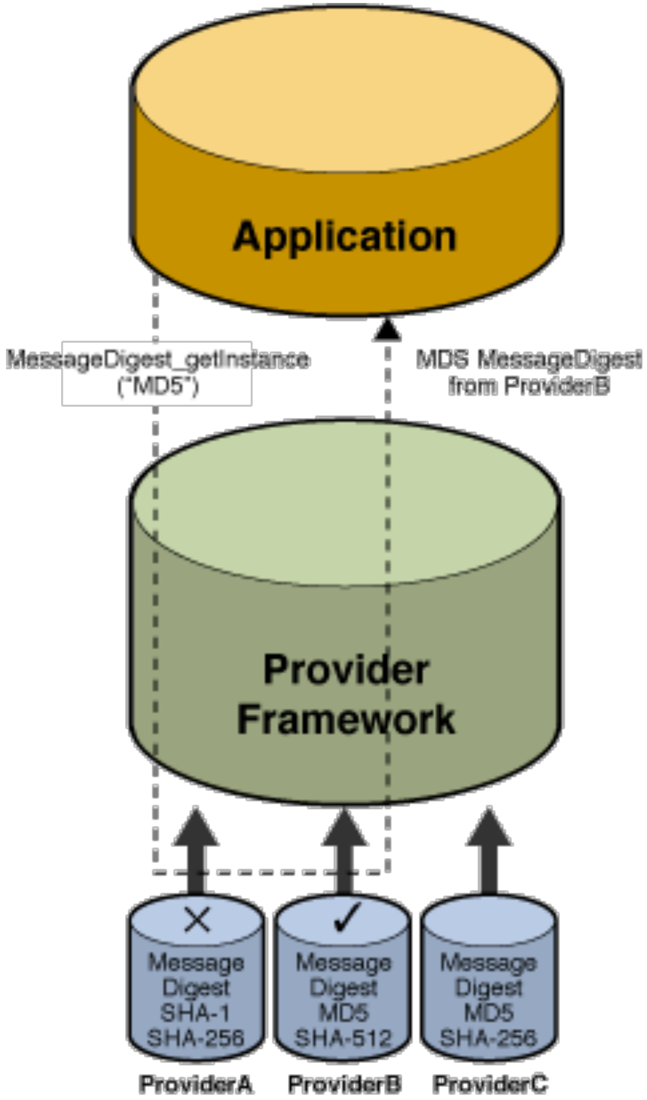
JCA/JCE

Java Crypto Architecture / Java Crypto Extensions

Dušan Klinec
deadcode.me

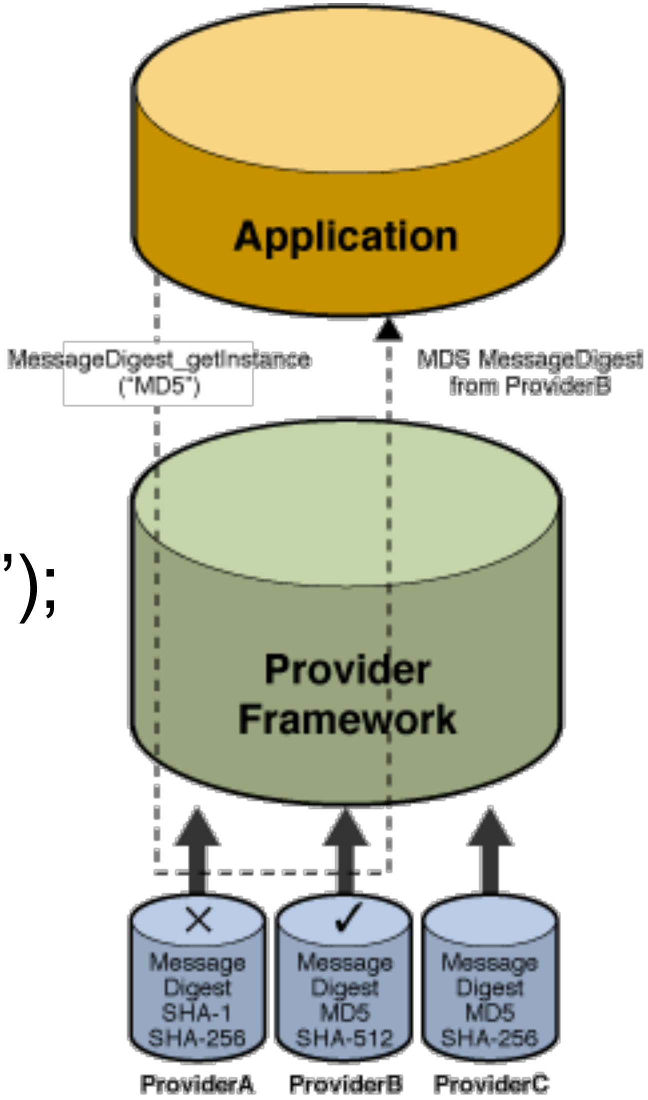
CROCS
Centre for Research on
Cryptography and Security

Provider architecture



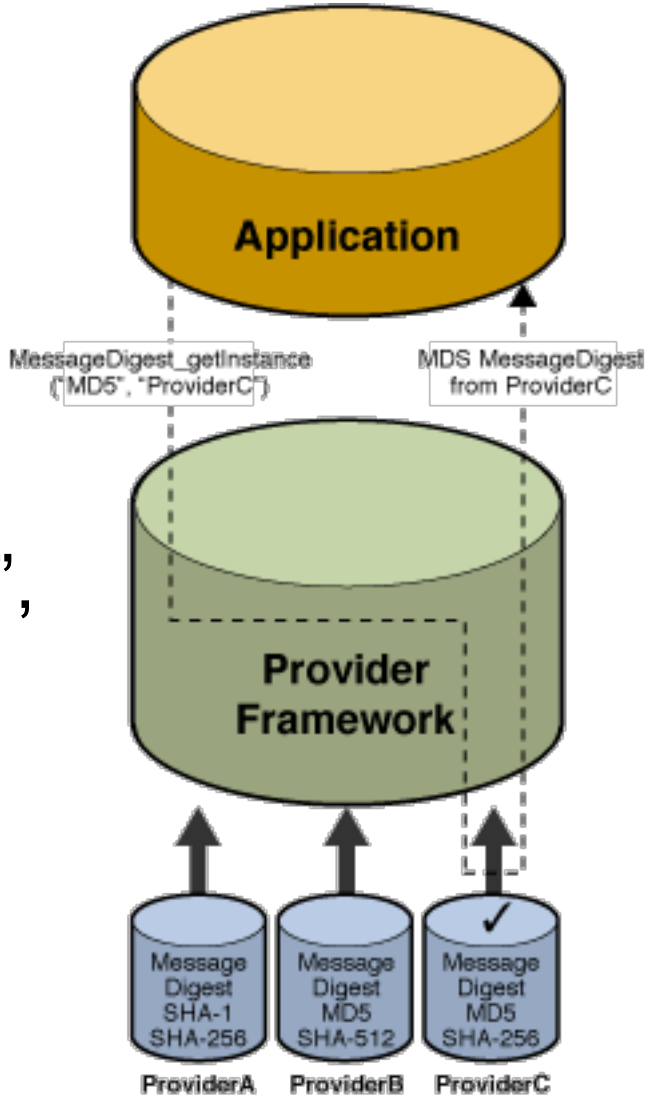
Provider architecture

MessageDigest.
getInstance("MD5");


Application
MessageDigest.getInstance ("MD5")
MD5 MessageDigest from ProviderB
Provider Framework
×
✓
Message Digest SHA-1 SHA-256
Message Digest MD5 SHA-512
Message Digest MD5 SHA-256
ProviderA
ProviderB
ProviderC

Provider architecture

MessageDigest.
getInstance("MD5", "ProviderC");


Application
MessageDigest.getInstance ("MD5", "ProviderC")
MD5 MessageDigest from ProviderC
Provider Framework
Message Digest SHA-1 SHA-256
Message Digest MD5 SHA-512
✓
Message Digest MD5 SHA-256
ProviderA
ProviderB
ProviderC

JCA

- java.security.*
  - SecureRandom - PRNG
  - MessageDigest – SHA256, MD5, ...
  - Signature – RSA, DSA
  - KeyStore – PKCS12
  - KeyPairGenerator, KeyFactory, CertificateFactory,

JCE

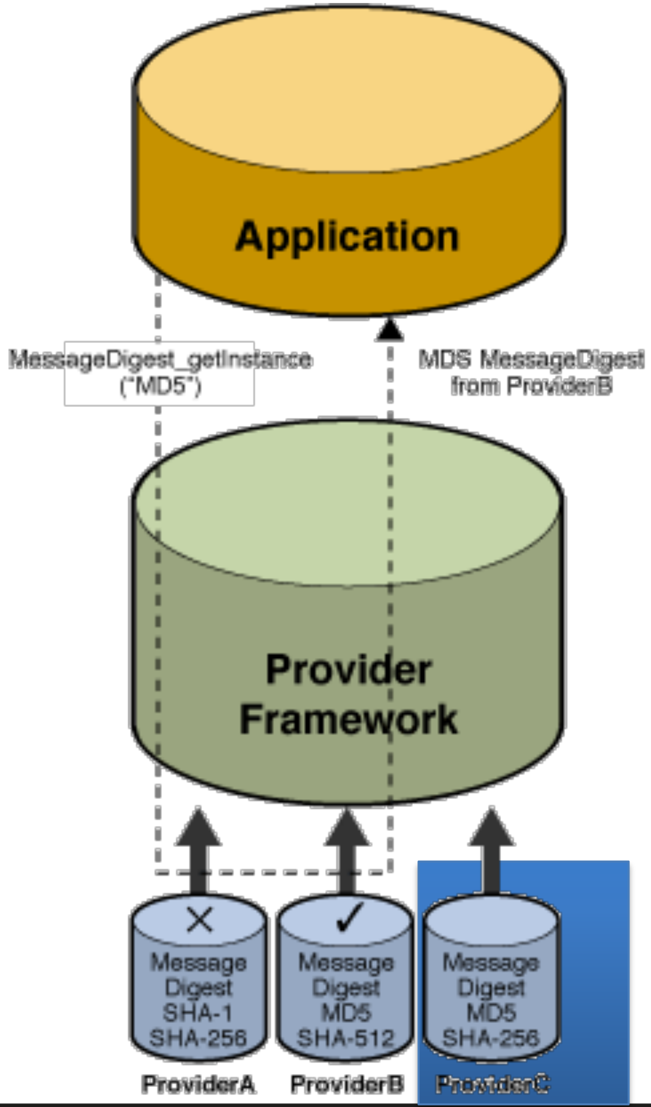
- javax.crypto.*
  - Cipher – AES, RSA, ElGamal, RC4, Salsa20
  - Mac – HMACWithSHA256
  - KeyGenerator

Provider architecture

- Implementation independence
- Implementation interoperability
- Algorithm extensibility

Bouncy Castle

Bouncy Castle

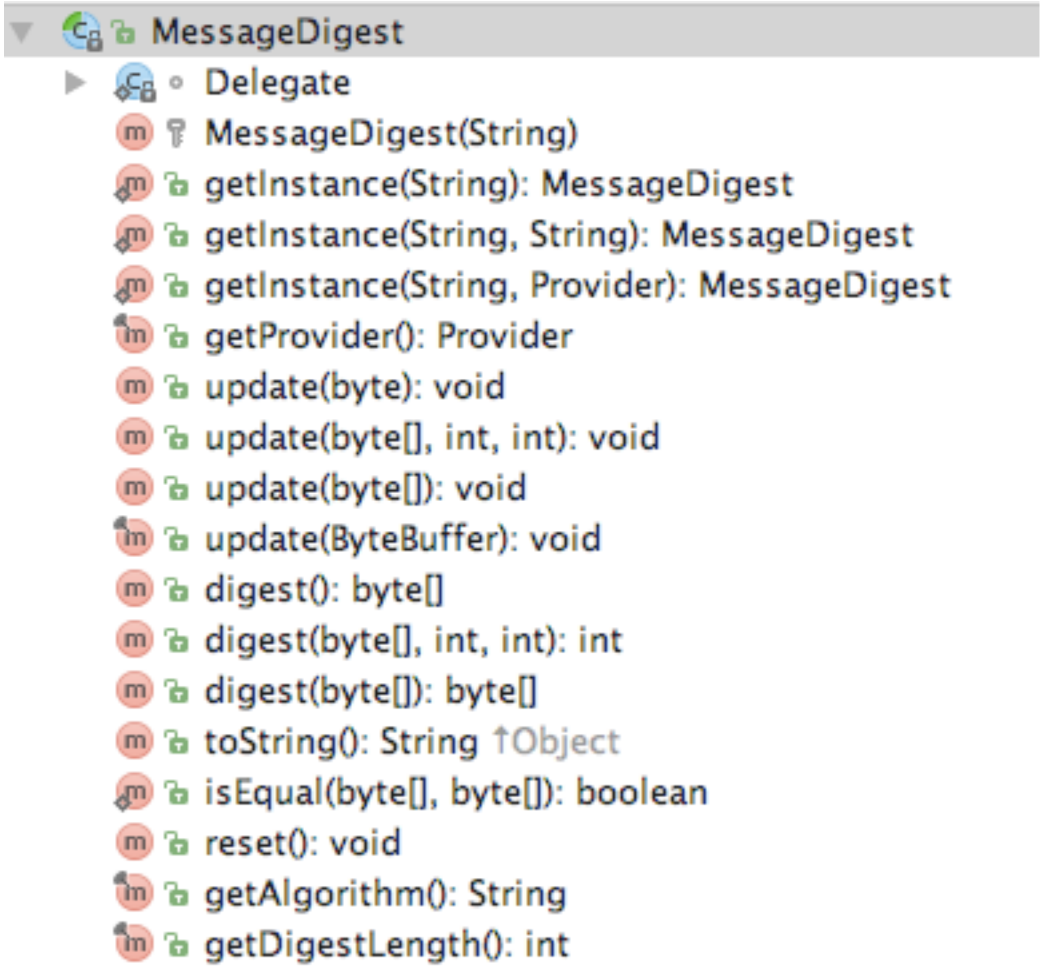


BouncyCastle

Bouncy Castle

- Implements a LOT OF ciphers, cipher suites, algorithms, modes, ASN.1, PEM, Certs, …
- Origin: Australian, former advantage (crypto regulations)
- Android

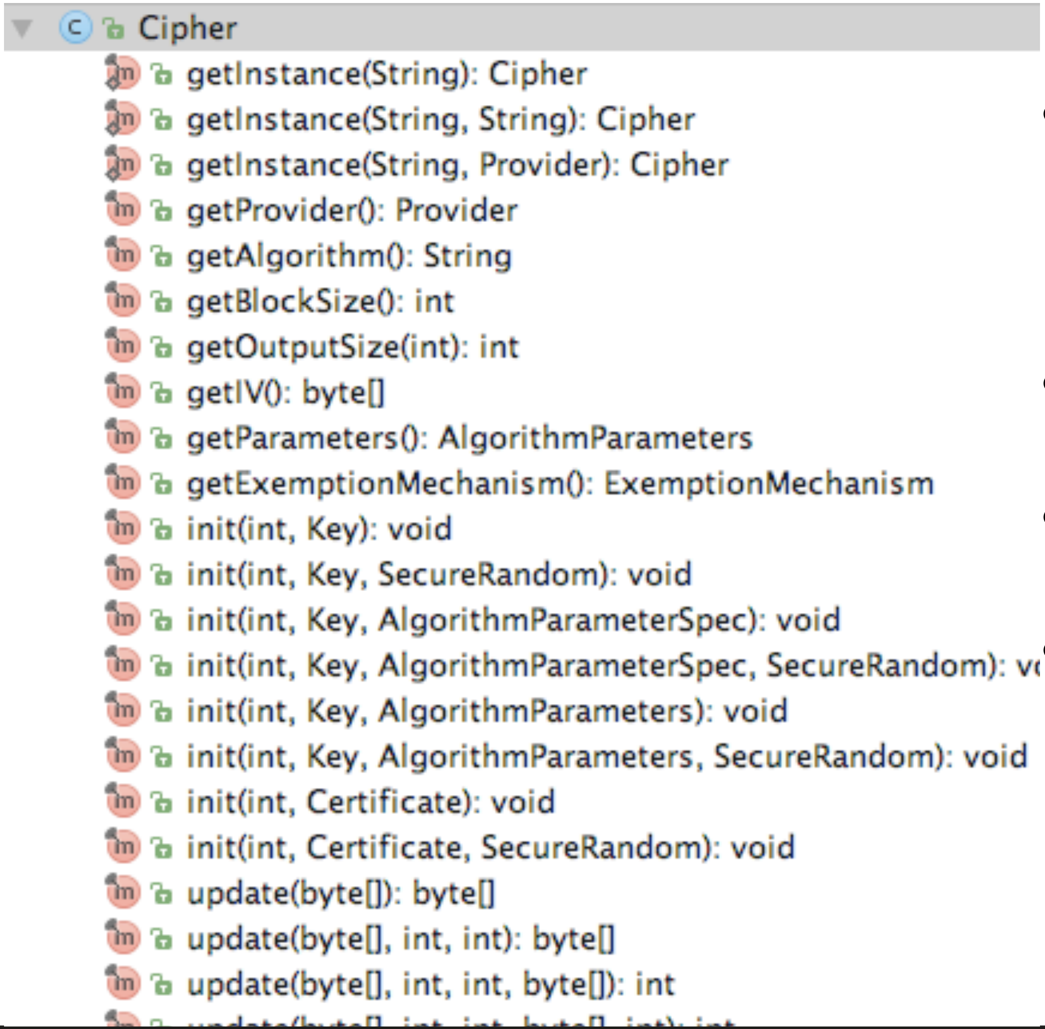
Provider architecture – Engine classes



```
MessageDigest
  Delegate
  MessageDigest(String)
  getInstance(String): MessageDigest
  getInstance(String, String): MessageDigest
  getInstance(String, Provider): MessageDigest
  getProvider(): Provider
  update(byte): void
  update(byte[], int, int): void
  update(byte[]): void
  update(ByteBuffer): void
  digest(): byte[]
  digest(byte[], int, int): int
  digest(byte[]): byte[]
  toString(): String ↑Object
  isEqual(byte[], byte[]): boolean
  reset(): void
  getAlgorithm(): String
  getDigestLength(): int
```

- getInstance()
- update()
- digest()
- reset()

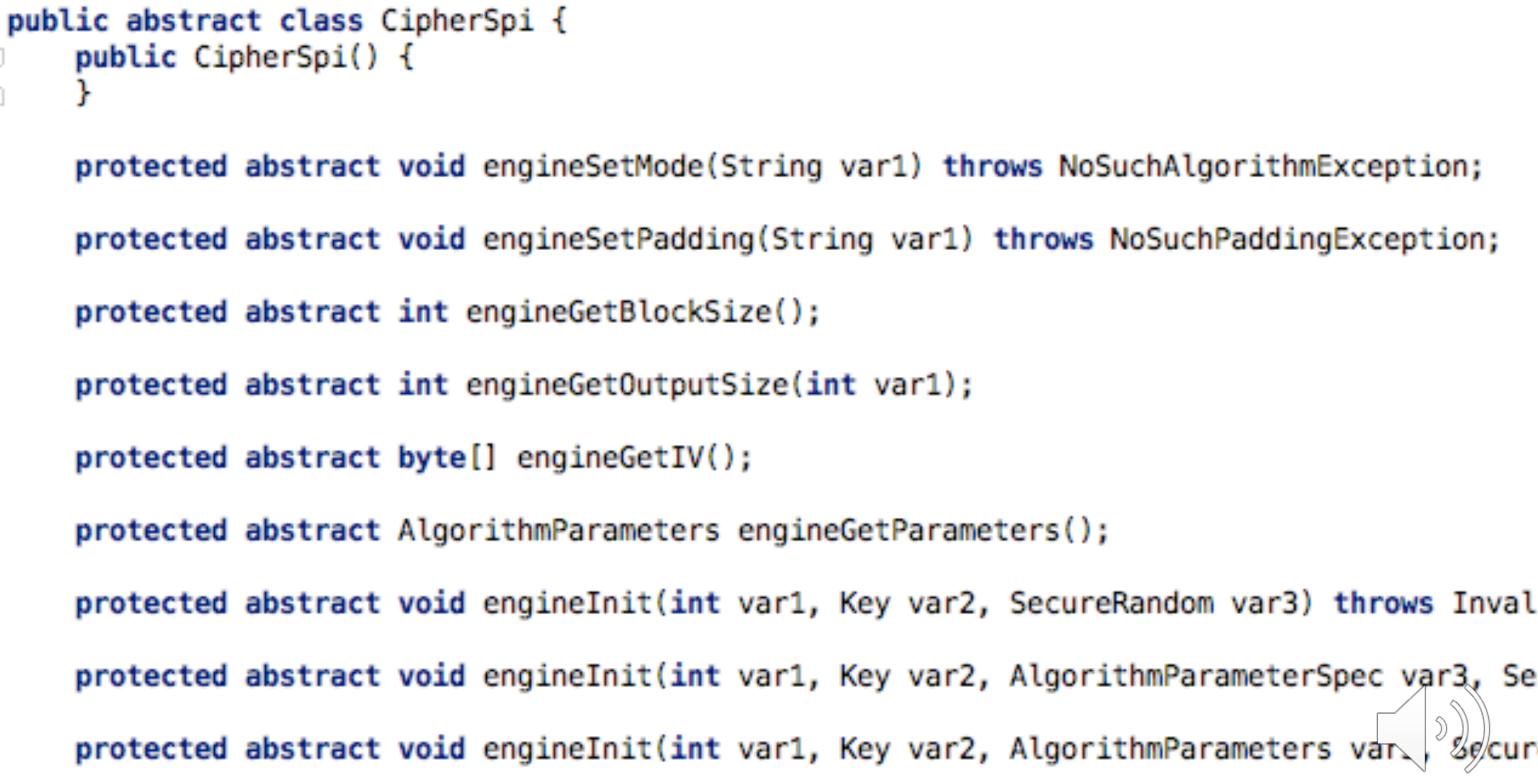
Provider architecture – Engine classes



```
Cipher
    getInstance(String): Cipher
    getInstance(String, String): Cipher
    getInstance(String, Provider): Cipher
    getProvider(): Provider
    getAlgorithm(): String
    getBlockSize(): int
    getOutputSize(int): int
    getIV(): byte[]
    getParameters(): AlgorithmParameters
    getExemptionMechanism(): ExemptionMechanism
    init(int, Key): void
    init(int, Key, SecureRandom): void
    init(int, Key, AlgorithmParameterSpec): void
    init(int, Key, AlgorithmParameterSpec, SecureRandom): vo
    init(int, Key, AlgorithmParameters): void
    init(int, Key, AlgorithmParameters, SecureRandom): void
    init(int, Certificate): void
    init(int, Certificate, SecureRandom): void
    update(byte[]): byte[]
    update(byte[], int, int): byte[]
    update(byte[], int, int, byte[]): int
    update(byte[], int, int, byte[], int): int
```

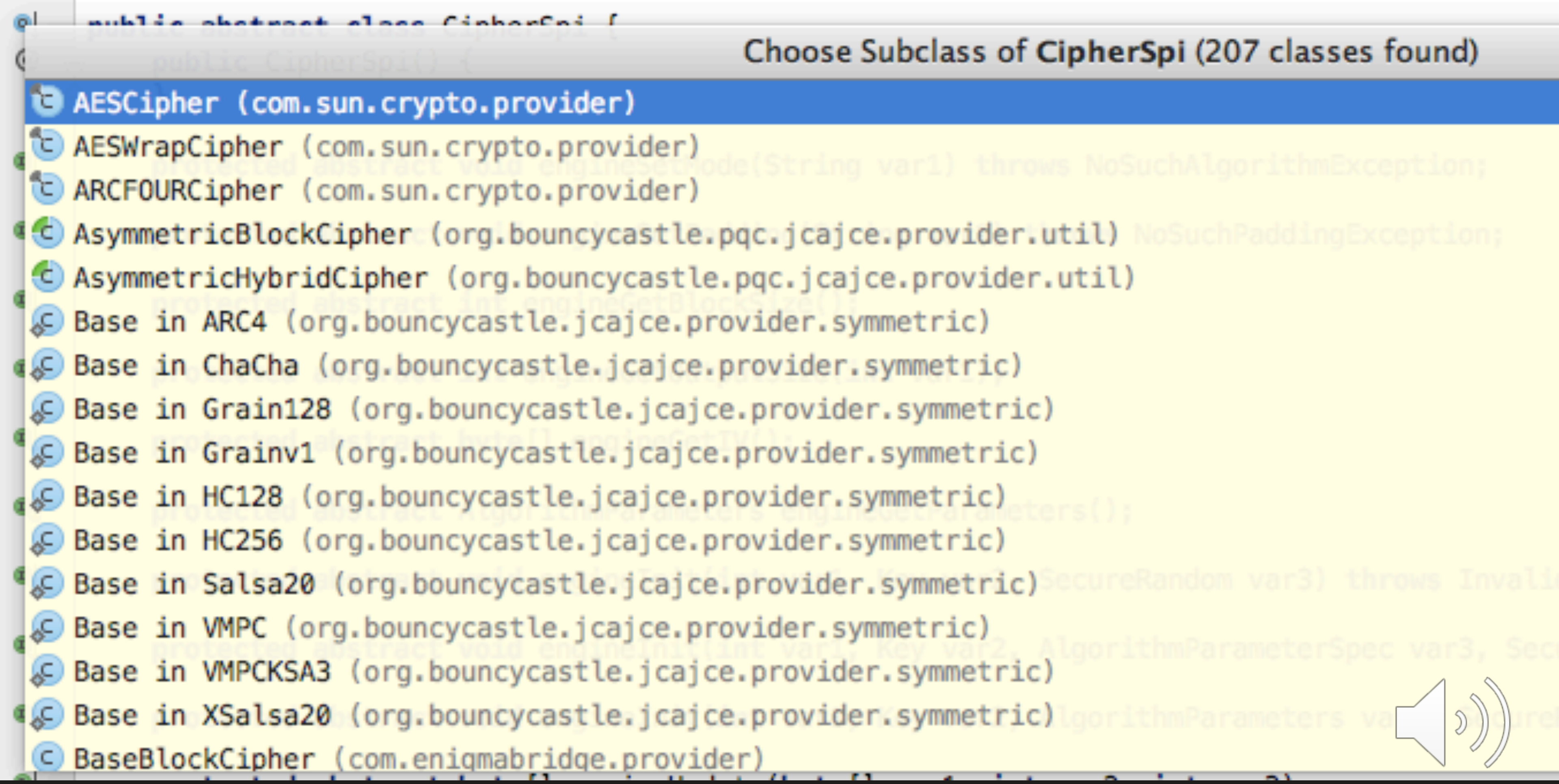
- getInstance()
- init()
- update()
- doFinal()

Provider architecture – Spi skeleton



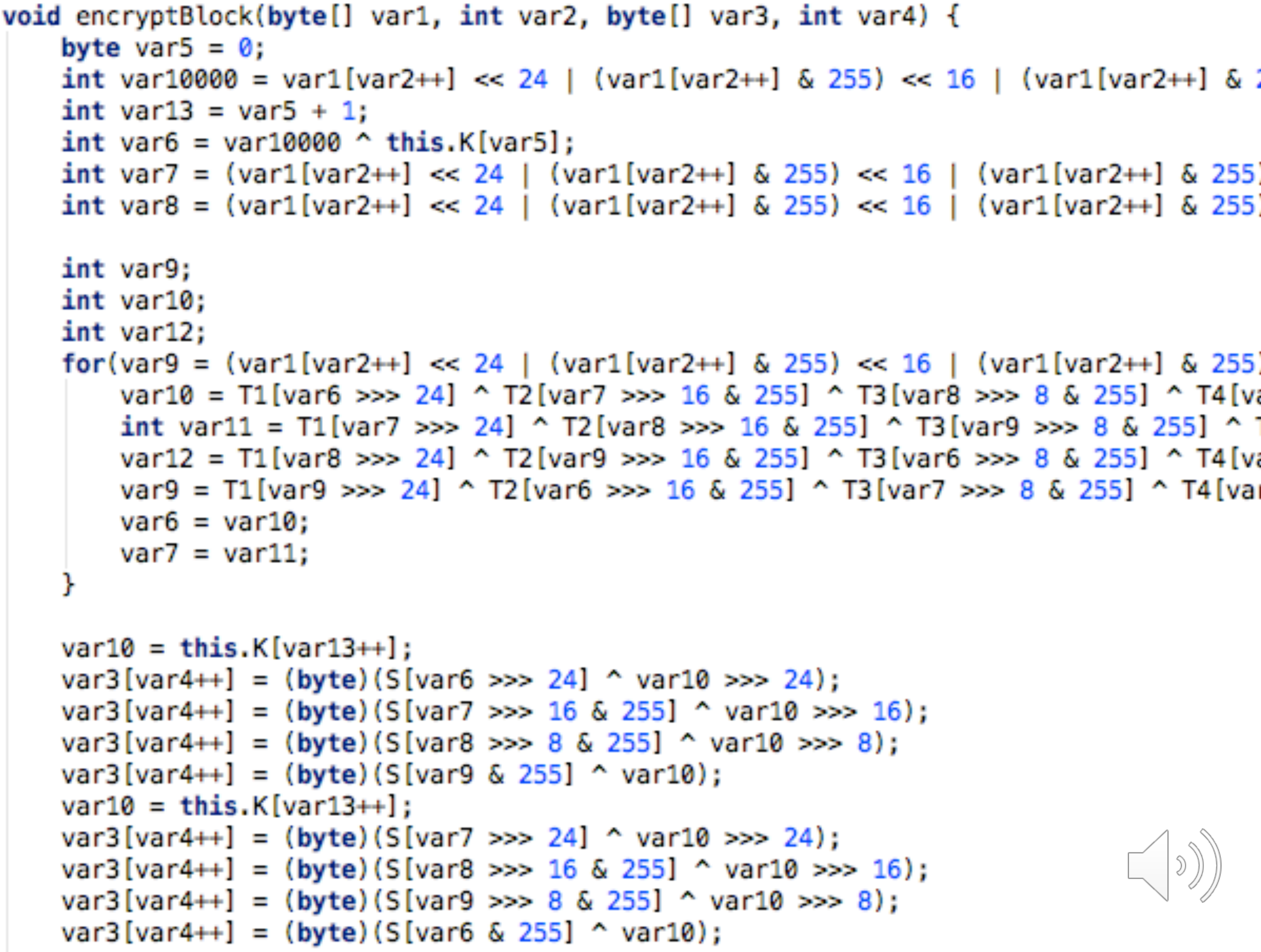
```
public abstract class CipherSpi {
    public CipherSpi() {
    }

    protected abstract void engineSetMode(String var1) throws NoSuchAlgorithmException;

    protected abstract void engineSetPadding(String var1) throws NoSuchPaddingException;

    protected abstract int engineGetBlockSize();

    protected abstract int engineGetOutputSize(int var1);

    protected abstract byte[] engineGetIV();

    protected abstract AlgorithmParameters engineGetParameters();

    protected abstract void engineInit(int var1, Key var2, SecureRandom var3) throws Inval

    protected abstract void engineInit(int var1, Key var2, AlgorithmParameterSpec var3, Se

    protected abstract void engineInit(int var1, Key var2, AlgorithmParameters var3, Secur
```

Provider architecture – Spi skeleton



Choose Subclass of **CipherSpi** (207 classes found)

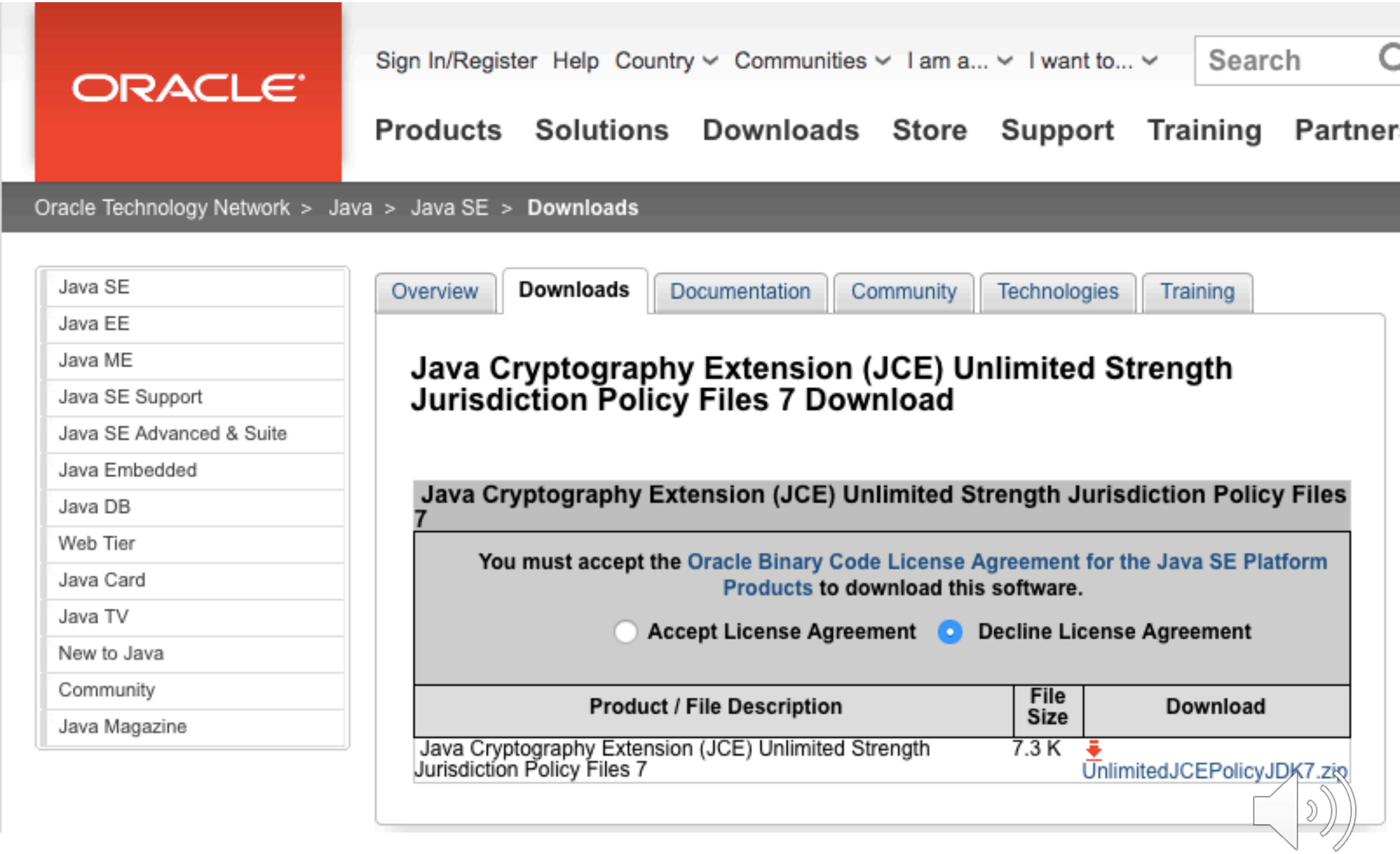
```
AESCipher (com.sun.crypto.provider)
AESWrapCipher (com.sun.crypto.provider)
ARCFOURCipher (com.sun.crypto.provider)
AsymmetricBlockCipher (org.bouncycastle.pqc.jcajce.provider.util)
AsymmetricHybridCipher (org.bouncycastle.pqc.jcajce.provider.util)
Base in ARC4 (org.bouncycastle.jcajce.provider.symmetric)
Base in ChaCha (org.bouncycastle.jcajce.provider.symmetric)
Base in Grain128 (org.bouncycastle.jcajce.provider.symmetric)
Base in Grainv1 (org.bouncycastle.jcajce.provider.symmetric)
Base in HC128 (org.bouncycastle.jcajce.provider.symmetric)
Base in HC256 (org.bouncycastle.jcajce.provider.symmetric)
Base in Salsa20 (org.bouncycastle.jcajce.provider.symmetric)
Base in VMPC (org.bouncycastle.jcajce.provider.symmetric)
Base in VMPCKSA3 (org.bouncycastle.jcajce.provider.symmetric)
Base in XSalsa20 (org.bouncycastle.jcajce.provider.symmetric)
BaseBlockCipher (com.enigmabridge.provider)
```



```
void encryptBlock(byte[] var1, int var2, byte[] var3, int var4) {
    byte var5 = 0;
    int var10000 = var1[var2++] << 24 | (var1[var2++] & 255) << 16 | (var1[var2++] & 2
    int var13 = var5 + 1;
    int var6 = var10000 ^ this.K[var5];
    int var7 = (var1[var2++] << 24 | (var1[var2++] & 255) << 16 | (var1[var2++] & 255
    int var8 = (var1[var2++] << 24 | (var1[var2++] & 255) << 16 | (var1[var2++] & 255

    int var9;
    int var10;
    int var12;
    for(var9 = (var1[var2++] << 24 | (var1[var2++] & 255) << 16 | (var1[var2++] & 255
        var10 = T1[var6 >>> 24] ^ T2[var7 >>> 16 & 255] ^ T3[var8 >>> 8 & 255] ^ T4[va
        int var11 = T1[var7 >>> 24] ^ T2[var8 >>> 16 & 255] ^ T3[var9 >>> 8 & 255] ^ T
        var12 = T1[var8 >>> 24] ^ T2[var9 >>> 16 & 255] ^ T3[var6 >>> 8 & 255] ^ T4[va
        var9 = T1[var9 >>> 24] ^ T2[var6 >>> 16 & 255] ^ T3[var7 >>> 8 & 255] ^ T4[var
        var6 = var10;
        var7 = var11;
    }

    var10 = this.K[var13++];
    var3[var4++] = (byte)(S[var6 >>> 24] ^ var10 >>> 24);
    var3[var4++] = (byte)(S[var7 >>> 16 & 255] ^ var10 >>> 16);
    var3[var4++] = (byte)(S[var8 >>> 8 & 255] ^ var10 >>> 8);
    var3[var4++] = (byte)(S[var9 & 255] ^ var10);
    var10 = this.K[var13++];
    var3[var4++] = (byte)(S[var7 >>> 24] ^ var10 >>> 24);
    var3[var4++] = (byte)(S[var8 >>> 16 & 255] ^ var10 >>> 16);
    var3[var4++] = (byte)(S[var9 >>> 8 & 255] ^ var10 >>> 8);
    var3[var4++] = (byte)(S[var6 & 255] ^ var10);
```

Strong cryptography

- Limits the strength of your crypto
  - the size of the Key
- AES-256, RSA-2048 not available by default
- Java Cryptography Extension (JCE) Unlimited Strength Jurisdiction Policy Files



Java Cryptography Extension (JCE) Unlimited Strength Jurisdiction Policy Files 7 Download

Java Cryptography Extension (JCE) Unlimited Strength Jurisdiction Policy Files 7

You must accept the Oracle Binary Code License Agreement for the Java SE Platform Products to download this software.

○ Accept License Agreement ● Decline License Agreement

Product / File Description	File Size	Download
Java Cryptography Extension (JCE) Unlimited Strength Jurisdiction Policy Files 7	7.3 K	UnlimitedJCEPolicyJDK7.zip

Strong cryptography

Algorithm	Key size
DES	64
DESede	*
RC2	128
RC4	128
RC5	128
RSA	* (KeyPairGenerator 1024)
other	128

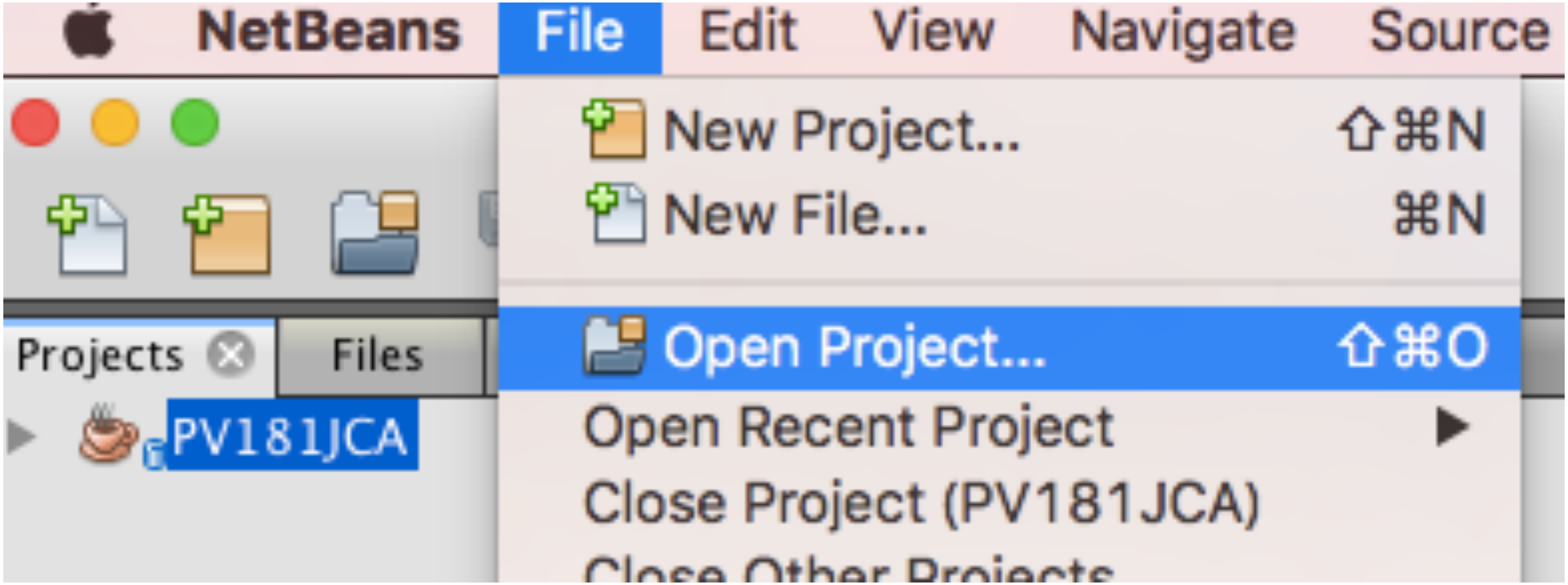
Download NetBeans project

Visit study materials

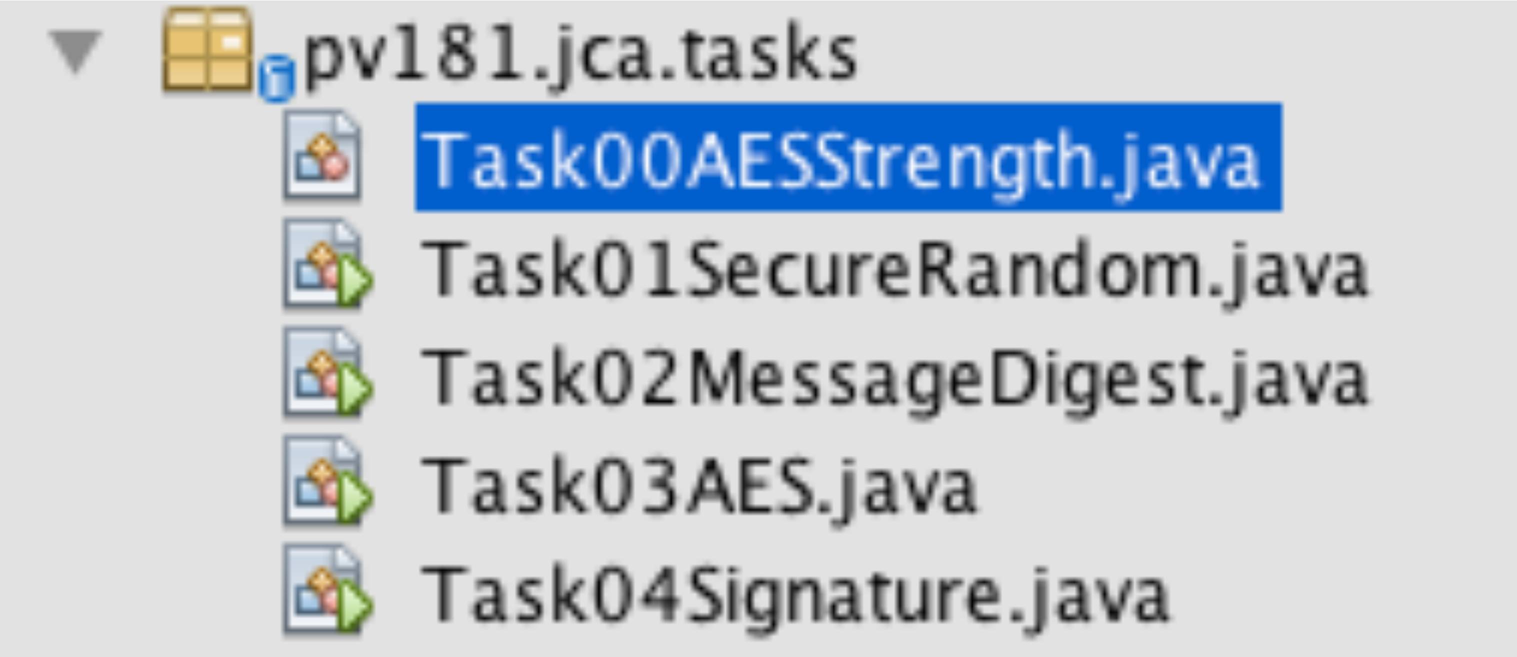
Pls open / download

NetBeans 12

Pls open

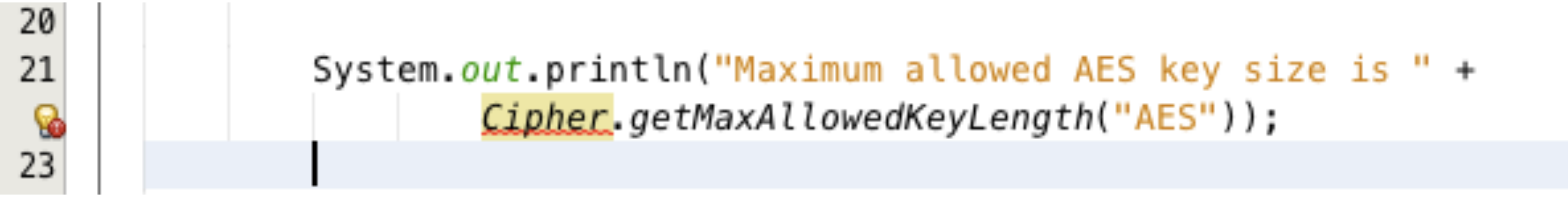

NetBeans
File
Edit
View
Navigate
Source
New Project... ⇧⌘N
New File... ⌘N
Open Project... ⇧⌘O
Open Recent Project ▶
Close Project (PV181JCA)
Close Other Projects
Projects
Files
PV181JCA

Getting started



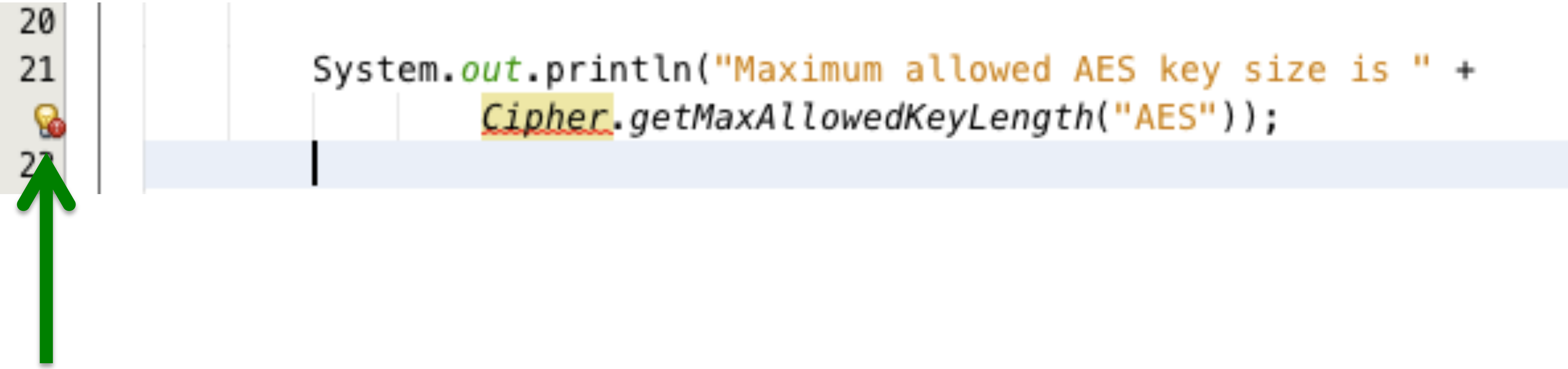
pv181.jca.tasks
- Task00AESStrength.java
- Task01SecureRandom.java
- Task02MessageDigest.java
- Task03AES.java
- Task04Signature.java

Cipher – import missing



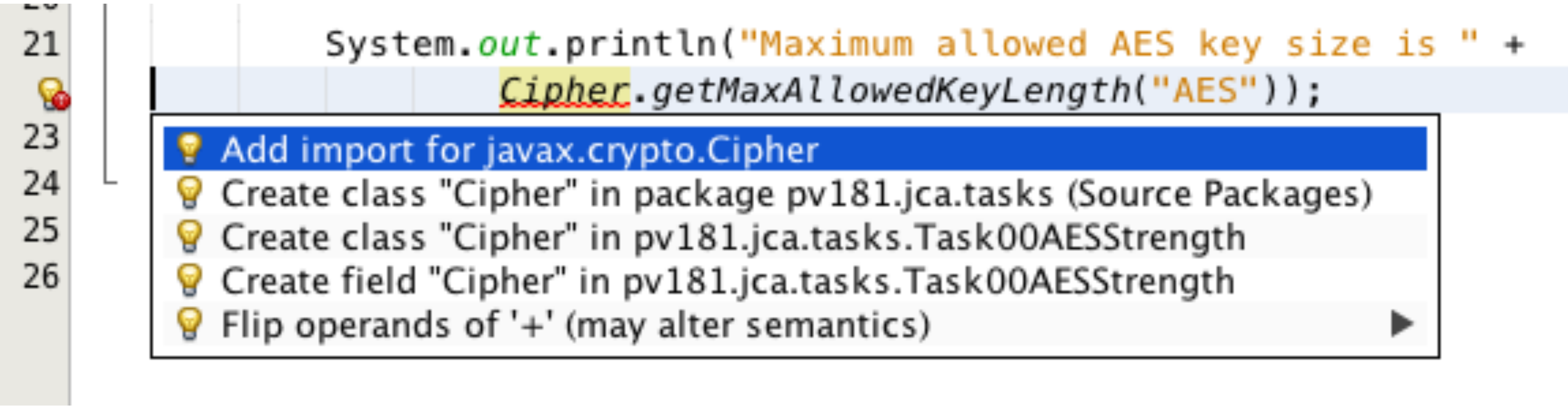
```
            System.out.println("Maximum allowed AES key size is " +
                    Cipher.getMaxAllowedKeyLength("AES"));

```

Cipher – import missing



```
        System.out.println("Maximum allowed AES key size is " +
                Cipher.getMaxAllowedKeyLength("AES"));
```

Lighbulb helps

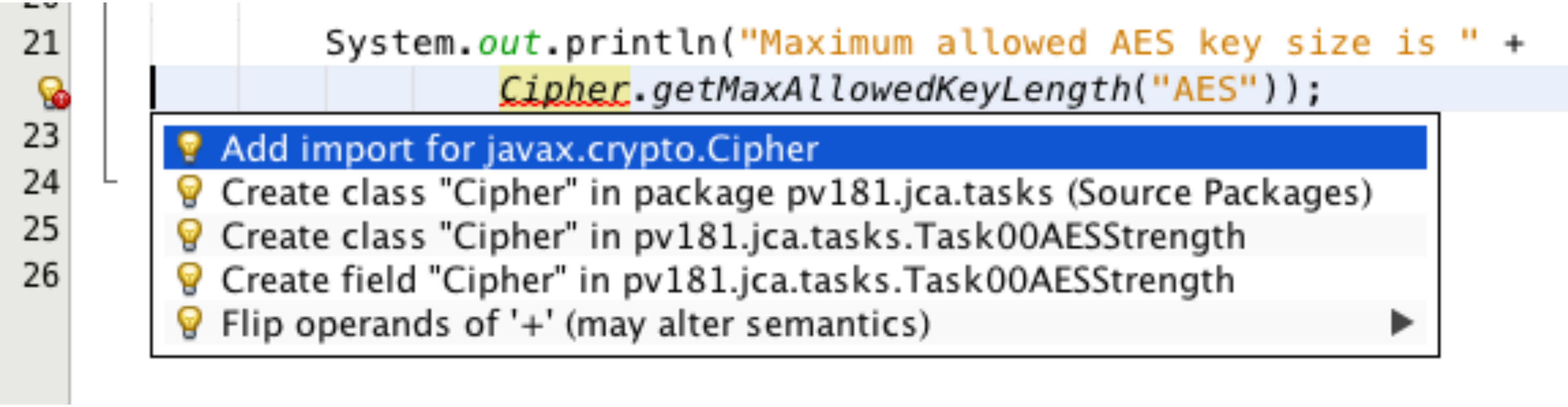


```
System.out.println("Maximum allowed AES key size is " +
        Cipher.getMaxAllowedKeyLength("AES"));
```

Add import for javax.crypto.Cipher
Create class "Cipher" in package pv181.jca.tasks (Source Packages)
Create class "Cipher" in pv181.jca.tasks.Task00AESStrength
Create field "Cipher" in pv181.jca.tasks.Task00AESStrength
Flip operands of '+' (may alter semantics) ▶

Getting started

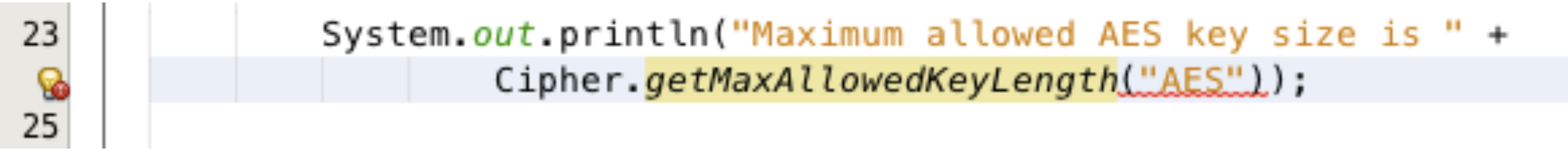
CTRL+SHIFT+I



```
        System.out.println("Maximum allowed AES key size is " +
                Cipher.getMaxAllowedKeyLength("AES"));
```

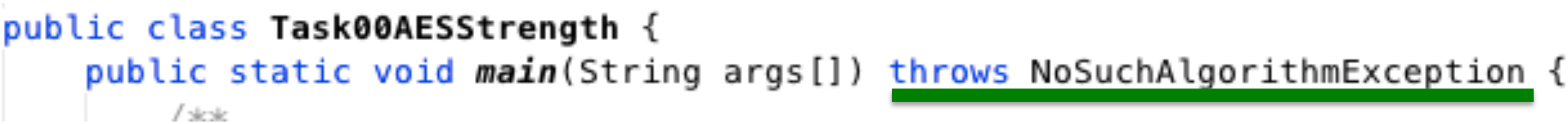
Add import for javax.crypto.Cipher
Create class "Cipher" in package pv181.jca.tasks (Source Packages)
Create class "Cipher" in pv181.jca.tasks.Task00AESStrength
Create field "Cipher" in pv181.jca.tasks.Task00AESStrength
Flip operands of '+' (may alter semantics)

Problem again



```
System.out.println("Maximum allowed AES key size is " +
        Cipher.getMaxAllowedKeyLength("AES"));
```

Problem again



```
public class Task00AESStrength {
    public static void main(String args[]) throws NoSuchAlgorithmException {
        /**
```

The web

Pls open – the guide

goo.gl/4Ztqen

Case sensitive

Task01 - SecureRandom

- SecureRandom rnd = new SecureRandom()
- rnd.nextDouble()
- rnd.nextByte()
- rnd. ....

Task01 - SecureRandom

1. SecureRandom

Generate 1024 random bytes using **SecureRandom** and print it with *Globals.bytesToHex()*

- SecureRandom uses PRNG implementations from Cryptography Service Providers. It may be:
  - *SHA1PRNG* default on Windows.
  - *NativePRNG* which reads /dev/urandom on Linux & Solaris.
  - Something completely different, depending on the architecture, provider, configuration, ...
- SecureRandom is automatically seeded by OS means before a first call for a random data.
- This seeding mechanism is bypassed if *java.security.SecureRandom.setSeed(byte[])* is called.
- You should not explicitly seed SecureRandom with your own data unless you are sure seed is not predictable.
- For long term SecureRandom object is is recommended to re-seed it from time to time: *rand.setSeed(java.security.SecureRandom.generateSeed(int))*.

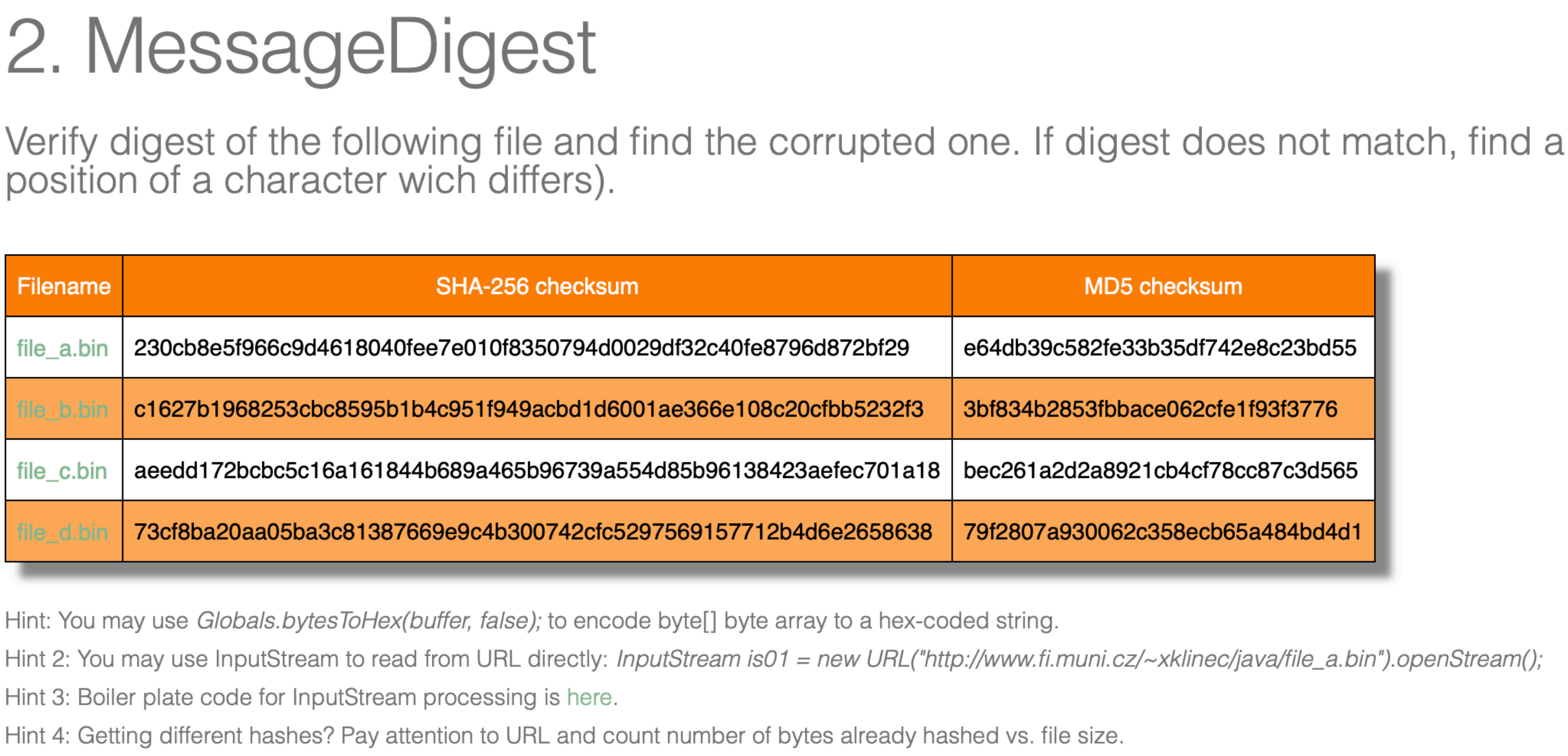
Task 01

- You can work now

SecureRandom - solution

- SecureRandom rnd = new SecureRandom();
- rnd.nextBytes(buffer);
- System.out.println(Globals.bytesToHex(buffer));

Task02 - MessageDigest



2. MessageDigest

Verify digest of the following file and find the corrupted one. If digest does not match, find a difference (i.e., position of a character wich differs).

Filename	SHA-256 checksum	MD5 checksum
file_a.bin	230cb8e5f966c9d4618040fee7e010f8350794d0029df32c40fe8796d872bf29	e64db39c582fe33b35df742e8c23bd55
file_b.bin	c1627b1968253cbc8595b1b4c951f949acbd1d6001ae366e108c20cfbb5232f3	3bf834b2853fbbace062cfe1f93f3776
file_c.bin	aeedd172bcbc5c16a161844b689a465b96739a554d85b96138423aefec701a18	bec261a2d2a8921cb4cf78cc87c3d565
file_d.bin	73cf8ba20aa05ba3c81387669e9c4b300742cfc5297569157712b4d6e2658638	79f2807a930062c358ecb65a484bd4d1

Hint: You may use *Globals.bytesToHex(buffer, false);* to encode byte[] byte array to a hex-coded string.

Hint 2: You may use InputStream to read from URL directly: *InputStream is01 = new URL("http://www.fi.muni.cz/~xklinec/java/file_a.bin").openStream();*

Hint 3: Boiler plate code for InputStream processing is here.

Hint 4: Getting different hashes? Pay attention to URL and count number of bytes already hashed vs. file size.

Task02 - MessageDigest

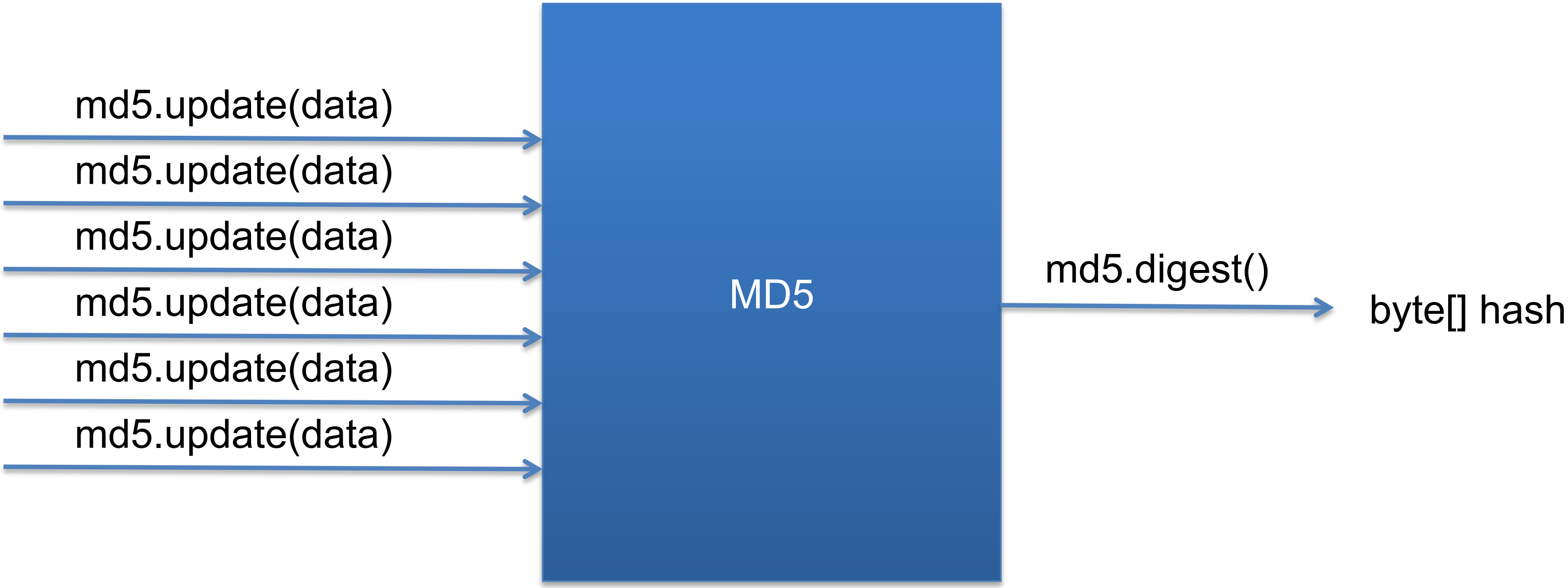
- MessageDigest md5 = MessageDigest.getInstance(“MD5”);

MessageDigest

- MessageDigest md5 = MessageDigest.getInstance(“MD5”);

- md5.update(inputBuffer, 0, bytesRead);
- md5.update(inputBuffer, 0, bytesRead);
- md5.update(inputBuffer, 0, bytesRead);
- byte[] md5hash = md5.digest()

MessageDigest – incremental API



md5.update(data)

md5.update(data)

md5.update(data)

md5.update(data)

md5.update(data)

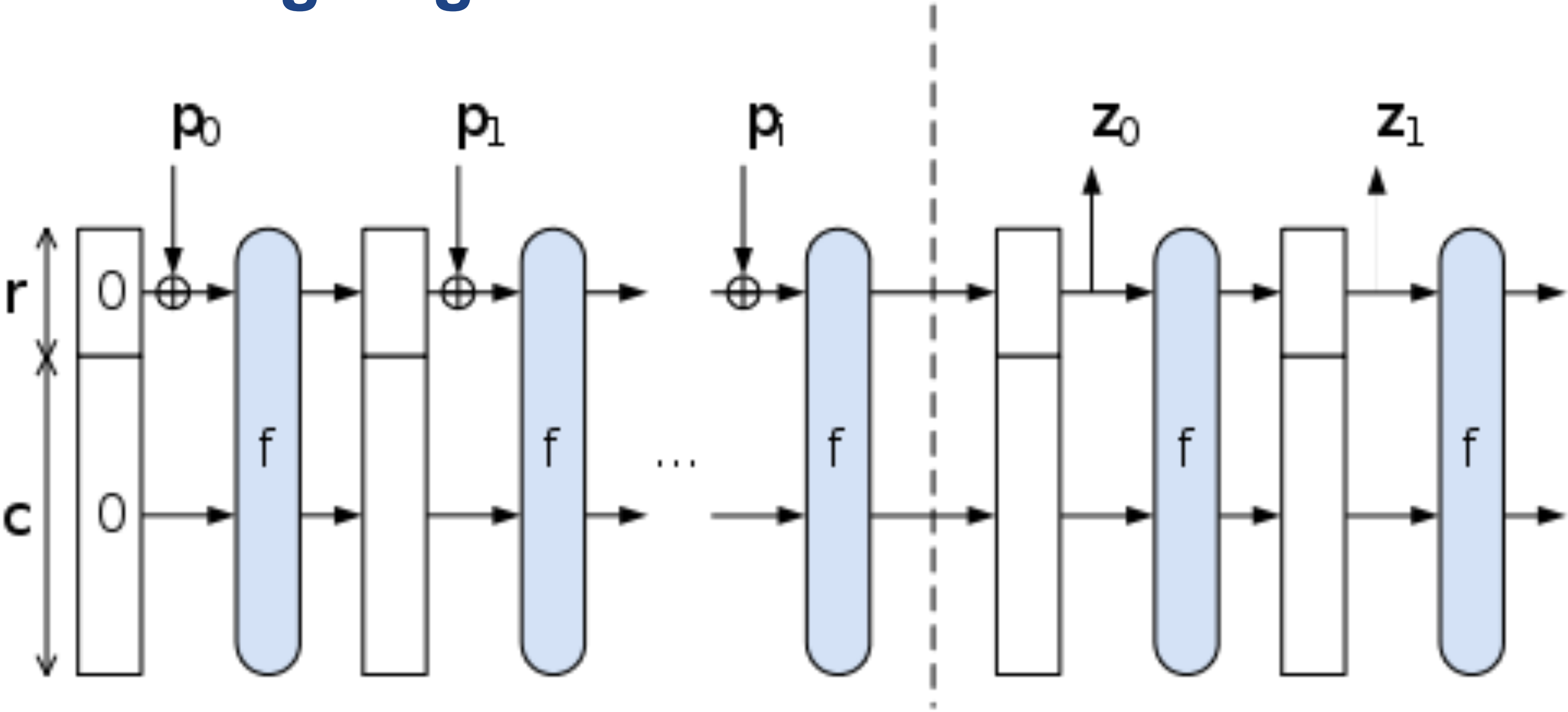
md5.update(data)

MD5

md5.digest()

byte[] hash

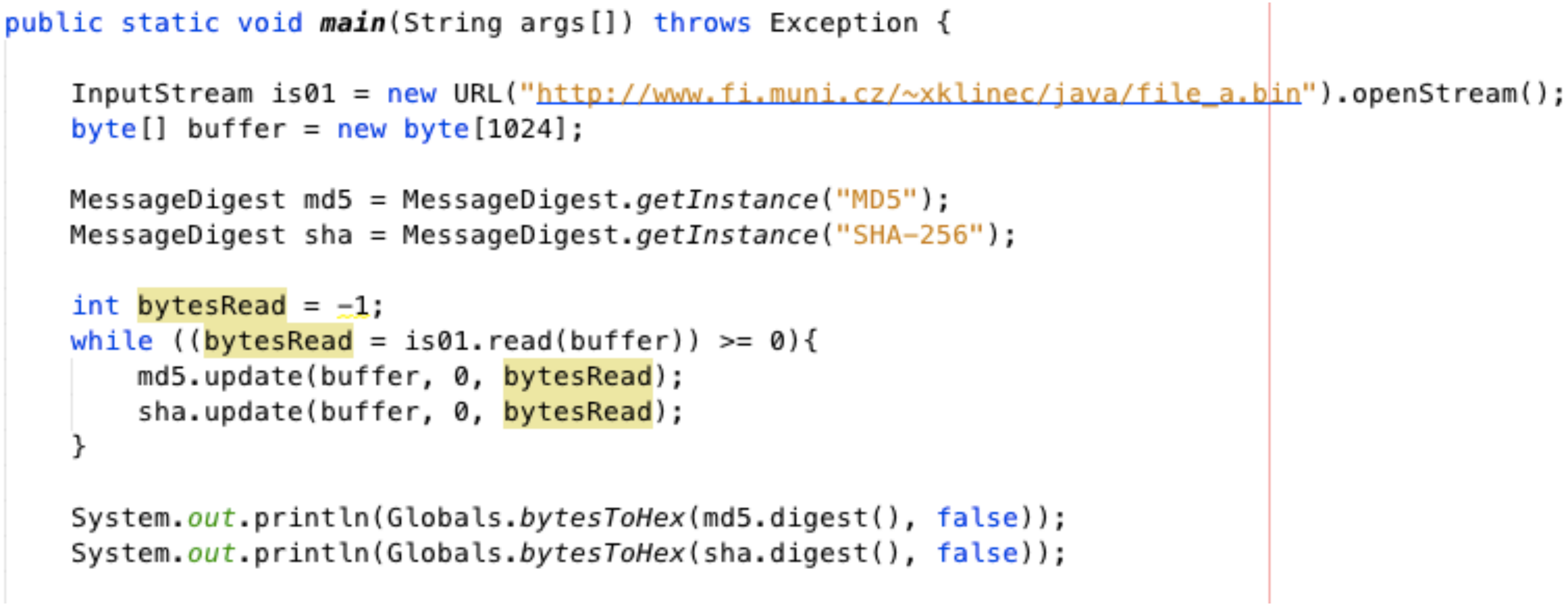
MessageDigest – incremental API



Task 02

- You can work now

MessageDigest – solution



```
public static void main(String args[]) throws Exception {

    InputStream is01 = new URL("http://www.fi.muni.cz/~xklinec/java/file_a.bin").openStream();
    byte[] buffer = new byte[1024];

    MessageDigest md5 = MessageDigest.getInstance("MD5");
    MessageDigest sha = MessageDigest.getInstance("SHA-256");

    int bytesRead = -1;
    while ((bytesRead = is01.read(buffer)) >= 0){
        md5.update(buffer, 0, bytesRead);
        sha.update(buffer, 0, bytesRead);
    }

    System.out.println(Globals.bytesToHex(md5.digest(), false));
    System.out.println(Globals.bytesToHex(sha.digest(), false));
```

Task03 - Cipher

3. AES Encryption

Decrypt the following ciphertexts. Cipher used: AES/CBC/PKCS5Padding

Question: How many bits does IV have? How many bits does key have?

IV	KEY	Ciphertext
AAAAAAAAAAAAAAAAAAAAAA==	AAAAAAAAAAAAAAAAAAAAAA==	6VMSY9xFduwNsiyn8mGZdLG6/NXb3ziw81MBSfaKozs=
FiikDkkW+k+oW2biRnC1zQ==	eUaq9at/s29swOs5EEWv8Q==	vDoRZgpnJ2/yCnW7ogatKoBlR3XBsViSz5Dfj2ExLl8=
tPIljLHaDSa8vXwrnDZiCg==	0y4bBloL0Ppbuy3o8AK6Vw==	N2HNL2GCfEahFJ+9ieUuKzns4zp10nsWqN3SKN5s0x1uOn2BNn1s7bkqbQuTSYLFf/ow3kUQL

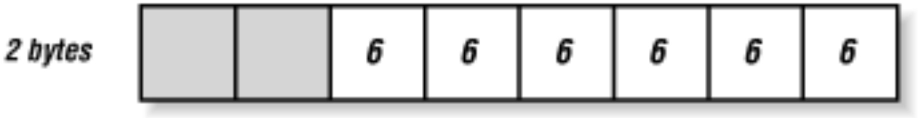
Hint1: You may use javax.xml.bind.DatatypeConverter class for Base64 encoding & decoding. Java 10+ may need JVM param: *--add-modules java.xml.bind*

Hint2: To construct AES encryption key you may use: new SecretKeySpec(key0, "AES");

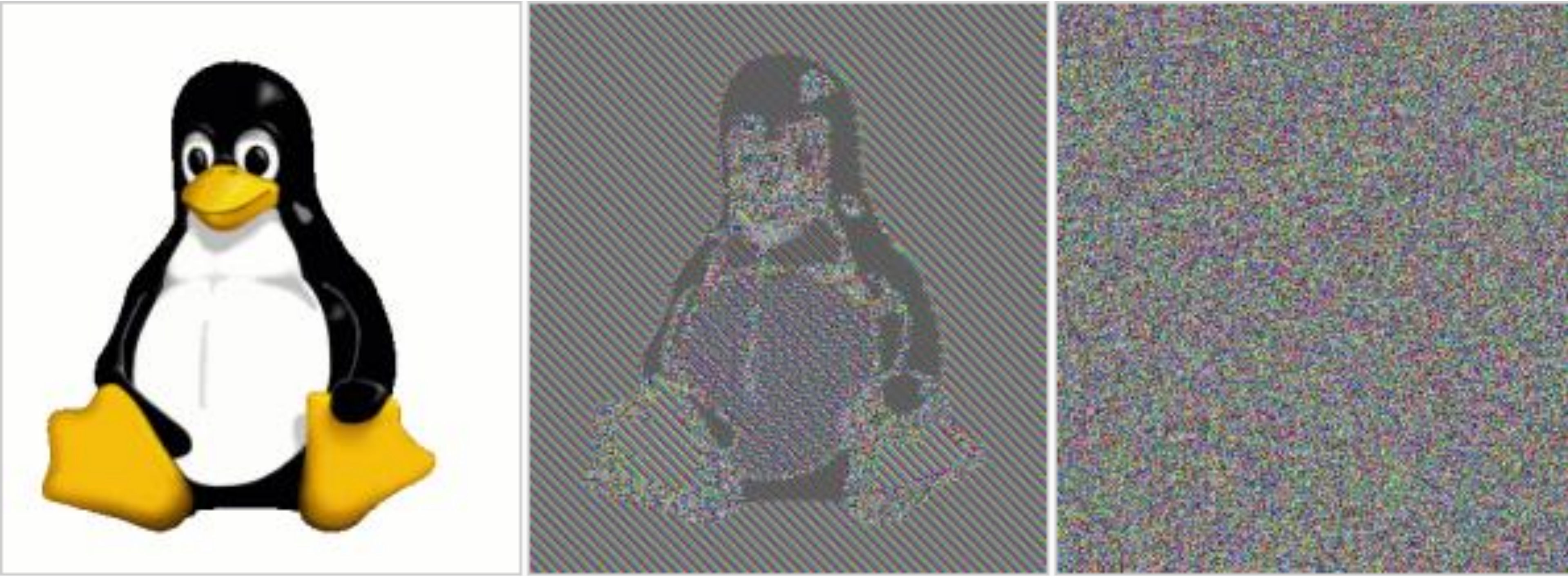
Hint2: To construct IV you may use: new IvParameterSpec(iv0);

Task03 - Cipher

- getInstance(“**algorithm/mode/padding**”);
  - Default mode: ECB
  - Default padding: PKCS5



Cipher



Cipher

- init(mode, key, algorithmParameterSpec)
  - Cipher.DECRYPT_MODE
  - **new** SecretKeySpec(aesKey, **"AES"**)
  - **new** IvParameterSpec(iv)

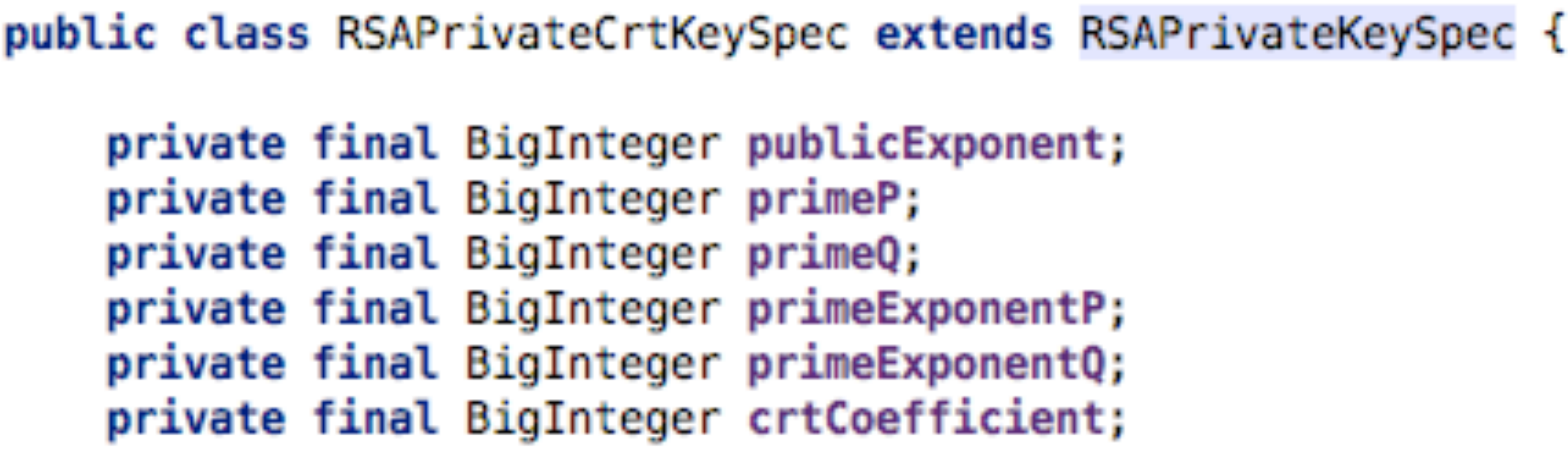
Cipher – Key vs KeySpec

- Key – opaque key, used in engine
  - getAlgoritm(), getEncoded()
- KeySpec – key specification, transport & storage
  - getP(), getQ(), getN()

Cipher – Key vs KeySpec

- SecretKeySpec = Spec & Key in the same time

Cipher – Key vs KeySpec



```
public class RSAPrivateCrtKeySpec extends RSAPrivateKeySpec {

    private final BigInteger publicExponent;
    private final BigInteger primeP;
    private final BigInteger primeQ;
    private final BigInteger primeExponentP;
    private final BigInteger primeExponentQ;
    private final BigInteger crtCoefficient;
```

Cipher – Key vs KeySpec

- Why separated?

Cipher – Key vs KeySpec

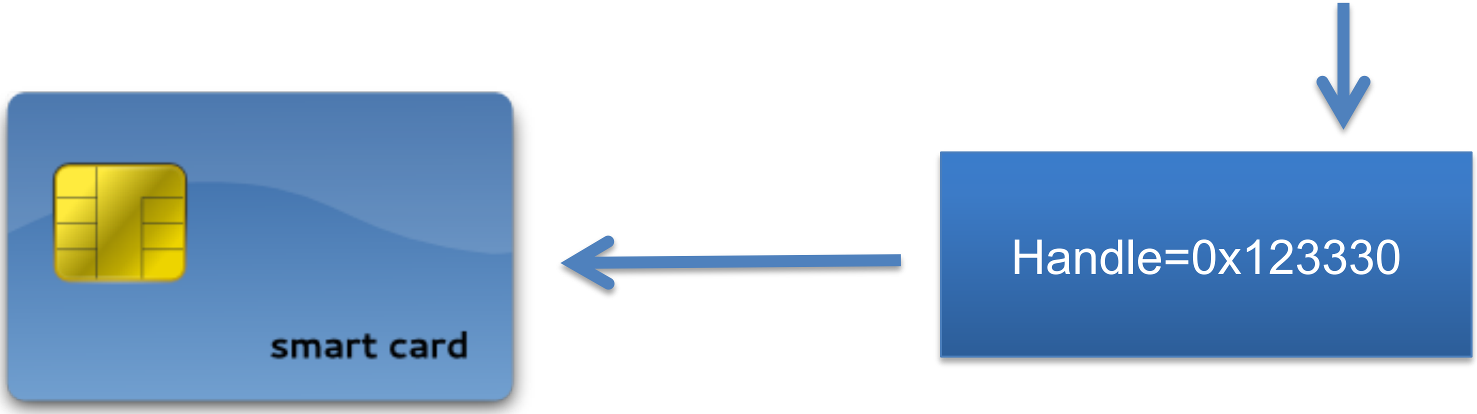
- Why separated?

```
Cipher.init(Cipher.DECRYPT_MODE, key)
```

Cipher – Key vs KeySpec

- Why separated?

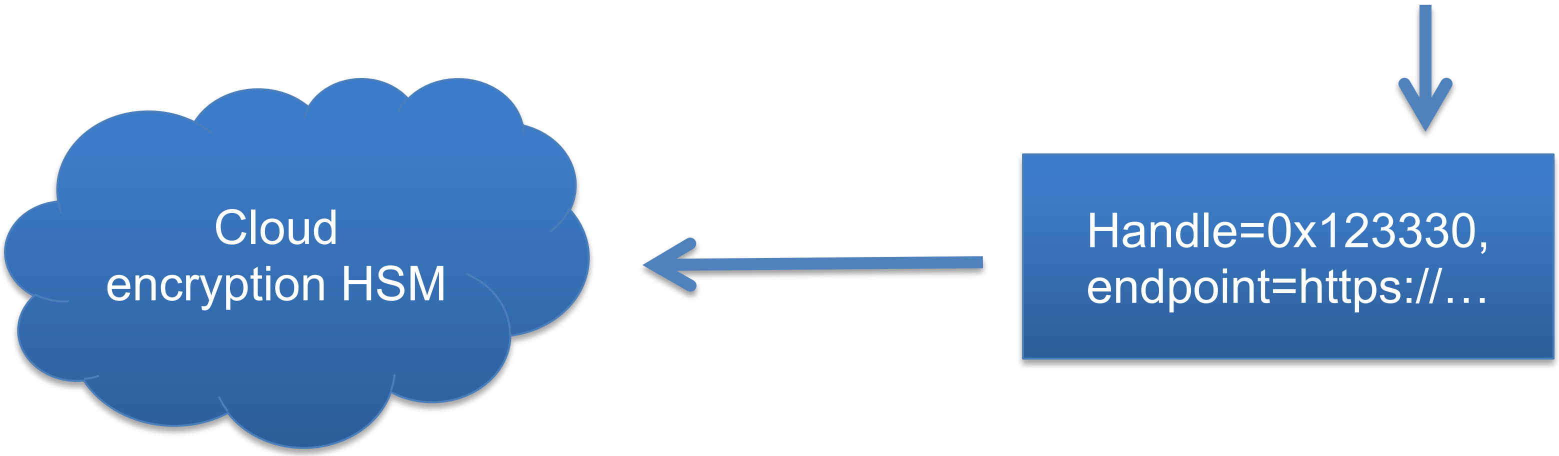
Cipher.init(Cipher.DECRYPT_MODE, key)



Cipher – Key vs KeySpec

- Why separated?

Cipher.init(Cipher.DECRYPT_MODE, key)



Cloud encryption HSM

Handle=0x123330, endpoint=https://…

Cipher – Key materials

- String vs. char[]
  - String is immutable, cannot zero out
- Zero-out mutable byte[] after use to prevent key leakage to swap files (or Heartblead)

Cipher – Key materials

- GC deallocates but does not zero-out – key still there
- Modern GC can copy, reorder mem (heap defrag), unable to properly delete keys from memory nowadays (Java does not specify behaviour, can differ).

Key Factories

- KeySpec → Key
- Key → KeySpec

- KeyFactory – asymetric keys
- SecretKeyFactory – symmetric keys

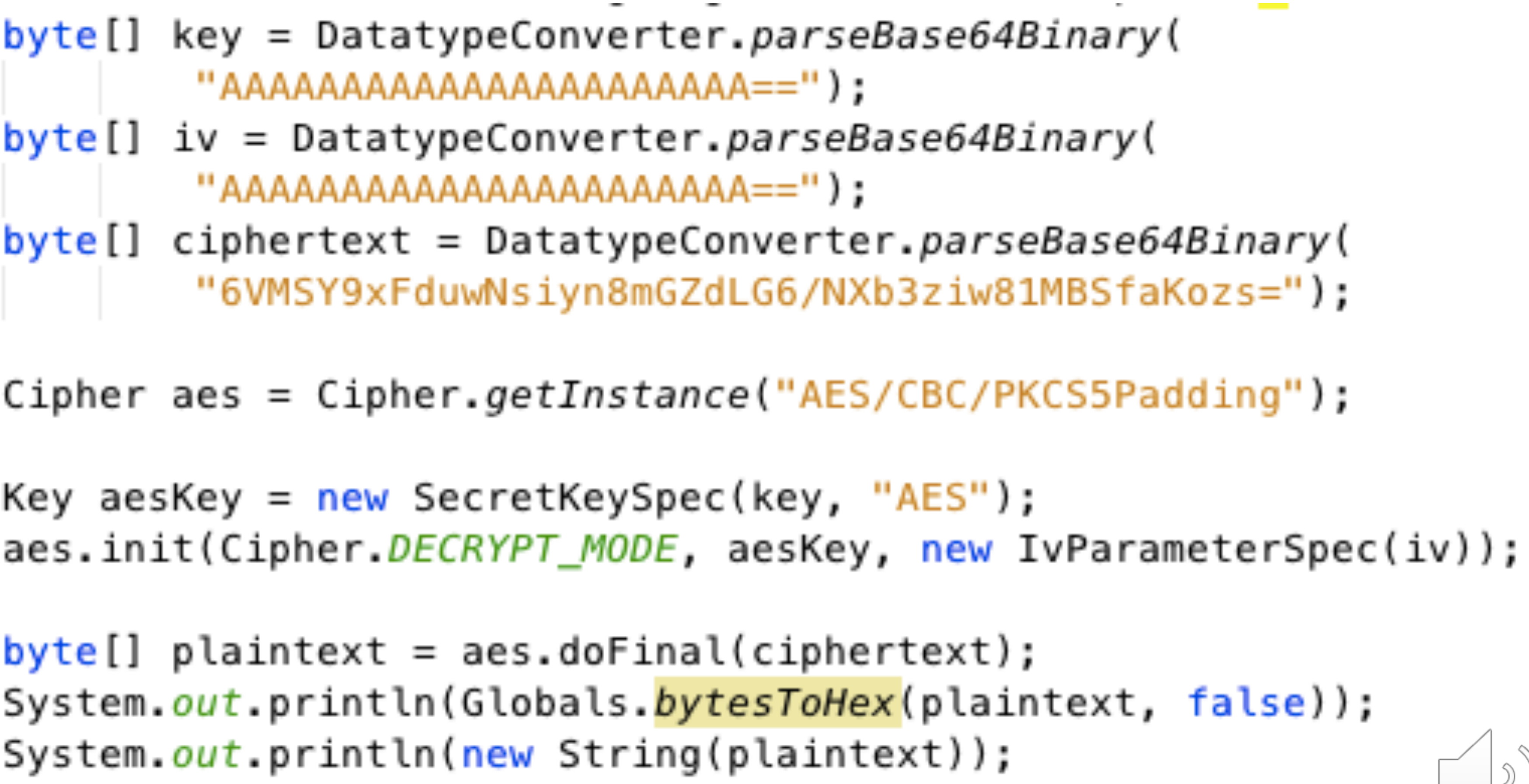
Key generators

- KeyGenerator – symmetric
  - generateSecret() → SecretKey
- KeyPairGenerator – asymmetric
  - generateKeyPair() → KeyPair

Task 03

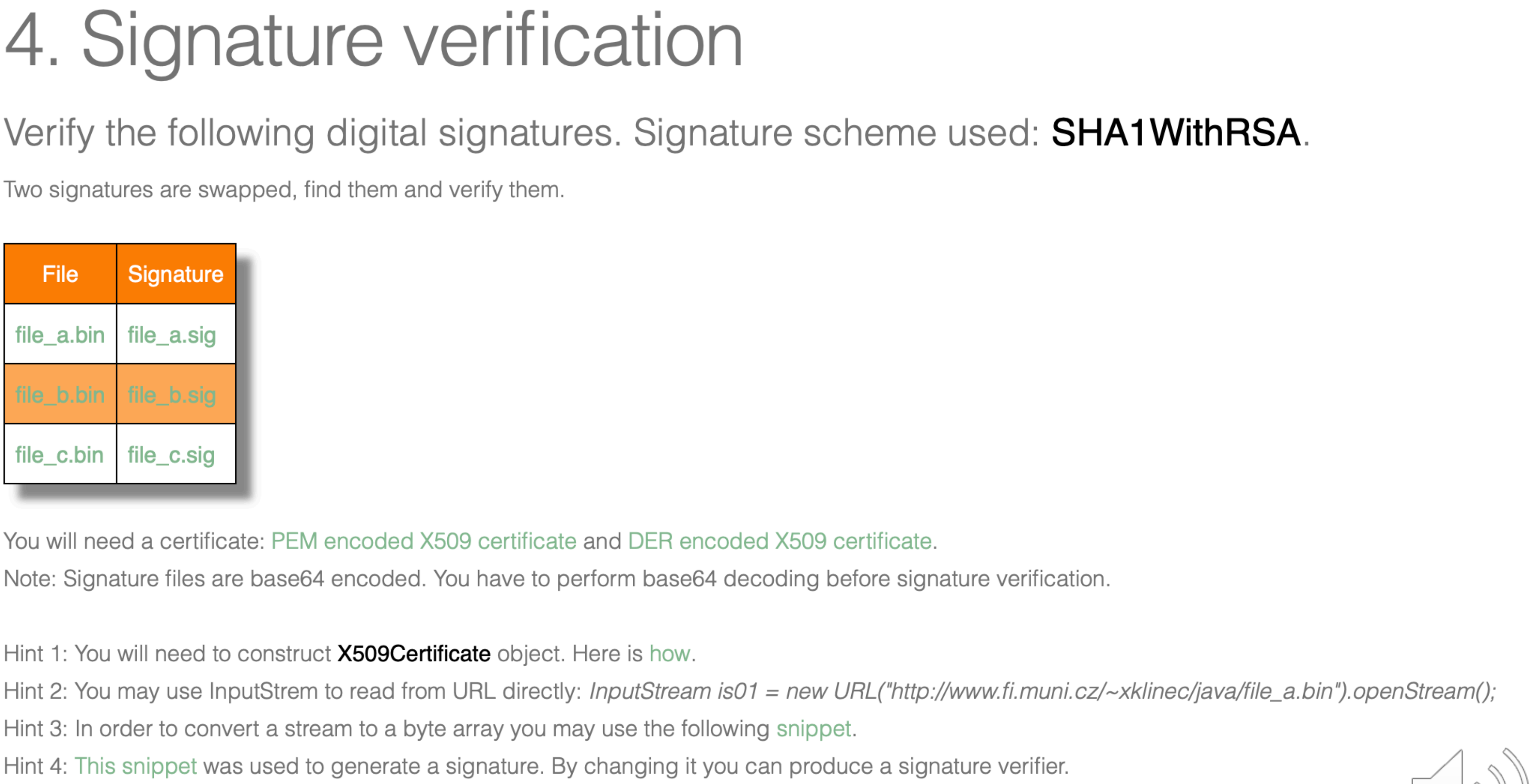
- You can work now

Cipher – Solution



```
byte[] key = DatatypeConverter.parseBase64Binary(
        "AAAAAAAAAAAAAAAAAAAAAA==");
byte[] iv = DatatypeConverter.parseBase64Binary(
        "AAAAAAAAAAAAAAAAAAAAAA==");
byte[] ciphertext = DatatypeConverter.parseBase64Binary(
        "6VMSY9xFduwNsiyn8mGZdLG6/NXb3ziw81MBSfaKozs=");

Cipher aes = Cipher.getInstance("AES/CBC/PKCS5Padding");

Key aesKey = new SecretKeySpec(key, "AES");
aes.init(Cipher.DECRYPT_MODE, aesKey, new IvParameterSpec(iv));

byte[] plaintext = aes.doFinal(ciphertext);
System.out.println(Globals.bytesToHex(plaintext, false));
System.out.println(new String(plaintext));
```

Task04 - Signature



4. Signature verification

Verify the following digital signatures. Signature scheme used: **SHA1WithRSA**.

Two signatures are swapped, find them and verify them.

File	Signature
file_a.bin	file_a.sig
file_b.bin	file_b.sig
file_c.bin	file_c.sig

You will need a certificate: PEM encoded X509 certificate and DER encoded X509 certificate.

Note: Signature files are base64 encoded. You have to perform base64 decoding before signature verification.

Hint 1: You will need to construct **X509Certificate** object. Here is how.

Hint 2: You may use InputStrem to read from URL directly: *InputStream is01 = new URL("http://www.fi.muni.cz/~xklinec/java/file_a.bin").openStream();*

Hint 3: In order to convert a stream to a byte array you may use the following snippet.

Hint 4: This snippet was used to generate a signature. By changing it you can produce a signature verifier.

Certificate Builder

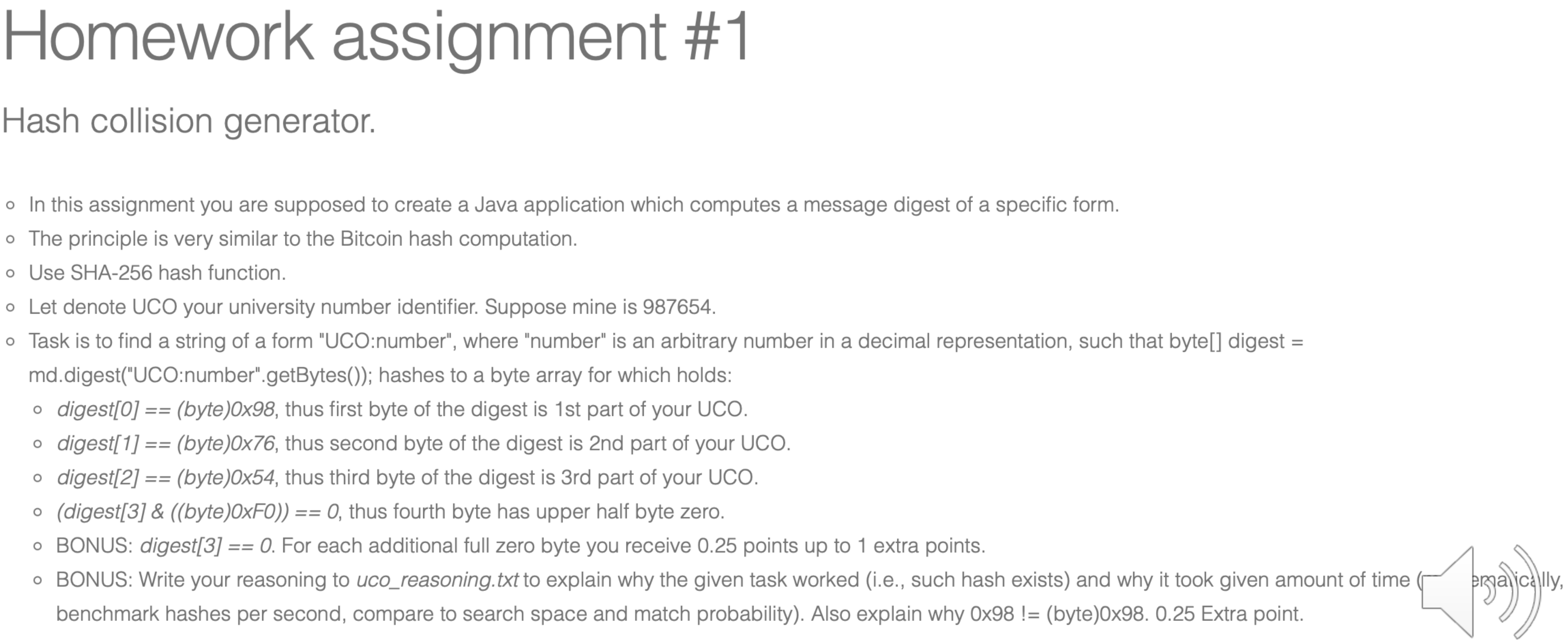
- X509V3CertificateGenerator
- goo.gl/I9WLUD

Diffie Hellman

- KeyPairGenerator
- KeyAgreement
- goo.gl/Lus40Y

Homework

Check study materials for the assignment.



Homework assignment #1

Hash collision generator.

- In this assignment you are supposed to create a Java application which computes a message digest of a specific form.
- The principle is very similar to the Bitcoin hash computation.
- Use SHA-256 hash function.
- Let denote UCO your university number identifier. Suppose mine is 987654.
- Task is to find a string of a form "UCO:number", where "number" is an arbitrary number in a decimal representation, such that byte[] digest = md.digest("UCO:number".getBytes()); hashes to a byte array for which holds:
  - *digest[0] == (byte)0x98*, thus first byte of the digest is 1st part of your UCO.
  - *digest[1] == (byte)0x76*, thus second byte of the digest is 2nd part of your UCO.
  - *digest[2] == (byte)0x54*, thus third byte of the digest is 3rd part of your UCO.
  - *(digest[3] & ((byte)0xF0)) == 0*, thus fourth byte has upper half byte zero.
  - BONUS: *digest[3] == 0*. For each additional full zero byte you receive 0.25 points up to 1 extra points.
  - BONUS: Write your reasoning to *uco_reasoning.txt* to explain why the given task worked (i.e., such hash exists) and why it took given amount of time (mathematically, benchmark hashes per second, compare to search space and match probability). Also explain why 0x98 != (byte)0x98. 0.25 Extra point.

Thank you for your attention!

Questions