Amazon DynamoDB Encryption Client

デベロッパーガイド

Amazon DynamoDB Encryption Client: デベロッパーガイド

Table of Contents

Amazon DynamoDB 暗号化クライアントとは

Amazon DynamoDB 暗号化クライアントは、クライアント側の暗号化をAmazon DynamoDBの設計。DynamoDB 暗号化クライアントは、新しいデータベースで実装されるように設計されています。伝送中および保管時の機密データを暗号化することで、AWS などのサードパーティーがお客様のプレーンテキストデータを使用することはできません。DynamoDB 暗号化クライアントは、Apache 2.0 ライセンスに基づいて、無償で提供されています。

> **Important**
>
> DynamoDB 暗号化クライアントでは、暗号化されていない DynamoDB テーブルデータの暗号化をサポートしていません。

このデベロッパーガイドでは、DynamoDB 暗号化クライアントの概念的な概要を説明します。例えば、アーキテクチャの説明 (p. 5)、DynamoDB テーブルデータを保護する方法 (p. 3)の詳細、DynamoDB サーバー側の暗号化 (p. 7)との相違点、アプリケーションの重要なコンポーネントの選択 (p. 12)に関するガイダンス、開始時に役立つ各プログラミング言語 (p. 32)での例などが含まれています。

DynamoDB 暗号化クライアントには、以下の利点があります。

DynamoDB アプリケーション用の特別設計

DynamoDB 暗号化クライアントは、暗号化の専門家でなくても使用できます。この実装には、既存の DynamoDB アプリケーションで動作するように設計されたヘルパーメソッドが含まれます。

必要なコンポーネントを作成して設定すると、DynamoDB 暗号化クライアントは、テーブルへの追加時に項目を透過的に暗号化して署名し、取得時に検証後、復号します。

DynamoDB 暗号化クライアントでは、DynamoDB データ型そして、以下を含むほとんどの Amazon DynamoDB 機能グローバルテーブル。ただし、古いバージョンのグローバルテーブルを使用している場合は、設定の変更が必要になることがあります。詳細については、「古いバージョンのグローバルテーブルに関する問題 (p. 55)」を参照してください

セキュアな暗号化と署名を含む

DynamoDB 暗号化クライアントには、一意の暗号化キーを使用して、各テーブル項目の属性値を暗号化するセキュア実装を含み、属性の追加や削除、または暗号化された値のスワップなどの不正な変更を防ぐために項目に署名します。

ソースの暗号化マテリアルを使用する

DynamoDB 暗号化クライアントは、カスタム実装や、AWS Key Management Service (AWS KMS) または AWS CloudHSM などの暗号化サービスを含む、任意のソースの暗号化キーで使用できます。DynamoDB 暗号化クライアントには、AWS アカウント または任意の AWSサービスは必要ありません。

プログラミング言語実装はすべて相互運用可能

DynamoDB 暗号化クライアントライブラリは、のオープンソースプロジェクトで開発されています。GitHub。これらのライブラリは、現在 Java および Python で使用できます。DynamoDB 暗号化クラ

イアントのサポートされているプログラミング言語実装はすべて相互運用可能です。たとえば、Java クライアントでデータを暗号化 (および署名) し、Python クライアントで復号することができます。

ただし、DynamoDB 暗号化クライアントは、AWS Encryption SDK または Amazon S3 暗号化クライアントとは互換性がありません。一方のクライアント側ライブラリで暗号化し、もう一方のライブラリを使用して復号することはできません。

オープンソースリポジトリで開発されています

Amazon DynamoDB Encryption Client は、のオープンソースリポジトリで開発されています。 GitHub。これらのリポジトリを使用して、コードの表示、Issue の読み取りと送信、および言語実装に固有の情報を見つけることができます。

- Java DynamoDB 暗号化クライアント —aws-dynamodb-encryption-java
- Python DynamoDB 暗号化クライアント —aws-dynamodb-encryption-python

Support とメンテナンス

DynamoDB 暗号化クライアントでは、メンテナンスポリシーそのAWSSDKとToolsは、バージョン管理とライフサイクルフェーズを含めて使用します。ベストプラクティスとして、DynamoDB 暗号化クライアントの使用可能なバージョンを使用することをお勧めします。

DynamoDB 暗号化クライアントのプログラミング言語実装は、それぞれ別のオープンソースで開発されています。 GitHub repository. 各バージョンのライフサイクルとサポートフェーズは、リポジトリによって異なる可能性があります。たとえば、DynamoDB 暗号化クライアントの特定のバージョンが 1 つのプログラミング言語で一般公開 (フルサポート) フェーズにある場合がありますが、 end-of-support別のプログラミング言語でのフェーズ。可能な限り完全にサポートされているバージョンを使用し、サポートされなくなったバージョンは避けることをお勧めします。

使用しているプログラミング言語の DynamoDB 暗号化クライアントバージョンの詳細については、`SUPPORT_POLICY.rst`ファイルは各 DynamoDB 暗号化クライアントリポジトリにあります。

- Java DynamoDB 暗号化クライアント —support_policy.rst
- Python DynamoDB 暗号化クライアント —support_policy.rst

詳細については、「」を参照してください。AWSSDK とツールのメンテナンスポリシーのAWSSDK とツールのリファレンスガイド

フィードバックを送る

当社では、お客様からのフィードバックをお待ちしております。質問、コメント、ご報告いただく問題がある場合は、以下のリソースをご利用ください。

- DynamoDB 暗号化クライアントで潜在的なセキュリティの脆弱性を発見した場合は、AWS セキュリティまでご報告ください。公開されている情報はご報告いただく必要はありません GitHub 問題。
- DynamoDB 暗号化クライアントに関するフィードバックについては、aws-dynamodb-encryption-javaまたはaws-dynamodb-encryption-python GitHub repository.
- このドキュメントに関するフィードバックを提供するには、任意のページのフィードバックリンクを使用します。問題を提起したり、貢献したりすることもできますaws-dynamodb-encryption-docs、このドキュメントのオープンソースリポジトリ GitHub。

どのフィールドが暗号化および署名されますか?

DynamoDBでは、テーブルは項目の集合です。各項目は、属性の集合です。各属性には名前と値があります。

DynamoDB 暗号化クライアントは、属性の値を暗号化します。次に、属性に対する署名を計算します。暗号化される属性値、および署名に含めるか指定できます。ただし、DynamoDB 暗号化クライアントは、新しい未入力データベースに実装されるように設計されています。DynamoDB にデータを送信する前に、暗号化機能を DynamoDB アプリケーションに追加する必要があります。

暗号化は、属性値の機密保持を保護します。署名は、署名されたすべての属性とその相互の関係を保全し、認証を提供します。これにより、属性の追加や削除、暗号化された値の別の値への置換など、項目全体への不正な変更を検出することができます。

暗号化された項目では、テーブル名、すべての属性名、暗号化されていない属性値、およびプライマリキー (パーティションキーとソートキー) 属性の名前と値など、一部のデータはプレーンテキストで残ります。これらのフィールドに機密データを保存しないでください。

トピック

暗号化の属性値

DynamoDB 暗号化クライアントは、指定した属性の値 (名前ではない) を暗号化します。どの属性値が暗号化されているかを確認するには、属性アクション (p. 9)を使用します。

たとえば、この項目には `example` および `test` 属性が含まれます。

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

`example` 属性を暗号化し、`test` 属性を暗号化しない場合、結果は次のようになります。暗号化された `example` 属性値は、文字列ではなくバイナリデータです。

```
'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'
...
```

DynamoDB ではテーブル内の項目を検索するためにそれらを使用するため、各項目のプライマリキー属性 (パーティションキーとソートキー) はプレーンテキストのままにする必要があります。署名は必要ですが、暗号化の必要はありません。

> Warning
>
> プライマリキー属性を暗号化しないでください。でテーブル全体のスキャンを実行せずに項目を見つけDynamoDB ように、プレーンテキストの状態を維持する必要があります。

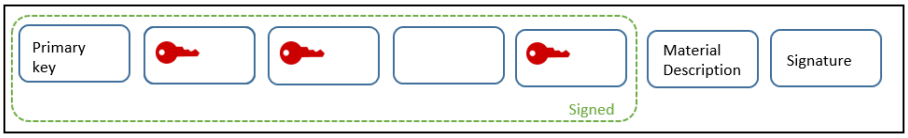
各プログラミング言語のヘルパーは、プライマリキーの属性を識別し、その値が署名されているが暗号化されていないことを確認します。また、プライマリキーを特定してそれを暗号化しようとすると、クライアントは例外をスローします。特別なユースケースのプライマリキーを暗号化する必要がある場合は、低

レベルのレベルを使用します。項目エンクリプタ (p. 9)直接ですが、DynamoDB はテーブル全体のスキャンを実行しないとが項目を見つけることができません。

DynamoDB 暗号化クライアントは、マテリアル記述属性 (p. 10)。これは、DynamoDB 暗号化クライアントが項目を検証および復号するための情報を保存しているためです。

項目の署名

指定された属性値を暗号化した後、DynamoDB 暗号化クライアントは、で指定した属性の名前と値に対するデジタル署名を計算します。属性アクション (p. 9)オブジェクト。クライアントは、署名を項目に追加する属性に保存します。



テーブル名を指定すると、それは署名に含まれます。これにより、従業員レコードが `AllEmployees` テーブルから `TrustedEmployees` テーブルに移動するなど、署名された項目が別のテーブルに移動したことを悪意のあるものとして検出することができます。DynamoDB 暗号化クライアントは、テーブル名をDynamoDB 暗号化コンテキスト (p. 10)です。ここで、オプションのフィールドです。

署名には必ずプライマリキーを含めてください。これは、ヘルパーを使用するときのデフォルトの動作です。署名は、プライマリキーと項目内の他の属性との関係をキャプチャし、署名の検証により、関係が変更されていないことが確認されます。

マテリアル記述属性 (p. 10)は暗号化も署名もされていません。

暗号化および署名された項目

DynamoDB 暗号化クライアントがテーブル項目を暗号化して署名すると、その結果は暗号化された属性値を含む標準の DynamoDB テーブル項目になります。

次の図は、暗号化および署名されたテーブル項目の一部の例を示しています。

```
{
    '*amzn-ddb-map-desc*': Binary(b'\x00\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'
}
```

この図は、DynamoDB 暗号化クライアントが暗号化して署名するテーブル項目の次の特性を示しています。

- すべての属性名はプレーンテキストです。
- プライマリキーの属性の値はプレーンテキストです。この例では、パーティションキー名は `partition_attribute`、ソートキー名は `sort_attribute` です。

- 暗号化しないようにクライアントに指示する属性の値は、プレーンテキストのままです。この例では、`test` 属性の値はプレーンテキストです。
- 暗号化された属性の値は、バイナリデータです。
- クライアントは、項目に署名属性 (`*amzn-ddb-map-sig*`) を追加します。その値は項目の署名です。
- クライアントは、項目にマテリアル説明属性 (p. 10) (`*amzn-ddb-map-desc*`) を追加します。その値は、属性がどのように暗号化および署名されたかを示します。クライアントは、この情報を使用して項目の検証と復号を行います。マテリアル記述属性は暗号化も署名もされていません。

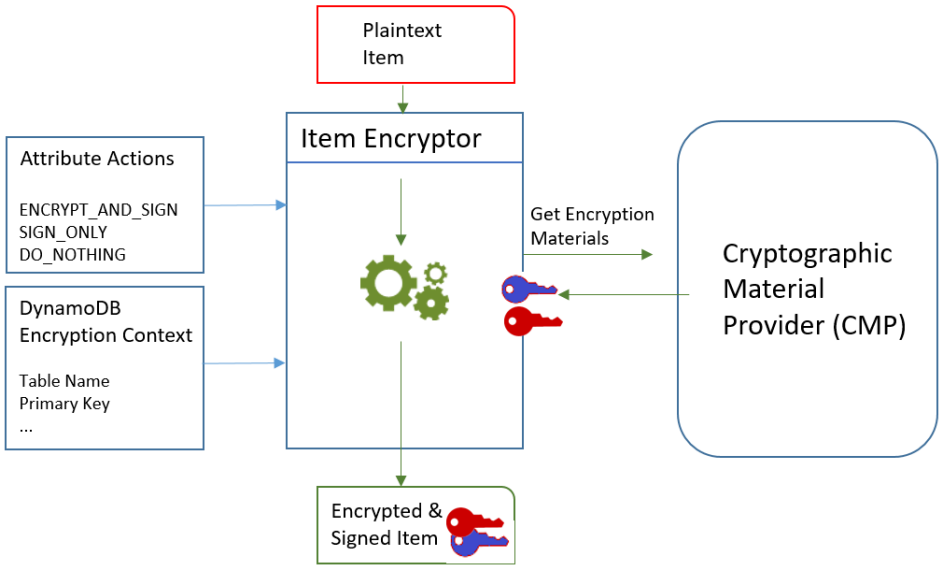
DynamoDB 暗号化クライアントの動作

DynamoDB 暗号化クライアントは、DynamoDB に保存されているデータを保護するように特別に設計されています。ライブラリには、拡張が可能でまた変更なしで使用できる安全な実装が含まれています。また、ほとんどの要素は抽象要素で表されるため、互換性のあるカスタムコンポーネントを作成して使用できます。

テーブル項目の暗号化と署名

DynamoDB 暗号化クライアントの中核となるのは、項目暗号化テーブル項目を暗号化、署名、検証、復号します。テーブル項目に関する情報と、暗号化して署名する項目に関する指示が取り込まれます。選択して設定した暗号化マテリアルプロバイダー (p. 8)から、暗号化マテリアルとその使用方法に関する指示が取得されます。

次の図は、このプロセスの高レベルのビューを示しています。



テーブル項目を暗号化し署名するために、DynamoDB 暗号化クライアントには次のものが必要です。

- テーブルについての情報。からテーブルに関する情報を取得します。DynamoDB 暗号化コンテキスト (p. 10)あなたが供給していること。一部のヘルパーは、DynamoDB から必要な情報を取得し、DynamoDB 暗号化コンテキストを作成します。

  Note

  -DynamoDB 暗号化コンテキストDynamoDB 暗号化クライアントでは、暗号化コンテキストに AWS Key Management Service(AWS KMS) とAWS Encryption SDK。

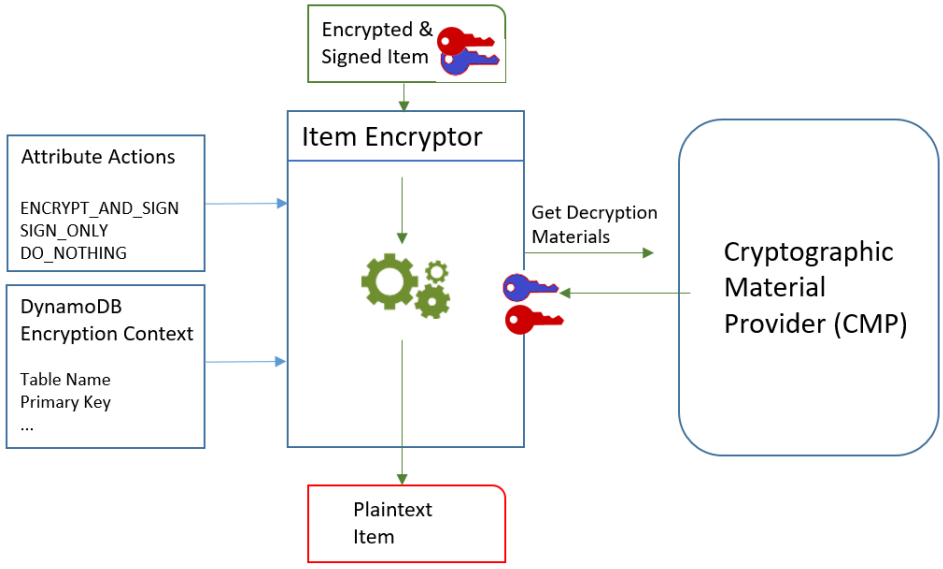
- 暗号化して署名する属性。この情報は、指定した属性アクション (p. 9)から取得されます。
- 暗号化および署名キーを含む、暗号化マテリアル。これらは、お客様が選択して設定する暗号化マテリアルプロバイダー (p. 8) (CMP) から取得されます。
- 項目の暗号化と署名の手順。CMP は、暗号化および署名アルゴリズムを含む、暗号化マテリアルを使用するための指示を実際のマテリアル説明 (p. 10)に追加します。

項目エンクリプタ (p. 9)は、これらの要素のすべてを使用して項目を暗号化して署名します。項目エンクリプタは、暗号化と署名の指示 (実際のマテリアル説明) を含むマテリアル説明属性 (p. 10)と、その署名を含む属性を項目に追加します。項目エンクリプタと直接やり取りすることができます。また、項目エンクリプタとやり取りするヘルパー機能を使用して、安全なデフォルトの動作を実装することもできます。

結果は、暗号化された署名済みデータを含む DynamoDB 項目です。

テーブル項目の検証と復号

これらのコンポーネントは、次の図に示すように、項目を検証および復号するために一緒に機能します。



項目を検証し、復号するために、DynamoDB 暗号化クライアントには、次のように、同じコンポーネント、同じ設定のコンポーネント、または項目を復号するために特に設計されたコンポーネントが必要です。

- テーブルについての情報からのDynamoDB 暗号化コンテキスト (p. 10)。
- 検証および復号する属性。これらは属性アクション (p. 9)から取得されます。
- 選択し、設定した暗号化マテリアルプロバイダー (p. 8) (CMP) からの検証キーおよび復号キーを含む復号マテリアル。

  暗号化された項目には、暗号化に使用された CMP のレコードは含まれません。同じ CMP、同じ設定の CMP、または項目を復号するように設計された CMP 指定する必要があります。
- 暗号化アルゴリズムと署名アルゴリズムを含む、項目の暗号化と項目の署名に関する情報。クライアントは、項目のマテリアル説明属性 (p. 10)からこれらを取得します。

項目エンクリプタ (p. 9)は、これらの要素のすべてを使用して項目の検証と復号を行います。また、マテリアル記述と署名属性も削除されます。結果はプレーンテキストの DynamoDB 項目です。

クライアント側とサーバー側の暗号化

DynamoDB 暗号化クライアントがサポートするクライアント側の暗号化では、テーブルデータを DynamoDB に送信する前に暗号化します。ただし、DynamoDB では、ディスクに保管されているテーブルを透過的に暗号化するサーバー側の保管時の暗号化機能を提供しており、ユーザーがテーブルにアクセスすると復号します。

選択するツールは、データの重要度と、アプリケーションのセキュリティ要件に応じて異なります。DynamoDB 暗号化クライアントと保管時の暗号化の両方を使用できます。暗号化されて署名された項目を DynamoDB に送信しても、保護されている項目は DynamoDB によって認識されません。バイナリ属性値を含む従来のテーブル項目を検出します。

サーバー側の保管時の暗号化

DynamoDB では、保管時の暗号化がサポートされています。これは、テーブルがディスクに保持されるときに DynamoDB がテーブルを透過的に暗号化し、ユーザーがテーブルデータにアクセスするときにテーブルを復号するサーバー側の暗号化機能です。

使用するバージョンAWSSDK では、データは HTTPS 接続で伝送時に暗号化され、DynamoDB エンドポイントで復号された後、DynamoDB に保管される前に再度暗号化されます。

- デフォルトでの暗号化。DynamoDB は、ディスクに書き込まれるときに、すべてのテーブルを透過的に暗号化および復号します。保管時の暗号化を有効または無効にするオプションはありません。
- DynamoDB は暗号化キーを作成および管理します。各テーブルの一意のキーは、AWS KMS key で保護されるため、AWS Key Management Service (AWS KMS) が未暗号化のままになることはありません。デフォルトでは、DynamoDB は DynamoDB サービス アカウントの AWS 所有のキー を使用しますが、一部またはすべてのテーブルを保護するために、自分のアカウントの AWS マネージドキー またはカスタマーマネージドキーを選択することもできます。
- テーブルデータはすべて、ディスク上で暗号化されます。暗号化されたテーブルがディスクに保存されると、DynamoDB は、プライマリキーおよびローカルとグローバルのセカンダリインデックスなど、すべてのテーブルデータを暗号化します。テーブルにソートキーが存在する場合、範囲の境界線を示すソートキーの一部が、プレーンテキスト形式でテーブルメタデータに保存されます。
- テーブルに関連するオブジェクトも暗号化されます。保管時の暗号化は、永続的なメディアに書き込まれるたびに、DynamoDBストリーム、グローバルテーブル、バックアップを保護します。
- アクセスすると、項目は復号されます。テーブルがアクセスされるとき、DynamoDB は、ターゲット項目を含むテーブル部分を復号し、プレーンテキスト形式で項目を返します。

DynamoDB 暗号化クライアント

クライアント側の暗号化では end-to-end ソースから DynamoDB のストレージまで、伝送時および保管時のデータを保護します。プレーンテキストデータが AWS などのサードパーティーに公開されることはありません。ただし、DynamoDB Encryption クライアントは、新しい未入力のデータベースに実装するように設計されています。暗号化機能を DynamoDB に追加する必要があります。

- 転送時と保管時のデータは保護されます。AWS も含め、このようなデータがサードパーティーに公開されることはありません。
- テーブル項目に署名できます。プライマリキー属性やテーブル名など、テーブル項目のすべてまたは一部の署名を計算するように、DynamoDB 暗号化クライアントに指示できます。この署名により、属性の追加や削除、属性値のスワップなど、項目全体への不正な変更を検出することができます。
- 暗号化キーの生成および保護方法を選択できます。キーを生成および保護するには、キーを作成して管理するか、AWS Key Management Service または AWS CloudHSM などの暗号化サービスを使用します。

- データの保護方法の決定は、暗号化マテリアルプロバイダー (p. 12) (CMP) を選択するか、独自のものを記述することで行います。CMP を使用して、一意のキーが生成されるタイミングや、使用する暗号化アルゴリズムおよび署名アルゴリズムなど、使用する暗号化の方法を判断します。
- DynamoDB 暗号化クライアントによってテーブル全体が暗号化されることはありません。暗号化する場合は、テーブルの項目を選択するか、一部またはすべての項目の属性値を選択できます。ただし、DynamoDB 暗号化クライアントによって項目全体が暗号化されることはありません。属性名、プライマリキー (パーティションキーおよびソートキー) 属性の名前または値は暗号化されません。暗号化される項目 (および暗号化されない項目) の詳細については、「どのフィールドが暗号化および署名されますか? (p. 3)」を参照してください。

AWS Encryption SDK

DynamoDB に保存しているデータを暗号化している場合は、DynamoDB 暗号化クライアントをお勧めします。

AWS Encryption SDK は、クライアント側暗号化ライブラリで、汎用データの暗号化および復号に役立ちます。任意のタイプのデータを保護することはできますが、データベースレコードなどの構造化データは操作できません。DynamoDB 暗号化クライアントとは異なり、AWS Encryption SDK では、項目レベルの整合性チェックを行うことはできません。属性を認識するか、プライマリキーの暗号化を回避するロジックはありません。

AWS Encryption SDK を使用してテーブルの要素を暗号化している場合、それに DynamoDB 暗号化クライアントとの互換性はありません。1 つのライブラリで暗号化し、もう 1 つのライブラリを使用して復号することはできません。

Amazon DynamoDB 暗号化クライアントの概念

このトピックでは、Amazon DynamoDB 暗号化クライアントで使用されている概念と用語について説明します。

DynamoDB 暗号化クライアントのコンポーネントがやり取りする方法については、「」DynamoDB 暗号化クライアントの動作 (p. 5)。

トピック

- 暗号化マテリアルプロバイダー (CMP) (p. 8)
- 項目エンクリプタ (p. 9)
- 属性アクション (p. 9)
- マテリアル記述 (p. 10)
- DynamoDB 暗号化コンテキスト (p. 10)
- プロバイダーストア (p. 11)

暗号化マテリアルプロバイダー (CMP)

DynamoDB 暗号化クライアントを実装する場合、最初のタスクの 1 つは次のとおりです。暗号化マテリアルプロバイダーを選択する (p. 12)(CMP) (別名暗号化マテリアルプロバイダー). 残りの実装の多くは、この選択によって決まります。

暗号化マテリアルプロバイダー (CMP) は項目エンクリプタ (p. 9)が、テーブル項目を暗号化し署名するのに使用する暗号化マテリアルを収集、アセンブルし、返します。CMP は、使用する暗号化アルゴリズムと、暗号化キーと署名キーを生成して保護する方法を決定します。

CMP は項目エンクリプタとやり取りします。項目エンクリプタは、暗号化または復号マテリアルを CMP に要求し、CMP はそれを項目エンクリプタに返します。次に、項目エンクリプタは、暗号化マテリアルを使用して、項目の暗号化、署名、検証、および復号を行います。

CMP は、クライアントの設定時に指定します。互換性のあるカスタム CMP を作成するか、ライブラリ内の多くの CMP のいずれかを使用できます。ほとんどの CMP は、複数のプログラミング言語で使用できます。

項目エンクリプタ

-項目エンクリプタは DynamoDB 暗号化クライアントの暗号化オペレーションを実行する低レベルのコンポーネントです。項目エンクリプタは、暗号化マテリアルプロバイダー (p. 8) (CMP) に暗号化マテリアルをリクエストし、CMP より返るマテリアルを使用して、テーブル項目を暗号化して署名するか、検証して復号します。

項目エンクリプタと直接やり取りするか、ライブラリにあるヘルパーを使用することができます。たとえば、Java 用 DynamoDB 暗号化クライアントには`AttributeEncryptor`ヘルパークラス`DynamoDBMapper`と直接やり取りする代わりに`DynamoDBEncryptor`項目エンクリプタ。Python ライブラリには、項目エンクリプタとやり取りする、`EncryptedTable`、`EncryptedClient`、および `EncryptedResource` ヘルパークラスが含まれています。

属性アクション

属性アクションは、項目の各属性に対して実行するアクションを項目エンクリプタに指示します。

属性アクションの値は、次のいずれかの値になります。

- 暗号化して署名— 属性値を暗号化します。項目の署名に属性 (名前と値) を含めます。
- 署名のみ— 項目署名に属性を含めます。
- 何もしない— 属性を暗号化または署名しないでください。

機密データを保存できるすべての属性は、暗号化と署名を使用します。プライマリキー属性 (パーティションキーとソートキー) は、署名のみを使用します。マテリアル説明属性 (p. 10)および署名属性は、署名も暗号化もされていません。これらの属性の属性アクションを指定する必要はありません。

属性アクションを慎重に選択します。不確かな場合は、暗号化と署名を使用します。DynamoDB 暗号化クライアントを使用してテーブル項目を保護した後は、署名検証エラーのリスクを冒すことなく、属性のアクションを変更することはできません。詳細については、データモデルの変更 (p. 50) を参照してください。

> Warning
>
> プライマリキー属性を暗号化しないでください。DynamoDB がテーブル全体のスキャンを実行せずに項目を見つけられるように、プレーンテキストの状態を維持する必要があります。

そのファイルにDynamoDB 暗号化コンテキスト (p. 10)がプライマリキー属性を識別します。それらを暗号化しようとするとクライアントはエラーをスローします。

属性アクションの指定に使用する手法は、プログラミング言語ごとに異なります。また、使用するヘルパークラスに固有の場合もあります。

詳細については、使用しているプログラミング言語のドキュメントを参照してください。

- Python (p. 44)
- Java (p. 34)

マテリアル記述

暗号化されたテーブル項目のマテリアル説明は、暗号化アルゴリズムなどの情報で構成されます。この情報は、テーブル項目が暗号化および署名される仕組みに関するものです。暗号化マテリアルプロバイダー (p. 8) (CMP) は、暗号化し、署名するための暗号化マテリアルをアセンブルするときに、マテリアル説明を記録します。後で、項目を検証および復号するために暗号化されたマテリアルをアセンブルする必要がある場合は、そのマテリアル記述をガイドとして使用します。

DynamoDB 暗号化クライアントでは、マテリアル記述は 3 つの関連する要素について参照します。

リクエストされたマテリアル説明

暗号化マテリアルプロバイダー (p. 8) (CMP) によっては、暗号化アルゴリズムなどの高度なオプションを指定できます。選択肢を示すために、名前と値のペアをDynamoDB 暗号化コンテキスト (p. 10)テーブルアイテムを暗号化するリクエストで指定します。この要素は、リクエストされたマテリアル説明と呼ばれます。リクエストされたマテリアル記述の有効値は、選択した CMP によって定義されます。

Note

マテリアル記述は安全なデフォルト値を上書きできるため、やむを得ない理由がない限り、リクエストされたマテリアル記述を省略することをお奨めします。

実際のマテリアル記述

暗号化マテリアルプロバイダー (p. 8) (CMP) が返すマテリアル説明は、実際のマテリアル説明と呼ばれます。CMP が暗号化マテリアルを構築したときに使用した実際の値について説明します。また、通常、リクエストされたマテリアル記述で構成され、ある場合は追加と変更を含みます。

マテリアル記述属性

クライアントは、実際のマテリアル説明を暗号化項目のマテリアル説明属性に保存します。このマテリアル記述属性名は、`amzn-ddb-map-desc` で、その値は実際のマテリアル記述です。クライアントは、マテリアル記述属性の値を使用して、項目の検証および復号を行います。

DynamoDB 暗号化コンテキスト

-DynamoDB 暗号化コンテキストは、テーブルと項目に関する情報をに提供します。暗号化マテリアルプロバイダー (p. 8)(CMP)。高度な実装では、DynamoDB 暗号化コンテキストにリクエストされたマテリアル記述 (p. 10)。

テーブル項目を暗号化すると、DynamoDB 暗号化コンテキストが暗号化された属性値に暗号化でバインドされます。暗号化したときに、DynamoDB 暗号化コンテキストが暗号化に使用された DynamoDB 暗号化コンテキストに対して大文字と小文字を区別して完全に一致しない場合、復号オペレーションは失敗します。とやり取りする場合項目エンクリプタ (p. 9)暗号化メソッドまたは復号メソッドを呼び出すときに DynamoDB 暗号化コンテキストを提供する必要があります。ほとんどのヘルパーは DynamoDB 暗号化コンテキストを作成します。

Note

-DynamoDB 暗号化コンテキストDynamoDB 暗号化クライアントでは、暗号化コンテキストに AWS Key Management Service(AWS KMS) とAWS Encryption SDK。

DynamoDB 暗号化コンテキストには、次のフィールドを含めることができます。すべてのフィールドと値はオプションです。

- テーブル名
- パーティションキー名
- ソートキー名

- 属性名と値のペア
- リクエストされたマテリアル説明 (p. 10)

プロバイダーストア

プロバイダーストアは、暗号化マテリアルプロバイダー (p. 8) (CMP) を返すコンポーネントです。プロバイダーストアは、CMP を作成するか、別のプロバイダーストアなどの別のソースから CMP を取得できます。プロバイダーストアは、作成した CMP のバージョンを、保存されたそれぞれの CMP がリクエスタのマテリアル名とバージョン番号によって識別される永続的ストレージに保存します。

-最新プロバイダー (p. 21)DynamoDB 暗号化クライアントでは、プロバイダストアから CMP を取得しますが、プロバイダストアを使用して任意のコンポーネントに CMP を提供できます。各最新のプロバイダーは 1 つのプロバイダーストアに関連付けられていますが、プロバイダーストアは複数のホスト間で多くのリクエスタに CMP を提供できます。

プロバイダーストアは、オンデマンドで新しいバージョンの CMP を作成し、新しいバージョンと既存のバージョンを返します。また、指定されたマテリアル名の最新バージョン番号も返されます。これにより、リクエスタは、プロバイダーストアからリクエストできる新しいバージョンの CMP がリリースされるタイミングを把握することができます。

DynamoDB 暗号化クライアントにはMetaStore (p. 24)。これは、DynamoDB に格納され、内部 DynamoDB 暗号化クライアントを使用して暗号化されるキーを使用してラップされた CMP を作成するプロバイダーストアです。

詳細はこちら:

- プロバイダーストア: Java,Python
- MetaStore: Java,Python

暗号化マテリアルプロバイダーを選択する方法

DynamoDB 暗号化クライアントを使用するときに行う最も重要な決定の 1 つは、暗号化マテリアルのプロバイダー (p. 8)(CMP)。CMP は、暗号化マテリアルをアセンブルして、項目エンクリプタに返します。また、暗号化キーと署名キーの生成方法、新しいキーマテリアルが項目ごとに生成されるか、または再利用されるか、使用する暗号化アルゴリズムおよび署名アルゴリズムも指定されます。

DynamoDB 暗号化クライアントライブラリに含まれている実装から CMP を選択するか、互換性のあるカスタム CMP を構築できます。また、CMP の選択も、使用するプログラミング言語 (p. 32)によって異なります。

このトピックでは、一般的な CMP について説明するとともに、アプリケーションに最適な CMP を選択するのに役立ついくつかのアドバイスを提供します。

Direct KMS マテリアルプロバイダー

Direct KMS マテリアルプロバイダーは、AWS KMS keyそれは決して去らないでAWS Key Management Service(AWS KMS) 暗号化されていません。アプリケーションで、暗号化マテリアルを生成または管理する必要はありません。AWS KMS key を使用して、項目ごとに一意の暗号化キーと署名キーを生成するため、項目を暗号化または復号する際は必ず、このプロバイダーによって AWS KMS が呼び出されます。

AWS KMS を使用し、アプリケーションでトランザクションごとに 1 つの AWS KMS を呼び出す必要がある場合は、このプロバイダーを選択することをお勧めします。

詳細については、Direct KMS マテリアルプロバイダー (p. 13) を参照してください。

ラップされたマテリアルプロバイダー (ラップされた CMP)

ラップされたマテリアルプロバイダー (ラップされた CMP) では、DynamoDB 暗号化クライアントの外で、ラッピングおよび署名キーを生成および管理することができます。

ラップされた CMP は、項目ごとに一意の暗号化キーを生成します。次に、生成したラップキー (またはアンラップキー) および署名キーを使用します。したがって、ラップキーおよび署名キーの生成方法と、それらが各項目に一意か、または再利用されたものかを判断します。ラップされた CMP は、Direct KMS プロバイダー (p. 13)使用しないアプリケーションの場合AWS KMS暗号化資料を安全に管理できます。

詳細については、ラップされたマテリアルプロバイダー (p. 18) を参照してください。

最新プロバイダー

-最新プロバイダーは暗号化マテリアルのプロバイダー (p. 8)で機能するよう設計された (CMP)プロバイダーストア (p. 11)。プロバイダーストアから CMP を取得し、CMP から返る暗号化マテリアルを取得します。最新プロバイダーでは通常、各 CMP を使用して暗号化マテリアルの複数の要求を満たしますが、プロバイダーストアの機能を使用して、マテリアルの再利用範囲を制御したり、CMP の回転頻度を判断したりできるほか、最新プロバイダーを変更せずに使用される CMP のタイプを変更することもできます。

最新プロバイダーは互換性のあるプロバイダーストアで使用できます。DynamoDB 暗号化クライアントには、ラップされた CMP を返すプロバイダーストアである MetaStore が含まれます。

最新プロバイダーは、その暗号ソースへの呼び出しを最小限に抑える必要のあるアプリケーションや、セキュリティ要件に違反せずに一部の暗号化マテリアルを再利用できるアプリケーションに適

しています。たとえば、暗号化マテリアルをAWS KMS keyにAWS Key Management Service(AWS KMS) を呼び出さずにAWS KMS項目を暗号化または復号する度に

詳細については、最新プロバイダー (p. 21) を参照してください。

静的マテリアルプロバイダー

静的マテリアルプロバイダーは、検証や概念実証のデモンストレーション、および従来の互換性を目的として設計されています。項目ごとに一意の暗号化マテリアルが生成されることはありません。指定した暗号化キーと署名キーが返ります。これらのキーは、テーブル項目の暗号化、復号、および署名に直接使用されます。

Note

Java ライブラリ内の非対称静的プロバイダーは静的プロバイダーではありません。これは、ラップされた CMP (p. 18) の代替コンストラクタを指定するだけです。本稼働環境での使用は安全ですが、できるだけラップされた CMP を直接使用する必要があります。

トピック

- Direct KMS マテリアルプロバイダー (p. 13)
- ラップされたマテリアルプロバイダー (p. 18)
- 最新プロバイダー (p. 21)
- 静的マテリアルプロバイダー (p. 29)

Direct KMS マテリアルプロバイダー

Direct KMS マテリアルプロバイダー (Direct KMS プロバイダー) は、AWS KMS key によってテーブル項目を保護しているため、AWS Key Management Service (AWS KMS) は必ず暗号化されます。この暗号化マテリアルプロバイダー (p. 8)より、テーブル項目ごとに一意の暗号化キーと署名キーが返ります。そのためには、項目を暗号化または復号する度に AWS KMS を呼び出します。

高頻度で大規模の DynamoDB 項目を処理している場合は、AWS KMS の 1 秒あたりのリクエスト数の制限を超過し、処理が遅延する可能性があります。制限を超過する必要がある場合は、AWS Support センターでケースを作成してください。また、最新プロバイダー (p. 21)など、キーの再利用が制限された暗号化マテリアルプロバイダーの使用を検討することもできます。

Direct KMS プロバイダーを使用するには、呼び出し元に AWS アカウント、および少なくとも 1 つの AWS KMS key が必要であり、さらに、AWS KMS key で GenerateDataKey および Decrypt オペレーションを呼び出すためのアクセス許可も必要です。-AWS KMS keyは対称暗号化キーである必要があります。DynamoDB 暗号化クライアントは非対称暗号化をサポートしていません。DynamoDB グローバルテーブルを使用している場合、AWS KMSマルチリージョンキーを指定することもできます。詳細については、「使用方法 (p. 14)」を参照してください。

Note

Direct KMS プロバイダーを使用する場合は、プライマリーキー属性の名前と値がプレーンテキストで AWS KMS 暗号化コンテキストおよび関連 AWS KMS オペレーションの AWS CloudTrail ログに表示されます。ただし、DynamoDB 暗号化クライアントが、暗号化された属性値をプレーンテキストで公開することはありません。

Direct KMS プロバイダーは、DynamoDB 暗号化クライアントがサポートしている複数の暗号化マテリアルプロバイダー (p. 8) (CMP) の 1 つです。他の CMP の詳細については、「暗号化マテリアルプロバイダーを選択する方法 (p. 12)」を参照してください。

サンプルコードについては、以下を参照してください。

- Java: AwsKmsEncryptedItem

- Python: aws-kms-encrypted-table、aws-kms-encrypted-item

トピック
- 使用方法 (p. 14)
- 使用方法 (p. 16)

使用方法

Direct KMS プロバイダーを作成するには、キー ID パラメーターを使用して対称暗号化を指定します。KMS キーアカウントに。キー ID パラメータの値は、キー ID、キー ARN、エイリアス名、または AWS KMS key のエイリアス ARN にすることができます。キー ID の詳細については、AWS Key Management Service デベロッパーガイドの「キー識別子」を参照してください。

Direct KMS プロバイダーでは、対称暗号化 KMS キーが必要です。非対称 KMS キーを使用することはできません。ただし、マルチリージョン KMS キー、インポートされたキーマテリアルを含む KMS キー、またはカスタムキーストア内の KMS キーを使用できます。KMS キーに kms:GenerateDataKey アクセス許可と kms:Decrypt アクセス許可がある必要があります。そのため、AWS が管理する KMS キーや AWS が所有する KMS キーではなく、カスタマー管理のキーを使用する必要があります。

Python 用 DynamoDB 暗号化クライアントは、キー ID パラメータ値でリージョンから AWS KMS を呼び出すためのリージョンを決定します (リージョンが含まれている場合)。リージョンが含まれていない場合、AWS KMS クライアントで指定されているリージョンを使用するか、AWS SDK for Python (Boto3) で設定されているリージョンを使用します。Python でのリージョンの選択の詳細については、AWS SDK for Python (Boto3) API リファレンスの「設定」を参照してください。

Java 用 DynamoDB 暗号化クライアントは、指定したクライアントにリージョンが含まれている場合、AWS KMS クライアントのリージョンから AWS KMS を呼び出すリージョンを決定します。リージョンが含まれていない場合、AWS SDK for Java で設定されたリージョンが使用されます。AWS SDK for Java でのリージョンの選択の詳細については、AWS SDK for Java デベロッパーガイドの「AWS リージョン の選択」を参照してください。

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

次の例では、キー ARN を使用して AWS KMS key を指定しています。キー ID に AWS リージョン が含まれていない場合、DynamoDB 暗号化クライアントは、設定された Botocore セッションがある場合はそのセッションから、あるいは Boto デフォルトからリージョンを取得します。

```
# 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)
```

使用するものAmazon DynamoDB グローバルテーブルでは、データを暗号化してAWS KMSマルチリージョンキー。マルチリージョンキーとは、さまざまな AWS リージョン にある AWS KMS keys であり、キー ID とキーマテリアルが同じであるため、交換して使用できます。詳細については、AWS Key Management Service デベロッパーガイドの「マルチリージョンキーを使用する」を参照してください。

Note

グローバルテーブルを使用している場合バージョン 2017.11.29では、予約済みレプリケーションフィールドが暗号化または署名されないように、属性アクションを設定する必要があります。詳細については、「古いバージョンのグローバルテーブルに関する問題 (p. 55)」を参照してください。

DynamoDB 暗号化クライアントでマルチリージョンキーを使用するには、マルチリージョンキーを作成し、アプリケーションを実行するリージョンにレプリケートします。次に、DynamoDB 暗号化クライアントが AWS KMS を呼び出すリージョンでマルチリージョンキーを使用するように Direct KMS プロバイダーを設定します。

次の例では、マルチリージョンキーを使用して、米国東部 (バージニア北部) (us-east-1) リージョンのデータを暗号化し、米国西部 (オレゴン) (us-west-2) リージョンのデータを復号するように DynamoDB 暗号化クライアントを設定します。

Java

この例では、DynamoDB 暗号化クライアントは AWS KMS クライアントのリージョンから AWS KMS を呼び出すためのリージョンを取得します。`keyArn` 値は、同じリージョンのマルチリージョンキーを識別します。

```
// 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

この例では、DynamoDB 暗号化クライアントはキー ARN のリージョンから AWS KMS を呼び出すためのリージョンを取得します。

```
# 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)
```

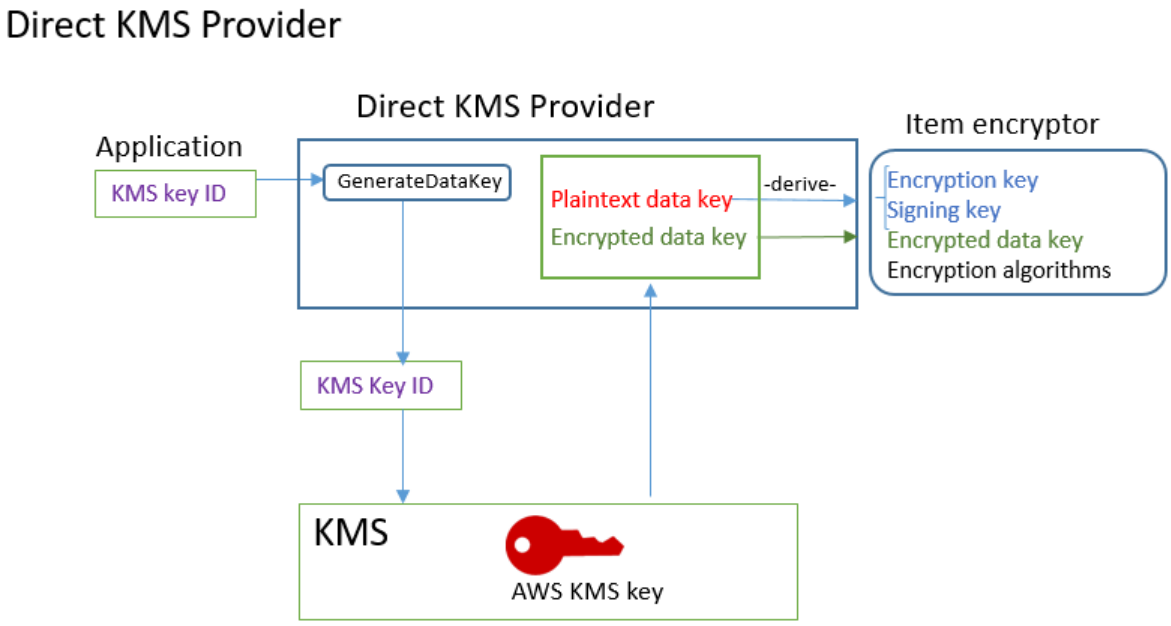
```
# Decrypt in us-west-2

# 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)
```

使用方法

以下の図に示されているように、Direct KMS プロバイダーは、指定した AWS KMS key で保護されている暗号化キーおよび署名キーを返します。



- 暗号化マテリアルを生成するために、Direct KMS プロバイダーは AWS KMS に、ユーザーが指定した AWS KMS key を使用して項目ごとに一意のデータキーを生成するように要求します。これにより、データキーのプレーンテキストコピーから項目の暗号化キーと署名キーが導出され、暗号化データキーと一緒に返ります。このデータキーは、項目のマテリアル記述属性 (p. 10)に保存されます。

  項目エンクリプタでは、この暗号化キーおよび署名キーを使用します。また、メモリから可能な限り早くそれらを削除します。導出されたデータキーの暗号化されたコピーのみ、暗号化された項目に保存されます。
- 復号マテリアルを生成するには、Direct KMS プロバイダーは暗号化されたデータキーを復号するよう AWS KMS に求めます。これにより、プレーンテキストデータキーより検証キーおよび署名キーが導出され、項目エンクリプタに返されます。

  項目エンクリプタは項目を検証し、検証が成功すると、暗号化された値が復号されます。次に、可能な限り早く、メモリよりキーが削除されます。

暗号化マテリアルを取得する

このセクションでは、項目エンクリプタ (p. 9)より暗号化マテリアルのリクエストを受け取るときの Direct KMS プロバイダーの入力、出力、処理の詳細について説明します。

入力 (アプリケーションから)

- のキー ID。AWS KMS key。

入力 (項目エンクリプタから)

- DynamoDB 暗号化コンテキスト (p. 10)

出力 (項目エンクリプタへ)

- 暗号化キー (プレーンテキスト)
- 署名キー
- In (イン)実際のマテリアル記述 (p. 10): これらの値は、クライアントが項目に追加するマテリアル説明属性に保存されます。
  - amzn-ddb-env-key: で暗号化されている Base64 エンコードのデータAWS KMS key
  - amzn-ddb-env-alg: 暗号化アルゴリズム (デフォルト)AES/256
  - amzn-ddb-sig-alg: 署名アルゴリズム。デフォルトでは、HmacSHA256/256
  - amzn-ddb-wrap-alg: kms

Processing

1. Direct KMS プロバイダーは AWS KMS に、指定された AWS KMS key を使用して項目の一意のデータキーを生成するように要求します。このオペレーションによって、プレーンテキストキーと、AWS KMS key で暗号化されたコピーが返ります。これは、初期のキーマテリアルと呼ばれます。

   このリクエストの AWS KMS 暗号化テキストには、次のプレーンテキスト形式の値が含まれています。これらのシークレットではない値は、暗号化されたオブジェクトに暗号的にバインドされているため、復号時には同じ暗号化コンテキストが必要です。これらの値を使用して、AWS KMS への呼び出しを AWS CloudTrail ログで識別します。
   - amzn-ddb-env-alg - 暗号化アルゴリズム。デフォルトは AES/256
   - amzn-ddb-sig-alg - 署名アルゴリズム。デフォルトは HmacSHA256/256
   - (オプション) aws-kms-table –*#####*
   - (オプション) *##########* – *##########* (バイナリ値は Base64 エンコード形式)
   - (オプション) *######* – *######* (バイナリ値は Base64 エンコード形式)

   Direct KMS プロバイダーは、項目の DynamoDB 暗号化コンテキスト (p. 10)から AWS KMS 暗号化コンテキストの値を取得します。DynamoDB 暗号化コンテキストにテーブル名などの値が含まれていない場合は、その名前と値のペアが AWS KMS 暗号化コンテキストから除外されます。
2. Direct KMS プロバイダーは、対称暗号化キーおよび署名キーをデータキーから導出します。デフォルトでは、セキュアハッシュアルゴリズム (SHA) 256 および RFC5869 HMAC ベースのキー導出関数を使用して、256 ビット AES 対称暗号化キーおよび 256 ビット HMAC-SHA-256 署名キーを導出します。
3. Direct KMS プロバイダーは、項目エンクリプタに出力を返します。
4. 項目エンクリプタは、暗号化キーを使用して、指定された属性を暗号化し、署名キーを使用して署名します。この際、実際のマテリアル記述で指定されたアルゴリズムを使用します。可能な限り早く、メモリよりプレーンテキストキーが削除されます。

復号マテリアルを取得する

このセクションでは、項目エンクリプタ (p. 9)より復号マテリアルのリクエストを受け取るときの Direct KMS プロバイダーの入力、出力、処理の詳細について説明します。

入力 (アプリケーションから)

- のキー ID。AWS KMS key。

  キー ID の値は、キー ID、キー ARN、エイリアス名、または AWS KMS key のエイリアス ARN にすることができます。キー ID に含まれていない値 (リージョンなど) はすべて、AWS 名前付きプロファイルで入手できる必要があります。キー ARN により、AWS KMS で必要なすべての値が提供されます。

入力 (項目エンクリプタから)

- DynamoDB 暗号化コンテキスト (p. 10)のコピー (マテリアル説明属性の内容を含む)。

出力 (項目エンクリプタへ)

- 暗号化キー (プレーンテキスト)
- 署名キー

Processing

1. Direct KMS プロバイダーは、暗号化された項目のマテリアル記述属性から暗号化されたデータキーを取得します。
2. AWS KMS に、指定された AWS KMS key を使用して暗号化されたデータキーを復号するように求めます。オペレーションでプレーンテキストのキーが返ります。

   このリクエストでは、データキーの生成および暗号化に使用したのと同じ AWS KMS 暗号化コンテキストを使用する必要があります。

   - aws-kms-table – *#####*
   - *##########* – *##########* (バイナリ値は Base64 エンコード形式)
   - (オプション) *######* – *######* (バイナリ値は Base64 エンコード形式)
   - amzn-ddb-env-alg - 暗号化アルゴリズム。デフォルトは AES/256
   - amzn-ddb-sig-alg - 署名アルゴリズム。デフォルトは HmacSHA256/256
3. Direct KMS プロバイダーでは、セキュアハッシュアルゴリズム (SHA) 256 および RFC5869 HMAC ベースのキー導出関数を使用して、データキーから 256 ビット AES 対称暗号化キーおよび 256 ビット HMAC-SHA-256 署名キーを導出します。
4. Direct KMS プロバイダーは、項目エンクリプタに出力を返します。
5. 項目エンクリプタは、署名キーを使用して項目を検証します。成功すると、暗号化された属性値は対称暗号化キーを使用して復号されます。これらのオペレーションでは、実際のマテリアル記述で指定された暗号化アルゴリズムおよび署名アルゴリズムが使用されます。項目エンクリプタによって、可能な限り早く、メモリよりプレーンテキストキーが削除されます。

ラップされたマテリアルプロバイダー

-ラップされたマテリアルプロバイダー(ラップされた CMP) では、DynamoDB 暗号化クライアントを使用して任意のソースからラッピングおよび署名キーを使用できます。ラップされた CMP は、いずれにも依存しません。AWSサービス。ただし、クライアントの外部にあるラップキーと署名キーを生成して管理する必要があります。これには、項目を検証および復号するための正しいキーを提供することが含まれます。

ラップされた CMP は、項目ごとに固有の項目暗号化キーを生成します。項目暗号化キーを指定したラップキーでラップし、ラップされた項目暗号化キーを項目のマテリアル説明属性 (p. 10)に保存します。ラップキーと署名キーを指定するため、ラップキーと署名キーの生成方法と、それらが各項目に固有のものか再利用されたものかを判断します。

ラップされた CMP は、安全な実装であり、暗号化マテリアルを管理できるアプリケーションに適しています。

ラップされたCMPは、いくつかのうちの1つです。暗号化マテリアルプロバイダー (p. 8)DynamoDB 暗号化クライアントがサポートしている (CMP)。他の CMP の詳細については、「暗号化マテリアルプロバイダーを選択する方法 (p. 12)」を参照してください。

サンプルコードについては、以下を参照してください。

- Java: AsymmetricEncryptedItem
- Python: wrapped-rsa-encrypted-table、wrapped-symmetric-encrypted-table

トピック

使用方法

ラップされた CMP を作成するには、ラップキー (暗号化に必要)、ラップ解除キー (復号に必要)、および署名キーを指定します。項目を暗号化および復号するときには、キーを指定する必要があります。

ラップキー、ラップ解除キー、および署名キーは、対称キーまたは非対称キーペアにすることができます。

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);
```

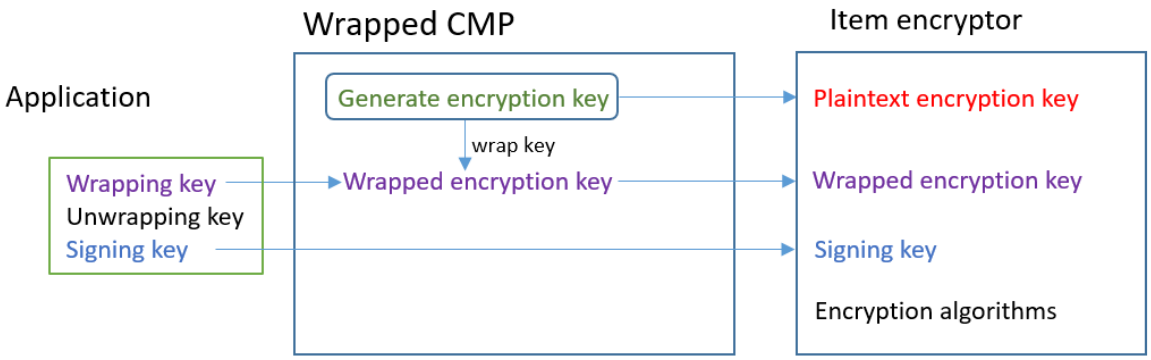
Python

```
# 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
)
```

仕組み

ラップされた CMP は、すべての項目に新しい項目暗号化キーを生成します。次の図に示すように、ラップキー、ラップ解除キー、および署名キーを使用します。



暗号化マテリアルを取得する

このセクションでは、暗号化マテリアルのリクエストを受け取る際のラップされたマテリアルプロバイダー (ラップされた CMP) の入力、出力、処理の詳細について説明します。

入力 (アプリケーションから)

- ラップキー: AnAdvanced Encryption Standard(AES) 対称キー、またはRSAパブリックキー。属性値が暗号化されている場合は必須です。それ以外の場合はオプションであり、無視されます。
- アンラッピングキー: オプションで無視されます。
- 署名キー

入力 (項目エンクリプタから)

- DynamoDB 暗号化コンテキスト (p. 10)

出力 (項目エンクリプタへ):

- プレーンテキスト項目暗号化キー
- 署名キー (変更されません)
- 実際のマテリアル記述 (p. 10): これらの値は、マテリアル記述属性 (p. 10)クライアントがアイテムに追加すること。
  - `amzn-ddb-env-key`: Base64 でエンコードされたラップされた項目暗号化キー
  - `amzn-ddb-env-alg`: 項目を暗号化するために使用される暗号化アルゴリズム。デフォルトは AES-256-CBC です。
  - `amzn-ddb-wrap-alg`: ラップされた CMP が項目暗号化キーをラップするために使用したラップアルゴリズム。ラッピングキーが AES キーの場合、キーはパッドなしを使用してラップされます。`AES-Keywrap`で定義された。RFC 3394。ラップキーが RSA キーの場合、キーは MGF1 パディング付き RSA OAEP を使用して暗号化されます。

Processing

項目を暗号化する際は、ラップキーと署名キーで渡します。ラップ解除キーは、オプションで無視されます。

1. ラップされた CMP は、テーブル項目に固有の対称項目暗号化キーを生成します。
2. 項目暗号化キーをラップするために指定したラップキーを使用します。次に、可能な限り早く、メモリより削除されます。

3. これは、プレーンテキスト項目暗号化キー、指定した署名キー、実際のマテリアル説明 (p. 10) (ラップされた項目暗号化キー、暗号化およびラップアルゴリズムを含む) を返します。
4. 項目エンクリプタは、プレーンテキスト暗号化キーを使用して項目を暗号化します。項目に署名するために指定した署名キーを使用します。次に、可能な限り早く、メモリよりプレーンテキストキーが削除されます。ラップされた暗号化キー (`amzn-ddb-env-key`) を含む、実際のマテリアル記述のフィールドを項目のマテリアル記述属性にコピーします。

復号マテリアルを取得する

このセクションでは、復号マテリアルのリクエストを受け取る際のラップされたマテリアルプロバイダー (ラップされた CMP) の入力、出力、処理の詳細について説明します。

入力 (アプリケーションから)

- ラップキー: オプションで無視されます。
- アンラッピングキー: 同じAdvanced Encryption Standard(AES) 対称キーまたはRSA暗号化に使用された RSA パブリックキーに対応するプライベートキー。属性値が暗号化されている場合は必須です。それ以外の場合はオプションであり、無視されます。
- 署名キー

入力 (項目エンクリプタから)

- の写しDynamoDB 暗号化コンテキスト (p. 10)これには、マテリアル記述属性の内容が含まれます。

出力 (項目エンクリプタへ)

- プレーンテキスト項目暗号化キー
- 署名キー (変更されません)

Processing

項目を復号する際は、ラップ解除キーと署名キーで渡します。ラップキーは、オプションで無視されます。

1. ラップされた CMP は、項目のマテリアル記述属性からラップされた項目暗号化キーを取得します。
2. 項目暗号化キーをラップ解除するためにラップ解除キーとアルゴリズムを使用します。
3. それは、項目エンクリプタにプレーンテキスト項目暗号化キー、署名キー、および暗号化および署名アルゴリズムを返します。
4. 項目エンクリプタは、署名キーを使用して項目を検証します。成功すると、項目暗号化キーを使用して項目を復号します。次に、可能な限り早く、メモリよりプレーンテキストキーが削除されます。

最新プロバイダー

-最新プロバイダーは暗号化マテリアルプロバイダー (p. 8)で機能するよう設計された (CMP)プロバイダーストア (p. 11)。プロバイダーストアから CMP を取得し、CMP から返る暗号化マテリアルを取得します。これは、通常、各 CMP を使用して複数の暗号化マテリアルをリクエストします。ただし、プロバイダーストアの機能を使用して、マテリアルが再利用される範囲を制御し、CMP のローテーション頻度を決定し、最新プロバイダーを変更せずに使用する CMP のタイプを変更することもできます。

Note

に関連付けられたコードMostRecentProvider最新のプロバイダのシンボルは、プロセスの存続期間中、暗号化材料をメモリに保存することがあります。これにより、呼び出し元が使用する権限がなくなったキーを使用できる場合があります。
-MostRecentProviderシンボルは、DynamoDB 暗号化クライアントの古いサポートされているバージョンでは廃止され、バージョン 2.0.0 から削除されます。と置き換えられる。CachingMostRecentProvider記号。詳細については、最新プロバイダーの更新 (p. 28)を参照してください。

最新プロバイダーは、プロバイダーストアとその暗号ソースへの呼び出しを最小限に抑える必要のあるアプリケーションや、セキュリティ要件に違反せずに一部の暗号化マテリアルを再利用できるアプリケーションに適しています。たとえば、暗号化マテリアルをAWS KMS keyにAWS Key Management Service(AWS KMS) を呼び出さずにAWS KMS項目を暗号化または復号する度に。

選択したプロバイダーストアによって、最新プロバイダーが使用する CMP のタイプと、新しい CMP を取得する頻度が決まります。設計したカスタムプロバイダーストアを含む、最新プロバイダーと互換性のある任意のプロバイダーストアを使用できます。

DynamoDB 暗号化クライアントには、MetaStoreそれは作成して返すラップされたマテリアルプロバイダー (p. 18)(ラップされた CMP)。MetaStore は、生成したラップされた CMP の複数のバージョンを内部 DynamoDB テーブルに保存し、DynamoDB 暗号化クライアントの内部インスタンスによるクライアント側の暗号化でそれらを保護します。

MetaStore は、テーブル内のマテリアルを保護するために、任意のタイプの内部 CMP を使用するように構成できます。Direct KMS プロバイダー (p. 13)によって保護されている暗号資料が生成されるAWS KMS key、指定したラッピングおよび署名キーを使用するラップ CMP、または設計する互換性のあるカスタム CMP。

サンプルコードについては、以下を参照してください。

- Java: MostRecentEncryptedItem
- Python: most_recent_provider_encrypted_table

トピック
- 使用方法 (p. 22)
- 仕組み (p. 24)
- 最新プロバイダーの更新 (p. 28)

使用方法

最新プロバイダーを作成するには、プロバイダーストアを作成して構成した後、プロバイダーストアを使用する最新プロバイダーを作成する必要があります。

次の例は、MetaStore を使用し、内部の DynamoDB テーブルのバージョンを暗号化マテリアルで保護する最新のプロバイダーを作成する方法を示しています。Direct KMS プロバイダー (p. 13)。以下の例では、`CachingMostRecentProvider` (p. 28)記号。

最新の各プロバイダには、MetaStore テーブル内の CMP を識別する名前があります。有効期限 (p. 25)(TTL) 設定、およびキャッシュが保持できるエントリの数を決定するキャッシュサイズ設定。以下の例では、キャッシュサイズを 1000 エントリ、TTL を 60 秒に設定します。

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
)
```

仕組み

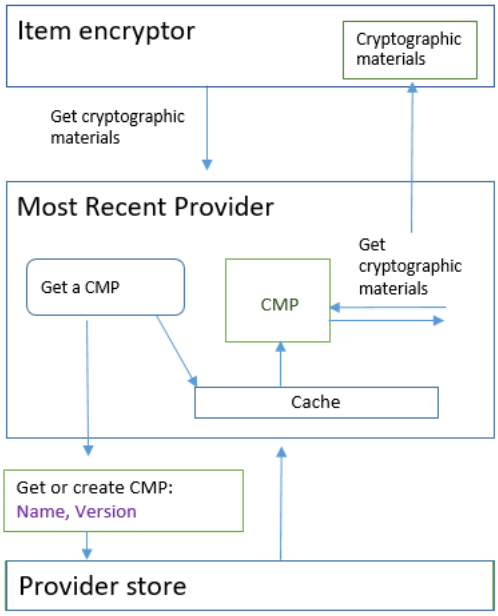
最新プロバイダーがプロバイダーストアから CMP を取得します。次に、CMP を使用して、暗号化マテリアルを生成し、それを項目エンクリプタに返します。

最新プロバイダーについて

最新プロバイダーは暗号化マテリアルプロバイダー (p. 8)(CMP)プロバイダーストア (p. 11)。次に、CMP を使用して、それが返す暗号化マテリアルを生成します。各最新プロバイダーは 1 つのプロバイダーストアに関連付けられていますが、プロバイダーストアは複数のホスト間で複数のプロバイダーに CMP を提供できます。

最新プロバイダーは、任意のプロバイダーストアから互換性のある CMP を使用できます。これは CMP の暗号化または復号マテリアルをリクエストし、その出力を項目エンクリプタに返します。暗号化オペレーションは実行されません。

最新プロバイダーは、そのプロバイダーストアから CMP を要求するために、使用する既存の CMP のマテリアル名とバージョンを提供します。暗号化マテリアルでは、最新プロバイダーは常に最大 (「最新の」) バージョンをリクエストします。復号マテリアルの場合、次の図に示すように、暗号化マテリアルの作成に使用された CMP のバージョンをリクエストします。



最新プロバイダーは、プロバイダーストアが返す CMP のバージョンをメモリ内のローカル最小使用 (LRU) キャッシュに保存します。キャッシュにより、最新プロバイダーは、すべての項目のプロバイダーストアを呼び出さずに必要な CMP を取得できます。必要に応じてキャッシュをクリアすることができます。

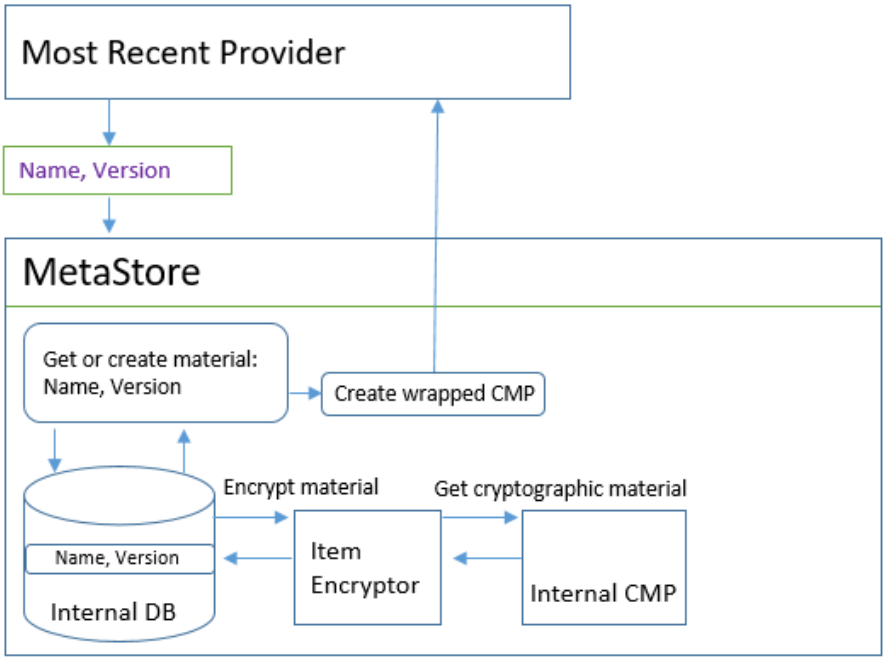
最新プロバイダーは、構成可能を使用します。有効期限 — 有効期限 (p. 25)アプリケーションの特性に基づいて調整できます。

MetaStore について

互換性のあるカスタムプロバイダーストアなどの任意のプロバイダーストアで最新プロバイダーを使用できます。DynamoDB 暗号化クライアントには、構成してカスタマイズできる安全な実装である MetaStore が含まれています。

あるMetaStoreはプロバイダーストア (p. 11)それは作成して返すラップされた CMP (p. 18)Wrapped CMP が必要とするラッピングキー、アンラッピングキー、および署名キーで設定されています。ラップされた CMP は常にすべての項目に対して一意の項目暗号化キーを生成するため、MetaStore は最新プロバイダーにとって安全なオプションです。項目暗号化キーと署名キーを保護するラップキーのみが再利用されます。

次の図は、MetaStore のコンポーネントと、最新プロバイダーとのやり取りの方法を示しています。



MetaStore は、ラップされた CMP を生成し、内部のDynamoDB テーブルに保存します。パーティションキーは、最新のプロバイダマテリアルの名前で、ソートキーはそのバージョン番号です。テーブル内のマテリアルは、アイテム暗号化と内部を含む、内部 DynamoDB 暗号化クライアントによって保護されています。暗号化マテリアルプロバイダー (p. 8)(CMP)。

MetaStore では、Direct KMS プロバイダー (p. 18)、提供する暗号化マテリアルを使用したラップされた CMP、または互換性のあるカスタム CMP など、あらゆるタイプの内部 CMP を使うことができます。MetaStore の内部 CMP が Direct KMS プロバイダーの場合、再利用可能なラッピングおよび署名キーは、AWS KMS keyにAWS Key Management Service(AWS KMS). MetaStore は、内部テーブルに新しい CMP バージョンを追加するか、内部テーブルから CMP バージョンを取得するたびに AWS KMS を呼び出します。

有効期限 (有効期限)

作成した最新プロバイダーの有効期限 (TTL) 値を設定できます。通常、アプリケーションに対して実用的である最小の TTL 値を使用します。

TTL 値の使用は、`CachingMostRecentProvider`最新プロバイダーの記号。

> **Note**
>
> -`MostRecentProvider`最新のプロバイダーのシンボルは、DynamoDB 暗号化クライアントの古いサポートバージョンでは廃止され、バージョン 2.0.0 から削除されました。と置き換えられる。`CachingMostRecentProvider`記号。コードはできる限り早く更新することをお勧めします。詳細については、最新プロバイダーの更新 (p. 28) を参照してください。

`CachingMostRecentProvider`

`-CachingMostRecentProvider`では、TTL 値を 2 つの異なる方法で使用します。

- TTL は、[最新のプロバイダー] がプロバイダーストアで CMP の新しいバージョンをチェックする頻度を決定します。新しいバージョンが利用可能な場合、最新のプロバイダはその CMP を置き換え、暗号化マテリアルを更新します。それ以外の場合は、現在の CMP および暗号化マテリアルを引き続き使用します。
- TTL は、キャッシュ内の CMP を使用できる期間を決定します。キャッシュされた CMP を暗号化に使用する前に、最新のプロバイダはキャッシュ内の時間を評価します。CMP キャッシュ時間が TTL を超えると、CMP はキャッシュから削除され、最新のプロバイダはプロバイダストアから新しい最新バージョン CMP を取得します。

`MostRecentProvider`

左`MostRecentProvider`では、TTL は、最新のプロバイダーがプロバイダーストアで新しいバージョンの CMP をチェックする頻度を決定します。新しいバージョンが利用可能な場合、最新のプロバイダはその CMP を置き換え、暗号化マテリアルを更新します。それ以外の場合は、現在の CMP および暗号化マテリアルを引き続き使用します。

TTL では、新しい CMP バージョンが作成される頻度は決定されません。新しい CMP バージョンを作成するには、暗号化マテリアルの回転 (p. 26)。

理想的な TTL 値は、アプリケーションとそのレイテンシーと可用性の目標によって異なります。TTL を小さくすると、暗号化マテリアルがメモリに格納される時間が短縮され、セキュリティプロファイルが向上します。また、TTL が低いほど、重要な情報がより頻繁に更新されます。たとえば、内部 CMP がDirect KMS プロバイダー (p. 13)では、呼び出し元が引き続き使用する権限があることを、より頻繁に検証します。AWS KMS key。

ただし、TTL が短すぎると、プロバイダーストアへの頻繁な呼び出しによってコストが増加し、プロバイダーストアがアプリケーションやサービスアカウントを共有する他のアプリケーションからのリクエストを抑制する可能性があります。また、TTL を暗号化マテリアルを回転させる速度を調整することでメリットが得られる場合があります。

テスト中に、アプリケーションおよびセキュリティおよびパフォーマンス標準に適した構成が見つかるまで、さまざまなワークロードで TTL とキャッシュサイズを変更します。

暗号化マテリアルの回転

最新のプロバイダが暗号化マテリアルを必要とする場合、常に認識している最新バージョンの CMP を使用します。新しいバージョンをチェックする頻度は、有効期限 (p. 25)[最新のプロバイダ] を構成するときに設定する (TTL) 値。

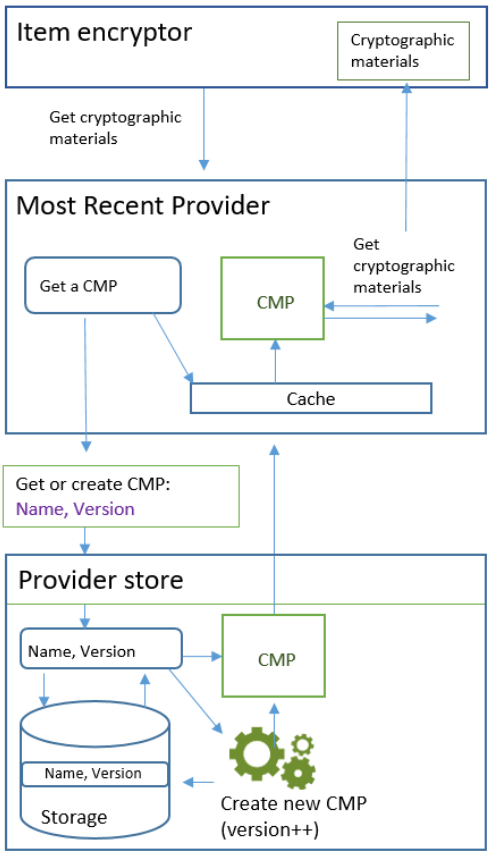
TTL の有効期限が切れると、[最新のプロバイダー] はプロバイダーストアに CMP の新しいバージョンがあるかどうかをチェックします。使用可能な場合は、最新のプロバイダがそれを取得し、キャッシュ内の CMP を置き換えます。プロバイダストアに新しいバージョンがあることが検出されるまで、この CMP とその暗号化マテリアルを使用します。

最新プロバイダーの新しいバージョンの CMP を作成するようにプロバイダーストアに指示するには、プロバイダーストアの新規プロバイダーの作成オペレーション作を、最新プロバイダーのマテリアル名で呼び出します。プロバイダーストアは新しい CMP を作成し、暗号化されたコピーをより大きなバージョン番号で内部ストレージに保存します。(CMP を返しますが、破棄することもできます。) その結果、次に最新プロバイダーがプロバイダーストアにその CMP の最大バージョン番号を問い合わせると、新しいより大きなバージョン番号を取得し、それをストアに対する後続のリクエストで使用して、CMP の新しいバージョンが作成されたかどうかを確認します。

時間、処理された項目または属性の数、またはアプリケーションに合ったその他のメトリクスに基づいて、新しいプロバイダー作成コールをスケジュールできます。

暗号化マテリアルを取得する

最新プロバイダーは、この図に示す次のプロセスを使用して、項目エンクリプタに返す暗号化マテリアルを取得します。出力は、プロバイダーストアが返す CMP のタイプによって異なります。最新プロバイダーは、DynamoDB 暗号化クライアントに含まれる MetaStore などの互換性のある任意のプロバイダーストアを使用できます。



を使用して最新のプロバイダを作成する場合`CachingMostRecentProvider`シンボル (p. 28)では、プロバイダストア、最新のプロバイダの名前、および有効期限 (p. 25)(TTL) 値。オプションで、キャッシュ内に存在できる暗号化マテリアルの最大数を決定するキャッシュサイズを指定することもできます。

項目エンクリプタが最新プロバイダーに暗号化マテリアルを要求すると、最新プロバイダーは、その CMP の最新バージョンのキャッシュの検索を開始します。

- キャッシュ内で最新バージョンの CMP を検出し、CMP が TTL 値を超えていない場合、最新プロバイダーは CMP を使用して暗号化マテリアルを生成します。次に、暗号化マテリアルを項目エンクリプタに返します。このオペレーションでは、プロバイダーストアへの呼び出しは必要ありません。
- CMP の最新バージョンがキャッシュにない場合、またはキャッシュにあるがその TTL 値を超えた場合、最新プロバイダーはそのプロバイダーストアに CMP をリクエストします。リクエストには、最新プロバイダーのマテリアル名と、既知の最大のバージョン番号が含まれています。
  1. プロバイダーストアは、永続的ストレージから CMP を返します。プロバイダーストアが MetaStore の場合、最新プロバイダーのマテリアル名をパーティションキーとして使用し、バージョン番号をソートキーとして使用して、内部のDynamoDB テーブルから暗号化された Wrapped CMP を取得します。MetaStore は、内部項目エンクリプタと内部 CMP を使用して、ラップされた CMP を復号します。次に、プレーンテキスト CMP を最新プロバイダーに返します。内部 CMP が Direct KMS Provider (p. 13) の場合、このステップには AWS Key Management Service (AWS KMS) コールが含まれます。

2. CMP は、`amzn-ddb-meta-id`フィールドを実際のマテリアル説明 (p. 10)に追加します。その値は、内部テーブルの CMP のマテリアル名とバージョンです。プロバイダーストアは CMP を最新プロバイダーに返します。
3. 最新プロバイダーは CMP をメモリにキャッシュします。
4. 最新プロバイダーは CMP を使用して暗号化マテリアルを生成します。次に、暗号化マテリアルを項目エンクリプタに返します。

復号マテリアルを取得する

項目エンクリプタが最新プロバイダーに復号マテリアルを要求すると、最新プロバイダーは以下のプロセスを使用して、それらを取得し返します。

1. 最新プロバイダーは、項目を暗号化するために使用された暗号化マテリアルのバージョン番号をプロバイダーストアに問い合わせます。項目のマテリアル説明属性 (p. 10)から実際のマテリアル説明を渡します。
2. プロバイダーストアは、実際のマテリアル説明の `amzn-ddb-meta-id` フィールドから暗号化 CMP バージョン番号を取得し、最新プロバイダーに返します。
3. 最新プロバイダーは、項目の暗号化と署名に使用された CMP のバージョンをキャッシュ内で検索します。

- CMP の一致するバージョンがキャッシュにあり、CMP が有効期限 (TTL) 値 (p. 25)の場合、最新プロバイダーは CMP を使用して復号マテリアルを生成します。次に、復号マテリアルを項目エンクリプタに返します。このオペレーションでは、プロバイダーストアまたは他の CMP への呼び出しは必要ありません。
- CMP の一致するバージョンがキャッシュにない場合、またはキャッシュされた CMP の一致するバージョンがキャッシュにない場合AWS KMS keyが TTL 値を超えている場合、最新プロバイダーはそのプロバイダーストアに CMP をリクエストします。リクエストには、マテリアル名および暗号化 CMP バージョン番号が送信されます。
  1. プロバイダーストアは、最新プロバイダー名をパーティションキーとして使用し、バージョン番号をソートキーとして使用して、CMP の永続的ストレージを検索します。
     - 名前とバージョン番号が永続的ストレージにない場合、プロバイダーストアは例外をスローします。CMP を生成するためにプロバイダーストアを使用した場合、意図的に削除されていない限り、CMP は永続的ストレージに保存する必要があります。
     - 一致する名前とバージョン番号を持つ CMP がプロバイダーストアの永続的ストレージにある場合、プロバイダーストアは指定された CMP を最新プロバイダーに返します。

       プロバイダーストアが MetaStore の場合、その DynamoDB テーブルから暗号化された CMP を取得します。次に、内部 CMP の暗号化マテリアルを使用して、CMP を最新プロバイダーに返す前に暗号化された CMP を復号します。内部 CMP が Direct KMS Provider (p. 13) の場合、このステップには AWS Key Management Service (AWS KMS) コールが含まれます。
  2. 最新プロバイダーは CMP をメモリにキャッシュします。
  3. 最新プロバイダーは CMP を使用して復号マテリアルを生成します。次に、復号マテリアルを項目エンクリプタに返します。

最新プロバイダーの更新

[最新のプロバイダ] のシンボルがから変更されます。`MostRecentProvider`に`CachingMostRecentProvider`。

Note

-`MostRecentProvider`最新のプロバイダを表すシンボルは、Java 用 DynamoDB 暗号化クライアントのバージョン 1.15 および Python 用 DynamoDB 暗号化クライアントのバージョン 1.3 で

廃止され、両方の言語実装で DynamoDB 暗号化クライアントのバージョン 2.0.0 から削除されました。代わりに、`CachingMostRecentProvider`。

-`CachingMostRecentProvider`には以下の変更が実装されています。

- -`CachingMostRecentProvider`メモリ内の時間が構成された時間を超えると、暗号化マテリアルをメモリから定期的に削除します。有効期限 (TTL) 値 (p. 25)。

  -`MostRecentProvider`は、プロセスの存続期間中、暗号化材料をメモリに保存することがあります。その結果、最新のプロバイダは承認の変更を認識しない可能性があります。呼び出し元のアクセス権限が取り消された後に、暗号化キーを使用する場合があります。

  この新しいバージョンにアップデートできない場合は、定期的に呼び出すことで同様の効果が得られます。`clear()`キャッシュ上のメソッドです。このメソッドは、キャッシュの内容を手動でフラッシュし、最新のプロバイダが新しい CMP と新しい暗号化マテリアルを要求する必要があります。
- -`CachingMostRecentProvider`また、キャッシュをより詳細に制御できるキャッシュサイズ設定も含まれています。

に更新するには`CachingMostRecentProvider`では、コード内のシンボル名を変更する必要があります。その他すべての点において、`CachingMostRecentProvider`と完全に下位互換性がありますか`MostRecentProvider`。テーブル項目を再暗号化する必要はありません。

ただし、`CachingMostRecentProvider`は、基礎となる主要なインフラストラクチャへの呼び出しをさらに生成します。これは、有効期限 (TTL) 間隔でプロバイダーストアを少なくとも 1 回呼び出します。多数のアクティブな CMP を持つアプリケーション (頻繁なローテーションによる) または大規模なフリートを持つアプリケーションは、この変更の影響を受ける可能性が最も高くなります。

更新されたコードをリリースする前に、コードを徹底的にテストして、頻繁な呼び出しがアプリケーションを損なったり、プロバイダーが依存するサービスによるスロットリングを引き起こさないようにしてください。AWS Key Management Service(AWS KMS) またはAmazon DynamoDB。パフォーマンスの問題を軽減するには、キャッシュサイズと有効期間を調整して`CachingMostRecentProvider`観察するパフォーマンス特性に基づきます。ガイダンスについては、「有効期限 (有効期限) (p. 25)」を参照してください。

静的マテリアルプロバイダー

-静的マテリアルプロバイダー(静的 CMP) は非常にシンプルです暗号化マテリアルプロバイダー (p. 8)(CMP) は、検証や概念実証のデモンストレーション、および従来の互換性を目的とした。

静的 CMP を使用してテーブル項目を暗号化するには、Advanced Encryption Standard (AES) 対称暗号化キーと署名キーまたはキーペアを指定します。暗号化された項目を復号するために同じキーを指定する必要があります。静的 CMP は暗号化オペレーションを実行しません。代わりに、項目エンクリプタに指定した暗号化キーをそのまま渡します。項目エンクリプタは、暗号化キーの直下の項目を暗号化します。次に、署名キーを直接使用して署名します。

静的 CMP は一意の暗号化マテリアルを生成しないため、処理するすべてのテーブル項目は同じ暗号化キーで暗号化され、同じ署名キーで署名されます。同じキーを使用して多数の項目の属性値を暗号化するか、同じキーまたはキーペアを使用してすべての項目に署名すると、キーの暗号化の制限を超える危険性があります。

> **Note**
>
> Java ライブラリ内の非対称静的プロバイダーは静的プロバイダーではありません。これは、ラップされた CMP (p. 18) の代替コンストラクタを指定するだけです。本稼働環境での使用は安全ですが、できるだけラップされた CMP を直接使用する必要があります。

スタティック CMP は、いくつかのうちの 1 つです。暗号化マテリアルプロバイダー (p. 8)DynamoDB 暗号化クライアントがサポートしている (CMP)。他の CMP の詳細については、「暗号化マテリアルプロバイダーを選択する方法 (p. 12)」を参照してください。

サンプルコードについては、以下を参照してください。

- Java: SymmetricEncryptedItem

トピック

使用方法

静的なプロバイダーを作成するには、暗号化キーやキーペアおよび署名キーやキーペアを指定します。テーブル項目を暗号化および復号するには、キーマテリアルを指定する必要があります。

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

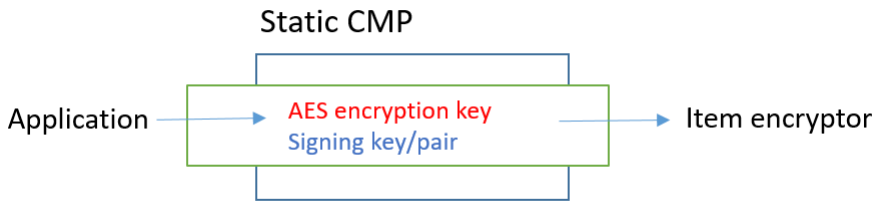
```
# You can provide encryption materials, decryption materials, or both
encrypt_keys = EncryptionMaterials(
    encryption_key = ...,
    signing_key = ...
)

decrypt_keys = DecryptionMaterials(
    decryption_key = ...,
    verification_key = ...
)

static_cmp = StaticCryptographicMaterialsProvider(
    encryption_materials=encrypt_keys
    decryption_materials=decrypt_keys
)
```

仕組み

静的プロバイダーは、指定した暗号化キーと署名キーを項目エンクリプタに渡します。ここで、これらのアイテムは、テーブル項目の暗号化と署名に直接使用されます。各項目に異なるキーを指定しない限り、すべての項目で同じキーが使用されます。



暗号化マテリアルを取得する

このセクションでは、暗号化マテリアルのリクエストを受け取る際の静的マテリアルプロバイダー (静的 CMP) の入力、出力、処理の詳細について説明します。

入力 (アプリケーションから)

- 暗号化キー — これは、などの対称キーであることが必要です。Advanced Encryption Standard(AES) キー。
- 署名キー — これは、対称キーまたは非対称key pair です。

入力 (項目エンクリプタから)

- DynamoDB 暗号化コンテキスト (p. 10)

出力 (項目エンクリプタへ)

- 入力として渡される暗号化キー。
- 入力として渡される署名キー。
- 実際のマテリアル記述: -リクエストされたマテリアル説明 (p. 10)。存在する場合は、変更されません。

復号マテリアルを取得する

このセクションでは、復号マテリアルのリクエストを受け取る際の静的マテリアルプロバイダー (静的 CMP) の入力、出力、処理の詳細について説明します。

暗号化マテリアルの取得と、復号マテリアルの取得のための異なるメソッドが含まれていますが、動作は同じです。

入力 (アプリケーションから)

- 暗号化キー — これは、などの対称キーであることが必要です。Advanced Encryption Standard(AES) キー。
- 署名キー — これは、対称キーまたは非対称key pair です。

入力 (項目エンクリプタから)

- DynamoDB 暗号化コンテキスト (p. 10)(未使用)

出力 (項目エンクリプタへ)

- 入力として渡される暗号化キー。
- 入力として渡される署名キー。

Amazon DynamoDB 暗号化クライアント使用可能なプログラミング言語

Amazon DynamoDB 暗号化クライアントは以下のプログラミング言語で使用できます。言語固有のライブラリはさまざまですが、結果として得られる実装は相互運用ができます。たとえば、Java クライアントで項目を暗号化 (および署名) し、Python クライアントで項目を復号することができます。

詳細については、該当するトピックを参照してください。

トピック

Java 用 Amazon DynamoDB 暗号化クライアント

このトピックでは、Java 用 Amazon DynamoDB 暗号化クライアントをインストールして使用する方法について説明します。DynamoDB 暗号化クライアントによるプログラミングの詳細については、Java の例 (p. 37)、例() aws-dynamodb-encryption-java リポジトリ GitHub、およびJavadocDynamoDB 暗号化クライアント用

> Note
>
> バージョン 1.x。xJava 用 Amazon DynamoDB 暗号化クライアントはend-of-support 段階 (p. 2)2022 年 7 月の発効。できるだけ早く新しいバージョンにアップグレードしてください。

トピック

前提条件

Java 用 Amazon DynamoDB 暗号化クライアントをインストールする前に、以下の前提条件が満たされていることを確認してください。

Java 開発環境

Java 8 以降が必要になります。Oracle のウェブサイトで Java SE のダウンロードに移動し、Java SE Development Kit (JDK) をダウンロードして、インストールします。

Oracle JDK を使用する場合は、Java Cryptography Extension (JCE) 無制限強度の管轄ポリシーファイルをダウンロードして、インストールする必要があります。

AWS SDK for Java

Amazon DynamoDB 暗号化クライアントでは、アプリケーションが DynamoDB とやり取りしていない場合でも、AWS SDK for Java の DynamoDB モジュールが必要です。SDK 全体またはこのモジュールだけをインストールできます。Maven を使用している場合は、`aws-java-sdk-dynamodb` を `pom.xml` ファイルに追加します。

AWS SDK for Java のインストールと設定についての詳細は、AWS SDK for Java を参照してください。

インストール

Java 用 Amazon DynamoDB 暗号化クライアントは、以下の方法でインストールできます。

手動

Java 用 Amazon DynamoDB 暗号化クライアントをインストールするには、aws-dynamodb-encryption-java GitHub リポジトリ。

Apache Maven の使用

Java 用 Amazon DynamoDB 暗号化クライアントは、以下の依存定義を使用して、Apache Maven を介して利用できます。

```
<dependency>
  <groupId>com.amazonaws</groupId>
  <artifactId>aws-dynamodb-encryption-java</artifactId>
  <version>version-number</version>
</dependency>
```

SDK をインストールしたら、このガイドと、このガイドとDynamoDB 暗号化クライアント Javadocオン GitHub。

Java 用 Amazon DynamoDB 暗号化クライアントの使用方法

このトピックでは、Java での Amazon DynamoDB 暗号化クライアントの機能の一部について説明します。他のプログラミング言語には実装されていない機能も含まれます。

DynamoDB 暗号化クライアントを使用したプログラミングの詳細については、Java の例 (p. 37)、例のaws-dynamodb-encryption-java repositoryオン GitHub、およびJavadocDynamoDB 暗号化クライアント用。

トピック

- 項目エンクリプタ AttributeEncryptor DynamoDBEncryptor (p. 33)
- 保存動作の設定 (p. 34)
- Java の属性アクション (p. 34)
- テーブル名の上書き (p. 36)

項目エンクリプタ AttributeEncryptor DynamoDBEncryptor

Java の DynamoDB 暗号化クライアントには、2 つがあります。項目エンクリプタ (p. 9): 下位レベルDynamoDBEncryptorとAttributeEncryptor (p. 33)。

`AttributeEncryptor` は、DynamoDB 暗号化クライアントで `DynamoDB Encryptor` を使用して AWS SDK for Java で DynamoDBMapper を使用するのに役立つヘルパークラスです。`DynamoDBMapper` で `AttributeEncryptor` を使用すると、項目の保存時に項目が透過的に暗号化および署名されます。また、項目のロード時に項目が透過的に検証および復号されます。

保存動作の設定

「」を使用できます。AttributeEncryptorそしてDynamoDBMapperテーブル項目を追加または暗号化および署名された属性を持つテーブル項目を追加または置換します。これらのタスクでは、次の例に示すように、PUT 保存動作を使用するよう設定することをお勧めします。そのように設定しない場合、データを復号できないことがあります。

```
DynamoDBMapperConfig mapperConfig =
 DynamoDBMapperConfig.builder().withSaveBehavior(SaveBehavior.PUT).build();
DynamoDBMapper mapper = new DynamoDBMapper(ddb, mapperConfig, new
 AttributeEncryptor(encryptor));
```

テーブル項目でモデル化された属性のみを更新するデフォルトの保存動作を使用する場合、テーブル書き込みによって変更されることはありません。その結果、すべての属性を後で読み取るときに、モデル化されていない属性が含まれていないため、署名は検証されません。

また、CLOBBER 保存動作を使用することもできます。この動作は、オプティミスティックロックを無効にしてテーブルの項目を上書きするという点を除いて、PUT 保存動作と同じです。

署名エラーを防ぐために、DynamoDB 暗号化クライアントは、次の場合にランタイム例外をスローします。AttributeEncryptorと併用されるDynamoDBMapperの保存動作が設定されていないCLOBBERまたはPUT。

例で使用されているこのコードを確認するには、DynamoDBMapper の使用 (p. 39)とAwsKmsEncryptedObjectjava例から始めることができます。aws-dynamodb-encryption-javarepository GitHub。

Java の属性アクション

属性アクション (p. 9)では、暗号化されて署名された属性値、署名のみされた属性値、無視される属性値を指定します。属性アクションの指定に使用するメソッドは、DynamoDBMapper および AttributeEncryptor、または下位レベルの DynamoDBEncryptor の使用有無によって異なります。

> **Important**
>
> 属性アクションを使用してテーブル項目を暗号化した後、データモデルから属性を追加または削除すると、署名の検証エラーが発生し、データの復号ができなくなることがあります。詳細な説明については、「データモデルの変更 (p. 50)」を参照してください。

DynamoDBMapper の属性アクション

DynamoDBMapper および AttributeEncryptor を使用する場合は、注釈を使用して属性アクションを指定します。DynamoDB 暗号化クライアントは標準の DynamoDB 属性の注釈を使用して、属性を保護する方法を決定する属性のタイプを定義します。デフォルトでは、プライマリキーを除く属性がすべて暗号化されます。これらの属性は署名されますが、暗号化はされません。

> **Note**
>
> 属性値を暗号化しないでください。@DynamoDBVersionAttribute注釈。ただし、署名することはできます (署名する必要があります)。それ以外の場合、その値を使用する条件によって、意図しない結果をもたらす場合があります。

```
// 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")
```

例外を指定するには、Java 用 Amazon DynamoDB 暗号化クライアントに定義されている暗号化注釈を使用します。クラスレベルで指定した場合は、クラスのデフォルト値になります。

```
// Sign only
@DoNotEncrypt

// Do nothing; not encrypted or signed
@DoNotTouch
```

たとえば、これらの注釈で署名するが、PublicationYear 属性を暗号化しない場合は、ISBN 属性値を暗号化または署名しないでください。

```
// Sign only (override the default)
@DoNotEncrypt
@DynamoDBAttribute(attributeName="PublicationYear")

// Do nothing (override the default)
@DoNotTouch
@DynamoDBAttribute(attributeName="ISBN")
```

DynamoDBEncryptor の属性アクション

DynamoDBEncryptor を使用する際に属性アクションを直接指定するには、名前と値のペアで属性名と指定されたアクションを表している HashMap オブジェクトを作成します。

属性アクションの有効な値は、列挙型の EncryptionFlags で定義されています。ENCRYPT と SIGN を一緒に使用したり、SIGN を単独で使用したりできます。また、両方除外することもできます。ただし、ENCRYPT を単独で使用すると、DynamoDB 暗号化クライアントはエラーをスローします。未署名の属性を暗号化することはできません。

```
ENCRYPT
SIGN
```

> **Warning**
>
> プライマリキー属性を暗号化しないでください。DynamoDB でテーブル全体のスキャンを実行せずに項目を見つけられるように、プレーンテキストの状態を維持する必要があります。

暗号化コンテキストでプライマリキーを指定し、いずれかのプライマリキー属性の属性アクションで ENCRYPT を指定した場合、DynamoDB 暗号化クライアントは例外をスローします。

たとえば、次の Java コードは、actions HashMap すべての属性を暗号化して署名しますrecord商品。例外は、署名されているが暗号化されていないパーティションキー属性とソートキー属性、および署名または暗号化されていない test 属性です。

```
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;
  }
}
```

その後、DynamoDBEncryptor の encryptRecord メソッドを呼び出すときに、attributeFlags パラメータの値としてマップを指定します。たとえば、この encryptRecord の呼び出しでは、actions マップが使用されます。

```
// Encrypt the plaintext record
final Map<String, AttributeValue> encrypted_record = encryptor.encryptRecord(record,
 actions, encryptionContext);
```

テーブル名の上書き

DynamoDB 暗号化クライアントでは、DynamoDB テーブルの名前は、暗号化メソッドおよび復号メソッドに渡される DynamoDB 暗号化コンテキスト (p. 10)の要素です。テーブル項目を暗号化または署名すると、テーブル名を含む DynamoDB 暗号化コンテキストが暗号化テキストに暗号でバインドされます。復号メソッドに渡される DynamoDB 暗号化コンテキストが、暗号化メソッドに渡された DynamoDB 暗号化コンテキストと一致しない場合、復号オペレーションは失敗します。

テーブルをバックアップする場合や、テーブルを実行する場合など、テーブルの名前が変更されることがあります。point-in-time リカバリ。これらの項目の署名を復号または検証する際、元のテーブル名を含む、項目の暗号化と署名に使用されたのと同じ DynamoDB 暗号化コンテキストを渡す必要があります。現在のテーブル名は必要ありません。

DynamoDBEncryptor を使用する場合、DynamoDB 暗号化コンテキストを手動で組み立てます。ただし、DynamoDBMapper を使用している場合は、AttributeEncryptor によって現在のテーブル名を含む DynamoDB 暗号化コンテキストが作成されます。異なるテーブル名で暗号化コンテキストを作成するよう AttributeEncryptor に指示するには、EncryptionContextOverrideOperator を使用します。

たとえば、次のコードは、暗号化マテリアルプロバイダー (CMP) と DynamoDBEncryptor のインスタンスを作成します。次に、DynamoDBEncryptor の setEncryptionContextOverrideOperator メソッドを呼び出します。これは、1 つのテーブル名を上書きする overrideEncryptionContextTableName 演算子を使用します。このように設定すると、AttributeEncryptor によって oldTableName の代わりに newTableName を含む DynamoDB 暗号化コンテキストが作成されます。詳しい例については、を参照してください。EncryptionContextOverridesWithDynamoDBMapper.java。

```
final DirectKmsMaterialProvider cmp = new DirectKmsMaterialProvider(kms, keyArn);
final DynamoDBEncryptor encryptor = DynamoDBEncryptor.getInstance(cmp);

encryptor.setEncryptionContextOverrideOperator(EncryptionContextOperators.overrideEncryptionContextTabl
                oldTableName, newTableName));
```

項目を復号および検証する DynamoDBMapper の load メソッドを呼び出す際、元のテーブル名を指定します。

```
mapper.load(itemClass, DynamoDBMapperConfig.builder()

 .withTableNameOverride(DynamoDBMapperConfig.TableNameOverride.withTableNameReplacement(oldTableName))
                .build());
```

また、複数のテーブル名を上書きする `overrideEncryptionContextTableNameUsingMap` 演算子を使用することもできます。

テーブル名の上書き演算子は通常、データの復号と署名の検証に使用されます。ただし、それらの演算子を使用して、暗号化および署名時に DynamoDB 暗号化コンテキスト内のテーブル名を別の値に設定することができます。

`DynamoDBEncryptor` を使用している場合は、テーブル名の上書き演算子を使用しないでください。代わりに、元のテーブル名で暗号化コンテキストを作成し、復号メソッドに送信します。

DynamoDB 暗号化クライアント for Java のサンプルコード

次の例では、DynamoDB 暗号化クライアントfor Java を使用して、アプリケーションの DynamoDB テーブル項目を保護する方法について説明します。その他の例については、を参照してください (独自の貢献など)例のディレクトリaws-dynamodb-encryption-JavaGitHub のリポジトリ。

トピック

- DynamoDBEncryptor の使用 (p. 37)
- DynamoDBMapper の使用 (p. 39)

DynamoDBEncryptor の使用

この例では、下位レベルの使用方法について説明します。DynamoDBEncryptorとDirect KMS プロバイダー (p. 13)。Direct KMS プロバイダーは、AWS KMS keyにAWS Key Management Service(AWS KMS) を指定します。

任意の互換性を使用できます。暗号化マテリアルプロバイダー (p. 8)(CMP) と`DynamoDBEncryptor`、Direct KMS プロバイダを`DynamoDBMapper`そしてAttributeEncryptor (p. 33)。

完全なコードサンプルの参照: AwsKmsEncryptedItem.java

ステップ 1: Direct KMS プロバイダーを作成する

指定されたリージョンを使用して AWS KMS クライアントのインスタンスを作成します。次に、クライアントインスタンスを使用して、任意の Direct KMS プロバイダーのインスタンスを作成します。AWS KMS key。

この例では、Amazon リソースネーム (ARN) を使用して、AWS KMS keyただし、を使用できます。任意の有効なキー識別子。

```
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);
```

ステップ 2: 項目を作成する

この例では、サンプルテーブル項目を表す `record` HashMap を定義します。

```
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"));
```

ステップ 3: DynamoDBEncryptor を作成する

Direct KMS プロバイダーを使用して `DynamoDBEncryptor` のインスタンスを作成します。

```
final DynamoDBEncryptor encryptor = DynamoDBEncryptor.getInstance(cmp);
```

ステップ 4: DynamoDB 暗号化コンテキストを作成する

-DynamoDB 暗号化コンテキスト (p. 10)には、テーブル構造に関する情報と、暗号化および署名方法が含まれています。`DynamoDBMapper` を使用する場合は、`AttributeEncryptor` で暗号化テキストが作成されます。

```
final String tableName = "testTable";

final EncryptionContext encryptionContext = new EncryptionContext.Builder()
    .withTableName(tableName)
    .withHashKeyName(partitionKeyName)
    .withRangeKeyName(sortKeyName)
    .build();
```

ステップ 5: 属性アクションオブジェクトを作成します

属性アクション (p. 9)では、暗号化されて署名された項目の属性値、署名のみされた項目の属性値、暗号化も署名もされていない項目の属性値を指定します。

Java で属性アクションを指定するには、属性名と `EncryptionFlags` 値のペアの HashMap を作成します。

たとえば、以下の Java コードでは、`actions` 項目のすべての属性を暗号化して署名する `record` HashMap を作成します。ただし、署名済みだが暗号化されていないパーティションキーおよびソートキー属性、暗号化されていない未署名の `test` 属性は除きます。

```
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;
  }
```

```
}
```

ステップ 6: 項目を暗号化して署名する

テーブル項目を暗号化して署名するには、`encryptRecord` のインスタンスで `DynamoDBEncryptor` メソッドを呼び出します。テーブル項目 (`record`)、属性アクション (`actions`)、暗号化テキスト (`encryptionContext`) を指定します。

```
final Map<String, AttributeValue> encrypted_record = encryptor.encryptRecord(record,
 actions, encryptionContext);
```

ステップ 7: 項目を DynamoDB テーブルに配置する

最後に、暗号化された署名済みの項目を DynamoDB テーブルに追加します。

```
final AmazonDynamoDB ddb = AmazonDynamoDBClientBuilder.defaultClient();
ddb.putItem(tableName, encrypted_record);
```

DynamoDBMapper の使用

次の例は、DynamoDB マッパーヘルパークラスとDirect KMS プロバイダー (p. 13)。Direct KMS プロバイダーは、AWS KMS keyにAWS Key Management Service(AWS KMS) を指定します。

任意の互換性を使用できます。暗号化マテリアルプロバイダー (p. 8)(CMP) と`DynamoDBMapper`で直通 KMS プロバイダを下位レベルで使用できる`DynamoDBEncryptor`。

完全なコードサンプルの参照: AwsKmsEncryptedObject.java

ステップ 1: Direct KMS プロバイダーを作成する

指定されたリージョンを使用して AWS KMS クライアントのインスタンスを作成します。次に、クライアントインスタンスを使用して、任意の Direct KMS プロバイダーのインスタンスを作成します。AWS KMS key。

この例では、Amazon リソースネーム (ARN) を使用して、AWS KMS keyただし、を使用できます。任意の有効なキー識別子。

```
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);
```

ステップ 2: DynamoDB 暗号化と DynamoDBMapper を作成する

前のステップで作成した Direct KMS プロバイダーを使用して、DynamoDB Encryptor (p. 33)。DynamoDB マッパーを使用するには、下位レベルの DynamoDB 暗号化をインスタンス化する必要があります。

次に、DynamoDB データベースのインスタンスとマッパー設定を作成し、それらを使用して DynamoDB マッパーのインスタンスを作成します。

> Important
>
> を使用する場合`DynamoDBMapper`署名済み (または暗号化および署名済み) アイテムを追加または編集するには、次のように設定します。保存動作を使用する (p. 34)または`PUT`次の例

に示すように、すべての属性が含まれます。そのように設定しない場合、データを復号できないことがあります。

```
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));
```

ステップ 3: DynamoDB テーブルの定義

次に、DynamoDB テーブルを定義します。注釈を使用して、属性アクション (p. 34)を指定します。この例では、DynamoDB テーブルを作成します。`ExampleTable`であり、`DataPoJo`テーブル項目を表すクラス。

このサンプルテーブルでは、プライマリキーの属性は署名されますが、暗号化されません。これは、`@DynamoDBHashKey` という注釈が付いた `partition_attribute` に適用されます。また、`@DynamoDBRangeKey` という注釈が付いた `sort_attribute` に適用されます。

`@DynamoDBAttribute` という注釈が付いた属性 (`some numbers` など) は暗号化されて署名されます。例外は、`@DoNotEncrypt`(記号のみ) または`@DoNotTouch`DynamoDB 暗号化クライアントで定義された (暗号化も署名もなし) 暗号化注釈です。たとえば、`leave me` 属性には `@DoNotTouch` 注釈が付いているため、暗号化も署名もされません。

```
@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 + "]";
  }
}
```

ステップ 4: テーブル項目を暗号化して保存する

これで、テーブル項目を作成し、DynamoDB マッパーを使用して項目を保存すると、項目はテーブルに追加される前に自動的に暗号化されて署名されます。

この例では、`record` というテーブル項目を定義しています。この項目がテーブルに保存される前に、`DataPoJo` クラスの注釈に基づいて、その属性は暗号化されて署名されます。この場合、`PartitionAttribute`、`SortAttribute`、`LeaveMe` を除くすべての属性が暗号化されて署名されます。`PartitionAttribute` と `SortAttributes` は署名のみされます。`LeaveMe` 属性は暗号化または署名されていません。

`record` 項目を暗号化して署名し、`ExampleTable` に追加するには、`DynamoDBMapper` クラスの `save` メソッドを呼び出します。DynamoDB マッパーは、`PUT`動作を保存すると、項目は更新されず、代わりに同じプライマリキーで置き換えられます。これにより、確実に署名が一致するようになり、その項目をテーブルからの取得時に復号化できます。

```
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);
```

Python 用 DynamoDB 暗号化クライアント

このトピックでは、Python 用 DynamoDB 暗号化クライアントをインストールして使用する方法について説明します。コードは、にありますaws-dynamodb-encryption-pythonrepository on GitHub、完全版とテスト済み版を含むサンプルコード使用開始に役立つためです。

> Note
>
> バージョン 1.x。xと 2.x。xPython 用 Amazon DynamoDB 暗号化クライアントは、end-of-support 段階 (p. 2)2022 年 7 月、有効になります できるだけ早く新しいバージョンにアップグレードしてください。

トピック

前提条件

Python 用 Amazon DynamoDB 暗号化クライアントをインストールする前に、以下の前提条件が満たされていることを確認してください。

Python のサポートされているバージョン

> Python 用 Amazon DynamoDB 暗号化クライアントバージョン 3.1.0 以降には、Python 3.6 以降が必要です。Python をダウンロードするには、「Python のダウンロード」を参照してください。
>
> Python 用 Amazon DynamoDB 暗号化クライアントの以前のバージョンでは Python 2.7 および Python 3.4 以降がサポートされていますが、最新バージョンの DynamoDB 暗号化クライアントを使用することをお勧めします。

Python 用 pip インストールツール

> Python 3.6 以降には pip が含まれていますが、アップグレードすることもできます。pip のアップグレードまたはインストールの詳細については、pip ドキュメント内のインストールを参照してください。

インストール

以下の例に示すように、pip を使用して Python 用 Amazon DynamoDB 暗号化クライアントをインストールします。

最新バージョンをインストールするには

```
pip install dynamodb-encryption-sdk
```

pip を使用してパッケージをインストールおよびアップグレードする方法の詳細については、「パッケージのインストール」を参照してください。

DynamoDB 暗号化クライアントでは、すべてのプラットフォームで cryptography ライブラリが必要です。pip のすべてのバージョンでは、Windows に cryptography ライブラリがインストールされて構築されます。pip 8.1 以降では、Linux に cryptography がインストールされて構築されます。以前のバージョンの pip を使用していて、Linux 環境に暗号ライブラリを構築するために必要なツールがない場合は、それらを

インストールする必要があります。詳細については、「Building cryptography on Linux」を参照してください。

DynamoDB 暗号化クライアントの最新開発バージョンは、aws-dynamodb-encryption-pythonrepository on GitHub。

DynamoDB 暗号化クライアントをインストールしたら、このガイドの Python コードの例を見ながら開始します。

Python 用 DynamoDB 暗号化クライアントを使用する

このトピックでは、Python の DynamoDB 暗号化クライアントの一部機能について説明します。他のプログラミング言語には実装されていない機能も含まれます。これらの機能は、DynamoDB 暗号化クライアントを最も安全な方法で簡単に使用できるように設計されています。通常とは異なるユースケースを除き、この方法を使用することをお勧めします。

DynamoDB 暗号化クライアントでのプログラミングの詳細については、を参照してください。Python の例 (p. 45)このガイドの「」例GitHub の aws-dynamodb-encryption-python リポジトリにあり、Python ドキュメントDynamoDB 暗号化クライアントの場合。

トピック

- クライアントのヘルパークラス (p. 43)
- TableInfo クラス (p. 44)
- Python の属性アクション (p. 44)

クライアントのヘルパークラス

Python の DynamoDB 暗号化クライアントには、DynamoDB の Boto 3 クラスをミラーリングする複数のクライアントヘルパークラスが含まれています。これらのヘルパークラスでは、次のように、暗号化の追加、既存の DynamoDB アプリケーションへの署名、一般的な問題の回避を簡単に行うことができます。

- 項目のプライマリキーを暗号化できないように、プライマリキーの上書きアクションを AttributeActions (p. 44) オブジェクトを追加するか、`AttributeActions` オブジェクトを使用してプライマリキーを暗号化するようにクライアントに明示的に指示している場合は例外をスローします。`AttributeActions` オブジェクトのデフォルトアクションが `DO_NOTHING` の場合、クライアントのヘルパークラスではプライマリキーのアクションが使用されます。それ以外の場合は、`SIGN_ONLY` を使用します。
- の作成TableInfo オブジェクト (p. 43)を入力し、DynamoDB 暗号化コンテキスト (p. 10)DynamoDB の呼び出しに基づきます。これにより、DynamoDB 暗号化コンテキストが正確になり、クライアントはプライマリキーを識別できます。
- 次のようなSupport メソッドput_itemそしてget_itemを使用すると、DynamoDB テーブルが読み書きされるときにテーブルを透過的に暗号化および復号できます。ただし、`update_item` メソッドはサポートされていません。

クライアントのヘルパークラスを使用します。低レベルの項目エンクリプタ (p. 9)を使用して直接やり取りする必要はありません。項目エンクリプタで高度オプションを設定する必要がある場合を除き、これらのクラスを使用します。

クライアントのヘルパークラスには、以下のものが含まれます。

- EncryptedTableを使用するアプリケーションの場合表DynamoDB のリソースを使用して、一度に 1 つのテーブルを処理します。
- EncryptedResourceを使用するアプリケーションの場合サービスリソースバッチ処理用の DynamoDB のクラス。
- EncryptedClientを使用するアプリケーションの場合下位レベルクライアントDynamoDB の「」

クライアントヘルパークラスを使用するには、DynamoDB を呼び出すアクセス許可が発信者に必要です。DescribeTableターゲットテーブルでの操作。

TableInfo クラス

-TableInfoclass は、DynamoDB テーブルを表すヘルパークラスです。プライマリキーとセカンダリインデックスのフィールドが含まれています。これにより、テーブルに関する正確なリアルタイム情報を簡単に取得できます。

クライアントのヘルパークラス (p. 43)を使用している場合は、`TableInfo` オブジェクトが作成、使用されます。それ以外の場合、オブジェクトを明示的に作成できます。例については、項目エンクリプタを使用する (p. 46) を参照してください。

電話すると`refresh_indexed_attributes`のメソッド`TableInfo`オブジェクトの場合、DynamoDB を呼び出して、オブジェクトのプロパティ値が入力されます。DescribeTableオペレーション. テーブルのクエリは、ハードコーディングのインデックス名よりも信頼性がはるかに高まります。`-TableInfo`クラスには`encryption_context_values`に必要な値を提供するプロパティDynamoDB 暗号化コンテキスト (p. 10)。

♪`refresh_indexed_attributes`メソッドを使用すると、DynamoDB を呼び出すアクセス権限が発信者に必要です。DescribeTableターゲットテーブルでの操作。

Python の属性アクション

属性アクション (p. 9)は、項目の各属性に対して実行するアクションを項目エンクリプタに指示します。属性アクションを Python で指定するには、デフォルトアクションで `AttributeActions` オブジェクトと、特定の属性の例外を作成します。有効な値は、列挙型の `CryptoAction` で定義されています。

> **Important**
>
> 属性アクションを使用してテーブル項目を暗号化した後、データモデルから属性を追加または削除すると、署名の検証エラーが発生し、データの復号ができなくなることがあります。詳細な説明については、「データモデルの変更 (p. 50)」を参照してください。

```
DO_NOTHING = 0
SIGN_ONLY = 1
ENCRYPT_AND_SIGN = 2
```

たとえば、この `AttributeActions` オブジェクトは、すべての属性のデフォルトとして `ENCRYPT_AND_SIGN` を確立し、`ISBN` 属性および `PublicationYear` 属性の例外を指定します。

```
actions = AttributeActions(
    default_action=CryptoAction.ENCRYPT_AND_SIGN,
    attribute_actions={
        'ISBN': CryptoAction.DO_NOTHING,
        'PublicationYear': CryptoAction.SIGN_ONLY
    }
)
```

クライアントのヘルパークラス (p. 43)を使用している場合は、プライマリキー属性の属性アクションを指定する必要はありません。クライアントのヘルパークラスを使用して、プライマリキーを暗号化することはできません。

クライアントのヘルパークラスを使用しておらず、デフォルトアクションが `ENCRYPT_AND_SIGN` の場合は、プライマリキーのアクションを指定する必要があります。プライマリキーに推奨されているアクションは `SIGN_ONLY` です。簡単に行うには、`set_index_keys` メソッドを使用します。このメソッドでは、プライマリキーに SIGN_ONLY、デフォルトアクションの場合には DO_NOTHING が使用されます。

Warning

プライマリキー属性を暗号化しないでください。DynamoDB がテーブル全体のスキャンを実行せずに項目を見つけられるように、プレーンテキストの状態を維持する必要があります。

```
actions = AttributeActions(
    default_action=CryptoAction.ENCRYPT_AND_SIGN,
)
actions.set_index_keys(*table_info.protected_index_keys())
```

Python の DynamoDB 暗号化クライアントのサンプルコード

次の例では、DynamoDB 暗号化クライアント for Python を使用して、アプリケーション内の DynamoDB データを保護する方法について説明します。でより多くの例を見つける (および独自の貢献をする) 例のディレクトリaws-dynamodb-encryption-PythonGitHub のリポジトリ。

トピック

- EncryptedTable クライアントヘルパークラスを使用する (p. 45)
- 項目エンクリプタを使用する (p. 46)

EncryptedTable クライアントヘルパークラスを使用する

次の例は、Direct KMS プロバイダー (p. 13)と`EncryptedTable` クライアントのヘルパークラス (p. 43)。この例では同じものを使用しています。暗号化マテリアルプロバイダ (p. 8)として項目エンクリプタを使用する (p. 46)以下に例を示します。ただし、`EncryptedTable`低レベルと直接やり取りするのではなく、クラス項目エントクリプタ (p. 9)。

これらの例を比較することで、クライアントのヘルパークラスが行う作業を確認できます。これには、DynamoDB 暗号化コンテキスト (p. 10)プライマリキー属性が常に署名されているが暗号化されていないことを確認します。暗号化コンテキストを作成し、プライマリキーを検出するには、クライアントのヘルパークラスでの DynamoDB を呼び出します。DescribeTableオペレーション. このコードを実行するには、このオペレーションを呼び出すアクセス許可が必要です。

完全なコードサンプルの参照: aws_kms_encrypted_table.py

ステップ 1: テーブルの作成

開始するには、テーブル名を指定して、標準 DynamoDB テーブルのインスタンスを作成します。

```
table_name='test-table'
table = boto3.resource('dynamodb').Table(table_name)
```

ステップ 2: 暗号化マテリアルプロバイダーを作成する

選択した暗号化マテリアルプロバイダー (p. 12) (CMP) のインスタンスを作成します。

この例では、Direct KMS プロバイダー (p. 13)を使用していますが、互換性のある CMP を使用することもできます。Direct KMS プロバイダーを作成するには、AWS KMS key。この例では、の Amazon リソースネーム (ARN) を使用します。AWS KMS keyただし、任意の有効なキー識別子を使用できます。

```
kms_key_id='arn:aws:kms:us-
west-2:111122223333:key/1234abcd-12ab-34cd-56ef-1234567890ab'
```

```
kms_cmp = AwsKmsCryptographicMaterialsProvider(key_id=kms_key_id)
```

ステップ 3: 属性アクションオブジェクトを作成する

属性アクション (p. 9)は、項目の各属性に対して実行するアクションを項目エンクリプタに指示します。この例の AttributeActions オブジェクトは、無視される test 属性を除くすべての項目を暗号化し、署名します。

クライアントのヘルパークラスを使用する場合は、プライマリキー属性の属性アクションを指定しないでください。EncryptedTable クラスでは、プライマリキー属性に署名しますが、暗号化しません。

```
actions = AttributeActions(
    default_action=CryptoAction.ENCRYPT_AND_SIGN,
    attribute_actions={'test': CryptoAction.DO_NOTHING}
)
```

ステップ 4: 暗号化されたテーブルを作成する

標準テーブル、Direct KMS プロバイダー、属性アクションを使用して、暗号化されたテーブルを作成します。このステップで設定を完了します。

```
encrypted_table = EncryptedTable(
    table=table,
    materials_provider=kms_cmp,
    attribute_actions=actions
)
```

ステップ 5: テーブル内でプレーンテキスト項目を入力する

あなたが電話したときput_itemでのメソッドencrypted_tableでは、テーブル項目は透過的に暗号化されて署名され、DynamoDB テーブルに追加されます。

まず、テーブル項目を定義します。

```
plaintext_item = {
    'partition_attribute': 'value1',
    'sort_attribute': 55
    'example': 'data',
    'numbers': 99,
    'binary': Binary(b'\x00\x01\x02'),
    'test': 'test-value'
}
```

次に、テーブルにデータを入力します。

```
encrypted_table.put_item(Item=plaintext_item)
```

暗号化された形式で DynamoDB テーブルから項目を取得するには、get_itemでのメソッドtableオブジェクト。復号された項目を取得するには、get_item オブジェクトの encrypted_table メソッドを呼び出します。

項目エンクリプタを使用する

この例では、で直接やり取りする方法について説明します。項目エントクリプタ (p. 9)DynamoDB 暗号化クライアントで、テーブルアイテムを暗号化する場合は、クライアントのヘルパークラス (p. 43)項目暗号化とやり取りします。

この方法を使用する場合、DynamoDB 暗号化コンテキストおよび設定オブジェクトを作成します。`CryptoConfig`) の手動作成。また、1 つの呼び出しで項目を暗号化し、別の呼び出しで DynamoDB テーブルに配置します。これにより、`put_item`DynamoDB 暗号化クライアントを呼び出して、を使用して、構造化データの暗号化および署名を行うことができます。DynamoDB に送信されることはありません。

この例では、Direct KMS プロバイダー (p. 13)を使用していますが、互換性のある CMP を使用することもできます。

完全なコードサンプルの参照: aws_kms_encrypted_item.py

ステップ 1: テーブルの作成

開始するには、テーブル名を指定して、標準 DynamoDB テーブルリソースのインスタンスを作成します。

```
table_name='test-table'
table = boto3.resource('dynamodb').Table(table_name)
```

ステップ 2: 暗号化マテリアルプロバイダーを作成する

選択した暗号化マテリアルプロバイダー (p. 12) (CMP) のインスタンスを作成します。

この例では、Direct KMS プロバイダー (p. 13)を使用していますが、互換性のある CMP を使用することもできます。Direct KMS プロバイダーを作成するには、AWS KMS key。この例では、の Amazon リソースネーム (ARN) を使用します。AWS KMS keyただし、任意の有効なキー識別子を使用できます。

```
kms_key_id='arn:aws:kms:us-
west-2:111122223333:key/1234abcd-12ab-34cd-56ef-1234567890ab'
kms_cmp = AwsKmsCryptographicMaterialsProvider(key_id=kms_key_id)
```

ステップ 3: TableInfo ヘルパークラスを使用する

DynamoDB からテーブルに関する情報を取得するには、TableInfo (p. 43)ヘルパークラス。項目エンクリプタで直接操作する場合は、`TableInfo` インスタンスを作成してそのメソッドを呼び出す必要があります。クライアントのヘルパークラス (p. 43)を使用してこの操作を行うことができます。

`-refresh_indexed_attributes`の方法`TableInfo`を使用します。DescribeTableDynamoDB オペレーションを使用して、テーブルに関するリアルタイムで正確な情報を取得します。この情報には、プライマリキーと、ローカルおよびグローバルセカンダリインデックスが含まれます。`DescribeTable` を呼び出すアクセス許可が発信者に必要です。

```
table_info = TableInfo(name=table_name)
table_info.refresh_indexed_attributes(table.meta.client)
```

ステップ 4: DynamoDB 暗号化コンテキストの作成

-DynamoDB 暗号化コンテキスト (p. 10)テーブル構造に関する情報と、暗号化および署名方法を示します。この例では、項目暗号化とやり取りするため、DynamoDB 暗号化コンテキストを明示的に作成します。-クライアントのヘルパークラス (p. 43)DynamoDB 暗号化コンテキストを作成します。

パーティションキーおよびソートキーを取得するには、TableInfo (p. 43) ヘルパークラスのプロパティを使用できます。

```
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)
)
```

ステップ 5: 属性アクションオブジェクトを作成する

属性アクション (p. 9)は、項目の各属性に対して実行するアクションを項目エンクリプタに指示します。この例の AttributeActions オブジェクトは、署名されているが暗号されていないプライマリキー属性と、無視される test 属性を除き、すべての項目を暗号化し、署名します。

項目エンクリプタを使用して直接やり取りし、デフォルトアクションが ENCRYPT_AND_SIGN の場合、プライマリキーの代替アクションを指定する必要があります。set_index_keys メソッドを使用できます。このメソッドでは、プライマリキーに SIGN_ONLY、デフォルトアクションの場合には DO_NOTHING が使用されます。

プライマリキーを指定するために、この例では、TableInfo (p. 43)オブジェクト。DynamoDB の呼び出しによって設定されます。この技術は、ハードコーディングのプライマリキー名より安全です。

```
actions = AttributeActions(
    default_action=CryptoAction.ENCRYPT_AND_SIGN,
    attribute_actions={'test': CryptoAction.DO_NOTHING}
)
actions.set_index_keys(*table_info.protected_index_keys())
```

ステップ 6: 項目の設定を作成します。

DynamoDB 暗号化クライアントを設定するには、で先ほど作成したオブジェクトを使用します。CryptoConfigテーブル項目の設定。クライアントのヘルパークラスでは、CryptoConfig が作成されます。

```
crypto_config = CryptoConfig(
    materials_provider=kms_cmp,
    encryption_context=encryption_context,
    attribute_actions=actions
)
```

ステップ 7: 項目を暗号化する

このステップでは、項目を暗号化および署名しますが、DynamoDB テーブルには入力されません。

クライアントのヘルパークラスを使用するときに、項目は透過的に暗号化されて署名され、その後、を呼び出すときに DynamoDB テーブルに追加されます。put_itemヘルパークラスのメソッド。項目エンクリプタを直接使用する場合、暗号化および入力アクションは独立しています。

まず、プレーンテキスト項目を作成します。

```
plaintext_item = {
    'partition_attribute': 'value1',
    'sort_key': 55,
    'example': 'data',
    'numbers': 99,
    'binary': Binary(b'\x00\x01\x02'),
    'test': 'test-value'
}
```

次に、その項目を暗号化して署名します。encrypt_python_item メソッドでは、CryptoConfig 設定オブジェクトが必要です。

```
encrypted_item = encrypt_python_item(plaintext_item, crypto_config)
```

ステップ 8: テーブル内で項目を入力する

このステップでは、暗号化された署名済みの項目を DynamoDB テーブルに配置します。

```
table.put_item(Item=encrypted_item)
```

暗号化された項目を表示するには、元の get_item オブジェクトの table メソッドを呼び出します。encrypted_table オブジェクトではありません。これにより、検証および復号せずに、DynamoDB テーブルより項目を取得することができます。

```
encrypted_item = table.get_item(Key=partition_key)['Item']
```

次の画像は、暗号化された署名済みのテーブル項目の例の一部を示します。

暗号化された属性値は、バイナリデータです。プライマリキー属性の名前および値 (partition_attribute および sort_attribute) と、test 属性は、プレーンテキスト形式のままです。また、この出力は、署名を含む属性 (*amzn-ddb-map-sig*) とマテリアル説明属性 (p. 10) (*amzn-ddb-map-desc*) を示します。

```
{
    '*amzn-ddb-map-desc*': Binary(b'\x00\x00\x00\x00\x00\x00\x00\x10amzn-ddb-env-
\x00\x00\x00\xe0AQEBAHhA84wnXjEJdBbBBylRUFcZZK2j7xwh6UyLoL28nQ
+0FAAAAH4wfAYJKoZIhvcNAQcGoG8wbQIBADBoBgkqhkiG9w0BBwEwHgYJYIZIAWUDBAEuMBEEDPeFBydr
izYl0R0C4M7wAK6E1/N/bgTmHI=\x00\x00\x00\x17amzn-ddb-map-signingAlg\x00\x00\x00\nHr
\x00\x00\x00\x11/CBC/PKCS5Padding\x00\x00\x00\x10amzn-ddb-sig-alg\x00\x00\x00\x0el
\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\xd
    'binary': Binary(b'!"\xc5\x92\xd7\x13\x1d\xe8Bs\x9b\x7f\xa8\x8e\x9c\xcf\x10\x
    'example': Binary(b"'b\x933\x9a+s\xf1\xd6a\xc5\xd5\x1aZ\xed\xd6\xce\xe9X\xf0T
    'numbers': Binary(b'\xd5\xa0\\d\xcc\x85\xf5\x1e\xb9-f!\xb9\xb8\x8a\x1aT\xbaq\x
    'partition_attribute': 'value1',
    'sort_attribute': 55,
    'test': 'test-value'
}
```

データモデルの変更

項目を暗号化または復号するたびに、属性アクション (p. 9)これにより、DynamoDB 暗号化クライアントに、暗号化と署名する属性、署名する (ただし、暗号化しない) 属性、無視する属性を指定します。属性アクションは、暗号化された項目に保存されるため、DynamoDB Encryption クライアントは属性アクションを自動的に処理しません。

> **Important**
>
> DynamoDB 暗号化クライアントは、暗号化されていない DynamoDB テーブルデータの暗号化をサポートしていません。

データモデルを変更するたびに、つまり、テーブル項目から属性を追加または削除すると、エラーが発生する危険性があります。指定した属性アクションが、項目のすべての属性で構成されていない場合、その項目は、意図した方法で暗号化および署名されない場合があります。さらに重要な点として、項目の復号時に指定する属性アクションが、項目の暗号化時に指定した属性アクションと異なる場合は、署名検証が失敗する場合があります。

たとえば、項目の暗号化に使用する属性アクションで、`test` 属性に署名するよう指示した場合、その項目の署名には `test` 属性が含まれます。項目の復号に使用する属性アクションが、`test` 属性で構成されていない場合、クライアントは `test` 属性を含まない署名の検証を試みるため、検証は失敗します。

これは、DynamoDB 暗号化クライアントはすべてのアプリケーションの項目に対して同じ署名を計算する必要があるため、複数のアプリケーションが同じDynamoDB 項目の読み取りおよび書き込みを行う場合に特に問題になります。また、属性アクションの変更がすべてのホストに反映される必要があるため、分散アプリケーションでも問題になります。DynamoDB テーブルが 1 つのプロセスで 1 つのホストによってアクセスされる場合でも、ベストプラクティスプロセスを確立すると、プロジェクトが複雑になった場合にエラーを防ぐことができます。

テーブル項目を読み取ることができない署名検証エラーを回避するには、次のガイダンスを使用します。

- 属性の追加 (p. 50)— 新しい属性によって属性アクションが変更される場合は、項目に新しい属性を含める前に属性アクションの変更を完全にデプロイします。
- 属性の削除 (p. 52)— 項目で属性の使用を中止する場合は、属性アクションを変更しないでください。
- アクションの変更 — 属性アクション設定を使用してテーブル項目を暗号化した後は、テーブル内のすべての項目を再暗号化しないと、デフォルトのアクションまたは既存の属性のアクションを安全に変更することはできません。

署名検証エラーは解決が非常に困難な場合があるため、最善の方法は、それらのエラーを回避することです。

トピック

属性の追加

テーブル項目に新しい属性を追加する場合、属性アクションの変更が必要になることがあります。署名検証エラーを回避するために、この変更を 2 ステージのプロセスで実装することをお勧めします。第 2 ステージを開始する前に、第 1 ステージが完了していることを確認します。

1. テーブルの読み取りまたは書き込みを行うすべてのアプリケーションで属性アクションを変更します。これらの変更をデプロイして、更新がすべての送信先ホストに反映されていることを確認します。
2. テーブル項目の新しい属性に値を書き込みます。

この 2 ステージのアプローチでは、すべてのアプリケーションおよびホストに同じ属性アクションが設定され、新しい属性が見つかる前に同じ署名が計算されます。これは、属性のアクションが何もしない (暗号化または署名しない) 場合でも重要です。その理由は、一部の暗号化では暗号化と署名がデフォルトであるためです。

次の例は、このプロセスの第 1 ステージのコードを示しています。新しい項目属性 `link` が追加されます。これには、別のテーブル項目へのリンクが保存されます。このリンクはプレーンテキストのままにする必要があるため、この例では署名のみアクションを割り当てます。この変更を完全にデプロイし、すべてのアプリケーションおよびホストに新しい属性アクションがあることを確認したら、テーブル項目で `link` 属性の使用を開始します。

Java DynamoDB Mapper

`DynamoDB Mapper` と `AttributeEncryptor` を使用すると、デフォルトでは、プライマリキーを除く属性がすべて暗号化されます。これらの属性は署名されますが、暗号化はされません。署名のみアクションを指定するには、`@DoNotEncrypt` 注釈を使用します。

この例では、新しい `link` 属性に `@DoNotEncrypt` 注釈を使用します。

```
@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

下位レベルの DynamoDB エンクリプタでは、属性ごとにアクションを設定する必要があります。この例では、switch 文を使用します。デフォルトは `encryptAndSign` で、パーティションキー、ソートキー、および新しい `link` 属性に例外が指定されています。この例では、リンク属性コードが使用前に完全にデプロイされていない場合、リンク属性の暗号化および署名を行うアプリケーションや、リンク属性の署名のみを行うアプリケーションがあります。

```
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

DynamoDB 暗号化クライアントfor Python では、すべての属性にデフォルトのアクションを指定してから、例外を指定できます。

Python クライアントのヘルパークラス (p. 43)を使用している場合は、プライマリキー属性の属性アクションを指定する必要はありません。クライアントのヘルパークラスを使用して、プライマリキーを暗号化することはできません。ただし、クライアントのヘルパークラスを使用していない場合は、パーティションキーとソートキーに SIGN_ONLY アクションを設定する必要があります。パーティションキーまたはソートキーを誤って暗号化した場合、完全なテーブルスキャンを行わないとデータを復元できません。

この例では、`SIGN_ONLY` アクションを取得する新しい `link` 属性の例外を指定します。

```
actions = AttributeActions(
    default_action=CryptoAction.ENCRYPT_AND_SIGN,
    attribute_actions={
      'example': CryptoAction.DO_NOTHING,
      'link': CryptoAction.SIGN_ONLY
    }
)
```

属性の削除

DynamoDB Encryption Client で暗号化された項目で属性が不要になった場合は、その属性の使用を中止できます。ただし、その属性のアクションを削除または変更しないでください。その削除または変更を行ってから、その属性を持つ項目が見つかった場合、その項目に対して計算された署名は元の署名と一致せず、署名の検証は失敗します。

コードから属性のすべてのトレースを削除したいと思うかもしれませんが、項目を削除するのではなく、項目が使用されなくなったというコメントを追加してください。完全なテーブルスキャンを実行して属性のすべてのインスタンスを削除しても、その属性を持つ暗号化された項目は、設定のどこかでキャッシュされるか、または処理中になる可能性があります。

DynamoDB 暗号化クライアントアプリケーションの問題のトラブルシューティング

このセクションでは、DynamoDB 暗号化クライアントを使用する際に直面する可能性のある問題を示すとともに、その問題の解決方法を提案します。

DynamoDB 暗号化クライアントに関するフィードバックを提供するには、aws-dynamodb-encryption-javaまたはaws-dynamodb-encryption-python GitHub repository。

このドキュメントに関するフィードバックを提供するには、任意のページのフィードバックリンクを使用します。また、問題を報告したり、貢献したりすることもできますaws-dynamodb-encryption-docs、このドキュメントのオープンソースリポジトリ GitHub。

トピック

アクセス が拒否されました

問題: アプリケーションは、必要なリソースへのアクセスを拒否されました。

提案: 必要な権限について学び、アプリケーションを実行するセキュリティコンテキストに追加します。

詳細

DynamoDB 暗号化クライアントライブラリを使用するアプリケーションを実行するには、そのコンポーネントを使用するためのアクセス許可が呼び出し元に必要です。それ以外の場合、必要な要素への発信者のアクセスは拒否されます。

- DynamoDB 暗号化クライアントは、Amazon Web Services (AWS) アカウントを必要とせず、どの AWS サービスにも依存しません。ただし、アプリケーションで AWS を使用している場合、アカウントを使用するために、AWS アカウントおよびアクセス許可を持つユーザーが必要です。
- DynamoDB 暗号化クライアントには Amazon DynamoDB は必要ありません。ただし、クライアントを使用するアプリケーションで DynamoDB テーブルを作成する、テーブルに項目を入力する、またはテーブルから項目を取得する場合、呼び出し元には、AWS アカウント で必要な DynamoDB オペレーションを使用するためのアクセス許可が必要です。詳細については、Amazon DynamoDB デベロッパーガイドのアクセスコントロールのトピックを参照してください。
- アプリケーションがクライアントヘルパークラス (p. 43)Python 用 Amazon DynamoDB 暗号化クライアントで、呼び出し元には、DynamoDB を呼び出すためのアクセス許可が必要ですDescribeTableオペレーション。
- DynamoDB 暗号化クライアントには、は必要ありませんAWS Key Management Service(AWS KMS). ただし、アプリケーションがダイレクト KMS マテリアルプロバイダー (p. 13)、またはそれが最新プロバイダー (p. 21)を使用するプロバイダーストアでAWS KMS、呼び出し元には、を使用するアクセス権限が必要ですAWS KMS GenerateDataKeyそしてDecryptオペレーション。

署名の検証失敗

問題: 署名の検証に失敗するため、アイテムを復号化できません。また、項目は、意図したように暗号化および署名されていない場合があります。

提案: 指定する属性アクションがアイテム内のすべての属性を考慮していることを確認してください。項目を復号する場合は、項目の暗号化に使用するアクションと一致する属性アクションを指定します。

詳細

指定する属性アクション (p. 9)によって、DynamoDB 暗号化クライアントに、暗号化して署名する属性、署名する (ただし、暗号化はしない) 属性、および無視する属性が伝達されます。

指定した属性アクションが、項目のすべての属性で構成されていない場合、その項目は、意図した方法で暗号化および署名されない場合があります。項目の復号時に指定する属性アクションが、項目の暗号化時に指定した属性アクションと異なる場合は、署名検証が失敗する場合があります。これは、分散アプリケーション固有の問題で、新しい属性アクションがすべてのホストに反映されていない可能性があります。

署名の検証エラーは解決が困難です。それらのエラーを防ぐために、データモデルの変更時に追加の対策を講じてください。詳細については、「データモデルの変更 (p. 50)」を参照してください。

古いバージョンのグローバルテーブルに関する問題

問題: 古いバージョンの Amazon DynamoDB グローバルテーブルの項目は、署名の検証に失敗するため復号化できません。

提案: リザーブドレプリケーションフィールドが暗号化または署名されないように、属性アクションを設定します。

詳細

DynamoDB 暗号化クライアントはDynamoDB グローバルテーブル。グローバルテーブルには、を使用することをお勧めします。マルチリージョン KMS キーそして KMS キーをすべてに複製しますAWS リージョングローバルテーブルが複製される場所。

グローバルテーブルから始めるバージョン 2019.11.21では、特別な設定なしで DynamoDB 暗号化クライアントでグローバルテーブルを使用できます。ただし、グローバルテーブルを使用する場合バージョン 2017.11.29では、予約済みレプリケーションフィールドが暗号化または署名されていないことを確認する必要があります。

グローバルテーブルバージョン 2017.11.29 を使用している場合は、次の属性の属性アクションを次のように設定する必要がありますDO_NOTHINGにJava (p. 34)または@DoNotTouchにPython (p. 44)。

- `aws:rep:deleting`
- `aws:rep:updatetime`
- `aws:rep:updateregion`

他のバージョンのグローバルテーブルを使用している場合は、何もする必要はありません。

最新プロバイダーのパフォーマンスが悪い

問題: 特に DynamoDB 暗号化クライアントの新しいバージョンに更新すると、アプリケーションの応答性が低下します。

提案: 調整値 time-to-live 値とキャッシュサイズ

詳細

最新のプロバイダーは、暗号化マテリアルの再利用を制限できるようにすることで、DynamoDB 暗号化クライアントを使用するアプリケーションのパフォーマンスを向上させるように設計されています。アプリケーションの最新プロバイダーを設定するときは、パフォーマンスの向上と、キャッシュと再利用によって生じるセキュリティ上の問題とのバランスを取る必要があります。

DynamoDB 暗号化クライアントの新しいバージョンでは、 time-to-live (TTTTTTT) の値によって、キャッシュされた暗号化マテリアルプロバイダー (CMP) の使用期間が決定されます。TTL により、最新プロバイダーが新しいバージョンの CMP をチェックする頻度も決定されます。

TTL が長すぎると、アプリケーションがビジネスルールやセキュリティ基準に違反する可能性があります。TTL が短すぎると、プロバイダーストアへの頻繁な呼び出しによって、プロバイダーストアがアプリケーションや、サービスアカウントを共有する他のアプリケーションからのリクエストを抑制する可能性があります。この問題を解決するには、レイテンシーと可用性の目標を満たし、セキュリティ基準に準拠する値に TTL とキャッシュサイズを調整します。詳細については、「有効期限 (有効期限) (p. 25)」を参照してください。

Amazon DynamoDB 暗号化クライアントデベロッパーガイドのドキュメント履歴

以下の表は、このドキュメントの大きな変更点をまとめたものです。主要な変更に加えて、その内容の説明と例を改善し、ユーザーから寄せられるフィードバックにも応える目的で、このドキュメントは頻繁に更新されます。重要な変更についての通知を受け取るには、RSS フィードをサブスクライブします。

変更	説明	日付
ドキュメントの変更	AWS Key Management Service の用語であるカスタマーマスターキー (CMK) が、AWS KMS key および KMS キーに置き換えられています。	2021 年 8 月 30 日
新機能	AWS Key Management Service (AWS KMS) マルチリージョンキーがサポートされるようになりました。マルチリージョンキーとは、さまざまな AWS リージョンにある AWS KMS キーであり、キー ID とキーマテリアルが同じであるため、交換して使用できます。	2021 年 6 月 8 日
新しいサンプル	Java で DynamoDBMapper を使用する例を追加しました。	2018 年 9 月 6 日
Python サポート	Java に加えて Python のサポートを追加しました。	2018 年 5 月 2 日
初回リリース	このドキュメントの初回リリース。	2018 年 5 月 2 日

翻訳は機械翻訳により提供されています。提供された翻訳内容と英語版の間で齟齬、不一致または矛盾がある場合、英語版が優先します。