Java Cryptography Extension

Didier DONSEZ

Université Joseph Fourier

IMA –IMAG/LSR/ADELE

Didier.Donsez@imag.fr

Motivation

- Fournit une API standard pour l'usage de fonctions cryptographique
  - Chiffrage, signature, …
- Plusieurs Cryptography Service Providers (CSP) cohabitent (priorité du choix)
  - SPI : Interface pour implanter un provider
  - SUN fournit un Provider de référence
  - Conçu pour être exporté en dehors des USA

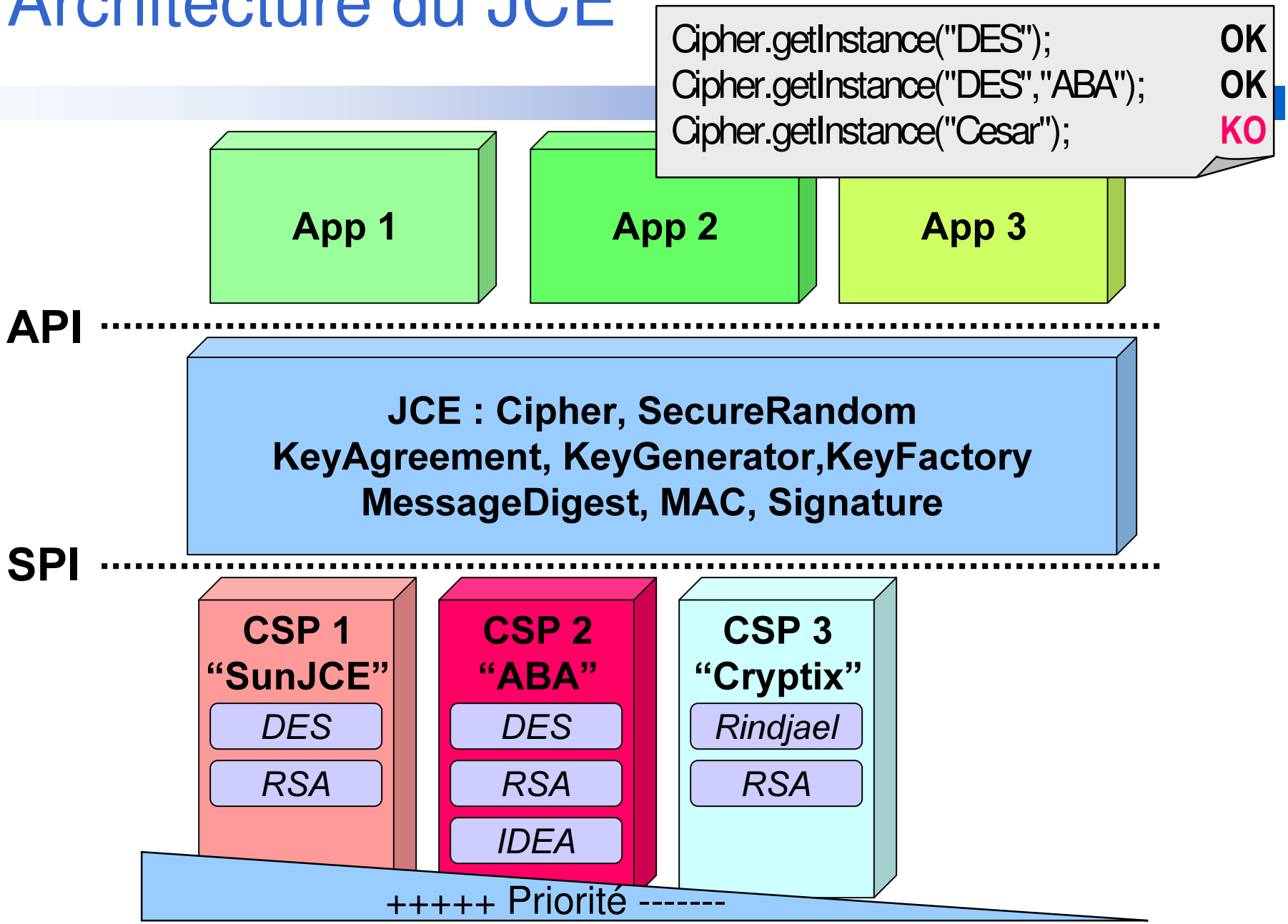
Principes

- Indépendance
- Intéropérabilité
- Extensibilité
  - Nouveaux algorithmes (AES par exemple)

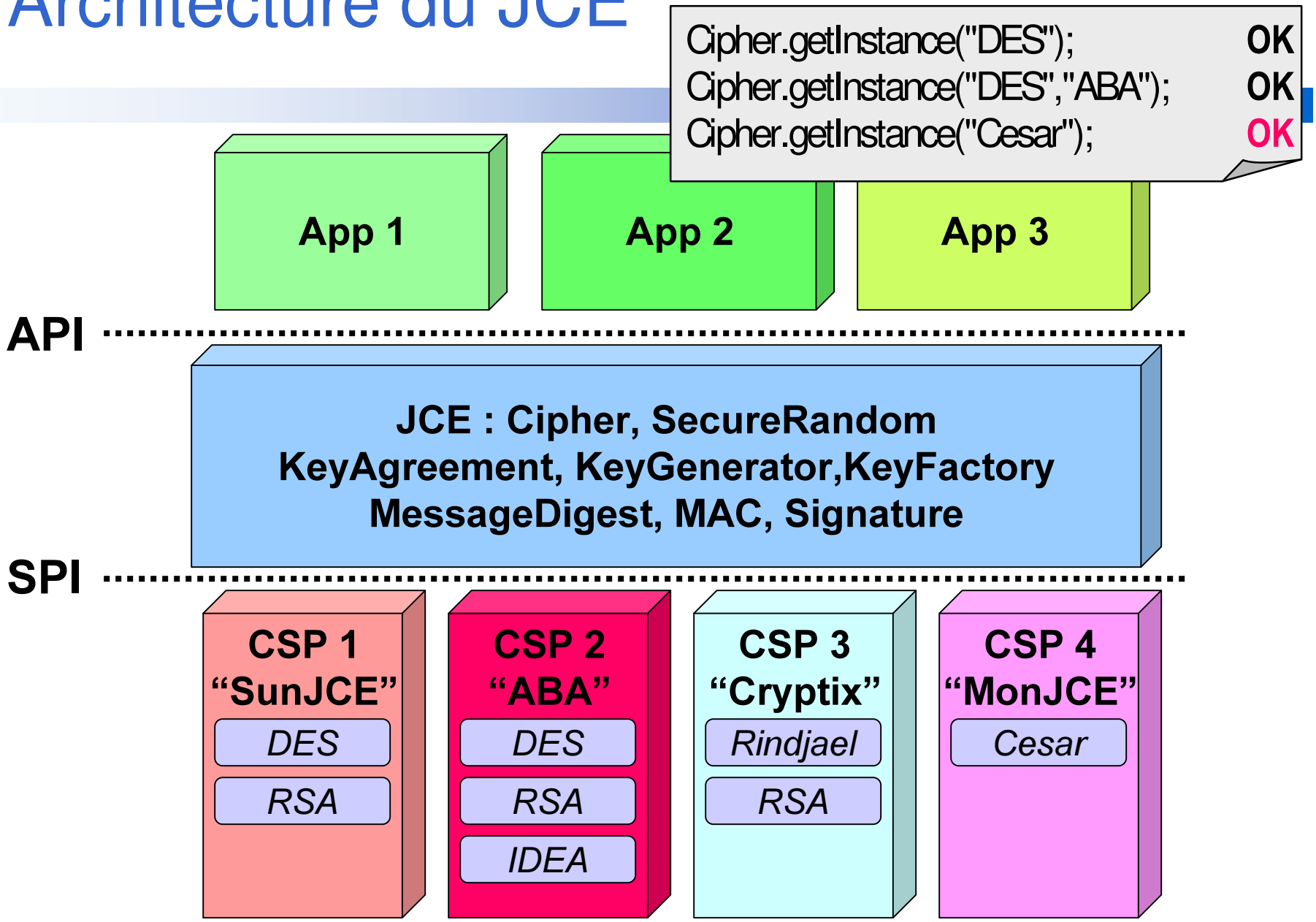
- Architecture
  - Providers & Key Management
- Concepts
  - Sélection d'implémentations chez les Providers
  - fabriques pour l'obtention des instances
    - XXX.getInstance(algo)

      // le premier qui offre l'implantation dans la liste des providers
    - XXX.getInstance(algo,provider)

Architecture du JCE



Architecture du JCE



Installation

- Chargez et décompressez le JCE de Sun et d'autres providers (ABA, …)
- Ajoutez les JAR
  - comme des extensions du JDK
  - ou au $CLASSPATH
- Positionnez les permissions pour le JCE

Add to <jre home>\lib\security\java.policy *the following*

```
grant codeBase "file:/work/sunjce_provider.jar" {
        permission java.io.FilePermission
          "/jdk1.2.2/jre/lib/ext/jce1_2_1.jar", "read";
        permission java.lang.RuntimePermission
          "getProtectionDomain";
        permission java.security.SecurityPermission
          "putProviderProperty.SunJCE";
      };
```

Installation

- Enregistrez les providers (selon leurs priorités)
- Statiquement
  - Fichier *($JAVA_HOME/jre/lib/security/java.security)*

```
security.provider.1=com.sun.crypto.provider.SunJCE
security.provider.2=au.net.aba.crypto.provider.ABAProvider
security.provider.3=cryptix.jce.provider.CryptixCrypto
security.provider.4=…
```

- A l'exécution

```
java.security.Security.addProvider(new com.sun.crypto.provider.SunJCE());
java.security.Security.addProvider(new au.net.aba.crypto.provider.ABAProvider());
…
```

Les classes et interfaces de l'API

- Les packages
  - java.security
  - java.security.cert
  - javax.crypto
  - javax.crypto.interfaces
  - javax.crypto.spec
- Les classes et interfaces
  - Provider
  - Security
  - Cipher
  - SecureRandom
  - Key
  - KeyPair
  - KeyPairGenerator MessageDigest
  - MAC
  - Signature
  - KeyStore
  - Certificate (java.security.cert.Certificate)
  - CipherSpi

SecureRandom
Générateur de nombres aléatoires

- Motivation
  - strong pseudo-random number generator (PRNG)
  - pour la génération de clés, dans les challenges, …
- Obtenir/construire une instance d'un générateur
  - **static SecureRandom getInstance(String algorithm)**
  - **static SecureRandom getInstance(String algorithm, String provider)**
    - algorithm = SHA1PRNG,
- Générer la graine (seed)
  - **byte[] generateSeed(int numBytes) // nombre d'octets de la graine**
- Initialiser et réinitialiser la graine du générateur
  - **byte[] generateSeed(int numBytes) // nombre d'octets de la graine**
  - **static byte[] getSeed(int numBytes)**
  - **void setSeed(byte[] seed) // avec seed.size octets de seed**
  - **void setSeed(long seed) // avec les 8 octets du long**
- Générer un nombre pseudo-aléatoire
  - **int next(int numBits) // généré numBits bits de poids faible aléatoire**
  - **// et 32-numBits bits de poids fort à 0**
  - **void nextBytes(byte[] bytes)**

Remarque sur la génération des nombres aléatoires (PRNG)

- Motivation
  - à la génération de clés
  - Aléa de session
- Risque
  - attaque brute de la recherche du nombre généré à partir d'hypothèse sur la graine
- Importance de la graine
  - Horloge de la machine n'est pas suffisante
  - Doit être complétée par
    - Memory Statistics, Process Statistics,
    - Mouse Movement, Keystroke Timing, …
- Voir
  - **http://www.rsasecurity.com/solutions/developers/whitepapers/Article4-PRNG.pdf**
  - **ftp://ftp.rsasecurity.com/pub/pdfs/bull-1.pdf**

Génération des clés

- Publique/privée
  - java.security.KeyPairGenerator
    - Crée une (ou des) clés à partir d'un objet existant ()
  - java.security.KeyFactory
    - Convertit (et verifie) une (ou des) clés à partir d'un objet existant ()
    - Exemple

```
X509EncodedKeySpec bobPubKeySpec
                    = new X509EncodedKeySpec(bobEncodedPubKey);
KeyFactory keyFactory = KeyFactory.getInstance("DSA");
PublicKey bobPubKey = keyFactory.generatePublic(bobPubKeySpec);
Signature sig = Signature.getInstance("DSA");
sig.initVerify(bobPubKey); sig.update(data); sig.verify(signature);
```

- Secrète
  - javax.crypto.SecretKeyFactory
    - Convertit (et verifie) une (ou des) clés à partir d'un objet existant ()
- Commun
  - javax.crypto.KeyGenerator

Générateur de clés privées/publiques

- Interface java.security.Key
  - PrivateKey, PublicKey
  - DSAPrivateKey, DSAPublicKey,RSAPrivateCrtKey, RSAPrivateKey, RSAPublicKey
  - Méthodes : String getAlgorithm(), byte[] getEncoded(), String getFormat()
- Obtenir/counstruire une instance d'un générateur **KeyPairGenerator**
  - **static KeyPairGenerator getInstance(String algorithm)**
  - **static KeyPairGenerator getInstance(String algorithm, String provider)**
    - algorithm =,
- Initialiser le générateur
  - **void initialize(AlgorithmParameterSpec params)**
  - **void initialize(AlgorithmParameterSpec params, SecureRandom random)**
  - **void initialize(int keysize)**
  - **void initialize(int keysize, SecureRandom random)**
    - **// Initializes the key pair generator for a certain keysize**
    - **// with the given source of randomness (and a default parameter set).**
- Générer une paire de clés **KeyPair**
  - **KeyPair genKeyPair()**
- Récupérer les clés de **KeyPair**
  - **PrivateKey getPrivate()**
  - **PublicKey getPublic()**

Génération des clés privées/publiques

```
// Random number
SecureRandom random = SecureRandom.getInstance("SHA1PRNG", "SUN");

// Digital Signature Algorithm
KeyPairGenerator keyGen = KeyPairGenerator.getInstance("DSA", "SUN");
int bitsize=1024;
keyGen.initialize(bitsize, random);

// Key pair
KeyPair pair = keyGen.generateKeyPair();
PrivateKey priv = pair.getPrivate();
PublicKey pub = pair.getPublic();
```

Convertion des clés privées/publiques

```
KeyFactory fact = KeyFactory.getInstance("RSA", "ABA");

    // fabriquer la clé privée
AsciiEncodedKeySpec privKeySpec = new AsciiEncodedKeySpec(args[0]);
PrivKey keyPriv = fact.generatePrivate(privKeySpec);
    // chiffrer avec la clé privée
Cipher enc = Cipher.getInstance("RSA/ECB/PKCS1Padding", "ABA");
enc.init(Cipher.ENCRYPT_MODE, keyPriv, rand);

    // fabriquer la clé publique
AsciiEncodedKeySpec pubKeySpec = new AsciiEncodedKeySpec(args[1]);
PublicKey keyPub = fact.generatePublic(pubKeySpec);
    // déchiffrer avec la clé publique
Cipher dec = Cipher.getInstance("RSA/ECB/PKCS1Padding", "ABA");
dec.init(Cipher.DECRYPT_MODE, keyPub);
```

Cipher (1/2)

- Fonction
  - De chiffrage/déchiffrage à clé symétrique (SKC) ou asymétrique (DSK)
- Obtenir une instance
  - **static Cipher getInstance(String algorithm)**
  - **static Cipher getInstance(String algorithm, String provider)**
    - algorithm spécifie l'algorithme / le mode / le « padding » (complétion)
    - Ex: "DES/CBC/PKCS5Padding", "DES/CFB8/NoPadding", "DES/OFB32/PKCS5Padding", "RSA/ECB/PKCS1Padding" …
- Initialiser avec l'opération (ENCRYPT_MODE | DECRYPT_MODE) et la clé
  - **void init(int opmode, Key key)**
  - **void init(int opmode, Key key, AlgorithmParameters params)**
  - **void init(int opmode, Key key, AlgorithmParameterSpec params)**
  - **void init(int opmode, Key key, AlgorithmParameterSpec params, SecureRandom random)**
  - **void init(int opmode, Key key, AlgorithmParameters params, SecureRandom random)**
  - **void init(int opmode, Key key, SecureRandom random)**

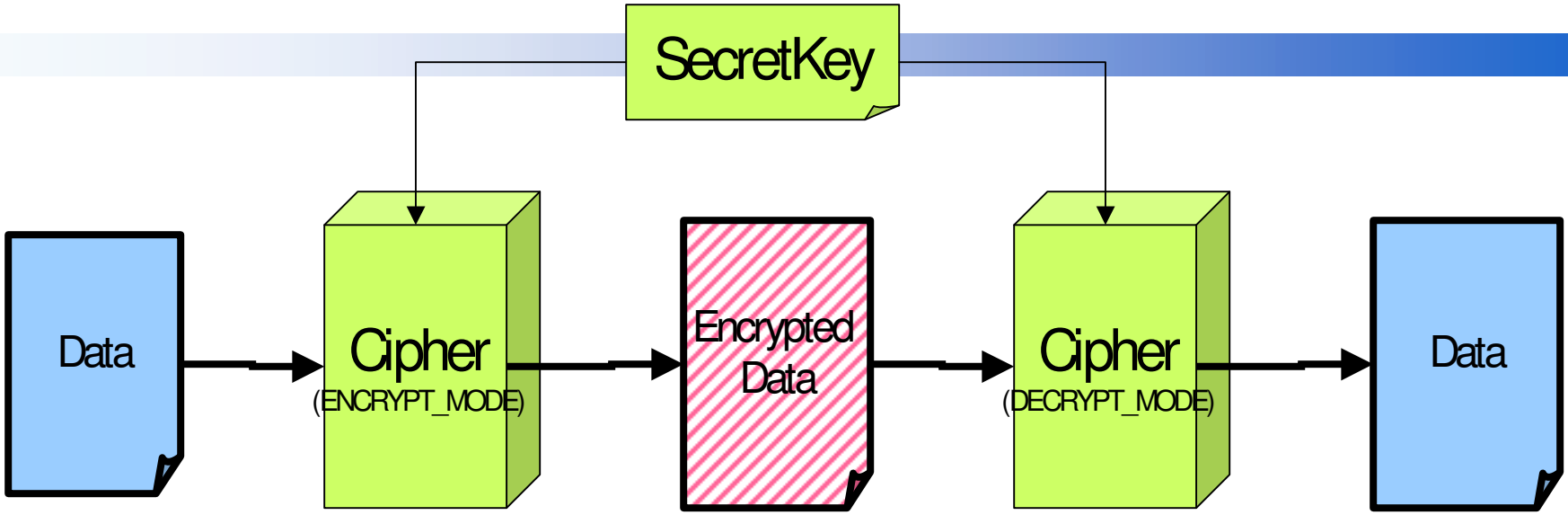
Cipher (2/2)

- Chiffrer/déchiffer l'entrée
  - **byte[] update(byte[] input)**
  - **byte[] update(byte[] input, int inputOffset, int inputLen)**
  - **int update(byte[] input, int inputOffset, int inputLen, byte[] output)**
  - **int update(byte[] input, int inputOffset, int inputLen, byte[] output, int outputOffset)**
- Finaliser
  - (car il reste des données qui n'ont pas été chiffrées dans l'instance)
  - **byte[] doFinal()**
  - **byte[] doFinal(byte[] input)**
  - **int doFinal(byte[] output, int outputOffset)**
  - **byte[] doFinal(byte[] input, int inputOffset, int inputLen)**
  - **int doFinal(byte[] input, int inputOffset, int inputLen, byte[] output)**
  - **int doFinal(byte[] input, int inputOffset, int inputLen, byte[] output, int outputOffset)**
- Extra
  - **String getAlgorithm(); Provider getProvider(); AlgorithmParameters getParameters()**
  - **Int getBlockSize() // Returns the block size (in bytes).**
  - **int getOutputSize(int inputLen)**
    - **// Returns the length in bytes that an output buffer would need to be in order to hold the result of the next update or doFinal operation, given the input length inputLen (in bytes).**

javax.crypto.CipherInputStream
javax.crypto.CipherOutputStream

- Motivation
  - Appliquer une fonction de chiffrage/déchiffrage sur un InputStream ou un OutputStream
  - étendent java.io.FilterInputStream et java.io.FilterOutputStream
- Constructeur
  - CipherInputStream(java.io.InputStream is, Cipher ciph)
  - CipherOutputStream(java.io.OutputStream is, Cipher ciph)
    - ciph doit être complétement initialisé

Exemples

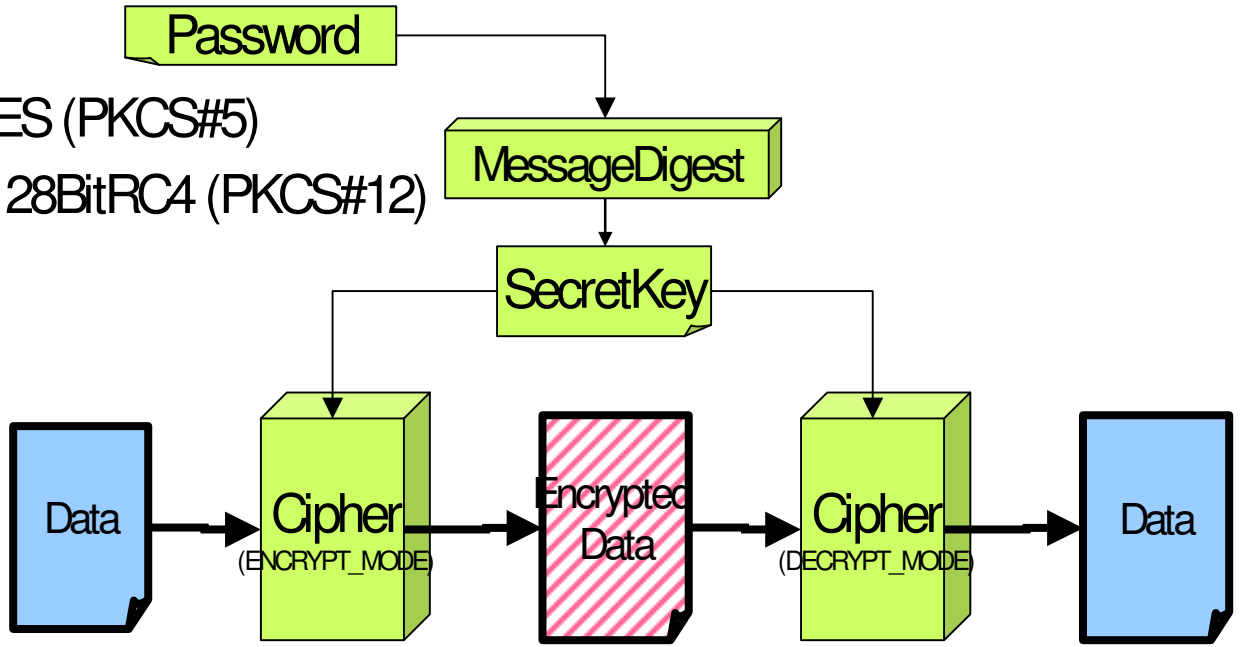



```
public void encrypt(InputStream is,
    OutputStream os,String algo, SecretKey secKey)
    throws Exception {
    Cipher c=cipher.getInstance(algo);
    c.init(Cipher.ENCRYPT_MODE, secKey);
    CipherInputStream cis
            = new CipherInputStream(is, c );
    int i; while ( (i = cis.read()) >= 0 ) {
            os.write(i);
    } }
```

```
public void decrypt(InputStream is,
    OutputStream os,String algo, SecretKey secKey)
    throws Exception {
    Cipher c=cipher.getInstance(algo);
    c.init(Cipher.DECRYPT_MODE, secKey);
    CipherInputStream cis
            = new CipherInputStream( is, c );
    int i; while ( (i = cis.read()) >= 0 ) {
            os.write(i);
    } }
```

Password Based Encryption (PBE)

- Chiffrage/Déchiffrer à partir d'un mot de passe
  - Les mots de passe ne sont pas des clés « sures »
- Idée
  - Appliquer une fonction de hachage sécurisé (MD5,SHA1, …) sur le mot de passe pour fabriquer une clé secrète
- Standard
  - PBEWithMD5AndDES (PKCS#5)
  - PBEWithSHA1And128BitRC4 (PKCS#12)



PBE Création du chiffreur/Déchiffreur

```
String mode=argv[0];
String password=argv[1];
byte[] salt = {          (byte)0xc7, (byte)0x73, (byte)0x21, (byte)0x8c,
                         (byte)0x7e, (byte)0xc8, (byte)0xee, (byte)0x99        };
int count = 20;
PBEParameterSpec paramSpec = new PBEParameterSpec( salt, count );
PBEKeySpec keySpec = new PBEKeySpec(password.toCharArray());
SecretKeyFactory kf = SecretKeyFactory.getInstance("PBEWithMD5AndDES");
SecretKey passwordKey = kf.generateSecret( keySpec );

Cipher c = Cipher.getInstance("PBEWithMD5AndDES");
if (mode.equals("-encrypt")) {
    c.init(Cipher.ENCRYPT_MODE, passwordKey, paramSpec);
    ...
} else if (mode.equals("-decrypt")) {
    c.init(Cipher.DECRYPT_MODE, passwordKey, paramSpec);
    ...
} else { ...
```

MessageDigest (Hachage Sécurisé)

- Fonction
  - produire un résumé avec une fonction de hachage sécurisé
- Obtenir une instance
  - static MessageDigest getInstance(String algorithm)
  - static MessageDigest getInstance(String algorithm, String provider)
    - algorithm=SHA0, SHA1, MD5, …
- Alimenter le calcul
  - void update(byte input)
  - void update(byte[] input)
  - void update(byte[] input, int offset, int len)
  - void reset() // remise à zéro du calcul
- Produire le résumé
  - public byte[] digest()
  - public byte[] digest(byte[] input)
- Voir aussi
  - java.security.DigestInputStream, java.security.DigestOutputStream

Hachage sécurisé des données d'un fichier

```
// Obtain a message digest object.
        MessageDigest md = MessageDigest.getInstance("MD5");

// Calculate the digest for the given file.
        FileInputStream fis = new FileInputStream(args[0]);
        BufferedInputStream bufin = new BufferedInputStream(fis);
        byte[] buffer = new byte[8192];

        int length;
        while ((length = bufin.read(buffer)) != -1)   {
              md.update(buffer, 0, length);
        };
        bufin.close();
//
        byte[] raw = md.digest();
```

MAC *Message Authentication Code*

- Fonction
  - produire un résumé avec une fonction de hachage sécurisé avec une clé
- Obtenir une instance
  - **static MAC getInstance(String algorithm)**
  - **static MAC getInstance(String algorithm, String provider)**
    - algorithm=DESMac, …
- Initialiser avec une clé
  - **void init(Key key)**
  - **void init(Key key, AlgorithmParameterSpec params)**
- Alimenter
  - **void update(byte input)**
  - **void update(byte[] input)**
  - **void update(byte[] input, int offset, int len)**
  - **void reset() // remise à zéro du calcul et de la clé (on doit réinitialiser avec une clé)**
- Générer le MAC
  - **byte[] doFinal()**
  - **void doFinal(byte[] output, int outOffset)**
  - **byte[] doFinal(byte[] input)**

Signature

- Fonction
  - Produire (vérifier) une signature à partir d'une clé privée (publique)
- Obtenir une instance
  - **static Signature getInstance(String algorithm)**
  - **static Signature getInstance(String algorithm, String provider)**
    - algorithm= SHA1withDSA, MD2withRSA, MD5withRSA, SHA1withRSA, ….
- Initialiser avec une clé privée pour la signature
  - **void initSign(PrivateKey privateKey)**
- Initialiser avec une clé publique pour la vérification
  - **void initVerify(PublicKey publicKey)**
- Alimenter pour la signature et la vérification
  - **void update(byte input)**
  - **void update(byte[] input)**
  - **void update(byte[] input, int offset, int len)**
  - **void reset()** *// remise à zéro du calcul*
- Générer la signature *(si initSign(priv))*
  - **byte[] sign()**
- Vérifier la signature *(si initVerify(pub))*
  - **boolean verify(byte[] signatureToVerify)**

Signature – Signer des données

```
//Get the private key
    PrivateKey priv = pair.getPrivate();
//Get ready to sign
    Signature dsa = Signature.getInstance("SHA1withDSA", "SUN");
//give it your private key
    dsa.initSign(priv);
//give the alg the data
    BufferedInputStream bufin = new BufferedInputStream(new FileInputStream(args[0]));
    byte[] buffer = new byte[1024];
    int len;
    while (bufin.available() != 0) {
            len = bufin.read(buffer);
            dsa.update(buffer, 0, len);
    };
    bufin.close();
// generate the signature
    byte[] realSig = dsa.sign();
```

Signature - Vérifier des données

```
//Get the signature to verify
        byte toBeVerifySig = …;
//Get the public key
        PublicKey pub = pair.getPublic();
//Get ready to sign
        Signature dsa = Signature.getInstance("SHA1withDSA", "SUN");
//give it your public key
        dsa.initVerify(pub);
//give the alg the data
        BufferedInputStream bufin = new BufferedInputStream(
                                        new FileInputStream(args[0]));
        byte[] buffer = new byte[1024]; int len;
        while (bufin.available() != 0) {
                len = bufin.read(buffer);
                dsa.update(buffer, 0, len);
        };
        bufin.close();
// verify the signature
        if(dsa.verify(toBeVerifySig)==true) { System.out.println("Signature is correct") }
        else                                { System.out.println("Signature is NOT correct") }
```

Certificat

- Motivation
  - Associer une identité (subject) avec sa clé publique et certifier le tout par une autorité de certification (CA)
  - Support de stockage et du diffusion de la clé publique
- Notions
  - PKI : Public Key Infrastructure
    - *Voir cours PKI*
  - Identity (Unix UID, X.500 Distinguished Name).
  - Entity (person, organization, program, computer, business, bank, …)
  - Issuer (celui qui a émis le certificat : ie le CA)
  - CRL (Certificate Revocation List)
- Classes
  - java.security.cert.CertificateFactory
  - java.security.cert.Certificat, java.security.cert.X509Certificat
  - java.security.cert.CRL, java.security.cert.X509CRL

CertificatFactory

- Récupérer une instance
  - static CertificateFactory getInstance(String type)
  - static CertificateFactory getInstance(String type, String provider)
    - type=X509
- Générer le ou les Certificate(s)
  - présents dans le inStream
  - Certificate generateCertificate(InputStream inStream)
  - Collection generateCertificates(InputStream inStream)
- Générer le ou les CRL
  - présents dans le inStream
  - CRL generateCRL(InputStream inStream)
  - Collection generateCRLs(InputStream inStream)

Certificate et X509Certificate

- Consulter un Certificat
  - PublicKey getPublicKey() // récupère la clé public (pour vérifier une signature)
  - byte[] getEncoded() // retourne l'encodage ASN.1 DER
- Consulter un Certificat X509
  - Principal getSubjectDN()
  - void checkValidity()
  - void checkValidity(Date date) // pour la période donnée.
  - Date getNotAfter() // date de fin la période de validité
  - Date getNotBefore() // date de fin la période de validité
  - Principal getIssuerDN() // DN de l'émétteur (CA)
  - BigInteger getSerialNumber()
  - String getSigAlgName() // algorithme de signature
  - byte[] getSigAlgParams() // paramêtre de l'algorithme
  - byte[] getSignature()
- Vérifier un Certificate
  - Lève une exception CertificateException, SignatureException, …
  - void verify(PublicKey key) throws CertificateException, SignatureException, …
  - void verify(PublicKey key, String sigProvider) throws CertificateException, …

CRL et X509CRLEntry

- Motivation
  - Liste de révocation des certificats
    - Clé corrompue, employé licencié, …
- Tester la révocation d'un Certificate dans une CRL
  - boolean isRevoked(Certificate cert)
- Parcourir les entrées d'une CRL
  - X509CRLEntry getRevokedCertificate(BigInteger serialNumber) .
  - Set getRevokedCertificates()
- Détail d'une X509CRLEntry
  - byte[] getEncoded() // retourne l'encodage ASN.1 DER.
  - Date getRevocationDate()
  - BigInteger getSerialNumber()
  - boolean hasExtensions()

Stockage des paires de clés

- Motivation
  - Stockage sécurisé de clés publiques, privées et certificats dans un fichier (keystore)
  - L'accès au fichier est protégé par une clé (**storepass**)
- Consultation/Modification
  - classe **java.security.KeyStore**
  - Outil **keytool** du JDK
- Concepts
  - Représente la structure de stockage des entrées (clés et certificats)
  - Key Entry
    - secret key, private key + référence vers le certificat.
  - Trusted Certificate Entry
    - Certificat appartenant à une entity dans laquelle le keystore fait confiance
  - Chaque entrée est désignée par un **alias**

KeyStore

- Protection
  - Le keystore est protégé par un mot de passe
  - Chaque entrée peut être protégée par un deuxième niveau de mot de passe
- Obtenir une instance
  - **static KeyStore getInstance(String type)**
  - **static KeyStore getInstance(String type, String provider)**
    - type=jks, PKCS12 (présent dans JSSE) …
  - **static String getDefaultType()**
    - Retourne la propriété Java keystore.type configurée dans le fichier java.security
  - **String getType()**
    - Retourne le type du keystore
- Charger/Stocker le KeyStore
  - **void load(InputStream stream, char[] password)**
    - **// charge les entrées protégées par un mot de passe**
  - **void store(OutputStream stream, char[] password)**
    - **// sauvegarde les entrées en protégeant avec le mot de passe**

KeyStore

- Parcourir le KeyStore
  - **int size() // nombre d'entrées**
  - **Enumeration aliases()**
  - **boolean containsAlias(String alias)**
  - **Certificate getCertificate(String alias)**
  - **Certificate[] getCertificateChain(String alias)**
  - **String getCertificateAlias(Certificate cert)**
  - **Date getCreationDate(String alias)**
  - **Key getKey(String alias, char[] password)**
  - **boolean isCertificateEntry(String alias)**
  - **boolean isKeyEntry(String alias)**
- Modifier une entrée
  - **void deleteEntry(String alias)**
  - **void setCertificateEntry(String alias, Certificate cert)**
  - **void setKeyEntry(String alias, byte[] key, Certificate[] chain)**
  - **void setKeyEntry(String alias, Key key, char[] password, Certificate[] chain)**
    - **// l'entrée est protégée par un autre mot de passe.**

KeyStore et formats

- Formats de KeyStore
  - Contient des paires de clés
  - Requiert une « passphase » pour protéger l'accès aux paires de clés (PBE)
  - JKS
    - format propriétaire de SUN
    - fournis et utilisé par défaut par les outils Java (Provider JCE)
  - PKCS#12 (.pfx, .p12)
    - format d'échange (import,export) pour les Navigateurs et les Mailers (IE, NS, …)
    - Fournis et utilisé par JSSE (Provider JCE de JSSE)
- Formats de Certificat
  - DER X509 (.cer)
  - PKCS#7 (.p7b) : contient une chaîne de certificats

Exemple avec KeyStore

```
KeyStore ks = KeyStore.getInstance("JKS");
ks.load(FileInputStream(argv[0]), "MotDePasseKeyStore".toCharArray()); printalias(ks);

CertificateFactory cf = CertificateFactory.getInstance("X.509");
InputStream is = new FileInputStream(argv[1]);
Certificate cert = cf.generateCertificate(is);
ks.setCertificateEntry("myfirst", cert); printalias(ks);

KeyPairGenerator kpg = KeyPairGenerator.getInstance("RSA");
kpg.initialize(1024, new SecureRandom());
KeyPair pair = kpg.generateKeyPair();
PrivateKey pkey = pair.getPrivate();

Certificate[] certs = { cert };
ks.setKeyEntry("mysecond", pkey,
        "MotDePasseKeyEntry".toCharArray(), certs); printalias(ks);

ks.store(FileOutputStream(argv[0]), "MotDePasseKeyStore".toCharArray());
Certificate cert = ks.getCertificate("myfirst");
```

```
private static void printalias(KeyStore store) {
  System.out.println("size = " + store.size());
  System.out.println("aliases = ");
  for( Enumeration enum = store.aliases();
                  enum.hasMoreElements();){
    System.out.print(enum.nextElement().toString() + " ");
  }
```

KeyTool

- Commande pour la gestion de fichiers keystore
  - keytool command [options]
- Options
  - -keystore keystorefile -storepass storepass –keypass keypass –alias alias
- Commandes
  - Création d'un fichier keystore et de ses entrées
    - -genkey — crée une entrée avec une paire de clé dans un certificat autosigné
    - -import — importe soit un certificat considéré de confiance ("trusted certificate") ou un certificat signé par un CA (il remplace le certificat autosigné)
    - -identitydb — importe des informations d'une base d'identité (JDK 1.1)
  - Visualisation
    - -list — Liste les entrées du keystore
    - -printcert — Affiche le contenu du certificat
  - Exportation d'une entrée
    - -export — Exporte la clé publique dans un certificat.
    - -certreq — Génère une demande CSR (Certificate Signing Request) pour envoi et signature à une autorité de certification (CA)
  - Gestion
    - -delete — supprime une entrée.
    - -keypasswd — assigne un mot de passe à une entrée.
    - -storepasswd — assigne un mot de passe à un keystore.

KeyTool Exemple 1/5
le KS des CA

```
>keytool –list –keystore $JAVA_HOME/jre/lib/security/cacerts –storepass changeit
Keystore type: jks
Keystore provider: SUN
Your keystore contains 8 entries:
verisignclass1ca, Mon Jun 29 19:06:17 CEST 1998, trustedCertEntry,
Certificate fingerprint (MD5): 51:86:E8:1F:BC:B1:C3:71:B5:18:10:DB:5F:DC:F6:20
verisignserverca, Mon Jun 29 19:07:34 CEST 1998, trustedCertEntry,
Certificate fingerprint (MD5): 74:7B:82:03:43:F0:00:9E:6B:B3:EC:47:BF:85:A5:93
thawteserverca, Fri Feb 12 21:14:33 CET 1999, trustedCertEntry,
Certificate fingerprint (MD5): C5:70:C4:A2:ED:53:78:0C:C8:10:53:81:64:CB:D0:1D
thawtepersonalbasicca, Fri Feb 12 21:11:01 CET 1999, trustedCertEntry,
Certificate fingerprint (MD5): E6:0B:D2:C9:CA:2D:88:DB:1A:71:0E:4B:78:EB:02:41
verisignclass2ca, Mon Jun 29 19:06:39 CEST 1998, trustedCertEntry,
Certificate fingerprint (MD5): EC:40:7D:2B:76:52:67:05:2C:EA:F2:3A:4F:65:F0:D8
thawtepersonalfreemailca, Fri Feb 12 21:12:16 CET 1999, trustedCertEntry,
Certificate fingerprint (MD5): 1E:74:C3:86:3C:0C:35:C5:3E:C2:7F:EF:3C:AA:3C:D9
thawtepersonalpremiumca, Fri Feb 12 21:13:21 CET 1999, trustedCertEntry,
Certificate fingerprint (MD5): 3A:B2:DE:22:9A:20:93:49:F9:ED:C8:D2:8A:E7:68:0D
thawtepremiumserverca, Fri Feb 12 21:15:26 CET 1999, trustedCertEntry,
Certificate fingerprint (MD5): 06:9F:69:79:16:66:90:02:1B:8C:8C:A2:C3:07:6F:3A
```

Liste les certificats des CA

KeyTool Exemple 2/5

```
>keytool –list
keytool error: Keystore file does not exist: /home/donsez/.keystore
```

Liste le KS par défaut

```
>keytool -genkey -dname "cn=Didier Donsez, ou=IMAG, o=UJF, l=Grenoble, c=FR"  \
      -alias didier -keypass totototo -keystore monks.jks -storepass tutututu -validity 360
```

Crée une entrée avec une paire de clé

```
>keytool -list -alias didier -keystore monks.jks -storepass tutututu
didier, Tue Nov 27 10:41:45 CET 2001, keyEntry,
Certificate fingerprint (MD5): DD:02:DE:A2:74:45:6A:07:18:31:1F:D3:3F:3F:5A:20
```

Liste l'alias dans le KS

```
>keytool -genkey -alias sylvain -keypass tatatata -keystore monks.jks -storepass tutututu
```

Crée une autre entrée avec une paire de clé

```
>keytool -list -keystore monks.jks -storepass tutututu
Keystore type: jks
Keystore provider: SUN
Your keystore contains 2 entries:
sylvain, Tue Nov 27 10:44:03 CET 2001, keyEntry,
Certificate fingerprint (MD5): 66:1A:23:B5:6A:99:D6:8A:27:6C:5D:A6:B4:4D:C8:7F
didier, Tue Nov 27 10:41:45 CET 2001, keyEntry,
Certificate fingerprint (MD5): DD:02:DE:A2:74:45:6A:07:18:31:1F:D3:3F:3F:5A:20
```

Liste les entrées du KS

KeyTool Exemple 3/5

```
>keytool -export -alias didier -file didier.cer \
                -keystore monks.jks -storepass tutututu
Certificate stored in file <didier.cer>

>keytool -printcert -file didier.cer
Owner: CN=Didier Donsez, OU=IMAG, O=UJF, L=Grenoble, C=FR
Issuer: CN=Didier Donsez, OU=IMAG, O=UJF, L=Grenoble, C=FR
Serial number: 3c035fd8
Valid from: Tue Nov 27 10:41:44 CET 2001 until: Fri Nov 22 10:41:44 CET 2002
Certificate fingerprints:
        MD5:  DD:02:DE:A2:74:45:6A:07:18:31:1F:D3:3F:3F:5A:20
        SHA1: 5E:56:55:C3:E8:35:9C:D4:41:E4:99:33:D4:6D:6A:B8:9D:0A:F6:53

>keytool -certreq -alias didier -file didier.csr \
       -keystore monks.jks -storepass tutututu \
       -keypass totototo

>keytool -list -keystore donsez.p12 -storetype PKCS12
```

Export un entrée sous la forme d'un certificat (DER X509)

Affiche ce certificat

Génère une demande de certification

Affiche le contenu du KS exporté de NS (un des providers doit supporter PKCS12)

KeyTool Exemple 4/5

```
>keytool -import -alias nadia -file nadia.p7b -keystore monks.jks -storepass tutututu
Owner: CN=Nadia Bennani, OU=ISTV, O=UVHC, L=Valenciennes, S=Nord, C=FR
Issuer: CN=Nadia Bennani, OU=ISTV, O=UVHC, L=Valenciennes, S=Nord, C=FR
Serial number: 3c036358
Valid from: Tue Nov 27 10:56:40 CET 2001 until: Sun May 26 11:56:40 CEST 2002
Certificate fingerprints:
        MD5:  AB:38:3A:2D:84:D1:A7:F2:BC:38:87:60:C7:21:14:D1
        SHA1: 86:4A:1B:28:5D:76:87:78:ED:3D:2B:45:24:65:E2:1F:AE:3F:F4:CD
Trust this certificate? [no]:  yes
Certificate was added to keystore

>keytool -list -keystore monks.jks -storepass tutututu
Keystore type: jks
Keystore provider: SUN
Your keystore contains 3 entries:
nadia, Tue Nov 27 11:01:23 CET 2001, trustedCertEntry,
Certificate fingerprint (MD5): AB:38:3A:2D:84:D1:A7:F2:BC:38:87:60:C7:21:14:D1
sylvain, Tue Nov 27 10:44:03 CET 2001, keyEntry,
Certificate fingerprint (MD5): 66:1A:23:B5:6A:99:D6:8A:27:6C:5D:A6:B4:4D:C8:7F
didier, Tue Nov 27 10:41:45 CET 2001, keyEntry,
Certificate fingerprint (MD5): DD:02:DE:A2:74:45:6A:07:18:31:1F:D3:3F:3F:5A:20

>keytool -delete -alias sylvain -keystore monks.jks -storepass tutututu
```

Importe une entrée à partir d'un certificat (PKCS#7) considéré de confiance

Si [no], le certificat n'est pas ajouté au KS

Liste les entrées du KS

Supprime une entrée

KeyTool Exemple 5/5

Export de Netscape Communicator avec:
Outils>Option Internet>Contenu>Certificats>Personnel
Export de IE avec :
Communicator>Outils>Informations sur la sécurité>Vos Certificats
Sélectionnez un certificat et exportez le

```
>keytool -printcert -file donsez.cer
Owner: EmailAddress=didier.donsez@imag.fr, CN=Didier Donsez,
            OU=Digital ID Class 1 - Netscape,
            OU=Persona Not Validated,
            OU="www.verisign.com/repository/RPA Incorp. by Ref.,LIAB.LTD(c)98",
            OU=VeriSign Trust Network,
            O="VeriSign, Inc."
Issuer: CN=VeriSign Class 1 CA Individual Subscriber-Persona Not Validated,
            OU="www.verisign.com/repository/RPA Incorp. By Ref.,LIAB.LTD(c)98",
            OU=VeriSign Trust Network,
            O="VeriSign, Inc."
Serial number: 6e867989ffb9f74b886d6f84e1abd80
Valid from: Tue Nov 27 01:00:00 CET 2001 until: Sun Jan 27 00:59:59 CET 2002
Certificate fingerprints:
      MD5:  2C:29:76:40:57:8D:76:2C:74:BB:1F:9D:9F:8C:D4:FF
      SHA1: AA:4A:50:85:20:5A:71:0F:62:5C:BA:39:3E:28:F6:DA:ED:52:46:C7
```

Affichage d'un certificat X509 DER signé par un CA et exporté de Netscape

Signature de code

- Commande `jarsigner`
  - Ajoute 2 entrées à l'archive pour chaque signataire (alias)
    - Aliasname.SF (Signature File)

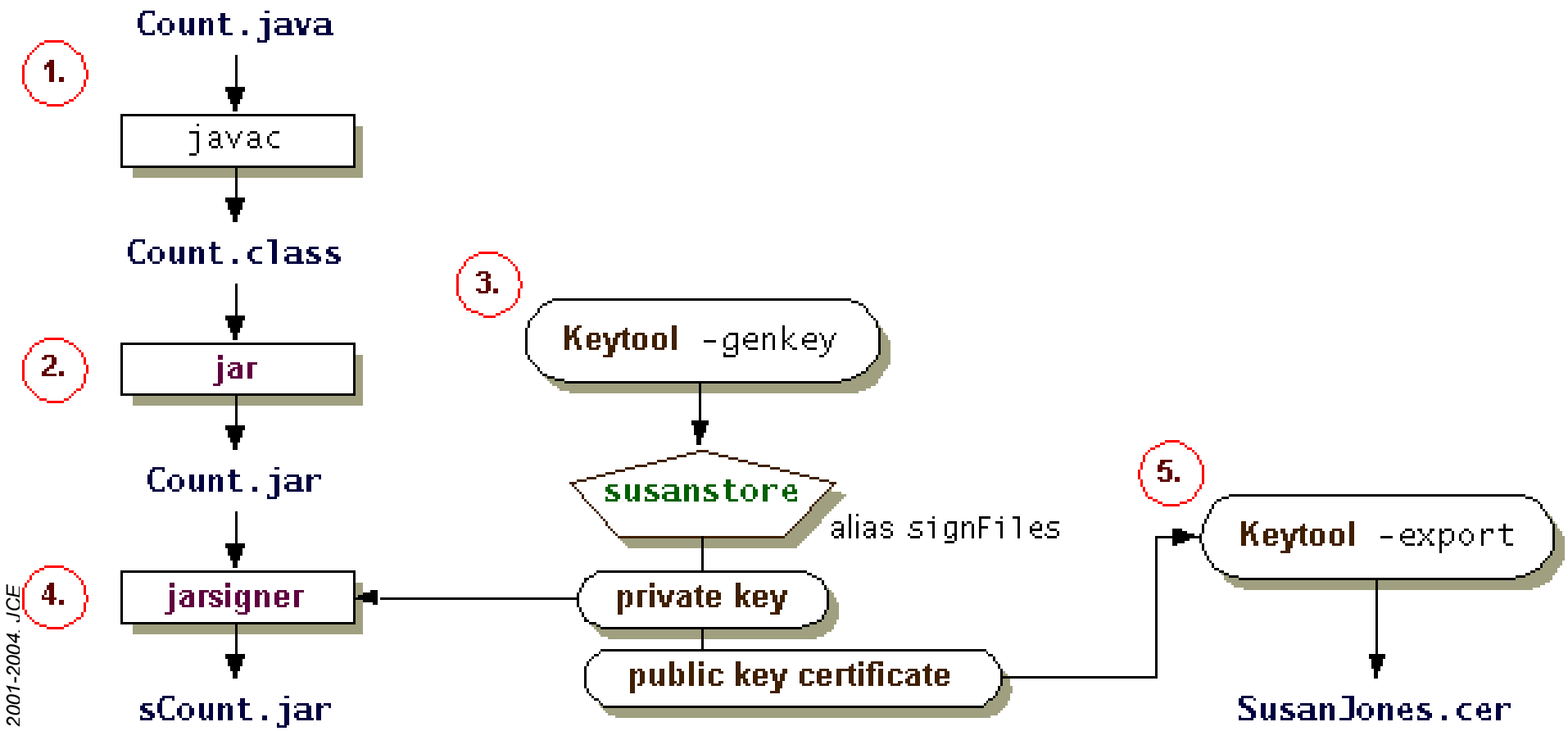
      Contient les noms des fichiers, le nom de l'algo de signature (SHA, …) et les valeurs des résumé
    - Aliasname.DSA

      Contient la signature du fichier Aliasname.SF
  - Ajoute le Manifest

    Contient les résumés des entrées
- Fonctions
  - Signature d'une archive Java
    - `jarsigner [ options ] jar-file alias`
  - Verification d'une archive Java
    - `jarsigner -verify [ options ] jar-file`
- Voir
  - *$JAVA_HOME*/docs/tooldocs/win32/jarsigner.html

Outils : Keytool et JarSigner

$JAVA_HOME/docs/tooldocs/tools.html#security



JarSigner Exemple

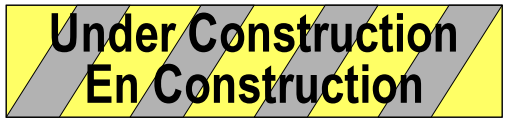
```
>jar cf archive.jar *.class *.java
>jarsigner -keystore dd.jks -signedjar sarchive.jar archive.jar  didier
>jarsigner -verify -verbose sarchive.jar

      403 Tue Nov 27 10:20:02 CET 2001 META-INF/DIDIER.SF
      926 Tue Nov 27 10:20:02 CET 2001 META-INF/DIDIER.DSA
        0 Tue Nov 27 10:19:30 CET 2001 META-INF/
sm     2180 Tue Feb 24 10:26:52 CET 1998 StrongClient.class
sm     2230 Tue Feb 24 10:28:06 CET 1998 StrongServer.class
sm     1532 Tue Feb 24 10:26:32 CET 1998 StrongClient.java
sm     1566 Tue Feb 24 10:27:24 CET 1998 StrongServer.java
  s = signature was verified
  m = entry is listed in manifest
  k = at least one certificate was found in keystore
  i = at least one certificate was found in identity scope

jar verified.
```

Echange sécurisé de clé (KAP Key Agreement Protocol)

- Motivation
  - Échange une clé secrète entre deux parties
  - Application : clé de session symétrique
- Protocoles
  - Diffie-Hellman (PKCS#3), …
- JCE : classe javax.crypto.KeyAgreement
  - static KeyAgreementgetInstance(String algorithm)
  - void init(java.security.Key key)
  - Key doPhase(Key key, boolean lastPhase)
  - DH est fournit par JCE de Baltimore



Echange sécurisé de clé Diffie-Hellman

1. Initialisation

1. choisir un couple approprié de nombres premiers **p et a**, tels que $2<\mathbf{a}<\mathbf{p}$
2. a et p sont publiées publiquement

2. Protocole

1. Message
   - ***A*** sends ***B***: $\mathbf{a}^x \% \mathbf{p}$, for any $1 <= x <= \mathbf{p}-2$
   - ***B*** sends ***A***: $\mathbf{a}^y \% \mathbf{p}$, for any $1 <= y <= \mathbf{p}-2$
2. **Actions:**
   - **B** receives $\mathbf{a}^x$ and calculates $K=(\mathbf{a}^x)^y$
   - **B** receives $\mathbf{a}^y$ and calculates $K=(\mathbf{a}^y)^x$

Providers

- La liste
  - http://java.sun.com/products/jce/jce12_providers.html
- Quelques providers
  - Baltimore KeyTools http://www.baltimore.com
  - RSA Security http://www.rsasecurity.com
  - eSec (ex-ABA) http://www.openjce.org/jce.html
    - http://www.wumpus.com.au/crypto/aba.html
    - http://www.bouncycastle.org/
  - Cryptix http://www.cryptix.org/products/jce/
  - Phaos Technology http://www.phaos.com/
  - Entrust(R) Technologies http://www.entrust.com/toolkit/java/index.htm
  - DSTC http://security.dstc.edu.au/projects/java/release3.html
  - IAIK http://jcewww.iaik.at/

ABA JCE Provider

- JDK 1.02, 1.1, 1.2, 1.3
- Cipher
  - RSA, DES, DESede, IDEA, Twofish, Blowfish, and RC4.
- Password Based Encryption (PBE)
  - MD5 with DES, SHA1 with RC4 128 bits
- Message digests
  - SHA-0, SHA-1, and MD5.
- MAC
  - DES, DESede, IDEA, HMACwithSHA1, and HMACwithMD5.
- Keystore.
- Key and Key pair generation
- Signing
  - MD5withRSA.

Cryptix JCE Provider

- URL http://www.cryptix.org
- JDKs 1.1, 1.2 and 1.3
- Ciphers
  - Blowfish, CAST5, DES, IDEA, MARS, RC2, RC4, RC6, **Rijndael**, Serpent, SKIPJACK, Square, TripleDES, Twofish
- Password Based Encryption (PBE)
  - none
- KeyAgreements
  - Diffie-Hellman
- Modes
  - CBC, CFB-(8, 16, 24, ..., blocksize), ECB, OFB-(blocksize), openpgpCFB
- Hashes
  - MD2, MD4, MD5, RIPEMD-128, RIPEMD-160, SHA-0, SHA-1, SHA-256/384/512, Tiger
- MACs
  - HMAC-MD2, HMAC-MD4, HMAC-MD5, HMAC-RIPEMD-128, HMAC-RIPEMD-160, HMAC-SHA-0, HMAC-SHA-1, HMAC-Tiger
- Signatures
  - RawDSA, RSASSA-PKCS1, RSASSA-PSS
- Assymetric ciphers
  - RSA/PKCS#1, ElGamal/PKCS#1
- SecureRandom SPIs
  - /dev/urandom on systems that support it (FreeBSD, Linux, OpenBSD and possibly other UNIXen)

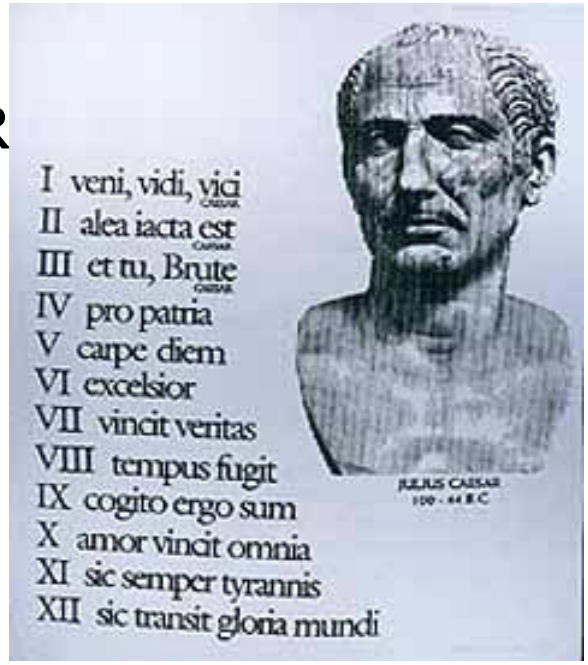
Implémentation d'un Provider JCE

- Implémenter
  - Une classe final Provider et des classes SPI
    - CipherSpi, MessageDigestSpi, …
- Voir
  - %JAVA-HOME%\docs\guide\security\HowToImplAProvider.html
- Exercice
  - Le faire pour l'algorithme de CESAR !

Implémentation d'un Provider JCE

Exercice

- Un Provider pour l'algorithme de CESAR
- La classe Provider à installer




```
package monjce.jce.provider;
public final class MonJCEProvider extends java.security.Provider {
   private static final String NAME    = "MonJCE",
        INFO    = "MonJCE JCE Weak Crypto Provider";
   private static final double VERSION = 1.0;

   public MonJCEProvider() {
      super(NAME, VERSION, INFO);
      // CESAR
      put("Cipher.CESAR","monjce.jce.provider.cipher.CESAR");
      put("Alg.Alias.Cypher.CAESAR","CESAR");

      put("KeyGenerator.CESAR","monjce.jce.provider.key.CESARKeyGenerator");
      put("SecretKeyFactory.DES","monjce.jce.provider.keyfactory.CESARKeyFactory");
   }
}
```

Extra JCE

PKCS *« Public-Key Cryptography Standards »*

- ensemble de standards pour la mise en place des IGC, coordonné par RSA
- définissent les formats des éléments de cryptographie :
  - PKCS #1:RSA Cryptography Standard
    - Inclut PKCS #2 et PKCS #4
  - PKCS #3:Diffie-Hellman Key Agreement Standard
  - PKCS #5:Password-Based Cryptography Standard
  - PKCS #6:Extended-Certificate Syntax
  - PKCS #7:Cryptographic Message Syntax
  - PKCS #8:Private-Key Information Syntax
  - PKCS #9:Selected Attribute Types Standard
  - PKCS #10:Certification Request Syntax Standard
  - PKCS #11:Cryptographic Token Interface Standard
  - PKCS #12:Personal Information Exchange Syntax Standard
  - PKCS #13: Elliptic Curve Cryptography Standard
  - PKCS #15: Cryptographic Token Information Format Standard
- Voir http://www.rsa.com/rsalabs/pkcs/

Cerificats X509, PKI et CA

- PKI : Private Key Infrastructure
  - http://www.verisign.com/resources/wp/index.html
  - http://www.thawte.com/
  - https://www.entrust.com/developer/java/faqs.htm
- CA
  - Public
    - Verign, Thawte, Entrust, …
  - Privé
    - Motivation : une société est sa propre autorité de certification pour ses employés et ses applications dans les échanges intranet.
    - iPlanet Certificate Management Server (developer.iplanet.com) , OpenCA (www.openca.org), IDX-PKI (idx-pki.idealx.org)

JSSE et JAAS

- Java Secure Socket Extension (JSSE)
  - Extension aux sockets pour sécuriser les protocoles basés sur TCP/IP (HTTP, Telnet, NNTP, FTP, RMI …)
    - SSL (Secure Sockets Layer) et TLS (Transport Layer Security)

      SecureSocket, SecureServerSocket, HttpsURLConnection, …
  - Utilise le JCE et ajoute un provider
  - Intégré au J2SE1.4
    - Remarque : Avant il fallait bricoler avec les SocketFactory
- Java Authentication and Authorization Service (JAAS)
  - Les PAM peuvent être utilisés des moyens cryptographiques

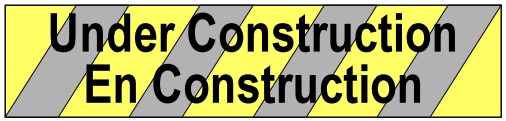
XML et Sécurité

- Evolution vers le gestion des clés en XML et vers les documents XML signés et chiffrés
  - Spécifications
    - XKMS XML Key Management Specification
      http://www.xmltrustcenter.org/xkms
    - XML Digital Signature http://www.w3.org/TR/xmldsig-core/
    - XML Digital Encryption http://www.w3.org/TR/xmlenc-core/
  - API Java
    - JSR 104 XML Trust Service APIs
    - JSR 105 XML Digital Signature APIs (javax.security.xml.dsig)
    - JSR 106 XML Digital Encryption APIs

Cryptographie, JavaCard, Carte Crypto

- Cartes à puce pourvue de fonctions cryptographiques
  - (logicielles ou matérielles)
  - Cartes PKI (GemSafe, CryptoSafe, …)
- JavaCard
  - Packages `javacard.security.* javacardx.crypto.*` pour les applets utilisant de la cryptographie
- OCF
  - CardService

JCE et J2ME



Motivation

- Faible mémoire, KVM, Processeur cryptographique

Pistes : les API JavaCard

- Packages `javacard.crypto.*` `javacardx.crypto.*`

Bibliographie

- Livre
  - Jonathan B. Knudsen , Java Cryptography, Oreilly, 1st Edition, May 1998, 1-56592-402-9
  - Scott Oaks, Java Security, Oreilly, 2nd Edition, May 2001, 0-596-00157-6
  - Jamie Jaworski & Paul J. Perrone, Java Security , Ed CampusPress, 24/9/2001, 570 pages, ISBN 2-7440-1230-0, (en Français) http://www.campuspress.net
- Guide
  - %JAVA_HOME%\docs\guide\security\CryptoSpec.html
  - %JAVA_HOME%\docs\guide\security\HowToImplAProvider.html
- Tutorial
  - http://java.sun.com/docs/books/tutorial/security1.2/index.html
- Présentation JavaOne
  - http://servlet.java.sun.com/javaone/javaone99/pdfs/e611.pdf
  - http://java.sun.com/products/jsse/JavaOne00_JSSE.pdf
  - http://jsp.java.sun.com/javaone/javaone2000/pdfs/TS-1175.pdf
- Articles
  - http://www.onjava.com/onjava/security/