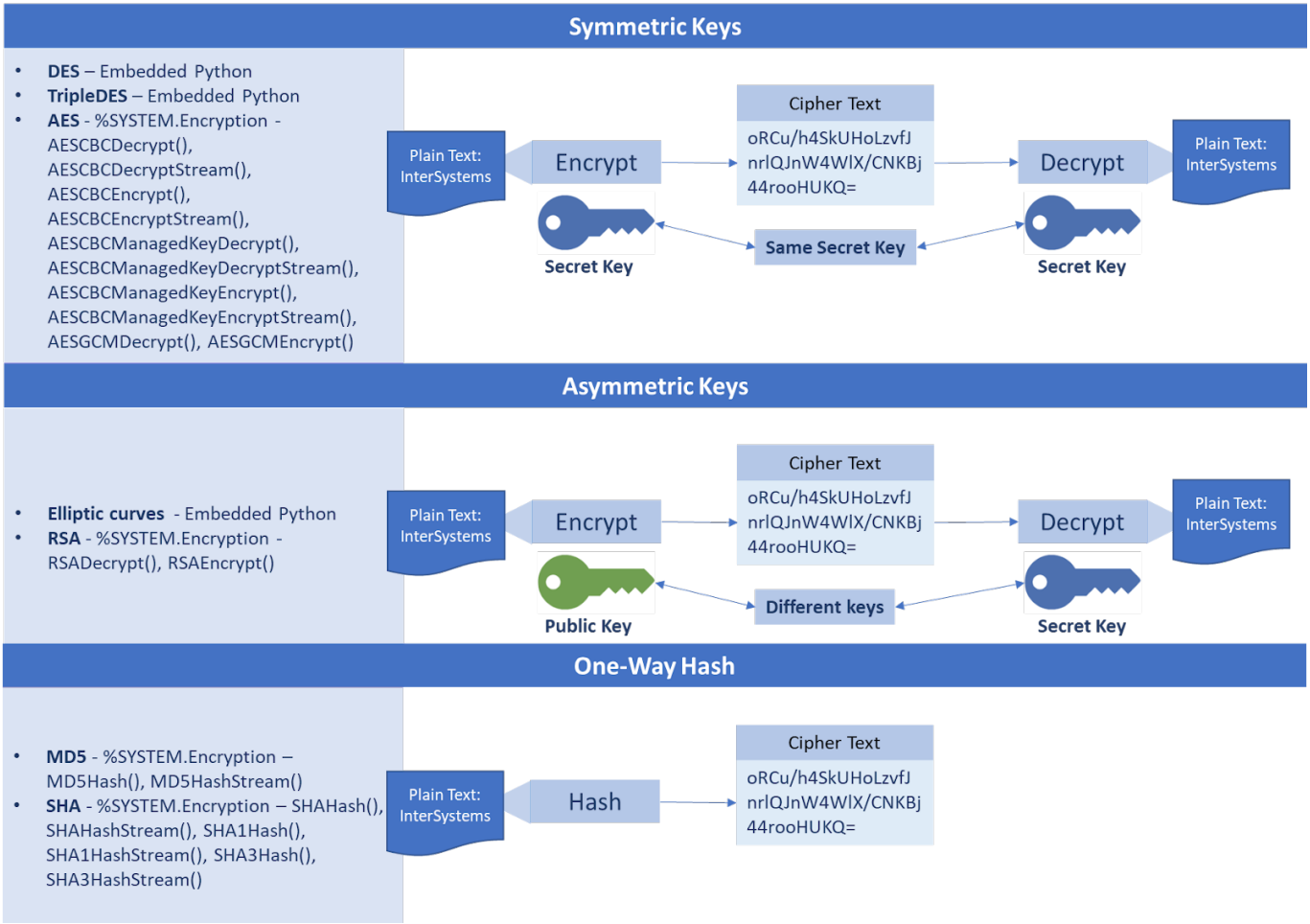
記事
Toshihiko Minamoto · 2022年8月17日 11m read

Open Exchange

%SYSTEM.Encryption クラスを習得する



InterSystems IRIS には、暗号化、復号化、およびハッシュ操作の優れたサポートが備わっています。 クラス %SYSTEM.Encryption（https://docs.intersystems.com/iris20212/csp/documatic/%25CSP.Documatic.c... ）の中には、市場に出回っている主なアルゴリズムのクラスメソッドがあります。

IRIS アルゴリズムと暗号化/復号化の方式

ご覧のとおり、操作は鍵に基づいており、3 つのオプションが含まれます。

- **対称鍵**: 暗号化と復号化の操作を実行する部分で同じ秘密鍵が使用されます。
- **非対称鍵**: 暗号化と復号化の操作を実行する部分で、暗号化に同じ秘密鍵が使用されますが、復号化においては、各パートナーが秘密鍵を所有します。
  この鍵は身元を証明するものであるため、他人と共有することはできません。
- **ハッシュ**: 暗号化だけが必要で、復号化が不要である場合に使用されます。強力なユーザーパスワードを保存する際に一般的なアプローチです。

対称暗号化と非対称暗号化の違い

- 対称暗号化では、メッセージを受信する必要のある人の間で単一の鍵を共有して使用しますが、非対称暗号化では、通信時に公開鍵と秘密鍵のペアを使用してメッセージの暗号化と復号化を行います。
- 対称暗号化は古いテクニックであるのに対し、非対称暗号化は比較的最近のテクニックです。
- 非対称暗号化は、対称暗号化モデルで鍵を共有する必要性に関わる固有の問題を補完するために導入されました。公開鍵と秘密鍵のペアを使用することで、鍵を共有する必要がなくなっています。
- 非対称暗号化には、対称暗号化よりも比較的時間がかかります。

主な違い	対称暗号化	非対称暗号化
暗号文のサイズ	元のプレーンテキストファイルより小さい暗号文。	元のプレーンテキストファイルより大きい暗号文。
データサイズ	ビッグデータの送信に使用される。	スモールデータの送信に使用される。
リソース使用率	対称鍵暗号化は、リソース使用率が低い場合に機能します。	非対称暗号化では、大量のリソースを消費する必要があります。
鍵の長さ	128 または 256 ビットの鍵サイズ。	RSA 2048 ビット以上の鍵サイズ。
セキュリティ	暗号化に単一の鍵を使用するため、セキュリティが低下。	暗号化と復号化を 2 つの異なる鍵で行うため、はるかに安全。
鍵の数	対称暗号化では、暗号化と復号化に単一の鍵を使用する。	非対称暗号化では、暗号化と復号化に 2 つの異なる鍵を使用する。
テクニック	古いテクニック。	最新のテクニック。
機密性	暗号化と復号化に使用される単一の鍵には、その鍵が改ざんされる可能性がある。	暗号化と復号化で個別に 2 つの鍵が作成されるため、鍵を共有する必要がない。
速度	対称暗号化は高速な手法。	非対称暗号化は速度が低下する。
アルゴリズム	RC4、AES、DES、3DES、および QUAD。	RSA、Diffie-Hellman、ECC アルゴリズム。

出典: https://www.ssl2buy.com/wiki/symmetric-vs-asymmetric-encryption-what-are-differences

%SYSTEM.Encryption クラスを使用して暗号化、復号化、ハッシュを実行

IRIS
の暗号化、復号化、およびハッシュ操作のサ
ポートを使用するには、https://github.com/yurimarx/cryptography-samples
にアクセスし、以下の手順に従ってください。

1. リポジトリを任意のローカルディレクトリに Clone/git pull します。

```
$ git clone https://github.com/yurimarx/cryptography-samples.git
```

2. このディレクトリで Docker ターミナルを開き、以下を実行します。

```
$ docker-compose build
```

3. IRIS コンテナを実行します。

```
$ docker-compose up -d
```

4. IRIS ターミナルを開きます。

```
$ docker-compose exec iris iris session iris -U IRISAPP

IRISAPP>
```

5. 非対称暗号化の RSA Encrypt を実行するには、以下を実行します。

```
IRISAPP>Set ciphertext = ##class(dc.cryptosamples.Samples).DoRSAEncrypt("InterSystems
")
IRISAPP>Write ciphertext
Ms/eR7pPmE39KBJu75EOYIxpFEd7qqoji61EfahJE1r9mGZX1NYuw5i2cPS5YwE3Aw6vPAeiEKXF
rYW++WtzMeRIRdCMbLG9PrCHD3iQHfZobBnuzx/JMXVc6a4TssbY9gk7qJ5BmlqRTU8zNJiiVmd8
pCFpJgwKzKkNrIgaQn48EgnwblmVkxSFnF2jwXpBt/naNudBguFUBthef2wfULl4uY00aZzHHNxA
bi15mzTdlSJu1vRtCQaEahng9ug7BZ6dyWCHOv74O/L5NEHI+jU+kHQeF2DJneE2yWNESzqhSECa
ZbRjjxNxiRn/HVAKyZdAjkGQVKUkyG8vjnc3Jw==
```

6. 非対称復号化の RSA Decrypt を実行するには、以下を実行します。

```
IRISAPP>Set plaintext = ##class(dc.cryptosamples.Samples).DoRSADecrypt(ciphertext)
IRISAPP>Write plaintext
InterSystems
```

7. 非対称暗号化の AES CBC Encrypt を実行するには、以下を実行します。

```
IRISAPP>Do ##class(dc.cryptosamples.Samples).DoAESCBCEncrypt("InterSystems")
8sGVUikDZaJF+Z9UljFVAA==
```

8. 対称暗号化の AES CBC Decrypt を行うには、以下を実行します。

```
IRISAPP>Do ##class(dc.cryptosamples.Samples).DoAESCBCDecrypt("8sGVUikDZaJF+Z9UljFVAA=
=")
InterSystems
```

9. 古いハッシュアプローチの MD5 hash を実行するには、以下を実行します。

```
IRISAPP>Do ##class(dc.cryptosamples.Samples).DoHash("InterSystems")
rOs6HXfrnbEY5+JBdUJ8hw==
```

10. 推奨されるハッシュアプローチの SHA hash を実行するには、以下を実行します。

```
IRISAPP>Do ##class(dc.cryptosamples.Samples).DoSHAHash("InterSystems")
+X0hDlyoViPlWOm/825KvN3rRKB5cTU5EQTDLvPWM+E=
```

11. ターミナルを終了するには、以下を実行します。

```
HALT ??? H ??????????????????
```

ソースコードについて

1. 対称鍵について

```
# 対称暗号化/復号化で使用する
ENV SECRETKEY=InterSystemsIRIS
```

Dockerfile に、対称操作で秘密鍵として使用される環境鍵が作成されました。

2. 非対称鍵について

非対称暗号化/復号化で使用する RUN openssl req -new -x509 -sha256 -config example-com.conf -newkey rsa:2048 -nodes -keyout example-com.key.pem -days 365 -out example-com.cert.pem
Dockerfile に、非対称操作で使用される秘密鍵と公開鍵が生成されました。

3. 対称暗号化

```
// 暗号化するための対称鍵の例
ClassMethod DoAESCBCEncrypt(plaintext As %String) As %Status
{
  // utf-8 に変換
  Set text=$ZCONVERT(plaintext,"O","UTF8")

  // 秘密鍵を設定
  Set secretkey = $system.Util.GetEnviron("SECRETKEY")
  Set IV = $system.Util.GetEnviron("SECRETKEY")

  // テキストを暗号化
  Set text = $SYSTEM.Encryption.AESCBCEncrypt(text, secretkey, IV)
  Set ciphertext = $SYSTEM.Encryption.Base64Encode(text)

  Write ciphertext
}
```

テキストの暗号化に AES CBC Encrypt 操作が使用されました。
Base64 Encode は、プリティ/読み取り可能なテキストとしてユーザーに結果を返します。

4. 対称復号化

```
// 復号化するための対称鍵の例
ClassMethod DoAESCBCDecrypt(ciphertext As %String) As %Status
{
  // 秘密鍵を設定
  Set secretkey = $system.Util.GetEnviron("SECRETKEY")
  Set IV = $system.Util.GetEnviron("SECRETKEY")

  // テキストを復号化
  Set text=$SYSTEM.Encryption.Base64Decode(ciphertext)
  Set text=$SYSTEM.Encryption.AESCBCDecrypt(text,secretkey,IV)

  Set plaintext=$ZCONVERT(text,"I","UTF8")
  Write plaintext
}
```

テキストの復号化に AES CBC Decrypt 操作が使用されました。
Base64 Decode は、復号化に使用できるように、暗号化されたテキストをバイナリテキストに返します。

5. 非対称暗号化

```
// 暗号化するための非対称鍵の例
```

```
ClassMethod DoRSAEncrypt(plaintext As %String) As %Status
{
  // 公開鍵証明書を取得
  Set pubKeyFileName = "/opt/irisbuild/example-com.cert.pem"
  Set objCharFile = ##class(%Stream.FileCharacter).%New()
  Set objCharFile.Filename = pubKeyFileName
  Set pubKey = objCharFile.Read()
  // RSA を使って暗号化
  Set binarytext = $System.Encryption.RSAEncrypt(plaintext, pubKey)
  Set ciphertext = $SYSTEM.Encryption.Base64Encode(binarytext)

  Return ciphertext
}
```

RSA で暗号化するには、公開鍵ファイルの内容を取得する必要があります。
テキストの暗号化に、RSA Encrypt 操作が使用されました。

6. 非対称復号化

```
 // 復号化するための非対称鍵の例
ClassMethod DoRSADecrypt(ciphertext As %String) As %Status
{
  // 秘密鍵を取得
  Set privKeyFileName = "/opt/irisbuild/example-com.key.pem"
  Set privobjCharFile = ##class(%Stream.FileCharacter).%New()
  Set privobjCharFile.Filename = privKeyFileName
  Set privKey = privobjCharFile.Read()
  // 暗号文をバイナリ形式で取得
  Set text=$SYSTEM.Encryption.Base64Decode(ciphertext)
  // RSA を使って復号化
  Set plaintext = $System.Encryption.RSADecrypt(text, privKey)
  Return plaintext
}
```

RSA で復号化するには、秘密鍵ファイルの内容を取得する必要があります。
テキストの復号化に、RSA Decrypt 操作が使用されました。

7. MD5 を使ったテキストのハッシュ化（古いアプローチ）

```
 // ハッシュの例
ClassMethod DoHash(plaintext As %String) As %Status
{
  // utf-8 に変換
```

```
    Set text=$ZCONVERT(plaintext,"O","UTF8")

    // テキストをハッシュ化
    Set hashtext = $SYSTEM.Encryption.MD5Hash(text)

    Set base64text = $SYSTEM.Encryption.Base64Encode(hashtext)
    // 16 進テキストを以下のベストプラクティスに変換
    Set hextext = ..GetHexText(base64text)
    // 小文字を使って返す
    Write $ZCONVERT(hextext,"L")
}
```

MD5 Hash 操作でテキストを暗号化すると、それを復号化することはできません。
MD5 を使ったハッシュ化は、安全でないと見なされているため、新しいプロジェクトには推奨されません。
そのため、この方式は SHA に置き換えられています。 InterSystems IRIS は SHA
をサポートしています（次の例で紹介します）。

8. SHA を使ったテキストのハッシュ化（推奨されるアプローチ）

この例では、SHA-3 Hash 方式を使用します。 InterSystems のドキュメントによると、この方式は、US Secure Hash Algorithm - 3 を使ってハッシュを生成します。 （詳細は、連邦情報処理規格 202 をご覧ください。）

```
// SHA を使ったハッシュ
ClassMethod DoSHAHash(plaintext As %String) As %Status
{
    // utf-8 に変換
    Set text=$ZCONVERT(plaintext,"O","UTF8")

    // テキストをハッシュ化
    Set hashtext = $SYSTEM.Encryption.SHA3Hash(256, text)

    Set base64text = $SYSTEM.Encryption.Base64Encode(hashtext)
    // 16 進テキストを以下のベストプラクティスに変換
    Set hextext = ..GetHexText(base64text)
    // 小文字を使って返す
    Write $ZCONVERT(hextext,"L")
}
```

SHA 方式では、ハッシュ操作で使用するビット長を設定できます。
ビット数が多いほど、ハッシュを解読するのがより困難になりますが、
ハッシュ化の処理速度が低下してしまいます。 この例では、256 ビットが使用されました。
ビット長については以下のオプションを選択できます。

- 224（SHA-224）
- 256（SHA-256）
- 384（SHA-384）

- 512（SHA-512）

#セキュリティ #ベストプラクティス #InterSystems IRIS
InterSystems Open Exchangeで関連アプリケーションを確認してください

ソースURL:
https://jp.community.intersystems.com/post/systemencryption-%E3%82%AF%E3%83%A9%E3%82%B9%E3%82%92%E7%BF%92%E5%BE%97%E3%81%99%E3%82%8B