What every Java developer needs to know about

cryptography

Angelo van der Sijpt
angelo.vandersijpt@luminis.eu
@_angelos

academy.luminis.eu
academy@luminis.eu
luminis
Conversing worlds

Representations

abc	UTF-8
0110001 01100010 01100011	Binary
YWJj	Base64
61 62 63	Hexadecimal

Base64 is the most common format for transferring key- and certificate material because of its hardening against text-based mangling.
PEM files contain armored Base64, guarded by -----BEGIN----- and -----END-----

Hashing

CRC32	Checksum	Quick, stable
SHA1, SHA2	Cryptographic hash	Tamper-resistant, prevents collisions
HMAC	Message auth code	Authenticated, allows verifying sender
PBKDF2	Password key deriv	Turn passwords into encryption keys
SCRYPT	Password hashing	PW hashing with salt & hardness for storage

Choose a hash suitable to your needs: properties, time- and memory-complexity, lifetime of the result.

Symmetric crypto

AES (RIjndael)
Is a block cipher.
Block ciphers have a fixed block size, and need
- padding
- operation mode

to work on streams.

Operation modes
ECB encrypts block independently.
CBC mixes in previous ciphertext, needs init vector

```java
// generate or import key
Key key = new SecretKeySpec(new byte[16], "AES");

// Set up cipher and data; provide algorithm/mode/padding
Cipher cipher = Cipher.getInstance("AES/CBC/PKCS5Padding");
byte[] data = "input".getBytes("UTF-8");

// Set up initialization vector
IvParameterSpec iv = new IvParameterSpec(new byte[16]);

cipher.init(Cipher.ENCRYPT_MODE, key, iv);
byte[] encrypted = cipher.doFinal(data);
// use update() for more data

cipher.init(Cipher.DECRYPT_MODE, key, iv);
byte[] decrypted = cipher.doFinal(encrypted);
```

Asymmetric crypto

RSA
Asymmetric crypto mechanism: one key of pair can encrypt, other decrypts.
Simple math, but expensive.

```java
KeyPairGenerator kpg = KeyPairGenerator.getInstance("RSA");
keyPair = kpg.generateKeyPair();
byte[] data = "input".getBytes("UTF-8");
// Set up cipher for encryption
Cipher rsa = Cipher.getInstance("RSA");
rsa.init(Cipher.ENCRYPT_MODE, keyPair.getPublic());
byte[] encrypted = rsa.doFinal(data);
// Set up cipher for decryption
rsa.init(Cipher.DECRYPT_MODE, keyPair.getPrivate());
byte[] decrypted = rsa.doFinal(encrypted);
```

Resources

https://github.com/angelos/javacrypto
https://www.cs.auckland.ac.nz/~pgut001/dumpasn1.c
https://www.grc.com/miscfiles/SChannel_Cipher_Suites.txt

TLS

Generate private key & certificate

Use OpenSSL to generate your private key, and a certificate with your site's properties in it.
(This is self-signed, for development only)

```
openssl req \
-x509 \
-sha256 \
-newkey rsa:2048 \
-keyout private.key \
-out certificate.cer \
-subj "/C=NL/O=<company>/OU=<dept>/CN=<domain>" \
-nodes
```

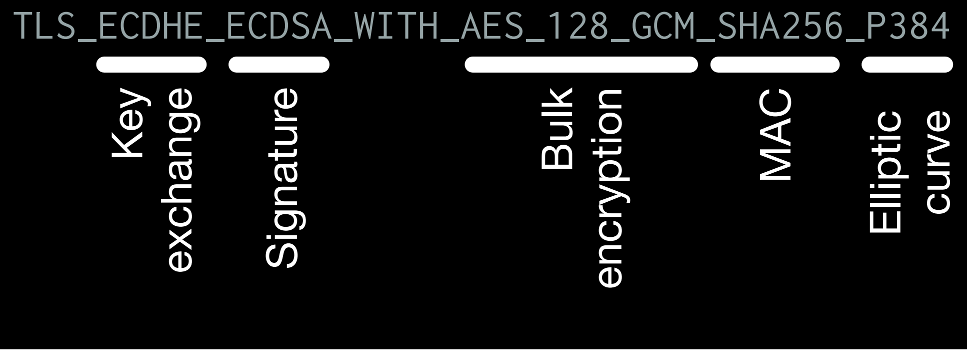
Build keystore

Make the key and certificate into a P12 that Java can use.

```
openssl pkcs12 \
-export \
-nodes \
-out keystore.p12 \
-inkey private.key \
-in certificate.cer \
-passin pass:<pass> \
-passout pass:<pass>
```

Run application

```
java \
-Djavax.net.ssl.keyStore=keystore.p12 \
-Djavax.net.ssl.keyStorePassword=<pass> \
-Djavax.net.debug=all \
<main class>
```



TLS_ECDHE_ECDSA_WITH_AES_128_GCM_SHA256_P384

Caveat

To play in the "real" world, you will
a) need a real, signed certificate. Your CA can usually help you generate one.
b) need to make sure your (private) key material stays safe; probably an application server which supports different configurations.
c) never email key material with its password; text the pw in stead.

Java Crypto Developer no-foot-shooting-pledge

I promise I will never, ever, try to implement my own crypto system for any other purpose than to learn how one works.