F5 TMOS v11.6/v12.0 – SSL/TLS Profile Cipher Cheat Sheet v0.1 – Protocols and keywords (1/4)

F5 TMOS supports cipher specifications for several purposes. The cheat sheet covers methods to define ciphers for client-ssl profiles and must not be understand as a recommendation for settings.

A list of supported ciphers and available protcols will be provided during the inital SSL handshake by the client in the CLIENT_HELLO message. The server now looks up it´s own ordered cipher list and picks the first match. That´s why the order of list elements in the servers cipher string matters.

It is recommended to leave the original client-ssl profile untouched and to create a new "master" client-ssl profile with modified ciphers to be used as parent for specific client-ssl profiles.

Ciphers or sets of ciphers can be defined by using sets and/or combinations of protocols, keywords and literal cipher suites. Multiple sets can be combined in an ordered list using the colon ':', comma ',' or space '␣' character as separator. Protocols, methods for key exchange and authentication, bulk crypto and message authentication can be generally excluded by using the exclamation mark '!' character as a prefix. Using the plus '+' character as prefix to a keyword results in a lowered priority. Using the minus '-' character as a prefix to a keyword generally disables the usage but still allows following specific definitions. TMOS allows to sort the resulting list based on the bulk crypto strength. Definitions are not case sensitive.

TMOS cipher syntax differs slightly from other environments and may differ between TMOS versions.

Ciphers can be tested from the TMOS advanced shell (using single quotes for the evaluated cipher/list):

`tmm --clientciphers '`*`your-cipher-string-here`*`'`

Evaluating ciphers in the shell does not modify the configuration.

Configuration changes need to be done in the client-ssl profile settings (advanced properties).

The 1st example shows two fully defined ciphers of protocol, key exchange authentication, bulk crypo ('AES-GCM' is a single keyword) and message authentication in preferred order and separated by a colon character:

```
$ tmm --clientciphers 'TLSv1_2+ECDHE+AES-GCM+SHA256:TLSv1_2+ECDHE+3DES+SHA'
       ID SUITE                        BITS PROT   METHOD CIPHER  MAC     KEYX
 0: 49199 ECDHE-RSA-AES128-GCM-SHA256 128  TLS1.2 Native AES-GCM SHA256  ECDHE_RSA
 1: 49170 ECDHE-RSA-DES-CBC3-SHA      168  TLS1.2 Native DES     SHA     ECDHE_RSA
```

The 2nd example returns exactly the same results by using the same combination of keywords in lower case letters separated by a space character (quotes only required for command line testing; not used in profile):

```
$ tmm --clientciphers 'tlsv1_2+ecdhe+aes-gcm+sha256 tlsv1_2+ecdhe+3des+sha'
```

Configure ciphers by combining keywords using the plus '+' character by considering protocol cipher support. (Cipher support is protocol dependent. I.e. cipher suites using AES-GCM bulk crypto are available in TLS v1.2 only.):

Protocol	+ KeyX/Auth	+ BulkCrypto	+ MessageAuth
SSLv2	RSA *	NULL	MD5
SSLv3	ADH	RC4	SHA
TLSv1	EDH or DHE *	DES	SHA256
TLSv1_1	DHE_DSS	3DES	SHA384
TLSv1_2	ECDH_RSA	AES	
DTLSv1	ECDH_ECDSA	AES-GCM	
	ECDHE *	CAMELLIA **	
	ECDHE_ECDSA		

Notes:
* for RSA Auth
** v12.0 only

Always cross check ciphers with your IT Security!

Ciphers can be also defined by using a subset of combined keywords only, sorted according to bulk crypto strength (bits!) in ascending or descending order, lowering the keyword priority and by excluding keywords.

The 3rd example shows a partially defined cipher of combined key exchange and authentication, bulk crypo and message authentication (allowing all available protocol versions and both AES128 and AES256) followed by the exclusion of TLS v1, DTLS v1 and SSL v3, lowered priority of TLS v1.1 and ordered by bulk crypto bit length (note different output order depending on placing the @STRENGTH switch):

```
$ tmm --clientciphers 'RSA+AES+SHA:!TLSv1:!SSLv3:!DTLSv1:+TLSv1_1:@STRENGTH'
     ID  SUITE                BITS PROT    METHOD  CIPHER    MAC     KEYX
 0:  53  AES256-SHA           256  TLS1.2  Native  AES       SHA     RSA
 1:  53  AES256-SHA           256  TLS1.1  Native  AES       SHA     RSA
 2:  47  AES128-SHA           128  TLS1.2  Native  AES       SHA     RSA
 3:  47  AES128-SHA           128  TLS1.1  Native  AES       SHA     RSA
$ tmm --clientciphers 'RSA+AES+SHA:!TLSv1:!SSLv3:!DTLSv1:@STRENGTH:+TLSv1_1'
     ID  SUITE                BITS PROT    METHOD  CIPHER    MAC     KEYX
 0:  53  AES256-SHA           256  TLS1.2  Native  AES       SHA     RSA
 1:  47  AES128-SHA           128  TLS1.2  Native  AES       SHA     RSA
 2:  53  AES256-SHA           256  TLS1.1  Native  AES       SHA     RSA
 3:  47  AES128-SHA           128  TLS1.1  Native  AES       SHA     RSA
```

The 4th example returns exactly the same results by using a specific and ordered combination of keywords:

```
$ tmm --clientciphers 'TLSv1_2+RSA+AES+SHA:TLSv1_1+RSA+AES+SHA:@STRENGTH '
$ tmm --clientciphers 'TLSv1_2+RSA+AES+SHA:TLSv1_1+RSA+AES+SHA'
```

List of additional characters and keywords to combine, sort, prioritize and exclude list entries and list of aliases containing complete lists of ciphers (result will vary depending on TMOS version):

Options	Explanation
: , '␣'	Use as separator between keywords, combined keywords and literal ciphers to form a list
+	Use to combine keywords to form a cipher, i.e. 'TLSv1_2+RSA+AES+SHA'
!	Use as prefix to exclude the property represented by the keyword, i.e. SSL v3 and DTLS v1 are excluded from the resulting list by using 'RSA+AES+SHA:!SSLv3:!DTLSv1'
+	Use as prefix to lower priority, i.e. 'RSA+AES+SHA:!SSLv3:!DTLSv1:+TLSv1_1:+TLSv1' allows all protocols but puts TLS v1.1 and TLS v1.0 protocols to the end of the list
-	Use as prefix to disable and re-enable suites, i.e. 'DEFAULT:-DES:DES+TLSv1_2:DES+TLSv1_1' allows all DEFAULT ciphers but uses DES in combination with TLS v1.1 and TLS v1.1 only. (The suite DES does not cover CBC usage in suites of 3DES and AES!)
@STRENGTH	Sort the resulting cipher list according to bulk crypto bit length in descending order
@SPEED	Sort the resulting cipher list according to bulk crypto bit length in ascending order

Aliases	Explanation
NATIVE	List of ciphers supported by F5 TMOS internal protocol stack
COMPAT *	List of additional ciphers not natively supported in the F5 TMOS internal protocol stack
ALL	List of all available ciphers. Does not include NULL. Does not include SSL v3 in TMOS v12.0
DEFAULT	TMOS dependent list of ciphers. Default setting in client-ssl profile
EXPORT	List of ciphers with 40 bit ** and 56 bit *** bulk crypto algorithm
LOW	List of ciphers with 64 bit bulk crypto algorithm
MEDIUM	List of ciphers with 128 bit bulk crypto algorithm
HIGH	List of ciphers with 168 bit / 192 bit **** and 256 bit bulk crypto algorithm

Notes:
* changes in v12.0
** use EXPORT40
*** use EXPORT56
**** version dep.

Always cross check ciphers with your IT Security!

A complete list of supported ciphers (depending on TMOS version) can be dumped from command line:

```
tmm --clientciphers 'ALL:COMPAT:COMPAT+SSLv2:COMPAT+SSLv3:SSLv3:NULL'
```

A sorted list of literal ciphers suites (by ignoring protocol version) can be dumped from command line:

```
tmm --clientciphers 'ALL:COMPAT:COMPAT+SSLv2:COMPAT+SSLv3:SSLv3:NULL' | \
awk '/^[[:blank:]]*[0-9]+/ {print $3}' | sort | uniq
```

Literal cipher suite strings are a ready-to-use combinations of key exchange and auth method, bulk crypto and message authentication and may be used in definitions with or without a specific protocol. A merged list of literal cipher suites of F5 TMOS v11.6 and v12.0 can be found on the reference page of this cheat sheet.

Cipher suite support is protocol dependent! I.e. ciphers using AES-GCM are supported in TLS v1.2 only.

The 5th example shows a combination of protocol and literal cipher suite string (subset of 1st example):

```
$ tmm --clientciphers 'TLSv1_2+ECDHE-RSA-AES128-GCM-SHA256'
       ID SUITE                       BITS PROT    METHOD CIPHER  MAC    KEYX
 0: 49199 ECDHE-RSA-AES128-GCM-SHA256 128  TLS1.2 Native AES-GCM SHA256 ECDHE_RSA
```

The 6th example shows the resulting list of defining a cipher by the literal cipher suite string only without specifying the protocol (lists all protocols supported natively in the TMOS protocol stack):

```
$ tmm --clientcipher 'RC4-MD5'
     ID  SUITE            BITS PROT    METHOD  CIPHER    MAC     KEYX
 0:   4  RC4-MD5          128  SSL3    Native  RC4       MD5     RSA
 1:   4  RC4-MD5          128  TLS1    Native  RC4       MD5     RSA
 2:   4  RC4-MD5          128  TLS1.1  Native  RC4       MD5     RSA
 3:   4  RC4-MD5          128  TLS1.2  Native  RC4       MD5     RSA
```

The 7th example shows the resulting list of allowing a cipher suite to be used in compatibility mode as well:

```
$ tmm --clientcipher 'COMPAT+RC4-MD5:RC4-MD5'
     ID  SUITE            BITS PROT    METHOD  CIPHER    MAC     KEYX
 0:   0  RC4-MD5          128  SSL2    Compat  RC4       MD5     RSA
 1:   4  RC4-MD5          128  SSL3    Native  RC4       MD5     RSA
 2:   4  RC4-MD5          128  TLS1    Native  RC4       MD5     RSA
 3:   4  RC4-MD5          128  TLS1.1  Native  RC4       MD5     RSA
 4:   4  RC4-MD5          128  TLS1.2  Native  RC4       MD5     RSA
```

(COMPAT mode can be combined with cipher suites like EXP-EDH-RSA-DES-CBC-SHA and keywords like RC2.)

The 8th example adds the export version of RC4-MD5 to the results (requires the literal cipher suite string):

```
$ tmm --clientcipher 'COMPAT+RC4-MD5:RC4-MD5:EXP-RC4-MD5'
     ID  SUITE            BITS PROT    METHOD  CIPHER    MAC     KEYX
 0:   0  RC4-MD5          128  SSL2    Compat  RC4       MD5     RSA
 1:   4  RC4-MD5          128  SSL3    Native  RC4       MD5     RSA
 2:   4  RC4-MD5          128  TLS1    Native  RC4       MD5     RSA
 3:   4  RC4-MD5          128  TLS1.1  Native  RC4       MD5     RSA
 4:   4  RC4-MD5          128  TLS1.2  Native  RC4       MD5     RSA
 5:   3  EXP-RC4-MD5      40   SSL3    Native  RC4       MD5     RSA
 6:   3  EXP-RC4-MD5      40   TLS1    Native  RC4       MD5     RSA
```

The 9th example shows a combination of mode [method], protocol and cipher suite
(to limit the resulting list to TLS v1 in the example; otherwise SSL v3 would be included):

```
$ tmm --clientcipher 'COMPAT+TLSv1+EXP-EDH-RSA-DES-CBC-SHA'
     ID  SUITE                      BITS PROT    METHOD  CIPHER  MAC     KEYX
 0:  20  EXP-EDH-RSA-DES-CBC-SHA    40   TLS1    Compat  DES     SHA     EDH/RSA
```

F5 TMOS v11.6/v12.0 – SSL/TLS Profile Cipher Cheat Sheet v0.1 – Reference (4/4)

RSA Key/Cert with RSA KeyX:

AES128-GCM-SHA256
AES128-SHA
AES128-SHA256
AES256-GCM-SHA384
AES256-SHA
AES256-SHA256
CAMELLIA128-SHA
CAMELLIA256-SHA
DES-CBC3-MD5
DES-CBC3-SHA
DES-CBC-MD5
DES-CBC-SHA
EXP1024-DES-CBC-SHA
EXP1024-RC4-SHA
EXP-DES-CBC-SHA
EXP-RC2-CBC-MD5
EXP-RC4-MD5
NULL-MD5
NULL-SHA
RC2-CBC-MD5
RC4-MD5
RC4-SHA

RSA Key/Cert with EDH KeyX:

DHE-RSA-AES128-GCM-SHA256
DHE-RSA-AES128-SHA
DHE-RSA-AES128-SHA256
DHE-RSA-AES256-GCM-SHA384
DHE-RSA-AES256-SHA
DHE-RSA-AES256-SHA256
DHE-RSA-CAMELLIA128-SHA
DHE-RSA-CAMELLIA256-SHA
DHE-RSA-DES-CBC3-SHA
DHE-RSA-DES-CBC-SHA
EXP-EDH-RSA-DES-CBC-SHA

RSA Key/Cert with ECDH KeyX:

ECDH-RSA-AES128-GCM-SHA256
ECDH-RSA-AES128-SHA
ECDH-RSA-AES128-SHA256
ECDH-RSA-AES256-GCM-SHA384
ECDH-RSA-AES256-SHA
ECDH-RSA-AES256-SHA384
ECDH-RSA-DES-CBC3-SHA

RSA Key/Cert with ECDHE KeyX:

ECDHE-RSA-AES128-CBC-SHA
ECDHE-RSA-AES128-GCM-SHA256
ECDHE-RSA-AES128-SHA256
ECDHE-RSA-AES256-CBC-SHA
ECDHE-RSA-AES256-GCM-SHA384
ECDHE-RSA-AES256-SHA384
ECDHE-RSA-DES-CBC3-SHA

ECDSA Key/Cert with ECDH KeyX:

ECDH-ECDSA-AES128-GCM-SHA256
ECDH-ECDSA-AES128-SHA
ECDH-ECDSA-AES128-SHA256
ECDH-ECDSA-AES256-GCM-SHA384
ECDH-ECDSA-AES256-SHA
ECDH-ECDSA-AES256-SHA384
ECDH-ECDSA-DES-CBC3-SHA

ECDSA Key/Cert with ECDHE KeyX:

ECDHE-ECDSA-AES128-GCM-SHA256
ECDHE-ECDSA-AES128-SHA
ECDHE-ECDSA-AES128-SHA256
ECDHE-ECDSA-AES256-GCM-SHA384
ECDHE-ECDSA-AES256-SHA
ECDHE-ECDSA-AES256-SHA384
ECDHE-ECDSA-DES-CBC3-SHA

DSA Key/Cert with DHE KeyX:

DHE-DSS-AES128-GCM-SHA256
DHE-DSS-AES128-SHA
DHE-DSS-AES128-SHA256
DHE-DSS-AES256-GCM-SHA384
DHE-DSS-AES256-SHA
DHE-DSS-AES256-SHA256
DHE-DSS-CAMELLIA128-SHA
DHE-DSS-CAMELLIA256-SHA

No Key/Cert with DH KeyX:

ADH-AES128-GCM-SHA256
ADH-AES128-SHA
ADH-AES256-GCM-SHA384
ADH-AES256-SHA
ADH-DES-CBC3-SHA
ADH-DES-CBC-SHA
ADH-RC4-MD5
EXP-ADH-DES-CBC-SHA
EXP-ADH-RC4-MD5

Separators, Prefixes, Options:

: or , or ' '	-
+	@STRENGTH
!	@SPEED

Protocol:

SSLv2
SSLv3
TLSv1
TLSv1_1
TLSv1_2
DTLSv1

KeyX/Auth:

RSA
ADH
EDH or DHE
DHE_DSS
ECDH_RSA
ECDH_ECDSA
ECDHE
ECDHE_ECDSA

BulkCrypto:

NULL
RC2
RC4
DES
3DES
AES
AES-GCM
CAMELLIA

MessageAuth (MAC):

MD5
SHA
SHA256
SHA384

Aliases (Suites):

NATIVE
COMPAT
ALL
DEFAULT
LOW
MEDIUM
HIGH
EXPORT or EXP
EXPORT40
EXPORT56

List of Abbreviations:

ADH	Anonymous Diffie-Hellman
AES	Advanced Encryption Standard
CBC	Cipher Block Chaining
DES	Data Encryption Standard
DHE/EDH	Diffie-Hellman Ephemeral
DSA/DSS	Digital Signature Algorithm, Digital Signature Standard
DES	Data Encryption Standard
EC	Elliptic Curve
GCM	Galois/Counter Mode
MD	Message Digest
RSA	Rivest-Shamir-Adleman
SHA	Secure Hash Algorithm

Always cross check ciphers with your IT Security!