

# 國家科學及技術委員會補助 大專學生研究計畫研究成果報告

計畫名稱：適用於IIoT環境且具備洩漏存活的免除金鑰託管之異質簽章加密機制之設計與實作

報告類別：成果報告

執行計畫學生：王嘉羽

學生計畫編號：NSTC 112-2813-C-019-015-E

研究期間：112年07月01日至113年02月29日止，計8個月

指導教授：蔡東佐

處理方式：本計畫可公開查詢

執行單位：國立臺灣海洋大學資訊工程學系

中華民國 113年03月28日

|                           |     |
|---------------------------|-----|
| 摘要 .....                  | II  |
| ABSTRACT .....            | III |
| 誌謝 .....                  | IV  |
| 1 前言 .....                | 1   |
| 2 研究目的 .....              | 3   |
| 3 文獻探討 .....              | 5   |
| 3.1 具相等性測試的異質簽章加密機制 ..... | 5   |
| 3.2 基於憑證的密碼機制 .....       | 6   |
| 3.3 洩漏存活的密碼機制 .....       | 7   |
| 4 研究方法 .....              | 8   |
| 5 結果與討論 .....             | 10  |
| 5.1 符號 .....              | 10  |
| 5.2 具體的機制 .....           | 12  |
| 5.3 機制的正確性驗證與安全分析 .....   | 19  |
| 5.4 與其他機制的比較 .....        | 21  |
| 5.5 機制的實作與成果 .....        | 22  |
| 5.6 討論與建議 .....           | 32  |
| 6 參考文獻 .....              | 33  |
| 7 附錄 .....                | 35  |
| 7.1 完整程式碼 .....           | 35  |

## 摘要

隨著邊緣運算、雲端運算、人工智慧等 IT 領域技術發展日漸成熟，藉助數位化的力量來推動企業轉型，已是各產業共通的課題。台灣製造業面對全球市場需求快速轉變與競爭壓力，數位化與商業模式轉型儼然成為維持競爭優勢的必要發展方向。工業物聯網 (industrial internet of things, IIoT) 作為物聯網 (internet of things, IoT) 的一個分支，迅速崛起。但仍有許多機構因為安全問題，未能成功部署工業物聯網及營運技術。基於現狀，我們不經對目前 IIoT 環境產生好奇，於是開始大量瀏覽 IIoT 在資安方面的論文，目的是想了解現在 IIoT 發展到什麼程度或是還存在哪些問題可以探討。其中有一篇論文讓我們很感興趣，是在 *IEEE Internet of Things Journal* 上看到的一篇論文，名稱為適用於 IIoT 環境之具相等性測試的異質簽章加密機制 (Heterogeneous signcryption with equality test for IIoT environment, HSC-ET for IIoT environment)。異質機制指的是存在兩種以上的密碼機制，像是在 HSC-ET 機制中，他的感測器是使用公開金鑰基礎建設 (public-key infrastructure, PKI)，而使用者端則是使用基於身分的密碼機制 (identity-based cryptosystem, IBC)，異質機制支持使用者在不同機制下可以對資料進行查詢和比對。但是我們認為現有的機制有幾個可以改良的地方：

1. **金鑰託管問題:** 由於 HSC-ET 機制是使用基於身分的密碼機制，所以存在金鑰託管問題 (key escrow problems)。我們認為可以改成使用基於憑證的密碼機制 (certificate-based cryptosystem, CBC)，解決這個問題。
2. **無法抵擋旁路攻擊:** 攻擊者可以使用旁路攻擊 (side-channel attacks) 竊取部分私密金鑰的資訊。攻擊者持續收集這些部分的私密金鑰後是有可能還原完整的私鑰，導致機制失去安全性。因此，我們認為應該要建立一個可抵擋旁路攻擊的機制，亦即在部分的私密金鑰洩漏的狀況下，仍然可以確保此機制是安全的。

在本研究計畫中，我們提出一個全新的機制，名為適用於 IIoT 環境之具洩漏存活與相等性測試的異質基於憑證簽章加密機制 (Leakage-resilient heterogeneous hybrid certificate-based signcryption with equality test for IIoT environment, LR-HHCB-SCET)，除了保留原始 HSC-ET 機制在異質環境下提供相等性測試的功能之外，還解決了金鑰託管的問題且可抵擋旁路攻擊。除了提出新機制之外，本研究計畫也使用 Java 函式庫中的 Java Pairing Based Cryptography Library (JPBC) 實作此機制並放在 Github 提供開源使用。

關鍵字：工業物聯網、異質機制、相等性測試、公開金鑰基礎建設、基於憑證的密碼機制、金鑰託管問題、旁路攻擊問題

## ABSTRACT

As the technologies in the IT field, such as edge computing, cloud computing, and artificial intelligence, continue to mature, leveraging the power of digitization to drive business transformation has become a common challenge across industries. Facing rapid changes in global market demand and competitive pressures, the digitalization and transformation of business models have become essential directions for sustaining competitive advantages in the Taiwanese manufacturing industry. Industrial Internet of Things (IIoT), as a branch of the Internet of Things (IoT), has rapidly emerged. However, many organizations have faced challenges in successfully deploying IIoT and operational technologies due to security issues.

Given the current situation, we became curious about the current state of the IIoT environment and started exploring papers on the cybersecurity aspects of IIoT. Our goal was to understand the extent of IIoT development and identify any existing issues that could be explored. One paper that caught our interest was titled "Heterogeneous signcryption with equality test for IIoT environment (HSC-ET for IIoT environment)", which we found in the *IEEE Internet of Things Journal*. Heterogeneous systems refer to those involving two or more cryptographic systems. In the case of the HSC-ET mechanism, sensors operate under a public-key infrastructure (PKI), while the user end employs an identity-based cryptosystem (IBC). Heterogeneous systems support users in different systems to query and compare data. However, we believe there are several areas where the existing mechanism can be improved:

1. **Key escrow problem:** Since the system uses an identity-based cryptosystem, there is a key escrow problem. We propose switching to a certificate-based cryptosystem (CBC) to address this issue.
2. **Side-channel attacks problem:** Attackers can use side-channel attacks to steal information about partial private keys. Continuous collection of these partial private keys may lead to the reconstruction of the complete private key, compromising the security of the mechanism. Therefore, we suggest establishing a mechanism that can resist side-channel attacks, ensuring security even in the event of partial private key leakage.

In this research project, we introduce a novel mechanism called leakage-resilient heterogeneous hybrid certificate-based signcryption with equality test for IIoT environment (LR-HHCB-SCET). This mechanism not only retains the original functionality of the HSC-ET mechanism in providing equality tests in a heterogeneous system but also addresses key escrow problem and can resist side-channel attacks. In addition to proposing the new mechanism, this research project also implements it using the Java Pairing Based Cryptography Library (JPBC) and provides it as open source on GitHub.

keyword: industrial internet of things, heterogeneous system, equality test, public key infrastructure cryptosystem, certificate-based cryptosystem, key escrow problems, side-channel attacks

## 誌謝

首先感謝科技部計畫（112-2813-C-019-015-E）對本計畫的支持與研究經費補助。並感謝指導教授蔡東佐老師在計畫過程中的指導與協助，從建構方法、報告撰寫到最後的實驗，提供多面向的想法與訓練我解決問題的能力，計畫中遇到瓶頸或是無法理清思緒時，皆感謝老師慷慨地提供意見及協助，使本計畫能照進度施行。

另外，也特別感謝密碼學實驗室的學長姐們，在我初入密碼學領域時，給我很多建議與方向。當我遇到問題時，也不厭其煩地與我討論。也很高興能與實驗室進行每周一次的密碼學論文討論會，在知識方面，我從中了解到不同機制的特性與用途；在報告方面，我也學到了很多技巧與肢體表達。在此感謝國科會、蔡東佐教授與學長姐們。

王嘉羽 誌於  
國立臺灣海洋大學資訊工程學系  
中華民國一一三年三月

## 1 前言

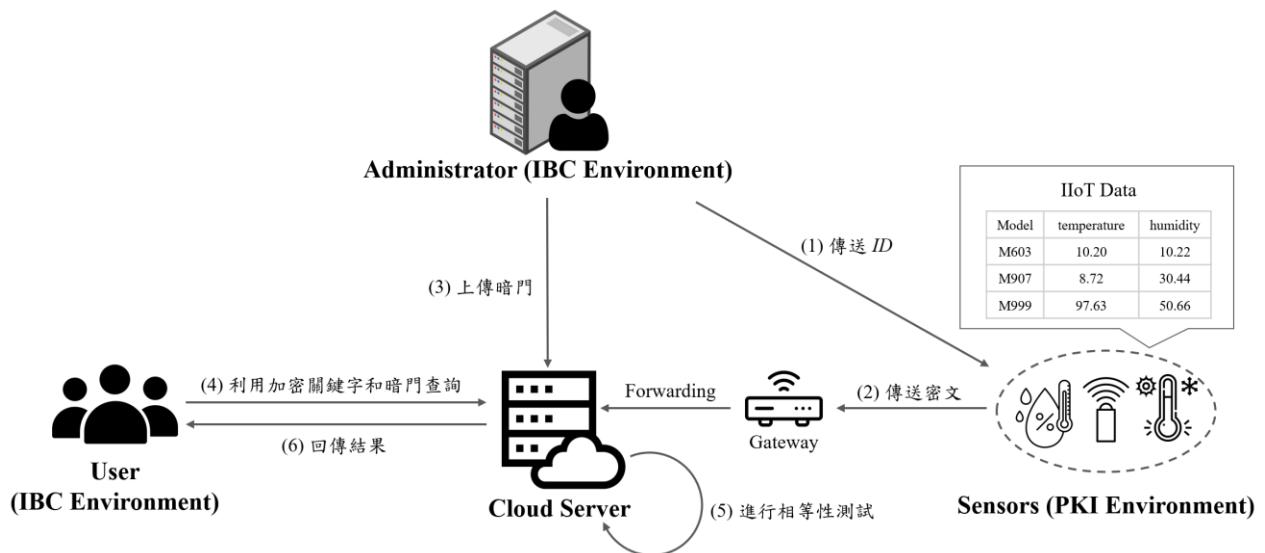
近年來，工業物聯網 (industrial internet of things, IIoT) 作為物聯網 (internet of things, IoT) 的一個分支[1]，迅速崛起，主要集中在工業領域的應用。IIoT 是一個複雜的異質 (heterogeneous) 網路，通過無線感測器網絡 (wireless sensor networks, WSNs) 連接大量的執行器 (actuators)、感測器 (sensors) 或智能設備 (smart devices) [2, 3]。在 IIoT 環境中，這些智能設備可以在最小的人為干擾 (human intervention) 下收集和交換數據，以提供方便的服務，像是生產控制、設備維護等。與傳統工業相比，IIoT 的資本和營運費用大幅減少，同時生產效率明顯提高。然而，隨著 IIoT 數據的激增，大數據儲存和處理成為不可或缺的任務。幸運的是，雲端運算技術 (cloud computing)，可以在沒有時間和地點限制的情況下，為大多數用戶提供可靠的計算服務。依靠雲端強大的計算能力，可以緩解 IIoT 設備的高儲存和計算成本。儘管雲端輔助的 IIoT 對社會和個人帶來了大量的好處，但仍需保護數據的機密性和隱私，否則會引發許多問題。

在 2022 年 7 月，有一篇有關工業物聯網的新聞指出：工業物聯網已被確認對工廠運作產生實質效益，然而，要成功部署這項技術並非易事。據報告顯示，高達 93% 的機構未能成功部署工業物聯網及相應的營運技術，其中主要原因歸咎於安全問題[4]。這則新聞引起了我們的興趣，因為根據我們先前的了解，雖然資安仍有提升空間，但並不至於只有 7% 的機構達到安全標準。於是我們開始對目前 IIoT 環境產生好奇，開始大量瀏覽 IIoT 在資安方面的論文，想要知道現在 IIoT 發展到什麼程度或是還存在哪些問題可以探討。在我們的探索中，注意到一篇發表於 *IEEE Internet of Things Journal* 的論文，標題為 適用於 IIoT 環境之具相等性測試的異質簽章加密機制 (Heterogeneous signcryption with equality test for IIoT environment, HSC-ET for IIoT environment) [5]。這篇論文引起了我們的興趣，特別是其探討在工業物聯網環境中使用異質簽章加密機制的應用。在這個機制中，異質機制指的是同時存在兩種以上的密碼機制。以 HSC-ET 機制為例，感測器採用公開金鑰基礎建設 (public-key infrastructure, PKI)，而使用者端則使用基於身分的密碼機制 (identity-based cryptosystem, IBC)，這種異質機制支持使用者在不同密碼機制下進行資料查詢和比對。

如圖一所示，HSC-ET 機制包含四個角色，分別是管理員 (administrator)、使用者、感測器和雲端伺服器。管理員和使用者都是使用基於身分密碼機制，而感測器是使用公開金鑰基礎建設，雲端伺服器則是在進行相等性測試。這些不同的角色將完成六個步驟以達成具相等性測試的異質簽章加密機制。

- (1) 管理員傳送使用者的身分給感測器，這個身分指的是使用者的 *ID* 也就是公開金鑰。目的是讓感測器知道要用誰的身分去進行加密程序。
- (2) 接下來，感測器會記錄一些 IIoT 的資料，這些資料是敏感資訊，需要被保護，感測器會用自己的私鑰對資料簽章，再用剛剛拿到的使用者 *ID* 去執行加密程序，經過簽章和加密的步驟產生密文 (ciphertext)。之後再透過中間的轉換介面 (gateway)，將密文傳送到雲端伺服器。
- (3) 這時管理員會另外上傳一個暗門 (trapdoor) 到雲端伺服器。
- (4) 使用者也會將暗門和加密後的關鍵字傳給伺服器。
- (5) 有了這些暗門和密文，雲端伺服器會進行資料比對。

(6) 最後，雲端伺服器會回傳比對成功的密文給使用者，而不是將所有密文都回傳。



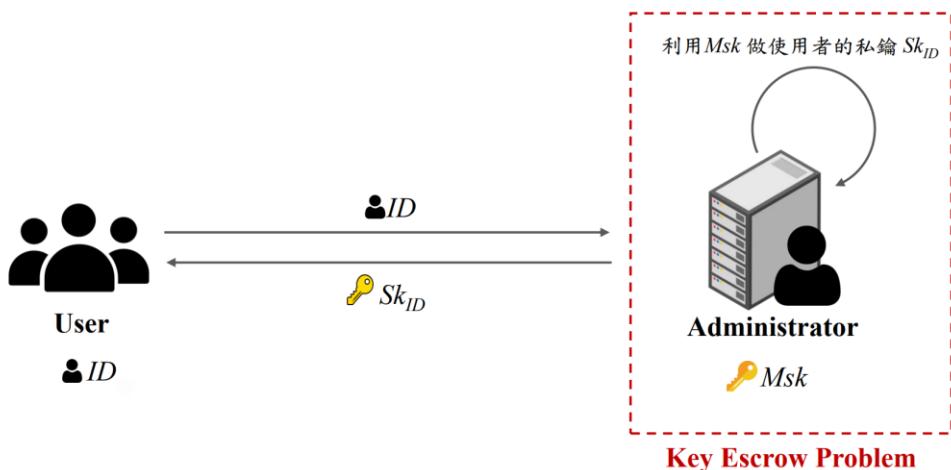
圖一、HSC-ET 機制的架構圖[5]

HSC-ET 機制是在上述架構下完成的，然而，這個機制存在兩個安全性問題，使其不具備完全的安全性。首先，問題出現在使用者所屬的基於身分的密碼機制上，IBC 以使用者的身分 ID 作為公鑰，並將此公鑰提供給管理員以取得對應的私鑰。然而，這種方式可能導致金鑰託管的問題，因為除了使用者外，第三者（管理員）也知道私鑰，這構成一種安全風險。為了解決這個金鑰託管問題，Gentry[6] 提出了基於憑證的密碼機制（certificate-based cryptosystem, CBC）。在這種機制中，使用者不再向管理員請求私鑰，而是改為請求憑證，這一改變有效地解決了金鑰託管問題。此外，HSC-ET 機制還存在第二個問題，即無法有效抵擋旁路攻擊（side-channel attacks）[7]。旁路攻擊是指攻擊者透過感知能量消耗或執行時間等途徑，取得計算操作中涉及秘密值或私鑰的部分資訊。然而，HSC-ET 機制在遭遇旁路攻擊時，無法維持其安全性。

本研究的主要目標是在原有 HSC-ET 環境的基礎上進行修改，以針對金鑰託管問題提出有效解決方案，同時強化機制對旁路攻擊的抵禦能力。透過結合基於憑證的密碼機制來替代原有的基於身分的密碼機制，我們致力於建立更安全、穩固的環境。進一步深入研究旁路攻擊防禦機制，以金鑰更新方法和技術提升機制的整體安全性，確保在資訊交換過程中保護用戶敏感資訊免於未授權存取。這項改進將有助於在工業物聯網環境中建立更為堅固的數位安全基礎。

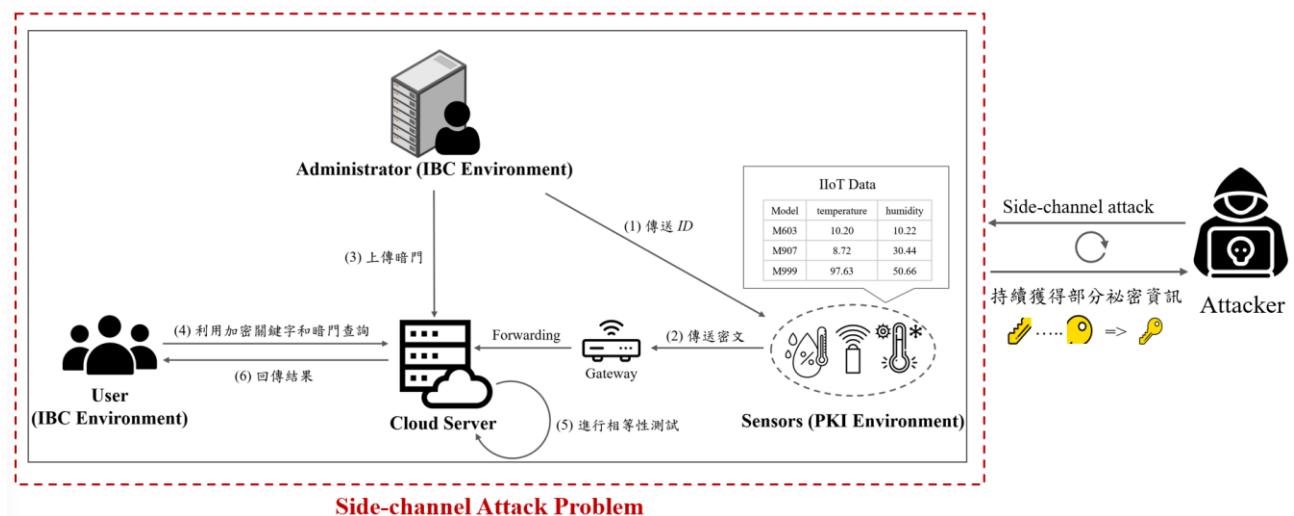
## 2 研究目的

在前述提及的 HSC-ET 機制中，我們認為還有幾個可以更完善的地方。首先，值得關注的是使用者所屬的基於身分的密碼機制。如圖二所示，IBC 的運作方式有兩個主要角色，即使用者和管理員。其中， $Msk$  代表管理員的私鑰，而  $ID$  則是使用者的身分。在 IBC 中，使用者以自身身分作為公鑰，並使用自己的身分  $ID$  向管理員請求私鑰  $Sk_{ID}$ 。然而，這帶來一個潛在的問題，即金鑰託管問題。當使用者向管理員請求私鑰時，不僅使用者本人知道私鑰，管理員也擁有每位使用者的私鑰。這種情況下，如果管理員成為攻擊者，他可能利用使用者的私鑰來取得資料。



圖二、IBC 存在金鑰託管的問題

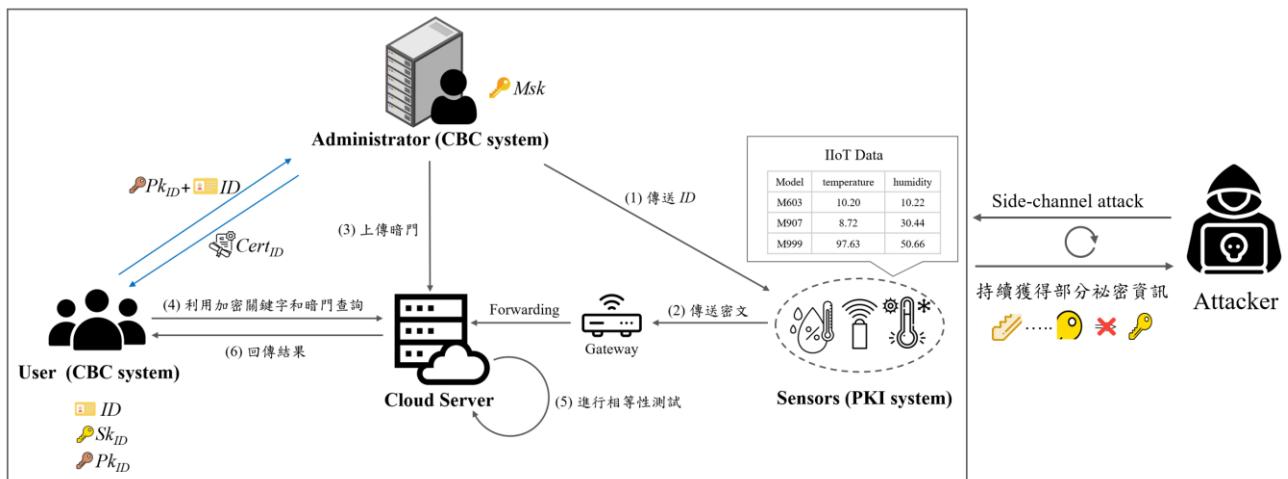
此外，HSC-ET 機制在另一方面也有改進的空間，即在抵擋旁路攻擊方面表現不佳。如圖三所示，無論是 IBC 的使用者使用私鑰解密，還是 PKI 的感測器使用私鑰進行簽章，攻擊者都有可能在此時發動旁路攻擊 (side-channel attacks)，以竊取私鑰的部分資訊。若機制未能有效抵禦旁路攻擊，導致部分私鑰洩漏給攻擊者，則攻擊者有可能在持續收集這些部分私鑰後還原出完整的私鑰。



圖三、HSC-ET 無法有效抵擋旁路攻擊

因此，本計畫的研究目的是對上述兩點進行其安全機制的研究以及設計實作，希望可以設計並實作出適用於 IIoT 環境之具洩漏存活與相等性測試的異質基於憑證簽章加密機制（Leakage-resilient heterogeneous hybrid certificate-based signcryption with equality test for IIoT environment, LR-HHCB-SCET）。以下兩點為初步擬定的解決方法：

- (1) 使用 CBC 取代 IBC，CBC 是由憑證中心（certificate authority, CA）結合公鑰和使用者的身分發放憑證，此憑證可以視為使用者的部分私密金鑰，因此使用者解密時是利用自己所選的私鑰和憑證，所以不存在金鑰託管的問題。
- (2) 提出一個可以滿足洩漏存活（leakage-resilient, LR）的機制，此機制可以抵擋旁路攻擊，亦即在部分的私密金鑰洩漏的狀況下，仍然可以確保此機制是安全的。



圖四、基於 HSC-ET 機制修改後的架構圖

圖四為我們提出基於 HSC-ET 機制修改後的架構圖，可以發現將使用者和管理員改為 CBC 後，從原本的使用身分請求私鑰，更改为使用公鑰及身分請求憑證，透過這樣的修改，再搭配使用者會自己選擇私密金鑰，管理員就不會擁有使用者完整的私鑰，可以解決金鑰託管的問題。並且我們將此機制修改為可抵擋旁路攻擊，亦即在遭遇旁路攻擊且部分私密金鑰洩漏的狀況下，仍然可以確保此機制是安全的。

表一比較了目前機制的特性。從最初的單一密碼機制，具相等性測試的公開金鑰簽章加密機制[8]（PKSCET）和具相等性測試的基於身分的簽章加密機制[9]（IBSCET）開始，進一步發展出結合 PKI 和 IBC 的機制 HSC-ET[5]。然而，HSC-ET 雖然具備異質性，卻無法避免金鑰託管的問題。此外，前述三個機制均缺乏抵抗旁路攻擊的能力。因此，我們在研究中旨在提出一個全新的機制，具備異質性、解決金鑰託管問題，同時能夠有效抵擋旁路攻擊。

表一、各機制具備的特性比較表

|        | PKSCET[8] | IBSCET[9] | HSC-ET[5] | Our LR-HHCB-SCET |
|--------|-----------|-----------|-----------|------------------|
| 異質性    | No        | No        | Yes       | Yes              |
| 金鑰系統   | PKI       | IBC       | PKI + IBC | <b>PKI + CBC</b> |
| 避免金鑰託管 | Yes       | No        | No        | <b>Yes</b>       |
| 抵擋旁路攻擊 | No        | No        | No        | <b>Yes</b>       |

### 3 文獻探討

在第二節中，我們的研究目的是進一步完善適用於工業物聯網的密碼學機制。於是，在這節中，我們將深入探討當前的機制，特別著墨於具相等性測試功能的異質簽章加密機制。同時，為了提升機制的完整性，我們也提出兩點解決方法。第一點，我們計劃將原本基於身分的密碼機制改成基於憑證的密碼機制。因此，我們將深入探討基於憑證的密碼機制，討論其原理、應用和相關研究。第二點，我們希望設計一種滿足洩漏存活的密碼機制。因此，我們也探討洩漏存活的密碼機制之概念及應用。為了完成以上兩點目標，我們將在以下 3.1 討論具相等性的異質簽章加密機制，在 3.2 討論基於憑證的密碼機制，最後在 3.3 討論洩漏存活機制概念。

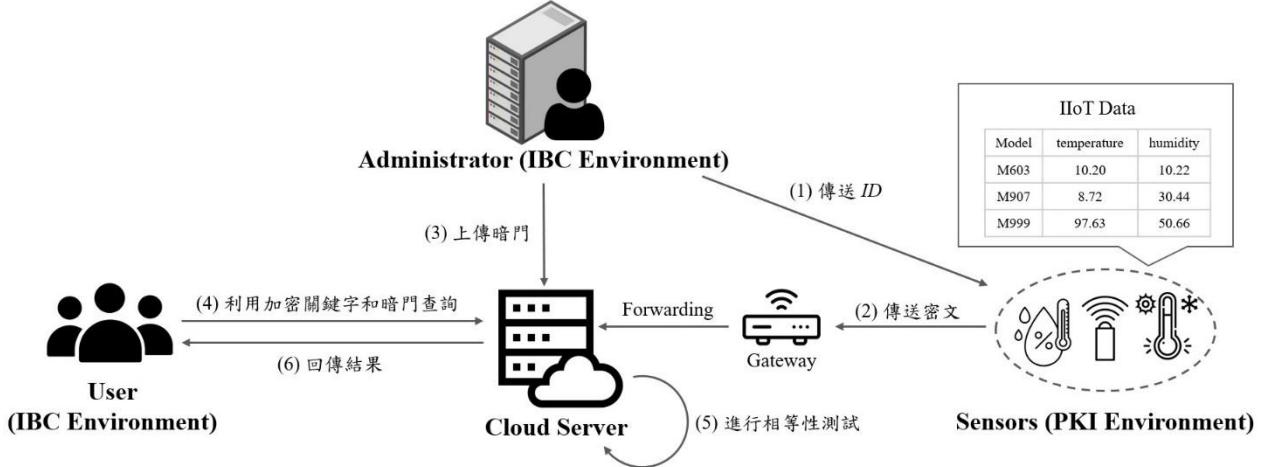
#### 3.1 具相等性測試的異質簽章加密機制 (HSC-ET)

為了確保訊息的機密性，會將訊息加密成密文後再傳輸。每位使用者都有一組對應的公鑰和私鑰，當訊息要傳輸時，傳輸者會使用接收者的公鑰進行加密，而接收者收到密文時可以用自己的私鑰去解密。雖然這樣的過程可以確保訊息的機密性，但是接收者是無法確定這個訊息的來源。因此，在傳輸訊息之前，傳輸者還必須執行簽章的程序，以確保這份密文的傳輸者是誰。當接收者收到這一份簽章加密的密文時，會先進行認證的程序以確定訊息的來源，最後再使用自己的私鑰對密文進行解密。

上述的過程中，加密是用接收者的公鑰，簽章是用傳輸者私鑰。異質簽章加密機制指的就是，在這個機制下存在兩種不同的密碼機制。舉例來說，傳輸者是使用公開金鑰基礎建設，接收者則是使用基於身分的密碼機制，當一份訊息要簽章加密時，會使用接收者 (PKI) 的公鑰加密，再用傳輸者 (IBC) 的私鑰進行簽章。這種先由 PKI 加密再由 IBC 簽章的方式，就可以稱為異質簽章加密機制。

最近，為了使簽章加密機制能夠更好地應用於異質系統，如 IIoT、智慧都市、車聯網等，提出了一系列異質系統的簽章加密機制。Sun 和 Li[10]製定了一種異質簽章加密機制，其中 PKI 中的使用者和 IBC 中的使用者可以相互發送訊息。在[10]的研究之後，Huang 等人[11]構建了一個異質簽章加密機制，該機制只允許將訊息從 IBC 傳遞到 PKI。Li 等人[12]為異質系統提供了兩種簽章加密構造。在裡面可以實現 PKI 和 IBC 中訊息的雙向傳輸。

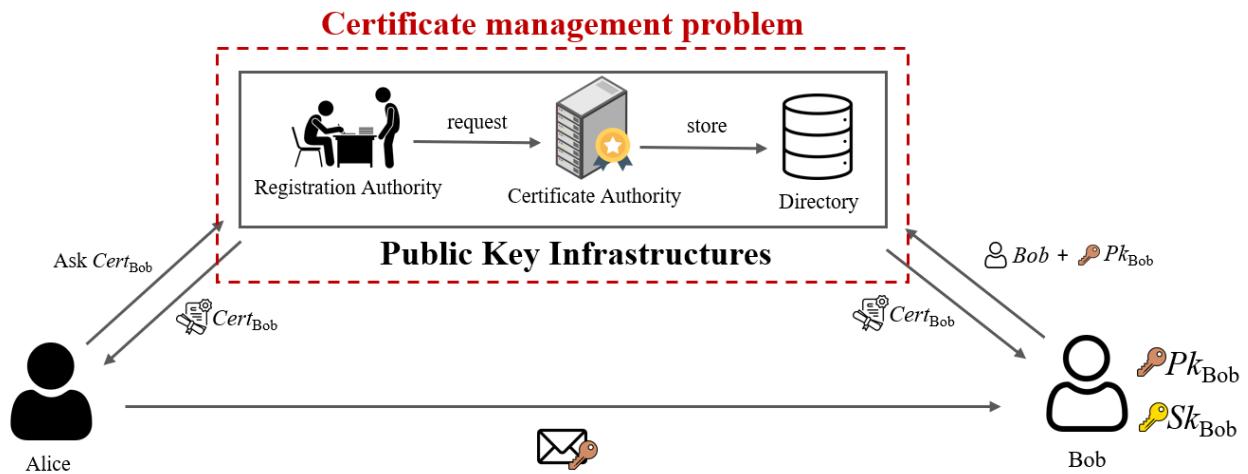
異質簽章加密機制還能有更多元的應用，像是相等性測試。具相等性測試的異質簽章加密機制是指在兩種不同的密碼機制下，可以將不同密碼機制加密的密文進行比對或查詢。如圖五所示，管理員和使用者都是使用 IBC，而感測器是使用 PKI，雲端伺服器則是將這兩種不同密碼機制產生的密文進行比對，亦即相等性測試。



圖五、具相等性測試的異質簽章加密機制

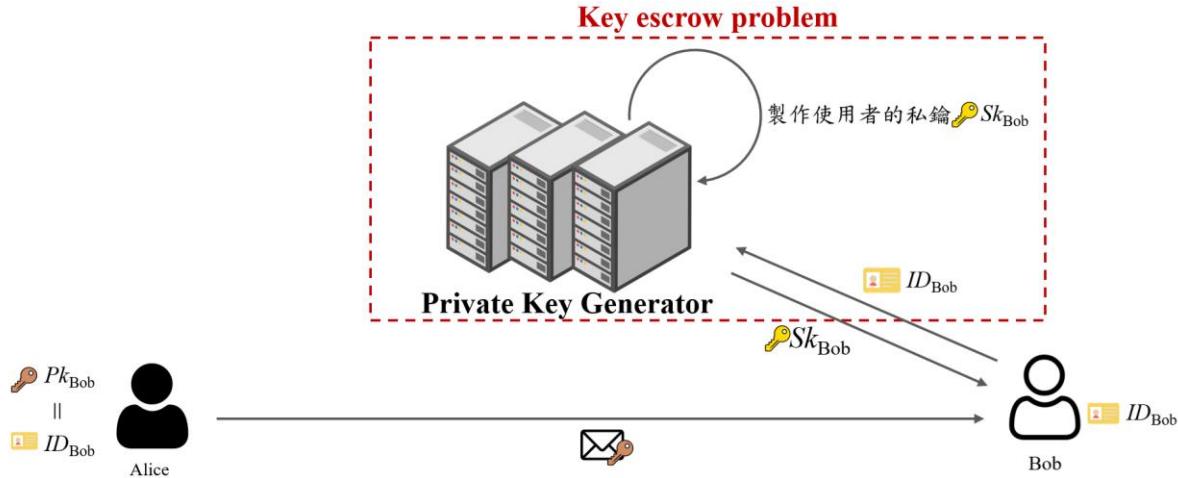
### 3.2 基於憑證的密碼機制 (CBC)

如圖六，在傳統的公開金鑰 (public-key, PK) 機制中[13]，每位使用者會有一組由亂數產生的公鑰和對應的私鑰。由於公鑰是亂數產生，因此不具任何意義。為了讓使用者身分和公鑰產生連結，使用者 (Bob) 會利用自己的公鑰 ( $Pk_{Bob}$ ) 和自己的身分向公開金鑰基礎建設註冊憑證 (certificate)。當其他使用者 (Alice) 要傳送資訊時，會向 PKI 請求對方的憑證 ( $Cert_{Bob}$ )，透過此憑證獲得對方的公鑰 ( $Pk_{Bob}$ )，並用此公鑰對訊息進行加密，之後再傳送密文給對方 (Bob)。但這樣公鑰基礎建設必須經過多道手續才能註冊憑證並且還需管理所有使用者的憑證，會耗費大量的成本，此問題為憑證管理問題。



圖六、傳統的公開金鑰機制

產生憑證管理問題的主因是由於原本的公鑰不具意義，為了使公鑰和使用者身分產生連結，才需註冊憑證。基於這個公鑰不具意義的問題，之後提出了基於身分的密碼機制，如圖七。在這個機制下，使用有意義的身分 ( $ID$ ) 代替原本由亂數產生、不具意義的公鑰。這樣在傳輸訊息時，使用者 (Alice) 可以用對方的身分 ( $ID_{Bob}$ ) 作為公鑰來加密訊息再傳送給對方。然而，因為使用者是使用自己的身分向私鑰生成器 (private key generator, PKG) 拿取私鑰，所以 PKG 會知道所有使用者的私鑰，這會造成金鑰託管問題 (key escrow problem)。



圖七、基於身分的密碼機制

為了解決金鑰託管的問題，Gentry[6] 提出了基於憑證的密碼機制概念，如圖八。在 CBC 中，有兩個角色，即使用者和可信賴的憑證機構（certificate authority, CA），使用者首先選擇一個私鑰並產生相應的公鑰。然後，將使用者身分和對應的公鑰給 CA。透過使用者的身分和公鑰，CA 會計算並發送相關的憑證給使用者。而當其他使用者（Alice）要傳送訊息時，會使用公鑰 ( $Pk_{Bob}$ ) 和身分 ( $ID_{Bob}$ ) 進行加密，而對方收到密文時是使用私鑰 ( $Sk_{Bob}$ ) 和憑證 ( $Cert_{Bob}$ ) 去解密。



圖八、基於憑證的密碼機制

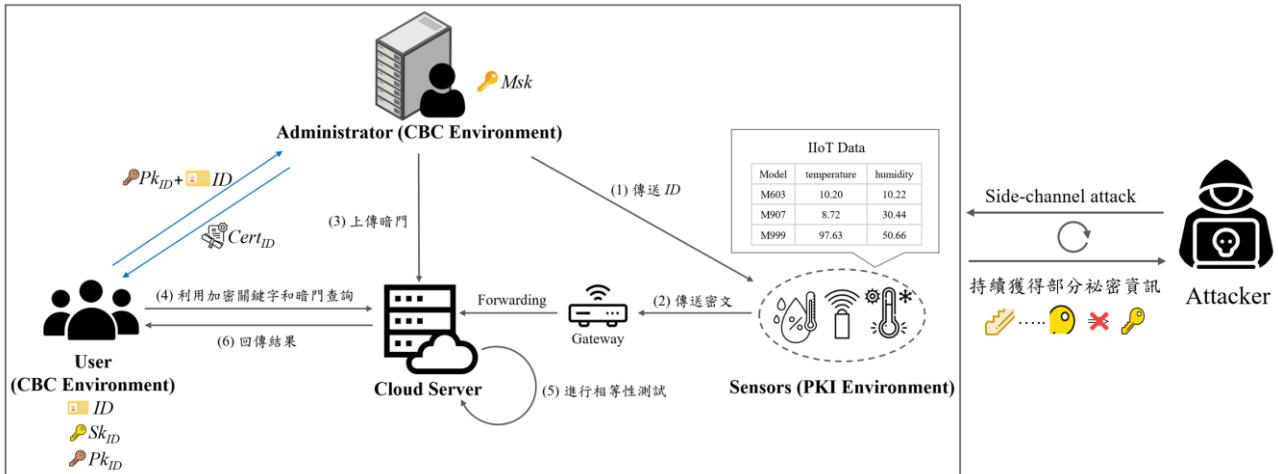
### 3.3 洩漏存活的密碼機制

一般而言，這些密碼機制的安全性是建立在理想模型上，意思是私密金鑰對攻擊者來說是絕對安全的，不允許攻擊者知道任何私密金鑰的資訊。但在現實情況下，攻擊者可以使用旁路攻擊[7]，因此理想的模型並不適用，旁路攻擊是指攻擊者可以透過感知能量消耗或執行時間來獲得計算操作中涉及這些秘密值的部分資訊。在這種情況下，這些基於理想模型的密碼系統可能會遭受到這種旁路攻擊，是不安全的。

最近，可以抵擋旁路攻擊的洩漏存活密碼學研究是一個新興的研究主題。這些洩漏存活密碼機制允許攻擊者獲得部分的私密金鑰，同時保持這些洩漏存活密碼機制的安全性。在過去十年中，已經提出了許多基於傳統公鑰設置的洩漏存活密碼機制，例如洩漏存活簽章機制（leakage-resilient signature schemes）[14] 和洩漏存活加密機制（leakage-resilient encryption schemes）[15]。

## 4 研究方法

經過閱讀眾多文獻後，我們了解到目前在 IIoT 環境中的設計仍有改進的空間，以使機制更加完備。舉例而言，前述提到的基於身分的密碼機制中存在金鑰託管的問題。此外，在遇到旁路攻擊時，無法確保機制的安全性。基於以上兩點，我們在研究目的中已提出一個新的架構，如圖九，將管理員和使用者皆由 IBC 轉換為基於憑證的密碼機制，以解決金鑰託管問題並賦予機制洩漏存活的特性。參考 HSC-ET 機制的論文，我們學到了從設計方法到實作的整體流程，並依循以下四大步驟完成此機制的研究。



圖九、基於 HSC-ET 機制修改後的架構圖

- ✓ **步驟一、定義演算法:** 我們新提出的環境是在原有環境的基礎上進行修改，因此我們學習原本環境的設計步驟。先定義出環境中角色的演算法，我們將在第一步驟重新定義出管理員、使用者（CBC）、雲端系統以及感測器相關的演算法，以適應我們新的環境。由於原本環境是使用 IBC，我們希望能夠解決金鑰託管問題，因此我們將其改為使用 CBC。在第一步驟定義演算法中，我們定義出了 5 個角色、6 個階段和 7 個算法，分別為：
  - **5 個角色:** PKI 的憑證中心、PKI 的實體、CBC 的憑證中心、CBC 的實體、雲端伺服器。
  - **6 個階段:** 系統設定階段 (*System setup phase*)、實體金鑰生成階段 (*Entity keys generation phase*)、簽章加密階段 (*Signcryption phase*)、解密驗證階段 (*Unsigncryption phase*)、暗門生成階段 (*Trapdoor generation phase*) 與相等性測試階段 (*Equality test phase*)。
  - **7 個算法:** 系統設定算法 (*Setup*)、自有私鑰生成算法 (*SelfKeyGen*)、憑證生成算法 (*CrtGen*)、簽章加密 (*Signcryption*)、解密驗證算法 (*Unsigncryption*)、暗門生成算法 (*TrapdoorGeneration*) 與相等性測試算法 (*Equality test*)。
- ✓ **步驟二、提出具體方法:** 在這一階段，我們著手思考如何實現這個機制，並根據定義的演算法，深入研究與探討以下五個方向，以提出具體方法：
  - **加密與簽章加密算法的區別:** 由於我們是簽章加密機制，因此閱讀了加密和簽章加密機制的文獻[16, 17]，針對它們之間的區別進行研究，並加以利用。

- **CBC 的建構:** 考慮到我們的機制將使用 CBC 取代原本 HSC-ET 機制中的 IBC，因此我們研讀了 CBC 的論文[18]，深入了解其他學者是如何建構 CBC 的。
  - **異質性的研究:** 由於我們的機制結合了 PKI 與 CBC，除了學習建構 CBC 外，還參考了其他具有異質機制的文獻[5, 19]，學習並研究如何將兩種密碼機制結合使用。
  - **抵抗旁路攻擊的方法:** 我們的目標是提出一個能夠抵擋旁路攻擊的機制，所以瀏覽了許多抵擋旁路攻擊的論文[16, 17, 18, 19]。我們研究與學習他們的算法，並加以改良以符合我們的機制。
  - **支援異質性的相等性測試:** 由於最初的相等性測試是建立在同質環境上，不符合我們的需求。因此，參考了多篇具有異質性相等性測試的文獻，如 PKI 和 IBC 的相等性測試[5]、PKI 和免憑證的密碼機制 (certificateless cryptosystem, CLC) 的相等性測試[19]。參考他們建構的算法，並建構出支援 PKI 和 CBC 的異質性相等性測試。
- ✓ **步驟三、提出安全分析:** 最後，我們將對所提出的具體方法進行安全分析，安全分析的項目大致分為三個方向。透過這三個方向的安全分析，我們將證明所提出的機制具有安全性。
- **方法中的式子推導:** 透過橢圓曲線上的加法、純量乘法以及雙線性映射算式，我們將證明所提方法中式子的正確性。
  - **方法中的機密性與不可偽造性:** 利用數學證明中的反證法以及橢圓曲線離散對數問題 (elliptic curve discrete logarithm problem, ECDLP) [20]，我們將證明所提機制加密後的密文具有機密性，且實體的簽章具有不可偽造性。
  - **方法可抵抗旁路攻擊:** 透過金鑰更新算法，我們將證明在遭遇旁路攻擊的情況下，所提機制仍能保持系統的安全性。
- ✓ **步驟四、實作:** 在完成相關的安全機制後，我們選擇使用 Java 程式語言進行實作，並選用 IntelliJ IDEA 社群版作為開發環境。為了實現橢圓曲線上的運算，我們將在網路上下載 Java Pairing Based Cryptography (JPBC) 函式庫的開源檔案 (.jar 檔)，並導入到 IDEA 中，以完成環境建置的步驟。環境建置完成後，我們將進行系統和元素的初始化。JPBC 的元素生成順序為: Pairing ->  $G_1, G_2, G_T, Z_{q^*}$  (Field) -> Element。換言之，首先生成 Pairing 物件，主要根據雙線性對的類型生成 Field，最後使用 Field 來生成 Element。初始化完成之後，我們將著手規劃整個系統的架構，並根據架構繪製 UML 類別圖。確認沒有問題後，即可按照架構進行實作。

藉由以上四步驟，我們的研究目標是希望實作出適用於 IIoT 環境之具洩漏存活與相等性測試的異質基於憑證簽章加密機制(LR-HHCB-SCET)，以改善目前現有機制之不足的地方。

## 5 結果與討論

在本節中，我們將介紹我們的成果與相關討論。首先，在 5.1 小節中，我們將介紹建構機制會使用到的符號，以幫助讀者更好地理解後續的內容。接著，在 5.2 小節中，我們將呈現 LR-HHCB-SCET 機制的具體算法，讓讀者能夠深入了解我們的機制。在讀者了解機制細節後，我們將在 5.3 節中討論機制的安全性及相關證明。隨後，在 5.4 節中，我們將與其他機制進行比較，以突顯我們機制的重要性與實用性。在 5.5 節中，我們將簡要說明使用 Java 搭配 JPBC 函式庫實作的過程及成果。最後，在 5.6 節中，我們將進行討論與建議，反思機制上仍有哪些地方可以進一步完善。

### 5.1 符號

在這小節中，我們將介紹 LR-HHCB-SCET 中符號對應的含義，希望藉由表格幫助讀者更好了解後續內容。在表二中描述了機制算法定義的符號，左邊表示符號，右邊表示符號的含意。

表二、符號定義

| 符號  | 含義  |
|---|---|
| $SPP$   | 系統公開參數                                      |
| $G, G_T$  | 橢圓曲線上的循環群 (cyclic group)                    |
| $\hat{e}$   | 雙線性映射 $\hat{e}: G \times G \rightarrow G_T$ |
| $q$   | 很大的質數，群的階 (order of group)                  |
| $P$   | $G$ 群的生成元 (generator)                       |
| $S, T, U_1, U_2, V_1, V_2$                                  | $G$ 群上的隨機點                                  |
| $SE$  | 對稱加密  |
| $SD$  | 對稱解密  |
| $HF_0, HF_1, HF_2, HF_3, HF_4,$<br>$HF_5, HF_6, HF_7, HF_8$ | 單向雜湊函數 (one-way hash function)              |
| PB-PKS  | 基於公開金鑰基礎建設的公開金鑰系統                           |
| CB-PKS  | 基於憑證的公開金鑰系統                                 |
| $C_{APB}$   | PB-PKS 的憑證中心                                |
| $PBPK_{CA}$   | $C_{APB}$ 的公鑰                               |
| $PBSK_{CA}$   | $C_{APB}$ 的私鑰                               |
| $(PBSK_{CA,0,0}, PBSK_{CA,0,1})$                            | $C_{APB}$ 的初始私鑰對                            |
| $(PBSK_{CA,i,0}, PBSK_{CA,i,1})$                            | $C_{APB}$ 在第 $i$ 次迭代時的私鑰對                   |
| $C_{ACB}$   | CB-PKS 的憑證中心                                |
| $CBPK_{CA}$   | $C_{ACB}$ 的公鑰                               |
| $CBSK_{CA}$   | $C_{ACB}$ 的私鑰                               |
| $(CBSK_{CA,0,0}, CBSK_{CA,0,1})$                            | $C_{ACB}$ 的初始私鑰對                            |
| $(CBSK_{CA,i,0}, CBSK_{CA,i,1})$                            | $C_{ACB}$ 在第 $i$ 次迭代時的私鑰對                   |

|  |  |
|--|--|
| $ID_{PB}$  | 屬於 PB-PKS 的實體的身分                             |
| $PBPK_{ID_{PB}}^{1st}$                                   | 身分為 $ID_{PB}$ 實體的第一把公鑰                       |
| $PBSK_{ID_{PB}}^{1st}$                                   | 身分為 $ID_{PB}$ 實體的第一把私鑰                       |
| $(PBSK_{ID_{PB},0,0}^{1st}, PBSK_{ID_{PB},0,1}^{1st})$   | 身分為 $ID_{PB}$ 實體的第一把私鑰的初始私鑰對                 |
| $(PBSK_{ID_{PB},j,0}^{1st}, PBSK_{ID_{PB},j,1}^{1st})$   | 身分為 $ID_{PB}$ 實體的第一把私鑰在第 $j$ 次迭代時私鑰對         |
| $PBPK_{ID_{PB}}^{2nd}$                                   | 身分為 $ID_{PB}$ 實體的第二把公鑰                       |
| $PBSK_{ID_{PB}}^{2nd}$                                   | 身分為 $ID_{PB}$ 實體的第二把私鑰                       |
| $(PBSK_{ID_{PB},0,0}^{2nd}, PBSK_{ID_{PB},0,1}^{2nd})$   | 身分為 $ID_{PB}$ 實體的第二把私鑰的初始私鑰對                 |
| $(PBSK_{ID_{PB},j,0}^{2nd}, PBSK_{ID_{PB},j,1}^{2nd})$   | 身分為 $ID_{PB}$ 實體的第二把私鑰在第 $j$ 次迭代時私鑰對         |
| $PBCRT_{ID_{PB}}$  | 身分為 $ID_{PB}$ 實體的憑證                          |
| $PBTD_{ID_{PB}}$   | 身分為 $ID_{PB}$ 實體的暗門                          |
| $ID_{CB}$  | 屬於 CB-PKS 的實體的身分                             |
| $CBPK_{ID_{CB}}^{1st}$                                   | 身分為 $ID_{CB}$ 實體的第一把公鑰                       |
| $CBSK_{ID_{CB}}^{1st}$                                   | 身分為 $ID_{CB}$ 實體的第一把私鑰                       |
| $(CBSK_{ID_{CB},0,0}^{1st}, CBSK_{ID_{CB},0,1}^{1st})$   | 身分為 $ID_{CB}$ 實體的第一把私鑰的初始私鑰對                 |
| $(CBSK_{ID_{CB},j,0}^{1st}, CBSK_{ID_{CB},j,1}^{1st})$   | 身分為 $ID_{CB}$ 實體的第一把私鑰在第 $j$ 次迭代時私鑰對         |
| $CBPK_{ID_{CB}}^{2nd}$                                   | 身分為 $ID_{CB}$ 實體的第二把公鑰                       |
| $CBSK_{ID_{CB}}^{2nd}$                                   | 身分為 $ID_{CB}$ 實體的第二把私鑰                       |
| $(CBSK_{ID_{CB},0,0}^{2nd}, CBSK_{ID_{CB},0,1}^{2nd})$   | 身分為 $ID_{CB}$ 實體的第二把私鑰的初始私鑰對                 |
| $(CBSK_{ID_{CB},j,0}^{2nd}, CBSK_{ID_{CB},j,1}^{2nd})$   | 身分為 $ID_{CB}$ 實體的第二把私鑰在第 $j$ 次迭代時私鑰對         |
| $CBPK_{ID_{CB}}^{3rd}$                                   | 身分為 $ID_{CB}$ 實體的第三把公鑰                       |
| $CBCRT_{ID_{CB}}^{1st}$                                  | 身分為 $ID_{CB}$ 實體的第一個憑證                       |
| $(CBCRT_{ID_{CB},0,0}^{1st}, CBCRT_{ID_{CB},0,1}^{1st})$ | 身分為 $ID_{CB}$ 實體的第一個憑證的初始憑證對                 |
| $(CBCRT_{ID_{CB},j,0}^{1st}, CBCRT_{ID_{CB},j,1}^{1st})$ | 身分為 $ID_{CB}$ 實體的第一個憑證在第 $j$ 次迭代時憑證對         |
| $CBCRT_{ID_{CB}}^{2nd}$                                  | 身分為 $ID_{CB}$ 實體的第二個憑證                       |
| $(CBCRT_{ID_{CB},0,0}^{2nd}, CBCRT_{ID_{CB},0,1}^{2nd})$ | 身分為 $ID_{CB}$ 實體的第二個憑證的初始憑證對                 |
| $(CBCRT_{ID_{CB},j,0}^{2nd}, CBCRT_{ID_{CB},j,1}^{2nd})$ | 身分為 $ID_{CB}$ 實體的第二個憑證在第 $j$ 次迭代時憑證對         |
| $CBTD_{ID_{CB}}^{1st}, CBTD_{ID_{CB}}^{2nd}$             | 身分為 $ID_{CB}$ 實體的暗門對                         |
| $ID_s$   | 發送者 (可能是 $ID_{PB}$ 或是 $ID_{CB}$ )            |
| $ID_r$   | 接收者 (可能是 $ID_{PB}$ 或是 $ID_{CB}$ )            |
| $ID_A$   | 身分為 $ID_A$ 的實體 (可能是 $ID_{PB}$ 或是 $ID_{CB}$ ) |
| $ID_B$   | 身分為 $ID_B$ 的實體 (可能是 $ID_{PB}$ 或是 $ID_{CB}$ ) |
| $msg$  | 明文訊息   |
| $CT$   | 密文   |

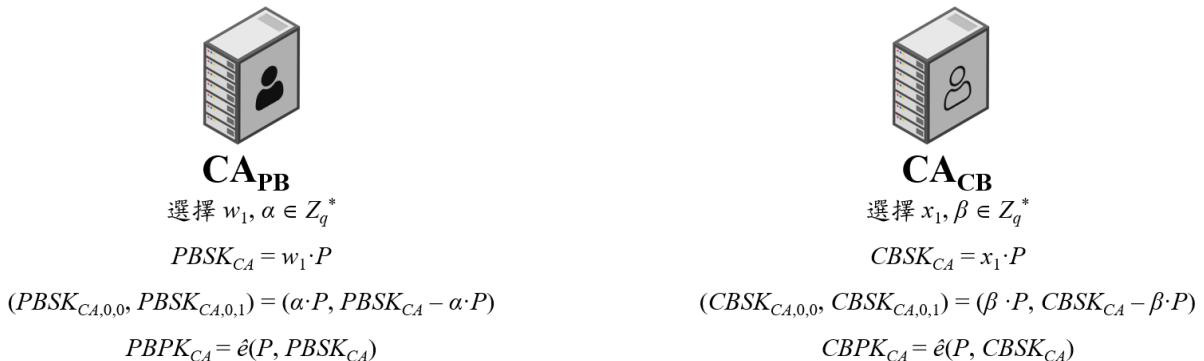
## 5.2 具體的機制

現有的機制，亦即適用於 IIoT 環境之具相等性測試的異質簽章加密機制(HSC-ET for IIoT environment)，存在著金鑰託管問題與無法抵擋旁路攻擊的缺點。在本章節中，我們將解決這兩個缺點並保留原始機制的優點提出一個新的機制，我們稱之為適用於 IIoT 環境之具洩漏存活與相等性測試的異質基於憑證簽章加密機制(LR-HHCB-SCET)。具體的機制包含如下的六個階段：

- **系統設定階段 (System setup phase)**：系統公開參數  $SPP$  包含一個雙線性配對的集合  $\{G, G_T, \hat{e}, q, P\}$ 、六個從  $G$  中隨機選取的點  $S, T, U_1, U_2, V_1, V_2 \in G$ 、對稱式加密/解密  $SE/SD$  演算法與九個雜湊函數  $HF_0: \{0, 1\}^* \times G_T \times G_T \times G \rightarrow \{0, 1\}^u$ 、 $HF_1: \{0, 1\}^* \times G_T \times G_T \times G \rightarrow \{0, 1\}^u$ 、 $HF_2: G_T \times G_T \rightarrow \{0, 1\}^u$ 、 $HF_3: G_T \rightarrow G$ 、 $HF_4: G_T \times G_T \times G_T \times G \rightarrow \{0, 1\}^u$ 、 $HF_5: G_T \times G_T \rightarrow G$ 、 $HF_6: \{0, 1\}^u \rightarrow \{0, 1\}^u$ 、 $HF_7: \{0, 1\}^* \rightarrow G$  與  $HF_8: G \times G \times \{0, 1\}^* \rightarrow \{0, 1\}^u$ ，其中  $u$  是一個整數常數。因此，完整的系統公開參數  $SPP = \{G, G_T, \hat{e}, q, P, S, T, U_1, U_2, V_1, V_2, SE/SD, HF_0, HF_1, HF_2, HF_3, HF_4, HF_5, HF_6, HF_7, HF_8\}$ 。我們提出的異質系統包含基於公開金鑰基礎建設(PKI-based, PB)與基於憑證(certificate-based, CB)。在 PB 公開金鑰系統(PB public key system, PB-PKS)中，有一個憑證中心，我們稱之為  $CA_{PB}$ 。同樣的，在 CB 公開金鑰系統(CB public key system, CB-PKS)中，也有一個憑證中心，我們稱之為  $CA_{CB}$ 。此外，如下所示， $CA_{PB}$  與  $CA_{CB}$  亦會生成屬於自己的私鑰與公鑰，完整的系統設定階段可參考圖十。
- PB-PKS: 憑證中心  $CA_{PB}$  隨機挑選一個數值  $w_1 \in Z_q^*$ ，並計算其在 PB-PKS 中的私鑰  $PBSK_{CA} = w_1 \cdot P$  與公鑰  $PBPK_{CA} = \hat{e}(P, PBSK_{CA})$ 。此外，憑證中心  $CA_{PB}$  會隨機挑選一個數值  $\alpha \in Z_q^*$ ，並計算  $(PBSK_{CA,0,0}, PBSK_{CA,0,1}) = (\alpha \cdot P, PBSK_{CA} - \alpha \cdot P)$ 。
- CB-PKS: 憑證中心  $CA_{CB}$  隨機挑選一個數值  $x_1 \in Z_q^*$ ，並計算其在 CB-PKS 中的私鑰  $CBSK_{CA} = x_1 \cdot P$  與公鑰  $CBPK_{CA} = \hat{e}(P, CBSK_{CA})$ 。此外，憑證中心  $CA_{CB}$  會隨機挑選一個數值  $\beta \in Z_q^*$ ，並計算  $(CBSK_{CA,0,0}, CBSK_{CA,0,1}) = (\beta \cdot P, CBSK_{CA} - \beta \cdot P)$ 。

### 系統公開參數 $SPP$

$$\{G, G_T, \hat{e}, q, P, S, T, U_1, U_2, V_1, V_2, SE/SD, HF_0, HF_1, HF_2, HF_3, HF_4, HF_5, HF_6, HF_7, HF_8\}$$

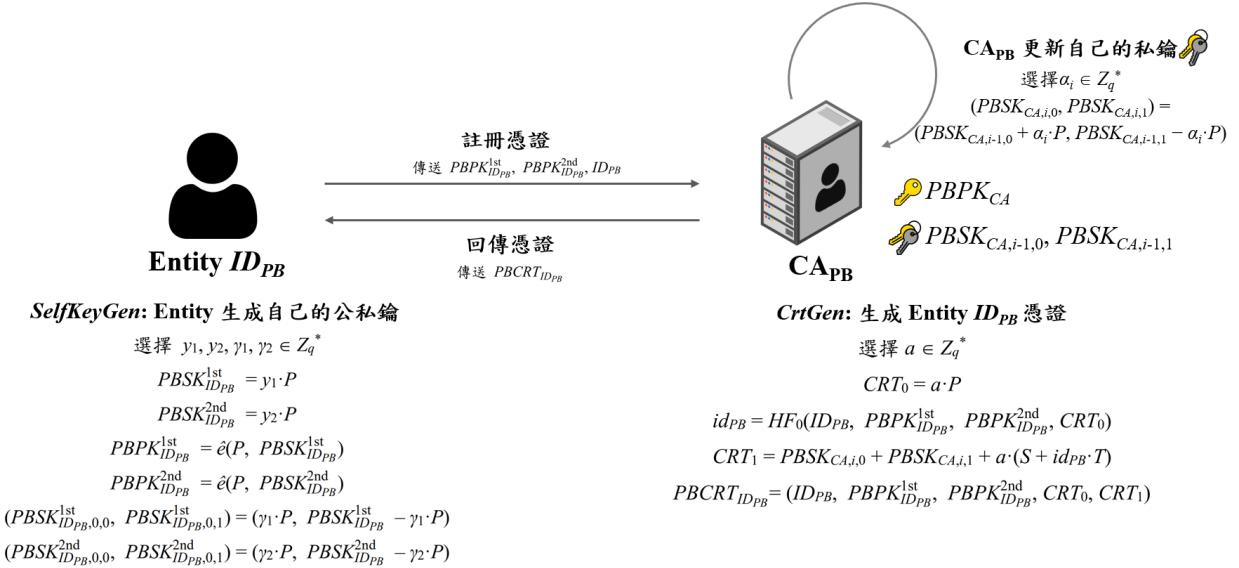


圖十、系統設定階段

- 實體金鑰生成階段 (*Entity keys generation phase*)：我們提出的異質環境包含 PB-PKS 與 CB-PKS。因此，實體可能是 PB-PKS 或 CB-PKS 的成員。接下來，我們將呈現針對新的實體成員在 PB-PKS 與 CB-PKS 中的金鑰生成過程。

➤ PB-PKS: 對於具有身分  $ID_{PB}$  的實體，其實體金鑰生成過程包含以下兩個演算法。圖十一為 PB-PKS 的實體金鑰生成階段的流程。

- ✓ PB-PKS 自有金鑰生成 (*PB-PKS SelfKeyGen*)：實體  $ID_{PB}$  隨機選取兩個數值  $y_1, y_2 \in Z_q^*$ ，並且計算兩個私鑰  $PBSK_{ID_{PB}}^{1st} = y_1 \cdot P$ ,  $PBSK_{ID_{PB}}^{2nd} = y_2 \cdot P$  與兩個公鑰  $PBPK_{ID_{PB}}^{1st} = \hat{e}(P, PBSK_{ID_{PB}}^{1st})$ ,  $PBPK_{ID_{PB}}^{2nd} = \hat{e}(P, PBSK_{ID_{PB}}^{2nd})$ 。同時，實體  $ID_{PB}$  隨機選取另外兩個數值  $\gamma_1, \gamma_2 \in Z_q^*$ ，並計算  $(PBSK_{ID_{PB},0,0}^{1st}, PBSK_{ID_{PB},0,1}^{1st}) = (\gamma_1 \cdot P, PBSK_{ID_{PB}}^{1st} - \gamma_1 \cdot P)$ ,  $(PBSK_{ID_{PB},0,0}^{2nd}, PBSK_{ID_{PB},0,1}^{2nd}) = (\gamma_2 \cdot P, PBSK_{ID_{PB}}^{2nd} - \gamma_2 \cdot P)$ 。
- ✓ PB-PKS 憑證生成 (*PB-PKS CrtGen*)：在這個演算法的第  $i$  次執行中，憑證中心  $CA_{PB}$  隨機挑選一個數值  $\alpha_i \in Z_q^*$ ，並計算  $(PBSK_{CA,i,0}, PBSK_{CA,i,1}) = (PBSK_{CA,i-1,0} + \alpha_i \cdot P, PBSK_{CA,i-1,1} - \alpha_i \cdot P)$ 。當接收到實體成員的兩個公鑰  $PBPK_{ID_{PB}}^{1st}, PBPK_{ID_{PB}}^{2nd}$  與其身分  $ID_{PB}$  後，憑證中心  $CA_{PB}$  使用  $(PBSK_{CA,i,0}, PBSK_{CA,i,1})$  生成並送發實體成員的憑證  $PBCRT_{ID_{PB}} = (ID_{PB}, PBPK_{ID_{PB}}^{1st}, PBPK_{ID_{PB}}^{2nd}, CRT_0, CRT_1)$  給實體成員，其中可以透過以下方法計算  $CRT_0$  與  $CRT_1$ 。憑證中心  $CA_{PB}$  會隨機挑選一個數值  $a \in Z_q^*$ ，並計算  $CRT_0 = a \cdot P$  與  $CRT_1 = PBSK_{CA,i,0} + TC$ ，其中  $TC = PBSK_{CA,i,1} + a \cdot (S + id_{PB} \cdot T)$  且  $id_{PB} = HF_0(ID_{PB}, PBPK_{ID_{PB}}^{1st}, PBPK_{ID_{PB}}^{2nd}, CRT_0)$ 。

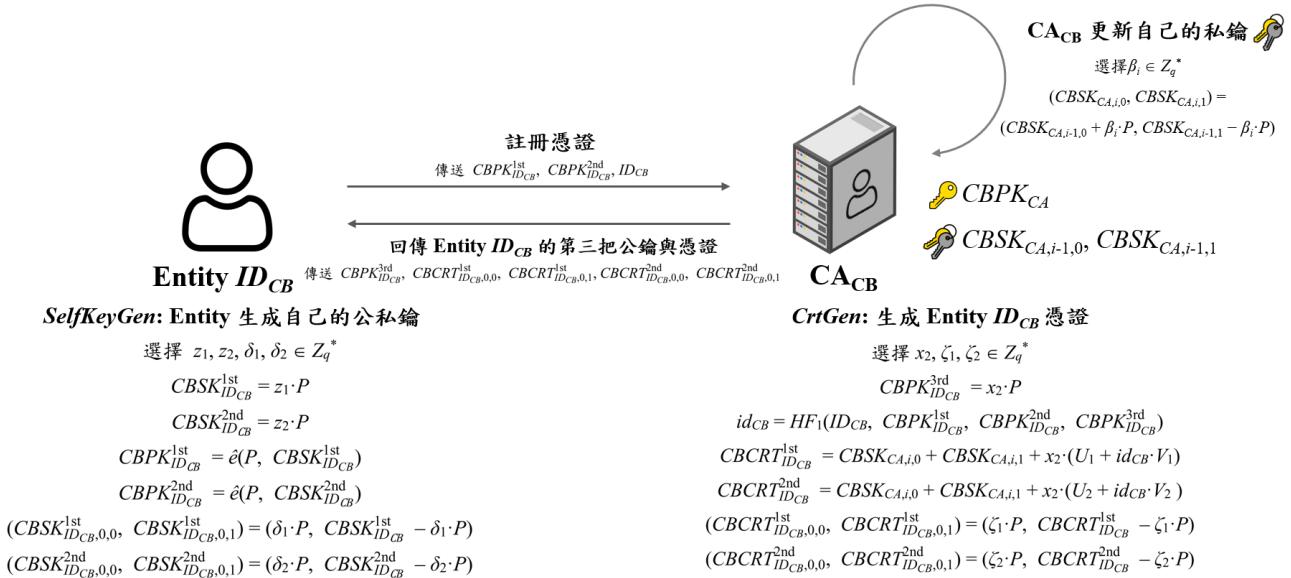


圖十一、PB-PKS 的實體金鑰生成階段的流程

➤ CB-PKS: 對於具有身分  $ID_{CB}$  的實體，其實體金鑰生成過程包含以下兩個演算法。圖十二為 CB-PKS 的實體金鑰生成階段的流程。

- ✓ CB-PKS 自有金鑰生成 (*CB-PKS SelfKeyGen*)：實體  $ID_{CB}$  隨機選取兩個數值  $z_1, z_2 \in Z_q^*$ ，並且計算兩個私鑰  $CBSK_{ID_{CB}}^{1st} = z_1 \cdot P$ ,  $CBSK_{ID_{CB}}^{2nd} = z_2 \cdot P$  與兩個公鑰  $CBPK_{ID_{CB}}^{1st} = \hat{e}(P, CBSK_{ID_{CB}}^{1st})$ ,  $CBPK_{ID_{CB}}^{2nd} = \hat{e}(P, CBSK_{ID_{CB}}^{2nd})$ 。同時，實體  $ID_{CB}$  隨機選取另外兩個數值  $\delta_1, \delta_2 \in Z_q^*$ ，並計算  $(CBSK_{ID_{CB},0,0}^{1st}, CBSK_{ID_{CB},0,1}^{1st}) = (\delta_1 \cdot P, CBSK_{ID_{CB}}^{1st} - \delta_1 \cdot P)$ ,  $(CBSK_{ID_{CB},0,0}^{2nd}, CBSK_{ID_{CB},0,1}^{2nd}) = (\delta_2 \cdot P, CBSK_{ID_{CB}}^{2nd} - \delta_2 \cdot P)$ 。

✓ CB-PKS 憑證生成 (*CB-PKS CrtGen*)：在這個演算法的第  $i$  次執行中，憑證中心 CA<sub>CB</sub> 隨機挑選一個數值  $\beta_i \in Z_q^*$ ，並計算  $(CBSK_{CA,i,0}, CBSK_{CA,i,1}) = (CBSK_{CA,i-1,0} + \beta_i \cdot P, CBSK_{CA,i-1,1} - \beta_i \cdot P)$ 。當接收到實體成員的兩個公鑰  $CBPK_{ID_{CB}}^{1st}$ ,  $CBPK_{ID_{CB}}^{2nd}$  與其身分  $ID_{CB}$  後，憑證中心 CA<sub>CB</sub> 使用  $(CBSK_{CA,i,0}, CBSK_{CA,i,1})$  生成並送發實體成員的公鑰  $CBPK_{ID_{CB}}^{3rd} = x_2 \cdot P$  與兩個憑證  $CBCRT_{ID_{CB}}^{1st} = CBSK_{CA,i,0} + TK_1$ ,  $CBCRT_{ID_{CB}}^{2nd} = CBSK_{CA,i,0} + TK_2$ ，其中  $x_2 \in Z_q^*$  且  $TK_1 = CBSK_{CA,i,1} + x_2 \cdot (U_1 + id_{CB} \cdot V_1)$ ,  $TK_2 = CBSK_{CA,i,1} + x_2 \cdot (U_2 + id_{CB} \cdot V_2)$ ，而  $id_{CB} = HF_1(ID_{CB}, CBPK_{ID_{CB}}^{1st}, CBPK_{ID_{CB}}^{2nd}, CBPK_{ID_{CB}}^{3rd})$ 。同時，實體成員  $ID_{CB}$  會隨機挑選兩個數值  $\zeta_1, \zeta_2 \in Z_q^*$ ，並計算  $(CBCRT_{ID_{CB},0,0}, CBCRT_{ID_{CB},0,1}) = (\zeta_1 \cdot P, CBCRT_{ID_{CB}}^{1st} - \zeta_1 \cdot P)$  與  $(CBCRT_{ID_{CB},0,0}, CBCRT_{ID_{CB},0,1}) = (\zeta_2 \cdot P, CBCRT_{ID_{CB}}^{2nd} - \zeta_2 \cdot P)$ 。



圖十二、CB-PKS 的實體金鑰生成階段的流程

- 簽章加密階段 (*Signcryption phase*)：針對這個簽章加密 (*signcryption, SC*) 演算法的第  $j$  次執行中，發送者  $ID_s$  執行這個演算法並生成密文傳送給接收者  $ID_r$ 。SC 演算法的輸入為訊息  $msg$ 、發送者身分  $ID_s$ 、接收者身分  $ID_r$ 、發送者私鑰  $SK_s$  與接收者公鑰  $PK_r$ ，其中根據發送者  $ID_s$  所屬的系統， $SK_s$  可能為  $(PBSK_{ID_{sj},0}^{1st}, PBSK_{ID_{sj},1}^{1st}, PBSK_{ID_{sj},0}^{2nd}, PBSK_{ID_{sj},1}^{2nd})$  或  $(CBSK_{ID_{sj},0}^{1st}, CBSK_{ID_{sj},1}^{1st})$ ,  $(CBSK_{ID_{sj},0}^{2nd}, CBSK_{ID_{sj},1}^{2nd})$ ,  $(CBCRT_{ID_{sj},0}^{1st}, CBCRT_{ID_{sj},1}^{1st})$ ,  $(CBCRT_{ID_{sj},0}^{2nd}, CBCRT_{ID_{sj},1}^{2nd})$ ，而根據接收者  $ID_r$  所屬的系統， $PK_r$  可能為  $PBPK_{ID_r}^{1st}, PBPK_{ID_r}^{2nd}$  或  $CBPK_{ID_r}^{1st}, CBPK_{ID_r}^{2nd}, CBPK_{ID_r}^{3rd}$ 。SC 演算法的示意圖如圖十三所示，SC 演算法的輸出為密文  $CT$ ，詳細的流程如下面步驟：

- (1) 發送者隨機挑選一個數值  $h \in Z_q^*$ ，並計算  $CT_0 = h \cdot P$ 。
- (2) 發送者會根據接收者所屬的系統計算  $H$  與  $TT$ 。

- 當接收者  $ID_r$  為 PB-PKS 的成員時，此接收者具有相對應的公鑰  $PBPK_{ID_r}^{1st}, PBPK_{ID_r}^{2nd}$ 。發送者計算  $PBH_1 = (PBPK_{ID_r}^{1st})^h$  與  $PBH_2 = (PBPK_{ID_r}^{2nd})^h$ 。接下來，發送者計算  $H = HF_2(PBH_1, PBH_2)$  與  $TT = HF_3(PBH_2)$ 。
- 當接收者  $ID_r$  為 CB-PKS 的成員時，此接收者具有相對應的公鑰  $CBPK_{ID_r}^{1st}, CBPK_{ID_r}^{2nd}, CBPK_{ID_r}^{3rd}$ 。發送者計算  $CBH_1 = (CBPK_{ID_r}^{1st})^h$ ,  $CBH_2 = (CBPK_{ID_r}^{2nd})^h$ ,  $CBH_3 = (CBPK_{ID_r}^{3rd})^h$ 。接下來，發送者計算  $H = HF_2(CBH_1, CBH_2)$  與  $TT = HF_3(CBH_3)$ 。

$\hat{e}(CBPK_{ID_r}^{3rd}, U_1 + id_r \cdot V_1)^h$  與  $CBH_4 = (CBPK_{CA} \cdot \hat{e}(CBPK_{ID_r}^{3rd}, U_2 + id_r \cdot V_2))^h$ ，其中  $id_r = HF_1(ID_r, CBPK_{ID_r}^{1st}, CBPK_{ID_r}^{2nd}, CBPK_{ID_r}^{3rd})$ 。接下來，發送者計算  $H = HF_4(CBH_1, CBH_2, CBH_3, CBH_4)$  與  $TT = HF_5(CBH_2, CBH_4)$ 。

(3) 發送者會根據自身所屬的系統計算  $TS$ 。

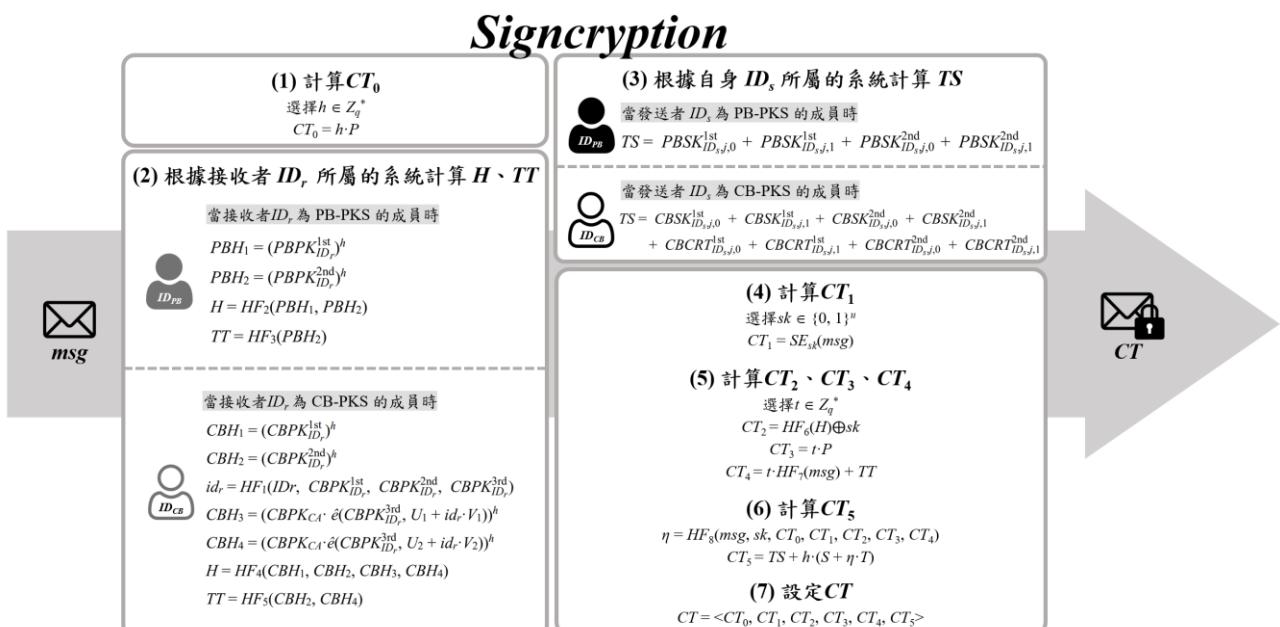
- 當發送者  $ID_s$  為 PB-PKS 的成員時，此發送者具有相對應的私鑰  $(PBSK_{ID_{sj-1,0}}^{1st}, PBSK_{ID_{sj-1,1}}^{1st}, PBSK_{ID_{sj-1,0}}^{2nd}, PBSK_{ID_{sj-1,1}}^{2nd})$ 。發送者隨機選取兩個數值  $\gamma_j^{1st}, \gamma_j^{2nd} \in Z_q^*$ ，並計算  $(PBSK_{ID_{sj,j,0}}^{1st}, PBSK_{ID_{sj,j,1}}^{1st}) = (PBSK_{ID_{sj-1,0}}^{1st} + \gamma_j^{1st} \cdot P, PBSK_{ID_{sj-1,1}}^{1st} - \gamma_j^{1st} \cdot P)$  與  $(PBSK_{ID_{sj,j,0}}^{2nd}, PBSK_{ID_{sj,j,1}}^{2nd}) = (PBSK_{ID_{sj-1,0}}^{2nd} + \gamma_j^{2nd} \cdot P, PBSK_{ID_{sj-1,1}}^{2nd} - \gamma_j^{2nd} \cdot P)$ 。接下來，發送者計算  $TS = PBSK_{ID_{sj,j,0}}^{1st} + PBSK_{ID_{sj,j,1}}^{1st} + PBSK_{ID_{sj,j,0}}^{2nd} + PBSK_{ID_{sj,j,1}}^{2nd}$ 。
- 當發送者  $ID_s$  為 CB-PKS 的成員時，此發送者具有相對應的私鑰  $(CBSK_{ID_{sj-1,0}}^{1st}, CBSK_{ID_{sj-1,1}}^{1st}, CBSK_{ID_{sj-1,0}}^{2nd}, CBSK_{ID_{sj-1,1}}^{2nd})$ ， $(CBCRT_{ID_{sj-1,0}}^{1st}, CBCRT_{ID_{sj-1,1}}^{1st}, CBCRT_{ID_{sj-1,0}}^{2nd}, CBCRT_{ID_{sj-1,1}}^{2nd})$ 。發送者隨機選取四個數值  $\delta_j^{1st}, \delta_j^{2nd}, \zeta_j^{1st}, \zeta_j^{2nd} \in Z_q^*$ ，並計算  $(CBSK_{ID_{sj,j,0}}^{1st}, CBSK_{ID_{sj,j,1}}^{1st}) = (CBSK_{ID_{sj-1,0}}^{1st} + \delta_j^{1st} \cdot P, CBSK_{ID_{sj-1,1}}^{1st} - \delta_j^{1st} \cdot P)$ ， $(CBSK_{ID_{sj,j,0}}^{2nd}, CBSK_{ID_{sj,j,1}}^{2nd}) = (CBSK_{ID_{sj-1,0}}^{2nd} + \delta_j^{2nd} \cdot P, CBSK_{ID_{sj-1,1}}^{2nd} - \delta_j^{2nd} \cdot P)$ ， $(CBCRT_{ID_{sj,j,0}}^{1st}, CBCRT_{ID_{sj,j,1}}^{1st}) = (CBCRT_{ID_{sj-1,0}}^{1st} + \zeta_j^{1st} \cdot P, CBCRT_{ID_{sj-1,1}}^{1st} - \zeta_j^{1st} \cdot P)$ ， $(CBCRT_{ID_{sj,j,0}}^{2nd}, CBCRT_{ID_{sj,j,1}}^{2nd}) = (CBCRT_{ID_{sj-1,0}}^{2nd} + \zeta_j^{2nd} \cdot P, CBCRT_{ID_{sj-1,1}}^{2nd} - \zeta_j^{2nd} \cdot P)$ 。接下來，發送者計算  $TS = CBSK_{ID_{sj,j,0}}^{1st} + CBSK_{ID_{sj,j,1}}^{1st} + CBSK_{ID_{sj,j,0}}^{2nd} + CBSK_{ID_{sj,j,1}}^{2nd} + CBCRT_{ID_{sj,j,0}}^{1st} + CBCRT_{ID_{sj,j,1}}^{1st} + CBCRT_{ID_{sj,j,0}}^{2nd} + CBCRT_{ID_{sj,j,1}}^{2nd}$ 。

(4) 發送者隨即選取一個數值  $sk \in \{0, 1\}^u$ ，並計算  $CT_1 = SE_{sk}(msg)$ 。

(5) 利用(2)獲得的結果，發送者計算  $CT_2 = HF_6(H) \oplus sk$ 。此外，發送者隨機選取一個數值  $t \in Z_q^*$  並計算  $CT_3 = t \cdot P$ ,  $CT_4 = t \cdot HF_7(msg) + TT$ 。

(6) 利用(3)獲得的結果，發送者計算  $CT_5 = TS + h \cdot (S + \eta \cdot T)$ ，其中  $\eta = HF_8(msg, sk, CT_0, CT_1, CT_2, CT_3, CT_4)$ 。

(7) 最後，發送者設定  $CT = <CT_0, CT_1, CT_2, CT_3, CT_4, CT_5>$ 。



圖十三、SC 演算法的流程

- **解密驗證階段 (Unsigncryption phase) :** 針對這個解密驗證 (unsigncryption, USC) 演算法的第  $k$  次執行中，接收者  $ID_r$  執行這個演算法，並將發送者  $ID_s$  傳來的密文  $CT$  解密，進而得到訊息  $msg$ 。USC 演算法的輸入為密文  $CT$ 、接收者身分  $ID_r$ 、發送者身分  $ID_s$ 、接收者私鑰  $SK_r$  與發送者公鑰  $PK_s$ ，其中  $CT = \langle CT_0, CT_1, CT_2, CT_3, CT_4, CT_5 \rangle$ ；根據發送者  $ID_s$  所屬的系統， $SK_r$  可能為  $(PBSK_{ID_r,k,0}^{1st}, PBSK_{ID_r,k,1}^{1st})$ ,  $(PBSK_{ID_r,k,0}^{2nd}, PBSK_{ID_r,k,1}^{2nd})$  或  $(CBSK_{ID_r,k,0}^{1st}, CBSK_{ID_r,k,1}^{1st})$ ,  $(CBSK_{ID_r,k,0}^{2nd}, CBSK_{ID_r,k,1}^{2nd})$ ,  $(CBCRT_{ID_r,k,0}^{1st}, CBCRT_{ID_r,k,1}^{1st})$ ,  $(CBCRT_{ID_r,k,0}^{2nd}, CBCRT_{ID_r,k,1}^{2nd})$ ；根據發送者  $ID_s$  所屬的系統， $PK_s$  可能為  $PBPK_{ID_s}^{1st}$ ,  $PBPK_{ID_s}^{2nd}$  或  $CBPK_{ID_s}^{1st}$ ,  $CBPK_{ID_s}^{2nd}$ ,  $CBPK_{ID_s}^{3rd}$ 。USC 演算法的輸出為訊息  $msg$ ，詳細的流程如下面步驟及圖十四：

(1) 接收者會根據自身所屬的系統計算  $H'$ 。

- 當接收者  $ID_r$  為 PB-PKS 的成員時，此接收者具有相對應的私鑰  $(PBSK_{ID_r,k-1,0}^{1st}, PBSK_{ID_r,k-1,1}^{1st})$ ,  $(PBSK_{ID_r,k-1,0}^{2nd}, PBSK_{ID_r,k-1,1}^{2nd})$ 。接收者隨機選取兩個數值  $\gamma_k^{1st}, \gamma_k^{2nd} \in Z_q^*$ ，並計算  $(PBSK_{ID_r,k,0}^{1st}, PBSK_{ID_r,k,1}^{1st}) = (PBSK_{ID_r,k-1,0}^{1st} + \gamma_k^{1st} \cdot P, PBSK_{ID_r,k-1,1}^{1st} - \gamma_k^{1st} \cdot P)$  與  $(PBSK_{ID_r,k,0}^{2nd}, PBSK_{ID_r,k,1}^{2nd}) = (PBSK_{ID_r,k-1,0}^{2nd} + \gamma_k^{2nd} \cdot P, PBSK_{ID_r,k-1,1}^{2nd} - \gamma_k^{2nd} \cdot P)$ 。接下來，接收者計算  $PBH_1' = \hat{e}(CT_0, PBSK_{ID_r,k,0}^{1st} + PBSK_{ID_r,k,1}^{1st})$ ,  $PBH_2' = \hat{e}(CT_0, PBSK_{ID_r,k,0}^{2nd} + PBSK_{ID_r,k,1}^{2nd})$  與  $H' = HF_2(PBH_1', PBH_2')$ 。
- 當接收者  $ID_r$  為 CB-PKS 的成員時，此接收者具有相對應的私鑰  $(CBSK_{ID_r,k-1,0}^{1st}, CBSK_{ID_r,k-1,1}^{1st})$ ,  $(CBSK_{ID_r,k-1,0}^{2nd}, CBSK_{ID_r,k-1,1}^{2nd})$ ,  $(CBCRT_{ID_r,k-1,0}^{1st}, CBCRT_{ID_r,k-1,1}^{1st})$ ,  $(CBCRT_{ID_r,k-1,0}^{2nd}, CBCRT_{ID_r,k-1,1}^{2nd})$ 。接收者隨機選取四個數值  $\delta_k^{1st}, \delta_k^{2nd}, \zeta_k^{1st}, \zeta_k^{2nd} \in Z_q^*$ ，並計算  $(CBSK_{ID_r,k,0}^{1st}, CBSK_{ID_r,k,1}^{1st}) = (CBSK_{ID_r,k-1,0}^{1st} + \delta_k^{1st} \cdot P, CBSK_{ID_r,k-1,1}^{1st} - \delta_k^{1st} \cdot P)$ ,  $(CBSK_{ID_r,k,0}^{2nd}, CBSK_{ID_r,k,1}^{2nd}) = (CBSK_{ID_r,k-1,0}^{2nd} + \delta_k^{2nd} \cdot P, CBSK_{ID_r,k-1,1}^{2nd} - \delta_k^{2nd} \cdot P)$ ,  $(CBCRT_{ID_r,k,0}^{1st}, CBCRT_{ID_r,k,1}^{1st}) = (CBCRT_{ID_r,k-1,0}^{1st} + \zeta_k^{1st} \cdot P, CBCRT_{ID_r,k-1,1}^{1st} - \zeta_k^{1st} \cdot P)$ ,  $(CBCRT_{ID_r,k,0}^{2nd}, CBCRT_{ID_r,k,1}^{2nd}) = (CBCRT_{ID_r,k-1,0}^{2nd} + \zeta_k^{2nd} \cdot P, CBCRT_{ID_r,k-1,1}^{2nd} - \zeta_k^{2nd} \cdot P)$ 。接下來，接收者計算  $CBH_1' = \hat{e}(CT_0, CBSK_{ID_r,k,0}^{1st} + CBSK_{ID_r,k,1}^{1st})$ ,  $CBH_2' = \hat{e}(CT_0, CBSK_{ID_r,k,0}^{2nd} + CBSK_{ID_r,k,1}^{2nd})$ ,  $CBH_3' = \hat{e}(CT_0, CBCRT_{ID_r,k,0}^{1st} + CBCRT_{ID_r,k,1}^{1st})$ ,  $CBH_4' = \hat{e}(CT_0, CBCRT_{ID_r,k,0}^{2nd} + CBCRT_{ID_r,k,1}^{2nd})$  與  $H' = HF_4(CBH_1', CBH_2', CBH_3', CBH_4')$ 。

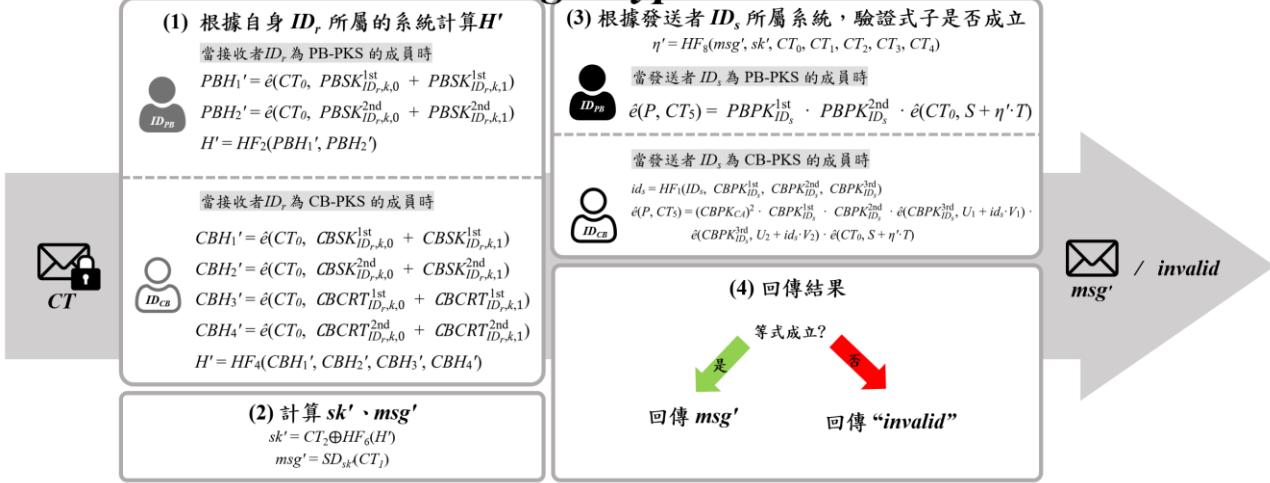
(2) 利用(1)獲得的結果，接收者計算  $sk' = CT_2 \oplus HF_6(H')$  並可計算出  $msg' = SD_{sk'}(CT_1)$ 。

(3) 為了確保訊息  $msg$  的來源，接收者計算  $\eta' = HF_8(msg', sk', CT_0, CT_1, CT_2, CT_3, CT_4)$ 。並根據發送者  $ID_s$  所屬的系統驗證對應的方程式。

- 當發送者  $ID_s$  為 PB-PKS 的成員時，此發送者具有相對應的公鑰  $PBPK_{ID_s}^{1st}, PBPK_{ID_s}^{2nd}$ 。這時，接收者可以透過驗證方程式  $\hat{e}(P, CT_5) = PBPK_{ID_s}^{1st} \cdot PBPK_{ID_s}^{2nd} \cdot \hat{e}(CT_0, S + \eta' \cdot T)$  是否相等，來確定此封訊息的來源是否屬實。
- 當發送者  $ID_s$  為 CB-PKS 的成員時，此發送者具有相對應的公鑰  $CBPK_{ID_s}^{1st}, CBPK_{ID_s}^{1st}, CBPK_{ID_s}^{3rd}$ 。這時，接收者計算  $ids = HF_1(ID_s, CBPK_{ID_s}^{1st}, CBPK_{ID_s}^{2nd}, CBPK_{ID_s}^{3rd})$ ，並驗證方程式  $\hat{e}(P, CT_5) = (CBPK_{CA})^2 \cdot CBPK_{ID_s}^{1st} \cdot CBPK_{ID_s}^{2nd} \cdot \hat{e}(CBPK_{ID_s}^{3rd}, U_1 + ids \cdot V_1) \cdot \hat{e}(CBPK_{ID_s}^{3rd}, U_2 + ids \cdot V_2) \cdot \hat{e}(CT_0, S + \eta' \cdot T)$  是否相等，來確定此封訊息的來源是否屬實。

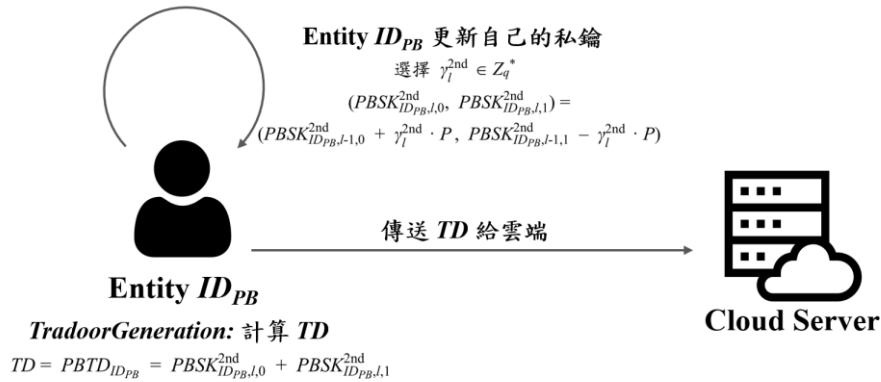
(4) 如果等式成立，則回傳  $msg'$ ，此外回傳 “invalid”。

## Unsigncryption



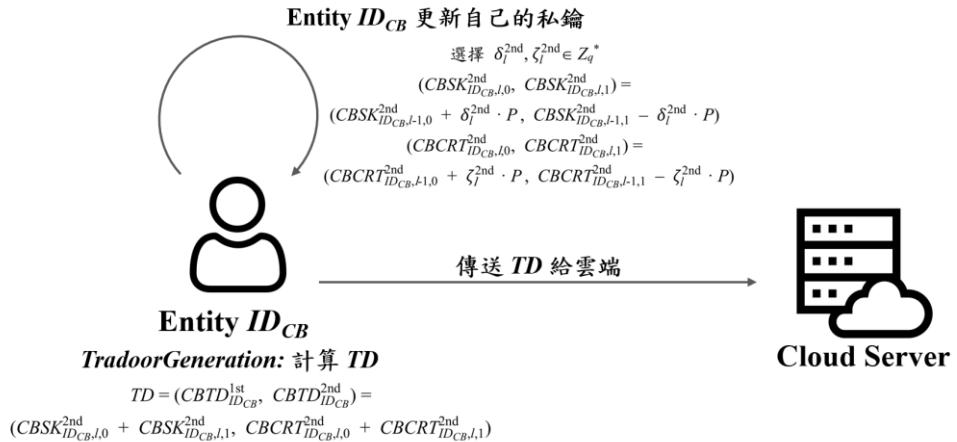
圖十四、USC 演算法的流程

- **暗門生成階段 (Trapdoor generation phase) :** 對暗門生成 (*TrapdoorGeneration, TG*) 演算法的第  $l$  次執行中，具有身分為  $ID_{PB}$  或  $ID_{CB}$  的實體執行此演算法並獲得暗門  $TD$ 。
  - 當實體的身分為  $ID_{PB}$ ，此實體具有相對應的私鑰  $(PBSK_{ID_{PB},l-1,0}^{2nd}, PBSK_{ID_{PB},l-1,1}^{2nd})$ 。實體隨機選取數值  $\gamma_l^{2nd} \in Z_q^*$ ，並計算  $(PBSK_{ID_{PB},l,0}^{2nd}, PBSK_{ID_{PB},l,1}^{2nd}) = (PBSK_{ID_{PB},l-1,0}^{2nd} + \gamma_l^{2nd} \cdot P, PBSK_{ID_{PB},l-1,1}^{2nd} - \gamma_l^{2nd} \cdot P)$ 。接下來，實體計算  $TD = PBTD_{ID_{PB}} = PBSK_{ID_{PB},l,0}^{2nd} + PBSK_{ID_{PB},l,1}^{2nd}$ ，如圖十五所示。



圖十五、實體身分為  $ID_{PB}$  的暗門生成階段

- 當實體的身分為  $ID_{CB}$ ，此實體具有相對應的私鑰  $(CBSK_{ID_{CB},l-1,0}^{2nd}, CBSK_{ID_{CB},l-1,1}^{2nd})$ ， $(CBCRT_{ID_{CB},l-1,0}^{2nd}, CBCRT_{ID_{CB},l-1,1}^{2nd})$ 。實體隨機選取兩個數值  $\delta_l^{2nd}, \zeta_l^{2nd} \in Z_q^*$ ，並計算  $(CBSK_{ID_{CB},l,0}^{2nd}, CBSK_{ID_{CB},l,1}^{2nd}) = (CBSK_{ID_{CB},l-1,0}^{2nd} + \delta_l^{2nd} \cdot P, CBSK_{ID_{CB},l-1,1}^{2nd} - \delta_l^{2nd} \cdot P)$  與  $(CBCRT_{ID_{CB},l,0}^{2nd}, CBCRT_{ID_{CB},l,1}^{2nd}) = (CBCRT_{ID_{CB},l-1,0}^{2nd} + \zeta_l^{2nd} \cdot P, CBCRT_{ID_{CB},l-1,1}^{2nd} - \zeta_l^{2nd} \cdot P)$ 。接下來，實體計算  $TD = (CBTD_{ID_{CB}}^{1st}, CBTD_{ID_{CB}}^{2nd}) = (CBSK_{ID_{CB},l,0}^{2nd} + CBSK_{ID_{CB},l,1}^{2nd}, CBCRT_{ID_{CB},l,0}^{2nd} + CBCRT_{ID_{CB},l,1}^{2nd})$ ，如圖十六所示。



圖十六、實體身分為  $ID_{CB}$  的暗門生成階段

- 相等性測試階段 (*Equality test phase*)：在此階段中，雲端會收到來自實體  $ID_A$  與實體  $ID_B$  的密文暗門對 (*ciphertext-trapdoor pair*)，分別是  $(CT_A, TDA)$  與  $(CT_B, TDB)$ 。實體  $ID_A$  與實體  $ID_B$  可能是 PB-PKS 或 CB-PKS 的成員，根據其身分不同，雲端會執行不同的運算。如果兩密文相同，雲端會回傳 *true*，否則回傳 *false*。詳細的流程如下面步驟及圖十七所示：

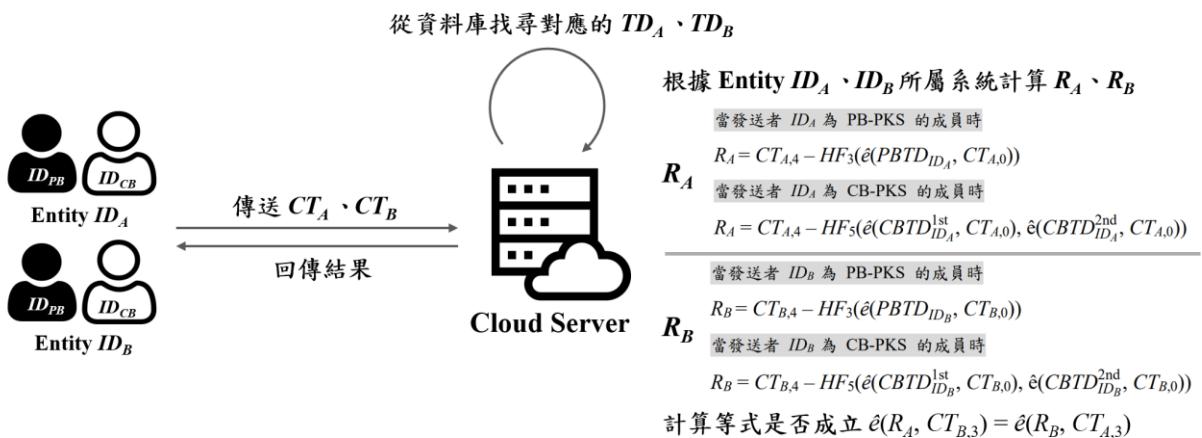
(1) 雲端會根據實體  $ID_A$  所屬的系統計算  $R_A$ 。

- 當實體  $ID_A$  為 PB-PKS 的成員時，雲端會計算  $R_A = CT_{A,4} - HF_3(\hat{e}(PBTD_{ID_A}, CT_{A,0}))$ 。
- 當實體  $ID_A$  為 CB-PKS 的成員時，雲端會計算  $R_A = CT_{A,4} - HF_5(\hat{e}(CBTD_{ID_A}^{1st}, CT_{A,0}), \hat{e}(CBTD_{ID_A}^{2nd}, CT_{A,0}))$ 。

(2) 雲端會根據實體  $ID_B$  所屬的系統計算  $R_B$ 。

- 當實體  $ID_B$  為 PB-PKS 的成員時，雲端會計算  $R_B = CT_{B,4} - HF_3(\hat{e}(PBTD_{ID_B}, CT_{B,0}))$ 。
- 當實體  $ID_B$  為 CB-PKS 的成員時，雲端會計算  $R_B = CT_{B,4} - HF_5(\hat{e}(CBTD_{ID_B}^{1st}, CT_{B,0}), \hat{e}(CBTD_{ID_B}^{2nd}, CT_{B,0}))$ 。

(3) 最後，雲端計算等式  $\hat{e}(R_A, CT_{B,3}) = \hat{e}(R_B, CT_{A,3})$  是否相等，如果相等回傳 *true*，否則回傳 *false*。



圖十七、相等性測試階段的流程

### 5.3 機制的正確性驗證與安全分析

在本小節中，我們將證明並推導解密驗證階段的式子與相等性測試階段的式子。同時，我們將驗證我們機制中密文的機密性與簽章的不可偽造性。最後，我們還將證明透過金鑰更新可以有效避免因旁路攻擊而導致的部分資訊洩漏，確保系統的安全性。

#### 5.3.1 解密驗證階段中的式子推導

在解密驗證階段的 *unsigncryption* (*USC*) 演算法中，接收者  $ID_r$  會收到來自發送者  $ID_s$  的密文  $CT$ 。接收者透過 *USC* 演算法，計算出訊息  $msg'$ ，為了確保訊息  $msg'$  的來源及正確性，接收者接著計算  $\eta' = HF_8(msg', sk', CT_0, CT_1, CT_2, CT_3, CT_4)$ ，並根據發送者  $ID_s$  所屬的系統驗證對應的方程式。只有當  $\eta = \eta'$  時，方程式才會成立，也代表著訊息的來源及內容沒有被竄改的。

- 當發送者  $ID_s$  為 PB-PKS 的成員時，此式  $\hat{e}(P, CT_5) = PBPK_{ID_s}^{1st} \cdot PBPK_{ID_s}^{2nd} \cdot \hat{e}(CT_0, S + \eta' \cdot T)$  成立。

$$(1) CT_5 = TS + h \cdot (S + \eta T)$$

$$(2) TS = PBSK_{ID_s,j,0}^{1st} + PBSK_{ID_s,j,1}^{1st} + PBSK_{ID_s,j,0}^{2nd} + PBSK_{ID_s,j,1}^{2nd} = PBSK_{ID_s}^{1st} + PBSK_{ID_s}^{2nd}$$

$$(3) \hat{e}(P, CT_5) = \hat{e}(P, TS + h \cdot (S + \eta T))$$

$$= \hat{e}(P, PBSK_{ID_s}^{1st} + PBSK_{ID_s}^{2nd} + h \cdot (S + \eta T))$$

$$= PBPK_{ID_s}^{1st} \cdot PBPK_{ID_s}^{2nd} \cdot \hat{e}(CT_0, h \cdot (S + \eta T))$$

- 當發送者  $ID_s$  為 CB-PKS 的成員時，此式  $\hat{e}(P, CT_5) = (CBPK_{CA})^2 \cdot CBPK_{ID_s}^{1st} \cdot CBPK_{ID_s}^{2nd} \cdot \hat{e}(CBPK_{ID_s}^{3rd}, U_1 + id_s \cdot V_1) \cdot \hat{e}(CBPK_{ID_s}^{3rd}, U_2 + id_s \cdot V_2) \cdot \hat{e}(CT_0, S + \eta' \cdot T)$  成立，其中  $id_s = HF_1(ID_s, CBPK_{ID_s}^{1st}, CBPK_{ID_s}^{2nd}, CBPK_{ID_s}^{3rd})$ 。

$$(1) CT_5 = TS + h \cdot (S + \eta T)$$

$$(2) TS = CBSK_{ID_s,j,0}^{1st} + CBSK_{ID_s,j,1}^{1st} + CBSK_{ID_s,j,0}^{2nd} + CBSK_{ID_s,j,1}^{2nd} + CBCRT_{ID_s,j,0}^{1st} + CBCRT_{ID_s,j,1}^{1st} + CBCRT_{ID_s,j,0}^{2nd} + CBCRT_{ID_s,j,1}^{2nd} = CBSK_{ID_s}^{1st} + CBSK_{ID_s}^{2nd} + CBCRT_{ID_s}^{1st} + CBCRT_{ID_s}^{2nd}$$

$$(3) CBCRT_{ID_s}^{1st} = CBSK_{CA,i,0} + CBSK_{CA,i,1} + x_2 \cdot (U_1 + id_s \cdot V_1) = CBSK_{CA} + x_2 \cdot (U_1 + id_s \cdot V_1)$$

$$(4) CBCRT_{ID_s}^{2nd} = CBSK_{CA,i,0} + CBSK_{CA,i,1} + x_2 \cdot (U_2 + id_s \cdot V_2) = CBSK_{CA} + x_2 \cdot (U_2 + id_s \cdot V_2)$$

$$(5) \hat{e}(P, CT_5) = \hat{e}(P, TS + h \cdot (S + \eta T))$$

$$= \hat{e}(P, CBSK_{ID_s}^{1st} + CBSK_{ID_s}^{2nd} + CBCRT_{ID_s}^{1st} + CBCRT_{ID_s}^{2nd} + h \cdot (S + \eta T))$$

$$= CBPK_{ID_s}^{1st} \cdot CBPK_{ID_s}^{2nd} \cdot \hat{e}(P, CBCRT_{ID_s}^{1st}) \cdot \hat{e}(P, CBCRT_{ID_s}^{2nd}) \cdot \hat{e}(P, h \cdot (S + \eta T))$$

$$= CBPK_{ID_s}^{1st} \cdot CBPK_{ID_s}^{2nd} \cdot \hat{e}(P, CBSK_{CA} + x_2 \cdot (U_1 + id_s \cdot V_1))$$

$$\cdot \hat{e}(P, CBSK_{CA} + x_2 \cdot (U_2 + id_s \cdot V_2)) \cdot \hat{e}(P, h \cdot (S + \eta T))$$

$$= CBPK_{ID_s}^{1st} \cdot CBPK_{ID_s}^{2nd} \cdot (CBPK_{CA})^2 \cdot \hat{e}(CBPK_{ID_s}^{3rd}, U_1 + id_s \cdot V_1) \cdot$$

$$\hat{e}(CBPK_{ID_s}^{3rd}, U_2 + id_s \cdot V_2) \cdot \hat{e}(CT_0, S + \eta' \cdot T)$$

#### 5.3.2 相等性測試階段中的式子推導

在相等性測試階段中，雲端會收到來自實體  $ID_A$  與實體  $ID_B$  的密文暗門對 (*ciphertext-trapdoor pair*)，分別是  $(CT_A, TDA)$  與  $(CT_B, TD_B)$ 。以下將證明只有當  $msg_A = msg_B$  時，不管實體身分為何，相等性測試皆成立。

- 當實體  $ID_A$  為 PB-PKS 的成員時，雲端會計算  $R_A = CT_{A,4} - HF_3(\hat{e}(PBTD_{ID_A}, CT_{A,0}))$

- (1)  $CT_{A,4} = t_A \cdot HF_7(msg_A) + TT_A$
- (2)  $TT_A = HF_3(PBH_{A,2})$
- (3)  $PBH_{A,2} = (PBPK_{ID_A}^{2nd})^{h_A}$ .
- (4)  $PBT_{ID_A} = PBSK_{ID_A,l,0}^{2nd} + PBSK_{ID_A,l,1}^{2nd} = PBSK_{ID_A}^{2nd}$
- (5)  $CT_{A,0} = k_A \cdot P$
- (6)  $R_A = CT_{A,4} - HF_3(\hat{e}(PBT_{ID_A}, CT_{A,0}))$   
 $= t_A \cdot HF_7(msg_A) + HF_3((PBPK_{ID_A}^{2nd})^{h_A}) - HF_3(\hat{e}(PBSK_{ID_A}^{2nd}, h_A \cdot P))$   
 $= t_A \cdot HF_7(msg_A) + HF_3((PBPK_{ID_A}^{2nd})^{h_A}) - HF_3((PBPK_{ID_A}^{2nd})^{h_A})$   
 $= t_A \cdot HF_7(msg_A)$

➤ 當實體  $ID_B$  為 CB-PKS 的成員時，雲端會計算  $R_B = CT_{B,4} - HF_5(\hat{e}(CBTD_{ID_B}^{1st}, CT_{B,0}), \hat{e}(CBTD_{ID_B}^{2nd}, CT_{B,0}))$

- (1)  $CT_{B,4} = t_B \cdot HF_7(msg_B) + TT_B$
- (2)  $TT_B = HF_5(CBH_2, CBH_4)$
- (3)  $CBH_{B,2} = (CBPK_{ID_B}^{2nd})^{h_B}$
- (4)  $CBH_{B,4} = (CBPK_{CA} \cdot \hat{e}(CBPK_{ID_B}^{3rd}, U_2 + id_B \cdot V_2))^{h_B}$
- (5)  $CBTD_{ID_B}^{1st} = CBSK_{ID_B,l,0}^{2nd} + CBSK_{ID_B,l,1}^{2nd} = CBSK_{ID_B}^{2nd}$
- (6)  $CBTD_{ID_B}^{2nd} = CBCRT_{ID_B,l,0}^{2nd} + CBCRT_{ID_B,l,1}^{2nd} = CBCRT_{ID_B}^{2nd}$
- (6)  $CBCRT_{ID_B}^{2nd} = CBSK_{CA,i,0} + CBSK_{CA,i,1} + x_2 \cdot (U_2 + id_B \cdot V_2) = CBSK_{CA} + x_2 \cdot (U_2 + id_B \cdot V_2)$
- (7)  $CT_{B,0} = h_B \cdot P$
- (8)  $R_B = CT_{B,4} - HF_5(\hat{e}(CBTD_{ID_B}^{1st}, CT_{B,0}), \hat{e}(CBTD_{ID_B}^{2nd}, CT_{B,0}))$   
 $= t_B \cdot HF_7(msg_B) + HF_5(CBH_{B,2}, CBH_{B,4})$   
 $- HF_5(\hat{e}(CBSK_{ID_B}^{2nd}, h_B \cdot P), \hat{e}(CBCRT_{ID_B}^{2nd}, h_B \cdot P))$   
 $= t_B \cdot HF_7(msg_B) + HF_5((CBPK_{ID_B}^{2nd})^{h_B}, (CBPK_{CA} \cdot \hat{e}(CBPK_{ID_B}^{3rd}, U_2 + id_B \cdot V_2))^{h_B})$   
 $- HF_5((CBPK_{ID_B}^{2nd})^{h_B}, (CBPK_{CA} \cdot \hat{e}(CBPK_{ID_B}^{3rd}, U_2 + id_B \cdot V_2))^{h_B})$   
 $= t_B \cdot HF_7(msg_B)$

得到  $R_A$  與  $R_B$  後，進行相等性測試，當  $msg_A = msg_B$  時， $\hat{e}(R_A, CT_{B,5}) = \hat{e}(R_B, CT_{A,5})$  成立。

- (1)  $\hat{e}(R_A, CT_{B,5}) = \hat{e}(t_A \cdot HF_7(msg_A), t_B \cdot P) = \hat{e}(HF_7(msg_A), P)^{t_A t_B}$
- (2)  $\hat{e}(R_B, CT_{A,5}) = \hat{e}(t_B \cdot HF_7(msg_B), t_A \cdot P) = \hat{e}(HF_7(msg_B), P)^{t_A t_B}$
- (3) 當  $msg_A = msg_B$  時，等式將成立。

### 5.3.3 機制中密文的機密性與簽章的不可偽造性

在簽章加密階段中，發送者  $ID_s$  欲將訊息  $msg$  加密成密文並且對密文簽章後才會傳送給接收者  $ID_r$ 。在這個過程中，只有發送者本人知道訊息，由於發送者是使用接收者  $ID_r$  的公鑰去做加密，再用自己的私鑰去簽章，因此只有對應的私鑰 ( $ID_r$  的私鑰) 可以將訊息解開，也只有對應的公鑰 ( $ID_s$  的公鑰) 可以認證這封密文的來源。

以下使用反證法證明密文的機密性：假設此封密文可以被  $ID_r$  以外的實體  $ID_A$  解開，又因為只有  $ID_r$  的私鑰可以解開此封密文，則可以推論  $ID_A$  一定擁有  $ID_r$  的私鑰。然而，根據實體金鑰生成階段的自有金鑰生成演算法，如果實體  $ID_r$  屬於 PB 系統，他會隨機選取一個數值  $y_1 \in Z_q^*$ ，並且計算第一把私鑰  $PBSK_{ID_r}^{1st} = y_1 \cdot P$ ；如果實體  $ID_r$  屬於 CB 系統，他會隨機選取一

個數值  $z_1 \in Z_q^*$ ，並且計算第一把私鑰  $CBSK_{ID_r}^{1st} = z_1 \cdot P$ 。因此， $ID_A$  擁有  $ID_r$  的私鑰的假設會導致  $ID_A$  必須計算出  $ID_r$  的私鑰  $y_1 \cdot P$  或是  $z_1 \cdot P$ ，但是，根據橢圓曲線離散對數問題 (elliptic curve discrete logarithm problem, ECDLP) [20] 的性質， $y_1$  和  $z_1$  是不可能被求出來的。因此，無法計算出私鑰  $PBSK_{ID_r}^{1st}$  或是  $CBSK_{ID_r}^{1st}$ 。此結論和前面的推論  $ID_A$  一定擁有  $ID_r$  的私鑰矛盾，因此可以證明沒有任何實體會獲得  $ID_r$  的私鑰，所以密文只會被  $ID_r$  本人解開，密文是具有機密性的。

接下來證明簽章的不可偽造性，證明方法和密文的機密性相似，一樣使用反證法證明：假設有實體  $ID_A$  可以偽造  $ID_s$  的簽名，亦即，實體  $ID_A$  使用  $ID_s$  的私鑰對密文簽名並傳送給實體  $ID_B$ ，實體  $ID_B$  收到後認為此封訊息來自  $ID_s$  而不是  $ID_A$ 。若此假設成立，則推論  $ID_A$  一定擁有  $ID_s$  的私鑰。然而，根據實體金鑰生成階段的自有金鑰生成演算法，如果實體  $ID_s$  屬於 PB 系統，他會隨機選取一個數值  $y_1 \in Z_q^*$ ，並且計算第一把私鑰  $PBSK_{ID_s}^{1st} = y_1 \cdot P$ ；如果實體  $ID_s$  屬於 CB 系統，他會隨機選取一個數值  $z_1 \in Z_q^*$ ，並且計算第一把私鑰  $CBSK_{ID_s}^{1st} = z_1 \cdot P$ 。因此， $ID_A$  擁有  $ID_s$  的私鑰的假設會導致  $ID_A$  必須計算出  $ID_s$  的私鑰  $y_1 \cdot P$  或是  $z_1 \cdot P$ ，但是，根據橢圓曲線離散對數問題[20]的性質， $y_1$  和  $z_1$  是不可能被求出來的。因此，無法計算出私鑰  $PBSK_{ID_s}^{1st}$  或是  $CBSK_{ID_s}^{1st}$ 。此結論和前面的推論  $ID_A$  一定擁有  $ID_s$  的私鑰矛盾，因此可以證明沒有任何實體會獲得  $ID_s$  的私鑰，所以無法偽造簽章。

### 5.3.4 利用金鑰更新避免因旁路攻擊洩漏的部分資訊而使系統不安全

旁路攻擊，是指攻擊者可以透過感知能量消耗或執行時間來獲得計算操作中涉及這些秘密值的部分資訊。以私鑰為例，每次旁路攻擊會使得攻擊者獲得部分私鑰資訊，如幾個 bit 的值。攻擊發生次數增加，攻擊者或許能透過這些部分資訊組合出完整的私鑰。因此，我們認為要抵抗旁路攻擊，可以從部分資訊組成完整鑰匙這部分下手。如果今天可以建構一個機制，使得私鑰使用時的值都不相同，這樣，即使每次使用都洩漏一部分 bit，攻擊者也無法透過這些資訊組成完整的私鑰。

具體概念如下，在無法抵擋旁路攻擊的機制中，實體  $ID_A$  擁有一把長度為  $n$  bit 的私鑰  $SK_A$ 。假設在使用私鑰時遭到旁路攻擊，每次被攻擊會洩漏  $m$  bits 的金鑰內容。這種情況下，攻擊者有可能在未來透過這些部分資訊推算出私鑰  $SK_A$  的值。

相對應地，我們提出的機制的概念是這樣的，實體  $ID_B$  先計算出一把長度為  $n$  bits 的私鑰  $SK_B$ ，並計算另一個  $n$  bits 的祕密值  $s$ 。我們的機制會將私鑰拆成兩部分  $(SK_{B,0,1}, SK_{B,0,2}) = (SK_B - s, SK_B + s)$  並作為私鑰對使用。其中， $SK_{B,0,1}$  的 0 指的是目前是第 0 次使用私鑰，而後面的 1 指的是這是私鑰對的第一把私鑰。當實體  $ID_B$  在第  $i$  次使用私鑰時，不管是利用私鑰對訊息做簽章，或是利用私鑰解密，實體  $ID_B$  都會先計算另一個祕密值  $r_i$ ，並將私鑰對  $(SK_{B,i,1}, SK_{B,i,2})$  更新成  $(SK_{B,i-1,1} + r_i, SK_{B,i-1,2} - r_i)$  在做使用，使用時只需要將兩把私鑰相加，就能變回原來的私鑰，可以正常加解密。這樣的更新操作使得每次使用的私鑰對數值都不同，即使遭遇旁路攻擊導致每次洩漏  $m$  個 bits，這些  $m$  個 bits 之間並無直接關聯，所以攻擊者無法使用這些資訊還原成私鑰對的。因此，可以證明我們機制使用金鑰更新的方法可以有效避免旁路攻擊造成系統不安全的問題。

## 5.4 與其他機制的比較

在此小節中，我們分析了機制的效能並與其他機制[5, 8, 9]進行比較。為了方便描述，我

們定義以下符號：

$T_{bp}$ : 雙線性映射操作  $\hat{e}: G \times G \rightarrow G_T$

$T_{sm}$ : 加法群 ( $G$ ) 中的純量乘法操作

$T_{exp}$ : 乘法群 ( $G_T$ ) 中的指數乘法操作

我們採用了 Java Pairing Based Cryptography Library (JPBC) 函式庫實現 LR-HHCB-SCET 機制。模擬環境的配置包括作業系統: Windows 11、CPU: 13th Gen Intel(R) Core(TM) i7-13700H，2400 Mhz、Ram: 16.0 GB。所選用的橢圓曲線為 Type-A ( $y^2 = x^3 + x$ ) 曲線， $G$ 、 $G_T$ 、 $Z_q^*$  及  $u$  的大小分別為 128Bytes、128Bytes、20Bytes 及 20Bytes。在模擬成本的實驗中，我們進行了 1000 次的雙線性映射 ( $T_{bp}$ )、加法群 ( $G$ ) 中的純量乘法 ( $T_{sm}$ ) 與乘法群 ( $G_T$ ) 中的指數乘法 ( $T_{exp}$ ) 計算，並取平均執行時間，將結果列於表三中，時間單位為毫秒。

表三、每個操作的執行時間(in milliseconds)

| $T_{bp}$ | $T_{sm}$ | $T_{exp}$ |
|----------|----------|-----------|
| 3.242 ms | 4.947 ms | 4.82 ms   |

在表四中列出了比較項目，包括加密成本、解密成本、相等性測試成本和一些重要特性。可以觀察到我們的機制在加密和解密的時間成本上稍高於其他三種機制，而相等性測試則位居第二。儘管時間成本相對較高，但我們的機制具有異質性，較[8, 9]機制有彈性。此外，我們的機制能夠避免金鑰託管的問題，相較於同樣具有異質性的機制[5]更為安全。最後，我們是唯一一個能夠抵擋旁路攻擊的機制。儘管在時間成本上有些犧牲，但我們提供了更加安全的解決方案。

表四、LR-HHCB-SCET 機制與其他機制的比較

|        | PKSCET[8]                           | IBSCET[9]                                     | HSC-ET[5]                           | 我們的機制  |
|--------|-------------------------------------|---|-------------------------------------|--|
| 加密     | $1T_{bp} + 4T_{exp}$<br>(22.522 ms) | $2T_{bp} + 6T_{sm} + 2T_{exp}$<br>(45.806 ms) | $5T_{sm} + 2T_{exp}$<br>(34.375 ms) | $2T_{bp} + 11T_{sm} + 4T_{exp}$<br>(80.181 ms) |
|        |                                     |   |                                     |  |
| 解密     | $3T_{bp} + 2T_{exp}$<br>(19.366 ms) | $5T_{bp} + 3T_{sm}$<br>(31.051 ms)            | $3T_{bp} + 1T_{exp}$<br>(11.304 ms) | $8T_{bp} + 7T_{sm}$<br>(60.565 ms)             |
|        |                                     |   |                                     |  |
| 相等性測試  | $4T_{bp} + 2T_{exp}$<br>(22.608 ms) | $4T_{bp}$<br>(12.968 ms)                      | $4T_{bp} + 4T_{exp}$<br>(32.248 ms) | $6T_{bp}$<br>(19.451 ms)                       |
|        |                                     |   |                                     |  |
| 異質性    | No                                  | No  | Yes                                 | Yes  |
| 金鑰系統   | PKI                                 | IBC   | PKI + IBC                           | PKI + CBC                                      |
| 避免金鑰託管 | Yes                                 | No  | No                                  | Yes  |
| 抵擋旁路攻擊 | No                                  | No  | No                                  | Yes  |

## 5.5 機制的實作與成果

在本節中，我們將詳細描述實驗環境的搭建過程，同時提供相關的實作程式碼。最終，我們將呈現實驗的具體成果。

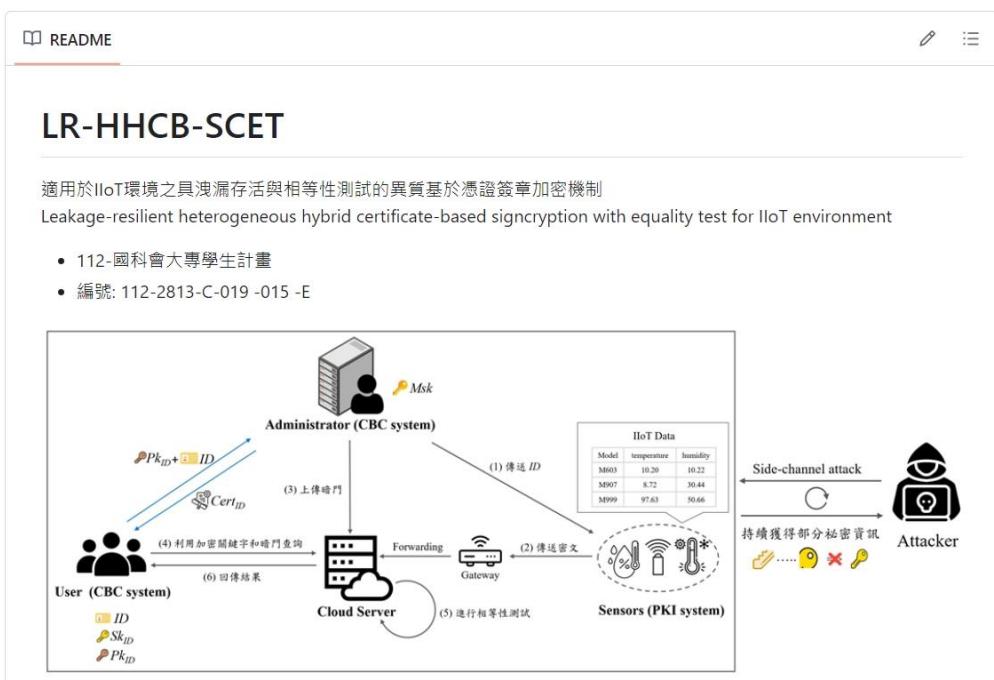
### 5.5.1 實驗環境架設

本計劃使用 Java 程式語言撰寫，並選用 IntelliJ IDEA 社群版做為開發環境；另外為了實

現橢圓曲線上的運算，我們使用 Java Pairing Based Cryptography (JPBC)函式庫作為輔助，以下為實驗環境架設的流程：

- (1) 至 IntelliJ IDEA 官網 (<https://www.jetbrains.com/idea/download/?section=windows>) 下載應用程式，本計劃選用社群版。
- (2) 至 Oracle 官網 (<https://www.oracle.com/tw/java/technologies/downloads/#jdk21-windows>) 下載 jdk 檔，本計劃使用 jdk-11.0.12，並設定環境變數。
- (3) 至 JPBC 官網 (<http://gas.dia.unisa.it/projects/jpbc/>) 下載 JPBC 2.0。
- (4) 完成以上步驟後，打開 IDEA 建立空白專案。
- (5) 再將下載好的 JPBC 解壓縮，並將 jpbc2.0.0/jar 資料夾中的 jpbc-plaf-2.0.0.jar 和 jpbc-api-2.0.0.jar 加入專案的外部函式庫中。

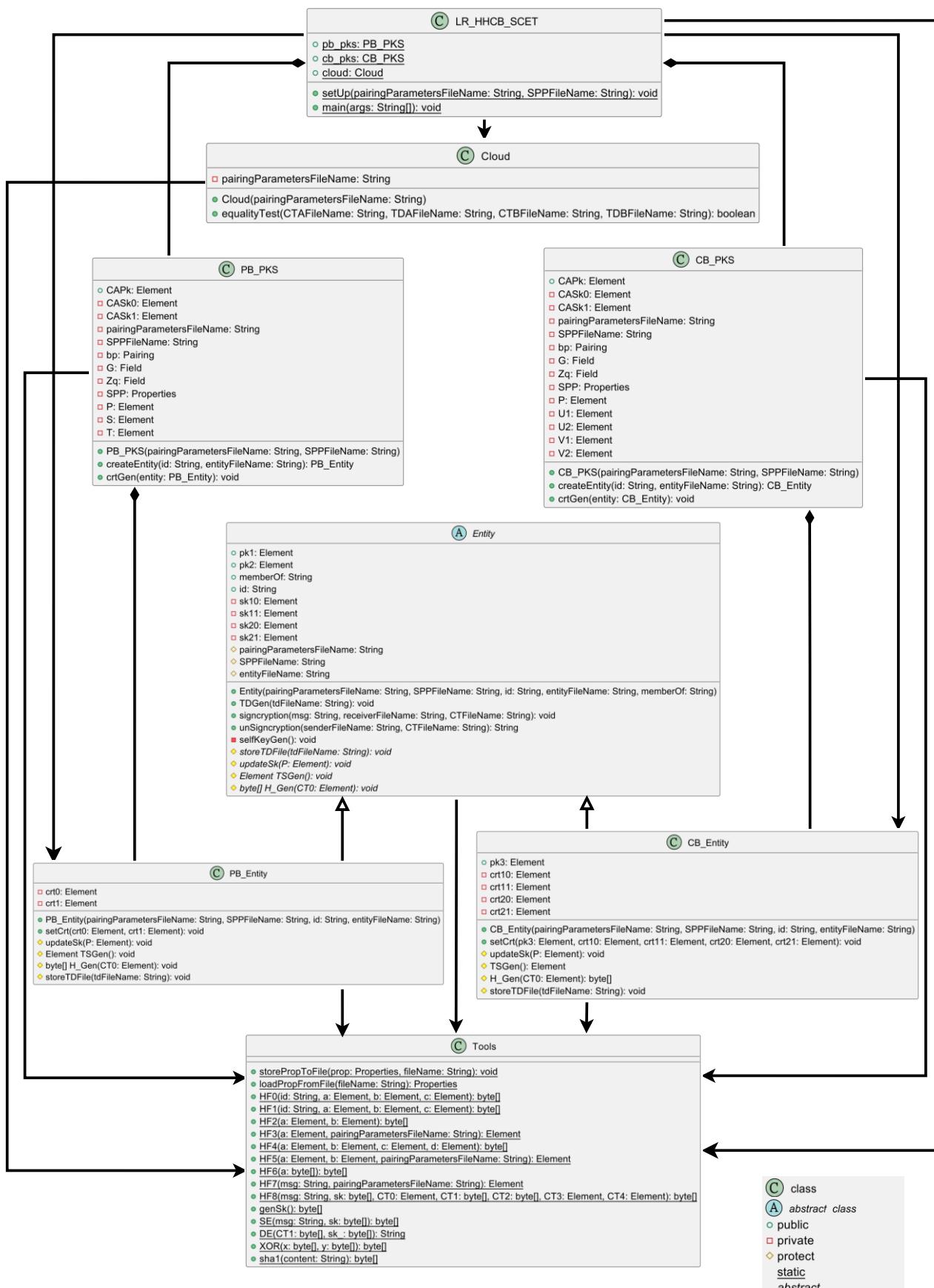
完成以上步驟後，就建立好實驗環境了。另外，我們已將此計畫的完整程式碼上傳至 Github，如圖十八，作為開源使用。可以至 <https://github.com/vayne1125/LR-HHCB-SCET> 下載，下載後，將其放入新建立好的空白專案即可。



圖十八、LR-HHCB-SCET 的 Github 網站

### 5.5.2 類別圖及程式碼說明

我們按照 5.1 小節具體的機制規劃整體的架構，並畫出類別圖。圖十九為 LR-HHCB-SCET 機制的類別圖，總共由 8 個 Java 檔案組成，分別為 LR\_HHCB\_SCET.java、PB\_PKS.java、CB\_PKS.java、Entity.java、PB\_Entity.java、CB\_Entity.java、Cloud.java、Tools.java。其中 LR\_HHCB\_SCET.java 是程式的進入點，亦即 Main 函式的所在處。他與 class PB\_PKS 以及 class CB\_PKS 存在著組合 (composition) 關係；與 class PB\_Entity、class CB\_Entity、class Cloud 以及 class Tools 有關聯 (association) 關係。另外，class PB\_Entity 與 class CB\_Entity 都繼承了 class Entity，並與 class PB\_PKS 以及 class CB\_PKS 有著組合關係。最後，class Tools 是一個工具類別，他提供了各式函數供其他類別使用，所以和其他 class 都有著關聯關係。



圖十九、類別圖

完成類別圖的設計後，接下來我們將根據類別圖開始進行程式碼的實作。LR\_HHCB\_SCET.java 是程式的進入點，亦即 Main 函式的所在處。程式一開始時，會先進行 *setUp()* 函式，對應機制中的系統設定階段，並將公開參數寫入 *spp.properties* 檔案，如圖二十和圖二十一。而 PB\_PKS.java 及 CB\_PKS.java 則是對應機制中的 PB-PKS 公開金鑰系統及 CB-PKS 公開金鑰系統。在 *setUp()* 函式中，創建 *pb\_pks* 和 *cb\_pks* 的變數，並將公開金鑰參數檔案傳入並進行初始化，如圖二十二。

```

public static void setUp(String pairingParametersFileName, String SPPFileName){
    // Pairing
    Pairing bp = PairingFactory.getPairing(pairingParametersFileName);
    Field G = bp.getG1();

    // 建立系統公開參數 SPP，並以文件方式存起來
    Element P = G.newRandomElement().getImmutable(); // G 的 generator
    Element S = G.newRandomElement().getImmutable();
    Element T = G.newRandomElement().getImmutable();
    Element U1 = G.newRandomElement().getImmutable();
    Element V1 = G.newRandomElement().getImmutable();
    Element U2 = G.newRandomElement().getImmutable();
    Element V2 = G.newRandomElement().getImmutable();

    Properties SPP = new Properties();
    SPP.setProperty("P", Base64.getEncoder().encodeToString(P.toBytes()));
    SPP.setProperty("S", Base64.getEncoder().encodeToString(S.toBytes()));
    SPP.setProperty("T", Base64.getEncoder().encodeToString(T.toBytes()));
    SPP.setProperty("U1", Base64.getEncoder().encodeToString(U1.toBytes()));
    SPP.setProperty("U2", Base64.getEncoder().encodeToString(V1.toBytes()));
    SPP.setProperty("V1", Base64.getEncoder().encodeToString(U2.toBytes()));
    SPP.setProperty("V2", Base64.getEncoder().encodeToString(V2.toBytes()));
    Tools.storePropToFile(SPP, SPPFileName);

    pb_pks = new PB_PKS(pairingParametersFileName, SPPFileName);
    cb_pks = new CB_PKS(pairingParametersFileName, SPPFileName);
}

```

圖二十、*setUp()* 函式部分程式碼

```

P=p506S1vEK7pBhDayoRqCM4d0+sCBZJLK4h6yGWIDN5As3KYTa0e6abKfjgRdRx007iKvAvJg0F20QqiuBaPoQVL0wmCdA3peAw3SnKf/RhD+FiW06MvP7D6MvWsMLE9ZaOK3prSkT20ERksX9WTNSemAUMWIIkBmguszt2K6I\=
S=BsjqlFl2UZM/0MEP4FrXN2JydU5o3qqC1SY26c2py6Z0aVv0NWHLE/6e1db04Xmzf/08eMm8AuWf/sRD9ejgn0ECPYeSJrF/HgQkL/E/LFobhwMgy6gbuU8U8XLzFuwe3zL7C6q2a18E8x7qA3IsEhcPackwBA+iAZJR63atECs\=
T=gNETI3/aEPg0ELNMgRrzvY4FB16k10YjprZkCge1X7WPBRJ83wQt1ePJTJZHEMbj6tgP3rMoms7rCgZ+4bxYr1uAStL+LowUT6J6tRwe04105dkQmf3veqDazl5w2F2GKBAKodv+ibNLKKIqfLb6zso70sMYK0egCmW8\=
V1=Vhqu6lo00C1epzygNyIvpFBJxxn0-WBg7CmxpmHbL6gSk6t8tZNA+h0tFcevyThf/J/qJ7hn6s3DwutV2zP41tBc0D//r5QnAH0J0Kg1fkPy+2AdUvrKQIq8yC0eo+KgcXJiqJp3tzoYUKtWD1zPnebzTmxzr0F8\=
U1=c3v/VMuVdthBv+oaeXznkYg67FK2zVbzN2/1xrpq0fw/ZXYzh9P0XbzKrvFoyCsaBg8AD5FRWg+H5BSUuoSXIt31L+24pSK7ePL+SmmGVPBgm5Kx0z2FNCXKybQNhdjvCsgym61r461UD0rn/m32LXIF+ll/V08dP9FTKnA6M\=
V2=E4cndTXKLj5KtAcblzQ6yq9ze8nSfTQgDrNum+sIUmj3pa4bhHv8vIErAlVtilIAmYzmkFaEvliuIn/z3GLwlQ/Z8Kxt3av9f1SY98Yn5yaMkMaIqXjp4qKx1z10A0jFXh0896lBKyhHC9E7cJsreAnptyTxF3Qb/K1ktZQ\=
U2=km7ofvphXnpf2LN7zCaHjjeVXKtttsILtLsya4/ILZh/+ePKg9eoty19j+k+iaR+j+w+1e+jhdtsnwLxz0c1Y43/5gKt//QKG621IX4kHX8dM2+C1zXAP1wnctKbkudwdh/tiXkoarP56RVRR636duauyyzzpuRfrYXpnw\=
PBCAPK=jtlk0OKuLDD0V915+Xy0a1B7I3Y26160VvBaTwz6mqNdaBxKmz8AKNcc/CpnK13KWRBUB0n3eBjsYm60keYREVEVm0PucJluJcpq6ct2Uu0P8J0WtxE40VkJ8ZEtiy0Jjs/j+e76YFzJszNbJlArZswj1Bffos565w/o\=
CBCAPK=f0b/JDN8qY1LrwN2B9nREeWLmp1bbByXjx91K/ToxvTnuYrl1nt9nqPIB3xy5eJbwRh52sm2XYo3DKKRIS+BExAJakFoVki74ZguahHFYV8snMqlQlnzCP6IEuQNxhqruidPcb7170W9Rit6WCaG3J/8c3Gn1L2Hzmhba\=

```

圖二十一、*spp.properties* 檔案內容

```

public PB_PKS(String pairingParametersFileName, String SPPFileName){
    this.pairingParametersFileName = pairingParametersFileName;
    this.SPPFileName = SPPFileName;

    // pairing
    this.bp = PairingFactory.getPairing(pairingParametersFileName);
    this.G = bp.getG1();
    this.Zq = bp.getZr();

    // 將公開參數 SPP 讀入
    this.SPP = Tools.loadPropFromFile(SPPFileName);

    // 還原 P(G 的 generator), S, T
    String P_str = SPP.getProperty("P");
    String S_str = SPP.getProperty("S");
    String T_str = SPP.getProperty("T");
    this.P = G.newElementFromBytes(Base64.getDecoder().decode(P_str)).getImmutable();
    this.S = G.newElementFromBytes(Base64.getDecoder().decode(S_str)).getImmutable();
    this.T = G.newElementFromBytes(Base64.getDecoder().decode(T_str)).getImmutable();
}

```

圖二十二、PB\_PKS 建構子的部分程式碼

Entity.java、PB\_Entity.java 和 CB\_Entity.java 都在實作機制中的實體。Entity.java 為一個抽象物件（abstract class），他實作了自有私鑰生成函式（*selfKeyGen()*）、加密函式（*signcryption()*）、解密函式（*unSigncryption()*）、暗門生成函式（*TDGen()*）等等。而 class PB\_Entity 和 class CB\_Entity 繼承了 class Entity，在 class Entity 的基礎上，增加對應公開金鑰系統的特色。在此實作中，如果想要創建 PB\_Entity 或是 CB\_Entity 型態的變數時，可以利用 class PB\_PKS 和 class CB\_PKS 中的 *createEntity()*函式，創建對應的實體，如圖二十三與圖二十四所示。

```
public PB_Entity createEntity(String id, String entityFileName) {
    PB_Entity entity = new PB_Entity(pairingParametersFileName, SPPFileName, id, entityFileName);
    crtGen(entity);
    return entity;
}
```

圖二十三、PB\_PKS 中的 *createEntity()*函式

```
public CB_Entity createEntity(String id, String entityFileName) {
    CB_Entity entity = new CB_Entity(pairingParametersFileName, SPPFileName, id, entityFileName);
    crtGen(entity);
    return entity;
}
```

圖二十四、CB\_PKS 中的 *createEntity()*函式

當實體被創建時，實體會呼叫自有私鑰生成函式(*selfKeyGen()*)，如圖二十五與圖二十六，創建完實體後，PKS 會呼叫憑證生成函式（*crtGen()*）並將憑證傳送給實體，如圖二十七，以上對應機制中的實體金鑰生成階段。

```
public Entity(String pairingParametersFileName, String SPPFileName, String id, String entityFileName, String memberOf) {
    this.pairingParametersFileName = pairingParametersFileName;
    this.SPPFileName = SPPFileName;
    this.id = id;
    this.entityFileName = entityFileName;
    this.memberOf = memberOf;
    selfKeyGen();
}
```

圖二十五、Entity 的建構子中呼叫 *selfKeyGen()*函式

```
private void selfKeyGen() {
    // pairing
    Pairing bp = PairingFactory.getPairing(pairingParametersFileName);
    Field G = bp.getG1();
    Field Zq = bp.getZr();

    // 將公開參數 SPP 讀入
    Properties SPP = Tools.loadPropFromFile(SPPFileName);

    // 還原 P (G 的 generator)
    String P_str = SPP.getProperty("P");
    Element P = G.newElementFromBytes(Base64.getDecoder().decode(P_str));

    // 做 Entity 的公私鑰
    Element y1 = Zq.newRandomElement().getImmutable();
    Element y2 = Zq.newRandomElement().getImmutable();
    Element sk1 = P.mulZn(y1).getImmutable();
    Element sk2 = P.mulZn(y2).getImmutable();
    pk1 = bp.pairing(P, sk1).getImmutable();
    pk2 = bp.pairing(P, sk2).getImmutable();

    public void crtGen(PB_Entity entity) {
        // get entity pk1, pk2, id
        String entityId = entity.id;
        Element entityPk1 = entity.pk1;
        Element entityPk2 = entity.pk2;

        // 更新 sk
        Element a = Zq.newRandomElement().getImmutable();
        Element aP = P.mulZn(a).getImmutable();
        CASK0 = CASK0.add(aP).getImmutable();
        CASK1 = CASK1.sub(aP).getImmutable();

        // 製作 CRT0, CRT1
        a = Zq.newRandomElement().getImmutable();
        Element crt0 = P.mulZn(a).getImmutable();
        byte[] idPB_Byte = Tools.HF0(entityId, entityPk1, entityPk2, crt0);
        Element idPB = Zq.newElementFromHash(idPB_Byte, 0, idPB_Byte.length);
        Element SidPBT = S.add(T.mulZn(idPB)).getImmutable();
        Element TC = CASK1.add(SidPBT.mulZn(a)).getImmutable();
        Element crt1 = CASK0.add(TC).getImmutable();
        entity.setCrt(crt0, crt1);
    }
}
```

圖二十六、*selfKeyGen()*函式的部分程式碼

圖二十七、*crtGen()*函式的部分程式碼

有了實體後，可以透過呼叫 *signcryption()* 函式加密訊息並生成密文，對應機制中的簽章加密階段，如圖二十八。也可以透過呼叫 *unSigncryption()* 函式解密密文並回傳訊息，對應機制中的解密驗證階段，如圖二十九。或是呼叫 *TDGen()* 函式，生成暗門並傳給雲端，對應機制中的暗門生成階段。另外，我們機制會在使用私鑰前，進行更新的動作，以抵抗旁路攻擊，PB 系統的私鑰更新程式碼如圖三十、CB 系統的私鑰更新程式碼如圖三十一。

```
// signcryption-----
// CTO
Element h = Zq.newRandomElement().getImmutable();
Element CTO = P.mulZn(h).getImmutable();

// H · TT
byte[] H = new byte[20];
Element TT = G.newZeroElement();
if(receiverMemberOf.equals("PB")){
    Element PBH1 = receiverPk1.powZn(h).getImmutable();
    Element PBH2 = receiverPk2.powZn(h).getImmutable();
    H = Tools.HF2(PBH1,PBH2);
    TT = Tools.HF3(PBH2,pairingParametersFileName).getImmutable();
} else if(receiverMemberOf.equals("CB")){
    byte[] idR_Byte = Tools.HF0(receiverId,receiverPk1,receiverPk2,receiverPk3);
    Element idR = Zq.newElementFromHash(idR_Byte, 0,idR_Byte.length).getImmutable();
    Element PBH1 = receiverPk1.powZn(h).getImmutable();
    Element PBH2 = receiverPk2.powZn(h).getImmutable();
    Element PBH3 = (CBCAPK.mul(bp.pairing(receiverPk3,U1.add(V1.mulZn(idR))))).powZn(h).getImmutable();
    Element PBH4 = (CBCAPK.mul(bp.pairing(receiverPk3,U2.add(V2.mulZn(idR))))).powZn(h).getImmutable();
    H = Tools.HF4(PBH1,PBH2,PBH3,PBH4);
    TT = Tools.HF5(PBH2,PBH4,pairingParametersFileName).getImmutable();
} else{
    System.out.println("非法 Entity");
    System.exit( status: -1);
}
```

圖二十八、*signcryption()*函式的部分程式碼

```
// unSigncryption-----
// update self sk
updateSk(P);

// compute H', sk'
byte[] H_ = H_Gen(CTO);
byte[] sk_ = Tools.XOR(Tools.HF6(H_),CT2);
String msg_ = Tools.SD(CT1,sk_);

// verify
byte[] n_byte = Tools.HF8(msg_,sk_,CT0,CT1,CT2,CT3,CT4);
Element n_ = Zq.newElementFromBytes(n_byte).getImmutable();

Element testL = bp.pairing(P,CT5).getImmutable();
Element testR = GT.newZeroElement();
if(senderMemberOf.equals("PB")){
    testR = senderPk1.mul(senderPk2).mul(bp.pairing(CT0,S.add(T.mulZn(n_))).getImmutable();
} else if (senderMemberOf.equals("CB")){
    byte[] idCB_Byte = Tools.HF1(senderId,senderPk1,senderPk2,senderPk3);
    Element idCB = Zq.newElementFromHash(idCB_Byte, 0,idCB_Byte.length).getImmutable();
    testR = CBCAPK.mul(CBCAPK).mul(senderPk1).mul(senderPk2).mul(bp.pairing(senderPk3,U1.add(V1.mulZn(idCB))));
} else {
    System.out.println("非法 Entity");
    System.exit( status: -1);
}
```

圖二十九、*unSigncryption()*函式的部分程式碼

```

@Override
protected void updateSk(Element P){
    Pairing bp = PairingFactory.getPairing(pairingParametersFileName);
    Field Zq = bp.getZr();
    Element r1 = Zq.newRandomElement().getImmutable();
    Element r2 = Zq.newRandomElement().getImmutable();
    Element r1P = P.mulZn(r1).getImmutable();
    Element r2P = P.mulZn(r2).getImmutable();
    sk10 = sk10.add(r1P).getImmutable();
    sk11 = sk11.sub(r1P).getImmutable();
    sk20 = sk20.add(r2P).getImmutable();
    sk21 = sk21.sub(r2P).getImmutable();
}

```

圖三十、class PB\_Entity 中的 *updateSk()* 函式

```

@Override
protected void updateSk(Element P){
    Pairing bp = PairingFactory.getPairing(pairingParametersFileName);
    Field Zq = bp.getZr();
    Element z1 = Zq.newRandomElement().getImmutable();
    Element z2 = Zq.newRandomElement().getImmutable();
    Element d1 = Zq.newRandomElement().getImmutable();
    Element d2 = Zq.newRandomElement().getImmutable();
    Element z1P = P.mulZn(z1).getImmutable();
    Element z2P = P.mulZn(z2).getImmutable();
    Element d1P = P.mulZn(d1).getImmutable();
    Element d2P = P.mulZn(d2).getImmutable();
    sk10 = sk10.add(z1P).getImmutable();
    sk11 = sk11.sub(z1P).getImmutable();
    sk20 = sk20.add(z2P).getImmutable();
    sk21 = sk21.sub(z2P).getImmutable();
    crt10 = crt10.add(d1P).getImmutable();
    crt11 = crt11.sub(d1P).getImmutable();
    crt20 = crt20.add(d2P).getImmutable();
    crt21 = crt21.sub(d2P).getImmutable();
}

```

圖三十一、class PB\_Entity 中的 *updateSk()* 函式

最後，Cloud.java 實作了雲端的功能，提供相等性測試的服務 (*equalityTest()*)，對應機制中的相等性測試階段，如圖三十二。而 Tool.java 則是包含了實作機制的過程中會用到的工具，像是哈希函式、對稱加密算法 (SE)、對稱解密算法 (SD) 或是儲存文件的函式，如圖三十三與圖三十四。

```

// RA
Element RA = G.newZeroElement();
if(memberOfA.equals("PB")){
    String PBTDA_str = TDA.getProperty("TD");
    Element PBTDA = G.newElementFromBytes(Base64.getDecoder().decode(PBTDA_str)).getImmutable();
    RA = CTA4.sub(Tools.HF3(bp.pairing(CTA0,PBTDA), pairingParametersFileName)).getImmutable();
} else if (memberOfA.equals("CB")) {
    String CBTDA1_str = TDA.getProperty("TD1");
    String CBTDA2_str = TDA.getProperty("TD2");
    Element CBTDA1 = G.newElementFromBytes(Base64.getDecoder().decode(CBTDA1_str)).getImmutable();
    Element CBTDA2 = G.newElementFromBytes(Base64.getDecoder().decode(CBTDA2_str)).getImmutable();
    RA = CTA4.sub(Tools.HF5(bp.pairing(CTA0,CBTDA1), bp.pairing(CTA0,CBTDA2), pairingParametersFileName)).getImmutable();
} else {
    System.out.println("非法 Entity");
    System.exit( status: -1);
}

// RB
Element RB = G.newZeroElement();
if(memberOfB.equals("PB")){
    String PBTDB_str = TDB.getProperty("TD");
    Element PBTDB = G.newElementFromBytes(Base64.getDecoder().decode(PBTDB_str)).getImmutable();
    RB = CTB4.sub(Tools.HF3(bp.pairing(CTB0,PBTDB), pairingParametersFileName)).getImmutable();
} else if (memberOfB.equals("CB")) {
    String CBTDB1_str = TDB.getProperty("TD1");
    String CBTDB2_str = TDB.getProperty("TD2");
    Element CBTDB1 = G.newElementFromBytes(Base64.getDecoder().decode(CBTDB1_str)).getImmutable();
    Element CBTDB2 = G.newElementFromBytes(Base64.getDecoder().decode(CBTDB2_str)).getImmutable();
    RB = CTB4.sub(Tools.HF5(bp.pairing(CTB0,CBTDB1), bp.pairing(CTB0,CBTDB2), pairingParametersFileName)).getImmutable();
} else {
    System.out.println("非法 Entity");
    System.exit( status: -1);
}

```

圖三十二、*equalityTest()* 函式的部分程式碼

```

public static byte[] HF0(String id, Element a, Element b, Element c) {
    String a_str = Base64.getEncoder().encodeToString(a.toBytes());
    String b_str = Base64.getEncoder().encodeToString(b.toBytes());
    String c_str = Base64.getEncoder().encodeToString(c.toBytes());
    String hash_str = id + a_str + b_str + c_str;
    // byte[] rt;
    try {
        return sha1(hash_str);
    } catch (NoSuchAlgorithmException e) {
        e.printStackTrace();
        System.out.println("HF0、HF1: error!");
        System.exit( status: -1);
    }
    return null;
}

```

圖三十三、Tool.java 中的哈希函式

```

public static byte[] SE(String msg, byte[] sk){
    byte[] msgByte = msg.getBytes();
    int skLen = sk.length;
    byte[] rt = new byte[msgByte.length];
    for(int i=0;i<msgByte.length;i++){
        rt[i] = (byte) (msgByte[i] ^ sk[i%skLen]);
    }
    return rt;
}

```

圖三十四、Tool.java 中的 SE() 函式

### 5.5.3 成果展示

在 LR\_HHCB\_SCET.java 的 Main 函式中，我們創建了 4 個 Entity 分別是 *PBSender*、*PBReceiver*、*CBSender*、*CBReceiver*，前兩者的型態為 *PB\_Entity*，後兩者的型態為 *CB\_Entity*，如圖三十五。在這個演示中，*PBSender* 和 *CBSender* 扮演者傳輸訊息的腳色，他們會將訊息透過 *signcryption()* 加密成密文，並傳輸給 *PBReceiver* 和 *CBReceiver*。而 *PBReceiver* 和 *CBReceiver* 收到密文後，會透過 *unSigncryption()* 解密，程式碼如圖三十六，其中一封密文的內容如圖三十七，執行的結果如圖三十八。

```

// 加解密測試
// 創造 Entity
String entityDir = "data/entityinfo/";
String PBSenderFileName = entityDir + "pbsender.properties";
String PBSenderId = "pbsender@gmail.com";
PB_Entity PBSender = pb_pk.createEntity(PBSenderId,PBSenderFileName);

String PBReceiverFileName = entityDir + "pbreceiver.properties";
String PBReceiverId = "pbreceiver@gmail.com";
PB_Entity PBReceiver = pb_pk.createEntity(PBReceiverId,PBReceiverFileName);

String CBSenderFileName = entityDir + "cbsender.properties";
String CBSenderId = "cbsender@gmail.com";
CB_Entity CBSender = cb_pk.createEntity(CBSenderId,CBSenderFileName);

String CBReceiverFileName = entityDir + "cbreceiver.properties";
String CBReceiverId = "cbreceiver@gmail.com";
CB_Entity CBReceiver = cb_pk.createEntity(CBReceiverId,CBReceiverFileName);

```

圖三十五、創建 Entity 的部分程式碼

```

// -----test PB PB ----- //
// PBSender 加密訊息給 PBReceiver
String CTDir = "data/ct/";
String CT_PB2PB_FileName = CTDir + "pb2pb.properties";
String msg = "Hello, I'm PBSender! How are you PBReceiver? 我來自 PB 系統歐~";
PBSender.signcryption(msg,PBReceiverFileName,CT_PB2PB_FileName);

// PBReceiver 解密 PBSender 的訊息
String msg_ = PBReceiver.unSigncryption(PBSenderFileName,CT_PB2PB_FileName
System.out.println("PBReceiver 解 PBSender 的密文: " + msg_);

// -----test CB CB ----- //
// CBSender 加密訊息給 CBReceiver
String CT_CB2CB_FileName = CTDir + "cb2cb.properties";
msg = "Hello, I'm CBSender! How are you CBReceiver? 我來自 CB 系統歐~";
CBSender.signcryption(msg,CBReceiverFileName,CT_CB2CB_FileName);

// CBReceiver 解密 CBSender 的訊息
msg_ = CBReceiver.unSigncryption(CBSenderFileName,CT_CB2CB_FileName);
System.out.println("CBReceiver 解 CBSender 的密文: " + msg_);

// -----test PB CB ----- //
// PBSender 加密訊息給 CBReceiver
String CT_PB2CB_FileName = CTDir + "pb2cb.properties";
msg = "Hello, I'm PBSender! How are you CBReceiver? 我來自 PB 系統歐~";
PBSender.signcryption(msg,CBReceiverFileName,CT_PB2CB_FileName);

// CBReceiver 解密 PBSender 的訊息
msg_ = CBReceiver.unSigncryption(PBSenderFileName,CT_PB2CB_FileName);
System.out.println("CBReceiver 解 PBSender 的密文: " + msg_);

// -----test CB PB ----- //
// CBSender 加密訊息給 PBReceiver
String CT_CB2PB_FileName = CTDir + "cb2pb.properties";
msg = "Hello, I'm CBSender! How are you PBReceiver? 我來自 PB 系統歐~";
CBSender.signcryption(msg,PBReceiverFileName,CT_CB2PB_FileName);

// PBReceiver 解密 CBSender 的訊息
msg_ = PBReceiver.unSigncryption(CBSenderFileName,CT_CB2PB_FileName);
System.out.println("PBReceiver 解 CBSender 的密文: " + msg_);

```

圖三十六、Sender 和 Receiver 互動的部分程式碼

```

CT2=xAkD3VqyJzZ7nbrIStIEr0kh7g=
CT1=vabajK1KS16hFP0Idfx0jxUVurujVm6CIHd+6REIr7SsfY4T5R615oI02vcAdWHywhuEtirK5uhjKzt6TjR6TtsTzf8=
CT4=Bnzyg0opRY9s4a6TKhLmLp+6zchqfJFJj1YCgrPRSJxJxGzdkvFTt45IfVRowKzxJ9dy9HZKuCd3Jr7oVW7YHg1hMVojo3oMcj0A7MCe8gfJalwQugMaySu6oNM+vko/g4RMSGVDZ6q3xaq6mz3Hmb2HAEmUfr7pUAH7UIia0\=
CT3=fM+slH/kb0ULxARpAjxcs85JmmNphA7J9FvhldjwsBXcseLvnWSRpqNvFvrbxXMu/33yDEISWXARbxNvAv8r+pgxiVUp7VotBnCi4ml0o0Bc88tluJHJKbBQTGW6Avgx098cGo0Ppl4WeftLlQRQLD0j1FbrT/0qD0F3LEwoUoc\=
CT5=qOKH1JJdhrj04bwj77tqQW11sbtBNBqsm0fLzbuQFCDHUhukVWU7LMTLWeR1sP+8wP+&ndvxF2KZJBq3yNc065/pccp740KLmMmUnubbsE/FQhzshxWAzcIShLWk5GrAnPH5L/1TnuSpLzvjBce200soExDv42mnrf7nh\=
CT0=HNvkPPPMcd1aVKA/KTP9JBd4ArvukLn03+Mo2bzRNxpUF3YKah02yEfM+zr6ZdjxYR6r2ejEDTK+gxKSVyIJz6AKyHRQcuMKH6uMS4C5A7AFDT2Eir1kh8fc/Ev02vx6sYV0gFgJ1TE2Tf7NnT5n0AnvBka19LznraO/ArsAxoo\=

```

圖三十七、其中一封密文的內容

```

"C:\Program Files\Microsoft\jdk-11.0.12.7-hotspot\bin\java.exe" "-javaagent:C:\Program Files\JetBrains\IntelliJ IDEA
PBReceiver 解 PBSender 的密文: Hello, I'm PBSender! How are you PBReceiver? 我來自 PB 系統歐~
CBReceiver 解 CBSender 的密文: Hello, I'm CBSender! How are you CBReceiver? 我來自 CB 系統歐~
CBReceiver 解 PBSender 的密文: Hello, I'm PBSender! How are you CBReceiver? 我來自 PB 系統歐~
PBReceiver 解 CBSender 的密文: Hello, I'm CBSender! How are you PBReceiver? 我來自 PB 系統歐~

```

圖三十八、程式執行的結果

以上是成功解開的演示，這邊接著展示無法解密的例子。假設今天 *PBSender* 加密訊息給 *PBReceiver*，但這封密文不小心落入 *CBReceiver* 手中，由於並不是要傳給 *CBReceiver* 的，所以 *CBReceiver* 即使收到這封密文，也無法解開訊息，圖三十九為程式碼，圖四十為程式執行的解果，結果顯示會輸出”invalid”。

```

// 加解密測試-----
// 解不開的例子：PBSender 加密訊息給 PBReceiver，但這封密文不小心落入 CBReceiver 手中
msg = "Hello, I'm PBSender! How are you PBReceiver? 我來自 PB 系統歐~";
PBSender.signcryption(msg,PBReceiverFileName,CT_PB2PB_FileName);

msg_ = CBReceiver.unSigncryption(PBSenderFileName,CT_PB2PB_FileName);
System.out.println("CBReceiver 解 PBSender 傳送給 PBReceiver 的密文：" + msg_);

```

圖三十九、無法解密例子的程式碼

Test: PBSender 加密訊息給 PBReceiver，但這封密文不小心落入 CBReceiver 手中  
CBReceiver 解 PBSender 傳送給 PBReceiver 的密文: invalid

圖四十、程式執行的結果

接著演示相等性測試，一開始會先創立 4 個變數，如圖四十一，分別是 PB\_Entity 型態的 PBA、PBB 變數以及 CB\_Entity 型態的 CBA、CBB 變數，並將他們的 TD 傳給雲端，如圖四十二。這 4 個變數都扮演接收者的腳色，他們會將收到的密文傳送給雲端比對，看結果是否相同，程式碼如圖四十三，執行的結果如圖四十四。

```

// 相等性測試-----
// 創造 Entity;
String PBAFileName = entityDir + "pbA.properties";
String PBAId = "pbA@gmail.com";
PB_Entity PBA = pb_pk.createEntity(PBAId,PBAFileName);

String PBBFileName = entityDir + "pbB.properties";
String PBBId = "pbB@gmail.com";
PB_Entity PBB = pb_pk.createEntity(PBBId,PBBFileName);

String CBAFileName = entityDir + "cbA.properties";
String CBAId = "cbA@gmail.com";
CB_Entity CBA = cb_pk.createEntity(CBAId,CBAFileName);

String CBBFileName = entityDir + "cbB.properties";
String CBBId = "cbB@gmail.com";
CB_Entity CBB = cb_pk.createEntity(CBBId,CBBFileName);

String TDDir = "data/td/";
String PBATDFFileName = TDDir + "pbA.properties";
PBA.TDGen(PBATDFFileName);

String PBBTDFFileName = TDDir + "pbB.properties";
PBB.TDGen(PBBTDFFileName);

String CBATDFFileName = TDDir + "cbA.properties";
CBA.TDGen(CBATDFFileName);

String CBBTDFFileName = TDDir + "cbB.properties";
CBB.TDGen(CBBTDFFileName);

```

圖四十一、創建 Entity 的程式碼

圖四十二、製造 TD 的程式碼

```

// PB <-> PB
String CTAFFileName = CTDdir + "pbA.properties";
String CTBFileName = CTDdir + "pbB.properties";
PBSender.signcryption( msg: "Hello, 你好!", PBAFileName, CTAFFileName );
CBSender.signcryption( msg: "Hello, 你好!", PBBFileName, CTBFileName );
System.out.println("PBA 和 PBB 收到的密文進行相等性測試的結果：" + cloud.equalityTest(CTAFileName, PBATDFFileName, CTBFileName, PBBTDFFileName));

// CB <-> CB
CTAFileName = CTDdir + "cbA.properties";
CTBFileName = CTDdir + "cbB.properties";
PBSender.signcryption( msg: "Hello, 你好!", CBAFileName, CTAFFileName );
CBSender.signcryption( msg: "Hello, 你好!", CBBFileName, CTBFileName );
System.out.println("CBA 和 CBB 收到的密文進行相等性測試的結果：" + cloud.equalityTest(CTAFileName, CBATDFFileName, CTBFileName, CBBTDFFileName));

// PB <-> CB
CTAFileName = CTDdir + "pbA.properties";
CTBFileName = CTDdir + "cbB.properties";
PBSender.signcryption( msg: "Hello, 你好!", PBAFileName, CTAFFileName );
PBSender.signcryption( msg: "Hello, 你好!", CBBFileName, CTBFileName );
System.out.println("PBA 和 CBB 收到的密文進行相等性測試的結果：" + cloud.equalityTest(CTAFileName, PBATDFFileName, CTBFileName, CBBTDFFileName));

// CB <-> PB
CTAFileName = CTDdir + "cbA.properties";
CTBFileName = CTDdir + "pbB.properties";
CBSender.signcryption( msg: "Hello, 你好!", CBAFileName, CTAFFileName );
CBSender.signcryption( msg: "Hello, 你好!", PBBFileName, CTBFileName );
System.out.println("CBA 和 PBB 收到的密文進行相等性測試的結果：" + cloud.equalityTest(CTAFileName, CBATDFFileName, CTBFileName, PBBTDFFileName));

```

圖四十三、進行相等性測試的程式碼

PBA 和 PBB 收到的密文進行相等性測試的結果 : true  
 CBA 和 CBB 收到的密文進行相等性測試的結果 : true  
 PBA 和 CBB 收到的密文進行相等性測試的結果 : true  
 CBA 和 PBB 收到的密文進行相等性測試的結果 : true

圖四十四、進行相等性測試的結果

另外，在圖四十五演示了當收到的密文不一樣時，相等性測試的例子，可以觀察到不管是英文字母的大小寫不一致或是符號不一致，都會準確地判定為訊息不一致，結果如圖四十六。

```
// 相等性測試-----  

// 密文不一樣的例子  

// PB <-> PB  

CTAFileName = CTDdir + "pbA.properties";  

CTBFileName = CTDdir + "pbB.properties";  

PBSender.signcryption( msg: "Hello!", PBAFileName, CTAFileName );  

CBSender.signcryption( msg: "你好!", PBBFileName, CTBFileName );  

System.out.println("PBA 和 PBB 收到的密文進行相等性測試的結果:" + cloud.equalityTest(CTAFileName, PBATDFileName, CTBFileName, PBBTDFileName));  

// CB <-> CB  

CTAFileName = CTDdir + "cbA.properties";  

CTBFileName = CTDdir + "cbB.properties";  

PBSender.signcryption( msg: "你好!", CBAFileName, CTAFileName );  

CBSender.signcryption( msg: "你好~", CBBFileName, CTBFileName );  

System.out.println("CBA 和 CBB 收到的密文進行相等性測試的結果:" + cloud.equalityTest(CBAFileName, CBATDFileName, CTBFileName, CBBTDFileName));  

// PB <-> CB  

CTAFileName = CTDdir + "pbA.properties";  

CTBFileName = CTDdir + "cbB.properties";  

PBSender.signcryption( msg: "NI~", PBAFileName, CTAFileName );  

PBSender.signcryption( msg: "ni~", CBBFileName, CTBFileName );  

System.out.println("PBA 和 CBB 收到的密文進行相等性測試的結果:" + cloud.equalityTest(CTAFileName, PBATDFileName, CTBFileName, CBBTDFileName));  

// CB <-> PB  

CTAFileName = CTDdir + "cbA.properties";  

CTBFileName = CTDdir + "pbB.properties";  

CBSender.signcryption( msg: "你好!", CBAFileName, CTAFileName );  

CBSender.signcryption( msg: "你好~", PBBFileName, CTBFileName );  

System.out.println("CBA 和 PBB 收到的密文進行相等性測試的結果:" + cloud.equalityTest(CBAFileName, CBATDFileName, CTBFileName, PBBTDFileName));
```

圖四十五、密文不一致時進行相等性測試的程式碼

PBA 和 PBB 收到的密文進行相等性測試的結果 : false  
 CBA 和 CBB 收到的密文進行相等性測試的結果 : false  
 PBA 和 CBB 收到的密文進行相等性測試的結果 : false  
 CBA 和 PBB 收到的密文進行相等性測試的結果 : false

圖四十六、密文不一致時相等性測試的結果

## 5.6 討論與建議

儘管 LR-HHCB-SCRT 機制在免除金鑰託管和抗洩漏等方面具有優勢，然而仍有一些改進的空間，主要聚焦在時間成本的優化和撤銷問題上。在時間成本方面，我們應考慮進一步優化執行效率，以提升系統的整體性能。此外，我們的機制目前未考慮到實體解除註冊的情境，一旦實體加入系統並與憑證中心註冊，即無法解除。在實際應用中，撤銷功能是必要的，因此未來的改進方向之一是加入這項功能，以提供更完善的系統靈活性與安全性。

## 6 參考文獻

- [1] Hao Wang, Ottar L. Osen, Guoyuan Li, Wei Li, Hong-Ning Dai, and Wei Zeng. Big data and industrial internet of things for the maritime industry in northwestern norway. In *TENCON 2015 - 2015 IEEE Region 10 Conference*, pages 1–5, 2015.
- [2] Cheng Wang, Changjun Jiang, Yunhao Liu, Xiang-Yang Li, Shaojie Tang, and Huadong Ma. Aggregation capacity of wireless sensor networks: Extended network case. In *2011 Proceedings IEEE INFOCOM*, pages 1701–1709, 2011.
- [3] Rodrigo Roman and Javier Lopez. Integrating wireless sensor networks and the internet: a security analysis. *Internet Research*, 19(2):246–259, 2009.
- [4] 【 IoT 安全 】過 9 成工業物聯網失敗，資安問題成最大挑戰 (<https://inews.hket.com/article/3306065/%E3%80%90IoT%E5%AE%89%E5%85%A8%E3%80%91%E9%81%8E9%E6%88%90%E5%B7%A5%E6%A5%AD%E7%89%A9%E8%81%AF%E7%B6%B2%E5%A4%B1%E6%95%97%E3%80%80%E8%B3%87%E5%AE%89%E5%95%8F%E9%A1%8C%E6%88%90%E6%9C%80%E5%A4%A7%E6%8C%91%E6%88%B0>)
- [5] Hu Xiong, Yanan Zhao, Yingzhe Hou, Xin Huang, Chuanjie Jin, Lili Wang, and Saru Kumari. Heterogeneous signcryption with equality test for iiot environment. *IEEE Internet of Things Journal*, 8(21):16142–16152, 2021.
- [6] Craig Gentry. Certificate-based encryption and the certificate revocation problem. In *International Conference on the Theory and Applications of Cryptographic Techniques*, pages 272–293. Springer, 2003.
- [7] Paul C Kocher. Timing attacks on implementations of diffie-hellman, rsa, dss, and other systems. In *Advances in Cryptology—CRYPTO ’96: 16th Annual International Cryptology Conference Santa Barbara, California, USA August 18–22, 1996 Proceedings 16*, pages 104–113. Springer, 1996.
- [8] Yujue Wang, HweeHwa Pang, Robert H Deng, Yong Ding, Qianhong Wu, and Bo Qin. Securing messaging services through efficient signcryption with designated equality test. *Information Sciences*, 490:146–165, 2019.
- [9] Hu Xiong, Yingzhe Hou, Xin Huang, and Yanan Zhao. Secure message classification services through identity-based signcryption with equality test towards the internet of vehicles. *Vehicular Communications*, 26:100264, 2020.
- [10] YinXia Sun and Hui Li. Efficient signcryption between tpkc and idpkc and its multi-receiver construction. *Science China Information Sciences*, 53:557–566, 2010.
- [11] Qiong Huang, Duncan S Wong, and Guomin Yang. Heterogeneous signcryption with key privacy. *The Computer Journal*, 54(4):525–536, 2011.
- [12] Fagen Li, Hui Zhang, and Tsuyoshi Takagi. Efficient signcryption for heterogeneous

- systems. *IEEE Systems Journal*, 7(3):420–429, 2013.
- [13] Ronald L Rivest, Adi Shamir, and Leonard Adleman. A method for obtaining digital signatures and public-key cryptosystems. *Communications of the ACM*, 21(2):120–126, 1978.
- [14] Joël Alwen, Yevgeniy Dodis, and Daniel Wichs. Leakage-resilient public-key cryptography in the bounded-retrieval model. In *Advances in Cryptology-CRYPTO 2009: 29th Annual International Cryptology Conference, Santa Barbara, CA, USA, August 16-20, 2009. Proceedings*, pages 36–54. Springer, 2009.
- [15] Adi Akavia, Shafi Goldwasser, and Vinod Vaikuntanathan. Simultaneous hardcore bits and cryptography against memory attacks. In *Theory of cryptography conference*, pages 474–495. Springer, 2009.
- [16] Tung-Tso Tsai, Yuh-Min Tseng, and Sen-Shan Huang. Leakage-resilient certificateless signcryption scheme under a continual leakage model. *IEEE Access*, 11:54448–54461, 2023.
- [17] Jui-Di Wu, Yuh-Min Tseng, Sen-Shan Huang, and Wei-Chieh Chou. Leakage-resilient certificateless key encapsulation scheme. *Informatica*, 29(1):125–155, 2018.
- [18] Yuh-Min Tseng, Tung-Tso Tsai, and Sen-Shan Huang. Fully continuous leakage-resilient certificate-based signcryption scheme for mobile communications. *Informatica*, 34(1):199–222, 2023.
- [19] Yuh-Min Tseng, Tung-Tso Tsai, Sen-Shan Huang, and Ting-Chieh Ho. Leakage-resilient anonymous heterogeneous multi-receiver hybrid encryption in heterogeneous public-key system settings. *IEEE Access*, 12:28155–28168, 2024.
- [20] Steven D Galbraith and Pierrick Gaudry. Recent progress on the elliptic curve discrete logarithm problem. *Designs, Codes and Cryptography*, 78:51–72, 2016.

## 7 附錄

### 7.1 完整程式碼

#### **LR\_HHCB\_SCET.java**

```
import it.unisa.dia.gas.jpbc.Element;
import it.unisa.dia.gas.jpbc.Field;
import it.unisa.dia.gas.jpbc.Pairing;
import it.unisa.dia.gas.plaf.jpbc.pairing.PairingFactory;

import java.util.Base64;
import java.util.Properties;

public class LR_HHCB_SCET {
    public static PB_PKS pb_pks;
    public static CB_PKS cb_pks;
    public static Cloud cloud;
    public static void setUp(String pairingParametersFileName, String SPPFileName){
        // Pairing
        Pairing bp = PairingFactory.getPairing(pairingParametersFileName);
        Field G = bp.getG1();

        // 建立系統公開參數 SPP，並以文件方式存起來
        Element P = G.newRandomElement().getImmutable(); // G 的 generator
        Element S = G.newRandomElement().getImmutable();
        Element T = G.newRandomElement().getImmutable();
        Element U1 = G.newRandomElement().getImmutable();
        Element V1 = G.newRandomElement().getImmutable();
        Element U2 = G.newRandomElement().getImmutable();
        Element V2 = G.newRandomElement().getImmutable();

        Properties SPP = new Properties();
        SPP.setProperty("P", Base64.getEncoder().encodeToString(P.toBytes()));
        SPP.setProperty("S", Base64.getEncoder().encodeToString(S.toBytes()));
        SPP.setProperty("T", Base64.getEncoder().encodeToString(T.toBytes()));
        SPP.setProperty("U1", Base64.getEncoder().encodeToString(U1.toBytes()));
        SPP.setProperty("U2", Base64.getEncoder().encodeToString(V1.toBytes()));
        SPP.setProperty("V1", Base64.getEncoder().encodeToString(U2.toBytes()));
        SPP.setProperty("V2", Base64.getEncoder().encodeToString(V2.toBytes()));
        Tools.storePropToFile(SPP,SPPFileName);
```

```

pb_pk = new PB_PKS(pairingParametersFileName,SPPFileName);
cb_pk = new CB_PKS(pairingParametersFileName,SPPFileName);

SPP.setProperty("PBCAPK",
Base64.getEncoder().encodeToString(pb_pk.CAPk.toBytes()));
SPP.setProperty("CBCAPK",
Base64.getEncoder().encodeToString(cb_pk.CAPk.toBytes()));
Tools.storePropToFile(SPP,SPPFileName);

cloud = new Cloud(pairingParametersFileName);
}

// 建立 pb_pk, cb_pk
public static void main(String[] args){
// setup-----
String dir = "data/";
String pairingParametersFileName = dir + "a.properties";
String SPPFileName = dir + "spp.properties";
setUp(pairingParametersFileName, SPPFileName);

// 加解密測試-----
// 創造 Entity
String entityDir = "data/entityinfo/";
String PBSenderFileName = entityDir + "pbsender.properties";
String PBSenderId = "pbsender@gmail.com";
PB_Entity PBSender = pb_pk.createEntity(PBSenderId,PBSenderFileName);

String PBReceiverFileName = entityDir + "pbreceiver.properties";
String PBReceiverId = "pbreceiver@gmail.com";
PB_Entity PBReceiver = pb_pk.createEntity(PBReceiverId,PBReceiverFileName);

String CBSenderFileName = entityDir + "cbsender.properties";
String CBSenderId = "cbsender@gmail.com";
CB_Entity CBSender = cb_pk.createEntity(CBSenderId,CBSenderFileName);

String CBReceiverFileName = entityDir + "cbreceiver.properties";
String CBReceiverId = "cbreceiver@gmail.com";
CB_Entity CBReceiver = cb_pk.createEntity(CBReceiverId,CBReceiverFileName);

// -----test PB PB ----- //

```

```

// PBSender 加密訊息給 PBReceiver
String CTDir = "data/ct/";
String CT_PB2PB_FileName = CTDir + "pb2pb.properties";
String msg = "Hello, I'm PBSender! How are you PBReceiver? 我來自 PB 系統歐~";
PBSender.signcryption(msg,PBReceiverFileName,CT_PB2PB_FileName);

// PBReceiver 解密 PBSender 的訊息
String msg_ = PBReceiver.unSigncryption(PBSenderFileName,CT_PB2PB_FileName);
System.out.println("PBReceiver 解 PBSender 的密文: " + msg_);

// -----test CB CB -----
// CBSender 加密訊息給 CBReceiver
String CT_CB2CB_FileName = CTDir + "cb2cb.properties";
msg = "Hello, I'm CBSender! How are you CBReceiver? 我來自 CB 系統歐~";
CBSender.signcryption(msg,CBReceiverFileName,CT_CB2CB_FileName);

// CBReceiver 解密 CBSender 的訊息
msg_ = CBReceiver.unSigncryption(CBSenderFileName,CT_CB2CB_FileName);
System.out.println("CBReceiver 解 CBSender 的密文: " + msg_);

// -----test PB CB -----
// PBSender 加密訊息給 CBReceiver
String CT_PB2CB_FileName = CTDir + "pb2cb.properties";
msg = "Hello, I'm PBSender! How are you CBReceiver? 我來自 PB 系統歐~";
PBSender.signcryption(msg,CBReceiverFileName,CT_PB2CB_FileName);

// CBReceiver 解密 PBSender 的訊息
msg_ = CBReceiver.unSigncryption(PBSenderFileName,CT_PB2CB_FileName);
System.out.println("CBReceiver 解 PBSender 的密文: " + msg_);

// -----test CB PB -----
// CBSender 加密訊息給 PBReceiver
String CT_CB2PB_FileName = CTDir + "cb2pb.properties";
msg = "Hello, I'm CBSender! How are you PBReceiver? 我來自 PB 系統歐~";
CBSender.signcryption(msg,PBReceiverFileName,CT_CB2PB_FileName);

// PBReceiver 解密 CBSender 的訊息
msg_ = PBReceiver.unSigncryption(CBSenderFileName,CT_CB2PB_FileName);
System.out.println("PBReceiver 解 CBSender 的密文: " + msg_);

```

```

System.out.println("\n\nTest: PBSender 加密訊息給 PBReceiver，但這封密文不小心
落入 CBReceiver 手中");
// 加解密測試-----
// 解不開的例子: PBSender 加密訊息給 PBReceiver，但這封密文不小心落入
CBReceiver 手中
msg = "Hello, I'm PBSender! How are you PBReceiver? 我來自 PB 系統歐~";
PBSender.signcryption(msg,PBReceiverFileName,CT_PB2PB_FileName);

msg_ = CBReceiver.unSigncryption(PBSenderFileName,CT_PB2PB_FileName);
System.out.println("CBReceiver 解 PBSender 傳送給 PBReceiver 的密文: " + msg_);

// 相等性測試-----
// 創造 Entity;
String PBAFileName = entityDir + "pbA.properties";
String PBAId = "pbA@gmail.com";
PB_Entity PBA = pb_pks.createEntity(PBAId,PBAFileName);

String PBBFileName = entityDir + "pbB.properties";
String PBBrId = "pbB@gmail.com";
PB_Entity PBB = pb_pks.createEntity(PBBrId,PBBFileName);

String CBAFileName = entityDir + "cbA.properties";
String CBAId = "cbA@gmail.com";
CB_Entity CBA = cb_pks.createEntity(CBAId,CBAFileName);

String CBBFileName = entityDir + "cbB.properties";
String CBBRId = "cbB@gmail.com";
CB_Entity CBB = cb_pks.createEntity(CBBRId,CBBFileName);

String TDDir = "data/td/";
String PBATDFFileName = TDDir + "pbA.properties";
PBA.TDGen(PBATDFFileName);

String PBBTDFFileName = TDDir + "pbB.properties";
PBB.TDGen(PBBTDFFileName);

String CBATDFFileName = TDDir + "cbA.properties";
CBA.TDGen(CBATDFFileName);

```

```

String CBBTDFFileName = TDDir + "cbB.properties";
CBB.TDGen(CBBTDFFileName);

// PB <-> PB
String CTAFFileName = CTDdir + "pbA.properties";
String CTBFileName = CTDdir + "pbB.properties";
PBSender.signcryption("Hello , 你好!",PBAFileName,CTAFileName);
CBSender.signcryption("Hello , 你好!",PBBFileName,CTBFileName);
System.out.println("PBA 和 PBB 收到的密文進行相等性測試的結果:" +
cloud.equalityTest(CTAFileName,PBATDFFileName,CTBFileName ,PBTDFFileName));

// CB <-> CB
CTAFileName = CTDdir + "cbA.properties";
CTBFileName = CTDdir + "cbB.properties";
PBSender.signcryption("Hello , 你好!",CBAFileName,CTAFileName);
CBSender.signcryption("Hello , 你好!",CBBFileName,CTBFileName);
System.out.println("CBA 和 CBB 收到的密文進行相等性測試的結果:" +
cloud.equalityTest(CTAFileName,CBATDFFileName,CTBFileName ,CBBTDFFileName));

// PB <-> CB
CTAFileName = CTDdir + "pbA.properties";
CTBFileName = CTDdir + "cbB.properties";
PBSender.signcryption("Hello , 你好!",PBAFileName,CTAFileName);
PBSender.signcryption("Hello , 你好!",CBBFileName,CTBFileName);
System.out.println("PBA 和 CBB 收到的密文進行相等性測試的結果:" +
cloud.equalityTest(CTAFileName,PBATDFFileName,CTBFileName ,CBBTDFFileName));

// CB <-> PB
CTAFileName = CTDdir + "cbA.properties";
CTBFileName = CTDdir + "pbB.properties";
CBSender.signcryption("Hello , 你好!",CBAFileName,CTAFileName);
CBSender.signcryption("Hello , 你好!",PBBFileName,CTBFileName);
System.out.println("CBA 和 PBB 收到的密文進行相等性測試的結果:" +
cloud.equalityTest(CTAFileName,CBATDFFileName,CTBFileName ,PBTDFFileName));

// 相等性測試-----
// 密文不一樣的例子
// PB <-> PB
CTAFileName = CTDdir + "pbA.properties";

```

```

CTBFileName = CTDdir + "pbB.properties";
PBSender.signcryption("Hello!",PBAFileName,CTAFileName);
CBSender.signcryption("你好!",PBBFileName,CTBFileName);
System.out.println("PBA 和 PBB 收到的密文進行相等性測試的結果:" +
cloud.equalityTest(CTAFileName,PBATDFilename,CTBFileName ,PBTDFilename));

// CB <-> CB
CTAFileName = CTDdir + "cbA.properties";
CTBFileName = CTDdir + "cbB.properties";
PBSender.signcryption("你好!",CBAFileName,CTAFileName);
CBSender.signcryption("你好~",CBBFileName,CTBFileName);
System.out.println("CBA 和 CBB 收到的密文進行相等性測試的結果:" +
cloud.equalityTest(CTAFileName,CBATDFilename,CTBFileName ,CBBTDFilename));

// PB <-> CB
CTAFileName = CTDdir + "pbA.properties";
CTBFileName = CTDdir + "cbB.properties";
PBSender.signcryption("HI~",PBAFileName,CTAFileName);
PBSender.signcryption("hi~",CBBFileName,CTBFileName);
System.out.println("PBA 和 CBB 收到的密文進行相等性測試的結果:" +
cloud.equalityTest(CTAFileName,PBATDFilename,CTBFileName ,CBBTDFilename));

// CB <-> PB
CTAFileName = CTDdir + "cbA.properties";
CTBFileName = CTDdir + "pbB.properties";
CBSender.signcryption("你好!",CBAFileName,CTAFileName);
CBSender.signcryption("你好~",PBBFileName,CTBFileName);
System.out.println("CBA 和 PBB 收到的密文進行相等性測試的結果:" +
cloud.equalityTest(CTAFileName,CBATDFilename,CTBFileName ,PBTDFilename));
}

}

```

## PB\_PKS.java

```
import it.unisa.dia.gas.jpbc.Element;
import it.unisa.dia.gas.jpbc.Field;
import it.unisa.dia.gas.jpbc.Pairing;
import it.unisa.dia.gas.plaf.jpbc.pairing.PairingFactory;

import java.util.Base64;
import java.util.Properties;

public class PB_PKS{
    public Element CAPk;
    private Element CASk0,CASk1;
    private String pairingParametersFileName, SPPFileName;
    private Pairing bp;
    private Field G,Zq;
    private Properties SPP;
    private Element P,S,T;
    public PB_PKS(String pairingParametersFileName, String SPPFileName){
        this.pairingParametersFileName = pairingParametersFileName;
        this.SPPFileName = SPPFileName;
        // pairing
        this.bp = PairingFactory.getPairing(pairingParametersFileName);
        this.G = bp.getG1();
        this.Zq = bp.getZr();

        // 將公開參數 SPP 讀入
        this.SPP = Tools.loadPropFromFile(SPPFileName);

        // 還原 P(G 的 generator), S, T
        String P_str = SPP.getProperty("P");
        String S_str = SPP.getProperty("S");
        String T_str = SPP.getProperty("T");
        this.P = G.newElementFromBytes(Base64.getDecoder().decode(P_str)).getImmutable();
        this.S = G.newElementFromBytes(Base64.getDecoder().decode(S_str)).getImmutable();
        this.T = G.newElementFromBytes(Base64.getDecoder().decode(T_str)).getImmutable();

        // 做 CA 的公私鑰
        Element w1 = Zq.newRandomElement().getImmutable();
        Element CASK = P.mulZn(w1).getImmutable();
```

```

CAPk = bp.pairing(P,CASK).getImmutable();

// 將 CASK 一拆為二
Element a = Zq.newRandomElement().getImmutable();
CASK0 = P.mulZn(a).getImmutable();
CASK1 = CASK.sub(CASK0).getImmutable();
}

public PB_Entity createEntity(String id,String entityFileName){
    PB_Entity entity = new PB_Entity(pairingParametersFileName,SPPFileName,id,
entityFileName);
    crtGen(entity);
    return entity;
}

public void crtGen(PB_Entity entity) {
    // get entity pk1, pk2, id
    String entityId = entity.id;
    Element entityPk1 = entity.pk1;
    Element entityPk2 = entity.pk2;

    // 更新 sk
    Element a = Zq.newRandomElement().getImmutable();
    Element aP = P.mulZn(a).getImmutable();
    CASK0 = CASK0.add(aP).getImmutable();
    CASK1 = CASK1.sub(aP).getImmutable();

    // 製作 CRT0, CRT1
    a = Zq.newRandomElement().getImmutable();
    Element crt0 = P.mulZn(a).getImmutable();
    byte[] idPB_Byte = Tools.HF0(entityId,entityPk1,entityPk2,crt0);
    Element idPB =
        Zq.newElementFromHash(idPB_Byte,0,idPB_Byte.length).getImmutable();
    Element SidPBT = S.add(T.mulZn(idPB)).getImmutable();
    Element TC = CASK1.add(SidPBT.mulZn(a)).getImmutable();
    Element crt1 = CASK0.add(TC).getImmutable();
    entity.setCrt(crt0,crt1);
}

}

```

### **PB\_Entity.java**

```
import it.unisa.dia.gas.jpbc.Element;
import it.unisa.dia.gas.jpbc.Field;
import it.unisa.dia.gas.jpbc.Pairing;
import it.unisa.dia.gas.plaf.jpbc.pairing.PairingFactory;

import java.util.Base64;
import java.util.Properties;
import java.util.Random;

public class PB_Entity extends Entity{
    public Element crt0, crt1;
    public PB_Entity(String pairingParametersFileName, String SPPFileName, String id, String entityFileName){
        super(pairingParametersFileName,SPPFileName,id,entityFileName,"PB");
    }

    public void setCrt(Element crt0, Element crt1){
        this.crt0 = crt0;
        this.crt1 = crt1;
    }

    @Override
    protected void updateSk(Element P){
        Pairing bp = PairingFactory.getPairing(pairingParametersFileName);
        Field Zq = bp.getZr();
        Element r1 = Zq.newRandomElement().getImmutable();
        Element r2 = Zq.newRandomElement().getImmutable();
        Element r1P = P.mulZn(r1).getImmutable();
        Element r2P = P.mulZn(r2).getImmutable();
        sk10 = sk10.add(r1P).getImmutable();
        sk11 = sk11.sub(r1P).getImmutable();
        sk20 = sk20.add(r2P).getImmutable();
        sk21 = sk21.sub(r2P).getImmutable();
    }

    @Override
    protected Element TSGen(){
        return sk10.add(sk11.add(sk20.add(sk21))).getImmutable();
    }
}
```

```

}

@Override
protected byte[] H_Gen(Element CT0){
    Pairing bp = PairingFactory.getPairing(pairingParametersFileName);
    Element PBH1_ = bp.pairing(CT0,sk10.add(sk11)).getImmutable();
    Element PBH2_ = bp.pairing(CT0,sk20.add(sk21)).getImmutable();
    return Tools.HF2(PBH1_,PBH2_);
}

@Override
protected void storeTDFFile(String tdFileName){
    Properties prop = new Properties();
    prop.setProperty("TD",
Base64.getEncoder().encodeToString((sk20.add(sk21)).toBytes()));
    prop.setProperty("memberOf", memberOf);
    Tools.storePropToFile(prop,tdFileName);
}
}

```

## **CB\_PKS.java**

```
import it.unisa.dia.gas.jpbc.Element;
import it.unisa.dia.gas.jpbc.Field;
import it.unisa.dia.gas.jpbc.Pairing;
import it.unisa.dia.gas.plaf.jpbc.pairing.PairingFactory;

import java.util.Base64;
import java.util.Properties;

public class CB_PKS{
    public Element CAPk;
    private Element CASk0,CASk1;
    private String pairingParametersFileName, SPPFileName;
    private Pairing bp;
    private Field G,Zq;
    private Properties SPP;
    private Element P,U1,U2,V1,V2;
    public CB_PKS(String pairingParametersFileName, String SPPFileName){
        this.pairingParametersFileName = pairingParametersFileName;
        this.SPPFileName = SPPFileName;

        // pairing
        this.bp = PairingFactory.getPairing(pairingParametersFileName);
        this.G = bp.getG1();
        this.Zq = bp.getZr();

        // 將公開參數 SPP 讀入
        this.SPP = Tools.loadPropFromFile(SPPFileName);
        String P_str = SPP.getProperty("P");
        String U1_str = SPP.getProperty("U1");
        String U2_str = SPP.getProperty("U2");
        String V1_str = SPP.getProperty("V1");
        String V2_str = SPP.getProperty("V2");
        this.P = G.newElementFromBytes(Base64.getDecoder().decode(P_str)).getImmutable();
        this.U1=
        G.newElementFromBytes(Base64.getDecoder().decode(U1_str)).getImmutable();
        this.U2=
        G.newElementFromBytes(Base64.getDecoder().decode(U2_str)).getImmutable();
        this.V1=
```

```

G.newElementFromBytes(Base64.getDecoder().decode(V1_str)).getImmutable();
    this.V2=
G.newElementFromBytes(Base64.getDecoder().decode(V2_str)).getImmutable();

// 做 CA 的公私鑰
Element w1 = Zq.newRandomElement().getImmutable();
Element CASK = P.mulZn(w1).getImmutable();
CAPk = bp.pairing(P,CASK).getImmutable();

// 將 CASK 一拆為二
Element b = Zq.newRandomElement().getImmutable();
CASK0 = P.mulZn(b).getImmutable();
CASK1 = CASK.sub(CASK0).getImmutable();

}

public CB_Entity createEntity(String id,String entityFileName){
    CB_Entity entity = new CB_Entity(pairingParametersFileName,SPPFileName,id,
entityFileName);
    crtGen(entity);
    return entity;
}

public void crtGen(CB_Entity entity) {
    // get entity pk1, pk2, id
    String entityId = entity.id;
    Element entityPk1 = entity.pk1;
    Element entityPk2 = entity.pk2;

    // 更新 sk
    Element b = Zq.newRandomElement().getImmutable();
    Element bP = P.mulZn(b).getImmutable();
    CASK0 = CASK0.add(bP).getImmutable();
    CASK1 = CASK1.sub(bP).getImmutable();

    // pk3
    Element x2 = Zq.newRandomElement().getImmutable();
    Element entityPk3 = P.mulZn(x2).getImmutable();

    // 製作 CRT10, CRT11, CRT20, CRT21
}

```

```

byte[] idCB_Byte = Tools.HF1(entityId,entityPk1,entityPk2,entityPk3);
Element idCB =
Zq.newElementFromHash(idCB_Byte,0,idCB_Byte.length).getImmutable();
Element crt1 =
(CASk0.add(CASk1)).add((U1.add(V1.mulZn(idCB))).mulZn(x2)).getImmutable();
Element crt2 =
(CASk0.add(CASk1)).add((U2.add(V2.mulZn(idCB))).mulZn(x2)).getImmutable();
Element z1 = Zq.newRandomElement().getImmutable();
Element z2 = Zq.newRandomElement().getImmutable();
Element crt10 = P.mulZn(z1).getImmutable();
Element crt11 = crt1.sub(crt10).getImmutable();
Element crt20 = P.mulZn(z2).getImmutable();
Element crt21 = crt2.sub(crt20).getImmutable();
entity.setCrt(entityPk3,crt10,crt11,crt20,crt21);
}
}

```

## **CB\_Entity.java**

```
import it.unisa.dia.gas.jpbc.Element;
import it.unisa.dia.gas.jpbc.Field;
import it.unisa.dia.gas.jpbc.Pairing;
import it.unisa.dia.gas.plaf.jpbc.pairing.PairingFactory;

import java.util.Base64;
import java.util.Properties;

public class CB_Entity extends Entity{
    public Element pk3;
    private Element crt10, crt11, crt20, crt21;
    public CB_Entity(String pairingParametersFileName, String SPPFileName, String id, String entityFileName){
        super(pairingParametersFileName,SPPFileName,id,entityFileName,"CB");
    }

    public void setCrt(Element pk3, Element crt10, Element crt11, Element crt20, Element crt21) {
        this.pk3 = pk3;
        this.crt10 = crt10;
        this.crt11 = crt11;
        this.crt20 = crt20;
        this.crt21 = crt21;
        // 將 pk3 放進文件中
        Properties prop = Tools.loadPropFromFile(entityFileName);
        prop.setProperty("pk3", Base64.getEncoder().encodeToString(pk3.toBytes()));
        Tools.storePropToFile(prop,entityFileName);
    }

    @Override
    protected void updateSk(Element P){
        Pairing bp = PairingFactory.getPairing(pairingParametersFileName);
        Field Zq = bp.getZr();
        Element z1 = Zq.newRandomElement().getImmutable();
        Element z2 = Zq.newRandomElement().getImmutable();
        Element d1 = Zq.newRandomElement().getImmutable();
        Element d2 = Zq.newRandomElement().getImmutable();
        Element z1P = P.mulZn(z1).getImmutable();
        Element z2P = P.mulZn(z2).getImmutable();
    }
}
```

```

Element d1P = P.mulZn(d1).getImmutable();
Element d2P = P.mulZn(d2).getImmutable();
sk10 = sk10.add(z1P).getImmutable();
sk11 = sk11.sub(z1P).getImmutable();
sk20 = sk20.add(z2P).getImmutable();
sk21 = sk21.sub(z2P).getImmutable();
crt10 = crt10.add(d1P).getImmutable();
crt11 = crt11.sub(d1P).getImmutable();
crt20 = crt20.add(d2P).getImmutable();
crt21 = crt21.sub(d2P).getImmutable();
}

@Override
protected Element TSGen(){
    Element tp = sk10.add(sk11.add(sk20.add(sk21))).getImmutable();
    Element tp2 = crt10.add(crt11.add(crt20.add(crt21))).getImmutable();
    return tp.add(tp2).getImmutable();
}

@Override
protected byte[] H_Gen(Element CT0){
    Pairing bp = PairingFactory.getPairing(pairingParametersFileName);
    Element PBH1_ = bp.pairing(CT0,sk10.add(sk11)).getImmutable();
    Element PBH2_ = bp.pairing(CT0,sk20.add(sk21)).getImmutable();
    Element PBH3_ = bp.pairing(CT0,crt10.add(crt11)).getImmutable();
    Element PBH4_ = bp.pairing(CT0,crt20.add(crt21)).getImmutable();
    return Tools.HF4(PBH1_,PBH2_,PBH3_,PBH4_);
}

@Override
protected void storeTDFfile(String tdFileName){
    Properties prop = new Properties();
    prop.setProperty("TD1",
Base64.getEncoder().encodeToString((sk20.add(sk21)).toBytes()));
    prop.setProperty("TD2",
Base64.getEncoder().encodeToString((crt20.add(crt21)).toBytes()));
    prop.setProperty("memberOf", memberOf);
    Tools.storePropToFile(prop,tdFileName);
}
}

```

## Entity.java

```
import it.unisa.dia.gas.jpbc.Element;
import it.unisa.dia.gas.jpbc.Field;
import it.unisa.dia.gas.jpbc.Pairing;
import it.unisa.dia.gas.plaf.jpbc.pairing.PairingFactory;

import java.util.Base64;
import java.util.Properties;

public abstract class Entity {

    public Element pk1, pk2;
    protected Element sk10,sk11,sk20,sk21;
    public String memberOf, id;
    protected String pairingParametersFileName, SPPFileName, entityFileName;
    public Entity(String pairingParametersFileName, String SPPFileName, String id, String
entityFileName, String memberOf){
        this.pairingParametersFileName = pairingParametersFileName;
        this.SPPFileName = SPPFileName;
        this.id = id;
        this.entityFileName = entityFileName;
        this.memberOf = memberOf;
        selfKeyGen();
    }
    private void selfKeyGen(){
        // pairing
        Pairing bp = PairingFactory.getPairing(pairingParametersFileName);
        Field G = bp.getG1();
        Field Zq = bp.getZr();
        // 將公開參數 SPP 讀入
        Properties SPP = Tools.loadPropFromFile(SPPFileName);
        // 還原 P (G 的 generator)
        String P_str = SPP.getProperty("P");
        Element P =
        G.newElementFromBytes(Base64.getDecoder().decode(P_str)).getImmutable();
        // 做 Entity 的公私鑰
        Element y1 = Zq.newRandomElement().getImmutable();
        Element y2 = Zq.newRandomElement().getImmutable();
        Element sk1 = P.mulZn(y1).getImmutable();
        Element sk2 = P.mulZn(y2).getImmutable();
    }
}
```

```

pk1 = bp.pairing(P,sk1).getImmutable();
pk2 = bp.pairing(P,sk2).getImmutable();

// 將 SK1, SK2 一拆為二
Element r1 = Zq.newRandomElement().getImmutable();
Element r2 = Zq.newRandomElement().getImmutable();
sk10 = P.mulZn(r1).getImmutable();
sk11 = sk1.sub(sk10).getImmutable();
sk20 = P.mulZn(r2).getImmutable();
sk21 = sk2.sub(sk20).getImmutable();

// 將 pk 以文件方式存起來
Properties prop = new Properties();
prop.setProperty("pk1", Base64.getEncoder().encodeToString(pk1.toBytes()));
prop.setProperty("pk2", Base64.getEncoder().encodeToString(pk2.toBytes()));
prop.setProperty("id", id);
prop.setProperty("memberOf", memberOf);
Tools.storePropToFile(prop,entityFileName);

}

public void TDGen(String tdFileName){
    // pairing
    Pairing bp = PairingFactory.getPairing(pairingParametersFileName);
    Field G = bp.getG1();

    // 將公開參數 SPP 讀入
    Properties SPP = Tools.loadPropFromFile(SPPFileName);

    // 還原 P (G 的 generator)
    String P_str = SPP.getProperty("P");
    Element P =
        G.newElementFromBytes(Base64.getDecoder().decode(P_str)).getImmutable();

    // update self sk
    updateSk(P);
    // 將 pk 以文件方式存起來
    storeTDFFile(tdFileName);
}

protected abstract void storeTDFFile(String tdFileName);
protected abstract void updateSk(Element P);

```

```

public void signcryption(String msg, String receiverFileName, String CTFileName) {
    // pairing
    Pairing bp = PairingFactory.getPairing(pairingParametersFileName);
    Field G = bp.getG1();
    Field GT = bp.getGT();
    Field Zq = bp.getZr();

    // 將公開參數 SPP 、receiverInfo 讀入
    Properties SPP = Tools.loadPropFromFile(SPPFileName);
    Properties receiverInfo = Tools.loadPropFromFile(receiverFileName);

    // SPP
    String P_str = SPP.getProperty("P");
    String S_str = SPP.getProperty("S");
    String T_str = SPP.getProperty("T");
    String U1_str = SPP.getProperty("U1");
    String U2_str = SPP.getProperty("U2");
    String V1_str = SPP.getProperty("V1");
    String V2_str = SPP.getProperty("V2");
    String CBCAPK_str = SPP.getProperty("CBCAPK");
    Element P =
        G.newElementFromBytes(Base64.getDecoder().decode(P_str)).getImmutable();
    Element U1 =
        G.newElementFromBytes(Base64.getDecoder().decode(U1_str)).getImmutable();
    Element U2 =
        G.newElementFromBytes(Base64.getDecoder().decode(U2_str)).getImmutable();
    Element V1 =
        G.newElementFromBytes(Base64.getDecoder().decode(V1_str)).getImmutable();
    Element V2 =
        G.newElementFromBytes(Base64.getDecoder().decode(V2_str)).getImmutable();
    Element S =
        G.newElementFromBytes(Base64.getDecoder().decode(S_str)).getImmutable();
    Element T =
        G.newElementFromBytes(Base64.getDecoder().decode(T_str)).getImmutable();
    Element CBCAPK =
        GT.newElementFromBytes(Base64.getDecoder().decode(CBCAPK_str)).getImmutable();

    // receiverInfo
    String receiverId = receiverInfo.getProperty("id");
    String receiverMemberOf = receiverInfo.getProperty("memberOf");

```

```

String pk1_str = receiverInfo.getProperty("pk1");
String pk2_str = receiverInfo.getProperty("pk2");
Element receiverPk1 =
GT.newElementFromBytes(Base64.getDecoder().decode(pk1_str)).getImmutable();
Element receiverPk2 =
GT.newElementFromBytes(Base64.getDecoder().decode(pk2_str)).getImmutable();
Element receiverPk3 = G.newOneElement();

if(receiverMemberOf.equals("CB")){
    String pk3_str = receiverInfo.getProperty("pk3");
    receiverPk3 =
G.newElementFromBytes(Base64.getDecoder().decode(pk3_str)).getImmutable();
}

// signcryption-----
// CT0
Element h = Zq.newRandomElement().getImmutable();
Element CT0 = P.mulZn(h).getImmutable();

// H、TT
byte[] H = new byte[20];
Element TT = G.newZeroElement();
if(receiverMemberOf.equals("PB")){
    Element PBH1 = receiverPk1.powZn(h).getImmutable();
    Element PBH2 = receiverPk2.powZn(h).getImmutable();
    H = Tools.HF2(PBH1,PBH2);
    TT = Tools.HF3(PBH2,pairingParametersFileName).getImmutable();
}else if(receiverMemberOf.equals("CB")){
    byte[] idR_Byte = Tools.HF0(receiverId,receiverPk1,receiverPk2,receiverPk3);
    Element idR =
Zq.newElementFromHash(idR_Byte,0,idR_Byte.length).getImmutable();
    Element PBH1 = receiverPk1.powZn(h).getImmutable();
    Element PBH2 = receiverPk2.powZn(h).getImmutable();
    Element PBH3 =
(CBCAPK.mul(bp.pairing(receiverPk3,U1.add(V1.mulZn(idR))))).powZn(h).getImmutable();
    Element PBH4 =
(CBCAPK.mul(bp.pairing(receiverPk3,U2.add(V2.mulZn(idR))))).powZn(h).getImmutable();
    H = Tools.HF4(PBH1,PBH2,PBH3,PBH4);
    TT = Tools.HF5(PBH2,PBH4,pairingParametersFileName).getImmutable();
}

```

```

}else{
    System.out.println("非法 Entity");
    System.exit(-1);
}

// update self sk
updateSk(P);

// TS
Element TS = TSGen();

// CT1
byte[] sk = Tools.genSk();
byte[] CT1 = Tools.SE(msg,sk);

// CT2
byte[] CT2 = Tools.XOR(Tools.HF6(H),sk);

// CT3
Element t = Zq.newRandomElement().getImmutable();
Element CT3 = P.mulZn(t).getImmutable();

// CT4
Element msgHash2G = Tools.HF7(msg,pairingParametersFileName).getImmutable();
Element CT4 = TT.add(msgHash2G.mulZn(t)).getImmutable();

// CT5
byte[] n_byte = Tools.HF8(msg,sk,CT0,CT1,CT2,CT3,CT4);
Element n = Zq.newElementFromBytes(n_byte).getImmutable();
Element SnT = S.add(T.mulZn(n)).getImmutable();
Element hSnT = SnT.mulZn(h).getImmutable();
Element CT5 = TS.add(hSnT).getImmutable();

// write CT -----
Properties prop = new Properties();
prop.setProperty("CT0", Base64.getEncoder().encodeToString(CT0.toBytes()));
prop.setProperty("CT1", Base64.getEncoder().encodeToString(CT1));
prop.setProperty("CT2", Base64.getEncoder().encodeToString(CT2));

```

```

prop.setProperty("CT3", Base64.getEncoder().encodeToString(CT3.toBytes()));
prop.setProperty("CT4", Base64.getEncoder().encodeToString(CT4.toBytes()));
prop.setProperty("CT5", Base64.getEncoder().encodeToString(CT5.toBytes()));
Tools.storePropToFile(prop, CTFileName);
}

protected abstract Element TSGen();
protected abstract byte[] H_Gen(Element CT0);
public String unSigncryption(String senderFileName, String CTFileName) {
    // pairing
    Pairing bp = PairingFactory.getPairing(pairingParametersFileName);
    Field G = bp.getG1();
    Field GT = bp.getGT();
    Field Zq = bp.getZr();

    // 將公開參數 SPP, senderInfo, CT 讀入
    Properties SPP = Tools.loadPropFromFile(SPPFileName);
    Properties receiverInfo = Tools.loadPropFromFile(senderFileName);
    Properties CT = Tools.loadPropFromFile(CTFileName);

    // SPP
    String P_str = SPP.getProperty("P");
    String S_str = SPP.getProperty("S");
    String T_str = SPP.getProperty("T");
    String U1_str = SPP.getProperty("U1");
    String U2_str = SPP.getProperty("U2");
    String V1_str = SPP.getProperty("V1");
    String V2_str = SPP.getProperty("V2");
    String CBCAPK_str = SPP.getProperty("CBCAPK");
    Element P =
        G.newElementFromBytes(Base64.getDecoder().decode(P_str)).getImmutable();
    Element S =
        G.newElementFromBytes(Base64.getDecoder().decode(S_str)).getImmutable();
    Element T =
        G.newElementFromBytes(Base64.getDecoder().decode(T_str)).getImmutable();
    Element U1 =
        G.newElementFromBytes(Base64.getDecoder().decode(U1_str)).getImmutable();
    Element U2 =
        G.newElementFromBytes(Base64.getDecoder().decode(U2_str)).getImmutable();
}

```

```

Element V1 =
G.newElementFromBytes(Base64.getDecoder().decode(V1_str)).getImmutable();
Element V2 =
G.newElementFromBytes(Base64.getDecoder().decode(V2_str)).getImmutable();
Element CBCAPK =
GT.newElementFromBytes(Base64.getDecoder().decode(CBCAPK_str)).getImmutable();

// senderInfo
String senderId = receiverInfo.getProperty("id");
String senderMemberOf = receiverInfo.getProperty("memberOf");
String pk1_str = receiverInfo.getProperty("pk1");
String pk2_str = receiverInfo.getProperty("pk2");
Element senderPk1 =
GT.newElementFromBytes(Base64.getDecoder().decode(pk1_str)).getImmutable();
Element senderPk2 =
GT.newElementFromBytes(Base64.getDecoder().decode(pk2_str)).getImmutable();
Element senderPk3 = G.newZeroElement();

if(senderMemberOf.equals("CB")){
    String pk3_str = receiverInfo.getProperty("pk3");
    senderPk3 =
G.newElementFromBytes(Base64.getDecoder().decode(pk3_str)).getImmutable();
}

// CT
String CT0_str = CT.getProperty("CT0");
String CT1_str = CT.getProperty("CT1");
String CT2_str = CT.getProperty("CT2");
String CT3_str = CT.getProperty("CT3");
String CT4_str = CT.getProperty("CT4");
String CT5_str = CT.getProperty("CT5");
Element CT0 =
G.newElementFromBytes(Base64.getDecoder().decode(CT0_str)).getImmutable();
byte[] CT1 = Base64.getDecoder().decode(CT1_str);
byte[] CT2 = Base64.getDecoder().decode(CT2_str);
Element CT3 =
G.newElementFromBytes(Base64.getDecoder().decode(CT3_str)).getImmutable();
Element CT4 =
G.newElementFromBytes(Base64.getDecoder().decode(CT4_str)).getImmutable();
Element CT5 =

```

```

G.newElementFromBytes(Base64.getDecoder().decode(CT5_str)).getImmutable();

// unSigncryption-----
// update self sk
updateSk(P);

// compute H', sk'
byte[] H_ = H_Gen(CT0);
byte[] sk_ = Tools.XOR(Tools.HF6(H_),CT2);
String msg_ = Tools.SD(CT1,sk_);

// varify
byte[] n_byte = Tools.HF8(msg_,sk_,CT0,CT1,CT2,CT3,CT4);
Element n_ = Zq.newElementFromBytes(n_byte).getImmutable();

Element testL = bp.pairing(P,CT5).getImmutable();
Element testR = GT.newZeroElement();
if(senderMemberOf.equals("PB")){
    testR =
senderPk1.mul(senderPk2).mul(bp.pairing(CT0,S.add(T.mulZn(n_))).getImmutable());
} else if (senderMemberOf.equals("CB")) {
    byte[] idCB_Byte = Tools.HF1(senderId, senderPk1, senderPk2, senderPk3);
    Element idCB =
Zq.newElementFromHash(idCB_Byte,0,idCB_Byte.length).getImmutable();
    testR =
CBCAPK.mul(CBCAPK).mul(senderPk1).mul(senderPk2).mul(bp.pairing(senderPk3,U1.add(V1.
mulZn(idCB)))).mul(bp.pairing(senderPk3,U2.add(V2.mulZn(idCB)))).mul(bp.pairing(CT0,S.add(
T.mulZn(n_))).getImmutable();
} else {
    System.out.println("非法 Entity");
    System.exit(-1);
}
// System.out.println("testMsg: " + msg_);
if(testL.isEqual(testR)) return msg_;

return "invalid";
}
}

```

## Cloud.java

```
import it.unisa.dia.gas.jpbc.Element;
import it.unisa.dia.gas.jpbc.Field;
import it.unisa.dia.gas.jpbc.Pairing;
import it.unisa.dia.gas.plaf.jpbc.pairing.PairingFactory;

import java.util.Base64;
import java.util.Properties;

public class Cloud {
    private String pairingParametersFileName;

    public Cloud(String pairingParametersFileName){
        this.pairingParametersFileName = pairingParametersFileName;
    }

    public boolean equalityTest(String CTAFileName, String TDAFileName, String
CTBFileName, String TDBFileName){
        // pairing
        Pairing bp = PairingFactory.getPairing(pairingParametersFileName);
        Field G = bp.getG1();

        // read CTA, CTB, TDA, TDB
        Properties CTA = Tools.loadPropFromFile(CTAFileName);
        Properties CTB = Tools.loadPropFromFile(CTBFileName);
        Properties TDA = Tools.loadPropFromFile(TDAFileName);
        Properties TDB = Tools.loadPropFromFile(TDBFileName);

        // CTA
        String CTA0_str = CTA.getProperty("CT0");
        String CTA3_str = CTA.getProperty("CT3");
        String CTA4_str = CTA.getProperty("CT4");
        Element CTA0 =
        G.newElementFromBytes(Base64.getDecoder().decode(CTA0_str)).getImmutable();
        Element CTA3 =
        G.newElementFromBytes(Base64.getDecoder().decode(CTA3_str)).getImmutable();
        Element CTA4 =
        G.newElementFromBytes(Base64.getDecoder().decode(CTA4_str)).getImmutable();

        // CTB
        String CTB0_str = CTB.getProperty("CT0");
```

```

String CTB3_str = CTB.getProperty("CT3");
String CTB4_str = CTB.getProperty("CT4");
Element CTB0 =
G.newElementFromBytes(Base64.getDecoder().decode(CTB0_str)).getImmutable();
Element CTB3 =
G.newElementFromBytes(Base64.getDecoder().decode(CTB3_str)).getImmutable();
Element CTB4 =
G.newElementFromBytes(Base64.getDecoder().decode(CTB4_str)).getImmutable();

// member
String memberOfA = TDA.getProperty("memberOf");
String memberOfB = TDB.getProperty("memberOf");

// RA
Element RA = G.newZeroElement();
if(memberOfA.equals("PB")){
    String PBTDA_str = TDA.getProperty("TD");
    Element PBTDA =
G.newElementFromBytes(Base64.getDecoder().decode(PBTDA_str)).getImmutable();
    RA = CTA4.sub(Tools.HF3(bp.pairing(CTA0,PBTDA),
pairingParametersFileName)).getImmutable();
} else if (memberOfA.equals("CB")) {
    String CBTDA1_str = TDA.getProperty("TD1");
    String CBTDA2_str = TDA.getProperty("TD2");
    Element CBTDA1 =
G.newElementFromBytes(Base64.getDecoder().decode(CBTDA1_str)).getImmutable();
    Element CBTDA2 =
G.newElementFromBytes(Base64.getDecoder().decode(CBTDA2_str)).getImmutable();
    RA = CTA4.sub(Tools.HF5(bp.pairing(CTA0,CBTDA1),
bp.pairing(CTA0,CBTDA2), pairingParametersFileName)).getImmutable();
} else {
    System.out.println("非法 Entity");
    System.exit(-1);
}

// RB
Element RB = G.newZeroElement();
if(memberOfB.equals("PB")){
    String PBTDB_str = TDB.getProperty("TD");

```

```

Element PBTDB =
G.newElementFromBytes(Base64.getDecoder().decode(PBTDB_str)).getImmutable();
    RB = CTB4.sub(Tools.HF3(bp.pairing(CTB0,PBTDB),
pairingParametersFileName)).getImmutable();
}else if (memberOfB.equals("CB")) {
    String CBTDB1_str = TDB.getProperty("TD1");
    String CBTDB2_str = TDB.getProperty("TD2");
    Element CBTDB1 =
G.newElementFromBytes(Base64.getDecoder().decode(CBTDB1_str)).getImmutable();
    Element CBTDB2 =
G.newElementFromBytes(Base64.getDecoder().decode(CBTDB2_str)).getImmutable();
    RB = CTB4.sub(Tools.HF5(bp.pairing(CTB0,CBTDB1),
bp.pairing(CTB0,CBTDB2), pairingParametersFileName)).getImmutable();
} else {
    System.out.println("非法 Entity");
    System.exit(-1);
}

return bp.pairing(RA,CTB3).isEqual(bp.pairing(RB,CTA3));
}
}

```

## Tool.java

```
import it.unisa.dia.gas.jpbc.Element;
import it.unisa.dia.gas.jpbc.Field;
import it.unisa.dia.gas.jpbc.Pairing;
import it.unisa.dia.gas.plaf.jpbc.pairing.PairingFactory;

import java.io.FileInputStream;
import java.io.FileOutputStream;
import java.io.IOException;
import java.security.MessageDigest;
import java.security.NoSuchAlgorithmException;
import java.util.Base64;
import java.util.Properties;
import java.util.Random;

public class Tools {

    public static void storePropToFile(Properties prop, String fileName){
        try(FileOutputStream out = new FileOutputStream(fileName)){
            prop.store(out,null);
        }
        catch (IOException e){
            e.printStackTrace();
            System.out.println(fileName + " save failed!");
            System.exit(-1);
        }
    }

    public static Properties loadPropFromFile(String fileName){
        Properties prop = new Properties();
        try(FileInputStream in = new FileInputStream(fileName)){
            prop.load(in);
        }
        catch(IOException e){
            e.printStackTrace();
            System.out.println(fileName + " load failed!");
            System.exit(-1);
        }
        return prop;
    }
}
```

```
}
```

```
public static byte[] HF0(String id, Element a, Element b, Element c) {
    String a_str = Base64.getEncoder().encodeToString(a.toBytes());
    String b_str = Base64.getEncoder().encodeToString(b.toBytes());
    String c_str = Base64.getEncoder().encodeToString(c.toBytes());
    String hash_str = id + a_str + b_str + c_str;
    // byte[] rt;
    try {
        return sha1(hash_str);
    } catch (NoSuchAlgorithmException e) {
        e.printStackTrace();
        System.out.println("HF0、HF1: error!");
        System.exit(-1);
    }
    return null;
}
```

```
public static byte[] HF1(String id, Element a, Element b, Element c) {
    return HF0(id,a,b,c);
}
```

```
public static byte[] HF2( Element a, Element b) {
    String a_str = Base64.getEncoder().encodeToString(a.toBytes());
    String b_str = Base64.getEncoder().encodeToString(b.toBytes());
    String hash_str = a_str + b_str;
    // byte[] rt;
    try {
        return sha1(hash_str);
    } catch (NoSuchAlgorithmException e) {
        e.printStackTrace();
        System.out.println("HF2: error!");
        System.exit(-1);
    }
    return null;
}
```

```
public static Element HF3(Element a,String pairingParametersFileName) {
    String hash_str = Base64.getEncoder().encodeToString(a.toBytes());
```

```

Pairing bp = PairingFactory.getPairing(pairingParametersFileName);
Field G = bp.getG1();
try {
    byte[] hashByte = sha1(hash_str);
    Element rt = G.newElementFromHash(hashByte,0,hashByte.length).getImmutable();
    return rt;
} catch (NoSuchAlgorithmException e) {
    e.printStackTrace();
    System.out.println("HF3: error!");
    System.exit(-1);
}
return null;
}

public static byte[] HF4( Element a, Element b, Element c, Element d ) {
    String a_str = Base64.getEncoder().encodeToString(a.toBytes());
    String b_str = Base64.getEncoder().encodeToString(b.toBytes());
    String c_str = Base64.getEncoder().encodeToString(c.toBytes());
    String d_str = Base64.getEncoder().encodeToString(d.toBytes());
    String hash_str = a_str + b_str + c_str + d_str;
    // byte[] rt;
    try {
        return sha1(hash_str);
    } catch (NoSuchAlgorithmException e) {
        e.printStackTrace();
        System.out.println("HF2: error!");
        System.exit(-1);
    }
    return null;
}

public static Element HF5(Element a,Element b, String pairingParametersFileName) {
    String a_str = Base64.getEncoder().encodeToString(a.toBytes());
    String b_str = Base64.getEncoder().encodeToString(b.toBytes());
    String hash_str = a_str + b_str;
    Pairing bp = PairingFactory.getPairing(pairingParametersFileName);
    Field G = bp.getG1();
    try {
        byte[] hashByte = sha1(hash_str);

```

```

Element rt = G.newElementFromHash(hashByte,0,hashByte.length).getImmutable();
return rt;
} catch (NoSuchAlgorithmException e) {
    e.printStackTrace();
    System.out.println("HF3: error!");
    System.exit(-1);
}
return null;
}

public static byte[] HF6(byte[] a){
try {
    return sha1(Base64.getEncoder().encodeToString(a));
} catch (NoSuchAlgorithmException e) {
    e.printStackTrace();
    System.out.println("HF6: error!");
    System.exit(-1);
}
return null;
}

public static Element HF7(String msg,String pairingParametersFileName){
    Pairing bp = PairingFactory.getPairing(pairingParametersFileName);
    Field G = bp.getG1();
    try {
        byte[] hashByte = sha1(msg);
        Element rt = G.newElementFromHash(hashByte,0,hashByte.length).getImmutable();
        return rt;
    } catch (NoSuchAlgorithmException e) {
        e.printStackTrace();
        System.out.println("HF7: error!");
        System.exit(-1);
    }
    return null;
}

public static byte[] HF8(String msg,byte[] sk, Element CT0, byte[] CT1, byte[] CT2,Element CT3, Element CT4){
    String sk_str = Base64.getEncoder().encodeToString(sk);
    String CT0_str = Base64.getEncoder().encodeToString(CT0.toBytes());

```

```

String CT1_str = Base64.getEncoder().encodeToString(CT1);
String CT2_str = Base64.getEncoder().encodeToString(CT2);
String CT3_str = Base64.getEncoder().encodeToString(CT3.toBytes());
String CT4_str = Base64.getEncoder().encodeToString(CT4.toBytes());
String hash_str = msg + sk_str + CT0_str + CT1_str + CT2_str + CT3_str + CT4_str;
try {
    return sha1(hash_str);
} catch (NoSuchAlgorithmException e) {
    e.printStackTrace();
    System.out.println("HF8: error!");
    System.exit(-1);
}
return null;
}

public static byte[] genSk(){
    Random random = new Random();
    byte[] randomBytes = new byte[20];
    random.nextBytes(randomBytes);
    return randomBytes;
}

public static byte[] SE(String msg, byte[] sk){
    byte[] msgByte = msg.getBytes();
    int skLen = sk.length;
    byte[] rt = new byte[msgByte.length];
    for(int i=0;i<msgByte.length;i++){
        rt[i] = (byte) (msgByte[i] ^ sk[i%skLen]);
    }
    return rt;
}

public static String SD(byte[] CT1, byte[] sk_){
    int skLen = sk_.length;
    byte[] rt = new byte[CT1.length];
    for(int i=0;i<CT1.length;i++){
        rt[i] = (byte) (CT1[i] ^ sk_[i%skLen]);
    }
    return new String(rt);
}

```

```
}

public static byte[] XOR(byte[] x, byte[] y){

    byte[] rt = new byte[x.length];
    for(int i=0;i<x.length;i++){
        rt[i] = (byte) (x[i] ^ y[i]);
    }
    // System.out.println(new String(msgByte));
    return rt;
}

public static byte[] sha1(String content) throws NoSuchAlgorithmException{
    MessageDigest instance = MessageDigest.getInstance("SHA-1");
    instance.update(content.getBytes());
    return instance.digest();
}

}
```