# *Java Cryptography Architecture* (JCA)

Notas para a UC de “Segurança Informática”

Pedro Félix (pedrofelix *em* cc.isel.ipl.pt)

José Simão (jsimao *em* cc.isel.ipl.pt)

Instituto Superior de Engenharia de Lisboa

# Sumário

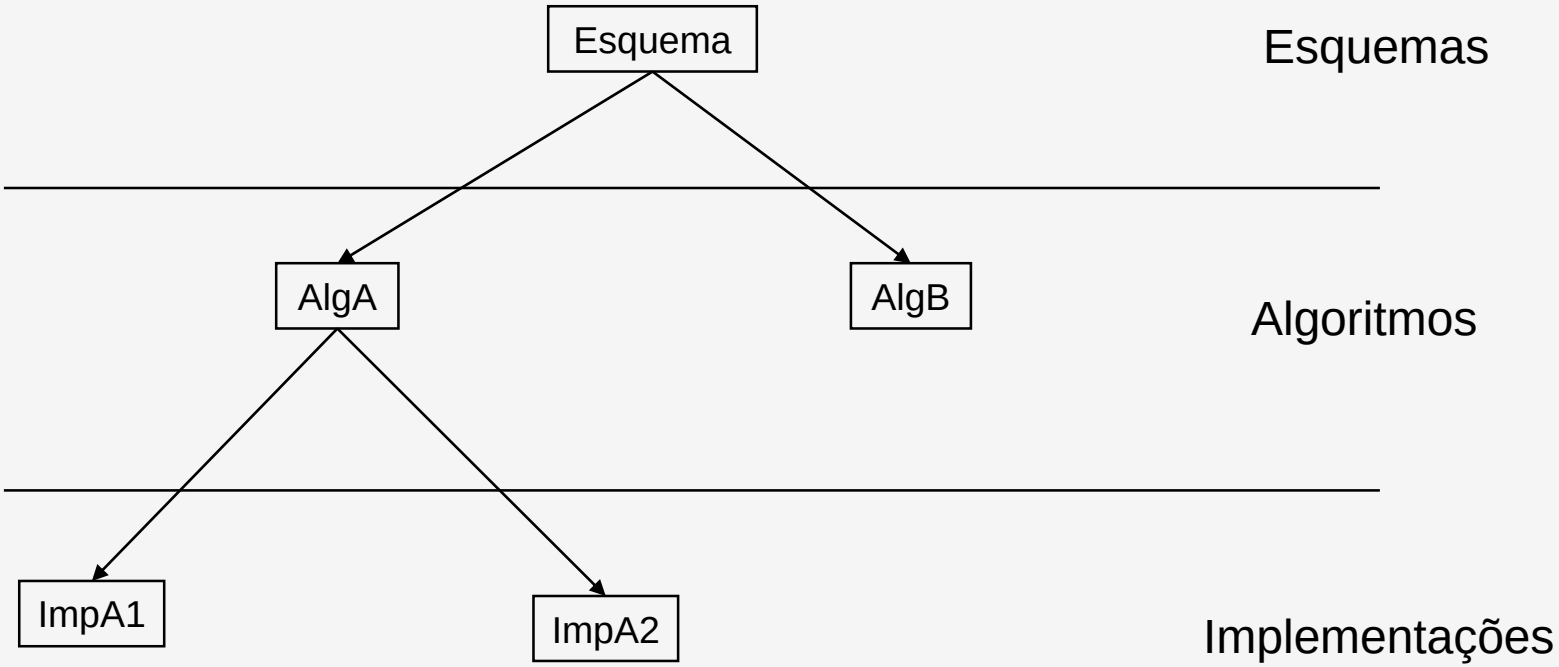
- Critérios de desenho da JCA
- Constituintes da arquitectura: *providers*, *engine classes*, *specification classes,* interfaces
- Tipos de *engine classes* e a relação com chaves
- Representações opacas e transparentes de chaves e parâmetros

# Princípios de desenho

- Independência dos algoritmos e expansibilidade
  - Utilização de esquemas criptográficos, como a assinatura digital e a cifra simétrica, independentemente dos algoritmos que os implementam
  - Capacidade de acrescentar novos algoritmos para os mecanismos criptográficos considerados

- Independência da implementação e interoperabilidade
  - Várias implementações para o mesmo algoritmo
  - Interoperabilidade entre várias implementações, por exemplo, assinar com uma implementação e verificar com outra
  - Acesso normalizado a características próprias dos algoritmos

# Esquemas, algoritmos e implementações

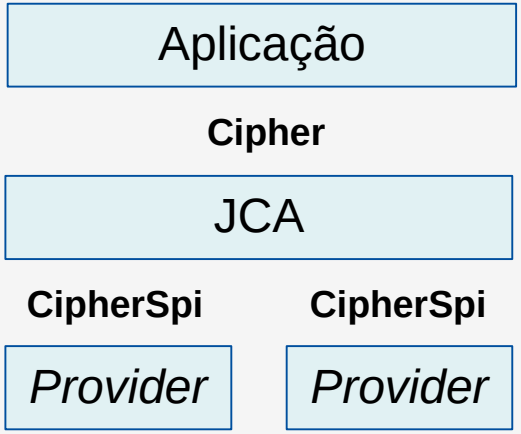


# Arquitectura

- Arquitectura baseada em:
  - CSP – *Cryptographic Service Provider*
    - *package* ou conjunto de *packages* que implementam um ou mais mecanismos criptográficos (serviços criptográficos)
  - *Engine Classes*
    - Definição abstracta (sem implementação) dum mecanismo criptográfico
    - A criação dos objectos é realizada através de métodos estáticos **getInstance**
  - *Specification Classes*
    - Representações normalizadas e transparentes de objectos criptográficos, tais como chaves e parâmetros de algoritmos.

# *Providers*

- Fornecem a *implementação* para as *engines classes*
  - Implementam as classes abstractas <*EngineClass*>Spi, onde *EngineClass* é o nome duma *engine class*
- Classe **Provider** é base para todos os *providers*
- Instalação
  - Colocar *package* na *classpath* ou na directoria de extensões
  - Registrar no ficheiro java.security
  - Em alternativa, usar a classe **Security**
- Classe **Security**
  - Registo dinâmico de *providers*
  - Listagem de *providers* e algoritmos



# *Engines classes*

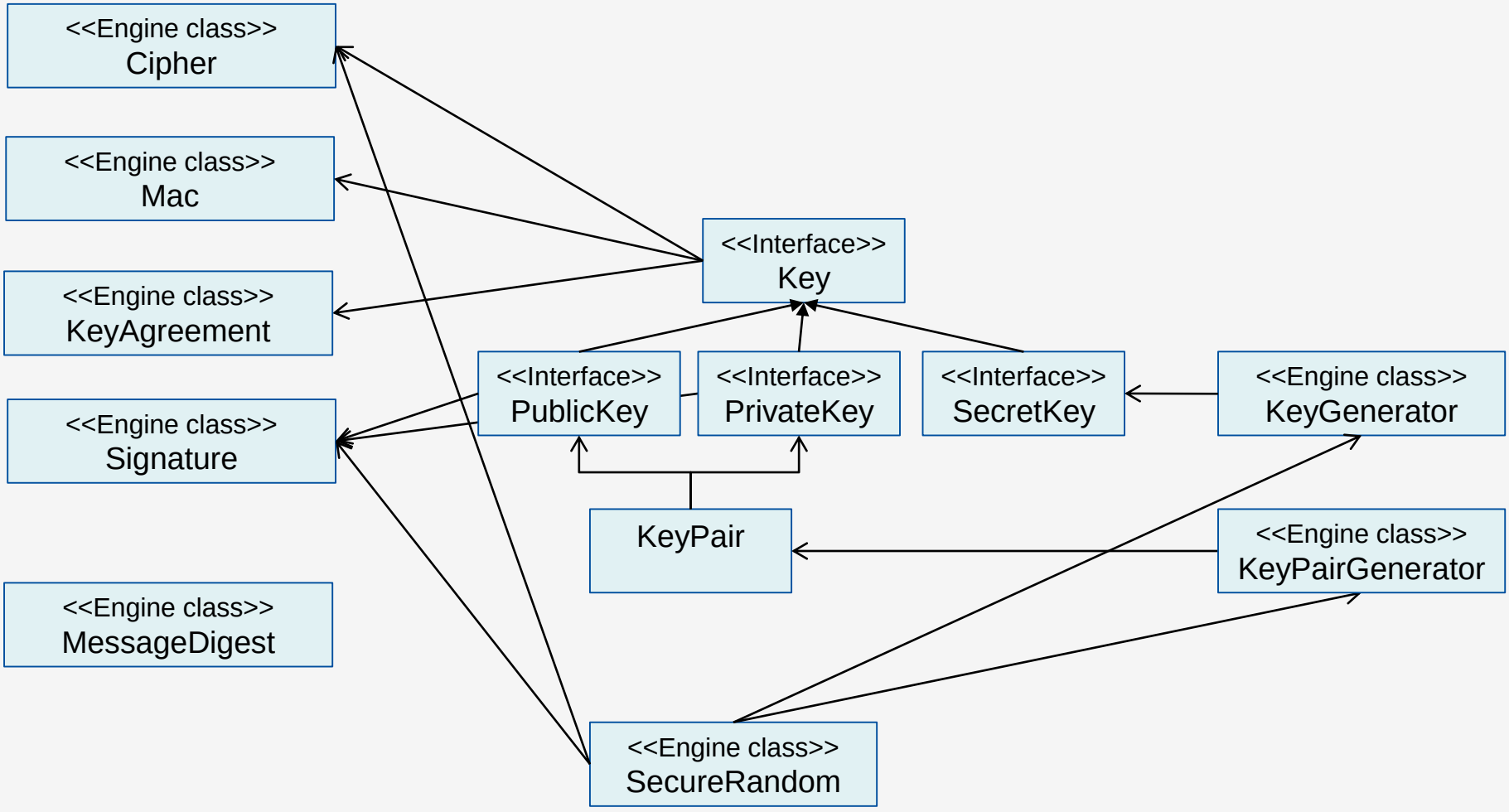
- Classes: **Cipher** (esquemas simétricos e assimétricos), **Mac**, **Signature**, **MessageDigest**, **KeyGenerator**, **KeyPairGenerator**, **SecureRandom**, …

- Métodos *factory*
  - **static Cipher getInstance(String transformation)**
  - **static Cipher getInstance(String transformation, String provider)**
  - **static Cipher getInstance(String transformation, Provider provider)**

- Os algoritmos concretos e as implementações concretas (*providers*) são identificados por *strings*

- *Delayed Provider Selection*
  - A Selecção do *provider* adequado é adiada até à iniciação com a chave
  - Permite a selecção do *provider* com base no tipo concreto da chave

# *Engines classes* e chaves



# *Streams*

- Classe **CipherInputStream : FilterInputStream**
  - Processa (cifra ou decifra) os *bytes* lidos através do *stream*
  - **ctor(InputStream, Cipher)**

- Classe **CipherOutputStream : FilterOutputStream**
  - Processa (cifra ou decifra) os *bytes* escritos para o *stream*

- Class **DigestInputStream : FilterInputStream**
  - Processa (calcula o *hash*) os *bytes* lidos através do *stream*
  - **ctor(InputStream, MessageDigest)**
  - **MessageDigest getMessageDigest()**

- Class **DigestOutputStream : FilterOutputStream**
  - Processa (calcula o *hash*) os *bytes* escritos para o *stream*

# Transformações normalizadas

- Ver apêndice A de “Java Cryptography Architecture (JCA) Reference Guide”
- **Cipher**
  - “algorithm/mode/padding” ou “algorithm”
  - *algorithm*: AES, DES, DESede, RSA, …
  - *mode*: ECB, CBC, CFB, CTR, OFB, …
  - *padding*: PKCS5Padding, PKCS1Padding, OAEPPadding
- **Mac**
  - hmac[MD5 | SHA1 | SHA256 | SHA384 | SHA512], …
- **Signature**
  - [MD5 | SHA1 | …]withRSA, SHA1withDSA, …
- **KeyGenerator**
  - AES, DES, DESede, Hmac[MD5 | SHA1 |…], …
- **KeyPairGenerator**
  - RSA, DSA, …

# Chaves

- Interface **Key**
  - **String getAlgorithm()**
  - **byte[] getEncoded()**
  - **String getFormat()**
- Interfaces **SecretKey**, **PublicKey** e **PrivateKey**
  - Derivam de **Key**
  - Não acrescentam métodos (*marker interfaces*)
- Classe concreta **KeyPair**
  - Contém uma **PublicKey** e uma **PrivateKey**
- Geração através das *engine classes* **KeyGenerator** e **KeyPairGenerator**

# Representações: opacas e transparentes

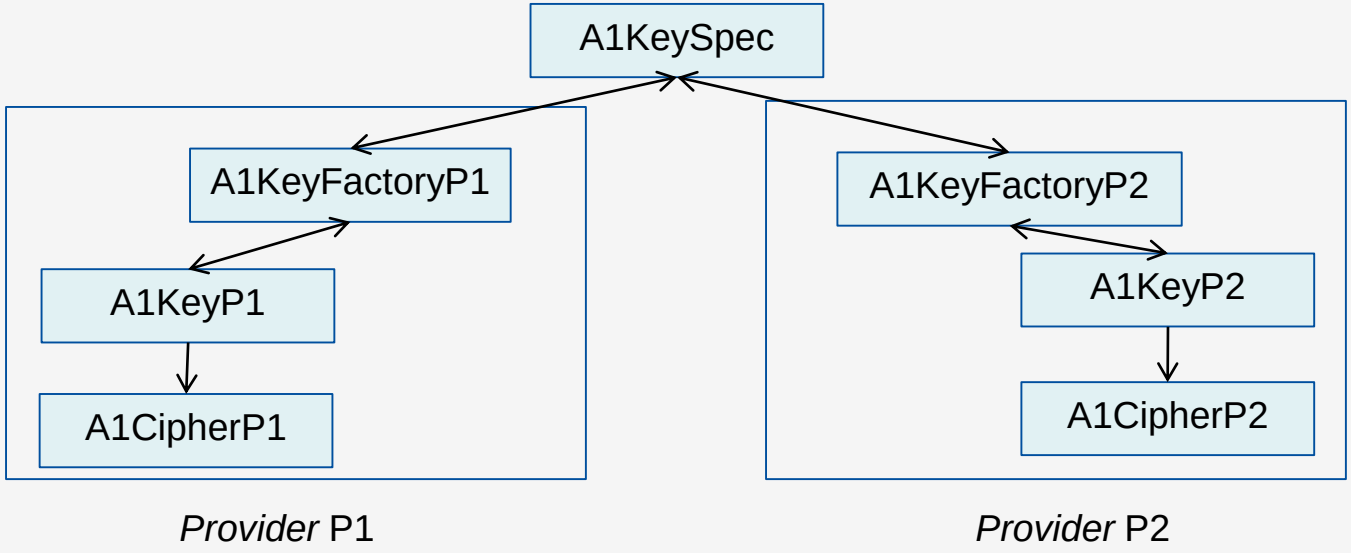
- Chaves opacas: representações de chaves sem acesso aos seus constituintes
  - Derivadas da interface **Key**
  - Específicas de cada *provider*
  - Geradas pelas *engine classes* **KeyGenerator** e **KeyPairGenerator**

- Chaves transparentes: representações de chaves com acesso aos seus constituintes
  - Derivadas da interface **KeySpec**
  - Os packages **java.security.spec e javax.crypto.spec** definem um conjunto de classes **<*nome*>Spec** com interface normalizada para o acesso aos constituintes das chave de diversos algoritmos.
  - Exemplos: **RsaPublicKeySpec**, **DESKeySpec**, **SecretKeySpec**, …

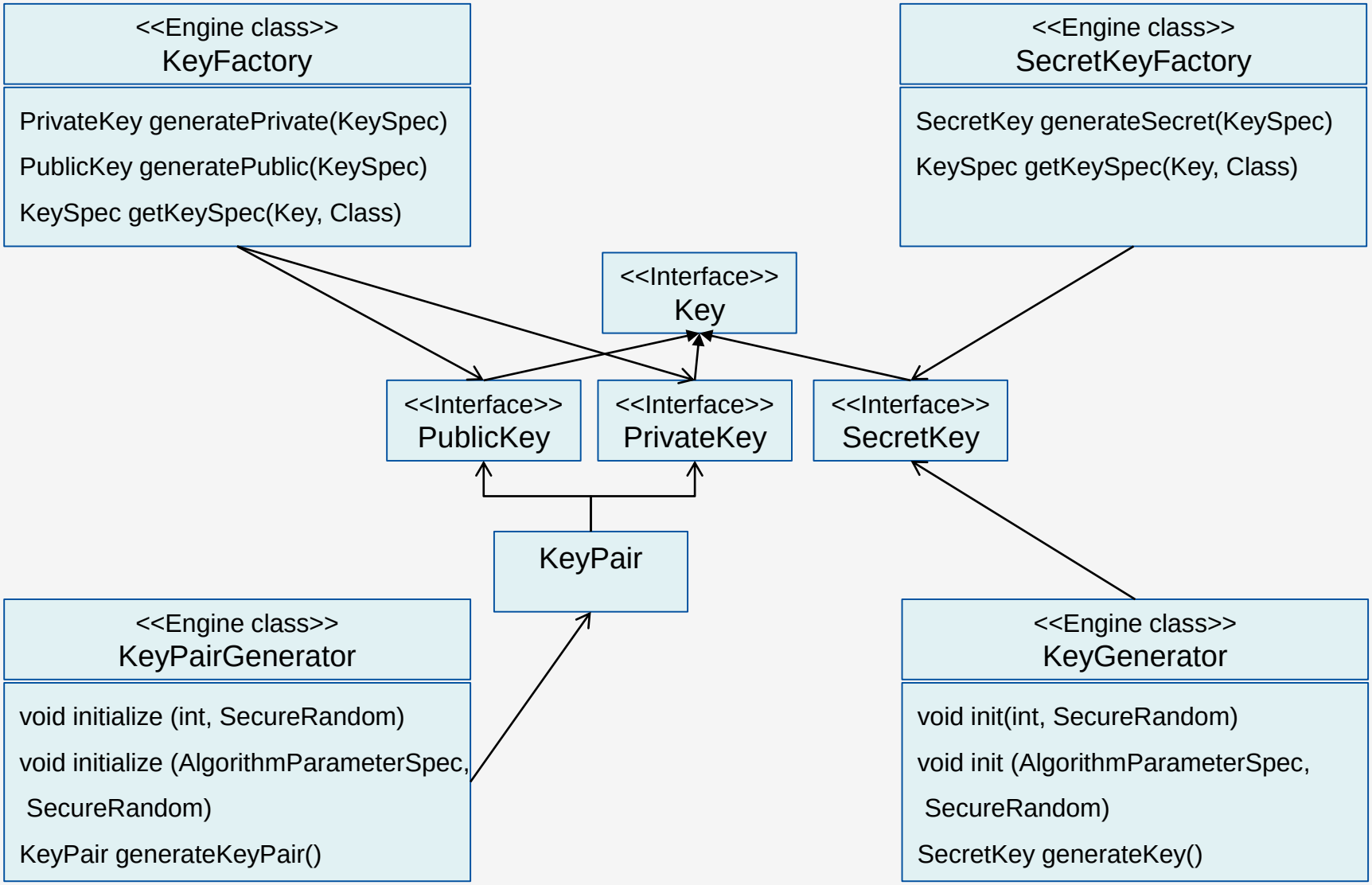
- **KeyFactory** – *engine class* para conversão entre representações opacas e transparentes

# *Key factories*

- **KeyFactory**
  - *engine class* – implementações específicas do *provider*
  - **static KeyFactory getInstance(…)**
  - **PrivateKey generatePrivate(KeySpec)**
  - **PublicKey generatePublic(KeySpec)**
  - **KeySpec getKeySpec(Key key, Class spec)**
- Independência da implementação e interoperabilidade



# Chaves, geradores e fábricas



# Parâmetros

- Parâmetros adicionais dos algoritmos
  - Exemplos: vector inicial (IV), dimensões das chaves

- Representação opaca: engine class **AlgorithmParameters**
- Representação transparentes: classes que implementam a interface **AlgorithmParameterSpec**
  - Exemplos: **IvParameterSpec**, **RsaKeyGenParameterSpec**,…

- Geração: *engine class* **AlgorithmParameterGenerator**
- Transformação entre representações transparentes e representações opacas: métodos de **AlgorithmParameters**

# Classe **Cipher**

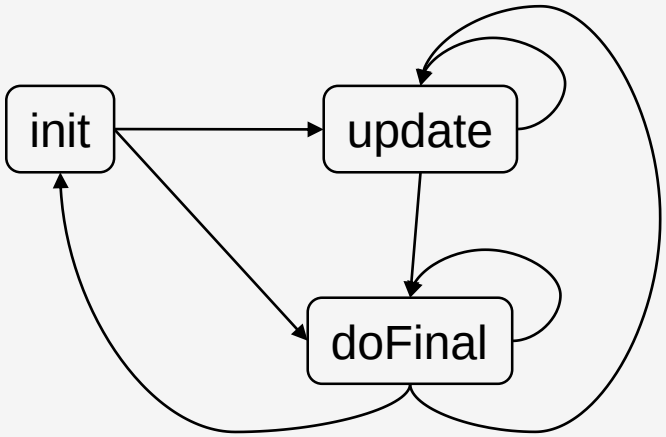
- Método **init** (várias sobrecargas)
  - Parâmetros: modo (cifra, decifra, *wrap* ou *unwrap*), chave, parâmetros específicos do algoritmo e gerador aleatorio

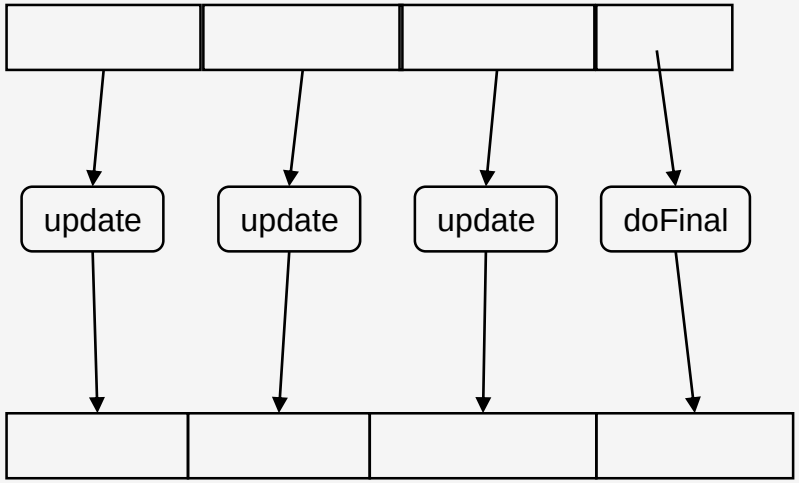
- Métodos de cifra
  - **update**: **byte[]** → **byte[]** – continua a operação incremental
  - **doFinal: byte[]** → **byte[]** – finaliza a operação incremental
  - **wrap**: **Key** → **byte[]** – cifra chave
  - **unwrap: byte[], …** → **Key** – decifra chave

- Métodos auxiliares
  - **byte[] getIV()**
  - **AlgorithmParameters getParameters()**
  - …

# **Cipher**: operação incremental

- Nota: **E(k)(m1|| m2) ≠ E(k)(m1) || E(k)(m2)**
- Cifra de mensagens com grande dimensão ou disponibilizadas parcialmente
  - **Método update** recebe parte da mensagem e retorna parte do criptograma
  - **Método doFinal** recebe o final da mensagem e retorna o final do criptograma



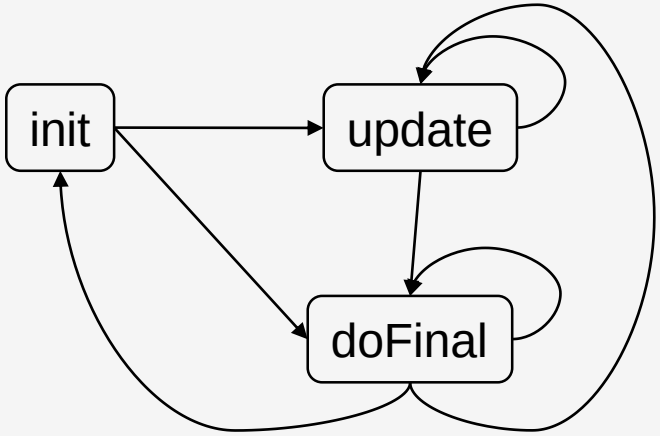



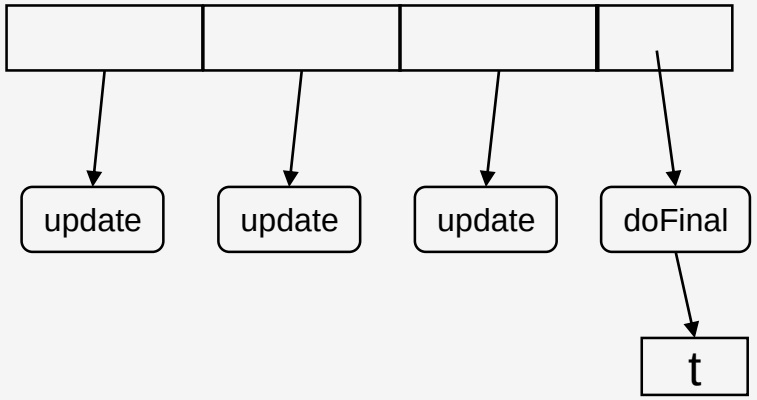
# Classe **Mac**

- Método **init** (várias sobrecargas)
  - Parâmetros: chave e parâmetros específicos do algoritmo
- Métodos de geração de marca
  - **update**: **byte[]** → **void** – continua a operação incremental
  - **doFinal: byte[]** → **byte[]** – finaliza a operação incremental, retornando a marca
- Métodos auxiliares
  - **int getMacLength()**
  - …

# **Mac**: operação incremental

- MAC de mensagens com grande dimensão ou disponibilizadas parcialmente
  - Método **update** recebe parte da mensagem
  - Método **doFinal** recebe o final da mensagem e retorna a marca



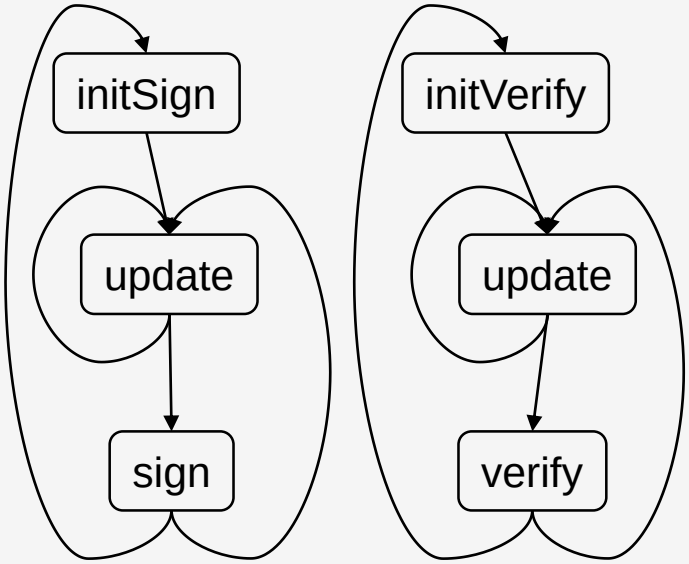
# Classe **Signature**

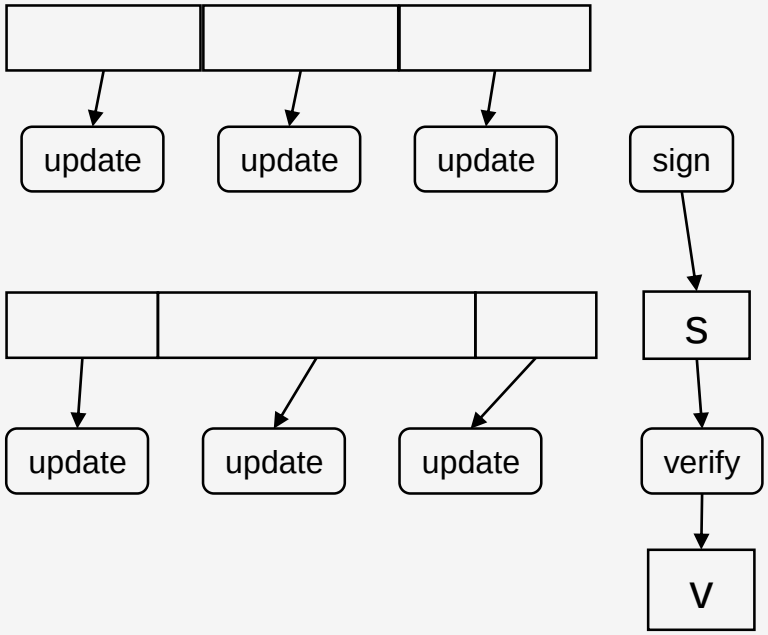
- Método **initSign** (várias sobrecargas)
  - Parâmetros: chave privada e gerador aleatório
- Método **initVerify** (várias sobrecargas)
  - Parâmetros: chave pública

- Métodos de geração de assinatura
  - **update**: **byte[]** → **void** – continua a operação incremental
  - **sign: void** → **byte[]** – finaliza operação incremental, retornando a assinatura
- Métodos de verificação de assinatura
  - **update**: **byte[]** → **void** – continua a operação incremental
  - **verify**: **byte[]** → **{true,false}** – finaliza a operação incremental, retornando a validade da assinatura

# **Signature**: operação incremental

- Assinatura/verificação de mensagens com grande dimensão ou disponibilizadas parcialmente
  - **update** recebe as parte da mensagem
  - **sign/verify** produz/verifica a assinatura



# *Key Stores*

- Armazenamento de chaves e certificados
- Representação através da *engine class* **KeyStore**
- Três tipos de entrada
  - Chaves privadas e certificados associados (incluindo a cadeia)
  - Chaves simétricas
  - Certificados representando *trust anchors*
- Protecção baseada em *passwords*
  - Integridade de todo o repositório - uma password por repositório
  - Confidencialidade das entradas contendo chaves privadas ou secretas - uma password por entrada do repositório
- Formatos de ficheiro (tipos de *provider*)
  - JKS – Formato proprietário da *Sun*
  - JCEKS – Evolução do formato JCE com melhor protecção
  - PKCS12 – Norma PKCS#12 (usada nos ficheiros .pfx criados pelo *Windows*)

# Entradas

- Cada entrada tem associado um *alias*, do tipo **String**
- Interface base **Entry**

- **PrivateKeyEntry -** Chaves privadas e certificados associados
  - **ctor(PrivateKey, Certificate[])**
  - **Certificate getCertificate()**
  - **Certificate[] getCertificateChain()**
  - **PrivateKey getPrivateKey()**
- **SecretKeyEntry -** Chaves simétricas
  - **ctor(SecretKey)**
  - **SecretKey getSecretKey()**
- **TrustedCertificateEntry -** *Trust anchors*
  - **ctor(Certificate)**
  - **Certificate getTrustedCertificate()**

# Classe **KeyStore**

- *Load* e *store*
  - **void load(InputStream, char[] password)**
  - **void load(LoadStoreParameter)**
  - **void store(OutputStream, char[] password)**
  - **void store(LoadStoreParameter)**
- Listagem
  - **Enumeration<String> aliases()**
- Acesso a entradas
  - **Entry getEntry(String alias, ProtectionParameter)**
  - **void setEntry(String alias, Entry, ProtectionParameters)**
  - **boolean is*Xxx*Entry(String alias)**
- Métodos especializados
  - **Key getKey(String alias, char[] password)**
  - …