 台灣軟體工程研討會 (TCSE), 桃園, 台灣, 2019. (Best Paper Award)

2. Chun-An Lin, Chun-Feng Liao and Kung Chen, " Design Patterns for Blockchain-assisted Accountable Data Dissemination between IoT Devices and Edge Server, " in *Proceedings of 9th Asian Conference on Pattern Languages of Programs* (AsianPLoP), Taipei, Taiwan, 2020.

DOI:10.6814/NCCU202101738