Criptografia utilizando Java Introdução

- A plataforma Java inclui um grande conjunto de APIs, ferramentas e implementações de algoritmos relacionados com a segurança
- API abrange áreas como:
  - Criptografia
  - Infra-estrutura de Chave pública (PKI – Public Key Infraestructure)
  - Segurança em comunicação
  - Autenticação
  - Controle de Acesso
- Para maiores informações:
  - http://java.sun.com/javase/6/docs/technotes/guides/security/index.html

Java Cryptography Architecture (JCA)

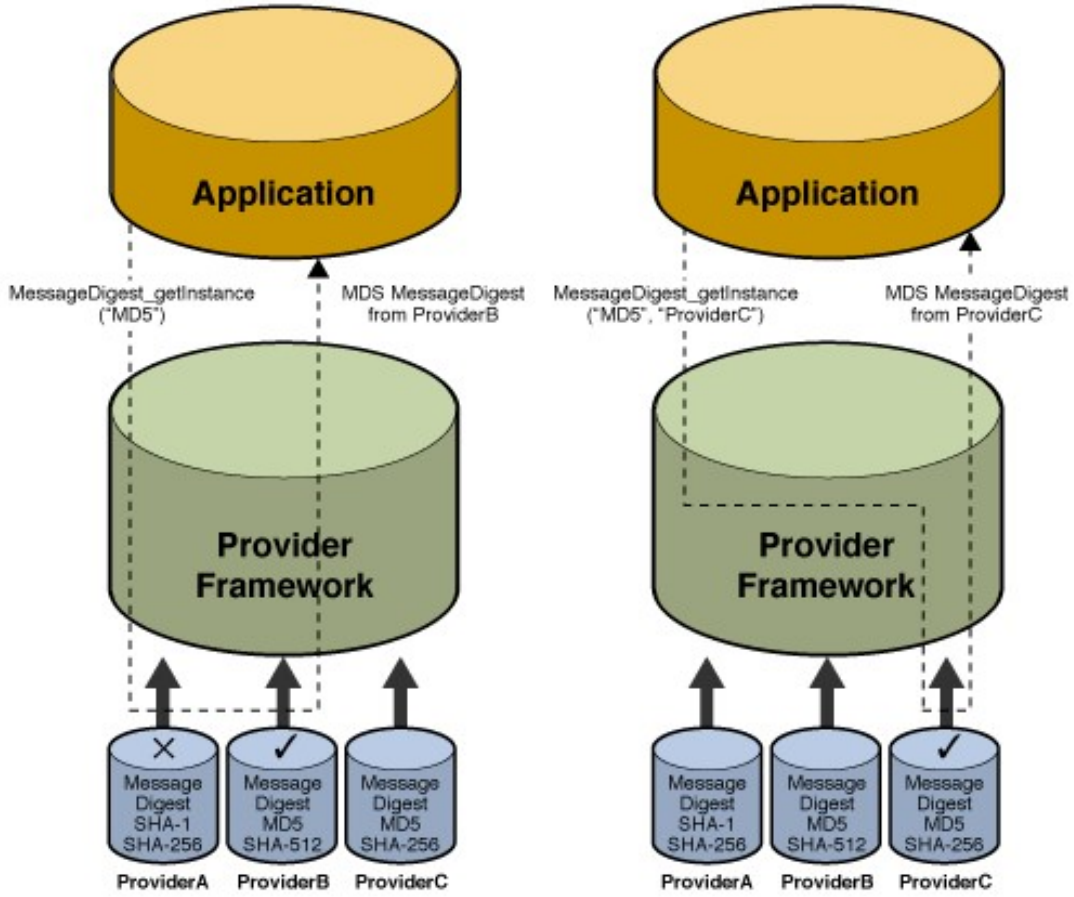
- É a maior parte da plataforma de segurança Java
- Contém a arquitetura do “Provider” e um APIs para os principais algoritmos
- Princípios de projeto
  - Independência de Implementação
    - Aplicações não necessitam implementar algoritmos, apenas requisitar serviços
    - Serviços são implementados por provedores
    - Classes da JCA provedoras de serviços são chamadas “engines” e que utilizam extensivamente o padrão de projeto factory - MessageDigest, Signature, KeyFactory, KeyPairGenerator, and Cipher
  - Interoperabilidade
    - Aplicações não estão ligadas a um provedor específico e vice-versa
    - Provedores são utilizados por várias aplicações
  - Extensiblidade
    - Plataforma pode ser estendida através da instalação de provedores específicos

Java Cryptography Architecture Conceito Provedor

- Base da arquitetura são os provedores de serviço (Service Providers)
- Contém a implementação dos algoritmos
- Podem ser estendidos a partir da classe **java.security.Provider**
- Exemplo:
  - md = MessageDigest.getInstance("MD5");
  - md = MessageDigest.getInstance("MD5", "ProviderC");
- Observação
  - Na segunda linha de código o principio da interoperabilidade não é satisfeito pois o código está associado a um provedor específico

Java Cryptography Architecture Conceito Provedor - Exemplo

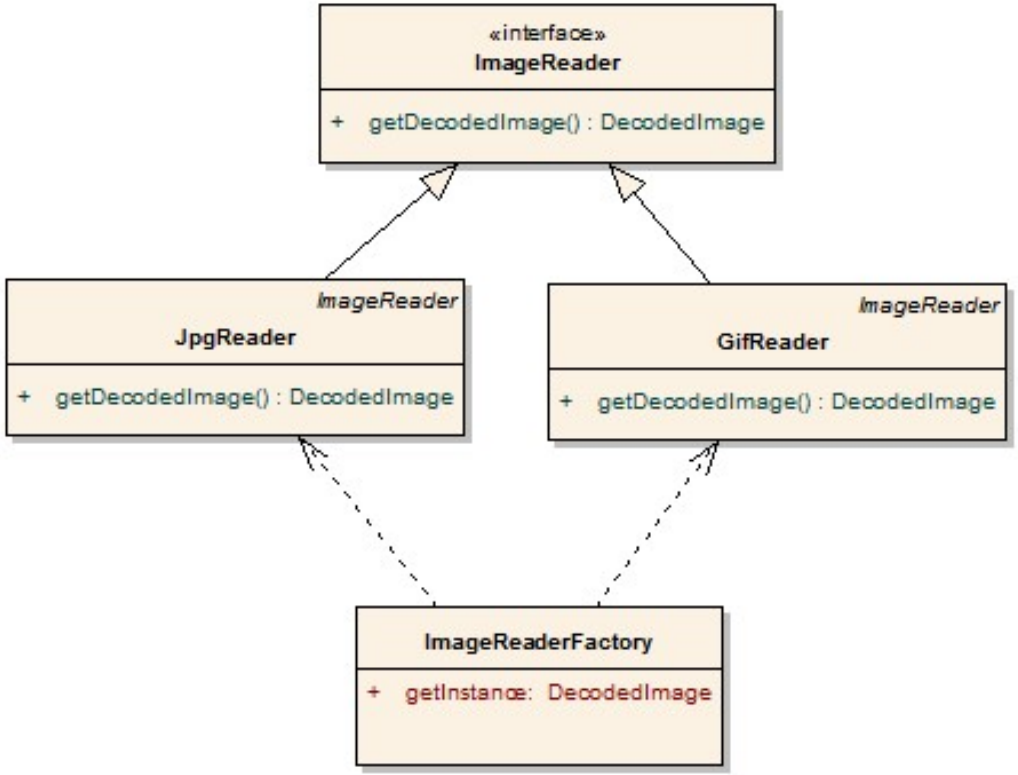
- Exemplo:
  - md = MessageDigest.getInstance("MD5");
  - md = MessageDigest.getInstance("MD5", "ProviderC");



Java Cryptography Architecture(JCA) Padrão de Projeto Factory

- O padrão de projeto (design pattern) factory é amplamente utilizado na JCA
- Exemplo

```
public interface ImageReader {
     public DecodedImage getDecodedImage();
}
public class GifReader implements ImageReader  {
      public DecodedImage getDecodedImage() {
          return decodedImage;
      }
}
public class JpgReader implements ImageReader {
     //....
}
public class ImageReaderFactory {
    public static ImageReader getInstance( InputStream  is)
        int imageType = figureOutImageType( is );
        switch( imageType ) {
            case ImageReaderFactory.GIF:
                return new GifReader( is );
            case ImageReaderFactory.JPEG:
                return new JpegReader( is );
            // etc.
        }
    }
}
```



JCA – Conceitos Engines

- Provê uma interface para um tipo específico de serviço
- Independe tanto do algoritmo quanto do provedor
- Serviços fornecidos:
  - Criptografia
    - Simétrica, Assimétrica, Hash
  - Geradores e Conversores
    - Produção de chaves e parâmetros utilizados em algoritmos criptográficos
  - Repositórios de Objetos
    - Certificados e Chaves

JCA – Conceitos
Engines - Criptografia

- Cipher
  - Utilizado para cifrar e decifrar dados
  - Necessário ser inicializado com chaves
  - Algoritmos simétricos (bloco e stream), assimétricos e cifragem baseada em senha (Password Based Encryption – PBE)
- MessageDigest
  - Calcula um sumário (digest) ou hash de uma seqüência de dados
- Signature
  - Utilizado para assinar e verificar assinaturas digitais
  - Necessário ser inicializado com chaves
- Message Authentication Codes (Mac)
  - Semelhante a classe MessageDigest
  - Necessita ser inicializado com chaves a fim de proteger a integridade dos dados
- Secure Random
  - Utilizado na geração de números aleatórios e pseudo-aleatórios

JCA – Conceitos
Engines – Geradores e Conversores

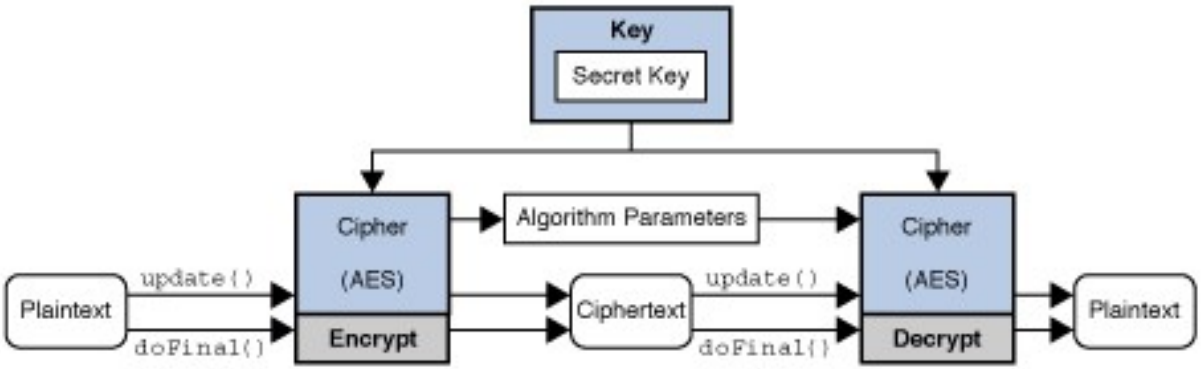
- KeyFactory
  - Converte chaves criptográficas (objetos da classe Key) em especificações de chaves e vice-versa
  - Especificações de chaves representam a chave e o material utilizado na criação da mesma
- SecretKeyFactory
  - Converte chaves criptográficas simétricas (objetos da classe SecretKey) em especificações de chaves e vice-versa
  - São especializações da classe KeyFactory para uso em algoritmos simétricos
- KeyPairGenerator
  - Utilizado para gerar um novo par de chaves (pública e privada) para uso de um algoritmo específico
- KeyGenerator
  - Utilizado na geração e chaves secretas para uso de algum algoritmo específico
- KeyAgreement
  - Utilizado por duas ou mais partes para acordar sobre o uso de uma chave específica em operações criptográficas
- AlgorithmParameters
  - Utilizado para armazenar parâmetros de um algoritmo especifico, incluindo parâmetros de encoding e decoding.
- AlgorithmParameterGenerator
  - Utilizado na geração de objetos AlgorithmParameters para um determinado algoritimo

JCA – Conceitos
Engines – Repositórios de Objetos

- KeyStore
  - Representa um banco de dados de chaves e certificados de entidades acreditadas (trusted)
  - Chaves privadas armazenadas contém uma cadeia de certificados associados a fim de autenticar as chaves públicas correspondentes
  - Pode conter um certificado
- CertificateFactory
  - Utilizado na criação de certificados de chaves públicas e CRLs (Certification Revocation Lists)
- CertPathBuilder
  - Utilizado na criação de cadeias de certificados, também conhecidas como “Certification Paths”
- CertPathValidator
  - Utilizado na validação de cadeias de certificados
- CertStore
  - Utilizado na recuperação de certificados e CRLs de um repositório

JCA
Cifradores

- A classe Cipher provê a funcionalidade de cifragem e decifragem



- Normalmente o processo envolve
  - Geração da Chave
  - Uso do Algoritmo

JCA
Cifradores

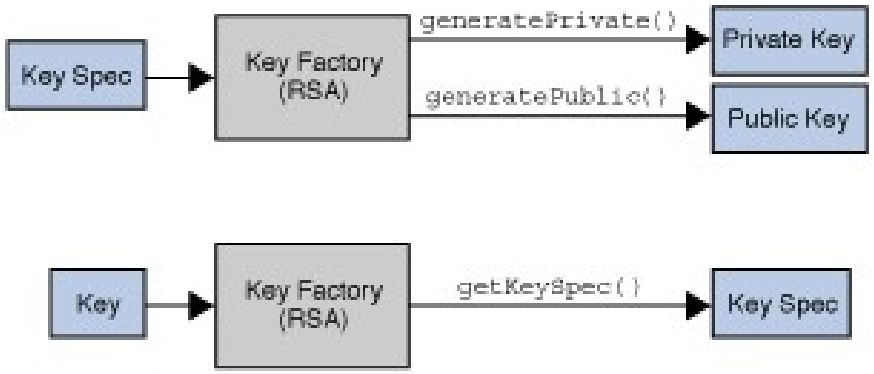
- O processo de uso dos cifradores envolve:
  - Geração da Chave
    - Simétrica
      - SecretKey e KeyGenerator
      - Password Based Encryption (PBE)
        - PBEKey
    - Assimétrica
      - KeyPairGenerator ou KeyGenerator
  - Uso do Algoritmo
    - Obtenção do Objeto Cipher (**javax.crypto.Cipher**)
      - Método getInstance()
    - Inicialização do Algoritmo
      - Método init() – recebe informações de acordo com o algoritmo.
    - Cifgragem e Decifragem
      - Método doFinal()
      - Método update() – permite cifragem ou decifragem de múltiplas partes

JCA – Conceitos Representações de Chaves

- Uma chave possui duas representações
  - Opaca
    - Implementam interface **java.security.Key**
    - Não permite o acesso direto ao material que constitui a chave
    - Acesso limitado, sendo possível obter somente:
      - Algoritmo (getAlgorithm)
      - Formato (getFormat)
      - Codificação (getEncoded)
    - Chaves utilizadas pelos cifradores
  - Transparente
    - Implementam interface **java.security.spec.KeySpec**
    - Permite o acesso ao material utilizado na criação da chave
    - Utilizadas para transmissão ou armazenamento das chaves
    - Acesso é feito através das classes que representam a especificação de cada chave

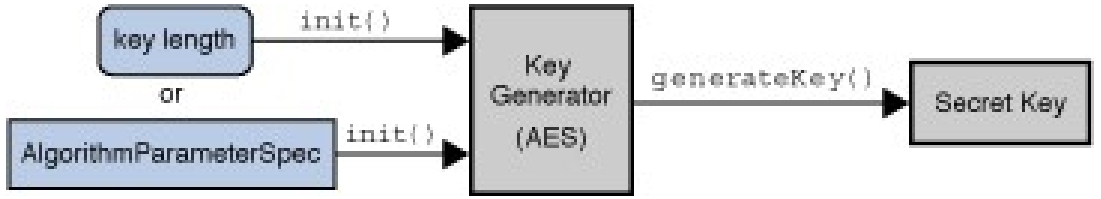
JCA – Conceitos Representações de Chaves

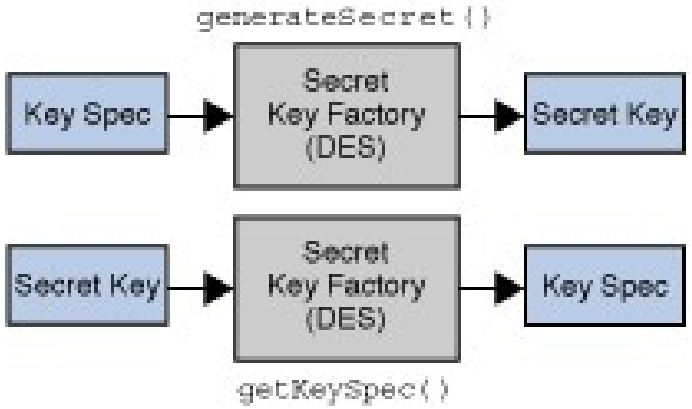
- Classes que representam chaves opacas
  - SecretKey; PBEKey; PrivateKey; DHPrivateKey; DSAPrivateKey; ECPrivateKey; RSAMultiPrimePrivateCrtKey; RSAPrivateCrtKey; RSAPrivateKey; PublicKey; DHPublicKey; DSAPublicKey; ECPublicKey; RSAPublicKey
- Classes que representam chaves transparentes
  - SeccretKeySpec; EncodedKeySpec; PKCS8EncodedKeySpec; X509EncodedKeySpec; DESKeySpec; DESedeKeySpec; PBEKeySpec; DHPrivateKeySpec; DSAPrivateKeySpec; ECPrivateKeySpec; RSAMultiPrimePrivateCrtKeySpec; RSAPrivateCrtKeySpec; RSAPrivateKeySpec; DHPublicKeySpec; DSAPublicKeySpec; ECPublicKeySpec; RSAPublicKeySpec
- Conversão entre chaves opacas e transparentes
  - Classe KeyFactory



JCA – Geração de Chaves Simétricas

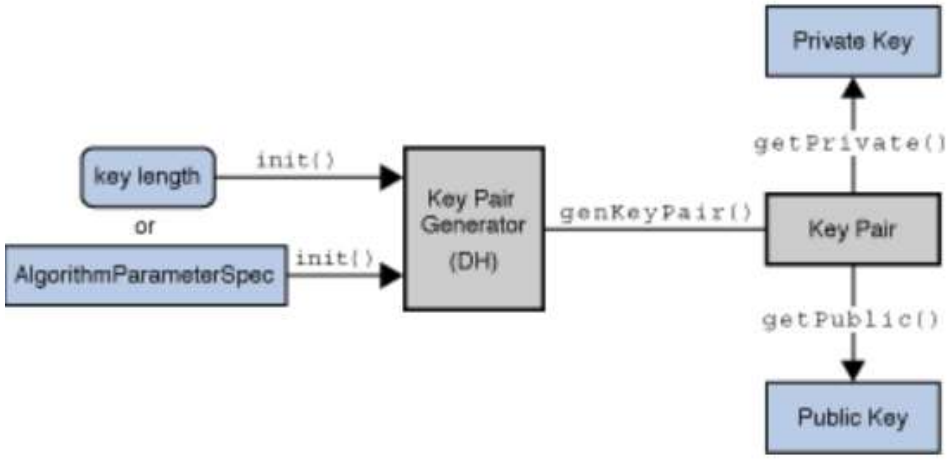
- Chaves simétricas podem ser produzidas através de duas classes:
  - KeyGenerator
    - Utiliza o método generatekey()
    - Necessário informar tamanho chave ou um objeto AlgoritmParameterSpec no método init()
    - Necessário obter instância do objeto, informando o algoritmo
  - SecretKeyFactory
    - Utiliza o método generateSecret()
    - Necessita de um objeto KeySpec relacionado como algortimo a ser utilizado
    - Necessário obter instância do objeto, informando o algoritmo





JCA – Geração de Chaves Assimétricas

- Chaves assimétricas podem ser produzidas através da classe:
  - KeyPairGenerator
    - Utiliza o método genkeyPair()
    - Necessário informar tamanho chave ou um objeto AlgoritmParameterSpec no método initialize()
    - Necessário obter instância do objeto, informando o algoritmo



JCA – Cifradores
Modos de Operação e Padding

- Cifrador pode ser utilizado da seguinte forma:
  - getInstance(“NomeAlgoritmo”) ou então:
  - getInstance(“NomeAlgoritmo/ModoOperacao/Padding”)
- Exemplos:
  - Cipher.getInstance("DES/CBC/PKCS5Padding")
  - Cipher. getInstance("DES")
- Caso o modo de operação ou o padding não seja especificado serão utilizados valores default definidos
  - Provider SunJCE utiliza como padrão: ECB e PKCS5Padding
    - Cipher c = Cipher.getInstance("DES/ECB/PKCS5Padding");
    - Cipher c = Cipher.getInstance("DES");

JCA – Cifradores
Modos de Operação

- Modos de Operação
  - NONE (Modo padrão)
  - CBC (Cipher Block Chaining Mode - FIPS PUB 81)
    - CBCx – Permite reduzir o tamanho do bloco definido pelo cifrador.
      - CBC8 indica que serão cifrados 8 bits por vez (Byte Oriented Stream)
  - CFB (Cipher Feedback Mode - FIPS PUB 81)
  - ECB (Electronic Codebook Mode)
  - OFB (Output Feedback Mode)
    - OFBx – Permite reduzir o tamanho do bloco definido pelo cifrador
      - OFB32 indica que serão cifrados 32 bits por vez
  - PCBC (Propagating Cipher Block Chaining, definido em Kerberos V4)
  - CTR (Counter mode)

JCA – Cifradores Padding (Inserção de Bits)

- Técnicas para inserção de bits a fim de completar o tamanho do bloco requerido pelo algoritmo
  - NoPadding (Não utiliza o padding)
  - ISO10126Padding
    - Definido no documento "XML Encryption Syntax and Processing" (W3C)
  - OAEPWith<digest>And<mgf>Padding:
    - Optimal Asymmetric Encryption Padding, definido em PKCS #1
    - <digest> - MD5, MD2
    - <mgf> - Mask Generation Functin
    - Exemplo: OAEPWithMD5AndMGF1Padding.
  - PKCS1Padding
    - Utilizado com algoritimo RSA e definido em PKCS1
  - PKCS5Padding
    - Descrito em "PKCS #5: Password-Based Encryption Standard
  - SSL3Padding
    - Padding utlizado pelo protocolo SSL.

JCA
Cifradores – Geração Chave

- Simétrica

```
KeyGenerator keygen;
keygen = KeyGenerator.getInstance("ALGORITHM_NAME");
keygen.init(KEY_SIZE);
SecretKey secretKey = keygen.generateKey();
```

//Utilizando material transparente para criação de chave opaca

```
SecretKeyFactory keyfactory = SecretKeyFactory.getInstance("DES");
SecretKey  secretKey;
byte[] keyMaterial = "01234567".getBytes();
DESKeySpec desKeySpec = new DESKeySpec(keyMaterial);
secretKey = keyfactory.generateSecret(desKeySpec);
```

JCA
Cifradores – Uso Algoritmo

- Obtenção do Objeto Cipher (**javax.crypto.Cipher**) – Método getInstance()

```
Cipher c;
c = Cipher.getInstance("DES");
```

- Inicialização do Algoritmo - Método init()

```
c.init(Cipher.ENCRYPT_MODE, secretKey);
```

- Cifragem - Método doFinal()

```
String plainText;
byte[] cipheredText, recoveredText;
cipheredText = c.doFinal(plainText.getBytes());
```

- Decifragem - Método doFinal()

```
c.init(Cipher.DECRYPT_MODE,secretKey);
recoveredText = c.doFinal(cipheredText);
```

JCA – Hash MessageDigest

- Responsável pela funcionalidade de criação de sumários (digest) ou hash de mensagens
- Recebe uma entrada de tamanho variável e produz saida de tamanho fixo
- Sumários representam “impressões digitais” da entrada de dados

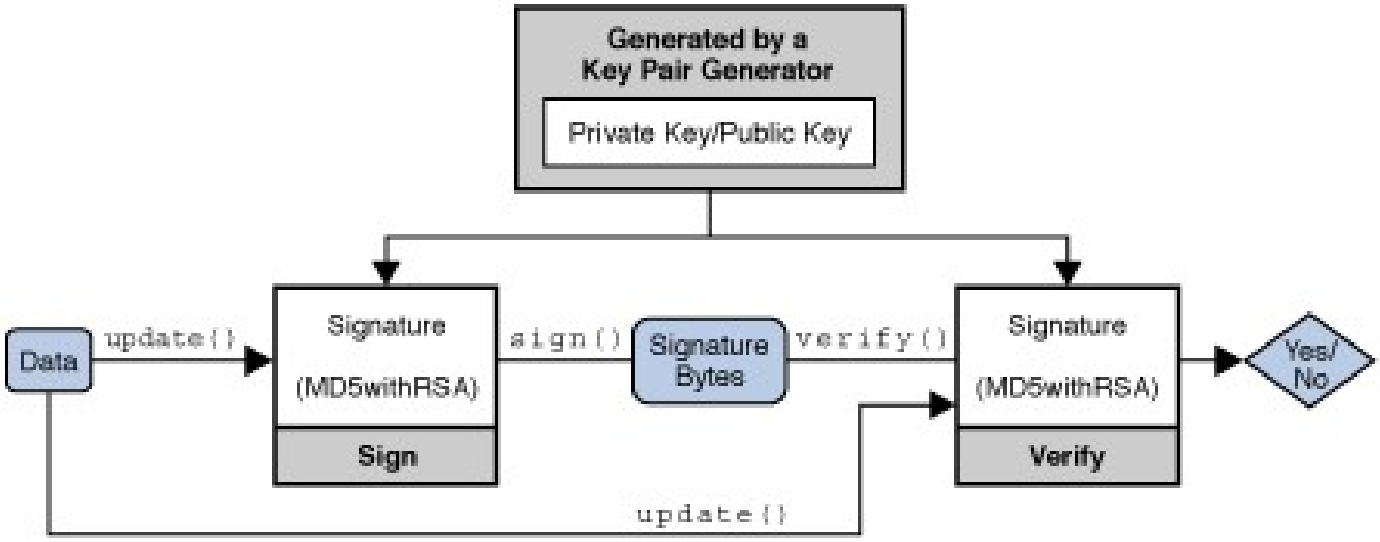
- Uso
  - Obter Instância do objeto
    - MessageDigest.getInstance
  - Inserir os dados de entrada
    - update()
  - Calcular o sumário
    - digest()

JCA – Hash MessageDigest - Algoritmos

- JCA utiliza os seguintes algortimos:
  - MD2 – Definido em RFC 1319
  - MD5 – Definido em RFC 1321.
  - SHA (SHA-1; SHA-256; SHA-384 e SHA-512)
    - Definido em FIPS PUB 180-2

JCA – Assinatura Digital Signature

- Produz uma assinatura digital
- Entrada
  - Conjunto de tamanho variável de bytes e uma chave privada
- Saida
  - Pequeno array de bytes chamado “assinatura digital”



JCA – Assinatura Digital Signature

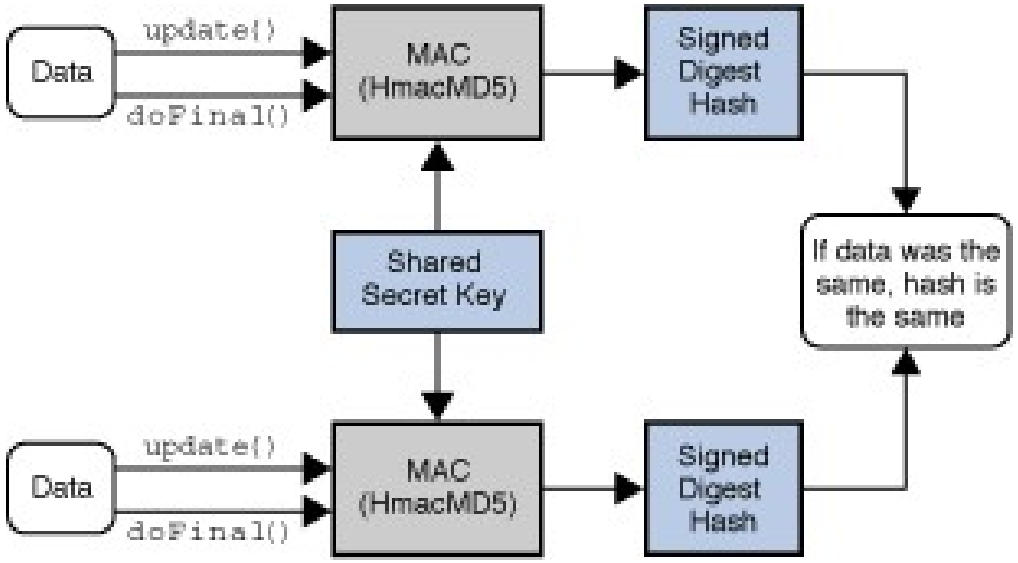
- Apesar de semelhante a um Hash a assinatura digital utiliza duas etapas
  - Hash – Compactar dados
  - Algoritmo Assimétrico - Assinatura do Hash
  - Algoritmo é referenciado como “<digest>with<encryption>”
    - SHA1withRSA
- Uso
  - Obtenção do objeto
    - Signature.getInstance()
  - Inicialização
    - Dependo do tipo de uso do objeto
      - Assinatura - initSign(PrivateKey)
      - Verificação - initVerify(PublicKey)
  - Execução do processo
    - Informar os dados – update()
    - Assinatura - byte[] sign()
    - Verificação - boolean verify(byte[] signature)

JCA – Assinatura Digital Signature - Algoritmos

- **ECDSA (Elliptic Curve Digital Signature Algorithm), descrito em ECC Cipher** Suites for TLS
  - NONEwithECDSA; SHA1withECDSA; SHA256withECDSA; SHA384withECDSA e SHA512withECDSA
- **MD2withRSA – Utiliza MD2 com o algoritmo RSA**
- **MD5withRSA**
- **NONEwithDSA – Assinaturas criadas e verificadas com o algoritmos DAS (FIPS PUB 186)**
- **SHA1withDSA**
- **SHA1withRSA**
- **<digest>with<encryption>**
  - **Forma Geral**
  - Digest – MD2 ou MD5
  - Encryption - RSA or DAS
- **<digest>with<encryption>and<mgf>**
  - Para os novos esquemas definidos em PKCS #1 v 2.0
  - **MGF – Mask Generation Function**
  - Exemplo: MD5withRSAandMGF1.

JCA – Message Autentiction Codes (Mac)

- Similar ao MessageDigest
- Permite verificar a integridade de uma informação que transitou ou foi armazenada em um meio inseguro
- Inclui uma chave secreta na verificação e somente quem possuí-la poderá verificar a integridade da mensagem
- MAC baseado em funções hash é chamado HMAC



JCA – Message Autentiction Codes (Mac)

- USO
  - Obtenção da Instância
    - Mac.getInstance()
  - Inicialização
    - init(Key key)
  - Cálculo do Mac
    - doFinal()

    Ou então
    - update() – informar pacotes de dados
    - doFinal()
- Algoritmos
  - HmacMD5
  - HmacSHA1; HmacSHA256; HmacSHA384; HmacSHA512
  - PBEWith<mac>, sendo <mac> um dos Hmac definidos acima