UNIVERSITÀ DI CATANIA

DIPARTIMENTO DI MATEMATICA E INFORMATICA
CORSO DI LAUREA IN INFORMATICA, SECONDO LIVELLO

Progetto di Sicurezza

Implementazione di una demo che permetta di eseguire crittografia con diversi algoritmi

Oreste Delitala W82000025 Anno 2013/2014

JCA e JCE

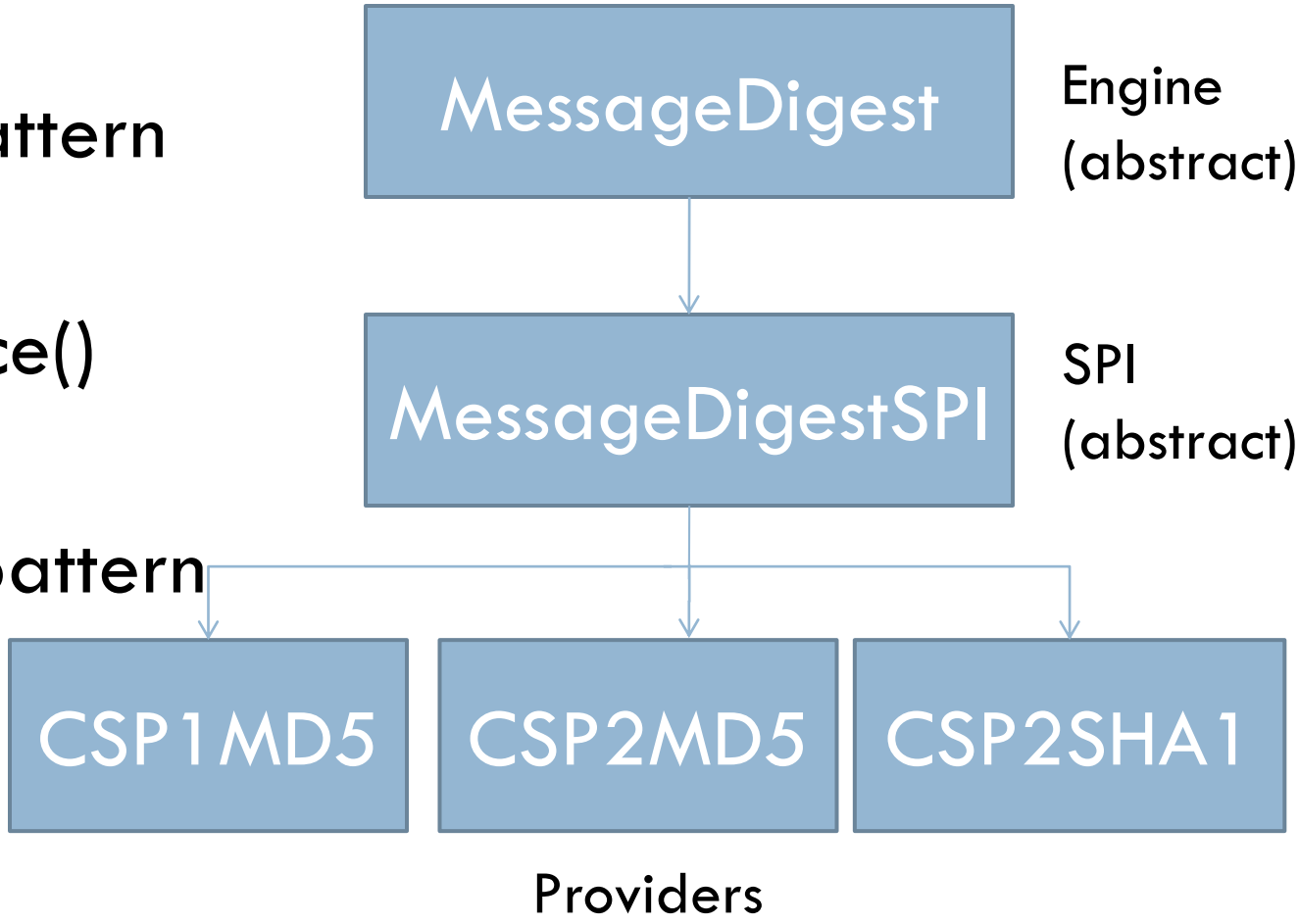
- Linguaggio Java
  - Java Cryptography Architecture (JCA)
  - Java Cryptography Extension (JCE)

Java Cryptography Architecture (JCA)

- MessageDigest
- Signature
- KeyPairGeneration
- KeyFactory
- CertificateFactory
- KeyStore
- AlgorithmParameters
- AlgorithmParameterGenerator
- SecureRandom

Architettura JCA

- Factory pattern
  - getIstance()
- Strategy pattern



Provider

- Un provider è un insieme di implementazioni di vari algoritmi
  - SUN
    - MD5, SHA-1
    - DSA
    - Creazioni di certificati X.509
    - Generazione di numeri random proprietaria
    - Keystore proprietario

Java Crtptography Extension JCE

- ha la stessa architettura di JCA
- È costituita da javax.crypto e sotto-package
- Classi principali
  - Cipher
  - KeyAgreement
  - Mac
  - SecretKey
  - SecretKeyFactor

Principali Provider per JCE

- Bouncy Castke
  - Open Source
  - Provider più completo
- SunJCE
  - Open Source
  - Non compatibile con altri Provider
  - Non supporta RSA

Cifratura Simmetrica

- Principali classi ed interfacce
  - Javax.crypto.Cipher
    - getInstance, init, update, doFinal
  - Java.security.Key(interfaccia)
    - Creato da:
      - Javax.crypto.KeyGenerator
      - Java.security.KeyFactory
  - Javax.crypto.KeyGenerator
    - getInstance, init, generateKey

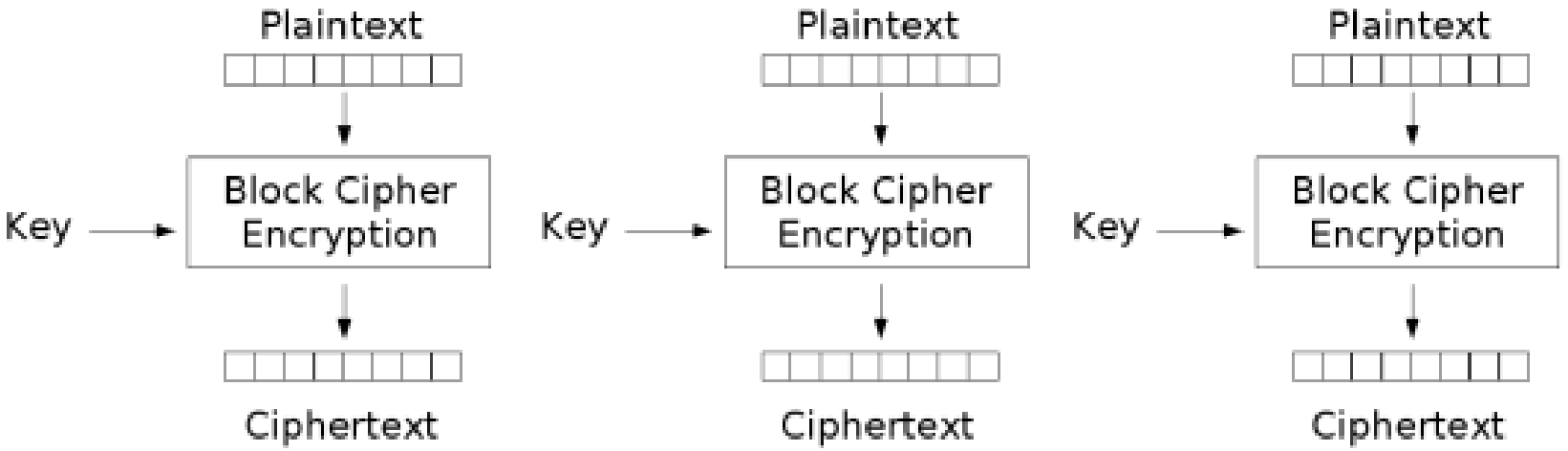
La classe Cipher

- getInstance("algorithm/mode/padding", "provider")
  - Algorithm
    - DES
    - DESede
    - AES
  - Mode
    - CBC
    - ECB
  - Padding
    - PKCS5Padding
    - NoPadding

La classe Cipher

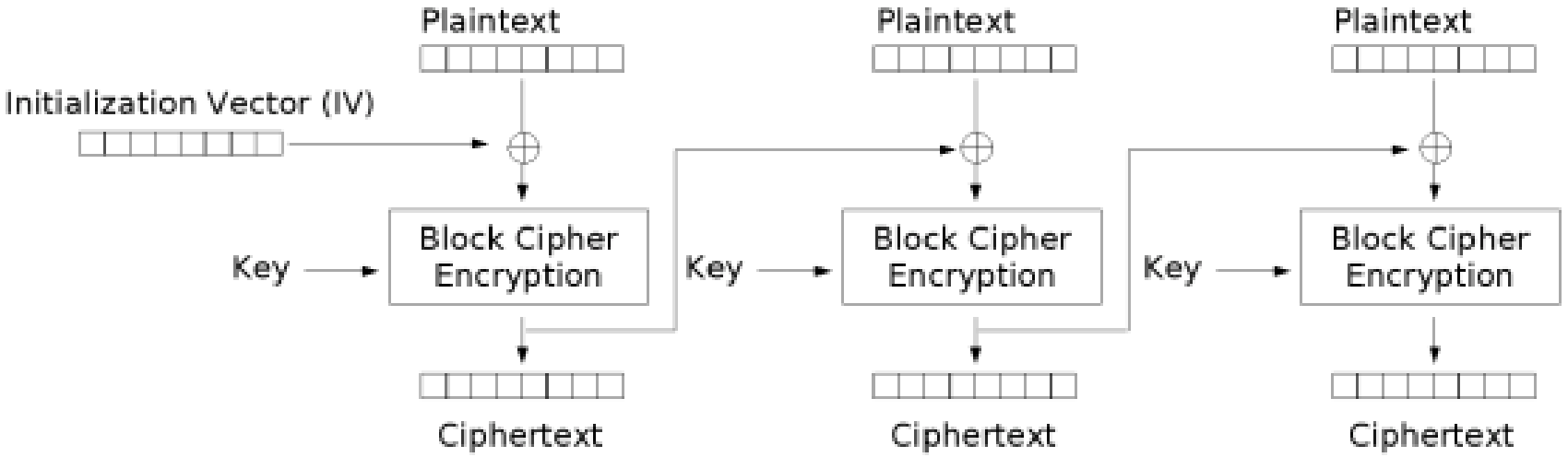
- init(mode, key)
  - Cipher.ENCRYPT_MODE
  - Cipher.DECRYPT_MODE
- update(bytes)
- doFinal(bytes)

Modalità ECB



Electronic Codebook (ECB) mode encryption

Modalità CBC



Cipher Block Chaining (CBC) mode encryption

I passi della cifratura ECB

- Creazione di una chiave
  1. KeyGenerator keyGenerator = KeyGenerator.getIstance("DES");
  2. keyGenerator.init(64);
  3. SecretKey key = keyGenerator.generateKey();

- Creazione ed inizializzazione di un cifrario
  1. Chiper cipher = Cipher.getInstance("DES/ECB/PKCS5Padding");

I passi della cifratura ECB

- Cifratura
  1. cipher.init(Cipher.ENCRYPT_MODE, key);
  2. byte[] cipherText = cipher.doFinal(stringToEncrypt.getBytes());

- Decifratura
  1. cipher.init(Cipher.DECRYPT_MODE, key);
  2. byte[] plainText = cipher.doFinal(cipherText);

I passi della cifratura CBC

- Creazione di una chiave
  1. KeyGenerator keyGenerator = KeyGenerator.getInstance("DES");
  2. SecretKey key = keyGenerator.generateKey();
- Creazione ed inizializzazione di un cifrario
  1. byte[] randomBytes = new byte[8]; //iv size = block size
  2. SecureRandom random = new SecureRandom();
  3. random.nextBytes(randomBytes);
  4. IvParameterSpec ivparams = new IvParameterSpec(randomBytes);
  5. Cipher cipher = Cipher.getInstance("DES/CBC/PKCS5Padding");
  6. Cipher.init(Cipher.ENCRYPT_MODE, key, ivparams); // cifratura
  7. Cipher.init(Cipher.DECRYPT_MODE, key, ivparams); // decifratura

Generare una chiave da una Stringa

1. String password = “password”;
2. byte[] desKeyData = password.getBytes();
3. DESKeySpec desKeySpec = new DESKeySpec(desKeyData);
4. SecretKeyFactory keyFactory = SecretKeyFactory.getInstance("DES");
5. SecretKey key = keyFactory.generateSecret(desKeySpec);

Sicurezza Cifratura Simmetrica

- ECB
  - Vulnerabile alla critto analisi
- CBC
  - Proteggere l'IV
- DES
  - Chiave a 64 bit vulnerabile all'attacco del compleanno
- DESede
  - Chiave sicura a 192 bit
  - Blocchi da 64 bit troppo piccoli
- AES
  - Sicuro perché usa chiave e blocco di lunghezza 128 bit

Cifratura Asimmetrica

- Generalmente usata per cifrare dati di "piccole" dimensioni (in bit)

- Per "grandi" dimensioni si utilizza la cifratura asimmetrica a chiave di sessione
  - **(e, d)** un public-private key pair
  - **P** un "grande" payload
  - **K** ⟵ *newSimmetricKey()*
  - **M = [E_k(P), E_e(K)]**

Cifratura Asimmetrica

- Principali classi ed interfacce
  - java.security.KeyPair
    - getPublic(), getPrivate()
  - java.security.PublicKey(interfaccia)
  - javax.crypto.PrivateKey(interfaccia)
  - Java.security.KeyPairGenerator
    - genKeyPair()

La classe Cipher

- getInstance("algorithm/mode/padding", provider)
  - Algorithm
    - RSA
  - Mode
    - ECB
  - Padding
    - NoPadding
    - PKCS1Padding
    - OAEPWithSHA-1AndMGF1Padding
    - OAEPWithSHA-256AndMGF1Padding

La classe Cipher

- init(mode, key)
  - Cipher.ENCRYPT_MODE
    - publicKey
  - Cipher.DECRYPT_MODE
    - privateKey
- update(bytes)
- doFinal(bytes)

OAEP

$\mathbf{Enc}(N, m)$ // $m \in \{0,1\}^n$

$r \leftarrow_R \{0,1\}^{k_0}$;

$s \leftarrow G(r) \oplus (m||0^{k_1})$ // $s \in \{0,1\}^{n+k_1}$

$t \leftarrow H(s) \oplus r$; // $t \in \{0,1\}^{k_0}$;

$w \leftarrow s||t$; // $w \in \{0,1\}^k$

$y \leftarrow \mathtt{RSA}(w)$; // $y \in \{0,1\}^k$;

Return y

I passi della cifratura

- Creazione di una chiave
  1. KeyPairGenerator generator = KeyPairGenerator.getIstance("RSA");
  2. keyGenerator.init(1024);
  3. KeyPair pair = generator.generateKeyPair();
  4. Key pubKey = pair.getPublic();
  5. Key privKey = pair.getPrivate();

- Creazione ed inizializzazione di un cifrario
  1. Chiper cipher = Cipher.getInstance("RSA/ECB/PKCS1Padding");

I passi della cifratura

- Cifratura
  1. cipher.init(Cipher.ENCRYPT_MODE, pubKey);
  2. byte[] cipherText = cipher.doFinal(stringToEncrypt.getBytes());

- Decifratura
  1. cipher.init(Cipher.DECRYPT_MODE, privKey);
  2. byte[] plainText = cipher.doFinal(cipherText);

Codifica e decodifica di chiavi (RSA)

- Codifica:
  - Chiave pubblica: X.509
  - Chiave privata: PKCS#8

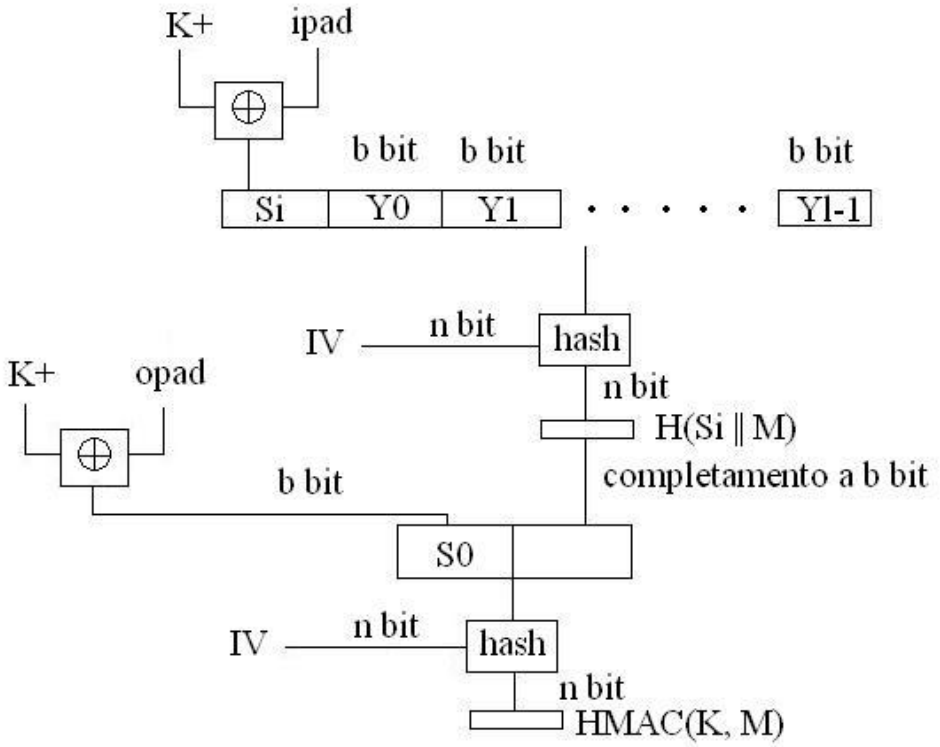
Codifica e decodifica di chiavi (RSA)

- Decodifica
  - Public key
    1. KeyFactory keyFactory = KeyFactory.getInstance("RSA");
    2. X509EncodedKeySpec publicKeySpec = new X509EncodedKeySpec(keyBytes);
    3. PublicKey publicKey = keyFactory.generatePublic(publicKeySpec);
  - Private key
    1. KeyFactory keyFactory = KeyFactory.getInstance("RSA");
    2. PKCS8EncodedKeySpec privateKeySpec = new PKCS8EncodedKeySpec(keyBytes);
    3. PrivateKey privateKey = keyFactory.generatePrivate(privateKeySpec);

Sicurezza Crittografia Asimmetrica

- Problema della Fattorizzazione

HMAC


K+
ipad
b bit
b bit
b bit
Si
Y0
Y1
Yl-1
IV
n bit
hash
K+
opad
n bit
H(Si || M)
completamento a b bit
b bit
S0
IV
n bit
hash
n bit
HMAC(K, M)

HMAC

- Classe
  - Javax.crypto.Mac
    - getInstance(“algorithm”, “provider”)
      - Algorithm
        - HmacSHA1
        - HmacSHA256
    - init(key)
    - update(bytes)
    - doFinal()

I passi di cifratura

- Generazione di una chiave
  - KeyGenerator keygen = KeyGenerator.*getInstance("HmacSHA1");*
  - Key macKey = keygen.generateKey();
- Creazione di un oggetto MAC
  - Mac mac = Mac.getInstance(“HmacSHA1”);
- Inizializzazione del MAC
  - mac.init(macKey);
- Restituzione del MAC
  - mac.update(text.getByte())
  - byte[] result = mac.doFinal();

I passi di cifratura

- Verifica del MAC
  - si prende in input ***(message, mac)***
  - si calcola il mac di ***message*** con la chiave segreta
  - si confrontano i 2 mac se sono uguali

Sicurezza HMAC

- Attacchi Replay
  - Soluzione Cifrare il Digest
    - Simmetrica
      - CBC
    - Asimmetrica
    - Valore segreto SAB condiviso
      - HMAC(SAB||M)
- Non si ha sicurezza su chi ha eseguito l'HMAC tra mittente e ricevente

Firma Digitale

- Classe principale
  - java.security.Signature
    - getInstance(“algorithm”,”provider”)
      - Algorithm
        - SHA1withDSA
    - initSign(privKey), initVerify(pubKey)
    - update(bytes)
    - sign()
    - verify(sign)

I passi di cifratura

- Generazione di una coppia di chiavi
  1. KeyPairGenerator keyGen = KeyPairGenerator.getInstance("DSA", "SUN");
  2. SecureRandom random = SecureRandom.getInstance("SHA1PRNG", "SUN");
  3. keyGen.initialize(1024, random);
  4. KeyPair keyPair = keyGen.generateKeyPair();
- Generazione di un Signature engine
  1. Signature sig = Signature.getInstance("SHA1withDSA", "SUN");

I passi di cifratura

- Generazione della firma digitale
  1. sig.initSign(privKey);
  2. sig.update(data);
  3. byte[] signBytes = sig.sign();
- Verifica della firma digitale
  1. sig.initVerify(pubKey);
  2. sig.update(data);
  3. try{
  4. verified = sig.verify(signBytes);
  5. } catch(SignatureException e){
  6. verified = false;
  7. }

Sicurezza Firma Digitale

- Non repudiabilità