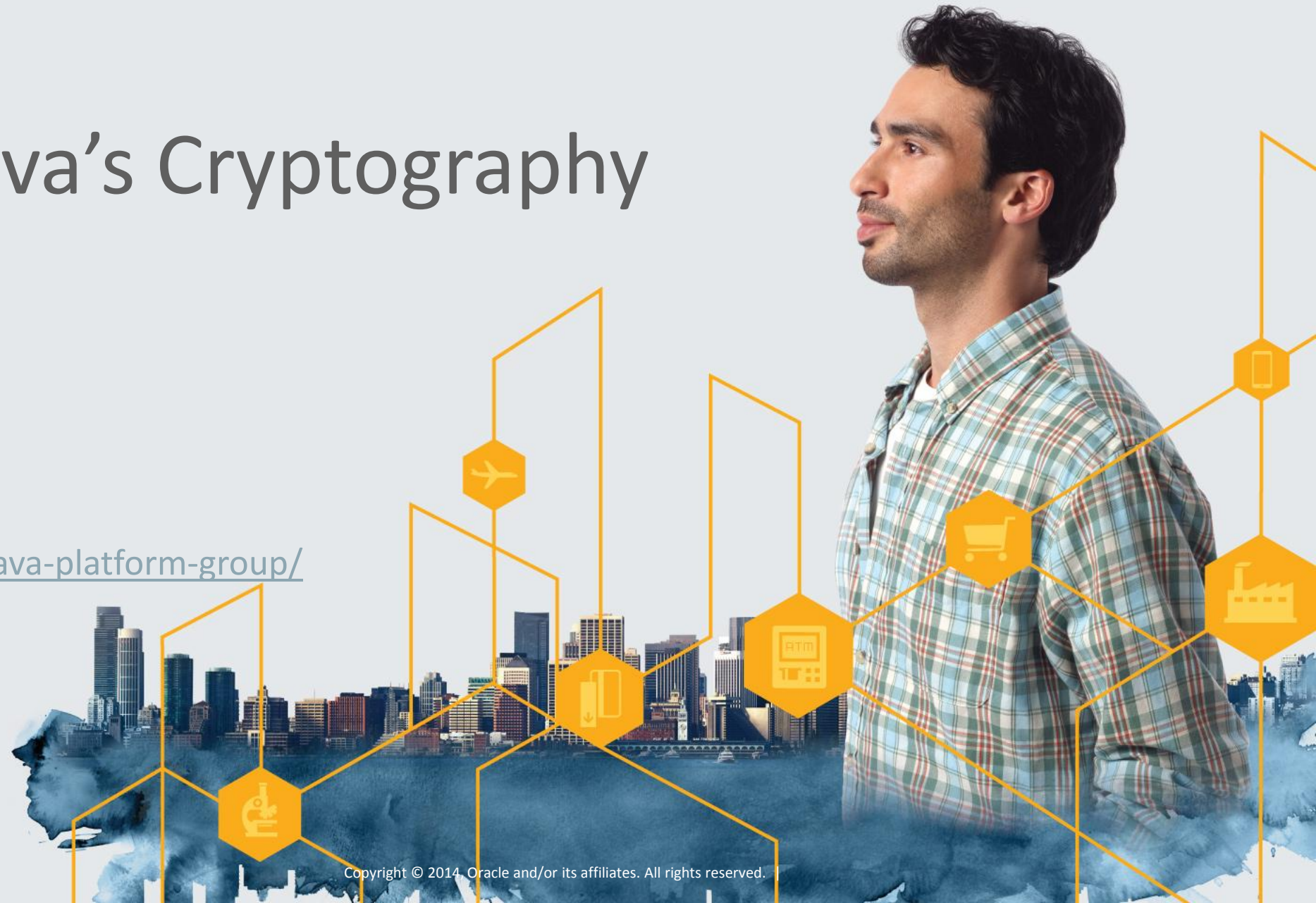


Applying Java's Cryptography

Erik Costlow
Product Manager
Java Platform Group
September 2014

Twitter @costlow
<https://blogs.oracle.com/java-platform-group/>



Safe Harbor Statement

The following is intended to outline our general product direction. It is intended for information purposes only, and may not be incorporated into any contract. It is not a commitment to deliver any material, code, or functionality, and should not be relied upon in making purchasing decisions. The development, release, and timing of any features or functionality described for Oracle's products remains at the sole discretion of Oracle.

Erik Costlow

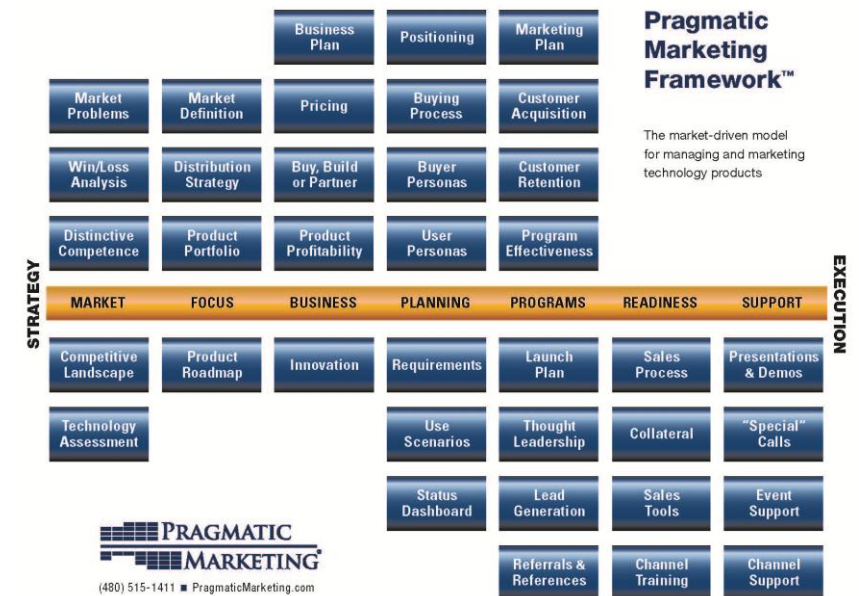
Product Manager

Java Architecture/Development/Deployment.
Software Security:

- Threat modeling
- Program analysis

Technical reading & writing.
Listening.

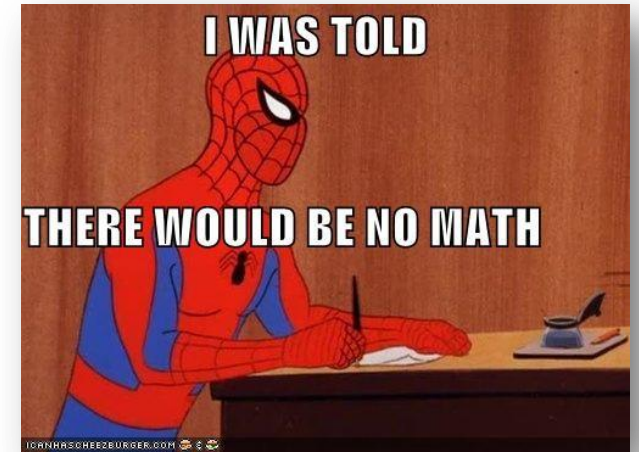
<https://blogs.oracle.com/java-platform-group/>



Introductory Cryptography Talk

Intended for Junior to Mid Developers / System Administrators

- Learn what types of cryptography do/don't protect things.
- Learn to use that cryptography.
- Learn to tune Java applications / servers.
- Favor clarity over complete accuracy (without being inaccurate).



Why cryptography?

IEEE Center for Secure Design: “Use Cryptography Correctly.”

- When used incorrectly, it can be circumvented.
- Often circumvented, rarely broken.

Too easy to do incorrectly:

PKCS 10, 11, 12 are not “newer versions.”

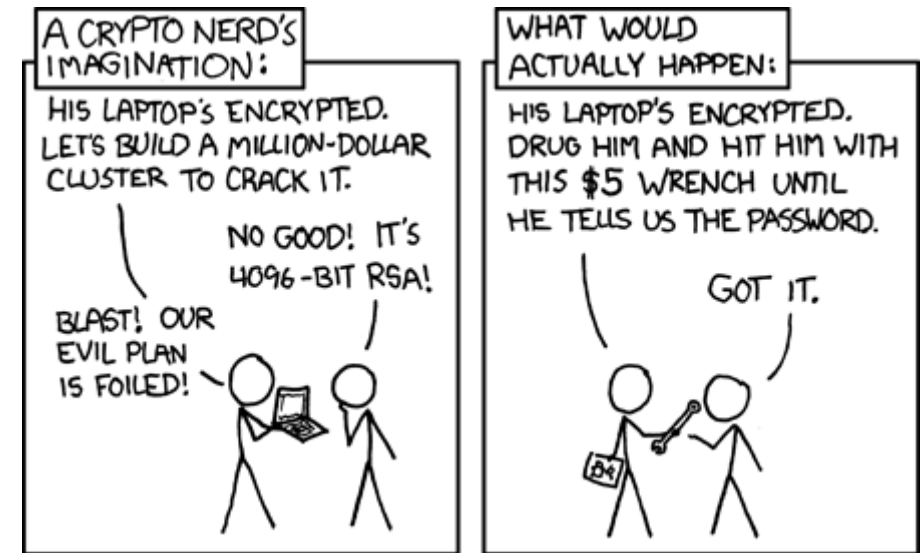
Naming doesn't fit well:

SHA-1 = SHA

SHA-2 = SHA-256, SHA-384, SHA-512

Bit strength not always meaningful.

“Correct” may differ based on goal.



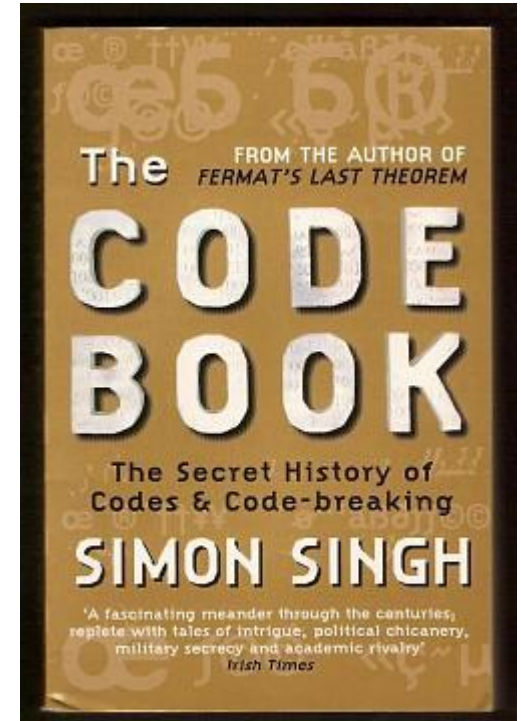
XKCD 538

Agenda

- 1 ➤ Background: security & cryptography.
- 2 ➤ Encryption in motion & at rest.
- 3 ➤ Using existing cryptography APIs in applications.
- 4 ➤ Recap.

Purpose of Cryptography

1. Communicate secrets.
2. Authenticate information.
3. Store sensitive information.



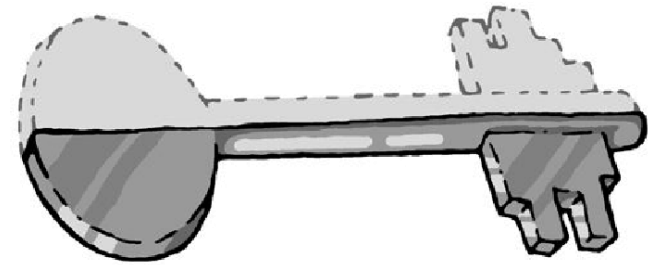
Things cryptography does not do



Secure applications against hacking.
Ensure you use it correctly with ease.
Protect unencrypted details, like the
encryption's key or password.

Types of cryptography

1. Communicate secrets. **“Public Key”**
 - Examples: RSA, EC.
2. Authenticate information. **“Hashing algorithms”**
 - Examples: SHA256, SHA1, MD5
3. Store sensitive information. **“Public Key”** or **“Symmetric Key”**
 - Examples: AES, 3DES



Most cryptographic systems
involve all three.

Cryptography in Java

Available since JDK 1.1 (1997)

- Java Cryptography Architecture
 - In packages `java.security` and `javax.crypto`
 - <http://docs.oracle.com/javase/8/docs/technotes/guides/security/crypto/CryptoSpec.html>
- Java Secure Socket Extensions
 - Customizable SSL/TLS. For HTTPS and others.
- Swappable implementation.
- Lots of common algorithms.



Do not make your own algorithms.
Do use existing algorithms or providers.

java.security.cert.Certificate

- The one in “javax.security.cert” is compatibility from Feb 2002; please avoid.
- The industry uses **X509Certificate** types. Use instanceof and typecasting.
- SSL Certificates cannot sign code.

X509Certificate.getKeyUsage() is boolean[8]

0	Digital Signatures
5	Sign other certificates

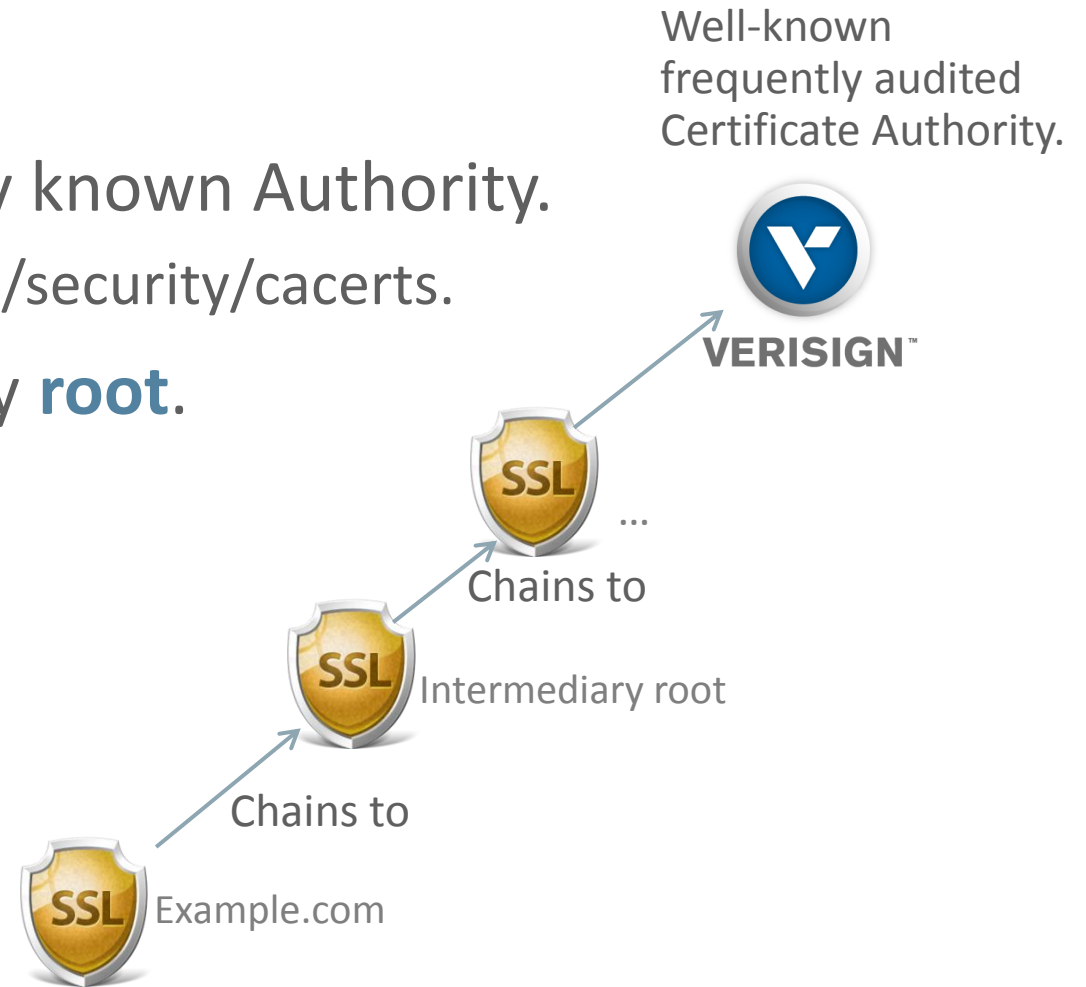
X509Certificate.getExtendedKeyUsage() is Strings

1.3.6.1.5.5.7.3.1	TLS Web Server Authentication
1.3.6.1.5.5.7.3.3	Code signatures

Trusted Certificate Authorities enforce these.

Public Key systems: Certificates

- Public key, *private* key.
- Public key shared in advance or verified by known Authority.
 - Certificate Authority: About 80 in Oracle jre/lib/security/cacerts.
- Certificates **chain** to a Certificate Authority **root**.
 - Chain validated through “**hashing algorithms.**”



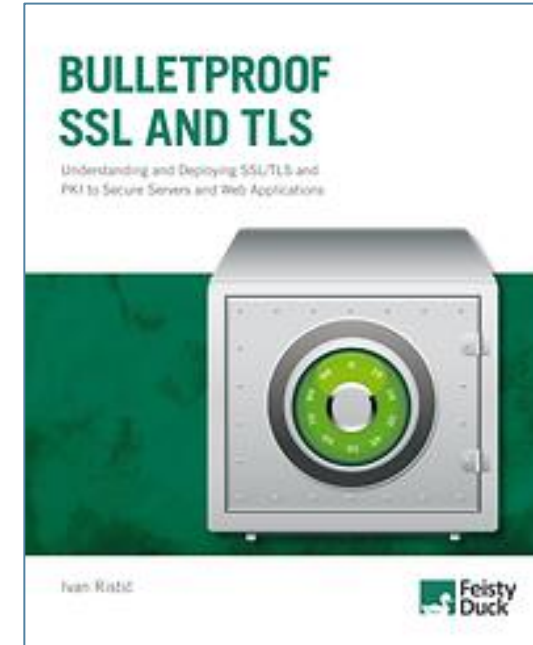
Agenda

- 1 Background: security & cryptography.
- 2 Encryption in motion & at rest.**
- 3 Using existing cryptography APIs in applications.
- 4 Recap.

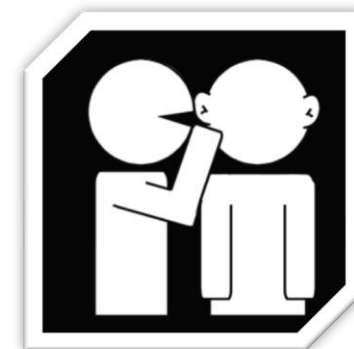
Java with SSL/TLS

HTTPS and others.

- Configure web servers & other systems.
- Java JSSE unrelated to OpenSSL.
 - Unrelated configurations.
 - Compatible algorithms and keys.
- SSL and TLS are the same thing.



Chapter 14 is dedicated to well-researched Java configurations.



Popular things to tune in SSL/TLS

Protocol version

- Choose newest protocol as the default.
- Remove older protocols and clients.

`-Dhttps.protocols=TLSv1.1,TLSv1.2`

Forward Secrecy

- Change ciphers.
- Usually configured in app server.
- Or in JVM:

`-Dhttps.cipherSuites=A,B,C...`

Unlimited Cryptography Extension

- Stronger versions of existing algorithms.
- Downloaded through Oracle Tech Network.
- JAR file placed in JRE.

Change TLS protocol

- Disabling older protocols removes:
 - Any attacks against the older protocol (BEAST, CRIME, etc).
 - Any legitimate clients needing that protocol.
- Configured through: `-Dhttps.protocols=TLSv1.1,TLSv1.2`

	JDK 8 (March 2014 to present)	JDK 7 (July 2011 to present)	JDK 6 (2006 to end of public updates 2013)*
TLS Protocols	TLSv1.2 (default) TLSv1.1 TLSv1 SSLv3	TLSv1.2 TLSv1.1 TLSv1 (default) SSLv3	TLSv1 (default) SSLv3

* End of public updates does not mean end of life.

Enable Forward Secrecy

- Future compromise does not reveal the past.
 - Became popular after large-scale spying scandals.
- Look for “DHE.”
- List of good algorithms in book, “**Bulletproof SSL/TLS**” page 439 that does security and compatibility.
- Configured through:
 - `Dhttps.cipherSuites=...`

Algorithm Names: Next slide explains how to read.

TLS_ECDHE_ECDSA_WITH_AES_128_GCM_SHA256

TLS_ECDHE_RSA_WITH_AES_256_GCM_SHA384

TLS_ECDHE_RSA_WITH_AES_256_CBC_SHA384

TLS_DHE_RSA_WITH_AES_128_GCM_SHA256

TLS_DHE_RSA_WITH_AES_256_GCM_SHA384

TLS_DHE_RSA_WITH_AES_128_CBC_SHA256

Etc.

Some need Unlimited Cryptography Extension.

Properties to tune:

<http://docs.oracle.com/javase/8/docs/technotes/guides/security/jsse/JSSERefGuide.html>

Names:

<http://docs.oracle.com/javase/8/docs/technotes/guides/security/StandardNames.html#ciphersuites>

Reading Algorithm Names

Usage

SSL means “named before the term TLS was chosen.” Otherwise no difference.

Symmetric speed-up after public keys

After validating public keys, use this to go faster.
AES better than DES. RSA or DSA are fine.

Hashing Algorithm

SHAs are newer. Higher means stronger. MD is not as strong.

TLS_ECDHE_ECDSA_WITH_AES_128_GCM_SHA256

Exchange

DHE means **ephemeral**, aka **forward secrecy**. Regular DH does not.
EC means Elliptical Curve, an optional algorithm choice.

Cipher Mode

GCM is newer than CBC, both are ok. Deals with blocks.

Personal opinion favoring clarity over complete accuracy.

<http://docs.oracle.com/javase/8/docs/technotes/guides/security/StandardNames.html#ciphersuites>

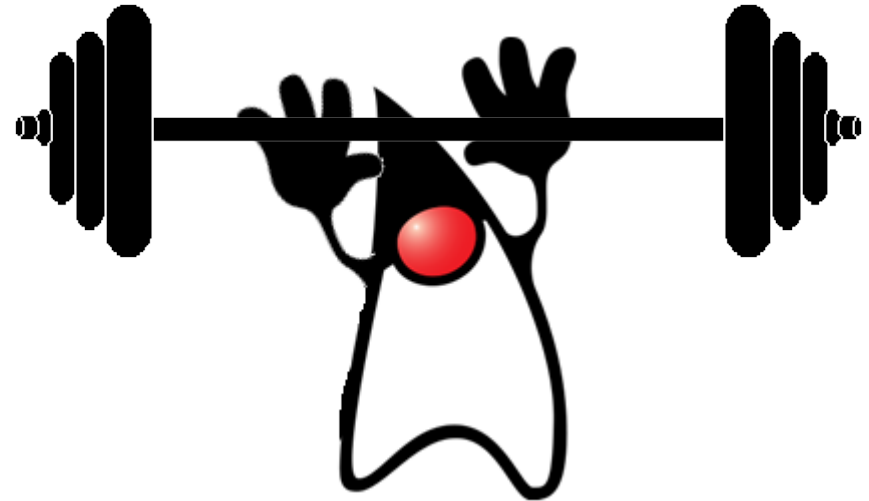
“More Secure” compared to “Defaults”



- Client chooses protocols and ciphers.
 - I just limit what the client can choose.
- More secure often means more CPU/Memory/Battery.
- Forcing ciphers that some clients can't use is like a Denial of Service.

Use Unlimited Cryptographic Extension

- Separate download in JDK 8, JDK 7, JDK 6, etc.
- Stronger versions of existing algorithms.
- Fixes some “why can’t *older_version* connect to *modern_service*?”
javax.net.ssl.SSLHandshakeException: Received fatal alert: handshake_failure

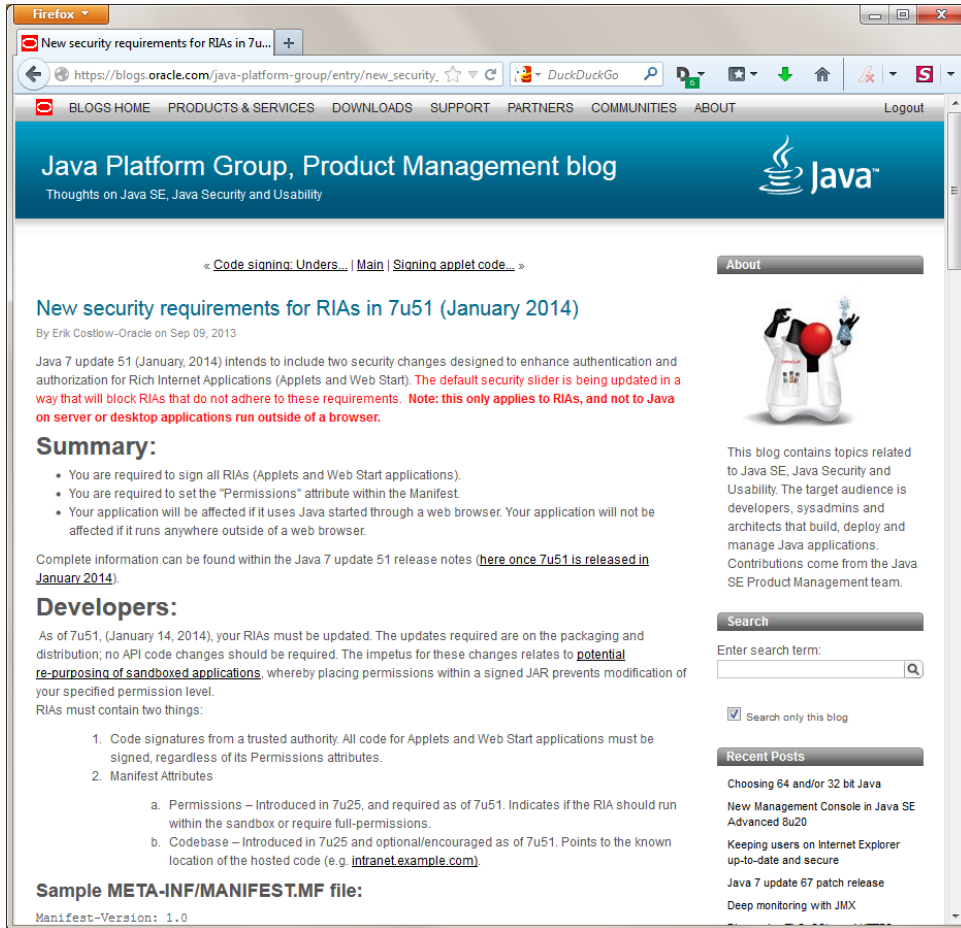


<http://www.oracle.com/technetwork/java/javase/downloads/jce8-download-2133166.html>
https://blogs.oracle.com/java-platform-group/entry/diagnosing_tls_ssl_and_https

Code signatures

- Same technology different purpose:
 - SSL answers “who sent this.”
 - Code addresses “who provided this.”
 - Does not mean authorship. Sign your dependencies too.
- JAR File allows any Java language:
 - Java, JRuby, Jython, Scala, etc.

Java 7 update 51 required signature for RIAs



- For Applet & Web Start applications only. Not for server-side.
- Lets user say “do I want to run this?”
- To avoid signing: Exception Site List.

Signature or no signature

No signature

- Verify file(s) against checksum.
 - If hash appears in a flat file, did someone change that file?
- If an ISV, did the customer roll their own patch?

Signed JAR file

- Every file in JAR has hash value.
 - So does the JAR itself. If changed, it will not verify.
- Easy to verify in support situations.

I sign my JAR files unless I have a reason not to.

Extend your signature through Timestamps (JDK 1.5+)

- Public key encryption requires that certificates expire.
- Most code signing certificates valid for 2 years. Audited industry practice.
- Timestamp extends lifetime.
 - JarSigner warns if you do not timestamp.

Date you sign.

Your certificate expires.

Timestamp Authority Expires.

`jarsigner ... -tsa tsa.starfieldtech.com`

Maybe 2 years.

Maybe 10 years.

Agenda

- 1 Background: security & cryptography
- 2 Encryption in motion & at rest
- 3 Using existing cryptography APIs in applications**
- 4 Recap



- Using existing providers and algorithms.
- Using Random or SecureRandom as appropriate.
- Using unpredictable salt/seed.



- Creating new encryption algorithms.
- Only using Random because it's called random.
- Re-using values over and over.

“I want to...”

- Verify that something hasn't been corrupted/modified
 - **Hashing**
- Encrypt something to be shared in uncontrolled environments
 - **Public Key**
- Encrypt something to be shared in controlled environments
 - **Secret Key**

Hashing Functions, aka MessageDigest

Used for: **Verifying things**

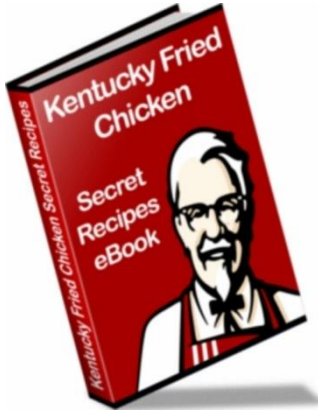
- Used for:
 - Validating files, checksums.
 - Comparing passwords without needing the password.
- One-way only. Cannot reverse.
 - Although it is possible to pre-compute hashes and string-compare.
- My preference: SHA256+, SHA1, MD5 only when needed.
- Names under “Message Digest” in documentation.

<http://docs.oracle.com/javase/8/docs/technotes/guides/security/StandardNames.html>



Salting Hashes

Salt – a new unpredictable byte[] that prevents pre-computing values.

	Predictable non-seeded hash	Unpredictable salted hash
Example:	Is this file corrupt or modified?	Is this password correct?
Why:	File hash is not a secret.	Password is a secret. Prevent “rainbow tables.”
Code:	<div></div> <pre>MessageDigest hash = MessageDigest.getInstance("SHA-512"); hash.update(uniqueSaltNotReUsedForOtherThings); //When something is a secret. hash.update(input.getBytes()); byte[] hashedBytes = hash.digest(); StringBuilder hexString = new StringBuilder(mdbytes.length); for (byte b : mdbytes) { hexString.append(Integer.toHexString(0xFF & b)); } hexString.toString().toUpperCase();</pre>	
Guidance:		Do not re-use salts. Keep them separate.

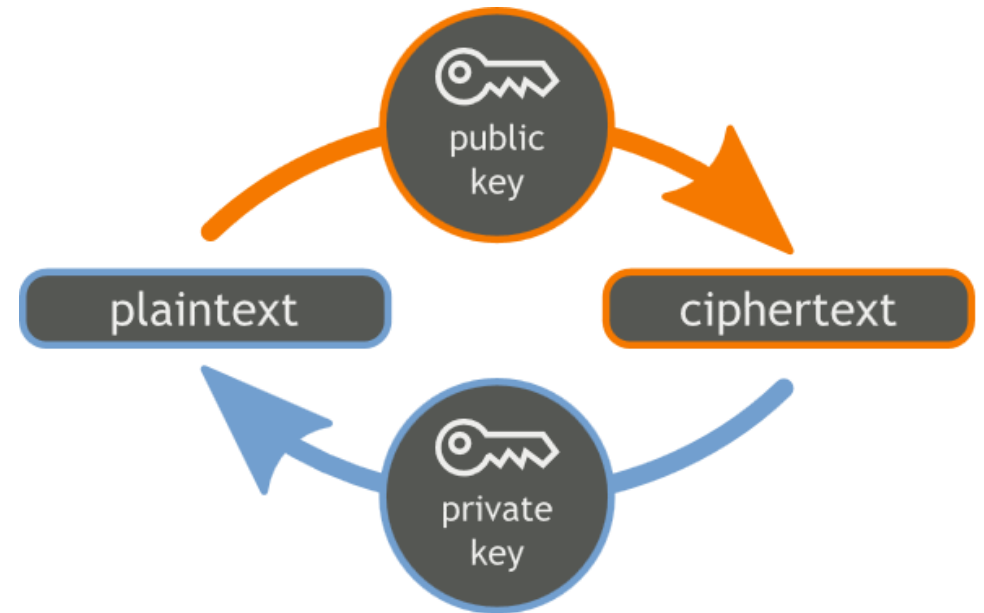
Public Key

Used for: **Encryption in uncontrolled environments**

- Public Key & Private Key go into **keystore**.
 - **Command-Line**: keytool
 - **Code**: java.security.KeyStore, KeyPair, PublicKey, PrivateKey.
- Share public keys. Do not share private keys.
- “KeyPairGenerator Algorithms”

<http://docs.oracle.com/javase/8/docs/technotes/guides/security/StandardNames.html>

– RSA or EC



Making and Sharing Keys

- Make a key:

```
keytool -genkeypair -alias BLAH -keyalg RSA -keysize 4096 -validity  
730 -keystore javakeystore_keepsecret.jks
```

KeyAlg numbers I use:

- RSA key increments:
 - 2048, 3072, 4096
- EC key increments:
 - 256, 384, 571

- Export a key for others:

```
keytool -exportcert -keystore javakeystore_keepsecret.jks -alias BLAH  
-file BLAH.cer
```

- Others can import:

```
keytool -importcert -keystore TheirNewStore.jks -alias BLAH -file  
BLAH.cer
```

*I generally do not make keys from code. Will explain why shortly...
Remember: Favoring clarity over complete accuracy.*

Example code of public key encryption

Access a KeyStore

To get access to your keys and certificates.

```
KeyStore ks = KeyStore.getInstance(KeyStore.getDefaultType());
try (InputStream fis = Files.newInputStream("keyStoreName")){
    ks.load(fis, password);
}
```

Encrypt or Decrypt Something

To secure or read something by others.

```
Cipher cipher = Cipher.getInstance("RSA"); //or EC, per StandardNames
cipher.init(Cipher.ENCRYPT_MODE, keyPair.getPublic()); //or DECRYPT_MODE with getPrivate
cipher.update(YOUR_BYTE[]);
final byte[] encrypted = cipher.doFinal();
```

Also CipherInputStream, CipherOutputStream.

Learn by unit test: Encrypt something then decrypt it. Every byte should be equal.

Key Management is the hard part



Easy to make keys and encrypt/decrypt.

Who should get private keys?

Encryption for others:

Whose keys do you need to use?

How many different people/keys?

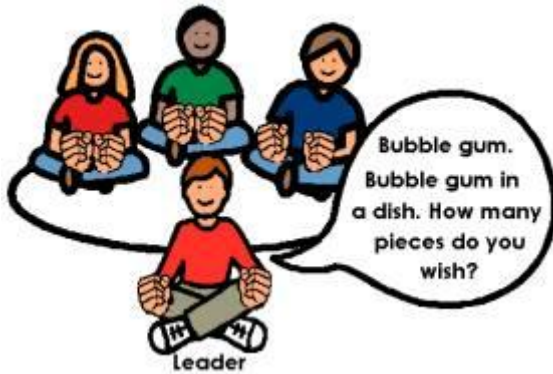
How do you authenticate their keys?

This is one reason why I try to avoid making a lot of keys.

Random or SecureRandom

Random

- For general use.
- About 10x speed or faster.
- Predictable when attacked in controllable environment.



SecureRandom

- For sensitive things.
- Uses slow-replenished resource.
 - May thread-block when it runs out. Some don't.
 - Hardware Random Number Generators improve speed.



<http://docs.oracle.com/javase/8/docs/technotes/guides/security/StandardNames.html#SecureRandom>

Agenda

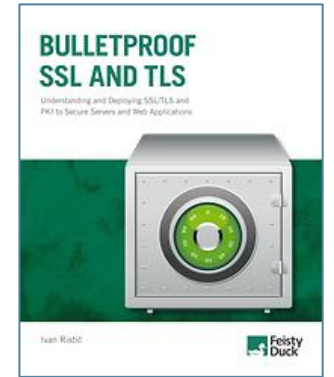
- 1 Background: security & cryptography
- 2 Encryption in motion & at rest
- 3 Using existing cryptography APIs in applications
- 4 **Recap**

Recap for cryptography

- SSL Certificates != Code Signing Certificates.
- Use existing algorithms.
- Avoid re-using salt values for hashes.
- If you want to copy code, copy from the actual crypto spec.
<http://docs.oracle.com/javase/8/docs/technotes/guides/security/crypto/CryptoSpec.html>
 - Try to avoid copying “I got it to work” posts.

Recap for web apps

Chapter 14



Protocol version

- Choose newest protocol as the default.
- Remove older protocols and clients.

```
-Dhttps.protocols=TLSv1.1,TLSv1.2
```

Forward Secrecy

- Change ciphers.
- Usually configured in app server.
- Or in JVM:

```
-Dhttps.cipherSuites=A,B,C...
```

Unlimited Cryptography Extension

- Stronger versions of existing algorithms.
- Downloaded through Oracle Tech Network.
- JAR file placed in JRE.

Questions?

Intended for Junior to Mid Developers / System Administrators

- Learn what types of cryptography do/don't protect things.
- Learn to use that cryptography.
- Learn to tune Java applications / servers.
- Favor clarity over complete accuracy (without being inaccurate).

Hardware and Software Engineered to Work Together



JavaOne™

ORACLE®