THE DOCUMENT COMPANY
FUJI XEROX

Java JCE：機能と利用法

富士ゼロックス株式会社
稲田 龍
<Ryu.Inada@fujixerox.co.jp>

THE DOCUMENT COMPANY
FUJI XEROX

JAVAにおけるセキュリティAPI

2系列のインターフェイス群

THE DOCUMENT COMPANY
FUJI XEROX

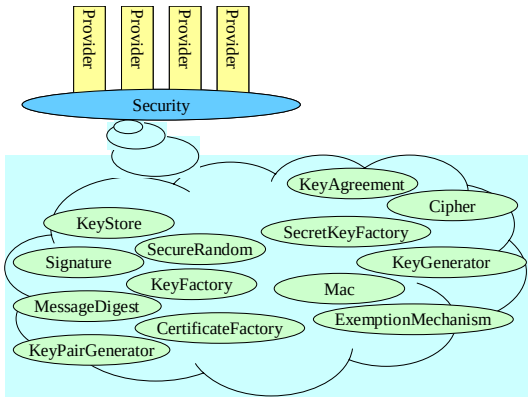
- java.seceurity.*
  - 基本となる機能を提供
    - 鍵管理、高度乱数発生器、署名、鍵入出力、一方向性ハッシュ、証明書パス検証、鍵対生成等
- javax.crypto.*
  - 暗号機能/免責機能など強力な機能を提供
    - 鍵交換、暗号、共通鍵入出力、共通鍵生成、MAC (Message Authentication Code)、免責機能

- 公開鍵系(電子署名系)の機能はjava.security
- 共通鍵系(暗号系)の機能はjavax.crypto

プロバイダモデルの採用

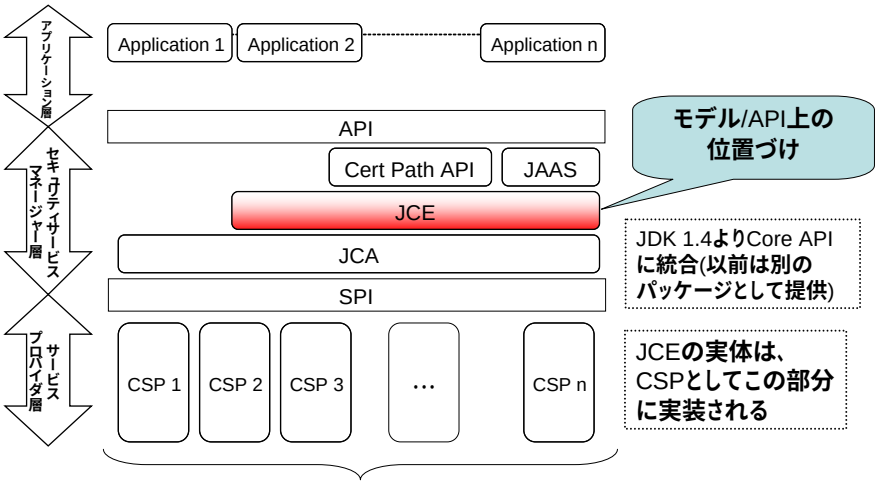
THE DOCUMENT COMPANY
FUJI XEROX

- getInstance()メソッド使ってプロバイダを検索し利用



JAVAにおけるJCEの位置

THE DOCUMENT COMPANY
FUJI XEROX



JDK 1.4のJCEで提供される機能

THE DOCUMENT COMPANY
FUJI XEROX

- 暗号
  - 共通鍵暗号/公開鍵暗号
- メッセージダイジェスト(一方向性ハッシュ)
- MAC (Message Authentication Code)
  - 秘密鍵ベースのデータ改ざん検出
- 電子署名
- 鍵管理
- 安全な乱数
- 証明書管理機能
- 認証
  - Kerberos/パスワードベース/X500系
- 免責機構 (Exemption Mechanism)
  - アクセスコントロール
    - 暗号系の動作に関してのアクセスコントロール機能

定義済みの暗号アルゴリズム

THE DOCUMENT COMPANY FUJI XEROX

AES	**A**dvanced **E**ncryption **S**tandardとしてNISTによってFIPSに上げられたブロック暗号 (JDK 1.4では利用不可)
Blowfish	Bruce Schneier設計によるブロック暗号 (JDK 1.4では利用不可)
DES	FIPS PUB 46-2で定義されているブロック暗号
DESede	DESを3重に使用するブロック暗号 (JDK 1.4では利用不可)
PBEWith<digest>And<encryption>	パスワードベースの暗号化アルゴリズム
RC2, RC4, RC5	RSA Data Security社が策定したブロック暗号。鍵長/段数を可変に出来るのが特徴 (JDK 1.4では利用不可)
RSA	PKCS#1で定義されている公開鍵暗号 (JDK 1.4では利用不可)

定義済みの暗号アルゴリズムがすべて提供されているわけではない

定義済み動作モード

THE DOCUMENT COMPANY FUJI XEROX

NONE	指定なし
CBC	Cipher Block Chaining
CFB	Cipher Feed Back
ECB	Electric Code Book
OFB	Output Feed Back
PCBC	Propagating CBC

すべての暗号アルゴリズムにおいて、定義済みの動作モードがすべて提供されているわけではない

定義済みパディング

THE DOCUMENT COMPANY FUJI XEROX

NoPadding	パディングなし
OAEPWith<digest>And<mgf>Padding	PKCS#1 v2で定義されているパディングスキーマ
PKCS5Padding	PKCS#5で定義されているパディングスキーマ
SSL3Padding	SSL v3/TLS v1で定義されているパディングスキーマ

すべての暗号アルゴリズムにおいて、定義済みのパディング
がすべて提供されているわけではない

定義済みハッシュアルゴリズム

THE DOCUMENT COMPANY FUJI XEROX

MD2	RFC 1319で定義されている128ビットの精度を持ったハッシュアルゴリズム。
MD5	RFC 1321で定義されている128ビットの精度を持ったハッシュアルゴリズム。一般的に使用されているが、研究の結果攻撃方法が見つかっている。
SHA-1	FIPS 180-1で定義されている160ビットの精度を持つハッシュアルゴリズム。MD5と比較し、こちらのほうが安全とされている。
SHA-256, SHA-384, SHA-512	FIPS 180-2で定義されているSHA-1の後継として策定されたアルゴリズム。 よりセキュアなハッシュを必要とする場合はこちらの使用を検討すると良い。

定義済みのハッシュアルゴリズムがすべて提供されているわけではない

共通の使い方

THE DOCUMENT COMPANY
FUJI XEROX

- インスタンスの取得
  - Providerモデルを使っているため、必要とされる機能を持ったインスタンスの取得が必要となる
    - 動作環境、動作状況により利用するProviderが異なる
    - 特定のProviderの機能を利用する場合は、要注意。
- 初期化
- データを流し込み
- 終了処理をする

THE DOCUMENT COMPANY
FUJI XEROX

暗号化･復号アプリケーション

Example1

THE DOCUMENT COMPANY
FUJI XEROX

- Exapmle1.javaとして公開中
- **共通鍵暗号を用いた暗号化/復号**

暗号化機能を提供しているクラス

THE DOCUMENT COMPANY
FUJI XEROX

- javax.crypto.Cipher
  - **暗号アルゴリズムの抽象化を行っている**
  - **共通鍵/公開鍵の両暗号系をサポート**

javax.crypto.Cipherの主要メソッド

THE DOCUMENT COMPANY FUJI XEROX

- init
  - アルゴリズムを初期化するメソッド
  - 動作に使用する鍵と動作モードを指定することによって初期化を行なう
- update
  - アルゴリズムへデータを入力し、変換結果を受け取る
- doFinal
  - 終了処理を行うメソッド
  - アルゴリズム中にフラッシュすべきデータなどがある場合、そのデータが返ってくる。そのデータもupdate()の処理と同様に処理を行わなければならない。

- その他
  - wrap
    - 鍵の暗号化 (wrap) を行う
  - unwrap
    - 暗号化 (wrap) されている鍵を取り出す

処理の流れ

THE DOCUMENT COMPANY FUJI XEROX

- 利用可能な暗号アルゴリズムを取得
- 鍵の生成
  - IVの生成
- 初期化
- データの暗号化/復号

暗号アルゴリズムを取得

THE DOCUMENT COMPANY
FUJI XEROX

```
57: protected Example1(
58:         String     algorithm,
59:         byte[]     key,
60:         byte[]     iv,
61:         String     inPath,
62:         String     outPath)
63: throws NoSuchAlgorithmException,
             NoSuchPaddingException,
             InvalidKeySpecException,
             InvalidKeyException {
64:         // 入出力情報を保存
65:         this.inPath          = inPath;
66:         this.outPath         = outPath;
67:
68:         // 暗号アルゴリズムの実装を取得
69:         this.cipher          = Cipher.getInstance(algorithm);
```

暗号アルゴリズムを
指定して実装を取得

注意点

THE DOCUMENT COMPANY
FUJI XEROX

- Providerモデルを採用しているため
  - 使える暗号アルゴリズムは実行時に決定
  - 同じアルゴリズムでも、複数のProviderがサポートしている場合もあるので、要注意
- アルゴリズム指定は文字列で行なう
  - “暗号アルゴリズム”のみを指定
    - “DES”, “RC2”, “RSA”など
  - “暗号アルゴリズム／動作モード／パディング”を指定
    - “DES/CBC”, “RC2/ECB/NoPadding”など

鍵の生成

THE DOCUMENT COMPANY
FUJI XEROX

```
71:     // 鍵の生成
72:     String    name     = cipher.getAlgorithm();
73:     int       offset   = name.indexOf('/');
74:
75:     if(offset >= 0) {
76:               name = name.substring(0, offset);
77:     }
78:     SecretKeyFactory factory = SecretKeyFactory.getInstance(name);

79:     KeySpec             spec = null;
80:     if(name.startsWith("DES")) {
81:               spec = new DESKeySpec(key);
82:     } else if(name.startsWith("TripleDES") || name.startsWith("DESede")) {
83:               spec = new DESedeKeySpec(key);
84:     } else {
85:               spec = new SecretKeySpec(key, name);
86:     }

87      this.secretKey = factory.generateSecret(spec);
```

利用している暗号アルゴリズム名を取得

秘密鍵を作るためのFactoryを取得

暗号アルゴリズムによってはKeySpecを指定する必要がある

Factory/Specが決まったら鍵を生成

注意点

THE DOCUMENT COMPANY
FUJI XEROX

- 秘密鍵(共通鍵)生成のステップ
  - SecretKeyFactoryを決定
  - KeySpecを決定
  - SecretKeyFactory.generateSecret()
- SecreteKeyFactory
  - 利用するプロバイダ/インスタンスにより異なる
    - Cipher.getAlgorithm()でアルゴリズム名を取得
    - SecretKeyFactory.getInstance()で取得
- KeySpec
  - 利用するアルゴリズムにより固有のクラスを要求するものがあるので要注意

初期化(暗号化の場合)

THE DOCUMENT COMPANY
FUJI XEROX

```
93: /**
94: * 暗号化を行うメソッド
95: **/
96: protected void doEncryption()
97: throws InvalidKeyException, IOException,
        IllegalBlockSizeException,
        BadPaddingException, InvalidAlgorithmParameterException {
98:       // 暗号化用に初期化
99:       cipher.init(Cipher.ENCRYPT_MODE, secretKey, algorithmSpec);
100:
101:      // 暗号化・復号化に共通な処理
102:      doProcess();
103:}
```

暗号化用に初期化を実施

初期化(復号の場合)

THE DOCUMENT COMPANY
FUJI XEROX

```
105: /**
106:  * 復号を行うメソッド
107:  **/
108: protected void doDecryption()
109: throws InvalidKeyException, IOException,
              IllegalBlockSizeException, BadPaddingException,
              InvalidAlgorithmParameterException {
110:      // 復号用に初期化
111:      cipher.init(Cipher.DECRYPT_MODE, secretKey, algorithmSpec);
112:
113:      // 暗号化・復号に共通な処理
114:      doProcess();
115: }
```

復号用に初期化を実施

データの暗号化/復号

THE DOCUMENT COMPANY
FUJI XEROX

```
130:    while(true) {
131:            // データの読み取りと変換を行う
132:            int size = in.read(buffer);
133:            if(size < 0) {
134:                    break;
135:            }
136:            byte[] processed = cipher.update(buffer, 0, size);
137:            if(processed != null && processed.length > 0) {
138:                    out.write(processed);
139:            }
140:    }
141:    // 残りのデータを取り出して処理を終了
142:    byte[] processed = cipher.doFinal();
143:    if(processed != null && processed.length > 0) {
144:            out.write(processed);
145     }
```

データを入れる

終了処理は必須

注意点

THE DOCUMENT COMPANY
FUJI XEROX

- cipher.update()/cipher.doFinal()
  - 入力されるデータ（平文）の長さと、返されるデータ（暗号文）の長さが一致しない可能性がある
- 終了時にcipher.doFinal()を呼び出すこと
- Cipherのインスタンスはスレッドセーフではない

THE DOCUMENT COMPANY
FUJI XEROX

一方向性ハッシュ

THE DOCUMENT COMPANY
FUJI XEROX

Example2

- Example2.javaとして公開中
- **いわゆる暗号学的一方向性関数を利用したハッシュ値計算**

一方向性ハッシュを提供するクラス

THE DOCUMENT COMPANY FUJI XEROX

- java.security.MessageDigest
  - ハッシュ値を計算するための機能を抽象化
  - データ入力/ハッシュ値取得の2ステップで動作

java.security.MessageDigestの主要メソッド

THE DOCUMENT COMPANY FUJI XEROX

- reset
  - コンテキストをリセットし、初期状態へ戻す
- update
  - アルゴリズムへデータを入力する
- digest
  - 入力したデータ全体に対するハッシュ値を取得する
  - 現コンテキストの使用終了を宣言する
- getDigestLength
  - ダイジェストの大きさを返す

処理の流れ

THE DOCUMENT COMPANY
FUJI XEROX

- 利用可能なハッシュアルゴリズムの取得
- データ入力
- ハッシュ値取得

ハッシュアルゴリズムの取得

THE DOCUMENT COMPANY
FUJI XEROX

```
36: public Example2(
37:         String     algorithm,
38:         String     inPath)
39: throws NoSuchAlgorithmException {
40:         messageDigest    = MessageDigest.getInstance(algorithm);
41:         inPathName       = inPath;
42: }
```

利用するハッシュアルゴリズムを取得

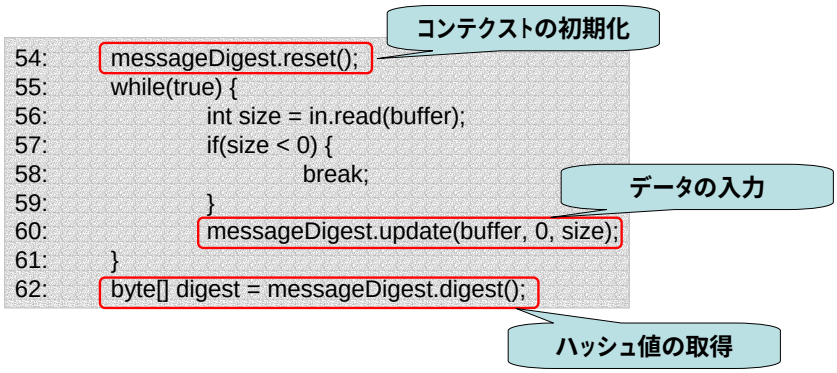
注意点

THE DOCUMENT COMPANY FUJI XEROX

- Providerモデルを採用しているため
  - 使えるハッシュアルゴリズムは実行時に決定
  - 同じアルゴリズムでも、複数のProviderがサポートしている場合もあるので、要注意
- アルゴリズム指定は文字列で行なう
  - “ハッシュアルゴリズム”のみを指定
    - “MD5”, “SHA-1”

データ入力・ハッシュ値取得

THE DOCUMENT COMPANY FUJI XEROX



注意点

THE DOCUMENT COMPANY
FUJI XEROX

- **初期化は必須**
  - MessageDigest.reset()
- **データの流し込みは、**MessageDigest.update()
- **ハッシュ値の取得は、**MessageDigest.digest()
  - doFinal()**ではない。**
  - **ハッシュ値は**byte[]**で返ってくる**
  - **データ量は利用するハッシュアルゴリズムに依存**
    - **固定的な量を仮定してはならない**
  - **ハッシュデータのサイズを得るには**……
    - getDigestLength()
    - **配列の長さを調べる**
- MessageDigest**のインスタンスは、スレッドセーフではない**

THE DOCUMENT COMPANY
FUJI XEROX

電子署名アプリケーション

Example4

THE DOCUMENT COMPANY
FUJI XEROX

- Example4.javaとして公開中
- **電子署名の作成と検証を行う**

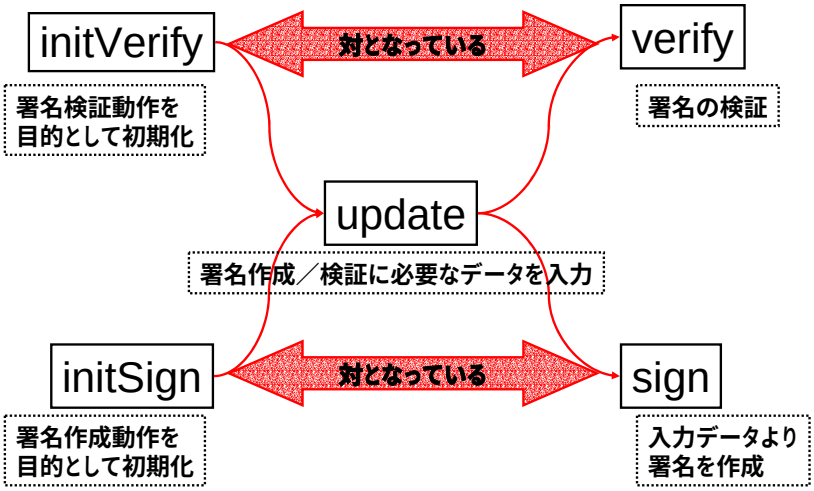
電子署名機能を提供するクラス

THE DOCUMENT COMPANY
FUJI XEROX

- java.security.Signature

java.security.Signatureの主要メソッド

THE DOCUMENT COMPANY FUJI XEROX



処理の流れ

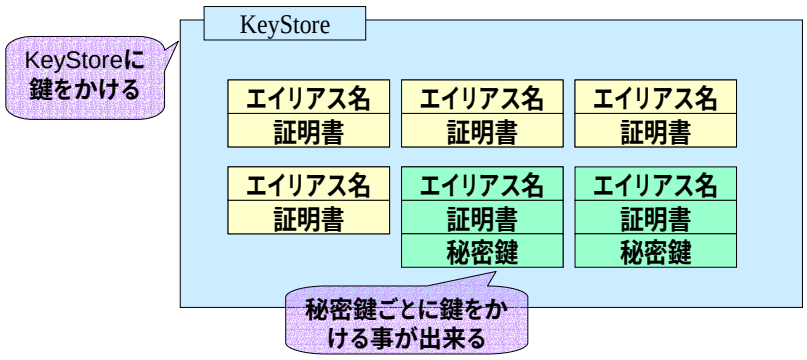
THE DOCUMENT COMPANY FUJI XEROX

- 鍵の読み込み
- 署名アルゴリズムの取得
- 初期化
- データの入力
- 署名作成もしくは署名検証

KeyStoreとは?

THE DOCUMENT COMPANY
FUJI XEROX

- JCEにおける暗号鍵・証明書リポジトリィなどの情報を一元的に扱うための仮想的な構造
  - 主に公開鍵証明書と私有鍵の管理を行う



KeyStoreの機能

THE DOCUMENT COMPANY
FUJI XEROX

- エントリ調査
  - 一覧の作成や当該エントリーの種類判別など
- エントリ操作
  - 追加と削除
  - 上書きは出来ない
- KeyStore入出力
  - KeyStoreへの入出力(load/store)

鍵の読み込み(KeyStoreの読み込み)

THE DOCUMENT COMPANY
FUJI XEROX

```
65: protected void loadKey(
66:         String     storeType,
67:         String     storeFileName,
68:         String     entryName,
69:         char[]     passwd)
70: throws NoSuchAlgorithmException,
             KeyStoreException, IOException,
             CertificateException {

71:         // 鍵ストアの実装を獲得します
72:         KeyStore store = KeyStore.getInstance(storeType);

73:         // 保存されているストアを読み取ります
74:         store.load(new FileInputStream(storeFileName), passwd);
```

KeyStoreの実装を取得する

KeyStoreへ読み込む

KeyStoreのパスワード

鍵の読み込み(私有鍵の取り出し)

THE DOCUMENT COMPANY
FUJI XEROX

```
65: protected void loadKey(
66:         String     storeType,
67:         String     storeFileName,
68:         String     entryName,
69:         char[]     passwd)
                       ……

76:         // 秘密鍵を再生させます
77:         try {
78:                    privateKey = (PrivateKey)(store.getKey(entryName, passwd));
79:         } catch(Exception e) {
80:                    privateKey = null;
81:         }

                       ……
```

私有鍵を取り出す

エイリアス名を指定

私有鍵のパスワード

鍵の読み込み(証明書/公開鍵の取り出し)

THE DOCUMENT COMPANY FUJI XEROX

```
68: protected void loadKey(
…       String    storeType,String  storeFileName,String entryName,
…       char[]    passwd)
                  ……
82:       // 公開鍵を再生させます
83:       try {
84:                 // 証明書を取り出し
85:                 Certificate certificate = store.getCertificate(entryName);
86:                 if(certificate != null) {
87:                           publicKey = certificate.getPublicKey();
88:                 } else {
89:                           publicKey = null;
90:                 }
91:       } catch(Exception e) {
92:                 publicKey = null;
93:       }
94: }
```

エイリアス名を指定して証明書を取り出す

証明書から公開鍵を取り出す

注意点

THE DOCUMENT COMPANY FUJI XEROX

- KeyStore/私有鍵の各々にパスワードがかけられる
  - Example4では、KeyStore/私有鍵のパスワードは共通になっている
  - 本来は、別々にすべき
- Providerモデルを採用しているため
  - 利用するKeyStoreは実行時に決定
  - 同じKeyStoreでも、複数のProviderがサポートしている場合もあるので、要注意

署名アルゴリズムの取得

THE DOCUMENT COMPANY
FUJI XEROX

利用している公開鍵暗号アルゴリズムを取得

```
57:     // 署名アルゴリズムを特定
58:     String keyAlgorithm = publicKey.getAlgorithm();
59:     signatureAlgorithm = Signature.getInstance(hashAlgorith +
60:                                 "with" + keyAlgorithm);
```

署名アルゴリズムを取得

署名アルゴリズムは
SHA1withRSA
の様に「ハッシュアルゴリズム」with「公開鍵暗号アルゴリズム」で指定

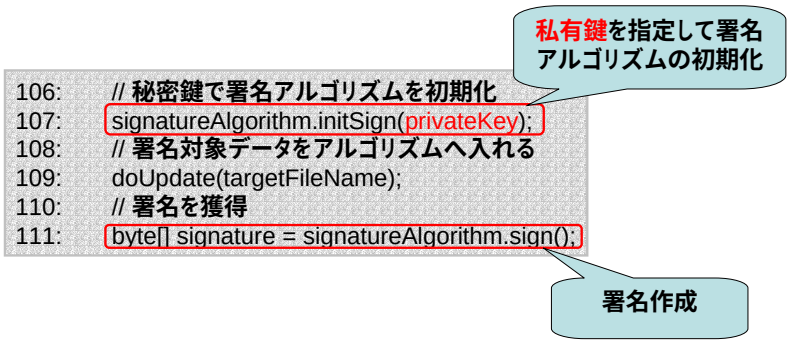
注意点

THE DOCUMENT COMPANY
FUJI XEROX

- Providerモデルを採用しているため
  - 使える署名アルゴリズムは実行時に決定
  - 同じアルゴリズムでも、複数のProviderがサポートしている場合もあるので、要注意
- アルゴリズム指定は文字列で行なう
  - “<HashAlgorithm>with<KeyAlgorithm> ”で指定
    - 最も一般的な形式
      - “MD5withRSA”, “SHA1withRSA”など
  - “鍵アルゴリズム名”で指定
    - 鍵アルゴリズムだけで署名アルゴリズム全体が記述できる場合
      - “DSA”など
- JDK単体で利用できるのは
  - MD2withRSA/MD5withRSA/SHA1withRSA/DSA

署名作成

THE DOCUMENT COMPANY
FUJI XEROX



注意点

THE DOCUMENT COMPANY
FUJI XEROX

- 私有鍵を指定してinitSign()を呼び出す
- データ入力後、sign()を呼び出す
  - 署名はbyte[]で返され、サイズの仮定をすることは出来ない
    - RSA系を利用した場合
      - 鍵のサイズに等しい
    - DSA系を使用した場合
      - 利用したハッシュアルゴリズムの出力の2倍程度
- Signatureのインスタンスは、スレッドセーフではない。

署名検証

THE DOCUMENT COMPANY
FUJI XEROX

```
164:    // 公開鍵で署名アルゴリズムを初期化
165:    signatureAlgorithm.initVerify(publicKey);
166:    // 署名対象データをアルゴリズムへ入力
167:    doUpdate(targetFileName);
168:    // 署名の確認
169:    if(signatureAlgorithm.verify(signature)) {
170:            System.out.println("Signature is valid");
171:    } else {
172:            System.out.println("Signature is invalid");
173:    }
```

公開鍵を指定して署名アルゴリズムの初期化

署名検証

注意点

THE DOCUMENT COMPANY
FUJI XEROX

- 公開鍵を指定してinitVerify()を呼び出す
- verify()を呼び出す
  - 署名データと現在のコンテキストを比較するために存在する
    - 比較結果はbooleanで返される
    - 署名データとして不正なものを与えられた場合、不正の原因に応じた例外を発生する
- Signatureのインスタンスは、スレッドセーフではない。

データ入力の注意点

THE DOCUMENT COMPANY
FUJI XEROX

- update()を用いる
  - 使用方法はMessageDigest/Cipherと同じ
  - 必要な分だけ、update()にデータを渡す

THE DOCUMENT COMPANY
FUJI XEROX

証明書リポジトリ操作

Example5

THE DOCUMENT COMPANY
FUJI XEROX

- Example5.javaとして公開
- JCEの証明書リポジトリィの1種であるキーストアの操作を行なう

処理の流れ

THE DOCUMENT COMPANY
FUJI XEROX

- KeyStore読み込み
- 操作
  - Copy/Delete/List
- KeyStore書き出し

- 細かい処理はExampe5.javaを参照のこと
  - ここでは、証明書および私有鍵のload/storeに関して説明する

証明書のload

THE DOCUMENT COMPANY
FUJI XEROX

- 実装はExample5.loadCertificate() 339-356行目

```
343:      in = new FileInputStream(certificateFile);
344:      CertificateFactory factory = CertificateFactory.getInstance("X.509");
345:
346:      return factory.generateCertificate(in);
```

取得したFactoryに対してストリームを指定してloadする

証明書の形式として"X509"を指定

CertificateFactoryの取得の際は、loadしたい証明書の形式に合わせた物を使う

証明書のstore

THE DOCUMENT COMPANY
FUJI XEROX

- 実装はExample5.storeCertificate() 317-334行目

```
322:      out = new FileOutputStream(certificateFile);
323:      out.write(certificate.getEncoded());
```

Certificateクラスのを使って書き出す

私有鍵のload

THE DOCUMENT COMPANY
FUJI XEROX

- 実装はExample5.loadPrivateKey() 276-312行目

DSA/RSAしかない

```
065: static protected final String[]     knownKeyType = {"DSA", "RSA" };
```

PKCS8形式でエンコードされている私有鍵を設定している

私有鍵がエンコードされているバイト列

```
294: PKCS8EncodedKeySpec spec = new PKCS8EncodedKeySpec(encoded);
295: for(int i = 0 ; i < knownKeyType.length ; i++) {
296:       try {
297:             KeyFactory factory = KeyFactory.getInstance(knownKeyType[i]);
298:             return factory.generatePrivate(spec);
299:       } catch(Exception e) {}
300: }
```

私有鍵のアルゴリズムを決定する

私有鍵を読み出す

私有鍵のstore

THE DOCUMENT COMPANY
FUJI XEROX

- 実装はExample5.storePrivateKey() 251-271行目

私有鍵のアルゴリズムに合わせたKeyFactoryを取得

私有鍵のアルゴリズムを取得

```
255:      KeyFactory factory = KeyFactory.getInstance(key.getAlgorithm());
256:      KeySpec spec = factory.getKeySpec(key, PKCS8EncodedKeySpec.class);
257:
258:      System.out.println("Store private key into " + privateKe
259:      out = new FileOutputStream(privateKeyFile);
260:      out.write(((PKCS8EncodedKeySpec)(spec)).getEncoded());
```

PKCS8で出力する事を指定している

getEncoded()で書き出す

THE DOCUMENT COMPANY
FUJI XEROX

証明書検証

THE DOCUMENT COMPANY
FUJI XEROX

Example3

- Example3.java
- **証明書の検証を行なう**

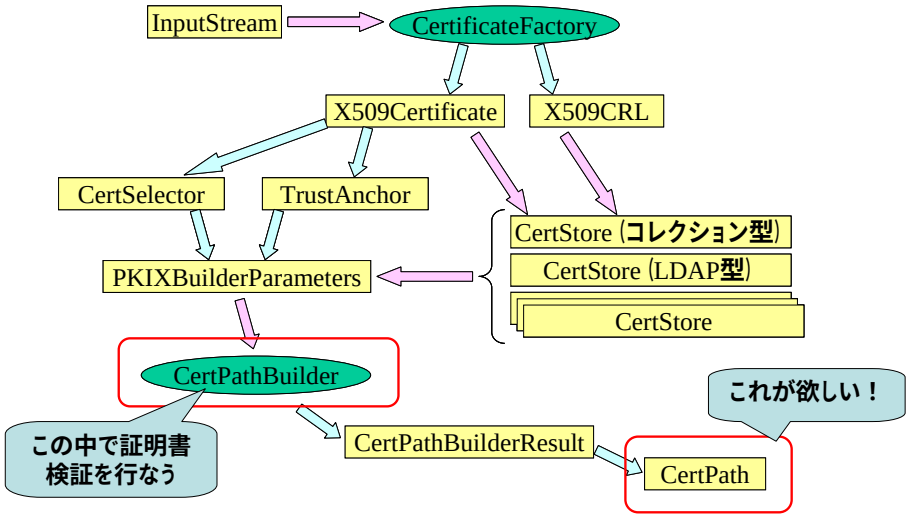
証明書検証機能を提供するクラス

THE DOCUMENT COMPANY FUJI XEROX

- java.security.cert.CertPathValidator
  - 証明書パスについて証明書の妥当性検証を行う
- java.security.cert.CertPathBuilder
  - 検証したい証明書から検証パスを生成する
  - 生成されたパスは「証明書パス検証」プロセスを経由することにより妥当性検証が行われている

証明書検証の全体像

THE DOCUMENT COMPANY FUJI XEROX



パス構築の例

THE DOCUMENT COMPANY
FUJI XEROX

- 実装はExample3.run 222-230行目

```
222: public void run() {
223:       try {
224:                 CertPathBuilderResult result = builder.build(params);
225:                 printResult(result);
226:       } catch(Exception e) {
227:                 e.printStackTrace();
228:                 System.exit(3);
229:       }
230: }
```

パス構築と検証を行う

構築パラメータの作成

THE DOCUMENT COMPANY
FUJI XEROX

- 実装はExample3のコンストラクタ 121-135行目

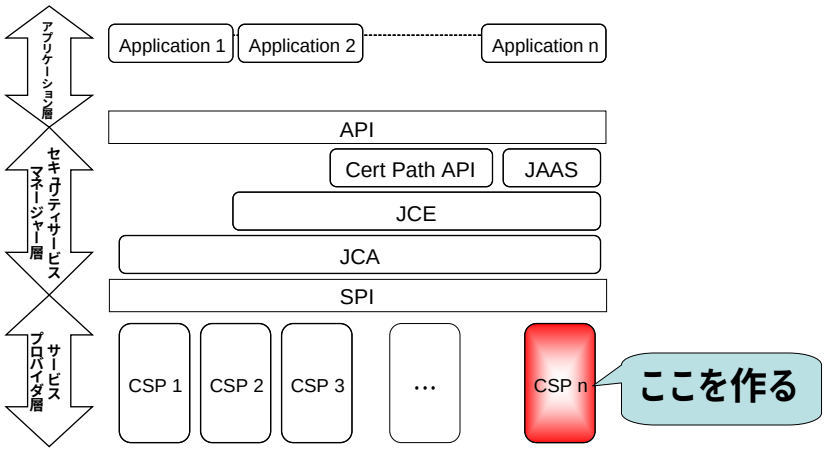
```
// 検証パラメータの作成
PKIXBuilderParameters pkixParams = new PKIXBuilderParameters(trustAnchor,
                                                    targetConstraint);

// リポジトリを設定します
pkixParams.addCertStore(localStore);
pkixParams.setRevocationEnabled(false);     // テストプログラムでは
                                            // CRLの検証を行わないようにします
params = pkixParams;
```

THE DOCUMENT COMPANY
FUJI XEROX

暗号プロバイダ

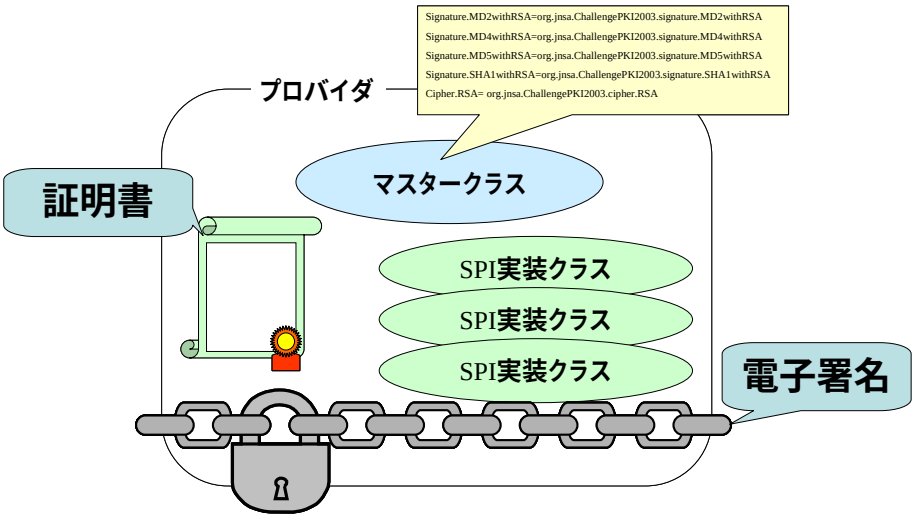
暗号プロバイダを作る ＝ CSPを作る

THE DOCUMENT COMPANY
FUJI XEROX



JAVAにおけるプロバイダの概念

THE DOCUMENT COMPANY
FUJI XEROX



マスタークラス

THE DOCUMENT COMPANY
FUJI XEROX

- 提供可能なアルゴリズム名と実装を結合させるためのプロパティを保持するクラス
- 実装は各フレームワークのSPI(Service Provider Interface)を継承したクラス
- Java.security.Securityへ登録
  - 各フレームワークはそれらを参照
- 実装はExample6のMasterClass.java

MasterClass.java

THE DOCUMENT COMPANY
FUJI XEROX

```
29: public MasterClass() {
30:         super(PROVIDER_NAME, PROVIDER_VERSION, PROVIDER_INFO);
31:
32:         AccessController.doPrivileged(new java.security.PrivilegedAction(){
33:                 public Object run() {
34:                         put("Cipher.Caesar",
                                "org.jnsa.ChallengePKI2003.Example6.CaesarCipher");
35:
36:                         return null;
37:                 }
38:         });
39:
40: }
```

Provaider情報を登録

おまじない

アルゴリズムの識別名
実装クラス名の登録

MasterClass.java

THE DOCUMENT COMPANY
FUJI XEROX

```
21: public class MasterClass extends Provider {
22: static protected final String  PROVIDER_NAME = "JNSA";
23: static protected final double PROVIDER_VERSION = 1.0;
24: static protected final String  PROVIDER_INFO
            = "This provider is sample implementation for ChallengePKI 2003";
```

- これらの情報は、Providerの登録情報である。
- 30行目のsuper()によりjava.security.Providerに登録される。
- Providerのインスタンスである自身は、java.security.Secuiryに登録される。

参考資料

THE DOCUMENT COMPANY
FUJI XEROX

- ASN.1 Standard
  - http://asn1.elibel.tm.fr/en/standards/index.htm
- DIGITAL SIGNATURE STANDARD(DSS)
  - http://www.itl.nist.gov/fipspubs/fip186.htm
- PKCS#1/5/7/8/12
  - http://www.rsasecurity.com/rsalabs/pkcs/
- Java 2 Platform, Standard Edition, 1.4.0 API**仕様**
  - http://Java.sun.com/j2se/1.4/ja/docs/ja/api/index.html
- Java **暗号化アーキテクチャ**API **の仕様およびリファレンス**
  - http://Java.sun.com/j2se/1.4/ja/docs/ja/guide/security/CryptoSpec.html
- Java Certification Path API **プログラマーズガイド**
  - http://Java.sun.com/j2se/1.4/ja/docs/ja/guide/security/certpath/CertPathProgGuide.html
- Java **暗号化拡張機能用プロバイダの実装方法**
  - http://Java.sun.com/j2se/1.4/ja/docs/ja/guide/security/jce/HowToImplAJCEProvider.html
- Java **暗号化拡張機能** (JCE) **リファレンスガイド**
  - http://Java.sun.com/j2se/1.4/ja/docs/ja/guide/security/jce/JCERefGuide.html

参考資料

THE DOCUMENT COMPANY
FUJI XEROX

- RFC 1310 - The MD2 Message-Digest Algorithm
  - http://www.ietf.org/rfc/rfc1319.txt
- RFC 1321 - The MD5 Message-Digest Algorithm
  - http://www.ietf.org/rfc/rfc1321.txt
- RFC 3174 - US Secure Hash Algorithm 1 (SHA1)
  - http://www.ietf.org/rfc/rfc3174.txt
- RFC 3280 - Internet X.509 Public Key Infrastructure Certificate and Certificate Revocation List (CRL) Profile
  - http://www.ietf.org/rfc/rfc3280.txt
- **平成**15**年度 情報処理振興事業協会**
  - **セキュリティ**API**に関する技術調査**：ー Part 1 ー **概要 アーキテクチャ・機能・暗号技術とアルゴリズム**」
  - **セキュリティ**API**に関する技術調査**：ー Part 2 ー Java JCE（Java Cryptographic Extensions）：**機能と利用法**