SSL implementieren – aber sicher!

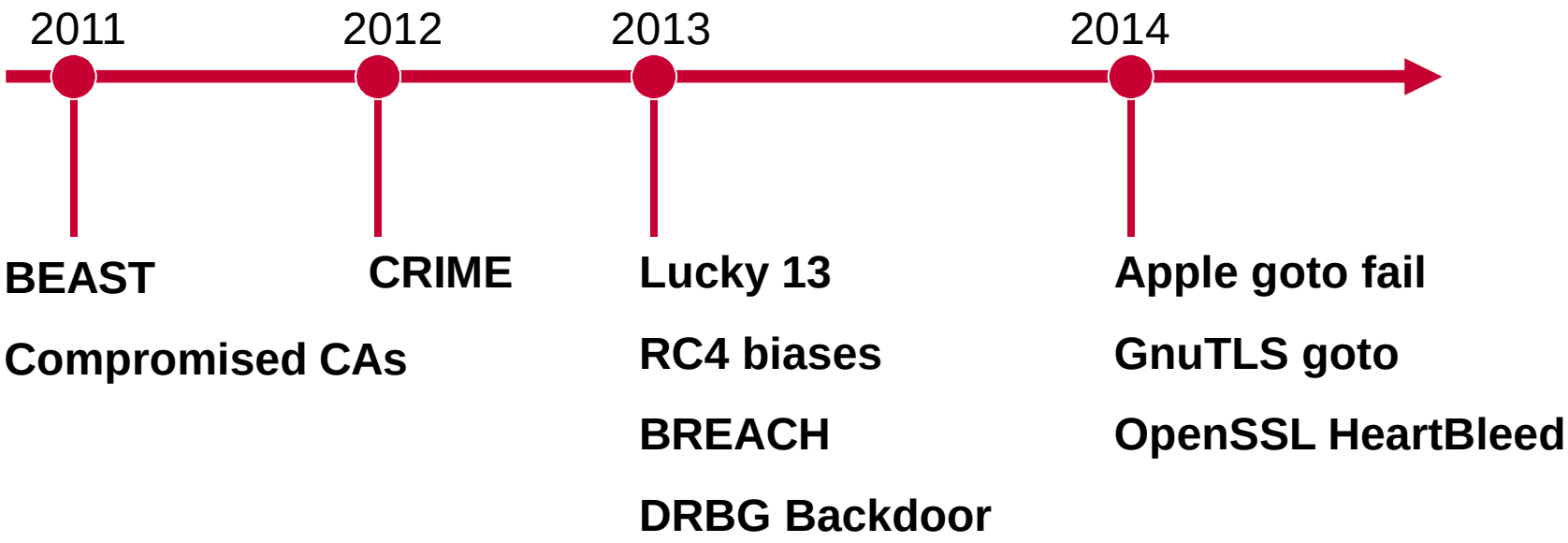
Karlsruher Entwicklertag 2014

21.05.2014

Dr. Yun Ding

secorvo
security consulting

SSL in the news



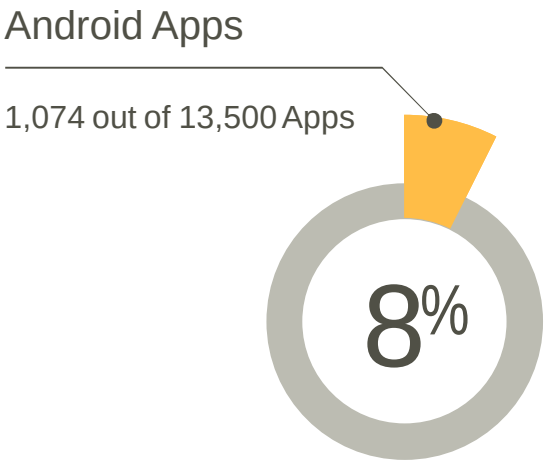
SSL in scientific publications

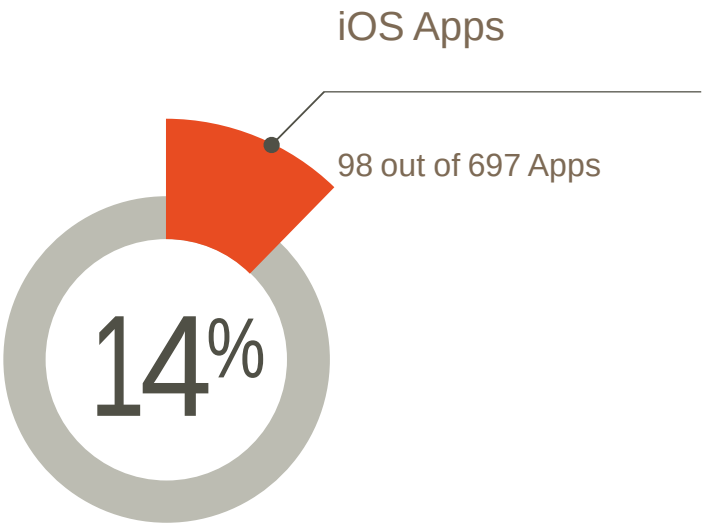
“The Most Dangerous Code in the World: Validating SSL Certificates …”

“Why Eve and Mallory Love Android: An Analysis of Android SSL (In)Security”

“Rethinking SSL Development in an Applied World”

Apps vulnerable to MITM





Layers of SSL-based applications

Human				
Application	Banking	Shopping	Messaging	Browser
Middleware/ Wrappers	Apache HttpClient	cURL	PhoneGap	MKNetworkKit
SSL Libraries	GnuTLS	Apple Secure Transport	OpenSSL	JSSE
SSL Protocols	Secure Protocols	Cipher Suites	Renegotiation	Compression
Cryptographic Primitives	Random Number Generators	Hash	Encryption	Authentication

How does SSL work?

Handshake: Key Exchange

Encrypted Data Communication

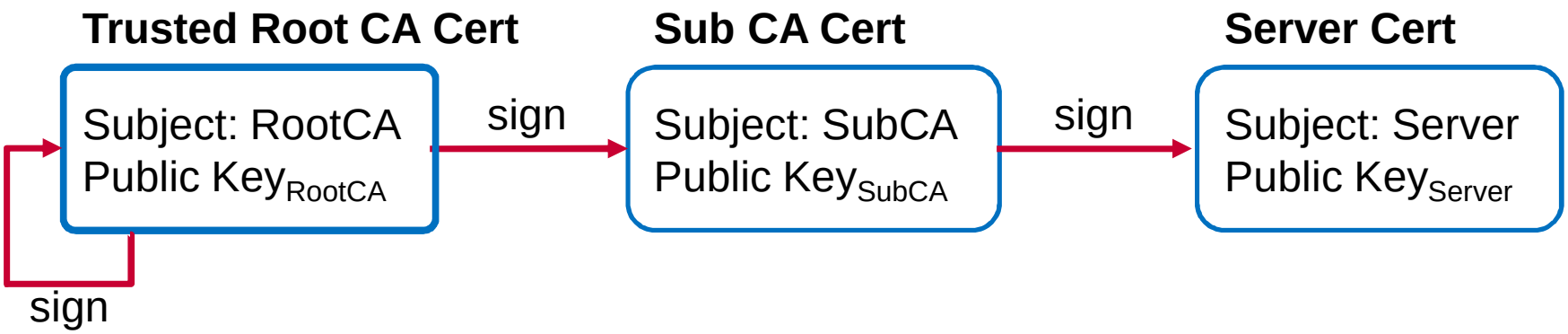
Copyright: http://openclipart.org/image/800px/svg_to_png/33457/Padlock-gold.png
gvictoria, bigstock.com, [2009] Joerg Habermeier, bigstock.com, Scanrail, bigstock.com

How does SSL break?

	Trick user *not* to encrypt	SSL stripping
	Predict the key	DRBG backdoor
	Trick user to use attacker's key	Apple goto fail, GnuTLS goto,MITM
	Trick server to expose keys	OpenSSL Heartbleed
	Perform cryptographic analysis to decrypt	RC4 biases, Lucky13, CRIME, BEAST, Breach

Copyright: © 2012 dny3d, bigstock.com, Scanrail, bigstock.com

SSL relies on Trust in Certificates ...



SSL relies on Valid Certificates

1. Make sure certificate validation is not turned off!
2. Verify the certificate is valid: not expired, not revoked
3. Validate “Chain of Trust”
4. Don’t accept self-signed certificates
5. Make sure hostname validation is set

What went wrong

- Insecure coding
  - Skipped or broken certificate validation
- Badly designed APIs
  - Expose low-level SSL protocol details, complex options
  - Complex relationship between return values and error status
  - Unsafe defaults (+missing warning in API Doc)
- Delegate responsibility to application developers

Default behavior in SSL lib. & wrappers

Libraries/Wrappers	Chain of Trust	Hostname Verification
OpenSSL	✔	✗
GnuTLS	✔	✔
CyaSSL	✔	✗
JSSE SSLSocketFactory HttpsURLConnection	 ✔ ✔	 ✗ ✔
Apache HttpClient 3.* HttpClient 4.*	 ✔ ✔	 ✗ ✔
Python ssl module	 ✔	 ✗

What went wrong

- Lack of understanding of how SSL works and breaks
- Misinterpretation of manifold SSL parameters & options
- Delegate responsibility to end users with warnings
- “Security gets in the way”

Human
Application
Middleware/ Wrappers
SSL Libraries
SSL Protocols
Cryptographic Primitives

When Security gets in the way ...

Override (secure) standard certificate validation

- disable or break certificate validation
- disabled in development & forget to remove in production

Customized Trust Manager in Java

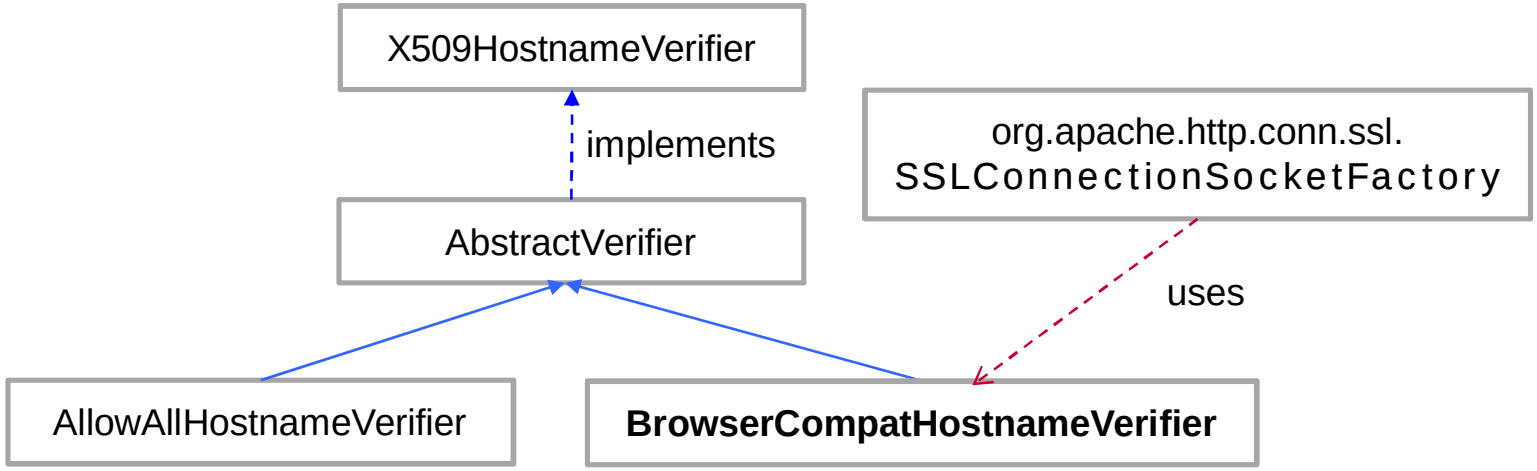
SSLTest.java | DisableValidationTrustManager.java

```java
import javax.net.ssl.*;
import java.security.cert.*;

public class DisableValidationTrustManager implements X509TrustManager {
    public DisableValidationTrustManager() {
    }

    public void checkServerTrusted(java.security.cert.X509Certificate[] pl, String p2) {
          System.out.println("I don't validate any certs!");
          return;
   }
```

SSLTest.java | DisableValidationTrustManager.java

```java
TrustManager tm[] = {new DisableValidationTrustManager()};
SSLContext context;
try {
  context = SSLContext.getInstance("TLS");
  context.init(null, tm, null);
} catch (NoSuchAlgorithmException e) {
  e.printStackTrace();
} catch (KeyManagementException e) {
      e.printStackTrace();
}
```

Hostname Verification in HttpClient (4.3)

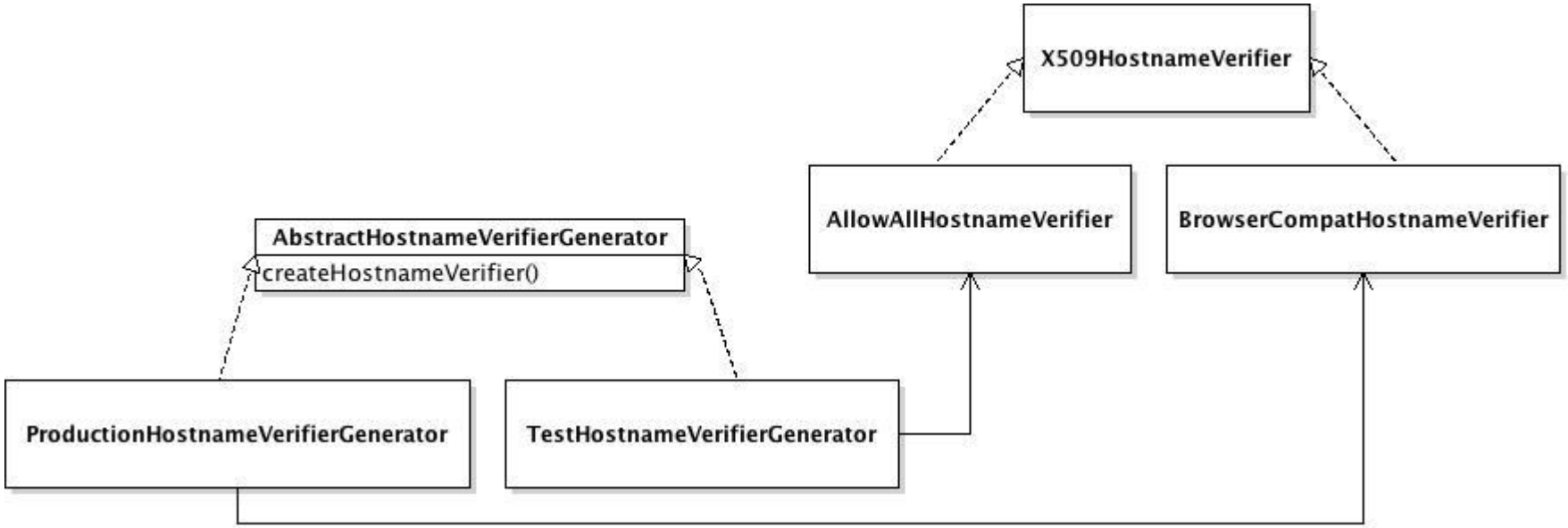


- Skip hostname verification: communicate with another host
- Customization to skip hostname verification

```
new SSLConnectionSocketFactory(sslContext, new AllowAllHostnameVerifier())
```

Decouple test and production code

- Don’t hardcode insecure certificate validation (and forget)
- Use best practices in software architecture for decoupling
  - Abstract Factory Design Pattern
  - Dependency Injection, configuration instead of programming



Customization for more Security!

- SSL Certificate or Public Key Pinning
  - Whitelist expected Certificates or Public Keys
  - Pre-existing binding between the server and its certificate/public key

Sample code available on OWASP
https://www.owasp.org/index.php/Certificate_and_Public_Key_Pinning#Examples_of_Pinning

Secure SSL configuration

- Use secure protocols: TLS v1.2, TLS v1.1, TLS v1.0
- Use secure cipher suites
  - Support authentication & encryption ≥ 128 bit Avoid
  - Use ECDHE for forward secrecy
  - Avoid anonymous DH, null cipher, RC4, 3DES
- RSA and DSA key must be ≥ 2048 bits
- Disable client-initiated Renegotiation
- Disable TLS compression

Secure SSL configuration

- Avoid mixed TLS and non-TLS content
- Secure cookies
- Deploy HTTP Strict Transport Security (HSTS)
- Prevent caching of sensitive content

Human
Application
Middleware/ Wrappers
SSL Libraries
SSL Protocols
Cryptographic Primitives

“SSL/TLS Deployment Best Practices” of Qualys SSL Labs
https://www.ssllabs.com/downloads/SSL_TLS_Deployment_Best_Practices_1.3.pdf

OWASP “Transport Layer Protection Cheat Sheet”
https://owasp.org/index.php/Transport_Layer_Protection_Cheat_Sheet

Test SSL

- Perform adversarial testing: abnormal certificates, MITM attacking tools (sslsniff, mitmproxy)
- Testing for SSL/TLS ciphers, protocols, keys and know vulnerabilities (e.g., BEAST, CRIME, Heartbleed)

Configuration

Protocols

TLS 1.2	No
TLS 1.1	No
TLS 1.0	Yes
SSL 3	Yes
SSL 2	No

Cipher Suites (SSL 3+ suites in server-preferred order, then SSL 2 suites where used)

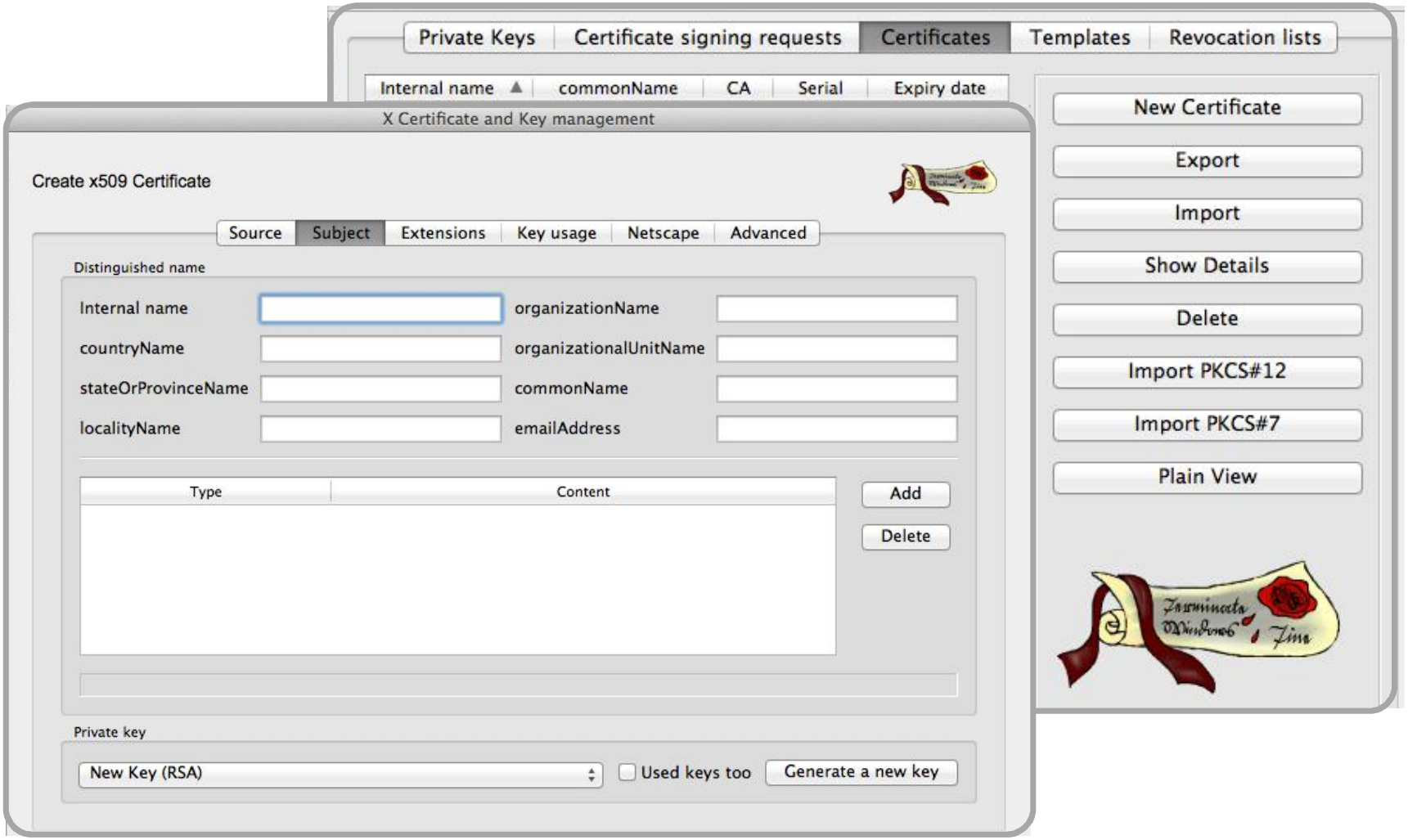
TLS_RSA_WITH_RC4_128_SHA (0x5)		128
TLS_RSA_WITH_RC4_128_MD5 (0x4)		128
TLS_ECDHE_RSA_WITH_AES_128_CBC_SHA (0xc013)	ECDH 256 bits (eq. 3072 bits RSA) FS	128
TLS_ECDHE_RSA_WITH_AES_256_CBC_SHA (0xc014)	ECDH 256 bits (eq. 3072 bits RSA) FS	256

- https://www.owasp.org/index.php/Testing_for_Weak_SSL/TSL_Ciphers,_Insufficient_Transport_Layer_Protection_%28OWASP-EN-002%29
- http://thoughtcrime.org/software/sslsniff/
- http://mitmproxy.org/

Tools: Creating Keys and Certs

- Java Keytool
- OpenSSL: powerful, but complex
- Xca: http://sourceforge.net/projects/xca/
  - Based on OpenSSL
  - Provides a Graphical User Interface (GUI)
- gnoMint: http://gnomint.sourceforge.net
  - Based on GnuTLS
  - Provides GUI and command line support

Tools: Creating Keys and Certs with xca



Securely implement SSL!

- Understand how SSL works and breaks
- Use SSL libraries and middleware securely
  - Don’t rely on default settings of SSL libraries and middleware/wrappers
  - Look out for badly designed SSL API (return value, error status)
- Perform certificate validation properly
  - Verify the certificate is valid: not expired, not revoked
  - Validate “Chain of Trust”
  - Don’t accept self-signed certificates
  - Make sure hostname validation is set
- Decouple insecure customized certificate handling from production code
- Test for insecure SSL configurations

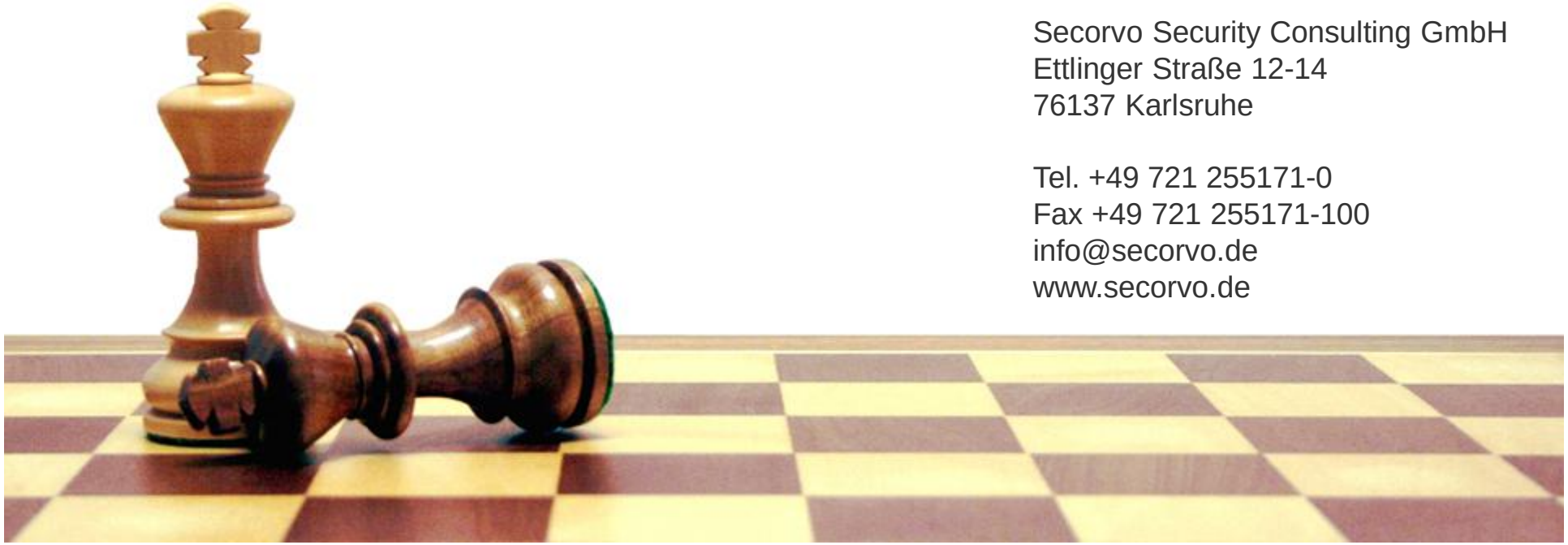
Engineering SSL is System Security Engineering

References

- M. Georgiev, S. Iyengar, S. Jana et al., “The Most Dangerous Code in the World: Validating SSL Certificates in Non-Browser Software”, 2012, http://www.cs.utexas.edu/~shmat/shmat_ccs12.pdf
- S. Fahl, M. Harbach, L. Baumgaertner and B. Freisleben, “Why Eve and Mallory Love Android: An Analysis of Android SSL (In)Security”, 2012, http://www2.dcsec.uni-hannover.de/files/android/p50-fahl.pdf
- S. Fahl, M. Harbach, H. Perl et al., “Rethinking SSL Development in an Applied World”, 2013, http://android-ssl.org/files/p49.pdf

- Comparison of TLS implementations http://en.wikipedia.org/wiki/Comparison_of_TLS_implementations

Copyright: © Kotist, bigstock.com



secorvo
security consulting

Secorvo Security Consulting GmbH
Ettlinger Straße 12-14
76137 Karlsruhe

Tel. +49 721 255171-0
Fax +49 721 255171-100
info@secorvo.de
www.secorvo.de