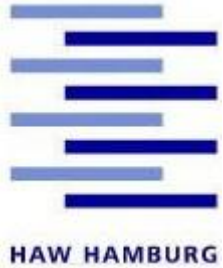
JAVA Kryptographie API



JAVA – Security (Guide to Features)

- General Security
  - Security Architecture (***Zugriffskontrolle**, überwachter Code, Sandbox-Modell, signierte Applets)*
  - **Cryptography Architecture JCA**
  - How to Implement a Provider for JCA
  - Policy Permissions (*Zugriffsrechtebeschreibungen inkl. Bedrohungsanalyse*)
  - Default Policy Implementation and Policy File Syntax *(Default-Zugriffsrechte)*
  - API for Privileged Blocks *(Dynamische Zuweisung von Rechten)*
  - **X.509 Certificates and Certificate Revocation Lists** *(Grundlagen)*
- **Certification Path** *(Verwaltung und Überprüfung von Zertifikaten, Zertifikatsketten und CRLs)*
- JAAS Authentication and Authorization Service *(Login-Module/Code-Ausführungsrechte)*
- Java GSS-API Generic Security Services *(Authentifikation in verteilten Systemen* ***à*** ***Kerberos**)*
- **JCE Cryptography Extension**
- JSSE Secure Socket Extension *(inkl. **SSL** 3.0-Implementierung)*

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

è Verwendung von „**Engine Classes**“

- Abstrakte Klassen stellen allgemeine Zugriffsmethoden zur Verfügung

è 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 (**à** 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
- **`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
- **`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

Methoden zur Erzeugung einer Objekt-Instanz für eine Engine Class

<Typ> = Engine Class

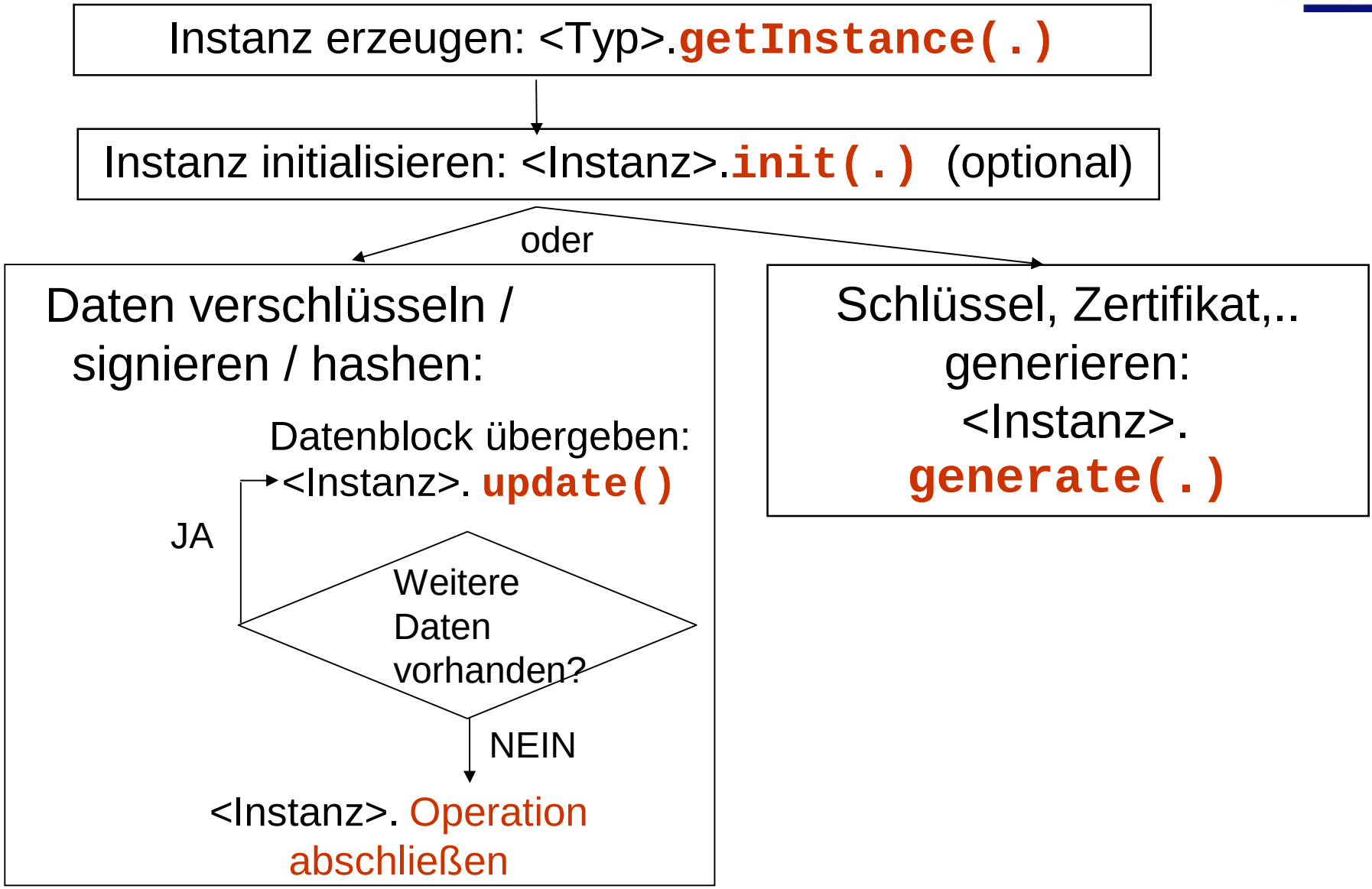
```
public static <Typ> getInstance(String algorithm,
                                String provider)
```

- Sucht nach einer Algorithmus-Implementierung des angegebenen Providers und erzeugt damit ein Objekt des Typs
- Beispiel:
```
MessageDigest md =
        MessageDigest.getInstance("MD5","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)

- **`Prov.java:`** Alle installierten Provider anzeigen
- **`MD.java:`** Message Digest für eine Datei berechnen und anzeigen
- **`SignMessage.java:`** Öffentlichen Schlüssel generieren, Nachricht signieren und zusammen mit der Signatur und dem öffentlichen Schlüssel in einer Datei speichern

Schlüsselrepräsentationen

- **Abgeschlossene** Schlüssel (providerabhängig)
  - Interface: **`Key / SecretKey`**
  - Nach Erzeugung kein Zugriff auf Parameter möglich, sondern nur auf
    - Name des Algorithmus (z.B. „RSA“, „DES“)
    - Externe Repräsentation (Bytefolge)
    - Formatname der externen Repräsentation (z.B. X.509, PKCS#8)
- **Transparente** Schlüssel (providerunabhängig)
  - Interface: **`KeySpec / SecretKeySpec`**
  - Zugriff über Schlüsselspezifikation (Key Specification)
  - Identifikation von externen Schlüssel möglich (z.B. auf einer Chipkarte)

KeyFactory - Classes

- Aufgabe der Klassen
  - **`KeyFactory`** (asymmetrische Schlüssel)
  - **`SecretKeyFactory`** (symmetrische Schlüssel)
- **Konvertierung von Schlüsselrepräsentationen**
  - transparent **à** abgeschlossen:

    Beispiel:

    1. Lesen der Schlüssel-Bytefolge aus einer Datei und Erzeugung einer entsprechenden Spezifikation (KeySpec)
    2. KeyFactory: Konvertierung der Spezifikation (KeySpec) in ein providerabhängiges Schlüsselobjekt
  - abgeschlossen **à** transparent
  - abgeschlossen **à** abgeschlossen
    - Wechsel der Provider-Implementierung möglich

Beispiele (2)

- **`ReadSignedFile.java:`**
  Nachricht, Signatur und Schlüssel aus Datei lesen und Signatur verifizieren

- **`CipherEncryption.java:`**
  Verschlüsseln und Entschlüsseln von Daten