# Red Hat JBoss Enterprise Application Platform 8.0

## JBoss EAP における認証情報のセキュアなストレージ

認証情報ストアに認証情報をセキュアに保存するためのガイド

Last Updated: 2024-02-10

# Red Hat JBoss Enterprise Application Platform 8.0 JBoss EAP における認証情報のセキュアなストレージ

認証情報ストアに認証情報をセキュアに保存するためのガイド

## 法律上の通知



## 概要

認証情報ストアに認証情報をセキュアに保存するためのガイド。

# 目次

# JBOSS EAP ドキュメントへのフィードバック (英語のみ)

エラーを報告したり、ドキュメントを改善したりするには、Red Hat Jira アカウントにログインし、課題を送信してください。Red Hat Jira アカウントをお持ちでない場合は、アカウントを作成するように求められます。

**手順**

1. **このリンクをクリック** してチケットを作成します。
2. **Summary** に課題の簡単な説明を入力します。
3. **Description** に課題や機能拡張の詳細な説明を入力します。問題があるドキュメントのセクションへの URL を含めてください。
4. **Submit** をクリックすると、課題が作成され、適切なドキュメントチームに転送されます。

# 多様性を受け入れるオープンソースの強化

Red Hat では、コード、ドキュメント、Web プロパティーにおける配慮に欠ける用語の置き換えに取り組んでいます。まずは、マスター (master)、スレーブ (slave)、ブラックリスト (blacklist)、ホワイトリスト (whitelist) の 4 つの用語の置き換えから始めます。この取り組みは膨大な作業を要するため、今後の複数のリリースで段階的に用語の置き換えを実施して参ります。詳細は、Red Hat CTO である Chris Wright のメッセージ をご覧ください。

# 第1章 ELYTRON のクレデンシャルとクレデンシャルストア

## 1.1. ELYTRON が提供するクレデンシャルストアのタイプ

Elytron には、クレデンシャルの保存に使用できるデフォルトのクレデンシャルストアタイプが 2 つ ( KeyStoreCredentialStore と PropertiesCredentialStore) 用意されています。JBoss EAP 管理 CLI でクレデンシャルストアを管理することも、Wild Fly Elytron ツールを使用してオフラインで管理することもできます。2 つのデフォルトストアタイプに加えて、独自のカスタムクレデンシャルストアを作成、使用、管理することができます。

### 1.1.1. Elytron の KeyStoreCredentialStore/credential-store

Elytron のすべてのクレデンシャルタイプを Key Store Credential Store に格納できます。**elytron** サブシステムの Key Store Credential Store のリソース名は **credential-store** です。Key Store Credential Store は、Java Development Kit (JDK) の Key Store 実装が提供するメカニズムを使用して、認証情報を保護します。

管理用 CLI で Key Store Credential Store に次のようにアクセスします。

```
/subsystem=elytron/credential-store
```

**関連情報**

- スタンドアロンサーバー用 **credential-store** の作成
- 管理対象ドメイン用 **credential-store** の作成
- WildFly Elytron ツールを使用した **credential-store** の作成
- Creating a **credential-store** using the Bouncy Castle provider
- **credential-store** 属性

### 1.1.2. Elytron の PropertiesCredentialStore/secret-key-credential-store

JBoss EAP を正しく起動するには、特定のセキュアリソースのロックを解除する初期キーが必要です。PropertiesCredentialStore を使用して、この初期シークレットキーを指定し、これらの必要なサーバーリソースのロックを解除します。また、Properties Credential Store を使用して、AES(Advanced Encryption Standard) の秘密鍵の保存をサポートする SecretKeyCredential を保存することもできます。ファイルシステムのパーミッションを使用して、クレデンシャルストアへのアクセスを制限します。アプリケーションサーバーのみにアクセス権を付与して、このクレデンシャルストアへのアクセスを制限するのが理想的です。

PropertiesCredentialStore の **elytron** サブシステムでのリソース名は **secret-key-credential-store** で、管理 CLI で以下のようにアクセスできます。

```
/subsystem=elytron/secret-key-credential-store
```

**関連情報**

- スタンドアロンサーバー用の **secret-key-credential-store** の作成
- WildFly Elytron ツールを使用した **secret-key-credential-store** の作成

- **secret-key-credential-store** 属性

## 1.2. ELYTRON のクレデンシャルタイプ

Elytron では、さまざまなセキュリティーの用途に合わせて以下の 3 種類のクレデンシャルがあり、これらのクレデンシャルを Elytron のクレデンシャルストアの 1 つに保存できます。

**PasswordCredential**

このクレデンシャルタイプでは、プレーンテキスト (暗号化されていない) パスワードを安全に保存できます。パスワードを必要とする JBoss EAP リソースでは、パスワードの機密性を維持するために、プレーンテキストのパスワードの代わりに PasswordCredential への参照を使用します。

**データベースへの接続の例**

```
data-source add ... --user-name=db_user --password=StrongPassword
```

このデータベース接続コマンドの例では、**StrongPassword** とパスワードが表示されます。つまり、他の人もサーバー設定ファイルでパスワードを確認できるということです。

**PasswordCredential を使用したデータベースへの接続例**

```
data-source add ... --user-name=db_user --credential-reference=
{store=exampleKeyStoreCredentialStore, alias=passwordCredentialAlias}
```

データベースへの接続にパスワードではなくクレデンシャルリファレンスを使用した場合に、他の人が確認できるのは設定ファイル内のクレデンシャルリファレンスのみで、パスワードは表示されません。

**KeyPairCredential**

KeyPairCredential には、Secure Shell (SSH) と Public-Key Cryptography Standards (PKCS) の両方のキーペアを使用できます。キーペアには、共有される公開鍵と、特定のユーザーだけが知っている秘密鍵の両方が含まれています。
KeyPairCredential の管理は、Wild Fly Elytron ツールのみで行うことができます。

**SecretKeyCredential**

SecretKeyCredential は Elytron で暗号化式の作成に使用できる Advanced Encryption Standard (AES) キーです。

**関連情報**

- Elytron が提供するクレデンシャルストア
- クレデンシャルストアがサポートする認証情報タイプ

## 1.3. ELYTRON のクレデンシャルストアでサポートされる認証情報タイプ

次の表は、どのクレデンシャルストアがどの認証情報のタイプをサポートしているかを示しています。

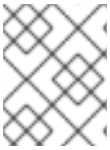
| 認証情報のタイプ | KeyStoreCredentialStore/credential-store | PropertiesCredentialStore/secret-key-credential-store |
|---|---|---|
| PasswordCredential | はい | いいえ |
| KeyPairCredential | はい | いいえ |
| SecretKeyCredential | はい | はい |

**関連情報**

- Elytron のクレデンシャルタイプ
- Elytron が提供するクレデンシャルストア

## 1.4. JBOSS EAP 管理 CLI を使用したクレデンシャルストアの操作

実行中の JBoss EAP サーバーで JBoss EAP クレデンシャルを管理するには、提供されている管理 CLI 操作を使用します。**PasswordCredential** および **SecretKeyCredential** は、JBoss EAP 管理 CLI を使用して管理できます。



**注記**

これらの操作は、変更可能なクレデンシャルストアに対してのみ行うことができます。デフォルトでは、クレデンシャルストアタイプはすべて変更可能です。

### 1.4.1. スタンドアロンサーバー用の credential-store の作成

ファイルシステムの任意のディレクトリーに、スタンドアロンサーバーとして動作する JBoss EAP 用の **credential-store** を作成します。セキュリティーの理由から、このストアを含むディレクトリーは、一部のユーザーのみがアクセスできるようにする必要があります。

**前提条件**

- 最低でも、JBoss EAP が実行されているユーザーアカウントの KeyStoreCredentialStore を含むディレクトリーへの読み取り/書き込みアクセスが割り当てられている。

**注記**

**credential-store** と **secret-key-credential-store** は同じ Elytron 機能 (**org.wildfly.security.credential-store**) を実装しているため、同じ名前を指定できません。

**手順**

- 以下の管理用 CLI コマンドで KeyStoreCredentialStore を作成します。

  **構文**

```
/subsystem=elytron/credential-store=<name_of_credential_store>:add(path="<path_to_store_file>", relative-
```

```
to=<base_path_to_store_file>, credential-reference={clear-text=<store_password>},
create=true)
```

**例**

```
/subsystem=elytron/credential-
store=exampleKeyStoreCredentialStore:add(path="exampleKeyStoreCredentialStore.jceks",
relative-to=jboss.server.data.dir, credential-reference={clear-text=password}, create=true)
{"outcome" => "success"}
```

**関連情報**

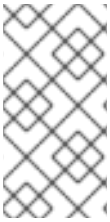
- Elytron の KeyStoreCredentialStore/ **credential-store**
- JBoss EAP 管理 CLI を使用したクレデンシャルストアの操作
- **credential-store** 属性

### 1.4.2. マネージドドメインでの credential-store の作成

マネージドドメインに **credential-store** を作成できますが、最初に Wild Fly Elytron ツールを使用して、KeyStoreCredentialStore を準備する必要があります。1 つのマネージドドメインに複数のホストコントローラーがある場合は、以下のいずれかのオプションを選択します。

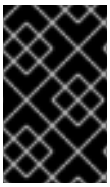
- 各ホストコントローラーに **credential-store** を作成し、各 **credential-store** にクレデンシャルを追加します。
- 1 つのホストコントローラーから他のすべてのホストコントローラーに、入力された **credential-store** をコピーします。
- **credential-store** ファイルをネットワークファイルシステム (NFS) に保存し、作成するすべての **credential-store** リソースにそのファイルを使用します。

また、Wild Fly Elytron ツールを使用せずに、ホストコントローラー上でクレデンシャルを含む **credential-store** ファイルを作成することもできます。



**注記**

同じプロファイルのすべてのサーバーには、**credential-store** ファイルが含まれているため、全サーバーに **credential-store** リソースを定義する必要はありません。**credential-store** ファイルは、サーバー **data** ディレクトリー **relative-to=jboss.server.data.dir** にあります。



**重要**

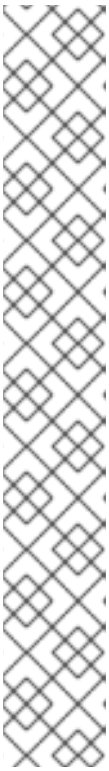
**credential-store** と **secret-key-credential-store** は同じ Elytron 機能 (**org.wildfly.security.credential-store**) を実装しているため、同じ名前を指定できません。

次の手順では、NFS を使用してすべてのホストコントローラーに **credential-store** を提供する方法について説明します。

**手順**

1. WildFly Elytron ツールを使用して、**credential-store** ストレージファイルを作成します。詳細は、WildFly Elytron tool **credential-store** operations を参照してください。

2. ストレージファイルを配布します。たとえば、**scp** コマンドを使用して各ホストコントローラーに割り当てたり、NFS に保存してすべての **credential-store** リソースに使用したりします。



**注記**

複数のリソースとホストコントローラーが使用し、NFS に保存した **credential-store** ファイルの一貫性を維持するには、**credential-store** を読み取り専用モードで使用する必要があります。さらに、**credential-store** ファイルの絶対パスを必ず指定してください。

**構文**

```
/profile=<profile_name>/subsystem=elytron/credential-store=<name_of_credential_store>:add(path=<absolute_path_to_store_keystore>,credential-reference={clear-text="<store_password>"},create=false,modifiable=false)
```

**例**

```
/profile=full-ha/subsystem=elytron/credential-store=exampleCredentialStoreDomain:add(path=/usr/local/etc/example-cred-store.cs,credential-reference={clear-text="password"},create=false,modifiable=false)
```

3. **オプション:**プロファイルに **credential-store** リソースを定義する必要がある場合は、ストレージファイルを使用してリソースを作成します。

**構文**

```
/profile=<profile_name>/subsystem=elytron/credential-store=<name_of_credential_store>:add(path=<path_to_store_file>,credential-reference={clear-text="<store_password>"})
```

**例**

```
/profile=full-ha/subsystem=elytron/credential-store=exampleCredentialStoreHA:add(path=/usr/local/etc/example-cred-store-ha.cs,
credential-reference={clear-text="password"})
```

4. **オプション:** ホストコントローラーの **credential-store** リソースを作成します。

**構文**

```
/host=<host_controller_name>/subsystem=elytron/credential-store=<name_of_credential_store>:add(path=<path_to_store_file>,credential-reference={clear-text="<store_password>"})
```

**例**

```
/host=master/subsystem=elytron/credential-
store=exampleCredentialStoreHost:add(path=/usr/local/etc/example-cred-store-host.cs,
credential-reference={clear-text="password"})
```

**関連情報**

- Elytron の KeyStoreCredentialStore/ **credential-store**
- Credential store operations using the WlidFly Elytron tool
- **credential-store** 属性

### 1.4.3. スタンドアロンサーバー用の secret-key-credential-store の作成

管理 CLI を使用して **secret-key-credential-store** を作成します。**secret-key-credential-store** を作成すると、JBoss EAP はデフォルトでシークレットキーを生成します。生成された鍵の名前は **key** で、そのサイズは 256 ビットです。

**前提条件**

- JBoss EAP が実行されている。
- 少なくとも、JBoss EAP を実行しているユーザーアカウントの **secret-key-credential-store** を格納したディレクトリーへの読み取り/書き込みアクセスが割り当てられている。

**手順**

- 以下のコマンドで、管理 CLI を使用して **secret-key-credential-store** を作成します。

  **構文**

  ```
  /subsystem=elytron/secret-key-credential-
  store=<name_of_credential_store>:add(path="<path_to_the_credential_store>", relative-
  to=<path_to_store_file>)
  ```

  **例**

  ```
  /subsystem=elytron/secret-key-credential-
  store=examplePropertiesCredentialStore:add(path=examplePropertiesCredentialStore.cs,
  relative-to=jboss.server.config.dir)
  {"outcome" => "success"}
  ```

### 1.4.4. credential-store への PasswordCredential の追加

PasswordCredential としてパスワードを必要とするリソースのプレインテキストパスワードを **credential-store** に追加し、そのパスワードを設定ファイルで非表示にします。この保存したクレデンシャルを参照して、パスワードを公開することなく、これらのリソースにアクセスできます。

**前提条件**

- **credential-store** を作成している。

**credential-store** の作成にについて、詳しくは スタンドアロンサーバー用 **credential-store** の作成 を参照してください。

**手順**

- 新しい PasswordCredential を **credential-store** に追加します。

  **構文**

  ```
  /subsystem=elytron/credential-store=<name_of_credential_store>:add-alias(alias=<alias>, secret-value=<secret-value>)
  ```

  **例**

  ```
  /subsystem=elytron/credential-store=exampleKeyStoreCredentialStore:add-alias(alias=passwordCredentialAlias, secret-value=StrongPassword)
  {"outcome" => "success"}
  ```

**検証**

- 次のコマンドを実行して、PasswordCredential が **credential-store** に追加されたことを確認します。

  **構文**

  ```
  /subsystem=elytron/credential-store=<name_of_credential_store>:read-aliases()
  ```

  **例**

  ```
  /subsystem=elytron/credential-store=exampleKeyStoreCredentialStore:read-aliases()
  {
      "outcome" => "success",
      "result" => ["passwordcredentialalias"]
  }
  ```

**関連情報**

- Elytron の KeyStoreCredentialStore/ **credential-store**
- **credential-store** 属性

### 1.4.5. credential-store での SecretKeyCredential の生成

**credential-store** で SecretKeyCredential を生成します。デフォルトでは、Elytron は 256 ビットの鍵を作成します。別のサイズにする場合は、**key-size** 属性に 128 ビットまたは 192 ビットの鍵を指定します。

**前提条件**

- **credential-store** を作成している。
  **credential-store** の作成にについて、詳しくは スタンドアロンサーバー用 **credential-store** の作成 を参照してください。

**手順**

- 以下の管理 CLI コマンドを使用して、**credential-store** に SecretKeyCredential を生成します。

  **構文**

  ```
  /subsystem=elytron/credential-store=<name_of_credential_store>:generate-secret-key(alias=<alias>, key-size=<128_or_192>)
  ```

  **例**

  ```
  /subsystem=elytron/credential-store=exampleKeyStoreCredentialStore:generate-secret-key(alias=secretKeyCredentialAlias)
  ```

**検証**

- 以下のコマンドを実行して、Elytron が SecretKeyCredential を **credential-store** に保存したことを確認します。

  **構文**

  ```
  /subsystem=elytron/credential-store=<name_of_credential_store>:read-aliases()
  ```

  **例**

  ```
  /subsystem=elytron/credential-store=exampleKeyStoreCredentialStore:read-aliases()
  {
      "outcome" => "success",
      "result" => [
          "secretkeycredentialalias"
      ]
  }
  ```

**関連情報**

- Elytron の KeyStoreCredentialStore/ **credential-store**
- **credential-store** 属性

### 1.4.6. secret-key-credential-store で SecretKeyCredential の生成

**secret-key-credential-store** で SecretKeyCredential を生成します。デフォルトでは、Elytron は 256 ビットの鍵を作成します。別のサイズにする場合は、**key-size** 属性に 128 ビットまたは 192 ビットの鍵を指定します。

SecretKeyCredential を生成すると、Elytron は新しいランダムな秘密鍵を生成して SecretKeyCredential として保存します。**secret-key-credential-store** でエクスポート操作を使用して、クレデンシャルの内容を表示できます。

**重要**

JBoss EAP は失われた Elytron クレデンシャルを復号化または取得できないため、**secret-key-credential-store** か SecretKeyCredential、またはその両方のバックアップを必ず作成してください。

**secret-key-credential-store** で **export** 操作を使用して、SecretKeyCredential の値を取得できます。この値をバックアップとして保存できます。

**前提条件**

- **secret-key-credential-store** を作成している。
**secret-key-credential-store** の作成について、詳しくは スタンドアロンサーバー用 **secret-key-credential-store** の作成 を参照してください。

**手順**

- 以下の管理 CLI コマンドを使用して、**secret-key-credential-store** に SecretKeyCredential を生成します。

  **構文**

  ```
  /subsystem=elytron/secret-key-credential-store=<name_of_the_properties_credential_store>:generate-secret-key(alias=<alias>, key-size=<128_or_192>)
  ```

  **例**

  ```
  /subsystem=elytron/secret-key-credential-store=examplePropertiesCredentialStore:generate-secret-key(alias=secretKeyCredentialAlias)
  {"outcome" => "success"}
  ```

**検証**

- 以下のコマンドを実行して、Elytron が SecretKeyCredential を作成したことを確認します。

  **構文**

  ```
  /subsystem=elytron/secret-key-credential-store=<name_of_the_properties_credential_store>:read-aliases()
  ```

  **例**

  ```
  /subsystem=elytron/secret-key-credential-store=examplePropertiesCredentialStore:read-aliases()
  {
      "outcome" => "success",
      "result" => [
          "secretkeycredentialalias",
          "key"
      ]
  }
  ```

**関連情報**

- Elytron の PropertiesCredentialStore/**secret-key-credential-store**
- **secret-key-credential-store** 属性

### 1.4.7. secret-key-credential-store への SecretKeyCredential のインポート

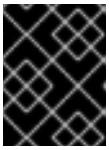
**secret-key-credential-store** 以外で作成された SecretKeyCredential を Elytron の **secret-key-credential-store** にインポートできます。たとえば別のクレデンシャルストア (例: **credential-store**) から SecretKeyCredential をエクスポートした場合、それを **secret-key-credential-store** にインポートできます。

**前提条件**

- **secret-key-credential-store** を作成している。
  **secret-key-credential-store** の作成について、詳しくは スタンドアロンサーバー用 **secret-key-credential-store** の作成 を参照してください。
- Secret Key Credential がエクスポートされました。
  SecretKeyCredential のエクスポートについて、詳しくは **secret-key-credential-store** からの SecretKeyCredential のエクスポート を参照してください。

**手順**

1. 以下のコマンドを使用すると、管理 CLI でコマンドのキャッシュを無効にできます。



**重要**

キャッシングを無効にしない場合は、管理 CLI の履歴ファイルにアクセスできるユーザーは誰でも、秘密鍵を確認できます。

```
history --disable
```

2. 次の管理 CLI コマンドを使用して、秘密鍵をインポートします。

**構文**

```
/subsystem=elytron/secret-key-credential-store=<name_of_credential_store>:import-secret-key(alias=<alias>, key="<secret_key>")
```

**例**

```
/subsystem=elytron/secret-key-credential-store=examplePropertiesCredentialStore:import-secret-key(alias=imported,
key="RUxZAUs+Y1CzEPw0g2AHHOZ+oTKhT9osSabWQtoxR+O+42o11g==")
```

3. 以下の管理 CLI コマンドを使用して、コマンドのキャッシングを再度有効にしてください。

```
history --enable
```

**関連情報**

- Elytron の PropertiesCredentialStore/**secret-key-credential-store**
- **secret-key-credential-store** 属性

### 1.4.8. credential-store 内のクレデンシャルリスト

**credential-store** に保存されているすべてのクレデンシャルを表示するには、管理 CLI を使用してリストを表示できます。

**手順**

- 以下の管理 CLI コマンドを使用して、**credential-store** に保存されているクレデンシャルをリスト表示します。

  **構文**

  ```
  /subsystem=elytron/credential-store=<name_of_credential_store>:read-aliases()
  ```

  **例**

  ```
  {
      "outcome" => "success",
      "result" => [
          "passwordcredentialalias",
          "secretkeycredentialalias"
      ]
  }
  ```

**関連情報**

- Elytron の KeyStoreCredentialStore/ **credential-store**
- **credential-store** 属性

### 1.4.9. credential-store からの SecretKeyCredential のエクスポート

**credential-store** から既存の SecretKeyCredential をエクスポートして、SecretKeyCredential を使用したり、SecretKeyCredential のバックアップを作成したりできます。

**前提条件**

- **credential-store** に SecretKeyCredential を生成している。
  **credential-store** で SecretKeyCredential を生成する方法について、詳しくは **credential-store** での SecretKeyCredential の生成 を参照してください。

**手順**

- 以下の管理 CLI コマンドを使用して、**credential-store** から SecretKeyCredential をエクスポートします。

  **構文**

```
/subsystem=elytron/credential-store=<name_of_credential_store>:export-secret-key(alias=<alias>)
```

**例**

```
/subsystem=elytron/credential-store=exampleKeyStoreCredentialStore:export-secret-key(alias=secretKeyCredentialAlias)
{
    "outcome" => "success",
    "result" => {"key" => "RUxZAUui+8JkoDCE6mFyA3cCIbSAZaXq5wgYejj1scYgdDqWiw=="}
}
```

**関連情報**

- Elytron の KeyStoreCredentialStore/ **credential-store**
- **credential-store** 属性

### 1.4.10. secret-key-credential-store からの SecretKeyCredential のエクスポート

**secret-key-credential-store** から既存の SecretKeyCredential をエクスポートして、SecretKeyCredential を使用したり、SecretKeyCredential のバックアップを作成したりできます。

**前提条件**

- **secret-key-credential-store** で SecretKeyCredential を生成したか、インポートしている。**secret-key-credential-store** で SecretKeyCredential を生成する方法について、詳しくは **secret-key-credential-store**での SecretKeyCredential の生成 を参照してください。

  SecretKeyCredential を **secret-key-credential-store** にインポートする方法について、詳しくは ISecretKeyCredential を **secret-key-credential-store** にインポートする方法 を参照してください。

**手順**

- 以下の管理 CLI コマンドを使用して、**secret-key-credential-store** から SecretKeyCredential をエクスポートします。

  **構文**

```
/subsystem=elytron/secret-key-credential-store=<name_of_credential_store>:export-secret-key(alias=<alias>)
```

  **例**

```
/subsystem=elytron/secret-key-credential-store=examplePropertiesCredentialStore:export-secret-key(alias=secretkeycredentialalias)
{
    "outcome" => "success",
    "result" => {"key" => "RUxZAUtxXcYvz0aukZu+odOynIr0ByLhC72iwzlJsi+ZPmONgA=="}
}
```

**関連情報**

- Elytron の PropertiesCredentialStore/**secret-key-credential-store**
- **secret-key-credential-store** 属性

### 1.4.11. credential-store からのクレデンシャルの削除

**credential-store** にすべてのクレデンシャルタイプを保存できますが、クレデンシャルを削除すると、デフォルトで Elytron は PasswordCredential であると想定します。別のクレデンシャルタイプを削除する場合は、**entry-type** 属性で指定します。

**手順**

- 以下の管理 CLI コマンドを使用して、**credential-store** からクレデンシャルを削除します。

  **構文**

  ```
  /subsystem=elytron/credential-store=<name_of_credential_store>:remove-alias(alias=<alias>, entry-type=<credential_type>)
  ```

  **PasswordCredential の削除例**

  ```
  /subsystem=elytron/credential-store=exampleKeyStoreCredentialStore:remove-alias(alias=passwordCredentialAlias)
  {
      "outcome" => "success",
      "response-headers" => {"warnings" => [{
          "warning" => "Update dependent resources as alias 'passwordCredentialAlias' does not exist anymore",
          "level" => "WARNING",
          "operation" => {
              "address" => [
                  ("subsystem" => "elytron"),
                  ("credential-store" => "exampleKeyStoreCredentialStore")
              ],
              "operation" => "remove-alias"
          }
      }]}
  }
  ```

  **SecretKeyCredential の削除例**

  ```
  /subsystem=elytron/credential-store=exampleKeyStoreCredentialStore:remove-alias(alias=secretKeyCredentialAlias, entry-type=SecretKeyCredential)
  {
      "outcome" => "success",
      "response-headers" => {"warnings" => [{
          "warning" => "Update dependent resources as alias 'secretKeyCredentialAlias' does not exist anymore",
          "level" => "WARNING",
          "operation" => {
              "address" => [
                  ("subsystem" => "elytron"),
  ```

```
                ("credential-store" => "exampleKeyStoreCredentialStore")
            ],
            "operation" => "remove-alias"
        }
    }]}
}
```

**検証**

- 次のコマンドを実行して、Elytron がクレデンシャルを削除したことを確認します。

  **構文**

  ```
  /subsystem=elytron/credential-store=<name_of_credential_store>:read-aliases()
  ```

  **例**

  ```
  /subsystem=elytron/credential-store=exampleKeyStoreCredentialStore:read-aliases()
  {
      "outcome" => "success",
      "result" => []
  }
  ```

  削除したクレデンシャルはリストにありません。

**関連情報**

- Elytron の KeyStoreCredentialStore/ **credential-store**
- **credential-store** 属性

### 1.4.12. secret-key-credential-store からのクレデンシャルの削除

**secret-key-credential-store** には SecretKeyCredential タイプのみ保存できます。つまり、**secret-key-credential-store** からクレデンシャルを削除する時に、**entry-type** を指定する必要はありません。

**手順**

- 次のコマンドを使用して、**secret-key-credential-store** から SecretKeyCredential を削除します。

  **構文**

  ```
  /subsystem=elytron/secret-key-credential-store=<name_of_credential_store>:remove-alias(alias=<alias>)
  ```

  **例**

  ```
  /subsystem=elytron/secret-key-credential-store=examplePropertiesCredentialStore:remove-alias(alias=secretKeyCredentialAlias)
  {
      "outcome" => "success",
      "response-headers" => {"warnings" => [{
  ```

```
        "warning" => "Update dependent resources as alias 'secretKeyCredentialAlias' does not
exist anymore",
        "level" => "WARNING",
        "operation" => {
            "address" => [
                ("subsystem" => "elytron"),
                ("secret-key-credential-store" => "examplePropertiesCredentialSt
ore")
            ],
            "operation" => "remove-alias"
        }
    }]}
}
```

**検証**

- 次のコマンドを実行して、Elytron がクレデンシャルを削除したことを確認します。

  **構文**

  ```
  /subsystem=elytron/secret-key-credential-store=<name_of_credential_store>:read-aliases()
  ```

  **例**

  ```
  /subsystem=elytron/secret-key-credential-store=examplePropertiesCredentialStore:read-
  aliases()
  {
      "outcome" => "success",
      "result" => []
  }
  ```

  削除したクレデンシャルはリストにありません。

**関連情報**

- Elytron の PropertiesCredentialStore/**secret-key-credential-store**
- **secret-key-credential-store** 属性

## 1.5. WILD FLY ELYTRON ツールを使用したクレデンシャルストア操作

WildFly Elytron ツールを使用して、オフラインでクレデンシャルストアのさまざまな操作を実行できます。

### 1.5.1. WildFly Elytron ツールを使用した credential-store の作成

Elytron では、すべてのクレデンシャルタイプを保存できる **credential-store** をオフラインで作成できます。

**手順**

- 以下のコマンドで、WildFly Elytron ツールを使用して **credential-store** を作成します。

**構文**

```
$ EAP_HOME/bin/elytron-tool.sh credential-store --create --location "<path_to_store_file>"
--password <store_password>
```

**例**

```
$ EAP_HOME/bin/elytron-tool.sh credential-store --create --location "../cred_stores/example-
credential-store.jceks" --password storePassword
Credential Store has been successfully created
```

ストアのパスワードをコマンドに含めない場合は、この引数を省略して、プロンプトでパスワードを手動で入力してください。また、Wild Fly Elytron ツールで生成したマスキングされたパスワードを使用することもできます。マスクされたパスワードの生成については、Generating masked encrypted strings using the WildFly Elytron tool を参照してください。

**関連情報**

- Elytron の KeyStoreCredentialStore/ **credential-store**
- WildFly Elytron Tool を使用した、マスクされた文字列の生成
- WildFly Elytron ツールの **credential-store** 操作

### 1.5.2. Bouncy Castle プロバイダーを使用した credential-store の作成

Bouncy Castle プロバイダーを使用して **credential-store** を作成します。

**前提条件**

- お使いの環境が Bouncy Castle を使用するように設定されていることを確認する。

**注記**

**credential-store** と **secret-key-credential-store** は同じ Elytron 機能 (**org.wildfly.security.credential-store**) を実装しているため、同じ名前を指定できません。

**手順**

1. **BCFKS** (Bouncy Castle FIPS Keystore) のキーストアを定義します。FIPS とは Federal Information Processing Standards の略です。このキーストアがすでにある場合は、次のステップに進んでください。

```
$ keytool -genkeypair -alias <key_pair_alias> -keyalg <key_algorithm> -keysize
<key_size> -storepass <key_pair_and_keystore_password> -keystore
<path_to_keystore> -storetype BCFKS -keypass <key_pair_and_keystore_password>
```

**重要**

キーストアの **keypass** 属性と **storepass** 属性が同一であることを確認してください。そうでない場合は、**elytron** サブシステムの **BCFKS** キーストアでは定義できません。

2. **credential-store** のシークレットキーを生成します。

```
$ keytool -genseckey -alias <key_alias> -keyalg <key_algorithm> -keysize <key_size> -
keystore <path_to_keystore> -storetype BCFKS -storepass
<key_and_keystore_password> -keypass <key_and_keystore_password>
```

3. 以下のコマンドで、WildFly Elytron ツールを使用して **credential-store** を定義します。

```
$ EAP_HOME/bin/elytron-tool.sh credential-store -c -a <alias> -x <alias_password> -p
<key_and_keystore_password> -l <path_to_keystore> -u
"keyStoreType=BCFKS;external=true;keyAlias=<key_alias>;externalPath=<path_to_crede
ntial_store>"
```

**関連情報**

- Elytron の KeyStoreCredentialStore/ **credential-store**
- WildFly Elytron ツールの **credential-store** 操作

### 1.5.3. WildFly Elytron ツールを使用した secret-key-credential-store の作成

Elytron では、**secret-key-credential-store** をオフラインで作成し、SecretKeyCredential インスタンスを保存できます。

**手順**

- 以下のコマンドで Wild Fly Elytron ツールを使用して PropertiesCredentialStore を作成します。

  **構文**

```
$ EAP_HOME/bin/elytron-tool.sh credential-store --create --location "<path_to_store_file>"
--type PropertiesCredentialStore
```

  **例**

```
$ bin/elytron-tool.sh credential-store --create --location=standalone/configuration/properties-
credential-store.cs --type PropertiesCredentialStore
Credential Store has been successfully created
```

**関連情報**

- Elytron の PropertiesCredentialStore/**secret-key-credential-store**
- WildFly Elytron ツールの **secret-key-credential-store** 操作

### 1.5.4. WildFly Elytron ツールの credential-store 操作

WildFly Elytron ツールを使用して、次のようなさまざまな **credential-store** タスクを実行できます。

**PasswordCredential の追加**

次の WildFly Elytron ツールコマンドを使用して、PasswordCredential を **credential-store** に追加できます。

**構文**

```
$ EAP_HOME/bin/elytron-tool.sh credential-store --location "<path_to_store_file>" --password <store_password> --add <alias> --secret <sensitive_string>
```

**例**

```
$ EAP_HOME/bin/elytron-tool.sh credential-store --location "../cred_stores/example-credential-store.jceks" --password storePassword --add examplePasswordCredential --secret speci@l_db_pa$$_01
Alias "examplePasswordCredential" has been successfully stored
```

コマンドにシークレットを指定しない場合は、この引数を省略して、プロンプトが表示されたら手動でシークレットを入力します。

### SecretKeyCredential の生成

次の WildFly Elytron ツールコマンドを使用して、SecretKeyCredential を **credential-store** に追加できます。

**構文**

```
$ EAP_HOME/bin/elytron-tool.sh credential-store --generate-secret-key=example --location=<path_to_the_credential_store> --password <store_password>
```

**例**

```
$ EAP_HOME/bin/elytron-tool.sh credential-store --generate-secret-key=example --location "../cred_stores/example-credential-store.jceks" --password storePassword
Alias "example" has been successfully stored
```

コマンドにシークレットを指定しない場合は、この引数を省略して、プロンプトが表示されたら手動でシークレットを入力します。

デフォルトでは、JBoss EAP で SecretKeyCredential を作成すると、256 ビットの秘密鍵が作成されます。サイズを変更する場合は、 **--size=128** または **--size=192** を指定すると、それぞれ 128 ビット、192 ビットの鍵を作成できます。

### SecretKeyCredential のインポート

Secret Key Credential のインポートは、以下の Wild FLy Elytron ツールのコマンドで行うことができます。

**構文**

```
$ EAP_HOME/bin/elytron-tool.sh credential-store --import-secret-key=imported --location=<path_to_credential_store> --password=<store_password>
```

**例**

```
$ EAP_HOME/bin/elytron-tool.sh credential-store --import-secret-key=imported --
location=../cred_stores/example-credential-store.jceks --password=storePassword
```

インポートする秘密鍵を入力します。

#### すべてのクレデンシャルのリスト表示

次の WildFly Elytron ツールコマンドを使用して、**credential-store** 内のクレデンシャルをリスト表示できます。

**構文**

```
$ EAP_HOME/bin/elytron-tool.sh credential-store --location "<path_to_store_file>" --password
<store_password> --aliases
```

**以下に例を示します。**

```
$ EAP_HOME/bin/elytron-tool.sh credential-store --location "../cred_stores/example-credential-
store.jceks" --password storePassword --aliases
Credential store contains following aliases: examplepasswordcredential example
```

#### エイリアスが存在するかどうかの確認

クレデンシャルストアにエイリアスが存在するかどうかを確認するには、次のコマンドを使用します。

**構文**

```
$ EAP_HOME/bin/elytron-tool.sh credential-store --location "<path_to_store_file>" --password
<store_password> --exists <alias>
```

**例**

```
$ EAP_HOME/bin/elytron-tool.sh credential-store --location "../cred_stores/example-credential-
store.jceks" --password storePassword --exists examplepasswordcredential
Alias "examplepasswordcredential" exists
```

#### SecretKeyCredential のエクスポート

次のコマンドを使用して、**credential-store** から SecretKeyCredential をエクスポートできます。

**構文**

```
$ EAP_HOME/bin/elytron-tool.sh credential-store --export-secret-key=<alias> --
location=<path_to_credential_store> --password=storePassword
```

**例**

```
$ EAP_HOME/bin/elytron-tool.sh credential-store --export-secret-key=example --
location=../cred_stores/example-credential-store.jceks --password=storePassword
Exported SecretKey for alias
example=RUxZAUtBiAnoLP1CA+i6DtcbkZHfybBJxPeS9mlVOmEYwjjmEA==
```

#### クレデンシャルの削除

以下のコマンドを使用して、クレデンシャルストアから認証情報を削除できます。

**構文**

```
$ EAP_HOME/bin/elytron-tool.sh credential-store --location "<path_to_store_file>" --password <store_password> --remove <alias>
```

**例**

```
$ EAP_HOME/bin/elytron-tool.sh credential-store --location "../cred_stores/example-credential-store.jceks" --password storePassword --remove examplepasswordcredential
Alias "examplepasswordcredential" has been successfully removed
```

#### 関連情報

- Elytron の KeyStoreCredentialStore/ **credential-store**
- Elytron のクレデンシャルタイプ

### 1.5.5. WildFly Elytron ツールの **secret-key-credential-store** 操作

以下のように、Wild Fly Elytron ツールを使って、SecretKeyCredential に対する **secret-key-credential-store** 操作を実行できます。

#### SecretKeyCredential の生成

次の WildFly Elytron ツールコマンドを使用して、**secteKeyCredential** を **secret-key-credential-store** に生成できます。

**構文**

```
$ EAP_HOME/bin/elytron-tool.sh credential-store --generate-secret-key=example --location "<path_to_the_credential_store>" --type PropertiesCredentialStore
```

**例**

```
$ EAP_HOME/bin/elytron-tool.sh credential-store --generate-secret-key=example --location "standalone/configuration/properties-credential-store.cs" --type PropertiesCredentialStore
Alias "example" has been successfully stored
```

#### SecretKeyCredential のインポート

Secret Key Credential のインポートは、以下の Wild FLy Elytron ツールのコマンドで行うことができます。

**構文**

```
$ EAP_HOME/bin/elytron-tool.sh credential-store --import-secret-key=imported --location=<path_to_credential_store> --type PropertiesCredentialStore
```

**例**

```
$ EAP_HOME/bin/elytron-tool.sh credential-store --import-secret-key=imported --location
"standalone/configuration/properties-credential-store.cs" --type PropertiesCredentialStore
```

#### すべてのクレデンシャルのリスト表示

次の WildFly Elytron ツールコマンドを使用して、**secret-key-credential-store** 内のクレデンシャルをリスト表示できます。

**構文**

```
$ EAP_HOME/bin/elytron-tool.sh credential-store --location "<path_to_store_file>"  --aliases --
type PropertiesCredentialStore
```

**例**

```
$ EAP_HOME/bin/elytron-tool.sh credential-store  --location "standalone/configuration/properties-
credential-store.cs" --aliases --type PropertiesCredentialStore
Credential store contains following aliases: example
```

#### SecretKeyCredential のエクスポート

次のコマンドを使用して、**secret-key-credential-store** から SecretKeyCredential をエクスポートできます。

**構文**

```
$ EAP_HOME/bin/elytron-tool.sh credential-store --export-secret-key=<alias> --location
"<path_to_credential_store>"  --type PropertiesCredentialStore
```

**例**

```
$ EAP_HOME/bin/elytron-tool.sh credential-store --export-secret-key=example --location
"standalone/configuration/properties-credential-store.cs" --type PropertiesCredentialStore
Exported SecretKey for alias
example=RUxZAUt1EZM7PsYRgMGypkGirSeI+5Eix4aSgwop6jfxGYUQaQ==
```

#### クレデンシャルの削除

以下のコマンドを使用して、クレデンシャルストアから認証情報を削除できます。

**構文**

```
$ EAP_HOME/bin/elytron-tool.sh credential-store --location "<path_to_store_file>" --remove
<alias> --type PropertiesCredentialStore
```

**例**

```
$ EAP_HOME/bin/elytron-tool.sh credential-store --location "standalone/configuration/properties-
credential-store.cs"  --remove example --type PropertiesCredentialStore
Alias "example" has been successfully removed
```

#### 関連情報

- Elytron の PropertiesCredentialStore/**secret-key-credential-store**
- Elytron のクレデンシャルタイプ

### 1.5.6. WildFly Elytron Tool で作成された credential-store の JBoss EAP サーバーへの追加

Wild Fly Elytron ツールで **credential-store** を作成したら、実行中の JBoss EAP サーバーに追加できます。

**前提条件**

- Wild Fly Elytron ツールでクレデンシャルストアを作成しておく。
  詳細は、Creating a **credential-store** using the WildFly Elytron tool を参照してください。

**手順**

- 以下の管理 CLI コマンドを使用して、実行中の JBoss EAP サーバーにクレデンシャルストアを追加します。

  **構文**

  ```
  /subsystem=elytron/credential-store=<store_name>:add(location="<path_to_store_file>",credential-reference={clear-text=<store_password>})
  ```

  **例**

  ```
  /subsystem=elytron/credential-store=my_store:add(location="../cred_stores/example-credential-store.jceks",credential-reference={clear-text=storePassword})
  ```

クレデンシャルストアを JBoss EAP 設定に追加することで、**credential-reference 属性** を使用してクレデンシャルストアに保存されているパスワードまたは機密文字列を参照できます。

詳細は、**EAP_HOME/bin/elytron-tool.sh credential-store --help** コマンドを使用して、利用可能なオプションの詳細な一覧を表示してください。

**関連情報**

- **credential-store** 属性

### 1.5.7. Wild Fly Elytron ツールによるキーペア管理の操作

以下の引数を使用して、クレデンシャルストアのエイリアスに保存するキーペアを新規生成するなど、**elytron-tool.sh** を実行してクレデンシャルストアを操作できます。

**キーペアの生成**

**generate-key-pair** コマンドでキーペアを作成します。その後、キーペアをクレデンシャルストアのエイリアスのに保存できます。以下の例では、割り当てられたサイズが3072ビットのRSAキーペアを作成し、クレデンシャルストアに指定された場所に保存しています。キーペアに割り当てられたエイリアスは **example** です。

```
$ EAP_HOME/bin/elytron-tool.sh credential-store --location=<path_to_store_file> --generate-key-pair example --algorithm RSA --size 3072
```

**キーペアのインポート**

既存の SSH キーペアを、指定したエイリアスでクレデンシャルストアにインポートするには、**import-key-pair** コマンドを使用します。以下の例では、Open SSH 形式の秘密鍵を含む**/home/user/.ssh/id_rsa** ファイルから**example**というエイリアスのキーペアをインポートしています。

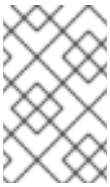
```
$ EAP_HOME/bin/elytron-tool.sh credential-store --import-key-pair example --private-key-location /home/user/.ssh/id_rsa --location=<path_to_store_file>
```

**キーペアのエクスポート**

キーペアの公開鍵を表示するには、**export-key-pair-public-key** コマンドを使用します。公開鍵には、OpenSSH 形式のエイリアスが指定されています。以下の例では、エイリアス**example**の公開鍵を表示しています。

```
$ EAP_HOME/bin/elytron-tool.sh credential-store --location=<path_to_store_file> --export-key-pair-public-key example

Credential store password:
Confirm credential store password:
ecdsa-sha2-nistp256
AAAAE2VjZHNhLXNoYTItbmlzdHAyNTYAAAAIbmlzdHAyNTYAAABBBMfncZuHmR7uglb0M96ieArRFtp42xPn9+ugukbY8dyjOXoi
cZrYRyy9+X68fylEWBMzyg+nhjWkxJlJ2M2LAGY=
```



**注記**

**export-key-pair-public-key** コマンドを実行すると、クレデンシャルストアのパスフレーズの入力が求められます。パスフレーズが存在しない場合は、プロンプトを空白にします。

### 1.5.8. Elytron の設定ファイルに保存されたキーペアの使用例

キーペアは対応する 2 つの個別の暗号化キー (公開鍵と秘密鍵) で構成されます。キーペアをクレデンシャルストアに保存してから、**elytron** の設定ファイルでキーペアを参照する必要があります。これにより、スタンドアロンサーバーの設定データ管理用のアクセス権を Git に割り当てることができます。

以下の例では、**elytron** の設定ファイルの **<credential-stores>** 要素でクレデンシャルストアとそのプロパティーを参照しています。**<credential>** 要素は、クレデンシャルストアとキーペアを格納するエイリアスを参照します。

```
<?xml version="1.0" encoding="UTF-8"?>
<configuration>
  <authentication-client xmlns="urn:elytron:client:1.6">

    <credential-stores>
      <credential-store name="${credential_store_name}">
        <protection-parameter-credentials>
          <clear-password password="${credential_store_password}"/>
```

```
        </protection-parameter-credentials>
        <attributes>
          <attribute name="path" value="${path_to_credential_store}"/>
        </attributes>
      </credential-store>
    </credential-stores>

    <authentication-rules>
      <rule use-configuration="${configuration_file_name}"/>
    </authentication-rules>

    <authentication-configurations>
      <configuration name="${configuration_file_name}">
        <credentials>
          <credential-store-reference store="${credential_store_name}" alias="${alias_of_key_pair}"/>
        </credentials>
      </configuration>
    </authentication-configurations>

  </authentication-client>
</configuration>
```

**elytron** 設定ファイルの設定後に、そのキーペアを SSH 認証に使用できます。

**関連情報**

- Wild Fly Elytron ツールによるキーペア管理の操作

### 1.5.9. WildFly Elytron Tool を使用した、マスクされた文字列の生成

WildFly Elytron ツールで、クレデンシャルストアのプレーンテキストパスワードの代わりに使用する暗号化した文字列を生成できます。

**手順**

- マスクされた文字列を生成するには、以下のコマンドを使用して salt および iteration count の値を指定します。

  ```
  $ EAP_HOME/bin/elytron-tool.sh mask --salt <salt> --iteration <iteration_count> --secret <password>
  ```

  以下に例を示します。

  ```
  $ EAP_HOME/bin/elytron-tool.sh mask --salt 12345678 --iteration 123 --secret supersecretstorepassword

  MASK-8VzWsSNwBaR676g8ujiIDdFKwSjOBHCHgnKf17nun3v;12345678;123
  ```

  コマンドでシークレットを指定しない場合は、その引数を省略し、標準入力を使用して手動でシークレットを入力するように求められます。

詳細は、**EAP_HOME/bin/elytron-tool.sh mask --help** コマンドを使用して、利用可能なオプションの詳細な一覧を表示してください。

## 1.6. クレデンシャルストアのクレデンシャルの自動更新

クレデンシャルストアがある場合には、クレデンシャル参照から取得する前に、クレデンシャルを追加したり、既存のクレデンシャルを更新したりする必要はありません。Elytron はこのプロセスを自動化します。クレデンシャル参照の設定時には、**store** 属性と **clear-text** 属性の両方を指定します。Elytron は、**store** 属性で指定されたクレデンシャルストアにクレデンシャルを自動的に追加または更新します。オプションで、**エイリアス** 属性を指定すできます。

Elytron はクレデンシャルストアを以下のように更新する。

- エイリアスを指定した場合:
  - エイリアスのエントリーが存在する場合に、既存のクレデンシャルは指定のクリアテキストのパスワードに置き換えられます。
  - エイリアスのエントリーが存在しない場合は、指定のエイリアスとクリアテキストのパスワードを含む新しいエントリーがクレデンシャルストアに追加されます。
- エイリアスが指定されない場合、Elytron はエイリアスを生成して生成されたエイリアスと指定されたクリアテキストパスワードが指定されたエントリーを新たに、クレデンシャルストアに追加します。

クレデンシャルストアが更新されると、**clear-text** 属性は管理モデルから削除されます。

次の例は、**store**、**clear-text**、**alias** の各属性を指定したクレデンシャル参照を作成する方法を示しています。

```
/subsystem=elytron/key-store=exampleKS:add(relative-to=jboss.server.config.dir,
path=example.keystore, type=JCEKS, credential-reference=
{store=exampleKeyStoreCredentialStore, alias=myNewAlias, clear-text=myNewPassword})
{
    "outcome" => "success",
    "result" => {"credential-store-update" => {
        "status" => "new-entry-added",
        "new-alias" => "myNewAlias"
    }}
}
```

以前に定義されたクレデンシャルストアに追加された **myNewAlias** エントリーのクレデンシャルを更新するには、次のコマンドを使用します。

```
/subsystem=elytron/key-store=exampleKS:write-attribute(name=credential-reference.clear-
text,value=myUpdatedPassword)
{
    "outcome" => "success",
    "result" => {"credential-store-update" => {"status" => "existing-entry-updated"}},
    "response-headers" => {
        "operation-requires-reload" => true,
        "process-state" => "reload-required"
    }
}
```

**注記**

**credential-reference** パラメーターを含む操作に失敗した場合、クレデンシャルストアの自動更新は行われません。

**credential-reference** 属性で指定されていたクレデンシャルストアは変更されません。

## 1.7. ELYTRON クライアントでのクレデンシャルストアの使用例

Jakarta Enterprise Beans などの JBoss EAP に接続するクライアントは、Elytron クライアントを使用して認証できます。実行中の JBoss EAP サーバーにアクセスできないユーザーは、WildFly Elytron ツールを使用してクレデンシャルストアを作成および変更できます。そのため、クライアントは Elytron クライアントを使用してクレデンシャルストア内の機密情報にアクセスできます。

以下の例は、Elytron クライアント設定ファイルでクレデンシャルストアを使用する方法を示しています。

**クレデンシャルストアを含む custom-config.xml の例**

```
<configuration>
  <authentication-client xmlns="urn:elytron:client:1.7">
    ...
    <credential-stores>
      <credential-store name="my_store"> 1
        <protection-parameter-credentials>
          <credential-store-reference clear-text="pass123"/> 2
        </protection-parameter-credentials>
        <attributes>
          <attribute name="location" value="/path/to/my_store.jceks"/> 3
        </attributes>
      </credential-store>
    </credential-stores>
    ...
    <authentication-configurations>
      <configuration name="my_user">
        <set-host name="localhost"/>
        <set-user-name name="my_user"/>
        <set-mechanism-realm name="ManagementRealm"/>
        <use-provider-sasl-factory/>
        <credentials>
          <credential-store-reference store="my_store" alias="my_user"/> 4
        </credentials>
      </configuration>
    </authentication-configurations>
    ...
  </authentication-client>
</configuration>
```

1. Elytron クライアント設定ファイル内で使用するクレデンシャルストアの名前。
2. クレデンシャルストアのパスワード。
3. クレデンシャルストアファイルへのパス。
4. クレデンシャルストアに保存される機密文字列のクレデンシャル参照。

**関連情報**

- WildFly Elytron ツールを使用したクレデンシャルストアの操作。

## 1.8. FIPS 140-2 準拠のクレデンシャルストアの作成

Elytron で、Federal Information Processing Standard (FIPS) 140-2 に準拠したクレデンシャルストアを設定できます。FIPS 140-2 は、米国が開発したコンピューターセキュリティー規格です。暗号化モジュールの品質を検証する政府および業界の作業グループ。FIPS の出版物 (140-2 を含む) は、http://csrc.nist.gov/publications/PubsFIPS.html にあります。

Elytron では、2 つの異なるプロバイダーを使用して FIPS 140-2 に準拠したクレデンシャルストアを設定できます。

- SunPKCS#11 プロバイダーと Network Security Services (NSS) データベース。
  詳細は、SunPKCS#11 プロバイダーおよび NSS データベースを使用した FIPS 140-2 準拠クレデンシャルストアの作成 を参照してください。

- BouncyCastle プロバイダー。
  詳細は、BouncyCastle プロバイダーを使用した FIPS 140-2 準拠クレデンシャルストアの作成 を参照してください。

**重要**

JBoss EAP 自体は FIPS 認定を受けていません。JBoss EAP の FIPS 対応のレベルは、JBoss EAP を FIPS 認定を受けた暗号化実装で使用できることです。テスト済みの実装は、BouncyCastle と SunPKCS#11 です。

### 1.8.1. SunPKCS#11 プロバイダーおよび NSS データベースを使用した FIPS 140-2 準拠クレデンシャルストアの作成

NSS は、クロスプラットフォームセキュリティー対応のクライアントおよびサーバーアプリケーションをサポートする一連のライブラリーです。JBoss EAP の NSS ライブラリーで SunPKCS#11 プロバイダーを使用して、FIPS 140-2 準拠の暗号化を実装できます。NSS の詳細は、Mozilla docs - Network Security Services (NSS) を参照してください。SunPKCS#11 プロバイダーの詳細は、PKCS#11 Reference Guide を参照してください。

#### 1.8.1.1. SunPKCS#11 プロバイダーと NSS データベースを使用した場合に FIPS をサポートする JDK

Federal Information Processing Standard (FIPS) 140-2 に準拠するためには、NSS ライブラリーによって実装される Network Security Services (NSS) ソフトウェアトークンを使用して SunPKCS#11 セキュリティープロバイダーを設定する必要がありますが、すべての Java Development Kit (JDK) ベンダーがこの設定をサポートしているわけではありません。JBoss EAP で SunPKCS#11 プロバイダーおよび NSS データベースを使用して FIPS を設定する前に、JDK がそれに対応していることを確認してください。

以下は、SunPKCS#11 セキュリティーの設定に対応している JBoss EAP のサポート対象 JDK の一覧です。

- OpenJDK 11

- OpenJDK 17

**関連情報**

- FIPS を使用した RHEL での OpenJDK 11 の設定
- FIPS を使用した RHEL での OpenJDK 17 の設定

#### 1.8.1.2. FIPS 対応 RHEL で SUNPKCS#11 プロバイダーと NSS データベースを使用して FIPS 140-2 準拠のクレデンシャルストアを作成する方法

Red Hat Enterprise Linux 8.4 以降、Federal Information Processing Standard (FIPS) システム全体の暗号化ポリシーを有効にすると、FIPS for Java も自動的に有効になります。デフォルトの Network Security Services (NSS) データベースを使用して、FIPS 140-2 準拠のクレデンシャルストアを作成できます。

この手順では、**$JAVA_HOME** は JDK インストールパスを参照します。この手順のコマンドを root ユーザーとして実行します。

**前提条件**

- RHEL で FIPS が有効になっている。
  以下のコマンドを使用して、FIPS が有効化されているか確認できます。

  ```
  # fips-mode-setup --check
  ```

  RHEL で FIPS を有効にする方法は、以下のリソースを参照してください。

  - Red Hat Enterprise Linux ドキュメントの FIPS モードでのシステムのインストール
  - Red Hat Enterprise Linux ドキュメントの FIPS モードへのシステムの切り替え

- NSS ツールがインストールされている。
  Red Hat Enterprise Linux では、以下のように DNF パッケージマネージャーを使用して NSS ツールをインストールできます。

  ```
  # dnf install -y nss-tools
  ```

- Java Development Kit (JDK) は、NSS ライブラリーを使用した PKCS#11 の設定をサポートしています。
  FIPS 対応 JDK の詳細は、FIPS 対応 JDK を参照してください。

- JBoss EAP が実行されている。

**手順**

1. **$JAVA_HOME/conf/security/nss.fips.cfg** ファイルの **nssDbMode** の値を **readWrite** に更新します。

   **nss.fips.cfg コンテンツの例**

   ```
   name = NSS-FIPS
   nssLibraryDirectory = /usr/lib64
   nssSecmodDirectory = sql:/etc/pki/nssdb
   nssDbMode = readWrite
   ```

```
nssModule = fips

attributes(*,CKO_SECRET_KEY,CKK_GENERIC_SECRET)={ CKA_SIGN=true }
```

2. AES シークレットキーを生成してクレデンシャルストアを暗号化します。

**注記**

コマンドでストアパスワード **NONE** を使用する必要があります。

**構文**

```
# keytool -genseckey -keystore NONE -storetype PKCS11 -storepass NONE -alias
<key_alias> -keyalg <symmetric_key_algorithm> -keysize <key_size>
```

**例**

```
# keytool -genseckey -keystore NONE -storetype PKCS11 -storepass NONE -alias
exampleKeyAlias -keyalg AES -keysize 256
```

3. シークレットキーの読み取りが可能か検証します。

```
# keytool -list -storetype pkcs11 -storepass NONE

Keystore type: PKCS11
Keystore provider: SunPKCS11-NSS-FIPS

Your keystore contains 1 entry

exampleKeyAlias, SecretKeyEntry,
```

4. **$JAVA_HOME/conf/security/nss.fips.cfg** ファイルの **nssDbMode** の値を **readOnly** に更新します。

**nss.fips.cfg コンテンツの例**

```
name = NSS-FIPS
nssLibraryDirectory = /usr/lib64
nssSecmodDirectory = sql:/etc/pki/nssdb
nssDbMode = readOnly
nssModule = fips

attributes(*,CKO_SECRET_KEY,CKK_GENERIC_SECRET)={ CKA_SIGN=true }
```

5. 管理 CLI のプロバイダーリストに SunJCE プロバイダーを追加します。

a. SunJCE のプロバイダーローダーを追加します。

```
/subsystem=elytron/provider-loader=SunJCE:add(class-names=
[com.sun.crypto.provider.SunJCE])
{"outcome" => "success"}
```

b. アグリゲートプロバイダーで Elytron および SunJCE を設定します。

**構文**

```
/subsystem=elytron/aggregate-providers=<aggregate_provider_name>:add(providers=
[elytron,SunJCE])
```

**例**

```
/subsystem=elytron/aggregate-providers=exampleAggregateProvider:add(providers=
[elytron,SunJCE])
{"outcome" => "success"}
```

**注記**

プロバイダーはコマンドで定義されている順序で呼び出されます。

6. SunJCE プロバイダーを使用して Elytron でクレデンシャルストアを作成します。

**構文**

```
/subsystem=elytron/credential-store=<credential_store_name>:add(implementation-
properties={keyStoreType => PKCS11, external => true, keyAlias => <key_alias>,
externalPath => <path_where_credential_store_is_to_be_saved>}, modifiable=true,
credential-reference={clear-text=<password>}, create=true, other-
providers=<aggregate_provider_name>)
```

**例**

```
/subsystem=elytron/credential-store=exampleFipsCredentialStore:add(implementation-
properties={keyStoreType => PKCS11, external => true, keyAlias => exampleKeyAlias,
externalPath => /home/ashwin/example.store}, modifiable=true, credential-reference={clear-
text=secret}, create=true, other-providers=exampleAggregateProvider)
{"outcome" => "success"}
```

**検証**

1. クレデンシャルストアにエイリアスを追加します。

**構文**

```
/subsystem=elytron/credential-store=<credential_store_name>:add-alias(alias=<alias>,
secret-value=<secret_value>)
```

**例**

```
/subsystem=elytron/credential-store=exampleFipsCredentialStore:add-
alias(alias=exampleAlias, secret-value=secret)
{"outcome" => "success"}
```

2. クレデンシャルストアのエイリアスを一覧表示します。

**構文**

```
/subsystem=elytron/credential-store=<credential_store_name>:read-aliases()
```

**例**

```
/subsystem=elytron/credential-store=exampleFipsCredentialStore:read-aliases()
{
    "outcome" => "success",
    "result" => ["examplealias"]
}
```

作成されたクレデンシャルストアは FIPS 140-2 に準拠しています。

**関連情報**

- **aggregate-providers** 属性
- **credential-store** 属性
- **credential-store** 実装プロパティー

## 1.8.2. BouncyCastle プロバイダーを使用した FIPS 140-2 準拠クレデンシャルストアの作成

BouncyCastle は、Java および C# 用の軽量な暗号化 API を提供します。JBoss EAP で次の BouncyCastle プロバイダーを使用すると、FIPS 140-2 準拠のクレデンシャルストアを作成できます。

- Java Cryptography Extension (JCE) および Java Cryptography Architecture (JCA) 用の BouncyCastle FIPS プロバイダー。
- Java Secure Socket Extension (JSSE) 用の BouncyCastle FIPS プロバイダー。

BouncyCastle については、The Legion of the Bouncy Castle を参照してください。

### 1.8.2.1. BouncyCastle プロバイダーを使用した FIPS 140-2 準拠クレデンシャルストアの作成

Red Hat Enterprise Linux 8.4 以降、Federal Information Processing Standard (FIPS) のシステム全体の暗号化ポリシーを有効にすると、OpenJDK によってさまざまなセキュリティープロバイダーが自動的に有効になります。セキュリティープロバイダーの 1 つは、FIPS モードで設定された SunPKCS11 プロバイダーです。その代わりに BouncyCastle プロバイダーを使用する場合は、以下の手順に従って Federal Information Processing Standard (FIPS) 140-2 準拠のクレデンシャルストアを作成できます。

**前提条件**

- RHEL で FIPS が有効になっている。
  以下のコマンドを使用して、FIPS が有効化されているか確認できます。

  ```
  # fips-mode-setup --check
  ```

  RHEL で FIPS を有効にする方法は、以下のリソースを参照してください。

  - Red Hat Enterprise Linux ドキュメントの FIPS モードでのシステムのインストール
  - Red Hat Enterprise Linux ドキュメントの FIPS モードへのシステムの切り替え

- お使いの Java Development Kit (JDK) が、BouncyCastle プロバイダーを使用した FIPS の設定をサポートしている。
  詳細は、The Legion of the Bouncy Castle - FIPS Resources Page の Java Related Questions を参照してください。

**手順**

1. 次のリンクから BouncyCastle jar をダウンロードします。
   - bc-fips-**N** jar: Bouncy Castle Provider (FIPS Distribution) Maven
   - bctls-fips-**N** jar: Bouncy Castle TLS/JSSE APIs (FIPS Distribution)
     **N** は BouncyCastle FIPS プロバイダーのバージョンを表します。

     ご使用の環境に適合する BouncyCastle FIPS プロバイダーの最新認定バージョンに関する情報は、The Legion of the Bouncy Castle を参照してください。
2. 次の内容を含む **java.security** という設定ファイルを作成します。

```
fips.provider.1=org.bouncycastle.jcajce.provider.BouncyCastleFipsProvider
fips.provider.2=org.bouncycastle.jsse.provider.BouncyCastleJsseProvider fips:BCFIPS
fips.provider.3=SUN
fips.provider.4=SunEC
fips.provider.5=com.sun.net.ssl.internal.ssl.Provider
```

**注記**

デフォルトの **java.security** ファイル内の FIPS プロバイダーを変更しないでください。この手順で説明されているように、独自の **java.security** プロパティーファイルを使用することを推奨します。

3. AES シークレットキーを生成してクレデンシャルストアを暗号化します。

   **構文**

```
$ keytool -J-Djava.security.properties=<java_security_file> -genseckey -keystore
"<keystore_name>" -storetype BCFKS -storepass <store_password> -alias <key_alias> -
keyalg <symmetric_key_algorithm> -keysize <key_size> -keypass <key_password>  -
provider org.bouncycastle.jcajce.provider.BouncyCastleFipsProvider -providerpath
<path_to_bc-fips_jar>  -dname "<certificate_contents>" -validity <validity_in_days>
```

   **例**

```
$ keytool -J-Djava.security.properties=<path_to_java_security_file>/java.security -
genseckey -keystore "examplekeystore.bcfks" -storetype BCFKS -storepass password -alias
exampleKeyAlias -keyalg AES -keysize 256 -keypass password -provider
org.bouncycastle.jcajce.provider.BouncyCastleFipsProvider -providerpath <path_to_bc-
fips_jar>/bc-fips-1.0.2.jar -dname "CN=localhost" -validity 365
```

4. シークレットキーの読み取りが可能か検証します。

   **構文**

```
$ keytool -J-Djava.security.properties=<java_security_file> -list -keystore
<keystore_name> -storetype BCFKS -storepass <store_password> -provider
org.bouncycastle.jcajce.provider.BouncyCastleFipsProvider -providerpath <path_to_bc-
fips_jar>
```

**例**

```
$ keytool -J-Djava.security.properties=<path_to_java_security_file>/java.security -list -
keystore examplekeystore.bcfks -storetype BCFKS -storepass password -provider
org.bouncycastle.jcajce.provider.BouncyCastleFipsProvider -providerpath <path_to_bc-
fips_jar>/bc-fips-1.0.2.jar
```

**出力例**

```
Keystore type: BCFKS
Keystore provider: BCFIPS

Your keystore contains 1 entry

exampleKeyAlias, Mar 1, 2023, SecretKeyEntry,
```

5. サービスを起動します。
6. 管理 CLI を使用して、BouncyCastle プロバイダーを使用するように JBoss EAP を設定します。
   a. SunJCE プロバイダーをプロバイダーのリストに追加します。

   ```
   /subsystem=elytron/provider-loader=SunJCE:add(class-names=
   [com.sun.crypto.provider.SunJCE])
   ```

   b. BouncyCastle プロバイダー jar を JBoss EAP のモジュールとして追加します。

   **構文**

   ```
   module add --name=org.bouncycastle.fips --resources=<path_to_bc-
   fips_jar>:<path_to_bctls-fips_jar>
   ```

   **例**

   ```
   module add --name=org.bouncycastle.fips --resources=<path_to_bc-fips_jar>/bc-fips-
   1.0.2.jar:<path_to_bctls-fips_jar>/bctls-fips-1.0.2.jar
   ```

   c. BouncyCastle プロバイダーのプロバイダーローダーを追加します。

   **構文**

   ```
   /subsystem=elytron/provider-
   loader=<provider_loader_name>:add(module=org.bouncycastle.fips)
   ```

   **例**

```
/subsystem=elytron/provider-
loader=exampleProviderLoader:add(module=org.bouncycastle.fips)
```

d. アグリゲートプロバイダーに、BouncyCastle、SunJCE、および combined-providers を設定します。

**構文**

```
/subsystem=elytron/aggregate-providers=<aggregate_provider_name>:add(providers=
[<provider_loader_name>,SunJCE,combined-providers])
```

**例**

```
/subsystem=elytron/aggregate-providers=exampleAggregateProvider:add(providers=
[exampleProviderLoader,SunJCE,combined-providers])
```

**注記**

プロバイダーはコマンドで定義されている順序で呼び出されます。

e. サーバーをリロードします。

```
reload
```

7. BouncyCastle プロバイダーを使用して、Elytron にクレデンシャルストアを作成します。

**構文**

```
/subsystem=elytron/credential-store=<credential_store_name>:add(credential-reference=
{clear-text=<key_and_keystore_password>},implementation-properties=
{keyAlias=<key_alias>,external=true,externalPath=<path_to_BCFKS_credential_store>,ke
yStoreType=BCFKS},create=true,path=<path_to_keystore>,modifiable=true, other-
providers=<aggregate_provider_name>)
```

**例**

```
/subsystem=elytron/credential-store=exampleFipsCredentialStore:add(credential-reference=
{clear-text=password},implementation-properties={keyAlias=exampleKeyAlias,external=true,
externalPath=credentialStore.bcfks, keyStoreType=BCFKS}, create=true,
path=__<path_to_keystore>__/examplekeystore.bcfks, modifiable=true, other-
providers=exampleAggregateProvider)
```

**検証**

1. クレデンシャルストアにエイリアスを追加します。

**構文**

```
/subsystem=elytron/credential-store=<credential_store_name>:add-alias(alias=<alias>,
secret-value=<secret_value>)
```

**例**

```
/subsystem=elytron/credential-store=exampleFipsCredentialStore:add-
alias(alias=exampleAlias, secret-value=secret)
```

2. クレデンシャルストアのエイリアスを一覧表示します。

**構文**

```
/subsystem=elytron/credential-store=<credential_store_name>:read-aliases()
```

**例**

```
/subsystem=elytron/credential-store=exampleFipsCredentialStore:read-aliases()
{
    "outcome" => "success",
    "result" => ["examplealias"]
}
```

作成されたクレデンシャルストアは FIPS 140-2 に準拠しています。

**関連情報**

- **aggregate-providers** 属性
- **credential-store** 属性
- **credential-store** 実装プロパティー
- **provider-loader** 属性
- **module add** コマンドの詳細は、JBoss EAP 管理 CLI で **--help** コマンドを実行して確認できます。

```
module add --help
```

# 第2章 セキュリティーで保護されたリソースを解除するための初期キーの JBOSS EAP への指定

## 2.1. ELYTRON の暗号化式

機密な文字列を秘匿するには、サーバー設定ファイルで機密な字列を指定する代わりに、暗号化表現を使用できます。

暗号化式とは、文字列を SecretKeyCredential で暗号化し、その文字列をエンコーディングの接頭辞とリゾルバー名と組み合わせたものです。エンコーディング接頭辞は、Elytron に式が暗号化されていることを伝えます。リゾルバーは、暗号化式をクレデンシャルストア内で対応する SecretKeyCredential にマッピングします。

Elytron の **expression=encryption** リソースは、暗号化式を使用して、実行時にその中の暗号化された文字列を復号します。設定ファイルの中で、機密な文字列そのものではなく、暗号化式を使用して文字列の機密性を守ることができます。暗号化式は、以下のような形式になります。

**特定のリゾルバーを使用する場合の構文**

```
${ENC::RESOLVER_NAME:ENCRYPTED_STRING}
```

**ENC** は、暗号化式を表す接頭語です。

**RESOLVER_NAME** は、Elytron が暗号化された文字列の復号時に使用するリゾルバーです。

**例**

```
${ENC::initialresolver:RUxZAUMQE+L5zx9LmCRLyh5fjdfl1WM7lhfthKjeoEU+x+RMi6s=}
```

デフォルトのリゾルバーで暗号化式を作成すると、以下のようになります。

**デフォルトのリゾルバーを使用する場合の構文**

```
${ENC::ENCRYPTED_STRING}
```

**例**

```
${ENC::RUxZAUMQE+L5zx9LmCRLyh5fjdfl1WM7lhfthKjeoEU+x+RMi6s=}
```

この場合、Elytron は **expression=encryption** リソースで定義したデフォルトのリゾルバーを使用して式を復号化します。暗号化式は、対応の全リソース属性で使用できます。ある属性が暗号化式をサポートしているかどうかを調べるには、**read-resource-description** 操作などを使用します。

**例: mail/mail-session の read-resource-description**

```
/subsystem=mail/mail-session=*/:read-resource-description(recursive=true,access-control=none)
{
  "outcome"=>"success",
  "result"=>[{
  ...
    "from"=>{
      ...
```

```
        "expression-allowed"=>true,
        ...
    }]
}
```

この例では、**from** 属性が暗号化式をサポートしています。つまり、暗号化して、暗号化式を使用し、**from** フィールドのメールアドレスを非表示にできます。

**関連情報**

- Elytron での暗号化式の作成
- **expression=encryption** 属性

## 2.2. ELYTRON での暗号化式の作成

機密な文字列と SecretKeyCredential から、暗号化式を作成します。この暗号化式を、管理モデルであるサーバー設定ファイルの機密文字列の代わりに使用すして、機密文字列を秘匿できます。

**前提条件**

- PropertiesCredentialStore とその中に秘密鍵を作成している。
  詳細は、スタンドアロンサーバー用 PropertiesCredentialStore/**secret-key-credential-store** の作成 を参照してください。

**手順**

1. 次の管理 CLI コマンドを使用して、クレデンシャルストア内の既存の SecretKeyCredential のエイリアスを参照するリゾルバーを作成します。

   **構文**

   ```
   /subsystem=elytron/expression=encryption:add(resolvers=
   [{name=<name_of_the_resolver>, credential-store=<name_of_credential_store>, secret-
   key=<secret_key_alias>}])
   ```

   **例**

   ```
   /subsystem=elytron/expression=encryption:add(resolvers=[{name=exampleResolver,
   credential-store=examplePropertiesCredentialStore, secret-key=key}])
   ```

   リソースの重複に関するエラーメッセージが表示された場合は、以下のように、**add** ではなく **list-add** 操作を行ってください。

   **構文**

   ```
   /subsystem=elytron/expression=encryption:list-add(name=resolvers, value=
   {name=<name_of_the_resolver>, credential-store=<name_of_credential_store>, secret-
   key=<secret_key_alias>})
   ```

   **例**

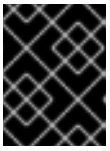
   ```
   /subsystem=elytron/expression=encryption:list-add(name=resolvers,value=
   ```

```
{name=exampleResolver, credential-store=examplePropertiesCredentialStore, secret-key=key})
{
    "outcome" => "success",
    "response-headers" => {
        "operation-requires-reload" => true,
        "process-state" => "reload-required"
    }
}
```

2. サーバーをリロードします。

```
reload
```

3. 管理 CLI でのコマンドのキャッシングを無効にします。



**重要**

キャッシングを無効にしない場合は、管理 CLI の履歴ファイルにアクセスできるユーザーは誰でも、秘密鍵を確認できます。

```
history --disable
```

4. 以下の管理 CLI コマンドを使用して、暗号化式を作成します。

**構文**

```
/subsystem=elytron/expression=encryption:create-expression(resolver=<existing_resolver>, clear-text=<sensitive_string_to_protect>)
```

**例**

```
/subsystem=elytron/expression=encryption:create-expression(resolver=exampleResolver, clear-text=TestPassword)
{
    "outcome" => "success",
    "result" => {"expression" => "${ENC::exampleResolver:RUxZAUMQgtpG7oFlHR2j1Gkn3GKlHff+HR8GcMX1QXHvx2uGurI=}"}
}
```

**${ENC::example Resolver:RUx ZAUMQgtp G7o Fl HR2j1Gkn3GKlHff+HR8Gc MX1QXHvx2u Gur I=}** は、管理モデルで **Test Password** の代わりに使用する暗号化式です。

同じプレーンテキストをさまざまな場所で使用する場合は、対処の場所でプレーンテキストの代わりに、毎回このコマンドを繰り返してから暗号化式を使用します。同じプレーンテキストに対して同じコマンドを繰り返すと、同じキーでも異なる結果が得られます。これは、Elytron が呼び出しごとに固有の初期化ベクトルを使用しているためです。

異なる暗号化式を使用して、文字列の暗号化式 1 つが侵害された場合に、他の暗号化式にも同じ文字列が含まれていることをユーザーが発見できないようにします。

5. 以下の管理 CLI コマンドを使用して、コマンドキャッシングを再度有効にします。

```
history --enable
```

**関連情報**

- 暗号化式を使用した KeyStoreCredentialStore/**credential-store**の保護
- **expression=encryption** 属性

## 2.3. 暗号化式を使用した KEYSTORECREDENTIALSTORE/CREDENTIAL-STORE の保護

Key Store Credential Store を保護するために、暗号化式を使用できます。

**前提条件**

- 暗号化式が作成されている。
  暗号化式の作成について、詳しくは Elytron での暗号化式の作成 を参照してください。

**手順**

- 暗号化式を **clear-text** として使用する KeyStoreCredentialStore を作成します。

  **構文**

  ```
  /subsystem=elytron/credential-store=<name_of_credential_store>:add(path=<path_to_the_credential_store>, create=true, modifiable=true, credential-reference={clear-text=<encrypted_expression>})
  ```

  **例**

  ```
  /subsystem=elytron/credential-store=secureKeyStoreCredentialStore:add(path="secureKeyStoreCredentialStore.jceks", relative-to=jboss.server.data.dir, create=true, modifiable=true, credential-reference={clear-text=${ENC::exampleResolver:RUxZAUMQgtpG7oFlHR2j1Gkn3GKlHff+HR8GcMX1QXHvx2uGurl=}})
  {"outcome" => "success"}
  ```

**関連情報**

- **expression=encryption** 属性
- **credential-store** 属性

KeyStoreCredentialStore を暗号化式で保護した後、KeyStoreCredentialStore で **SecretKeyCredential** を生成し、その秘密鍵を使用して別の暗号化式を作成できます。そして、この新しい暗号化式は、管理モデル (サーバー設定ファイル) の機密な文字列の代わりに使用できます。クレデンシャルストアのチェーン全体を作成してセキュリティーを確保できます。このようなチェーンは、文字列が以下のように保護されているため、機密な文字列を推測することが難しくなります。

- 最初の暗号化式は、KeyStoreCredentialStore のセキュリティーを保護します。
- また、別の暗号化式では、機密文字列のセキュリティーを確保します。

- 機密な文字列の解読には、暗号化式を両方、解読する必要があります。

暗号化式のチェーンが長くなればなるほど、機密な文字列の復号化は困難になります。

# 第3章 参照

## 3.1. AGGREGATE-PROVIDERS 属性

**providers** 属性を設定することで **aggregate-providers** を設定できます。

**表3.1 aggregate-providers 属性**

| 属性 | 説明 |
| --- | --- |
| providers | 集約するプロバイダーの一覧。Elytron はリストにある最初の適切なプロバイダーを使用します。 |

## 3.2. CREDENTIAL-STORE 属性

**credential-store** は、その属性を設定することで設定できます。

**表3.2 credential-store 属性**

| 属性 | 説明 |
| --- | --- |
| create | クレデンシャルストアが存在しない場合に、ストレージを作成するかどうかを指定します。デフォルト値は **false** です。 |
| credential-reference | 保護パラメーターの作成に使用される認証情報への参照。これはクリアテキストで、または **credential-store** に保存されている認証情報への参照として指定できます。 |
| implementation-properties | クレデンシャルストアの実装固有のプロパティーのマッピング。 |
| modifiable | クレデンシャルストアを変更できるかどうか。デフォルト値は **true** です。 |
| other-providers | 必要な Jakarta Connectors オブジェクトをクレデンシャルストア内で作成できるものを検索するためにプロバイダーを取得するプロバイダーの名前。これはキーストアベースのクレデンシャルストアにのみ有効です。これが指定されていない場合は、代わりにプロバイダーのグローバルリストが使用されます。 |
| path | クレデンシャルストアのファイル名。 |
| provider-name | **CredentialStoreSpi** をインスタンス化するために使用するプロバイダーの名前。プロバイダーが指定されていない場合は、指定したタイプのインスタンスを作成できる、見つかった最初のプロバイダーが使用されます。 |

| 属性 | 説明 |
| --- | --- |
| providers | 必要なクレデンシャルストアタイプを作成できるプロバイダーを検索するプロバイダーの名前。これが指定されていない場合は、代わりにプロバイダーのグローバルリストが使用されます。 |
| relative-to | このクレデンシャルストアパスが相対するベースパス。 |
| type | クレデンシャルストアのタイプ (例: KeyStoreCredentialStore)。 |

## 3.3. CREDENTIAL-STORE 実装プロパティー

**credential-store** 実装を設定するには、その属性を設定します。

**表3.3 credential-store 実装プロパティー**

| 属性 | 説明 |
| --- | --- |
| cryptoAlg | 外部ストレージでのエントリーの暗号化と複合化に使用される暗号化アルゴリズム名。この属性は、**external** が有効な場合にのみ有効です。デフォルトは **AES** です。 |
| external | データが外部ストレージに保存され、**keyAlias** によって暗号化されるかどうか。デフォルトは **false** です。 |
| externalPath | 外部ストレージへのパスを指定します。この属性は、**external** が有効な場合にのみ有効です。 |
| keyAlias | 外部ストレージへのデータの暗号化または復号化に使用されるクレデンシャルストア内のシークレットキーエイリアス。 |
| keyStoreType | **PKCS11** など、キーストアタイプデフォルトは **KeyStore.getDefaultType()** です。 |

## 3.4. EXPRESSION=ENCRYPTION 属性

**expression=encryption** は、その属性を設定することで設定できます。

**表3.4 expression=encryption 属性**

| 属性 | 説明 |
| --- | --- |

| 属性 | 説明 |
| --- | --- |
| default-resolver | 任意の属性です。暗号化式がレスポンダなしで定義されている場合に使用するレスポンダです。たとえば、**default-resolver** として "exampleResolver" に設定し、**/subsystem=elytron/expression=encryption:create-expression (clear-text=Test Password)** コマンドで暗号化式を作成した場合に、Elytron はこの暗号化式のリゾルバーとして "exampleResolver" を使用します。 |
| prefix | 暗号化式で使用される接頭辞です。デフォルトは **ENC** です。この属性は、**ENC** がすでに定義済みの場合に指定します。すでに定義されている **ENC** 接頭辞と競合しない限り、この値を変更すべきではありません。 |
| リゾルバー | 定義されたリゾルバーのリストです。リゾルバーには以下の属性があります。<br>● **name**: 参照に使用される個別の設定名。<br>● **credential-store**: このリゾルバーが使用するシークレットキーを含む、クレデンシャルストアインスタンスへの参照。<br>● **secret-key**: 指定されたクレデンシャルストア内で Elytron が使用するシークレットキーのエイリアス。 |

## 3.5. PROVIDER-LOADER 属性

属性を設定することで、**provider-loader** を設定できます。

**表3.5 provider-loader 属性**

| 属性 | 説明 |
| --- | --- |
| argument | **Provider** がインスタンス化されるとき、コンストラクターに渡される引数。 |
| class-names | 読み込むプロバイダーの完全修飾クラス名のリスト。これらは service-loader がプロバイダーを検出した後にロードされ、重複はスキップされます。 |
| configuration | 初期化するためにプロバイダーに渡されるキーおよび値の設定。 |
| module | プロバイダーのロード元となるモジュールの名前。 |
| path | プロバイダーの初期化に使用するファイルのパス。 |
| relative-to | 設定ファイルのベースパス。 |

## 3.6. SECRET-KEY-CREDENTIAL-STORE 属性

**secret-key-credential-store** は、その属性を設定することで設定できます。

**表3.6 secret-key-credential-store 属性**

| 属性 | 説明 |
| --- | --- |
| create | まだ存在していない場合に Elytron に作成させたくない場合は、値を **false** に設定してください。デフォルトは **true** です。 |
| default-alias | デフォルトで生成されるキーのエイリアス名です。デフォルト値は **key** です。 |
| key-size | 生成される鍵のサイズです。デフォルトのサイズは 256 ビットです。以下のいずれかの値を設定できます。<br>• **128**<br>• **192**<br>• **256** |
| path | クレデンシャルストアへのパス。 |
| populate | クレデンシャルストアに **default-alias** が含まれていない場合、この属性は Elytron が作成するかどうかを示します。デフォルトは **true** です。 |
| relative-to | 属性 **path** と相対パスにある、以前に定義済みのパスへの参照。 |