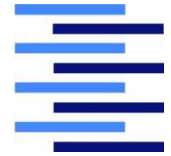

JAVA Kryptographie API

Weitere Infos:

Java Cryptography Architecture (JCA) Reference Guide

<http://docs.oracle.com/javase/8/docs/technotes/guides/security/>





JAVA Kryptographie API

- **Design-Anforderungen**

- Unabhängigkeit des API von speziellen kryptographischen Algorithmen (*RSA, DSA, DES, AES, ..*)
- Unterstützung verschiedener unabhängiger Implementierungen eines Algorithmus über dasselbe API

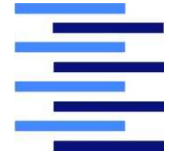
➔ Verwendung von „**Engine Classes**“

- ➔ Abstrakte Klassen stellen allgemeine Zugriffsmethoden zur Verfügung

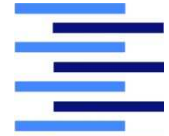
➔ **Provider-Konzept**

- ➔ Verschiedene Provider können konkrete Implementierungen für die abstrakten Klassen zur Verfügung stellen

JAVA Cryptography Architecture (JCA) / JAVA Cryptography Extension (JCE)

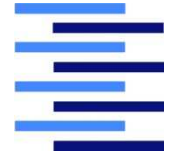


- JCA
 - Umfasst alle Funktionalitäten, die nicht den US-Exportbeschränkungen (bis 2000) unterlagen
 - Schlüsselerzeugung für asymmetrische Verfahren
 - Kryptographische Hashfunktionen / Signaturen
 - Zertifikate
 - Sichere Zufallszahlen
- JCE
 - Gehörte nicht zum Standard
 - Wurde außerhalb der USA mit schwachen Implementierungen ausgeliefert (→ Provider!)
 - Schlüsselerzeugung für symmetrische Verfahren
 - Verschlüsselung / Entschlüsselung



JCA – Engine Classes (java.security, java.security.cert)

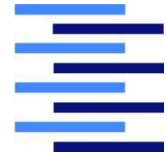
- **MessageDigest**
 - Berechnung eines Hashwerts mittels kryptographischer Hashfunktion
- **Signature**
 - Erzeugung und Verifikation einer Digitalen Signatur
- **KeyPairGenerator**
 - Generierung eines Schlüsselpaares für asymmetrische Verfahren (public + private key)
- **KeyFactory**
 - Format-Konvertierung eines Schlüssels eines asymmetrischen Verfahrens (public oder private key)
- **CertificateFactory**
 - Format-Konvertierung eines Zertifikats oder einer CRL
- **KeyStore**
 - Verwaltung einer Schlüsseldatenbank
- **AlgorithmParameters**
 - Verwaltung von Parametern für kryptographische Algorithmen (z.B. IV)
- **SecureRandom**
 - Erzeugung sicherer kryptographischer Pseudo-Zufallszahlen



JCE – Engine Classes (javax.crypto)

- **Cipher**
 - Ver- und Entschlüsselung
(für symmetrische und asymmetrische Verfahren!)
- **KeyGenerator**
 - Generierung eines geheimen Schlüssels
("geheim" = für ein symmetrisches Verfahren)
- **SecretKeyFactory**
 - Format-Konvertierung eines geheimen Schlüssels
- **KeyAgreement**
 - Austauschverfahren für geheime Schlüssel (Diffie-Hellmann)
- **Mac**
 - Berechnung eines Message Authentication Codes (HMAC)

Methoden zur Erzeugung einer Objekt-Instanz für eine Engine Class



<Typ> = Engine Class

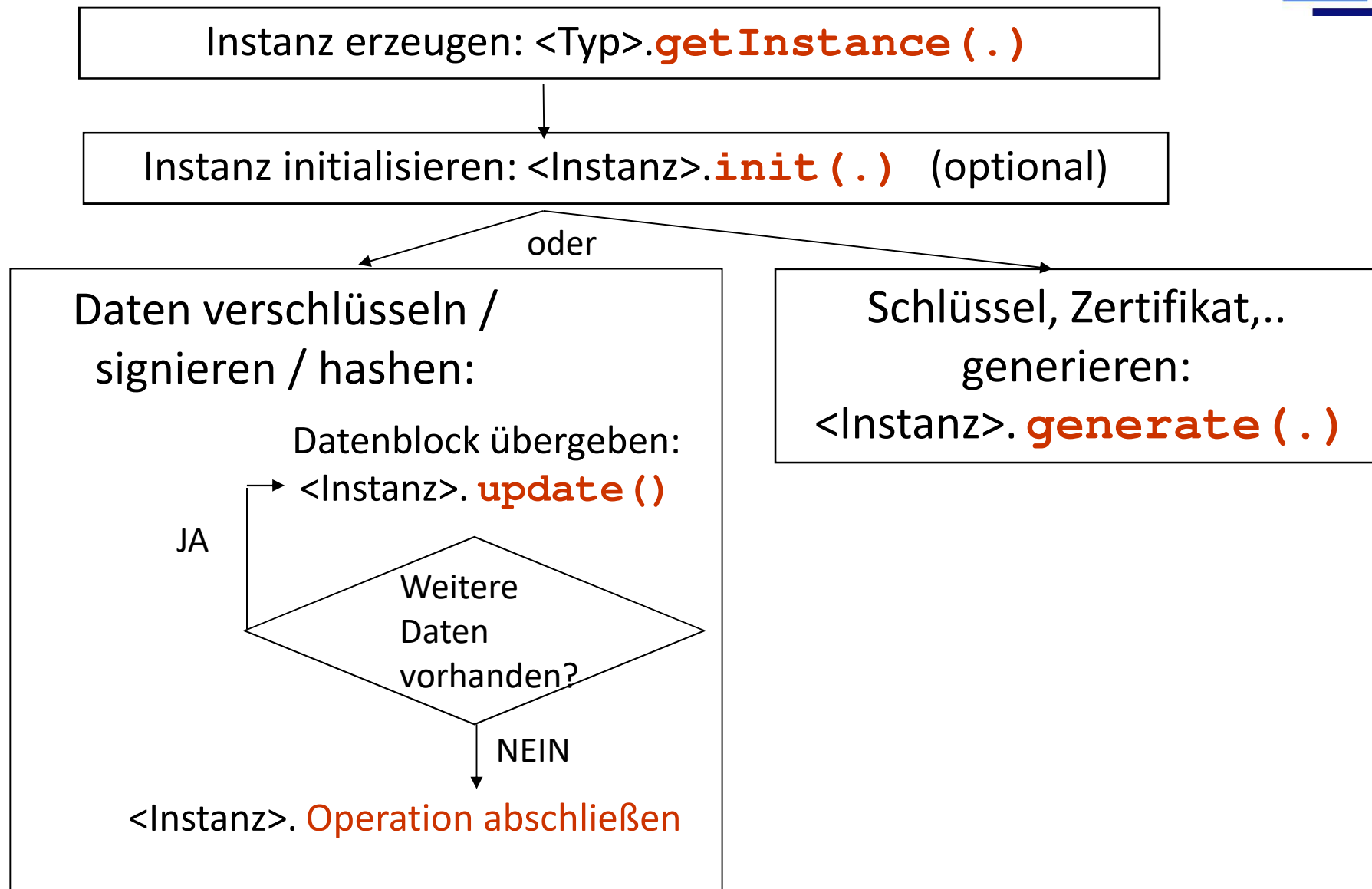
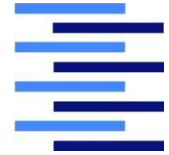
```
public static <Typ> getInstance(String algorithm,  
                                String provider)
```

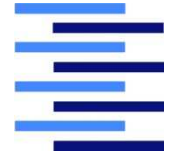
- Sucht nach einer Algorithmus-Implementierung des angegebenen Providers und erzeugt damit ein Objekt des Typs
- Beispiel: `MessageDigest md = MessageDigest.getInstance("SHA-256", "SUN");`

```
public static <Typ> getInstance(String algorithm)
```

- Sucht den ersten Provider, der eine Algorithmus-Implementierung liefert und erzeugt damit ein Objekt des Typs

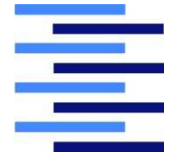
Typische Anwendung einer Engine-Class





Beispiele (1)

- **ShowProv.java**: Alle installierten Provider anzeigen
- **ShowMD.java**: Message Digest (kryptographischen Hashwert) für eine Datei berechnen und anzeigen
- **SignMessage.java**: Schlüsselpaar generieren, Nachricht signieren und zusammen mit der Signatur und dem öffentlichen Schlüssel in einer Datei speichern



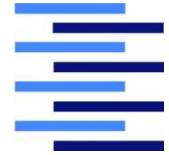
Schlüsselrepräsentationen

- **Abgeschlossene** Schlüssel
 - Interface: **Key** / **SecretKey**
 - Nach Erzeugung kein Zugriff auf Parameter möglich, sondern nur auf
 - Name des Algorithmus (z.B. „RSA“, „AES“)
 - Externe Repräsentation (Bytefolge)
 - Formatname der externen Repräsentation (z.B. X.509, PKCS#8)
- **Transparente** Schlüssel
 - Interface: **KeySpec** / **SecretKeySpec**
 - Zugriff über Schlüsselspezifikation (Key Specification)
 - Änderung der Parameter ist möglich!



KeyFactory - Classes

- **Aufgabe der Klassen**
 - **KeyFactory** (asymmetrische Schlüssell)
 - **SecretKeyFactory** (symmetrische Schlüssell)
- **ist die Konvertierung von Schlüsselrepräsentationen**
 - transparent → abgeschlossen
 - abgeschlossen → transparent
 - abgeschlossen → abgeschlossen



Beispiele (2)

- **ReadSignedFile.java:**
Nachricht, Signatur und Schlüssel aus Datei lesen und Signatur verifizieren
- **CipherEncryption.java:**
Verschlüsseln und Entschlüsseln von Daten