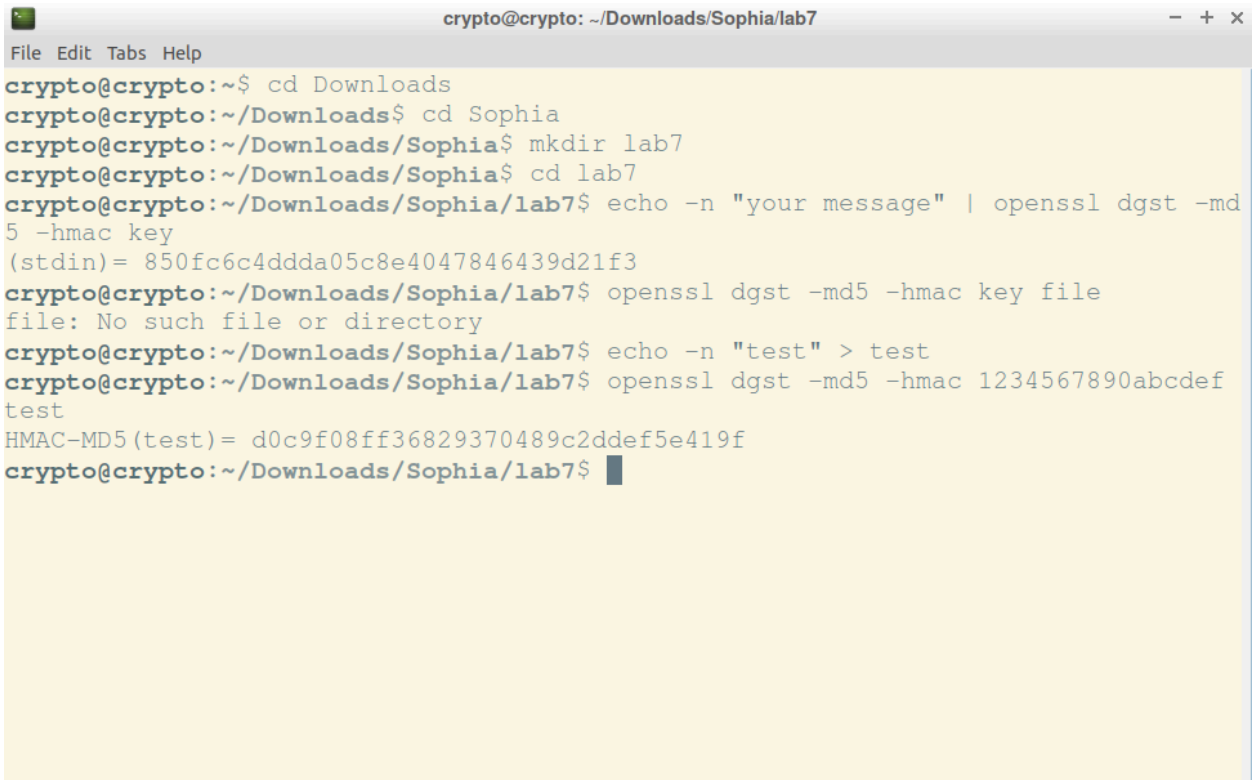
MAC and SSL/TLS handshake

Generating HMAC using keys

Task 1 - Practice HMAC

- Key in hex
- MAC functions need keys for authentication



Q: Compared to hash functions, why MAC can resist man-in-the middle attack?

- MAC needs secret key shared between sender/receiver
- Hash functions - doesn't need secret key, attacker can compute msg hash and alter it

Q: Suppose we have a MAC function with c binary digits MAC code and k binary digits key, what are the levels of effort of the brute-force attacks on the MAC code and the key? What is the best strategy for attackers?

- Effort of brute force attacks on MAC
  - C = binary digits
  - 2^c = attempts, probability of guessing correct answer is ½^c
- Effort of brute force attacks on key
  - K = binary digits key
  - 2^K = attempts
- Best strategy for attackers
  - C < K - forging MAC is easier than brute force attacking key
  - K < C - easier to find key than brute force attacking MAC

Task 2 - Practice HMAC on files with different sizes

```
crypto@crypto:~/Downloads/Sophia/lab7$ head -c 500M /dev/urandom > xlfile
crypto@crypto:~/Downloads/Sophia/lab7$  openssl dgst -md5 -hmac 1234567890abcdef
 test
HMAC-MD5(test)= d0c9f08ff36829370489c2ddef5e419f
crypto@crypto:~/Downloads/Sophia/lab7$ openssl dgst -md5 -hmac 1234567890abcdef
xlfile
HMAC-MD5(xlfile)= b92969041485e9ce1b5a9a58ff190753
crypto@crypto:~/Downloads/Sophia/lab7$
```

Q: Does the file size affect the length of HMAC values? Why?

- No, HMAC fixed length output

Task 3 - Practice HMAC with different keys and file contents

```
crypto@crypto:~/Downloads/Sophia/lab7$ openssl dgst -sha256 -hmac 1 xlfile
HMAC-SHA256(xlfile)= 2c8e0f7bd405bb9c06e6497ef5982a054751b0d21163c951bd4f9dd2638
2940d
crypto@crypto:~/Downloads/Sophia/lab7$ openssl dgst -sha256 -hmac 2 xlfile
HMAC-SHA256(xlfile)= 85f18d18d1ff8805aba384a10592cc8a3e1110872f0d8efbdf135b21ad4
ab030
crypto@crypto:~/Downloads/Sophia/lab7$ openssl dgst -sha256 -hmac 1234567890 - 1
23456789012345678901234567890123456789012345678901234567890 xlfile
123456789012345678901234567890123456789012345678901234567890: No such file or directory
HMAC-SHA256(xlfile)= e8a9c72ba5eb6dd3910b995cb2fde59b45ff35b0f76d78d62607b82d120
d38a6
crypto@crypto:~/Downloads/Sophia/lab7$ tail -c 5 xlfile
crypto@crypto:~/Downloads/Sophia/lab7$
crypto@crypto:~/Downloads/Sophia/lab7$ echo -n "0" >> xlfile
crypto@crypto:~/Downloads/Sophia/lab7$ tail -c 5 xlfile
crypto@crypto:~/Downloads/Sophia/lab7$
crypto@crypto:~/Downloads/Sophia/lab7$ openssl dgst -sha256 -hmac 1 xlfile
HMAC-SHA256(xlfile)= 4c5d1814b0e57d0bd5c597af2ac8be88e9a20122cb4120412072a05165e
e4b69
crypto@crypto:~/Downloads/Sophia/lab7$
```

Q: Will different keys generate different HMAC values on the same file? Is there any limitation of the length of the key, why? What can you summarise from this task? Which security property does this feature ensure?

- Different keys generates different HMAC values - small change in key = big change in HMAC output
- No limitations on length of key
    - Recommended key length = block size of hash function
    - Too long = key will be hashed first to reduce length
    - Too short = key will be padded with zeros
- Summary:
    - HMAC values depend on key and file contents - any change will affect HMAC
- Security property - HMAC ensures msg authentication + integrity (key dependent, tampering on file contents easily evident))

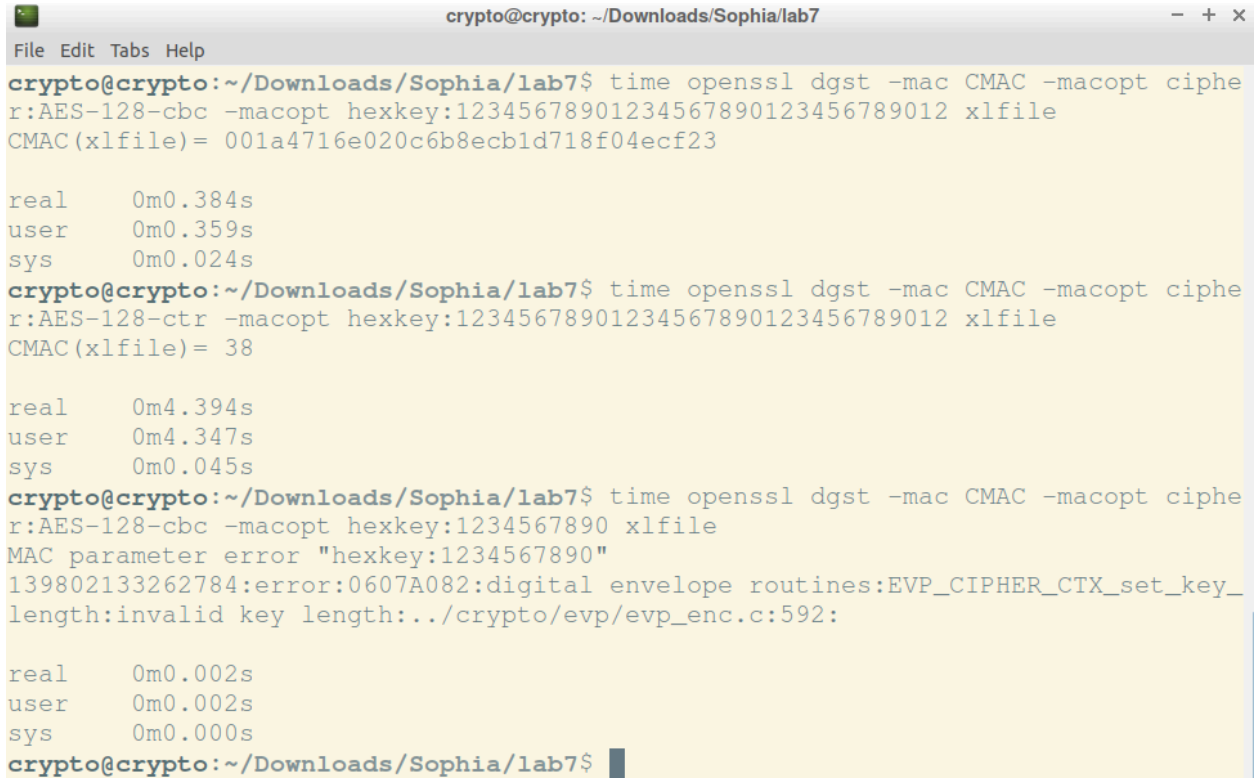
Task 4 - Try HMAC with different hash algorithms

```
crypto@crypto:~/Downloads/Sophia/lab7$ openssl dgst -sha1 -hmac 1 xlfile
HMAC-SHA1(xlfile)= 3a8ac53da02b2f4281cc5f8cc0c2202366b1ee27
crypto@crypto:~/Downloads/Sophia/lab7$ openssl dgst -sha256 -hmac 1 xlfile
HMAC-SHA256(xlfile)= 4c5d1814b0e57d0bd5c597af2ac8be88e9a20122cb4120412072a05165e
e4b69
crypto@crypto:~/Downloads/Sophia/lab7$ openssl dgst -sha512-hmac 1 xlfile
dgst: Unknown digest sha512-hmac
dgst: Use -help for summary.
crypto@crypto:~/Downloads/Sophia/lab7$ openssl dgst -sha512 hmac 1 xlfile
hmac: No such file or directory
1: No such file or directory
SHA512(xlfile)= 911c2235d5c587a996a26cf73c301e1ec3603a3dd83aaf72ece1c5ecab40993f
a2977f95f7bee2773a4ef65371afb8a9b6787f8ff08847dc267006400f079af5
crypto@crypto:~/Downloads/Sophia/lab7$ openssl dgst -sha224 -hmac 1 xlfile
HMAC-SHA224(xlfile)= 772c1671d698dfab5fe91bb32d59650f57b45b6143f6915cb81e2cb9
crypto@crypto:~/Downloads/Sophia/lab7$
```

Q: Why HMAC is compatible with different hash algorithms?

- Independent from hash function, only relies on standard structure of hash function (inner/outer padding)
- HMAC flexible - enables developers to choose which hash function to use

Task 5 - Try CMAC

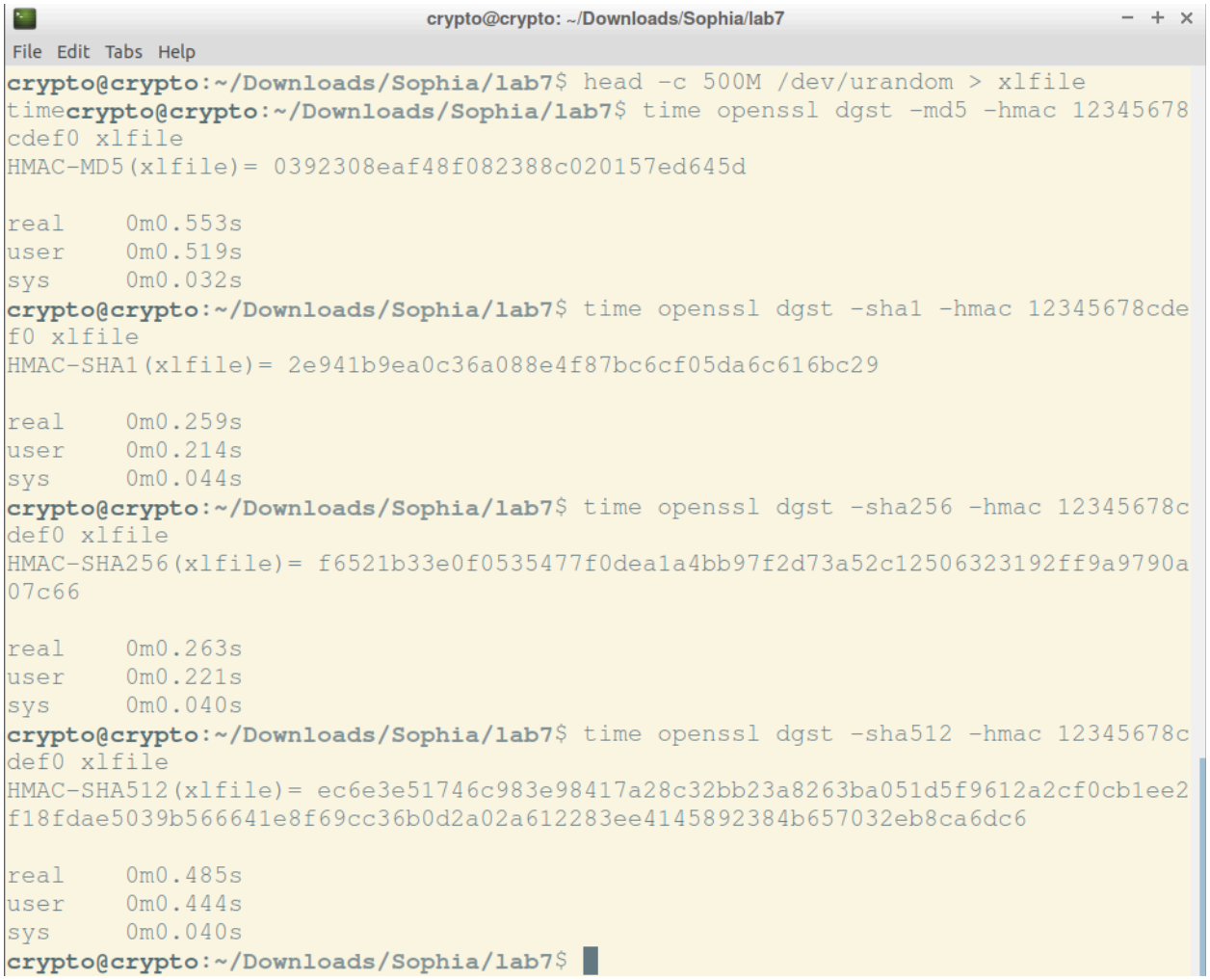


Q: Read the slide of CMAC (page 16) and explain why the last two commands are failed

- Command1: CMAC doesn't do CTR mode (stream cipher mode)
    - Designed for block cipher modes (CBC)
- Command2: key length must be 16 byte, 128 bit

Efficiency test

Task 1 - Compare the time efficiency of HMAC functions with different hash functions



Q: How long do these HMAC functions run? Which one is the fastest?

- Less than a second, fastest is MD5: 0.032 seconds

Task 2 - Compare the time efficiency of HMAC functions and corresponding hash functions

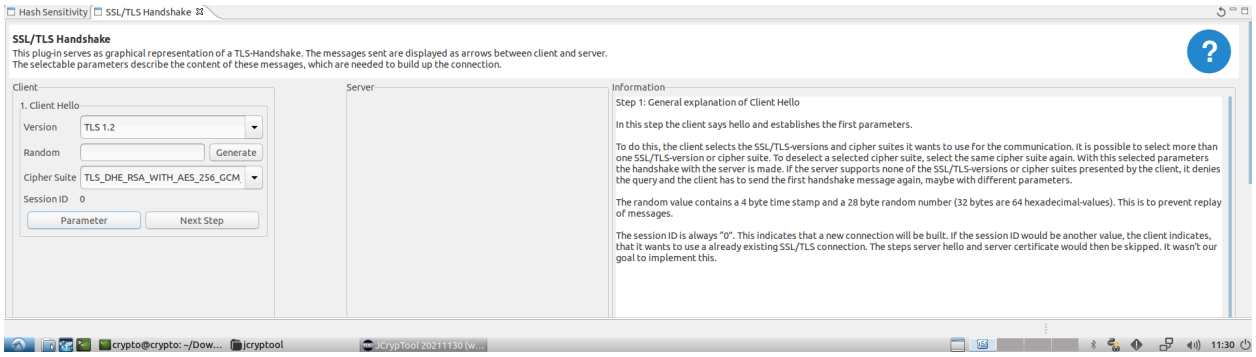
```
crypto@crypto:~/Downloads/Sophia/lab7$ head -c 500M /dev/urandom > xlfile
crypto@crypto:~/Downloads/Sophia/lab7$ time openssl dgst -sha512 -hmac 12345678c
def0 xlfile
HMAC-SHA512(xlfile)= d87bf80d8ba0489189848d00a7eb715aecd366d26f93cdf208007903bdb
f306566909208a50ea14ad84ef2093691233704b9677278173a2ac709f7e860db8755

real    0m0.487s
user    0m0.438s
sys     0m0.048s
crypto@crypto:~/Downloads/Sophia/lab7$ time openssl dgst -sha512 xlfile
SHA512(xlfile)= 688b78178a0d66648784560a8c635fa637b94e1497967f5a81111a318ced2a91
e1c7f5d38a07e00a6c06308504ae9dc6f2225a919436899dd09ffd56c68515f8

real    0m0.502s
user    0m0.455s
sys     0m0.044s
crypto@crypto:~/Downloads/Sophia/lab7$
```

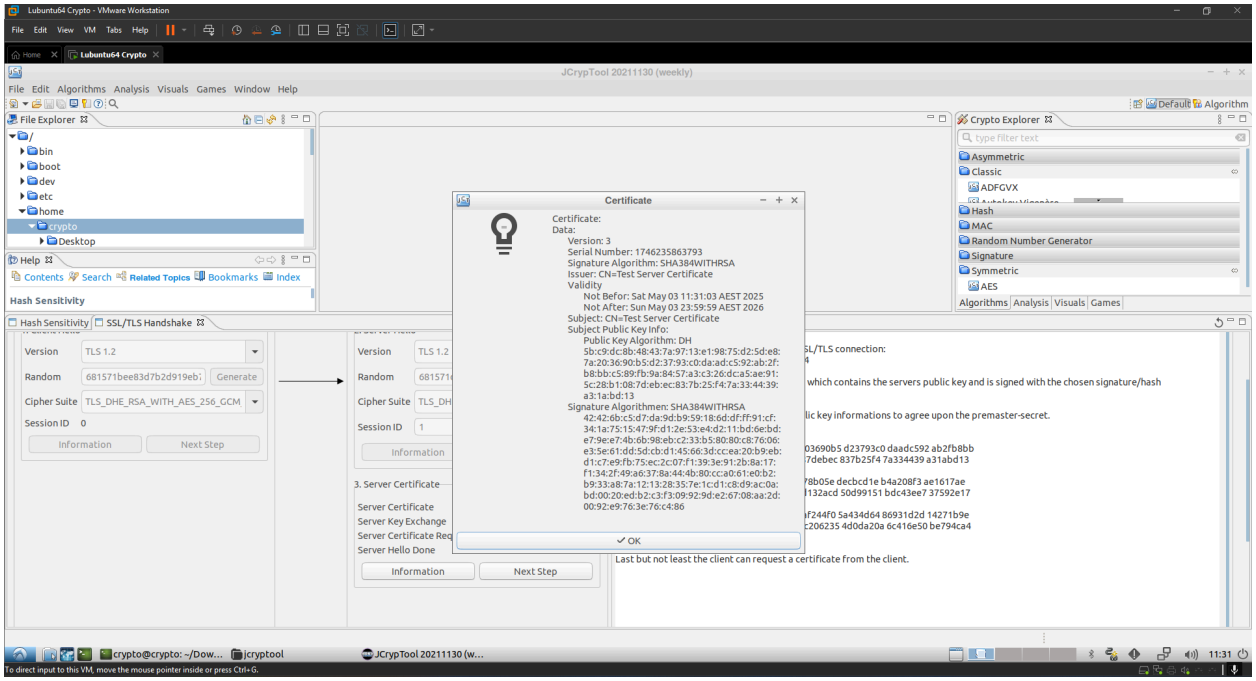
Q: Does the running time of HMAC function and corresponding hash function differ much, why?

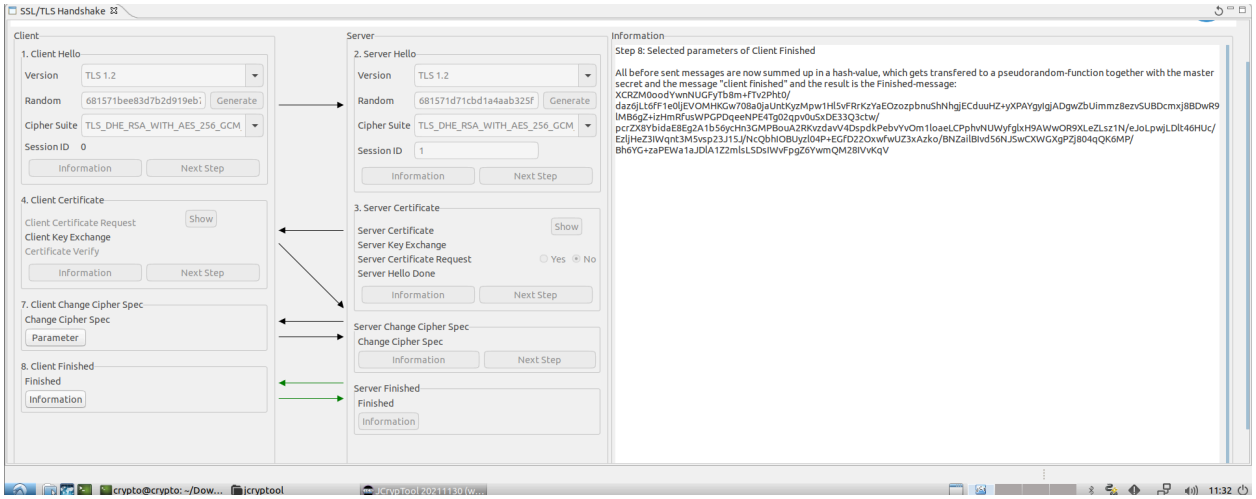
- HMAC function takes twice as long - performs hash function twice + padding for security

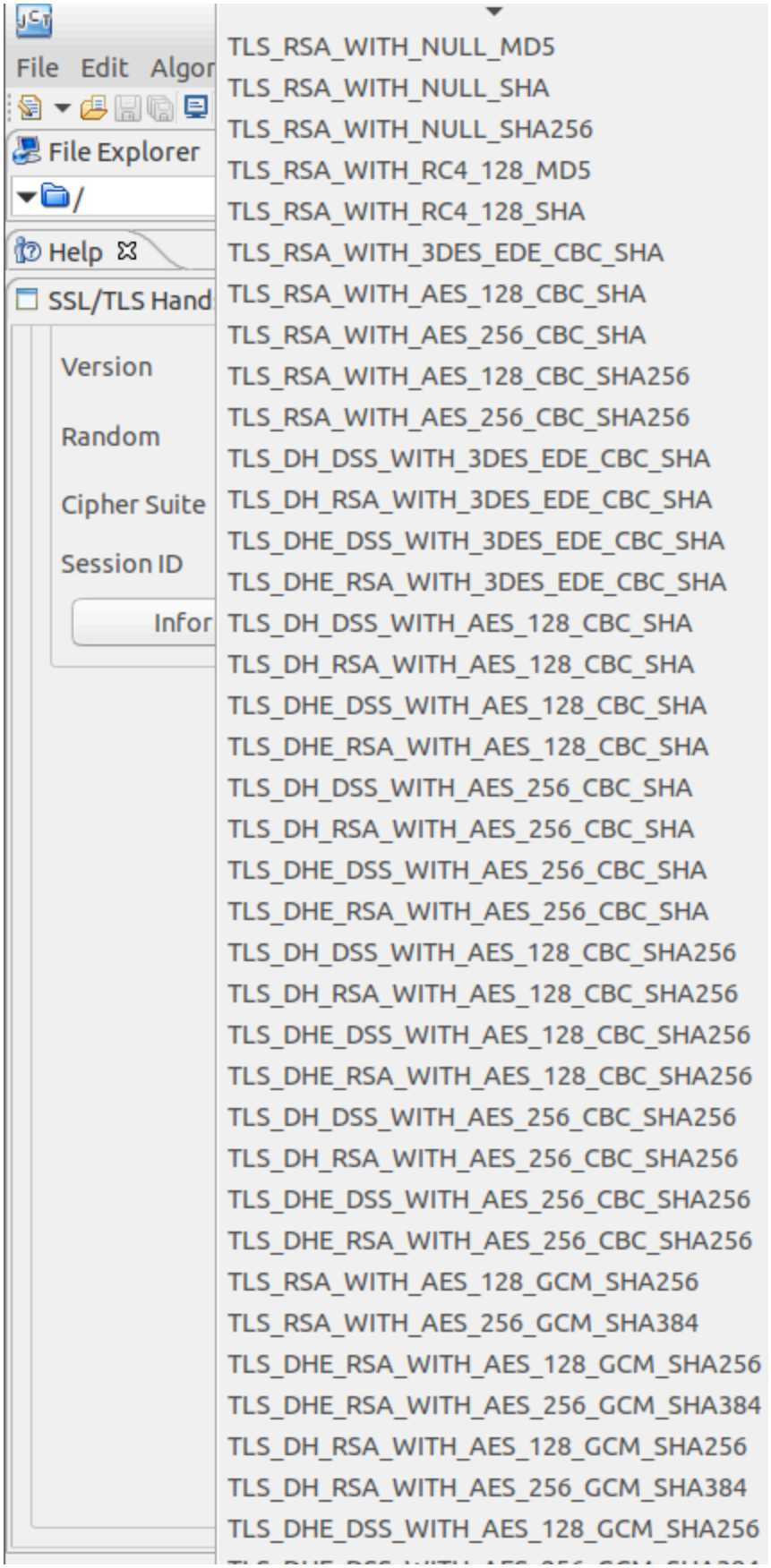
SSL/TLS Handshake

- Secure communication channel








File Edit Algor
File Explorer
/
Help
SSL/TLS Hand
Version
Random
Cipher Suite
Session ID
Infor
TLS_RSA_WITH_NULL_MD5
TLS_RSA_WITH_NULL_SHA
TLS_RSA_WITH_NULL_SHA256
TLS_RSA_WITH_RC4_128_MD5
TLS_RSA_WITH_RC4_128_SHA
TLS_RSA_WITH_3DES_EDE_CBC_SHA
TLS_RSA_WITH_AES_128_CBC_SHA
TLS_RSA_WITH_AES_256_CBC_SHA
TLS_RSA_WITH_AES_128_CBC_SHA256
TLS_RSA_WITH_AES_256_CBC_SHA256
TLS_DH_DSS_WITH_3DES_EDE_CBC_SHA
TLS_DH_RSA_WITH_3DES_EDE_CBC_SHA
TLS_DHE_DSS_WITH_3DES_EDE_CBC_SHA
TLS_DHE_RSA_WITH_3DES_EDE_CBC_SHA
TLS_DH_DSS_WITH_AES_128_CBC_SHA
TLS_DH_RSA_WITH_AES_128_CBC_SHA
TLS_DHE_DSS_WITH_AES_128_CBC_SHA
TLS_DHE_RSA_WITH_AES_128_CBC_SHA
TLS_DH_DSS_WITH_AES_256_CBC_SHA
TLS_DH_RSA_WITH_AES_256_CBC_SHA
TLS_DHE_DSS_WITH_AES_256_CBC_SHA
TLS_DHE_RSA_WITH_AES_256_CBC_SHA
TLS_DH_DSS_WITH_AES_128_CBC_SHA256
TLS_DH_RSA_WITH_AES_128_CBC_SHA256
TLS_DHE_DSS_WITH_AES_128_CBC_SHA256
TLS_DHE_RSA_WITH_AES_128_CBC_SHA256
TLS_DH_DSS_WITH_AES_256_CBC_SHA256
TLS_DH_RSA_WITH_AES_256_CBC_SHA256
TLS_DHE_DSS_WITH_AES_256_CBC_SHA256
TLS_DHE_RSA_WITH_AES_256_CBC_SHA256
TLS_RSA_WITH_AES_128_GCM_SHA256
TLS_RSA_WITH_AES_256_GCM_SHA384
TLS_DHE_RSA_WITH_AES_128_GCM_SHA256
TLS_DHE_RSA_WITH_AES_256_GCM_SHA384
TLS_DH_RSA_WITH_AES_128_GCM_SHA256
TLS_DH_RSA_WITH_AES_256_GCM_SHA384
TLS_DHE_DSS_WITH_AES_128_GCM_SHA256

Q: Which cipher suite do you prefer? Explain the cipher suite and give your reason choosing the cipher suite.

- TLS_AES_256_GCM_SHA384
    - Strong encryption using 256 bit keys = more secure
    - GCM - galois counter mode = resistant to padding, timing attacks
    - SHA384 = reduces risk of collision, preimage attacks