開發人員指南

AWS 資料庫加密 SDK

AWS 資料庫加密 SDK: 開發人員指南

Table of Contents

什麼是資 AWS 料庫加密 SDK？

我們的用戶端加密程式庫已重新命名為 AWS 資料庫加密 SDK。本開發人員指南仍提供 DynamoDB 加密用戶端的相關資訊。

資 AWS 料庫加密 SDK 是一組軟體程式庫，可讓您在資料庫設計中包含用戶端加密。 AWS 資料庫加密 SDK 提供記錄層級的加密解決方案。您可以指定要加密的欄位以及簽名中包含哪些欄位，以確保資料的真實性。對傳輸中和靜態的敏感資料進行加密，有助於確保您的純文字資料無法供任何第三方使用，包括. AWS數 AWS 據庫加密 SDK 是根據 Apache 2.0 許可證免費提供的。

本開發人員指南提供 AWS Database Encryption SDK 的概念性概觀，包括其架構簡介、如何保護您的資料、與伺服器端加密有何不同之處，以及為應用程式選取關鍵元件以協助您開始使用的指引。

資 AWS 料庫加密開發套件支援具有屬性層級加密功能的 Amazon DynamoDB。版本 3. 適用於 DynamoDB 的 Java 用戶端加密程式庫的 x 是針對 Java 的 DynamoDB 加密用戶端的主要重新寫入。它包含許多更新，例如新的結構化資料格式、改進的多租戶支援、可搜尋的加密，以及無縫結構描述變更的支援。

資 AWS 料庫加密 SDK 具有以下優點：

專為資料庫應用程式設計

您不需要成為密碼編譯專家即可使用 AWS 資料庫加密 SDK。這些實現包括旨在與現有應用程序配合使用的幫助程序方法。

建立並設定必要元件之後，當您將記錄新增至資料庫時，加密用戶端會透明地加密和簽署記錄，並在擷取記錄時驗證和解密它們。

包含安全加密和簽署

AWS Database Encryption SDK 包含安全實作，這些實作會使用唯一的資料加密金鑰加密每筆記錄中的欄位值，然後簽署記錄以防止未經授權的變更，例如新增或刪除欄位，或交換加密值。

使用來自任何來源的密碼編譯資料

資料 AWS 庫加密 SDK 使用金鑰環來產生、加密和解密唯一的資料加密金鑰，以保護您的記錄。金鑰圈會決定加密該資料金鑰的包裝金鑰。

您可以使用包裝來自任何來源的金鑰，包括密碼編譯服務，例如 AWS Key Management Service(AWS KMS) 或 AWS CloudHSM. 數 AWS 據庫加密 SDK 不需要 AWS 帳戶 或任何 AWS 服務。

Support 加密材料快取

AWS KMS 階層式金鑰圈是一種加密材料快取解決方案，可透過使用保存在 Amazon DynamoDB 表格中的 AWS KMS 受保護分支金鑰，然後在本機快取用於加密和解密作業的分支金鑰材料，以減少 AWS KMS 呼叫次數。它可讓您以對稱式加密 KMS 金鑰保護您的加密資料，而無需在 AWS KMS 每次加密或解密記錄時呼叫。 AWS KMS 分層式鑰匙圈對於需要盡量減少呼叫的應用程序來說是一個不錯的選擇。 AWS KMS

可搜索加密

您可以設計可以在不解密整個數據庫的情況下搜索加密記錄的數據庫。根據您的威脅模型和查詢需求，您可以使用可搜尋的加密，在加密的資料庫上執行完全比對搜尋或更自訂的複雜查詢。

Support 多租戶資料庫結構描述

資料 AWS 庫加密 SDK 可讓您使用不同的加密材料隔離每個租用戶，以共用結構描述保護儲存在資料庫中的資料。如果您有多個使用者在資料庫中執行加密作業，請使用其中一個金 AWS KMS 鑰環，為每位使用者提供不同的金鑰，以便在其密碼編譯作業中使用。如需詳細資訊，請參閱 使用多租戶資料庫。

Support 無縫結構描述更新

當您設定 AWS 資料庫加密 SDK 時，您會提供密碼編譯動作，告訴用戶端要加密和簽署哪些欄位、要簽署 (但不加密) 的欄位，以及要忽略哪些欄位。使用資料 AWS 庫加密 SDK 保護記錄之後，您仍然可以對資料模型進行變更。您可以在單一部署中更新加密動作，例如新增或移除加密欄位。

在開源存儲庫中開發

數 AWS 據庫加密 SDK 是在開源存儲庫中開發的 GitHub。您可以使用這些存放庫來檢視程式碼、讀取和提交問題，以及尋找實作專屬的資訊。

適用於 DynamoDB 的 AWS 資料庫加密開發套件

- 上的 aws-database-encryption-sdk-dynamodb 儲存庫 GitHub 支援第 3 版。 x 及更新版本的 AWS 資料庫加密開發 DynamoDB 適用於 Java 和 .NET)。

版本 3. DynamoDB 的 AWS 資料庫加密 SDK 的 x 是 Dafny 的產品，Dafny 是您撰寫規格時所使用的驗證感知語言、實作規格的程式碼，以及測試它們的校樣。其結果是一個程式 AWS 庫，在可確保功能正確性的架構中實作適用於 DynamoDB 的資料庫加密 SDK 功能。

Support 與維護

資 AWS 料庫加密 SDK 使用與 AWS SDK 和工具使用的相同維護原則，包括其版本控制和生命週期階段。最佳作法是，建議您使用最新版本的 AWS 資料庫加密 SDK 來實作資料庫，並在新版本發行時升級。

如需詳細資訊，請參閱 AWS SDK 和工具參考指南中的 AWS SDK 和工具維護原則。

傳送意見回饋

我們誠摯歡迎您提供意見回饋。如果您有問題或意見、問題或報告，請使用以下資源。

如果您在 AWS 資料庫加密 SDK 中發現潛在的安全漏洞，請通知 AWS 安全性。請勿建立公開 GitHub 問題。

若要提供有關此文件的意見回饋，請使用任何頁面上的意見回饋連結。

AWS 資料庫加密 SDK 概念

我們的用戶端加密程式庫已重新命名為 AWS 資料庫加密 SDK。本開發人員指南仍提供 DynamoDB 加密用戶端的相關資訊。

本主題說明資 AWS 料庫加密 SDK 中使用的概念和術語。

若要瞭解 AWS 資料庫加密 SDK 的元件如何互動，請參閱AWS 資料庫加密 SDK 的運作方式。

若要深入了解資 AWS 料庫加密 SDK，請參閱下列主題。

- 瞭解資料 AWS 庫加密 SDK 如何使用信封加密來保護您的資料。
- 瞭解信封加密的元素：保護記錄的資料金鑰，以及保護資料金鑰的包裝金鑰。
- 瞭解決定您使用哪些環繞金鑰的鑰匙圈。

- 瞭解為您的加密程序增加完整性的加密內容。
- 瞭解加密方法新增至記錄的材料說明。
- 瞭解告知 AWS 資料庫加密 SDK 要加密和簽署哪些欄位的加密動作。

主題

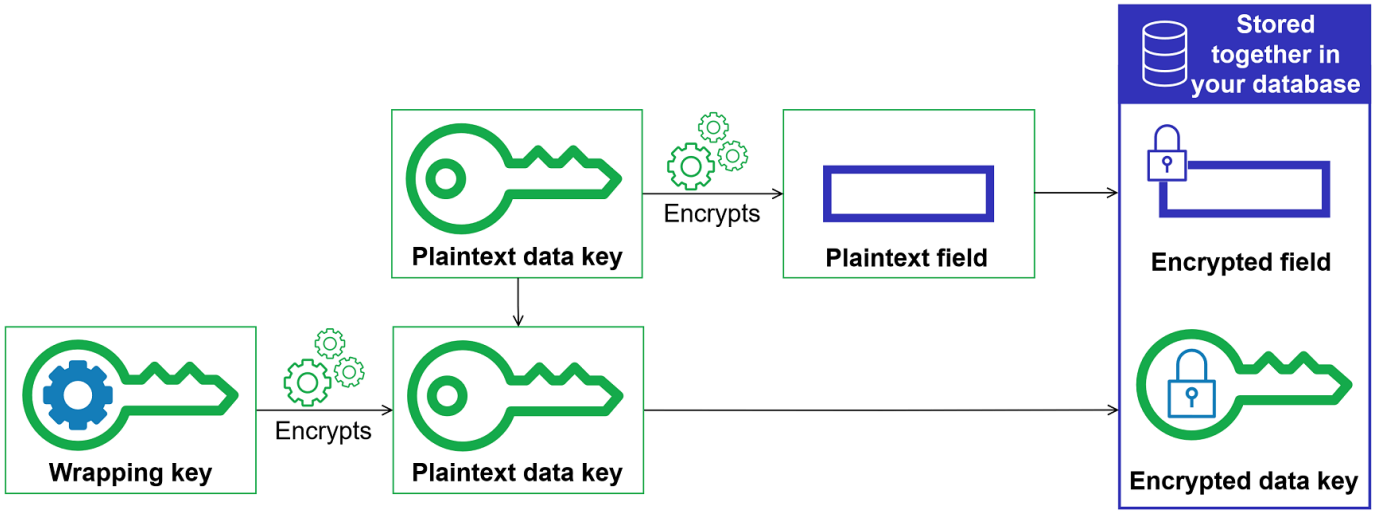
封套加密

加密資料的安全性有一部分取決於保護能夠解密資料的資料金鑰。加密處理金鑰是保護資料金鑰的一種最佳實務。若要這麼做，您需要另一個加密金鑰，稱為金鑰加密金鑰或包裝金鑰。使用包裝金鑰來加密資料金鑰的做法稱為信封加密。

保護資料金鑰

數據 AWS 庫加密 SDK 使用唯一的數據密鑰對每個字段進行加密。然後它加密您指定的包裝密鑰下的每個數據密鑰。它存儲在材料描述中的加密數據密鑰。

若要指定包裝金鑰，請使用金鑰圈。



在多個包裝密鑰下加密相同的數據

您可以使用多個包裝金鑰來加密資料金鑰。您可能希望為不同的用戶提供不同的包裝鍵，或者包裝不同類型或不同位置的密鑰。每個包裝密鑰都會加密相同的數據密鑰。數據 AWS 庫加密 SDK 將所有加密的數據密鑰與材料描述中的加密字段一起存儲。

要解密數據，您需要提供至少一個可以解密加密數據密鑰的包裝密鑰。

結合多種演算法的優勢

為了加密您的數據，默認情況下，數據 AWS 庫加密 SDK 使用具有 AES-GCM 對稱加密的算法套件，基於 HMAC 的密鑰派生功能（HKDF）和 ECDSA 簽名。若要加密資料金鑰，您可以指定適用於包裝金鑰的對稱或非對稱加密演算法。

一般而言，相較於非對稱或公有金鑰加密，對稱金鑰加密演算法速度較快，產生的加密文字較小。但是公鑰算法提供角色固有的分離。若要結合每個項目的優點，您可以使用公開金鑰加密來加密資料金鑰。

我們建議盡可能使用其中一個 AWS KMS 鑰匙圈。使用AWS KMS 金鑰圈時，您可以指定非對稱 RSA AWS KMS key 作為包裝金鑰，來選擇結合多種演算法的優勢。您也可以使用對稱加密 KMS 金鑰。

資料金鑰

資料金鑰是資料 AWS 庫加密 SDK 用來加密記錄中標記在加密動作ENCRYPT_AND_SIGN中的欄位的加密金鑰。每個資料金鑰是符合密碼編譯金鑰需求的位元組陣列。資料 AWS 庫加密 SDK 使用唯一的資料金鑰來加密每個屬性。

您不需要指定、產生、實作、擴充、保護或使用資料金鑰。當您調用加密和解密操作時， AWS 數據庫加密 SDK 可以為您工作。

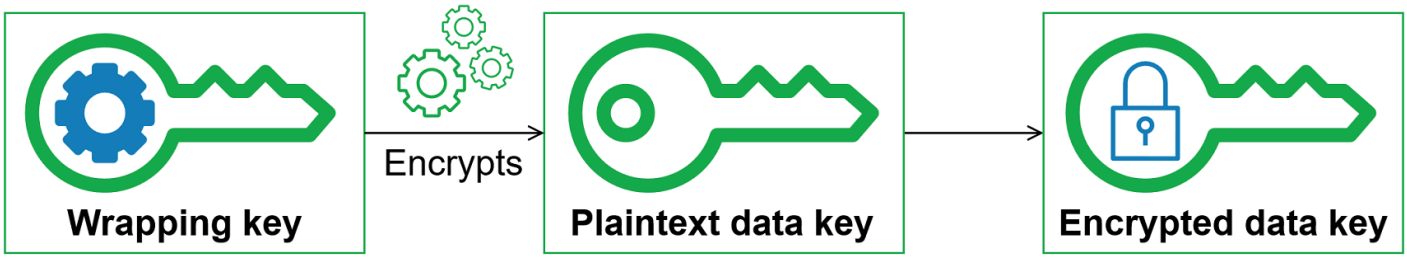
為了保護您的資料金鑰，資料 AWS 庫加密 SDK 會使用一或多個稱為包裝金鑰的金鑰加密金鑰來加密這些金鑰。資料 AWS 庫加密 SDK 使用您的純文字資料金鑰加密資料之後，會盡快將其從記憶體中移除。然後將加密的數據密鑰存儲在材料描述中。如需詳細資訊，請參閱 AWS 資料庫加密 SDK 的運作方式。

> **Tip**
>
> 在資料 AWS 庫加密 SDK 中，我們將資料金鑰與資料加密金鑰區分開來。作為最佳實踐，所有支持的算法套件都使用密鑰派生函數。密鑰派生函數接受一個數據密鑰作為輸入，並返回實際用於加密記錄的數據加密密鑰。因此，我們通常會說資料是在資料金鑰「底下」加密，而不是「由」資料金鑰加密。

每個加密的資料金鑰都包含中繼資料，包括加密之環繞金鑰的識別碼。此中繼資料可讓資料 AWS 庫加密 SDK 在解密時識別有效的包裝金鑰。

包裝鍵

包裝金鑰是一種金鑰加密金鑰，資料 AWS 庫加密 SDK 用來加密記錄的資料金鑰。每個數據密鑰可以在一個或多個包裝密鑰下進行加密。設定金鑰圈時，您可以決定使用哪些包裝金鑰來保護您的資料。



AWS 資料庫加密 SDK 支援數種常用的包裝金鑰，例如 AWS Key Management Service(AWS KMS) 對稱加密 KMS 金鑰 (包括多區域金鑰) 和非對稱 RSA KMS 金 AWS KMS 鑰、原始 AES-GCM (進階加密標準/Galois 計數器模式) 金鑰，以及原始 RSA 金鑰。我們建議盡可能使用 KMS 金鑰。若要決定您應該使用哪個包裝金鑰，請參閱選取包裝金鑰。

當您使用信封加密時，您需要保護包裝密鑰免受未經授權的訪問。您可以透過下列任何一種方式執行此操作：

- 使用專為此目的而設計的服務，例如 AWS Key Management Service (AWS KMS)。

- 使用硬體安全模組 (HSM)，例如 AWS CloudHSM 所提供的功能。
- 使用其他金鑰管理工具和服務。

如果您沒有金鑰管理系統，我們建議您使用 AWS KMS。資 AWS 料庫加密 SDK 與 AWS KMS 整合，可協助您保護和使用包裝金鑰。

鑰匙圈

若要指定用於加密和解密的包裝金鑰，請使用金鑰環。您可以使用資 AWS 料庫加密 SDK 提供的金鑰環或設計您自己的實作。

Keying 會產生、加密和解密資料金鑰。它還會生成用於計算簽名中基於散列的消息身份驗證碼（HMAC）的 MAC 密鑰。當您定義金鑰環時，您可以指定加密資料金鑰的包裝金鑰。大多數金鑰圈至少會指定一個包裝金鑰或提供並保護包裝金鑰的服務。加密時，資料 AWS 庫加密 SDK 會使用金鑰圈中指定的所有包裝金鑰來加密資料金鑰。如需有關選擇和使用 AWS 資料庫加密 SDK 定義之金鑰環的說明，請參閱使用金鑰環。

密碼編譯動作

加密動作會告訴加密程式要對記錄中的每個欄位執行哪些動作。

密碼編譯動作值可以是下列其中一項：

- 加密和簽署 — 加密欄位。在簽名中包含加密的欄位。
- 僅簽署 — 在簽名中包含欄位。
- 簽署並包含在加密內容中 — 在簽名和加密內容中包含欄位。

  依預設，分割區和排序索引鍵是加密內容中包含的唯一屬性。您可以考慮定義其他欄位，以`SIGN_AND_INCLUDE_IN_ENCRYPTION_CONTEXT`便AWS KMS 階層式金鑰圈的分支金鑰 ID 供應商可識別從加密內容解密所需的分支金鑰。如需詳細資訊，請參閱分支金鑰 ID 供應商。

  > **Note**
  >
  > 若要使用加`SIGN_AND_INCLUDE_IN_ENCRYPTION_CONTEXT`密動作，您必須使用 3.3 版或更新版本的 AWS 資料庫加密 SDK。在更新要包含的資料模型之前，請先將新版本部署至所有讀取器`SIGN_AND_INCLUDE_IN_ENCRYPTION_CONTEXT`。

- 不執行任何動作 — 請勿加密或在簽名中包含欄位。

對於任何可以儲存敏感資料的欄位，請使用「加密並簽署」。對於主索引鍵值 (例如 DynamoDB 表中的分區索引鍵和排序金鑰)，請使用「僅簽署」或「簽署並包含在加密內容中」。如果您指定任何 Sign 並包含在加密內容屬性中，則分割區和排序屬性也必須是 Sign 並包含在加密內容中。您不需要為材料描述指定密碼編譯動作。資料 AWS 庫加密 SDK 會自動簽署儲存材料描述的欄位。

仔細選擇您的加密操作。如有疑問，請使用加密並簽署。使用 AWS 資料庫加密 SDK 來保護記錄之後，您無法將現有、或SIGN_AND_INCLUDE_IN_ENCRYPTION_CONTEXT欄位變更為指派給現有ENCRYPT_AND_SIGN欄位的密碼編譯動作DO_NOTHING，也無法變更指派給現有DO_NOTHING欄位的密碼編譯動作。SIGN_ONLY不過，您仍然可以對資料模型進行其他變更。例如，您可以在單一部署中新增或移除加密欄位。

資料描述

材料描述作為加密記錄的標頭。當您使用 AWS Database Encryption SDK 加密和簽署欄位時，加密程式會在組合加密資料時記錄材料描述，並將材料描述儲存在加密程式新增至記錄的新欄位 (aws_dbe_head) 中。

材料描述是一種可攜式格式化的資料結構，其中包含資料金鑰和其他資訊的加密副本，例如加密演算法、加密內容，以及加密和簽署指示。加密程式會在組合加密材料以進行加密和簽署時記錄材料描述。稍後，當它需要組裝密碼材料來驗證和解密字段時，它會使用材料描述作為其指南。

將加密的資料金鑰與加密欄位一起儲存，可簡化解密作業，讓您無須獨立儲存和管理加密的資料金鑰，而不受其加密資料的影響。

如需材料描述的技術資訊，請參閱材料描述格式。

加密內容

為了改善密碼編譯作業的安全性， AWS Database Encryption SDK 會在所有要求中包含加密內容，以加密和簽署記錄。

加密內容是一組名稱/值對，其中包含任意非私密的額外驗證資料。資 AWS 料庫加密 SDK 在加密內容中包含資料庫的邏輯名稱和主索引鍵值 (例如，DynamoDB 表格中的分區索引鍵和排序金鑰)。當您加密並簽署欄位時，加密內容會以密碼編譯方式繫結至加密記錄，因此必須使用相同的加密內容來解密欄位。

如果您使用 AWS KMS 金鑰環，資料 AWS 庫加密 SDK 也會使用加密內容，在金鑰環進行的呼叫中提供額外的驗證資料 (AAD)。 AWS KMS

每當您使用預設演算法套件時，密碼編譯材料管理員 (CMM) 都會在加密內容中新增名稱-值配對，該內容包含保留名稱`aws-crypto-public-key`，以及代表公開驗證金鑰的值。公開驗證金鑰會儲存在材料描述中。

密碼編譯資料管理員

密碼材料管理員 (CMM) 會組合用來加密、解密和簽署資料的加密材料。每當您使用預設演算法套件時，加密資料都包括純文字和加密的資料金鑰、對稱簽章金鑰和非對稱簽章金鑰。您永遠不會直接與 CMM 互動。加密和解密方法會為您代勞。

由於 CMM 充當 AWS 資料庫加密 SDK 和金鑰環之間的聯絡，因此它是自訂和延伸 (例如對原則強制執行的支援) 的理想選擇。您可以明確指定 CMM，但這不是必需的。當您指定金鑰環時， AWS 資料庫加密 SDK 會為您建立預設的 CMM。預設 CMM 會從您指定的金鑰圈取得加密或解密資料。這可能牽涉到呼叫密碼編譯服務，例如 AWS Key Management Service (AWS KMS)。

對稱與非對稱式加密

對稱加密使用相同的金鑰來加密和解密資料。

非對稱加密使用與數學相關的資料 key pair。配對中的一個金鑰會加密資料；只有配對中的另一個金鑰可以解密資料。如需詳細資訊，請參閱《加密服務和工具指南》中的AWS 加密演算法。

資 AWS 料庫加密 SDK 使用信封加密。它使用對稱數據密鑰對您的數據進行加密。它使用一個或多個對稱或非對稱包裝密鑰對稱數據密鑰進行加密。它將材料描述添加到記錄，其中包括至少一個數據密鑰的加密副本。

加密您的資料 (對稱加密)

> 為了加密您的資料，資料 AWS 庫加密 SDK 會使用對稱資料金鑰和包含對稱加密演算法的演算法套件。為了解密數據，數據 AWS 庫加密 SDK 使用相同的數據密鑰和相同的算法套件。

加密您的資料金鑰 (對稱或非對稱式加密)

> 您提供給加密和解密作業的金鑰圈會決定對稱資料金鑰的加密與解密方式。您可以選擇使用對稱加密的金鑰圈，例如具有對稱加密 KMS 金鑰的金鑰 AWS KMS 圈，或使用非對稱加密的金鑰圈，例如具有非對稱 RSA KMS 金 AWS KMS 鑰的金鑰圈。

主要承諾

資 AWS 料庫加密 SDK 支援金鑰承諾 (有時稱為穩健性)，這是一種安全性屬性，可確保每個加密文字只能解密為單一純文字。若要這麼做，金鑰承諾可確保只有加密記錄的資料金鑰才會被用來解密。資 AWS 料庫加密 SDK 包含所有加密和解密作業的金鑰承諾。

大多數現代對稱密碼（包括 AES）會在單個密鑰下加密純文本，例如數據 AWS 庫加密 SDK 用於加密記錄中標記的每個明文字字段的唯一數據密鑰。ENCRYPT_AND_SIGN使用相同的資料金鑰解密此記錄會傳回與原始資料相同的純文字。使用不同的金鑰解密通常會失敗。雖然困難，但技術上可以在兩個不同的密鑰下解密密文。在極少數情況下，找到一個可以將密文部分解密為不同但仍然可以理解的明文密鑰是可行的。

數據 AWS 庫加密 SDK 始終對一個唯一數據密鑰下的每個屬性進行加密。它可能會在多個包裝密鑰下加密該數據密鑰，但包裝密鑰始終加密相同的數據密鑰。儘管如此，複雜、手動製作的加密記錄實際上可能包含不同的資料金鑰，每個金鑰都由不同的包裝金鑰加密。例如，如果一個使用者解密加密記錄，則會傳回 0x0 (false)，而另一位使用者解密相同的加密記錄則會得到 0x1 (true)。

為了避免這種情況發生，資 AWS 料庫加密 SDK 會在加密和解密時包含金鑰承諾。加密方法會以密碼編譯方式將產生密文的唯一資料金鑰繫結至金鑰承諾產生，這是一種以雜湊為基礎的訊息驗證碼 (HMAC)，透過資料金鑰衍生計算在材料描述上。然後，它將關鍵承諾存儲在材料描述中。當它使用金鑰承諾解密記錄時，資料 AWS 庫加密 SDK 會驗證資料金鑰是否為該加密記錄的唯一金鑰。如果資料金鑰驗證失敗，解密作業就會失敗。

數位簽章

若要確保資料在系統之間傳送時的真實性，您可以將數位簽章套用至記錄。數位簽章永遠是不對稱的。您可以使用私鑰來創建簽名，並將其附加到原始記錄。您的收件者會使用公開金鑰來驗證記錄在您簽署後尚未修改。如果加密資料的使用者和解密資料的使用者不受同等信任，您應該使用數位簽章。

資料 AWS 庫加密 SDK 會使用經過驗證的加密演算法 AES-GCM 來加密您的資料，但是由於 AES-GCM 使用對稱金鑰，因此任何可以解密用來解密密文資料金鑰的人也可以手動建立新的加密密文字，造成潛在的安全性考量。

為了避免這個問題，預設演算法套件會在加密的記錄中加入橢圓曲線數位簽章演算法 (ECDSA) 簽章。預設演算法套件會加密記錄中ENCRYPT_AND_SIGN使用驗證加密演算法 AES-GCM 標記的欄位。然後，它會在記錄中標ENCRYPT_AND_SIGN記、和的欄位上計算以雜湊為基礎的訊息驗證碼 (HMacs) 和非對稱 ECDSA 簽名。SIGN_ONLY SIGN_AND_INCLUDE_IN_ENCRYPTION_CONTEXT解密程序會使用簽章來驗證授權使用者是否已加密記錄。

使用預設演算法套件時， AWS 資料庫加密 SDK 會為每個加密記錄產生一個臨時私密金鑰和公開 key pair。資料 AWS 庫加密 SDK 會將公開金鑰儲存在材料說明中，並捨棄私密金鑰，而且沒有人可以建立另一個使用公開金鑰進行驗證的簽章。由於演算法會將公開金鑰與加密的資料金鑰做為材料描述中的額外驗證資料繫結，因此只能解密記錄的使用者無法更改公開金鑰。

資 AWS 料庫加密 SDK 一律包含 HMAC 驗證。ECDSA 數位簽章預設為啟用，但不是必要的。如果加密資料的使用者和解密資料的使用者同樣受到信任，您可以考慮使用不包含數位簽章的演算法套件來改善您的效能。如需有關選取替代演算法套件的詳細資訊，請參閱選擇演算法套件。

AWS 資料庫加密 SDK 的運作方式

> 我們的用戶端加密程式庫已重新命名為 AWS 資料庫加密 SDK。本開發人員指南仍提供 DynamoDB 加密用戶端的相關資訊。

資料 AWS 庫加密 SDK 提供用戶端加密程式庫，這些程式庫專門設計用來保護您儲存在資料庫中的資料。程式庫包括您可以擴展或以原狀使用的安全實作。如需有關定義和使用自訂元件的詳細資訊，請參閱資料庫實作的 GitHub 存放庫。

本節中的工作流程說明資料 AWS 庫加密 SDK 如何加密、簽署、解密和驗證資料庫中的資料。這些工作流程描述了使用抽象元素和默認功能的基本過程。如需有關資 AWS 料庫加密 SDK 如何與資料庫實作搭配運作的詳細資訊，請參閱資料庫的加密內容主題。

資料 AWS 庫加密 SDK 使用信封加密來保護您的資料。每個記錄都在唯一的數據密鑰下進行加密。資料金鑰可用來衍生加密動作*ENCRYPT_AND_SIGN*中標記的每個欄位的唯一資料加密金鑰。然後，資料金鑰的副本會由您指定的包裝金鑰加密。若要解密加密的記錄，資料 AWS 庫加密 SDK 會使用您指定的包裝金鑰來解密至少一個加密的資料金鑰。然後它可以解密密文並返回一個明文條目。

如需有關資 AWS 料庫加密 SDK 中使用的術語的詳細資訊，請參閱AWS 資料庫加密 SDK 概念。

加密和簽署

在其核心， AWS 數據庫加密 SDK 是一個記錄加密器，用於加密，簽名，驗證和解密數據庫中的記錄。它會取得您的記錄相關資訊，以及要加密和簽署哪些欄位的指示。它從您指定的包裝密鑰配置的加密材料管理器中獲取加密材料以及有關如何使用它們的說明。

以下逐步解說說明資料 AWS 庫加密 SDK 如何加密和簽署您的資料項目。

1. 加密材料管理員為資料 AWS 庫加密 SDK 提供唯一的資料加密金鑰：一個純文字資料金鑰、以指定包裝金鑰加密的資料金鑰副本，以及一個 MAC 金鑰。

> **Note**
>
> 您可以在多個包裝密鑰下加密數據密鑰。每個包裝密鑰都會加密數據密鑰的單獨副本。數據 AWS 庫加密 SDK 將所有加密的數據密鑰存儲在材料描述中。資料 AWS 庫加密 SDK 會將新欄位 (aws_dbe_head) 新增至儲存材料描述的記錄。
> 系統會針對資料金鑰的每個加密複本衍生一個 MAC 金鑰。MAC 鍵不會存儲在材料描述中。而是，解密方法使用包裝密鑰再次導出 MAC 密鑰。

2. 加密方法會加密您指定的加密動作ENCRYPT_AND_SIGN中標記為的每個欄位。
3. 加密方法會commitKey從資料金鑰衍生出來，並使用它來產生金鑰承諾值，然後捨棄資料金鑰。
4. 加密方法會將材料描述新增至記錄。材料描述包含加密的數據密鑰和有關加密記錄的其他信息。若要取得材料描述中所包含資訊的完整清單，請參閱〈材料描述格式〉。
5. 加密方法會使用步驟 1 中傳回的 MAC 金鑰，根據材料描述、加密內容以及每個標記ENCRYPT_AND_SIGN的欄位或SIGN_AND_INCLUDE_IN_ENCRYPTION_CONTEXT加密動作的規範化來計算「雜湊型訊息驗證碼」(HMAC) 值。SIGN_ONLYHMAC 值會儲存在加密方法新增至記錄的新欄位 (aws_dbe_foot) 中。
6. 加密方法會根據材料描述、加密內容的規範化以及標記ENCRYPT_AND_SIGN、或SIGN_AND_INCLUDE_IN_ENCRYPTION_CONTEXT的每個欄位計算 ECD SA 簽章，並將 ECDSA 簽章儲存在欄位中。SIGN_ONLY aws_dbe_foot

> **Note**
>
> ECDSA 簽章預設為啟用，但並非必要。

7. 加密方法將加密和簽名記錄存儲在數據庫中

解密和驗證

1. 密碼材料管理器（CMM）提供了解密方法，其中包括明文數據密鑰和關聯的 MAC 密鑰存儲在材料描述中的解密材料。
   - CMM 會使用指定金鑰環中的包裝金鑰來解密加密的資料金鑰，並傳回純文字資料金鑰。
2. 解密方法比較和驗證材料描述中的關鍵承諾值。
3. 解密方法會驗證簽名欄位中的簽名。

它標識哪些字段被標記 ENCRYPT_AND_SIGNSIGN_ONLY，或SIGN_AND_INCLUDE_IN_ENCRYPTION_CONTEXT從您定義的允許未驗證字段的列表。解密方法會使用步驟 1 中傳回的 MAC 金鑰，重新計算並比較標記為ENCRYPT_AND_SIGNSIGN_ONLY、或的欄位的 HMAC 值。SIGN_AND_INCLUDE_IN_ENCRYPTION_CONTEXT然後，它會使用儲存在加密內容中的公開金鑰來驗證 ECDSA 簽章。

4. 解密方法會使用純文字資料金鑰來解密每個標記的值。ENCRYPT_AND_SIGN然後， AWS 資料庫加密 SDK 會捨棄純文字資料金鑰。
5. 解密方法返回明文記錄。

AWS 資料庫加密 SDK 中支援的演算法套件

我們的用戶端加密程式庫已重新命名為 AWS 資料庫加密 SDK。本開發人員指南仍提供 DynamoDB 加密用戶端的相關資訊。

演算法套件是加密演算法與相關數值的集合。密碼編譯系統使用演算法實作來產生加密文字訊息。

資 AWS 料庫加密 SDK 使用演算法套件來加密和簽署資料庫中的欄位。資 AWS 料庫加密 SDK 支援兩個演算法套件。所有支援的套件都使用進階加密標準 (AES) 做為主要演算法，並將它與其他演算法和值結合。

預設演算法套件

資 AWS 料庫加密 SDK 演算法套件在 Galo/ 計數器模式 (GCM) 中使用進階加密標準 (AES) 演算法來加密原始資料。資 AWS 料庫加密 SDK 支援 256 位元加密金鑰。驗證標籤的長度一律是 16 個位元組。

默認情況下， AWS 數據庫加密 SDK 使用帶有 AES-GCM 的算法套件，具有基於 HMAC 的密 extract-and-expand 鑰派生功能（ HKDF ），密鑰承諾，對稱和非對稱簽名以及 256 位加密密鑰。

數據 AWS 庫加密 SDK 使用算法套件，該算法套件通過向基於 HMAC extract-and-expand 的密鑰派生功能（ HKDF ）提供 256 位數據加密密鑰來導出 AES-GCM 數據密鑰。它也會衍生資料金鑰的 MAC 金鑰。資料 AWS 庫加密 SDK 使用此資料金鑰衍生唯一的資料加密金鑰，以加密每個欄位。然後，資料 AWS 庫加密 SDK 會使用 MAC 金鑰來計算資料金鑰的每個加密副本的雜湊型訊息驗證碼 (HMAC)，並將橢圓曲線數位簽章演算法 (ECDSA) 新增至記錄。此演算法套件也會衍生一個關鍵承諾 — HMAC，可將資料金鑰與記錄相關聯。主要承諾值是根據材料描述和承諾密鑰計算的 HMAC，該密鑰是通過 HKDF 使用類似於導出數據加密密鑰的程序獲得的。然後，關鍵承諾值會儲存在材料描述中。

加密演算法	資料加密金鑰長度 (以位元為單位)	对称签名算法	非对称签名算法	主要承諾
AES-G 厘米	256	哈馬克-沙 -384	ECDSA 超過 384	香港文憑基金與 SHA-512

該算法套件序列化材料描述和標記的所有字段 ENCRYPT_AND_SIGNSIGN_ONLY，並SIGN_AND_INCLUDE_IN_ENCRYPTION_CONTEXT在加密操作中，然後使用 HMAC 與加密哈希函數算法（SHA-512）簽名規範化。然後它計算一個 ECDSA 數字簽名。HMAC 和 ECDSA 簽章會儲存在 AWS 資料庫加密 SDK 新增至記錄的新欄位 (aws_dbe_foot) 中。當授權原則允許一組使用者加密資料，而另一組使用者解密資料時，數位簽章特別有用。

關鍵承諾確保每個密文解密只能解密為一個純文本。他們通過驗證用作加密算法輸入的數據密鑰來實現這一點。加密時，這些演算法套件會衍生金鑰承諾 HMAC。在解密之前，他們會驗證資料金鑰產生相同的金鑰承諾 HMAC。如果沒有，解密呼叫就會失敗。

不含數位簽章的 AES-GCM

雖然預設演算法套件可能適用於大多數應用程式，但您可以選擇替代演算法套件。例如，某些信任模型會由沒有數位簽章的演算法套件來滿足。僅當加密資料的使用者和解密資料的使用者受到同等信任時，才使用此套件。

所有 AWS 資料庫加密 SDK 演算法套件都支援 HMAC-SHA-384 對稱簽章。唯一的區別是，沒有數位簽章的 AES-GCM 演算法套件缺少 ECDSA 簽章，可提供額外的真實性和不可否認性層。

例如，如果您的金鑰圈、、和中有多個包裝金鑰 wrappingKeyA wrappingKeyBwrappingKeyC，而您使用解密記錄wrappingKeyA，則 HMAC-SHA-384 對稱簽章會驗證記錄是否已由具有存取權的使用者加密。wrappingKeyA如果您使用預設演算法，HMAC 會提供相同的驗證wrappingKeyA，並且另外使用 ECDSA 簽章來確保記錄是由具有的加密權限的使用者加密。wrappingKeyA

若要選取不含數位簽章的 AES-GCM 演算法套件，請在加密組態中指定。

搭配使用AWS資料庫加密 SDK AWS KMS

我們的用戶端加密程式庫已重新命名為AWS資料庫加密 SDK。本開發人員指南仍提供 DynamoDB 加密用戶端的相關資訊。

若要使用AWS資料庫加密 SDK，您需要設定金鑰環並指定一或多個包裝金鑰。如果您沒有金鑰基礎架構，建議您使用 AWS Key Management Service (AWS KMS)。

資AWS料庫加密 SDK 支援兩種類型的AWS KMS金鑰環。傳統的金AWS KMS鑰圈用AWS KMS keys來產生、加密和解密資料金鑰。您可以使用對稱加密 (SYMMETRIC_DEFAULT) 或非對稱 RSA KMS 金鑰。由於數據AWS庫加密 SDK 使用唯一的數據密鑰AWS KMS對每個記錄進行加密和簽名，因此必須調用每個加密和解密操作的密AWS KMS鑰環。對於需要最小化呼叫次數的應用程式AWS KMS，AWS資料庫加密 SDK 也支援AWS KMS階層式金鑰圈。階層式金鑰圈是一種加密材料快取解決方案，可透過使用保存在 Amazon DynamoDB 表格中的AWS KMS受保護分支金鑰，然後在本機快取用於加密和解密作業的分支金鑰材料，以減少AWS KMS呼叫次數。我們建議盡可能使用AWS KMS鑰匙圈。

若要與互動AWS KMS，AWS資料庫加密 SDK 需要AWS SDK for Java. AWS KMS

若要準備搭配使用AWS資料庫加密 SDK AWS KMS

1. 建立 AWS 帳戶。要了解如何操作，請參閱如何創建和激活新的亞馬遜網絡服務帳戶？ 在AWS知識中心。
2. 建立對稱加密AWS KMS key。如需說明，請參閱AWS Key Management Service開發人員指南中的建立金鑰。

   > **Tip**
   >
   > 若要以AWS KMS key程式設計方式使用，您將需要. AWS KMS key 如需尋找 ARN 的相關協助AWS KMS key，請參閱AWS Key Management Service開發人員指南中的尋找金鑰 ID 和 ARN。

3. 產生存取金鑰 ID 和安全性存取金鑰。您可以使用 IAM 使用者的存取金鑰 ID 和秘密存取金鑰，也可以使用建立包含存AWS Security Token Service取金鑰 ID、秘密存取金鑰和工作階段 Token 的臨時安全登入資料的新工作階段。為了安全性最佳做法，我們建議您使用臨時登入資料，而不要使用與 IAM 使用者或 AWS (root) 使用者帳戶相關聯的長期登入資料。

若要使用存取金鑰建立 IAM 使用者，請參閱 IAM 使用者指南中的建立 IAM 使用者。

若要產生臨時安全登入資料，請參閱 IAM 使用者指南中的要求臨時安全登入資料。

4. 使用您在步驟 3 中產生的AWS SDK for Java存取金鑰 ID 和秘密存取金鑰中的指示來設定您的AWS認證。如果您產生了臨時認證，則還需要指定會話令牌。

   此程序可允許 AWS 軟體開發套件為您簽署向 AWS 提出的請求。AWS資料庫加密 SDK 中與互動的程式碼範例AWS KMS假設您已完成此步驟。

設定資 AWS 料庫加密 SDK

我們的用戶端加密程式庫已重新命名為 AWS 資料庫加密 SDK。本開發人員指南仍提供 DynamoDB 加密用戶端的相關資訊。

數 AWS 據庫加密 SDK 被設計為易於使用。雖然 AWS 資料庫加密 SDK 有數個設定選項，但是預設值是經過仔細選擇，對於大多數應用程式來說是實用且安全的。不過，您可能需要調整組態以改善效能，或在設計中包含自訂功能。

主題

選擇一種編程語言

適用於 DynamoDB 的 AWS 資料庫加密開發套件提供多種程式設計語言。語言實現被設計為完全可互操作，並提供相同的功能，儘管它們可能以不同的方式實現。一般而言，您會使用與應用程式相容的程式庫。

選擇包裝鍵

資料 AWS 庫加密 SDK 會產生唯一的對稱資料金鑰來加密每個欄位。您不需要設定、管理或使用資料金鑰。數 AWS 據庫加密 SDK 為您完成。

但是，您必須選取一或多個包裝金鑰來加密每個資料金鑰。資 AWS 料庫加密 SDK 支援 AWS Key Management Service(AWS KMS) 對稱式加密 KMS 金鑰和非對稱 RSA KMS 金鑰。它還支持 AES 對稱密鑰和您提供不同大小的 RSA 非對稱密鑰。您必須為包裝金鑰的安全性和耐久性負責，因此建議您在硬體安全性模組或金鑰基礎架構服務 (例如) 中使用加密金鑰 AWS KMS。

若要指定用於加密和解密的包裝金鑰，請使用金鑰環。根據您使用的金鑰圈類型，您可以指定一個包裝金鑰或多個相同或不同類型的包裝金鑰。如果您使用多個包裝金鑰來包裝資料金鑰，每個包裝金鑰都會

加密相同資料金鑰的副本。加密的數據密鑰（每個包裝密鑰一個）存儲在與加密字段一起存儲的材料描述中。若要解密資料，資料 AWS 庫加密 SDK 必須先使用其中一個包裝金鑰來解密加密的資料金鑰。

我們建議盡可能使用其中一個 AWS KMS 鑰匙圈。資 AWS 料庫加密 SDK 提供AWS KMS 金鑰環和AWS KMS 階層式金鑰圈，可減少呼叫的次數。 AWS KMS若要在金鑰環 AWS KMS key 中指定，請使用支援的 AWS KMS 金鑰識別碼。如果您使用 AWS KMS 階層式金鑰圈，則必須指定金鑰 ARN。如需金鑰的金鑰識別碼的詳細資訊，請參閱AWS Key Management Service 開發人員指南中的金鑰識別碼。 AWS KMS

- 使用金 AWS KMS 鑰圈加密時，您可以為對稱加密 KMS 金鑰指定任何有效的金鑰識別碼 (金鑰 ARN、別名名稱、別名 ARN 或金鑰識別碼)。如果您使用非對稱 RSA KMS 金鑰，則必須指定金鑰 ARN。

  如果您在加密時為 KMS 金鑰指定別名或別名 ARN， AWS 資料庫加密 SDK 會儲存目前與該別名關聯的金鑰 ARN，但不會儲存別名。變更別名不會影響用於解密資料金鑰的 KMS 金鑰。
- 根據預設，金 AWS KMS 鑰環會以嚴謹模式 (您指定特定 KMS 金鑰) 解密記錄。您必須使用金鑰 ARN 來識別以進行 AWS KMS keys 解密。

  當您使用金 AWS KMS 鑰圈加密時，資料 AWS 庫加密 SDK 會將材料描述 AWS KMS key 中的金鑰 ARN 與加密的資料金鑰一起儲存。在嚴謹模式下解密時， AWS 資料庫加密 SDK 會在嘗試使用環繞金鑰解密加密的資料金鑰之前，先驗證金鑰圈中是否出現相同的金鑰 ARN。如果您使用不同的金鑰識別碼， AWS 資料庫加密 SDK 將無法辨識或使用 AWS KMS key，即使識別碼參照相同的金鑰。
- 在探索模式下解密時，您不會指定任何包裝金鑰。首先，數 AWS 據庫加密 SDK 嘗試使用存儲在材料描述中的密鑰 ARN 解密記錄。如果沒有作用， AWS 資料庫加密 SDK 會 AWS KMS 要求使用加密記錄的 KMS 金鑰來解密記錄，無論誰擁有或擁有該 KMS 金鑰的存取權。

若要將原始 AES 金鑰或原始 RSA key pair 指定為金鑰環中的環繞金鑰，您必須指定命名空間和名稱。解密時，您必須為每個原始包裝金鑰使用與加密時使用完全相同的命名空間和名稱。如果您使用不同的命名空間或名稱，即使金鑰材料相同，資料 AWS 庫加密 SDK 也不會辨識或使用包裝金鑰。

建立探索篩選器

解密使用 KMS 金鑰加密的資料時，最佳做法是在嚴謹模式下進行解密，也就是說，將使用的包裝金鑰限制為只有您指定的金鑰。不過，如有必要，您也可以在探索模式中解密，而您不需要指定任何包裝金鑰。在此模式中， AWS KMS 無論誰擁有或擁有該 KMS 金鑰的存取權，都可以使用加密的 KMS 金鑰來解密加密的資料金鑰。

如果您必須在探索模式中解密，建議您一律使用探索篩選器，這會限制可用於指定 AWS 帳戶 和分割區中的 KMS 金鑰。探索篩選器是選用的，但這是最佳作法。

使用下表決定探索篩選器的分割區值。

區域	分區
AWS 區域	aws
中國區域	aws-cn
AWS GovCloud (US) Regions	aws-us-gov

下列範例顯示如何建立探索篩選器。在使用代碼之前，請將示例值替換為您的 AWS 帳戶 和分區的有效值。

Java

```
// Create the discovery filter
DiscoveryFilter discoveryFilter = DiscoveryFilter.builder()
        .partition("aws")
        .accountIds(111122223333)
        .build();
```

C# / .NET

```
var discoveryFilter = new DiscoveryFilter
{
    Partition = "aws",
    AccountIds = 111122223333
};
```

使用多租戶資料庫

透過資料 AWS 庫加密 SDK，您可以使用不同的加密材料隔離每個租用戶，藉此為具有共用結構描述的資料庫設定用戶端加密。考量多租戶資料庫時，請花一些時間檢閱您的安全性需求，以及多租戶可能對其造成的影響。例如，使用多租戶資料庫可能會影響您將資料 AWS 庫加密 SDK 與其他伺服器端加密解決方案結合的能力。

如果您有多個使用者在資料庫中執行加密作業，您可以使用其中一個金 AWS KMS 鑰環，為每位使用者提供不同的金鑰，以便在其加密編譯作業中使用。管理多租戶用戶端加密解決方案的資料金鑰可能很複雜。我們建議盡可能由租戶組織您的數據。如果租用戶是由主索引鍵值識別 (例如，Amazon DynamoDB 表格中的分區金鑰)，則管理金鑰會更容易。

您可以使用AWS KMS 金鑰圈，透過不同的金 AWS KMS 鑰圈和隔離每個承租人。 AWS KMS keys根據每個租用戶 AWS KMS 撥打的通話量，您可能想要使用 AWS KMS 階層式金鑰圈來將您的通話降至AWS KMS最低。AWS KMS 階層式金鑰圈是一種加密材料快取解決方案，可透過使用保存在 Amazon DynamoDB 表格中的 AWS KMS 受保護分支金鑰，然後在本機快取用於加密和解密作業的分支金鑰材料，以減少 AWS KMS 呼叫次數。您必須使用 AWS KMS 階層式金鑰環，在資料庫中實作可搜尋的加密。

建立簽署的信標

AWS 資料庫加密 SDK 使用標準信標和複合信標來提供可搜尋的加密解決方案，讓您搜尋加密的記錄，而無需解密查詢的整個資料庫。不過， AWS 資料庫加密 SDK 也支援已簽署的信標，這些信標可以完全從純文字簽署欄位進行設定。簽名信標是一種複合信標，可對和SIGN_AND_INCLUDE_IN_ENCRYPTION_CONTEXT字段進行複雜查詢編制索引SIGN_ONLY和執行。

例如，如果您有多租戶資料庫，您可能想要建立一個已簽署的信標，讓您查詢資料庫中的特定承租人金鑰加密的記錄。如需詳細資訊，請參閱 查詢多租戶資料庫中的信標。

您必須使用 AWS KMS 階層式金鑰圈來建立已簽署的信標。

若要設定已簽署的信標，請提供下列值。

Java

複合信標配置

下列範例會在已簽署的信標 (Beacon) 組態中本機定義已簽署零件清單

```
List<CompoundBeacon> compoundBeaconList = new ArrayList<>();
CompoundBeacon exampleCompoundBeacon = CompoundBeacon.builder()
    .name("compoundBeaconName")
    .split(".")
    .signed(signedPartList)
    .constructors(constructorList)
    .build();
compoundBeaconList.add(exampleCompoundBeacon);
```

信標版本定義

下列範例會在信標 (Beacon) 版本中全域定義已簽署零件清單。如需有關定義信標版本的詳細資訊，請參閱使用信標。

```
 List<BeaconVersion> beaconVersions = new ArrayList<>();
beaconVersions.add(
    BeaconVersion.builder()
        .standardBeacons(standardBeaconList)
        .compoundBeacons(compoundBeaconList)
        .signedParts(signedPartList)
        .version(1) // MUST be 1
        .keyStore(keyStore)
        .keySource(BeaconKeySource.builder()
            .single(SingleKeyStore.builder()
                .keyId(branchKeyId)
                .cacheTTL(6000)
                .build())
            .build())
        .build()
);
```

C# / .NET

請參閱完整的程式碼範例：BeaconConfig.cs

已簽署信標組態

下列範例會在已簽署的信標 (Beacon) 組態中本機定義已簽署零件清單

```
var compoundBeaconList = new List<CompoundBeacon>();
var exampleCompoundBeacon = new CompoundBeacon
 {
    Name = "compoundBeaconName",
    Split = ".",
    Signed = signedPartList,
    Constructors = constructorList
 };
compoundBeaconList.Add(exampleCompoundBeacon);
```

信標版本定義

下列範例會在信標 (Beacon) 版本中全域定義已簽署零件清單。如需有關定義信標版本的詳細資訊，請參閱使用信標。

```
var beaconVersions = new List<BeaconVersion>
{
    new BeaconVersion
    {
        StandardBeacons = standardBeaconList,
        CompoundBeacons = compoundBeaconList,
        SignedParts = signedPartsList,
        Version = 1, // MUST be 1
        KeyStore = keyStore,
        KeySource = new BeaconKeySource
        {
            Single = new SingleKeyStore
            {
                KeyId = branchKeyId,
                CacheTTL = 6000
            }
        }
    }
};
```

您可以在本機或全域定義的清單中定義已簽署的零件。我們建議您盡可能在信標版本的全域清單中定義已簽署的零件。透過全域定義已簽署零件，您可以定義每個零件一次，然後在多個複合信標 (Beacon) 組態中重複使用零件 如果您只打算使用一次已簽署零件，則可以在已簽署的信標 (Beacon) 組態的本機清單中定義該零件。您可以在構造函數列表中引用本地和全局部分。

如果您全域定義已簽署零件清單，則必須提供建構函式零件清單，以識別已簽署信標 (Beacon) 組合信標 (Beacon) 組態中欄位的所有可能方式。

> **Note**
>
> 若要全域定義已簽署零件清單，您必須使用 3.2 版或更新版本的 AWS 資料庫加密 SDK。在全域定義任何新零件之前，請先將新版本部署至所有讀取器。
> 您無法更新現有的信標組態以全域定義已簽署零件清單。

信標名稱

查詢信標時使用的名稱。

已簽署的信標名稱不能與未加密欄位的名稱相同。沒有兩個信標可以具有相同的信標名稱。

分割字元

用來分隔組成已簽署信標之零件的字元。

分割字元不能出現在已簽署信標所建構之任何欄位的純文字值中。

已簽署零件清單

識別已簽署信標中包含的已簽署欄位。

每個零件都必須包含名稱、來源和字首。來源是零件識別的SIGN_ONLY或SIGN_AND_INCLUDE_IN_ENCRYPTION_CONTEXT欄位。來源必須是欄位名稱或參照巢狀欄位值的索引。如果您的零件名稱識別來源，您可以省略來源， AWS 資料庫加密 SDK 會自動使用該名稱作為其來源。建議您盡可能將來源指定為零件名稱。前綴可以是任何字符串，但它必須是唯一的。簽署的信標中沒有兩個已簽署的零件可以具有相同的前綴。我們建議使用短值來區分零件與複合信標服務的其他零件。

我們建議您盡可能在全域定義已簽署的零件。如果您只打算在一個複合信標中使用已簽署零件，則可以考慮在本機定義已簽署零件。本機定義的零件不能具有與全域定義零件相同的字首或名稱。

Java

```
List<SignedPart> signedPartList = new ArrayList<>);
    SignedPart signedPartExample = SignedPart.builder()
        .name("signedFieldName")
        .prefix("S-")
        .build();
    signedPartList.add(signedPartExample);
```

C# / .NET

```
var signedPartsList = new List<SignedPart>
{
    new SignedPart { Name = "signedFieldName1", Prefix = "S-" },
    new SignedPart { Name = "signedFieldName2", Prefix = "SF-" }
};
```

構造函數列表（可選）

識別建構函式，這些建構函式會定義已簽署零件可由簽署的信標組裝的不同方式。

如果您未指定建構函式清單， AWS 資料庫加密 SDK 會使用下列預設建構函式組合已簽署的信標。

- 所有已簽署零件的順序，均按其新增至已簽署零件清單的順序
- 所有零件皆為必填

建構函式

每個構造函數都是構造函數部分的有序列表，它定義了有符號信標可以組裝的一種方式。構造函數部分按照它們被添加到列表中的順序連接在一起，每個部分由指定的拆分字符分隔。

每個構造函數部分命名一個帶符號的部分，並定義構造函數中該部分是必需的還是可選的。例如，如果您想要在、和上查詢已簽署的信標 `Field1`、`Field1.Field2`，請將 and 標記`Field3`為可選`Field1.Field2.Field3`，`Field2`並建立一個建構函式。

每個構造函數必須至少有一個必需的部分。我們建議在每個所需的構造函數中進行第一部分，以便您可以在查詢中使用`BEGINS_WITH`運算符。

如果記錄中存在其所有必需的部分，則構造函數成功。當您撰寫新記錄時，已簽署的信標會使用建構函式清單來判斷是否可以從提供的值組合信標。它嘗試按照構造函數添加到構造函數列表的順序組裝信標，並使用成功的第一個構造函數。如果沒有建構函式成功，信標就不會寫入記錄。

所有讀者和作者都應該指定相同的構造函數順序，以確保他們的查詢結果是正確的。

請使用下列程序來指定您自己的建構函式清單。

1. 為每個已簽署的零件建立建構函式零件，以定義是否需要該部分。

   構造函數部分名稱必須是有符號字段的名稱。

   下面的例子演示瞭如何創建一個帶符號字段的構造部分。

   Java

```
ConstructorPart field1ConstructorPart = ConstructorPart.builder()
        .name("Field1")
        .required(true)
        .build();
```

C# / .NET

```
var field1ConstructorPart = new ConstructorPart { Name = "Field1", Required
 = true };
```

2. 為可以使用在步驟 1 中創建的構造函數部分組裝簽名信標的每種可能方式創建構函數。

   例如，如果你想查詢Field1.Field2.Field3和Field4.Field2.Field3，那麼你必須創建兩個構造函數。 Field1並且都Field4可以是必需的，因為它們是在兩個單獨的構造函數中定義的。

   Java

```
// Create a list for Field1.Field2.Field3 queries
List<ConstructorPart> field123ConstructorPartList = new ArrayList<>();
field123ConstructorPartList.add(field1ConstructorPart);
field123ConstructorPartList.add(field2ConstructorPart);
field123ConstructorPartList.add(field3ConstructorPart);
Constructor field123Constructor = Constructor.builder()
        .parts(field123ConstructorPartList)
        .build();
// Create a list for Field4.Field2.Field1 queries
List<ConstructorPart> field421ConstructorPartList = new ArrayList<>();
field421ConstructorPartList.add(field4ConstructorPart);
field421ConstructorPartList.add(field2ConstructorPart);
field421ConstructorPartList.add(field1ConstructorPart);
Constructor field421Constructor = Constructor.builder()
        .parts(field421ConstructorPartList)
        .build();
```

   C# / .NET

```
// Create a list for Field1.Field2.Field3 queries
 var field123ConstructorPartList = new Constructor
{
    Parts = new List<ConstructorPart> { field1ConstructorPart,
 field2ConstructorPart, field3ConstructorPart }
};
// Create a list for Field4.Field2.Field1 queries
var field421ConstructorPartList = new Constructor
{
```

```
    Parts = new List<ConstructorPart> { field4ConstructorPart,
 field2ConstructorPart, field1ConstructorPart }
};
```

3. 創建一個構造函數列表，其中包含您在步驟 2 中創建的所有構造函數。

   Java

   ```
   List<Constructor> constructorList = new ArrayList<>();
   constructorList.add(field123Constructor)
   constructorList.add(field421Constructor)
   ```

   C# / .NET

   ```
   var constructorList = new List<Constructor>
   {
       field123Constructor,
       field421Constructor
   };
   ```

4. 指定建`constructorList`立已簽署信標的時間。

使用 keyring

我們的用戶端加密程式庫已重新命名為 AWS 資料庫加密 SDK。本開發人員指南仍提供 DynamoDB 加密用戶端的相關資訊。

資 AWS 料庫加密 SDK 使用金鑰環來執行信封加密。Keyring 會產生、加密及解密資料金鑰。金鑰圈會決定保護每個加密記錄的唯一資料金鑰來源，以及加密該資料金鑰的包裝金鑰。您可以在加密時指定 keyring，並在解密時指定相同或不同的 keyring。

您可以個別使用每個 keyring 或是結合 keyring 成為多重 keyring。雖然多數 keyring 可以產生、加密及解密資料金鑰，您可能想要建立僅執行一個特定操作的 keyring，例如只會產生資料金鑰的 keyring，並將該 keyring 與其他 keyring 結合使用。

我們建議您使用可保護包裝金鑰的金鑰圈，並在安全邊界內執行密碼編譯作業，例如使用永不離開 AWS Key Management Service()AWS KMS未加密 AWS KMS keys 的金 AWS KMS 鑰圈。您也可以撰寫使用儲存在硬體安全性模組 (HSM) 中或受其他主要金鑰服務保護的包裝金鑰環。

本主題說明如何使用資 AWS 料庫加密 SDK 的金鑰圈功能，以及如何選擇金鑰圈。

主題

keyring 如何運作

我們的用戶端加密程式庫已重新命名為 AWS 資料庫加密 SDK。本開發人員指南仍提供 DynamoDB 加密用戶端的相關資訊。

當您加密並簽署資料庫中的欄位時，資料 AWS 庫加密 SDK 會詢問金鑰圈是否有加密材料。金鑰圈會傳回純文字資料金鑰、金鑰圈中每個包裝金鑰所加密的資料金鑰副本，以及與資料金鑰相關聯的 MAC 金鑰。資料 AWS 庫加密 SDK 使用純文字金鑰來加密資料，然後儘快從記憶體中移除純文字資料金鑰。然後，數據 AWS 庫加密 SDK 添加一個材料描述，其中包括加密的數據密鑰和其他信息，例如加密和簽名指令。 AWS 數據庫加密 SDK 使用 MAC 密鑰來計算基於散列的消息身份驗證代碼（HMAC），以規範化材料描述和標記為或的所有字段。`ENCRYPT_AND_SIGN` `SIGN_ONLY`

解密資料時，您可以使用用來加密資料的相同金鑰環，或使用不同的金鑰環。若要解密資料，解密金鑰圈必須至少有權存取加密金鑰環中的一個包裝金鑰。

數據 AWS 庫加密 SDK 將加密的數據密鑰從材料描述傳遞到密鑰環，並要求密鑰環解密其中的任何一個。keyring 使用其包裝金鑰來解密其中一個加密的資料金鑰，並傳回純文字資料金鑰。資料 AWS 庫加密 SDK 使用純文字資料金鑰來解密資料。如果 keyring 中沒有任何包裝金鑰可以解密任何加密的資料金鑰，則解密操作會失敗。

您可以使用單一 keyring，也可以將相同類型或不同類型的 keyring 結合成多重 keyring。當您加密資料時，多重金鑰圈會傳回由包含多重金鑰圈的所有金鑰環中的所有包裝金鑰加密的資料金鑰副本，以及與資料金鑰相關聯的 MAC 金鑰。您可以使用包含多重金鑰圈中任何一個包裝金鑰的金鑰圈來解密資料。

選擇鑰匙圈

> 我們的用戶端加密程式庫已重新命名為 AWS 資料庫加密 SDK。本開發人員指南仍提供 DynamoDB 加密用戶端的相關資訊。

您的金鑰圈會決定保護資料金鑰的包裝金鑰，以及最終保護資料的金鑰。使用對您的任務實用的最安全的包裝密鑰。盡可能使用包裝受硬體安全性模組 (HSM) 或金鑰管理基礎結構保護的金鑰，例如 AWS Key Management Service(AWS KMS) 中的 KMS 金鑰或中的加密金鑰。AWS CloudHSM

資 AWS 料庫加密 SDK 提供數個金鑰環和金鑰圈設定，您可以建立自己的自訂金鑰環。您也可以建立包含一或多個相同或不同類型之鑰匙圈的多重金鑰圈。

主題

- AWS KMS 鑰匙圈
- AWS KMS 階層式鑰匙圈
- AWS KMS ECDH 鑰匙圈
- 原始 AES keyring
- 原始 RSA keyring
- 原始 ECDH 鑰匙圈
- 多重 keyring

AWS KMS 鑰匙圈

我們的客戶端加密庫被重命名為 AWS 數據庫加密 SDK。本開發人員指南仍提供 DynamoDB 加密用戶端的相關資訊。

AWS KMS 金鑰圈使用對稱加密或非對稱 RSA AWS KMS keys來產生、加密和解密資料金鑰。 AWS Key Management Service (AWS KMS) 保護您的 KMS 金鑰，並在 FIPS 界限內執行密碼編譯作業。我們建議您盡可能使用 AWS KMS 金鑰圈或具有類似安全性屬性的鑰匙圈。

您也可以在金鑰圈中使用對稱的多區域 KMS 金鑰。 AWS KMS 如需使用多區域的詳細資訊和範例 AWS KMS keys，請參閱使用多地區 AWS KMS keys。如需有關多區域金鑰的詳細資訊，請參閱AWS Key Management Service 開發人員指南中的使用多區域金鑰。

AWS KMS 鑰匙圈可以包含兩種類型的環繞鍵：

- 產生器金鑰：產生純文字資料金鑰並加密。加密資料的金鑰環必須有一個產生器金鑰。
- 其他金鑰：加密產生器金鑰所產生的純文字資料金鑰。 AWS KMS 鑰匙圈可以有零個或多個其他金鑰。

您必須具有生成器密鑰才能加密記錄。當金 AWS KMS 鑰圈只有一個 AWS KMS 金鑰時，就會使用該金鑰來產生和加密資料金鑰。

與所有鑰匙圈一樣， AWS KMS 鑰匙圈可以單獨使用，也可以與其他相同或不同類型的鑰匙圈一起使用。

主題

AWS KMS 金鑰圈所需的權限

數 AWS 據庫加密 SDK 不需要 AWS 帳戶 ，它不依賴於任何 AWS 服務. 但是，若要使用 AWS KMS 金鑰圈，您需要在 AWS 帳戶 金鑰圈 AWS KMS keys 中具有以下最低權限。

- 要使用密 AWS KMS 鑰環進行加密，您需要 kms：對生成器密鑰的GenerateDataKey權限。您需要 KMS: 加密金鑰環中所有其他金鑰的權限。 AWS KMS
- 若要使用金 AWS KMS 鑰圈進行解密，您需要 KMS: 在金鑰環中至少有一個金鑰的 Decryp t 權限。 AWS KMS
- 要使用由密鑰環組成的多密 AWS KMS 鑰環進行加密，您需要 kms：對生成器密鑰環中的生成器密鑰的GenerateDataKey權限。您需要 KMS：對所有其他金鑰環中所有其他金鑰的加密權限。 AWS KMS
- 若要使用非對稱 RSA 金 AWS KMS 鑰圈加密，您不需要 kms: GenerateDataKey 或 KMS: Encrypt，因為您必須在建立金鑰環時指定要用於加密的公開金鑰材料。使用此金鑰圈加密時，不會進行任何 AWS KMS 呼叫。若要使用非對稱 RSA AWS KMS 金鑰圈進行解密，您需要 K MS: 解密權限。

如需有關權限的詳細資訊 AWS KMS keys，請參閱AWS Key Management Service 開發人員指南中的驗證和存取控制。

在 AWS KMS 鑰匙圈 AWS KMS keys 中識別

一個 AWS KMS 鑰匙圈可以包括一個或多 AWS KMS keys個。若要在金 AWS KMS 鑰環 AWS KMS key 中指定，請使用支援的 AWS KMS 金鑰識別碼。您可以用來識別金鑰圈 AWS KMS key 中的金鑰識別碼會隨作業和語言實作而有所不同。有關的密鑰標識符的詳細信息 AWS KMS key，請參閱AWS Key Management Service 開發人員指南中的密鑰標識符。

最佳做法是使用適合您工作的最特定金鑰識別碼。

- 若要使用金 AWS KMS 鑰圈加密，您可以使用金鑰識別碼、金鑰 ARN、別名或別名 ARN 來加密資料。

> **Note**
>
> 如果您在加密金鑰圈中為 KMS 金鑰指定別名或別名 ARN，則加密作業會將目前與別名關聯的金鑰 ARN 儲存在加密資料金鑰的中繼資料中。它不會儲存別名。變更別名不會影響用於解密加密資料金鑰的 KMS 金鑰。

- 若要使用金 AWS KMS 鑰圈進行解密，您必須使用金鑰 ARN 來識別。 AWS KMS keys如需詳細資訊，請參閱 選擇包裝鍵。
- 在用於加密和解密的 keyring 中，您必須使用金鑰 ARN 來識別 AWS KMS keys。

解密時，資料 AWS 庫加密 SDK 會在 AWS KMS 金鑰環中搜尋可以解密其中一個加密資料金鑰的金鑰。 AWS KMS key 具體而言，資料 AWS 庫加密 SDK 會針對材料描述中的每個加密資料金鑰使用下列模式。

- 數據 AWS 庫加密 SDK 從材料描述的元數據中 AWS KMS key 獲取加密數據密鑰的密鑰 ARN。
- 資 AWS 料庫加密 SDK 會在解密金鑰環中搜尋 AWS KMS key 具有相符金鑰 ARN 的。
- 如果在金鑰環中找到 AWS KMS key 具有相符金鑰 ARN 的金鑰，資料 AWS 庫加密 SDK 會 AWS KMS 要求使用 KMS 金鑰來解密加密的資料金鑰。
- 否則會跳到下一個加密的資料金鑰 (如果有)。

建立 AWS KMS 金鑰圈

您可以將每個 AWS KMS 鑰匙圈配置為一個 AWS KMS key 或多 AWS KMS keys 個相同或不同的 AWS 帳戶 和 AWS 區域。 AWS KMS key 必須是對稱加密金鑰 (SYMMETRIC_DEFAULT) 或非對稱 RSA KMS 金鑰。您也可以使用對稱加密多區域 KMS 金鑰。您可以在多重鑰匙圈中使用一或多個 AWS KMS 鑰匙圈。

您可以建立加密和解密資料的 AWS KMS 金鑰環，也可以建立專門用於加密或解密的 AWS KMS 金鑰環。當您建立金 AWS KMS 鑰環來加密資料時，您必須指定產生器金鑰， AWS KMS key 這是用來產生純文字資料金鑰並加密的產生器金鑰。資料金鑰在數學上與 KMS 金鑰無關。然後，如果您選擇，您可以指定其他加密 AWS KMS keys 相同的純文字資料金鑰。若要解密受此金鑰圈保護的加密欄位，您使用的解密金鑰圈必須至少包含金鑰圈中 AWS KMS keys 定義的其中一個，或者 no。 AWS KMS keys (沒有的 AWS KMS 鑰匙圈稱 AWS KMS keys 為AWS KMS 探索鑰匙圈。)

加密金鑰圈或多重金鑰環中的所有包裝金鑰都必須能夠加密資料金鑰。如果有任何包裝金鑰無法加密，則加密方法會失敗。因此，呼叫者必須擁有金鑰環中所有金鑰的必要權限。如果您單獨使用探索金鑰環加密資料，或在多重金鑰環中加密資料，則加密作業會失敗。

下列範例使用此方`CreateAwsKmsMrkMultiKeyring`法建立具有對稱加密 KMS 金 AWS KMS 鑰的金鑰環。此方`CreateAwsKmsMrkMultiKeyring`法會自動建立用 AWS KMS 戶端，並確保金鑰環能夠正確處理單一區域和多區域金鑰。這些範例使用金鑰 ARN 來識別 KMS 金鑰。如需詳細資訊，請參閱在 AWS KMS 鑰匙圈 AWS KMS keys 中識別

Java

```
final MaterialProviders matProv = MaterialProviders.builder()
        .MaterialProvidersConfig(MaterialProvidersConfig.builder().build())
        .build();
final CreateAwsKmsMrkMultiKeyringInput keyringInput =
 CreateAwsKmsMrkMultiKeyringInput.builder()
        .generator(kmsKeyArn)
        .build();
final IKeyring kmsKeyring = matProv.CreateAwsKmsMrkMultiKeyring(keyringInput);
```

C# / .NET

```
var matProv = new MaterialProviders(new MaterialProvidersConfig());
var createAwsKmsMrkMultiKeyringInput = new CreateAwsKmsMrkMultiKeyringInput
{
    Generator = kmsKeyArn
};
var awsKmsMrkMultiKeyring =
 matProv.CreateAwsKmsMrkMultiKeyring(createAwsKmsMrkMultiKeyringInput);
```

下列範例使用此方`CreateAwsKmsRsaKeyring`法建立具有非對稱 RSA KMS 金 AWS KMS 鑰的金鑰圈。若要建立非對稱的 RSA AWS KMS 金鑰圈，請提供下列值。

- `kmsClient`：建立新 AWS KMS 用戶端
- `kmsKeyID`：識別您非對稱 RSA KMS 金鑰的金鑰 ARN
- `publicKey`: 一個 UTF-8 編碼 ByteBuffer 的 PEM 檔案，代表您傳遞給的金鑰的公開金鑰 `kmsKeyID`
- `encryptionAlgorithm`：加密演算法必須是`RSAES_OAEP_SHA_256`或 `RSAES_OAEP_SHA_1`

Java

```
 final MaterialProviders matProv = MaterialProviders.builder()
    .MaterialProvidersConfig(MaterialProvidersConfig.builder().build())
    .build();
final CreateAwsKmsRsaKeyringInput createAwsKmsRsaKeyringInput =
    CreateAwsKmsRsaKeyringInput.builder()
        .kmsClient(KmsClient.create())
        .kmsKeyId(rsaKMSKeyArn)
```

```
        .publicKey(publicKey)
        .encryptionAlgorithm(EncryptionAlgorithmSpec.RSAES_OAEP_SHA_256)
        .build();
IKeyring awsKmsRsaKeyring =
 matProv.CreateAwsKmsRsaKeyring(createAwsKmsRsaKeyringInput);
```

C# / .NET

```
var matProv = new MaterialProviders(new MaterialProvidersConfig());
var createAwsKmsRsaKeyringInput = new CreateAwsKmsRsaKeyringInput
{
    KmsClient = new AmazonKeyManagementServiceClient(),
    KmsKeyId = rsaKMSKeyArn,
    PublicKey = publicKey,
    EncryptionAlgorithm = EncryptionAlgorithmSpec.RSAES_OAEP_SHA_256
};
IKeyring awsKmsRsaKeyring =
 matProv.CreateAwsKmsRsaKeyring(createAwsKmsRsaKeyringInput);
```

使用多地區 AWS KMS keys

您可以在 AWS 資料庫加密 SDK 中使用多區域 AWS KMS keys 作為包裝金鑰。如果您使用一個多區域金鑰加密 AWS 區域，則可以使用不同的相關多區域金鑰來解密。 AWS 區域

多區域 KMS 金鑰是一組具有 AWS 區域 相同金鑰材料和金鑰 ID 的 AWS KMS keys 不同金鑰。您可以使用這些相關鍵字，就好像它們在不同區域中是相同的索引鍵一樣。多區域金鑰支援常見的災難復原和備份案例，這些案例需要在一個區域中進行加密，並在不同區域進行解密，而不需要跨區域呼叫。AWS KMS如需有關多區域金鑰的詳細資訊，請參閱AWS Key Management Service 開發人員指南中的使用多區域金鑰。

為了支援多區域金鑰， AWS 資料庫加密 SDK 包含 AWS KMS 多區域感知金鑰環。該`CreateAwsKmsMrkMultiKeyring`方法同時支持單區域和多區域鍵。

- 對於單一區域金鑰，多區域感知符號的行為與單一區域金鑰圈一樣。 AWS KMS 它只會嘗試使用加密資料的單一區域金鑰來解密密文。為了簡化金 AWS KMS 鑰圈體驗，我們建議您在使用對稱加密 KMS 金鑰時使用此`CreateAwsKmsMrkMultiKeyring`方法。
- 對於多區域金鑰，多區域感知符號會嘗試使用加密資料的相同多區域金鑰，或使用您指定之區域中的相關多區域金鑰來解密密文。

在採用多個 KMS 金鑰的多重區域感知金鑰環中，您可以指定多個單一區域和多區域金鑰。不過，您只能從每組相關的多區域金鑰中指定一個金鑰。如果您使用相同的金鑰 ID 指定多個金鑰識別碼，建構函式呼叫會失敗。

下列範例會建立具有多區域 KMS 金 AWS KMS 鑰的金鑰圈。範例指定多區域鍵做為產生器鍵，並指定單一區域鍵做為子關鍵字。

Java

```
final MaterialProviders matProv = MaterialProviders.builder()
        .MaterialProvidersConfig(MaterialProvidersConfig.builder().build())
        .build();
final CreateAwsKmsMrkMultiKeyringInput createAwsKmsMrkMultiKeyringInput =
    CreateAwsKmsMrkMultiKeyringInput.builder()
            .generator(multiRegionKeyArn)
            .kmsKeyIds(Collections.singletonList(kmsKeyArn))
            .build();
IKeyring awsKmsMrkMultiKeyring =
 matProv.CreateAwsKmsMrkMultiKeyring(createAwsKmsMrkMultiKeyringInput);
```

C# / .NET

```
var matProv = new MaterialProviders(new MaterialProvidersConfig());
var createAwsKmsMrkMultiKeyringInput = new CreateAwsKmsMrkMultiKeyringInput
{
    Generator = multiRegionKeyArn,
    KmsKeyIds = new List<String> { kmsKeyArn }
};
var awsKmsMrkMultiKeyring =
 matProv.CreateAwsKmsMrkMultiKeyring(createAwsKmsMrkMultiKeyringInput);
```

使用多區域 AWS KMS 金鑰圈時，您可以在嚴謹模式或探索模式下解密密文。若要在嚴謹模式中解密密文，請在您要解密密文的區域中，使用相關多區域金鑰的金鑰 ARN 來實體化多區域感知符號。如果您在不同的區域中指定相關多區域金鑰的金鑰 ARN (例如，記錄已加密的區域)，則多區域感知符號會針對此進行跨區域呼叫。 AWS KMS key

在嚴謹模式下解密時，多區域感知符號需要金鑰 ARN。它只接受來自每組相關多區域鍵的一個關鍵 ARN。

您也可以使用 AWS KMS 多區域金鑰在探索模式下解密。在探索模式下解密時，您不會指定任何 AWS KMS keys。(如需有關單一區域 AWS KMS 探索金鑰圈的資訊，請參閱使用 AWS KMS 探索鑰匙圈。)

如果您使用多區域金鑰加密，則探索模式下的多區域感知符號會嘗試使用本機區域中的相關多區域金鑰來解密。如果不存在，則呼叫失敗。在探索模式下， AWS 資料庫加密 SDK 不會嘗試跨區域呼叫用於加密的多區域金鑰。

使用 AWS KMS 探索鑰匙圈

解密時，最佳做法是指定 AWS 資料庫加密 SDK 可以使用的包裝金鑰。若要遵循此最佳作法，請使用 AWS KMS 解密金鑰圈，將 AWS KMS 包裝金鑰限制為您指定的金鑰。不過，您也可以建立AWS KMS 探索金鑰圈，也就是不會指定任何 AWS KMS 環繞金鑰的金鑰圈。

資 AWS 料庫加密 SDK 為 AWS KMS 多區域金鑰提供標準 AWS KMS 探索金鑰環和探索金鑰圈。如需將多區域金鑰搭配資 AWS 料庫加密 SDK 使用的詳細資訊，請參閱使用多地區 AWS KMS keys。

因為它不會指定任何包裝金鑰，因此探索金鑰環無法加密資料。如果您單獨使用探索金鑰環加密資料，或在多重金鑰環中加密資料，則加密作業會失敗。

解密時，探索金鑰圈允許 AWS Database Encryption SDK AWS KMS 要求使用 AWS KMS key 該加密資料金鑰來解密任何加密的資料金鑰，而不論誰擁有或具有存取權。 AWS KMS key只有當呼叫`kms:Decrypt`者具有. AWS KMS key

> **Important**
>
> 如果您在解密多重金鑰環中包含 AWS KMS 探索金鑰圈，探索金鑰圈會覆寫多重金鑰圈中其他金鑰環指定的所有 KMS 金鑰限制。多重金鑰環的行為與其限制性最小的鑰匙圈一樣。如果您單獨使用探索金鑰圈加密資料，或在多重金鑰環中加密資料，則加密作業會失敗

為了方便起見， AWS 資料庫加密 SDK 提供 AWS KMS 探索金鑰環。不過，基於下列原因，建議您在可能時使用較具限制的 keyring。

- 真實性 — AWS KMS 探索金鑰圈可以使用任何 AWS KMS key 用來加密材料描述中的資料金鑰，只要呼叫者有權使用該金鑰 AWS KMS key 來解密該金鑰的權限。這可能不是呼 AWS KMS key 叫者打算使用的。例如，其中一個加密的資料金鑰可能以較不安全的方式加密 AWS KMS key ，任何人都可以使用。
- 延遲和效能 — AWS KMS 探索金鑰圈的速度可能會比其他金鑰環慢，因為 AWS Database Encryption SDK 會嘗試解密所有加密的資料金鑰，包括 AWS KMS keys 在其他 AWS 帳戶 和區域中加密的資料金鑰，而且 AWS KMS keys 呼叫者沒有用於解密的權限。

如果您使用探索金鑰圈，建議您使用探索篩選器來限制可用於指定 AWS 帳戶 和磁碟分割中的 KMS 金鑰。如需尋找帳戶 ID 和分割區的 AWS 帳戶 ARN 助，請參閱. AWS 一般參考

下列程式碼範例會使用 AWS KMS 探索篩選器來實體化探索金鑰環，該篩選器會限制 AWS Database Encryption SDK 可用於aws分割區和範111122223333例帳戶中的 KMS 金鑰。

在使用此代碼之前，請將示例 AWS 帳戶 和分區值替換為您的 AWS 帳戶 和分區的有效值。如果您的 KMS 金鑰位於中國區域，請使用aws-cn分割區值。如果您的 KMS 金鑰位於中 AWS GovCloud (US) Regions，請使用aws-us-gov分割區值。對於所有其他 AWS 區域，請使用aws分區值。

Java

```
// Create discovery filter
DiscoveryFilter discoveryFilter = DiscoveryFilter.builder()
        .partition("aws")
        .accountIds(111122223333)
        .build();
// Create the discovery keyring
CreateAwsKmsMrkDiscoveryMultiKeyringInput createAwsKmsMrkDiscoveryMultiKeyringInput
 = CreateAwsKmsMrkDiscoveryMultiKeyringInput.builder()
        .discoveryFilter(discoveryFilter)
        .build();
IKeyring decryptKeyring =
 matProv.CreateAwsKmsMrkDiscoveryMultiKeyring(createAwsKmsMrkDiscoveryMultiKeyringInput);
```

C# / .NET

```
// Create discovery filter
var discoveryFilter = new DiscoveryFilter
{
    Partition = "aws",
    AccountIds = 111122223333
};
// Create the discovery keyring
var createAwsKmsMrkDiscoveryMultiKeyringInput = new
 CreateAwsKmsMrkDiscoveryMultiKeyringInput
{
    DiscoveryFilter = discoveryFilter
};
var decryptKeyring =
 matProv.CreateAwsKmsMrkDiscoveryMultiKeyring(createAwsKmsMrkDiscoveryMultiKeyringInput);
```

使用 AWS KMS 區域探索金鑰圈

AWS KMS 區域探索金鑰圈是不會指定 KMS 金鑰 ARN 的金鑰圈。相反地，它允許 AWS 資料庫加密 SDK 僅使用 KMS 金鑰進行解密 AWS 區域。

使用 AWS KMS 區域探索金鑰圈進行解密時，資料 AWS 庫加密 SDK 會將 AWS KMS key 在指定的中加密的任何加密資料金鑰解密。 AWS 區域若要成功，呼叫者必須`kms:Decrypt`擁有至少一個加密資料金鑰 AWS 區域 的指定 AWS KMS keys 中的權限。

與其他探索金鑰環一樣，區域探索金鑰圈對加密沒有影響。它僅在解密加密字段時才起作用。如果您在用於加密和解密的多重金鑰環中使用區域探索金鑰圈，則只有在解密時才有效。如果您單獨或在多重金鑰環中使用多區域探索金鑰環來加密資料，則加密作業會失敗。

> **Important**
>
> 如果您在解密多重金鑰圈中包含 AWS KMS 區域探索金鑰圈，則區域探索金鑰圈會覆寫多重金鑰圈中其他金鑰圈所指定的所有 KMS 金鑰限制。多重金鑰環的行為與其限制性最小的鑰匙圈一樣。 AWS KMS 探索金鑰圈在單獨使用或多重金鑰環中使用時，對加密沒有影響。

AWS 資料庫加密 SDK 中的地區探索金鑰圈只會嘗試使用指定區域中的 KMS 金鑰進行解密。當您使用探索金鑰圈時，您可以在用 AWS KMS 戶端上設定 [區域]。這些 AWS 資料庫加密 SDK 實作不會依區域篩選 KMS 金鑰，但 AWS KMS 會失敗指定區域以外的 KMS 金鑰的解密要求。

如果您使用探索金鑰圈，建議您使用探索篩選器，將用於解密的 KMS 金鑰限制為指定 AWS 帳戶 和分割區中的 KMS 金鑰。

例如，下列程式碼會使用探索篩選器建立 AWS KMS 區域探索金鑰圈。此金鑰圈會將 AWS 資料庫加密 SDK 限制在美國西部 (奧勒岡) 區域 (美國西部 -2) 帳戶 111122223333 中的 KMS 金鑰。

Java

```
// Create the discovery filter
DiscoveryFilter discoveryFilter = DiscoveryFilter.builder()
        .partition("aws")
        .accountIds(111122223333)
        .build();
// Create the discovery keyring
CreateAwsKmsMrkDiscoveryMultiKeyringInput createAwsKmsMrkDiscoveryMultiKeyringInput
 = CreateAwsKmsMrkDiscoveryMultiKeyringInput.builder()
```

```
        .discoveryFilter(discoveryFilter)
        .regions("us-west-2")
        .build();
IKeyring decryptKeyring =
 matProv.CreateAwsKmsMrkDiscoveryMultiKeyring(createAwsKmsMrkDiscoveryMultiKeyringInput);
```

C# / .NET

```
// Create discovery filter
var discoveryFilter = new DiscoveryFilter
{
    Partition = "aws",
    AccountIds = 111122223333
};
// Create the discovery keyring
var createAwsKmsMrkDiscoveryMultiKeyringInput = new
 CreateAwsKmsMrkDiscoveryMultiKeyringInput
{
    DiscoveryFilter = discoveryFilter,
    Regions = us-west-2
};
var decryptKeyring =
 matProv.CreateAwsKmsMrkDiscoveryMultiKeyring(createAwsKmsMrkDiscoveryMultiKeyringInput);
```

AWS KMS 階層式鑰匙圈

我們的用戶端加密程式庫已重新命名為 AWS 資料庫加密 SDK。本開發人員指南仍提供 DynamoDB 加密用戶端的相關資訊。

> **Note**
>
> 自 2023 年 7 月 24 日起，不支援在開發人員預覽期間建立的分支金鑰。建立新的分支金鑰，以繼續使用您在開發人員預覽期間建立的分支金鑰存放區。

使用 AWS KMS 階層式金鑰圈，您可以使用對稱式加密 KMS 金鑰保護加密資料，而無需在 AWS KMS 每次加密或解密記錄時呼叫。對於需要盡量減少呼叫的應用程式，以及可重複使用某些加密資料而不違反其安全性需求的應用程式來說 AWS KMS，這是一個不錯的選擇。

階層式金鑰圈是一種加密材料快取解決方案，可透過使用保存在 Amazon DynamoDB 表格中的 AWS KMS 受保護分支金鑰，然後在本機快取用於加密和解密作業的分支金鑰材料，以減少 AWS KMS 呼叫次數。DynamoDB 表可做為管理和保護分支金鑰的分支金鑰存放區。它存儲活動分支密鑰和分支密鑰的所有以前版本。活動分支密鑰是最新的分支密鑰版本。分層密鑰環使用唯一的數據密鑰來加密每個字段，並使用從活動分支密鑰派生的唯一包裝密鑰對每個數據密鑰進行加密。分層密鑰環取決於活動分支密鑰及其派生包裝密鑰之間建立的層次結構。

分層密鑰環通常使用每個分支密鑰版本來滿足多個請求。但是您可以控制重複使用活動分支密鑰的程度，並確定活動分支鍵的旋轉頻率。分支索引鍵的作用中版本會保持作用中狀態，直到您旋轉它為止。以前版本的 Active 分支密鑰將不會用於執行加密操作，但仍然可以在解密操作中查詢和使用它們。

當您實例化分層密鑰環時，它會創建一個本地緩存。您可以指定一個快取限制，以定義分支索引鍵材料在到期並從快取中逐出之前儲存在本機快取中的時間上限。在作業中第一次指定 a `branch-key-id` 時，階層式金鑰圈會進行一次 AWS KMS 呼叫，以解密分支金鑰並組裝分支金鑰材料。然後，分支金鑰材料會儲存在本機快取中，並重複用於所有指定的加密和解密作業，`branch-key-id`直到快取限制到期為止。在本機快取中儲存分支金鑰材料可減少 AWS KMS 呼叫。例如，假設快取限制為 15 分鐘。如果您在該快取限制內執行 10,000 個加密作業，則傳統 AWS KMS 金鑰圈需要進行 10,000 次 AWS KMS 呼叫，才能滿足 10,000 個加密作業。如果您有一個作用中`branch-key-id`，階層式金鑰圈只需要進行一次 AWS KMS 呼叫即可滿足 10,000 個加密作業。

本機快取由兩個分割區組成，一個用於加密作業，另一個用於解密作業。加密分割區會儲存從使用中分支金鑰組合而來的分支金鑰材料，並在所有加密作業中重複使用，直到快取限制到期為止。解密分割區會儲存為解密作業中識別的其他分支金鑰版本所組合的分支金鑰材料。解密分區可以一次存儲多個活動分支密鑰材料版本。當它設定為針對多租戶資料庫使用分支金鑰 ID 供應商時，該加密分割區也可以一次儲存多個分支金鑰材料版本。如需詳細資訊，請參閱 搭配多租戶資料庫使用階層式金鑰圈。

> **Note**
>
> AWS 數據庫加密 SDK 中分層密鑰環的所有提及都是指 AWS KMS 分層密鑰環。

主題

- 使用階層式金鑰圈進行可搜尋的加密

運作方式

下列逐步解說說明階層式金鑰環如何組合加密和解密資料，以及金鑰圈針對加密和解密作業所進行的不同呼叫。如需有關包裝金鑰衍生和純文字資料金鑰加密程序的技術詳細資訊，請參閱AWS KMS 階層式金鑰圈技術詳細資料。

加密和簽署

以下逐步解說說明「階層式金鑰圈」如何組合加密材料並衍生唯一的環繞金鑰。

1. 加密方法會詢問加密材料的階層式金鑰圈。金鑰圈會產生純文字資料金鑰，然後檢查本機快取中是否有有效的分支材料來產生環繞金鑰。如果有有效的分支金鑰材料，金鑰圈會進入步驟 5。
2. 如果沒有有效的分支金鑰材料，階層式金鑰圈會查詢分支金鑰存放區以取得使用中的分支金鑰。
   a. 分支密鑰存儲調用 AWS KMS 以解密活動分支密鑰並返回明文活動分支密鑰。識別活動分支密鑰的數據被序列化，以在解密調用中提供額外的身份驗證數據（AAD）。 AWS KMS
   b. 分支密鑰存儲庫返回明文分支密鑰和標識它的數據，例如分支密鑰版本。
3. 階層式金鑰圈會組合分支金鑰材料 (純文字分支金鑰和分支金鑰版本)，並將它們的副本儲存在本機快取中。
4. 分層密鑰環從明文分支密鑰和 16 字節隨機鹽派生一個唯一的包裝密鑰。它使用派生的包裝密鑰來加密純文本數據密鑰的副本。

加密方法使用加密材料對記錄進行加密和簽名。如需有關如何在資 AWS 料庫加密 SDK 中加密和簽署記錄的詳細資訊，請參閱加密和簽署。

解密和驗證

以下逐步解說說明「階層式金鑰圈」如何組合解密材料並解密加密的資料金鑰。

1. 解密方法可從加密記錄的材料描述欄位中識別加密的資料金鑰，並將其傳遞至階層式金鑰圈。
2. 階層式金鑰環還原序列化識別加密資料金鑰的資料，包括分支金鑰版本、16 位元組 salt，以及說明資料金鑰加密方式的其他資訊。

   如需詳細資訊，請參閱 AWS KMS分層鑰匙圈技術細節。
3. 階層式金鑰圈會檢查本機快取中是否有符合步驟 2 中所識別的分支金鑰版本的有效分支金鑰材料。如果有有效的分支金鑰材料，金鑰圈會進入步驟 6。

4. 如果沒有有效的分支金鑰材料，階層式金鑰圈會查詢分支金鑰存放區，找出與步驟 2 中所識別的分支金鑰版本相符的分支金鑰存放區。

   a. 分支密鑰存儲調用 AWS KMS 解密分支密鑰並返回明文活動分支密鑰。識別活動分支密鑰的數據被序列化，以在解密調用中提供額外的身份驗證數據（AAD）。 AWS KMS

   b. 分支密鑰存儲庫返回明文分支密鑰和標識它的數據，例如分支密鑰版本。

5. 階層式金鑰圈會組合分支金鑰材料 (純文字分支金鑰和分支金鑰版本)，並將它們的副本儲存在本機快取中。
6. 階層式金鑰圈使用組裝好的分支金鑰材料和步驟 2 中識別的 16 位元組鹽來重現加密資料金鑰的唯一包裝金鑰。
7. 階層式金鑰圈使用複製的包裝金鑰來解密資料金鑰，並傳回純文字資料金鑰。

解密方法使用解密材料和明文數據密鑰來解密和驗證記錄。如需有關如何在資 AWS 料庫加密 SDK 中解密和驗證記錄的詳細資訊，請參閱解密和驗證。

必要條件

數 AWS 據庫加密 SDK 不需要 AWS 帳戶 ，它不依賴於任何 AWS 服務. 不過，階層式金鑰圈取決於 AWS KMS 和 Amazon DynamoDB。

若要使用階層式金鑰圈，您需要使用 K MS: 解密權限的對稱 AWS KMS key 加密。您也可以使用對稱加密多區域金鑰。如需有關權限的詳細資訊 AWS KMS keys，請參閱AWS Key Management Service 開發人員指南中的驗證和存取控制。

在建立和使用階層式金鑰圈之前，您必須先建立分支金鑰存放區，並使用第一個使用中的分支金鑰來填入該分支金鑰存放區。

步驟 1：設定新的金鑰存放區服務

金鑰存放區服務提供數種作業，例如`CreateKeyStore`和`CreateKey`，可協助您組合階層式金鑰圈必要條件並管理分支金鑰存放區。

下列範例會建立金鑰存放區服務。您必須指定 DynamoDB 資料表名稱做為分支金鑰存放區的名稱、分支金鑰存放區的邏輯名稱，以及識別可保護分支金鑰之 KMS 金鑰的 KMS 金鑰 ARN。

邏輯金鑰存放區名稱會以密碼方式繫結至儲存在資料表中的所有資料，以簡化 DynamoDB 還原作業。邏輯金鑰存放區名稱可以與 DynamoDB 表名稱相同，但不一定要這樣做。強烈建議您在第一次設定金鑰存放區服務時，將 DynamoDB 表名稱指定為邏輯資料表名稱。您必須永遠指定相同的邏輯資料表名稱。如果您的分支金鑰存放區名稱在從備份還原 DynamoDB 表後變更，邏輯金鑰存

放區名稱會對應至您指定的 DynamoDB 表名稱，以確保階層式金鑰圈仍可存取您的分支金鑰存放區。

Java

```
final KeyStore keystore = KeyStore.builder().KeyStoreConfig(
                KeyStoreConfig.builder()
                        .ddbClient(DynamoDbClient.create())
                        .ddbTableName(keyStoreName)
                        .logicalKeyStoreName(logicalKeyStoreName)
                        .kmsClient(KmsClient.create())
                        .kmsConfiguration(KMSConfiguration.builder()
                                .kmsKeyArn(kmsKeyArn)
                                .build())
                        .build()).build();
```

C# / .NET

```
var kmsConfig = new KMSConfiguration { KmsKeyArn = kmsKeyArn };
var keystoreConfig = new KeyStoreConfig
{
    KmsClient = new AmazonKeyManagementServiceClient(),
    KmsConfiguration = kmsConfig,
    DdbTableName = keyStoreName,
    DdbClient = new AmazonDynamoDBClient(),
    LogicalKeyStoreName = logicalKeyStoreName
};
var keystore = new KeyStore(keystoreConfig);
```

步驟 2：**CreateKeyStore**打電話創建分支密鑰存儲

以下操作會創建將持續存儲並保護您的分支密鑰的分支密鑰存儲區。

Java

```
keystore.CreateKeyStore(CreateKeyStoreInput.builder().build());
```

C# / .NET

```
var createKeyStoreOutput = keystore.CreateKeyStore(new CreateKeyStoreInput());
```

此CreateKeyStore作業會使用您在步驟 1 中指定的資料表名稱和下列必要值建立 DynamoDB 表格。

	分割區索引鍵	排序索引鍵
基本資料表	branch-key-id	type

> **Note**
>
> 您可以手動建立做為分支金鑰存放區的 DynamoDB 表格，而非使用此作業。CreateKeyStore如果您選擇手動建立分支金鑰存放區，則必須為分割區和排序索引鍵指定下列字串值：
>
> - 分區索引鍵: branch-key-id
> - 排序索引鍵：type

步驟 3：打電話創CreateKey建一個新的活動分支密鑰

下列作業會使用您在步驟 1 中指定的 KMS 金鑰建立新的作用中分支金鑰，並將作用中分支金鑰新增至您在步驟 2 中建立的 DynamoDB 表格。

撥打電話時CreateKey，您可以選擇指定下列選用值。

- branchKeyIdentifier：定義一個自定義branch-key-id。

  若要建立自訂branch-key-id，您還必須在encryptionContext參數中包含其他加密內容。

- encryptionContext: 定義一組選用的非秘密金鑰-值配對，在 kms: 呼叫中包含的加密內容中提供額外的驗證資料 (AAD)。GenerateDataKeyWithoutPlaintext

  此額外的加密內容會以aws-crypto-ec:前置詞顯示。

Java

```
final Map<String, String> additionalEncryptionContext =
 Collections.singletonMap("Additional Encryption Context for",
        "custom branch key id");

final String BranchKey = keystore.CreateKey(
        CreateKeyInput.builder()
                .branchKeyIdentifier(custom-branch-key-id) //OPTIONAL
                .encryptionContext(additionalEncryptionContext) //OPTIONAL
                .build()).branchKeyIdentifier();
```

C# / .NET

```
var additionalEncryptionContext = new Dictionary<string, string>();
additionalEncryptionContext.Add("Additional Encryption Context for", "custom
 branch key id");

var branchKeyId = keystore.CreateKey(new CreateKeyInput
{
    BranchKeyIdentifier = "custom-branch-key-id", // OPTIONAL
    EncryptionContext = additionalEncryptionContext // OPTIONAL
});
```

首先，作CreateKey業會產生下列值。

- 版本 4 的通用唯一識別碼 (UUID) branch-key-id (除非您指定了自訂)。branch-key-id
- 分支密鑰版本的版本 4 UUID
- 以 ISO 8601 日期和時間格式表示的 Atimestamp，採用國際標準時間 (UTC) 表示。

然後，作CreateKey業會呼叫 kms：GenerateDataKeyWithoutPlaintext使用下列要求。

```
{
   "EncryptionContext": {
      "branch-key-id" : "branch-key-id",
      "type" : "type",
      "create-time" : "timestamp",
      "logical-key-store-name" : "the logical table name for your branch key store",
      "kms-arn" : the KMS key ARN,
      "hierarchy-version" : "1",
      "aws-crypto-ec:contextKey": "contextValue"
   },
   "KeyId": "the KMS key ARN you specified in Step 1",
   "NumberOfBytes": "32"
}
```

> **Note**
>
> 即使您尚未將資料庫設定為可搜尋的加密，CreateKey作業也會建立作用中的分支金鑰和信標金鑰。兩個密鑰都存儲在您的分支密鑰存儲中。如需詳細資訊，請參閱使用階層式金鑰圈進行可搜尋的加密。

接下來，CreateKey作業會呼叫 kms: ReEncrypt 透過更新加密內容來建立分支金鑰的使用中記錄。

最後，CreateKey操作調用 ddb: TransactWriteItems 編寫一個新項目，該項目將在步驟 2 中創建的表中保留分支密鑰。此項目具有下列屬性。

```
{
    "branch-key-id" : branch-key-id,
    "type" : "branch:ACTIVE",
    "enc" : the branch key returned by the GenerateDataKeyWithoutPlaintext call,
    "version": "branch:version:the branch key version UUID",
    "create-time" : "timestamp",
    "kms-arn" : "the KMS key ARN you specified in Step 1",
    "hierarchy-version" : "1",
    "aws-crypto-ec:contextKey": "contextValue"
}
```

建立階層式金鑰圈

若要初始化階層式金鑰圈，您必須提供下列值：

- 分支金鑰存放區名稱

  您建立做為分支金鑰存放區使用的 DynamoDB 表的名稱。

-

  快取限制存留時間 (TTL)

  本機快取中的分支金鑰材料項目在到期前可以使用的時間 (以秒為單位)。快取限制 TTL 指定用戶端呼叫 AWS KMS 授權使用分支金鑰的頻率。該值必須大於零。當快取限制 TTL 到期時，會從本機快取中逐出項目。

- 分支密鑰標識符

  識別分支金鑰存放區中作用中分支金鑰的。branch-key-id

> **Note**
>
> 若要初始化多租戶使用的階層式金鑰環，您必須指定分支金鑰 ID 供應商，而不是指定. branch-key-id 如需詳細資訊，請參閱 搭配多租戶資料庫使用階層式金鑰圈。

- （可選）授予令牌列表

  如果您使用授權控制階層式金鑰圈中 KMS 金鑰的存取權，則必須在初始化金鑰環時提供所有必要的授權權杖。

下列範例示範如何使用 DynamoDB 用戶端的 AWS 資料庫加密 SDK 初始化階層式金鑰圈。下列範例會指定 600 秒的快取限制 TTL。

Java

```
final MaterialProviders matProv = MaterialProviders.builder()
        .MaterialProvidersConfig(MaterialProvidersConfig.builder().build())
        .build();
final CreateAwsKmsHierarchicalKeyringInput keyringInput =
 CreateAwsKmsHierarchicalKeyringInput.builder()
        .keyStore(branchKeyStoreName)
        .branchKeyId(branch-key-id)
        .ttlSeconds(600)
        .build();
final Keyring hierarchicalKeyring =
 matProv.CreateAwsKmsHierarchicalKeyring(keyringInput);
```

C# / .NET

```
var matProv = new MaterialProviders(new MaterialProvidersConfig());
var keyringInput = new CreateAwsKmsHierarchicalKeyringInput
{
   KeyStore = keystore,
   BranchKeyIdSupplier = branchKeyIdSupplier,
   TtlSeconds = 600
};
var hierarchicalKeyring = matProv.CreateAwsKmsHierarchicalKeyring(keyringInput);
```

旋轉您的活動分支密鑰

每個分支密鑰一次只能有一個活動版本。分層密鑰環通常使用每個活動分支密鑰版本來滿足多個請求。但是您可以控制重複使用活動分支密鑰的程度，並確定活動分支鍵的旋轉頻率。

分支密鑰不用於加密純文本數據密鑰。它們被用來派生加密明文數據密鑰的唯一包裝密鑰。包裝密鑰派生過程會產生具有 28 個字節隨機性的唯一 32 字節包裝密鑰。這意味著在發生密碼編譯耗損之前，分

支密鑰可以導出超過 79 個八進制或 2^{96} 個唯一包裝密鑰。儘管這種用盡風險非常低，但由於業務或合約規則或政府法規，您可能需要輪換活動中的分支密鑰。

分支索引鍵的作用中版本會保持作用中狀態，直到您旋轉它為止。以前版本的 Active 分支密鑰將不會用於執行加密操作，也無法用於導出新的包裝密鑰。但是仍然可以查詢它們並提供包裝密鑰以解密它們在活動時加密的數據密鑰。

使用金鑰存放區服務`VersionKey`作業輪換作用中的分支金鑰。當您旋轉作用中的分支索引鍵時，會建立新的分支索引鍵來取代先前的版本。當您旋轉作用中的分支索引鍵時，不`branch-key-id`會變更。您必須在`branch-key-id`呼叫時指定識別目前作用中分支索引鍵的`VersionKey`。

Java

```
keystore.VersionKey(
    VersionKeyInput.builder()
        .branchKeyIdentifier("branch-key-id")
        .build()
);
```

C# / .NET

```
keystore.VersionKey(new VersionKeyInput{BranchKeyIdentifier = branchKeyId});
```

搭配多租戶資料庫使用階層式金鑰圈

透過為資料庫中的每個租用戶建立分支金鑰，您可以使用在作用中分支金鑰及其衍生包裝金鑰之間建立的金鑰階層來支援多租戶資料庫。然後，階層式金鑰圈會使用其不同的分支金鑰加密並簽署指定租用戶的所有資料。這可讓您將多租戶資料儲存在單一資料庫中，並透過分支金鑰隔離租用戶資料。

每個承租人都有自己的分支索引鍵，由唯一定義`branch-key-id`。一次只能有一個`branch-key-id`作用中版本。

分支金鑰 ID 供應商

您必須先為每個承租人建立分支金鑰，並建立分支金鑰 ID 供應商，才能初始化多租戶使用的階層式金鑰環。分支金鑰 ID 供應商會使用儲存在加密內容中的欄位來決定解密記錄所需的租用戶分支金鑰。根據預設，只有分割區和排序索引鍵會包含在加密內容中。但是，您可以使用`SIGN_AND_INCLUDE_IN_ENCRYPTION_CONTEXT`密碼編譯動作在加密內容中包含其他欄位。

> **Note**
>
> 若要使用加SIGN_AND_INCLUDE_IN_ENCRYPTION_CONTEXT密動作，您必須使用 3.3 版或更新版本的 AWS 資料庫加密 SDK。在更新要包含的資料模型之前，請先將新版本部署至所有讀取器SIGN_AND_INCLUDE_IN_ENCRYPTION_CONTEXT。

您可以使用分支金鑰 ID 供應商為您建立易記名稱，branch-key-ids以便輕鬆識別租用戶branch-key-id的正確名稱。例如，易記名稱可讓您參考分支索引鍵，tenant1而不是b3f61619-4d35-48ad-a275-050f87e15122。

對於解密作業，您可以靜態設定單一階層式金鑰環，以限制對單一承租人的解密，或者您可以使用分支金鑰識別碼供應商來識別負責解密記錄的承租人。

首先，遵循先決條件程序的步驟 1 和步驟 2。然後，使用下列程序為每個承租人建立分支金鑰、建立分支金鑰識別碼供應商，並初始化階層式金鑰圈以供多租戶使用。

步驟 1：為資料庫中的每個租用戶建立分支金鑰

呼叫CreateKey資料庫中的每個租用戶。

下列作業會使用您在建立金鑰存放區服務時指定的 KMS 金鑰建立兩個分支金鑰，並將分支金鑰新增至您建立做為分支金鑰存放區使用的 DynamoDB 表格。相同的 KMS 金鑰必須保護所有分支金鑰。

Java

```
CreateKeyOutput branchKeyId1 =
 keystore.CreateKey(CreateKeyInput.builder().build());
CreateKeyOutput branchKeyId2 =
 keystore.CreateKey(CreateKeyInput.builder().build());
```

C# / .NET

```
var branchKeyId1 = keystore.CreateKey(new CreateKeyInput());
var branchKeyId2 = keystore.CreateKey(new CreateKeyInput());
```

步驟 2：建立分支金鑰 ID 供應商

下列範例會從步驟 1 中建立的兩個分支金鑰建立好記名稱，並呼叫`CreateDynamoDbEncryptionBranchKeyIdSupplier`使用 DynamoDB 用戶端的 AWS 資料庫加密 SDK 建立分支金鑰 ID 供應商。

Java

```
// Create friendly names for each branch-key-id
class ExampleBranchKeyIdSupplier implements IDynamoDbKeyBranchKeyIdSupplier {
    private static String branchKeyIdForTenant1;
    private static String branchKeyIdForTenant2;

    public ExampleBranchKeyIdSupplier(String tenant1Id, String tenant2Id) {
        this.branchKeyIdForTenant1 = tenant1Id;
        this.branchKeyIdForTenant2 = tenant2Id;
    }
// Create the branch key ID supplier
final DynamoDbEncryption ddbEnc = DynamoDbEncryption.builder()
        .DynamoDbEncryptionConfig(DynamoDbEncryptionConfig.builder().build())
        .build();
final BranchKeyIdSupplier branchKeyIdSupplier =
 ddbEnc.CreateDynamoDbEncryptionBranchKeyIdSupplier(
    CreateDynamoDbEncryptionBranchKeyIdSupplierInput.builder()
            .ddbKeyBranchKeyIdSupplier(new ExampleBranchKeyIdSupplier(branch-key-
ID-tenant1, branch-key-ID-tenant2))
            .build()).branchKeyIdSupplier();
```

C# / .NET

```
// Create friendly names for each branch-key-id
 class ExampleBranchKeyIdSupplier : DynamoDbKeyBranchKeyIdSupplierBase {
    private String _branchKeyIdForTenant1;
    private String _branchKeyIdForTenant2;

    public ExampleBranchKeyIdSupplier(String tenant1Id, String tenant2Id) {
        this._branchKeyIdForTenant1 = tenant1Id;
        this._branchKeyIdForTenant2 = tenant2Id;
    }
// Create the branch key ID supplier
var ddbEnc = new DynamoDbEncryption(new DynamoDbEncryptionConfig());
var branchKeyIdSupplier = ddbEnc.CreateDynamoDbEncryptionBranchKeyIdSupplier(
    new CreateDynamoDbEncryptionBranchKeyIdSupplierInput
    {
```

```
        DdbKeyBranchKeyIdSupplier = new ExampleBranchKeyIdSupplier(branch-key-ID-
tenant1, branch-key-ID-tenant2)
    }).BranchKeyIdSupplier;
```

步驟 3：使用分支密鑰 ID 供應商初始化分層密鑰環

若要初始化階層式金鑰圈，您必須提供下列值：

- 分支金鑰存放區名稱
- 快取限制存留時間 (TTL)
- 分支金鑰 ID 供應商
- (選擇性) 快取

  如果您想要自訂快取類型或可儲存在本機快取中的分支金鑰材料項目數目，請在初始化金鑰環時指定快取類型和項目容量。

  緩存類型定義了線程模型。階層式金鑰環提供三種支援多租戶資料庫的快取類型：預設值、、MultiThreaded. StormTracking

  如果您未指定快取，階層式金鑰圈會自動使用預設快取類型，並將項目容量設定為 1000。

  Default (Recommended)

  對於大多數用戶來說，默認緩存滿足他們的線程要求。默認緩存被設計為支持大量多線程環境。當分支密鑰材料條目到期時，默認緩存通知一個線程，該分支密鑰材料條目將提前 10 秒到期，以防止多個線程調 AWS KMS 用。這確保只有一個線程發送一個請求 AWS KMS 來刷新緩存。

  若要使用預設快取來初始化階層式金鑰圈，請指定下列值：

  - 項目容量：限制可儲存在本機快取中的分支金鑰材料項目數量。

  Java

  ```
  .cache(CacheType.builder()
          .Default(DefaultCache.builder()
          .entryCapacity(100)
          .build())
  ```

  C # /.

  ```
  CacheType defaultCache = new CacheType
  {
  ```

```
    Default = new DefaultCache{EntryCapacity = 100}
};
```

預設值和 StormTracking 快取支援相同的執行緒模型，但您只需要指定項目容量，即可使用預設快取來初始化階層式金鑰環。如需更精細的快取自訂項目，請使用 StormTracking 快取。

MultiThreaded

MultiThreaded 快取可在多執行緒環境中安全使用，但不提供任何可減少 AWS KMS 或 Amazon DynamoDB 呼叫的功能。因此，當分支金鑰材料項目到期時，所有執行緒都會同時收到通知。這可能會導致多次 AWS KMS 呼叫以重新整理快取。

若要使用 MultiThreaded 快取來初始化階層式金鑰圈，請指定下列值：

- 項目容量：限制可儲存在本機快取中的分支金鑰材料項目數量。
- 項目修剪尾端大小：定義達到入口容量時要修剪的項目數。

Java

```
.cache(CacheType.builder()
        .MultiThreaded(MultiThreadedCache.builder()
        .entryCapacity(100)
        .entryPruningTailSize(1)
        .build())
```

C # /.

```
CacheType multithreadedCache = new CacheType
{
    MultiThreaded = new MultiThreadedCache
    {
        EntryCapacity = 100,
        EntryPruningTailSize = 1
    }
};
```

StormTracking

該 StormTracking 緩存被設計為支持大量多線程環境。當分支密鑰材料條目到期時，StormTracking 緩存通知一個線程，該分支密鑰材料條目將提前過期，以防止多個線程調 AWS KMS 用。這確保只有一個線程發送一個請求 AWS KMS 來刷新緩存。

若要使用 StormTracking 快取來初始化階層式金鑰圈，請指定下列值：

- 項目容量：限制可儲存在本機快取中的分支金鑰材料項目數量。
- 項目修剪尾部大小：定義一次要修剪的分支關鍵材料項目數。

  預設值：1 個項目
- 寬限期：定義嘗試重新整理分支金鑰材料到期前的秒數。

  預設值：10 秒
- 寬限間隔：定義嘗試重新整理分支金鑰材料之間的秒數。

  預設值：1 秒
- 扇出：定義可同時嘗試重新整理分支金鑰材料的次數。

  預設值：20 次嘗試
- 存留時間 (TTL)：定義嘗試重新整理分支金鑰材料逾時之前的秒數。每當緩存返回`NoSuchEntry`以響應 a 時`GetCacheEntry`，該分支密鑰都會被視為處於飛行狀態，直到用條目寫`PutCache`入相同的密鑰為止。

  預設值：20 秒
- 睡眠：定義如果超過執行緒應該睡眠的秒數。`fanOut`

  預設值：20 毫秒

Java

```
.cache(CacheType.builder()
        .StormTracking(StormTrackingCache.builder()
        .entryCapacity(100)
        .entryPruningTailSize(1)
        .gracePeriod(10)
        .graceInterval(1)
        .fanOut(20)
        .inFlightTTL(20)
        .sleepMilli(20)
        .build())
```

C# /.

```
CacheType stormTrackingCache = new CacheType
```

```
{
    StormTracking = new StormTrackingCache
    {
        EntryCapacity = 100,
        EntryPruningTailSize = 1,
        FanOut = 20,
        GraceInterval = 1,
        GracePeriod = 10,
        InFlightTTL = 20,
        SleepMilli = 20
    }
};
```

- （可選）授予令牌列表

  如果您使用授權控制階層式金鑰圈中 KMS 金鑰的存取權，則必須在初始化金鑰環時提供所有必要的授權權杖。

下列範例會使用在步驟 2 中建立的分支金鑰 ID 供應商、快取限制 TLL 600 秒，最大快取大小為 1000，初始化階層式金鑰圈。此範例使用 DynamoDB 用戶端的 AWS 資料庫加密 SDK 初始化階層式金鑰圈。

Java

```
final MaterialProviders matProv = MaterialProviders.builder()
        .MaterialProvidersConfig(MaterialProvidersConfig.builder().build())
        .build();
final CreateAwsKmsHierarchicalKeyringInput keyringInput =
 CreateAwsKmsHierarchicalKeyringInput.builder()
        .keyStore(keystore)
        .branchKeyIdSupplier(branchKeyIdSupplier)
        .ttlSeconds(600)
        .cache(CacheType.builder() //OPTIONAL
                .Default(DefaultCache.builder()
                .entryCapacity(100)
                .build())
        .build();
final Keyring hierarchicalKeyring =
 matProv.CreateAwsKmsHierarchicalKeyring(keyringInput);
```

C# / .NET

```
var matProv = new MaterialProviders(new MaterialProvidersConfig());
```

```
var keyringInput = new CreateAwsKmsHierarchicalKeyringInput
{
   KeyStore = keystore,
   BranchKeyIdSupplier = branchKeyIdSupplier,
   TtlSeconds = 600,
   Cache = new CacheType
   {
        Default = new DefaultCache { EntryCapacity = 100 }
   }
};
var hierarchicalKeyring = matProv.CreateAwsKmsHierarchicalKeyring(keyringInput);
```

使用階層式金鑰圈進行可搜尋的加密

可搜索的加密使您無需解密整個數據庫即可搜索加密記錄。這是透過使用信標對加密欄位的純文字值進行索引來完成的。若要實作可搜尋的加密，您必須使用階層式金鑰圈。

金鑰儲存區CreateKey作業會同時產生分支索引鍵和信標金鑰。分支密鑰用於記錄加密和解密操作。信標索引鍵是用來產生信標。

分支金鑰和信標金鑰會受到您在建立金鑰存放區服務時指定的相同 AWS KMS key 保護。CreateKey作業呼叫 AWS KMS 產生分支金鑰後，會呼叫 kms：GenerateDataKeyWithoutPlaintext第二次使用下列要求產生信標金鑰。

```
{
   "EncryptionContext": {
      "branch-key-id" : "branch-key-id",
      "type" : type,
      "create-time" : "timestamp",
      "logical-key-store-name" : "the logical table name for your branch key store",
      "kms-arn" : the KMS key ARN,
      "hierarchy-version" : 1
   },
   "KeyId": "the KMS key ARN",
   "NumberOfBytes": "32"
}
```

產生這兩個金鑰之後，CreateKey作業會呼叫 ddb: TransactWriteItems 撰寫兩個新項目，這兩個新項目會在您的分支金鑰存放區中保留分支索引鍵和 Beacon 索引鍵。

設定標準信標時， AWS 資料庫加密 SDK 會查詢信標金鑰的分支金鑰存放區。然後，它使用基於 HMAC 的 extract-and-expand 密鑰派生函數（ HKDF ）將信標密鑰與標準信標的名稱相結合，以創建給定信標的 HMAC 密鑰。

與分支金鑰不同，分支金鑰存放區`branch-key-id`中每個只有一個信標金鑰版本。信標鍵永遠不會旋轉。

定義信標金鑰來源

當您定義標準和複合信標的信標版本時，必須識別信標 (Beacon) 索引鍵，並定義信標主要原物料的快取存留時間 (TTL)。信標金鑰材料會儲存在與分支金鑰不同的本機快取中。下列程式碼片段示範如何為單租`keySource`用戶資料庫定義。透過與其關聯的信標金鑰`branch-key-id`來識別您的信標金鑰。

Java

```
keySource(BeaconKeySource.builder()
        .single(SingleKeyStore.builder()
                .keyId(branch-key-id)
                .cacheTTL(6000)
                .build())
        .build())
```

C# / .NET

```
KeySource = new BeaconKeySource
{
    Single = new SingleKeyStore
    {
       KeyId = branch-key-id,
       CacheTTL = 6000
    }
}
```

在多租戶資料庫中定義信標來源

如果您有多租戶資料庫，則必須在設定. `keySource`

- keyFieldName

定義欄位名稱，此欄位名稱會儲存branch-key-id與指定承租人產生之信標所用之信標之信標關聯的信標索引鍵。keyFieldName可以是任何字串，但資料庫中所有其他欄位必須是唯一的。當您將新記錄寫入資料庫時，此欄位會儲存識別用來產生該記錄之任何信標的信標索引鍵。branch-key-id您必須在信標查詢中包含此欄位，並識別重新計算信標所需的適當信標主要材料。如需詳細資訊，請參閱 查詢多租戶資料庫中的信標。

- 快取記憶體

  區域信標快取記憶體中的信標主要材料項目在到期之前，可以使用的時間 (秒)。該值必須大於零。當快取限制 TTL 到期時，會從本機快取中逐出項目。

- (選擇性) 快取

  如果您想要自訂快取類型或可儲存在本機快取中的分支金鑰材料項目數目，請在初始化金鑰環時指定快取類型和項目容量。

  緩存類型定義了線程模型。階層式金鑰環提供三種支援多租戶資料庫的快取類型：預設值、、MultiThreaded. StormTracking

  如果您未指定快取，階層式金鑰圈會自動使用預設快取類型，並將項目容量設定為 1000。

  Default (Recommended)

  對於大多數用戶來說，默認緩存滿足他們的線程要求。默認緩存被設計為支持大量多線程環境。當分支密鑰材料條目到期時，默認緩存通知一個線程，該分支密鑰材料條目將提前 10 秒到期，以防止多個線程調 AWS KMS 用。這確保只有一個線程發送一個請求 AWS KMS 來刷新緩存。

  若要使用預設快取來初始化階層式金鑰圈，請指定下列值：

  - 項目容量：限制可儲存在本機快取中的分支金鑰材料項目數量。

  Java

  ```
  .cache(CacheType.builder()
          .Default(DefaultCache.builder()
          .entryCapacity(100)
          .build())
  ```

  C # /.

  ```
  CacheType defaultCache = new CacheType
  {
  ```

```
    Default = new DefaultCache{EntryCapacity = 100}
};
```

預設值和 StormTracking 快取支援相同的執行緒模型，但您只需要指定項目容量，即可使用預設快取來初始化階層式金鑰環。如需更精細的快取自訂項目，請使用 StormTracking 快取。

MultiThreaded

MultiThreaded 快取可在多執行緒環境中安全使用，但不提供任何可減少 AWS KMS 或 Amazon DynamoDB 呼叫的功能。因此，當分支金鑰材料項目到期時，所有執行緒都會同時收到通知。這可能會導致多次 AWS KMS 呼叫以重新整理快取。

若要使用 MultiThreaded 快取來初始化階層式金鑰圈，請指定下列值：

- 項目容量：限制可儲存在本機快取中的分支金鑰材料項目數量。
- 項目修剪尾端大小：定義達到入口容量時要修剪的項目數。

Java

```
.cache(CacheType.builder()
        .MultiThreaded(MultiThreadedCache.builder()
        .entryCapacity(100)
        .entryPruningTailSize(1)
        .build())
```

C # /.

```
CacheType multithreadedCache = new CacheType
{
    MultiThreaded = new MultiThreadedCache
    {
        EntryCapacity = 100,
        EntryPruningTailSize = 1
    }
};
```

StormTracking

該 StormTracking 緩存被設計為支持大量多線程環境。當分支密鑰材料條目到期時，StormTracking 緩存通知一個線程，該分支密鑰材料條目將提前過期，以防止多個線程調 AWS KMS 用。這確保只有一個線程發送一個請求 AWS KMS 來刷新緩存。

若要使用 StormTracking 快取來初始化階層式金鑰圈，請指定下列值：

- 項目容量：限制可儲存在本機快取中的分支金鑰材料項目數量。
- 項目修剪尾部大小：定義一次要修剪的分支關鍵材料項目數。

  預設值：1 個項目
- 寬限期：定義嘗試重新整理分支金鑰材料到期前的秒數。

  預設值：10 秒
- 寬限間隔：定義嘗試重新整理分支金鑰材料之間的秒數。

  預設值：1 秒
- 扇出：定義可同時嘗試重新整理分支金鑰材料的次數。

  預設值：20 次嘗試
- 存留時間 (TTL)：定義嘗試重新整理分支金鑰材料逾時之前的秒數。每當緩存返回`NoSuchEntry`以響應 a 時`GetCacheEntry`，該分支密鑰都會被視為處於飛行狀態，直到用條目寫`PutCache`入相同的密鑰為止。

  預設值：20 秒
- 睡眠：定義如果超過執行緒應該睡眠的秒數。`fanOut`

  預設值：20 毫秒

Java

```
.cache(CacheType.builder()
        .StormTracking(StormTrackingCache.builder()
        .entryCapacity(100)
        .entryPruningTailSize(1)
        .gracePeriod(10)
        .graceInterval(1)
        .fanOut(20)
        .inFlightTTL(20)
        .sleepMilli(20)
        .build())
```

C# /.

```
CacheType stormTrackingCache = new CacheType
```

```
{
    StormTracking = new StormTrackingCache
    {
        EntryCapacity = 100,
        EntryPruningTailSize = 1,
        FanOut = 20,
        GraceInterval = 1,
        GracePeriod = 10,
        InFlightTTL = 20,
        SleepMilli = 20
    }
};
```

下列範例會使用在步驟 2 中建立的分支金鑰 ID 供應商、快取限制 TLL 600 秒，以及 1000 的項目容量，初始化階層式金鑰圈。

Java

```
final MaterialProviders matProv = MaterialProviders.builder()
        .MaterialProvidersConfig(MaterialProvidersConfig.builder().build())
        .build();
final CreateAwsKmsHierarchicalKeyringInput keyringInput =
 CreateAwsKmsHierarchicalKeyringInput.builder()
        .keyStore(branchKeyStoreName)
        .branchKeyIdSupplier(branchKeyIdSupplier)
        .ttlSeconds(600)
        .cache(CacheType.builder() //OPTIONAL
                .Default(DefaultCache.builder()
                        .entryCapacity(1000)
                        .build())
                .build();
final IKeyring hierarchicalKeyring =
 matProv.CreateAwsKmsHierarchicalKeyring(keyringInput);
```

C# / .NET

```
var matProv = new MaterialProviders(new MaterialProvidersConfig());
var keyringInput = new CreateAwsKmsHierarchicalKeyringInput
{
   KeyStore = keystore,
   BranchKeyIdSupplier = branchKeyIdSupplier,
   TtlSeconds = 600,
   Cache = new CacheType
```

```
    {
        Default = new DefaultCache { EntryCapacity = 1000 }
    }
};
var hierarchicalKeyring = matProv.CreateAwsKmsHierarchicalKeyring(keyringInput);
```

AWS KMS ECDH 鑰匙圈

我們的用戶端加密程式庫已重新命名為 AWS 資料庫加密 SDK。本開發人員指南仍提供 DynamoDB 加密用戶端的相關資訊。

Important

AWS KMS ECDH 鑰匙圈僅適用於材料提供者資料庫的 1.5.0 版。

AWS KMS ECDH 金鑰圈使用非對稱金鑰協定AWS KMS keys來衍生雙方之間的共用對稱包裝金鑰。首先，金鑰圈會使用橢圓曲線迪菲-赫爾曼 (ECDH) 金鑰合約演算法，從寄件者的 KMS key pair 中的私密金鑰和收件者的公開金鑰衍生共用密碼。然後，金鑰圈會使用共用密碼衍生共用包裝金鑰，以保護您的資料加密金鑰。 AWS 資料庫加密 SDK 使用 (KDF_CTR_HMAC_SHA384) 衍生共用包裝金鑰的金鑰衍生函數，符合 NIST 金鑰衍生的建議。

密鑰派生函數返回 64 個字節的鍵控材料。為了確保雙方使用正確的金鑰資料，資料 AWS 庫加密 SDK 會使用前 32 個位元組做為承諾金鑰，最後 32 個位元組作為共用包裝金鑰。解密時，如果金鑰圈無法重現儲存在加密記錄之 [資料描述] 欄位中的相同承諾金鑰和共用包裝金鑰，作業就會失敗。例如，如果您使用以 Alice 私密金鑰和 Bob 公開金鑰設定的金鑰圈來加密記錄，則使用 Bob 的私密金鑰和 Alice 的公開金鑰設定的金鑰圈將會重現相同的承諾金鑰和共用包裝金鑰，並且能夠解密記錄。如果 Bob 的公開金鑰不是來自 KMS key pair，則 Bob 可以建立原始 ECDH 金鑰圈來解密記錄。

AWS KMS ECDH 密鑰環使用 AES-GCM 使用對稱密鑰對記錄進行加密。然後使用 AES-GCM 使用派生的共享包裝密鑰對數據密鑰進行包絡加密。每個 AWS KMS ECDH 鑰匙圈只能有一個共用包裝金鑰，但您可以將多個 AWS KMS ECDH 鑰匙圈 (單獨或與其他鑰匙圈一起) 包含在一個多鑰匙圈中。

主題

- AWS KMS ECDH 金鑰圈所需的權限
- 建立一個 AWS KMS ECDH 金鑰圈

- 建立 AWS KMS ECDH 探索金鑰圈

AWS KMS ECDH 金鑰圈所需的權限

數 AWS 據庫加密 SDK 不需要 AWS 帳戶，也不依賴於任何 AWS 服務。但是，若要使用 AWS KMS ECDH 金鑰圈，您需要一個 AWS 帳戶以及金鑰圈 AWS KMS keys 中的下列最低權限。權限會根據您使用的金鑰合約結構描述而有所不同。

- 若要使用金KmsPrivateKeyToStaticPublicKey鑰合約結構描述加密GetPublicKey和解密記錄，您需要寄件者的非對稱 KMS key pair DeriveSharedSecret 上的 kms: 和 kms:。如果您在實體化金鑰圈時直接提供寄件者的 DER 編碼公開金鑰，則只需要 kms: 寄件者非對稱 KMS key pair 的 DeriveSharedSecret權限。
- 若要使用金KmsPublicKeyDiscovery鑰合約結構描述解密記錄，您需要指定的非對稱 KMS key pair 的 kms: DeriveSharedSecret 和 kms: GetPublicKey 權限。

建立一個 AWS KMS ECDH 金鑰圈

若要建立加密和解密資料的 AWS KMS ECDH 金鑰圈，您必須使用金鑰合約結構描述KmsPrivateKeyToStaticPublicKey。若要使用KmsPrivateKeyToStaticPublicKey金鑰合約結構描述初始化 AWS KMS ECDH 金鑰圈，請提供下列值：

- 寄件者 AWS KMS key 識別碼

  必須識別非對稱 NIST 建議的橢圓曲線 (ECC) KMS key pair，其值為。KeyUsage KEY_AGREEMENT發件人的私鑰用於導出共享密鑰。

- (選擇性) 寄件者的公開金鑰

  必須是一個德編碼的 X.509 公開金鑰，也稱為 SubjectPublicKeyInfo (SPKI)，如 RFC 5280 中所定義。

  此 AWS KMS GetPublicKey作業會以必要的 DER 編碼格式傳回非對稱 KMS key pair 的公開金鑰。

  若要減少金鑰圈 AWS KMS 撥打的通話次數，您可以直接提供寄件者的公開金鑰。如果沒有為寄件者的公開金鑰提供值，金鑰圈會呼叫擷 AWS KMS 取寄件者的公開金鑰。

- 收件者的公開金鑰

  您必須提供收件者的 DER 編碼 X.509 公開金鑰，也稱為 SubjectPublicKeyInfo (SPKI)，如 RFC 5280 中所定義。

此 AWS KMS GetPublicKey作業會以必要的 DER 編碼格式傳回非對稱 KMS key pair 的公開金鑰。

- 曲線規格

  識別指定索引鍵對中的橢圓曲線規格。寄件者和收件者的金鑰配對必須具有相同的曲線規格。

  有效值：ECC_NIST_P256、ECC_NIS_P384、ECC_NIST_P512

- （可選）授予令牌列表

  如果您使用授權控制 AWS KMS ECDH 金鑰圈中 KMS 金鑰的存取權，則必須在初始化金鑰環時提供所有必要的授權權杖。

Java

下列範例會使用寄件者的 KMS 金鑰、寄件者的公開金鑰和收件者的公開金鑰建立 AWS KMS ECDH 金鑰圈。此範例使用選用senderPublicKey參數來提供寄件者的公開金鑰。如果您未提供寄件者的公開金鑰，金鑰圈會呼叫 AWS KMS 以擷取寄件者的公開金鑰。寄件者和收件者的金鑰配對都在ECC_NIST_P256曲線上。

```
// Retrieve public keys
// Must be DER-encoded X.509 public keys
ByteBuffer BobPublicKey = getPublicKeyBytes("arn:aws:kms:us-
west-2:111122223333:key/1234abcd-12ab-34cd-56ef-1234567890ab");
        ByteBuffer AlicePublicKey = getPublicKeyBytes("arn:aws:kms:us-
west-2:111122223333:key/0987dcba-09fe-87dc-65ba-ab0987654321");

// Create the AWS KMS ECDH static keyring
        final CreateAwsKmsEcdhKeyringInput senderKeyringInput =
          CreateAwsKmsEcdhKeyringInput.builder()
            .kmsClient(KmsClient.create())
            .curveSpec(ECDHCurveSpec.ECC_NIST_P256)
            .KeyAgreementScheme(
              KmsEcdhStaticConfigurations.builder()
                .KmsPrivateKeyToStaticPublicKey(
                  KmsPrivateKeyToStaticPublicKeyInput.builder()
                    .senderKmsIdentifier("arn:aws:kms:us-
west-2:111122223333:key/1234abcd-12ab-34cd-56ef-1234567890ab")
                    .senderPublicKey(BobPublicKey)
                    .recipientPublicKey(AlicePublicKey)
                    .build()).build()).build();
```

建立 AWS KMS ECDH 探索金鑰圈

解密時，最佳做法是指定 AWS 資料庫加密 SDK 可以使用的金鑰。若要遵循此最佳做法，請使用 AWS KMS ECDH 金鑰圈搭配`KmsPrivateKeyToStaticPublicKey`金鑰合約結構描述。不過，您也可以建立 AWS KMS ECDH 探索金鑰圈，也就是 AWS KMS ECDH 金鑰圈，該金鑰圈可以解密指定 KMS 金鑰 key pair 的公開金鑰與儲存在加密記錄之 [材料描述] 欄位中的收件者公開金鑰相符的任何記錄。

> **Important**
>
> 當您使用金`KmsPublicKeyDiscovery`鑰合約結構描述解密記錄時，您會接受所有公開金鑰，不論其擁有者為何。

若要使用`KmsPublicKeyDiscovery`金鑰合約結構描述初始化 AWS KMS ECDH 金鑰圈，請提供下列值：

- 收件人的 AWS KMS key 識別碼

  必須識別非對稱 NIST 建議的橢圓曲線 (ECC) KMS key pair，其值為。`KeyUsage` `KEY_AGREEMENT`
- 曲線規格

  識別收件者的 KMS key pair 中的橢圓曲線規格。

  有效值：`ECC_NIST_P256`、`ECC_NIS_P384`、`ECC_NIST_P512`
- （可選）授予令牌列表

  如果您使用授權控制 AWS KMS ECDH 金鑰圈中 KMS 金鑰的存取權，則必須在初始化金鑰環時提供所有必要的授權權杖。

Java

下列範例會在曲線上建立具有 KMS key pair 的 AWS KMS ECDH 探索金鑰圈。`ECC_NIST_P256`您必須擁有指定 KMS key pair 的 kms: GetPublicKey 和 kms: DeriveSharedSecret 權限。此金鑰圈可以解密指定 KMS key pair 的公開金鑰與儲存在加密記錄之材料描述欄位中的收件者公開金鑰相符的任何記錄。

```
// Create the AWS KMS ECDH discovery keyring
final CreateAwsKmsEcdhKeyringInput recipientKeyringInput =
  CreateAwsKmsEcdhKeyringInput.builder()
```

```
    .kmsClient(KmsClient.create())
    .curveSpec(ECDHCurveSpec.ECC_NIST_P256)
    .KeyAgreementScheme(
      KmsEcdhStaticConfigurations.builder()
        .KmsPublicKeyDiscovery(
          KmsPublicKeyDiscoveryInput.builder()
            .recipientKmsIdentifier("arn:aws:kms:us-
west-2:111122223333:key/0987dcba-09fe-87dc-65ba-ab0987654321").build()
        ).build())
    .build();
```

原始 AES keyring

我們的用戶端加密程式庫已重新命名為 AWS 資料庫加密 SDK。本開發人員指南仍提供 DynamoDB 加密用戶端的相關資訊。

資料 AWS 庫加密 SDK 可讓您使用您提供的 AES 對稱金鑰做為保護資料金鑰的包裝金鑰。您必須產生、儲存及保護金鑰資料，最好是在硬體安全模組 (HSM) 或金鑰管理系統中。當您需要提供包裝金鑰並在本機或離線加密資料金鑰時，請使用 Raw AES 金鑰圈。

Raw AES 金鑰環會使用 AES-GCM 演算法和您指定為位元組陣列的包裝金鑰來加密資料。您只能在每個 Raw AES 金鑰圈中指定一個環繞金鑰，但您可以在多重金鑰圈中包含多個 Raw AES 鑰匙圈，單獨或與其他鑰匙圈一起包含多個 Raw AES 鑰匙圈。

關鍵命名空間和名稱

若要識別金鑰圈中的 AES 金鑰，Raw AES 金鑰圈會使用您提供的金鑰命名空間和金鑰名稱。這些值並非機密。它們以純文本顯示在 AWS 數據庫加密 SDK 添加到記錄的材料描述中。我們建議您使用 HSM 或金鑰管理系統的金鑰命名空間，以及識別該系統中 AES 金鑰的金鑰名稱。

> **Note**
>
> 金鑰命名空間和金鑰名稱等同於中的「提供者 ID」(或「提供者」) 和「金鑰 ID」欄位`JceMasterKey`。

如果您構建不同的密鑰環來加密和解密給定字段，則命名空間和名稱值非常重要。如果解密金鑰圈中的金鑰命名空間和金鑰名稱與加密金鑰圈中的金鑰命名空間和金鑰名稱不完全相符且區分大小寫，即使金鑰材料位元組相同，也不會使用解密金鑰圈。

例如，您可以使用金鑰命名空間HSM_01和金鑰名稱AES_256_012來定義 Raw AES 金鑰環。然後，您可以使用該密鑰環來加密某些數據。若要解密該資料，請使用相同的金鑰命名空間、金鑰名稱和金鑰材料建構 Raw AES 金鑰環。

下列範例說明如何建立 Raw AES 金鑰圈。AESWrappingKey變數代表您提供的關鍵材料。

Java

```
final CreateRawAesKeyringInput keyringInput = CreateRawAesKeyringInput.builder()
        .keyName("AES_256_012")
        .keyNamespace("HSM_01")
        .wrappingKey(AESWrappingKey)
        .wrappingAlg(AesWrappingAlg.ALG_AES256_GCM_IV12_TAG16)
        .build();
final MaterialProviders matProv = MaterialProviders.builder()
        .MaterialProvidersConfig(MaterialProvidersConfig.builder().build())
        .build();
IKeyring rawAesKeyring = matProv.CreateRawAesKeyring(keyringInput);
```

C# / .NET

```
var keyNamespace = "HSM_01";
var keyName = "AES_256_012";

// This example uses the key generator in Bouncy Castle to generate the key
 material.
// In production, use key material from a secure source.
var aesWrappingKey = new
 MemoryStream(GeneratorUtilities.GetKeyGenerator("AES256").GenerateKey());

// Create the keyring
var keyringInput = new CreateRawAesKeyringInput
{
    KeyNamespace = keyNamespace,
    KeyName = keyName,
    WrappingKey = AESWrappingKey,
    WrappingAlg = AesWrappingAlg.ALG_AES256_GCM_IV12_TAG16
};
```

```
var matProv = new MaterialProviders(new MaterialProvidersConfig());
IKeyring rawAesKeyring = matProv.CreateRawAesKeyring(keyringInput);
```

原始 RSA keyring

我們的用戶端加密程式庫已重新命名為 AWS 資料庫加密 SDK。本開發人員指南仍提供 DynamoDB 加密用戶端的相關資訊。

Raw RSA 金鑰圈會使用您提供的 RSA 公開金鑰和私密金鑰，對本機記憶體中的資料金鑰執行非對稱加密和解密。您必須產生、儲存及保護私密金鑰，最好是在硬體安全模組 (HSM) 或金鑰管理系統中。加密函數會根據 RSA 公開金鑰加密資料金鑰。解密函數會使用私有金鑰解密資料金鑰。您可以從數個 RSA 填補模式中選擇。

加密和解密的原始 RSA keyring，必須包含非對稱公開金鑰和私有金鑰對。不過，您可以使用只有公開金鑰的 RSA 金鑰圈來加密資料，也可以使用只有私密金鑰的 RSA 金鑰圈來解密資料。您可以在多鑰匙圈中包含任何 Raw RSA 鑰匙圈。如果您使用公開金鑰和私密金鑰來設定 Raw RSA 金鑰圈，請確定它們是相同 key pair 的一部分。

原始 RSA 金鑰圈等同於中與 RSA 非對稱加密金鑰搭配使用 適用於 JAVA 的 AWS Encryption SDK 時，與JceMasterKey中相互操作。

> **Note**
>
> 原始 RSA 金鑰圈不支援非對稱 KMS 金鑰。若要使用非對稱 RSA KMS 金鑰，請建構金鑰AWS KMS 圈。

命名空間和名稱

若要識別金鑰圈中的 RSA 金鑰材料，Raw RSA 金鑰圈會使用您提供的金鑰命名空間和金鑰名稱。這些值並非機密。它們以純文本顯示在 AWS 數據庫加密 SDK 添加到記錄的材料描述中。我們建議您在 HSM 或金鑰管理系統中使用識別 RSA key pair (或其私密金鑰) 的金鑰命名空間和金鑰名稱。

> **Note**
>
> 金鑰命名空間和金鑰名稱等同於中的「提供者 ID」(或「提供者」) 和「金鑰 ID」欄位`JceMasterKey`。

如果您構建不同的密鑰環來加密和解密給定的記錄，則命名空間和名稱值非常重要。如果解密金鑰圈中的金鑰命名空間和金鑰名稱與加密金鑰圈中的金鑰命名空間和金鑰名稱不完全相符且區分大小寫，即使金鑰來自相同的金鑰組，也不會使用解密金鑰圈。

無論金鑰圈包含 RSA 公開金鑰、RSA 私密金鑰或 key pair 中的兩個金鑰，加密和解密金鑰環中的金鑰資料的金鑰名稱空間和金鑰資料都必須相同。例如，假設您使用具有金鑰命名空間`HSM_01`和金鑰名稱之 RSA 公開金鑰的 RSA 金鑰圈來加密資料。`RSA_2048_06`若要解密該資料，請使用私密金鑰 (或金鑰組) 以及相同的金鑰命名空間和名稱來建構 Raw RSA 金鑰環。

填充模式

您必須為用於加密和解密的 RSA 金鑰環指定填補模式，或使用語言實作的功能來為您指定該密鑰環。

AWS Encryption SDK 支持以下填充模式，受到每種語言的約束。我們建議使用 OAEP 填充模式，特別是使用 SHA-256 和 MGF1 搭配 SHA-256 填充模式。只有向下相容性才支援 PKCS1 填補模式。

- 使用 SHA-1 和 MGF1 配備 SHA-1 填充的「老有增值計劃」
- 使用 SHA-256 和 MGF1 配備 SHA-256 填充的「老有增值計劃」
- 使用 SHA-384 和 MGF1 配備 SHA-384 填充的「老有增值計劃」
- 使用 SHA-512 和 MGF1 配備 SHA-512 填充的「老有增值計劃」
- PKCS1 填充

下列 Java 範例會示範如何使用 RSA 金鑰組的公開和私密金鑰，以及使用 SHA-256 和 MGF1 (含 SHA-256 填補模式) 建立原始 RSA 金鑰圈。`RSAPublicKey`和`RSAPrivateKey`變數代表您提供的關鍵材料。

Java

```
final CreateRawRsaKeyringInput keyringInput = CreateRawRsaKeyringInput.builder()
        .keyName("RSA_2048_06")
        .keyNamespace("HSM_01")
        .paddingScheme(PaddingScheme.OAEP_SHA256_MGF1)
        .publicKey(RSAPublicKey)
        .privateKey(RSAPrivateKey)
        .build();
final MaterialProviders matProv = MaterialProviders.builder()
        .MaterialProvidersConfig(MaterialProvidersConfig.builder().build())
        .build();
IKeyring rawRsaKeyring = matProv.CreateRawRsaKeyring(keyringInput);
```

C# / .NET

```
var keyNamespace = "HSM_01";
var keyName = "RSA_2048_06";

// Get public and private keys from PEM files
var publicKey = new
 MemoryStream(System.IO.File.ReadAllBytes("RSAKeyringExamplePublicKey.pem"));
var privateKey = new
 MemoryStream(System.IO.File.ReadAllBytes("RSAKeyringExamplePrivateKey.pem"));

// Create the keyring input
var keyringInput = new CreateRawRsaKeyringInput
{
    KeyNamespace = keyNamespace,
    KeyName = keyName,
    PaddingScheme = PaddingScheme.OAEP_SHA512_MGF1,
    PublicKey = publicKey,
    PrivateKey = privateKey
};

// Create the keyring
var matProv = new MaterialProviders(new MaterialProvidersConfig());
var rawRsaKeyring = matProv.CreateRawRsaKeyring(keyringInput);
```

原始 ECDH 鑰匙圈

我們的用戶端加密程式庫已重新命名為 AWS 資料庫加密 SDK。本開發人員指南仍提供 DynamoDB 加密用戶端的相關資訊。

Important

原始 ECDH 鑰匙圈僅適用於材料提供者資料庫的 1.5.0 版本。

Raw ECDH 金鑰圈會使用您提供的橢圓曲線公開-私密金鑰配對，以衍生雙方之間的共用包裝金鑰。首先，金鑰圈會使用寄件者的私密金鑰、收件者的公開金鑰和橢圓曲線迪菲-赫爾曼 (ECDH) 金鑰協定演算法衍生共用密碼。然後，金鑰圈會使用共用密碼衍生共用包裝金鑰，以保護您的資料加密金

鑰。 AWS 資料庫加密 SDK 使用 (KDF_CTR_HMAC_SHA384) 衍生共用包裝金鑰的金鑰衍生函數，符合 NIST 金鑰衍生的建議。

密鑰派生函數返回 64 個字節的鍵控材料。為了確保雙方使用正確的金鑰資料，資料 AWS 庫加密 SDK 會使用前 32 個位元組做為承諾金鑰，最後 32 個位元組作為共用包裝金鑰。解密時，如果金鑰圈無法重現儲存在加密記錄之 [資料描述] 欄位中的相同承諾金鑰和共用包裝金鑰，作業就會失敗。例如，如果您使用以 Alice 私密金鑰和 Bob 公開金鑰設定的金鑰圈來加密記錄，則使用 Bob 的私密金鑰和 Alice 的公開金鑰設定的金鑰圈將會重現相同的承諾金鑰和共用包裝金鑰，並且能夠解密記錄。如果 Bob 的公開金鑰來自一 AWS KMS key 對，則 Bob 可以建立 AWS KMS ECDH 金鑰圈來解密記錄。

原始 ECDH 密鑰環使用 AES-GCM 使用對稱密鑰對記錄進行加密。然後使用 AES-GCM 使用派生的共享包裝密鑰對數據密鑰進行包絡加密。每個 Raw ECDH 鑰匙圈只能有一個共用包裝金鑰，但您可以將多個 Raw ECDH 鑰匙圈（單獨或與其他鑰匙圈一起）包含在多重鑰匙圈中。

您必須負責產生、儲存及保護私密金鑰，最好是在硬體安全模組 (HSM) 或金鑰管理系統中。寄件者和收件者的金鑰配對大部分位於相同的橢圓曲線上。資 AWS 料庫加密 SDK 支援下列橢圓切割規格：

- ECC_NIST_P256
- ECC_NIST_P384
- ECC_NIST_P512

建立原始 ECDH 金鑰圈

Raw ECDH 金鑰圈支援三個金鑰合約結構描述：RawPrivateKeyToStaticPublicKey、EphemeralPrivateKeyToStaticPublicKey和。PublicKeyDiscovery您選取的金鑰協定結構描述會決定您可以執行哪些密碼編譯作業，以及鍵控材料的組合方式。

主題

- RawPrivateKeyToStaticPublicKey
- EphemeralPrivateKeyToStaticPublicKey
- PublicKeyDiscovery

RawPrivateKeyToStaticPublicKey

使用RawPrivateKeyToStaticPublicKey金鑰合約結構描述，在金鑰圈中以靜態方式設定寄件者的私密金鑰和收件者的公開金鑰。此金鑰合約結構描述可以加密和解密記錄。

若要使用RawPrivateKeyToStaticPublicKey金鑰合約結構描述初始化 Raw ECDH 金鑰圈，請提供下列值：

- 寄件者的私密金鑰

  您必須提供寄件者的 PEM 編碼私密金鑰 (PKCS #8 PrivateKeyInfo 結構)，如 RFC 5958 中所定義。

- 收件者的公開金鑰

  您必須提供收件者的 DER 編碼 X.509 公開金鑰，也稱為 SubjectPublicKeyInfo (SPKI)，如 RFC 5280 中所定義。

  您可以指定非對稱金鑰合約 KMS key pair 的公開金鑰，或從外部產生的 key pair 中指定公開金鑰 AWS。

- 曲線規格

  識別指定索引鍵對中的橢圓曲線規格。寄件者和收件者的金鑰配對必須具有相同的曲線規格。

  有效值：ECC_NIST_P256、ECC_NIS_P384、ECC_NIST_P512

Java

下列 Java 範例會使用RawPrivateKeyToStaticPublicKey金鑰合約結構描述，以靜態方式設定寄件者的私密金鑰和收件者的公開金鑰。兩個鍵對都在ECC_NIST_P256曲線上。

```
private static void StaticRawKeyring() {
    // Instantiate material providers
    final MaterialProviders materialProviders =
      MaterialProviders.builder()
        .MaterialProvidersConfig(MaterialProvidersConfig.builder().build())
        .build();

    KeyPair senderKeys = GetRawEccKey();
    KeyPair recipient = GetRawEccKey();

    // Create the Raw ECDH static keyring
    final CreateRawEcdhKeyringInput rawKeyringInput =
      CreateRawEcdhKeyringInput.builder()
        .curveSpec(ECDHCurveSpec.ECC_NIST_P256)
        .KeyAgreementScheme(
          RawEcdhStaticConfigurations.builder()
            .RawPrivateKeyToStaticPublicKey(
```

```
                RawPrivateKeyToStaticPublicKeyInput.builder()
                  // Must be a PEM-encoded private key

 .senderStaticPrivateKey(ByteBuffer.wrap(senderKeys.getPrivate().getEncoded()))
                  // Must be a DER-encoded X.509 public key

 .recipientPublicKey(ByteBuffer.wrap(recipient.getPublic().getEncoded()))
                  .build()
            )
            .build()
        ).build();

    final IKeyring staticKeyring =
 materialProviders.CreateRawEcdhKeyring(rawKeyringInput);
}
```

EphemeralPrivateKeyToStaticPublicKey

使用`EphemeralPrivateKeyToStaticPublicKey`金鑰合約結構描述設定的金鑰圈會在本機建立新的 key pair，並為每個加密呼叫衍生唯一的共用包裝金鑰。

此金鑰合約結構描述只能加密記錄。若要解密使用`EphemeralPrivateKeyToStaticPublicKey`金鑰合約結構描述加密的記錄，您必須使用以相同收件者公開金鑰設定的探索金鑰合約結構描述。若要解密，您可以使用 Raw ECDH 金鑰圈搭配金`PublicKeyDiscovery`鑰合約演算法，或者，如果收件者的公開金鑰來自非對稱金鑰合約 KMS 金鑰組，您可以使用 AWS KMS ECDH 金鑰圈搭配金鑰合約結構描述`KmsPublicKeyDiscovery`。

若要使用`EphemeralPrivateKeyToStaticPublicKey`金鑰合約結構描述初始化 Raw ECDH 金鑰圈，請提供下列值：

- 收件者的公開金鑰

  您必須提供收件者的 DER 編碼 X.509 公開金鑰，也稱為 SubjectPublicKeyInfo (SPKI)，如 RFC 5280 中所定義。

  您可以指定非對稱金鑰合約 KMS key pair 的公開金鑰，或從外部產生的 key pair 中指定公開金鑰 AWS。

- 曲線規格

  識別指定公開金鑰中的橢圓曲線規格。

在加密時，密鑰環在指定的曲線上創建一個新的密鑰對，並使用新的私鑰和指定的公鑰來導出共享包裝密鑰。

有效值：ECC_NIST_P256、ECC_NIS_P384、ECC_NIST_P512

Java

下列範例會使用EphemeralPrivateKeyToStaticPublicKey金鑰合約結構描述建立 Raw ECDH 金鑰圈。在加密時，密鑰環將在指定的ECC_NIST_P256曲線上本地創建一個新的密 key pair。

```
private static void EphemeralRawEcdhKeyring() {
    // Instantiate material providers
    final MaterialProviders materialProviders =
      MaterialProviders.builder()
        .MaterialProvidersConfig(MaterialProvidersConfig.builder().build())
        .build();

    ByteBuffer recipientPublicKey = getPublicKeyBytes();

    // Create the Raw ECDH ephemeral keyring
    final CreateRawEcdhKeyringInput ephemeralInput =
      CreateRawEcdhKeyringInput.builder()
        .curveSpec(ECDHCurveSpec.ECC_NIST_P256)
        .KeyAgreementScheme(
          RawEcdhStaticConfigurations.builder()
            .EphemeralPrivateKeyToStaticPublicKey(
              EphemeralPrivateKeyToStaticPublicKeyInput.builder()
                .recipientPublicKey(recipientPublicKey)
                .build()
            )
            .build()
        ).build();

    final IKeyring ephemeralKeyring =
 materialProviders.CreateRawEcdhKeyring(ephemeralInput);
}
```

PublicKeyDiscovery

解密時，最佳做法是指定 AWS 資料庫加密 SDK 可以使用的包裝金鑰。若要遵循此最佳做法，請使用指定寄件者私密金鑰和收件者公開金鑰的 ECDH 金鑰圈。不過，您也可以建立 Raw ECDH 探索金鑰圈，也就是 Raw ECDH 金鑰圈，可以解密指定金鑰的公開金鑰與加密記錄材料描述欄位中儲存之收件者公開金鑰相符的任何記錄。此金鑰合約結構描述只能解密記錄。

> **Important**
>
> 當您使用金`PublicKeyDiscovery`鑰合約結構描述解密記錄時，您會接受所有公開金鑰，不論其擁有者為何。

若要使用`PublicKeyDiscovery`金鑰合約結構描述初始化 Raw ECDH 金鑰圈，請提供下列值：

- 收件者的靜態私密金鑰

  您必須提供收件者的 PEM 編碼私密金鑰 (PKCS #8 PrivateKeyInfo 結構)，如 RFC 5958 中所定義。

- 曲線規格

  識別指定私密金鑰中的橢圓曲線規格。寄件者和收件者的金鑰配對必須具有相同的曲線規格。

  有效值：`ECC_NIST_P256`、`ECC_NIS_P384`、`ECC_NIST_P512`

Java

下列範例會使用`PublicKeyDiscovery`金鑰合約結構描述建立 Raw ECDH 金鑰圈。此金鑰圈可以解密指定私密金鑰的公開金鑰與儲存在加密記錄的材料描述欄位中的收件者公開金鑰相符的任何記錄。

```
private static void RawEcdhDiscovery() {
    // Instantiate material providers
    final MaterialProviders materialProviders =
      MaterialProviders.builder()
        .MaterialProvidersConfig(MaterialProvidersConfig.builder().build())
        .build();

    KeyPair recipient = GetRawEccKey();

    // Create the Raw ECDH discovery keyring
```

```
    final CreateRawEcdhKeyringInput rawKeyringInput =
      CreateRawEcdhKeyringInput.builder()
        .curveSpec(ECDHCurveSpec.ECC_NIST_P256)
        .KeyAgreementScheme(
          RawEcdhStaticConfigurations.builder()
            .PublicKeyDiscovery(
              PublicKeyDiscoveryInput.builder()
                // Must be a PEM-encoded private key

 .recipientStaticPrivateKey(ByteBuffer.wrap(sender.getPrivate().getEncoded()))
                .build()
            )
            .build()
        ).build();

    final IKeyring publicKeyDiscovery  =
 materialProviders.CreateRawEcdhKeyring(rawKeyringInput);
}
```

多重 keyring

我們的用戶端加密程式庫已重新命名為 AWS 資料庫加密 SDK。本開發人員指南仍提供 DynamoDB 加密用戶端的相關資訊。

您可以結合 keyring 成為多重 keyring。多重 keyring 是一種 keyring，其中包含相同或不同類型的一或多個個別 keyring。效果就像是使用系列中的數個 keyring。使用多重 keyring 來加密資料時，其任何 keyring 中的任何包裝金鑰均可以解密該資料。

建立多重 keyring 來加密資料時，您會指定其中一個 keyring 做為產生器 keyring。所有其他 keyring 稱為子 keyring。產生器 keyring 會產生並加密純文字資料金鑰。然後，所有子 keyring 中的所有包裝金鑰會加密相同的純文字資料金鑰。該多重 keyring 會為多重 keyring 中的每個包裝金鑰傳回純文字金鑰和一個加密的資料金鑰。如果產生器金鑰圈是 KMS 金鑰圈，金鑰圈中的產生器金鑰會產生並加密 AWS KMS 純文字金鑰。然後，鑰匙 AWS KMS 圈 AWS KMS keys 中的所有附加密鑰以及多密鑰環中所有子密鑰環中的所有包裝密鑰，對相同的明文密鑰進行加密。

解密時，資料 AWS 庫加密 SDK 會使用金鑰環嘗試解密其中一個加密的資料金鑰。按照在多重 keyring 中指定的順序呼叫 keyring。只要任何 keyring 中的任何金鑰可以解密已加密的資料金鑰，處理就會停止。

若要建立多重 keyring，請先將子 keyring 執行個體化。在此範例中，我們使用 AWS KMS 鑰匙圈和 Raw AES 鑰匙圈，但您可以將任何支援的鑰匙圈組合在多重鑰匙圈中。

Java

```
// 1. Create the raw AES keyring.
final MaterialProviders matProv = MaterialProviders.builder()
        .MaterialProvidersConfig(MaterialProvidersConfig.builder().build())
        .build();
final CreateRawAesKeyringInput createRawAesKeyringInput =
 CreateRawAesKeyringInput.builder()
        .keyName("AES_256_012")
        .keyNamespace("HSM_01")
        .wrappingKey(AESWrappingKey)
        .wrappingAlg(AesWrappingAlg.ALG_AES256_GCM_IV12_TAG16)
        .build();
IKeyring rawAesKeyring = matProv.CreateRawAesKeyring(createRawAesKeyringInput);

// 2. Create the AWS KMS keyring.
final CreateAwsKmsMrkMultiKeyringInput createAwsKmsMrkMultiKeyringInput =
 CreateAwsKmsMrkMultiKeyringInput.builder()
        .generator(kmsKeyArn)
        .build();
IKeyring awsKmsMrkMultiKeyring =
 matProv.CreateAwsKmsMrkMultiKeyring(createAwsKmsMrkMultiKeyringInput);
```

C# / .NET

```
// 1. Create the raw AES keyring.
var keyNamespace = "HSM_01";
var keyName = "AES_256_012";

var matProv = new MaterialProviders(new MaterialProvidersConfig());
var createRawAesKeyringInput = new CreateRawAesKeyringInput
{
    KeyName = "keyName",
    KeyNamespace = "myNamespaces",
    WrappingKey = AESWrappingKey,
    WrappingAlg = AesWrappingAlg.ALG_AES256_GCM_IV12_TAG16
};
var rawAesKeyring = matProv.CreateRawAesKeyring(createRawAesKeyringInput);

// 2. Create the AWS KMS keyring.
```

```
//    We create a MRK multi keyring, as this interface also supports
//    single-region KMS keys,
//    and creates the KMS client for us automatically.
var createAwsKmsMrkMultiKeyringInput = new CreateAwsKmsMrkMultiKeyringInput
{
    Generator = keyArn
};
var awsKmsMrkMultiKeyring =
 matProv.CreateAwsKmsMrkMultiKeyring(createAwsKmsMrkMultiKeyringInput);
```

接著，建立多重 keyring，並指定其產生器 keyring (如果有)。在此範例中，我們建立了一個多鑰匙圈，其中 AWS KMS 鑰匙圈是產生器金鑰圈，而 AES 鑰匙圈則是子鑰匙圈。

Java

Java `CreateMultiKeyringInput` 構造函數允許您定義生成器密鑰環和子密鑰環。生成的`createMultiKeyringInput`對象是不可變的。

```
final CreateMultiKeyringInput createMultiKeyringInput =
 CreateMultiKeyringInput.builder()
        .generator(awsKmsMrkMultiKeyring)
        .childKeyrings(Collections.singletonList(rawAesKeyring))
        .build();
IKeyring multiKeyring = matProv.CreateMultiKeyring(createMultiKeyringInput);
```

C# / .NET

.NET `CreateMultiKeyringInput` 建構函式可讓您定義產生器金鑰環和子金鑰環。生成的`CreateMultiKeyringInput`對象是不可變的。

```
var createMultiKeyringInput = new CreateMultiKeyringInput
{
    Generator = awsKmsMrkMultiKeyring,
    ChildKeyrings = new List<IKeyring> { rawAesKeyring }
};
var multiKeyring = matProv.CreateMultiKeyring(createMultiKeyringInput);
```

現在，您可以使用多重 keyring 來加密和解密資料。

可搜索加密

我們的用戶端加密程式庫已重新命名為AWS資料庫加密 SDK。本開發人員指南仍提供 DynamoDB 加密用戶端的相關資訊。

可搜索的加密使您無需解密整個數據庫即可搜索加密記錄。這是使用信標（Beacon）完成的，該信標會在寫入字段的純文本值和實際存儲在數據庫中的加密值之間創建一個映射。資AWS料庫加密 SDK 會將信標儲存在新增至記錄的新欄位中。根據您使用的信標類型，您可以對加密的資料執行完全比對搜尋或更自訂的複雜查詢。

> **Note**
>
> AWS資料庫加密 SDK 中的可搜尋加密與學術研究中定義的可搜尋對稱加密不同，例如可搜尋的對稱加密。

信標 (Beacon) 是截斷的雜湊型訊息驗證碼 (HMAC) 標記，可在欄位的純文字和加密值之間建立對應。當您將新值寫入設定為可搜尋加密的加密欄位時，AWS資料庫加密 SDK 會透過純文字值計算 HMAC。此 HMAC 輸出是該欄位的純文字值的一對一 (1:1) 相符項目。HMAC 輸出會被截斷，以便多個不同的明文值對應到相同的截斷 HMAC 標籤。這些誤報會限制未經授權的使用者識別有關純文字值的區別資訊的能力。當您查詢信標時，AWS資料庫加密 SDK 會自動篩選掉這些誤報，並傳回查詢的純文字結果。

針對每個信標產生的平均誤判數，取決於截斷後剩餘的信標長度。如需決定實作適當信標長度的說明，請參閱判斷信標長度。

> **Note**
>
> 可搜索的加密被設計為在新的，未填充的數據庫中實現。在現有資料庫中設定的任何信標只會對應上傳至資料庫的新記錄，信標無法對應現有資料。

主題

- 可搜尋加密案例

信標是否適合我的數據集？

使用信標對加密資料執行查詢，可降低與用戶端加密資料庫相關的效能成本。當您使用信標時，查詢的效率和有關數據分佈的信息量之間存在固有的權衡。信標不會改變欄位的加密狀態。當您使用AWS資料庫加密 SDK 加密和簽署欄位時，欄位的純文字值永遠不會公開給資料庫。資料庫會儲存欄位的隨機加密值。

信標與計算它們的加密字段一起存儲。這表示即使未經授權的使用者無法檢視加密欄位的純文字值，他們也可以對信標執行統計分析，進一步瞭解資料集的分佈情況，並在極端情況下識別信標對應的純文字值。您設定信標的方式可以減輕這些風險。特別是，選擇正確的信標長度可以幫助您保持數據集的機密性。

安全性與效能

- 信標長度越短，保留的安全性就越高。
- 信標長度越長，保留的效能就越多。

可搜尋的加密可能無法為所有資料集提供所需的效能和安全性等級。在設定任何信標之前，請先檢閱您的威脅模型、安全需求和效能需求。

判斷可搜尋的加密是否適合您的資料集時，請考量下列資料集唯一性需求。

發佈

> Beacon 保留的安全性量取決於資料集的分佈情況。當您設定可搜尋加密的加密欄位時，AWS資料庫加密 SDK 會根據寫入該欄位的純文字值來計算 HMAC。針對指定欄位計算的所有信標均使用相同的索引鍵計算，但對每個承租人使用不同索引鍵的多租戶資料庫除外。這表示如果將相同的純文字值寫入欄位多次，則會針對該純文字值的每個執行個體建立相同的 HMAC 標籤。
>
> 您應該避免從包含非常常見值的字段構建信標。例如，考慮一個存儲伊利諾伊州每個居民的地址的數據庫。如果您從加密`City`字段構建信標，則由於居住在芝加哥的伊利諾伊州人口的比例很大，因此在「芝加哥」上計算出的信標將被過度佔用。即使未經授權的使用者只能讀取加密的值和信標值，如果信標保留此分配，他們也許能夠識別哪些記錄包含芝加哥居民的資料。為了最大限度地減少顯示有關分佈的區別信息量，您必須充分截斷您的信標。隱藏這種不均勻分佈所需的信標長度具有可能無法滿足您應用程式需求的顯著效能成本。

您必須仔細分析數據集的分佈情況，以確定需要截斷多少信標。截斷後剩餘的信標長度與可識別的有關分佈的統計資訊量直接相關。您可能需要選擇較短的信標長度，以充分減少顯示有關資料集的區別資訊量。

在極端情況下，您無法計算分佈不均勻的資料集的信標長度，以有效平衡效能和安全性。例如，您不應從儲存罕見疾病醫學測試結果的欄位建構信標。由於預計NEGATIVE結果在數據集中顯著更加普遍，因此可以通過它們的稀有程度輕鬆識別POSITIVE結果。當欄位只有兩個可能的值時，隱藏分佈是非常具有挑戰性的。如果您使用的信標長度足以隱藏分佈，則所有純文字值都會對應至相同的 HMAC 標籤。如果使用較長的信標長度，很明顯哪個信標映射到純文POSITIVE本值。

相关性

我們強烈建議您避免從具有相關值的欄位建構不同的信標。從相關欄位建構的信標需要較短的信標長度，以便充分減少每個資料集散佈給未經授權使用者的相關資訊量。您必須仔細分析資料集，包括其熵和相關值的聯合分佈，以判斷需要截斷多少信標。如果產生的信標長度不符合您的效能需求，則信標可能不適合您的資料集。

例如，您不應該從City和ZIPCode欄位建構兩個不同的信標，因為郵遞區號可能只與一個城市產生關聯。通常，信標產生的誤報會限制未經授權的使用者識別資料集相關資訊的能力。但是City和ZIPCode欄位之間的關聯意味著未經授權的使用者可以輕鬆識別哪些結果是誤報，並區分不同的郵遞區號。

您也應該避免從包含相同純文字值的欄位建構信標。例如，您不應從mobilePhone和preferredPhone欄位建構信標，因為它們可能包含相同的值。如果您從這兩個欄位建構不同的信標，AWS資料庫加密 SDK 會在不同金鑰下為每個欄位建立信標。這會為相同的純文字值產生兩個不同的 HMAC 標籤。這兩個不同的信標不太可能具有相同的誤報，並且未經授權的用戶可能能夠區分不同的電話號碼。

即使您的資料集包含相關欄位或分佈不均，您也可以使用較短的信標長度來建構保護資料集機密性的信標。但是，Beacon length 並不保證資料集中的每個唯一值都會產生一些誤報，這些誤報可有效地減少資料集所揭露的區別資訊量。信標長度僅估計產生誤報的平均數量。您的資料集分佈越不均，信標長度的效率就越低，就是決定產生的誤報的平均數目。

仔細考慮從中構建信標的字段的分佈情況，並考慮截斷信標長度以滿足您的安全需求所需的數量。本章的下列主題假設您的信標均勻分佈，且不包含相關資料。

可搜尋加密案例

下列範例會示範簡單的可搜尋加密解決方案。在應用程式中，此範例中使用的範例欄位可能不符合信標的分佈和相互關聯唯一性建議。當您閱讀本章中可搜尋的加密概念時，您可以使用此範例作為參考。

考慮一個名Employees為跟踪公司員工數據的數據庫。在數據庫中的每個記錄包含稱為員工 ID，LastNameFirstName，和地址字段。Employees數據庫中的每個字段由主鍵標識EmployeeID。

以下是資料庫中的明文記錄範例。

```
{
    "EmployeeID": 101,
    "LastName": "Jones",
    "FirstName": "Mary",
    "Address": {
                "Street": "123 Main",
                "City": "Anytown",
                "State": "OH",
                "ZIPCode": 12345
    }
}
```

如果您ENCRYPT_AND_SIGN在加密動作中將LastName和FirstName欄位標示為，則這些欄位中的值在上傳到資料庫之前會先在本機加密。上傳的加密數據是完全隨機的，數據庫不會將此數據識別為受保護。它只是檢測典型的數據條目。這意味著實際存儲在數據庫中的記錄可能如下所示。

```
{
    "PersonID": 101,
    "LastName": "1d76e94a2063578637d51371b363c9682bad926cbd",
    "FirstName": "21d6d54b0aaabc411e9f9b34b6d53aa4ef3b0a35",
    "Address": {
                "Street": "123 Main",
                "City": "Anytown",
                "State": "OH",
                "ZIPCode": 12345
    }
}
```

如果您需要查詢資料庫LastName欄位中的完全相符項目，請設定名為的標準信標，LastName將寫入LastName欄位的純文字值對應至儲存在資料庫中的加密值。

此信標會根據欄位中的純文字值計算 HMAC。LastName每個 HMAC 輸出都會被截斷，使其不再與純文字值完全相符。例如，完整的雜湊值和截斷的雜湊值Jones可能如下所示。

完整雜湊

2aa4e9b404c68182562b6ec761fcca5306de527826a69468885e59dc36d0c3f824bdd44cab45526f

截斷散列

b35099d408c833

設定標準信標之後，您可以在LastName欄位上執行相等搜尋。例如，如果您要搜尋Jones，請使用LastName信標 (Beacon) 來執行下列查詢。

```
LastName = Jones
```

資AWS料庫加密 SDK 會自動篩選出誤報，並傳回查詢的純文字結果。

信標

> 我們的用戶端加密程式庫已重新命名為 AWS 資料庫加密 SDK。本開發人員指南仍提供 DynamoDB 加密用戶端的相關資訊。

信標 (Beacon) 是截斷的雜湊型訊息驗證碼 (HMAC) 標記，可在寫入欄位的純文字值與實際儲存在資料庫中的加密值之間建立對應。信標不會改變欄位的加密狀態。信標通過字段的純文本值計算 HMAC，並將其與加密值一起存儲。此 HMAC 輸出是該欄位的純文字值的一對一 (1:1) 相符項目。HMAC 輸出會被截斷，以便多個不同的明文值對應到相同的截斷 HMAC 標籤。這些誤報會限制未經授權的使用者識別有關純文字值的區別資訊的能力。

信標只能從標記ENCRYPT_AND_SIGN的字段或SIGN_AND_INCLUDE_IN_ENCRYPTION_CONTEXT在您的加密操作中構建。SIGN_ONLY信標本身並未簽署或加密。您無法建構含有標記欄位的信標DO_NOTHING。

您設定的信標類型會決定您可以執行的查詢類型。有兩種類型的信標支持可搜索的加密。標準信標執行相等搜尋。複合信標會結合常值純文字字串和標準信標，以執行複雜的資料庫作業。設定信標之後，您必須先設定每個信標的次要索引，才能搜尋加密的欄位。如需詳細資訊，請參閱 使用信標設定次要索引。

主題

- 標準信標
- 複合信標

標準信標

標準信標是在數據庫中實現可搜索加密的最簡單方法。它們只能針對單一加密或虛擬欄位執行相等搜尋。若要瞭解如何設定標準信標，請參閱設定標準信標。

建構標準信標的欄位稱為信標來源。它會識別信標需要對應之資料的位置。信標來源可以是加密欄位或虛擬欄位。每個標準信標中的信標來源必須是唯一的。您無法設定兩個具有相同信標來源的信標。

標準信標可用於對加密或虛擬欄位執行相等搜尋。或者，它們可用於構建複合信標以執行更複雜的數據庫操作。為了協助您組織及管理標準信標，Datab AWS ase Encryption SDK 提供下列選擇性信標樣式，以定義標準信標的預定用途。如需詳細資訊，請參閱定義信標樣式。

您可以建立對單一加密欄位執行相等搜尋的標準信標，也可以建立標準信標，透過建立虛擬欄位，在多個ENCRYPT_AND_SIGNSIGN_ONLY、和SIGN_AND_INCLUDE_IN_ENCRYPTION_CONTEXT欄位的串連上執行相等搜尋。

虛擬領域

虛擬欄位是由一或多個來源欄位建構的概念欄位。建立虛擬欄位不會將新欄位寫入記錄。虛擬欄位未明確儲存在資料庫中。它可用於標準信標組態，提供如何識別欄位的特定區段或串連記錄中的多個欄位以執行特定查詢的信標指示。虛擬欄位至少需要一個加密欄位。

> **Note**
>
> 下列範例會示範轉換的類型，以及您可以使用虛擬欄位執行的查詢。在應用程式中，此範例中使用的範例欄位可能不符合信標的分佈和相互關聯唯一性建議。

例如，如果您想要對FirstName和LastName欄位的串連執行相等搜尋，您可以建立下列其中一個虛擬欄位。

- 一個虛擬字NameTag段，由字段的第一個字母構成，後跟字LastName段，全部以小寫形式構成。FirstName此虛擬欄位可讓您進行查詢NameTag=mjones。

- 一個虛擬LastFirst字段，它是從LastName現場構建的，然後是字FirstName段。此虛擬欄位可讓您進行查詢LastFirst=JonesMary。

或者，如果您想要在加密欄位的特定區段上執行相等搜尋，請建立一個虛擬欄位來識別您要查詢的區段。

例如，如果您想要使用 IP 位址的前三個區段來查詢加密IPAddress欄位，請建立下列虛擬欄位。

- 一個虛擬IPSegment領域，從構造Segments('.', 0, 3)。此虛擬欄位可讓您進行查詢IPSegment=192.0.2。查詢會傳回以「192.0.2」開頭IPAddress值的所有記錄。

虛擬欄位必須是唯一的。無法從完全相同的來源欄位建構兩個虛擬欄位。

如需設定虛擬欄位及其使用它們的信標的說明，請參閱建立虛擬欄位。

複合信標

複合信標會建立可改善查詢效能的索引，並可讓您執行更複雜的資料庫作業。您可以使用複合信標來結合常值純文字字串和標準信標，以對加密記錄執行複雜的查詢，例如從單一索引查詢兩種不同的記錄類型，或查詢具有排序索引鍵的欄位組合。如需更多複合信標解決方案範例，請參閱選擇信標類型。

複合信標可以使用標準信標或標準信標和簽名字段的組合來構建。它們是從零件清單中建構的。所有複合信標都應包含一份加密零件清單，以識別信標中包含的ENCRYPT_AND_SIGN欄位。每個ENCRYPT_AND_SIGN欄位都必須由標準信標識。更複雜的複合信標也可能包括一份已簽署零件的清單，這些零件可識別信標中包含的一SIGN_ONLY或多個純文SIGN_AND_INCLUDE_IN_ENCRYPTION_CONTEXT字欄位，以及識別複合信標可組合欄位的所有可能方式的建構函式零件清單。

> **Note**
>
> 資 AWS 料庫加密 SDK 也支援可完全從純文字SIGN_ONLY和SIGN_AND_INCLUDE_IN_ENCRYPTION_CONTEXT欄位設定的已簽署信標。簽署的信標是一種複合信標，可在已簽署但未加密的欄位上編製索引和執行複雜查詢。如需詳細資訊，請參閱 建立簽署的信標。

如需設定複合信標的說明，請參閱設定複合信標。

您配置複合信標的方式決定了它可以執行的查詢類型。例如，您可以將某些加密和簽署的零件設定為選用，以便在查詢中獲得更大的彈性。如需複合信標可執行之查詢類型的詳細資訊，請參閱查詢信標。

規劃信標

我們的用戶端加密程式庫已重新命名為AWS資料庫加密 SDK。本開發人員指南仍提供 DynamoDB 加密用戶端的相關資訊。

信標的設計是要在新的、未填入的資料庫中實作。在現有資料庫中設定的任何信標只會對應寫入資料庫的新記錄。信標是根據欄位的純文字值計算的，一旦欄位經過加密，信標就無法對應現有資料。使用信標寫入新記錄之後，就無法更新信標的組態。不過，您可以為新增至記錄的新欄位新增信標。

若要實作可搜尋的加密，您必須使用AWS KMS階層式金鑰環來產生、加密和解密用於保護記錄的資料金鑰。如需詳細資訊，請參閱使用階層式金鑰圈進行可搜尋的加密。

在設定可搜尋加密的信標之前，您必須先檢閱加密需求、資料庫存取模式和威脅模型，以判斷資料庫的最佳解決方案。

您設定的信標類型會決定您可以執行的查詢類型。您在標準信標組態中指定的信標長度，會決定為指定信標產生的預期誤判數。強烈建議您在設定信標之前，先識別並規劃您需要執行的查詢類型。使用信標之後，就無法更新組態。

強烈建議您在設定任何信標之前，先檢閱並完成下列工作。

- 判斷信標是否適合您的資料集
- 選擇信標類型
- 選擇信標長度
- 選擇信標名稱

規劃資料庫的可搜尋加密解決方案時，請記住下列信標唯一性需求。

- 每個標準信標都必須有唯一的信標來源

  無法從相同的加密欄位或虛擬欄位建構多個標準信標。

  不過，單一標準信標可用來建構多個複合信標。
- 避免建立含有與現有標準信標重疊的來源欄位的虛擬欄位

  從虛擬欄位 (包含用來建立其他標準信標的來源欄位) 建構標準信標，可降低兩個信標的安全性。

  如需詳細資訊，請參閱虛擬欄位的安全性考量。

多租戶資料庫的考量

若要查詢在多租戶資料庫中設定的信標，您必須包含儲存branch-key-id與在查詢中加密記錄之承租人相關聯的欄位。您可以在定義信標 (Beacon) 主要來源時定義此欄位。若要成功查詢，此欄位中的值必須識別重新計算信標所需的適當信標主要原物料。

在設定信標之前，您必須決定如何在查詢branch-key-id中包含信標。如需有關可branch-key-id在查詢中包含的不同方式的詳細資訊，請參閱查詢多租戶資料庫中的信標。

選擇信標類型

> 我們的用戶端加密程式庫已重新命名為 AWS 資料庫加密 SDK。本開發人員指南仍提供 DynamoDB 加密用戶端的相關資訊。

透過可搜尋的加密，您可以透過將加密欄位中的明文值與信標對應，來搜尋加密的記錄。您設定的信標類型會決定您可以執行的查詢類型。

強烈建議您在設定信標之前，先識別並規劃您需要執行的查詢類型。設定信標之後，您必須先設定每個信標的次要索引，才能搜尋加密的欄位。如需詳細資訊，請參閱 使用信標設定次要索引。

信標會在寫入欄位的純文字值與實際儲存在資料庫中的加密值之間建立對應。您無法比較兩個標準信標的值，即使它們包含相同的基礎純文字也一樣。兩個標準信標將為相同的明文值產生兩個不同的 HMAC 標籤。因此，標準信標無法執行下列查詢。

- *beacon1* = *beacon2*
- *beacon1* IN (*beacon2*)
- *value* IN (*beacon1*, *beacon2*, ...)
- CONTAINS(*beacon1*, *beacon2*)

只有在比較複合信標的已簽署部分時，才能執行上述查詢，但CONTAINS運算子除外，您可以搭配複合信標使用該運算子來識別組合信標所包含之加密或已簽署欄位的整個值。比較已簽署零件時，您可以選擇性地包含加密零件的前置詞，但不能包括欄位的加密值。如需有關標準和複合信標可執行之查詢類型的詳細資訊，請參閱查詢信標。

檢閱資料庫存取模式時，請考慮下列可搜尋的加密解決方案。下列範例定義要設定哪個信標，以滿足不同的加密和查詢需求。

標準信標

標準信標只能執行相等搜尋。您可以使用標準信標來執行下列查詢。

查詢單一加密欄位

如果要識別包含加密欄位之特定值的記錄，請建立標準信標。

範例

在下列範例中，請考慮名UnitInspection為追蹤生產設施之檢驗資料的資料庫。資料庫中的每個記錄都包含稱為work_id、inspection_dateinspector_id_last4、和的欄位unit。完整的檢查員識別碼是介於 0 至 99,999,999 之間的數字。但是，為了確保數據集均勻分佈，inspector_id_last4只存儲檢查器 ID 的最後四位數。數據庫中的每個字段由主鍵標識work_id。inspector_id_last4和unit欄位會標示ENCRYPT_AND_SIGN在密碼編譯動作中。

以下是資料庫中純文字項目的範例。UnitInspection

```
{
    "work_id": "1c7fcff3-6e74-41a8-b7f7-925dc039830b",
    "inspection_date": 2023-06-07,
    "inspector_id_last4": 8744,
    "unit": 229304973450
}
```

查詢記錄中的單一加密欄位

> 如果inspector_id_last4欄位需要加密，但您仍需要查詢它是否完全相符，請從inspector_id_last4欄位建構標準信標。然後，使用標準信標建立次要索引。您可以使用此次要索引來查詢加密inspector_id_last4欄位。

如需設定標準信標的說明，請參閱設定標準信標。

查詢虛擬欄位

虛擬欄位是由一個或多個來源欄位建構的概念欄位。如果您想要針對加密欄位的特定區段執行相等搜尋，或對多個欄位的串連執行相等搜尋，請從虛擬欄位建構標準信標。所有虛擬欄位必須至少包含一個加密來源欄位。

範例

下列範例會建立Employees資料庫的虛擬欄位。以下是資料庫中的明文記錄範例。Employees

```
{
    "EmployeeID": 101,
    "SSN": 000-00-0000,
    "LastName": "Jones",
    "FirstName": "Mary",
    "Address": {
                "Street": "123 Main",
                "City": "Anytown",
                "State": "OH",
                "ZIPCode": 12345
    }
}
```

查詢加密欄位的區段

在此範例中，SSN欄位已加密。

如果您要使用社會安全號碼的最後四位數來查詢SSN欄位，請建立一個虛擬欄位來識別您計劃查詢的區段。

構造的虛擬Last4SSN字段Suffix(4)使您能夠查詢Last4SSN=0000。使用此虛擬欄位可建構標準信標。然後，使用標準信標建立次要索引。您可以使用此次要索引來查詢虛擬欄位。這個查詢返回與您指定的最後四個數字結束的SSN值的所有記錄。

查詢多個字段的連接

> **Note**
>
> 下列範例會示範轉換的類型，以及您可以使用虛擬欄位執行的查詢。在應用程式中，此範例中使用的範例欄位可能不符合信標的分佈和相互關聯唯一性建議。

如果您想要對FirstName和LastName欄位的串連執行相等搜尋，您可以建立一個虛擬欄位，從NameTag欄位的第一個字母建構，後面接著FirstName欄位，全部以小寫形式建構。LastName使用此虛擬欄位可建構標準信標。然後，使用標準信標建立次要索引。您可以使用此次要索引來查詢NameTag=mjones虛擬欄位。

必須至少加密其中一個來源欄位。FirstName或者LastName可以加密，或者它們都可以加密。任何純文字來源欄位都必須SIGN_AND_INCLUDE_IN_ENCRYPTION_CONTEXT在您的密碼編譯動作中標示為SIGN_ONLY或。

如需設定虛擬欄位及其使用這些欄位的信標的說明，請參閱建立虛擬欄位。

複合信標

複合信標會從常值純文字字串和標準信標建立索引，以執行複雜的資料庫作業。您可以使用複合信標來執行下列查詢。

在單一索引上查詢加密欄位的組合

如果您需要查詢單一索引上加密欄位的組合，請建立複合信標，該信標會結合為每個加密欄位建構的個別標準信標，以形成單一索引。

設定複合信標之後，您可以建立次要索引，將複合信標指定為執行完全相符查詢的分割區索引鍵，或使用排序索引鍵來執行更複雜的查詢。將複合信標指定為排序索引鍵的次要索引可以執行完全比對查詢和更多自訂的複雜查詢。

範例

對於下列範例，請考慮名UnitInspection為追蹤生產設施之檢驗資料的資料庫。資料庫中的每個記錄都包含稱為work_id、inspection_dateinspector_id_last4、和的欄位unit。完整的檢查員識別碼是介於 0 至 99,999,999 之間的數字。但是，為了確保數據集均勻分佈，inspector_id_last4只存儲檢查器 ID 的最後四位數。數據庫中的每個字段由主鍵標識work_id。inspector_id_last4和unit欄位會標示ENCRYPT_AND_SIGN在密碼編譯動作中。

以下是資料庫中純文字項目的範例。UnitInspection

```
{
    "work_id": "1c7fcff3-6e74-41a8-b7f7-925dc039830b",
    "inspection_date": 2023-06-07,
    "inspector_id_last4": 8744,
    "unit": 229304973450
}
```

對加密欄位的組合執行相等搜尋

如果您要查詢UnitInspection資料庫中的完全相符項目inspector_id_last4.unit，請先為inspector_id_last4和unit欄位建立不同的標準信標。然後，從兩個標準信標創建一個複合信標。

設定複合信標之後，請建立次要索引，將複合信標指定為分割區索引鍵。使用此次要索引來查詢上的完全相符項目inspector_id_last4.unit。例如，您可以查詢此信標，以尋找檢查員針對指定單位執行的檢驗清單。

對加密欄位的組合執行複雜的查詢

如果您要查詢和的UnitInspection資料庫inspector_id_last4.unit，請先為inspector_id_last4和unit欄位建立不同的inspector_id_last4標準信標。然後，從兩個標準信標創建一個複合信標。

設定複合信標之後，請建立次要索引，將複合信標指定為排序索引鍵。使用此次要索引來查詢UnitInspection資料庫中是否有以特定檢查器開頭的項目，或查詢資料庫中特定單位 ID 範圍內的所有單位清單，這些單位已由特定檢查器檢查。您也可以在上執行完全相符搜尋inspector_id_last4.unit。

如需設定複合信標的說明，請參閱設定複合信標。

在單一索引上查詢加密和純文字欄位的組合

如果您需要查詢單一索引上加密和純文字欄位的組合，請建立結合個別標準信標和純文字欄位的複合信標，以形成單一索引。用於建構複合信標的純文字欄位必須標記SIGN_ONLY或SIGN_AND_INCLUDE_IN_ENCRYPTION_CONTEXT在您的密碼編譯動作中。

設定複合信標之後，您可以建立次要索引，將複合信標指定為執行完全相符查詢的分割區索引鍵，或使用排序索引鍵來執行更複雜的查詢。將複合信標指定為排序索引鍵的次要索引可以執行完全比對查詢和更多自訂的複雜查詢。

範例

對於下列範例，請考慮名UnitInspection為追蹤生產設施之檢驗資料的資料庫。資料庫中的每個記錄都包含稱為work_id、inspection_dateinspector_id_last4、和的欄位unit。完整的檢查員識別碼是介於 0 至 99,999,999 之間的數字。但是，為了確保數據集均勻分佈，inspector_id_last4只存儲檢查器 ID 的最後四位數。數據庫中的每個字段由主鍵標識work_id。inspector_id_last4和unit欄位會標示ENCRYPT_AND_SIGN在密碼編譯動作中。

以下是資料庫中純文字項目的範例。UnitInspection

```
{
    "work_id": "1c7fcff3-6e74-41a8-b7f7-925dc039830b",
    "inspection_date": 2023-06-07,
    "inspector_id_last4": 8744,
    "unit": 229304973450
}
```

對欄位組合執行相等搜尋

如果您要查詢UnitInspection資料庫，以查詢特定檢查員在特定日期進行的檢查，請先為inspector_id_last4欄位建立標準信標。該inspector_id_last4字段在密碼編譯操作ENCRYPT_AND_SIGN中標記。所有加密零件都需要自己的標準信標。此inspection_date欄位已標記，SIGN_ONLY且不需要標準信標。接下來，從inspection_date現場和標準信標創建一個複合信inspector_id_last4標。

設定複合信標之後，請建立次要索引，將複合信標指定為分割區索引鍵。使用此次要索引來查詢資料庫中與特定檢查程式和檢查日期完全相符的記錄。例如，您可以查詢資料庫，以取得 ID 在特定日期結束的檢8744查員的所有檢查清單。

對欄位組合執行複雜的查詢

如果您要查詢資料庫是否在某個inspection_date範圍內進行的檢驗，或查詢資料庫是否有針對inspector_id_last4或inspection_date限制的特定執行的檢驗inspector_id_last4.unit，請先為inspector_id_last4和unit欄位建立不同的標準信標。然後，從純文inspection_date字欄位和兩個標準信標建立複合信標。

設定複合信標之後，請建立次要索引，將複合信標指定為排序索引鍵。使用此次要索引可針對特定檢查員在特定日期進行的檢查執行查詢。例如，您可以在資料庫中查詢在同一日期檢查的所有單位清單。或者，您可以查詢資料庫，以取得在指定檢驗日期範圍內，針對特定單位執行的所有檢驗清單。

如需設定複合信標的說明，請參閱設定複合信標。

選擇信標長度

> 我們的客戶端加密庫被重命名為AWS數據庫加密 SDK。本開發人員指南仍提供 DynamoDB 加密用戶端的相關資訊。

當您將新值寫入設定為可搜尋加密的加密欄位時，AWS資料庫加密 SDK 會透過純文字值計算 HMAC。此 HMAC 輸出是該欄位的純文字值的一對一 (1:1) 相符項目。HMAC 輸出會被截斷，以便多個不同的明文值對應到相同的截斷 HMAC 標籤。這些衝突或誤報會限制未經授權的使用者識別明文值相關資訊的能力。

針對每個信標產生的平均誤判數，取決於截斷後剩餘的信標長度。您只需在設定標準信標時定義信標長度。複合信標使用其構造的標準信標的信標長度。

信標不會改變欄位的加密狀態。但是，當您使用信標時，在查詢的效率和披露有關數據分佈的信息量之間存在固有的權衡。

可搜尋加密的目標是透過使用信標對加密資料執行查詢，以降低與用戶端加密資料庫相關的效能成本。信標與計算它們的加密字段一起存儲。這意味著它們可以顯示有關數據集分佈的區別信息。在極端情況下，未經授權的使用者可能能夠分析您發佈的相關資訊，並使用它來識別欄位的純文字值。選擇正確的信標長度有助於減輕這些風險，並保持分配的機密性。

檢閱您的威脅模型，以判斷所需的安全層級。例如，擁有資料庫存取權但無法存取純文字資料的人越多，您就越想要保護資料集散發的機密性。為了提高機密性，信標需要產生更多誤報。增加機密性會降低查詢效能。

安全性與效能

- 信標長度太長會產生太少誤報，並且可能會顯示有關資料集分佈的區別資訊。
- 信標長度太短會產生太多誤判，並增加查詢的效能成本，因為它需要對資料庫進行更廣泛的掃描。

在決定解決方案的適當信標長度時，您必須找到能夠充分保留資料安全性的長度，而不會影響查詢效能，超過絕對必要的範圍。Beacon 保留的安全性程度取決於資料集的分佈情況，以及建構信標所使用之欄位的關聯性。下列主題假設您的信標均勻分佈，且不包含相關資料。

主題

計算信標長度

信標長度以位定義，並指的是截斷後保留的 HMAC 標籤的位數。建議的信標長度會根據資料集分佈、相關值的存在，以及您的特定安全性和效能需求而有所不同。如果您的資料集均勻分佈，您可以使用下列方程式和程序來協助識別實作的最佳信標長度。這些方程式只會估計信標產生的誤報平均數量，並不保證資料集中的每個唯一值都會產生特定數量的誤報。

> **Note**
>
> 這些方程式的有效性取決於資料集的分佈。如果您的資料集未均勻分佈，請參閱。信標是否適合我的數據集？
> 一般而言，您的資料集越是來自統一分佈，您就越需要縮短信標長度。

1.

估計人口

人口是建構標準信標之欄位中預期的唯一值數目，而不是預期儲存在欄位中的值總數。例如，考慮一個識別員工會議位置的加密Room欄位。該Room欄位預計會儲存 100,000 個總值，但是員工只能預留 50 個不同的會議室供會議使用。這意味著人口是 50，因為只有 50 個可能的唯一值可以存儲在Room字段中。

> **Note**
>
> 如果您的標準信標是從虛擬欄位建構，用來計算信標長度的填入量就是虛擬欄位所建立的唯一組合數目。

估計人口時，請務必考慮資料集的預計增長情況。使用信標寫入新記錄之後，就無法更新信標長度。檢閱您的威脅模型和任何現有的資料庫解決方案，以建立預期此欄位在未來五年內儲存的唯一值數量的估算值。

您的人口不需要精確。首先，識別目前資料庫中唯一值的數目，或估計您預期在第一年儲存的唯一值數目。接下來，使用以下問題來幫助您確定未來五年內獨特價值的預計增長情況。

- 您是否希望唯一值乘以 10？
- 您是否希望唯一值乘以 100？
- 您是否希望唯一值乘以 1000？

50,000 和 60,000 個唯一值之間的差異並不重要，它們都會產生相同的建議信標長度。但是，50,000 到 500,000 個唯一值之間的差異將會對建議的信標長度產生重大影響。

請考慮檢閱一般資料類型頻率的公開資料，例如郵遞區號或姓氏。例如，在美國有 41,707 個郵遞區號。您使用的人口應與您自己的數據庫成正比。如果資料庫中的ZIPCode欄位包含來自整個美國的資料，則您可以將人口定義為 41,707，即使該ZIPCode欄位目前沒有 41,707 個唯一值。如果資料庫中的ZIPCode欄位只包含來自單一狀態的資料，而且只會包含單一州的資料，則您可以將人口定義為該州的郵遞區號總數，而不是 41,704。

2. 計算預期衝突次數的建議範圍

若要判斷指定欄位的適當信標長度，您必須先找出預期衝突次數的適當範圍。預期的衝突次數代表映射到特定 HMAC 標籤的唯一純文本值的平均預期數。一個唯一純文字值的預期誤判數目比預期的衝突數少一個。

我們建議預期的衝突次數大於或等於 2，且小於人口的平方根。只有當您的人口具有 16 個或更多唯一值時，以下方程式才有效。

```
2 ≤ number of collisions < √(Population)
```

如果碰撞次數少於兩個，信標會產生太少誤判。我們建議使用兩個作為預期衝突的最小數量，因為這意味著，平均而言，欄位中的每個唯一值都會透過對應到另一個唯一值來產生至少一個誤報。

3. 計算信標長度的建議範圍

   在確定預期衝突的最小和最大數量之後，請使用下列方程式來識別適當的信標長度範圍。

   $$number\ of\ collisions = Population * 2^{-(beacon\ length)}$$

   首先，求解預期衝突次數等於 2 的信標長度（建議的預期衝突次數下限）。

   $$2 = Population * 2^{-(beacon\ length)}$$

   然後，解決信標長度，其中預期的衝突次數等於人口的平方根（建議的預期衝突次數上限）。

   $$\sqrt{(Population)} = Population * 2^{-(beacon\ length)}$$

   我們建議將此方程式產生的輸出四捨五入到較短的信標長度。例如，如果方程式產生 15.6 的信標長度，我們建議將該值四捨五入為 15 位元，而不是四捨五入至 16 位元。

4. 選擇信標長度

   這些方程式只會為您的欄位識別建議的信標長度範圍。我們建議盡可能使用較短的信標長度來保護資料集的安全性。不過，您實際使用的信標長度取決於您的威脅模型。在檢閱威脅模型以判斷欄位的最佳信標長度時，請考慮效能需求。

   使用較短的信標長度可降低查詢效能，而使用較長的信標長度會降低安全性。一般而言，如果您的資料集分佈不均勻，或者您是從相關欄位建構不同的信標，您必須使用較短的信標長度，將資料集分佈的相關資訊量降到最低。

如果您檢閱您的威脅模型，並決定顯示的任何有關欄位分佈的區別資訊不會對您的整體安全性構成威脅，您可能會選擇使用比您計算的建議範圍長度還長的信標長度。例如，如果您將欄位的建議信標長度範圍計算為 9—16 位元，您可以選擇使用 24 位元的信標長度，以避免任何效能損失。

仔細選擇您的信標長度。使用信標寫入新記錄之後，就無法更新信標長度。

範例

考慮將該unit字段標記為ENCRYPT_AND_SIGN在密碼編譯操作的數據庫。若要設定unit欄位的標準信標，我們需要判斷unit欄位的預期誤判數和信標長度。

1. 估計人口

   在審查了我們的威脅模型和當前的數據庫解決方案之後，我們預計該unit領域最終具有 100,000 個唯一值。

   這意味著人口 = 10 萬。

2. 計算預期衝突次數的建議範圍。

   在此範例中，預期的衝突次數應介於 2—316 之間。

   ```
   2 ≤ number of collisions < √(Population)
   ```

   a.
   ```
   2 ≤ number of collisions < √(100,000)
   ```

   b.
   ```
   2 ≤ number of collisions < 316
   ```

3. 計算建議的信標長度範圍。

   在此範例中，信標長度應介於 9—16 位元之間。

   $$number\ of\ collisions = Population * 2^{-(beacon\ length)}$$

   a. 計算信標長度，其中預期的衝突次數等於步驟 2 中識別的最小值。

      $$2 = 100{,}000 * 2^{-(beacon\ length)}$$

      信標長度 = 15.6 或 15 位元

b. 計算信標長度，其中預期的衝突次數等於步驟 2 中識別的最大值。

```
316 = 100,000 * 2^-(beacon length)
```

信標長度 = 8.3 或 8 位元

4. 決定適合您安全性和效能需求的信標長度。

對於每一位低於 15，性能成本和安全性翻倍。

- 16 位元
  - 平均而言，每個唯一值將映射到另外 1.5 個單位。
  - 安全性：具有相同截斷 HMAC 標籤的兩個記錄有 66% 可能具有相同的純文本值。
  - 效能：查詢會針對您實際要求的每 10 筆記錄擷取 15 筆記錄。
- 十四位元
  - 平均而言，每個唯一值將映射到其他 6.1 個單位。
  - 安全性：具有相同截斷 HMAC 標籤的兩個記錄 33% 可能具有相同的純文本值。
  - 性能：查詢將為您實際請求的每 10 條記錄檢索 30 條記錄。

選擇信標名稱

> 我們的用戶端加密程式庫已重新命名為AWS資料庫加密 SDK。本開發人員指南仍提供 DynamoDB 加密用戶端的相關資訊。

每個信標都是以唯一的信標名稱來識別。設定信標之後，信標名稱就是您在查詢加密欄位時使用的名稱。信標名稱可以與加密欄位或虛擬欄位相同的名稱，但不能與未加密欄位的名稱相同。兩個不同的信標不能具有相同的信標名稱。

如需示範如何命名及設定信標的範例，請參閱設定信標。

命名標準信標

命名標準信標時，我們強烈建議您盡可能將信標名稱解析為信標來源。這表示建構標準信標的所在加密或虛擬欄位的信標名稱和名稱相同。例如，如果您要為名為的加密欄位建立標準信標`LastName`，您的信標名稱也應該是`LastName`。

當您的信標名稱與信標來源相同時，您可以省略組態中的信標來源，而「AWS資料庫加密 SDK」會自動使用信標名稱作為信標來源。

設定信標

> 我們的用戶端加密程式庫已重新命名為 AWS 資料庫加密 SDK。本開發人員指南仍提供 DynamoDB 加密用戶端的相關資訊。

有兩種類型的信標支持可搜索的加密。標準信標執行相等搜尋。它們是在數據庫中實現可搜索加密的最簡單方法。複合信標結合常值純文字字串和標準信標，以執行更複雜的查詢。

信標的設計是要在新的、未填入的資料庫中實作。在現有資料庫中設定的任何信標只會對應寫入資料庫的新記錄。信標是根據欄位的純文字值計算的，一旦欄位經過加密，信標就無法對應現有資料。使用信標寫入新記錄之後，就無法更新信標的組態。不過，您可以為新增至記錄的新欄位新增信標。

決定存取模式之後，設定信標應該是實作資料庫的第二個步驟。然後，在設定所有信標之後，您需要建立AWS KMS 階層式金鑰環、定義信標版本、為每個信標設定次要索引、定義加密動作，以及設定資料 AWS 庫和 Database Encryption SDK 用戶端。如需詳細資訊，請參閱使用信標。

為了更容易定義信標版本，我們建議您建立標準和複合信標的清單。在設定時，將您建立的每個信標新增至個別的標準或複合信標清單。

主題

- 設定標準信標
- 設定複合信標
- 範例組態

設定標準信標

標準信標是在數據庫中實現可搜索加密的最簡單方法。它們只能針對單一加密或虛擬欄位執行相等搜尋。

設定語法範例

Java

```
List<StandardBeacon> standardBeaconList = new ArrayList<>();
```

```
StandardBeacon exampleStandardBeacon = StandardBeacon.builder()
    .name("beaconName")
    .length(beaconLengthInBits)
    .build();
standardBeaconList.add(exampleStandardBeacon);
```

C# / .NET

```
var standardBeaconList = new List<StandardBeacon>();
StandardBeacon exampleStandardBeacon = new StandardBeacon
  {
    Name = "beaconName",
    Length = 10
  };
standardBeaconList.Add(exampleStandardBeacon);
```

若要設定標準信標，請提供下列值。

信標名稱

查詢加密欄位時使用的名稱。

信標名稱可以與加密欄位或虛擬欄位相同的名稱，但不能與未加密欄位的名稱相同。我們強烈建議盡可能使用標準信標建構的加密欄位或虛擬欄位的名稱。兩個不同的信標不能具有相同的信標名稱。如需決定實作之最佳信標名稱的說明，請參閱選擇信標名稱。

信標長度

截斷之後保留的信標雜湊值位元數目。

信標長度會決定指定信標產生誤報的平均數目。如需詳細資訊並協助判斷實作的適當信標長度，請參閱判斷信標長度。

信標來源 (選擇性)

建構標準信標的欄位。

信標來源必須是欄位名稱或參照巢狀欄位值的索引。當您的信標名稱與信標來源相同時，您可以省略組態中的信標來源， AWS 資料庫加密 SDK 會自動使用信標名稱作為信標來源。

建立虛擬欄位

若要建立虛擬欄位，您必須提供虛擬欄位的名稱和來源欄位清單。將來源欄位加入至虛擬零件表的順序決定了連接來源欄位以建置虛擬欄位的順序。下列範例將兩個來源欄位全部連結起來，以建立虛擬欄位。

> **Note**
>
> 我們建議您在填入資料庫之前，先確認您的虛擬欄位是否會產生預期的結果。如需詳細資訊，請參閱測試信標輸出。

Java

請參閱完整的代碼示例：VirtualBeaconSearchableEncryptionExample.java

```
List<VirtualPart> virtualPartList = new ArrayList<>();
    virtualPartList.add(sourceField1);
    virtualPartList.add(sourceField2);

VirtualField virtualFieldName = VirtualField.builder()
    .name("virtualFieldName")
    .parts(virtualPartList)
    .build();

List<VirtualField> virtualFieldList = new ArrayList<>();
    virtualFieldList.add(virtualFieldName);
```

C# / .NET

請參閱完整的代碼示例：VirtualBeaconSearchableEncryptionExample.cs

```
var virtualPartList = new List<VirtualPart> { sourceField1, sourceField2 };

var virtualFieldName = new VirtualField
{
    Name = "virtualFieldName",
    Parts = virtualPartList
};

var virtualFieldList = new List<VirtualField> { virtualFieldName };
```

若要使用來源欄位的特定區段建立虛擬欄位，您必須先定義該轉換，然後再將來源欄位新增至虛擬零件清單。

虛擬欄位的安全性考量

信標不會改變欄位的加密狀態。但是，當您使用信標時，在查詢的效率和披露有關數據分佈的信息量之間存在固有的權衡。您設定信標的方式會決定該信標所保留的安全層級。

避免建立含有與現有標準信標重疊的來源欄位的虛擬欄位。建立包含已用來建立標準信標之來源欄位的虛擬欄位，可降低兩個信標的安全層次。安全性降低的程度取決於其他源字段添加的熵級別。熵層級取決於其他來源欄位中唯一值的分佈，以及其他來源欄位對虛擬欄位整體大小所貢獻的位元數。

您可以使用人口和信標長度來判斷虛擬欄位的來源欄位是否保留資料集的安全性。人口是一個字段中唯一值的預期數量。您的人口不需要精確。有關估計字段人口的幫助，請參閱估計人口。

檢閱虛擬欄位的安全性時，請考慮下列範例。

- 信標 1 是從構造的。FieldA FieldA人口大於 $2^{(\text{信標 1 長度})}$。
- 信標 2 是從構造的VirtualField，它是從FieldA，FieldBFieldC，和構造。FieldD在一起FieldB，FieldC，並且FieldD有一個人口大於 2^{N}

如果下列陳述式為真，則信標 2 會保留信標 1 和信標 2 的安全性：

```
N ≥ (Beacon1 length)/2
```

以及

```
N ≥ (Beacon2 length)/2
```

定義信標樣式

標準信標可用於對加密或虛擬欄位執行相等搜尋。或者，它們可用於構建複合信標以執行更複雜的數據庫操作。為了協助您組織及管理標準信標，Datab AWS ase Encryption SDK 提供下列選擇性信標樣式，以定義標準信標的預定用途。

> **Note**
>
> 若要定義信標樣式，您必須使用 3.2 版或更新版本的 AWS 資料庫加密 SDK。在將信標樣式新增至信標組態之前，請先將新版本部署至所有讀取器。

PartOnly

定義為的標準信標只PartOnly能用來定義複合信標的加密部分。您無法直接查詢標PartOnly準信標。

Java

```
List<StandardBeacon> standardBeaconList = new ArrayList<>();
StandardBeacon exampleStandardBeacon = StandardBeacon.builder()
    .name("beaconName")
    .length(beaconLengthInBits)
    .style(
        BeaconStyle.builder()
           .partOnly(PartOnly.builder().build())
        .build()
    )
    .build();
standardBeaconList.add(exampleStandardBeacon);
```

C# /.

```
new StandardBeacon
{
    Name = "beaconName",
    Length = beaconLengthInBits,
    Style = new BeaconStyle
    {
        PartOnly = new PartOnly()
    }
}
```

Shared

根據預設，每個標準信標都會產生一個唯一的 HMAC 金鑰來計算信標。因此，您無法對來自兩個不同標準信標的加密欄位執行相等搜尋。定義為Shared使用其他標準信標的 HMAC 金鑰進行計算的標準信標。

例如，如果您需要將beacon1欄位與欄位進beacon2行比較，請定義beacon2為使用 HMAC 金鑰進行計算beacon1的Shared信標。

> **Note**
>
> 設定任何Shared信標之前，請考慮您的安全性和效能需求。 Shared信標可能會增加可識別的有關數據集分佈的統計信息量。例如，它們可能會顯示哪些共用欄位包含相同的純文字值。

Java

```
List<StandardBeacon> standardBeaconList = new ArrayList<>();
StandardBeacon exampleStandardBeacon = StandardBeacon.builder()
    .name("beacon2")
    .length(beaconLengthInBits)
    .style(
        BeaconStyle.builder()
           .shared(Shared.builder().other("beacon1").build())
        .build()
    )
    .build();
standardBeaconList.add(exampleStandardBeacon);
```

C # /.

```
new StandardBeacon
{
    Name = "beacon2",
    Length = beaconLengthInBits,
    Style = new BeaconStyle
    {
        Shared = new Shared { Other = "beacon1" }
    }
}
```

AsSet

根據預設，如果欄位值是集合， AWS 資料庫加密 SDK 會計算集合的單一標準信標。因此，您無法執行加密欄位的查詢CONTAINS(*a*, :*value*)。*a*標準信標 (Beacon) 定義為AsSet計算集合中每個個別元素的個別標準信標值，並以集合形式儲存在項目中的信標值。這可讓資 AWS 料庫加密 SDK 執行查詢CONTAINS(*a*, :*value*)。

若要定義AsSet標準信標，集合中的元素必須來自相同的人口，以便它們都可以使用相同的信標長度。如果在計算信標值時發生衝突，信標集合的元素可能會少於純文字集。

> **Note**
>
> 設定任何AsSet信標之前，請考慮您的安全性和效能需求。 AsSet信標可能會增加可識別的有關數據集分佈的統計信息量。例如，它們可能會顯示明文集的大小。

Java

```
List<StandardBeacon> standardBeaconList = new ArrayList<>();
StandardBeacon exampleStandardBeacon = StandardBeacon.builder()
    .name("beaconName")
    .length(beaconLengthInBits)
    .style(
        BeaconStyle.builder()
           .asSet(AsSet.builder().build())
        .build()
    )
    .build();
standardBeaconList.add(exampleStandardBeacon);
```

C# /.

```
new StandardBeacon
{
    Name = "beaconName",
    Length = beaconLengthInBits,
    Style = new BeaconStyle
    {
        AsSet = new AsSet()
    }
}
```

SharedSet

定義為SharedSet結合Shared和AsSet函數的標準信標，以便您可以對集合和欄位的加密值執行相等搜尋。這使得 AWS 數據庫加密 SDK 能夠執行查詢，CONTAINS(*a*, *b*)其中*a*是加密集，並且*b*是一個加密的字段。

> **Note**
>
> 設定任何Shared信標之前，請考慮您的安全性和效能需求。 SharedSet信標可能會增加可識別的有關數據集分佈的統計信息量。例如，它們可能會顯示純文字集的大小，或哪些共用欄位包含相同的純文字值。

Java

```
List<StandardBeacon> standardBeaconList = new ArrayList<>();
StandardBeacon exampleStandardBeacon = StandardBeacon.builder()
    .name("beacon2")
    .length(beaconLengthInBits)
    .style(
        BeaconStyle.builder()
           .sharedSet(SharedSet.builder().other("beacon1").build())
        .build()
    )
    .build();
standardBeaconList.add(exampleStandardBeacon);
```

C# /.

```
new StandardBeacon
{
    Name = "beacon2",
    Length = beaconLengthInBits,
    Style = new BeaconStyle
    {
        SharedSet = new SharedSet { Other = "beacon1" }
    }
}
```

設定複合信標

複合信標會結合常值純文字字串和標準信標，以執行複雜的資料庫作業，例如從單一索引查詢兩種不同的記錄類型，或使用排序索引鍵查詢欄位組合。複合信標可以從ENCRYPT_AND_SIGNSIGN_ONLY、和SIGN_AND_INCLUDE_IN_ENCRYPTION_CONTEXT字段構造。您必須為複合信標中包含的每個加密欄位建立標準信標。

> **Note**
>
> 我們建議您在填入資料庫之前，先確認您的複合信標產生預期的結果。如需詳細資訊，請參閱測試信標輸出。

設定語法範例

Java

複合信標配置

下列範例會在複合信標組態中於本機定義加密和已簽署的零件清單。

```
List<CompoundBeacon> compoundBeaconList = new ArrayList<>();
CompoundBeacon exampleCompoundBeacon = CompoundBeacon.builder()
    .name("compoundBeaconName")
    .split(".")
    .encrypted(encryptedPartList)
    .signed(signedPartList)
    .constructors(constructorList)
    .build();
compoundBeaconList.add(exampleCompoundBeacon);
```

信標版本定義

下列範例會在信標版本中全域定義加密和已簽署的零件清單。如需有關定義信標版本的詳細資訊，請參閱使用信標。

```
 List<BeaconVersion> beaconVersions = new ArrayList<>();
beaconVersions.add(
    BeaconVersion.builder()
        .standardBeacons(standardBeaconList)
        .compoundBeacons(compoundBeaconList)
        .encryptedParts(encryptedPartList)
        .signedParts(signedPartList)
        .version(1) // MUST be 1
        .keyStore(keyStore)
        .keySource(BeaconKeySource.builder()
            .single(SingleKeyStore.builder()
                .keyId(branchKeyId)
```

```
                .cacheTTL(6000)
                .build())
            .build())
        .build()
);
```

C# / .NET

請參閱完整的程式碼範例：BeaconConfig.cs

複合信標配置

下列範例會在複合信標組態中於本機定義加密和已簽署的零件清單。

```
var compoundBeaconList = new List<CompoundBeacon>();
var exampleCompoundBeacon = new CompoundBeacon
 {
    Name = "compoundBeaconName",
    Split = ".",
    Encrypted = encryptedPartList,
    Signed = signedPartList,
    Constructors = constructorList
 };
compoundBeaconList.Add(exampleCompoundBeacon);
```

信標版本定義

下列範例會在信標版本中全域定義加密和已簽署的零件清單。如需有關定義信標版本的詳細資訊，請參閱使用信標。

```
var beaconVersions = new List<BeaconVersion>
{
    new BeaconVersion
    {
        StandardBeacons = standardBeaconList,
        CompoundBeacons = compoundBeaconList,
        EncryptedParts = encryptedPartsList,
        SignedParts = signedPartsList,
        Version = 1, // MUST be 1
        KeyStore = keyStore,
        KeySource = new BeaconKeySource
        {
            Single = new SingleKeyStore
```

```
            {
                KeyId = branchKeyId,
                CacheTTL = 6000
            }
        }
    }
};
```

您可以在本機或全域定義的清單中定義加密零件和已簽署零件。我們建議盡可能在信標版本的全域清單中定義加密和簽署的零件。透過全域定義加密和簽署的零件，您可以定義每個零件一次，然後在多個複合信標 (Beacon) 組態中重複使用零件。如果您只打算使用一次加密或已簽署的零件，則可以在複合信標組態的本機清單中定義該零件。您可以在構造函數列表中引用本地和全局部分。

如果您全域定義加密和已簽署的零件清單，則必須提供建構函式零件清單，以識別複合信標在複合信標組態中組合欄位的所有可能方式。

> **Note**
>
> 若要全域定義加密和已簽署的零件清單，您必須使用 3.2 版或更新版本的 AWS 資料庫加密 SDK。在全域定義任何新零件之前，請先將新版本部署至所有讀取器。
> 您無法更新現有信標組態，以全域定義加密和已簽署的零件清單。

若要設定複合信標，請提供下列值。

信標名稱

查詢加密欄位時使用的名稱。

信標名稱可以與加密欄位或虛擬欄位相同的名稱，但不能與未加密欄位的名稱相同。沒有兩個信標可以具有相同的信標名稱。如需決定實作之最佳信標名稱的說明，請參閱選擇信標名稱。

分割字元

用於分隔組成複合信標的部分的字符。

分割字元不能出現在複合信標所建構之任何欄位的純文字值中。

加密零件清單

識別複合信標中包含的ENCRYPT_AND_SIGN欄位。

每個零件都必須包含名稱和字首。零件名稱必須是從加密欄位建構的標準信標的名稱。前綴可以是任何字符串，但它必須是唯一的。加密零件不能具有與已簽署零件相同的字首。我們建議使用短值來區分零件與複合信標服務的其他零件。

我們建議您盡可能在全域定義加密零件。如果您只打算在一個複合信標中使用加密零件，則可以考慮在本機定義加密零件。本機定義的加密零件不能具有與全域定義的加密零件相同的字首或名稱。

Java

```
List<EncryptedPart> encryptedPartList = new ArrayList<>);
EncryptedPart encryptedPartExample = EncryptedPart.builder()
    .name("standardBeaconName")
    .prefix("E-")
    .build();
encryptedPartList.add(encryptedPartExample);
```

C# / .NET

```
var encryptedPartList = new List<EncryptedPart>();
var encryptedPartExample = new EncryptedPart
 {
    Name = "compoundBeaconName",
    Prefix = "E-"
 };
encryptedPartList.Add(encryptedPartExample);
```

已簽署零件清單

識別複合信標中包含的已簽署欄位。

> **Note**
>
> 簽署的零件是選用的。您可以設定不參照任何已簽署零件的複合信標。

每個零件都必須包含名稱、來源和字首。來源是零件識別的SIGN_ONLY或SIGN_AND_INCLUDE_IN_ENCRYPTION_CONTEXT欄位。來源必須是欄位名稱或參照巢狀欄位值的索引。如果您的零件名稱識別來源，您可以省略來源， AWS 資料庫加密 SDK 會自動使用該名稱作為其來源。建議您盡可能將來源指定為零件名稱。前綴可以是任何字符串，但它

必須是唯一的。簽署的零件不能具有與加密零件相同的字首。我們建議使用短值來區分零件與複合信標服務的其他零件。

我們建議您盡可能在全域定義已簽署的零件。如果您只打算在一個複合信標中使用已簽署零件，則可以考慮在本機定義已簽署零件。本機定義的已簽署零件不能具有與全域定義的已簽署零件相同的字首或名稱。

Java

```
List<SignedPart> signedPartList = new ArrayList<>);
SignedPart signedPartExample = SignedPart.builder()
    .name("signedFieldName")
    .prefix("S-")
    .build();
signedPartList.add(signedPartExample);
```

C# / .NET

```
var signedPartsList = new List<SignedPart>
{
    new SignedPart { Name = "signedFieldName1", Prefix = "S-" },
    new SignedPart { Name = "signedFieldName2", Prefix = "SF-" }
};
```

構造器列表

識別建構函式，這些建構函式定義複合信標可以組裝加密和已簽署零件的不同方式。您可以在構造函數列表中引用本地和全局部分。

如果您從全局定義的加密和簽名部分構造複合信標，則必須提供構造函數列表。

如果您不使用任何全局定義的加密或簽名部分來構建複合信標，則構造函數列表是可選的。如果您未指定建構函式清單， AWS 資料庫加密 SDK 會使用下列預設建構函式組合複合信標。

- 所有已簽署零件的順序，均按其新增至已簽署零件清單的順序
- 所有加密零件均按其新增至加密零件清單的順序
- 所有零件皆為必填

建構函式

每個構造函數都是構造函數部分的有序列表，它定義了可以組裝複合信標的一種方式。構造函數部分按照它們被添加到列表中的順序連接在一起，每個部分由指定的拆分字符分隔。

每個構造函數部分命名一個加密的部分或一個簽名的部分，並定義構造函數中該部分是必需的還是可選的。例如，如果您想在、和Field1.Field2.Field3上查詢複合信標Field1Field1.Field2，請將 and 標記Field3為可選Field2並創建一個構造函數。

每個構造函數必須至少有一個必需的部分。我們建議在每個所需的構造函數中進行第一部分，以便您可以在查詢中使用BEGINS_WITH運算符。

如果記錄中存在其所有必需的部分，則構造函數成功。當您撰寫新記錄時，複合信標會使用建構函式清單來判斷是否可以從提供的值組合信標。它嘗試按照構造函數添加到構造函數列表的順序組裝信標，並使用成功的第一個構造函數。如果沒有建構函式成功，就不會將信標寫入記錄。

所有讀者和作者都應該指定相同的構造函數順序，以確保他們的查詢結果是正確的。

請使用下列程序來指定您自己的建構函式清單。

1. 為每個加密零件和已簽署的零件建立建構函式零件，以定義是否需要該零件。

   構造函數部分名稱必須是它表示的標準信標或帶符號字段的名稱。

   Java

   ```
   ConstructorPart field1ConstructorPart = ConstructorPart.builder()
           .name("Field1")
           .required(true)
           .build();
   ```

   C# / .NET

   ```
   var field1ConstructorPart = new ConstructorPart { Name = "Field1", Required
    = true };
   ```

2. 為可以使用您在步驟 1 中創建的構造函數部件組裝複合信標的每種可能方式創建構函數。

   例如，如果你想查詢Field1.Field2.Field3和Field4.Field2.Field3，那麼你必須創建兩個構造函數。 Field1並且都Field4可以是必需的，因為它們是在兩個單獨的構造函數中定義的。

   Java

   ```
   // Create a list for Field1.Field2.Field3 queries
   List<ConstructorPart> field123ConstructorPartList = new ArrayList<>();
   field123ConstructorPartList.add(field1ConstructorPart);
   ```

```
field123ConstructorPartList.add(field2ConstructorPart);
field123ConstructorPartList.add(field3ConstructorPart);
Constructor field123Constructor = Constructor.builder()
        .parts(field123ConstructorPartList)
        .build();
// Create a list for Field4.Field2.Field1 queries
List<ConstructorPart> field421ConstructorPartList = new ArrayList<>();
field421ConstructorPartList.add(field4ConstructorPart);
field421ConstructorPartList.add(field2ConstructorPart);
field421ConstructorPartList.add(field1ConstructorPart);
Constructor field421Constructor = Constructor.builder()
        .parts(field421ConstructorPartList)
        .build();
```

C# / .NET

```
// Create a list for Field1.Field2.Field3 queries
 var field123ConstructorPartList = new Constructor
{
    Parts = new List<ConstructorPart> { field1ConstructorPart,
 field2ConstructorPart, field3ConstructorPart }
};
// Create a list for Field4.Field2.Field1 queries
var field421ConstructorPartList = new Constructor
{
    Parts = new List<ConstructorPart> { field4ConstructorPart,
 field2ConstructorPart, field1ConstructorPart }
};
```

3. 創建一個構造函數列表，其中包含您在步驟 2 中創建的所有構造函數。

   Java

   ```
   List<Constructor> constructorList = new ArrayList<>();
   constructorList.add(field123Constructor)
   constructorList.add(field421Constructor)
   ```

   C# / .NET

   ```
   var constructorList = new List<Constructor>
   {
       field123Constructor,
       field421Constructor
   ```

```
};
```

4. 指定建constructorList立複合信標的時間。

範例組態

> 我們的用戶端加密程式庫已重新命名為 AWS 資料庫加密 SDK。本開發人員指南仍提供 DynamoDB 加密用戶端的相關資訊。

下列範例示範如何設定標準和複合信標。下列組態不提供信標長度。如需決定組態適當信標長度的說明，請參閱選擇信標長度。

若要查看示範如何設定和使用信標的完整程式碼範例，請參閱上的 aws-database-encryption-sdk-dynamodb 儲存庫中的 Java 和 .NET 可搜尋加密範例。 GitHub

主題

- 標準信標
- 複合信標

標準信標

如果您要查詢inspector_id_last4欄位中的完全相符項目，請使用下列組態建立標準信標。

Java

```
List<StandardBeacon> standardBeaconList = new ArrayList<>();
StandardBeacon exampleStandardBeacon = StandardBeacon.builder()
    .name("inspector_id_last4")
    .length(beaconLengthInBits)
    .build();
standardBeaconList.add(exampleStandardBeacon);
```

C# / .NET

```
var standardBeaconList = new List<StandardBeacon>>);
StandardBeacon exampleStandardBeacon = new StandardBeacon
  {
    Name = "inspector_id_last4",
```

```
    Length = 10
  };
standardBeaconList.Add(exampleStandardBeacon);
```

複合信標

如果要在inspector_id_last4和上查詢資UnitInspection料庫inspector_id_last4.unit，請使用下列組態建立複合信標。此複合信標僅需要加密的零件。

Java

```
// 1. Create standard beacons for the inspector_id_last4 and unit fields.
List<StandardBeacon> standardBeaconList = new ArrayList<>);
StandardBeacon inspectorBeacon = StandardBeacon.builder()
    .name("inspector_id_last4")
    .length(beaconLengthInBits)
    .build();
standardBeaconList.add(inspectorBeacon);

StandardBeacon unitBeacon = StandardBeacon.builder()
    .name("unit")
    .length(beaconLengthInBits)
    .build();
standardBeaconList.add(unitBeacon);

// 2. Define the encrypted parts.
List<EncryptedPart> encryptedPartList = new ArrayList<>);

// Each encrypted part needs a name and prefix
// The name must be the name of the standard beacon
// The prefix must be unique
// For this example we use the prefix "I-" for "inspector_id_last4"
// and "U-" for "unit"
EncryptedPart encryptedPartInspector = EncryptedPart.builder()
    .name("inspector_id_last4")
    .prefix("I-")
    .build();
encryptedPartList.add(encryptedPartInspector);

EncryptedPart encryptedPartUnit = EncryptedPart.builder()
    .name("unit")
```

```
    .prefix("U-")
    .build();
encryptedPartList.add(encryptedPartUnit);

// 3. Create the compound beacon.
// This compound beacon only requires a name, split character,
// and list of encrypted parts
CompoundBeacon inspectorUnitBeacon = CompoundBeacon.builder()
    .name("inspectorUnitBeacon")
    .split(".")
    .sensitive(encryptedPartList)
    .build();
```

C# / .NET

```
// 1. Create standard beacons for the inspector_id_last4 and unit fields.
StandardBeacon inspectorBeacon = new StandardBeacon
 {
   Name = "inspector_id_last4",
   Length = 10
 };
standardBeaconList.Add(inspectorBeacon);
StandardBeacon unitBeacon = new StandardBeacon
 {
    Name = "unit",
    Length = 30
 };
standardBeaconList.Add(unitBeacon);

// 2. Define the encrypted parts.
var last4EncryptedPart = new EncryptedPart

// Each encrypted part needs a name and prefix
// The name must be the name of the standard beacon
// The prefix must be unique
// For this example we use the prefix "I-" for "inspector_id_last4"
// and "U-" for "unit"
var last4EncryptedPart = new EncryptedPart
 {
   Name = "inspector_id_last4",
   Prefix = "I-"
 };
encryptedPartList.Add(last4EncryptedPart);
```

```
var unitEncryptedPart = new EncryptedPart
 {
   Name = "unit",
   Prefix = "U-"
 };
encryptedPartList.Add(unitEncryptedPart);

// 3. Create the compound beacon.
// This compound beacon only requires a name, split character,
// and list of encrypted parts
var compoundBeaconList = new List<CompoundBeacon>>);
var inspectorCompoundBeacon = new CompoundBeacon
  {
      Name = "inspector_id_last4",
      Split = ".",
      Encrypted = encryptedPartList
  };
compoundBeaconList.Add(inspectorCompoundBeacon);
```

使用信標

我們的用戶端加密程式庫已重新命名為 AWS 資料庫加密 SDK。本開發人員指南仍提供 DynamoDB 加密用戶端的相關資訊。

信標 (Beacon) 可讓您搜尋加密的記錄，而無需解密所查詢的整個資料庫。信標的設計是要在新的、未填入的資料庫中實作。在現有資料庫中設定的任何信標只會對應寫入資料庫的新記錄。信標是根據欄位的純文字值計算的，一旦欄位經過加密，信標就無法對應現有資料。使用信標寫入新記錄之後，就無法更新信標的組態。不過，您可以為新增至記錄的新欄位新增信標。

設定信標之後，您必須先完成下列步驟，才能開始填入資料庫並對信標執行查詢。

1. 建立 AWS KMS 階層式金鑰圈

   若要使用可搜尋的加密，您必須使用AWS KMS 階層式金鑰環來產生、加密和解密用於保護記錄的資料金鑰。

   設定信標之後，請組合階層式金鑰圈先決條件並建立階層式金鑰圈。

如需有關為何需要階層式金鑰圈的詳細資訊，請參閱使用階層式金鑰圈進行可搜尋的加密。

2.

定義信標版本

指定您的keyStorekeySource、您設定的所有標準信標清單、您設定的所有複合信標清單、加密零件清單、已簽署零件的清單，以及信標版本。您必須1為信標版本指定。如需定義您的指引keySource，請參閱定義信標金鑰來源。

下列 Java 範例會定義單一承租人資料庫的信標版本。如需定義多租戶資料庫之信標版本的說明，請參閱多租戶資料庫的可搜尋加密。

Java

```
 List<BeaconVersion> beaconVersions = new ArrayList<>();
beaconVersions.add(
    BeaconVersion.builder()
        .standardBeacons(standardBeaconList)
        .compoundBeacons(compoundBeaconList)
        .encryptedParts(encryptedPartsList)
        .signedParts(signedPartsList)
        .version(1) // MUST be 1
        .keyStore(keyStore)
        .keySource(BeaconKeySource.builder()
            .single(SingleKeyStore.builder()
                .keyId(branchKeyId)
                .cacheTTL(6000)
                .build())
            .build())
        .build()
);
```

C# / .NET

```
var beaconVersions = new List<BeaconVersion>
{
    new BeaconVersion
    {
        StandardBeacons = standardBeaconList,
        CompoundBeacons = compoundBeaconList,
        EncryptedParts = encryptedPartsList,
```

```
        SignedParts = signedPartsList,
        Version = 1, // MUST be 1
        KeyStore = branchKeyStoreName,
        KeySource = new BeaconKeySource
        {
            Single = new SingleKeyStore
            {
                KeyId = branch-key-id,
                CacheTTL = 6000
            }
        }
    }
};
```

3. 設定次要索引

   設定信標之後，您必須先設定反映每個信標的次要索引，才能搜尋加密欄位。如需詳細資訊，請參閱 使用信標設定次要索引。

4. 定義您的密碼編譯動作

   必須標記用於建構標準信標的所有欄位ENCRYPT_AND_SIGN。用於構造信標的所有其他字段必須標記SIGN_ONLY或SIGN_AND_INCLUDE_IN_ENCRYPTION_CONTEXT。

5. 設定資 AWS 料庫加密 SDK 用戶端

   若要設定可保護 DynamoDB 表格中表項目的 AWS 資料庫加密 SDK 用戶端，請參閱 DynamoDB 的 Java 用戶端加密程式庫。

查詢信標

您設定的信標類型會決定您可以執行的查詢類型。標準信標使用篩選運算式來執行相等搜尋。複合信標結合文字純文字字串和標準信標，以執行複雜的查詢。當您查詢加密的資料時，您會搜尋信標名稱。

您無法比較兩個標準信標的值，即使它們包含相同的基礎純文字也一樣。兩個標準信標將為相同的明文值產生兩個不同的 HMAC 標籤。因此，標準信標無法執行下列查詢。

- *beacon1* = *beacon2*
- *beacon1* IN (*beacon2*)
- *value* IN (*beacon1*, *beacon2*, ...)
- CONTAINS(*beacon1*, *beacon2*)

複合信標可以執行以下查詢。

- BEGINS_WITH(*a*)，其中*a*反映組合化合物信標開始於之欄位的整個值。您無法使用BEGINS_WITH運算子來識別以特定子字串開頭的值。但是，您可以使用BEGINS_WITH(*S_*)，其中*S_*反映組裝化合物信標開頭之零件的字首。
- CONTAINS(*a*)，其中*a*反映組合化合物信標所包含之欄位的整個值。您無法使用CONTAINS運算子來識別包含特定子字串或集合中值的記錄。

  例如，您無法執行*a*反映集合中值的查詢CONTAINS(*path*, "*a*"。
- 您可以比較複合信標的簽名部分。比較已簽署零件時，您可以選擇性地將加密零件的前置詞附加到一或多個已簽署零件，但不能在任何查詢中包含加密欄位的值。

  例如，您可以比較已簽署的零件，並在*signedField1* = *signedField2*或上進行查詢*value* IN (*signedField1*, *signedField2*, ...)。

  您也可以透過上的查詢來比較已簽署零件與加密零件的字首*signedField1.A_* = *signedField2.B_*。
- *field* BETWEEN *a* AND *b*，在哪裡*a*和簽名*b*的部分。您可以選擇性地將加密零件的前置詞附加到一或多個已簽署零件，但不能在任何查詢中包含加密欄位的值。

您必須包含在複合信標的查詢中的每個零件的前置詞。例如，如果您從兩個欄位建構複合信標compoundBeaconsignedField，encryptedField並且在查詢信標時，必須包含為這兩個部分設定的首碼。

```
compoundBeacon = E_encryptedFieldValue.S_signedFieldValue
```

可搜尋的多租戶資料庫加密

> 我們的用戶端加密程式庫已重新命名為 AWS 資料庫加密 SDK。本開發人員指南仍提供 DynamoDB 加密用戶端的相關資訊。

若要在資料庫中實作可搜尋的加密，您必須使用AWS KMS 階層式金鑰圈。 AWS KMS 階層式金鑰圈會產生、加密和解密用於保護記錄的資料金鑰。它也會建立用來產生信標的信標金鑰。將 AWS KMS 階層式金鑰圈與多租戶資料庫搭配使用時，每個租用戶都有不同的分支索引鍵和信標金鑰。若要查詢多租戶資料庫中的加密資料，您必須識別用來產生要查詢之信標的信標金鑰材料。

當您定義多租戶資料庫的信標 (Beacon) 版本時，請指定您設定的所有標準信標清單、您設定的所有複合信標清單、信標 (Beacon) 版本，以及 keySource 您必須將信標主要來源定義為MultiKeyStore區域信標主要快取的存留快取時間，以及區域信標主要快取記憶體的快取大小上限。keyFieldName

如果您設定了任何已簽署的信標，它們必須包含在您的compoundBeaconList. 簽名信標是一種複合信標，可對和SIGN_AND_INCLUDE_IN_ENCRYPTION_CONTEXT字段進行複雜查詢編制索引SIGN_ONLY和執行。

Java

```
List<BeaconVersion> beaconVersions = new ArrayList<>();
    beaconVersions.add(
        BeaconVersion.builder()
                .standardBeacons(standardBeaconList)
                .compoundBeacons(compoundBeaconList)
                .version(1) // MUST be 1
                .keyStore(branchKeyStoreName)
                .keySource(BeaconKeySource.builder()
                        .multi(MultiKeyStore.builder()
                                .keyFieldName(keyField)
                                .cacheTTL(6000)
                                .maxCacheSize(10)
                        .build())
                .build())
        .build()
    );
```

C# / .NET

```
var beaconVersions = new List<BeaconVersion>
{
    new BeaconVersion
    {
        StandardBeacons = standardBeaconList,
        CompoundBeacons = compoundBeaconList,
        EncryptedParts = encryptedPartsList,
        SignedParts = signedPartsList,
        Version = 1, // MUST be 1
        KeyStore = branchKeyStoreName,
        KeySource = new BeaconKeySource
        {
```

```
            Multi = new MultiKeyStore
            {
                KeyId = branch-key-id,
                CacheTTL = 6000,
                MaxCacheSize = 10
            }
        }
    }
};
```

keyFieldName

定keyFieldName義欄位名稱，此欄位名稱會儲存branch-key-id與指定承租人產生信標所用之信標之信標關聯的信標索引鍵。

當您將新記錄寫入資料庫時，此欄位會儲存識別用來產生該記錄之任何信標的信標索引鍵。branch-key-id

依預設，keyField是未明確儲存在資料庫中的概念欄位。Datab AWS ase Encryption SDK會branch-key-id從材料說明中識別加密的資料金鑰，並將值儲存在概念中，keyField供您在複合信標和已簽署的信標中參照。由於材料描述已簽署，因此概念keyField被視為已簽署的零件。

您也可以在密碼編譯動作keyField中包含作為SIGN_ONLY或SIGN_AND_INCLUDE_IN_ENCRYPTION_CONTEXT欄位，以將欄位明確儲存在資料庫中。如果您這樣做，則keyField每次將記錄寫入資料庫時，都必須手動包含 branch-key-id in。

查詢多租戶資料庫中的信標

若要查詢信標，您必須在查詢keyField中包含，以識別重新計算信標所需的適當信標主要原物料。您必須指定branch-key-id與用來產生記錄之信標之信標之信標關聯的信標索引鍵。您無法指定可識別分支金鑰 ID 供應商branch-key-id中承租人的易記名稱。您可以透過下列方式在查詢中包含。keyField

複合信標

無論您是否在記錄keyField中明確存儲，都可以將其作為簽名部分keyField直接包含在複合信標中。必須要有keyField簽署的零件。

例如，如果您想要從兩個欄位建構複合信標compoundBeaconsignedField，encryptedField並且還必須包含keyField作為已簽署零件。這可讓您在上執行下列查詢compoundBeacon。

```
compoundBeacon = E_encryptedFieldValue.S_signedFieldValue.K_branch-key-id
```

簽署的信標

資 AWS 料庫加密 SDK 使用標準和複合信標來提供可搜尋的加密解決方案。這些信標必須至少包含一個加密欄位。不過， AWS 資料庫加密 SDK 也支援已簽署的信標，這些信標可以完全從純文字SIGN_ONLY和SIGN_AND_INCLUDE_IN_ENCRYPTION_CONTEXT欄位進行設定。

簽名信標可以從單個部分構建。無論您是否明確地儲存keyField在記錄中，都可以從建構已簽署的信標，keyField並使用它來建立複合查詢，將keyField已簽署信標上的查詢與其他信標上的查詢結合在一起。例如，您可以執行下列查詢。

```
keyField = K_branch-key-id AND compoundBeacon =
 E_encryptedFieldValue.S_signedFieldValue
```

如需設定簽署信標的說明，請參閱 建立簽署的信標

直接在 **keyField**

如果您keyField在密碼編譯動作中指定了，並將欄位明確儲存在記錄中，則可以建立複合查詢，將信標上的查詢與上的查詢結合在keyField一起。keyField如果您要查詢標準信標，可以選擇直接查詢。例如，您可以執行下列查詢。

```
keyField = branch-key-id AND standardBeacon = S_standardBeaconValue
```

AWS 適用於資料庫加密開發套件

我們的客戶端加密庫被重命名為 AWS 數據庫加密 SDK。本開發人員指南仍提供 DynamoDB 加密用戶端的相關資訊。

適用於 DynamoDB 的 AWS 資料庫加密開發套件是一種軟體程式庫，可讓您在 Amazon DynamoDB 設計中加入用戶端加密。DynamoDB 的 AWS 資料庫加密 SDK 提供屬性層級加密，可讓您指定要加密的項目，以及要在簽章中包含哪些項目，以確保資料的真實性。對傳輸中和靜態的敏感資料進行加密，有助於確保您的純文字資料無法供任何第三方使用，包括. AWS

> **Note**
>
> 下列主題著重於版本 3。 Java 用戶端加密程式庫 DynamoDB 個。
> 我們的客戶端加密庫被重命名為 AWS 數據庫加密 SDK。資 AWS 料庫加密 SDK 會繼續支援舊版 DynamoDB 加密用戶端版本。

在 DynamoDB 中，資料表是項目的集合。每個項目都是屬性的集合。每個屬性都有名稱和數值。適用於 DynamoDB 的 AWS 資料庫加密開發套件會加密屬性的值。接著，它會透過屬性計算簽章。您可以指定要加密的屬性值，以及要在密碼編譯動作的簽章中包含哪些屬性值。

本章中的主題提供 DynamoDB 適用的 AWS 資料庫加密 SDK 概觀，包括哪些欄位已加密、用戶端安裝和設定的指導，以及協助您開始使用的 Java 範例。

主題

- 用戶端加密和伺服器端加密
- 哪些欄位已加密並簽署？
- 在 DynamoDB 中可搜尋的加密
- 更新您的資料模型
- AWS 適用於 DynamoDB 的資料庫加密 SDK 可用程式設計語言
- 舊 DynamoDB 加密用戶端

用戶端加密和伺服器端加密

我們的用戶端加密程式庫已重新命名為AWS資料庫加密 SDK。本開發人員指南仍提供 DynamoDB 加密用戶端的相關資訊。

適用於 DynamoDB 的資料AWS庫加密 SDK 支援用戶端加密，您可以在其中加密表格資料，然後再將資料表資料傳送到資料庫。不過，DynamoDB 提供伺服器端靜態加密功能，可在資料表保留至磁碟時透明地加密資料表，並在您存取資料表時對其進行解密。

您所應選擇的工具，取決於資料的敏感度以及應用程式的安全性需求。您可以針對 DynamoDB 使用AWS資料庫加密開發套件，也可以使用靜態加密。當您將加密和簽署的項目傳送到 DynamoDB 時，DynamoDB 無法將這些項目識別為受保護。它只會偵測具有二進位屬性值的一般資料表項目。

伺服器端靜態加密

DynamoDB 支援靜態加密，這是一項伺服器端加密功能，其中 DynamoDB 會在資料表保留至磁碟時為您透明地為您加密資料表，並在您存取資料表資料時對其進行解密。

當您使用 AWS SDK 與 DynamoDB 互動時，預設情況下，您的資料會在透過 HTTPS 連線傳輸過程中加密、在 DynamoDB 端點解密，然後再重新加密，然後再儲存在 DynamoDB 中。

- 默認情況下加密。DynamoDB 會在寫入所有表格時透明地加密和解密這些資料表。沒有啟用或停用靜態加密的選項。
- DynamoDB 會建立和管理加密編譯金鑰。每個表的唯一AWS KMS key密鑰受到永遠不會離開 AWS Key Management Service（AWS KMS）未加密的保護。根據預設，DynamoDB 會AWS 擁有的金鑰在 DynamoDB 服務帳戶中使用，但您可以在帳戶中選擇AWS 受管金鑰或客戶管理的金鑰來保護部分或全部表格。
- 所有表格資料都會在磁碟上加密。將加密的表格儲存到磁碟時，DynamoDB 會加密所有表格資料，包括主索引鍵以及本機和全域次要索引。如果您的資料表有排序索引鍵，則某些標示範圍界限的排序索引鍵將會以純文字的格式存放在資料表中繼資料中。
- 與資料表相關的物件也會加密。靜態加密可在 DynamoDB 串流、全域資料表和備份寫入耐用媒體時提供保護。
- 當您訪問它們時，您的項目將被解密。當您存取資料表時，DynamoDB 會解密包含目標項目的表格部分，並將純文字項目傳回給您。

AWS適用於資料庫加密開發套件

無論是在傳輸中、靜態時，還是從來源儲存至 DynamoDB 時，用戶端加密都可為資料提供端對端的保護。您的純文字資料絕不會對任何第三方公開，包括 AWS 在內。您可以將適用於 DynamoDB 的AWS資料庫加密開發套件與新的 DynamoDB 表格搭配使用，也可以將現有的 Amazon DynamoDB 表格遷移到第 3 版。 適用於 Java 用戶端加密程式庫的 x 個。

- 傳輸中和靜態的資料都受到保護。它永遠不會暴露給任何第三方，包括AWS.
- 您可以簽署您的資料表項目。您可以指示 DynamoDB 的AWS資料庫加密 SDK 計算全部或部分表格項目 (包括主索引鍵屬性) 的簽章。此簽章可讓您偵測對整體項目的未授權變更，包括新增或刪除屬性，或是交換屬性值。
- 您可以透過選取金鑰圈來決定資料受到保護的方式。您的金鑰圈會決定保護資料金鑰的包裝金鑰，以及最終保護資料的金鑰。使用對您的任務實用的最安全的包裝密鑰。
- 適用於 DynamoDB 的AWS資料庫加密開發套件不會加密整個資料表。您可以選擇項目中要加密的屬性。適用於 DynamoDB 的AWS資料庫加密開發套件不會加密整個項目。它不會加密屬性名稱，或是主索引鍵 (分割區索引鍵和排序索引鍵) 屬性的名稱或值。

AWS Encryption SDK

如果您要加密儲存在 DynamoDB 中的資料，我們建議您使用適用於 DynamoDB 的資料AWS庫加密開發套件。

AWS Encryption SDK 是用戶端加密程式庫，可協助您進行一般資料的加密和解密。雖然它可保護任何類型的資料，但在設計上並不是用來處理結構化資料 (例如資料庫記錄)。與 DynamoDB 的AWS資料庫加密 SDK 不同，AWS Encryption SDK無法提供項目層級完整性檢查，也沒有邏輯可識別屬性或防止主金鑰加密。

如果您使用AWS Encryption SDK來加密表格的任何元素，請記住它與 DynamoDB 的AWS資料庫加密 SDK 不相容。您無法用一個程式庫加密，但用另一個程式庫解密。

哪些欄位已加密並簽署？

> 我們的用戶端加密程式庫已重新命名為 AWS 資料庫加密 SDK。本開發人員指南仍提供 DynamoDB 加密用戶端的相關資訊。

適用於 DynamoDB 的 AWS 資料庫加密開發套件是專為 Amazon DynamoDB 應用程式設計的用戶端加密程式庫。Amazon DynamoDB 會將資料存放在資料表中，這些資料是項目的集合。每個項目都是屬性的集合。每個屬性都有名稱和數值。適用於 DynamoDB 的 AWS 資料庫加密開發套件會加密屬

性的值。接著，它會透過屬性計算簽章。您可以指定哪些屬性值要加密，以及哪些屬性值要包含在簽章中。

加密可保護屬性值的機密性。簽署可提供所有已簽署屬性的完整性及其彼此的關係，並可提供驗證。它可讓您偵測對整體項目的未授權變更 (包括新增或刪除屬性)，或是替換已加密的值。

在加密項目中，有些資料會保持純文字格式，包括資料表名稱、所有屬性名稱、未加密的屬性值、主索引鍵 (分割索引鍵和排序索引鍵) 屬性的名稱和值，以及屬性類型。請勿在這些欄位中存放敏感性資訊。

如需 DynamoDB 的資 AWS 料庫加密 SDK 如何運作的詳細資訊，請參閱。AWS 資料庫加密 SDK 的運作方式

> **Note**
>
> DynamoDB 適用的 AWS 資料庫加密 SDK 主題中所有屬性動作的提及均參考加密動作。

主題

加密屬性值

DynamoDB 的 AWS 資料庫加密 SDK 會加密您指定之屬性的值 (但不會加密屬性名稱或類型)。若要決定加密的是哪些屬性值，請使用屬性動作。

例如，這個項目包括 `example` 和 `test` 屬性。

```
'example': 'data',
'test': 'test-value',
...
```

如果您將 `example` 屬性加密，但不加密 `test` 屬性，結果如下所示。已加密的 `example` 屬性值是二進位資料，而不是字串。

```
'example': Binary(b"'b\x933\x9a+s\xf1\xd6a\xc5\xd5\x1aZ\xed\xd6\xce\xe9X\xf0T\xcb\x9fY
\x9f\xf3\xc9C\x83\r\xbb\\"),
'test': 'test-value'
```

```
...
```

每個項目的主索引鍵屬性 (分區索引鍵和排序索引鍵) 必須保留為純文字，因為 DynamoDB 會使用這些屬性來尋找表格中的項目。這些屬性應加以簽署，但不要加密。

適用於 DynamoDB 的 AWS 資料庫加密 SDK 可為您識別主要金鑰屬性，並確保其值已簽署但未加密。而且，如果您找出您的主要索引鍵，然後試著將它加密，則用戶端會擲出例外狀況。

客戶端將材料描述存儲在一個新屬性（aws_dbe_head）中，該屬性將其添加到物件中。材料描述描述了項目的加密和簽名方式。用戶端會使用此資訊來驗證並解密項目。儲存材料描述的欄位未加密。

簽署項目

將指定的屬性值加密後，DynamoDB 的資料 AWS 庫加密 SDK 會根據材料描述、加密內容以及標記ENCRYPT_AND_SIGN的每個欄位或屬性動作的規範化，計算雜湊型訊息驗證碼 (HMAC) 和數位簽章。SIGN_ONLYSIGN_AND_INCLUDE_IN_ENCRYPTION_CONTEXTECDSA 簽章預設為啟用，但不是必要的。客戶端將 HMAC 和簽名存儲在添加到項目的新屬性（aws_dbe_foot）中。

在 DynamoDB 中可搜尋的加密

若要將 Amazon DynamoDB 表格設定為可搜尋的加密，您必須使用AWS KMS 階層式金鑰圈來產生、加密和解密用於保護項目的資料金鑰。您還必須SearchConfig在表格加密配置中包含。

> **Note**
>
> 如果您使用 DynamoDB 的 Java 用戶端加密程式庫，則必須使用適用於 DynamoDB API 的低階 AWS 資料庫加密開發套件來加密、簽署、驗證和解密資料表項目。DynamoDB 增強型用戶端和較低層級DynamoDBItemEncryptor不支援可搜尋的加密。

主題

使用信標設定次要索引

設定信標之後，您必須先設定反映每個信標的次要索引，才能搜尋加密的屬性。

當您設定標準或複合信標時，Datab AWS ase Encryption SDK 會將前置aws_dbe_b_詞新增至信標名稱，以便伺服器可以輕鬆識別信標。例如，如果您命名複合信標compoundBeacon，則實際上是完整的信標名稱aws_dbe_b_compoundBeacon。如果要設定包含標準或複合信標的次要索引，識別信標名aws_dbe_b_稱時必須包含前置詞。

分割和排序索引鍵

您無法加密主索引鍵值。您的分區和排序鍵必須簽名。您的主鍵值不能是標準或複合信標。

您的主鍵值必須是SIGN_ONLY，除非您指定任何SIGN_AND_INCLUDE_IN_ENCRYPTION_CONTEXT屬性，否則分區和排序屬性也必須是SIGN_AND_INCLUDE_IN_ENCRYPTION_CONTEXT。

您的主鍵值可以是簽名的信標。如果您為每個主索引鍵值設定不同的簽署信標，則必須指定可將主索引鍵值識別為已簽署信標名稱的屬性名稱。不過， AWS 資料庫加密 SDK 不會將aws_dbe_b_前置詞新增至已簽署的信標。即使您為主索引鍵值設定了不同的已簽署信標，在設定次要索引時，只需指定主索引鍵值的屬性名稱。

本機次要索引

本機次要索引的排序索引鍵可以是信標。

如果您指定排序索引鍵的信標，類型必須是「字串」。如果您為排序索引鍵指定標準或複合信標，則必須在指定信標名aws_dbe_b_稱時加入前置碼。如果您指定已簽署的信標，請指定不含任何前置碼的信標名稱。

全域次要索引

全域次要索引的分割區和排序索引鍵都可以是信標。

如果您指定分割區或排序索引鍵的信標，類型必須是「字串」。如果您為排序索引鍵指定標準或複合信標，則必須在指定信標名aws_dbe_b_稱時加入前置碼。如果您指定已簽署的信標，請指定不含任何前置碼的信標名稱。

屬性投影

投影是指從資料表複製到次要索引的屬性集合。資料表的分割區索引鍵和排序索引鍵一律會投影到索引中；您可以投影其他屬性來支援應用程式的查詢需求。DynamoDB 針對屬性投影提供了三種不同的選項：KEYS_ONLY、INCLUDE和。ALL

如果您使用 INCLUDE 屬性投影來搜尋信標，則必須指定建構信標之所有屬性的名稱，以及含有aws_dbe_b_前置碼的信標名稱。例如，如果您設定了複合信

標compoundBeacon，則來自field1field2field3、和，則必須在投影field3中指定aws_dbe_b_compoundBeaconfield1field2、和。

全域次要索引只能使用在投影中明確指定的屬性，但本機次要索引可以使用任何屬性。

測試信標輸出

如果您設定了複合信標或使用虛擬欄位建構信標，建議您在填入 DynamoDB 表之前先確認這些信標是否產生預期的輸出。

資 AWS 料庫加密 SDK 提供的DynamoDbEncryptionTransforms服務可協助您疑難排解虛擬欄位和複合信標輸出。

測試虛擬領域

下列程式碼片段會建立測試項目、使用 DynamoDB 表格加密組態定義DynamoDbEncryptionTransforms服務，並示範如何使用ResolveAttributes來驗證虛擬欄位是否產生預期的輸出。

Java

請參閱完整的代碼示例：VirtualBeaconSearchableEncryptionExample.java

```
// Create test items
final PutItemRequest itemWithHasTestResultPutRequest = PutItemRequest.builder()
    .tableName(ddbTableName)
    .item(itemWithHasTestResult)
    .build();

final PutItemResponse itemWithHasTestResultPutResponse =
 ddb.putItem(itemWithHasTestResultPutRequest);

final PutItemRequest itemWithNoHasTestResultPutRequest = PutItemRequest.builder()
    .tableName(ddbTableName)
    .item(itemWithNoHasTestResult)
    .build();

final PutItemResponse itemWithNoHasTestResultPutResponse =
 ddb.putItem(itemWithNoHasTestResultPutRequest);

// Define the DynamoDbEncryptionTransforms service
final DynamoDbEncryptionTransforms trans = DynamoDbEncryptionTransforms.builder()
```

```
    .DynamoDbTablesEncryptionConfig(encryptionConfig).build();

// Verify configuration
final ResolveAttributesInput resolveInput = ResolveAttributesInput.builder()
    .TableName(ddbTableName)
    .Item(itemWithHasTestResult)
    .Version(1)
    .build();
final ResolveAttributesOutput resolveOutput = trans.ResolveAttributes(resolveInput);

// Verify that VirtualFields has the expected value
Map<String, String> vf = new HashMap<>();
vf.put("stateAndHasTestResult", "CAt");
assert resolveOutput.VirtualFields().equals(vf);
```

C# / .NET

請參閱完整的程式碼範例：[VirtualBeaconSearchableEncryptionExample.cs]

```
 // Create item with hasTestResult=true
var itemWithHasTestResult = new Dictionary<String, AttributeValue>
{
    ["customer_id"] = new AttributeValue("ABC-123"),
    ["create_time"] = new AttributeValue { N = "1681495205" },
    ["state"] = new AttributeValue("CA"),
    ["hasTestResult"] = new AttributeValue { BOOL = true }
};

// Create item with hasTestResult=false
var itemWithNoHasTestResult = new Dictionary<String, AttributeValue>
{
    ["customer_id"] = new AttributeValue("DEF-456"),
    ["create_time"] = new AttributeValue { N = "1681495205" },
    ["state"] = new AttributeValue("CA"),
    ["hasTestResult"] = new AttributeValue { BOOL = false }
};

// Define the DynamoDbEncryptionTransforms service
var trans = new DynamoDbEncryptionTransforms(encryptionConfig);

// Verify configuration
var resolveInput = new ResolveAttributesInput
{
    TableName = ddbTableName,
```

```
    Item = itemWithHasTestResult,
    Version = 1
};
var resolveOutput = trans.ResolveAttributes(resolveInput);

// Verify that VirtualFields has the expected value
Debug.Assert(resolveOutput.VirtualFields.Count == 1);
Debug.Assert(resolveOutput.VirtualFields["stateAndHasTestResult"] == "CAt");
```

測試複合信標

下列程式碼片段會建立測試項目、使用 DynamoDB 表格加密組態定義DynamoDbEncryptionTransforms服務，並示範如何使用ResolveAttributes來驗證複合信標是否產生預期的輸出。

Java

請參閱完整的代碼示例：CompoundBeaconSearchableEncryptionExample.java

```
// Create an item with both attributes used in the compound beacon.
final HashMap<String, AttributeValue> item = new HashMap<>();
item.put("work_id", AttributeValue.builder().s("9ce39272-8068-4efd-a211-
cd162ad65d4c").build());
item.put("inspection_date", AttributeValue.builder().s("2023-06-13").build());
item.put("inspector_id_last4", AttributeValue.builder().s("5678").build());
item.put("unit", AttributeValue.builder().s("011899988199").build());

// Define the DynamoDbEncryptionTransforms service
final DynamoDbEncryptionTransforms trans = DynamoDbEncryptionTransforms.builder()
    .DynamoDbTablesEncryptionConfig(encryptionConfig).build();

// Verify configuration
final ResolveAttributesInput resolveInput = ResolveAttributesInput.builder()
    .TableName(ddbTableName)
    .Item(item)
    .Version(1)
    .build();

final ResolveAttributesOutput resolveOutput = trans.ResolveAttributes(resolveInput);

// Verify that CompoundBeacons has the expected value
Map<String, String> cbs = new HashMap<>();
```

```
cbs.put("last4UnitCompound", "L-5678.U-011899988199");
assert resolveOutput.CompoundBeacons().equals(cbs);
// Note : the compound beacon actually stored in the table is not
 "L-5678.U-011899988199"
// but rather something like "L-abc.U-123", as both parts are EncryptedParts
// and therefore the text is replaced by the associated beacon
```

C# / .NET

請參閱完整的程式碼範例：CompoundBeaconSearchableEncryptionExample.cs

```
// Create an item with both attributes used in the compound beacon
var item = new Dictionary<String, AttributeValue>
{
    ["work_id"] = new AttributeValue("9ce39272-8068-4efd-a211-cd162ad65d4c"),
    ["inspection_date"] = new AttributeValue("2023-06-13"),
    ["inspector_id_last4"] = new AttributeValue("5678"),
    ["unit"] = new AttributeValue("011899988199")
};

// Define the DynamoDbEncryptionTransforms service
var trans = new DynamoDbEncryptionTransforms(encryptionConfig);

// Verify configuration
var resolveInput = new ResolveAttributesInput
{
    TableName = ddbTableName,
    Item = item,
    Version = 1
};
var resolveOutput = trans.ResolveAttributes(resolveInput);


// Verify that CompoundBeacons has the expected value
Debug.Assert(resolveOutput.CompoundBeacons.Count == 1);
Debug.Assert(resolveOutput.CompoundBeacons["last4UnitCompound"] ==
 "L-5678.U-011899988199");
// Note : the compound beacon actually stored in the table is not
 "L-5678.U-011899988199"
// but rather something like "L-abc.U-123", as both parts are EncryptedParts
// and therefore the text is replaced by the associated beacon
```

更新您的資料模型

我們的用戶端加密程式庫已重新命名為 AWS 資料庫加密 SDK。本開發人員指南仍提供 DynamoDB 加密用戶端的相關資訊。

當您為 DynamoDB 設定 AWS 資料庫加密開發套件時，您會提供屬性動作。在加密時， AWS 資料庫加密 SDK 會使用屬性動作來識別要加密和簽署的屬性、要簽署 (但不加密) 的屬性，以及要忽略哪些屬性。您也可以定義允許的未簽署屬性，以明確告知用戶端哪些屬性已從簽章中排除。解密時， AWS 資料庫加密 SDK 會使用您定義的允許未簽署屬性來識別簽章中未包含的屬性。屬性動作不會儲存在加密項目中，且資 AWS 料庫加密 SDK 不會自動更新您的屬性動作。

請仔細選擇屬性動作。如有疑問，請使用加密並簽署。使用 AWS 資料庫加密 SDK 來保護您的項目之後，您就無法將現有的ENCRYPT_AND_SIGNSIGN_ONLY、或SIGN_AND_INCLUDE_IN_ENCRYPTION_CONTEXT屬性變更為DO_NOTHING。但是，您可以放心地進行以下更改。

- 新增ENCRYPT_AND_SIGNSIGN_ONLY、和SIGN_AND_INCLUDE_IN_ENCRYPTION_CONTEXT屬性
- 移除現有屬性
- 將現有ENCRYPT_AND_SIGN屬性變更為SIGN_ONLY或SIGN_AND_INCLUDE_IN_ENCRYPTION_CONTEXT
- 將現有SIGN_ONLY或SIGN_AND_INCLUDE_IN_ENCRYPTION_CONTEXT屬性變更為ENCRYPT_AND_SIGN
- 新增DO_NOTHING屬性
- 將現有SIGN_ONLY屬性變更為 SIGN_AND_INCLUDE_IN_ENCRYPTION_CONTEXT
- 將現有SIGN_AND_INCLUDE_IN_ENCRYPTION_CONTEXT屬性變更為 SIGN_ONLY

可搜尋加密的考量

在更新資料模型之前，請仔細考慮您的更新可能會對您透過屬性建構的任何信標造成的影響。使用信標寫入新記錄之後，就無法更新信標的組態。您無法更新與用來建構信標之屬性相關聯的屬性動作。如果您移除現有屬性及其相關聯的信標，您將無法使用該信標查詢現有記錄。您可以為新增至記錄的新欄位建立新信標，但無法更新現有信標以包含新欄位。

SIGN_AND_INCLUDE_IN_ENCRYPTION_CONTEXT屬性的考量

依預設，分割區和排序索引鍵是加密內容中包含的唯一屬性。您可以考慮定義其他欄位，以SIGN_AND_INCLUDE_IN_ENCRYPTION_CONTEXT便AWS KMS 階層式金鑰圈的分支金鑰 ID 供應商可識別從加密內容解密所需的分支金鑰。如需詳細資訊，請參閱分支金鑰 ID 供應商。如果您指定任何SIGN_AND_INCLUDE_IN_ENCRYPTION_CONTEXT屬性，則分割區和排序屬性也必須是SIGN_AND_INCLUDE_IN_ENCRYPTION_CONTEXT。

> **Note**
>
> 若要使用加SIGN_AND_INCLUDE_IN_ENCRYPTION_CONTEXT密動作，您必須使用 3.3 版或更新版本的 AWS 資料庫加密 SDK。在更新要包含的資料模型之前，請先將新版本部署至所有讀取器SIGN_AND_INCLUDE_IN_ENCRYPTION_CONTEXT。

新增ENCRYPT_AND_SIGNSIGN_ONLY、和SIGN_AND_INCLUDE_IN_ENCRYPTION_CONTEXT屬性

欲新增ENCRYPT_AND_SIGN、或SIGN_AND_INCLUDE_IN_ENCRYPTION_CONTEXT屬性SIGN_ONLY，請在屬性動作中定義新屬性。

您無法移除現有DO_NOTHING屬性並將其新增回為ENCRYPT_AND_SIGNSIGN_ONLY、或SIGN_AND_INCLUDE_IN_ENCRYPTION_CONTEXT屬性。

使用註釋的數據類

如果您使用 a 定義了屬性動作TableSchema，請將新屬性新增至已註解的資料類別。如果您沒有為新屬性指定屬性動作註釋，則用戶端預設會加密並簽署新屬性 (除非該屬性是主索引鍵的一部分)。如果您只想簽署新屬性，則必須新增具有@DynamoDBEncryptionSignOnly或@DynamoDBEncryptionSignAndIncludeInEncryptionConte釋的新屬性。

使用物件模型

如果您手動定義屬性動作，請將新屬性新增至物件模型中的屬性動作，並指定ENCRYPT_AND_SIGNSIGN_ONLY、或作SIGN_AND_INCLUDE_IN_ENCRYPTION_CONTEXT為屬性動作。

移除現有屬性

如果您決定不再需要屬性，可以停止將資料寫入該屬性，或者您可以將其從屬性動作中正式移除。當您停止將新資料寫入屬性時，屬性仍會顯示在屬性動作中。如果您將 future 需要再次開始使用屬性，這

可能會很有幫助。從屬性動作中正式移除屬性並不會將其從資料集中移除。您的資料集仍會包含包含該屬性的項目。

欲正式移除現有ENCRYPT_AND_SIGN、SIGN_ONLY、或DO_NOTHING屬性SIGN_AND_INCLUDE_IN_ENCRYPTION_CONTEXT，請更新您的屬性動作。

如果您移除DO_NOTHING屬性，則不得從允許的未簽署屬性中移除該屬性。即使您不再將新值寫入該屬性，用戶端仍需要知道該屬性未簽署，才能讀取包含該屬性的現有項目。

使用註釋的數據類

如果您使用 a 定義了屬性動作TableSchema，請從已註解的資料類別中移除屬性。

使用物件模型

如果您手動定義屬性動作，請從物件模型的屬性動作中移除屬性。

將現有ENCRYPT_AND_SIGN屬性變更為SIGN_ONLY或SIGN_AND_INCLUDE_IN_ENCRYPTION_CONTEXT

欲將現有ENCRYPT_AND_SIGN屬性變更為SIGN_ONLY或SIGN_AND_INCLUDE_IN_ENCRYPTION_CONTEXT，您必須更新屬性動作。部署更新之後，用戶端將能夠驗證和解密寫入屬性的現有值，但只會簽署寫入屬性的新值。

> **Note**
>
> 將現有屬性變更為SIGN_ONLY或之前，請仔細考量您的安全ENCRYPT_AND_SIGN性需求SIGN_AND_INCLUDE_IN_ENCRYPTION_CONTEXT。任何可以儲存敏感資料的屬性都應加密。

使用註釋的數據類

如果您使用 a 定義了屬性動作TableSchema，請更新現有屬性以將@DynamoDBEncryptionSignOnly或@DynamoDBEncryptionSignAndIncludeInEncryptionConte釋包括在已註解的資料類別中。

使用物件模型

如果您手動定義屬性動作，請將與現有屬性相關聯的屬性動作從物件模型更新ENCRYPT_AND_SIGN至SIGN_ONLY或物件模型SIGN_AND_INCLUDE_IN_ENCRYPTION_CONTEXT中。

將現有SIGN_ONLY或SIGN_AND_INCLUDE_IN_ENCRYPTION_CONTEXT屬性變更為 ENCRYPT_AND_SIGN

欲將現有SIGN_ONLY或SIGN_AND_INCLUDE_IN_ENCRYPTION_CONTEXT屬性變更為ENCRYPT_AND_SIGN，您必須更新屬性動作。部署更新之後，用戶端將能夠驗證寫入屬性的現有值，並將加密並簽署寫入屬性的新值。

使用註釋的數據類

如果您使用定義了屬性動作TableSchema，請從現有屬性中移除@DynamoDBEncryptionSignOnly或@DynamoDBEncryptionSignAndIncludeInEncryptionConte釋。

使用物件模型

如果您手動定義屬性動作，請ENCRYPT_AND_SIGN在物件模型中從SIGN_ONLY或SIGN_AND_INCLUDE_IN_ENCRYPTION_CONTEXT更新與屬性相關聯的屬性動作。

新增DO_NOTHING屬性

若要降低新增屬性時發生錯誤的風險，建議您DO_NOTHING在命名DO_NOTHING屬性時指定不同的前置詞，然後使用該前置詞來定義允許的未簽署屬性。

您無法從已註解的資料類別中移除現有ENCRYPT_AND_SIGN的SIGN_ONLY、或SIGN_AND_INCLUDE_IN_ENCRYPTION_CONTEXT屬性，然後再將屬性新增回來作為DO_NOTHING屬性。您只能新增全新DO_NOTHING屬性。

您要新增DO_NOTHING屬性所採取的步驟，取決於您是否在清單中明確定義了允許的未簽署屬性，還是使用前置詞定義。

使用允許的未簽名屬性前綴

如果您使用 a 定義了屬性動作TableSchema，請使用註釋將新DO_NOTHING屬性添加到帶@DynamoDBEncryptionDoNothing註釋的數據類中。如果您手動定義屬性動作，請更新屬性動作以包含新屬性。請務必使用屬性動作明確配置新DO_NOTHING屬性。您必須在新屬性的名稱中包含相同的不同前綴。

使用允許的未簽署屬性清單

1. 將新DO_NOTHING屬性新增至允許的未簽署屬性清單，並部署更新的清單。
2. 從步驟 1 部署變更。

   在變更已傳播到需要讀取此資料的所有主機之前，您無法移至步驟 3。
3. 將新DO_NOTHING屬性新增至屬性動作。

   a. 如果您使用 a 定義了屬性動作TableSchema，請使用註釋將新DO_NOTHING屬性添加到帶@DynamoDBEncryptionDoNothing註釋的數據類中。

   b. 如果您手動定義屬性動作，請更新屬性動作以包含新屬性。請務必使用屬性動作明確配置新DO_NOTHING屬性。
4. 從步驟 3 部署變更。

將現有SIGN_ONLY屬性變更為SIGN_AND_INCLUDE_IN_ENCRYPTION_CONTEXT

若要將現有SIGN_ONLY屬性變更為SIGN_AND_INCLUDE_IN_ENCRYPTION_CONTEXT，您必須更新屬性動作。部署更新之後，用戶端將能夠驗證寫入屬性的現有值，並繼續簽署寫入屬性的新值。寫入屬性的新值將包含在加密內容中。

如果您指定任何SIGN_AND_INCLUDE_IN_ENCRYPTION_CONTEXT屬性，則分割區和排序屬性也必須是SIGN_AND_INCLUDE_IN_ENCRYPTION_CONTEXT。

使用註釋的數據類

如果您使用 a 定義了屬性動作TableSchema，請將與屬性相關聯的屬性動作從更新@DynamoDBEncryptionSignOnly為@DynamoDBEncryptionSignAndIncludeInEncryptionConte

使用物件模型

如果您手動定義屬性動作，請SIGN_AND_INCLUDE_IN_ENCRYPTION_CONTEXT在物件模型中SIGN_ONLY將與屬性相關聯的屬性動作從更新為。

將現有SIGN_AND_INCLUDE_IN_ENCRYPTION_CONTEXT屬性變更為SIGN_ONLY

若要將現有SIGN_AND_INCLUDE_IN_ENCRYPTION_CONTEXT屬性變更為SIGN_ONLY，您必須更新屬性動作。部署更新之後，用戶端將能夠驗證寫入屬性的現有值，並繼續簽署寫入屬性的新值。寫入屬性的新值將不會包含在加密內容中。

將現有SIGN_AND_INCLUDE_IN_ENCRYPTION_CONTEXT屬性變更為之前SIGN_ONLY，請仔細考慮您的更新可能會對分支金鑰 ID 供應商的功能造成什麼影響。

使用註釋的數據類

如果您使用 a 定義了屬性動作TableSchema，請將與屬性相關聯的屬性動作從更新@DynamoDBEncryptionSignAndIncludeInEncryptionContext為@DynamoDBEncryptionSignOr

使用物件模型

如果您手動定義屬性動作，請SIGN_ONLY在物件模型中SIGN_AND_INCLUDE_IN_ENCRYPTION_CONTEXT將與屬性相關聯的屬性動作從更新為。

AWS 適用於 DynamoDB 的資料庫加密 SDK 可用程式設計語言

適用於 DynamoDB 的 AWS 資料庫加密開發套件適用於下列程式設計語言。語言專屬的程式庫有所不同，但是產生的實作都是可互通的。您可以使用一種語言實作加密，並使用另一種語言進行解密。互通性可能受到語言限制。如果是這樣，這些限制會在語言實作的主題中加以說明。

主題

- Java
- .NET

Java

我們的用戶端加密程式庫已重新命名為 AWS 資料庫加密 SDK。本開發人員指南仍提供 DynamoDB 加密用戶端的相關資訊。

本主題說明如何安裝和使用版本 3。 適用於 Java 用戶端加密程式 DynamoDB x 個。如需有關使用適用於 DynamoDB 之 AWS 資料庫加密開發套件進行程式設計的詳細資訊，請參閱上的 aws-database-encryption-sdk-dynamodb 儲存庫中的 Java 範例。 GitHub

> **Note**
>
> 下列主題著重於版本 3。 適用於 Java 用戶端加密程式 DynamoDB x 個。
> 我們的客戶端加密庫被重命名為 AWS 數據庫加密 SDK。資 AWS 料庫加密 SDK 會繼續支援舊版 DynamoDB 加密用戶端版本。

主題

必要條件

在您安裝版本 3 之前。 DynamoDB 的 Java 用戶端加密程式庫的 x 個，請確定您具有下列先決條件。

Java 開發環境

您會需要 Java 8 或更新版本。在 Oracle 網站上，移至 Java SE 下載，然後下載並安裝 Java SE 開發套件 (JDK)。

如果您使用 Oracle JDK，您還必須下載並安裝 Java Cryptography Extension (JCE) Unlimited Strength 管轄權政策檔案。

AWS SDK for Java 2.x

適用於 DynamoDB 的 AWS 資料庫加密開發套件需要的 DynamoDB 增強型用戶端模組。 AWS SDK for Java 2.x您可以安裝整個 SDK 或只安裝這個模組。

如需更新您的版本的相關資訊 AWS SDK for Java，請參閱從 1.x 版移轉至. AWS SDK for Java

可以 AWS SDK for Java 通過阿帕奇 Maven 的。您可以為整個 AWS SDK for Java或僅聲明dynamodb-enhanced模塊的依賴關係。

AWS SDK for Java 使用阿帕奇 Maven 的安裝

- 若要匯入整個 AWS SDK for Java 作為相依性，請在 pom.xml 檔案中宣告它。
- 若只要為中的 Amazon DynamoDB 模組建立相依性 AWS SDK for Java，請遵循指定特定模組的指示。groupId將設定為software.amazon.awssdk和artifactID為dynamodb-enhanced。

> **Note**
>
> 如果您使用 AWS KMS 鑰匙圈或 AWS KMS 分層密鑰環，則還需要為模塊創建依賴項。AWS KMS groupId將設定為software.amazon.awssdk和artifactID為kms。

安裝

您可以安裝版本 3。 使用下列方式適用於 DynamoDB 的 Java 用戶端加密程式庫的 x 個。

使用 Apache Maven

適用於 Java 的 Amazon DynamoDB 加密用戶端可透過 Apache Maven 使用，並具有下列相依性定義。

```
<dependency>
  <groupId>software.amazon.cryptography</groupId>
  <artifactId>aws-database-encryption-sdk-dynamodb</artifactId>
  <version>version-number</version>
</dependency>
```

使用搖籃科特林

您可以使用 Gradle 在適用於 Java 的 Amazon DynamoDB 加密用戶端上宣告相依性，方法是將下列項目新增至 Gradle 專案的相依性區段。

```
implementation("software.amazon.cryptography:aws-database-encryption-sdk-
dynamodb:version-number")
```

手動

若要安裝適用於 DynamoDB 的 Java 用戶端加密程式庫，請複製或下載 aws-database-encryption-sdk- GitHub dynamodb 儲存庫。

安裝 SDK 之後，請參閱本指南中的範例程式碼以及上的 aws-database-encryption-sdk-dynamodb 儲存庫中的 Java 範例來開始使用。 GitHub

使用 Java 用戶端加密 DynamoDB 式庫

我們的用戶端加密程式庫已重新命名為 AWS 資料庫加密 SDK。本開發人員指南仍提供 DynamoDB 加密用戶端的相關資訊。

本主題說明第 3 版中的一些函數和輔助類別。 適用於 Java 用戶端加密程式 DynamoDB x 個。

如需有關使用適用於 DynamoDB 之 Java 用戶端加密程式庫進行程式設計的詳細資訊，請參閱 Java 範例 (位於上的 aws-database-encryption-sdk-dynamodb 儲存庫中的 Java 範例)。 GitHub

主題

- 項目加密程式
- 適用於 DynamoDB 的 AWS 資料庫加密開發套件中的屬性動作
- 適用於 DynamoDB 的 AWS 資料庫加密開發套件中的加密組態
- 使用 AWS 資料庫加密 SDK 更新項目
- 解密簽署的集

項目加密程式

DynamoDB 的 AWS 資料庫加密開發套件在其核心是項目加密程式。您可以使用版本 3。 DynamoDB 的 Java 用戶端加密程式庫的 x 個，可透過下列方式加密、簽署、驗證和解密 DynamoDB 表格項目。

增強型用戶端

您可以將 DynamoDB 增強型用戶端設定為使用 Dynam `PutItem` oDB `DynamoDbEncryptionInterceptor` 請求自動加密和簽署用戶端項目。使用 DynamoDB 增強型用戶端，您可以使用帶註解的資料類別定義屬性動作。我們建議盡可能使用 DynamoDB 增強型用戶端。

DynamoDB 增強型用戶端不支援可搜尋的加密。

> **Note**
>
> 資 AWS 料庫加密 SDK 不支援巢狀屬性的註解。

低階 DynamoDB API

您可以使用設定低階 DynamoDB API，以使用 Dynam oDB `DynamoDbEncryptionInterceptor` 請求在用戶端自動加密和簽署項目。`PutItem`

您必須使用低階 DynamoDB API 才能使用可搜尋的加密。

較低級別 **`DynamoDbItemEncryptor`**

較低層級會`DynamoDbItemEncryptor`直接加密、簽署或解密並驗證您的資料表項目，而無需呼叫 DynamoDB。它不會發出 DynamoDB `PutItem` 或`GetItem`請求。例如，您可以使用較低層級直`DynamoDbItemEncryptor`接解密和驗證已擷取的 DynamoDB 項目。

較低級別`DynamoDbItemEncryptor`不支持可搜索的加密。

適用於 DynamoDB 的 AWS 資料庫加密開發套件中的屬性動作

屬性動作決定哪些屬性值已加密和簽署、哪些屬性值僅經過簽署、已簽署並包含在加密前後關聯中，以及忽略哪些值。

> **Note**
>
> 若要使用加`SIGN_AND_INCLUDE_IN_ENCRYPTION_CONTEXT`密動作，您必須使用 3.3 版或更新版本的 AWS 資料庫加密 SDK。在更新要包含的資料模型之前，請先將新版本部署至所有讀取器`SIGN_AND_INCLUDE_IN_ENCRYPTION_CONTEXT`。

如果您使用低階 DynamoDB API 或較低層級`DynamoDbItemEncryptor`，則必須手動定義屬性動作。如果您使用 DynamoDB 增強型用戶端，則可以手動定義屬性動作，也可以使用帶註解的資料類別來產生. `TableSchema` 為了簡化配置過程，我們建議使用帶註釋的數據類。當您使用已註解的資料類別時，您只需要對物件建模一次。

> **Note**
>
> 定義屬性動作之後，您必須定義哪些屬性要從簽章中排除。為了方便日後新增 future 簽署的屬性，我們建議您選擇不同的前置詞 (例如 ":「) 來識別未簽署的屬性。在定義 DynamoDB 結構描述和屬性動DO_NOTHING作時標記的所有屬性的屬性名稱中包含此前置詞。

使用帶註釋的數據類

使用帶註解的資料類別，透過 DynamoDB 增強型用戶端和指定您的屬性動作。DynamoDbEncryptionInterceptor適用於 DynamoDB 的 AWS 資料庫加密開發套件使用標準 DynamoDB 屬性註釋來定義屬性類型，以決定如何保護屬性。除了主要索引鍵 (簽署但不加密) 以外，所有屬性都預設會進行加密和簽署。

> **Note**
>
> 若要使用加SIGN_AND_INCLUDE_IN_ENCRYPTION_CONTEXT密動作，您必須使用 3.3 版或更新版本的 AWS 資料庫加密 SDK。在更新要包含的資料模型之前，請先將新版本部署至所有讀取器SIGN_AND_INCLUDE_IN_ENCRYPTION_CONTEXT。

如需有關 DynamoDB 增強型用戶端註釋的詳細指引，請參閱上 GitHub 的 aws-database-encryption-sdk-dynamodb 儲存庫中的 SimpleClass.java。

根據預設，主索引鍵屬性會經過簽署但未加密 (SIGN_ONLY)，而且所有其他屬性都會加密並簽署 (ENCRYPT_AND_SIGN)。如果您將任何屬性定義為SIGN_AND_INCLUDE_IN_ENCRYPTION_CONTEXT，則分割區和排序屬性也必須是SIGN_AND_INCLUDE_IN_ENCRYPTION_CONTEXT。若要指定例外狀況，請使用 DynamoDB 的 Java 用戶端加密程式庫中定義的加密註解。例如，如果您只想簽署特定屬性，請使用@DynamoDbEncryptionSignOnly註釋。如果您想要將特定屬性簽署並包含在加密內容中，請使用@DynamoDbEncryptionSignAndIncludeInEncryptionContext. 如果您希望特定屬性既不簽名也不加密（DO_NOTHING），請使用@DynamoDbEncryptionDoNothing註釋。

> **Note**
>
> 資 AWS 料庫加密 SDK 不支援巢狀屬性的註解。

下列範例顯示用來定義ENCRYPT_AND_SIGNSIGN_ONLY、和DO_NOTHING屬性動作的註釋。如需顯示用於定義之註釋的範例SIGN_AND_INCLUDE_IN_ENCRYPTION_CONTEXT，請參閱 SimpleClass4.java。

```
@DynamoDbBean
public class SimpleClass {

    private String partitionKey;
    private int sortKey;
    private String attribute1;
    private String attribute2;
    private String attribute3;

    @DynamoDbPartitionKey
    @DynamoDbAttribute(value = "partition_key")
    public String getPartitionKey() {
        return this.partitionKey;
    }

    public void setPartitionKey(String partitionKey) {
        this.partitionKey = partitionKey;
    }

    @DynamoDbSortKey
    @DynamoDbAttribute(value = "sort_key")
    public int getSortKey() {
        return this.sortKey;
    }

    public void setSortKey(int sortKey) {
        this.sortKey = sortKey;
    }

    public String getAttribute1() {
        return this.attribute1;
    }

    public void setAttribute1(String attribute1) {
        this.attribute1 = attribute1;
    }

    @DynamoDbEncryptionSignOnly
    public String getAttribute2() {
```

```
        return this.attribute2;
    }

    public void setAttribute2(String attribute2) {
        this.attribute2 = attribute2;
    }

    @DynamoDbEncryptionDoNothing
    public String getAttribute3() {
        return this.attribute3;
    }

    @DynamoDbAttribute(value = ":attribute3")
    public void setAttribute3(String attribute3) {
        this.attribute3 = attribute3;
    }

}
```

使用註釋的數據類來創建顯示在下面的代碼片段。TableSchema

```
final TableSchema<SimpleClass> tableSchema = TableSchema.fromBean(SimpleClass.class);
```

手動定義屬性動作

若要手動指定屬性動作，請建立一個Map物件，其中的名稱-值配對代表屬性名稱和指定動作。

指定ENCRYPT_AND_SIGN要加密和簽署屬性。指定SIGN_ONLY要簽署屬性，但不加密。指SIGN_AND_INCLUDE_IN_ENCRYPTION_CONTEXT定簽署屬性並將其包含在加密內容中。您無法在未簽署屬性的情況下加密屬性。指DO_NOTHING定忽略屬性。

分割區和排序屬性必須是SIGN_ONLY或SIGN_AND_INCLUDE_IN_ENCRYPTION_CONTEXT。如果您將任何屬性定義為SIGN_AND_INCLUDE_IN_ENCRYPTION_CONTEXT，則分割區和排序屬性也必須是SIGN_AND_INCLUDE_IN_ENCRYPTION_CONTEXT。

> **Note**
>
> 若要使用加SIGN_AND_INCLUDE_IN_ENCRYPTION_CONTEXT密動作，您必須使用 3.3 版或更新版本的 AWS 資料庫加密 SDK。在更新要包含的資料模型之前，請先將新版本部署至所有讀取器SIGN_AND_INCLUDE_IN_ENCRYPTION_CONTEXT。

```
final Map<String, CryptoAction> attributeActionsOnEncrypt = new HashMap<>();
// The partition attribute must be signed
attributeActionsOnEncrypt.put("partition_key",
 CryptoAction.SIGN_AND_INCLUDE_IN_ENCRYPTION_CONTEXT);
// The sort attribute must be signed
attributeActionsOnEncrypt.put("sort_key",
 CryptoAction.SIGN_AND_INCLUDE_IN_ENCRYPTION_CONTEXT);
attributeActionsOnEncrypt.put("attribute1", CryptoAction.ENCRYPT_AND_SIGN);
attributeActionsOnEncrypt.put("attribute2", CryptoAction.SIGN_ONLY);
attributeActionsOnEncrypt.put("attribute3",
 CryptoAction.SIGN_AND_INCLUDE_IN_ENCRYPTION_CONTEXT);
attributeActionsOnEncrypt.put(":attribute4", CryptoAction.DO_NOTHING);
```

適用於 DynamoDB 的 AWS 資料庫加密開發套件中的加密組態

使用 AWS 資料庫加密 SDK 時，您必須為 DynamoDB 表格明確定義加密組態。加密配置中所需的值取決於您是手動定義屬性動作還是使用帶註釋的資料類別定義。

下列程式碼片段使用 DynamoDB 增強型用戶端定義 DynamoDB 表格加密組態 `TableSchema`，以及允許透過不同前置詞定義的未簽署屬性。

```
final Map<String, DynamoDbEnhancedTableEncryptionConfig> tableConfigs = new
 HashMap<>();
tableConfigs.put(ddbTableName,
        DynamoDbEnhancedTableEncryptionConfig.builder()
            .logicalTableName(ddbTableName)
            .keyring(kmsKeyring)
            .allowedUnsignedAttributePrefix(unsignedAttrPrefix)
            .schemaOnEncrypt(tableSchema)
            // Optional: only required if you use beacons
            .search(SearchConfig.builder()
                    .writeVersion(1) // MUST be 1
                    .versions(beaconVersions)
                    .build())
            .build());
```

邏輯資料表名稱

您的 DynamoDB 資料表的邏輯資料表名稱。

邏輯資料表名稱會以密碼編譯方式繫結至儲存在資料表中的所有資料，以簡化 DynamoDB 還原作業。強烈建議您在第一次定義加密組態時，將 DynamoDB 表名稱指定為邏輯資料表名稱。您必須

永遠指定相同的邏輯資料表名稱。若要成功解密，邏輯資料表名稱必須與加密時指定的名稱相符。如果您的 DynamoDB 表名稱在從備份還原 DynamoDB 資料表後發生變更，邏輯資料表名稱可確保解密作業仍可辨識該資料表。

允許不帶正負號

屬性動作DO_NOTHING中標記的屬性。

允許的未簽名屬性會告訴用戶端哪些屬性會從簽章中排除。用戶端假設所有其他屬性都包含在簽章中。然後，在解密記錄時，用戶端會決定需要驗證哪些屬性，以及從您指定的允許未簽署屬性中忽略哪些屬性。您無法從允許的未簽署屬性中移除屬性。

您可以建立列出所有屬性的陣列，明確定義允許的無正負號DO_NOTHING屬性。您還可以在命名DO_NOTHING屬性時指定不同的前綴，並使用前綴告訴客戶端哪些屬性未簽名。我們強烈建議您指定不同的前置詞，因為這樣可以簡化 future 新增DO_NOTHING屬性的程序。如需詳細資訊，請參閱 更新您的資料模型。

如果您未指定所有DO_NOTHING屬性的前置詞，您可以設定allowedUnsignedAttributes陣列，明確列出用戶端在解密時遇到這些屬性時應該未簽署的所有屬性。您應該只在絕對必要時明確定義允許的未簽名屬性。

搜尋組態 (選擇性)

定SearchConfig義信標版本。

SearchConfig必須指定為使用可搜尋的加密或已簽署的信標。

演算法套件 (選擇性)

定algorithmSuiteId義了 AWS 資料庫加密 SDK 使用的演算法套件。

除非您明確指定替代演算法套件，否則資 AWS 料庫加密 SDK 會使用預設演算法套件。預設演算法套件使用 AES-GCM 演算法搭配金鑰衍生、數位簽章和金鑰承諾。雖然預設演算法套件可能適用於大多數應用程式，但您可以選擇替代演算法套件。例如，某些信任模型會由沒有數位簽章的演算法套件來滿足。如需有關「資 AWS 料庫加密 SDK」支援的演算法套件的資訊，請參閱AWS 資料庫加密 SDK 中支援的演算法套件。

若要選取不含 ECDSA 數位簽章的 AES-GCM 演算法套件，請在資料表加密組態中包含下列程式碼片段。

```
.algorithmSuiteId(
    DBEAlgorithmSuiteId.ALG_AES_256_GCM_HKDF_SHA512_COMMIT_KEY_SYMSIG_HMAC_SHA384)
```

使用 AWS 資料庫加密 SDK 更新項目

UpdateItem對於已加密或簽署的項目， AWS 資料庫加密 SDK 不支援 ddb:。若要更新加密或簽署的項目，您必須使用 ddb: PutItem。當您在請求中指定與現有項目相同的主PutItem索引鍵時，新項目會完全取代現有項目。更新項目後，您還可以使用 CLOBBER 清除和替換保存時的所有屬性。

解密簽署的集

在 AWS 資料庫加密 SDK 版本 3.0.0 和 3.1.0 版中，如果您將設定類型屬性定義為SIGN_ONLY，則會依照提供的順序規範化集合的值。DynamoDB 不會保留集合的順序。因此，包含該集合的項目的簽名驗證可能會失敗。當集合的值以與提供給 AWS 資料庫加密 SDK 不同的順序傳回時，簽章驗證會失敗，即使集合屬性包含相同的值也一樣。

> **Note**
>
> 版本 3.1.1 及更新版本的 AWS 資料庫加密 SDK 會規範化所有設定類型屬性的值，以便讀取值的順序與寫入 DynamoDB 的順序相同。

如果簽章驗證失敗，解密作業就會失敗，並傳回下列錯誤訊息。

```
加密軟件 .dbencryptionsdk 結構加密. StructuredEncryptionException：沒有符合的收件者標籤。
```

如果您收到上述錯誤訊息，並認為您嘗試解密的項目包含使用 3.0.0 或 3.1.0 版簽署的集合，請參閱上的 aws-database-encryption-sdk-dynamodb-java 存放庫DecryptWithPermute目錄，以取得有關如何成功驗證集的 GitHub 詳細資訊。

Java 範例

我們的用戶端加密程式庫已重新命名為 AWS 資料庫加密 SDK。本開發人員指南仍提供 DynamoDB 加密用戶端的相關資訊。

下列範例說明如何使用 DynamoDB 的 Java 用戶端加密程式庫來保護應用程式中的表項目。您可以在上的 aws-database-encryption-sdk-dynamodb 存儲庫中的 Java 示例中找到更多示例（並提供您自己的示例）。 GitHub

下列範例示範如何在新的未填入 Amazon DynamoDB 表格中設定 DynamoDB 的 Java 用戶端加密程式庫。如果您想要為用戶端加密設定現有的 Amazon DynamoDB 表格，請參閱。將版本 3.x 新增至現有資料表

主題

- 使用 DynamoDB 強型用戶端
- 使用低階 DynamoDB API
- 使用較低層級 DynamoDbItemEncryptor

使用 DynamoDB 強型用戶端

下列範例顯示如何使用 DynamoDB 增強型用戶端，以及使`DynamoDbEncryptionInterceptor`用AWS KMS 金鑰圈來加密 DynamoDB 表格項目，做為 DynamoDB API 呼叫的一部分。

您可以在 DynamoDB 增強型用戶端上使用任何支援的金鑰圈，但建議您盡可能使用其中一個 AWS KMS 金鑰環。

> **Note**
>
> DynamoDB 增強型用戶端不支援可搜尋的加密。`DynamoDbEncryptionInterceptor`搭配低階 DynamoDB API 使用，即可使用可搜尋的加密功能。

請參閱完整的代碼示例：EnhancedPutGetExample.java

步驟 1：建立 AWS KMS 鑰匙圈

下列範例使用`CreateAwsKmsMrkMultiKeyring`建立具有對稱加密 KMS 金 AWS KMS 鑰的金鑰圈。此方`CreateAwsKmsMrkMultiKeyring`法可確保金鑰圈能夠正確處理單一區域和多區域金鑰。

```
final MaterialProviders matProv = MaterialProviders.builder()
        .MaterialProvidersConfig(MaterialProvidersConfig.builder().build())
        .build();
final CreateAwsKmsMrkMultiKeyringInput keyringInput =
 CreateAwsKmsMrkMultiKeyringInput.builder()
        .generator(kmsKeyId)
        .build();
```

```
final IKeyring kmsKeyring = matProv.CreateAwsKmsMrkMultiKeyring(keyringInput);
```

第 2 步：從註釋的數據類創建一個表模式

下列範例會使用帶註解的資料類別來建立。TableSchema

此範例假設已註解的資料類別和屬性動作是使用 SimpleClass. java 定義的。如需註解屬性動作的詳細指引，請參閱。使用帶註釋的數據類

> **Note**
>
> 資 AWS 料庫加密 SDK 不支援巢狀屬性的註解。

```
final TableSchema<SimpleClass> schemaOnEncrypt =
 TableSchema.fromBean(SimpleClass.class);
```

步驟 3：定義哪些屬性從簽章中排除

下列範例假設所有DO_NOTHING屬性共用不同的前置詞 ":「，並使用前置詞來定義允許的未簽署屬性。用戶端假設任何具有 ":" 前置詞的屬性名稱都已從簽章中排除。如需詳細資訊，請參閱 允許不帶正負號。

```
final String unsignedAttrPrefix = ":";
```

步驟 4：建立加密設定

下列範例會定義表示 DynamoDB 表格加密組態的 tableConfigs Map。

此範例會將 DynamoDB 資料表名稱指定為邏輯資料表名稱。強烈建議您在第一次定義加密組態時，將 DynamoDB 表名稱指定為邏輯資料表名稱。如需詳細資訊，請參閱 適用於 DynamoDB 的 AWS 資料庫加密開發套件中的加密組態。

> **Note**
>
> 若要使用可搜尋的加密或已簽署的信標，您還必須SearchConfig在加密組態中包含。

```
final Map<String, DynamoDbEnhancedTableEncryptionConfig> tableConfigs = new
 HashMap<>();
```

```
tableConfigs.put(ddbTableName,
    DynamoDbEnhancedTableEncryptionConfig.builder()
        .logicalTableName(ddbTableName)
        .keyring(kmsKeyring)
        .allowedUnsignedAttributePrefix(unsignedAttrPrefix)
        .schemaOnEncrypt(tableSchema)
        .build());
```

步驟 5：建立 **DynamoDbEncryptionInterceptor**

下列範例會DynamoDbEncryptionInterceptor使用tableConfigs從步驟 4 建立新的。

```
final DynamoDbEncryptionInterceptor interceptor =
    DynamoDbEnhancedClientEncryption.CreateDynamoDbEncryptionInterceptor(
        CreateDynamoDbEncryptionInterceptorInput.builder()
            .tableEncryptionConfigs(tableConfigs)
            .build()
    );
```

步驟 6：建立新的 Dynam AWS oDB 戶端

下列範例會使用interceptor從步驟 5 開始建立新的 AWS SDK DynamoDB 用戶端。

```
final DynamoDbClient ddb = DynamoDbClient.builder()
        .overrideConfiguration(
                ClientOverrideConfiguration.builder()
                       .addExecutionInterceptor(interceptor)
                       .build())
        .build();
```

步驟 7：建立 DynamoDB 增強型用戶端並建立資料表

下列範例會使用在步驟 6 中建立的 AWS SDK DynamoDB 用戶端建立 DynamoDB 增強型用戶端，並使用附註的資料類別建立資料表。

```
final DynamoDbEnhancedClient enhancedClient = DynamoDbEnhancedClient.builder()
        .dynamoDbClient(ddb)
        .build();
final DynamoDbTable<SimpleClass> table = enhancedClient.table(ddbTableName,
 tableSchema);
```

步驟 8：加密並簽署表格項目

下列範例會使用 DynamoDB 增強型用戶端，將項目放入 DynamoDB 資料表中。在傳送至 DynamoDB 之前，該項目會經過加密並簽署用戶端。

```
final SimpleClass item = new SimpleClass();
item.setPartitionKey("EnhancedPutGetExample");
item.setSortKey(0);
item.setAttribute1("encrypt and sign me!");
item.setAttribute2("sign me!");
item.setAttribute3("ignore me!");

table.putItem(item);
```

使用低階 DynamoDB API

下列範例顯示如何使用含AWS KMS 金鑰圈的低階 DynamoDB API，在用戶端透過 DynamoDB 請求自動加密和簽署項目。`PutItem`

您可以使用任何支援的鑰匙圈，但我們建議您盡可能使用其中一個 AWS KMS 鑰匙圈。

請參閱完整的代碼示例：BasicPutGetExample.java

步驟 1：建立 AWS KMS 鑰匙圈

下列範例使用`CreateAwsKmsMrkMultiKeyring`建立具有對稱加密 KMS 金 AWS KMS 鑰的金鑰圈。此方`CreateAwsKmsMrkMultiKeyring`法可確保金鑰圈能夠正確處理單一區域和多區域金鑰。

```
final MaterialProviders matProv = MaterialProviders.builder()
         .MaterialProvidersConfig(MaterialProvidersConfig.builder().build())
         .build();
final CreateAwsKmsMrkMultiKeyringInput keyringInput =
 CreateAwsKmsMrkMultiKeyringInput.builder()
        .generator(kmsKeyId)
        .build();
final IKeyring kmsKeyring = matProv.CreateAwsKmsMrkMultiKeyring(keyringInput);
```

步驟 2：設定屬性動作

下列範例會定義 `attributeActionsOnEncrypt` Map，代表表格項目的範例屬性動作。

> **Note**
>
> 下列範例未將任何屬性定義為SIGN_AND_INCLUDE_IN_ENCRYPTION_CONTEXT。如果您指定任何SIGN_AND_INCLUDE_IN_ENCRYPTION_CONTEXT屬性，則分割區和排序屬性也必須是SIGN_AND_INCLUDE_IN_ENCRYPTION_CONTEXT。

```
final Map<String, CryptoAction> attributeActionsOnEncrypt = new HashMap<>();
// The partition attribute must be SIGN_ONLY
attributeActionsOnEncrypt.put("partition_key", CryptoAction.SIGN_ONLY);
// The sort attribute must be SIGN_ONLY
attributeActionsOnEncrypt.put("sort_key", CryptoAction.SIGN_ONLY);
attributeActionsOnEncrypt.put("attribute1", CryptoAction.ENCRYPT_AND_SIGN);
attributeActionsOnEncrypt.put("attribute2", CryptoAction.SIGN_ONLY);
attributeActionsOnEncrypt.put(":attribute3", CryptoAction.DO_NOTHING);
```

步驟 3：定義哪些屬性從簽章中排除

下列範例假設所有DO_NOTHING屬性共用不同的前置詞 ":「，並使用前置詞來定義允許的未簽署屬性。用戶端假設任何具有 ":" 前置詞的屬性名稱都已從簽章中排除。如需詳細資訊，請參閱 允許不帶正負號。

```
final String unsignedAttrPrefix = ":";
```

步驟 4：定義 DynamoDB 資料表加密組態

下列範例會定義表示此 DynamoDB 表tableConfigs格之加密組態的 Map。

此範例會將 DynamoDB 資料表名稱指定為邏輯資料表名稱。強烈建議您在第一次定義加密組態時，將 DynamoDB 表名稱指定為邏輯資料表名稱。如需詳細資訊，請參閱 適用於 DynamoDB 的 AWS 資料庫加密開發套件中的加密組態。

> **Note**
>
> 若要使用可搜尋的加密或已簽署的信標，您還必須SearchConfig在加密組態中包含。

```
final Map<String, DynamoDbTableEncryptionConfig> tableConfigs = new HashMap<>();
final DynamoDbTableEncryptionConfig config = DynamoDbTableEncryptionConfig.builder()
        .logicalTableName(ddbTableName)
```

```
        .partitionKeyName("partition_key")
        .sortKeyName("sort_key")
        .attributeActionsOnEncrypt(attributeActionsOnEncrypt)
        .keyring(kmsKeyring)
        .allowedUnsignedAttributePrefix(unsignedAttrPrefix)
        .build();
tableConfigs.put(ddbTableName, config);
```

步驟 5：建立 **DynamoDbEncryptionInterceptor**

下列範例會DynamoDbEncryptionInterceptor使用tableConfigs從步驟 4 建立。

```
DynamoDbEncryptionInterceptor interceptor = DynamoDbEncryptionInterceptor.builder()
        .config(DynamoDbTablesEncryptionConfig.builder()
                .tableEncryptionConfigs(tableConfigs)
                .build())
        .build();
```

步驟 6：建立新的 Dynam AWS oDB 戶端

下列範例會使用interceptor從步驟 5 開始建立新的 AWS SDK DynamoDB 用戶端。

```
final DynamoDbClient ddb = DynamoDbClient.builder()
        .overrideConfiguration(
                ClientOverrideConfiguration.builder()
                        .addExecutionInterceptor(interceptor)
                        .build())
        .build();
```

步驟 7：對 DynamoDB 表格項目進行加密和簽署

下列範例會定義表示範例資料表項目並將項目放入 DynamoDB 表格中的 item Map。在傳送至 DynamoDB 之前，該項目已經過加密並簽署用戶端。

```
final HashMap<String, AttributeValue> item = new HashMap<>();
item.put("partition_key", AttributeValue.builder().s("BasicPutGetExample").build());
item.put("sort_key", AttributeValue.builder().n("0").build());
item.put("attribute1", AttributeValue.builder().s("encrypt and sign me!").build());
item.put("attribute2", AttributeValue.builder().s("sign me!").build());
item.put(":attribute3", AttributeValue.builder().s("ignore me!").build());

final PutItemRequest putRequest = PutItemRequest.builder()
        .tableName(ddbTableName)
```

```
        .item(item)
        .build();

final PutItemResponse putResponse = ddb.putItem(putRequest);
```

使用較低層級 DynamoDbItemEncryptor

下列範例會示範如何使用較低層級DynamoDbItemEncryptor搭配AWS KMS 金鑰環來直接加密和簽署資料表項目。DynamoDbItemEncryptor不會將項目放入您的 DynamoDB 表格中。

您可以在 DynamoDB 增強型用戶端上使用任何支援的金鑰圈，但建議您盡可能使用其中一個 AWS KMS 金鑰環。

> **Note**
>
> 較低級別DynamoDbItemEncryptor不支持可搜索的加密。DynamoDbEncryptionInterceptor搭配低階 DynamoDB API 使用，即可使用可搜尋的加密功能。

請參閱完整的代碼示例：ItemEncryptDecryptExample.java

步驟 1：建立 AWS KMS 鑰匙圈

下列範例使用CreateAwsKmsMrkMultiKeyring建立具有對稱加密 KMS 金 AWS KMS 鑰的金鑰圈。此方CreateAwsKmsMrkMultiKeyring法可確保金鑰圈能夠正確處理單一區域和多區域金鑰。

```
final MaterialProviders matProv = MaterialProviders.builder()
         .MaterialProvidersConfig(MaterialProvidersConfig.builder().build())
         .build();
final CreateAwsKmsMrkMultiKeyringInput keyringInput =
 CreateAwsKmsMrkMultiKeyringInput.builder()
        .generator(kmsKeyId)
        .build();
final IKeyring kmsKeyring = matProv.CreateAwsKmsMrkMultiKeyring(keyringInput);
```

步驟 2：設定屬性動作

下列範例會定義 attributeActionsOnEncrypt Map，代表表格項目的範例屬性動作。

> **Note**
>
> 下列範例未將任何屬性定義為SIGN_AND_INCLUDE_IN_ENCRYPTION_CONTEXT。如果您指定任何SIGN_AND_INCLUDE_IN_ENCRYPTION_CONTEXT屬性，則分割區和排序屬性也必須是SIGN_AND_INCLUDE_IN_ENCRYPTION_CONTEXT。

```
final Map<String, CryptoAction> attributeActionsOnEncrypt = new HashMap<>();
// The partition attribute must be SIGN_ONLY
attributeActionsOnEncrypt.put("partition_key", CryptoAction.SIGN_ONLY);
// The sort attribute must be SIGN_ONLY
attributeActionsOnEncrypt.put("sort_key", CryptoAction.SIGN_ONLY);
attributeActionsOnEncrypt.put("attribute1", CryptoAction.ENCRYPT_AND_SIGN);
attributeActionsOnEncrypt.put("attribute2", CryptoAction.SIGN_ONLY);
attributeActionsOnEncrypt.put(":attribute3", CryptoAction.DO_NOTHING);
```

步驟 3：定義哪些屬性從簽章中排除

下列範例假設所有DO_NOTHING屬性共用不同的前置詞 ":「，並使用前置詞來定義允許的未簽署屬性。用戶端假設任何具有 ":" 前置詞的屬性名稱都已從簽章中排除。如需詳細資訊，請參閱 允許不帶正負號。

```
final String unsignedAttrPrefix = ":";
```

步驟 4：定義**DynamoDbItemEncryptor**配置

下列範例定義的組態DynamoDbItemEncryptor。

此範例會將 DynamoDB 資料表名稱指定為邏輯資料表名稱。強烈建議您在第一次定義加密組態時，將 DynamoDB 表名稱指定為邏輯資料表名稱。如需詳細資訊，請參閱 適用於 DynamoDB 的 AWS 資料庫加密開發套件中的加密組態。

```
final DynamoDbItemEncryptorConfig config = DynamoDbItemEncryptorConfig.builder()
        .logicalTableName(ddbTableName)
        .partitionKeyName("partition_key")
        .sortKeyName("sort_key")
        .attributeActionsOnEncrypt(attributeActionsOnEncrypt)
        .keyring(kmsKeyring)
        .allowedUnsignedAttributePrefix(unsignedAttrPrefix)
        .build();
```

步驟 5：建立 **DynamoDbItemEncryptor**

下列範例會DynamoDbItemEncryptor使用config從步驟 4 建立新的。

```
final DynamoDbItemEncryptor itemEncryptor = DynamoDbItemEncryptor.builder()
        .DynamoDbItemEncryptorConfig(config)
        .build();
```

步驟 6：直接加密並簽署表格項目

下列範例會使用直接加密和簽署項目DynamoDbItemEncryptor。DynamoDbItemEncryptor不會將項目放入您的 DynamoDB 表格中。

```
final Map<String, AttributeValue> originalItem = new HashMap<>();
originalItem.put("partition_key",
 AttributeValue.builder().s("ItemEncryptDecryptExample").build());
originalItem.put("sort_key", AttributeValue.builder().n("0").build());
originalItem.put("attribute1", AttributeValue.builder().s("encrypt and sign
 me!").build());
originalItem.put("attribute2", AttributeValue.builder().s("sign me!").build());
originalItem.put(":attribute3", AttributeValue.builder().s("ignore me!").build());

final Map<String, AttributeValue> encryptedItem = itemEncryptor.EncryptItem(
        EncryptItemInput.builder()
                .plaintextItem(originalItem)
                .build()
).encryptedItem();
```

將現有的 DynamoDB 表格設定為使用適用於 DynamoDB 的 AWS 資料庫加密開發套件

我們的用戶端加密程式庫已重新命名為 AWS 資料庫加密 SDK。本開發人員指南仍提供 DynamoDB 加密用戶端的相關資訊。

使用版本 3。 您可以將現有的 Amazon DynamoDB 用戶端加密程式庫中的 x 個設定為用戶端加密資料表。本主題提供有關新增版本 3 所必須採取的三個步驟的指引。 x 至現有、已填入的 DynamoDB 表格。

必要條件

版本 3. DynamoDB 的 Java 用戶端加密程式庫的 x 個需要中提供的 DynamoDB 增強型用戶端。 AWS SDK for Java 2.x 如果您仍然使用 DynamoDBMapper，則必須移轉至以使用 DynamoDB 增強型 AWS SDK for Java 2.x 用戶端。

請遵循從 1.x 版移轉至 2.x 版的指示。 AWS SDK for Java

然後，依照指示操作，使用 DynamoDB 增強型用戶端 API 開始使用。

在將表格設定為使用 DynamoDB 的 Java 用戶端加密程式庫之前，您需要產生`TableSchema`使用已註解的資料類別並建立增強型用戶端。

步驟 1：準備讀取和寫入加密項目

完成下列步驟，準備好資 AWS 料庫加密 SDK 用戶端以讀取和寫入加密項目。部署下列變更之後，用戶端將繼續讀取和寫入純文字項目。它不會對寫入表格的任何新項目進行加密或簽署，但一旦加密項目出現，它就能夠解密。這些變更可讓用戶端開始加密新項目。下列變更必須部署至每個讀取器，才能繼續進行下一個步驟。

1. 定義屬性動作

   更新已註解的資料類別，以包含屬性動作，這些動作會定義哪些屬性值將被加密和簽署，這些值只會經過簽署，而且會忽略這些值。

   如需有關 DynamoDB 增強型用戶端註釋的詳細指引，請參閱上 GitHub的 aws-database-encryption-sdk-dynamodb 儲存庫中的 SimpleClass.java。

   根據預設，主索引鍵屬性會經過簽署但未加密 (`SIGN_ONLY`)，而且所有其他屬性都會加密並簽署 (`ENCRYPT_AND_SIGN`)。若要指定例外狀況，請使用 DynamoDB 的 Java 用戶端加密程式庫中定義的加密註解。例如，如果您希望特定屬性成為符號，請僅使用`@DynamoDbEncryptionSignOnly`註釋。如果您想要將特定屬性簽署並包含在加密內容中，請使用`@DynamoDbEncryptionSignAndIncludeInEncryptionContext`註釋。如果您希望特定屬性既不簽名也不加密（`DO_NOTHING`），請使用`@DynamoDbEncryptionDoNothing`註釋。

   > **Note**
   >
   > 如果您指定任何`SIGN_AND_INCLUDE_IN_ENCRYPTION_CONTEXT`屬性，則分割區和排序屬性也必須是`SIGN_AND_INCLUDE_IN_ENCRYPTION_CONTEXT`。如需顯示用於定義之註釋的範例`SIGN_AND_INCLUDE_IN_ENCRYPTION_CONTEXT`，請參閱 SimpleClass4.java。

如需註釋的範例，請參閱使用帶註釋的數據類。

2. 定義要從簽章中排除的屬性

下列範例假設所有DO_NOTHING屬性共用不同的前置詞 ":「，並使用前置詞來定義允許的未簽署屬性。用戶端會假設任何具有 ":" 前置詞的屬性名稱已從簽章中排除。如需詳細資訊，請參閱 允許不帶正負號。

```
final String unsignedAttrPrefix = ":";
```

3. 建立鑰匙圈

下列範例會建立AWS KMS 金鑰圈。 AWS KMS 金鑰環使用對稱加密或非對稱 RSA AWS KMS keys 來產生、加密和解密資料金鑰。

此範例使用CreateMrkMultiKeyring建立具有對稱加密 KMS 金 AWS KMS 鑰的金鑰圈。此方CreateAwsKmsMrkMultiKeyring法可確保金鑰圈能夠正確處理單一區域和多區域金鑰。

```
final MaterialProviders matProv = MaterialProviders.builder()
        .MaterialProvidersConfig(MaterialProvidersConfig.builder().build())
        .build();
final CreateAwsKmsMrkMultiKeyringInput keyringInput =
 CreateAwsKmsMrkMultiKeyringInput.builder()
        .generator(kmsKeyId)
        .build();
final IKeyring kmsKeyring = matProv.CreateAwsKmsMrkMultiKeyring(keyringInput);
```

4. 定義表 DynamoDB 密組態

下列範例會定義表示此 DynamoDB 表tableConfigs格之加密組態的 Map。

此範例會將 DynamoDB 資料表名稱指定為邏輯資料表名稱。強烈建議您在第一次定義加密組態時，將 DynamoDB 表名稱指定為邏輯資料表名稱。如需詳細資訊，請參閱 適用於 DynamoDB 的 AWS 資料庫加密開發套件中的加密組態。

您必須指定FORCE_WRITE_PLAINTEXT_ALLOW_READ_PLAINTEXT為純文字取代。此原則會繼續讀取和寫入純文字項目、讀取加密項目，以及準備用戶端寫入加密的項目。

```
final Map<String, DynamoDbTableEncryptionConfig> tableConfigs = new HashMap<>();
final DynamoDbTableEncryptionConfig config = DynamoDbTableEncryptionConfig.builder()
        .logicalTableName(ddbTableName)
        .partitionKeyName("partition_key")
        .sortKeyName("sort_key")
```

```
        .schemaOnEncrypt(tableSchema)
        .keyring(kmsKeyring)
        .allowedUnsignedAttributePrefix(unsignedAttrPrefix)

 .plaintextOverride(PlaintextOverride.FORCE_WRITE_PLAINTEXT_ALLOW_READ_PLAINTEXT)
        .build();
tableConfigs.put(ddbTableName, config);
```

5. 建立 **DynamoDbEncryptionInterceptor**

   下列範例會DynamoDbEncryptionInterceptor使用tableConfigs從步驟 3 建立。

```
DynamoDbEncryptionInterceptor interceptor = DynamoDbEncryptionInterceptor.builder()
        .config(DynamoDbTablesEncryptionConfig.builder()
                .tableEncryptionConfigs(tableConfigs)
                .build())
        .build();
```

步驟 2：寫入加密和簽署的項目

更新DynamoDbEncryptionInterceptor配置中的純文本策略，以允許用戶端寫入加密和簽署的項目。部署下列變更之後，用戶端將根據您在步驟 1 中設定的屬性動作來加密並簽署新項目。用戶端將能夠讀取純文字項目以及加密和簽署的項目。

繼續執行步驟 3 之前，您必須先加密並簽署表格中所有現有的純文字項目。您無法執行任何單一量度或查詢來快速加密現有的純文字項目。使用對您的系統最有意義的過程。例如，您可以使用非同步程序來緩慢掃描表格，並使用您定義的屬性動作和加密配置重寫項目。若要識別資料表中的純文字項目，建議您掃描不包含 AWS Database Encryption SDK 在加密aws_dbe_head和簽署項目時新增至項目的和aws_dbe_foot屬性的所有項目。

下列範例會從步驟 1 更新資料表加密組態。您必須使用更新純文字取代。FORBID_WRITE_PLAINTEXT_ALLOW_READ_PLAINTEXT此原則會繼續讀取純文字項目，但也會讀取和寫入加密的項目。DynamoDbEncryptionInterceptor使用更新的創建一個新的tableConfigs。

```
final Map<String, DynamoDbTableEncryptionConfig> tableConfigs = new HashMap<>();
final DynamoDbTableEncryptionConfig config = DynamoDbTableEncryptionConfig.builder()
        .logicalTableName(ddbTableName)
        .partitionKeyName("partition_key")
        .sortKeyName("sort_key")
        .schemaOnEncrypt(tableSchema)
```

```
        .keyring(kmsKeyring)
        .allowedUnsignedAttributePrefix(unsignedAttrPrefix)

 .plaintextOverride(PlaintextOverride.FORBID_WRITE_PLAINTEXT_ALLOW_READ_PLAINTEXT)
        .build();
tableConfigs.put(ddbTableName, config);
```

步驟 3：只讀取加密和簽署的項目

加密並簽署所有項目之後，請更新DynamoDbEncryptionInterceptor組態中的純文字覆寫，以僅允許用戶端讀取和寫入加密和簽署的項目。部署下列變更之後，用戶端將根據您在步驟 1 中設定的屬性動作來加密並簽署新項目。用戶端只能讀取加密和簽署的項目。

下列範例會從步驟 2 更新資料表加密組態。您可以使用組態更新純文字覆寫，FORBID_WRITE_PLAINTEXT_FORBID_READ_PLAINTEXT或從組態中移除純文字原則。用戶端預設只會讀取和寫入加密和已簽署的項目。DynamoDbEncryptionInterceptor使用更新的創建一個新的tableConfigs。

```
final Map<String, DynamoDbTableEncryptionConfig> tableConfigs = new HashMap<>();
final DynamoDbTableEncryptionConfig config = DynamoDbTableEncryptionConfig.builder()
        .logicalTableName(ddbTableName)
        .partitionKeyName("partition_key")
        .sortKeyName("sort_key")
        .schemaOnEncrypt(tableSchema)
        .keyring(kmsKeyring)
        .allowedUnsignedAttributePrefix(unsignedAttrPrefix)
        // Optional: you can also remove the plaintext policy from your configuration

 .plaintextOverride(PlaintextOverride.FORBID_WRITE_PLAINTEXT_FORBID_READ_PLAINTEXT)
        .build();
tableConfigs.put(ddbTableName, config);
```

移轉至適用於 DynamoDB 的 Java 用戶端加密程式庫 3.x 版

我們的用戶端加密程式庫已重新命名為 AWS 資料庫加密 SDK。本開發人員指南仍提供 DynamoDB 加密用戶端的相關資訊。

版本 3. 適用於 DynamoDB 的 Java 用戶端加密程式庫的 x 是 2 的主要重新寫入。 x 代碼庫。它包含許多更新，例如新的結構化資料格式、改進的多租戶支援、順暢的結構描述變更，以及可搜尋的加密支援。本主題提供如何將程式碼移轉至第 3 版的指引。 x.

從 1.x 版移轉至 2.x 版

移轉至版本 2。 x，然後再移轉至版本 3。 x. 版本 2. x 將「最近提供者」的符號從變更`MostRecentProvider`為`CachingMostRecentProvider`。如果您目前使用版本 1。 具有`MostRecentProvider`符號的 DynamoDB 的 Java 用戶端加密程式庫的 x，您必須將程式碼中的符號名稱更新為。`CachingMostRecentProvider`如需詳細資訊，請參閱最新提供者的更新。

從 2.x 版移轉至 3.x 版

下列程序說明如何從版本 2 移轉程式碼。 x 轉換為版本 3。 適用於 Java 用戶端加密程式 DynamoDB x 個。

步驟 1. 準備閱讀新格式的項目

完成下列步驟，準備您的 AWS 資料庫加密 SDK 用戶端以讀取新格式的項目。部署下列變更之後，用戶端將繼續以與第 2 版相同的方式運作。 x. 您的客戶端將繼續讀取和寫入第 2 版中的項目。 x 格式，但這些變更會讓用戶端以新格式讀取項目做好準備。

AWS SDK for Java 將您的更新到 2.x 版

版本 3. DynamoDB 的 Java 用戶端加密程式庫的 x 個需要使用 DynamoDB 增強型用戶端。 DynamoDB 增強型用戶端會取代舊版中使用的 DynamoDBMapper。若要使用增強型用戶端，您必須使用 AWS SDK for Java 2.x.

請遵循從 1.x 版移轉至 2.x 版的指示。 AWS SDK for Java

如需需要哪些 AWS SDK for Java 2.x 模組的詳細資訊，請參閱必要條件。

設定您的用戶端讀取由舊版加密的項目

下列程序提供以下程式碼範例中所示步驟的概觀。

1. 建立鑰匙圈。

   金鑰圈和加密材料管理員會取代舊版 DynamoDB Java 用戶端加密程式庫中使用的加密材料提供者。

   > **Important**
   >
   > 您在建立金鑰圈時指定的包裝金鑰必須與您在第 2 版中搭配加密材料提供者使用的包裝金鑰相同。 x.

2. 在您的註釋類上創建一個表格模式。

此步驟定義開始以新格式撰寫項目時將使用的屬性動作。

如需使用新 DynamoDB 增強型用戶端的指引，請參閱AWS SDK for Java 開發人員指南TableSchema中的產生 a。

下列範例假設您已從版本 2 更新已註解的類別。 x 使用新的屬性動作註釋。如需註解屬性動作的詳細指引，請參閱。使用帶註釋的數據類

> **Note**
>
> 如果您指定任何SIGN_AND_INCLUDE_IN_ENCRYPTION_CONTEXT屬性，則分割區和排序屬性也必須是SIGN_AND_INCLUDE_IN_ENCRYPTION_CONTEXT。如需顯示用於定義之註釋的範例SIGN_AND_INCLUDE_IN_ENCRYPTION_CONTEXT，請參閱SimpleClass4.java。

3. 定義要從簽章中排除的屬性。
4. 配置在 2.x 版模型化類別中配置的屬性動作的明確對映。

   此步驟會定義用來以舊格式寫入項目的屬性動作。
5. 配置DynamoDBEncryptor您在版本 2 中使用的。 適用於 Java 用戶端加密程式 DynamoDB x 個。
6. 設定舊版行為。
7. 建立 DynamoDbEncryptionInterceptor。
8. 建立新的 Dynam AWS oDB 戶端。
9. 使用您的模型化類創建DynamoDBEnhancedClient並創建一個表。

   如需 DynamoDB 增強型用戶端的詳細資訊，請參閱建立增強型用戶端。

```
public class MigrationExampleStep1 {

    public static void MigrationStep1(String kmsKeyId, String ddbTableName, int
 sortReadValue) {
        // 1. Create a Keyring.
        //    This example creates an AWS KMS Keyring that specifies the
        //    same kmsKeyId previously used in the version 2.x configuration.
        //    It uses the 'CreateMrkMultiKeyring' method to create the
        //    keyring, so that the keyring can correctly handle both single
```

```
        //    region and Multi-Region KMS Keys.
        //    Note that this example uses the AWS SDK for Java v2 KMS client.
        final MaterialProviders matProv = MaterialProviders.builder()
                .MaterialProvidersConfig(MaterialProvidersConfig.builder().build())
                .build();
        final CreateAwsKmsMrkMultiKeyringInput keyringInput =
CreateAwsKmsMrkMultiKeyringInput.builder()
                .generator(kmsKeyId)
                .build();
        final IKeyring kmsKeyring = matProv.CreateAwsKmsMrkMultiKeyring(keyringInput);

        // 2. Create a Table Schema over your annotated class.
        //    For guidance on using the new attribute actions
        //    annotations, see SimpleClass.java in the
        //    aws-database-encryption-sdk-dynamodb GitHub repository.
        //    All primary key attributes must be signed but not encrypted
        //    and by default all non-primary key attributes
        //    are encrypted and signed (ENCRYPT_AND_SIGN).
        //    If you want a particular non-primary key attribute to be signed but
        //    not encrypted, use the 'DynamoDbEncryptionSignOnly' annotation.
        //    If you want a particular attribute to be neither signed nor encrypted
        //    (DO_NOTHING), use the 'DynamoDbEncryptionDoNothing' annotation.
        final TableSchema<SimpleClass> schemaOnEncrypt =
TableSchema.fromBean(SimpleClass.class);

        // 3. Define which attributes the client should expect to be excluded
        //    from the signature when reading items.
        //    This value represents all unsigned attributes across the entire
        //    dataset.
        final List<String> allowedUnsignedAttributes = Arrays.asList("attribute3");

        // 4. Configure an explicit map of the attribute actions configured
        //    in your version 2.x modeled class.
        final Map<String, CryptoAction> legacyActions = new HashMap<>();
        legacyActions.put("partition_key", CryptoAction.SIGN_ONLY);
        legacyActions.put("sort_key", CryptoAction.SIGN_ONLY);
        legacyActions.put("attribute1", CryptoAction.ENCRYPT_AND_SIGN);
        legacyActions.put("attribute2", CryptoAction.SIGN_ONLY);
        legacyActions.put("attribute3", CryptoAction.DO_NOTHING);

        // 5. Configure the DynamoDBEncryptor that you used in version 2.x.
        final AWSKMS kmsClient = AWSKMSClientBuilder.defaultClient();
        final DirectKmsMaterialProvider cmp = new DirectKmsMaterialProvider(kmsClient,
kmsKeyId);
```

```
        final DynamoDBEncryptor oldEncryptor = DynamoDBEncryptor.getInstance(cmp);

        // 6. Configure the legacy behavior.
        //    Input the DynamoDBEncryptor and attribute actions created in
        //    the previous steps. For Legacy Policy, use
        //    'FORCE_LEGACY_ENCRYPT_ALLOW_LEGACY_DECRYPT'. This policy continues to
read
        //    and write items using the old format, but will be able to read
        //    items written in the new format as soon as they appear.
        final LegacyOverride legacyOverride = LegacyOverride
                .builder()
                .encryptor(oldEncryptor)
                .policy(LegacyPolicy.FORCE_LEGACY_ENCRYPT_ALLOW_LEGACY_DECRYPT)
                .attributeActionsOnEncrypt(legacyActions)
                .build();

        // 7. Create a DynamoDbEncryptionInterceptor with the above configuration.
        final Map<String, DynamoDbEnhancedTableEncryptionConfig> tableConfigs = new
HashMap<>();
        tableConfigs.put(ddbTableName,
                DynamoDbEnhancedTableEncryptionConfig.builder()
                        .logicalTableName(ddbTableName)
                        .keyring(kmsKeyring)
                        .allowedUnsignedAttributes(allowedUnsignedAttributes)
                        .schemaOnEncrypt(tableSchema)
                        .legacyOverride(legacyOverride)
                        .build());
        final DynamoDbEncryptionInterceptor interceptor =
                DynamoDbEnhancedClientEncryption.CreateDynamoDbEncryptionInterceptor(
                        CreateDynamoDbEncryptionInterceptorInput.builder()
                                .tableEncryptionConfigs(tableConfigs)
                                .build()
                );

        // 8. Create a new AWS SDK DynamoDb client using the
        //    interceptor from Step 7.
        final DynamoDbClient ddb = DynamoDbClient.builder()
                .overrideConfiguration(
                        ClientOverrideConfiguration.builder()
                                .addExecutionInterceptor(interceptor)
                                .build())
                .build();

        // 9. Create the DynamoDbEnhancedClient using the AWS SDK DynamoDb client
```

```
        //    created in Step 8, and create a table with your modeled class.
        final DynamoDbEnhancedClient enhancedClient = DynamoDbEnhancedClient.builder()
                .dynamoDbClient(ddb)
                .build();
        final DynamoDbTable<SimpleClass> table = enhancedClient.table(ddbTableName,
 tableSchema);
    }
}
```

步驟 2. 以新格式寫入項目

將步驟 1 的變更部署到所有讀取器之後，請完成下列步驟，將資 AWS 料庫加密 SDK 用戶端設定為以新格式寫入項目。部署下列變更之後，用戶端會繼續讀取舊格式的項目，並開始以新格式撰寫和讀取項目。

下列程序提供以下程式碼範例中所示步驟的概觀。

1. 繼續設定金鑰圈、資料表結構描述、舊屬性動作`allowedUnsignedAttributes`，以及您在步驟 1 中`DynamoDBEncryptor`所做的一樣。
2. 將舊版行為更新為僅使用新格式寫入新項目。
3. 建立`DynamoDbEncryptionInterceptor`
4. 建立新的 Dynam AWS oDB 戶端。
5. 使用您的模型化類創建`DynamoDBEnhancedClient`並創建一個表。

   如需 DynamoDB 增強型用戶端的詳細資訊，請參閱建立增強型用戶端。

```
public class MigrationExampleStep2 {

    public static void MigrationStep2(String kmsKeyId, String ddbTableName, int
 sortReadValue) {
        // 1. Continue to configure your keyring, table schema, legacy
        //    attribute actions, allowedUnsignedAttributes, and
        //    DynamoDBEncryptor as you did in Step 1.
        final MaterialProviders matProv = MaterialProviders.builder()
                .MaterialProvidersConfig(MaterialProvidersConfig.builder().build())
                .build();
        final CreateAwsKmsMrkMultiKeyringInput keyringInput =
 CreateAwsKmsMrkMultiKeyringInput.builder()
                .generator(kmsKeyId)
                .build();
```

```
        final IKeyring kmsKeyring = matProv.CreateAwsKmsMrkMultiKeyring(keyringInput);

        final TableSchema<SimpleClass> schemaOnEncrypt =
TableSchema.fromBean(SimpleClass.class);

        final List<String> allowedUnsignedAttributes = Arrays.asList("attribute3");

        final Map<String, CryptoAction> legacyActions = new HashMap<>();
        legacyActions.put("partition_key", CryptoAction.SIGN_ONLY);
        legacyActions.put("sort_key", CryptoAction.SIGN_ONLY);
        legacyActions.put("attribute1", CryptoAction.ENCRYPT_AND_SIGN);
        legacyActions.put("attribute2", CryptoAction.SIGN_ONLY);
        legacyActions.put("attribute3", CryptoAction.DO_NOTHING);

        final AWSKMS kmsClient = AWSKMSClientBuilder.defaultClient();
        final DirectKmsMaterialProvider cmp = new DirectKmsMaterialProvider(kmsClient,
kmsKeyId);
        final DynamoDBEncryptor oldEncryptor = DynamoDBEncryptor.getInstance(cmp);

        // 2. Update your legacy behavior to only write new items using the new
        //    format.
        //    For Legacy Policy, use 'FORBID_LEGACY_ENCRYPT_ALLOW_LEGACY_DECRYPT'. This
policy
        //    continues to read items in both formats, but will only write items
        //    using the new format.
        final LegacyOverride legacyOverride = LegacyOverride
                .builder()
                .encryptor(oldEncryptor)
                .policy(LegacyPolicy.FORBID_LEGACY_ENCRYPT_ALLOW_LEGACY_DECRYPT)
                .attributeActionsOnEncrypt(legacyActions)
                .build();

        // 3. Create a DynamoDbEncryptionInterceptor with the above configuration.
        final Map<String, DynamoDbEnhancedTableEncryptionConfig> tableConfigs = new
HashMap<>();
        tableConfigs.put(ddbTableName,
                DynamoDbEnhancedTableEncryptionConfig.builder()
                        .logicalTableName(ddbTableName)
                        .keyring(kmsKeyring)
                        .allowedUnsignedAttributes(allowedUnsignedAttributes)
                        .schemaOnEncrypt(tableSchema)
                        .legacyOverride(legacyOverride)
                        .build());
        final DynamoDbEncryptionInterceptor interceptor =
```

```
                DynamoDbEnhancedClientEncryption.CreateDynamoDbEncryptionInterceptor(
                        CreateDynamoDbEncryptionInterceptorInput.builder()
                                .tableEncryptionConfigs(tableConfigs)
                                .build()
                );

        // 4. Create a new AWS SDK DynamoDb client using the
        //    interceptor from Step 3.
        final DynamoDbClient ddb = DynamoDbClient.builder()
                .overrideConfiguration(
                        ClientOverrideConfiguration.builder()
                                .addExecutionInterceptor(interceptor)
                                .build())
                .build();

        // 5. Create the DynamoDbEnhancedClient using the AWS SDK DynamoDb Client
 created
        //    in Step 4, and create a table with your modeled class.
        final DynamoDbEnhancedClient enhancedClient = DynamoDbEnhancedClient.builder()
                .dynamoDbClient(ddb)
                .build();
        final DynamoDbTable<SimpleClass> table = enhancedClient.table(ddbTableName,
 tableSchema);
    }
}
```

部署步驟 2 變更之後，您必須使用新格式重新加密資料表中的所有舊項目，才能繼續執行步驟 3。您無法執行單一指標或查詢來快速加密現有項目。使用對您的系統最有意義的過程。例如，您可以使用非同步程序來緩慢掃描表格，並使用您定義的新屬性動作和加密配置重寫項目。

步驟 3。只讀取和寫入新格式的項目

使用新格式重新加密表格中的所有項目之後，您可以從組態中移除舊版行為。完成下列步驟，將用戶端設定為只讀取和寫入新格式的項目。

下列程序提供以下程式碼範例中所示步驟的概觀。

1. 繼續設定金鑰圈、資料表結構描述，以`allowedUnsignedAttributes`及在步驟 1 中所做的一樣。移除舊有屬性動作，並`DynamoDBEncryptor`從您的組態中移除。
2. 建立 `DynamoDbEncryptionInterceptor`。
3. 建立新的 Dynam AWS oDB 戶端。
4. 使用您的模型化類創建`DynamoDBEnhancedClient`並創建一個表。

如需 DynamoDB 增強型用戶端的詳細資訊，請參閱建立增強型用戶端。

```
public class MigrationExampleStep3 {

    public static void MigrationStep3(String kmsKeyId, String ddbTableName, int
 sortReadValue) {
        // 1. Continue to configure your keyring, table schema,
        //    and allowedUnsignedAttributes as you did in Step 1.
        //    Do not include the configurations for the DynamoDBEncryptor or
        //    the legacy attribute actions.
        final MaterialProviders matProv = MaterialProviders.builder()
                .MaterialProvidersConfig(MaterialProvidersConfig.builder().build())
                .build();
        final CreateAwsKmsMrkMultiKeyringInput keyringInput =
 CreateAwsKmsMrkMultiKeyringInput.builder()
                .generator(kmsKeyId)
                .build();
        final IKeyring kmsKeyring = matProv.CreateAwsKmsMrkMultiKeyring(keyringInput);

        final TableSchema<SimpleClass> schemaOnEncrypt =
 TableSchema.fromBean(SimpleClass.class);

        final List<String> allowedUnsignedAttributes = Arrays.asList("attribute3");


        // 3. Create a DynamoDbEncryptionInterceptor with the above configuration.
        //    Do not configure any legacy behavior.
        final Map<String, DynamoDbEnhancedTableEncryptionConfig> tableConfigs = new
 HashMap<>();
        tableConfigs.put(ddbTableName,
                DynamoDbEnhancedTableEncryptionConfig.builder()
                        .logicalTableName(ddbTableName)
                        .keyring(kmsKeyring)
                        .allowedUnsignedAttributes(allowedUnsignedAttributes)
                        .schemaOnEncrypt(tableSchema)
                        .build());
        final DynamoDbEncryptionInterceptor interceptor =
                DynamoDbEnhancedClientEncryption.CreateDynamoDbEncryptionInterceptor(
                        CreateDynamoDbEncryptionInterceptorInput.builder()
                                .tableEncryptionConfigs(tableConfigs)
                                .build()
                );
```

```
        // 4. Create a new AWS SDK DynamoDb client using the
        //    interceptor from Step 3.
        final DynamoDbClient ddb = DynamoDbClient.builder()
                .overrideConfiguration(
                        ClientOverrideConfiguration.builder()
                                .addExecutionInterceptor(interceptor)
                                .build())
                .build();

        // 5. Create the DynamoDbEnhancedClient using the AWS SDK Client
        //    created in Step 4, and create a table with your modeled class.
        final DynamoDbEnhancedClient enhancedClient = DynamoDbEnhancedClient.builder()
                .dynamoDbClient(ddb)
                .build();
        final DynamoDbTable<SimpleClass> table = enhancedClient.table(ddbTableName,
 tableSchema);
    }
}
```

.NET

本主題說明如何安裝和使用版本 3。 適用於 DynamoDB 的 .NET 用戶端加密程式庫的 x 個。如需有關使用適用於 DynamoDB 之 AWS 資料庫加密開發套件進行程式設計的詳細資訊，請參閱上的 aws-database-encryption-sdk-dynamodb 儲存庫中的 .NET 範例。 GitHub

DynamoDB 的 .NET 用戶端加密程式庫適用於以 C# 和其他 .NET 程式設計語言撰寫應用程式的開發人員。Windows、macOS 和 Linux 都提供支援。

適用於 DynamoDB 的 AWS 資料庫加密 SDK 的所有程式設計語言實作都是可互通的。但是， AWS SDK for .NET 不支援清單或對映資料類型的空值。這表示，如果您使用 DynamoDB 的 Java 用戶端加密程式庫撰寫包含清單或對映資料類型空值的項目，則無法使用 DynamoDB 的 .NET 用戶端加密程式庫來解密和讀取該項目。

主題

- 為 DynamoDB 安裝 .NET 用戶端加密程式庫
- 使用. NET 進行偵
- 使用適用於 DynamoDB 的 .NET 用戶端加密程式庫
- .NET 範例
- 將現有的 DynamoDB 表格設定為使用適用於 DynamoDB 的 AWS 資料庫加密開發套件

為 DynamoDB 安裝 .NET 用戶端加密程式庫

DynamoDB 的 .NET 用戶端加密程式庫可作為 AWS 加密技術使用。 DbEncryption開發套件。 DynamoDb封裝於 NuGet。如需有關安裝和建置程式庫的詳細資訊，請參閱-dynamodb 儲存庫中的 .NET Readme.md 檔案。 aws-database-encryption-sdk即使您未使用 AWS Key Management Service (AWS KMS) 金鑰，DynamoDB 的 .NET 用戶端加密程式庫 AWS SDK for .NET 也需要使用。隨套件 AWS SDK for .NET 一起安 NuGet 裝。

版本 3. 適用於 .NET 用戶端加密程 DynamoDB 庫的 x 支援 .NET 6.0 及更新版本。

使用. NET 進行偵

DynamoDB 的 .NET 用戶端加密程式庫不會產生任何記錄。DynamoDB 的 .NET 用戶端加密程式庫中的例外狀況會產生例外狀況訊息，但不會產生堆疊追蹤。

若要協助您偵錯，請務必啟用 AWS SDK for .NET. 中的記錄和錯誤訊息 AWS SDK for .NET 可協助您區分發生的錯誤與 DynamoDB 的 AWS SDK for .NET .NET 用戶端加密程式庫中的錯誤。如需有關 AWS SDK for .NET 記錄的說明，請參閱AWSLogging開AWS SDK for .NET 發人員指南中的。若要查看主題，請展開 [開啟以檢視 .NET 架構內容] 區段。)

使用適用於 DynamoDB 的 .NET 用戶端加密程式庫

本主題說明第 3 版中的一些函數和輔助類別。 適用於 DynamoDB 的 .NET 用戶端加密程式庫的 x 個。

如需有關使用 DynamoDB 之 .NET 用戶端加密程式庫進行程式設計的詳細資訊，請參閱上的 aws-database-encryption-sdk-dynamodb 儲存庫中的 .NET 範例。 GitHub

主題

- 項目加密程式
- 適用於 DynamoDB 的 AWS 資料庫加密開發套件中的屬性動作
- 適用於 DynamoDB 的 AWS 資料庫加密開發套件中的加密組態
- 使用 AWS 資料庫加密 SDK 更新項目

項目加密程式

DynamoDB 的 AWS 資料庫加密開發套件在其核心是項目加密程式。您可以使用版本 3。 DynamoDB 的 .NET 用戶端加密程式庫的 x 個，可透過下列方式加密、簽署、驗證和解密 DynamoDB 表格項目。

適用於 DynamoDB API 的低階 AWS 資料庫加密開發套件

您可以使用表格加密組態來建構 DynamoDB 用戶端，該用戶端會自動對 DynamoDB 請求的項目進行加密和簽署。PutItem您可以直接使用此客戶端，也可以構建文檔模型或對象持久性模型。

您必須使用適用於 DynamoDB API 的低階 AWS 資料庫加密開發套件，才能使用可搜尋的加密。

較低級別 **DynamoDbItemEncryptor**

較低層級會DynamoDbItemEncryptor直接加密、簽署或解密並驗證您的資料表項目，而無需呼叫 DynamoDB。它不會發出 DynamoDB PutItem 或GetItem請求。例如，您可以使用較低層級直DynamoDbItemEncryptor接解密和驗證已擷取的 DynamoDB 項目。如果您使用較低層級的程式設計模型DynamoDbItemEncryptor，我們建議您使用 AWS SDK for .NET 提供與 DynamoDB 通訊的低階程式設計模型。

較低級別DynamoDbItemEncryptor不支持可搜索的加密。

適用於 DynamoDB 的 AWS 資料庫加密開發套件中的屬性動作

屬性動作會決定哪些屬性值已加密和簽署、哪些屬性值僅經過簽署、已簽署並包含在加密前後關聯中，以及忽略哪些值。

若要使用 .NET 用戶端指定屬性動作，請使用物件模型手動定義屬性動作。透過建立一個Dictionary物件，其中名稱-值對代表屬性名稱和指定動作來指定屬性動作。

指定ENCRYPT_AND_SIGN要加密和簽署屬性。指定SIGN_ONLY要簽署屬性，但不加密。指SIGN_AND_INCLUDE_IN_ENCRYPTION_CONTEXT定簽署屬性並將其包含在加密內容中。您無法在未簽署屬性的情況下加密屬性。指DO_NOTHING定忽略屬性。

分割區和排序屬性必須是SIGN_ONLY或SIGN_AND_INCLUDE_IN_ENCRYPTION_CONTEXT。如果您將任何屬性定義為SIGN_AND_INCLUDE_IN_ENCRYPTION_CONTEXT，則分割區和排序屬性也必須是SIGN_AND_INCLUDE_IN_ENCRYPTION_CONTEXT。

> **Note**
>
> 定義屬性動作之後，您必須定義哪些屬性要從簽章中排除。為了方便日後新增 future 簽署的屬性，我們建議您選擇不同的前置詞 (例如 ":「) 來識別未簽署的屬性。在定義 DynamoDB 結構描述和屬性動DO_NOTHING作時標記的所有屬性的屬性名稱中包含此前置詞。

下列物件模型示範如何使用 .NET 用戶端指定ENCRYPT_AND_SIGNSIGN_ONLYSIGN_AND_INCLUDE_IN_ENCRYPTION_CONTEXT、、和DO_NOTHING屬性動作。此範例使用字首 ":" 來識別DO_NOTHING屬性。

> **Note**
>
> 若要使用加SIGN_AND_INCLUDE_IN_ENCRYPTION_CONTEXT密動作，您必須使用 3.3 版或更新版本的 AWS 資料庫加密 SDK。在更新要包含的資料模型之前，請先將新版本部署至所有讀取器SIGN_AND_INCLUDE_IN_ENCRYPTION_CONTEXT。

```
var attributeActionsOnEncrypt = new Dictionary<string, CryptoAction>
{
    ["partition_key"] = CryptoAction.SIGN_AND_INCLUDE_IN_ENCRYPTION_CONTEXT, // The
 partition attribute must be signed
    ["sort_key"] = CryptoAction.SIGN_AND_INCLUDE_IN_ENCRYPTION_CONTEXT, // The sort
 attribute must be signed
    ["attribute1"] = CryptoAction.ENCRYPT_AND_SIGN,
    ["attribute2"] = CryptoAction.SIGN_ONLY,
    ["attribute3"] = CryptoAction.SIGN_AND_INCLUDE_IN_ENCRYPTION_CONTEXT,
    [":attribute4"] = CryptoAction.DO_NOTHING
};
```

適用於 DynamoDB 的 AWS 資料庫加密開發套件中的加密組態

使用 AWS 資料庫加密 SDK 時，您必須為 DynamoDB 表格明確定義加密組態。加密配置中所需的值取決於您是手動定義屬性動作還是使用帶註釋的資料類別定義。

下列程式碼片段使用適用於 DynamoDB API 的低階 AWS 資料庫加密 SDK 定義 DynamoDB 表格加密組態，並允許透過不同前置詞定義的未簽署屬性。

```
Dictionary<String, DynamoDbTableEncryptionConfig> tableConfigs =
    new Dictionary<String, DynamoDbTableEncryptionConfig>();
DynamoDbTableEncryptionConfig config = new DynamoDbTableEncryptionConfig
{
    LogicalTableName = ddbTableName,
    PartitionKeyName = "partition_key",
    SortKeyName = "sort_key",
    AttributeActionsOnEncrypt = attributeActionsOnEncrypt,
    Keyring = kmsKeyring,
```

```
    AllowedUnsignedAttributePrefix = unsignAttrPrefix,
    // Optional: SearchConfig only required if you use beacons
    Search = new SearchConfig
    {
        WriteVersion = 1, // MUST be 1
        Versions = beaconVersions
    }
};
tableConfigs.Add(ddbTableName, config);
```

邏輯資料表名稱

您的 DynamoDB 資料表的邏輯資料表名稱。

邏輯資料表名稱會以密碼編譯方式繫結至儲存在資料表中的所有資料，以簡化 DynamoDB 還原作業。強烈建議您在第一次定義加密組態時，將 DynamoDB 表名稱指定為邏輯資料表名稱。您必須永遠指定相同的邏輯資料表名稱。若要成功解密，邏輯資料表名稱必須與加密時指定的名稱相符。如果您的 DynamoDB 表名稱在從備份還原 DynamoDB 資料表後變更，邏輯資料表名稱可確保解密作業仍可辨識該資料表。

允許不帶正負號

屬性動作DO_NOTHING中標記的屬性。

允許的未簽名屬性會告訴用戶端哪些屬性會從簽章中排除。用戶端假設所有其他屬性都包含在簽章中。然後，在解密記錄時，用戶端會決定需要驗證哪些屬性，以及從您指定的允許未簽署屬性中忽略哪些屬性。您無法從允許的未簽署屬性中移除屬性。

您可以建立列出所有屬性的陣列，明確定義允許的無正負號DO_NOTHING屬性。您還可以在命名DO_NOTHING屬性時指定不同的前綴，並使用前綴告訴客戶端哪些屬性未簽名。我們強烈建議您指定不同的前置詞，因為這樣可以簡化 future 新增DO_NOTHING屬性的程序。如需詳細資訊，請參閱 更新您的資料模型。

如果您未指定所有DO_NOTHING屬性的前置詞，您可以設定allowedUnsignedAttributes陣列，明確列出用戶端在解密時遇到這些屬性時應該未簽署的所有屬性。您應該只在絕對必要時明確定義允許的未簽名屬性。

搜尋組態 (選擇性)

定SearchConfig義信標版本。

SearchConfig必須指定為使用可搜尋的加密或已簽署的信標。

演算法套件 (選擇性)

定`algorithmSuiteId`義了 AWS 資料庫加密 SDK 使用的演算法套件。

除非您明確指定替代演算法套件，否則資 AWS 料庫加密 SDK 會使用預設演算法套件。預設演算法套件使用 AES-GCM 演算法搭配金鑰衍生、數位簽章和金鑰承諾。雖然預設演算法套件可能適用於大多數應用程式，但您可以選擇替代演算法套件。例如，某些信任模型會由沒有數位簽章的演算法套件來滿足。如需有關「資 AWS 料庫加密 SDK」支援的演算法套件的資訊，請參閱AWS 資料庫加密 SDK 中支援的演算法套件。

若要選取不含 ECDSA 數位簽章的 AES-GCM 演算法套件，請在資料表加密組態中包含下列程式碼片段。

```
AlgorithmSuiteId =
 DBEAlgorithmSuiteId.ALG_AES_256_GCM_HKDF_SHA512_COMMIT_KEY_SYMSIG_HMAC_SHA384
```

使用 AWS 資料庫加密 SDK 更新項目

UpdateItem對於包含加密或已簽署屬性的項目， AWS 資料庫加密 SDK 不支援 ddb:。若要更新加密或已簽署的屬性，您必須使用 ddb: PutItem。當您在請求中指定與現有項目相同的主`PutItem`索引鍵時，新項目會完全取代現有項目。更新項目後，您還可以使用 CLOBBER 清除和替換保存時的所有屬性。

.NET 範例

下列範例說明如何使用 DynamoDB 的 .NET 用戶端加密程式庫來保護應用程式中的表項目。要查找更多示例（並提供您自己的示例），請參閱上的 aws-database-encryption-sdk-dynamodb 存儲庫中的 .NET 示例。 GitHub

下列範例示範如何在新的未填入 Amazon DynamoDB 表格中設定 DynamoDB 的 .NET 用戶端加密程式庫。如果您想要為用戶端加密設定現有的 Amazon DynamoDB 表格，請參閱。將版本 3.x 新增至現有資料表

主題

- 使用適用於 DynamoDB API 的低階 AWS 資料庫加密開發套件
- 使用較低層級 DynamoDbItemEncryptor

使用適用於 DynamoDB API 的低階 AWS 資料庫加密開發套件

下列範例顯示如何使用適用於 DynamoDB API 的低階 AWS 資料庫加密 SDK 搭配AWS KMS 金鑰圈，透過 DynamoDB 請求在用戶端自動加密和簽署項目。PutItem

您可以使用任何支援的鑰匙圈，但我們建議您盡可能使用其中一個 AWS KMS 鑰匙圈。

請參閱完整的程式碼範例：BasicPutGetExample.cs

步驟 1：建立 AWS KMS 鑰匙圈

下列範例使用CreateAwsKmsMrkMultiKeyring建立具有對稱加密 KMS 金 AWS KMS 鑰的金鑰圈。此方CreateAwsKmsMrkMultiKeyring法可確保金鑰圈能夠正確處理單一區域和多區域金鑰。

```
var matProv = new MaterialProviders(new MaterialProvidersConfig());
var keyringInput = new CreateAwsKmsMrkMultiKeyringInput { Generator = kmsKeyId };
var kmsKeyring = matProv.CreateAwsKmsMrkMultiKeyring(keyringInput);
```

步驟 2：設定屬性動作

下列範例會定義代表表格項目範例屬性動作的 attributeActionsOnEncrypt Dictionary。

> **Note**
>
> 下列範例未將任何屬性定義為SIGN_AND_INCLUDE_IN_ENCRYPTION_CONTEXT。如果您指定任何SIGN_AND_INCLUDE_IN_ENCRYPTION_CONTEXT屬性，則分割區和排序屬性也必須是SIGN_AND_INCLUDE_IN_ENCRYPTION_CONTEXT。

```
var attributeActionsOnEncrypt = new Dictionary<string, CryptoAction>
{
    ["partition_key"] = CryptoAction.SIGN_ONLY, // The partition attribute must be
 SIGN_ONLY
    ["sort_key"] = CryptoAction.SIGN_ONLY, // The sort attribute must be SIGN_ONLY
    ["attribute1"] = CryptoAction.ENCRYPT_AND_SIGN,
    ["attribute2"] = CryptoAction.SIGN_ONLY,
    [":attribute3"] = CryptoAction.DO_NOTHING
};
```

步驟 3：定義哪些屬性從簽章中排除

下列範例假設所有DO_NOTHING屬性共用不同的前置詞 ":「，並使用前置詞來定義允許的未簽署屬性。用戶端假設任何具有 ":" 前置詞的屬性名稱已從簽章中排除。如需詳細資訊，請參閱 允許不帶正負號。

```
const String unsignAttrPrefix = ":";
```

步驟 4：定義 DynamoDB 資料表加密組態

下列範例會定義表示此 DynamoDB 表tableConfigs格之加密組態的 Map。

此範例會將 DynamoDB 資料表名稱指定為邏輯資料表名稱。強烈建議您在第一次定義加密組態時，將 DynamoDB 表名稱指定為邏輯資料表名稱。如需詳細資訊，請參閱 適用於 DynamoDB 的 AWS 資料庫加密開發套件中的加密組態。

> **Note**
>
> 若要使用可搜尋的加密或已簽署的信標，您還必須SearchConfig在加密組態中包含。

```
Dictionary<String, DynamoDbTableEncryptionConfig> tableConfigs =
    new Dictionary<String, DynamoDbTableEncryptionConfig>();
DynamoDbTableEncryptionConfig config = new DynamoDbTableEncryptionConfig
{
    LogicalTableName = ddbTableName,
    PartitionKeyName = "partition_key",
    SortKeyName = "sort_key",
    AttributeActionsOnEncrypt = attributeActionsOnEncrypt,
    Keyring = kmsKeyring,
    AllowedUnsignedAttributePrefix = unsignAttrPrefix
};
tableConfigs.Add(ddbTableName, config);
```

步驟 5：建立新的 Dynam AWS oDB 戶端

下列範例會使用TableEncryptionConfigs從步驟 4 開始建立新的 AWS SDK DynamoDB 用戶端。

```
var ddb = new Client.DynamoDbClient(
```

```
    new DynamoDbTablesEncryptionConfig { TableEncryptionConfigs = tableConfigs });
```

步驟 6：對 DynamoDB 表格項目進行加密和簽署

下列範例會定義代表範例資料表項目的`item`字典，並將項目置於 DynamoDB 資料表中。在傳送至 DynamoDB 之前，該項目已經過加密並簽署用戶端。

```
var item = new Dictionary<String, AttributeValue>
{
    ["partition_key"] = new AttributeValue("BasicPutGetExample"),
    ["sort_key"] = new AttributeValue { N = "0" },
    ["attribute1"] = new AttributeValue("encrypt and sign me!"),
    ["attribute2"] = new AttributeValue("sign me!"),
    [":attribute3"] = new AttributeValue("ignore me!")
};

PutItemRequest putRequest = new PutItemRequest
{
    TableName = ddbTableName,
    Item = item
};

PutItemResponse putResponse = await ddb.PutItemAsync(putRequest);
```

使用較低層級 **`DynamoDbItemEncryptor`**

下列範例會示範如何使用較低層級`DynamoDbItemEncryptor`搭配AWS KMS 金鑰環來直接加密和簽署資料表項目。`DynamoDbItemEncryptor`不會將項目放入您的 DynamoDB 表格中。

您可以在 DynamoDB 增強型用戶端上使用任何支援的金鑰圈，但建議您盡可能使用其中一個 AWS KMS 金鑰環。

> **Note**
>
> 較低級別`DynamoDbItemEncryptor`不支持可搜索的加密。使用適用於 DynamoDB API 的低階 AWS 資料庫加密開發套件，以使用可搜尋的加密。

請參閱完整的程式碼範例：ItemEncryptDecryptExample.cs

步驟 1：建立 AWS KMS 鑰匙圈

下列範例使用CreateAwsKmsMrkMultiKeyring建立具有對稱加密 KMS 金 AWS KMS 鑰的金鑰圈。此方CreateAwsKmsMrkMultiKeyring法可確保金鑰圈能夠正確處理單一區域和多區域金鑰。

```
var matProv = new MaterialProviders(new MaterialProvidersConfig());
var keyringInput = new CreateAwsKmsMrkMultiKeyringInput { Generator = kmsKeyId };
var kmsKeyring = matProv.CreateAwsKmsMrkMultiKeyring(keyringInput);
```

步驟 2：設定屬性動作

下列範例會定義代表表格項目範例屬性動作的 attributeActionsOnEncrypt Dictionary。

> **Note**
>
> 下列範例未將任何屬性定義為SIGN_AND_INCLUDE_IN_ENCRYPTION_CONTEXT。如果您指定任何SIGN_AND_INCLUDE_IN_ENCRYPTION_CONTEXT屬性，則分割區和排序屬性也必須是SIGN_AND_INCLUDE_IN_ENCRYPTION_CONTEXT。

```
var attributeActionsOnEncrypt = new Dictionary<String, CryptoAction>
{
    ["partition_key"] = CryptoAction.SIGN_ONLY, // The partition attribute must be
 SIGN_ONLY
    ["sort_key"] = CryptoAction.SIGN_ONLY, // The sort attribute must be SIGN_ONLY
    ["attribute1"] = CryptoAction.ENCRYPT_AND_SIGN,
    ["attribute2"] = CryptoAction.SIGN_ONLY,
    [":attribute3"] = CryptoAction.DO_NOTHING
};
```

步驟 3：定義哪些屬性從簽章中排除

下列範例假設所有DO_NOTHING屬性共用不同的前置詞 ":「，並使用前置詞來定義允許的未簽署屬性。用戶端假設任何具有 ":" 前置詞的屬性名稱已從簽章中排除。如需詳細資訊，請參閱 允許不帶正負號。

```
String unsignAttrPrefix = ":";
```

步驟 4：定義DynamoDbItemEncryptor配置

下列範例定義的組態DynamoDbItemEncryptor。

此範例會將 DynamoDB 資料表名稱指定為邏輯資料表名稱。強烈建議您在第一次定義加密組態時，將 DynamoDB 表名稱指定為邏輯資料表名稱。如需詳細資訊，請參閱 適用於 DynamoDB 的 AWS 資料庫加密開發套件中的加密組態。

```
var config = new DynamoDbItemEncryptorConfig
{
    LogicalTableName = ddbTableName,
    PartitionKeyName = "partition_key",
    SortKeyName = "sort_key",
    AttributeActionsOnEncrypt = attributeActionsOnEncrypt,
    Keyring = kmsKeyring,
    AllowedUnsignedAttributePrefix = unsignAttrPrefix
};
```

步驟 5：建立 DynamoDbItemEncryptor

下列範例會DynamoDbItemEncryptor使用config從步驟 4 建立新的。

```
var itemEncryptor = new DynamoDbItemEncryptor(config);
```

步驟 6：直接加密並簽署表格項目

下列範例會使用直接加密項目並簽署項目DynamoDbItemEncryptor。DynamoDbItemEncryptor不會將項目放入您的 DynamoDB 表格中。

```
var originalItem = new Dictionary<String, AttributeValue>
{
    ["partition_key"] = new AttributeValue("ItemEncryptDecryptExample"),
    ["sort_key"] = new AttributeValue { N = "0" },
    ["attribute1"] = new AttributeValue("encrypt and sign me!"),
    ["attribute2"] = new AttributeValue("sign me!"),
    [":attribute3"] = new AttributeValue("ignore me!")
};

var encryptedItem = itemEncryptor.EncryptItem(
    new EncryptItemInput { PlaintextItem = originalItem }
).EncryptedItem;
```

將現有的 DynamoDB 表格設定為使用適用於 DynamoDB 的 AWS 資料庫加密開發套件

使用版本 3。 在 DynamoDB 的 .NET 用戶端加密程式庫中，您可以設定現有的 Amazon DynamoDB 表格以進行用戶端加密。本主題提供有關新增版本 3 所必須採取的三個步驟的指引。 x 至現有、已填入的 DynamoDB 表格。

步驟 1：準備讀取和寫入加密項目

完成下列步驟，準備好資 AWS 料庫加密 SDK 用戶端以讀取和寫入加密項目。部署下列變更之後，用戶端將繼續讀取和寫入純文字項目。它不會對寫入表格的任何新項目進行加密或簽署，但一旦加密項目出現，它就能夠解密。這些變更可讓用戶端開始加密新項目。下列變更必須部署至每個讀取器，才能繼續進行下一個步驟。

1. 定義屬性動作

   建立物件模型以定義要加密和簽署哪些屬性值，這些值只會經過簽署，而且會忽略這些值。

   根據預設，主索引鍵屬性會經過簽署但未加密 (SIGN_ONLY)，而且所有其他屬性都會加密並簽署 (ENCRYPT_AND_SIGN)。

   指定ENCRYPT_AND_SIGN要加密和簽署屬性。指定SIGN_ONLY要簽署屬性，但不加密。指SIGN_AND_INCLUDE_IN_ENCRYPTION_CONTEXT定簽署和屬性，並將其包含在加密內容中。您無法在未簽署屬性的情況下加密屬性。指DO_NOTHING定忽略屬性。如需詳細資訊，請參閱 適用於 DynamoDB 的 AWS 資料庫加密開發套件中的屬性動作。

   > **Note**
   >
   > 如果您指定任何SIGN_AND_INCLUDE_IN_ENCRYPTION_CONTEXT屬性，則分割區和排序屬性也必須是SIGN_AND_INCLUDE_IN_ENCRYPTION_CONTEXT。

```
var attributeActionsOnEncrypt = new Dictionary<string, CryptoAction>
{
    ["partition_key"] = CryptoAction.SIGN_ONLY, // The partition attribute must be
 SIGN_ONLY
    ["sort_key"] = CryptoAction.SIGN_ONLY, // The sort attribute must be SIGN_ONLY
    ["attribute1"] = CryptoAction.ENCRYPT_AND_SIGN,
    ["attribute2"] = CryptoAction.SIGN_ONLY,
    [":attribute3"] = CryptoAction.DO_NOTHING
};
```

2. 定義要從簽章中排除的屬性

下列範例假設所有DO_NOTHING屬性共用不同的前置詞 ":「，並使用前置詞來定義允許的未簽署屬性。用戶端會假設任何具有 ":" 前置詞的屬性名稱已從簽章中排除。如需詳細資訊，請參閱 允許不帶正負號。

```
const String unsignAttrPrefix = ":";
```

3. 建立鑰匙圈

下列範例會建立AWS KMS 金鑰圈。 AWS KMS 金鑰環使用對稱加密或非對稱 RSA AWS KMS keys 來產生、加密和解密資料金鑰。

此範例使用CreateMrkMultiKeyring建立具有對稱加密 KMS 金 AWS KMS 鑰的金鑰圈。此方CreateAwsKmsMrkMultiKeyring法可確保金鑰圈能夠正確處理單一區域和多區域金鑰。

```
var matProv = new MaterialProviders(new MaterialProvidersConfig());
var keyringInput = new CreateAwsKmsMrkMultiKeyringInput { Generator = kmsKeyId };
var kmsKeyring = matProv.CreateAwsKmsMrkMultiKeyring(keyringInput);
```

4. 定義表 DynamoDB 密組態

下列範例會定義表示此 DynamoDB 表tableConfigs格之加密組態的 Map。

此範例會將 DynamoDB 資料表名稱指定為邏輯資料表名稱。強烈建議您在第一次定義加密組態時，將 DynamoDB 表名稱指定為邏輯資料表名稱。

您必須指定FORCE_WRITE_PLAINTEXT_ALLOW_READ_PLAINTEXT為純文字取代。此原則會繼續讀取和寫入純文字項目、讀取加密項目，以及準備用戶端寫入加密的項目。

如需資料表加密組態中包含的值的詳細資訊，請參閱適用於 DynamoDB 的 AWS 資料庫加密開發套件中的加密組態。

```
Dictionary<String, DynamoDbTableEncryptionConfig> tableConfigs =
    new Dictionary<String, DynamoDbTableEncryptionConfig>();
DynamoDbTableEncryptionConfig config = new DynamoDbTableEncryptionConfig
{
    LogicalTableName = ddbTableName,
    PartitionKeyName = "partition_key",
    SortKeyName = "sort_key",
    AttributeActionsOnEncrypt = attributeActionsOnEncrypt,
    Keyring = kmsKeyring,
```

```
    AllowedUnsignedAttributePrefix = unsignAttrPrefix,
    PlaintextOverride = FORCE_WRITE_PLAINTEXT_ALLOW_READ_PLAINTEXT
};
tableConfigs.Add(ddbTableName, config);
```

5. 建立新的 Dynam AWS oDB 戶端

   他以下示例使用TableEncryptionConfigs從步驟 4 開始創建一個新的 AWS SDK DynamoDB 客戶端。

```
var ddb = new Client.DynamoDbClient(
    new DynamoDbTablesEncryptionConfig { TableEncryptionConfigs = tableConfigs });
```

步驟 2：寫入加密和簽署的項目

更新資料表加密組態中的純文字原則，以允許用戶端寫入加密和簽署的項目。部署下列變更之後，用戶端將根據您在步驟 1 中設定的屬性動作來加密並簽署新項目。用戶端將能夠讀取純文字項目以及加密和簽署的項目。

繼續執行步驟 3 之前，您必須先加密並簽署表格中所有現有的純文字項目。您無法執行任何單一量度或查詢來快速加密現有的純文字項目。使用對您的系統最有意義的過程。例如，您可以使用非同步程序來緩慢掃描表格，並使用您定義的屬性動作和加密配置重寫項目。若要識別資料表中的純文字項目，建議您掃描不包含 AWS Database Encryption SDK 在加密aws_dbe_head和簽署項目時新增至項目的和aws_dbe_foot屬性的所有項目。

下列範例會從步驟 1 更新資料表加密組態。您必須使用更新純文字取代。FORBID_WRITE_PLAINTEXT_ALLOW_READ_PLAINTEXT此原則會繼續讀取純文字項目，但也會讀取和寫入加密的項目。使用 AWS 更新的功能建立新的 SDK 用戶端。TableEncryptionConfigs

```
Dictionary<String, DynamoDbTableEncryptionConfig> tableConfigs =
    new Dictionary<String, DynamoDbTableEncryptionConfig>();
DynamoDbTableEncryptionConfig config = new DynamoDbTableEncryptionConfig
{
    LogicalTableName = ddbTableName,
    PartitionKeyName = "partition_key",
    SortKeyName = "sort_key",
    AttributeActionsOnEncrypt = attributeActionsOnEncrypt,
    Keyring = kmsKeyring,
    AllowedUnsignedAttributePrefix = unsignAttrPrefix,
    PlaintextOverride = FORBID_WRITE_PLAINTEXT_ALLOW_READ_PLAINTEXT
};
```

```
tableConfigs.Add(ddbTableName, config);
```

步驟 3：只讀取加密和簽署的項目

加密並簽署所有項目之後，請更新資料表加密組態中的純文字覆寫，以僅允許用戶端讀取和寫入加密和簽署的項目。部署下列變更之後，用戶端將根據您在步驟 1 中設定的屬性動作來加密並簽署新項目。用戶端只能讀取加密和簽署的項目。

下列範例會從步驟 2 更新資料表加密組態。您可以使用組態更新純文字覆寫，FORBID_WRITE_PLAINTEXT_FORBID_READ_PLAINTEXT或從組態中移除純文字原則。用戶端預設只會讀取和寫入加密和已簽署的項目。使用 AWS 更新的功能建立新的 SDK 用戶端。TableEncryptionConfigs

```
Dictionary<String, DynamoDbTableEncryptionConfig> tableConfigs =
    new Dictionary<String, DynamoDbTableEncryptionConfig>();
DynamoDbTableEncryptionConfig config = new DynamoDbTableEncryptionConfig
{
    LogicalTableName = ddbTableName,
    PartitionKeyName = "partition_key",
    SortKeyName = "sort_key",
    AttributeActionsOnEncrypt = attributeActionsOnEncrypt,
    Keyring = kmsKeyring,
    AllowedUnsignedAttributePrefix = unsignAttrPrefix,
    // Optional: you can also remove the plaintext policy from your configuration
    PlaintextOverride = FORBID_WRITE_PLAINTEXT_FORBID_READ_PLAINTEXT
};
tableConfigs.Add(ddbTableName, config);
```

舊 DynamoDB 加密用戶端

2023 年 6 月 9 日，我們的用戶端加密程式庫重新命名為 AWS 資料庫加密 SDK。資 AWS 料庫加密 SDK 會繼續支援舊版 DynamoDB 加密用戶端版本。如需隨重新命名而變更之用戶端加密程式庫不同部分的詳細資訊，請參閱亞馬遜加密用戶端重新命名。

若要移轉至最新版本的 DynamoDB 用戶端加密程式庫，請參閱。移轉至 3.x 版

主題

- AWS 適用於 DynamoDB 版本支援的資料庫加密 SDK
- 加密用戶端的運作方式
- 亞馬遜加密用戶端概念

- 密碼材料供應商
- Amazon DynamoDB 密用戶端可用的程式設計語言
- 變更您的資料模型
- 疑難排解 DynamoDB 加密用戶端應用程式中的問題

AWS 適用於 DynamoDB 版本支援的資料庫加密 SDK

舊版章節中的主題提供了有關版本 1 的資訊。 X-2. 適用於 Java 和版本 1 的 x 個加密用戶端。 X — 3. 適用於 Python 的 x 個 DynamoDB 密用戶端。

下表列出在 Amazon DynamoDB 中支援用戶端加密的語言和版本。

程式設計語言	版本	SDK 主要版本生命週期階段
Java	版本 1. x	終止 Support 階段，自 2022 年 7 月生效
Java	版本 2. x	一般可用性 (GA)
Java	版本 3. x	一般可用性 (GA)
Python	版本 1. x	終止 Support 階段，自 2022 年 7 月生效
Python	版本 2. x	終止 Support 階段，自 2022 年 7 月生效
Python	版本 3. x	一般可用性 (GA)

加密用戶端的運作方式

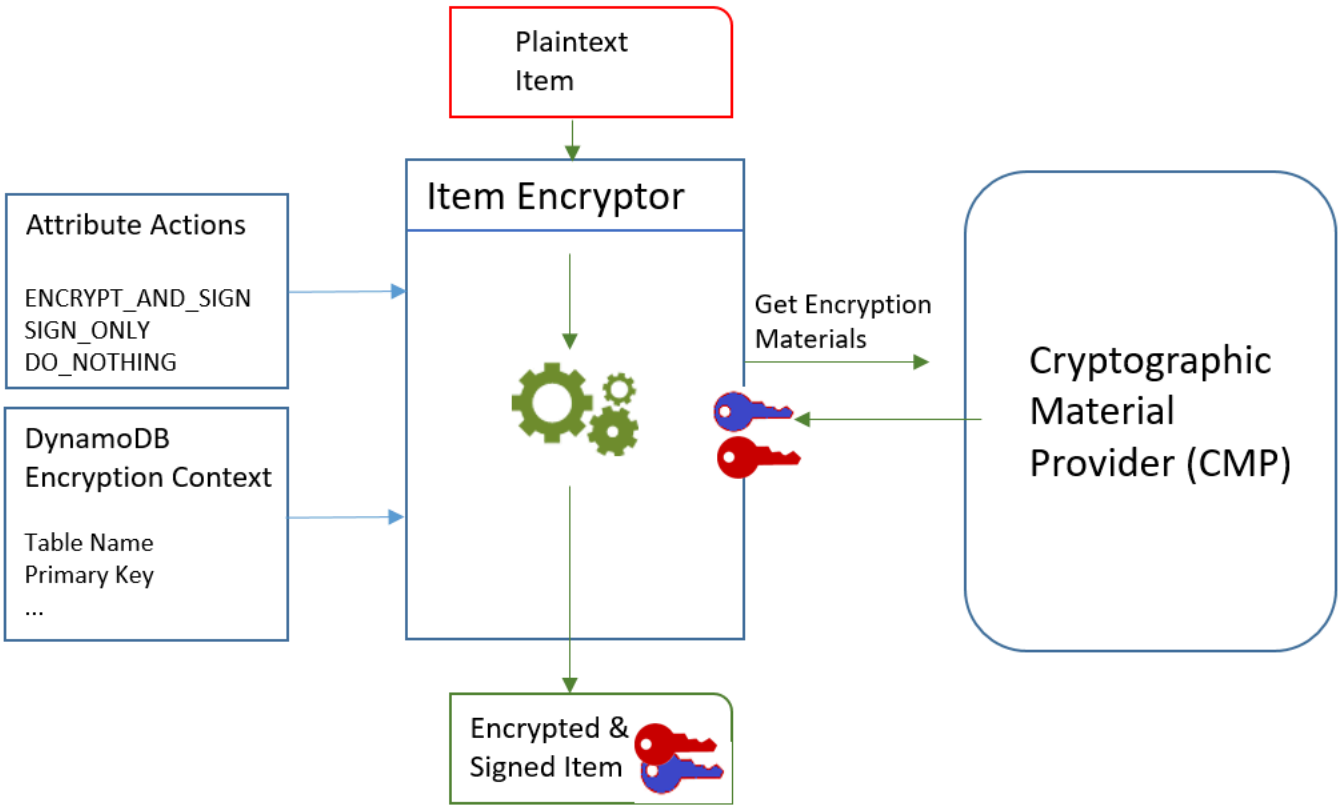
> **Note**
>
> 我們的客戶端加密庫被重命名為AWS數據庫加密 SDK。下列主題提供有關版本 1 的資訊。 X-2. 適用於 Java 和版本 1 的 x 個加密用戶端。 X — 3. 適用於 Python 的 x 個加密用戶端。如需詳細資訊，請參閱適用於 DynamoDB 版本支援的資AWS料庫加密 SDK。

DynamoDB 加密用戶端專為保護您儲存在 DynamoDB 中的資料而設計。程式庫包括您可以擴展或以原狀使用的安全實作。再者，大多數元素是由摘要元素表示，所以您可建立和使用相容的自訂元件。

加密並簽署資料表項目

DynamoDB 加密用戶端的核心是項目加密器，可加密、簽署、驗證和解密表格項目。它會接受資料表項目相關資訊，以及要加密並簽署項目的相關指示。其將從您選取及設定的密碼編譯資料提供者取得加密資料，以及該項資料使用方式的指示。

下圖顯示此程序的高階檢視。



若要加密和簽署表格項目，DynamoDB 加密用戶端需要：

- 有關表格的資訊。它會從您提供的 DynamoDB 加密內容取得有關表格的資訊。有些助手會從 DynamoDB 取得必要的資訊，並為您建立 DynamoDB 加密內容。

> **Note**
>
> DynamoDB 加密用戶端中的 DynamoDB 加密內容與 AWS Key Management Service () AWS KMS 和中的加密內容無關。AWS Encryption SDK

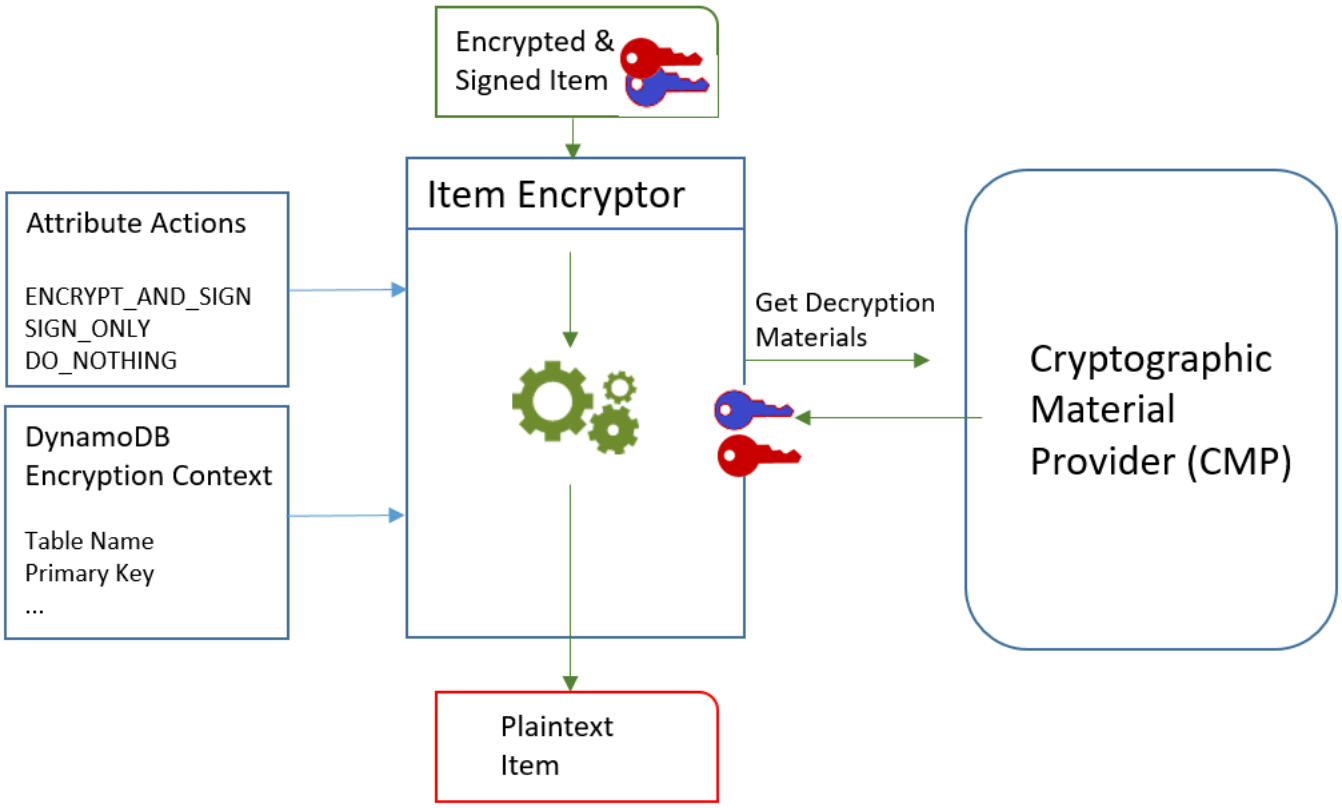
- 哪些屬性要加密並簽署。其將從您提供的屬性動作取得這項資訊。
- 加密資料，包括加密金鑰和簽署金鑰。其將從您選取及設定的密碼編譯資料提供者 (CMP) 取得這些資料。
- 加密並簽署項目的指示。CMP 會將可供使用加密資料 (包括加密和簽署演算法) 的指示新增至實際資料描述。

項目加密程式會使用上述所有元素來加密並簽署項目。項目加密程式也會將兩個屬性新增至此項目：包含加密和簽署指示 (實際資料描述) 的資料描述屬性，以及包含簽章的屬性。您可直接與項目加密程式互動，或使用為您與項目加密程式互動的協助程式功能來實作安全的預設行為。

結果是包含已加密並簽署之資料的 DynamoDB 項目。

驗證並解密資料表項目

如下圖所示，這些元件也會一起運作來驗證並解密項目。



若要驗證和解密項目，DynamoDB 加密用戶端需要相同的元件、具有相同組態的元件，或是專為解密項目而設計的元件，如下所示：

- 來自 DynamoDB 加密內容之表格的相關資訊。
- 要驗證和解密哪些屬性。其將從屬性動作取得這項資訊。
- 解密資料，包括驗證和加密金鑰，來自您選取和設定的密碼編譯資料提供者 (CMP)。

  已加密的項目不包含用來進行加密的 CMP 相關記錄。您必須提供相同的 CMP、具有相同組態的 CMP，或設計用來解密項目的 CMP。
- 如何加密並簽署項目的相關資訊，包括加密和簽署演算法。用戶端會從項目中的資料描述屬性取得這些資訊。

項目加密程式會使用上述所有元素來驗證並解密項目。此外，它不會移除資料描述和簽章屬性。結果為純文字 DynamoDB 項目。

亞馬遜加密用戶端概念

> **Note**
>
> 我們的客戶端加密庫被重命名為AWS數據庫加密 SDK。下列主題提供有關版本 1 的資訊。X-2. 適用於 Java 和版本 1 的 x 個加密用戶端。 X — 3. 適用於 Python 的 x 個加密用戶端。如需詳細資訊，請參閱適用於 DynamoDB 版本支援的資AWS料庫加密 SDK。

本主題說明 Amazon DynamoDB 加密用戶端中使用的概念和術語。

若要瞭解 DynamoDB 加密用戶端的元件如何互動，請參閱。加密用戶端的運作方式

主題

- 密碼編譯資料提供者 (CMP)
- 項目加密程式
- 屬性動作
- 資料描述
- DynamoDB 加密內容
- 提供者存放區

密碼編譯資料提供者 (CMP)

實作 DynamoDB 加密用戶端時，您的首要任務之一是選取加密材料提供者 (CMP) (也稱為加密材料提供者)。您的選擇會決定其餘的大部分實作。

密碼編譯資料提供者 (CMP) 會收集、整合及傳回項目加密程式用來加密並簽署資料表項目的密碼編譯資料。CMP 會決定要使用的加密演算法，以及如何產生及保護加密和簽署金鑰。

CMP 會與項目加密程式互動。項目加密程式會請求 CMP 提供加密和解密資料，而 CMP 會將這些資料傳回給項目加密程式。然後，項目加密程式會使用密碼編譯資料將項目加密並簽署，或驗證並解密。

您會在設定用戶端時指定 CMP。您可以建立相容的自訂 CMP，或使用程式庫的其中一個 CMP。大多數 CMP 都適用於多種程式設計語言。

項目加密程式

項目加密程式是較低層級的元件，可為 DynamoDB 加密用戶端執行密碼編譯作業。其將請求密碼編譯資料提供者 (CMP) 提供密碼編譯資料，然後使用 CMP 傳回的資料將資料表項目加密並簽署，或驗證並解密。

您可以直接與項目加密程式互動，或使用您的程式庫所提供的協助程式。例如，適用於 Java 的 DynamoDB 加密用戶端包含可與使用的`AttributeEncryptor`協助程式類別`DynamoDBMapper`，而不是直接與`DynamoDBEncryptor`項目加密程式互動。Python 程式庫包括 `EncryptedTable`、`EncryptedClient` 和 `EncryptedResource` 協助程式類別，這些類別會替您與項目加密程式互動。

屬性動作

屬性動作會告知項目加密程式要對項目的每個屬性執行什麼動作。

屬性動作值可以是下列其中一項：

- 加密並簽署 — 加密屬性值。在項目簽章中包含屬性 (名稱和值)。
- 僅簽署 — 將屬性包括在項目簽名中。
- 不執行任何動作 — 不加密或簽署屬性。

對於可以儲存敏感資料的任何屬性，請使用加密並簽署。針對主索引鍵屬性 (分割區索引鍵和排序索引鍵)，使用僅簽署。資料描述屬性和簽章屬性不會進行簽署或加密。您不需要指定這些屬性的屬性動作。

請仔細選擇屬性動作。如有疑問，請使用加密並簽署。一旦您使用 DynamoDB 加密用戶端來保護您的資料表項目，就無法變更屬性的動作，而不會冒著簽章驗證錯誤的風險。如需詳細資訊，請參閱 變更您的資料模型。

> **Warning**
>
> 請勿加密主索引鍵的屬性。它們必須保持純文字格式，以便 DynamoDB 可以在不執行完整表格掃描的情況下尋找項目。

如果 DynamoDB 加密內容識別出您的主索引鍵屬性，如果您嘗試加密，用戶端將會擲回錯誤。

您用來為每種程式設計語言指定屬性動作的技巧都不同。而且，該技巧可以專屬於您所使用的協助程式類別。

如需詳細資訊，請參閱程式設計語言的文件。

- Python
- Java

資料描述

已加密資料表項目的資料描述包含資料表項目加密和簽署方式的相關資訊 (例如加密演算法)。密碼編譯資料提供者 (CMP) 會在整合可供加密和簽署的密碼編譯資料時記錄資料描述。稍後，當它需要整合密碼編譯資料來驗證和解密項目時，它會使用資料描述作為其指南。

在 DynamoDB 加密用戶端中，材料說明涉及三個相關元素：

請求的資料描述

有些密碼編譯資料提供者 (CMP) 可讓您指定進階選項，例如加密演算法。為了指出您的選擇，您可以在加密表格項目的請求中，將名稱-值配對新增至 DynamoDB 加密內容的材料描述屬性。此元素又稱為請求的資料描述。請求的資料描述中的有效值由您所選的 CMP 定義。

> **Note**
>
> 由於資料描述可能覆寫安全的預設值，除非您有充分理由要使用請求的資料描述，否則建議您將其省略。

實際資料描述

密碼編譯資料提供者 (CMP) 傳回的資料描述又稱為實際資料描述。它會描述 CMP 在整合密碼編譯資料時所用的實際值。通常包含請求的資料描述及新增和變更 (如果有)。

資料描述屬性

用戶端會在已加密項目的資料描述屬性中儲存實際資料描述。資料描述屬性名稱為 `amzn-ddb-map-desc`，而其值為實際資料描述。用戶端會使用資料描述屬性中的值來驗證並解密項目。

DynamoDB 加密內容

DynamoDB 加密內容會將表格和項目的相關資訊提供給加密材料提供者 (CMP)。在進階實作中，DynamoDB 加密內容可以包含要求的材料描述。

當您加密表格項目時，DynamoDB 加密內容會以密碼編譯方式繫結至加密的屬性值。解密時，如果 DynamoDB 加密內容與用於加密的 DynamoDB 加密內容不完全相符、區分大小寫，則解密作業會失敗。如果您直接與項目加密程式互動，則必須在呼叫加密或解密方法時提供 DynamoDB 加密內容。大多數助手都會為您建立 DynamoDB 加密內容。

> **Note**
>
> DynamoDB 加密用戶端中的 DynamoDB 加密內容與 AWS Key Management Service () AWS KMS 和中的加密內容無關。AWS Encryption SDK

DynamoDB 加密內容可以包含下列欄位。所有欄位和值都是選用的。

- 資料表名稱
- 分割區索引鍵名稱
- 排序索引鍵名稱
- 屬性名稱值組
- 請求的資料描述

提供者存放區

提供者存放區 是一個可傳回密碼編譯資料提供者 (CMP) 的元件。提供者存放區可以建立 CMP，或從另一個來源 (例如另一個提供者存放區) 取得。提供者存放區會在持久性儲存體中儲存其建立的 CMP 版本，而存放的每個 CMP 都是依照請求者的資料名稱與版本號碼進行識別。

DynamoDB 加密用戶端中的最新提供者會從提供者存放區取得其 CMP，但您可以使用提供者存放區為任何元件提供 CMP。每個最近提供者都與一個提供者存放區相關聯，但提供者存放區可以將 CMP 提供給多部主機上的許多請求者。

提供者存放區會隨需建立新的 CMP 版本，並傳回新的和現有版本。它也會傳回特定資料名稱的最新版本號碼。這使請求者能夠得知提供者存放區何時有可請求的新版 CMP。

DynamoDB 加密用戶端包含一個提供者存放區 MetaStore，可使用儲存在 DynamoDB 中的金鑰建立包裝的 CMP，並使用內部 DynamoDB 加密用戶端進行加密。

進一步了解：

- 提供者存放區：Java、Python
- MetaStore: 爪哇, 蟒蛇

密碼材料供應商

> **Note**
>
> 我們的客戶端加密庫被重命名為AWS數據庫加密 SDK。下列主題提供有關版本 1 的資訊。X-2. 適用於 Java 和版本 1 的 x 個加密用戶端。 X — 3. 適用於 Python 的 x 個加密用戶端。如需詳細資訊，請參閱適用於 DynamoDB 版本支援的資AWS料庫加密 SDK。

使用 DynamoDB 加密用戶端時，最重要的決定之一就是選擇加密材料提供者 (CMP)。CMP 會整合密碼編譯資料，並將其傳回至項目加密程式。它也會決定加密和簽署金鑰的產生方式、要為每個項目產生新的金鑰資料還是重複使用這些資料，以及所使用的加密和簽署演算法等。

您可以從 DynamoDB 加密用戶端程式庫中提供的實作中選擇 CMP，或建立相容的自訂 CMP。您的 CMP 選擇也可能取決於您所使用的程式設計語言。

本主題說明最常見的 CMP，並提供相關建議以協助您選擇最適用於應用程式的 CMP。

直接 KMS 資料提供者

直接 KMS 材料提供商可以在永遠不會離開 AWS Key Management Service（AWS KMS）未加密AWS KMS key的情況下保護您的表格項目。您的應用程式不會產生或管理任何密碼編譯資料。此提供者會使用 AWS KMS key 為每個項目產生唯一的加密和簽署金鑰，因此在每次為項目加密或解密時，它都會呼叫 AWS KMS。

如果您使用 AWS KMS，且對每項交易發出一次 AWS KMS 呼叫對您的應用程式是可行的，那麼這個提供者將是理想的選擇。

如需詳細資訊，請參閱 直接 KMS 資料提供者。

包裝資料提供者 (包裝 CMP)

包裝材料提供者 (包裝的 CMP) 可讓您在 DynamoDB 加密用戶端之外產生和管理包裝和簽署金鑰。

包裝 CMP 會為每個項目產生唯一的加密金鑰。接著，它會使用您所提供的包裝 (或取消包裝) 和簽署金鑰。如此，您可決定包裝和簽署金鑰的產生方式，以及要讓每個項目各有唯一的金鑰還是重複使用這些金鑰。對於不使用AWS KMS且可以安全管理加密資料的應用程式，包裝的 CMP 是 Direct KMS 提供者的安全替代方案。

如需詳細資訊，請參閱 包裝資料提供者。

最近提供者

最近提供者是一個密碼編譯資料提供者 (CMP)，旨在與提供者存放區搭配使用。它會從提供者存放區取得 CMP，並取得它從 CMP 傳回的密碼編譯資料。最近提供者通常會使用各個 CMP 因應多次密碼編譯資料請求，但您也可以使用提供者存放區的功能來控制資料重複使用的程度、決定輪換 CMP 的頻率，甚至在不變更最近提供者的情況下變更所使用的 CMP 類型。

您可以將最近提供者與任何相容的提供者存放區搭配使用。DynamoDB 加密用戶端包含一個 MetaStore，這是傳回包裝 CMP 的提供者存放區。

對於需要盡可能避免呼叫其密碼編譯來源的應用程式，以及可重複使用部分密碼編譯資料而不會違反安全性需求的應用程式，最近提供者將是理想的選擇。例如，它允許您AWS KMS key在 in AWS Key Management Service(AWS KMS) 下保護您的加密材料，而無需AWS KMS每次加密或解密項目時都調用。

如需詳細資訊，請參閱 最近提供者。

靜態資料提供者

靜態材料供應商專為測試、proof-of-concept示範和舊版相容性而設計。它不會為每個項目產生任何唯一的密碼編譯資料。它會傳回您所提供的相同加密和簽署金鑰，並直接使用這些金鑰來加密、解密和簽署您的資料表項目。

> **Note**
>
> Java 程式庫中的非對稱靜態提供者不是靜態提供者。其僅提供包裝 CMP 的替代建構函數。此提供者可在生產環境中安全地使用，但只要情況允許，您即應直接使用包裝 CMP。

主題

- 直接 KMS 資料提供者
- 包裝資料提供者
- 最近提供者
- 靜態資料提供者

直接 KMS 資料提供者

> **Note**
>
> 我們的客戶端加密庫被重命名為AWS數據庫加密 SDK。下列主題提供有關版本 1 的資訊。X-2. 適用於 Java 和版本 1 的 x 個加密用戶端。 X — 3. 適用於 Python 的 x 個加密用戶端。如需詳細資訊，請參閱適用於 DynamoDB 版本支援的資AWS料庫加密 SDK。

直接 KMS 材料提供商（直接 KMS 提供商）保護您的表項下永遠不AWS KMS key會離開 AWS Key Management Service（AWS KMS）未加密。此密碼編譯資料提供者會為每個資料表項目傳回唯一的加密金鑰和簽署金鑰。為此，它會在您每次進行項目的加密或解密時呼叫 AWS KMS。

如果您以高頻率和大規模處理 DynamoDB 項目，則可能會超出AWS KMSrequests-per-second限制，導致處理延遲。如果您需要超過限制，請在AWS Support中心建立案例。您也可以考慮使用有限金鑰重複使用的加密材料提供者，例如「最新的提供者」。

若要使用直接 KMS 提供者AWS 帳戶，呼叫者必須具有至少一個GenerateDataKey和權限AWS KMS key，才能呼叫AWS KMS key. AWS KMS key必須是對稱加密金鑰；DynamoDB 加密用戶端不支援非對稱加密。如果您使用 DynamoDB 全域表，您可能需要指定AWS KMS多區域金鑰。如需詳細資訊，請參閱 使用方式。

> **Note**
>
> 當您使用 Direct KMS 提供者時，主索引鍵屬性的名稱和值會以純文字顯示在AWS KMS加密內容和相關AWS KMS作業的AWS CloudTrail記錄中。但是，DynamoDB 加密用戶端永遠不會公開任何加密屬性值的純文字。

直接 KMS 提供者是 DynamoDB 加密用戶端支援的數個加密材料提供者 (CMP) 之一。如需其他 CMP 的相關資訊，請參閱密碼材料供應商。

如需範例程式碼，請參閱：

- Java：AwsKmsEncryptedItem
- 蟒蛇:aws-kms-encrypted-table, aws-kms-encrypted-item

主題

- 使用方式
- 運作方式

使用方式

若要建立直接 KMS 提供者，請使用金鑰 ID 參數在您的帳戶中指定對稱加密 KMS 金鑰。索引鍵識別碼參數的值可以是的索引鍵識別碼、金鑰 ARN、別名或別名 ARN。AWS KMS key如需金鑰識別碼的詳細資訊，請參閱AWS Key Management Service開發人員指南中的金鑰識別碼。

直接 KMS 提供者需要對稱加密 KMS 金鑰。無法使用非對稱 KMS 金鑰。不過，您可以使用多區域 KMS 金鑰、包含匯入金鑰材料的 KMS 金鑰，或是自訂金鑰存放區中的 KMS 金鑰。您必須擁有 KMS 金鑰的公里:GenerateDataKey和 KMS: 解密權限。因此，您必須使用客戶管理的金鑰，而不是受AWS管或AWS擁有的 KMS 金鑰。

適用於 Python 的 DynamoDB 加密用戶端會在金鑰 ID 參數值中決定要AWS KMS從區域呼叫的區域 (如果該區域包含一個)。否則，它會使用用AWS KMS戶端中的「區域」(如果您指定區域) 或您在中設定的「區域」AWS SDK for Python (Boto3)。如需有關 Python 中區域選取的資訊，請參閱在AWS開發套件中的設定 (Boto3) API 參考。

如果您指定的用戶端包含區域，Java 的 DynamoDB 加密用戶端會決定AWS KMS從AWS KMS用戶端中的區域呼叫的區域。否則，它會使用您在中設定的區域AWS SDK for Java。如需有關「區域」選取的資訊AWS SDK for Java，請參閱AWS SDK for Java開發人員指南中的AWS 區域選取項目。

Java

```
// Replace the example key ARN and Region with valid values for your application
final String keyArn = 'arn:aws:kms:us-
west-2:111122223333:key/1234abcd-12ab-34cd-56ef-1234567890ab'
final String region = 'us-west-2'

final AWSKMS kms = AWSKMSClientBuilder.standard().withRegion(region).build();
final DirectKmsMaterialProvider cmp = new DirectKmsMaterialProvider(kms, keyArn);
```

Python

下列範例使用 ARN 索引鍵來指定AWS KMS key。如果您的金鑰識別碼未包含AWS 區域，DynamoDB 加密用戶端會從設定的 Botocore 工作階段 (如果有的話) 取得區域，或從 Boto 預設值取得該區域。

```
# Replace the example key ID with a valid value
```

```
kms_key = 'arn:aws:kms:us-
west-2:111122223333:key/1234abcd-12ab-34cd-56ef-1234567890ab'
kms_cmp = AwsKmsCryptographicMaterialsProvider(key_id=kms_key)
```

如果您使用 Amazon DynamoDB 全域表，建議您使用AWS KMS多區域金鑰加密資料。多區域金鑰是不同AWS KMS keys的，可AWS 區域以互換使用，因為它們具有相同的金鑰 ID 和金鑰材料。如需詳細資訊，請參閱AWS Key Management Service開發人員指南中的使用多區域金鑰。

> **Note**
>
> 如果您使用的是全域表 2017.11.29 版，則必須設定屬性動作，使保留的複寫欄位不會加密或簽署。如需詳細資訊，請參閱 舊版全域資料表的問題。

若要將多區域金鑰與 DynamoDB 加密用戶端搭配使用，請建立多區域金鑰，然後將其複寫到應用程式執行所在的區域。然後將直接 KMS 提供者設定為在 DynamoDB 加密用戶端呼叫的區域中使用多區域金鑰。AWS KMS

下列範例將 DynamoDB 加密用戶端設定為在美國東部 (維吉尼亞北部) (us-east-1) 區域中加密資料，並使用多區域金鑰在美國西部 (奧勒岡) (US-西部 -2) 區域中將其解密。

Java

在此範例中，DynamoDB 加密用戶端會取得要AWS KMS從用戶端中的區域呼叫的AWS KMS區域。此`keyArn`值可識別相同區域中的多區域金鑰。

```
// Encrypt in us-east-1

// Replace the example key ARN and Region with valid values for your application
final String usEastKey = 'arn:aws:kms:us-east-1:111122223333:key/
mrk-1234abcd12ab34cd56ef1234567890ab'
final String region = 'us-east-1'

final AWSKMS kms = AWSKMSClientBuilder.standard().withRegion(region).build();
final DirectKmsMaterialProvider cmp = new DirectKmsMaterialProvider(kms, usEastKey);
```

```
// Decrypt in us-west-2

// Replace the example key ARN and Region with valid values for your application
```

```
final String usWestKey = 'arn:aws:kms:us-west-2:111122223333:key/
mrk-1234abcd12ab34cd56ef1234567890ab'
final String region = 'us-west-2'

final AWSKMS kms = AWSKMSClientBuilder.standard().withRegion(region).build();
final DirectKmsMaterialProvider cmp = new DirectKmsMaterialProvider(kms, usWestKey);
```

Python

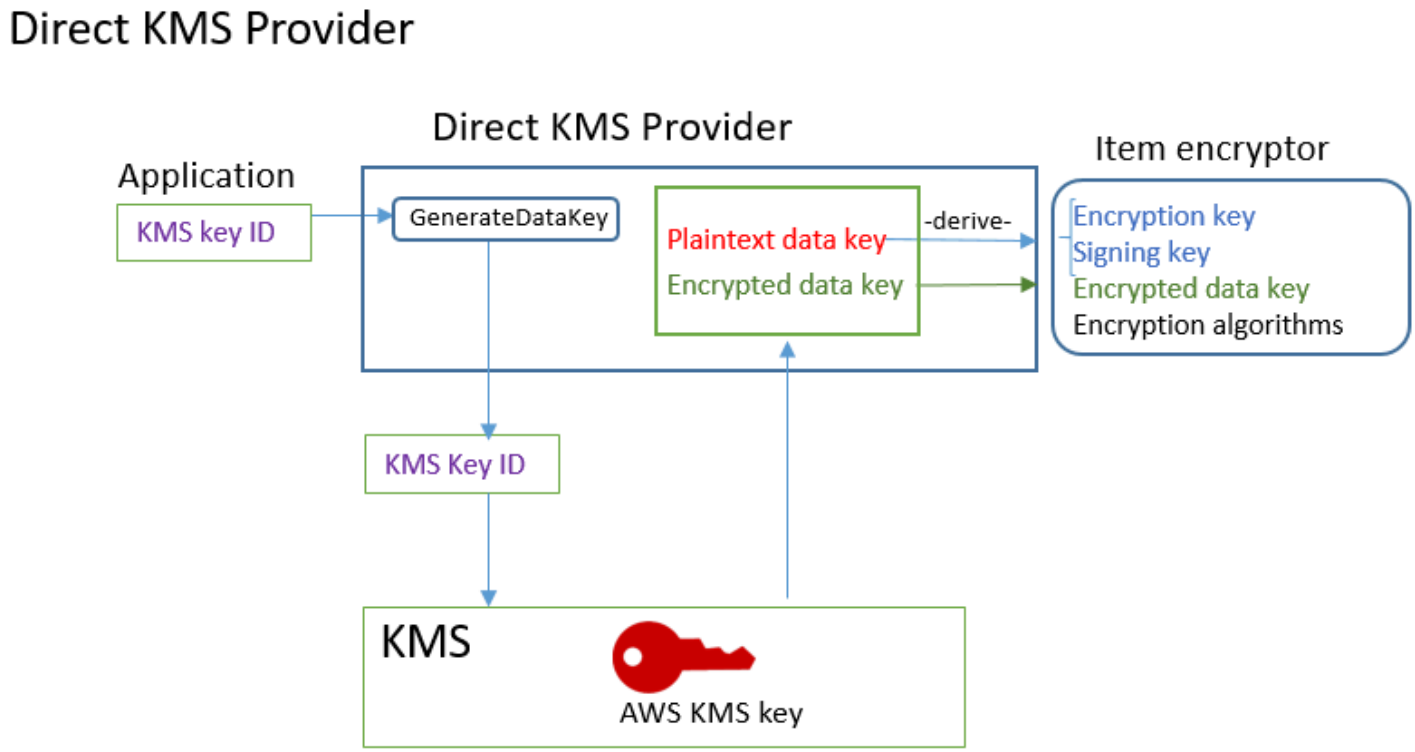
在此範例中，DynamoDB 加密用戶端會取得要AWS KMS從金鑰 ARN 中的區域呼叫的區域。

```
# Encrypt in us-east-1

# Replace the example key ID with a valid value
us_east_key = 'arn:aws:kms:us-east-1:111122223333:key/
mrk-1234abcd12ab34cd56ef1234567890ab'
kms_cmp = AwsKmsCryptographicMaterialsProvider(key_id=us_east_key)
```

```
# Decrypt in us-west-2

# Replace the example key ID with a valid value
us_west_key = 'arn:aws:kms:us-west-2:111122223333:key/
mrk-1234abcd12ab34cd56ef1234567890ab'
kms_cmp = AwsKmsCryptographicMaterialsProvider(key_id=us_west_key)
```

運作方式

Direct KMS 提供者會傳回受您指定之保護AWS KMS key的加密和簽署金鑰，如下圖所示。



- 若要產生加密資料，Direct KMS 提供者會AWS KMS要求使用您指定的項目，為每個項目產生唯一AWS KMS key的資料金鑰。它會從資料金鑰的純文字複本衍生項目的加密和簽署金鑰，然後傳回加密和簽署金鑰，以及在項目資料描述屬性中存放的加密資料金鑰。

  項目加密程式會在使用加密和簽署金鑰後，盡快將其從記憶體中移除。加密的項目中只會儲存用來衍生這些金鑰的資料金鑰加密複本。

- 為了產生解密資料，直接 KMS 提供者會要求 AWS KMS 將加密的資料金鑰解密。隨後，它會從純文字資料金鑰衍生驗證和簽署金鑰，並將它們傳回至項目加密程式。

  項目加密程式會驗證該項目，如果驗證成功，則會將加密的值解密。接著，它會盡快從記憶體中移除這些金鑰。

取得加密資料

本節將詳細說明直接 KMS 提供者在接收到來自項目加密程式的加密資料請求時的輸入、輸出和處理情形。

輸入 (從應用程式)

- 的金鑰識別碼AWS KMS key。

輸入 (從項目加密程式)

- 加密內容

輸出 (到項目加密程式)

- 加密金鑰 (純文字)
- 簽署金鑰
- 在實際資料描述中：這些值會儲存在用戶端新增至項目的資料描述屬性中。
  - amzn-ddb-env-key：由 Base64 編碼的資料金鑰加密 AWS KMS key
  - amzn-ddb-env-alg：加密演算法，預設為 A ES/256
  - amzn-ddb-sig-alg: 簽署演算法，預設情況下，
  - amzn-ddb-wrap-alg: 公里

處理

1. 直接 KMS 提供者會傳送AWS KMS要求，以使用指AWS KMS key定的產生項目的唯一資料金鑰。此操作會傳回以 AWS KMS key 加密的純文字金鑰和複本。這項資料又稱為初始金鑰資料。

   此要求包含 AWS KMS 加密內容中的下列純文字值。這些非機密值會以密碼編譯的方式繫結至加密的物件，而在解密時需要相同的加密細節。您可以使用這些值來識別AWS CloudTrail記錄AWS KMS中的呼叫。

   - amzn-ddb-env-alg— 加密演算法，預設為 AES/256
   - amzn-ddb-sig-alg— 簽名算法，默認情況下
   - (選擇性) aws-kms-table — ####
   - (選擇性) ######## — ####### (二進位值是 Base64 編碼)
   - (選擇性) ####### — ###### (二進位值是 Base64 編碼)

   直接 KMS 提供者會從項目的 DynamoDB AWS KMS 加密內容取得加密內容的值。如果 DynamoDB 加密內容未包含值 (例如資料表名稱)，則加密內容中會省略該名稱與值配對。AWS KMS
2. 直接 KMS 提供者會從資料金鑰衍生對稱加密金鑰和簽署金鑰。根據預設，其將使用安全雜湊演算法 (SHA) 256 和 RFC5869 HMAC 式金鑰衍生函數，衍生 256 位元 AES 對稱加密金鑰和 256 位元 HMAC-SHA-256 簽署金鑰。
3. 直接 KMS 提供者會將輸出傳回至項目加密程式。

4. 項目加密程式會採用實際資料描述中指定的演算法，使用加密金鑰為指定的屬性加密，並使用簽署金鑰加以簽署。它會盡快從記憶體中移除這些純文字金鑰。

取得解密資料

本節將詳細說明直接 KMS 提供者在接收到來自項目加密程式的解密資料請求時的輸入、輸出和處理情形。

輸入 (從應用程式)

- 的金鑰識別碼AWS KMS key。

  金鑰識別碼的值可以是的金鑰識別碼、金鑰 ARN、別名或別名 ARN。AWS KMS key金鑰 ID 中未包含的任何值 (例如「地區」) 都必須在AWS具名的設定檔中提供。關鍵 ARN 提供了所有AWS KMS需要的值。

輸入 (從項目加密程式)

- DynamoDB 加密內容的副本，其中包含材料描述屬性的內容。

輸出 (到項目加密程式)

- 加密金鑰 (純文字)
- 簽署金鑰

處理

1. 直接 KMS 提供者會從加密項目中的材料描述屬性取得加密的資料金鑰。
2. 它會要求 AWS KMS 使用指定的 AWS KMS key 將加密的資料金鑰解密。此操作會傳回純文字金鑰。

   此要求必須使用先前用來產生和加密資料金鑰的相同 AWS KMS 加密內容。

   - aws-kms-table— *####*
   - *########* — *#######* (二進位值是 Base64 編碼)
   - (選擇性) *#######* — *######* (二進位值是 Base64 編碼)
   - amzn-ddb-env-alg— 加密演算法，預設為 AES/256
   - amzn-ddb-sig-alg— 簽名算法，默認情況下

3. 直接 KMS 提供者會使用安全雜湊演算法 (SHA) 256 和 RFC5869 HMAC 式金鑰衍生函數，從資料金鑰衍生 256 位元 AES 對稱加密金鑰和 256 位元 HMAC-SHA-256 簽署金鑰。
4. 直接 KMS 提供者會將輸出傳回至項目加密程式。
5. 項目加密程式會使用簽署金鑰來驗證項目。如果成功，則會使用對稱加密金鑰將加密的屬性值解密。這些操作會使用實際資料描述中指定的加密和簽署演算法。項目加密程式會盡快從記憶體中移除這些純文字金鑰。

包裝資料提供者

> **Note**
>
> 我們的客戶端加密庫被重命名為AWS數據庫加密 SDK。下列主題提供有關版本 1 的資訊。X-2. Java 和版本 1 的動態驗證加密用戶端的 x。 X — 3. 適用於 Python 的 x 個加密用戶端。如需詳細資訊，請參閱適用於 DynamoDB 版本支援的資AWS料庫加密 SDK。

包裝材料提供者 (包裝的 CMP) 可讓您透過 DynamoDB 加密用戶端使用來自任何來源的包裝和簽署金鑰。包裝的 CMP 不依賴於任何AWS服務。不過，您必須在用戶端以外產生及管理包裝和簽署金鑰，包括提供正確的金鑰來驗證和解密項目。

包裝 CMP 會為每個項目產生唯一的項目加密金鑰。其將使用您所提供的包裝金鑰來包裝項目加密金鑰，並將包裝的項目加密金鑰儲存至項目的資料描述屬性。因為您提供包裝和簽署金鑰，所以由您決定包裝和簽署金鑰的產生方式，以及要讓每個項目各有唯一的金鑰還是重複使用。

對於可以管理密碼編譯資料的應用程式而言，包裝 CMP 是安全的實作與理想的選擇。

包裝的 CMP 是 DynamoDB 加密用戶端支援的數個加密材料提供者 (CMP) 之一。如需其他 CMP 的相關資訊，請參閱密碼材料供應商。

如需範例程式碼，請參閱：

- Java：AsymmetricEncryptedItem
- 蟒蛇:wrapped-rsa-encrypted-table, wrapped-symmetric-encrypted-table

主題

- 使用方式
- 運作方式

使用方式

若要建立包裝 CMP，請指定包裝金鑰 (加密時需要)、取消包裝金鑰 (解密時需要) 以及簽署金鑰。您必須在加密和解密項目時提供金鑰。

包裝、取消包裝和簽署金鑰可以是對稱金鑰或非對稱金鑰對。

Java

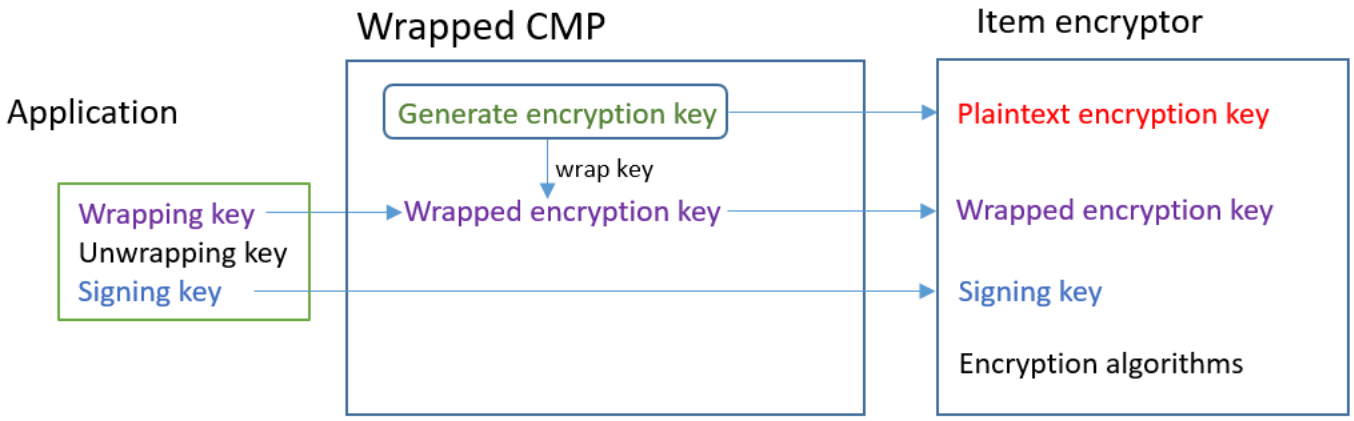
```
// This example uses asymmetric wrapping and signing key pairs
final KeyPair wrappingKeys = ...
final KeyPair signingKeys = ...

final WrappedMaterialsProvider cmp =
    new WrappedMaterialsProvider(wrappingKeys.getPublic(),
                                 wrappingKeys.getPrivate(),
                                 signingKeys);
```

Python

```
# This example uses symmetric wrapping and signing keys
wrapping_key = ...
signing_key  = ...

wrapped_cmp = WrappedCryptographicMaterialsProvider(
    wrapping_key=wrapping_key,
    unwrapping_key=wrapping_key,
    signing_key=signing_key
)
```

運作方式

包裝 CMP 會為每個項目產生新的項目加密金鑰。如下圖所示，它會使用您所提供的包裝、取消包裝和簽署金鑰。



取得加密資料

本節將詳細說明包裝資料提供者 (包裝 CMP) 在接收到加密資料請求時的輸入、輸出和處理情形。

輸入 (從應用程式)

- 包裝金鑰：進階加密標準 (AES) 對稱金鑰，或 RSA 公有金鑰。如有任何已加密的屬性值，則為必要。否則為選用並予以忽略。
- 取消包裝金鑰：選用並予以忽略。
- 簽署金鑰

輸入 (從項目加密程式)

- 加密內容

輸出 (到項目加密程式)：

- 純文字項目加密金鑰
- 簽署金鑰 (不變)
- 實際資料描述：這些值會儲存在用戶端新增至項目的資料描述屬性中。
  - `amzn-ddb-env-key`：Base64 編碼的包裝項目加密金鑰
  - `amzn-ddb-env-alg`：用來加密項目的加密演算法。預設為 AES-256-CBC。
  - `amzn-ddb-wrap-alg`：包裝 CMP 用來包裝項目加密金鑰的包裝演算法。如果包裝金鑰是 AES 金鑰，則會使用未填補的 `AES-Keywrap` (如 RFC 3394 定義) 來包裝此金鑰。如果包裝金鑰是 RSA 金鑰，則會使用 RSA OAEP (MGF1 填補) 來加密此金鑰。

處理

當您加密項目時，您會傳入包裝金鑰和簽署金鑰。取消包裝金鑰為選用並予以忽略。

1. 包裝 CMP 會為資料表項目產生唯一的對稱項目加密金鑰。
2. 它會使用您指定的包裝金鑰來包裝項目加密金鑰。接著，它會盡快從記憶體中移除此金鑰。
3. 其將傳回純文字項目加密金鑰、您所提供的簽署金鑰，以及包含包裝項目加密金鑰和加密與包裝演算法的實際資料描述。
4. 項目加密程式會使用純文字加密金鑰來加密項目。它會使用您所提供的簽署金鑰來簽署金鑰。接著，它會盡快從記憶體中移除這些純文字金鑰。它會將實際資料描述中的欄位 (包括包裝加密金鑰 (amzn-ddb-env-key)) 複製到項目的資料描述屬性。

取得解密資料

本節將詳細說明包裝資料提供者 (包裝 CMP) 在接收到解密資料請求時的輸入、輸出和處理情形。

輸入 (從應用程式)

- 包裝金鑰：選用並予以忽略。
- 取消包裝金鑰：相同的進階加密標準 (AES) 對稱金鑰，或與加密使用的 RSA 公有金鑰對應的 RSA 私有金鑰。如有任何已加密的屬性值，則為必要。否則為選用並予以忽略。
- 簽署金鑰

輸入 (從項目加密程式)

- DynamoDB 加密內容的副本，其中包含材料描述屬性的內容。

輸出 (到項目加密程式)

- 純文字項目加密金鑰
- 簽署金鑰 (不變)

處理

當您解密項目時，您會傳入取消包裝金鑰和簽署金鑰。包裝金鑰為選用並予以忽略。

1. 包裝 CMP 會從項目的資料描述屬性取得包裝項目加密金鑰。

2. 它會使用取消包裝金鑰和演算法來取消包裝項目加密金鑰。
3. 它會將純文字項目加密金鑰、簽署金鑰以及加密和簽署演算法傳回給項目加密程式。
4. 項目加密程式會使用簽署金鑰來驗證項目。如果成功，則會使用項目加密金鑰來將項目解密。接著，它會盡快從記憶體中移除這些純文字金鑰。

最近提供者

> **Note**
>
> 我們的客戶端加密庫被重命名為AWS數據庫加密 SDK。下列主題提供有關版本 1 的資訊。X-2. Java 和版本 1 的動態驗證加密用戶端的 x。 X — 3. 適用於 Python 的 x 個加密用戶端。如需詳細資訊，請參閱適用於 DynamoDB 版本支援的資AWS料庫加密 SDK。

最近提供者是一個密碼編譯資料提供者 (CMP)，旨在與提供者存放區搭配使用。它會從提供者存放區取得 CMP，並取得它從 CMP 傳回的密碼編譯資料。其通常會使用各個 CMP 因應多次密碼編譯資料請求。但您也可以使用提供者存放區的功能來控制資料的重複使用程度、決定 CMP 的輪換頻率，甚至在不變更「最近提供者」的情況下變更所使用的 CMP 類型。

> **Note**
>
> 與「最新提供者」`MostRecentProvider` 符號相關聯的程式碼可能會在程序的生命週期內將密碼編譯材料儲存在記憶體中。它可能允許來電者使用他們不再授權使用的金鑰。該`MostRecentProvider`符號已在舊版受支援的 DynamoDB 加密用戶端中取代，並從 2.0.0 版移除。它由符號所取`CachingMostRecentProvider`代。如需詳細資訊，請參閱 最新提供者的更新。

對於需要盡可能避免呼叫提供者存放區與其密碼編譯來源的應用程式，以及可重複使用部分密碼編譯資料而不會違反安全性需求的應用程式，最近提供者將是理想的選擇。例如，它允許您AWS KMS key在 in AWS Key Management Service（AWS KMS）下保護您的加密材料，而無需AWS KMS每次加密或解密項目時都調用。

您選擇的提供者存放區可決定最近提供者使用的 CMP 類型，以及其取得新 CMP 的頻率。您可以使用任何相容提供者存放區搭配最近提供者，包括您所設計的自訂提供者存放區。

DynamoDB 加密用戶端包含可建立並傳回MetaStore包裝材料提供者 (包裝 CMP) 的一個。這會MetaStore將產生的多個版本的已包裝 CMP 儲存在內部 DynamoDB 表格中，並透過 DynamoDB 加密用戶端的內部執行個體透過用戶端加密來保護它們。

您可以設定為使用任何類型的內部 CMP 來保護資料表中的資料，包括產生受您保護的加密資料的Direct KMS 提供者AWS KMS key、使用您提供的包裝和簽署金鑰的 Wraped CMP，或是您設計的相容自訂 CMP。MetaStore

如需範例程式碼，請參閱：

- Java：MostRecentEncryptedItem
- Python：most_recent_provider_encrypted_table

主題

- 使用方式
- 運作方式
- 最新提供者的更新

使用方式

若要建立最近提供者，您必須建立及設定提供者存放區，然後建立可使用該提供者存放區的最近提供者。

下列範例說明如何建立使用其內部 DynamoDB 表格中的版本的最新提供者，MetaStore並使用直接KMS 提供者的加密資料來保護版本。這些範例使用CachingMostRecentProvider符號。

每個「最近的提供者」都有一個可識別MetaStore表格中 CMP 的名稱、time-to-live(TTL) 設定，以及決定快取可保留多少個項目的快取大小設定。這些範例會將快取大小設定為 1000 個項目，TTL 設定為 60 秒。

Java

```
// Set the name for MetaStore's internal table
final String keyTableName = 'metaStoreTable'

// Set the Region and AWS KMS key
final String region = 'us-west-2'
final String keyArn = 'arn:aws:kms:us-
west-2:111122223333:key/1234abcd-12ab-34cd-56ef-1234567890ab'
```

```
// Set the TTL and cache size
final long ttlInMillis = 60000;
final long cacheSize = 1000;

// Name that identifies the MetaStore's CMPs in the provider store
final String materialName = 'testMRP'

// Create an internal DynamoDB client for the MetaStore
final AmazonDynamoDB ddb =
 AmazonDynamoDBClientBuilder.standard().withRegion(region).build();

// Create an internal Direct KMS Provider for the MetaStore
final AWSKMS kms = AWSKMSClientBuilder.standard().withRegion(region).build();
final DirectKmsMaterialProvider kmsProv = new DirectKmsMaterialProvider(kms,
 keyArn);

// Create an item encryptor for the MetaStore,
// including the Direct KMS Provider
final DynamoDBEncryptor keyEncryptor = DynamoDBEncryptor.getInstance(kmsProv);

// Create the MetaStore
final MetaStore metaStore = new MetaStore(ddb, keyTableName, keyEncryptor);

//Create the Most Recent Provider
final CachingMostRecentProvider cmp = new CachingMostRecentProvider(metaStore,
 materialName, ttlInMillis, cacheSize);
```

Python

```
# Designate an AWS KMS key
kms_key_id = 'arn:aws:kms:us-
west-2:111122223333:key/1234abcd-12ab-34cd-56ef-1234567890ab'

# Set the name for MetaStore's internal table
meta_table_name = 'metaStoreTable'

# Name that identifies the MetaStore's CMPs in the provider store
material_name = 'testMRP'

# Create an internal DynamoDB table resource for the MetaStore
meta_table = boto3.resource('dynamodb').Table(meta_table_name)
```

```
# Create an internal Direct KMS Provider for the MetaStore
kms_cmp = AwsKmsCryptographicMaterialsProvider(key_id=kms_key_id)

# Create the MetaStore with the Direct KMS Provider
meta_store = MetaStore(
    table=meta_table,
    materials_provider=kms_cmp
)

# Create a Most Recent Provider using the MetaStore
#    Sets the TTL (in seconds) and cache size (# entries)
most_recent_cmp = MostRecentProvider(
    provider_store=meta_store,
    material_name=material_name,
    version_ttl=60.0,
    cache_size=1000
)
```

運作方式

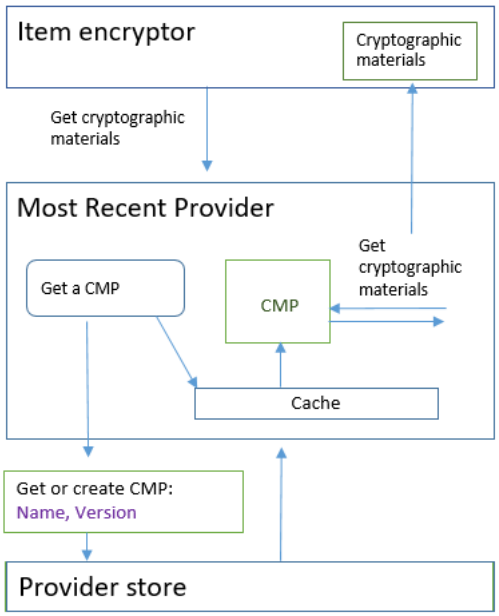
最近提供者會從提供者存放區取得 CMP。然後，它會使用 CMP 來產生密碼編譯資料並將其傳回給項目加密程式。

關於最近提供者

最新的供應商會從供應商商店取得加密材料提供者 (CMP)。然後，它會使用 CMP 來產生可傳回的密碼編譯資料。每個最近提供者都會與一個提供者存放區相關聯，但提供者存放區可以將 CMP 提供給多部主機上的多個提供者。

最近提供者可與任何提供者存放區中的任何相容 CMP 搭配使用。它要求從 CMP 加密或解密材料，並將輸出返回到項目加密器。並不會執行任何密碼編譯操作。

若要向提供者存取區請求 CMP，最近提供者可提供其資料名稱以及想要使用的現有 CMP 版本。針對加密資料，最近提供者一律會請求最大 (「最近」) 版本。針對解密資料，其將請求用來建立加密資料的 CMP 版本 (如下圖所示)。



最近提供者會將提供者存放區傳回的 CMP 版本儲存於記憶體中的本機最久未使用 (LRU) 快取。此快取可讓最近提供者取得它所需的 CMP，而不需針對每個項目呼叫提供者存放區。您可以隨需清除此快取。

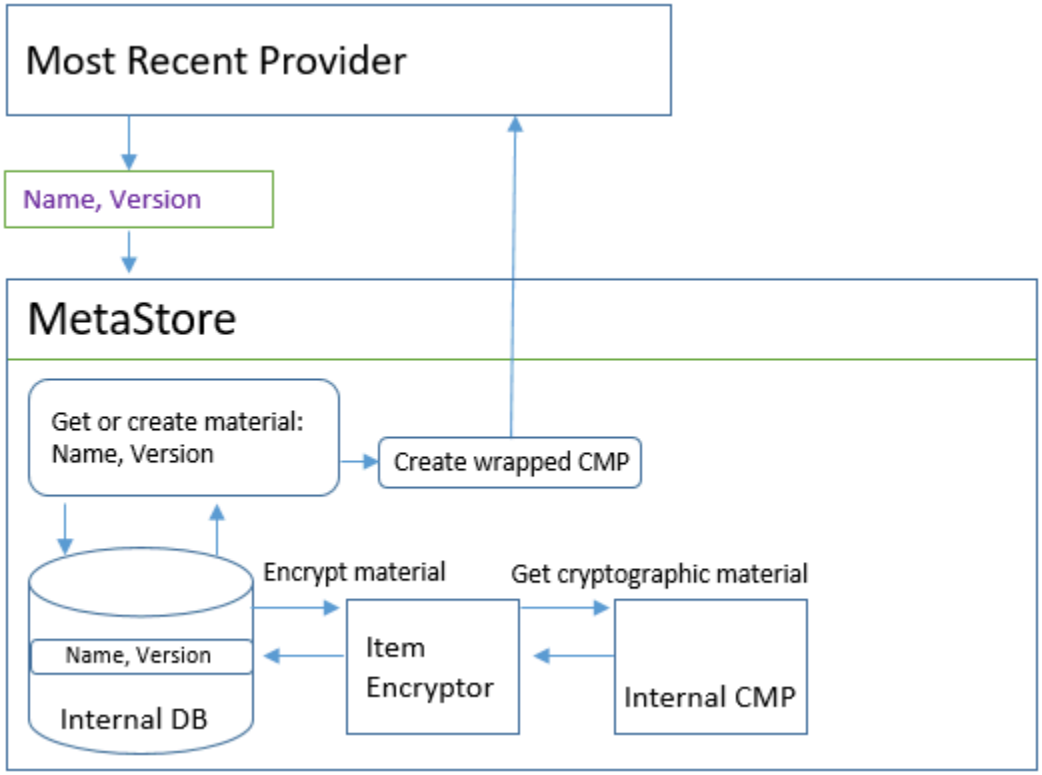
「最新的提供者」使用可設定的time-to-live值，您可以根據應用程式的特性進行調整。

關於 MetaStore

您可以使用最近提供者搭配任何提供者存放區，包括相容的自訂提供者存放區。DynamoDB 加密用戶端包含MetaStore可設定和自訂的安全實作。

A MetaStore是一個提供者存放區，可建立並傳回使用包裝 CMP 所需的包裝金鑰、解包金鑰和簽章金鑰設定的包裝 CMP。A MetaStore 是最新提供者的安全選項，因為包裝的 CMP 一律會為每個項目產生唯一的項目加密金鑰。只有可保護項目加密金鑰的包裝金鑰以及簽署金鑰能重複使用。

下圖顯示的元件，以MetaStore及它如何與最新的提供者互動。



會MetaStore產生已包裝的 CMP，然後將它們 (以加密格式) 儲存在內部 DynamoDB 表格中。分割索引鍵是「最新提供者」資料的名稱；排序索引鍵的版本號碼。表格中的資料受到內部 DynamoDB 加密用戶端的保護，包括項目加密器和內部加密材料提供者 (CMP)。

您可以在您的中使用任何類型的內部 CMPMetaStore，包括直接 KMS 提供商，包含您提供的加密材料的包裝 CMP，或兼容的自定義 CMP。如果您的內部 CMP MetaStore 是直接 KMS 提供者，則您可重複使用的包裝和簽署金鑰會受到 AWS KMS keyin AWS Key Management Service(AWS KMS) 保護。AWS KMS每次將新的 CMP 版本添加到其內部表或從其內部表中獲取 CMP 版本時MetaStore調用。

設置一個time-to-live值

您可以為您建立的每個最新提供者設定 time-to-live (TTL) 值。一般而言，請使用適合您應用程式的最低 TTL 值。

TTL 值的使用會在「最近的提供者」的`CachingMostRecentProvider`符號中變更。

> **Note**
>
> 最新提供者的`MostRecentProvider`符號已在舊版受支援的 DynamoDB 加密用戶端中取代，並從 2.0.0 版移除。它由符號所取`CachingMostRecentProvider`代。我們建議您盡快更新程式碼。如需詳細資訊，請參閱 最新提供者的更新。

`CachingMostRecentProvider`

`CachingMostRecentProvider`會以兩種不同的方式使用 TTL 值。

- TTL 會決定最新的提供者檢查提供者儲存區是否有新版 CMP 的頻率。如果有新版本可用，則「最新的提供者」會取代其 CMP 並重新整理其加密資料。否則，它將繼續使用其當前的 CMP 和密碼材料。
- TTL 決定可以使用快取記憶體中 CMP 的時間長度。在使用快取的 CMP 進行加密之前，「最新的提供者」會評估其在快取中的時間。如果 CMP 快取時間超過 TTL，則會從快取中逐出 CMP，而「最新的提供者」會從其提供者存放區取得新的最新版本 CMP。

`MostRecentProvider`

在中`MostRecentProvider`，TTL 決定最新的提供者檢查提供者存放區是否有新版 CMP 的頻率。如果有新版本可用，則「最新的提供者」會取代其 CMP 並重新整理其加密資料。否則，它將繼續使用其當前的 CMP 和密碼材料。

TTL 不會決定建立新 CMP 版本的頻率。您可以透過旋轉加密材料來建立新的 CMP 版本。

理想的 TTL 值會隨應用程式及其延遲和可用性目標而有所不同。較低的 TTL 可以減少加密材料儲存在記憶體中的時間，從而改善您的安全性設定檔。此外，較低的 TTL 會更頻繁地重新整理重要資訊。例如，如果您的內部 CMP 是直接 KMS 提供者，它會更頻繁地驗證呼叫者是否仍有權使用. AWS KMS key

但是，如果 TTL 過於短暫，頻繁撥打給提供者商店的電話可能會增加您的成本，並導致供應商商店限制來自您的應用程式以及共用您服務帳戶的其他應用程式的要求。您也可以從 TTL 與旋轉加密材料的速率協調中受益。

在測試期間，請在不同的工作負載下變更 TTL 和快取大小，直到找到適用於應用程式和安全性和效能標準的組態為止。

輪換密碼編譯資料

當最新的供應商需要加密材料時，它始終使用其知道的最新版本的 CMP。它檢查較新版本的頻率取決於您在設定最新提供者時設定的 time-to-live(TTL) 值。

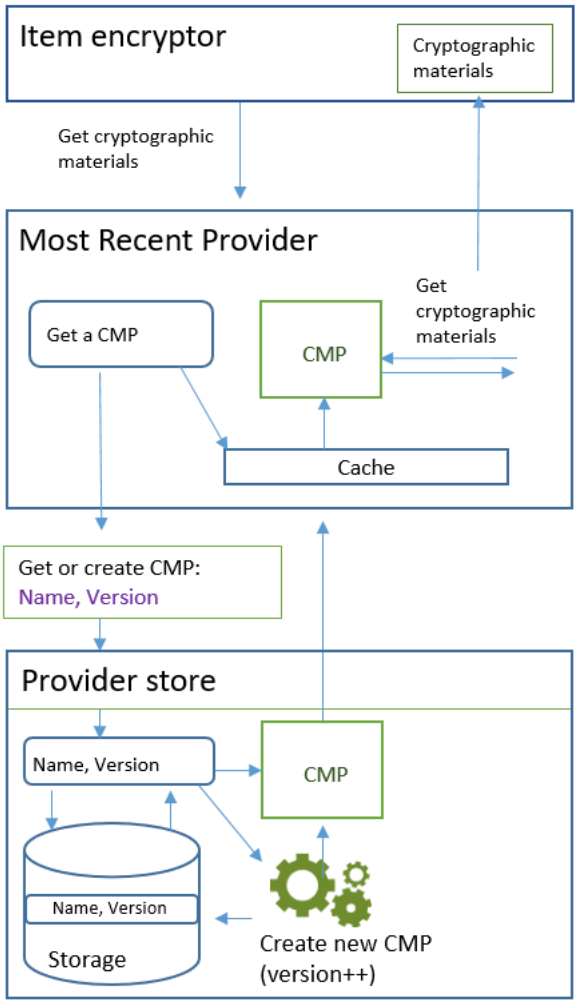
當 TTL 到期時，最新的提供者會檢查提供者存放區是否有較新版本的 CMP。如果有可用的話，最新的提供者會取得它，並取代其快取中的 CMP。它使用此 CMP 及其加密材料，直到發現提供商商店具有更新的版本為止。

若要告知提供者存放區為最近提供者建立新的 CMP 版本，請以最近提供者的資料名稱呼叫提供者存放區的「建立新提供者」操作。提供者存放區會建立新的 CMP，並且在其內部儲存體中使用較大版本號碼儲存已加密的複本。(它也會傳回 CMP，但您可予以捨棄。) 因此，下次「最新的提供者」查詢提供者存放區中其 CMP 的最大版本號時，它會取得新的較大版本號碼，並在後續對存放區的要求中使用它，以查看是否已建立新版本的 CMP。

您可以根據時間、處理的項目或屬性數目，或是對您應用程式有意義的任何其他度量，為您的「建立新提供者」呼叫進行排程。

取得加密資料

最近提供者會使用下列程序 (如下圖所示)，取得它傳回給項目加密程式的加密資料。輸出取決於提供者存放區所傳回的 CMP 類型。最新的提供者可以使用任何相容的提供者存放區，包括 DynamoDB 加密 MetaStore用戶端中包含的提供者存放區。



使用CachingMostRecentProvider符號建立最新的提供者時，您可以指定提供者存放區、最近提供者的名稱以及 time-to-live(TTL) 值。您也可以選擇性地指定快取大小，以決定快取記憶體中可存在的最大密碼編譯資料數目。

當項目加密程式向最近提供者詢問加密資料時，最近提供者會開始搜尋其快取中的 CMP 最新版本。

- 如果它在快取中找到最新版本的 CMP，且 CMP 未超過 TTL 值，則「最新的供應商」會使用 CMP 來產生加密材料。接著，它會將加密資料傳回給項目加密程式。此作業不需要呼叫提供者存放區。
- 如果 CMP 的最新版本不在其快取中，或者它位於快取中但已超過其 TTL 值，則最新版本的提供者會從其提供者存放區要求 CMP。此請求包括最近提供者資料名稱及其所知的最大版本號碼。
  1. 提供者存放區會從其持久性儲存體傳回 CMP。如果提供者存放區是MetaStore，則會使用最新的提供者材料名稱作為分區索引鍵，並使用版本號碼做為排序索引鍵，從其內部 DynamoDB 表格取得加密的「包裝的 CMP」。MetaStore使用其內部項目加密器和內部 CMP 來解密包裝的 CMP。接著，它會將純文字 CMP 傳回給最近提供者。如果內部 CMP 是直接 KMS 提供者，這個步驟就包括對 AWS Key Management Service (AWS KMS) 的呼叫。

2. CMP 會將 amzn-ddb-meta-id 欄位新增至實際資料描述。其值為資料名稱與其內部資料表中的 CMP 版本。提供者存放區會將 CMP 傳回給最近提供者。
3. 最近提供者會快取記憶體中的 CMP。
4. 最近提供者會使用 CMP 來產生加密資料。接著，它會將加密資料傳回給項目加密程式。

取得解密資料

當項目加密程式向最近提供者詢問解密資料時，最近提供者會使用下列程序來取得和傳回解密資料。

1. 最近提供者會向提供者存放區詢問用來加密項目的密碼編譯資料版本號碼。其將從項目的資料描述屬性傳入實際資料描述。
2. 提供者存放區會從實際資料描述中的 amzn-ddb-meta-id 欄位取得加密 CMP 版本號碼，並將它傳回給最近提供者。
3. 最近提供者會在其快取中搜尋用來加密和簽署項目的 CMP 版本。

- 如果發現 CMP 的相符版本位於其快取中，且 CMP 未超過 time-to-live(TTL) 值，則「最新提供者」會使用 CMP 來產生解密資料。接著，它會將解密資料傳回給項目加密程式。此操作不需要呼叫提供者存放區或任何其他 CMP。
- 如果 CMP 的相符版本不在其快取中，或者快取AWS KMS key已超過其 TTL 值，則最新的提供者會從其提供者存放區要求 CMP。其會在請求中傳送資料名稱與加密 CMP 版本號碼。
  1. 提供者存放區會使用最近提供者名稱作為分割區索引鍵並使用版本號碼作為排序索引鍵，在其持久性儲存體中搜尋 CMP。
     - 如果其持久性儲存體中沒有此名稱和版本號碼，則提供者存放區會擲出例外狀況。如果提供者存放區用來產生 CMP，則 CMP 應存放在其持久性儲存體中 (除非它遭到故意刪除)。
     - 如果提供者存放區的持久性儲存體中有具備相符名稱和版本號碼的 CMP，則提供者存放區會將指定的 CMP 傳回給最近提供者。

       如果提供者存放區是一個MetaStore，則會從其 DynamoDB 表格取得加密的 CMP。然後，它會先使用其內部 CMP 提供的密碼編譯資料、將已加密的 CMP 解密，再將 CMP 傳回給最近提供者。如果內部 CMP 是直接 KMS 提供者，這個步驟就包括對 AWS Key Management Service (AWS KMS) 的呼叫。
  2. 最近提供者會快取記憶體中的 CMP。
  3. 最近提供者會使用 CMP 來產生解密資料。接著，它會將解密資料傳回給項目加密程式。

最新提供者的更新

「最近提供者」的符號會從變更MostRecentProvider為CachingMostRecentProvider。

> **Note**
>
> 代表最新提供者的MostRecentProvider符號已在適用於 Java 的 DynamoDB 加密用戶端 1.15 版和 Python 的 DynamoDB 加密用戶端 1.3 版中取代，並在這兩種語言實作中從 DynamoDB 加密用戶端 2.0.0 版中移除。相反地，請使用CachingMostRecentProvider.

會CachingMostRecentProvider實作下列變更：

- 當加密材料在記憶體中的時間超過設定的 time-to-live(TTL) 值時，會CachingMostRecentProvider定期移除記憶體中的加密材料。

  在過程的生命週期內，MostRecentProvider可能會將加密材料存儲在內存中。因此，最新的提供者可能不知道授權變更。在來電者使用它們的權限被撤銷後，它可能會使用加密金鑰。

  如果您無法更新到此新版本，則可以通過定期調用緩存上的clear()方法來獲得類似的效果。此方法會手動清除快取內容，並要求「最新的提供者」要求新的 CMP 和新的密碼編譯材料。
- CachingMostRecentProvider也包含快取大小設定，可讓您更好地控制快取。

若要更新為CachingMostRecentProvider，您必須變更程式碼中的符號名稱。在所有其他方面，與MostRecentProvider. CachingMostRecentProvider 您不需要重新加密任何資料表項目。

不過，CachingMostRecentProvider會產生更多對基礎金鑰基礎結構的呼叫。它會在每個 time-to-live (TTL) 間隔中呼叫提供者儲存區至少一次。具有許多作用中 CMP 的應用程式 (由於頻繁旋轉) 或具有大型叢集的應用程式很可能對此變更很敏感。

在發行更新的程式碼之前，請徹底測試它，以確保更頻繁的呼叫不會損害您的應用程式，也不會因為提供者所依賴的服務造成限制，例如 AWS Key Management Service (AWS KMS) 或 Amazon DynamoDB。若要緩解任何效能問題，請CachingMostRecentProvider根據您觀察到time-to-live的效能特性調整快取大小和的大小。如需準則，請參閱設置一個time-to-live值。

靜態資料提供者

> **Note**
>
> 我們的客戶端加密庫被重命名為AWS數據庫加密 SDK。下列主題提供有關版本 1 的資訊。X-2. 適用於 Java 和版本 1 的 x 個加密用戶端。 X — 3. 適用於 Python 的 x 個加密用戶端。如需詳細資訊，請參閱適用於 DynamoDB 版本支援的資AWS料庫加密 SDK。

靜態材料提供者 (靜態 CMP) 是非常簡單的加密材料供應商 (CMP)，旨在用於測試、proof-of-concept 示範和傳統相容性。

若要使用靜態 CMP 加密資料表項目，請提供進階加密標準 (AES) 對稱加密金鑰與簽署金鑰或金鑰對。您必須提供相同的金鑰，才能將已加密的項目解密。靜態 CMP 不會執行任何密碼編譯操作。它反而會以原狀傳遞您提供給項目加密程式的加密金鑰。項目加密程式會直接以加密金鑰加密項目。然後，直接使用簽署金鑰進行簽署。

因為靜態 CMP 不會產生任何獨特的密碼編譯資料，您處理的所有資料表項目都會使用相同加密金鑰進行加密並由相同的簽署金鑰進行簽署。當您使用相同的金鑰加密眾多項目的屬性值，或使用相同金鑰或金鑰對來簽署所有項目時，可能會超出金鑰的密碼編譯限制。

> **Note**
>
> Java 程式庫中的非對稱靜態提供者不是靜態提供者。其僅提供包裝 CMP 的替代建構函數。此可在生產環境中安全地使用，但只要情況允許，您即應直接使用包裝 CMP。

靜態 CMP 是 DynamoDB 加密用戶端支援的數個加密材料提供者 (CMP) 之一。如需其他 CMP 的相關資訊，請參閱密碼材料供應商。

如需範例程式碼，請參閱：

- Java：SymmetricEncryptedItem

主題

- 使用方式
- 運作方式

使用方式

若要建立靜態提供者，請提供加密金鑰或金鑰對和簽署金鑰或金鑰對。您必須提供金鑰資料才能加密和解密資料表項目。

Java

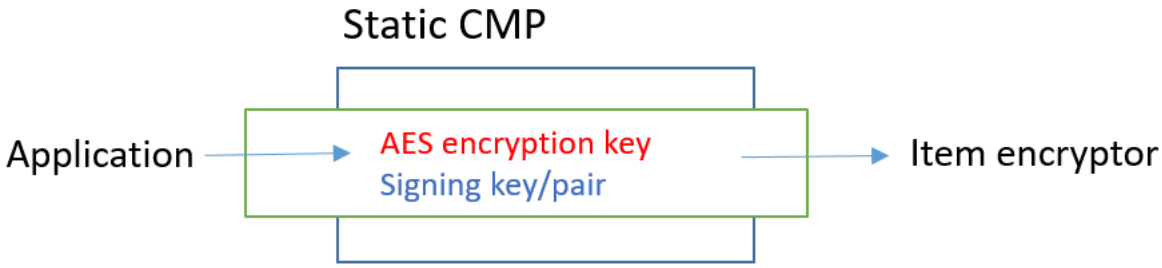
```
// To encrypt
SecretKey cek = ...;        // Encryption key
SecretKey macKey =  ...;    // Signing key
EncryptionMaterialsProvider provider = new SymmetricStaticProvider(cek, macKey);

// To decrypt
SecretKey cek = ...;        // Encryption key
SecretKey macKey =  ...;    // Verification key
EncryptionMaterialsProvider provider = new SymmetricStaticProvider(cek, macKey);
```

Python

```
# You can provide encryption materials, decryption materials, or both
encrypt_keys = EncryptionMaterials(
    encryption_key = ...,
    signing_key = ...
)

decrypt_keys = DecryptionMaterials(
    decryption_key = ...,
    verification_key = ...
)

static_cmp = StaticCryptographicMaterialsProvider(
    encryption_materials=encrypt_keys
    decryption_materials=decrypt_keys
)
```

運作方式

靜態提供者會傳遞您提供給項目加密程式的加密和簽署金鑰，直接用來加密和簽署資料表項目。除非您針對每個項目提供不同的金鑰，否則每個項目都會使用相同金鑰。



取得加密資料

本節將詳細說明靜態資料提供者 (靜態 CMP) 在接收到加密資料請求時的輸入、輸出和處理情形。

輸入 (從應用程式)

- 加密金鑰 — 必須是對稱金鑰，例如進階加密標準 (AES) 金鑰。
- 簽署金鑰 — 可以是對稱金鑰或非對稱金鑰配對。

輸入 (從項目加密程式)

- 加密內容

輸出 (到項目加密程式)

- 當作輸入傳遞的加密金鑰。
- 當作輸入傳遞的簽署金鑰。
- 實際資料描述：請求的資料描述 (如果有) 會維持原狀。

取得解密資料

本節將詳細說明靜態資料提供者 (靜態 CMP) 在接收到解密資料請求時的輸入、輸出和處理情形。

雖然它包括取得加密資料及取得解密資料的個別方法，但是行為相同。

輸入 (從應用程式)

- 加密金鑰 — 必須是對稱金鑰，例如進階加密標準 (AES) 金鑰。
- 簽署金鑰 — 可以是對稱金鑰或非對稱金鑰配對。

輸入 (從項目加密程式)

- 加密內容 (未使用)

輸出 (到項目加密程式)

- 當作輸入傳遞的加密金鑰。
- 當作輸入傳遞的簽署金鑰。

Amazon DynamoDB 密用戶端可用的程式設計語言

> **Note**
>
> 我們的客戶端加密庫被重命名為 AWS 數據庫加密 SDK。下列主題提供有關版本 1 的資訊。X-2. 適用於 Java 和版本 1 的 x 個加密用戶端。 X — 3. 適用於 Python 的 x 個 DynamoDB 密用戶端。如需詳細資訊，請參閱適用於 DynamoDB 版本支援的資AWS 料庫加密 SDK。

Amazon DynamoDB 加密用戶端可用於下列程式設計語言。語言專屬的程式庫有所不同，但是產生的實作都是可互通的。例如，您可以使用 Java 用戶端來加密 (和簽署) 項目，以及使用 Python 用戶端將項目解密。

如需詳細資訊，請參閱相關主題。

主題

- Amazon DynamoDB 加密客戶端
- 適用於 Python 的加密用戶端

Amazon DynamoDB 加密客戶端

> **Note**
>
> 我們的客戶端加密庫被重命名為 AWS 數據庫加密 SDK。下列主題提供有關版本 1 的資訊。X-2. 適用於 Java 和版本 1 的 x 個加密用戶端。 X — 3. 適用於 Python 的 x 個 DynamoDB 密用戶端。如需詳細資訊，請參閱適用於 DynamoDB 版本支援的資AWS 料庫加密 SDK。

本主題說明如何安裝和使用適用於 Java 的 Amazon DynamoDB 加密用戶端。如需使用 DynamoDB 加密用戶端進行程式設計的詳細資訊，請參閱 Java 範例、 aws-dynamodb-encryption-java 儲存庫中的範例 GitHub，以及 DynamoDB 加密用戶端的 Javadoc。

> **Note**
>
> 版本 1. x. Java 的 DynamoDB 驗證加密客戶端的 x 個正在end-of-support 階段生效 2022 年 7 月。請盡快升級至較新的版本。

主題

必要條件

在您安裝適用於 Java 的 Amazon DynamoDB 加密用戶端之前，請確定您具備下列先決條件。

Java 開發環境

您會需要 Java 8 或更新版本。在 Oracle 網站上，移至 Java SE 下載，然後下載並安裝 Java SE 開發套件 (JDK)。

如果您使用 Oracle JDK，您還必須下載並安裝 Java Cryptography Extension (JCE) Unlimited Strength 管轄權政策檔案。

AWS SDK for Java

DynamoDB 加密用戶端需要的 DynamoDB 模組， AWS SDK for Java 即使您的應用程式未與 DynamoDB 互動也一樣。您可以安裝整個 SDK 或只安裝這個模組。如果您使用 Maven，請將 `aws-java-sdk-dynamodb` 新增到 `pom.xml` 檔案。

如需有關安裝和配置的更多資訊 AWS SDK for Java，請參閱AWS SDK for Java。

安裝

您可以使用下列方式安裝適用於 Java 的 Amazon DynamoDB 加密用戶端。

手動

若要安裝適用於 Java 的 Amazon DynamoDB 加密用戶端，請複製或下載儲存aws-dynamodb-encryption-java GitHub 庫。

使用 Apache Maven

適用於 Java 的 Amazon DynamoDB 加密用戶端可透過 Apache Maven 使用，並具有下列相依性定義。

```
<dependency>
  <groupId>com.amazonaws</groupId>
  <artifactId>aws-dynamodb-encryption-java</artifactId>
  <version>version-number</version>
</dependency>
```

安裝 SDK 之後，請查看本指南中的範例程式碼以及開啟 DynamoDB 加密用戶端 Javadoc 以開始使用。 GitHub

使用 DynamoDB 密用戶端

> **Note**
>
> 我們的客戶端加密庫被重命名為 AWS 數據庫加密 SDK。下列主題提供有關版本 1 的資訊。 X-2. 適用於 Java 和版本 1 的 x 個加密用戶端。 X — 3. 適用於 Python 的 x 個 DynamoDB 密用戶端。如需詳細資訊，請參閱適用於 DynamoDB 版本支援的資AWS 料庫加密 SDK。

本主題說明 Java 中 DynamoDB 加密用戶端的一些功能，這些功能可能在其他程式設計語言實作中找不到這些功能。

如需有關使用 DynamoDB 加密用戶端進行程式設計的詳細資訊，請參閱`aws-dynamodb-encryption-java repository`上 GitHub的 Java 範例、中的範例以及 DynamoDB 加密用戶端的 Javadoc。

主題

- 項目加密器： AttributeEncryptor 和 DynamoDBEncryptor
- 設定儲存行為
- Java 中的屬性動作

- 覆寫表格名稱

項目加密器： AttributeEncryptor 和 DynamoDBEncryptor

Java 中的 DynamoDB 加密用戶端有兩個項目加密器：較低層級的. AttributeEncryptor

這AttributeEncryptor是一個協助程式類別，可協助您在 DynamoDB 加密用戶端中 AWS SDK for Java 搭配使用 DynamoDBMapper。DynamoDB Encryptor當您搭配 DynamoDBMapper 使用 AttributeEncryptor 時，它會在您保存項目時，透明地加密和簽署您的項目。當您載入項目時，它也會透明地驗證和解密您的項目。

設定儲存行為

您可以使用AttributeEncryptor和DynamoDBMapper來新增或取代為僅簽署或加密和已簽署的屬性的表格項目。對於這些任務，我們建議您將其設定為使用 PUT 儲存行為，如下列範例所示。否則，您將可能無法解密資料。

```
DynamoDBMapperConfig mapperConfig =
 DynamoDBMapperConfig.builder().withSaveBehavior(SaveBehavior.PUT).build();
DynamoDBMapper mapper = new DynamoDBMapper(ddb, mapperConfig, new
 AttributeEncryptor(encryptor));
```

如果您使用預設儲存行為 (此行為僅更新表格項目中模型化的屬性)，則未建模的屬性不會包含在簽章中，且不會因表格寫入而變更。因此，稍後讀取所有屬性時，簽名將不會驗證，因為它不包含未建模的屬性。

您也可以使用 CLOBBER 儲存行為。這種行為與 PUT 儲存行為完全相同，差別在於它會停用樂觀鎖定並覆寫表格中的項目。

為了防止簽章錯誤，如果未設定或的儲存行為，則 DynamoDB 加密用戶端會擲回執行階段例外狀況。AttributeEncryptor DynamoDBMapper CLOBBER PUT

若要查看範例中使用的此程式碼使用 DynamoDBMapper，請參閱中aws-dynamodb-encryption-java存放庫中 GitHub的 AwsKmsEncryptedObject.java 範例。

Java 中的屬性動作

屬性動作決定哪些屬性值會加密並簽署，哪些屬性值只會簽署，以及哪些屬性值會予忽略。您用來指定屬性動作的方法取決於您使用的是 DynamoDBMapper 和 AttributeEncryptor，還是較低層層級的 DynamoDBEncryptor。

> **Important**
>
> 使用屬性動作加密表格項目後，從資料模型新增或移除屬性可能會導致簽章驗證錯誤，讓您無法解密資料。如需更詳細的說明，請參閱變更您的資料模型。

DynamoDBMapper 的屬性動作

當您使用 DynamoDBMapper 和 AttributeEncryptor 時，您可使用註釋來指定屬性動作。DynamoDB 加密用戶端使用標準 DynamoDB 屬性註釋來定義屬性類型，以決定如何保護屬性。除了主要索引鍵 (簽署但不加密) 以外，所有屬性都預設會進行加密和簽署。

> **Note**
>
> 不要使用 @DynamoDB VersionAttribute 註釋加密屬性的值，儘管您可以（也應該）對其進行簽名。否則，使用其值的情況將會造成想不到的影響。

```
// Attributes are encrypted and signed
@DynamoDBAttribute(attributeName="Description")

// Partition keys are signed but not encrypted
@DynamoDBHashKey(attributeName="Title")

// Sort keys are signed but not encrypted
@DynamoDBRangeKey(attributeName="Author")
```

若要指定例外狀況，請使用在適用於 Java 的 DynamoDB 加密用戶端中定義的加密註解。如果您在類別層級指定例外狀況，這些例外狀況就會成為類別的預設值。

```
// Sign only
@DoNotEncrypt

// Do nothing; not encrypted or signed
@DoNotTouch
```

例如，這些註釋會簽署但不會加密 PublicationYear 屬性，且不會加密或簽署 ISBN 屬性值。

```
// Sign only (override the default)
```

```
@DoNotEncrypt
@DynamoDBAttribute(attributeName="PublicationYear")

// Do nothing (override the default)
@DoNotTouch
@DynamoDBAttribute(attributeName="ISBN")
```

DynamoDBEncryptor 的屬性動作

若要在您直接使用 DynamoDBEncryptor 時指定屬性動作，請建立 HashMap 物件，其中的名稱值組代表屬性名稱和指定的動作。

屬性動作的有效值會定義於 EncryptionFlags 列舉類型中。您可以同時使用 ENCRYPT 與 SIGN、單獨使用 SIGN，或省略兩者。但是，如果您ENCRYPT單獨使用，DynamoDB 加密用戶端會擲回錯誤。您無法加密您未簽署的屬性。

```
ENCRYPT
SIGN
```

> **Warning**
>
> 請勿加密主索引鍵的屬性。它們必須保持純文字格式，以便 DynamoDB 可以在不執行完整表格掃描的情況下尋找項目。

如果您在加密內容中指定主金鑰，然後ENCRYPT在任一主索引鍵屬性的屬性動作中指定，DynamoDB 加密用戶端會擲回例外狀況。

例如，下列 Java 程式碼會建立一actions HashMap 個加密並簽署record項目中的所有屬性。例外狀況是分割索引鍵和排序索引鍵屬性 (已簽署但未加密)，以及未簽署或加密的 test 屬性。

```
final EnumSet<EncryptionFlags> signOnly = EnumSet.of(EncryptionFlags.SIGN);
final EnumSet<EncryptionFlags> encryptAndSign = EnumSet.of(EncryptionFlags.ENCRYPT,
 EncryptionFlags.SIGN);
final Map<String, Set<EncryptionFlags>> actions = new HashMap<>();

for (final String attributeName : record.keySet()) {
  switch (attributeName) {
    case partitionKeyName: // no break; falls through to next case
    case sortKeyName:
      // Partition and sort keys must not be encrypted, but should be signed
```

```
        actions.put(attributeName, signOnly);
        break;
      case "test":
        // Don't encrypt or sign
        break;
      default:
        // Encrypt and sign everything else
        actions.put(attributeName, encryptAndSign);
        break;
    }
}
```

然後，當您呼叫的 En encryptRecord 方法時DynamoDBEncryptor，請將對應指定為參數的attributeFlags值。例如，encryptRecord 的這項呼叫會使用 actions 映射。

```
// Encrypt the plaintext record
final Map<String, AttributeValue> encrypted_record = encryptor.encryptRecord(record,
 actions, encryptionContext);
```

覆寫表格名稱

在 DynamoDB 加密用戶端中，DynamoDB 表格的名稱是 DynamoDB 加密內容的一個元素，會傳遞至加密和解密方法。當您加密或簽署表格項目時，DynamoDB 加密內容 (包括資料表名稱) 會以密碼編譯方式繫結至加密文字。如果傳遞至解密方法的 DynamoDB 加密內容與傳遞給加密方法的 DynamoDB 加密內容不符，解密作業就會失敗。

有時候，資料表名稱會變更，例如當您備份資料表或執行point-in-time 復原時。解密或驗證這些項目的簽章時，您必須傳入用於加密和簽署項目的相同 DynamoDB 加密內容，包括原始表格名稱。目前的表格名稱是不需要的。

使用時DynamoDBEncryptor，您可以手動組合 DynamoDB 加密內容。但是，如果您使用的是DynamoDBMapper，會為您AttributeEncryptor建立 DynamoDB 加密內容，包括目前的表格名稱。若要告訴 AttributeEncryptor 建立具有不同資料表名稱的加密內容，請使用 EncryptionContextOverrideOperator。

例如，下列程式碼會建立密碼編譯材料提供者 (CMP) 和 DynamoDBEncryptor。然後它會呼叫 DynamoDBEncryptor 的 setEncryptionContextOverrideOperator 方法。它使用覆寫一個表格名稱的 overrideEncryptionContextTableName 運算子。以這種方式進行設定時，AttributeEncryptor會建立一個 DynamoDB 加密內容，其newTableName中包含代替. oldTableName 如需完整範例，請參閱 EncryptionContextOverridesWithDynamoDBMapper.java。

```
final DirectKmsMaterialProvider cmp = new DirectKmsMaterialProvider(kms, keyArn);
final DynamoDBEncryptor encryptor = DynamoDBEncryptor.getInstance(cmp);

encryptor.setEncryptionContextOverrideOperator(EncryptionContextOperators.overrideEncryptionCon
                oldTableName, newTableName));
```

當您呼叫 DynamoDBMapper (解密和驗證項目) 的載入方法時，您可以指定原始資料表名稱。

```
mapper.load(itemClass, DynamoDBMapperConfig.builder()

 .withTableNameOverride(DynamoDBMapperConfig.TableNameOverride.withTableNameReplacement(oldTabl
                .build());
```

您也可以使用 overrideEncryptionContextTableNameUsingMap 運算子，該運算子會覆寫多個表格名稱。

表格名稱覆寫運算子通常用於解密資料和驗證簽章。不過，您可以使用它們在加密和簽署時，將 DynamoDB 加密內容中的表格名稱設定為不同的值。

如果您使用的是 DynamoDBEncryptor，請勿使用表格名稱覆寫運算子。請改為使用原始表格名稱建立加密內容，並將其提交至解密方法。

Java 加密用戶端的範例程式碼

> **Note**
>
> 我們的客戶端加密庫被重命名為 AWS 數據庫加密 SDK。下列主題提供有關版本 1 的資訊。X-2. 適用於 Java 和版本 1 的 x 個加密用戶端。 X — 3. 適用於 Python 的 x 個 DynamoDB 密用戶端。如需詳細資訊，請參閱適用於 DynamoDB 版本支援的資AWS 料庫加密 SDK。

下列範例說明如何使用適用於 Java 的 DynamoDB 加密用戶端來保護應用程式中的 DynamoDB 表格項目。您可以在aws-dynamodb-encryption-java存儲庫的示例目錄中找到更多示例（並提供您自己的示例）GitHub。

主題

- 使用 DynamoDBEncryptor
- 使用 DynamoDBMapper

使用 DynamoDBEncryptor

這個範例說明如何使用較低層級的 DynamoDBEncryptor 搭配直接 KMS 提供者。直接 KMS 提供者會根據您指定的 AWS KMS keyin AWS Key Management Service (AWS KMS) 產生並保護其加密資料。

您可以搭配使用任何相容的加密資料提供者 (CMP)DynamoDBEncryptor，也可以使用直接 KMS 提供者和. DynamoDBMapper AttributeEncryptor

請參閱完整的代碼示例：AwsKmsEncryptedItem.java

步驟 1：建立直接 KMS 提供者

建立具有指定區域的用 AWS KMS 戶端執行個體。然後，使用用戶端執行個體建立具有您偏好的 Direct KMS 提供者執行個體 AWS KMS key。

此範例使用 Amazon 資源名稱 (ARN) 來識別 AWS KMS key，但您可以使用任何有效的金鑰識別碼。

```
final String keyArn = 'arn:aws:kms:us-
west-2:111122223333:key/1234abcd-12ab-34cd-56ef-1234567890ab'
final String region = 'us-west-2'

final AWSKMS kms = AWSKMSClientBuilder.standard().withRegion(region).build();
final DirectKmsMaterialProvider cmp = new DirectKmsMaterialProvider(kms, keyArn);
```

步驟 2：建立項目

這個範例record HashMap 會定義代表範例資料表項目的一個。

```
final String partitionKeyName = "partition_attribute";
final String sortKeyName = "sort_attribute";

final Map<String, AttributeValue> record = new HashMap<>();
record.put(partitionKeyName, new AttributeValue().withS("value1"));
record.put(sortKeyName, new AttributeValue().withN("55"));
record.put("example", new AttributeValue().withS("data"));
record.put("numbers", new AttributeValue().withN("99"));
record.put("binary", new AttributeValue().withB(ByteBuffer.wrap(new byte[]{0x00,
 0x01, 0x02})));
record.put("test", new AttributeValue().withS("test-value"));
```

步驟 3：建立 DynamoDBEncryptor

使用直接 KMS 提供者建立 DynamoDBEncryptor 的執行個體。

```
final DynamoDBEncryptor encryptor = DynamoDBEncryptor.getInstance(cmp);
```

步驟 4：建立 DynamoDB 加密內容

DynamoDB 加密內容包含資料表結構及其加密和簽署方式的相關資訊。如果您使用 DynamoDBMapper，AttributeEncryptor 會為您建立加密細節。

```
final String tableName = "testTable";

final EncryptionContext encryptionContext = new EncryptionContext.Builder()
    .withTableName(tableName)
    .withHashKeyName(partitionKeyName)
    .withRangeKeyName(sortKeyName)
    .build();
```

步驟 5：建立屬性動作物件

屬性動作決定項目的哪些屬性會加密並簽署，哪些屬性只會簽署，以及哪些屬性不會加密或簽署。

在 Java 中，若要指定屬性動作，您可以建立屬性 HashMap 名稱和EncryptionFlags值配對的屬性。

例如，下列 Java 程式碼會建立一actions HashMap 個加密並簽署record項目中的所有屬性，但分割索引鍵和排序索引鍵屬性除外 (已簽署但未加密)，以及未簽署或加密的test屬性。

```
final EnumSet<EncryptionFlags> signOnly = EnumSet.of(EncryptionFlags.SIGN);
final EnumSet<EncryptionFlags> encryptAndSign = EnumSet.of(EncryptionFlags.ENCRYPT,
 EncryptionFlags.SIGN);
final Map<String, Set<EncryptionFlags>> actions = new HashMap<>();

for (final String attributeName : record.keySet()) {
  switch (attributeName) {
    case partitionKeyName: // fall through to the next case
    case sortKeyName:
      // Partition and sort keys must not be encrypted, but should be signed
      actions.put(attributeName, signOnly);
      break;
    case "test":
      // Neither encrypted nor signed
```

```
      break;
    default:
      // Encrypt and sign all other attributes
      actions.put(attributeName, encryptAndSign);
      break;
  }
}
```

步驟 6：將項目加密並簽署

若要加密並簽署資料表項目，請在 encryptRecord 的執行個體上呼叫 DynamoDBEncryptor 方法。指定資料表項目 (record)、屬性動作 (actions) 及加密細節 (encryptionContext)。

```
final Map<String, AttributeValue> encrypted_record = encryptor.encryptRecord(record,
 actions, encryptionContext);
```

步驟 7：將項目放入 DynamoDB 表格中

最後，將加密和已簽署的項目放入 DynamoDB 表格中。

```
final AmazonDynamoDB ddb = AmazonDynamoDBClientBuilder.defaultClient();
ddb.putItem(tableName, encrypted_record);
```

使用 DynamoDBMapper

下列範例顯示如何搭配直接 KMS 提供者使用 DynamoDB 對應程式協助程式類別。直接 KMS 提供者會根據您指定的 AWS KMS keyin AWS Key Management Service (AWS KMS) 產生並保護其加密資料。

您可以使用任何相容的密碼編譯資料提供者 (CMP) 搭配 DynamoDBMapper，而且可以使用直接 KMS 提供者搭配較低層級的 DynamoDBEncryptor。

請參閱完整的代碼示例：AwsKmsEncryptedObject.java

步驟 1：建立直接 KMS 提供者

建立具有指定區域的用 AWS KMS 戶端執行個體。然後，使用用戶端執行個體建立具有您偏好的 Direct KMS 提供者執行個體 AWS KMS key。

此範例使用 Amazon 資源名稱 (ARN) 來識別 AWS KMS key，但您可以使用任何有效的金鑰識別碼。

```
final String keyArn = 'arn:aws:kms:us-
west-2:111122223333:key/1234abcd-12ab-34cd-56ef-1234567890ab'
final String region = 'us-west-2'

final AWSKMS kms = AWSKMSClientBuilder.standard().withRegion(region).build();
final DirectKmsMaterialProvider cmp = new DirectKmsMaterialProvider(kms, keyArn);
```

步驟 2：建立 DynamoDB 加密程式和 DynamoDBMapper

使用您在上一個步驟中建立的直接 KMS 提供者建立 DynamoDB 加密器的執行個體。您必須具現化較低層級的 DynamoDB 加密程式，才能使用 DynamoDB 對應程式。

接下來，建立 DynamoDB 資料庫的執行個體和對應程式組態，並使用它們建立 DynamoDB 對應器的執行個體。

> **Important**
>
> 使用 DynamoDBMapper 加入或編輯已簽署 (或已加密並簽署) 的項目時，請將其設定成使用儲存行為 (例如 PUT) 納入所有屬性，如以下範例所示。否則，您將可能無法解密資料。

```
final DynamoDBEncryptor encryptor = DynamoDBEncryptor.getInstance(cmp)
final AmazonDynamoDB ddb =
 AmazonDynamoDBClientBuilder.standard().withRegion(region).build();

DynamoDBMapperConfig mapperConfig =
 DynamoDBMapperConfig.builder().withSaveBehavior(SaveBehavior.PUT).build();
DynamoDBMapper mapper = new DynamoDBMapper(ddb, mapperConfig, new
 AttributeEncryptor(encryptor));
```

步驟 3：定義您的 DynamoDB 資料表

接下來，定義您的 DynamoDB 資料表。請使用註釋指定屬性動作。此範例會建立 DynamoDB 資料表`ExampleTable`，以及代表資料表項目的`DataPoJo`類別。

該範例資料表的主索引鍵屬性將經過簽署但未加密。這包括了標註 @DynamoDBHashKey 的 `partition_attribute`，以及標註 @DynamoDBRangeKey 的 `sort_attribute`。

凡是標註 @DynamoDBAttribute 的屬性 (例如 `some numbers`) 都將進行加密並簽署。例外情況是使用 DynamoDB 加密用戶端定義的 @DoNotTouch (僅簽署) 或 (不加密或簽署) 加密註釋的屬

性。@DoNotEncrypt例如，leave me 屬性由於設有 @DoNotTouch 註釋，其將不會進行加密或簽署。

```
@DynamoDBTable(tableName = "ExampleTable")
public static final class DataPoJo {
  private String partitionAttribute;
  private int sortAttribute;
  private String example;
  private long someNumbers;
  private byte[] someBinary;
  private String leaveMe;

  @DynamoDBHashKey(attributeName = "partition_attribute")
  public String getPartitionAttribute() {
    return partitionAttribute;
  }

  public void setPartitionAttribute(String partitionAttribute) {
    this.partitionAttribute = partitionAttribute;
  }

  @DynamoDBRangeKey(attributeName = "sort_attribute")
  public int getSortAttribute() {
    return sortAttribute;
  }

  public void setSortAttribute(int sortAttribute) {
    this.sortAttribute = sortAttribute;
  }

  @DynamoDBAttribute(attributeName = "example")
  public String getExample() {
    return example;
  }

  public void setExample(String example) {
    this.example = example;
  }

  @DynamoDBAttribute(attributeName = "some numbers")
  public long getSomeNumbers() {
    return someNumbers;
  }
```

```
  public void setSomeNumbers(long someNumbers) {
    this.someNumbers = someNumbers;
  }

  @DynamoDBAttribute(attributeName = "and some binary")
  public byte[] getSomeBinary() {
    return someBinary;
  }

  public void setSomeBinary(byte[] someBinary) {
    this.someBinary = someBinary;
  }

  @DynamoDBAttribute(attributeName = "leave me")
  @DoNotTouch
  public String getLeaveMe() {
    return leaveMe;
  }

  public void setLeaveMe(String leaveMe) {
    this.leaveMe = leaveMe;
  }

  @Override
  public String toString() {
    return "DataPoJo [partitionAttribute=" + partitionAttribute + ", sortAttribute="
        + sortAttribute + ", example=" + example + ", someNumbers=" + someNumbers
        + ", someBinary=" + Arrays.toString(someBinary) + ", leaveMe=" + leaveMe +
 "]";
  }
}
```

步驟 4：加密並儲存資料表項目

現在，當您建立資料表項目並使用 DynamoDB 對應器儲存該項目時，項目會在新增至表格之前自動加密並簽署。

本範例定義的資料表項目名為 `record`。該項目儲存至資料表之前，其屬性將根據 `DataPoJo` 類別中的註釋進行加密與簽署。就本例而言，所有屬性除了 `PartitionAttribute`、`SortAttribute` 和 `LeaveMe` 以外都將加密並簽署。`PartitionAttribute` 和 `SortAttributes` 只會進行簽署，而 `LeaveMe` 屬性則完全未加密或簽署。

若要加密並簽署 `record` 項目，然後將其加入至 `ExampleTable`，請呼叫 `DynamoDBMapper` 類別的 `save` 方法。由於 DynamoDB 對應器已設定為使用PUT儲存行為，因此該項目會以相同的主索引鍵取代任何項目，而不是更新它。這可確保簽章相符，而且您從資料表取得該項目後便能將其解密。

```
DataPoJo record = new DataPoJo();
record.setPartitionAttribute("is this");
record.setSortAttribute(55);
record.setExample("data");
record.setSomeNumbers(99);
record.setSomeBinary(new byte[]{0x00, 0x01, 0x02});
record.setLeaveMe("alone");

mapper.save(record);
```

適用於 Python 的加密用戶端

> **Note**
>
> 我們的客戶端加密庫被重命名為 AWS 數據庫加密 SDK。下列主題提供有關版本 1 的資訊。X-2. 適用於 Java 和版本 1 的 x 個加密用戶端。 X — 3. 適用於 Python 的 x 個 DynamoDB 密用戶端。如需詳細資訊，請參閱適用於 DynamoDB 版本支援的資AWS 料庫加密 SDK。

本主題說明如何安裝和使用適用於 Python 的 DynamoDB 加密用戶端。您可以在的aws-dynamodb-encryption-python存放庫中找到程式碼 GitHub，包括完整且經過測試的範例程式碼，以協助您開始使用。

> **Note**
>
> 版本 1. x. x 和 2. x. 另一個適用於 Python 的 DynamoDB 加密用戶端將於 2022 年 7 月end-of-support 階段生效。請盡快升級至較新的版本。

主題

- 必要條件
- 安裝

- 使用 DynamoDB 密用戶端
- 適用於 Python 的加密用戶端範例程式碼

必要條件

在您安裝適用於 Python 的 Amazon DynamoDB 加密用戶端之前，請確定您具備下列先決條件。

支援的 Python 版本

Amazon DynamoDB 3.3.0 及更新版本的加密用戶端需要 Python 3.8 或更新版本。若要下載 Python，請參閱 Python 下載。

舊版 Amazon DynamoDB 加密用戶端支援 Python 2.7 和 Python 3.4 及更新版本，但我們建議您使用最新版本的 DynamoDB 加密用戶端。

適用於 Python 的 pip 安裝工具

Python 3.6 及更高版本包含點子，儘管您可能想升級它。有關升級或安裝 pip 的更多信息，請參閱 pip 文檔中的安裝。

安裝

使用 pip 安裝適用於 Python 的 Amazon DynamoDB 加密用戶端，如下列範例所示。

若要安裝最新版本

```
pip install dynamodb-encryption-sdk
```

如需使用 pip 來安裝及升級套件的詳細資訊，請參閱安裝套件。

DynamoDB 加密用戶端需要所有平台上的加密程式庫。Windows 上所有版本的 pip 都將安裝並建置密碼編譯程式庫，而 Linux 上的 pip 8.1 和更新版本則會安裝並建置密碼編譯。如果您使用舊版 pip，而且您的 Linux 環境沒有建置密碼編譯程式庫所需的工具，您就需要加以安裝。如需詳細資訊，請參閱在 Linux 上建置密碼編譯。

您可以從上的aws-dynamodb-encryption-python存放庫取得最新的 DynamoDB 加密用戶端開發版本。GitHub

安裝 DynamoDB 加密用戶端之後，請參閱本指南中的範例 Python 程式碼開始使用。

使用 DynamoDB 密用戶端

> **Note**
>
> 我們的客戶端加密庫被重命名為 AWS 數據庫加密 SDK。下列主題提供有關版本 1 的資訊。X-2. 適用於 Java 和版本 1 的 x 個加密用戶端。 X — 3. 適用於 Python 的 x 個 DynamoDB 密用戶端。如需詳細資訊，請參閱適用於 DynamoDB 版本支援的資AWS 料庫加密 SDK。

本主題說明適用於 Python 的 DynamoDB 加密用戶端的一些功能，這些功能可能在其他程式設計語言實作中找不到這些功能。這些功能旨在讓您以最安全的方式更輕鬆地使用 DynamoDB 加密用戶端。若非遇到罕見的使用案例，我們都建議您使用這些功能。

如需使用 DynamoDB 加密用戶端進行程式設計的詳細資訊，請參閱本指南中的 Python 範例、上的 aws-dynamodb-encryption-python 存放庫中的範例 GitHub，以及 DynamoDB 加密用戶端的 Python 文件。

主題

- 用戶端協助程式類別
- TableInfo 類
- Python 中的屬性動作

用戶端協助程式類別

適用於 Python 的 DynamoDB 加密用戶端包含數個用戶端協助程式類別，這些用戶端協助程式類別會鏡像 DynamoDB 的 3 個類別。這些輔助程式類別的設計可讓您更輕鬆地將加密和簽署新增至現有 DynamoDB 應用程式，並避免最常見的問題，如下所示：

- 通過向對象添加主鍵的覆蓋操作，或者在AttributeActions對`AttributeActions`象明確告訴客戶端加密主鍵時拋出異常，防止您加密項目中的主鍵。如果 `AttributeActions` 物件中的預設動作為 `DO_NOTHING`，則用戶端協助程式類別會將該動作用於主索引鍵。否則會使用 `SIGN_ONLY`。
- 建立TableInfo 物件並根據對 DynamoDB 的呼叫填入 DynamoDB 加密內容。這有助於確保 DynamoDB 加密內容正確無誤，而且用戶端可以識別主要金鑰。
- 當您寫入 DynamoDB 表格或從中讀取資料表項目時`get_item`，可透明地加密和解密資料表項目的 Support 援方法，例如和。`put_item`只有 `update_item` 方法不受支援。

您可以使用用戶端協助程式類別，而無須直接與較低層級的項目加密程式互動。只有在需要於項目加密程式中設定進階選項時，才使用這些類別。

用戶端協助程式類別包括：

- EncryptedTable適用於在 DynamoDB 中使用表格資源的應用程式，一次處理一個表格。
- EncryptedResource適用於在 DynamoDB 中使用服務資源類別進行批次處理的應用程式。
- EncryptedClient適用於在 DynamoDB 中使用較低層級用戶端的應用程式。

若要使用用戶端協助程式類別，呼叫者必須具有在目標資料表上呼叫 DynamoDB DescribeTable作業的權限。

TableInfo 類

此TableInfo類別是代表 DynamoDB 表的協助程式類別，其中包含主索引鍵和次要索引的欄位。它可協助您取得正確而即時的資料表相關資訊。

如果您使用的是用戶端協助程式類別，其將為您建立並使用 `TableInfo` 物件。否則，您可以明確建立一個物件。如需範例，請參閱使用項目加密程式。

當您呼叫`TableInfo`物件上的`refresh_indexed_attributes`方法時，它會透過呼叫 DynamoDB DescribeTable作業填入物件的屬性值。查詢資料表會比進行索引名稱的硬編碼可靠得多。此`TableInfo`類別也包含提`encryption_context_values`供 DynamoDB 加密內容所需值的屬性。

若要使用此方`refresh_indexed_attributes`法，呼叫者必須具有在目標資料表上呼叫 DynamoDB DescribeTable作業的權限。

Python 中的屬性動作

屬性動作會告知項目加密程式要對項目的每個屬性執行什麼動作。若要指定 Python 中的屬性動作，請建立具有預設動作和特定屬性之任何例外狀況的 `AttributeActions` 物件。有效值會在 `CryptoAction` 列舉類型中加以定義。

> **Important**
>
> 使用屬性動作加密表格項目後，從資料模型新增或移除屬性可能會導致簽章驗證錯誤，讓您無法解密資料。如需更詳細的說明，請參閱變更您的資料模型。

```
DO_NOTHING = 0
SIGN_ONLY = 1
ENCRYPT_AND_SIGN = 2
```

例如，此 AttributeActions 物件會建立 ENCRYPT_AND_SIGN 作為所有屬性的預設值，並指定 ISBN 和 PublicationYear 屬性的例外狀況。

```
actions = AttributeActions(
    default_action=CryptoAction.ENCRYPT_AND_SIGN,
    attribute_actions={
        'ISBN': CryptoAction.DO_NOTHING,
        'PublicationYear': CryptoAction.SIGN_ONLY
    }
)
```

若您使用的是用戶端協助程式類別，您不需要指定主索引鍵屬性的屬性動作。用戶端協助程式類別可防止您將主索引鍵加密。

若未使用用戶端協助程式類別，且預設動作為 ENCRYPT_AND_SIGN，則必須指定主索引鍵的動作。建議的主索引鍵動作為 SIGN_ONLY。若要加以簡化，請使用會對主索引鍵使用 SIGN_ONLY 的 set_index_keys 方法，或是在預設動作為 DO_NOTHING 時使用此動作。

> **Warning**
>
> 請勿加密主索引鍵的屬性。它們必須保持純文字格式，以便 DynamoDB 可以在不執行完整表格掃描的情況下尋找項目。

```
actions = AttributeActions(
    default_action=CryptoAction.ENCRYPT_AND_SIGN,
)
actions.set_index_keys(*table_info.protected_index_keys())
```

適用於 Python 的加密用戶端範例程式碼

> **Note**
>
> 我們的客戶端加密庫被重命名為 AWS 數據庫加密 SDK。下列主題提供有關版本 1 的資訊。X-2. 適用於 Java 和版本 1 的 x 個加密用戶端。 X — 3. 適用於 Python 的 x 個 DynamoDB 密用戶端。如需詳細資訊，請參閱適用於 DynamoDB 版本支援的資AWS 料庫加密 SDK。

下列範例說明如何使用適用於 Python 的 DynamoDB 加密用戶端來保護應用程式中的 DynamoDB 資料。您可以在aws-dynamodb-encryption-python存儲庫的示例目錄中找到更多示例（並提供您自己的示例） GitHub。

主題

- 使用用 EncryptedTable 戶端輔助程式類別
- 使用項目加密程式

使用用 EncryptedTable 戶端輔助程式類別

下列範例示範如何使用直接 KMS 提供者搭配用`EncryptedTable`戶端協助程式類別。此範例使用與下列範使用項目加密程式例相同的密碼編譯材料提供者。不過，其將使用 `EncryptedTable` 類別，而不是直接與較低層級的項目加密程式互動。

比較這些範例，您就可以看見用戶端協助程式類別為您所做的事情。這包括建立 DynamoDB 加密內容，以及確保主要金鑰屬性永遠簽署，但永遠不會加密。若要建立加密內容並探索主要金鑰，用戶端協助程式類別會呼叫 DynamoDB DescribeTable作業。若要執行此程式碼，您必須擁有呼叫此作業的權利。

查看完整的程式碼範例：aws_kms_encrypted_table.py

步驟 1：建立資料表

首先建立具有表格名稱的標準 DynamoDB 表格執行個體。

```
table_name='test-table'
table = boto3.resource('dynamodb').Table(table_name)
```

步驟 2：建立密碼編譯資料提供者

建立您所選密碼編譯資料提供者 (CMP) 的執行個體。

本範例使用直接 KMS 提供者，但您可以使用任何相容的 CMP。若要建立直接 KMS 提供者，請指定 AWS KMS key. 此範例使用的 Amazon 資源名稱 (ARN) AWS KMS key，但您可以使用任何有效的金鑰識別碼。

```
kms_key_id='arn:aws:kms:us-
west-2:111122223333:key/1234abcd-12ab-34cd-56ef-1234567890ab'
kms_cmp = AwsKmsCryptographicMaterialsProvider(key_id=kms_key_id)
```

步驟 3：建立屬性動作物件

屬性動作會告知項目加密程式要對項目的每個屬性執行什麼動作。這個範例中的 AttributeActions 物件會加密並簽署所有項目，但 test 屬性 (予以忽略) 除外。

當您使用用戶端協助程式類別時，請勿指定主要索引鍵屬性的屬性動作。EncryptedTable 類別會簽署 (但絕不會加密) 主要索引鍵屬性。

```
actions = AttributeActions(
    default_action=CryptoAction.ENCRYPT_AND_SIGN,
    attribute_actions={'test': CryptoAction.DO_NOTHING}
)
```

步驟 4：建立已加密的資料表

使用標準資料表、直接 KMS 提供者和屬性動作，來建立已加密的資料表。這個步驟可完成設定。

```
encrypted_table = EncryptedTable(
    table=table,
    materials_provider=kms_cmp,
    attribute_actions=actions
)
```

步驟 5：在資料表中放入純文字項目

當您呼叫上的put_item方法時encrypted_table，您的表格項目會透明地加密、簽署，並新增至 DynamoDB 表格。

首先，定義資料表項目。

```
plaintext_item = {
    'partition_attribute': 'value1',
    'sort_attribute': 55
    'example': 'data',
```

```
    'numbers': 99,
    'binary': Binary(b'\x00\x01\x02'),
    'test': 'test-value'
}
```

然後，在資料表中放入該項目。

```
encrypted_table.put_item(Item=plaintext_item)
```

若要以加密形式從 DynamoDB 表格取得項目，請呼叫物件上的get_itemtable方法。若要取得已解密的項目，請在 get_item 物件上呼叫 encrypted_table 方法。

使用項目加密程式

此範例說明如何在加密表格項目時直接與 DynamoDB 加密用戶端中的項目加密程式互動，而不是使用與項目加密器互動的用戶端協助程式類別。

使用此技術時，您可以手動建立 DynamoDB 加密內容和設定物件 (CryptoConfig)。此外，您可以在一次呼叫中加密項目，並將它們放入 DynamoDB 表格中的個別呼叫。這可讓您自訂put_item呼叫，並使用 DynamoDB 加密用戶端來加密和簽署永遠不會傳送至 DynamoDB 的結構化資料。

本範例使用直接 KMS 提供者，但您可以使用任何相容的 CMP。

查看完整的程式碼範例：aws_kms_encrypted_item.py

步驟 1：建立資料表

首先建立具有表格名稱的標準 DynamoDB 表格資源執行個體。

```
table_name='test-table'
table = boto3.resource('dynamodb').Table(table_name)
```

步驟 2：建立密碼編譯資料提供者

建立您所選密碼編譯資料提供者 (CMP) 的執行個體。

本範例使用直接 KMS 提供者，但您可以使用任何相容的 CMP。若要建立直接 KMS 提供者，請指定 AWS KMS key. 此範例使用的 Amazon 資源名稱 (ARN) AWS KMS key，但您可以使用任何有效的金鑰識別碼。

```
kms_key_id='arn:aws:kms:us-
west-2:111122223333:key/1234abcd-12ab-34cd-56ef-1234567890ab'
```

```
kms_cmp = AwsKmsCryptographicMaterialsProvider(key_id=kms_key_id)
```

第 3 步：使用 TableInfo 輔助類

若要從 DynamoDB 取得資料表的相關資訊，請建立TableInfo協助程式類別的執行個體。當您直接使用項目加密程式時，您需要建立 TableInfo 執行個體及呼叫其方法。用戶端協助程式類別會為您執行此作業。

的refresh_indexed_attributes方法TableInfo使用 DescribeTableDynamoDB 作業取得有關表格的即時、準確資訊。這包括其主要索引鍵及其本機和全域輔助索引。呼叫端必須具備呼叫 DescribeTable 的許可。

```
table_info = TableInfo(name=table_name)
table_info.refresh_indexed_attributes(table.meta.client)
```

步驟 4：建立動 DynamoDB 加密內容

DynamoDB 加密內容包含資料表結構及其加密和簽署方式的相關資訊。此範例會明確建立 DynamoDB 加密內容，因為它會與項目加密程式互動。用戶端協助程式類別會為您建立 DynamoDB 加密內容。

若要取得分割索引鍵和排序索引鍵，您可以使用 TableInfoHelper 類別的屬性。

```
index_key = {
    'partition_attribute': 'value1',
    'sort_attribute': 55
}

encryption_context = EncryptionContext(
    table_name=table_name,
    partition_key_name=table_info.primary_index.partition,
    sort_key_name=table_info.primary_index.sort,
    attributes=dict_to_ddb(index_key)
)
```

步驟 5：建立屬性動作物件

屬性動作會告知項目加密程式要對項目的每個屬性執行什麼動作。這個範例中的 AttributeActions 物件會加密並簽署所有項目，但主要索引鍵屬性 (簽署但不加密) 和 test 屬性 (予以忽略) 除外。

當您直接與項目加密程式互動且預設動作為 ENCRYPT_AND_SIGN 時，您必須為主要索引鍵指定替代動作。您可以使用 set_index_keys 方法，該方法針對主要索引鍵使用 SIGN_ONLY，或使用 DO_NOTHING (如果這是預設動作)。

若要指定主索引鍵，此範例會使用TableInfo物件中的索引鍵，該索引鍵會透過呼叫 DynamoDB 填入。這項技巧比硬編碼主要索引鍵名稱還要安全。

```
actions = AttributeActions(
    default_action=CryptoAction.ENCRYPT_AND_SIGN,
    attribute_actions={'test': CryptoAction.DO_NOTHING}
)
actions.set_index_keys(*table_info.protected_index_keys())
```

步驟 6：建立項目的組態

若要設定 DynamoDB 加密用戶端，請使用您剛才在表格項目的CryptoConfig組態中建立的物件。用戶端輔助程式類別會 CryptoConfig 為您建立。

```
crypto_config = CryptoConfig(
    materials_provider=kms_cmp,
    encryption_context=encryption_context,
    attribute_actions=actions
)
```

步驟 7：將項目加密

此步驟會加密並簽署項目，但不會將其放入 DynamoDB 表格中。

當您使用用戶端協助程式類別時，您的項目會以透明方式加密和簽署，然後在您呼叫協助程式類別的put_item方法時新增至 DynamoDB 表格。當您直接使用項目加密程式時，加密和放置動作都是獨立的。

首先，建立純文字項目。

```
plaintext_item = {
    'partition_attribute': 'value1',
    'sort_key': 55,
    'example': 'data',
    'numbers': 99,
    'binary': Binary(b'\x00\x01\x02'),
```

```
    'test': 'test-value'
}
```

然後，將它加密並簽署。encrypt_python_item 方法需要 CryptoConfig 組態物件。

```
encrypted_item = encrypt_python_item(plaintext_item, crypto_config)
```

步驟 8：在資料表中放入此項目

此步驟會將加密和已簽署的項目置於 DynamoDB 表格中。

```
table.put_item(Item=encrypted_item)
```

若要檢視已加密的項目，請在原始 get_item 物件 (而非 table 物件) 上呼叫 encrypted_table 方法。不需進行驗證或解密，即可從 DynamoDB 資料表取得項目。

```
encrypted_item = table.get_item(Key=partition_key)['Item']
```

下圖顯示一部分已加密並簽署資料表項目的範例。

已加密的屬性值為二進位資料。主要索引鍵屬性 (partition_attribute 和 sort_attribute) 及 test 屬性的名稱和值都保持純文字形式。輸出也會顯示包含簽章 (*amzn-ddb-map-sig*) 的屬性和資料描述屬性 (*amzn-ddb-map-desc*)。

```
{
    '*amzn-ddb-map-desc*': Binary(b'\x00\x00\x00\x00\x00\x00\x10amzn-ddb-env-alg\
\x00\x00\x00\xe0AQEBAHhA84wnXjEJdBbBBylRUFcZZK2j7xwh6UyLoL28nQ
+0FAAAAH4wfAYJKoZIhvcNAQcGoG8wbQIBADBoBgkqhkiG9w0BBwEwHgYJYIZIAWUDBAEuMBEEDPeFBydmoJD
izYl0R0C4M7wAK6E1/N/bgTmHI=\x00\x00\x00\x17amzn-ddb-map-signingAlg\x00\x00\x00\nHmacS
\x00\x00\x00\x11/CBC/PKCS5Padding\x00\x00\x00\x10amzn-ddb-sig-alg\x00\x00\x00\x0eHmac
\x00\x00\x00\x0faws-kms-ec-attr\x00\x00\x00\x06*keys*'),
    '*amzn-ddb-map-sig*': Binary(b"\xd3\xc6\xc7\n\xb7#\x13\xd1Y\xea\xe4.|^\xbd\xdf\xe
    'binary': Binary(b'!"\xc5\x92\xd7\x13\x1d\xe8Bs\x9b\x7f\xa8\x8e\x9c\xcf\x10\x1e\x
    'example': Binary(b"'b\x933\x9a+s\xf1\xd6a\xc5\xd5\x1aZ\xed\xd6\xce\xe9X\xf0T\xcb
    'numbers': Binary(b'\xd5\xa0\\d\xcc\x85\xf5\x1e\xb9-f!\xb9\xb8\x8a\x1aT\xbaq\xf7\
    'partition_attribute': 'value1',
    'sort_attribute': 55,
    'test': 'test-value'
}
```

變更您的資料模型

> **Note**
>
> 我們的客戶端加密庫被重命名為 AWS 數據庫加密 SDK。下列主題提供有關版本 1 的資訊。X-2. 適用於 Java 和版本 1 的 x 個加密用戶端。 X — 3. 適用於 Python 的 x 個 DynamoDB 密用戶端。如需詳細資訊，請參閱適用於 DynamoDB 版本支援的資AWS 料庫加密 SDK。

每次加密或解密項目時，都需要提供屬性動作，以告知 DynamoDB 加密用戶端要加密和簽署哪些屬性、要簽署 (但不加密) 的屬性，以及要忽略哪些屬性。屬性動作不會儲存在加密項目中，且 DynamoDB 加密用戶端不會自動更新您的屬性動作。

> **Important**
>
> DynamoDB 加密用戶端不支援對現有未加密的 DynamoDB 表格資料進行加密。

每當您變更資料模型時，也就是，當您新增或移除資料表項目中的屬性時，就可能發生錯誤。如果您指定的屬性動作並未考量項目中的所有屬性，此項目便無法如您預期的方式進行加密和簽署。更重要的是，如果您在加密項目時提供的屬性動作與您在解密項目時提供的屬性動作不同，則簽章驗證可能會失敗。

例如，如果用來加密項目的屬性動作告知要簽署 `test` 屬性，則項目中的簽章會包含 `test` 屬性。但是如果用來解密項目的屬性動作並未考量 `test` 屬性，則驗證會因為用戶端嘗試驗證不包含 `test` 屬性的簽章而失敗。

當多個應用程式讀取和寫入相同的 DynamoDB 項目時，這是一個特殊的問題，因為 DynamoDB 加密用戶端必須為所有應用程式中的項目計算相同的簽章。對於任何分散式應用程式來說，這也是一個問題，因為屬性動作的變更必須傳播到所有主機。即使您的 DynamoDB 表是由一個主機在一個程序中存取，建立最佳實務程序也有助於在專案變得更複雜時發生錯誤。

若要避免使您無法讀取資料表項目的簽章驗證錯誤，請使用下列指導。

- 新增屬性 — 如果新屬性變更了您的屬性動作，請先完整部署屬性動作變更，然後再將新屬性納入項目中。
- 移除屬性 — 如果您停止在項目中使用屬性，請勿變更屬性動作。
- 變更動作 — 使用屬性動作組態加密表格項目之後，如果不重新加密表格中的每個項目，就無法安全地變更現有屬性的預設動作或動作。

簽章驗證錯誤可能非常難以解決，因此最好的方法是避免這些錯誤的發生。

主題

新增屬性

當您將新屬性新增至資料表項目時，您可能需要變更屬性動作。若要避免簽章驗證錯誤，我們建議您在兩階段的程序中實作此變更。在啟動第二個階段之前，請確認第一個階段已完成。

1. 在所有讀取或寫入資料表的應用程式中變更屬性動作。部署這些變更，並確認更新已傳播到所有目的地主機。
2. 將值寫入資料表項目中的新屬性。

這個兩階段的方法可確保所有應用程式和主機都具有相同的屬性動作，並在遇到新屬性之前計算相同的簽章。即使屬性的動作是不執行任何動作 (不加密或簽署)，這個方法仍很重要，因為某些加密器的預設值是加密和簽署。

下列範例顯示此程序中第一個階段的程式碼。這些範例會新增項目屬性 `link`，此屬性會存放另一個資料表項目的連結。此連結必須維持純文字的形式，因此此範例會為其指派僅簽署動作。完全部署此變更，然後確認所有應用程式和主機都有新的屬性動作之後，您就可以開始在資料表項目中使用該 `link` 屬性。

Java DynamoDB Mapper

使用 DynamoDB Mapper 和 AttributeEncryptor 時，主要索引鍵 (簽署但不加密) 以外，所有屬性都預設會進行加密和簽署。若要指定僅簽署動作，請使用 @DoNotEncrypt 註釋。

此範例會針對新 `link` 屬性使用 @DoNotEncrypt 註釋。

```
@DynamoDBTable(tableName = "ExampleTable")
public static final class DataPoJo {
  private String partitionAttribute;
  private int sortAttribute;
  private String link;

  @DynamoDBHashKey(attributeName = "partition_attribute")
  public String getPartitionAttribute() {
```

```
    return partitionAttribute;
  }

  public void setPartitionAttribute(String partitionAttribute) {
    this.partitionAttribute = partitionAttribute;
  }

  @DynamoDBRangeKey(attributeName = "sort_attribute")
  public int getSortAttribute() {
    return sortAttribute;
  }

  public void setSortAttribute(int sortAttribute) {
    this.sortAttribute = sortAttribute;
  }

  @DynamoDBAttribute(attributeName = "link")
  @DoNotEncrypt
  public String getLink() {
    return link;
  }

  public void setLink(String link) {
    this.link = link;
  }

  @Override
  public String toString() {
    return "DataPoJo [partitionAttribute=" + partitionAttribute + ",
        sortAttribute=" + sortAttribute + ",
        link=" + link + "]";
  }
}
```

Java DynamoDB encryptor

在較低層級 DynamoDB 加密器中，您必須為每個屬性設定動作。此範例使用 switch 陳述式，其中預設值為 `encryptAndSign`，而且會針對分割索引鍵、排序索引鍵和新 `link` 屬性指定例外狀況。在此範例中，如果在使用連結屬性程式碼之前未將其完全部署，則某些應用程式會加密並簽署此連結屬性，但其他應用程式則僅簽署此連結屬性。

```
for (final String attributeName : record.keySet()) {
    switch (attributeName) {
```

```
        case partitionKeyName:
            // fall through to the next case
        case sortKeyName:
            // partition and sort keys must be signed, but not encrypted
            actions.put(attributeName, signOnly);
            break;
        case "link":
            // only signed
            actions.put(attributeName, signOnly);
            break;
        default:
            // Encrypt and sign all other attributes
            actions.put(attributeName, encryptAndSign);
            break;
    }
}
```

Python

在適用於 Python 的 DynamoDB 加密用戶端中，您可以為所有屬性指定預設動作，然後指定例外狀況。

若您使用的是 Python 用戶端協助程式類別，您不需要指定主索引鍵屬性的屬性動作。用戶端協助程式類別可防止您將主索引鍵加密。但是，如果您不使用用戶端協助程式類別，則必須在您的分割區索引鍵和排序索引鍵上設定 SIGN_ONLY 動作。如果您不小心加密了分割區或排序索引鍵，您將無法在沒有進行完整資料表掃描的情況下復原資料。

此範例會指定新 `link` 屬性 (取得 `SIGN_ONLY` 動作) 的例外狀況。

```
actions = AttributeActions(
    default_action=CryptoAction.ENCRYPT_AND_SIGN,
    attribute_actions={
      'example': CryptoAction.DO_NOTHING,
      'link': CryptoAction.SIGN_ONLY
    }
)
```

移除屬性

如果已使用 DynamoDB 加密用戶端加密的項目中不再需要屬性，您可以停止使用該屬性。但是，請勿刪除或變更該屬性的動作。如果您這樣做，然後遇到具有該屬性的項目，則針對該項目計算的簽章將不符合原始簽章，且簽章驗證將會失敗。

雖然您可能想要從程式碼中移除屬性的所有追蹤，但請新增註解，指出該項目已不再使用，而不是將其刪除。即使您執行完整資料表掃描以刪除該屬性的所有執行個體，具有該屬性的加密項目可能會被快取或在組態中的某處處理中。

疑難排解 DynamoDB 加密用戶端應用程式中的問題

> **Note**
>
> 我們的客戶端加密庫被重命名為AWS數據庫加密 SDK。下列主題提供有關版本 1 的資訊。X-2. Java 和版本 1 的動態驗證加密用戶端的 x。 X — 3. 適用於 Python 的 x 個加密用戶端。如需詳細資訊，請參閱適用於 DynamoDB 版本支援的資AWS料庫加密 SDK。

本節說明使用 DynamoDB 加密用戶端時可能遇到的問題，並提供解決這些問題的建議。

若要提供有關 DynamoDB 加密用戶端的意見反應，請在aws-dynamodb-encryption-java或aws-dynamodb-encryption-pythonGitHub存放庫中提出問題。

若要提供有關此文件的意見回饋，請使用任何頁面上的意見回饋連結。您也可以針對本文件的開放原始碼儲存庫提出問題或貢獻GitHub。aws-dynamodb-encryption-docs

主題

- 存取遭拒
- 簽章驗證失敗
- 舊版全域資料表的問題
- 最新提供商的表現不佳

存取遭拒

問題：您的應用程式遭拒，無法存取其所需的資源。

建議：了解必要的許可，並將這些許可新增至您的應用程式執行所在的安全性細節。

詳細資訊

若要執行使用 DynamoDB 加密用戶端程式庫的應用程式，呼叫者必須具有使用其元件的權限。否則他們會遭拒，無法存取所需的元素。

- DynamoDB 加密用戶端不需要 Amazon 網路服務 (AWS) 帳戶，也不需要任何AWS服務。但是，如果您的應用程序使用 AWS，則需要有權使用該帳戶的AWS 帳戶和用戶。
- 加密用戶端不需要使用亞馬遜動態 B。不過，如果使用用戶端的應用程式建立 DynamoDB 表、將項目放入資料表或從表格中取得項目，則呼叫者必須具有在您的. AWS 帳戶 如需詳細資訊，請參閱 Amazon DynamoDB 開發人員指南中的存取控制主題。
- 如果您的應用程式在 Python 的 DynamoDB 加密用戶端中使用用戶端協助程式類別，則呼叫者必須具有呼叫 DynamoDB DescribeTable作業的權限。
- DynamoDB 加密用戶端不需要 AWS Key Management Service () AWS KMS。但是，如果您的應用程式使用 Direct KMS 材料提供者，或者它使用的提供者存放區的最新提供者AWS KMS，則呼叫者必須具有使用AWS KMSGenerateDataKey和解密操作的權限。

簽章驗證失敗

問題：項目因為簽章驗證失敗而無法解密。此項目也無法如您所願進行加密和簽署。

建議：確保您提供的屬性動作會考量項目中的所有屬性。將項目解密時，請務必提供與用來解密項目之動作相符的屬性動作。

詳細資訊

您提供的屬性動作會告訴 DynamoDB 加密用戶端要加密和簽署哪些屬性、要簽署 (但不加密) 哪些屬性，以及要忽略哪些屬性。

如果您指定的屬性動作並未考量項目中的所有屬性，此項目便無法如您預期的方式進行加密和簽署。如果您在加密項目時提供的屬性動作與您在解密項目時提供的屬性動作不同，則簽章驗證可能會失敗。這是一個特殊問題，出現於新屬性動作尚未傳播到所有主機的分散式應用程式中。

簽章驗證錯誤很難解決。如需協助防範其發生，請在變更資料模型時採取額外的預防措施。如需詳細資訊，請參閱 變更您的資料模型。

舊版全域資料表的問題

問題：舊版 Amazon DynamoDB 全域資料表中的項目無法解密，因為簽章驗證失敗。

建議：設定屬性動作，使保留的複寫欄位不會加密或簽署。

詳細資訊

您可以將 DynamoDB 加密用戶端與 Dynam oDB 全域資料表搭配使用。建議您搭配多區域 KMS 金鑰使用全域資料表，並將 KMS 金鑰複寫到複寫全域資料表的所有AWS 區域位置。

從 2019.11.21 版全域表開始，您可以將全域表與 DynamoDB 加密用戶端搭配使用，而無需任何特殊設定。但是，如果您使用全域資料表版本 2017.11.29，則必須確保保留的複寫欄位未加密或簽署。

如果您使用的是全域表格版本 2017.11.29，您必須DO_NOTHING在 Java 或 @DoNotTouch Python 中將下列屬性的屬性動作設定為。

- `aws:rep:deleting`
- `aws:rep:updatetime`
- `aws:rep:updateregion`

如果您正在使用任何其他版本的全域表格，則不需要採取任何動作。

最新提供商的表現不佳

問題：您的應用程式的回應速度較低，尤其是在更新至較新版本的 DynamoDB 加密用戶端之後。

建議：調整time-to-live值和快取大小。

詳細資訊

最新的提供者旨在透過允許有限的加密材料重複使用，改善使用 DynamoDB 加密用戶端的應用程式效能。當您為應用程式設定最新的 Provider 時，您必須在改善的效能與快取和重複使用所產生的安全性考量之間取得平衡。

在較新版本的 DynamoDB 加密用戶端中，time-to-live(TTL) 值決定可以使用快取的加密材料提供者 (CMP) 的時間長度。TTL 也會決定「最新的提供者」檢查新版 CMP 的頻率。

如果您的 TTL 過長，您的應用程式可能會違反您的商業規則或安全性標準。如果您的 TTL 太短，頻繁撥打電話給提供者商店可能會導致您的供應商商店限制來自您的應用程式和其他共用您服務帳戶的應用程式的要求。若要解決此問題，請將 TTL 和快取大小調整為符合延遲和可用性目標的值，並符合您的安全性標準。如需詳細資訊，請參閱設置一個time-to-live值。

亞馬遜加密用戶端重新命名

我們的用戶端加密程式庫已重新命名為AWS資料庫加密 SDK。本開發人員指南仍提供 DynamoDB 加密用戶端的相關資訊。

2023 年 6 月 9 日，我們的用戶端加密程式庫重新命名為AWS資料庫加密 SDK。AWS資料庫加密開發套件與亞馬遜動態 B 相容。它可以解密和讀取由舊版 DynamoDB 加密用戶端加密的項目。如需舊版 DynamoDB 加密用戶端版本的詳細資訊，請參閱。AWS 適用於 DynamoDB 版本支援的資料庫加密 SDK

資AWS料庫加密 SDK 提供版本 3。 適用於 DynamoDB 的 Java 用戶端加密程式庫的 x 個，這是 Java 的 DynamoDB 加密用戶端的主要重新寫入。它包含許多更新，例如新的結構化資料格式、改進的多租戶支援、順暢的結構描述變更，以及可搜尋的加密支援。

若要深入了解資AWS料庫加密 SDK 所引入的新功能，請參閱下列主題。

可搜索加密

您可以設計可以在不解密整個數據庫的情況下搜索加密記錄的數據庫。根據您的威脅模型和查詢需求，您可以使用可搜尋的加密來對加密記錄執行完全比對搜尋或更自訂的複雜查詢。

鑰匙圈

資AWS料庫加密 SDK 使用金鑰環來執行信封加密。金鑰圈會產生、加密和解密可保護您記錄的資料金鑰。資料AWS庫加密 SDK 支援使用對稱加密或非對稱 RSA 的金AWS KMS鑰圈AWS KMS keys來保護您的資料金鑰，以及AWS KMS階層式金鑰圈，可讓您在對稱加密 KMS 金鑰下保護加密資料，而無需AWS KMS每次加密或解密記錄時都呼叫。您也可以使用 Raw AES 鑰匙圈和 RSA 原始鑰匙圈來指定自己的金鑰材料。

無縫模式更改

當您設定AWS資料庫加密 SDK 時，您會提供密碼編譯動作，告訴用戶端要加密和簽署哪些欄位、要簽署 (但不加密) 的欄位，以及要忽略哪些欄位。使用資料AWS庫加密 SDK 保護記錄之後，您仍然可以對資料模型進行變更。您可以在單一部署中更新加密動作，例如新增或移除加密欄位。

設定用戶端加密的現有 DynamoDB 表格

舊版 DynamoDB 加密用戶端的設計是要在新的未填入資料表中實作。使用適用於 DynamoDB 的AWS資料庫加密開發套件，您可以將現有的 Amazon DynamoDB 表格移轉到第 3 版。 Java 用戶端加密程式庫的 x 個。

參考

我們的用戶端加密程式庫已重新命名為AWS資料庫加密 SDK。本開發人員指南仍提供 DynamoDB 加密用戶端的相關資訊。

下列主題提供AWS資料庫加密 SDK 的技術詳細資料。

材料描述格式

材料描述作為加密記錄的標頭。當您使用 AWS Database Encryption SDK 加密和簽署欄位時，加密程式會在組合加密資料時記錄材料描述，並將材料描述儲存在加密程式新增至記錄的新欄位 (aws_dbe_head) 中。材料描述是可攜式格式化的資料結構，其中包含加密的資料金鑰，以及如何加密和簽署記錄的相關資訊。下表描述形成材料描述的值。位元組依顯示順序附加。

值	長度 (位元組)
版本	1
已啟用簽章	1
記錄 ID	32
加密圖例	變數
加密內容長度	2
加密內容	變數
加密資料金鑰計數	1
加密的資料金鑰	變數
記錄承諾	1

版本

此aws_dbe_head欄位格式的版本。

已啟用簽章

編碼是否為此記錄啟用簽名。

位元組值	意義
0x01	已啟用簽章 (預設值)
0x00	已停用簽章

記錄 ID

隨機產生的 256 位元值，可識別記錄。記錄識別碼：

- 唯一識別加密記錄。
- 將材料描述繫結至加密記錄。

加密圖例

已加密驗證欄位的序列化描述。加密圖例用於確定解密方法應嘗試解密哪些字段。

位元組值	意義
0x65	ENCRYPT_AND_SIGN
0x73	SIGN_ONLY

加密圖例序列化如下：

1. 按字母順序表示其規範路徑的字節序列。
2. 對於每個欄位，依序附加上述指定的其中一個位元組值，以指示該欄位是否應加密。

加密內容長度

加密內容的長度。它會以 2 位元組數值表示，並解譯為 16 位元的無符號整數。最大長度為 65,535 個字節。

加密內容

一組名稱-值對，其中包含任意，非秘密的附加驗證數據。

啟用數位簽章時，加密內容會包含索引鍵值配對{"aws-crypto-footer-ecdsa-key": Qtxt}。 Qtxt表示根據 SEC 1 版本 2.0 Q 壓縮，然後以 base64 編碼壓縮的橢圓曲線點。

加密資料金鑰計數

加密資料金鑰的數量。它是解譯為 8 位元組不帶正負號的整數的 1 位元組值，用於指定加密資料金鑰的數目。每筆記錄中的加密資料金鑰數目上限為 255。

加密的資料金鑰

加密資料金鑰的序列。序列長度取決於加密資料金鑰的數量與每個加密資料金鑰的長度。序列會包含至少一個加密資料金鑰。

下表將說明每個加密資料金鑰的組成欄位。位元組依顯示順序附加。

加密資料金鑰結構

欄位	長度 (位元組)
金鑰提供者 ID 長度	2
金鑰提供者 ID	變數. 等於前 2 個位元組中指定的值 (金鑰提供者 ID 長度)。
金鑰提供者資訊長度	2
金鑰提供者資訊	變數. 等於前 2 個位元組中指定的值 (金鑰提供者資訊長度)。
加密資料金鑰長度	2
加密資料金鑰	變數. 等於前 2 個位元組中指定的值 (加密資料金鑰長度)。

金鑰提供者 ID 長度

金鑰提供者識別碼的長度。它會以 2 元組數值表示，並解譯為 16 位元的無符號整數，指出包含金鑰提供者 ID 的位元組數量。

金鑰提供者 ID

金鑰提供者識別碼。它會用來指出加密資料金鑰的提供者，以而且可供擴充。

金鑰提供者資訊長度

金鑰提供者資訊的長度。它會以 2 元組數值表示，並解譯為 16 位元的無符號整數，指出包含金鑰提供者資訊的位元組數量。

金鑰提供者資訊

金鑰提供者資訊。它會取決於金鑰提供者。

當您使用AWS KMS金鑰圈時，此值會包含. AWS KMS key

加密資料金鑰長度

加密資料金鑰的長度。它會以 2 元組數值表示，並解譯為 16 位元的無符號整數，指出包含加密資料金鑰的位元組數量。

加密資料金鑰

加密資料金鑰。它是由金鑰提供者加密的資料金鑰。

記錄承諾

不同的 256 位元雜湊型訊息驗證碼 (HMAC) 雜湊，透過使用承諾金鑰計算所有先前的材料描述位元組。

AWS KMS分層鑰匙圈技術細節

AWS KMS分層密鑰環使用 unqiue 數據密鑰來加密每個字段，並使用從活動分支密鑰派生的唯一包裝密鑰對每個數據密鑰進行加密。它在計數器模式下使用密鑰派生，並帶有 HMAC SHA-256 的偽隨機函數，以導出具有以下輸入的 32 字節包裝密鑰。

- 一個 16 字節的隨機鹽
- 作用中的分支索引鍵
- 金鑰提供者識別碼 "aws-kms-hierarchy」的 UTF-8 編碼值

階層式金鑰環使用衍生的包裝金鑰，使用具有 16 位元組驗證標籤和下列輸入的 AES-GCM-256 來加密純文字資料金鑰的副本。

- 派生的包裝密鑰被用作 AES-GCM 密鑰
- 資料金鑰會用作 AES-GCM 訊息
- 一個 12 字節的隨機初始化向量（IV）被用作 AES-GCM IV

- 包含下列序列化值的其他驗證資料 (AAD)。

值	長度 (位元組)	解釋為
"aws-kms-hierarchy"	17	UTF-8 編碼
分支密鑰標識符	變數	UTF-8 編碼
分支密鑰版	16	UTF-8 編碼
加密內容	變數	UTF-8 編碼的金鑰值配對

AWS 資料庫加密 SDK 開發人員指南的文件歷史記錄

下表說明本文件的重大變更。除了下述各項主要變更外，我們也會經常更新文件以改進說明內容和範例，並且反映您傳送我們的意見回饋。如要接收重大變更的通知，請訂閱 RSS 摘要。

變更	描述	日期
新功能	已新增 AWS KMS ECDH 鑰匙圈和原始 E CDH 鑰匙圈的說明文件。	2024年6月17日
一般可用性 (GA) 版本	介紹對 DynamoDB 的 .NET 用戶端加密程式庫的支援。	2024年1月17日
一般可用性 (GA) 版本	針對第 3 版 GA 發行的更新說明文件。 Java 用戶端加密程式庫 DynamoDB 個。 **Warning** 不再支援開發人員預覽版本期間建立的分支金鑰。	2023 年 7 月 24 日
重新命 DynamoDB 加密用戶端	用戶端加密程式庫已重新命名為 AWS 資料庫加密 SDK。	2023 年 6 月 9 日
預覽版本	增加和更新的文檔版本 3。 DynamoDB 的 Java 用戶端加密程式庫的 x 個，其中包含新的結構化資料格式、改進的多租戶支援、順暢的結構描述變更，以及可搜尋的加密支援。	2023 年 6 月 9 日
文件變更	將 AWS Key Managemen t Service 術語客戶主金鑰	2021 年 8 月 30 日

	(CMK) 取代為AWS KMS key和 KMS 金鑰。	
新功能	增加了對 AWS Key Management Service （AWS KMS）多區域鍵的支持。多區域金鑰是不同的 AWS KMS 金鑰，可 AWS 區域 以互換使用，因為它們具有相同的金鑰 ID 和金鑰材料。	2021 年 6 月 8 日
新範例	新增了使用 DynamoDBM apper 的 Java 範例。	2018 年 9 月 6 日
Python 支持	除 Java 外另新增了對 Python 的支援。	2018 年 5 月 2 日
初始版本	本文件的初始版本。	2018 年 5 月 2 日

本文為英文版的機器翻譯版本，如內容有任何歧義或不一致之處，概以英文版為準。