# Content

- Introduction
- Signing, sealing and guarding Java object
  - Motivation
  - Design ( in terms of API )
  - Performance
- Summary

# Introduction

- Java Security Package
  - Police-based
  - Configurable
  - Extensible
  - Fine-grained access control

# Introduction

- Object Orientation
  - Data encapsulation
  - Object name space partition
  - Type safety

# Introduction

- Distributed Java Application
  - Java remote method Invocation package
  - Convenient and necessary to protect the state of an object for integrity and confidentiality

# Introduction

- Java.security.SignedObject and java.security.GuardedObject are part of JDK1.2
- Javax.crypto.SealedObject is included in JCE1.2

# Signing Java Object

- Motivation
  - Authorization token
  - Valid authentication across machines (JVMs)
  - Provide authenticity of the state of an object
  - Nested SignedObject
  - Provide confideniality

# Signing Java Object

- Design
  - SignedObject contains the signed object, must be serializable, and its signature
  - Signing algorithm
    - DSA
    - SHA-1

SignedObject and SealedObject

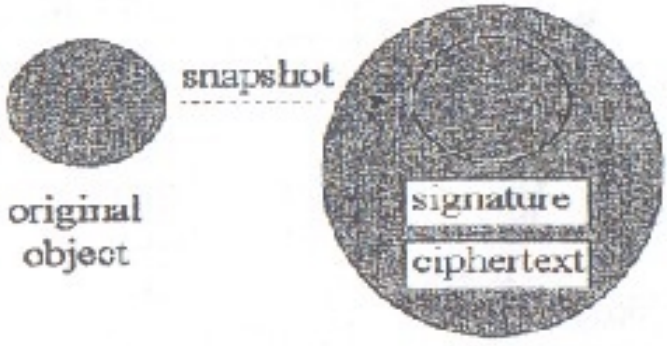


**Fig. 1.** Signed and Sealed Objects

# Signing Java Object

- API Design

```
public SignedObject(Serializable object, PrivateKey signingKey,
    Signature signingEngine)
public final void sign(PrivateKey signingkey, Signature
    signingEngine)
public final Object getContent()
public final byte[] getSignature();
public final String getAlgorithm();
public final boolean verify(PublicKey verificationKey, Signature
    verificationEngine);
```

# Signing Java Object

- Example - Signing an object

```
Signature signingEngine =
    Signature.getInstance(algorithm, provider)
SignedObject so = new SignedObject(myobject,
  privatekey, signingEngine)
```

# Signing Java Object

- Example - Verification

```
Signature verificationEngine =
    Signature.getInstance(algorithm, provider)
If(so.verify(publickey, verificationEngine))
   try {
     Object myobj = so.getContent();
   } catch (ClassNotFoundException e) {};
```

# Signing Java Object

| object size | serialization | signing | verification |
|---|---|---|---|
| | | 512-bit SHA-1/DSA | |
| 10 bytes | 0ms | 25ms | 43ms |
| 100 bytes | 0ms | 26ms | 44ms |
| 10K bytes | 1ms | 134ms | 153ms |
| 100K bytes | 9ms | 1119ms | 1138ms |

**Table 1.** Performance of SignedObject (09/05/97)

# Signing Java Object

| object size | serialization | signing | verification |
|---|---|---|---|
| | | 1024-bit SHA-1/DSA | |
| 10 bytes | 0ms | 80ms | 151ms |
| 100 bytes | 0ms | 83ms | 157ms |
| 10K bytes | 1ms | 189ms | 260ms |
| 100K bytes | 9ms | 1168ms | 1237ms |

**Table 2.** Performance of SignedObject (09/05/97)

# Sealing Java Object

- Motivation
  - Protect its confidentiality with cryptographic algorithm (e.g. DES)
  - Provide integrity to object

# Sealing Java Object

- API Design

Public SealedObject(Serializable object, Cipher c);

Public final Object getContent(Cipher c);

# Sealing Java Object

- Example - generate a DES cipher

```
KeyGenerator keyGen =
   KeyGenerator.getInstance("DES");
SecretKey desKey = KeyGen.generateKey();
Cipher cipher = Cipher.getInstance("Des");
Cipher.init(Cipher.ENCRYPT_MODE, desKey);
```

# Sealing Java Object

- Example - create a SealedObject

```
String s = new String("Greetings");
SealedObject so = new SealedObject(s, cipher);
```

# Sealing Java Object

- Example - decrypt the SealedObject

```
Cipher.init(Cipher.DECRYPT_MODE, desKey);
Try {
   String s = (String) so.getContent(cipher);
} catch(ClassNotFoundException e) {}
```

# Sealing Java Object

- Performance
  - Similar to SignedObject.
  - Depends on the serialization time and the speed of the underlaying cryptographic algorithm.

# Guarding Java Object

- Motivation
  - Security check done in the consumer side
  - Don’t know what information to provide
  - Performance (e.g. faster access)
  - Consumer environment too security sensitive
  - Too much information
  - Guaranteed to occur in a context where the protection mechanism would allow it
  - Simplify server program

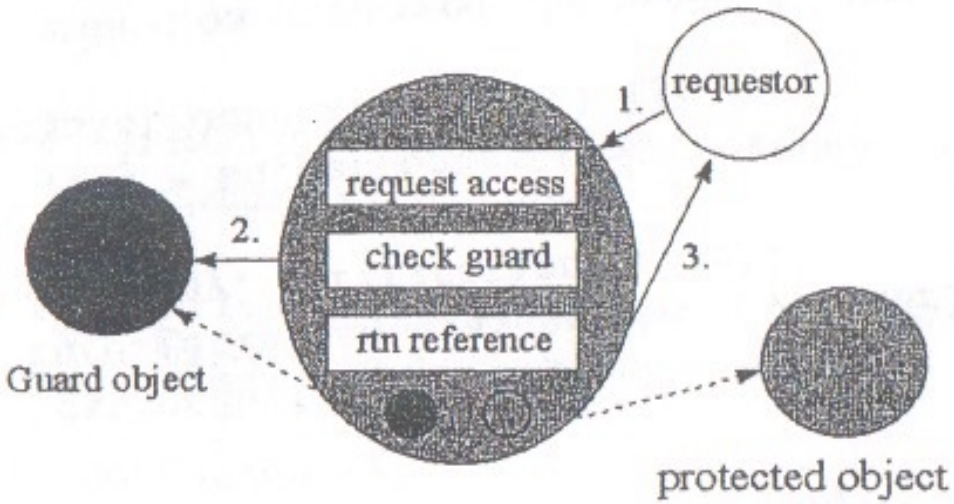


**Fig. 2.** Guard and GuardedObject

# Sealing Java Object

- API Design

Public abstract void checkGuard(Object object)

Public GuardedObject(Object object, Guard guard);

Public Object getObject();

# Sealing Java Object

- Example

```
FileInputStream fis = new
  FileInputStream("/a/b/c");
FilePermission p = new FilePermission("/a/b/c",
  "read");
GuardedObject g = new GuardedObject(fis, p);

FileInputStream fis = (FileInputStream)
  g.getObject();
```

# Related Work

- Modula-3 and Oblique is related to SignedObject and SealedObject.
- Gated Object model and Guard concept in programming language research is similar to the GuardedObject

# Summary

- Enrich the existing Java security APIs, so security aware applications can be much easier to build.
- Performance is satisfy for commercial use.

# Question