Holub Associates
www.holub.com

The Java Cryptography APIs

The Crypto APIs and related Tools

Allen I. Holub
www.holub.com

© 2009, Allen I. Holub <http://www.holub.com>

Updates

- Get the most recent version of these slides from

  http://www.holub.com/publications/notes_and_slides/index.html

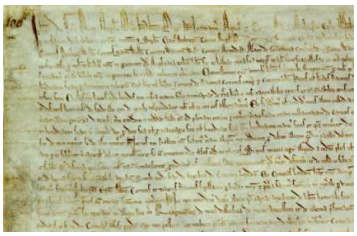
© 2009, Allen I. Holub <http://www.holub.com>

Caveat

- Cryptographers like to obscure stuff.
  - You shouldn't let cryptographers write documentation.
  - You shouldn't let cryptographers design APIs.
- The JAVA security docmentation and APIs were written by cryptographers.

© 2009, Allen I. Holub <http://www.holub.com>

Establishing Policies



© 2009, Allen I. Holub <http://www.holub.com>

Security Managers

- Objects ask the security manger for permission to do sensitive operations.
- Can subvert by replacing *rt.jar* with a version that doesn't ask the questions.
- Default version gets permissions from *$JAVA_HOME/jre/lib/security/java.policy*
  - *Client-side file. Maintenance is a headache.*
- You can also write your own from scratch to support centralized permissions, etc.
- To run under a security manager, use:
  java –Djava.security.manager

© 2009, Allen I. Holub <http://www.holub.com>

Writing your own Security Manager (1)

```
Public class PwdSecurityMgr
               extends SecurityManager
{
  private String password;

  public PwdSecurityMgr(String password)
  {   this.password = password;
  }
```

© 2009, Allen I. Holub <http://www.holub.com>

Writing your own Security Manager (2)

```
private void accessOk() throws SecurityException
{ try
  {  BufferedReader in = new BufferedReader(
          new InputStreamReader(System.in));
     System.out.print("Password: ");
     if( in.readLine().equals( password );
          return;
  }
  catch( IOException e )( /*... */ }
  throw new SecurityException("Bad Password");
}
```

© 2009, Allen I. Holub <http://www.holub.com>

Writing your own Security Manager (3)

```
  public void checkRead(FileDescriptor fd)
  {  accessOkay();  }

  public void checkWrite(FileDescriptor fd)
  {  accessOkay();  }

  public void checkDelete(FileDescriptor fd)
  {  accessOkay();  }
  }

  /* Many more checkXXX overrides go here */
}
```

© 2009, Allen I. Holub <http://www.holub.com>

Writing your own Security Manager (4)

- It can get a lot more complicated.
  - Can check that requesting class is indeed in correct package.
  - Can check that class was loaded by the correct class loader.
  - Can examine the runtime stack for suspicious calling sequence.
    - This is a very error prone, and not very reliable process.

© 2009, Allen I. Holub <http://www.holub.com>

Setting the Security Manager

- `System.setSecurityManager( new PwdSecurityManager());`
- Throws `SecurityException` if the calling code doesn't have permission to change the security manager.
- Only one security manager can be active!

© 2009, Allen I. Holub <http://www.holub.com>

The Security-Properties File

- $JAVA_HOME/jre/lib/secuirty/java.security
- Contains various security-related properties:
  - Security providers (discussed below)
  - Policy-file locations:

    policy.url.1=file:${java.home}/lib/security/java.policy
    policy.url.2=file:${user.home}/.java.policy
    - user.home is your home directory
    - you can add additional files to this list (policy.url.3 ...)

© 2009, Allen I. Holub <http://www.holub.com>

The Policy File

- Grants permissions for specified operations.
- http://java.sun.com/j2se/1.5.0/guide/security/PolicyFiles.html
- $JAVA_HOME/jre/lib/security/java.policy
  - Individual user's can also have files: */userHomeDirectory/*.java.policy
- Or specify custom file with:

  java -Djava.security.manager \
  -Djava.security.policy=*someURL*
- Can use JDK's **policytool** to get GUI, but not worth the trouble

© 2009, Allen I. Holub <http://www.holub.com>

Specifying contents with java -D

- Use *java –Dxxx* for macros

  On command line:

  java –Djava.ext.dirs=dir1;dir2

  In policy file:

```
grant codebase
      "file:${{java.ext.dirs}}/*"
```

© 2009, Allen I. Holub <http://www.holub.com>

The keystore directive

- Keystores are created by **keytool** utility.
  - Hold certificates, encryption keys, etc.
- Security manager needs it to verify "signed" code.
- Only one permitted in policy file, as follows:

```
keystore "path/.keystore"
```

  - path is relative to policy-file location.

    Given:

    policy.url.1=http://foo.bar.com/policies/some.policy

    Keystore is at:

    http://foo.bar.com/policies/path/.keystore
- Can also expose keystore password via policy file:

```
keystorePasswordURL "path/.password"
```

© 2009, Allen I. Holub <http://www.holub.com>

The "grant" statement

```
grant [signedby signer] [, codebase code-source>]
      [, principal principalClassName "principalName"]
{ permission class-name [name, [action-list]]
     [signedBy "signer-names"]
  ...
}
```

For example:

```
grant signedby "Holub"  codebase
          "http://www.holub.com/jars/Holub.jar"
{ permission FilePermission
          "c:/tmp/holub.tmp",  "read,write"
}
```

© 2009, Allen I. Holub <http://www.holub.com>

signedby

- Permission granted only if file signed by certificate holder.

Keystore "alias" for a certificate.

```
grant signedby "AHolub" ... {
  //...
}
```

Multiple aliases (comma separated) are okay.

```
grant signedby "Aholub,HolubAssociates" ... {
  //...
}
```

© 2009, Allen I. Holub <http://www.holub.com>

codeBase

- Authorizes all .class and .jar files in a specific directory
- If missing, current machine is assumed.

All .class files (not jars) in this directory

All .class and .jar files in this directory

```
grant codeBase "file:/home/"
grant codeBase "file:/home/sysadmin/*" ...
grant codeBase "http://www.holub.com/javabin/-"
```

All .class and .jar files in directory and, recursively, all subdirectories

© 2009, Allen I. Holub <http://www.holub.com>

principal (Authorization)

- Specifies a specific person who is authorized to execute this code.
- If missing, everyone can execute code.

Search keystore for a certificate associated with the alias "allen"

```
grant principal "AliasForAllen"
grant principal
  javax.security.auth.x500.x500Principal
  "cn=Allen" ..
```

Designated name in cert

© 2009, Allen I. Holub <http://www.holub.com>

Permissions

```
grant codeBase "file:${{java.ext.dirs}}/*" {
  permission java.security.AllPermission;  };
grant{
  permission java.lang.RuntimePermission
                                  "stopThread"
  permission java.net.SocketPermission
          "localhost:1024-", "listen"
  permission java.io.FilePermission
          "c:/tmp/*", "read,write"
  //...
}
```

These are java class names..

Constructor arguments. See javadoc for associated class.

© 2009, Allen I. Holub <http://www.holub.com>

Standard Permissions (1)

http://java.sun.com/j2se/1.5.0/docs/guide/security/permissions.html

Permission class	Name	Action
FilePermission	file or directory name (use * for all, - for subdirs), or **<<ALLFILES>>**	read, write, execute.
SocketPermission	domain name	accept, listen, connect
RuntimePermission	property name, e.g. java.version	read write
AWTPermission	facility: e.g.: topLevelWindow	n/a
AllPermission	n/a	n/a

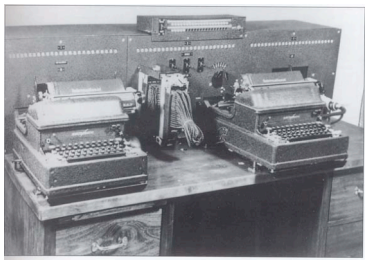
© 2009, Allen I. Holub <http://www.holub.com>

Standard Permissions (2)

Permission class	Name	Action
NetPermission	Authenticator.setDefault, Authenticator.requestPassword Authentication, ...	n/a
SecurityPermission	getPolicy, setPolicy, ...	n/a
SerializablePermission	enableSubstitution (enables replace/resolve)	n/a
ReflectPermission	access (enables core reflection)	n/a

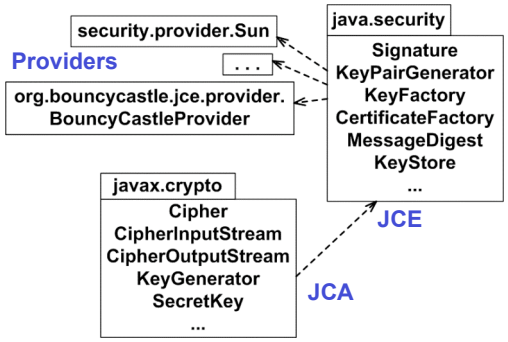
© 2009, Allen I. Holub <http://www.holub.com>

Crypto



© 2009, Allen I. Holub <http://www.holub.com>

Java Crypo Extensions & Architecture



© 2009, Allen I. Holub <http://www.holub.com>

Providers

- Implement Algorithms and do other low-level things.
- Sun ships a basic provider with the JDK
- Better providers are available:
  - http://www.bouncycastle.org
  - http://www.cryptix.org
  - http://java.sun.com/products/jce/jce122_providers.html

© 2009, Allen I. Holub <http://www.holub.com>

Abstract Factory & Chain of Responsibility

```
abstract class ServiceProvider
{   static public
    ServiceProvider getInstance(String algorithm)
    { for( AProvider p: AllProviders )
          if( p.supports(algorithm) )
               return p.create(algorithm);
    }

    abstract protected
        ServiceProvider create(String algorithm);

    abstract public void f();
    abstract public void g();
    //...
}
```

© 2009, Allen I. Holub <http://www.holub.com>

Static Provider Registration

- In $JAVA_HOME/jre/lib/security/java.security

security.provider.1=sun.security.provider.Sun
security.provider.2=sun.security.rsa.SunRsaSign
security.provider.3=com.sun.net.ssl.internal.ssl.Provider
security.provider.4=com.sun.crypto.provider.SunJCE
security.provider.5=sun.security.jgss.SunProvider
security.provider.6=org.bouncycastle.jce.provider.
BouncyCastleProvider
security.provider.7=com.sun.security.sasl.Provider

© 2009, Allen I. Holub <http://www.holub.com>

Static Provider Registration (2)

- Providers are searched in order.
  - "Can you provide algorithm X?"
  - First one that responds "yes" is used.
- Use positions > 1
  - This was mandated by a bug, which may or may not have been fixed in Java 5.
  - Your provider might have dependencies on Sun providers.
  - Position 5 seems to work pretty well.

© 2009, Allen I. Holub <http://www.holub.com>

Dynamic Provider Registration

- In *$JAVA_HOME/jre/lib/security/java.policy*

```
grant codeBase "file:/mydir/myApp.jar"
{ Permission
      java.security.SecurityPermission
      "insertProvider.BouncyCastle"
}
```

- In the code

```
Provider p =
   new org.bouncycastle.jce.provider.
                        BouncyCastleProvider();
Security.insertProviderAt( p, 5 );
```

© 2009, Allen I. Holub <http://www.holub.com>

Algorithm Names are Arbitrary

- You must request an algorithm by name:

```
Signature sig =

signature.getInstance("SHA1withDSA");
```

- Sun does not specify algorithm names
  - They're invented by the Providers.

© 2009, Allen I. Holub <http://www.holub.com>

Listing the Algorithms, etc.

- `Providers` are `Maps` `// ouput on next slide`

```
for( Provider provider : Security.getProviders() )
{ SortedSet<String> sortedListing =
                              new TreeSet<String>();
  System.out.println( provider.getName() );
  System.out.println("ver. " + provider.getVersion());
  System.out.println(        + provider.getInfo()   );
  for( Map.Entry entry : provider.entrySet() )
  {   String  description = entry.getKey().toString();
      description + (entry.getKey().toString().
                              startsWith("Alg.Alias.")
                  ? " is aliased to "
                  : " = " );
      sortedListing.add( description +
                        entry.getValue().toString() );
   }
   for( String s : sortedListing )
      System.out.println( s );
}
```

© 2009, Allen I. Holub <http://www.holub.com>

Sample Algorithm-List Output

```
SunRsaSign
ver. 1.5
Sun RSA signature provider
**************************************
Alg.Alias.KeyFactory.1... is aliased to RSA
Alg.Alias.Signature.1.... is aliased to SHA256withRSA
KeyFactory.RSA = sun.security.rsa.RSAKeyFactory
KeyPairGenerator.RSA = sun.security.rsa.RSAKeyPairGe...
Provider.id className = sun.security.rsa.SunRsaSign
Provider.id info = Sun RSA signature provider
Provider.id name = SunRsaSign
Provider.id version = 1.5
Signature.MD5withRSA = sun.security.rsa.RSASignatu  ...
Signature.SHA1withRSA = sun.security.rsa.RSASignatu ...
Signature.SHA256withRSA = sun.security.rsa.RSASigna ...
Signature.SHA512withRSA SupportedKeyClasses = java. ...
```

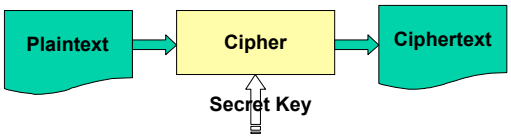
© 2009, Allen I. Holub <http://www.holub.com>

Symmetric Ciphers

© 2009, Allen I. Holub <http://www.holub.com>

Symmetric Algorithms

- The same key (and algorithm) is used for both enciphering and deciphering.
- Algorithms based on:
  - complex shuffling
  - adding pseudo-random numbers (noise) to plaintext.
  - both



© 2009, Allen I. Holub <http://www.holub.com>

Random Numbers

- CSPRNG: Cryptographically Secure Pueudo-Random Number Generators
- Do not use java.util.Random;

```
SecureRandom csprng =
       SecureRandom.
           getInstance("SHA1PRNG");
csprng.setSeed(918345903458);
long n   = csprng.next(64); //64 bit

byte r[] = new byte[4];
csprng.nextBytes(r);
```

© 2009, Allen I. Holub <http://www.holub.com>

Secret Keys

- Opaque
  - an array of bytes
- Transparent
  - lets us find out information about the key
    - how strong is it?
    - how many bits is it?
    - etc.

© 2009, Allen I. Holub <http://www.holub.com>

Create an Opaque Secret Key

Get key for DES algorithm

```
KeyGenerator g =
       KeyGenerator.getInstance("DES");
g.init( 56, SecureRandom.
                getInstance("SHA1PRNG"));
SecretKey key = g.generateKey();
byte[] myKey = key.getEncoded();
```

- You can skip the `init()` call, but then you're leaving itup to the provider.

© 2009, Allen I. Holub <http://www.holub.com>

Converting to a Transparent Key

```
try
{ //... (see previous slide)
  byte[] myKey = key.getEncoded()
  DESKeySpec spec = new DESKeySpec(myKey);
  SecretKeyFactory =
  SecretKeyFactory.getInstance("DES");
  SecretKey key =
  factory.geterateSecret(spec);
}
catch( InvalidKeyException e )
{ // key not legitimate DES key!
}
```

© 2009, Allen I. Holub <http://www.holub.com>

Verifying Untrusted Key

```
try
{ byte[] myKey = userInput();

  if( DESKeySpec.isWeak(myKey, 0)
     throw InvalidKeyException("weak key");

  // proceed as in previous slide.
}
catch( InvalidKeyException e )
{ // key not legitimate DES key!
}
```

© 2009, Allen I. Holub <http://www.holub.com>

Generic Transparent Keys

```
try
{ //... (see previous slide)
  byte[] myKey = ...;
  SecretKeySpec key =
               new DESKeySpec(myKey, "DES");
}
catch( InvalidKeyException e )
{ // key not legitimate DES key!
}
```

- **Viable only for symetric cyphers**
  - **Practically speaking, only DES & DES3**

© 2009, Allen I. Holub <http://www.holub.com>

Padding

- Cipher algorithms work in multi-byte blocks.
  - you can define a 1-byte block, but the algorithm typically works more slowly.
- If plaintext isn't an even multiple of the block size, it must be padded.
- Four algorithms supported:
  - None (if even multiple of block size)
  - PKCS5Padding (Public Key Cryptography Std.)
  - SSL3Padding (reserved, not implemented)
  - OAEPWith*Digest*And*Mask*

© 2009, Allen I. Holub <http://www.holub.com>

PKCS5 Padding

- Pad out with bytes indicated the length of the padding:

```
0x01
0x0202
0x030303
...
0x0808080808080808
```

- Last byte in block always tells you the padding size.
- Must add a complete block of padding if no padding is required!

© 2009, Allen I. Holub <http://www.holub.com>

What are Cipher Modes

- Repeating patterns are bad.
  - Eases the code-breaking process.
- If the plaintext contains repeating patterns, the ciphertext will also have repeating patterns.
- Cipher Modes "hide" the repeating patterns.
  - Algorithms are fast, and easy to undo.
  - Not encryption!

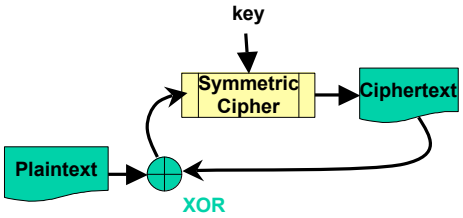
© 2009, Allen I. Holub <http://www.holub.com>

Cipher Modes

- ECB (no transformation is made)
  - Electronic Code Book Mode
  - Doesn't hide patterns
- CBC
  - Cipher Block Chaining
  - Doesn't work for stream ciphers (block size of 1).
- CFB
  - Variant on CBC for stream ciphers

© 2009, Allen I. Holub <http://www.holub.com>

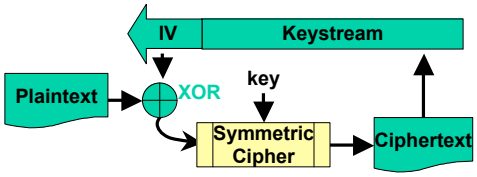
CBC



- Use *Initialization Vector* (IV) instead of ciphertext for first block of plaintext.
- Must use different IV for *every* message.

© 2009, Allen I. Holub <http://www.holub.com>

CFB



- Leftmost 8 bits of ciphertext block shifted into keystream
- Works well with stream ciphers (8-bit block)

© 2009, Allen I. Holub <http://www.holub.com>

The Cipher Engine

```
Cipher desCipher = Cipher.getInstance("DES")
```

- Leaves padding and cipher mode up to provider.
  - not a great idea.

```
Cipher desCipher = Cipher.getInstance(
                    "DES/CBC/PKCS5Padding");
```

- Much better
  - Ambiguity is dangerous

© 2009, Allen I. Holub <http://www.holub.com>

Create a Cipher

```
KeyGenerator g = KeyGenerator("DES");
SecretKey key = g.generateKey();
byte[] desKey = key.getEncoded();
SecretKeySpec spec =
               new SecretKeySpec(desKey,"DES");
try
{  Cipher cipher = Cipher.getInstance(
                    "DES/CBC/PKCS5Padding");
   cipher.init(Cipher.ENCRYPT_MODE, spec);
   //               Cipher.DECRYPT_MODE
}
catch( NoSuchAlgorithmException e )
{   // algorithm/mode/padding not available.
}
```

© 2009, Allen I. Holub <http://www.holub.com>

Encipher/Decipher the Data

- If you have all the data

```
String plaintext = ...;
byte[] cipherText =
   cipher.doFinal( plaintext.getBytes("UTF-8");
```

- If data size is unknown

```
byte[] chunkOfPlaintext;
while( chunkOfPlaintext = doRead() )
     cipher.update( chunkOfPlaintext );
byte[] cipherText = cipher.doFinal();
```

© 2009, Allen I. Holub <http://www.holub.com>

A Word about Strings

- Strings remain in memory, even when garbage collected.
  - For sensitive text, use byte arrays, and clear them out.

```
char[] password = new char[20];
for( int i = 0; i < 20; ++i )
{     char c = System.in.read();
      if( c == '\n' )
            break;
      password[i] = (char)c;
}
//...
Arrays.fill( password, '\u0000' );
```

© 2009, Allen I. Holub <http://www.holub.com>

Stream Ciphers

- Strings remain in memory, even when garbage collected.
  - For sensitive text, use byte arrays, and clear them out.

8-bit block size

```
Cipher cipher =
      Cipher.getInstance("DES/CFB8/PKCS5Padding");
InputStream in = new FileInputStream( ... );
in = new CipherInputStream( in, cipher );
//...
while(  ...  in.read(...)  ... )
      ...
in.close();
```

© 2009, Allen I. Holub <http://www.holub.com>

Password Encryption

- Use password instead of key.
- Add "salt" to randomize password.

```
byte[] salt = new byte[]{ (byte)0x3a, (byte)0x44,
                                ... , (byte)0x31 };
int iterations = 1000; // should be >= 1000
char[] password = getPasswordFromUser();
PBEKeySpec spec = new PBEKeySpec(
                        password, salt, iterations );
SecretKeyFactor factory =
    SecretKeyFactory.getInstance("PBEWithMD5AndDES");
SecretKey key = factory.generateSecret( spec );
Cipher    c = Cipher.getInstance("PBEWithMD5AndDES");
c.doFinal( plaintext );
```

© 2009, Allen I. Holub <http://www.holub.com>

There's a better way

- http://www.mindrot.org/projects/jBCrypt/
- `String hashed = BCrypt.hashpw(`
  `password, BCrypt.gensalt());`

```
//...
if( BCrypt.checkpw(candidate, hashed) )
   // candidate matches
```

- `public static boolean checkpw(`
  `String plaintext, String hashed)`

```
{  return hashed.compareTo(
        hashpw(plaintext, hashed)) == 0;
}
```

© 2009, Allen I. Holub <http://www.holub.com>

Asymmetric Ciphers

© 2009, Allen I. Holub <http://www.holub.com>

Key Encoding

- RAW
- PKCS8
  - `java.security.spec.PKCS8EncodedKeySpec`
  - Use for private keys
- ANSI X.509
  - `java.security.spec.X509EncodedKeySpec`
  - Use for public keys
  - Also used for certificates.

© 2009, Allen I. Holub <http://www.holub.com>

Create Keys In Pairs

- Public and private keys are created at the same time.
- Minimum secure key length is 1024 bits.
  - Public key is shorter
  - Public-key operations are faster.

```
KeyPairGenerator generator =
            KeyPairGenerator.getInstance("RSA");
generator.initialize(1024);  // 1024-bit key
KeyPair keys = generator.genKeyPair();
// get keys with proper encoding, ready to write
// to disk:
byte[] publicKey = keyPair.getPublic().getEncoded();
bute[] privateKey= keyPair.getPrivate().getEncoded();
```

© 2009, Allen I. Holub <http://www.holub.com>

Loading Encoded Public Key

```
ByteArrayOutputStream encoded =
                        new ByteArrayOutputStream();
FileInputStream storedKey = new FileInputStream(...);
for( int b; (b=storedKey.read()) != -1; )
      encoded.write( b );
X509EncodedKeySpec spec =
      new X509EncodedKeySpec(encoded.toByteArray());
encoded.close();

KeyFactory engine = KeyFactory.getInstance("RSA");
PublicKey key = engine.generatePublic(spec);
```

© 2009, Allen I. Holub <http://www.holub.com>

Loading Encoded Private Key

```
ByteArrayOutputStream encoded =
                        new ByteArrayOutputStream();
FileInputStream storedKey = new FileInputStream(...);
for( int b; (b=storedKey.read()) != -1; )
      encoded.write( b );
PKCS8EncodedKeySpec spec =
      new PKCS8EncodedKeySpec(encoded.toByteArray());
encoded.close();

KeyFactory engine = KeyFactory.getInstance("RSA");
PrivateKey key = engine.generatePrivate(spec);
```

© 2009, Allen I. Holub <http://www.holub.com>

En/Deciphering with Public Key

- Just like symmetric encryption, but use public or private key, as appropriate.

```
PublicKey public = ... ; // see previous slide
Cipher rsa = Cipher.getInstance("RSA/ECB/PKCS1Padding");
rsa.init( Cipher.ENCRYPT_MODE, public );
String plainText = ...;
Byte[] cipherText =
            rsa.doFinal( plainText.getBytes() );
//...

PrivateKey private = ...; // see previous slide
Cipher rsa = Cipher.getInstance("RSA/ECB/PKCS1Padding");
rsa.init( Cipher.DECRYPT_MODE, private );

Byte[] plainText =
            rsa.doFinal( cipherText );
```

© 2009, Allen I. Holub <http://www.holub.com>

Digital Signing

© 2009, Allen I. Holub <http://www.holub.com>

Hashing

- Numeric "fingerprint" of document
- MD5
  - 128-bit
  - Known weaknesses could lead to collisions
    - different documents generating same hash
- SHA-1
  - 160-bit
  - Slower than MD5 (about twice the time required)
  - No *known* weakness
- RIPEMD128, RIPEMD160
  - Similar to MD4
  - Developed for EU

© 2009, Allen I. Holub <http://www.holub.com>

Creating a Hash

- Very simple, since no keys required.

```
MessageDigest md5 =
     MessageDigest.getInstance("MD5");
FileInputStream in = new FileInputStream();
for( int c; (c = in.read()) != -1 ; )
     md5.update( (byte)c );
byte[] messageDigest = md5.digest();
```

© 2009, Allen I. Holub <http://www.holub.com>

Creating a MAC

- Key required
- HMACMD5 or HMACSHA1

```
KeyGenerator kg =
     KeyGenerator.getInstance("HMACMD5");
SecretKey key = kg.generateKey();
Mac mac = Mac.getInstance("HMACMD5");
mac.init( key );
byte[] plainText = ...;
byte[] digest = mac.doFinal( plainText );
```

© 2009, Allen I. Holub <http://www.holub.com>

Creating a MAC with CSPRNG

- Avoids opaque secret-key generation.

```
SecureRandom csprng =
     SecureRandom.getInstance("SHA1PRNG");
byte[] randBytes = newByte[16];
csprng.nextBytes( randBytes );
SecretKeySpec key =
     new SecretKeySpec(randBytes, "HMACMD5");
Mac mac = Mac.getInstance("HMACMD5");
mac.init( key );
byte[] plainText = ...;
byte[] digest = mac.doFinal( plainText );
```

© 2009, Allen I. Holub <http://www.holub.com>

Digital Signing

- A digital signature is a private-key-encrypted hash.
- You could do it manually, but don't have to:

```
PrivateKey key = ... ;
Signature engine =
          Signature.getInstance("MD5withRSA");
engine.initSign( key );

String document;
byte[] bytes = document.getBytes("UTF-8");
engine.update(bytes); // can call many times
byte[] signature = engine.sign();
```

© 2009, Allen I. Holub <http://www.holub.com>

Signature Verification

- A digital signature is a private-key-encrypted hash.
- You could do it manually, but don't have to:

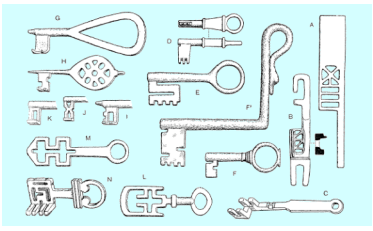
```
PublicKey key = ... ;
Signature engine =
          Signature.getInstance("MD5withRSA");
engine.initVerify( key );

byte[] bytes = ...;           // document to
verify
engine.update( bytes );

if( engine.verify() )
     // signature checks out!
```

© 2009, Allen I. Holub <http://www.holub.com>

Key Management



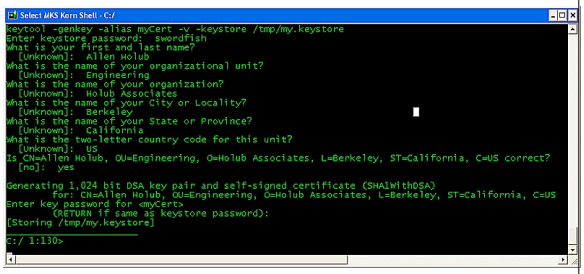
© 2009, Allen I. Holub <http://www.holub.com>

Root Certificates

- Certificates are stored in a keystore.
- Create them with keytool.
- JDK ships with a key store containing various trusted root certificates.
  - $JAVA_HOME/jre/lib/security/cacerts

© 2009, Allen I. Holub <http://www.holub.com>

Create a Certificate with Keytool



- keytool –genkey –alias myCert –v –keystore /tmp/my.keystore
- For RSA cert, add: -keyalg RSA

© 2009, Allen I. Holub <http://www.holub.com>

Create Commercial Certificates

- Use keytool to generate a key pair

  keytool –genkey –keyalg RSA
  –keysize 1024 –alias myCertPair
- BACK UP THE KEYSTORE. NOW!
- Create a certificate request:

  keytool –certreq –alias myCertPair
  –file /certrequest.csr
- Ship .csr file to the CA, who will send you back a PKCS7-encoded certificate, which you can publish.

© 2009, Allen I. Holub <http://www.holub.com>

Import Commercial Certificates

- **keytool -import –v** -alias *fredCert*
  -file *certificate.cer*
  –keystore /tmp/my.keystore
- If no –file is given, then a PKCS7 cert is imported from standard input.
- X.509 certificates (ver 1, 2, or 3) are also supported.
- I know of no way to create a certificate that uses a commercial certificate as its root!

© 2009, Allen I. Holub <http://www.holub.com>

Creating Keystores

- You have to create a keystore before you can use it.
- Various formats are available:
  - JKS (doesn't support symmetric keys!)
  - JCEKS (newer, more secure)
  - others (Providers other than sun, e.g. "BKS")
- The Keystore class is an **in-memory** version of the key store.
  - You must write it to disk to save it.

© 2009, Allen I. Holub <http://www.holub.com>

Creating Keystores (code)

```
Keystore store = KeyStore.getInstance("JKS");
// or getInstance(Keystore.getDefaultType());

// nul==create new keystore
// empty array for no password.
store.load( null, new char[0] );

FileOutputStream out = new FileOutputStream(
                    new
File("/tmp/my.keystore") );
char[] password = new char[]{'p','w','d'};
store.store(out,password);
Arrays.fill(password, '\u0000');
```

© 2009, Allen I. Holub <http://www.holub.com>

Listing Keystore Contents

```
String defaultFormat =
     Security.getProperty("keystore.type");
Keystore store =
     Keystore.geInstance(defaultFormat);

FileInputStream in = new FileInputStream(
          new File("/tmp/my.keystore") );
char[] password = new char[]{ 'p','w','d' };
store.load( in, password );
Arrays.fill(password, `\u0000`);

Enumeration e = store.aliases();
while( e.hasMoreElements() )
{    String name = (String) e.nextElement();
     boolean isKey = store.isKeyEntry(name);
}
```

© 2009, Allen I. Holub <http://www.holub.com>

Storing Secret Key

```
char[] password = new char[]{'p','w','d'};
SecretKey key = ...;
Keystore store = KeyStore.getInstance("BKS");
store.load(null, new char[0]);

store.setKeyEntry("myKey",key,password,null);
FileOutputStream out =
                    new FileOutputStream(...);
store.store(out, password);
Arrays.fill( password, '\u000' );
```

- Keystore and key passwords don't have to be the same, but often are.

© 2009, Allen I. Holub <http://www.holub.com>

Retrieving Secret Key

```
char[] password = new char[]{'p','w','d'};
Keystore store = KeyStore.getInstance("BKS");

FileInputStream in = new FileInputStream(
          new File("/tmp/my.keystore") );
store.load(in, password);

Key key = store.getKey("myKey", password);

Arrays.fill( password, '\u000' );
```

© 2009, Allen I. Holub <http://www.holub.com>

Dealing with Certificates

```
CertificateFactory factory =
     CertificateFactory.getInstance("X.509");
FileInputStream in = ...;
Certificate cert = generateCertificate(in);
```

- Two Representation Systems:
  - DER (Distinguished Encoding Rules – default for keytool)
  - Base64 encoded (RFC2045 [mime])
- Export certificate to file system:

```
keytool –export –alias myCert –keystore /tmp/my.keystore \
                                    –file /myCert.der
keytool –export –alias myCert –keystore /tmp/my.keystore \
                                    –file /myCert.b64 –rfc
```

- For Bas64-encoded certs:
  `generateCertificate(byteArrayHoldingCert);`

© 2009, Allen I. Holub <http://www.holub.com>

Getting Key from Certificate

```
CertificateFactory factory =
     CertificateFactory.getInstance("X.509");
FileInputStream in = ...;
Certificate cert = generateCertificate(in);

Cipher cipher =
      Cipher.getInstance("RSA/ECB/PKCS#1");
cipher.init(Cipher.ENCRYPT_MODE, cert );
cipher.doFinal( text.getBytes("UTF-8") );
```

- You can pass a certificate to `init()` instead of a *public* key.

© 2009, Allen I. Holub <http://www.holub.com>