Cryptography and Java
Cryptography and Java

Java provides cryptographic functionality using two APIs:

- **JCA** – Java Cryptography Architecture
  - *security framework integrated with the core Java API*

- **JCE** – Java Cryptography Extension
  - *Extensions for strong encryption (exported after 2000 US export policy)*
Basic Architecture

- Provider based architecture

```
  Application

  JCA
  java.security
  provider A
  provider B
  provider Z

  JCE
  javax.crypto
  provider A
  provider B
  ...... provider Z
```
Design principles

- Algorithm independence
  - specification of engine classes
- Algorithm extensibility
  - easy updatation of engine classes with new algorithms
- Implementation independence
  - use of cryptographic service providers
- Implementation interoperability
  - providers working with each other
Engine classes

- Cryptographic operations are classified into classes in JCA/JCE. This classes are called as engines.
  - JCA engines
  - JCE engines
JCA engines

- MessageDigest (produces hash value)
- Signature (produces digital signature)
  - KeyPairGenerator (produces pair of keys)
- KeyFactory (breaks down a key)
- KeyStore (manages and stores keys)
- SecureRandom (produces random numbers)
  - AlgorithmParameters (encoding and decoding)
  - AlgorithmParameterGenerator (generates parameters)
- CertificateFactory (public key cert, revocation)
- CertPathBuilder (establish relationship chains between certs)
- CertStore (stores certificates and revocation lists)
JCE engines

- Cipher (*encryption/decryption*)
- KeyGenerator (*produces secret keys used by ciphers*)
- SecretKeyFactory (*operates on SecretKey objects*)
- KeyAgreement (*key agreement protocol*)
- Mac (*message authentication code functionality*)
Location

- JCA engines are located in java.security package
- JCE engines are located in javax.crypto package
Example 1: Generate a DES/AES key and use cipher to encrypt a message.

```java
byte[] message = "I am a superman, sshhh don't tell anyone".getBytes();

KeyGenerator keygenerator = KeyGenerator.getInstance("DES");
SecretKey desKey = keygenerator.generateKey();

Cipher desCipher = Cipher.getInstance("DES/ECB/PKCS5Padding");
// Initialize the cipher for encryption
desCipher.init(Cipher.ENCRYPT_MODE, desKey);

// Encrypt message and return
byte[] encryptedMessage = desCipher.doFinal(message);
```
• Example 2: Generate random bytes using SecureRandom.

```java
import java.security.SecureRandom;

public class Main {
    public static void main(String[] argv) throws Exception {
        SecureRandom secRandom = SecureRandom.getInstance("SHA1PRNG");
        secRandom.setSeed(711);
        byte[] bytes = new byte[20];
        secRandom.nextBytes(bytes);
    }
}
```
Your Best Friend

• Look up API docs for the relevant packages
  - java.security
  - javax.crypto

• JCA reference guide
  - http://java.sun.com/j2se/1.5.0/docs/guide/security/CryptoSpec.html
Algorithm extensibility - example

MessageDigest ultrafastImplementation =

MessageDigest.getInstance("UltraFastHash");
Implementation independence

- Offers the developer a choice of how to handle the presence of providers

MessageDigest Dev1Md5Implementation =

MessageDigest.getInstance("MD5", "Provider1");
Implementation interoperability

- Providers are interoperable
- The developer might use provider A to generate a key pair, passing that key pair along to provider B's signature algorithm
Adding providers

There are two ways

- Adding statically
- Adding dynamically

Static addition:

- Copy the JCE provider JAR file to \texttt{java-home/jre/lib/ext/}
- Stop the Application Server
- Edit the \texttt{java-home/jre/lib/security/java.security} properties file in any text editor. Add the JCE provider you’ve just downloaded to this file.
  
  \texttt{security.provider.n=provider-class-name}
Adding providers

Static addition (cont..)

security.provider.2 = org.bouncycastle.jce.provider.BouncyCastleProvider

- Dynamic addition

// create a provider object
Provider bountyProvider = new org.bouncycastle.jce.provider.BouncyCastleProvider();
// Add the bouncycastle Provider to the current list of
// providers available on the system.
Security.addProvider (bountyProvider);