Java Cryptography Architecture (JCA)

The JCA is a part of the Java platform that contains provider architecture and a set of APIs for:

- Digital signatures
- Secure random generators
- Message digests
- Message authentication codes
- Certificates and certificate validation
- Encryption (symmetric/asymmetric block/stream ciphers)
- Key generation and management, Key agreement
- Access control

The JCA can be obtained with Java Standard Edition Development Kit (JDK) from Sun's website java.sun.com/javase/. It includes two software components:

1. the framework that defines and supports cryptographic services (APIs)
2. the providers such as **Sun, SunRsaSign, SunJCE** which contain the actual implementations of cryptographic services

The JCA documentation is part of JDK documentation, based on JavaDoc, therefore easy to use and very good. (java.sun.com/javase/6/docs/api/)

The JCA APIs are historically split into two main packages:

1. *java.security* (MessageDigest, Signature, SecureRandom, Key, KeyStore, Policy, Permission, Provider, Security)
2. *javax.crypto* (Cipher, Mac, KeyGenerator, KeyAgreement, SecretKey)

The main design principles of JCA:

1. **Implementation independence** – Security services are implemented in providers, which are plugged into Java platform.
2. **Implementation interoperability** – Providers are interoperable across applications.
3. **Algorithm extensibility** – The Java platform supports the installation of custom providers that implement new services.

Algorithm independence is achieved by defining types of crypto-engines and classes that provide the functionality of these crypto-engines. (MessageDigest, Signature, KeyFactory, KeyPairGenerator, Cipher, Mac) These engine classes provide the interface to a specific type of crypto-service. The concrete instance of crypto-service is returned as a result of calling the factory method of engine class.

```
MessageDigest md = MessageDigest.getInstance(“SHA-1”, “Sun”);
MessageDigest md = MessageDigest.getInstance(“SHA-1”);
```

Provider Class

Cryptographic Service Provider (CSP) refers to a package or set of packages that supply a concrete implementation of a subset of the JCA API cryptography features. The Provider abstract class is the interface to such a package. Every CSP have to implement a subclass of Provider, in which the security service aliases are bound with concrete implementation class. It contains also name, version and info about CSP. The concrete implementation of crypto-service must be a subclass of Service Provider Interface (SPI) class. (MessageDigestSpi, SignatureSpi, CipherSpi)

Security Class

The Security class manages installed providers and security-wide properties in one place. (centralization) It only contains static methods and is never instantiated. The CSPs can be added and deleted dynamically using Security class methods addProvider(), insertProviderAt() and removeProvider() or statically by editing java.security file in JRE.

SecureRandom Class

It is an engine class that provides the functionality of Random Number Generator. SecureRandom object is created by calling the static factory method getInstance(). The implementation of SecureRandom attempts to completely randomize the internal state, but it also can be seed by programmer using setSeed() method. The random bytes can be obtained by nextBytes() method.

MessageDigest Class

It is an engine class that provides the functionality of crypto-secure message digest. The initialized MessageDigest object is created by calling the static factory method getInstance(). The MD can be fed with data using one of the update() methods and the result obtained by calling one of the digest() methods. The reset() method initializes MessageDigest object.

```
MessageDigest md = MessageDigest.getInstance(“SHA-1”, “Sun”);
byte[] hash = md.digest(“Hashovana sprava”.getBytes());
```

Signature Class

It is an engine class that provides the functionality of cryptographic digital signature algorithm. The Signature object is created by calling the static factory method getInstance() and is in uninitialized state. Before using it must be initialized with appropriate private key, public key or certificate using initSign() or initVerify() method. For signing are used update() and sign() methods and for verifying signatures update() and verify() methods.

The verify() methods return Boolean value indicating whether or not the encoded signature is authentic with data supplied to the update() methods.

```
Signature sg = Signature.getInstance("SHA1withRSA");
KeyPairGenerator kpg = KeyPairGenerator.getInstance("RSA");
kpg.initialize(1024);
KeyPair kp = kpg.genKeyPair();
sg.initSign(kp.getPrivate());
sg.update("Sprava na podpisanie".getBytes());
byte[] signature = sg.sign();
```

Cipher Class

It is an engine class that provides the functionality of a cryptographic cipher for encryption and decryption. The Cipher object is created by calling the static factory method getInstance(). The transformation string in getInstance() is of the form “algorithm/mode/padding”. The Cipher object have to be initialized with one of the init() methods taking Key or Certificate, operational mode (Encrypt, Decrypt, Wrap_key, Unwrap_key) and other algorithm parameters. The data can be encrypted in one step using doFinal() method or in multiple steps using update() method followed by doFinal() method. Wrap() and unwrap() methods provide wrapping and unwrapping of keys. The cipher parameters can be obtained using getIV(), getParameters() or getAlgorithm() methods.

```
Cipher c = Cipher.getInstance(“DESede/CBC/PKCS5Padding”);
KeyGenerator kg = KeyGenerator.getInstance("DESede");
SecureRandom sr = SecureRandom.getInstance("SHA1PRNG", "SUN");
kg.init(168, sr);
SecretKey sk = kg.generateKey();
c. init(Cipher.ENCRYPT_MODE, sk);
```

Digest and Cipher Stream Classes

The CipherInputStream, CipherOutputStream, DigestInputStream and DigestOutputStream are FilterInputStream or FilterOutputStream classes that encrypt, decrypt or digest the data passing through. They are composed of an InputStream or OutputStream and Cipher or MessageDigest.

Mac Class

It is an engine class that provides the functionality of message authentication code. Can be obtained using getInstance() factory method and initialized with key and algorithm parameters using init() method. The MAC can be computed using update() and doFinal() metods.

Key Interface and KeySpec Interface

Key interface defines keys with no direct access to the key material. SecretKey, PrivateKey and PublicKey interfaces extend the Key interface.

The keys are generally obtained through key generators such as KeyGenerator, KeyPairKenerator, certificates, key specifications using KeyFactory or a KeyStore.
KeySpec interface and its implementations provide transparent representation of the key. It means that it is possible to access each key material value individually.

Generators vs. Factories
Generators are used to generate brand new objects. Generators can be initialized in either algorithm independent or dependent way. On the other hand, factories are used to convert data from one existing object type to another. For example to create PublicKey from Certificate.

Keys can be stored in and managed in key stores using KeyStore interface and its implementations. The KeyAgreement class provides the functionality of key agreement protocols.
The package *java.security.cert* provides classes and interfaces for parsing and managing certificates, certificate revocation lists CRL and certificate paths. (classes Certificate, CertificateFactory, CRL, CertStore)

Java Secure Socket Extension (JSSE) is a part of JDK and provides access to SSL and TSL implementations. The JSSE API is available in *javax.net* and *javax.net.ssl* packages. The Sun's implementation of JSSE is included in JDK as the SunJSSE provider.

Simple Authentication and Security Layer (SASL) specifies a protocol for authentication and optional establishment of a security layer between client and server applications. The API for SASL is in *javax.security.sasl* package. The Sun's implementation of SASL is included in JDK as the SunSASL provider.

The JDK contains also providers that enable applications to access the native crypto-libraries on MS Windows platform (SunMSCAPI), interact with the PC/SC Smart Cards (SunPCSC) using Java Smart Card I/O API (*javax.smartcardio* package) and access native PKCS11 libraries (SunPKCS11). This providers don't contain cryptographic functionality.