Software Interfaces to Cryptographic Algorithms

Pieter Philippaerts
Pieter.Philippaerts@cs.kuleuven.be

Research Group

Based on slides by Frank Piessens

Katholieke Universiteit Leuven
departement computerwetenschappen

Overview

- Design Principles
- The native Windows CryptoAPI
  - Cryptography API: Next Generation (CNG)
- The Java Cryptography Architecture and Extensions (JCA/JCE)
- The .NET Cryptographic Library
- The OpenBSD Cryptographic Framework
- Key management issues
- Conclusion

Katholieke Universiteit Leuven
departement computerwetenschappen

Crypto Timeline

- 1998: DES brute-forced in 22 hours
- 2000: AES selected as NIST standard
- 2001: RC4 weakness breaks WEP
- 2002: end-of-life of DES (inc. TripleDES), SHA2 introduced
- 2004: MD5, RIPEMD broken
- 2005: SHA1 broken
- 2012: SHA3 introduced

Katholieke Universiteit Leuven
departement computerwetenschappen

Design Requirements

- **New algorithms get introduced**
  - The architecture should be extensible
- **Algorithms get broken**
  - Developers should be able to easily replace one algorithm with another

Design Principles

- **Algorithm independence**
  - Engine classes / Factory methods
- **Implementation independence**
  - Provider-based architecture
- **Implementation interoperability**
  - Transparent and opaque data types

Bottom Line: security mechanisms should be easy to change over time

Katholieke Universiteit Leuven
departement computerwetenschappen

Opaque vs transparent data

- **Representation of data items like keys, algorithm parameters, initialization vectors:**
  - Opaque: chosen by the implementation object
  - Transparent: chosen by the designer of the cryptographic API
- **Transparent data allow for implementation interoperability**
- **Opaque data allow for efficiency or hardware implementation**

Katholieke Universiteit Leuven
departement computerwetenschappen

Crypto frameworks and CSP's

- A ***cryptographic framework*** **defines:**
  - Engine classes (and possibly algorithm classes)
  - Transparent key and parameter classes
  - Interfaces for opaque keys and parameters
- A ***cryptographic service provider*** **defines:**
  - Implementation classes
  - Opaque key and parameter classes
  - Possibly methods to convert between opaque and transparent data

Overview

- Design Principles
- The native Windows CryptoAPI
  - Cryptography API: Next Generation (CNG)
- The Java Cryptography Architecture and Extensions (JCA/JCE)
- The .NET Cryptographic Library
- The OpenBSD Cryptographic Framework
- Key management issues
- Conclusion

Katholieke Universiteit Leuven
departement computerwetenschappen

The Windows CryptoAPI

- Introduced with Windows 95/Windows NT4
- C-based library
- Still used by most Windows programs today

Katholieke Universiteit Leuven
departement computerwetenschappen

Cryptographic Service Providers

- **Pluggable libraries**
- **Implement different cryptographic algorithms**
- **Own a key database**
- **Windows and IE ship with a number of CSPs**
  - Depending from version to version and language to language

Katholieke Universiteit Leuven
departement computerwetenschappen

Key databases

- **Stores persistent keys**
- **Contains a number of *key containers***
  - Has a unique name
  - One for each user
  - Applications can create new containers
- **Saved in a secure file**
  - With access control
  - Optional *'strong protection'*

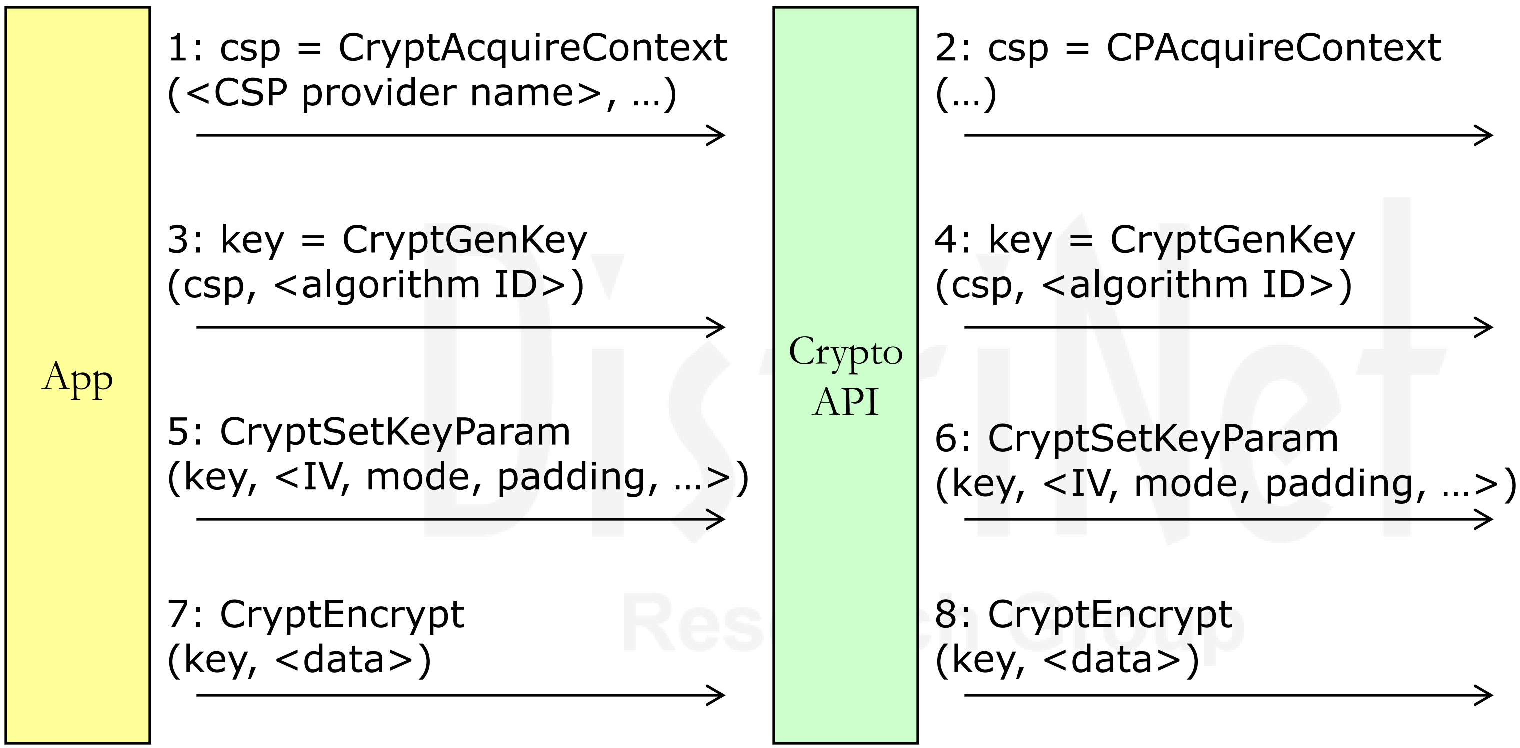
Keys

- **Session keys**
  - Used for symmetric encryption
  - Volatile
- **Public/private key pairs**
  - Typically two pairs per user (one for key exchange, one for digital signatures)
- **They are opaque**
  - All you get is an identification number (handle)
  - You can export them, though

Katholieke Universiteit Leuven
departement computerwetenschappen

The Windows CryptoAPI

- **Example: encrypt data**



Katholieke
Universiteit
Leuven
departement computerwetenschappen

Additional support for…

- **Cryptographic Message Syntax (CMS)**
- **Public key infrastructure**
- **Smart cards**
- **Authenticode**
- **XML signatures**
  - Windows 7 and higher

Katholieke Universiteit Leuven
departement computerwetenschappen

Summary

- **To add an algorithm, a new CSP must be implemented**
  - Not easy
- **Impossible to write algorithm independent code**
  - There is no notion of 'a default algorithm'
  - However, there *are* defaults for implementations
- **A CSP is an island; you cannot modify its behavior**

Katholieke Universiteit Leuven
departement computerwetenschappen

Cryptography: Next Generation

- **Introduced with Windows Vista**
- **Aims to replace Windows CryptoAPI**
  - Hence, also a C-based library
- **Has benefits over the CryptoAPI**
  - Easy plug-in creation, better extensibility
  - Crypto isolation
  - Support for algorithmic independence
    - In CMS, SSL/TLS, …, your application

Katholieke Universiteit Leuven
departement computerwetenschappen

Overview

- Design Principles
- The native Windows CryptoAPI
  - Cryptography API: Next Generation (CNG)
- The Java Cryptography Architecture and Extensions (JCA/JCE)
- The .NET Cryptographic Library
- The OpenBSD Cryptographic Framework
- Key management issues
- Conclusion

Katholieke Universiteit Leuven
departement computerwetenschappen

The JCA/JCE

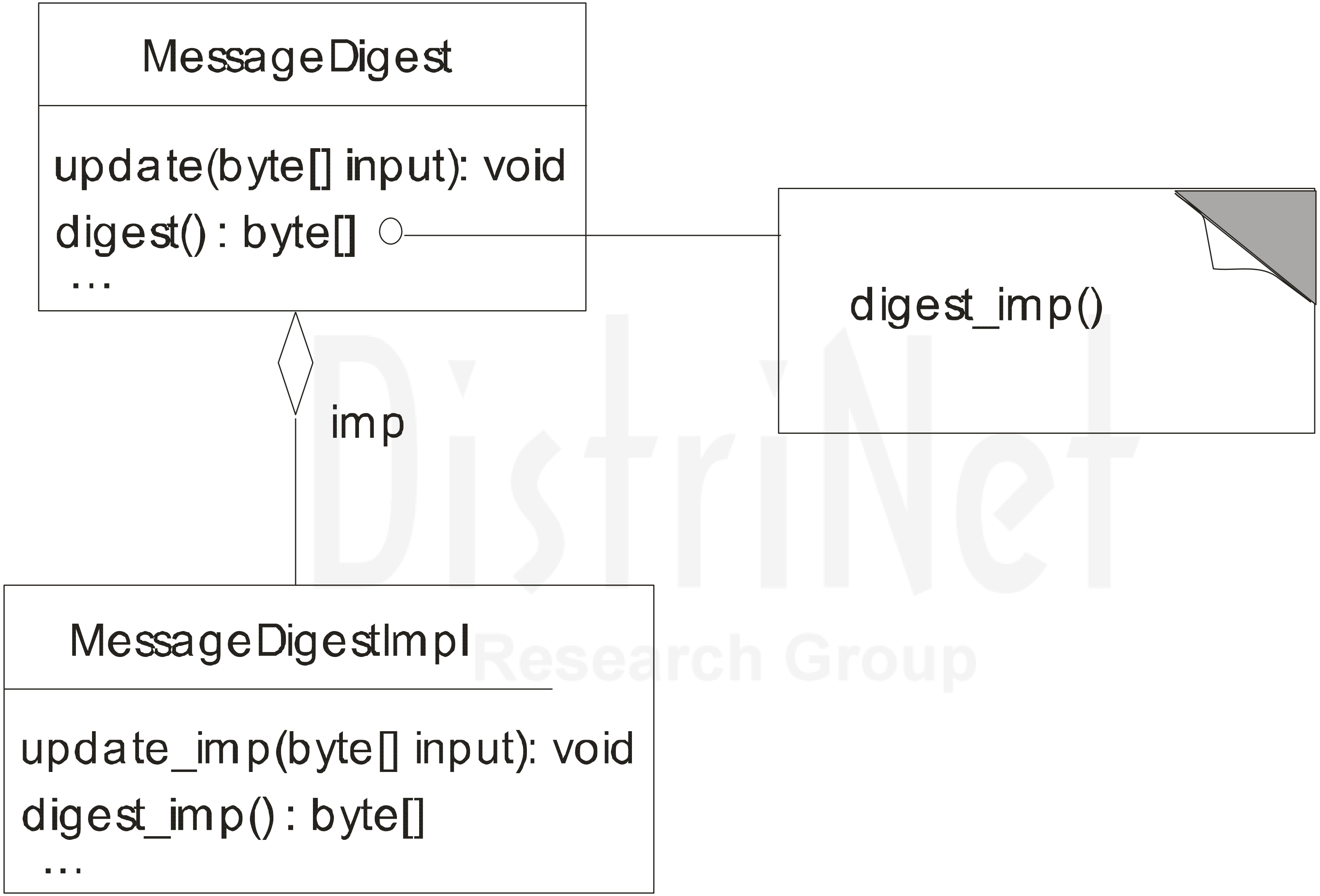
- **Java Crypto API structured as a cryptographic framework with CSPs**
- **Split in:**
  - The *Java Cryptography Architecure (JCA)*
  - The *Java Cryptography Extensions (JCE)*
- **This split is because of US export-control regulations for cryptography**

Engine classes

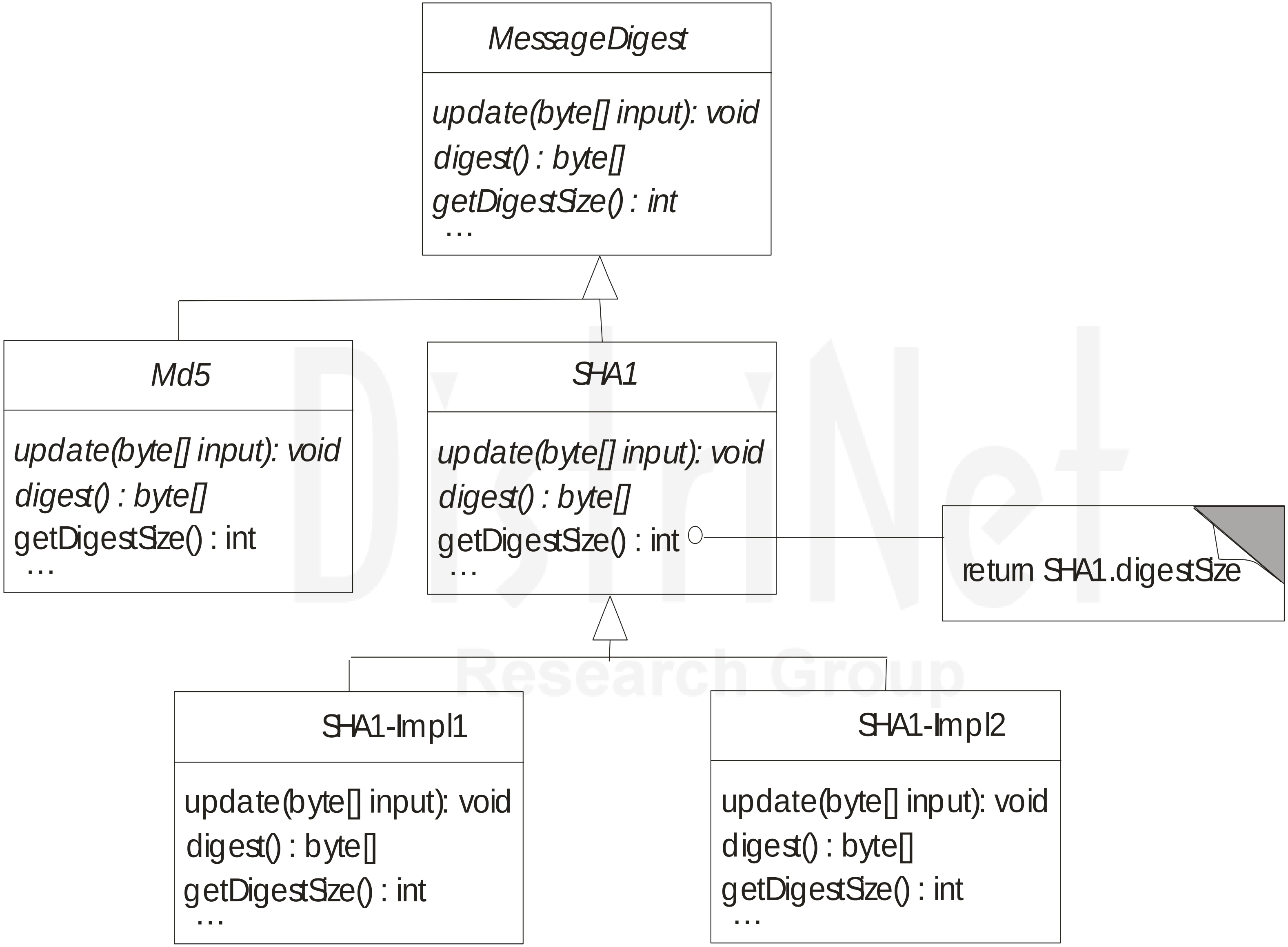
- **Abstraction for a cryptographic service**
  - Provide cryptographic operations
  - Generate/supply cryptographic material
  - Generate objects encapsulating cryptographic keys
- **Define the Cryptographic API**
- **Bridge pattern or inheritance hierarchy to allow for implementation independence**
- **Instances created by factory method**

Katholieke Universiteit Leuven
departement computerwetenschappen

Bridge Pattern



Inheritance-based decoupling



Engine classes (JCA)

java.security.*

- **MessageDigest**
  hash functions
- **Signature**
- **SecureRandom**
- **KeyPairGenerator**
  generate new key pairs
- **KeyFactory**
  convert existing keys
- **CerticateFactory**
  generate certificates from encoded form
- **KeyStore**
  database of keys
- **AlgorithmParameters**
- **AlgorithmParameter-Generator**

Katholieke Universiteit Leuven
departement computerwetenschappen

Engine classes (JCE)

javax.crypto.*

- **Cipher**

    encryption, decryption

- **Mac**
- **KeyGenerator**

    generate new symmetric keys

- **SecretKeyFactory**

    convert existing keys

- **KeyAgreement**

Katholieke Universiteit Leuven
departement computerwetenschappen

Key classes

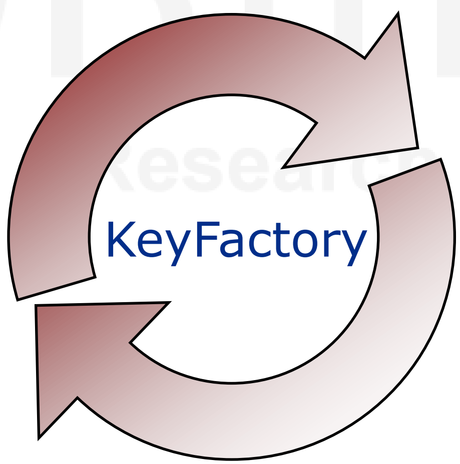
Katholieke Universiteit Leuven
departement computerwetenschappen

Opaque Representation

- No direct access to key material
- Encoded in provider-specific format
- **java.security.Key**

Transparent Representation

- Access each key material value individually
- Provider-independent format
- **java.security.KeySpec**



y = …
p = …
q = …
g = …

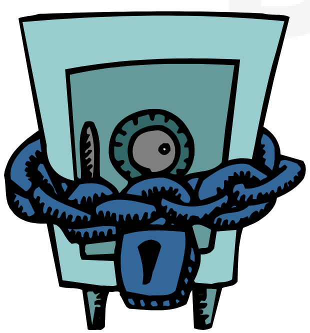
Parameter classes

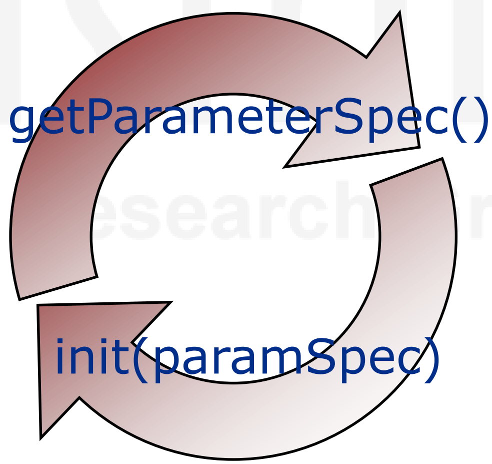
Opaque Representation

- No direct access to parameter fields
- Encoded in provider-specific format
- **AlgorithmParameters**

Transparent Representation

- Access each parameter value individually
- Provider-independent format
- **AlgorithmParameterSpec**





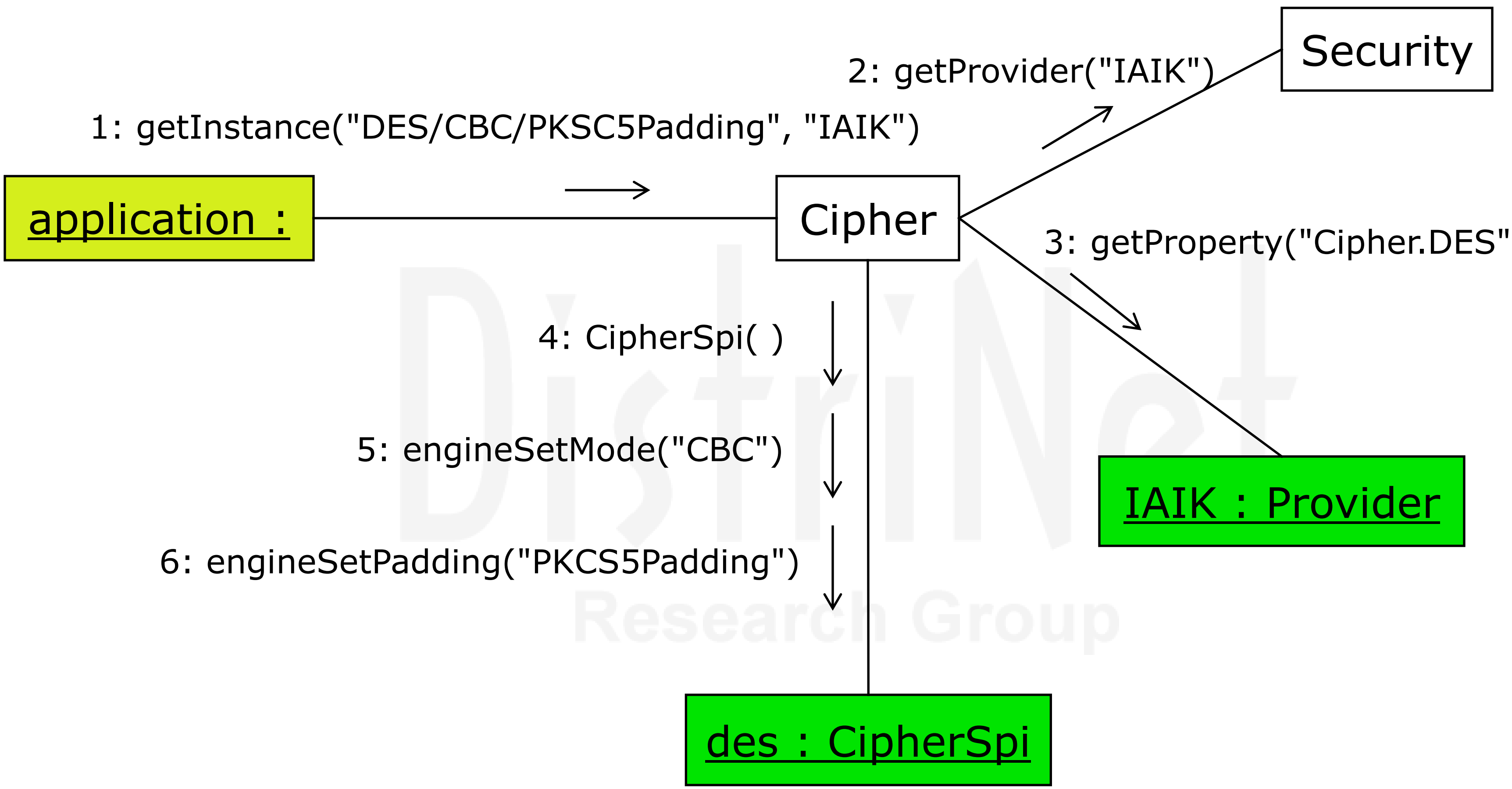
g = …
p = …
q = …

Overall structure of the framework

- Security class encapsulates configuration information (what providers are installed)
- Per provider, an instance of the provider class contains provider specific information (e.g. what algorithms are implemented in what classes)
- Factory method on the engine class interacts with the Security class and provider objects to instantiate a correct implementation object

Katholieke Universiteit Leuven
departement computerwetenschappen

Example: creating ciphers



Katholieke Universiteit Leuven

departement computerwetenschappen

Additional support

- **Secure streams**
  - For easy bulk encryption and decryption
- **Signed objects**
  - Integrity checked serialized objects
- **Sealed objects**
  - Confidentiality protected serialized objects
- **Working with certificates**
- **Keystores**

Katholieke Universiteit Leuven
departement computerwetenschappen

Summary

- **Very easy integration of new classes**
  - Inherit from the correct class
- **Cryptographic configuration**
  - To set the defaults

Katholieke Universiteit Leuven
departement computerwetenschappen

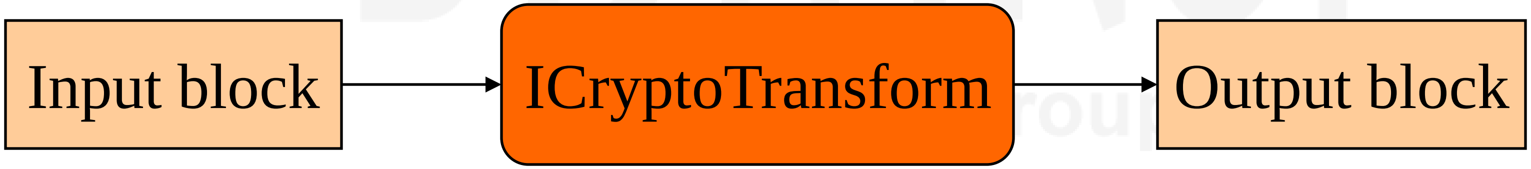
Overview

- Design Principles
- The native Windows CryptoAPI
  - Cryptography API: Next Generation (CNG)
- The Java Cryptography Architecture and Extensions (JCA/JCE)
- The .NET Cryptographic Library
- The OpenBSD Cryptographic Framework
- Key management issues
- Conclusion

Katholieke Universiteit Leuven
departement computerwetenschappen

The .NET cryptographic library

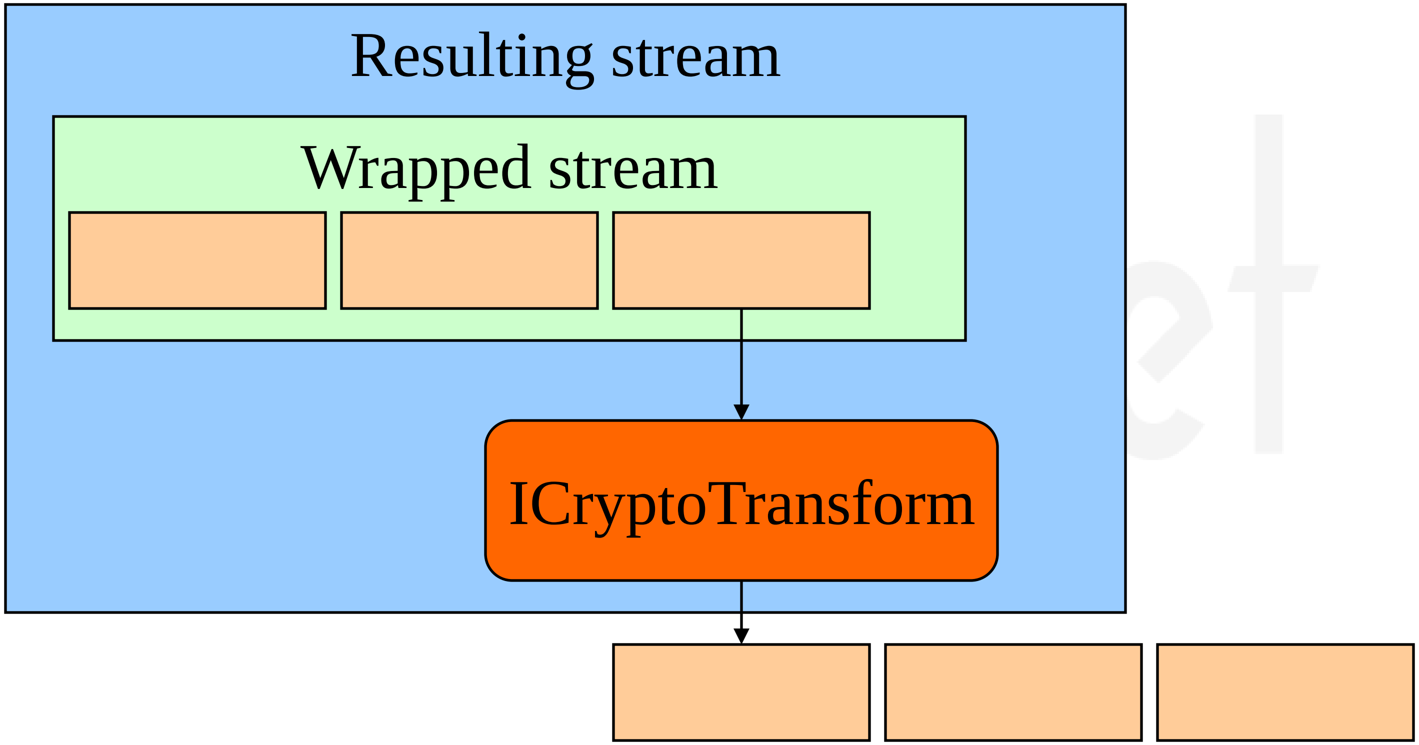
- CSP based library that uses inheritance based decoupling
- Bulk data processing algorithms are all made available as ICryptoTransforms
- Essentially 2 methods: TransformBlock() and TransformFinalBlock()



Katholieke Universiteit Leuven
departement computerwetenschappen

ICryptoTransform and CryptoStream

- ICryptoTransforms can wrap streams
  E.g. (in read mode)



Bulk data engine classes

- **SymmetricAlgorithm, with algorithm classes**
  - TripleDES, DES, Rijndael, …
- **HashAlgorithm, with algorithm classes**
  - SHA1, MD5, …
- **KeyedHashAlgorithm, with algorithm classes**
  - HMACSHA1, MACTripleDES, …

Katholieke Universiteit Leuven
departement computerwetenschappen

Asymmetric engine classes

- **Generic AsymmetricAlgorithm engine class**
  - RSA, (EC)DSA and ECDH algorithm classes
- **Specialized engine classes for typical uses of asymmetric cryptography, that take care of padding and formatting**
  - AsymmetricKeyExchangeFormatter
  - AsymmetricSignatureFormatter

Katholieke Universiteit Leuven
departement computerwetenschappen

Engine classes for key generation

- **RandomNumberGenerator**
  - For generating secure random numbers
- **DeriveBytes**
  - For deriving key material from passwords

Other functionality…

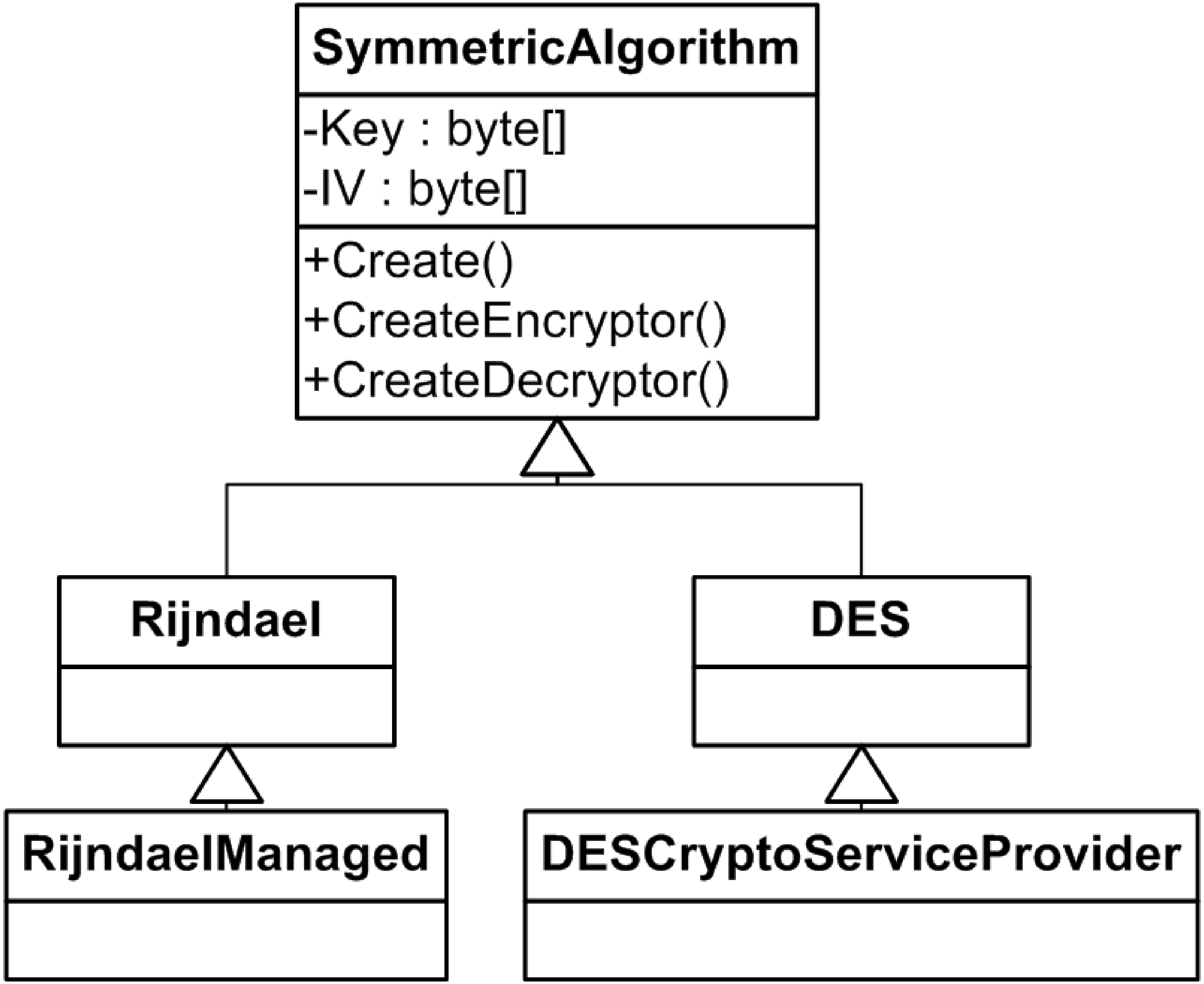
- **Facilities for interacting with Windows CryptoAPI / CNG**
  - To manage CryptoAPI Key containers manually
  - To call extended functionality in CryptoAPI
- **Configuration mechanism**
  - The factory methods that create engine classes are driven by a configuration file that can be edited to change default algorithms and implementations
- **On top of the .NET crypto API, an implementation of XML Digital Signatures is provided**

Katholieke Universiteit Leuven
departement computerwetenschappen

Differences between Java and .NET

- **.NET class structure is much simpler**
  - Hardly support for opaque keys
  - Wrappers around the CryptoAPI
  - Perhaps too simple?

Problems with transparent keys



Katholieke Universiteit Leuven
departement computerwetenschappen

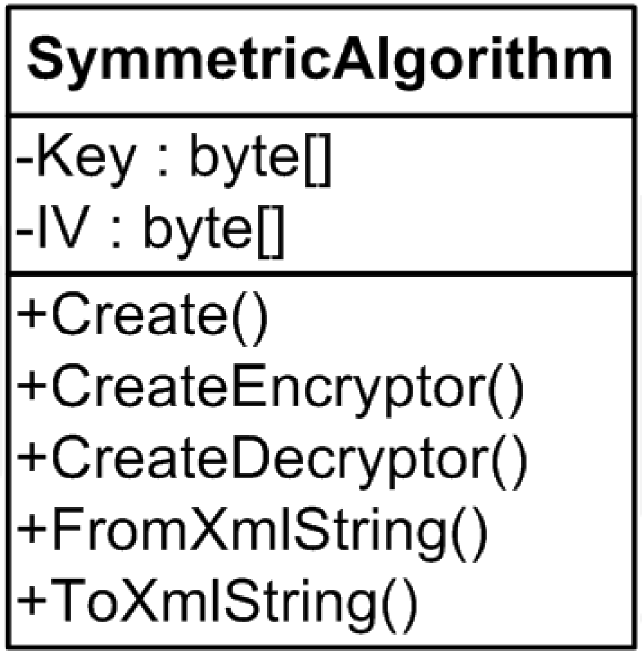
Problems with transparent keys

```
SymmetricAlgorithm algo
    = SymmetricAlgorithm.Create();
algo.Key = …                ◄ PROBLEM
algo.IV = …
encryptor = algo.CreateEncryptor();
encryptor.TransformBlock(…);
```


Katholieke Universiteit Leuven
departement computerwetenschappen

Problems with transparent keys

- **Solution: add opaque key support**

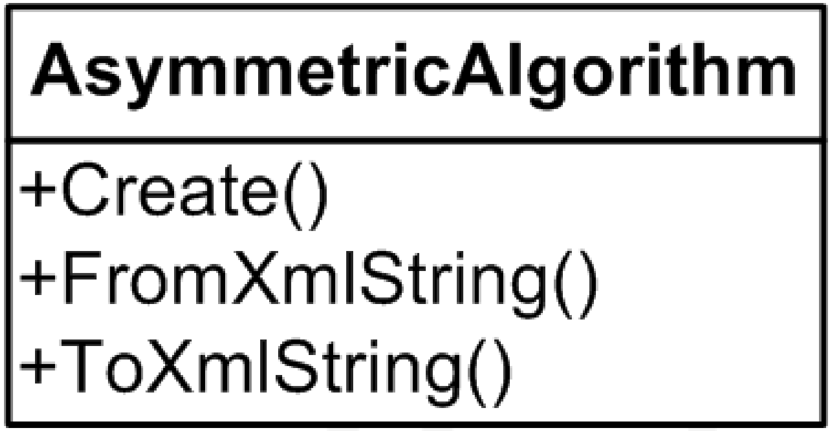


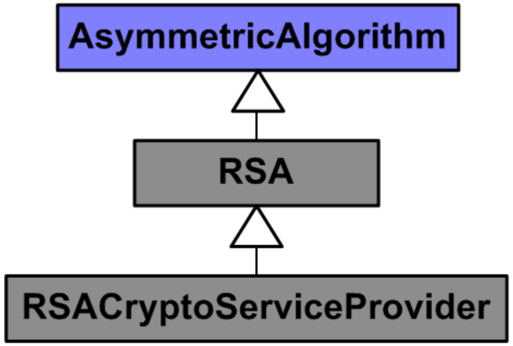
Katholieke Universiteit Leuven
departement computerwetenschappen

Problems with transparent keys

```
SymmetricAlgorithm algo
    = SymmetricAlgorithm.Create();

algo.FromXmlString(…);

encryptor = algo.CreateEncryptor();

encryptor.TransformBlock(…);
```

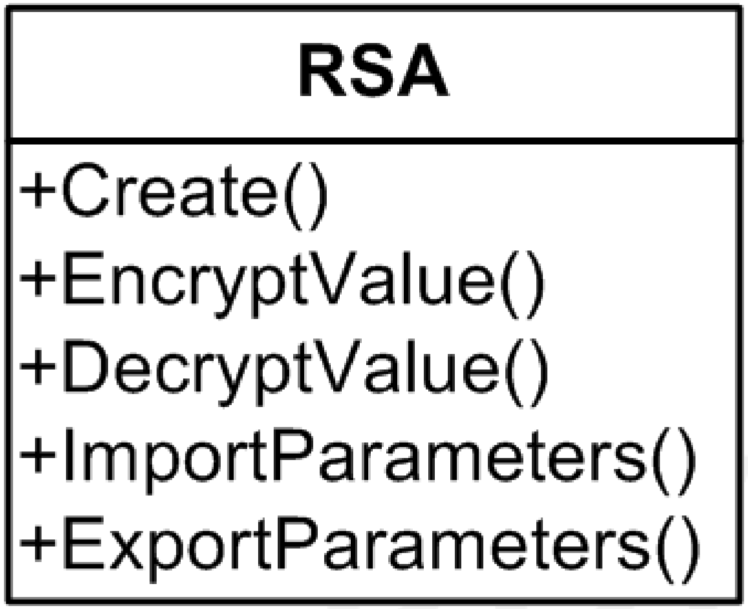
Problems with wrapper code

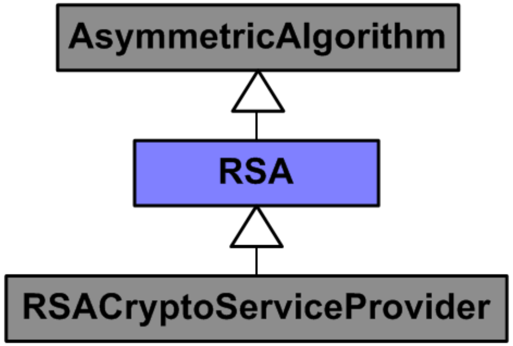




Katholieke
Universiteit
Leuven
departement computerwetenschappen

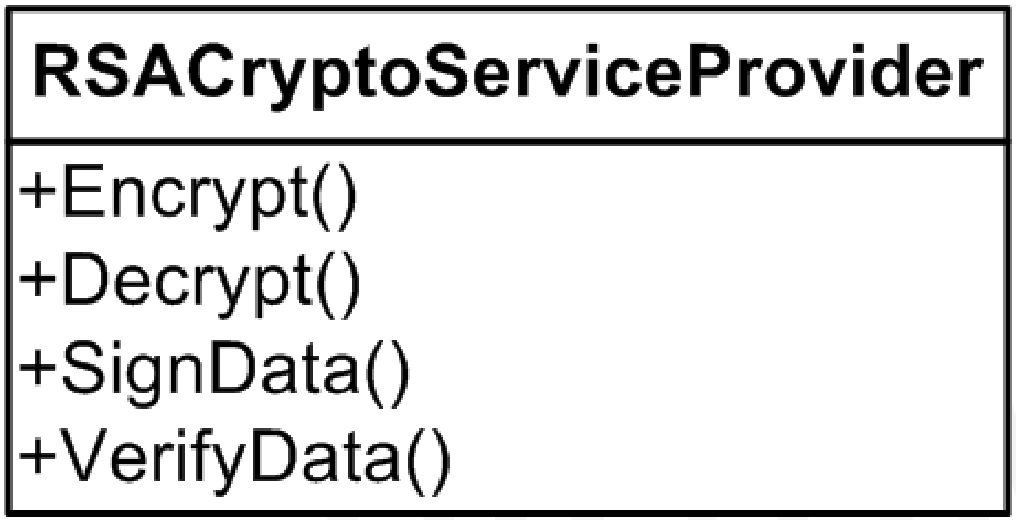
Problems with wrapper code

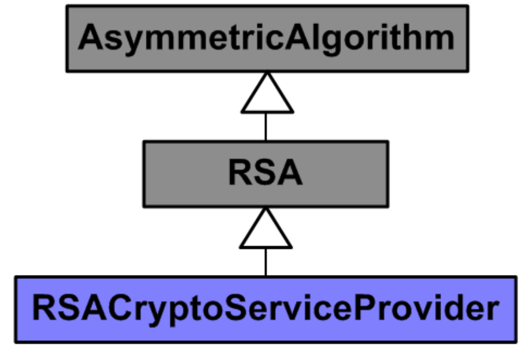




Katholieke Universiteit Leuven
departement computerwetenschappen

Problems with wrapper code

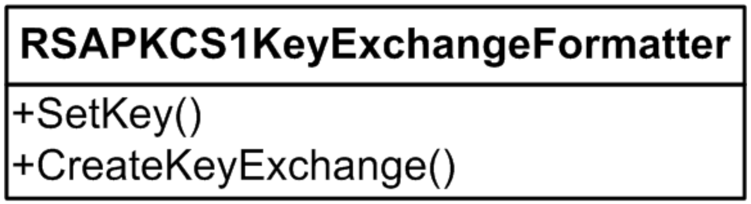


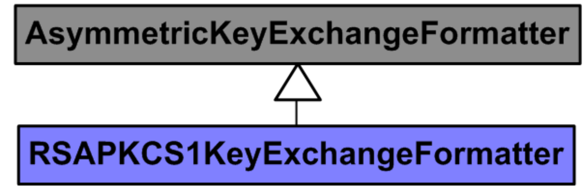


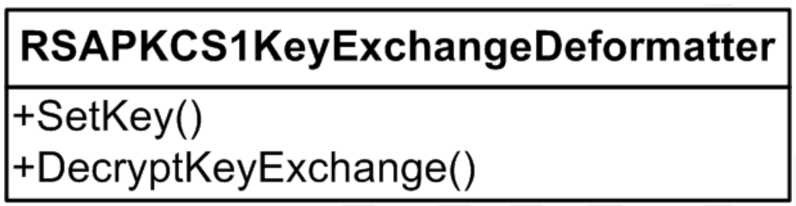
Katholieke Universiteit Leuven
departement computerwetenschappen

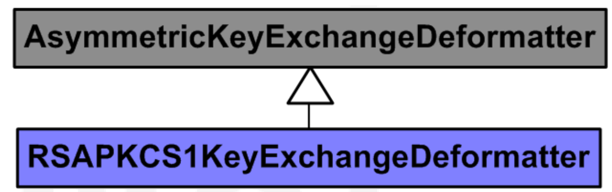
Problems with wrapper code

Katholieke Universiteit Leuven
departement computerwetenschappen









- **RSAOAEPKeyExchangeFormatter/Deformatter**
- **RSAPKCS1SignatureFormatter/Deformatter**
- **DSASignatureFormatter/Deformatter**

Problems with wrapper code

```
public byte[] CreateKeyExchange(...) {

if (rsaKey is RSACryptoServiceProvider) {
  return
   ((RSACryptoServiceProvider)rsaKey).Encrypt(...);
} else {
  <perform padding>
  return rsaKey.EncryptValue(<padded bytes>);
}

}
```

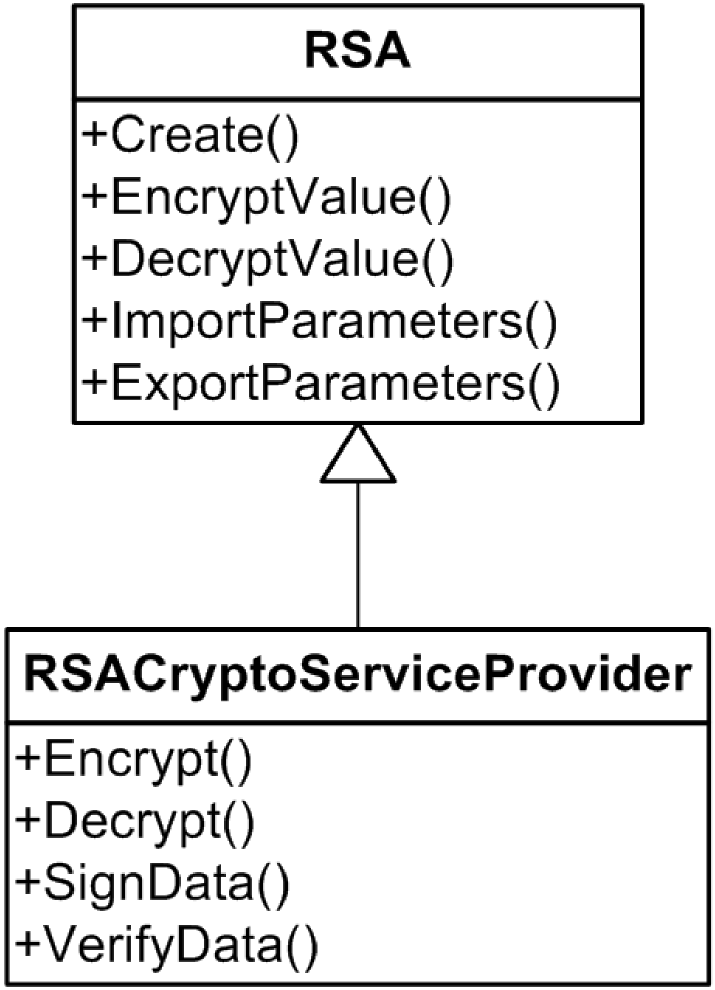

Katholieke Universiteit Leuven
departement computerwetenschappen

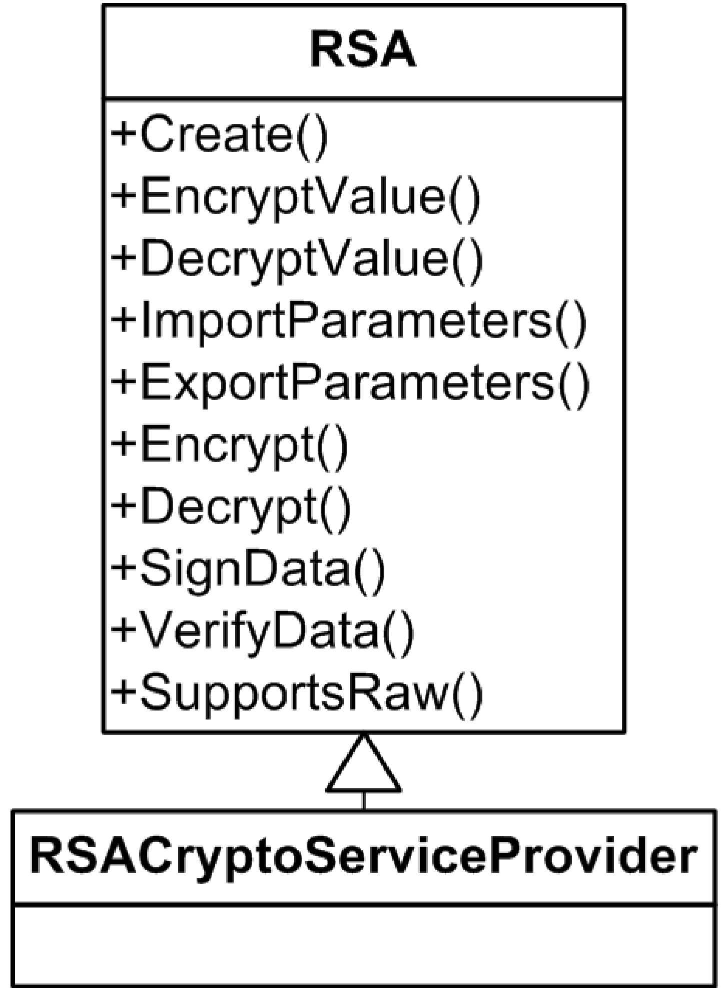
Problems with wrapper code

- **Problem: only RSACryptoServiceProvider gets a ‘special’ treatment**
  - Custom RSA implementations ***must*** support raw RSA

Problems with wrapper code

- **Solution:**





Problems with wrapper code

```
public byte[] CreateKeyExchange(...) {

if ( ! rsaKey.SupportsRaw) {
  return rsaKey.Encrypt(...);
} else {
  <perform padding>
  return rsaKey.EncryptValue(<padded bytes>);
}

}
```

Summary

- Extensible class hierarchy
- Cryptographic configuration support
- Some small issues
  - Can be resolved with some minor tweaks

Katholieke Universiteit Leuven
departement computerwetenschappen

Overview

- Design Principles
- The native Windows CryptoAPI
  - Cryptography API: Next Generation (CNG)
- The Java Cryptography Architecture and Extensions (JCA/JCE)
- The .NET Cryptographic Library
- The OpenBSD Cryptographic Framework
- Key management issues
- Conclusion

Katholieke Universiteit Leuven
departement computerwetenschappen

OpenBSD

- **NetBSD spin-off**
- **Focues on security**
  - Cryptography is the cornerstone of the system
  - Defensive programming
    - Periodically go through source

Katholieke Universiteit Leuven
departement computerwetenschappen

The OpenBSD Crypto Framework

- **OpenBSD Cryptographic Framework (OCF)**
  - “Asynchronous service virtualization layer”
  - Resides in kernel
  - Offers uniform access to crypto hardware
  - Used by
    - Producers (crypto hardware)
    - Consumers (other kernel modules)

Katholieke Universiteit Leuven
departement computerwetenschappen

The OpenBSD Crypto Framework

- **Two modi operandi**
  - Session-based
    - Symmetric crypto, hasing
    - Session caching features
  - Individual operations
    - Asymmetric crypto

Katholieke Universiteit Leuven
departement computerwetenschappen

The OpenBSD Crypto Framework

- **Producers**
  - Are drivers
  - Registers with OCF
    - Supported algorithms
    - Other capabilities (chaining, RNG, …)
  - One pseudo-driver
    - Software crypto

Katholieke Universiteit Leuven
departement computerwetenschappen

The OpenBSD Crypto Framework

- **Consumers**
  - Other modules in kernel (e.g. IPSec)
  - Send asynchronous requests to the OCF
    - Get notified when the work is complete
    - Synchronous requests not supported

Katholieke Universiteit Leuven
departement computerwetenschappen

The OpenBSD Crypto Framework

- **A consumer doesn’t know which producer it’s talking to**
  - The OCF takes care of this automatically
  - Enables load-balancing
  - Enables session-migration
    - When hardware is added/removed (i.e. PCMCIA card)
    - On-demand
  - Important difference between OCF and Java/.NET/CryptoAPI

Katholieke Universiteit Leuven
departement computerwetenschappen

The OpenBSD Crypto Framework

- **This is a kernel framework**
- **User-level support is added through the /dev/crypto interface**
  - Synchronous!
  - Based on ioctl() calls
  - Not very user friendly
  - Frameworks like OpenSSL offer abstractions over /dev/crypto

Katholieke Universiteit Leuven
departement computerwetenschappen

Summary

- Extensible (through device drivers)
- Crypto configuration is done by the framework behind the scenes
  - Applications do not see the different ‘CSPs’

Katholieke Universiteit Leuven
departement computerwetenschappen

Overview

- Design Principles
- The native Windows CryptoAPI
  - Cryptography API: Next Generation (CNG)
- The Java Cryptography Architecture and Extensions (JCA/JCE)
- The .NET Cryptographic Library
- The OpenBSD Cryptographic Framework
- Key management issues
- Conclusion

Katholieke Universiteit Leuven
departement computerwetenschappen

Key management issues

- Generating keys
- Key length
- Storing keys
- Key establishment
- Key renewal
- Key disposal

Katholieke Universiteit Leuven
departement computerwetenschappen

Generating keys

- **Algorithm security = key secrecy**
- **Key should be hard or impossible to guess**
  - Human password → dictionary attack!
  - Better: hash of entire pass-phrase
  - Machine-generated → use cryptographically secure pseudo-random generator

Katholieke Universiteit Leuven
departement computerwetenschappen

Key length

- **Trade-off: information value ↔ cracking cost**
- **Symmetric algorithms**
  - $1 000 000 investment in VLSI-implementation

56 bits	64 bits	128 bits
1 hour	10 days	10^{17} years

- **RSA**

Year	*vs. Individual*	*vs. Corporation*	*vs. Government*
2000	1024	1280	1536
2005	1280	1536	2048
2010	1280	1536	2048

Katholieke Universiteit Leuven
departement computerwetenschappen

Storing keys

- **Simplest: human memory**
  - Remember key itself
  - Key generated from pass-phrase
- **Use Operating System access control**
- **Key embedded in chip on smart card**
- **Storage in encrypted form**
  - *Key encryption* keys ↔ *data encryption* keys
- **Limit key lifetime depending on**
  - Value of the data
  - Amount of encrypted data

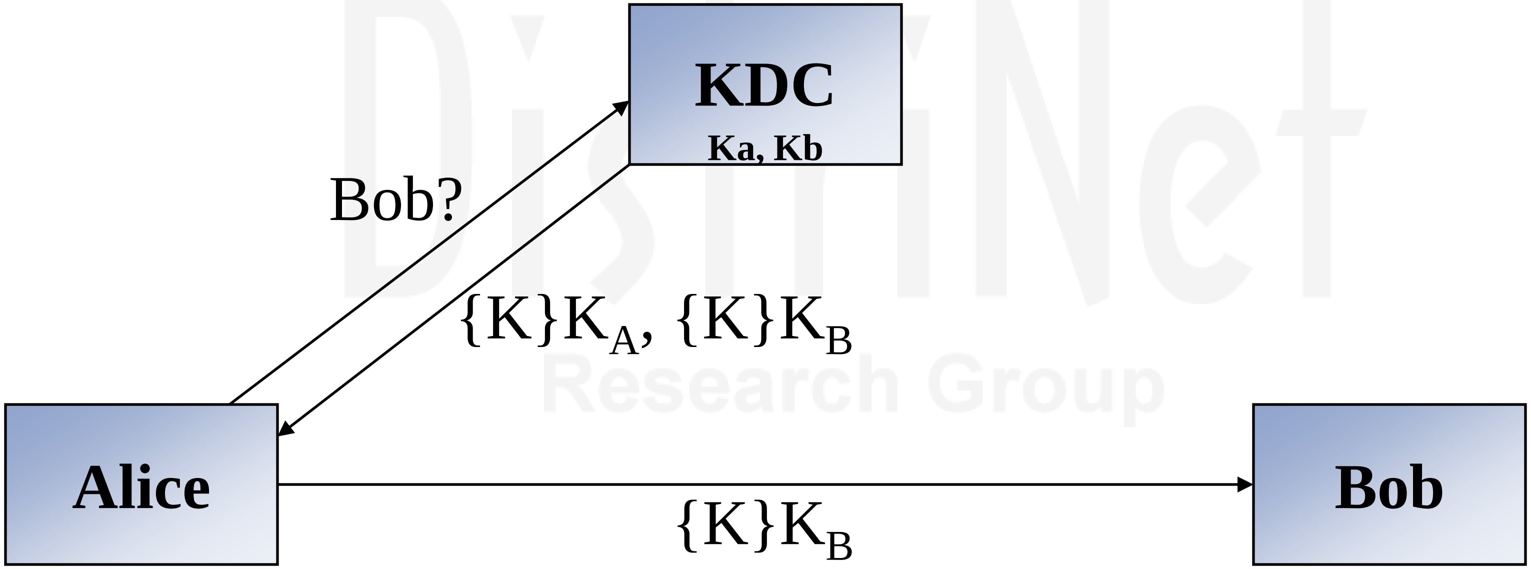
Key establishment

- **Key agreement = Two parties compute a secret key together**
  - E.g. Diffie – Hellman protocol
- **Key distribution or transport = One party generates a key and distributes it in a secure way to all authorized parties**

Katholieke Universiteit Leuven
departement computerwetenschappen

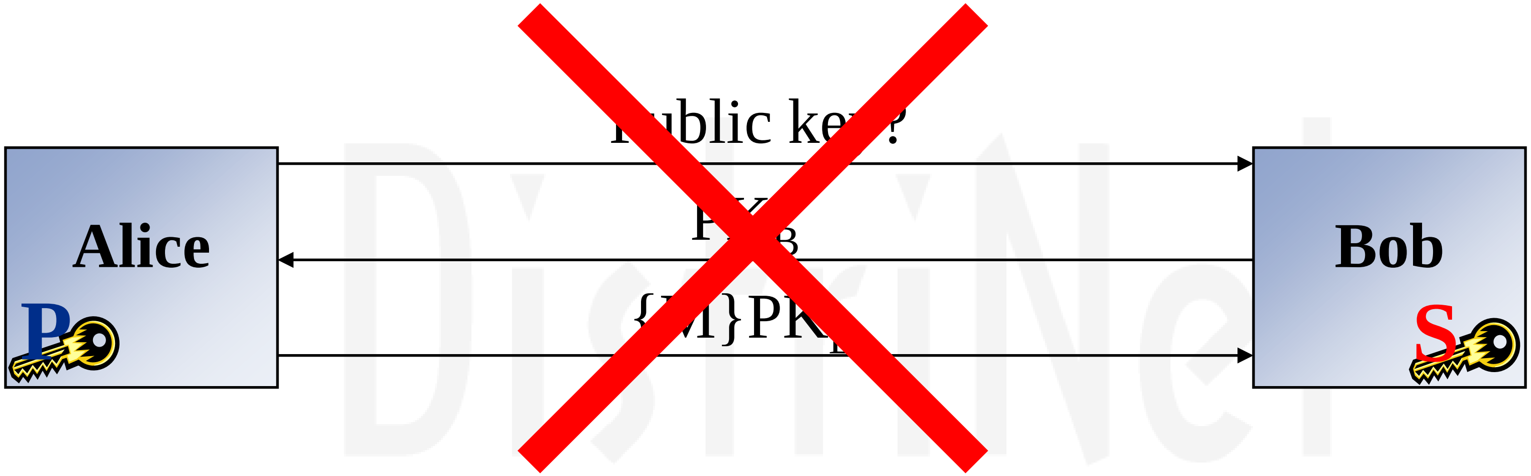
Key distribution

- **Using symmetric encryption**
  - Trusted party: Key Distribution Center (KDC)
  - General idea (oversimplified:)



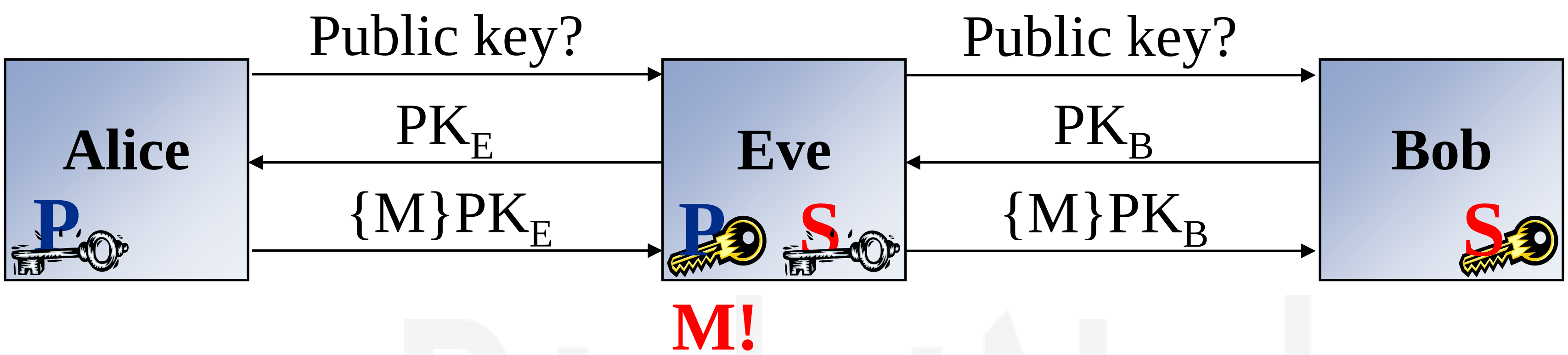
Key distribution

- **Using public-key encryption**
  - No need for KDC?



– Man-in-the-middle attack!

Katholieke Universiteit Leuven
departement computerwetenschappen



- **How can Alice be sure she got Bob’s public key?**
  - Solution: Certificates

    Public Key Infrastructure (PKI)
  - Discussed later

Katholieke Universiteit Leuven
departement computerwetenschappen

Key renewal

- **Best practice:**
  - Limit the amount of data encrypted with a single key
  - Limit the amount of time a key is in use
- **Hence:**
  - Need for mechanisms to renew keys

Key disposal

- **Once a key is no longer used, what should happen?**
  - Short-term keys:
    - Dispose in a secure way
  - Long-term keys:
    - Encryption:
      - Re-encrypt old data, or store key securely
    - Signing
      - Signing key should be disposed of securely
      - Verification key should be stored securely

Summary

- **Good key management is essential to achieve any security from cryptography**
- **Inappropriate**
  - Key generation
  - Key storage
  - Or key establishment

is often the cause of security breaches

Katholieke Universiteit Leuven
departement computerwetenschappen

Overview

- Design Principles
- The native Windows CryptoAPI
  - Cryptography API: Next Generation (CNG)
- The Java Cryptography Architecture and Extensions (JCA/JCE)
- The .NET Cryptographic Library
- The OpenBSD Cryptographic Framework
- Key management issues
- Conclusion

Katholieke Universiteit Leuven
departement computerwetenschappen

Conclusion

- **Cryptographic primitives offer well-defined but complex security guarantees**
  - Precisely saying what security a crypto primitive offers is non-trivial
- **As a consequence, cryptographic primitives are hard to use correctly**
  - Mainstream developers should typically **not** use them
  - Use API to higher-level protocols instead

Katholieke Universiteit Leuven
departement computerwetenschappen