APIs Cryptographiques

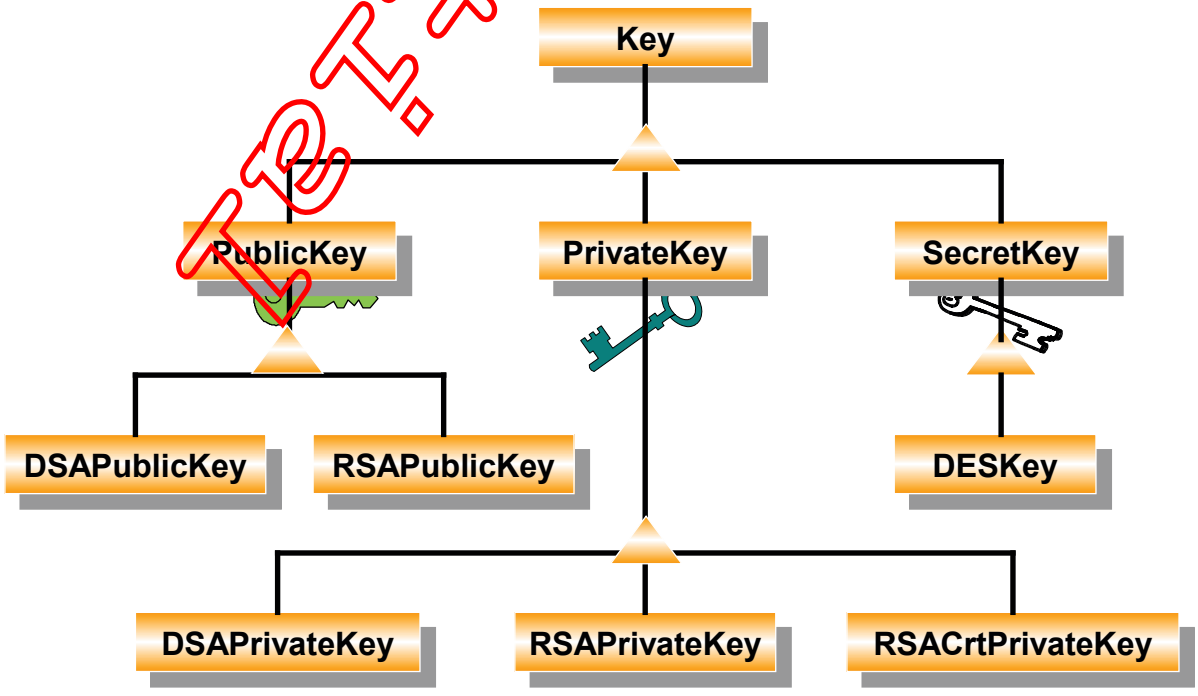
javacard.security

- Contains interfaces for managing keys
  - `Key, SecretKey, DESKey, PublicKey, RSAPublicKey, DSAPublicKey, PrivateKey, RSAPrivateKey, DSAPrivateKey, RSAPrivateCrtKey`
- Contains objects for realizing cryptographic operations
  - `KeyBuilder, Signature, MessageDigest, RandomData, CryptoException`

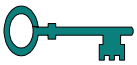
3.0 Classic Edition API

- AES: Advanced Encryption Standard (FIPS-197)
- SEED Algorithm Specification: KISA - Korea Information Security Agency Standard Names for Security and Crypto Packages
- SHA (SHA-1): Secure Hash Algorithm, as defined in Secure Hash Standard, NIST FIPS 180-1
- SHA-256,SHA-384,SHA-512: Secure Hash Algorithm, as defined in Secure Hash Standard, NIST FIPS 180-2
- MD5: The Message Digest algorithm RSA-MD5, as defined by RSA DSI in RFC 1321
- RIPEMD-160: as defined in ISO/IEC 10118-3:1998 Information technology - Security techniques - Hash-functions - Part 3: Dedicated hash-functions
- DSA: Digital Signature Algorithm, as defined in Digital Signature Standard, NIST FIPS 186
- DES: The Data Encryption Standard, as defined by NIST in FIPS 46-1 and 46-2
- RSA: The Rivest, Shamir and Adleman Asymmetric Cipher algorithm
- ECDSA: Elliptic Curve Digital Signature Algorithm
- ECDH: Elliptic Curve Diffie-Hellman algorithm
- AES: Advanced Encryption Standard (AES), as defined by NIST in FIPS 197
- HMAC: Keyed-Hashing for Message Authentication, as defined in RFC-2104

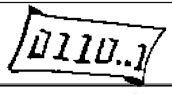
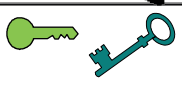
The key hierarchy in `javacard.security`



javacard.security

javacard.security	
Interfaces Summary	
Key	clearKey( ), getSize( ), getType( ), isInitialized( ).
SecretKey extends Key  DESKey extends SecretKey	getKey( ),setKey( ).
PrivateKey extends Key  RSAPrivateKey extends PrivateKey RSAPrivateCrtKey extends PrivateKey	getExponent(), getModulus(), setExponent (), setModulus(). set[P,Q,PQDP1,DQ1] (), get[P,Q,PQDP1,DQ1] ().
PublicKey extends Key  RSAPublicKey extendsPublicKey	getExponent(), getModulus(), setExponent (), setModulus().

javacard.security

javacard.security	
Class Summary	
KeyBuilder 	Allows to create a key object factory.
Signature 	Is the base class for signature algorithms.
RandomData ..8569765..	Generates random number.
MessageDigest 	Is the base class for hashing algorithms.
KeyPair 	Enable generation of a KeyPair

KeyBuilder

javacard.security	
KeyBuilder	
buildKey(type,length,encrypt)	Create a key with a specific type and length.
TYPE_DES TYPE_DES_TRANSIENT_DESELECT TYPE_DES_TRANSIENT_RESET TYPE_RSA_PUBLIC TYPE_RSA_PRIVATE TYPE_RSA_CRT_PRIVATE	
LENGTH_DES *(64 bits)* LENGTH_DES_2KEY *(128 bits)* LENGTH_DES_3KEY *(192 bits)* LENGTH_RSA_512 LENGTH_RSA_768 LENGTH_RSA_1024	

Signature

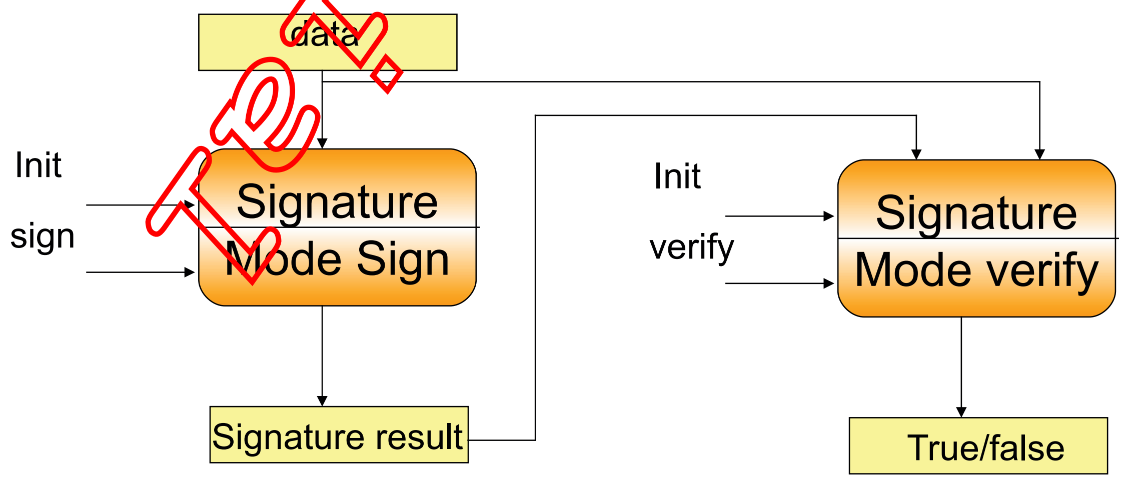
javacard.security	
Signature	**methods**
getInstance()	Creates a Signature object instance of the selected algorithm.
init()	Initializes the Signature object.
update()	Accumulates a signature of the input data.
sign()	Generates the signature of all/last input data.
verify()	Verifies the signature of all/last input data against the passed in signature.
getAlgorithm( )	Gets the Signature algorithm.
getLength( )	Returns the byte length of the signature.

	fields
ALG_DES_MAC4_NOPAD ALG_DES_MAC8_NOPAD ALG_RSA_SHA_PKCS1	
MODE_SIGN MODE_VERIFY	TheMode used in init( ) method.

javacard.security

- `KeyBuilder`
  - Creates non initialized cryptographic keys for signature and cipher algorithms
  - `buildKey` method create the key and returns a `Key` interface
- `Signature` class
  - Is an abstract class for signature algorithms
  - Implementations must extend this class and implement all the abstract methods,
  - Contains all signature constants value for `getInstance` algorithms parameters specification

Principe



Example of `Signature` for signature usage

- Once inside the applet:
  - Create a private key object using `builKey()` method

```
privrsakey =
(RSAPrivateKey)KeyBuilder.buildKey(TYPE_RSA_PRIVATE,LENGTH_RSA_512,false);
```

  - Initialize your private key object using `setExponent()` and `setModulus()` methods

```
privrsakey.setExponent(buffer_exp,offset_exp,length_exp);

privrsakey.setModulus(buffer_mod,offset_mod,length_mod);
```

  - Create a signature object using `getInstance( )` method

```
sgn = Signature.getInstance(ALG_RSA_SHA_PKCS1, false);
```

- At each usage:
  - Initialize your signature object using `init()` method

```
sgn.init(privrsakey,MODE_SIGN);
```

  - Compute your signature
    - feed the data using `update()` method
    - compute signature using `sign()` method

```
sgn.update(a1,offset1,length1);
...
sgn.sign(aX,offsetX,lengthX,sgn_buffer,sgn_offset);
```

Example of `Signature` for verification usage

- Once inside the applet:
  - Create a public key object using `builKey()` method

```
pubrsakey =
(RSAPublicKey)KeyBuilder.buildKey(TYPE_RSA_PUBLIC,LENGTH_RSA_512,false);
```

  - Initialize your public key object using `setExponent()` and `setModulus()` methods

```
pubrsakey.setExponent(buffer_exp,offset_exp,length_exp);

pubrsakey.setModulus(buffer_mod,offset_mod,length_mod);
```

  - Create a signature object using `getInstance( )` method

```
sgn = Signature.getInstance(ALG_RSA_SHA_PKCS1, false);
```

- At each usage:
  - Initialize your signature object using `init()` method

```
sgn.init(pubrsakey,MODE_VERIFY);
```

  - Compute your signature
    - feed the data using `update()` method
    - verify signature using `verify()` method

```
sgn.update(a1,offset1,length1);
...
sgn.verify(aX,offsetX,lengthX,sig_buffer,sig_offset,sig_length);
```

Another example of `Signature` usage

- Once inside the applet:

```
DESKey k =
(DESKey)KeyBuilder.buildKey( TYPE_DES,LENGTH_DES,false) ;
   k.setKey(buffer, offset, length) ;
   Signature s = Signature.getInstance( ALG_DES_MAC_NOPAD,
     false);
```

- At each usage:

```
   s.init(k, MODE_SIGN) ;
   s.update(in_buff, in_ofs, in_len) ;
   s.sign(in_buff, in_ofs, in_len, out_buff, out_ofs) ;
   s.init(k, MODE_VERIFY) ;
   s.update(in_buff, in_ofs, in_len) ;
   s.verify(in_buff, in_ofs, in_len, out_buff, out_ofs,
     out_len);
```


`MessageDigest`

javacard.security	
MessageDigest	
getInstance()	Creates a MessageDigest object instance of the selected algorithm.
update()	Accumulates a hash of the input data.
doFinal()	Generates a hash of all/last input data.
reset( )	Resets the MessageDigest object to the initial state for further use.
getAlgorithm( )	Gets the Message Digest algorithm.
getLength( )	Returns the byte length of the hash.
ALG_SHA ALG_MD5	

Example of `MessageDigest` usage

- Once inside the applet:
  - Create a message digest object using `getInstance( )` method

```
msgd = MessageDigest.getInstance(ALG_SHA, false);
```

- At each usage:
  - Re-Initialize your message digest object using `reset()` method

```
msgd.reset();
```

  - Compute your digest
    - feed the data using `update()` method
    - compute the message digest using `doFinal()` method

```
msgd.update(a1,offset1,length1);
...
msgd.doFinal(aX,offsetX,lengthX,msgd_buffer,msgd_offset);
```


RandomData

javacard.security	
RandomData	
getInstance() generateData() setSeed()	Creates a RandomData instance of the selected algorithm. Generates random data in a buffer. Seeds the random data generator*.
ALG_SECURE_RANDOM	

**software part which modifies the random number given by the hardware part to enhance security.*

Example of `RandomData` usage

- Once inside the applet:
  - Create a random number object using `getInstance( )` method

```
rd = RandomData.getInstance(RandomData.ALG_SECURE_RANDOM);
```

- At each usage:
  - If needed set a seed for your random number object using `setSeed()` method

```
rd.setSeed(seed_buffer, seed_offset, seed_length);
```

  - Generate random data using `generateData()` method

```
rd.generateData(rd_buffer,rd_offset,rd_length);
```


Example of `KeyPair` usage

- Once inside the applet:
  - Create a Key pair object by instancing the class

```
KeyPair kp = new KeyPair(ALG_RSA_CRT, (short) 1024);
```

  - Generate the Key pair using `genKeyPair( )` method

```
kp.genKeyPair();
```

- At each usage:
  - Public and private keys of the pair can be get using `getPublic()` and `getPrivate()` methods

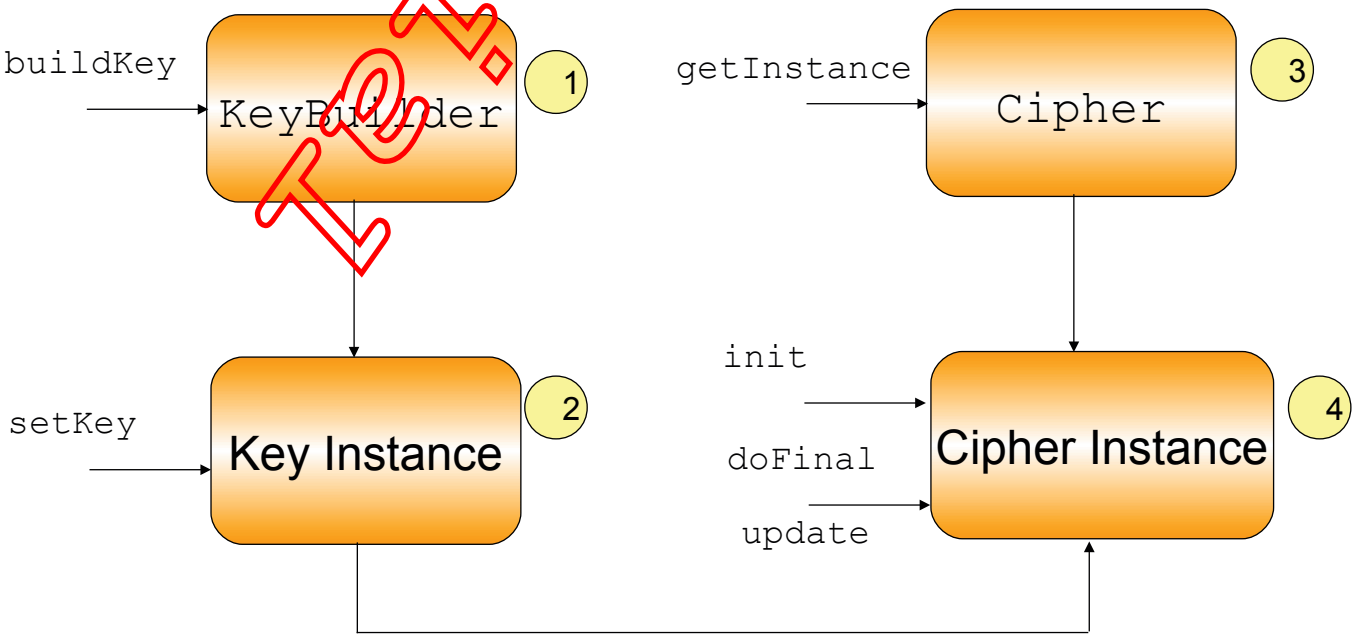
```
RSAPublicKey pub = (RSAPublicKey) kp.getPublic();
RSAPrivateCrtKey priv = (RSAPrivateCrtKey) kp.getPrivate();
```

javacardx.security.Cipher

javacardx.crypto	
Cipher	**methods**
getInstance()	Creates a Cipher object instance of the selected algorithm.
init()	Initializes the Cipher object with the appropriate Key and encrypt or decrypt mode.
update()	Generates encrypted/decrypted output from input data.
doFinal()	Generates encrypted/decrypted output from **all/last** input data.
getAlgorithm( )	Gets the Cipher algorithm.

	fields
MODE_DECRYPT MODE_ENCRYPT	To specify the mode used in the update() or doFinal() method
ALG_DES_CBC_NOPAD ALG_DES_ECB_NOPAD ALG_RSA_NOPAD	

Principe



Example of `Cipher` usage

- Once inside the applet:
  - Create a Cipher object using `getInstance()` method

```
cipher = Cipher.getInstance(ALG_RSA_NOPAD, false);
```

- At each usage:
  - Initialize your cipher object using `init()` method

```
cipher.init(pub,MOD_ENCRYPT); // pub is the PublicRSACrtKey
```

  - Compute your encrypted/decrypted data
    - feed the data using `update()` method
    - compute resulting data using `doFinal()` method

```
cipher.update(a1,offset1,length1);
...
cipher.doFinal(aX,offsetX,lengthX,cipher_buffer,cipher_offset);
```


Another example of `Cipher` usage

- Once inside the applet:

```
DESKey k = (DESKey) KeyBuilder.buildKey( TYPE_DES,
  LENGTH_DES, false) ;
k.setKey(buffer, offset, length) ;
c = Cipher.getInstance( ALG_DES_MAC_NOPAD, false);
```

- At each usage:

```
c.init(k, MODE_ENCRYPT) ;
s.update(in_buff, in_ofs, in_len) ;
s.doFinal(in_buff, in_ofs, in_len, out_buff, out_ofs) ;
```

Hints

- `getInstance (algorithm, external access)` method
  - “external access” boolean parameter
    - `true`: access from other applets via the Shareable Interface
    - `false`: access only from applets in the same package, if it is selected.
- `update()` method uses
  - temporary storage for intermediate results: used only if all input data cannot be contained in one byte array.

And some more…

- Package `javacardx.biometry`: Extension package that contains functionality for implementing a biometric framework on the Java Card platform.
  - `BioBuilder, BioException, BioTemplate, OwnerBioTemplate, SharedBioTemplate`
- Package `javacardx.framework.math`: Extension package that contains common utility functions for BCD math and parity computations.
  - `BCDUtil, BigNumber, ParityBit`
- `javacardx.apdu`: Extension package that enables support for ISO7816 specification defined optional APDU related mechanisms.
- `javacardx.tlv`: Extension package that contains functionality, for managing storage for BER TLV formatted data, based on the ASN.1 BER encoding rules of ISO/IEC 8825-1:2002, as well as parsing and editing BER TLV formatted data in I/O buffers.