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.


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 different hash algorithms)

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?\(^17\)

\(^{11}\) APK: http://duenndns.de/SSLCiphers.apk, source: http://duenndns.de/SSLCiphersProject.zip

\(^{12}\) https://android.googlesource.com/platform/libcore/+/

\(^{13}\) https://android.googlesource.com/platform/libcore/+/

\(^{14}\) https://android.googlesource.com/platform/libcore/+/

\(^{15}\) https://android.googlesource.com/platform/libcore/+/

\(^{16}\) https://android.googlesource.com/platform/libcore/+/

\(^{17}\) http://docs.oracle.com/javase/6/docs/technotes/guides/security/SSLJavaSSEProvider.html#SunJSSSEProvider 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 Suits Length: 70

Cipher Suits (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')

```java
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
<table>
<thead>
<tr>
<th>Android 2.2.1</th>
<th>Android 2.3.4, 2.3.7</th>
<th>Android 4.2.2, 4.3</th>
</tr>
</thead>
<tbody>
<tr>
<td>DHE-RSA-AES256-SHA</td>
<td>RC4-MD5</td>
<td>RC4-MD5</td>
</tr>
<tr>
<td>DHE-DSS-AES256-SHA</td>
<td>RC4-SHA</td>
<td>RC4-SHA</td>
</tr>
<tr>
<td>AES256-SHA</td>
<td>AES128-SHA</td>
<td>AES128-SHA</td>
</tr>
<tr>
<td>EDH-RSA-DES-CBC3-SHA</td>
<td>DHE-RSA-AES128-SHA</td>
<td>AES256-SHA</td>
</tr>
<tr>
<td>EDH-DSS-DES-CBC3-SHA</td>
<td>DHE-DSS-AES128-SHA</td>
<td>ECDH-ECDSA-RC4-SHA</td>
</tr>
<tr>
<td>DES-CBC3-SHA</td>
<td>DES-CBC3-SHA</td>
<td>ECDH-ECDSA-AES128-SHA</td>
</tr>
<tr>
<td>DES-CBC3-MD5</td>
<td>EDH-RSA-DES-CBC3-SHA</td>
<td>ECDH-ECDSA-AES256-SHA</td>
</tr>
<tr>
<td>DHE-RSA-AES128-SHA</td>
<td>EDH-DSS-DES-CBC3-SHA</td>
<td>ECDH-RSA-RC4-SHA</td>
</tr>
<tr>
<td>DHE-DSS-AES128-SHA</td>
<td>DES-CBC-SHA</td>
<td>ECDH-RSA-AES128-SHA</td>
</tr>
<tr>
<td>AES128-SHA</td>
<td>EDH-RSA-DES-CBC-SHA</td>
<td>ECDH-RSA-AES256-SHA</td>
</tr>
<tr>
<td>RC2-CBC-MD5</td>
<td>EDH-DSS-DES-CBC-SHA</td>
<td>ECDH-ECDSA-RC4-SHA</td>
</tr>
<tr>
<td>RC4-SHA</td>
<td>EXP-RC4-MD5</td>
<td>ECDH-ECDSA-AES128-SHA</td>
</tr>
<tr>
<td>RC4-MD5</td>
<td>EXP-DES-CBC-SHA</td>
<td>ECDH-ECDSA-AES256-SHA</td>
</tr>
<tr>
<td>RC4-MD5</td>
<td>EXP-EDH-RSA-DES-CBC-SHA</td>
<td>ECDH-RSA-RC4-SHA</td>
</tr>
<tr>
<td>EDH-RSA-DES-CBC-SHA</td>
<td>EXP-EDH-DSS-DES-CBC-SHA</td>
<td>ECDH-RSA-AES128-SHA</td>
</tr>
<tr>
<td>EDH-DSS-DES-CBC-SHA</td>
<td>EXP-EDH-DSS-DES-CBC-SHA</td>
<td>ECDH-RSA-AES256-SHA</td>
</tr>
<tr>
<td>DES-CBC-SHA</td>
<td>DES-CBC-SHA</td>
<td>ECDH-RSA-AES256-SHA</td>
</tr>
<tr>
<td>DES-CBC-MD5</td>
<td>EXP-EDH-RSA-DES-CBC-SHA</td>
<td>DHE-RSA-AES128-SHA</td>
</tr>
<tr>
<td>EXP-EDH-RSA-DES-CBC-SHA</td>
<td>DHE-DSS-AES128-SHA</td>
<td>DHE-DSS-AES256-SHA</td>
</tr>
<tr>
<td>EXP-EDH-DSS-DES-CBC-SHA</td>
<td>EXP-EDH-DSS-DES-CBC-SHA</td>
<td>DHE-DSS-AES256-SHA</td>
</tr>
<tr>
<td>EXP-DES-CBC-SHA</td>
<td>DES-CBC3-SHA</td>
<td>DES-CBC3-SHA</td>
</tr>
<tr>
<td>EXP-RC2-CBC-MD5</td>
<td>ECDH-ECDSA-DES-CBC3-SHA</td>
<td>EXP-EDH-RSA-DES-CBC-SHA</td>
</tr>
<tr>
<td>EXP-RC2-CBC-MD5</td>
<td>ECDH-RSA-DES-CBC-SHA</td>
<td>ECDH-ECDSA-DES-CBC3-SHA</td>
</tr>
<tr>
<td>EXP-RC4-MD5</td>
<td>ECDH-ECDSA-DES-CBC3-SHA</td>
<td>EXP-EDH-RSA-DES-CBC-SHA</td>
</tr>
<tr>
<td>EXP-RC4-MD5</td>
<td>ECDH-RSA-DES-CBC3-SHA</td>
<td>EXP-EDH-RSA-DES-CBC-SHA</td>
</tr>
<tr>
<td>EXP-RC4-MD5</td>
<td>ECDH-RSA-DES-CBC3-SHA</td>
<td>EXP-EDH-RSA-DES-CBC-SHA</td>
</tr>
<tr>
<td>EXP-RC4-MD5</td>
<td>ECDH-RSA-DES-CBC3-SHA</td>
<td>EXP-EDH-RSA-DES-CBC-SHA</td>
</tr>
</tbody>
</table>

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 CipherSuite18 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 224620 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 change 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 list21 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 200122 (and issues are widely known since WEP was compromised23), MD5 was broken in 200924.

The change from the strong OpenSSL cipher list to a hardcoded one starting with weak ciphers is either a sign of horrid ignorance, security incompetence or a clever disguise for an NSA-influenced manipulation – it is up to the reader to decide! (This was before BEAST25 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 SSLLabs26.

6.1 Changing the Client Cipher List

For client developers, the well-motivated browser cipher suite proposal written by Brian Smith at Mozilla27 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 broken28 when contacted using v1.2, so this approach seemed a good conservative choice over a year ago29.

At least for XMPP, an attempt to enforce TLS v1.230 is being made. You can follow with your own app easily, see Figure 5.
// 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_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",
    
};

Figure 4: Changing the Cipher Order in Android Apps

// 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);
...

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

Figure 5: Changing the Protocol Order in Android Apps
6.3 Use NetCipher!

NetCipher is an Android library made by the Guardian Project 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 using Orbot.

6.4 Use AndroidPinning!

AndroidPinning is another Android library, written by Moxie Marlinspike 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 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
EDH-RSA-DES-CBC3-SHA
EDH-RSA-DES-CBC3-SHA
ECRH-RSA-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
SRP-DES-RC4-SHA AES128-SHA
SRP-DSS-RC4-SHA RC4-SHA RC4-MD5

Surprisingly, the Android WebView 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 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.

---

33 https://www.torproject.org/: v. 2013-10-15
34 https://guardianproject.info/apps/orbot/: v. 2013-10-15
37 https://github.com/ge0rg/MemorizingTrustManager/: v. 2013-10-15
8 Further Reading

- “Real World Crypto 2013”\textsuperscript{39} by Adam Langley from Google
- “Why does the web still run on RC4?”\textsuperscript{40} by Luke Mather
- “SSL/TLS in a Post-PRISM Era”\textsuperscript{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


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

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

\textsuperscript{41} https://wiki.thc.org/ssl v. 2013-10-15
<table>
<thead>
<tr>
<th>Java 1.4.0 (2002)</th>
<th>Java 1.4.2_19, 1.5.0 (2004)</th>
<th>Java 1.6 (2006)</th>
</tr>
</thead>
<tbody>
<tr>
<td>SSL_RSA_WITH_RC4_128_SHA</td>
<td>SSL_RSA_WITH_RC4_128_SHA</td>
<td>SSL_RSA_WITH_RC4_128_SHA</td>
</tr>
<tr>
<td>SSL_RSA_WITH_RC4_128_MD5</td>
<td>SSL_RSA_WITH_RC4_128_MD5</td>
<td>SSL_RSA_WITH_RC4_128_MD5</td>
</tr>
<tr>
<td>SSL_RSA_WITH_DES_CBC_SHA</td>
<td>SSL_RSA_WITH_DES_CBC_SHA</td>
<td>SSL_RSA_WITH_DES_CBC_SHA</td>
</tr>
<tr>
<td>SSL_RSA_WITH_3DES_EDE_CBC_SHA</td>
<td>SSL_RSA_WITH_3DES_EDE_CBC_SHA</td>
<td>SSL_RSA_WITH_3DES_EDE_CBC_SHA</td>
</tr>
<tr>
<td>SSL_DH_DSS_WITH_3DES_EDE_CBC_SHA</td>
<td>SSL_DH_DSS_WITH_3DES_EDE_CBC_SHA</td>
<td>SSL_DH_DSS_WITH_3DES_EDE_CBC_SHA</td>
</tr>
<tr>
<td>SSL_RSA_EXPORT_WITH_RC4_40_MD5</td>
<td>SSL_RSA_EXPORT_WITH_RC4_40_MD5</td>
<td>SSL_RSA_EXPORT_WITH_RC4_40_MD5</td>
</tr>
<tr>
<td>SSL_RSA_EXPORT_WITH_NULL_SHA</td>
<td>SSL_RSA_EXPORT_WITH_NULL_SHA</td>
<td>SSL_RSA_EXPORT_WITH_NULL_SHA</td>
</tr>
<tr>
<td>SSL_DH_anon_WITH_RC4_128_MD5</td>
<td>SSL_DH_anon_WITH_RC4_128_MD5</td>
<td>SSL_DH_anon_WITH_RC4_128_MD5</td>
</tr>
<tr>
<td>SSL_DH_anon_EXPORT_WITH_DES40_CBC_SHA</td>
<td>SSL_DH_anon_EXPORT_WITH_DES40_CBC_SHA</td>
<td>SSL_DH_anon_EXPORT_WITH_DES40_CBC_SHA</td>
</tr>
<tr>
<td>SSL_DH_anon_EXPORT_WITH_RC4_40_MD5</td>
<td>SSL_DH_anon_EXPORT_WITH_RC4_40_MD5</td>
<td>SSL_DH_anon_EXPORT_WITH_RC4_40_MD5</td>
</tr>
<tr>
<td>SSL_DH_anon_EXPORT_WITH_3DES_EDE_CBC_SHA</td>
<td>SSL_DH_anon_EXPORT_WITH_3DES_EDE_CBC_SHA</td>
<td>SSL_DH_anon_EXPORT_WITH_3DES_EDE_CBC_SHA</td>
</tr>
</tbody>
</table>

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