Overview

A. Execution model

language features, sandbox, applet firewall (object isolation and sharing)

B. On-card Cryptography

algorithms and protocols, good cryptographic practice

C. Protecting against attacks

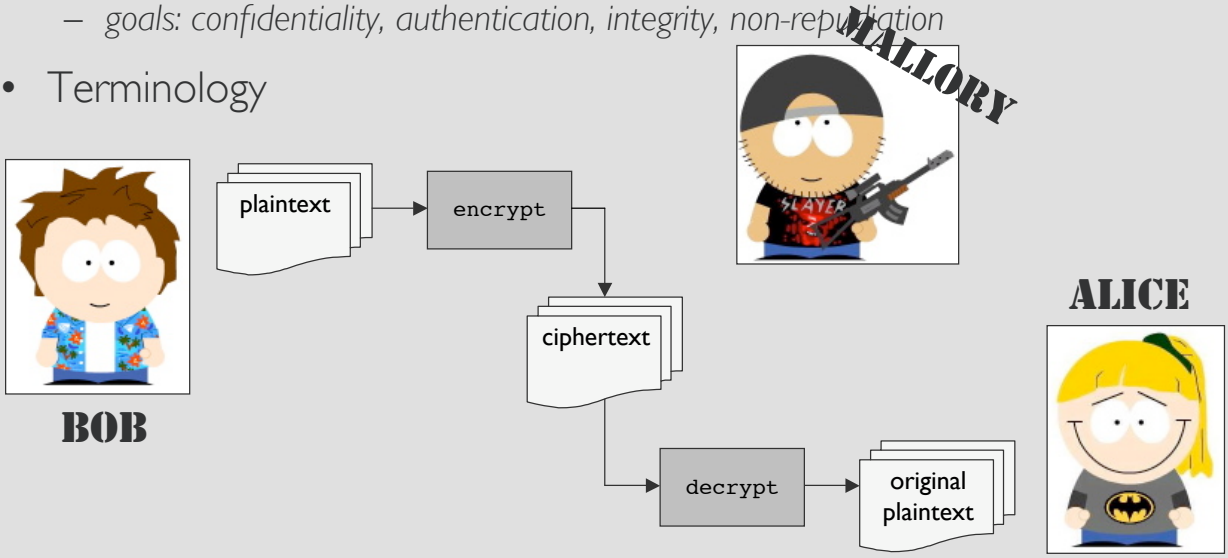
SPA/DPA, timing attacks, fault injection

B. On-Card: Cryptography Quick Tour

- Cryptography is a Tool
  - *cryptology vs. cryptanalysis*
  - *encryption/decryption, secure hashes, digital signatures, true randomness*
  - *goals: confidentiality, authentication, integrity, non-repudiation*
- Terminology
  - ***confidentiality:*** *the contents of a message should only be readable by the receiver (and sender)*
  - ***authentication:*** *a receiver should be able to ascertain a message's origin*
  - ***integrity:*** *a receiver should be able to verify that the message has not been modified in transit*
  - ***non-repudiation:*** *a sender should not be able to falsely deny later that he sent a message*

B. On-Card: Cryptography Quick Tour

- Cryptography is a Tool
  - *cryptology vs. cryptanalysis*
  - *encryption/decryption, secure hashes, digital signatures, true randomness*
  - *goals: confidentiality, authentication, integrity, non-repudiation*
- Terminology



B. On-Card: Cryptography Quick Tour

- Cryptography is a Tool
  - *cryptology vs. cryptanalysis*
  - *encryption/decryption, secure hashes, digital signatures, true randomness*
  - *goals: confidentiality, authentication, integrity, non-repudiation*
- "Good Cryptography" vs. "Security by Obscurity"
  - *algorithm is known and has been widely scrutinized*
  - *security rests only in the key*
  - *beware of "rolled-my-own" algorithms, a.k.a. "snake oil"*

B. On-Card: Cryptography Quick Tour

- Symmetric Cryptography
  - *a.k.a. "conventional" algorithms, single-key or secret-key algorithms*
  - *decryption key* **K'** *can be calculated from encryption key* **K** *and vice versa (usually* **K** = **K'***, so we use only* **K***)*

$$E_K(M) = C \qquad D_K(C) = M$$

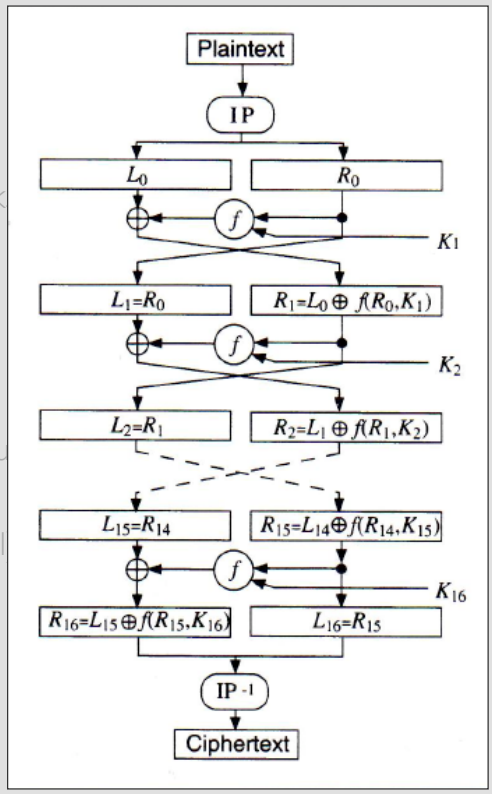
    - sender and receiver must agree on a key before communicating securely
      ☛ key distribution problem
  - ***stream ciphers:*** *operate on the plain text a single bit or byte at a time*
  - ***block ciphers:*** *operate on the plain text in blocks (groups of bits), e.g. 64 bits*

B. On-Card: Cryptography Quick Tour

- Symmetric Cryptography: DES
  - *since 1975*
  - *block cipher encrypting data in 64-bits blocks*
  - *key length is 56 bits (+ 8 bits parity checking)*
  - *algorithm*
    - initial permutation (optional)
    - block is broken into two halves, each one 32 bits longs
    - 16 rounds of identical operations, called function **f**
    - the two halves are joined again
    - inverse of the initial permutation (optional)

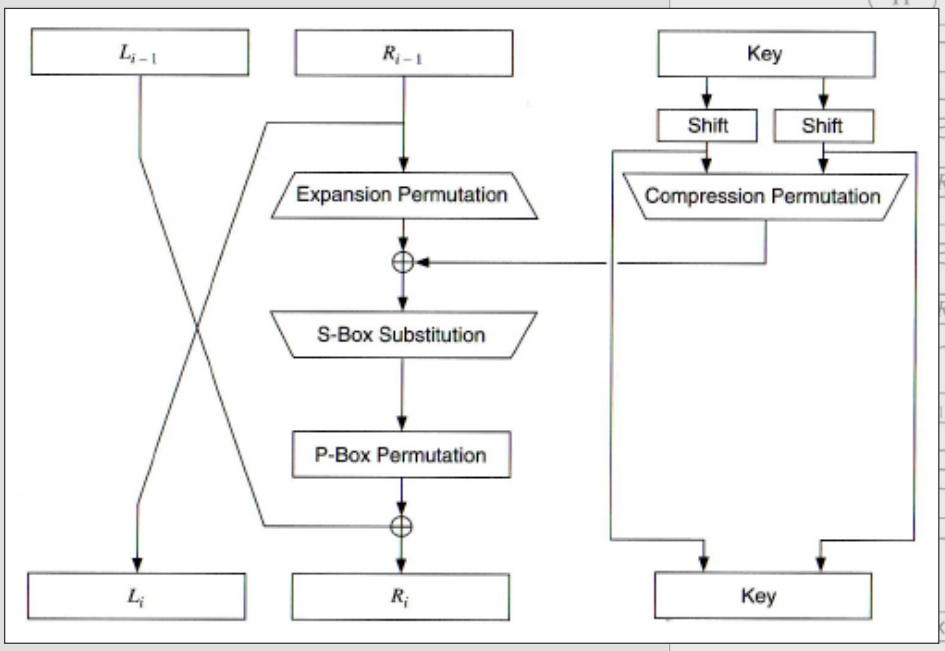
B. On-Card: Cryptography Quick Tour

- Symmetric Cryptography: DES
  - *since 1975*
  - *block cipher encrypting data in 64-bits block*
  - *key length is 56 bits (+ 8 parity checking)*
  - *algorithm*
    - initial permutation (optional)
    - block is broken into two halves, each one
    - 16 rounds of identical operations, called fu
    - the two halves are joined again
    - inverse of the initial permutation (optional



B. On-Card: Cryptography Quick Tour

- Symmetric Cryptography: DES

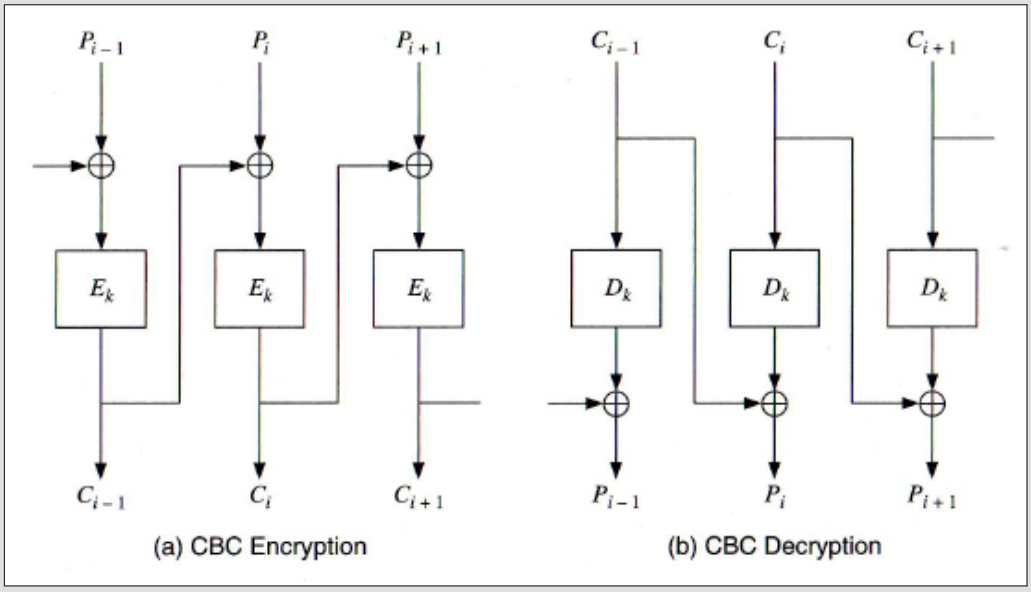


B. On-Card: Cryptography Quick Tour

- Symmetric Cryptography: DES (cont'd)
  - *ECB mode (electronic code book)*
    - a single block of plain text is converted into a single block of cipher text independently of order
    - it's theoretically possible to create a "code book"
    - prone to block substitution or replay
  - *CBC mode (cipher block chaining)*
    - before encrypting, the block of plain-text is XORed with the previous ciphertext block ☛ blocks are chained
    - first block is XORed with an initial vector (IV)
  - *Padding*
    - add some regular pattern to make the last block complete
    - use "ciphertext stealing"
  - *multiple DES ☛ 3DES w/ two key (112 bits)*

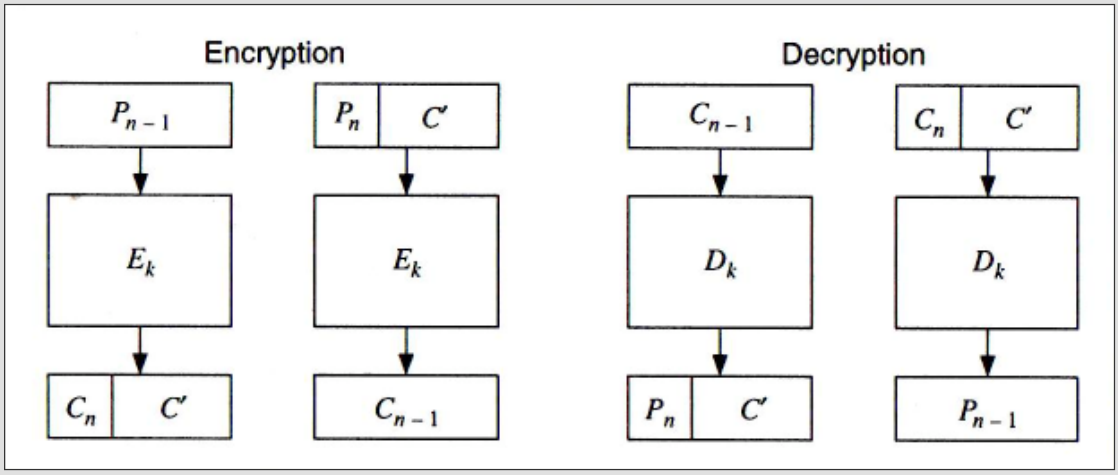
B. On-Card: Cryptography Quick Tour

- Symmetric Cryptography: DES (cont'd)
  - *ECB mode (electronic code book)*



B. On-Card: Cryptography Quick Tour

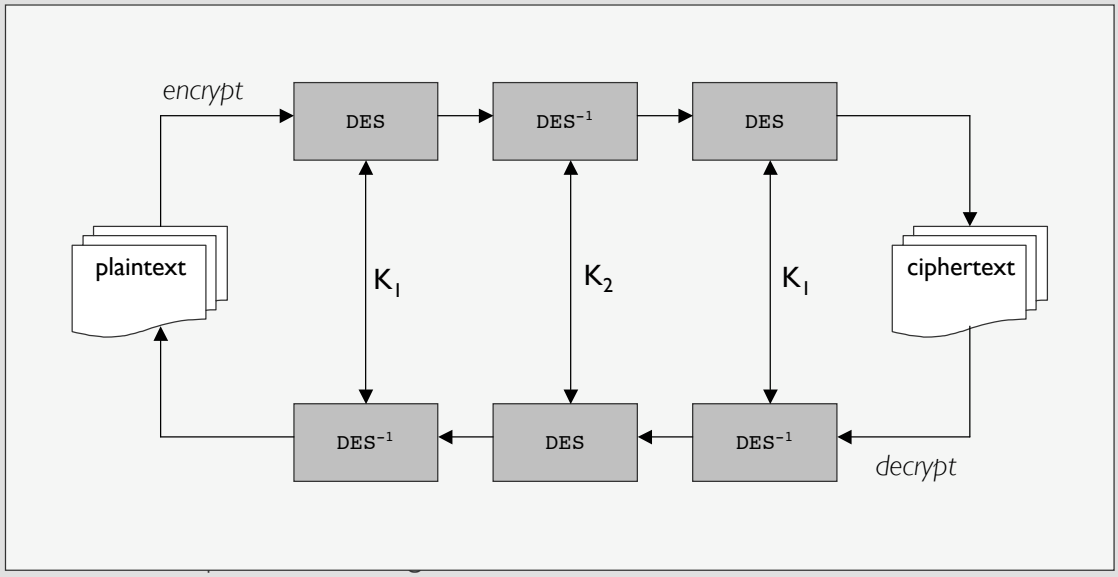
- Symmetric Cryptography: DES (cont'd)
  - *ECB mode (electronic code book)*



    - use "ciphertext stealing"
  - *multiple DES ☛ 3DES w/ two keys (112 bits)*

B. On-Card: Cryptography Quick Tour

- Symmetric Cryptography: DES (cont'd)



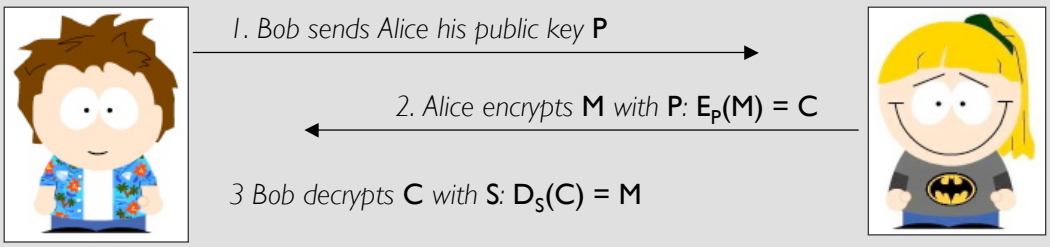
  - *multiple DES ☛ 3DES w/ two keys (112 bits)*

B. On-Card: Cryptography Quick Tour

- Symmetric Cryptography: AES
  - *chosen by NIST in 2001 as a replacement for DES*
  - *block cipher encrypting data in 128-bits blocks*
  - *key length is 128 bits, 192 bits, or 256 bits (may be extended in 32-bits steps)*
  - *works in EBC and CBC mode*
  - *time to crack (trying 255 keys per second)*
    - AES-192: 149 trillion years
    - 3DES: 4.6 billion years
  - *faster and less resource consuming than DES*

B. On-Card: Cryptography Quick Tour

- Public-Key Cryptography
  - *two keys: one public* **P** *and the other one private* **S**
  - *anyone with a public key can encrypt a message but not decrypt it*
  - *only the person with the private key can decrypt the message*
  - *it's computationally unfeasible to derive* **S** *from* **P**
  - *solves the key distribution problem*
  - *less efficient that symmetric key cryptography*



B. On-Card: Cryptography Quick Tour

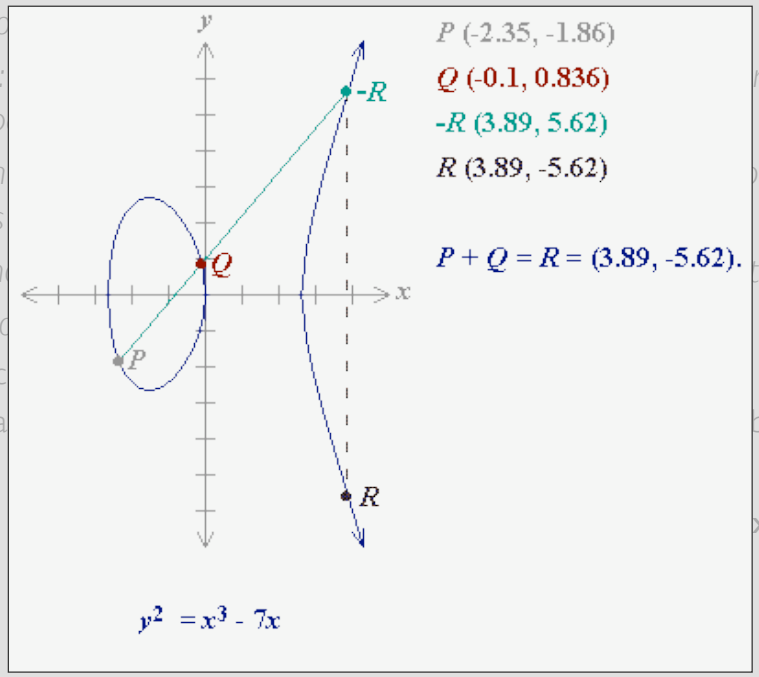
- Public-Key Cryptography: RSA
  - *invented by Ron Rivest, Adi Shamir, and Leonard Adleman (1978)*
  - *security from the difficulty of factoring large numbers (recovering the plaintext is equivalent to factoring the product of two primes)*
  - *key length: at least 1024 bits (as of 2004)*
  - *key generation*
    - choose two large prime numbers p, q and calculate $n = p \cdot q$
    - *randomly choose* e *such that* e *and* $(p-1) \cdot (q-1)$ *are relatively prime*
    - *calculate* d *such that* $e \cdot d = 1 \bmod (p-1) \cdot (q-1)$
    - *public key:* e, n *private key:* d
  - *algorithm*
    - encrypt: $c_i = m_i^e \bmod n$ with $c_i < 2^k < n$
    - decrypt: $m_i = c_i^d \bmod n$ since $c_i^d = (m_i^e)^d = m_i^{ed} = m_i \bmod n$

B. On-Card: Cryptography Quick Tour

- Public-Key Cryptography: Elliptic Curve Cryptography, ECC (1985)
  - *elliptic curve:* $y^2 + x \cdot y = x^3 + a_2 \cdot x + a_6$ *with* x, y *being complex, real, integers, polynomial basis, or any other kind of finite field element*
  - *security from the difficulty of the elliptic curve discrete logarithm problem (given points* F *and* $k \cdot F$ *find* k*)*
  - *key length: not less than 160 bits (which is comparable to 1024-bits RSA)*
  - *key generation*
    - choose curve C and a public base point B
    - choose a random number s as private key, then $P = s \cdot B$ is the public key
  - *algorithm*
    - encrypt: choose a random number k, calculate $R = k \cdot B$ and $S = (x_S, y_S) = k \cdot P$
encrypt message with a symmetric cipher using x_S as key
send R and the encrypted message
    - decrypt: from R calculate $S = b \cdot R = b \cdot (k \cdot B)$ to get x_S

B. On-Card: Cryptography Quick Tour

- Public-Key Cryptography: Elliptic Curve Cryptography, ECC (1985)
  - *alternative to*
  - *elliptic curve:* *real, integers, polynomial b*
  - *security from* *oblem (given points*
  - *key length: n* *ts RSA)*
  - *key generatio*
    - choose c
    - choose a *blic key*
  - *algorithm*
    - encrypt: $x_S, y_S) = k \cdot P$
    - decrypt:



B. On-Card: Cryptography Quick Tour

- Message Digest
  - *secure one-way hash function: arbitrary sized data* ⟶ *fixed length hash*
  - *reqs:* *easy to calculate but difficult to reverse the computation*
    collisions are rare (i.e., different data rarely generates same hash)
    difficult to fabricate data with some specific hash
  - *helps to achieve data integrity if sent along with the data*
  - *examples:*
    - MD5: generates an 128-bits hash
    - SHA-1: generates an 160-bits hash
  - *vulnerable to man-in-the-middle-attack* ☛ *message authentication code (MAC)*
    - calculate hash on both the input data and some key

B. On-Card: Cryptography Quick Tour

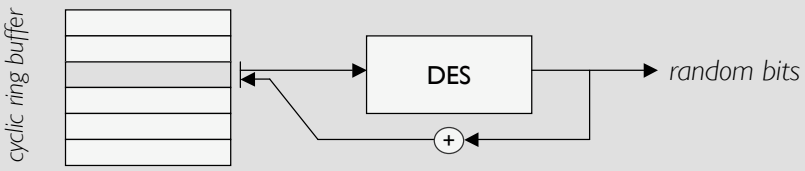
- Digital Signature
  - *computed by encrypting a message digest using public-key cryptography*
  - *provides authentication, integrity, and non-repudiation*

1. *calculates message digest* **H** *for message* **M**
2. *calculates* **Sig = E_S(H)**
3. *sends* **(M | Sig)**

calculates message digest **H** *for message* **M** 4.
calculates **H' = D_P(Sig)** 5.
compares **H** *and* **H'** 6.

B. On-Card: Cryptography Quick Tour

- Randomness
  - *crucial in cryptography: generating keys, blinding, and padding*
  - *random number generator should not be influenced by external conditions (e.g., temperature, voltage)*
  - *hardware: physical processes (e.g., unstable electronic circuit, radioactive decay)*
  - *software: pseudo-random generators using deterministic algorithms*
    - initialized with some "random" seed such as the current time
    - must be good enough to withstand statistical tests(e.g., χ^2 or FIPS 140-2)



B. On-Card: Usage Scenarios

- Cryptography is used for…
  - ***authentication:*** *an applet wants to authenticate the host and vice versa*
  - ***confidentiality:*** *sensitive data must be protected (on-card & communication)*
  - ***integrity:*** *secure communication against modification (e.g., via MAC)*
  - ***proof of authorization:*** *e.g., challenge/response between host and applet*
  - ***encryption:*** *e.g., digital signatures*
- But…
  - *avoid computationally expensive cryptographic mechanisms on voluminous data*
  - *always look at the big picture – the smart card is just one piece!*

B. On-Card: JavaCard Cryptography

- Functionality
  - *key storage and generation*
  - *encryption/decryption (symmetric and public key)*
  - *message digests*
  - *signatures*
  - *randomness*
- Designed for Algorithm Extensibility
  - *set of base classes/interfaces (e.g.,* `MessageDigest`*,* `Cipher`*)*
  - *instantiation via factory methods:* `getInstance()`

    e.g., `MessageDigest md5;`
    `md5 = MessageDigest.getInstance(MessageDigest.ALG_MD5, …);`

B. On-Card: `javacard.security`

- `javacard.security.Key`
  - *base interface for all keys*
- `javacard.security.KeyPair`
  - *container for a public/private key pair*
- `javacard.security.KeyBuilder`
  - *the key object factory*
- `javacard.security.MessageDigest`
  - *base class for hashing algorithms*
- `javacard.security.Signature`
  - *base class for signature algorithms*
- `javacard.security.RandomData`
  - *base class for random number generation*

B. On-Card: `javacard.security`

- `javacard.security.Key`
  - *base interface for all keys*
- `javacard.security.K`
  - *container for a public/priv*
- `javacard.security.K`
  - *the key object factory*
- `javacard.security.M`
  - *base class for hashing alg*
- `javacard.security.S`
  - *base class for signature al*
- `javacard.security.R`
  - *base class for random nur*

```
public interface Key {
   public void clearKey();

   public short getSize();
   public byte getType();

   public boolean isInitialized();
};
```

B. On-Card: javacard.security

- javacard.security.Key
  - *base interface for all keys*
- javacard.security.K
  - *container for a public/priv*
- javacard.security.K
  - *the key object factory*
- javacard.security.M
  - *base class for hashing alg*
- javacard.security.S
  - *base class for signature al*
- javacard.security.R
  - *base class for random nur*

```
public final class KeyPair {
   public KeyPair(byte algorithm, short keyLength);
   public KeyPair(PublicKey pub, PrivateKey priv);

   public void genKeyPair();
   public PrivateKey getPrivate();
   public PublicKey getPublic();
};


public interface PrivateKey extends Key {}

public interface PublicKey extends Key {}
```


B. On-Card: javacard.security

- javacard.security.Key
  - *base interface for all keys*
- javacard.security.K
  - *container for a public/priv*
- javacard.security.K
  - *the key object factory*
- javacard.security.M
  - *base class for hashing alg*
- javacard.security.S
  - *base class for signature al*
- javacard.security.R
  - *base class for random nur*

```
public class KeyBuilder {
   public Key buildKey(byte keyType, short keyLength,
                       boolean keyEncryption);
};
```

B. On-Card: javacard.security

- javacard.security.Key
  - *base interface for all keys*
- javacard.security.K
  - *container for a public/priv*
- javacard.security.K
  - *the key object factory*
- javacard.security.M
  - *base class for hashing alg*
- javacard.security.S
  - *base class for signature al*
- javacard.security.R
  - *base class for random nu*

```
public class MessageDigest {
   public static MessageDigest getInstance
      (byte algorithm, boolean externalAccess);

   public byte getAlgorithm();
   public byte getLength();

   public void reset();
   public void update
      (byte[] inbuf, short inoffset, short inlength);
   public short doFinal
      (byte[] inbuf, short inoffset, short inlength,
       byte[] outbuf, short outoffset);
};
```

Indicates that the instance will be shared among multiple applet instances and that the instance will also be accessable (via a Shareable interface) when the owner of the instance is not the currently selected applet.

B. On-Card: javacard.security

- javacard.security.Key
  - *base interface for all keys*
- javacard.security.K
  - *container for a public/priv*
- javacard.security.K
  - *the key object factory*
- javacard.security.M
  - *base class for hashing alg*
- javacard.security.S
  - *base class for signature al*
- javacard.security.R
  - *base class for random nu*

```
public class Signature {
   public static Signature getInstance
      (byte algorithm, boolean externalAccess);

   public byte getAlgorithm();
   public short getLength();

   public void init(Key key, byte mode);
   public void init
      (Key key, byte mode,
       byte[] buffer, short offset, short length);
   public void update
      (byte[] inbuf, short offset, short length);
   public short sign
      (byte[] inbuf, short inoffset, short inlength,
       byte[] sigbuf, short sigoffset);
   public short verify
      (byte[] inbuf, short inoffset, short inlength,
       byte[] sigbuf, short sigoffset, short siglength);
};
```

B. On-Card: `javacard.security`

- `javacard.security.Key`
  - *base interface for all keys*
- `javacard.security.K`
  - *container for a public/priv*
- `javacard.security.K`
  - *the key object factory*
- `javacard.security.M`
  - *base class for hashing alg*
- `javacard.security.S`
  - *base class for signature al*
- `javacard.security.R`
  - *base class for random nu*

```
public class RandomData {
   public static RandomData getInstance(byte algorithm);

   public void setSeed
      (byte[] buffer, short offset, short length);
   public void generateData
      (byte[] buffer, short offset, short length);
};
```


B. On-Card: `javacardx.crypto`

- `javacardx.crypto.Cipher`
  - *base class for all cipher algorithms*

```
public class Cipher {
   public static Cipher getInstance
      (byte algorithm, boolean externalAccess);

   public void init(Key key, byte mode);
   public void init
      (Key key, byte mode,
       byte[] buffer, short offset, short length);
   public short update
      (byte[] inbuf, short offset, short length
       byte[] outbuf, short outoffset);
   public short doFinal
      (byte[] inbuf, short inoffset, short inlength,
       byte[] outbuf, short outoffset);
};
```

B. On-Card: Examples

- Key Building and Generation

```
// persistent/transient DES key
DESKey desKey;
desKey = (DESkey)KeyBuilder.buildKey(KeyBuilder.TYPE_DES, // TYPE_DES_TRANSIENT_RESET
                                     KeyBuilder.LENGTH_DES3_2KEY,false);

desKey.setKey(...);


// RSA private key
RSAPrivateKey rsaPrivate;
rsaPrivate = (RSAPrivateKey)KeyBuilder.buildKey(KeyBuilder.TYPE_RSA_PRIVATE,
                                                KeyBuilder.LENGTH_RSA_1024,false);
rsaPrivate.setExponent(...);
rsaPrivate.setModulus(...);


// generate an RSA key pair
KeyPair rsaPair;
rsaPair = new KeyPair(KeyPair.ALG_RSA,KeyBuilder.LENGTH_RSA_1024);
rsaPair.genKeyPair();
```


B. On-Card: Examples

- Cipher

```
// allocate Cipher engine
Cipher cipher;
cipher = Cipher.getInstance(Cipher.ALG_DES_CBC_NO_PAD,false);

// init for DES encryption/decryption
cipher.init(desKey,Cipher.MODE_ENCRYPT);
           cipher.init(desKey,Cipher.MODE_DECRYPT);

// feed the data and encrypt/decrypt
short ofs = cipher.update(m1,(short)0,(short)(m1.length),outbuf,(short)0);
ofs += cipher.update(m2,(short)0,(short)8,outbuf,ofs);

cipher.doFinal(m3,(short)0,(short)(m3.length),outbuf,ofs);
```

B. On-Card: Examples

- Message Digest

```
// allocate SHA message digest
MessageDigest sha;
sha = MessageDigest.getInstance(MessageDigest.ALG_SHA,false);

...

// feed data and get hash
sha.update(m1,(short)0,(short)(m1.length));
sha.update(m2,(short)0,(short)8);

sha.doFinal(m3,(short)0,(short)(m3.length),hash,(short)0);
```


B. On-Card: Examples

- Signature

```
// allocate RSA/SHA signature object
Signature signature;
signature = Signature.getInstance(Signature.ALG_RSA_SHA_ISO9796,false);

// specifiy the key to use and whether to sign or to verify
signature.init(privateKey,Signature.MODE_SIGN);
          signature.init(publicKey,Signature.MODE_VERIFY);

// feed the data and sign/verify
signature.update(s1,(short)0,(short)(s1.length));

signature.sign(s2,(short)0,(short)(s2.length),sigbuf,(short)0);
          if (!signature.verify(s2,(short)0,(short)(s2.length),sigbuf,sigofs,siglen))
            ISOException.throwIt(SW_WRONG_SIGNATURE)
```

B. On-Card: Examples

- Random

```
// allocate Random engine
Random rnd;
rnd = Random.getInstance(RandomData.ALG_SECURE_RANDOM); // ALG_PSEUDO_RANDOM

// set seed
rnd.setSeed(seed,seedOffset,seedLength);

// generate random data
rnd.generateData(rndbuffer,rndoffset,rndlength);
```


DEMO