JPN_IWE216532: TEST SPECIFICATION

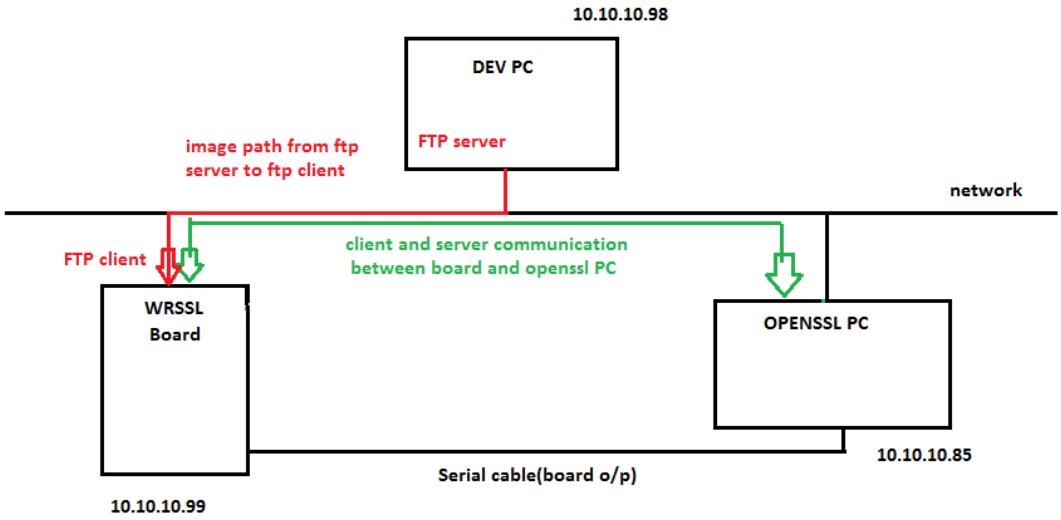
Step-by-step guide to test JPN_IWE216532 to Wind River VxWorks 6.2

08-Feb-2017

Contents

Development and Test Environment for JPN_IWE216532

Development and test environment for the JPN-IWE216532 project is as shown below.



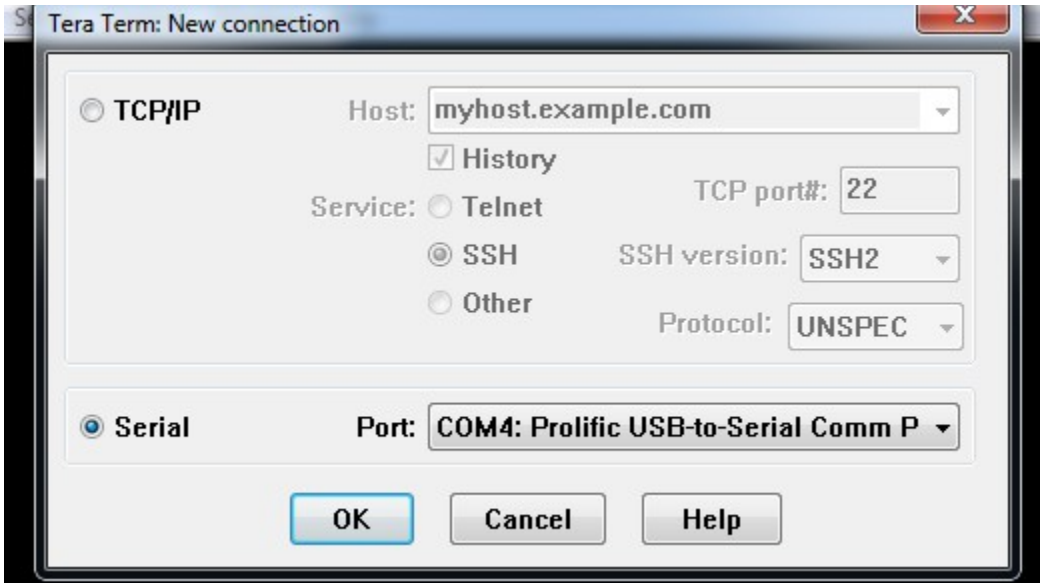
Ensure that required hardware and software are available to realize below setup before following instructions in this document.

Document also assumes that above setup is realized with required physical connections made as shown in above diagram.

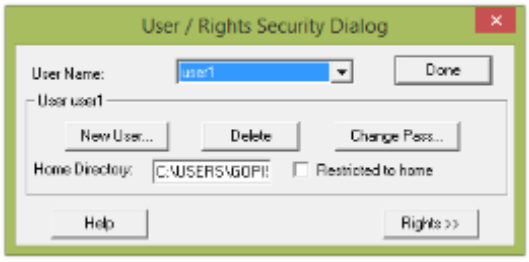
Bringing up Wind River Board

1. Power on the WindRiver target board.
2. Target board should be connected to OPENSSL PC using serial port. [use teraterm serial input/output or to see the console log of board.].

   In teraterm : Check in the serial option and select the port "COM4:Prolific USB-to-Serial Comm Port"



3. Make a LAN connection between Target board and system in which ftp server is running, i.e., DEV PC.[FTP server is part of windriver package and available under **start->all programs->windriver->vxworks-6.2->ftp server]**.
4. Board and DEV PC should be on same network.
5. Configure ftp server as below:
   - Launch ftp server from **start->all programs->windriver->vxworks-6.2->ftp server.**
   - Select **security->user/rights.** Pop up window as shown below fig will open.
   - Select username(if no user available then create new by pressing 'new user') and set password and home directory for that user.(set the directory where Vxworks images will be present as the home directory of ftp server).



6. Switch off and switch on the board and wait for '[VxWorks Boot]'command prompt.
7. Configure board as below: (press 'c' to get configure window)
   - Select boot device(type 'help' to get the list of supported boot device).
   - Set appropriate IP address for board.
   - Host IP and gateway should be same as FTP server system's IP address.
   - FTP user and password field should match with FTP server configuration.
   - Following fig. shows sample example

```
[VxWorks Boot]: @

boot device          : motfcc
unit number          : 0
processor number     : 0
host name            : host
file name            : vxWorks
inet on ethernet (e) : 10.10.10.99:ffffff00
host inet (h)        : 10.10.10.100
gateway inet (g)     : 10.10.10.100
user (u)             : target
ftp password (pw)    : target
flags (f)            : 0x0
other (o)            : motfcc0
```

8. Once board configuration is done, '[VxWorks Boot]' prompt will be displayed again. You may use command 'p' to ensure that configuration is correct.

9. Copy Vxworks images created to FTP home directory.

Test Environment Setup & Execution Procedure

A. Test Preparation

On OPENSSL-LINUX PC

1.1.1. Download and install OPENSSL-1.0.1s from https://www.openssl.org/source/openssl-1.0.1s.tar.gz

1.1.2. Following certificates are to be generated and should be kept under respective folders.

sha256WithRSAEncryption

```
$opensslgenrsa -out sha256rsakey.pem 2048

$openssl req -out sha256rsa.csr -key sha256rsakey.pem -new -sha256

$openssl req -in sha256rsa.csr -out sha256rsa.pem -text
```

SHA1withECDSA

```
$openssl ecparam -name secp256k1 -genkey -out sha1ecdsakey.pem

$openssl req -new -x509 -key sha1ecdsakey.pem -out sha1ecdsa.pem -days
 730
```

Note: You may verify the key and the certificate contents using

```
$openssl ecparam -in sha1ecdsakey.pem -text -noout

$openssl x509 -in sha1ecdsa.pem -text -noout
```

sha256withECDSA

```
$opensslecparam -name secp256k1 -genkey -out sha256ecdsakey.pem

$openssl req -new -x509 -key sha256ecdsakey.pem -out sha256ecdsa.pem
 -days 730
```

Note: You may verify the key and the certificate contents using

```
$opensslecparam -in sha256ecdsakey.pem -text -noout

$openssl x509 -in sha256ecdsa.pem -text -noout
```

1.1.3. Copy above generated certificate and key files to openssl-1.0.1s folder on OPENSSL-LINUX PC

1.1.4. Transfer above generated certificate and key files (via FTP or any other method) to home directory of FTP server on DEV-PC.

sha256RSA_ecPubKey-prime256v1

Note: Must use OpenSSL version 1.0.2 or above. (We used 1.0.2h)

```
A) Create CA certificate
      a) Create ca directory
         $mkdir $HOME/testca
         $cd $HOME/testca
         $mkdir certs crl newcerts private
         $chmod 700 private
         $touch index.txt
         $echo 1000 > serial

      b) Create configuration file testca.cnf as in APPENDIX 1
         (replace $HOME appropriately)

      c) Create root key
         $cd /home/testca
         $openssl genrsa -aes256 -out private/ca.key.pem 4096
         $chmod 400 private/ca.key.pem
         $openssl req -config openssl.cnf -key private/ca.key.pem -new -
          x509 -days 7300 -sha256 -extensions v3_ca \
          -out certs/ca.cert.pem
         $chmod 444 certs/ca.cert.pem

B) Create ECDH-RSA certificate

      a)Generate ECDH key
      $openssl ecparam -out ecparam.pem -name prime256v1
      $openssl genpkey -paramfile ecparam.pem -out ecdhkey.pem
```

b)Create the public key file:
$openssl pkey -in ecdhkey.pem -pubout -out ecdhpubkey.pem

c)Createa CSR file. This will be RSA signed.
$openssl genrsa -out rsakey.pem 1024
$openssl req -new -key rsakey.pem -out rsa.csr

d)Create ECDH-RSA certificate – force DH public key into the certificate
$openssl x509 -req -in rsa.csr -CAkey ca.key.pem -CA ca.cert.pem -force_pubkey ecdhpubkey.pem –out ecdhcert.pem -CAcreateserial

C)Run OpenSSL server
$openssl s_server -cert ecdhcert.pem -key ecdhpubkey.pem

APPENDIX 1

Configuration file: testca.cnf

```
# This definition stops the following lines choking if HOME isn't
# defined.
HOME       = .
RANDFILE   = $ENV::HOME/.rnd


# Extra OBJECT IDENTIFIER info:
#oid_file  = $ENV::HOME/.oid
oid_section = new_oids

[ new_oids ]

# Policies used by the TSA examples.
tsa_policy1 = 1.2.3.4.1
tsa_policy2 = 1.2.3.4.5.6
tsa_policy3 = 1.2.3.4.5.7

####################################################################
[ ca ]
default_ca          = CA_default         # The default ca section

####################################################################
[ CA_default ]
dir            = $HOME/testca
certs          = $dir/certs  # Where the issued certs are kept
crl_dir        = $dir/crl    # Where the issued crl are kept
database       = $dir/index.txt  # database index file.
#unique_subject = no  # Set to 'no' to allow creation of
                      # several ctificates with same subject.
new_certs_dir  = $dir/newcerts # default place for new certs.
certificate    = $dir/cacert.pem # The CA certificate
serial         = $dir/serial     # The current serial number
crlnumber      = $dir/crlnumber  # the current crl number

# must be commented out to leave a V1 CRL
crl                           = $dir/crl.pem                   # The
current CRL
```

```
private_key        = $dir/private/cakey.pem # The private key
RANDFILE           = $dir/private/.rand    # private random number file
x509_extensions    = usr_cert  # The extentions to add to the cert

# Comment out the following two lines for the "traditional"
# (and highly broken) format.
name_opt           = ca_default # Subject Name options
cert_opt           = ca_default # Certificate field options
default_days       = 365        # how long to certify for
default_crl_days   = 30         # how long before next CRL
default_md         = sha256     # use public key default MD
preserve           = no     # keep passed DN ordering

# A few difference way of specifying how similar the request should look
# For type CA, the listed attributes must be the same, and the optional
# and supplied fields are just that :-)
policy             = policy_match
# For the CA policy
[ policy_match ]
countryName        = match
stateOrProvinceName   = match
organizationName   = match
organizationalUnitName   = optional
commonName               = supplied
emailAddress             = optional

# For the 'anything' policy
# At this point in time, you must list all acceptable 'object'
# types.
[ policy_anything ]
countryName           = optional
stateOrProvinceName   = optional
localityName          = optional
organizationName      = optional
organizationalUnitName  = optional
commonName              = supplied
emailAddress            = optional

####################################################################
[ req ]
default_bits       = 1024
default_keyfile    = privkey.pem
distinguished_name = req_distinguished_name
attributes         = req_attributes
x509_extensions    = v3_ca # The extentions to add to the self signed
cert

# WARNING: ancient versions of Netscape crash on BMPStrings or
UTF8Strings.
string_mask = utf8only
# req_extensions = v3_req # The extensions to add to a certificate
request
[ req_distinguished_name ]
countryName = Country Name (2 letter code)
countryName_default  = AU
countryName_min    = 2
countryName_max    = 2
```

```
stateOrProvinceName = State or Province Name (full name)
stateOrProvinceName_default = Some-State

localityName    = Locality Name (eg, city)

0.organizationName = Organization Name (eg, company)
0.organizationName_default = Internet Widgits Pty Ltd

# we can do this but it is not needed normally :-)
#1.organizationName = Second Organization Name (eg, company)
#1.organizationName_default   = World Wide Web Pty Ltd

organizationalUnitName  = Organizational Unit Name (eg, section)
#organizationalUnitName_default =

commonName     = Common Name (e.g. server FQDN or YOUR name)
commonName_max = 64

emailAddress   = Email Address
emailAddress_max  = 64

# SET-ex3         = SET extension number 3

[ req_attributes ]
challengePassword  = A challenge password
challengePassword_min  = 4
challengePassword_max  = 20

unstructuredName  = An optional company name

[ usr_cert ]
# These extensions are added when 'ca' signs a request.

basicConstraints = CA:FALSE

# This will be displayed in Netscape's comment listbox.
nsComment   = "OpenSSL Generated Certificate"

# PKIX recommendations harmless if included in all certificates.
subjectKeyIdentifier = hash
authorityKeyIdentifier = keyid,issuer

[ v3_req ]
# Extensions to add to a certificate request
basicConstraints = CA:FALSE
keyUsage = nonRepudiation, digitalSignature, keyEncipherment

[ v3_ca ]
# Extensions for a typical CA
# PKIX recommendation.
subjectKeyIdentifier=hash
authorityKeyIdentifier=keyid:always,issuer
basicConstraints = CA:true

[ crl_ext ]
# CRL extensions.
```

```
# Only issuerAltName and authorityKeyIdentifier make any sense in a CRL.
# issuerAltName=issuer:copy
authorityKeyIdentifier=keyid:always

[ proxy_cert_ext ]
# These extensions should be added when creating a proxy certificate
basicConstraints=CA:FALSE
# This will be displayed in Netscape's comment listbox.
nsComment  = "OpenSSL Generated Certificate"

# PKIX recommendations harmless if included in all certificates.
subjectKeyIdentifier=hash
authorityKeyIdentifier=keyid,issuer

# This really needs to be in place for it to be a proxy certificate.
proxyCertInfo=critical,language:id-ppl-anyLanguage,pathlen:3,policy:foo

####################################################################
[ tsa ]
default_tsa = tsa_config1 # the default TSA section

[ tsa_config1 ]
# This really needs to be in place for it to be a proxy certificate.
proxyCertInfo=critical,language:id-ppl-anyLanguage,pathlen:3,policy:foo

####################################################################
[ tsa ]
default_tsa = tsa_config1 # the default TSA section

[ tsa_config1 ]
# These are used by the TSA reply generation only.
dir     = ./demoCA # TSA root directory
serial  = $dir/tsaserial  # The current serial number (mandatory)
crypto_device   = builtin # OpenSSL engine to use for signing
signer_cert     = $dir/tsacert.pem  # The TSA signing certificate
# (optional)
certs  = $dir/cacert.pem # Certificate chain to include in reply

# (optional)
signer_key  = $dir/private/tsakey.pem # The TSA private key (optional)
default_policy   = tsa_policy1 # Policy if request did not specify it
# (optional)
other_policies   = tsa_policy2, tsa_policy3  # acceptable policies
(optional)
digests  = md5, sha1 # Acceptable message digests (mandatory)
accuracy  = secs:1, millisecs:500, microsecs:100      # (optional)
clock_precision_digits  = 0   # number of digits after dot. (optional)
ordering = yes      # Is ordering defined for timestamps?
 # (optional, default: no)
tsa_name = yes      # Must the TSA name be included in the reply?
# (optional, default: no)
ess_cert_id_chain = no # Must the ESS cert id chain be included?
# (optional, default: no)
```

B. Test Execution

Sl.No	Cipher suite	TLS1	TLS1_1	TLS1_2
1	DHE-RSA-AES128-SHA256	NA	NA	YES
2	DHE-RSA-AES128-SHA	YES	YES	YES
3	ECDHE-ECDSA-AES128-SHA256	NA	NA	YES
4	ECDHE-RSA-AES128-SHA256	NA	NA	YES
5	ECDHE-ECDSA-AES128-SHA	YES	YES	YES
6	ECDHE-RSA-AES128-SHA	YES	YES	YES
7	AES128-SHA256	NA	NA	YES
8	AES128-SHA	YES	YES	YES
9	ECDH-ECDSA-AES128-SHA256	NA	NA	YES
10	ECDH-RSA-AES128-SHA256	NA	NA	YES
11	ECDH-ECDSA-AES128-SHA	YES	YES	YES
12	ECDH-RSA-AES128-SHA	YES	YES	YES
13	DHE-RSA-AES256-SHA256	NA	NA	YES
14	DHE-RSA-AES256-SHA	YES	YES	YES
15	ECDHE-ECDSA-AES256-SHA	YES	YES	YES
16	ECDHE-RSA-AES256-SHA	YES	YES	YES
17	AES256-SHA256	NA	NA	YES
18	AES256-SHA	YES	YES	YES

NA:- Not Applicable

C. Test Procedure

Note:- Repeat the above test for <PROTOCOLS> tls1,tls1_1 and tls1_2.

CASE: A01 DHE-RSA-AES128-SHA

On OPENSSL-LINUX -PC:

1. Open terminal, change directory to openssl-1.0.1s and run openssl server as shown below

   e.g.

```
$cd/home/user1/openssls/openssl-1.0.1s
$./apps/openssl s_server -accept 443 -cert sha256rsa.pem -key
 sha256rsakey.pem -<PROTOCOLS> -msg
```

On WRSSL-Board and teraterm on OPENSSL-PC:

2. Run WRSSL as client using following command in WRSSL-PC teraterm terminal.

```
->nm_client"5","-<PROTOCOLS>","-cipher","DHE-RSA-AES128-SHA","-
connect","<OPENSSL_PC_IPAddr>:443"
```

Expected result:

SSL connection should establish successfully. Verify by typing in a test message on the client terminal and seeing the same on server terminal and vice versa. Try to close connection from client. Connection should get closed without problems.

CASE: A02 DHE-RSA-AES128-SHA256

On OPENSSL-LINUX -PC:

1.Open terminal, change directory to openssl-1.0.1s and run openssl server as shown below

e.g.

```
$cd/home/user1/openssls/openssl-1.0.1s
$./apps/openssl s_server -accept 443 -cert sha256rsa.pem -key
 sha256rsakey.pem -<PROTOCOLS> -msg
```

On WRSSL-Board and teraterm on OPENSSL-PC:

2. Run WRSSL as client using following command in WRSSL-PC teraterm terminal.

```
->nm_client "5","-<PROTOCOLS>","-cipher","DHE-RSA-AES128-SHA256","-
 connect","<OPENSSL_PC_IPAddr>:443"
```

Expected result:

SSL connection should establish successfully. Verify by typing in a test message on the client terminal and seeing the same on server terminal and vice versa. Try to close connection from client. Connection should get closed without problems.

CASE: A03 ECDHE-ECDSA-AES128-SHA256

On OPENSSL-LINUX -PC:

1.Open terminal, change directory to openssl-1.0.1s and run openssl server as shown below

e.g.

```
$cd/home/user1/openssls/openssl-1.0.1s
$./apps/openssl s_server -accept 443 -cert sha256ecdsa.pem -key
sha256ecdsakey.pem -<PROTOCOLS> -msg
```

On WRSSL-Board and tera term on OPENSSL-PC:

2. Run WRSSL as client using following command in WRSSL-PC tera term terminal.

```
->nm_client"5","-<PROTOCOLS>","-cipher","ECDHE-ECDSA-AES128-SHA256","-
connect","<OPENSSL_PC_IPAddr>:443"
```

Expected result:

SSL connection should establish successfully. Verify by typing in a test message on the client terminal and seeing the same on server terminal and vice versa. Try to close connection from client. Connection should get closed without problems.

CASE: A04 ECDHE-RSA-AES128-SHA256

On OPENSSL-LINUX -PC:

1.1.1. Open terminal, change directory to openssl-1.0.1s and run openssl server as shown below e.g.

```
$cd/home/user1/openssls/openssl-1.0.1s
$./apps/openssl s_server -accept 443 -cert sha256rsa.pem -key
sha256rsakey.pem -<PROTOCOLS> -msg
```

On WRSSL-Board and teraterm on OPENSSL-PC:

2. Run WRSSL as client using following command in WRSSL PC tera term terminal.

```
->nm_client"5","-<PROTOCOLS>","-cipher","ECDHE-RSA-AES128-SHA256","-
connect","<OPENSSL_PC_IPAddr>:443"
```

Expected result:

SSL connection should establish successfully. Verify by typing in a test message on the client terminal and seeing the same on server terminal and vice versa. Try to close connection from client. Connection should get closed without problems.

CASE: A05 ECDHE-ECDSA-AES128-SHA

On OPENSSL-LINUX -PC:

1.Open terminal, change directory to openssl-1.0.1s and run openssl server as shown below e.g.

```
$cd/home/user1/openssls/openssl-1.0.1s
$./apps/openssl s_server -accept 443 -cert sha256ecdsa.pem -key
sha256ecdsakey.pem -<PROTOCOLS> -msg
```

On WRSSL-Board and teraterm on OPENSSL-PC:

2. Run WRSSL as client using following command in WRSSL PC tera term terminal .

```
->nm_client"5","-<PROTOCOLS>","-cipher","ECDHE-ECDSA-AES128-SHA","-
connect","<OPENSSL_PC_IPAddr>:443"
```

Expected result:

SSL connection should establish successfully. Verify by typing in a test message on the client terminal and seeing the same on server terminal and vice versa. Try to close connection from client. Connection should get closed without problems.

CASE: A06 ECDHE-RSA-AES128-SHA

On OPENSSL-LINUX -PC:

1.Open terminal, change directory to openssl-1.0.1s and run openssl server as shown below

e.g.

```
$cd /home/user1/openssls/openssl-1.0.1s
$./apps/openssl s_server -accept 443 -cert sha256rsa.pem -key
       sha256rsakey.pem -<PROTOCOLS> -msg
```

On WRSSL-Board and teraterm on OPENSSL-PC:

2.Run WRSSL as client using following command in WRSSL PC teraterm terminal.

```
->nm_client"5","-<PROTOCOLS>","-cipher","ECDHE-RSA-AES128-SHA","-
connect","<OPENSSL_PC_IPAddr>:443"
```

Expected result:

SSL connection should establish successfully. Verify by typing in a test message on the client terminal and seeing the same on server terminal and vice versa. Try to close connection from client. Connection should get closed without problems.

CASE: A07 AES128-SHA256

On OPENSSL-LINUX -PC:

1.Open terminal, change directory to openssl-1.0.1s and run openssl server as shown below

e.g.

```
$cd/home/user1/openssls/openssl-1.0.1s
$./apps/openssl s_server -accept 443 -cert sha256rsa.pem -key
sha256rsakey.pem -<PROTOCOLS> -msg
```

On WRSSL-Board and teraterm on OPENSSL-PC:

2.Run WRSSL as client using following command in WRSSL-PC teraterm terminal.

```
->nm_client"5","-<PROTOCOLS>","-cipher","AES128-SHA256","-
connect","<OPENSSL_PC_IPAddr>:443"
```

Expected result:

SSL connection should establish successfully. Verify by typing in a test message on the client terminal and seeing the same on server terminal and vice versa. Try to close connection from client. Connection should get closed without problems.

CASE: A08 AES128-SHA

On OPENSSL-LINUX -PC:

1.Open terminal, change directory to openssl-1.0.1s and run openssl server as shown below

e.g.

```
$cd/home/user1/openssls/openssl-1.0.1s
$./apps/openssl s_server -accept 443 -cert sha256rsa.pem -key
sha256rsakey.pem -<PROTOCOLS> -msg
```

On WRSSL-Board and teraterm on OPENSSL-PC:

1. Run WRSSL as client using following command in WRSSL-PC teraterm terminal.

```
->nm_client"5","-<PROTOCOLS>","-cipher","AES128-SHA","-
connect","<OPENSSL_PC_IPAddr>:443"
```

Expected result:

SSL connection should establish successfully. Verify by typing in a test message on the client terminal and seeing the same on server terminal and vice versa. Try to close connection from client. Connection should get closed without problems.

CASE: A09 DHE-RSA-AES256-SHA256

On OPENSSL-LINUX -PC:

1.Open terminal, change directory to openssl-1.0.1s and run openssl server as shown below e.g.

```
$cd/home/user1/openssls/openssl-1.0.1s
$./apps/openssl s_server -accept 443 -cert sha256rsa.pem -key
sha256rsakey.pem –<PROTOCOLS> -msg
```

On WRSSL-Board and teraterm on OPENSSL-PC:

2. Run WRSSL as client using following command in WRSSL-PC teraterm terminal.

```
->nm_client "5","-<PROTOCOLS>","-cipher","DHE-RSA-AES256-SHA256","-
connect","<OPENSSL_PC_IPAddr>:443"
```

Expected result:

SSL connection should establish successfully. Verify by typing in a test message on the client terminal and seeing the same on server terminal and vice versa. Try to close connection from client. Connection should get closed without problems.

CASE: A10 DHE-RSA-AES256-SHA

On OPENSSL-LINUX -PC:

1. Open terminal, change directory to openssl-1.0.1s and run openssl server as shown below e.g.

```
$cd/home/user1/openssls/openssl-1.0.1s
$./apps/openssl s_server -accept 443 -cert sha256rsa.pem -key
       sha256rsakey.pem –<PROTOCOLS> -msg
```

On WRSSL-Board and teraterm on OPENSSL-PC:

2. Run WRSSL as client using following command in WRSSL-PC teraterm terminal.

```
->nm_client "5","-<PROTOCOLS>","-cipher","DHE-RSA-AES256-SHA","-
connect","<OPENSSL_PC_IPAddr>:443"
```

Expected result:

SSL connection should establish successfully. Verify by typing in a test message on the client terminal and seeing the same on server terminal and vice versa. Try to close connection from client. Connection should get closed without problems.

CASE: A11 ECDHE-ECDSA-AES256-SHA

On OPENSSL-LINUX -PC:

1. Open terminal, change directory to openssl-1.0.1s and run openssl server as shown below

e.g.

```
$cd/home/user1/openssls/openssl-1.0.1s
$./apps/openssl s_server -accept 443 -cert sha256ecdsa.pem -key
sha256ecdsakey.pem -<PROTOCOLS> -msg
```

On WRSSL-Board and teratermon OPENSSL-PC:

2.Run WRSSL as client using following command in WRSSL-PC teraterm terminal.

```
->nm_client"5","-<PROTOCOLS>","-cipher","ECDHE-ECDSA-AES256-SHA","-
connect","<OPENSSL_PC_IPAddr>:443"
```

Expected result:

SSL connection should establish successfully. Verify by typing in a test message on the client terminal and seeing the same on server terminal and vice versa. Try to close connection from client. Connection should get closed without problems.

CASE: A12 ECDHE-RSA-AES256-SHA

On OPENSSL-LINUX -PC:

1. Open terminal, change directory to openssl-1.0.1s and run openssl server as shown below

e.g.

```
$cd/home/user1/openssls/openssl-1.0.1s
$./apps/openssl s_server -accept 443 -cert sha256rsa.pem -key
sha256ecdsakey.pem -<PROTOCOLS> -msg
```

On WRSSL-Board and teraterm on OPENSSL-PC:

2. Run WRSSL as client using following command in WRSSL-PC teraterm terminal.

```
->nm_client"5","-<PROTOCOLS>","-cipher","ECDHE-RSA-AES256-SHA","-
connect","<OPENSSL_PC_IPAddr>:443"
```

Expected result:

SSL connection should establish successfully. Verify by typing in a test message on the client terminal and seeing the same on server terminal and vice versa. Try to close connection from client. Connection should get closed without problems.

CASE: A13 AES256-SHA256

On OPENSSL-LINUX -PC:

1. Open terminal, change directory to openssl-1.0.1s and run openssl server as shown below

e.g.

```
$cd/home/user1/openssls/openssl-1.0.1s
$./apps/openssl s_server -accept 443 -cert sha256rsa.pem -key
sha256rsakey.pem -<PROTOCOLS> -msg
```

On WRSSL-Board and tearterm on OPENSSL-PC:

2. Run WRSSL as client using following command in WRSSL-PC teraterm terminal.

```
->nm_client"5","-<PROTOCOLS>","-cipher","AES256-SHA256","-
connect","<OPENSSL_PC_IPAddr>:443"
```

Expected result:

SSL connection should establish successfully. Verify by typing in a test message on the

client terminal and seeing the same on server terminal and vice versa. Try to close connection from client. Connection should get closed without problems.

CASE: A14 AES256-SHA

On OPENSSL-LINUX -PC:

1. Open terminal, change directory to openssl-1.0.1s and run openssl server as shown below

e.g.

```
$cd/home/user1/openssls/openssl-1.0.1s
$./apps/openssl s_server -accept 443 -cert sha256rsa.pem -key
sha256rsakey.pem -<PROTOCOLS> -msg
```

On WRSSL-Board and teraterm on OPENSSL-PC:

2. Run WRSSL as client using following command in WRSSL-PC teraterm terminal.

```
->nm_client"5","-<PROTOCOLS>","-cipher","AES256-SHA","-
connect","<OPENSSL_PC_IPAddr>:443"
```

Expected result:

SSL connection should establish successfully. Verify by typing in a test message on the client terminal and seeing the same on server terminal and vice versa. Try to close connection from client. Connection should get closed without problems.

CASE: A15ECDH-ECDSA-AES128-SHA256

On OPENSSL-LINUX -PC:

1.Open terminal, change directory to openssl-1.0.1s and run openssl server as shown below

e.g.

```
$cd/home/user1/openssls/openssl-1.0.1s
$./apps/openssl s_server -accept 443 -cert sha256ecdsa.pem -key
       sha256ecdsakey.pem -<PROTOCOLS> -msg
```

On WRSSL-Board and teraterm on OPENSSL-PC:

2. Run WRSSL as client using following command in WRSSL-PC teraterm terminal.

```
->nm_client"5","-<PROTOCOLS>","-cipher","ECDH-ECDSA-AES128-SHA256","-
connect","<OPENSSL_PC_IPAddr>:443"
```

Expected result:

SSL connection should establish successfully. Verify by typing in a test message on the client terminal and seeing the same on server terminal and vice versa. Try to close connection from client. Connection should get closed without problems.

CASE: A16ECDH-RSA-AES128-SHA256

On OPENSSL-LINUX -PC:

1.Open terminal, change directory to openssl-1.0.1s and run openssl server as shown below

e.g.

```
$cd/home/user1/openssls/openssl-1.0.1s
$./apps/openssl s_server -accept 443 -cert sha256rsa.pem -key
sha256rsakey.pem -<PROTOCOLS> -msg
```

On WRSSL-Board and teraterm on OPENSSL-PC:

2.Run WRSSL as client using following command in WRSSL PC teraterm terminal.

```
->nm_client"5","-<PROTOCOLS>","-cipher","ECDH-RSA-AES128-SHA256","-
 connect","<OPENSSL_PC_IPAddr>:443"
```

Expected result:

SSL connection should establish successfully. Verify by typing in a test message on the client terminal and seeing the same on server terminal and vice versa. Try to close connection from client. Connection should get closed without problems.

CASE: A17ECDH-ECDSA-AES128-SHA

On OPENSSL-LINUX -PC:

1.Open terminal, change directory to openssl-1.0.1s and run openssl server as shown below e.g.

```
$cd/home/user1/openssls/openssl-1.0.1s
$./apps/openssl s_server -accept 443 -cert sha256ecdsa.pem -key
sha256ecdsakey.pem -<PROTOCOLS> -msg
```

On WRSSL-Board and tearterm on OPENSSL-PC:

2.Run WRSSL as client using following command in WRSSL-PC teraterm terminal.

```
->nm_client"5","-<PROTOCOLS>","-cipher","ECDH-ECDSA-AES128-SHA","-
connect","<OPENSSL_PC_IPAddr>:443"
```

Expected result:

SSL connection should establish successfully. Verify by typing in a test message on the client terminal and seeing the same on server terminal and vice versa. Try to close connection from client. Connection should get closed without problems.

CASE: A18ECDH-RSA-AES128-SHA

On OPENSSL-LINUX -PC:

1.Open terminal, change directory to openssl-1.0.1s and run openssl server as shown below e.g.

```
$cd/home/user1/openssls/openssl-1.0.1s
$./apps/openssl s_server -accept 443 -cert ecdhcert.pem -key
ecdhpubkey.pem -<PROTOCOLS> -msg
```

On WRSSL-Board and teraterm on OPENSSL-PC:

2.Run WRSSL as client using following command in WRSSL PC teraterm terminal.

```
->nm_client"5","-<PROTOCOLS>","-cipher","ECDH-RSA-AES128-SHA","-
connect","<OPENSSL_PC_IPAddr>:443"
```

Expected result:

SSL connection should establish successfully. Verify by typing in a test message on the client terminal and seeing the same on server terminal and vice versa. Try to close connection from client. Connection should get closed without problems.

/EOD