Magdeburger Journal zur Sicherheitsforschung

Gegründet 2011 | ISSN: 2192-4260
Herausgegeben von Stefan Schumacher und Jörg Sambleben
Erschienen im Magdeburger Institut für Sicherheitsforschung

Why Android SSL was downgraded from AES256-SHA to RC4-MD5 in late 2010

Dr. Georg Lukas

Android is using the combination of broken RC4[1] and MD5[2] as the first default cipher on *all SSL connections*. This impacts all apps that did not change the list of enabled ciphers (i.e. almost all existing apps). This paper investigates why RC4-MD5 is the default cipher, and why it replaced better ciphers which were in use prior to the Android 2.3 release in December 2010.

This article first appeared on the author's blog[3] in October 2013.

Citation: Lukas, D. G. (2013). Why Android SSL was downgraded from AES256-SHA to RC4-MD5 in late 2010. *Magdeburger Journal zur Sicherheitsforschung*, *2*, 385–392. Retrieved December 1, 2013, from http://www.sicherheitsforschung-magdeburg.de/publikationen.html

1 http://www.isg.rhul.ac.uk/tls/ v. 2013-10-15

2 http://www.win.tue.nl/hashclash/rogue-ca/ v. 2013-10-15

3 http://op-co.de/blog/posts/android_ssl_downgrade/

1 Preface

Some time ago, I was adding secure authentication[4] to my APRSdroid app[5] for Amateur Radio geolocation. While debugging its TLS handshake, I noticed that RC4-MD5 is leading the client's list of supported ciphers and thus wins the negotiation. As the task at hand was about authentication, not about secrecy, I ignored the fact.

However, following speculations about what the NSA can decrypt[6] and a series of excellent posts about XMPP clients by Thijs Alkemade[7] brought it into my focus again and I seriously asked myself what reasons led to it.

2 Status Quo Analysis

To obtain some data, I used the Wireshark sniffer on my router, started yaxim[8] on my Android 4.2.2 phone (CyanogenMod 10.1.3 on a Galaxy Nexus) and checked the Client Hello packet sent. Indeed, RC4-MD5 was first, followed by RC4-SHA1 (see Figure 1).

To quote from RFC 2246[9]: »The CipherSuite list, passed from the client to the server in the client hello message, contains the combinations of cryptographic algorithms supported by the client in order of the client's preference (favorite choice first).« Thus, the server is encouraged to actually use RC4-MD5 if it is not explicitly forbidden by its configuration.

To complete the picture, I obtained samples from Android 2.2.1 (CyanogenMod 6.1.0 on HTC Dream), 2.3.4 (Samsung original ROM on Galaxy SII) and 2.3.7 (CyanogenMod 7 on a Galaxy 5). The results can be seen in Table 1.

As the table shows, Android 2.2.1 came with a set of AES256-SHA1 ciphers first, followed by 3DES and AES128. Android 2.3 *significantly reduced the security* by removing AES256 and putting the broken RC4-MD5 on the prominent first place, followed by the not-so-much-better RC4-SHA1.

Indeed, Android versions before 2.3 were using AES256 > 3DES > AES128 > RC4, and starting with 2.3 it was now: RC4 > AES128 > 3DES. Also, the recently broken MD5 suddenly became the favorite MAC (Update: MD5 in TLS is OK, as it is combining two differently keyed MD5 hashes[10]).

The relevant change, part of Android 2.3, was released in late 2010, which coincides with the reported beginning of the NSA funding for sabotaging communication security. To trace back the relevant piece of code in the large Android framework, I wrote a minimal test program[11] and single-stepped it to find the origin of the default cipher list.

It turned out to be in Android's libcore package, `NativeCrypto.getDefaultCipherSuites()` which returns a hardcoded String array starting with `SSL_RSA_WITH_RC4_128_MD5`.

3 Diving Into the Android Source

Going back on that file's change history revealed interesting things, like the addition of TLS v1.1 and v1.2[12] and their almost immediate removal with a suspicious commit message just referencing "Bug 6234791"[13] (taking place between Android 4.0 and 4.1, possibly to improve compatibility with HTTPS servers[14]), added support for Elliptic Curves and AES256[15] in Android 3.x, and finally the addition of our hardcoded string list[16] sometime before Android 2.3 (Listing 2).

The commit message tells us: »We now have a default cipher suite list that is chose to match RI behavior and priority, not based on OpenSSLs default and priorities.« Translated into English: before, Android just used the list from OpenSSL (which was really good), now the list is replaced by RC4 and MD5 to match some other entity.

The test suite comes with another hint: »Note these are added in priority order as defined by RI 6 documentation.«

That RI 6 for sure has nothing to do with MI-6 (the British Secret Intelligence Service), but stands for *Reference Implementation*, the Sun (now Oracle) Java SDK version 6.

So what the fine Google engineers did to reduce the security of their users was merely to copy what was there, defined by the inventors of Java[17]!

4 http://aprsdroid.org/ssl/ v. 2013-10-15

5 https://play.google.com/store/apps/details?id=org.aprsdroid.app v. 2013-10-15

6 http://www.theregister.co.uk/2013/09/06/nsa_cryptobreaking_bullrun_analysis/ v. 2013-10-15

7 https://blog.thijsalkema.de/blog/2013/08/26/the-state-of-tls-on-xmpp-1/ v. 2013-10-15

8 http://yaxim.org v. 2013-10-15

9 http://tools.ietf.org/html/rfc2246#section-7.4.1.2 v. 2013-10-15

10 https://news.ycombinator.com/item?id=6548545 v. 2013-10-15

11 APK: http://duenndns.de/SSLCiphers.apk, source: http://duenndns.de/SSLCiphersProject.zip

12 https://android.googlesource.com/platform/libcore/+/3e6dd45baa0d7f9b4fa06f4ade76e088b59cc7bf%5E! v. 2013-10-15

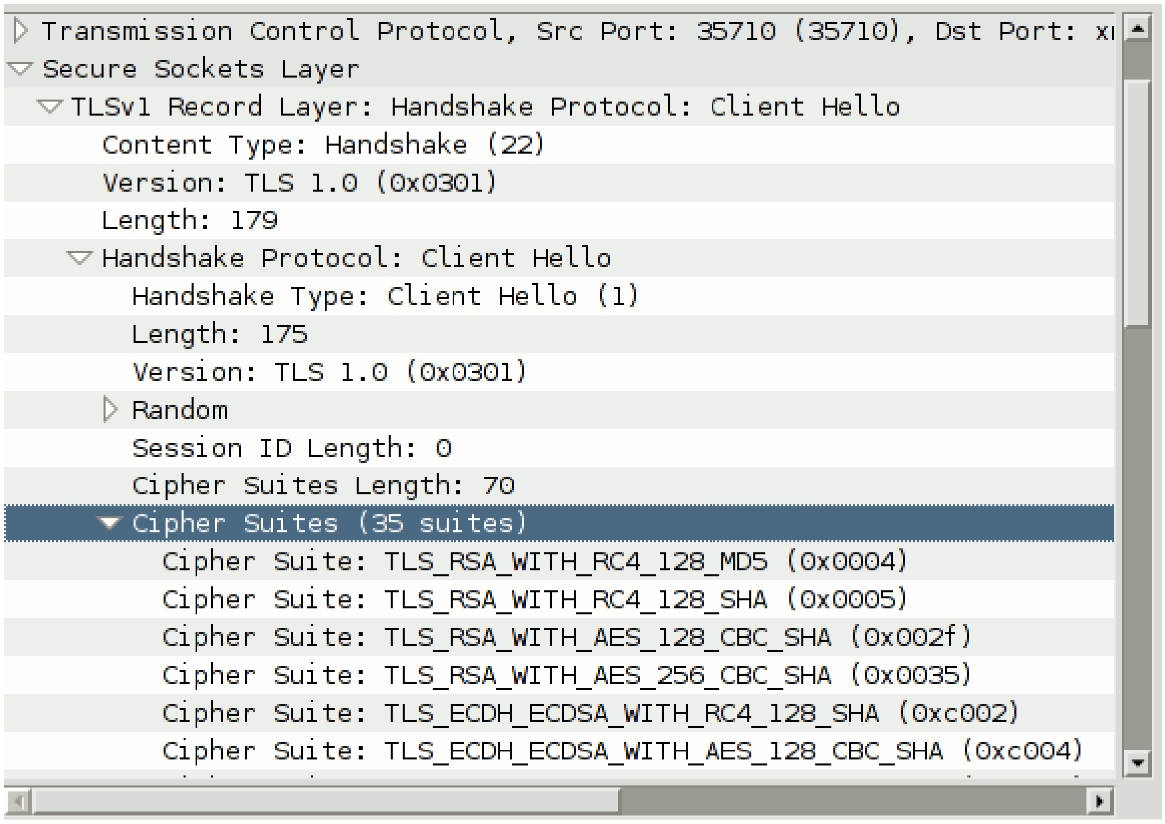
13 https://android.googlesource.com/platform/libcore/+/0731920fdf845358cc13ce78292f9e80e143f915%5E!/ v. 2013-10-15

14 https://code.google.com/p/android-source-browsing/source/detail?r=d473d7ae9135c9ca149a361b78366a753e1c0d5f&repo=platform--external--chromium v. 2013-10-15

15 https://android.googlesource.com/platform/libcore/+/4ae3fd787741bfe1b808f447dcb0785250024119%5E!/ v. 2013-10-15

16 https://android.googlesource.com/platform/libcore/+/9acacc36bafda869c6e9cc63786cdddd995ca96a%5E! v. 2013-10-15

17 http://docs.oracle.com/javase/6/docs/technotes/guides/security/SunProviders.html#SunJSSEProvider v. 2013-10-15



```
▷ Transmission Control Protocol, Src Port: 35710 (35710), Dst Port: x
▽ Secure Sockets Layer
  ▽ TLSv1 Record Layer: Handshake Protocol: Client Hello
      Content Type: Handshake (22)
      Version: TLS 1.0 (0x0301)
      Length: 179
    ▽ Handshake Protocol: Client Hello
        Handshake Type: Client Hello (1)
        Length: 175
        Version: TLS 1.0 (0x0301)
      ▷ Random
        Session ID Length: 0
        Cipher Suites Length: 70
      ▼ Cipher Suites (35 suites)
          Cipher Suite: TLS_RSA_WITH_RC4_128_MD5 (0x0004)
          Cipher Suite: TLS_RSA_WITH_RC4_128_SHA (0x0005)
          Cipher Suite: TLS_RSA_WITH_AES_128_CBC_SHA (0x002f)
          Cipher Suite: TLS_RSA_WITH_AES_256_CBC_SHA (0x0035)
          Cipher Suite: TLS_ECDH_ECDSA_WITH_RC4_128_SHA (0xc002)
          Cipher Suite: TLS_ECDH_ECDSA_WITH_AES_128_CBC_SHA (0xc004)
```

Figure 1: Wireshark Screenshot (Right-click your XMPP connection, then 'Decode as...' -> 'SSL')

```
 public static String[] getDefaultCipherSuites() {
-       int ssl_ctx = SSL_CTX_new();
-       String] supportedCiphers = SSL_CTX_get_ciphers(ssl_ctx);
-       SSL_CTX_free(ssl_ctx);
-       return supportedCiphers;
+        return new String] {
+            "SSL_RSA_WITH_RC4_128_MD5",
+            "SSL_RSA_WITH_RC4_128_SHA",
+            "TLS_RSA_WITH_AES_128_CBC_SHA",
...
+            "SSL_DHE_RSA_EXPORT_WITH_DES40_CBC_SHA",
+            "SSL_DHE_DSS_EXPORT_WITH_DES40_CBC_SHA"
+        };
 }
```

Figure 2: Hard-coded List of Default Client Ciphers, Introduced Between Android 2.2 and 2.3

Android 2.2.1	Android 2.3.4, 2.3.7	Android 4.2.2, 4.3
DHE-RSA-AES256-SHA	RC4-MD5	RC4-MD5
DHE-DSS-AES256-SHA	RC4-SHA	RC4-SHA
AES256-SHA	AES128-SHA	AES128-SHA
EDH-RSA-DES-CBC3-SHA	DHE-RSA-AES128-SHA	AES256-SHA
EDH-DSS-DES-CBC3-SHA	DHE-DSS-AES128-SHA	ECDH-ECDSA-RC4-SHA
DES-CBC3-SHA	DES-CBC3-SHA	ECDH-ECDSA-AES128-SHA
DES-CBC3-MD5	EDH-RSA-DES-CBC3-SHA	ECDH-ECDSA-AES256-SHA
DHE-RSA-AES128-SHA	EDH-DSS-DES-CBC3-SHA	ECDH-RSA-RC4-SHA
DHE-DSS-AES128-SHA	DES-CBC-SHA	ECDH-RSA-AES128-SHA
AES128-SHA	EDH-RSA-DES-CBC-SHA	ECDH-RSA-AES256-SHA
RC2-CBC-MD5	EDH-DSS-DES-CBC-SHA	ECDHE-ECDSA-RC4-SHA
RC4-SHA	EXP-RC4-MD5	ECDHE-ECDSA-AES128-SHA
RC4-MD5	EXP-DES-CBC-SHA	ECDHE-ECDSA-AES256-SHA
RC4-MD5	EXP-EDH-RSA-DES-CBC-SHA	ECDHE-RSA-RC4-SHA
EDH-RSA-DES-CBC-SHA	EXP-EDH-DSS-DES-CBC-SHA	ECDHE-RSA-AES128-SHA
EDH-DSS-DES-CBC-SHA		ECDHE-RSA-AES256-SHA
DES-CBC-SHA		DHE-RSA-AES128-SHA
DES-CBC-MD5		DHE-RSA-AES256-SHA
EXP-EDH-RSA-DES-CBC-SHA		DHE-DSS-AES128-SHA
EXP-EDH-DSS-DES-CBC-SHA		DHE-DSS-AES256-SHA
EXP-DES-CBC-SHA		DES-CBC3-SHA
EXP-RC2-CBC-MD5		ECDH-ECDSA-DES-CBC3-SHA
EXP-RC2-CBC-MD5		ECDH-RSA-DES-CBC3-SHA
EXP-RC4-MD5		ECDHE-ECDSA-DES-CBC3-SHA
EXP-RC4-MD5		ECDHE-RSA-DES-CBC3-SHA
		EDH-RSA-DES-CBC3-SHA
		EDH-DSS-DES-CBC3-SHA
		DES-CBC-SHA
		EDH-RSA-DES-CBC-SHA
		EDH-DSS-DES-CBC-SHA
		EXP-RC4-MD5
		EXP-DES-CBC-SHA
		EXP-EDH-RSA-DES-CBC-SHA
		EXP-EDH-DSS-DES-CBC-SHA

Table 1: Ciphers in the `ClientHello` Packets of the Same Application on Different Android Versions

4 Cipher Order in the Java Runtime

In the Java reference implementation, the code responsible for creating the cipher list is split into two files. First, a priority-ordered set of ciphers is constructed in the CipherSuite[18] class (Listing 3).

Then, all enabled ciphers with sufficient priority are added to the list for `CipherSuiteList.getDefault()`[19]. This cipher list has not experienced relevant changes since the initial import of Java 6 into Hg, when the OpenJDK was brought to life.

Going back in time reveals that even in the 1.4.0 JDK, the first one incorporating the Java Secure Socket Extension (JSEE) for SSL/TLS, the list was more or less the same (Table 2).

The original list resembles the CipherSpec definition in RFC 2246[20] from 1999, sorted numerically with the NULL and 40-bit ciphers moved down. Somewhere between the first release and 1.4.2, DES was deprecated, TLS was added to the mix (bringing in AES) and MD5 was pushed in front of SHA1 (which makes one wonder why). After that, the only chage was the addition of `TLS_EMPTY_RENEGOTIATION_INFO_SCSV`, which is not a cipher but just an information token for the server.

Java 7 added Elliptic Curves and significantly improved the cipher list[21] in 2011, but Android is based on JDK 6, making the effective default cipher list over 10 years old now.

5 Conclusion

The cipher order on the vast majority of Android devices was defined by Sun in 2002 and taken over into the Android project in 2010 as an attempt to improve compatibility. RC4 is considered problematic since 2001[22] (and issues are widely known since WEP was compromised[23]), MD5 was broken in 2009[24].

The change from the strong OpenSSL cipher list to a hardcoded one starting with weak ciphers is either a sign of horrible ignorance, security incompetence or a clever disguise for an NSA-influenced manipulation – it is up to the reader to decide! (This was before BEAST[25] made the other ciphers in TLS *less secure* in 2011 and RC4 gained momentum again).

All that notwithstanding, now is the time to get rid of RC4-MD5, in the applications as well as in the Android core! This change should go together with phasing out SSL v2 and v3, and introducing TLS v1.2 support (if not done so already).

6 Appendix A: Making Android App More Secure

If your app is only ever making contact to your own server, feel free to choose the best cipher that fits into your CPU budget! Otherwise, it is hard to give generic advice for an app to support a wide variety of different servers without producing obscure connection errors.

Server operators should read the excellent best practices document by SSLLabs[26].

6.1 Changing the Client Cipher List

For client developers, the well-motivated browser cipher suite proposal written by Brian Smith at Mozilla[27] is a good guideline, and also the basis for the following list. Figure 4 provides the Java code to change the cipher list to the subset of Brian's ciphers which are supported on Android 4.2.2. The last three ciphers are prefixed `SSL_` instead of `TLS_` in Android's SSL library.

6.2 Use TLS v1.2!

By default, TLS version 1.0 is used, and the more recent protocol versions are disabled. Some servers used to be broken[28] when contacted using v1.2, so this approach seemed a good conservative choice over a year ago[29].

At least for XMPP, an attempt to enforce TLS v1.2[30] is being made. You can follow with your own app easily, see Figure 5.

18 http://hg.openjdk.java.net/jdk6/jdk6/jdk/file/77af6e10b333/src/share/classes/sun/security/ssl/CipherSuite.java v. 2013-10-15

19 http://hg.openjdk.java.net/jdk6/jdk6/jdk/file/77af6e10b333/src/share/classes/sun/security/ssl/CipherSuiteList.java v. 2013-10-15

20 http://tools.ietf.org/html/rfc2246#appendix-A.5 v. 2013-10-15

21 http://docs.oracle.com/javase/7/docs/technotes/guides/security/SunProviders.html#SunJSSEProvider v. 2013-10-15

22 http://www.wisdom.weizmann.ac.il/~itsik/RC4/Papers/bc_rc4.ps v. 2013-10-15

23 http://en.m.wikipedia.org/wiki/Wired_Equivalent_Privacy v. 2013-10-15

24 http://www.win.tue.nl/hashclash/rogue-ca/ v. 2013-10-15

25 http://vnhacker.blogspot.de/2011/09/beast.html v. 2013-10-15

26 https://www.ssllabs.com/projects/best-practices/index.html v. 2013-10-15

27 https://briansmith.org/browser-ciphersuites-01.html v. 2013-10-15

28 https://bugs.launchpad.net/ubuntu/+source/openssl/+bug/965371 v. 2013-10-15

29 https://code.google.com/p/android-source-browsing/source/detail?r=d473d7ae9135c9ca149a361b78366a753e1c0d5f&repo=platform--external--chromium v. 2013-10-15

30 https://datatracker.ietf.org/doc/draft-saintandre-xmpp-tls/?include_text=1 v. 2013-10-15

```
// Definition of the CipherSuites that are enabled by default.
// They are listed in preference order, most preferred first.
int p = DEFAULT_SUITES_PRIORITY * 2;

add("SSL_RSA_WITH_RC4_128_MD5", 0x0004, --p, K_RSA, B_RC4_128, N);
add("SSL_RSA_WITH_RC4_128_SHA", 0x0005, --p, K_RSA, B_RC4_128, N);
...
```

Figure 3: Creation of a Priority-Ordered List of Ciphers in Java's CipherSuite class

```
// put this in a place where it can be reused
static final String ENABLED_CIPHERS] = {
                "TLS_ECDHE_RSA_WITH_AES_128_CBC_SHA",
                "TLS_ECDHE_ECDSA_WITH_AES_128_CBC_SHA",
                "TLS_ECDHE_RSA_WITH_AES_256_CBC_SHA",
                "TLS_ECDHE_ECDSA_WITH_AES_256_CBC_SHA",
                "TLS_DHE_RSA_WITH_AES_128_CBC_SHA",
                "TLS_DHE_RSA_WITH_AES_256_CBC_SHA",
                "TLS_DHE_DSS_WITH_AES_128_CBC_SHA",
                "TLS_ECDHE_RSA_WITH_RC4_128_SHA",
                "TLS_ECDHE_ECDSA_WITH_RC4_128_SHA",
                "TLS_RSA_WITH_AES_128_CBC_SHA",
                "TLS_RSA_WITH_AES_256_CBC_SHA",
                "SSL_RSA_WITH_3DES_EDE_CBC_SHA",
                "SSL_RSA_WITH_RC4_128_SHA",
                "SSL_RSA_WITH_RC4_128_MD5",
        };

// get a new socket from the factory
SSLSocket s = (SSLSocket)sslcontext.getSocketFactory().createSocket(host, port);
// IMPORTANT: set the cipher list before calling getSession(),
// startHandshake() or reading/writing on the socket!
s.setEnabledCipherSuites(ENABLED_CIPHERS);
...
```

Figure 4: Changing the Cipher Order in Android Apps

```
// put this in a place where it can be reused
static final String ENABLED_PROTOCOLS] = {
                "TLSv1.2", "TLSv1.1", "TLSv1"
        };

// put this right before setEnabledCipherSuites()!
s.setEnabledProtocols(ENABLED_PROTOCOLS);
```

Figure 5: Changing the Protocol Order in Android Apps

6.3 Use NetCipher!

NetCipher[31] is an Android library made by the Guardian Project[32] to improve network security for mobile apps. It comes with a StrongTrustManager to do more thorough certificate checks, an independent Root CA store, and code to easily route your traffic through the Tor network[33] using Orbot[34].

6.4 Use AndroidPinning!

AndroidPinning[35] is another Android library, written by Moxie Marlinspike[36] to allow pinning of server certificates, improving security against government-scale MitM attacks. Use this if your app is made to communicate with a specific server!

6.5 Use MemorizingTrustManager!

MemorizingTrustManager[37] by the author of this article is yet another Android library. It allows your app to ask the user if they want to trust a given self-signed/untrusted certificate, improving support for regular connections to private services. If you are writing an XMPP client or a private cloud sync app, use this!

7 Appendix B: Apps That Change the Cipher Ordering

In the following, a number of Android apps is listed that are using a custom cipher order. This is not a comprehensive list, merely a small snapshot.

7.1 Android Browser ("Internet")

Checks of the default Android Browser revealed that at least until Android 2.3.7 the Browser was using the default cipher list of the OS, participating in the RC4 regression. As of 4.2.2, the Browser comes with a better cipher list:

```
ECDHE-RSA-AES256-SHA
ECDHE-ECDSA-AES256-SHA
SRP-DSS-AES-256-CBC-SHA
SRP-RSA-AES-256-CBC-SHA
DHE-RSA-AES256-SHA DHE-DSS-AES256-SHA
ECDH-RSA-AES256-SHA
ECDH-ECDSA-AES256-SHA AES256-SHA
ECDHE-RSA-DES-CBC3-SHA
ECDHE-ECDSA-DES-CBC3-SHA
SRP-DSS-3DES-EDE-CBC-SHA
SRP-RSA-3DES-EDE-CBC-SHA
EDH-RSA-DES-CBC3-SHA
EDH-DSS-DES-CBC3-SHA
ECDH-RSA-DES-CBC3-SHA
ECDH-ECDSA-DES-CBC3-SHA
DES-CBC3-SHA ECDHE-RSA-AES128-SHA
ECDHE-ECDSA-AES128-SHA
SRP-DSS-AES-128-CBC-SHA
SRP-RSA-AES-128-CBC-SHA
DHE-RSA-AES128-SHA DHE-DSS-AES128-SHA
ECDH-RSA-AES128-SHA
ECDH-ECDSA-AES128-SHA AES128-SHA
ECDHE-RSA-RC4-SHA ECDHE-ECDSA-RC4-SHA
ECDH-RSA-RC4-SHA ECDH-ECDSA-RC4-SHA
RC4-SHA RC4-MD5
```

Surprisingly, the Android WebView[38] class (tested on Android 4.0.4) is also using the better cipher list.

7.2 Google Chrome

The Google Chrome browser (version 30.0.1599.82, 2013-10-11) serves the following list:

```
ECDHE-RSA-AES256-GCM-SHA384
ECDHE-ECDSA-AES256-GCM-SHA384
ECDHE-RSA-AES256-SHA
ECDHE-ECDSA-AES256-SHA
DHE-DSS-AES256-GCM-SHA384
DHE-RSA-AES256-GCM-SHA384
DHE-RSA-AES256-SHA256
DHE-DSS-AES256-SHA256
DHE-RSA-AES256-SHA DHE-DSS-AES256-SHA
AES256-GCM-SHA384 AES256-SHA256
AES256-SHA ECDHE-RSA-DES-CBC3-SHA
ECDHE-ECDSA-DES-CBC3-SHA
EDH-RSA-DES-CBC3-SHA
EDH-DSS-DES-CBC3-SHA DES-CBC3-SHA
ECDHE-RSA-AES128-GCM-SHA256
ECDHE-ECDSA-AES128-GCM-SHA256
ECDHE-RSA-AES128-SHA256
ECDHE-ECDSA-AES128-SHA256
ECDHE-RSA-AES128-SHA
ECDHE-ECDSA-AES128-SHA
DHE-DSS-AES128-GCM-SHA256
DHE-RSA-AES128-GCM-SHA256
DHE-RSA-AES128-SHA256
DHE-DSS-AES128-SHA256
DHE-RSA-AES128-SHA DHE-DSS-AES128-SHA
AES128-GCM-SHA256 AES128-SHA256
AES128-SHA ECDHE-RSA-RC4-SHA
ECDHE-ECDSA-RC4-SHA RC4-SHA RC4-MD5
```

This one comes with AES256-GCM *and* SHA384. Here, the different teams at Google should group up to provide the same proper level of security to developers as well as to users.

31 https://guardianproject.info/code/netcipher/ v. 2013-10-15

32 https://guardianproject.info/ v. 2013-10-15

33 https://www.torproject.org/ v. 2013-10-15

34 https://guardianproject.info/apps/orbot/ v. 2013-10-15

35 https://github.com/moxie0/AndroidPinning v. 2013-10-15

36 http://www.thoughtcrime.org/ v. 2013-10-15

37 https://github.com/ge0rg/MemorizingTrustManager/ v. 2013-10-15

38 http://developer.android.com/reference/android/webkit/WebView.html v. 2013-10-15

8 Further Reading

- "Real World Crypto 2013"[39] by Adam Langley from Google
- "Why does the web still run on RC4?"[40] by Luke Mather
- "SSL/TLS in a Post-PRISM Era"[41]

9 About the Author

Georg Lukas completed a Ph.D. degree in Computer Science in 2012, focusing the research on wireless communication and security. Currently, he is working as an IT Security consultant at rt-solutions.de GmbH in Cologne. He is developing smart-phone applications and working on mobile payment solutions.

You can contact the author at lukas@rt-solutions.de.

References

Lukas, D. G. (2013). Why Android SSL was downgraded from AES256-SHA to RC4-MD5 in late 2010. *Magdeburger Journal zur Sicherheitsforschung*, *2*, 385–392. Retrieved December 1, 2013, from http://www.sicherheitsforschung-magdeburg.de/publikationen.html

39 https://www.imperialviolet.org/2013/01/13/rwc03.html v. 2013-10-15

40 http://bristolcrypto.blogspot.fr/2013/08/why-does-web-still-run-on-rc4.html v. 2013-10-15

41 https://wiki.thc.org/ssl v. 2013-10-15

Java 1.4.0 (2002)	Java 1.4.2_19, 1.5.0 (2004)	Java 1.6 (2006)
SSL_RSA_WITH_RC4_128_SHA	SSL_RSA_WITH_RC4_128_MD5	SSL_RSA_WITH_RC4_128_MD5
SSL_RSA_WITH_RC4_128_MD5	SSL_RSA_WITH_RC4_128_SHA	SSL_RSA_WITH_RC4_128_SHA
SSL_RSA_WITH_DES_CBC_SHA	TLS_RSA_WITH_AES_128_CBC_SHA	TLS_RSA_WITH_AES_128_CBC_SHA
SSL_RSA_WITH_3DES_EDE_CBC_SHA	TLS_DHE_RSA_WITH_AES_128_CBC_SHA	TLS_DHE_RSA_WITH_AES_128_CBC_SHA
SSL_DHE_DSS_WITH_DES_CBC_SHA	TLS_DHE_DSS_WITH_AES_128_CBC_SHA	TLS_DHE_DSS_WITH_AES_128_CBC_SHA
SSL_DHE_DSS_WITH_3DES_EDE_CBC_SHA	SSL_RSA_WITH_3DES_EDE_CBC_SHA	SSL_RSA_WITH_3DES_EDE_CBC_SHA
SSL_RSA_EXPORT_WITH_RC4_40_MD5	SSL_DHE_RSA_WITH_3DES_EDE_CBC_SHA	SSL_DHE_RSA_WITH_3DES_EDE_CBC_SHA
SSL_DHE_DSS_EXPORT_WITH_DES40_CBC_SHA	SSL_DHE_DSS_WITH_3DES_EDE_CBC_SHA	SSL_DHE_DSS_WITH_3DES_EDE_CBC_SHA
SSL_RSA_WITH_NULL_MD5	SSL_RSA_WITH_DES_CBC_SHA	SSL_RSA_WITH_DES_CBC_SHA
SSL_RSA_WITH_NULL_SHA	SSL_DHE_RSA_WITH_DES_CBC_SHA	SSL_DHE_RSA_WITH_DES_CBC_SHA
SSL_DH_anon_WITH_RC4_128_MD5	SSL_DHE_DSS_WITH_DES_CBC_SHA	SSL_DHE_DSS_WITH_DES_CBC_SHA
SSL_DH_anon_WITH_DES_CBC_SHA	SSL_RSA_EXPORT_WITH_RC4_40_MD5	SSL_RSA_EXPORT_WITH_RC4_40_MD5
SSL_DH_anon_WITH_3DES_EDE_CBC_SHA	SSL_RSA_EXPORT_WITH_DES40_CBC_SHA	SSL_RSA_EXPORT_WITH_DES40_CBC_SHA
SSL_DH_anon_EXPORT_WITH_RC4_40_MD5	SSL_DHE_RSA_EXPORT_WITH_DES40_CBC_SHA	SSL_DHE_RSA_EXPORT_WITH_DES40_CBC_SHA
SSL_DH_anon_EXPORT_WITH_DES40_CBC_SHA	SSL_DHE_DSS_EXPORT_WITH_DES40_CBC_SHA	SSL_DHE_DSS_EXPORT_WITH_DES40_CBC_SHA
		TLS_EMPTY_RENEGOTIATION_INFO_SCSV

Table 2: Default Cipher Lists Provided by Different Releases of the Java Runtime Environment