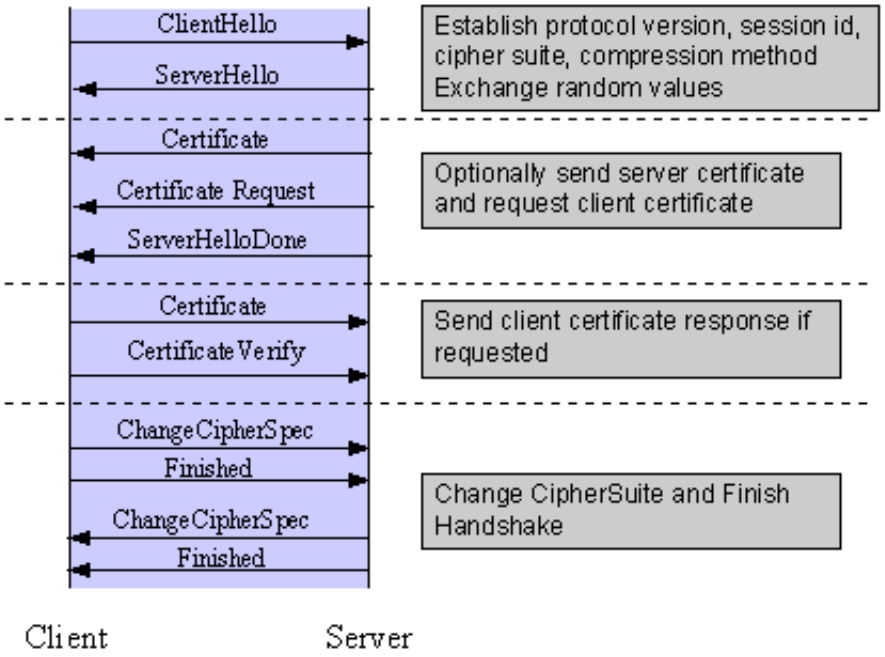
How to set the preferred cipher suite on Apache 2.2.x and Apache 2.4.x Reverse Proxy

Author : admin



1. Change default Apache (Reverse Proxy) SSL client cipher suite to end customer for Android Mobile applications to work

If you're a sys admin like me and you need to support client environments with multiple **Reverse Proxy Apache servers include old ones Apache version 2.2.x (with mod_ssl compiled in Apache or enabled as external module)**
and for that reason a certain specific Apache Reverse Proxy certificate SSL encoding cipher default served suite change to be **TLS_DHE_RSA_WITH_AES_128_CBC_SHA** in order for the application to properly communicate with the server backend application then this article might help you.

There is an end user client application which is Live on a production servers some of which running on backend WebSphere Application Servers (WAS) / SAP / Tomcat servers and f**or security and logging purposes the traffic is being forwarded from the Apache Reverse Proxies (whose traffic is incoming from a roundup Load Balancers).**

Here is a short background history of why cipher suite change is necessery?

The application worked fine and was used by a desktop PCs, however since recently there is an existent Android and Apple Store (iOS) mobile phone application and the Android Applications are unable to properly handle the default served Apache Reverse Proxy cipher suite and which forced the client to ask for change in the default SSL cipher suite to:

TLS_DHE_RSA_WITH_AES_128_CBC_SHA

By default, the way the client lists the cipher suites within its Client **Hello** will influence on Apache the selection of the cipher suite used between the client and server.

The current **httpd.conf** in Apache is configured so the ciphers for RP client **cipher suite Hello** transferred between **Reverse Proxy -> Client** are being provided in the following order:

1. **TLS_RSA_WITH_RC4_128_MD5**
2. **TLS_RSA_WITH_RC4_128_SHA**
3. **TLS_RSA_WITH_RC4_128_CBC_SHA**
4. **TLS_DHE_RSA_WITH_AES_128_CBC_SHA**

This has to be inverted so:

4. TLS_DHE_RSA_WITH_AES_128_CBC_SHA
becomes on the place of
1. TLS_RSA_WITH_RC4_128_MD5

A very good reading that helped me achieve the task as usual was Apache's official documentation about mod_ssl see here

So to fix the SSL/TLS cipher suite default served order use **SSLCipherSuite and SSLHonorCipherOrder** directives**.**

SSLCipherSuite directive is used to specify the cipher suites enabled on the server.

To dictate also preferred cipher suite order directive and that's why you need **SSLHonorCipherOrder** directive (note that this is not available for older Apache 2.x branch), the original bug for this directive can be seen within

For Example:

```
SSLHonorCipherOrder On
SSLCipherSuite RC4-SHA:AES128-SHA:AES256-SHA:DHE-RSA-AES128-SHA:DHE-RSA-AES256-SHA:DES-CBC3-SHA
```

So here is my fix for changing the Ciphersuite SSL Crypt order (notice the **TLS_DHE_RSA_WITH_AES_128_CBC_SHA** being given as first argument):

```
SSLHonorCipherOrder On
SSLCipherSuite TLS_DHE_RSA_WITH_AES_128_CBC_SHA:RC4-SHA:AES128-SHA:AES256-SHA:DHE-RSA-AES128-SHA:DHE-RSA-AES256-SHA:DES-CBC3-SHA
```

if you want also to enable TLSv1.2 certificate cipher support you can use also:

SSLProtocol -all +TLSv1.2

SSLHonorCipherOrder on

 # Old Commented configuration from my httpd.conf - no RC4, 3DES allowed
#SSLCipherSuite "EECDH+ECDSA+AESGCM EECDH+aRSA+AESGCM EECDH+ECDSA+SHA384 EECDH+ECDSA+SHA256 EECDH+aRSA+SHA384 EECDH+aRSA+SHA256 EECDH+aRSA+RC4 EECDH EDH+aRSA 3DES-EDE-CBC-SHA RC4 !aNULL !eNULL !LOW !MD5 !EXP !PSK !SRP !DSS !RC4"

Because there was also requirement for a multiple of SSL cipher encryption (to support large range of both mobile and desktop computers and operating systems the final) cipher suite configuration in ***httpd.conf*** *that worked for the client looked like so:*

SSLCipherSuite ECDHE-RSA-AES128-GCM-SHA256:ECDHE-ECDSA-AES128-GCM-SHA256:ECDHE-RSA-AES256-GCM-SHA384:ECDHE-ECDSA-AES256-GCM-SHA384:DHE-DSS-AES128-GCM-SHA256:kEDH+AESGCM:ECDHE-RSA-AES128-SHA256:ECDHE-ECDSA-AES128-SHA256:ECDHE-RSA-AES128-SHA:ECDHE-ECDSA-AES128-SHA:ECDHE-RSA-AES256-SHA384:ECDHE-ECDSA-AES256-SHA384:ECDHE-RSA-AES256-SHA:ECDHE-ECDSA-AES256-SHA:DHE-DSS-AES128-SHA256:DHE-DSS-AES256-SHA:AES128-GCM-SHA256:AES256-GCM-SHA384:AES128-SHA:AES256-SHA:AES:CAMELLIA:DES-CBC3-SHA:!aNULL:!eNULL:!EXPORT:!DES:!RC4:!MD5:!PSK:!aECDH:!EDH-DSS-DES-CBC3-SHA:!EDH-RSA-DES-CBC3-SHA:!KRB5-DES-CBC3-SHA:!DHE-RSA-AES128-GCM-SHA256:!DHE-RSA-AES256-GCM-SHA384:!DHE-RSA-AES128-SHA256:!DHE-RSA-AES256-SHA:!DHE-RSA-AES128-SHA:!DHE-RSA-AES256-SHA256:!DHE-RSA-CAMELLIA128-SHA:!DHE-RSA-CAMELLIA256-SHA

Once this was done **the customer requested HTTP cookie restriction to be added to the same virtual host.**
There initial request was to:

2. Set HTTP cookie secure flag and HttpOnly on every cookie that is not being accessed from Internal website JavaScript code

To make Apache Reverse Proxy to behave that way here is **the httpd.conf** config added **to httpd.conf**

vim **httpd.conf**

#Header edit Set-Cookie ^(.*)$ $1;HttpOnly;Secure
Header always edit Set-Cookie ^(.*)$ $1;HttpOnly;Secure

Finally an Apache restart was necessery