

## Project 2 – openSSL

Name: Muhfat Alam, Pennapa Aekaram, Ahmed M Shehab

Professor: Dr. Matluba Khodjaeva

Subject: CSCI 360

Date: August 8, 2022

## Project 2 (OpenSSL)

OpenSSL is an open-source command line tool that is commonly used to generate private keys, create CSRs, install your SSL/TLS certificate, and identify certificate information.

### Task 1, Getting started: Start your PC (Windows or Linux) with an OpenSSL installation

- Start the OpenSSL command line.** We used the commands “`openssl`” to open the source commands. The `openssl` program provides a rich variety of commands each of which often has a wealth of options and arguments

```
[pennapa.nate@localhost ~]$ openssl
help:

Standard commands
asn1parse      ca          ciphers       cmp
cms            crl         crl2pkcs7   dgst
dhparam        dsa         dsaparam     ec
ecparam        enc         engine       errstr
fipsinstall   gendsa     genpkey     genrsa
help           info        kdf          list
mac            nseq        ocsp         passwd
pkcs12         pkcs7      pkcs8       pkey
pkeyparam     pkeyutl    prime       rand
rehash         req         rsa          rsautl
s_client       s_server   s_time      sess_id
smime          speed      spkac      srp
storeutl      ts         verify     version
x509

Message Digest commands (see the `dgst' command for more details)
blake2b512    blake2s256  md2          md4
md5           rmd160     sha1         sha224
sha256        sha3-224   sha3-256    sha3-384
sha3-512     sha384     sha512     sha512-224
sha512-256   shake128   shake256   sm3

Cipher commands (see the `enc' command for more details)
aes-128-cbc   aes-128-ecb  aes-192-cbc  aes-192-ecb
aes-256-cbc   aes-256-ecb  aria-128-cbc  aria-128-cfb
aria-128-cfb1 aria-128-cfb8  aria-128-ctr  aria-128-ecb
aria-128-ofb  aria-192-cbc  aria-192-cfb  aria-192-cfb1
aria-192-cfb8 aria-192-ctr  aria-192-ecb  aria-192-ofb
aria-256-cbc  aria-256-cfb  aria-256-cfb1  aria-256-cfb8
aria-256-ctr  aria-256-ecb  aria-256-ofb  base64
bf             bf-cbc      bf-cfb      bf-ecb
bf-ofb        camellia-128-cbc  camellia-128-ecb  camellia-192-cbc
camellia-192-ecb camellia-256-cbc  camellia-256-ecb  cast
cast-cbc      cast5-cbc   cast5-cfb   cast5-ecb
cast5-ofb     des          des-cbc     des-cfb
des-ecb       des-edc     des-edc-cbc  des-edc-cfb
des-edc-ofb   des-edc3    des-edc3-cbc  des-edc3-cfb
des-edc3-ofb  des-ofb     des3        desx
idea          idea-cbc   idea-cfb   idea-ecb
idea-ofb      rc2          rc2-40-cbc  rc2-64-cbc
rc2-cbc       rc2-cfb    rc2-ecb   rc2-ofb
rc4           rc4-40     rc5         rc5-cbc
rc5-cfb      rc5-ecb   rc5-ofb   seed
seed-cbc      seed-cfb   seed-ecb   seed-ofb
zlib
```

## b. List commands by type

Now we see that all the standard commands, message digest commands, and cipher command after we use openssl. Next is we will use **openssl list -commands** to see all the list commands

```
[pennapa.nate@localhost ~]$ openssl list -commands
asn1parse          ca                  ciphers           cmp
cms                crl                 crl2pkcs7        dgst
dhparam            dsa                 dsaparam         ec
ecparam            enc                 engine           errstr
fipsinstall        gendsa             genpkey          genrsa
help               info                kdf              list
mac                nseq                ocsp             passwd
pkcs12             pkcs7              pkcs8            pkey
pkeyparam          pkeyutl            prime            rand
rehash             req                 rsa              rsautl
s_client           s_server           s_time           sess_id
smime              speed               spkac            srp
storeutl          ts                 verify           version
x509

[pennapa.nate@localhost ~]$ █
```

Then next we will use **openssl list -cipher-commands** to see all the list of ciphertext that you have on your operating system

```
[pennapa.nate@localhost ~]$ openssl list -cipher-commands
aes-128-cbc       aes-128-ecb      aes-192-cbc      aes-192-ecb
aes-256-cbc       aes-256-ecb      aria-128-cbc    aria-128-cfb
aria-128-cfb1     aria-128-cfb8    aria-128-ctr    aria-128-ecb
aria-128-ofb      aria-192-cbc    aria-192-cfb    aria-192-cfb1
aria-192-cfb8     aria-192-ctr    aria-192-ecb    aria-192-ofb
aria-256-cbc      aria-256-cfb    aria-256-cfb1   aria-256-cfb8
aria-256-ctr      aria-256-ecb    aria-256-ofb    camellia-128-cbc
camellia-128-ecb  camellia-192-cbc camellia-192-ecb camellia-256-cbc
camellia-256-ecb  des-edede       des-edede-cbc  des-edede-cfb
des-edede-ofb     des-edede3      des-edede3-cbc des-edede3-cfb
des-edede3-ofb    des3
```

Next is using **openssl list -digest-commands** to see all list all digest commands. These commands are used to generate as well as verify signatures.

```
[pennapa.nate@localhost ~]$ openssl list -digest-commands
blake2b512        blake2s256      md5              sha1
sha224            sha256          sha3-224        sha3-256
sha3-384          sha3-512        sha384          sha512
sha512-224        sha512-256     shake128       shake256
sm3
```

## c. Use the help to find out more about OpenSSL

Next, we will be using **openssl help**. This command helps to find out the further details.

```
[pennapa.nate@localhost ~]$ openssl help
help:

Standard commands
asn1parse      ca          ciphers       cmp
cms            crl         crl2pkcs7   dgst
dhparam        dsa         dsaparam     ec
ecparam        enc         engine       errstr
fipsinstall    gendsa     genpkey     genrsa
help           info        kdf          list
mac            nseq        ocsp         passwd
pkcs12         pkcs7      pkcs8       pkey
pkeyparam     pkeyutl    prime       rand
rehash         req         rsa          rsautl
s_client       s_server   s_time      sess_id
smime          speed      spkac      srp
storeutl      ts         verify     version
x509

Message Digest commands (see the `dgst' command for more details)
blake2b512    blake2s256   md2          md4
md5            rmd160      sha1         sha224
sha256         sha3-224   sha3-256    sha3-384
sha3-512      sha384     sha512     sha512-224
sha512-256   shake128   shake256   sm3

Cipher commands (see the `enc' command for more details)
aes-128-cbc   aes-128-ecb  aes-192-cbc  aes-192-ecb
aes-256-cbc   aes-256-ecb  aria-128-cbc  aria-128-cfb
aria-128-cfb1  aria-128-cfb8  aria-128-ctr  aria-128-ecb
aria-128-ofb   aria-192-cbc  aria-192-cfb  aria-192-cfb1
aria-192-cfb8  aria-192-ctr  aria-192-ecb  aria-192-ofb
aria-256-cbc   aria-256-cfb  aria-256-cfb1  aria-256-cfb8
aria-256-ctr   aria-256-ecb  aria-256-ofb  base64
bf              bf-cbc      bf-cfb      bf-ecb
bf-ofb         camellia-128-cbc  camellia-128-ecb  camellia-192-cbc
camellia-192-ecb  camellia-256-cbc  camellia-256-ecb  cast
cast-cbc       cast5-cbc   cast5-cfb   cast5-ecb
cast5-ofb      des         des-cbc    des-cfb
des-ecb        des-edc     des-edc-cbc  des-edc-cfb
des-edc        des-edc     des-edc-cbc  des-edc-cfb
des-edc-ofb    des-ede3    des-ede3-cbc  des-ede3-cfb
des-ede3-ofb   des-ofb    des3        desx
idea           idea-cbc   idea-cfb   idea-ecb
idea-ofb       rc2         rc2-40-cbc  rc2-64-cbc
rc2-cbc        rc2-cfb   rc2-ecb   rc2-ofb
rc4            rc4-40     rc5         rc5-cbc
rc5-cfb        rc5-ecb   rc5-ofb   seed
seed-cbc       seed-cfb   seed-ecb   seed-ofb
zlib
```

## Task2, Performance of OpenSSL

- Make a speed test on your PC-platform with the speed command. We use the “openssl speed” command to shows the speed test

```
[pennapa.nate@localhost ~]$ openssl speed
Doing md5 for 3s on 16 size blocks: 17517910 md5's in 2.99s
Doing md5 for 3s on 64 size blocks: 10304113 md5's in 3.00s
Doing md5 for 3s on 256 size blocks: 4574525 md5's in 3.00s
Doing md5 for 3s on 1024 size blocks: 1426212 md5's in 3.00s
Doing md5 for 3s on 8192 size blocks: 191421 md5's in 3.00s
Doing md5 for 3s on 16384 size blocks: 95685 md5's in 3.00s
Doing sha1 for 3s on 16 size blocks: 29939961 sha1's in 3.00s
Doing sha1 for 3s on 64 size blocks: 24879372 sha1's in 3.00s
Doing sha1 for 3s on 256 size blocks: 15635723 sha1's in 3.00s
Doing sha1 for 3s on 1024 size blocks: 5870292 sha1's in 2.99s
Doing sha1 for 3s on 8192 size blocks: 864789 sha1's in 3.00s
Doing sha1 for 3s on 16384 size blocks: 438620 sha1's in 3.00s
Doing sha256 for 3s on 16 size blocks: 31383242 sha256's in 3.00s
Doing sha256 for 3s on 64 size blocks: 26154422 sha256's in 3.00s
Doing sha256 for 3s on 256 size blocks: 15586933 sha256's in 3.00s
Doing sha256 for 3s on 1024 size blocks: 5977675 sha256's in 3.00s
Doing sha256 for 3s on 8192 size blocks: 878309 sha256's in 3.00s
Doing sha256 for 3s on 16384 size blocks: 444903 sha256's in 3.00s
Doing sha512 for 3s on 16 size blocks: 16718389 sha512's in 3.00s
Doing sha512 for 3s on 64 size blocks: 16797653 sha512's in 3.00s
Doing sha512 for 3s on 256 size blocks: 8558683 sha512's in 3.00s
Doing sha512 for 3s on 1024 size blocks: 3350119 sha512's in 3.00s
Doing sha512 for 3s on 8192 size blocks: 509089 sha512's in 3.00s
Doing sha512 for 3s on 16384 size blocks: 258928 sha512's in 3.00s
Doing hmac(md5) for 3s on 16 size blocks: 10064181 hmac(md5)'s in 3.00s
Doing hmac(md5) for 3s on 64 size blocks: 7513435 hmac(md5)'s in 3.00s
Doing hmac(md5) for 3s on 256 size blocks: 3969048 hmac(md5)'s in 3.00s
Doing hmac(md5) for 3s on 1024 size blocks: 1352173 hmac(md5)'s in 3.00s
Doing hmac(md5) for 3s on 8192 size blocks: 189733 hmac(md5)'s in 3.00s
Doing hmac(md5) for 3s on 16384 size blocks: 95822 hmac(md5)'s in 3.00s

Doing des-ede3 for 3s on 16 size blocks: 6234953 des-ede3's in 3.00s
Doing des-ede3 for 3s on 64 size blocks: 1595647 des-ede3's in 3.00s
Doing des-ede3 for 3s on 256 size blocks: 395467 des-ede3's in 3.00s
Doing des-ede3 for 3s on 1024 size blocks: 99669 des-ede3's in 3.00s
Doing des-ede3 for 3s on 8192 size blocks: 12325 des-ede3's in 3.00s
Doing des-ede3 for 3s on 16384 size blocks: 6161 des-ede3's in 3.00s
Doing aes-128-cbc for 3s on 16 size blocks: 126893909 aes-128-cbc's in 3.00s
Doing aes-128-cbc for 3s on 64 size blocks: 67939697 aes-128-cbc's in 3.00s
Doing aes-128-cbc for 3s on 256 size blocks: 17851817 aes-128-cbc's in 3.00s
Doing aes-128-cbc for 3s on 1024 size blocks: 4610737 aes-128-cbc's in 3.00s
Doing aes-128-cbc for 3s on 8192 size blocks: 579428 aes-128-cbc's in 3.00s
Doing aes-128-cbc for 3s on 16384 size blocks: 291390 aes-128-cbc's in 3.00s
Doing aes-192-cbc for 3s on 16 size blocks: 113680440 aes-192-cbc's in 3.00s
Doing aes-192-cbc for 3s on 64 size blocks: 56242598 aes-192-cbc's in 3.00s
Doing aes-192-cbc for 3s on 256 size blocks: 14856878 aes-192-cbc's in 3.00s
Doing aes-192-cbc for 3s on 1024 size blocks: 3779178 aes-192-cbc's in 3.00s
Doing aes-192-cbc for 3s on 8192 size blocks: 478608 aes-192-cbc's in 3.00s
Doing aes-192-cbc for 3s on 16384 size blocks: 239465 aes-192-cbc's in 3.00s
Doing aes-256-cbc for 3s on 16 size blocks: 104676853 aes-256-cbc's in 3.00s
Doing aes-256-cbc for 3s on 64 size blocks: 49454751 aes-256-cbc's in 3.00s
Doing aes-256-cbc for 3s on 256 size blocks: 12964720 aes-256-cbc's in 3.00s
Doing aes-256-cbc for 3s on 1024 size blocks: 3257592 aes-256-cbc's in 3.00s
Doing aes-256-cbc for 3s on 8192 size blocks: 409670 aes-256-cbc's in 3.00s
Doing aes-256-cbc for 3s on 16384 size blocks: 207147 aes-256-cbc's in 3.00s
```

```
Doing camellia-128-cbc for 3s on 16 size blocks: 31081993 camellia-128-cbc's in 3.00s
Doing camellia-128-cbc for 3s on 64 size blocks: 8035999 camellia-128-cbc's in 3.00s
Doing camellia-128-cbc for 3s on 256 size blocks: 2050957 camellia-128-cbc's in 3.00s
Doing camellia-128-cbc for 3s on 1024 size blocks: 516554 camellia-128-cbc's in 3.00s
Doing camellia-128-cbc for 3s on 8192 size blocks: 64528 camellia-128-cbc's in 3.00s
Doing camellia-128-cbc for 3s on 16384 size blocks: 32111 camellia-128-cbc's in 3.00s
Doing camellia-192-cbc for 3s on 16 size blocks: 24260512 camellia-192-cbc's in 3.00s
Doing camellia-192-cbc for 3s on 64 size blocks: 6196333 camellia-192-cbc's in 3.00s
Doing camellia-192-cbc for 3s on 256 size blocks: 1530614 camellia-192-cbc's in 3.00s
Doing camellia-192-cbc for 3s on 1024 size blocks: 384417 camellia-192-cbc's in 3.00s
Doing camellia-192-cbc for 3s on 8192 size blocks: 48289 camellia-192-cbc's in 3.00s
Doing camellia-192-cbc for 3s on 16384 size blocks: 23887 camellia-192-cbc's in 3.00s
Doing camellia-256-cbc for 3s on 16 size blocks: 23835174 camellia-256-cbc's in 3.00s
Doing camellia-256-cbc for 3s on 64 size blocks: 6082269 camellia-256-cbc's in 3.00s
Doing camellia-256-cbc for 3s on 256 size blocks: 1522761 camellia-256-cbc's in 2.99s
Doing camellia-256-cbc for 3s on 1024 size blocks: 385474 camellia-256-cbc's in 3.00s
Doing camellia-256-cbc for 3s on 8192 size blocks: 48407 camellia-256-cbc's in 3.00s
Doing camellia-256-cbc for 3s on 16384 size blocks: 23885 camellia-256-cbc's in 3.00s
Doing ghash for 3s on 16 size blocks: 232878193 ghash's in 3.00s
Doing ghash for 3s on 64 size blocks: 218374046 ghash's in 3.00s
Doing ghash for 3s on 256 size blocks: 77264851 ghash's in 3.00s
Doing ghash for 3s on 1024 size blocks: 20517089 ghash's in 3.00s
Doing ghash for 3s on 8192 size blocks: 2581362 ghash's in 3.00s
Doing ghash for 3s on 16384 size blocks: 1309961 ghash's in 3.00s
Doing rand for 3s on 16 size blocks: 6222793 rand's in 2.61s
Doing rand for 3s on 64 size blocks: 6171383 rand's in 2.51s
Doing rand for 3s on 256 size blocks: 5876957 rand's in 2.47s
Doing rand for 3s on 1024 size blocks: 5040039 rand's in 2.59s
Doing rand for 3s on 8192 size blocks: 1998884 rand's in 2.82s
Doing rand for 3s on 16384 size blocks: 1178064 rand's in 2.95s
Doing 512 bits private rsa's for 10s: 545331 512 bits private RSA's in 10.00s
Doing 512 bits public rsa's for 10s: 6838351 512 bits public RSA's in 10.00s
Doing 1024 bits private rsa's for 10s: 111203 1024 bits private RSA's in 10.00s
Doing 1024 bits public rsa's for 10s: 2469542 1024 bits public RSA's in 10.00s
Doing 2048 bits private rsa's for 10s: 17679 2048 bits private RSA's in 9.99s
Doing 2048 bits public rsa's for 10s: 727947 2048 bits public RSA's in 10.00s
Doing 3072 bits private rsa's for 10s: 6078 3072 bits private RSA's in 10.00s
Doing 3072 bits public rsa's for 10s: 338124 3072 bits public RSA's in 10.00s
Doing 4096 bits private rsa's for 10s: 2773 4096 bits private RSA's in 10.00s
Doing 4096 bits public rsa's for 10s: 192225 4096 bits public RSA's in 10.00s
Doing 7680 bits private rsa's for 10s: 337 7680 bits private RSA's in 10.03s
Doing 7680 bits public rsa's for 10s: 55865 7680 bits public RSA's in 10.00s
Doing 15360 bits private rsa's for 10s: 64 15360 bits private RSA's in 10.11s
Doing 15360 bits public rsa's for 10s: 14314 15360 bits public RSA's in 10.00s
Doing 512 bits sign dsa's for 10s: 286813 512 bits DSA signs in 9.98s
Doing 512 bits verify dsa's for 10s: 505377 512 bits DSA verify in 10.00s
Doing 1024 bits sign dsa's for 10s: 147387 1024 bits DSA signs in 9.97s
Doing 1024 bits verify dsa's for 10s: 193175 1024 bits DSA verify in 10.00s
Doing 2048 bits sign dsa's for 10s: 51637 2048 bits DSA signs in 9.94s
Doing 2048 bits verify dsa's for 10s: 55531 2048 bits DSA verify in 9.97s
Doing 224 bits sign ecdsa's for 10s: 326412 224 bits ECDSA signs in 9.95s
Doing 224 bits verify ecdsa's for 10s: 151064 224 bits ECDSA verify in 10.00s
Doing 256 bits sign ecdsa's for 10s: 577926 256 bits ECDSA signs in 9.90s
Doing 256 bits verify ecdsa's for 10s: 186511 256 bits ECDSA verify in 9.99s
Doing 384 bits sign ecdsa's for 10s: 17328 384 bits ECDSA signs in 10.00s
Doing 384 bits verify ecdsa's for 10s: 20769 384 bits ECDSA verify in 9.99s
Doing 521 bits sign ecdsa's for 10s: 47996 521 bits ECDSA signs in 9.98s
Doing 521 bits verify ecdsa's for 10s: 25420 521 bits ECDSA verify in 9.99s
Doing 224 bits ecdh's for 10s: 224933 224-bits ECDH ops in 10.00s
Doing 256 bits ecdh's for 10s: 244389 256-bits ECDH ops in 10.00s
Doing 384 bits ecdh's for 10s: 18073 384-bits ECDH ops in 10.00s
Doing 521 bits ecdh's for 10s: 43264 521-bits ECDH ops in 10.00s
Doing 253 bits ecdh's for 10s: 311897 253-bits ECDH ops in 9.99s
Doing 448 bits ecdh's for 10s: 63021 448-bits ECDH ops in 10.00s
Doing 253 bits sign Ed25519's for 10s: 330077 253 bits Ed25519 signs in 9.99s
Doing 253 bits verify Ed25519's for 10s: 119166 253 bits Ed25519 verify in 10.00s
Doing 456 bits sign Ed448's for 10s: 54637 456 bits Ed448 signs in 10.00s
Doing 456 bits verify Ed448's for 10s: 62884 456 bits Ed448 verify in 10.00s
Doing 2048 bits ffdh's for 10s: 5698 2048-bits FFDH ops in 9.98s
Doing 3072 bits ffdh's for 10s: 1816 3072-bits FFDH ops in 10.00s
```

```

version: 3.0.1
built on: Wed Mar 16 00:00:00 2022 UTC
options: bn(64,64)
compiler: gcc -fPIC -pthread -Wa,--noexecstack -Wall -O3 -O2 -flto=auto -ffat-lto-objects -fexceptions -g -frecord-gcc-switches -pipe -Wall -Werror=format-security -Wp,-D_FORTIFY_SOURCE=2 -Wp,-D_GLIBCXX_ASSERTIONS -specs=/usr/lib/rpm/redhat/redhat-hardened-cc1 -fstack-protector-strong -specs=/usr/lib/rpm/redhat/redhat-annobin-cc1 -fasynchronous-unwind-tables -fstack-clash-protection -Wa,--noexecstack -Wa,--generate-missing-build-notes=yes -specs=/usr/lib/rpm/redhat/redhat-hardened-ld -specs=/usr/lib/rpm/redhat/redhat-annobin-cc1 -DOPENSSL_USE_NODELETE -DOPENSSL_PIC -DOPENSSL_BUILDING_OPESSL -DZLIB -DNDEBUG -DPURIFY -DDEVRANDOM="\"/dev/urandom\" -DREDHAT_FIPS_VERSION="\\"3.0.1-20220316\\"" -DSYSTEM_CIPHERS_FILE="/etc/crypto-policies/back-ends/openssl.config"
CPUINFO: OPENSSL_armcap=0xff
The 'numbers' are in 1000s of bytes per second processed.
type          16 bytes     64 bytes    256 bytes   1024 bytes   8192 bytes   16384 bytes
md5           93741.32k   219821.08k   390359.47k   486813.70k   522706.94k   522567.68k
sha1          159679.79k   530759.94k   1334248.36k   2010427.76k   2361450.50k   2395450.03k
sha256         167377.29k   557961.00k   1330084.95k   2040379.73k   2398369.11k   2429763.58k
sha512         89164.74k    358349.93k   730340.95k   1143507.29k   1390152.36k   1414092.12k
hmac(md5)      53675.63k   160286.61k   338692.10k   461541.72k   518097.58k   523315.88k
des-ede3       33253.08k   34040.47k    33746.52k   34020.35k   33655.47k   33647.27k
aes-128-cbc   676767.51k   1449380.20k   1523355.05k   1573798.23k   1582224.73k   1591377.92k
aes-192-cbc   606295.68k   1199842.09k   1267786.92k   1289959.42k   1306918.91k   1307798.19k
aes-256-cbc   558276.55k   1055034.69k   1106322.77k   1111924.74k   1118672.21k   1131298.82k
camellia-128-cbc 165770.63k   171434.65k   175015.00k   176317.10k   176204.46k   175368.87k
camellia-192-cbc 129389.40k   132188.44k   130612.39k   131214.34k   131861.16k   130454.87k
camellia-256-cbc 127120.93k   129755.07k   130376.86k   131575.13k   132183.38k   130443.95k
ghash          1242017.03k   4658646.31k   6593267.29k   7003166.38k   7048839.17k   7154133.67k
rand           38147.39k    157357.97k   609109.71k   1992664.07k   5806687.14k   6542847.65k
              sign      verify      sign/s verify/s
rsa  512 bits 0.000018s 0.000001s  54533.1 683835.1
rsa 1024 bits 0.000090s 0.000004s 11120.3 246954.2
rsa 2048 bits 0.000565s 0.000014s 1769.7 72794.7
rsa 3072 bits 0.001645s 0.000030s  607.8 33812.4
rsa 4096 bits 0.003606s 0.000052s  277.3 19222.5
rsa 7680 bits 0.029763s 0.000179s   33.6  5586.5
rsa 15360 bits 0.157969s 0.000699s   6.3   1431.4
              sign      verify      sign/s verify/s
dsa  512 bits 0.000035s 0.000020s 28738.8 50537.7
dsa 1024 bits 0.000068s 0.000052s 14783.0 19317.5
dsa 2048 bits 0.000192s 0.000180s 5194.9 5569.8
              sign      verify      sign/s verify/s
224 bits ecdsa (nistp224) 0.0000s 0.0001s 32805.2 15106.4
256 bits ecdsa (nistp256) 0.0000s 0.0001s 58376.4 18669.8
384 bits ecdsa (nistp384) 0.0006s 0.0005s 1732.8 2079.0
521 bits ecdsa (nistp521) 0.0002s 0.0004s 4809.2 2544.5
              op      op/s
224 bits ecdh (nistp224) 0.0000s 22493.3
256 bits ecdh (nistp256) 0.0000s 24438.9
384 bits ecdh (nistp384) 0.0006s 1807.3
521 bits ecdh (nistp521) 0.0002s 4326.4
253 bits ecdh (X25519) 0.0000s 31220.9
448 bits ecdh (X448) 0.0002s 6302.1
              sign      verify      sign/s verify/s
253 bits EdDSA (Ed25519) 0.0000s 0.0001s 33040.7 11916.6
456 bits EdDSA (Ed448) 0.0002s 0.0002s 5463.7 6288.4
              op      op/s
2048 bits ffdh 0.0018s 570.9
3072 bits ffdh 0.0055s 181.6
4096 bits ffdh 0.0125s 80.3
6144 bits ffdh 0.0402s 24.9
8192 bits ffdh 0.0945s 10.6

```

[pennapa.nate@localhost ~]\$ █

b. Compare the results for symmetric encryption (e.g., AES-CBC) and RSA signature.

Next we will use **openssl speed rsa2048** to compare the results for symmetric encryption and RSA signature.

```
[pennapa.nate@localhost ~]$ openssl speed rsa2048
Doing 2048 bits private rsa's for 10s: 17735 2048 bits private RSA's in 10.00s
Doing 2048 bits public rsa's for 10s: 726605 2048 bits public RSA's in 10.00s
version: 3.0.1
built on: Wed Mar 16 00:00:00 2022 UTC
options: bn(64,64)
compiler: gcc -fPIC -pthread -Wa,--noexecstack -Wall -O3 -O2 -fstack-protector-strong -fexceptions -g -frecord-gcc-switches -pipe -Wall -Werror=format-security -Wp,-D_FORTIFY_SOURCE=2 -Wp,-D_GLIBCXX_ASSERTIONS -specs=/usr/lib/rpm/redhat/redhat-hardened-cc1 -fstack-protector-strong -specs=/usr/lib/rpm/redhat/redhat-annobin-cc1 -fasynchronous-unwind-tables -fstack-clash-protection -Wa,--noexecstack -Wa,--generate-missing-build-notes=yes -specs=/usr/lib/rpm/redhat/redhat-hardened-ld -specs=/usr/lib/rpm/redhat/redhat-annobin-cc1 -DOPENSSL_USE_NODELETE -DOPENSSL_PIC -DOPENSSL_BUILDING_OPENSSL -DZLIB -DNDEBUG -DPURIFY -DDEVRANDOM="\\"/dev/urandom\\" -DREDHAT_FIPS_VERSION="\\"3.0.1-20220316\\" -DSYSTEM_CIPHERS_FILE="/etc/crypto-policies/back-ends/openssl.config"
CPUINFO: OPENSSL_armcap=0xff
          sign      verify    sign/s verify/s
rsa 2048 bits 0.000564s 0.000014s   1773.5  72660.5
[pennapa.nate@localhost ~]$
```

**Taks3, Using OpenSSL from the command line interface**

Now, our goal is to create a text file that can be encrypted in three different types. For this, I have to create a directory for this project and access this directory. So, I used my terminal and went to my desire folder, and type

```
```  
mkdir Project 2 – openssl  
cd Project 2 - openssl
```

Then for the text file created, we used nano editor and named as “textFile”

```  
nano textFile

Here we typed our text, “Hi everyone, This is a group project about OpenSSL. Our goal is to encrypt and decrypt using OpenSSL.”

Then press **ctrl + X**, then **v**, and press **enter** to save this text.

```
Last login: Sat Aug 6 09:00:35 on ttys000
muhfatalam@Muhfats-MBP Project 2 - openssl % nano textFile
muhfatalam@Muhfats-MBP Project 2 - openssl % cat textFile
Hi everyone, This is a group project about openssl. Our goal is encrypt and decrypt using openssl.
muhfatalam@Muhfats-MBP Project 2 - openssl %
```

a. Create a text file with some input and encrypt it using

- I. AES-128 CBC
- II. AES-256 CTR
- III. DES

Now, we are going to encrypt with three following different types of encryption methods.

- i. AES-128 CBC
- ii. AES-256 CTR
- iii. DES

To do this encryption, In the terminal, we used three commands that are similar to each other and just changed the encryption method.

- i. For AES-128 CBC

...

```
openssl enc -aes-128-cbc -base64 -in textFile -out text_aes_128_cbc
```

...

Here are “textFile” will encrypted and it will save as “text\_aes\_128\_cbc”. Besides terminal is required for the password for extra security. For the password we used “password.” We also need to verify our password with our given password.

```
Project 2 - openssl -- zsh - 132x44

Last login: Sat Aug 6 09:00:35 on ttys000
muhfatalam@Muhfats-MBP Project 2 - openssl % nano textFile
muhfatalam@Muhfats-MBP Project 2 - openssl % cat textFile
Hi everyone, This is a group project about openssl. Our goal is encrypt and decrypt using openssl.
muhfatalam@Muhfats-MBP Project 2 - openssl enc -aes-128-cbc -base64 -in textFile -out text_aes_128_cbc
[enter aes-128-cbc encryption password:
[Verifying - enter aes-128-cbc encryption password:
muhfatalam@Muhfats-MBP Project 2 - openssl % cat text_aes_128_cbc
U2FsdGVkX1/JLHVAcBsebvPn/7nUVGY0kg0lhEg7ZHhDg2j1DN8MAdkyXcrmdvP
7BkPu1Sg6LxjZER8J247AiIbii0jnnMsW3H4T/0QDww7Cib+mxZrxVXSNd2Uxja0
l+o1HYgMkH1hd0gh07yIr4i65gBg1Emz1Mws16zlzUU=
muhfatalam@Muhfats-MBP Project 2 - openssl %
```

- ii. For AES-256 CTR

...

```
openssl enc -aes-256-ctr -base64 -in textFile -out text_aes_128_cbc
```

...

Here “textFile” will be encrypted and it will save as “text\_aes\_256\_ctr” and password we used “password”. We also need to verify our password with our given password.

```
[muhfatalam@Muhfats-MBP Project 2 - openssl % openssl enc -aes-256-ctr -base64 -in textFile -out text_aes_256_ctr
[enter aes-256-ctr encryption password:
[Verifying - enter aes-256-ctr encryption password:
[muhfatalam@Muhfats-MBP Project 2 - openssl % cat text_aes_256_ctr
U2FsdGVkX19wwwAwbqbBLXFA3s6kcGo6Gk0pBricV9zWj6rgdQYNd/vqyLnnSON
E21txZ414/DkoGrxBuGt1UhDoM4KNuJ3HSFAXt4V89vdvZl5AhEeUAfsTZBFNSOU
USPP0hDYFF7vKtzN+2C8g5aZxng=
muhfatalam@Muhfats-MBP Project 2 - openssl % ]
```

### iii. For AES-256 CTR

```
...  
openssl enc -des -base64 -in textFile -out text_des  
...
```

Here “textFile” will be encrypted and it will save as “text\_des” and password we used “password”. We also need to verify our password with our given password.

```
[muhfatalam@Muhfats-MBP Project 2 - openssl % openssl enc -des -base64 -in textFile -out text_des
[enter des-cbc encryption password:
[Verifying - enter des-cbc encryption password:
[muhfatalam@Muhfats-MBP Project 2 - openssl % cat text_des
U2FsdGVkX18S23npabsmxhbef7eJqb604tP609Xd010pnUKSwv/U4/w22eA9wxml
LNMB//xEVczlusUp1R5RyhCYgt70rsemxVnanpRAvvvdL6izh+ZjLnQ451/zk0tm
uL3ppPVzneovETAfkJQbFtyPKU7DNGNkk
muhfatalam@Muhfats-MBP Project 2 - openssl % ]
```

Now, we need to verify we will have 3 different encrypted file and one original files.

```
...  
ls  
...
```

```
[muhfatalam@Muhfats-MBP Project 2 - openssl % ls
Screenshots      textFile      text_aes_128_cbc      text_aes_256_ctr      text_des
muhfatalam@Muhfats-MBP Project 2 - openssl % ]
```

### b. Create a 2048 bit RSA public and private key.

After we verify there are three different encrypted files, and one original file are this directory now we will create a 2048-bit RSA public and private key. For this, we need to create directories for two people where they will have each other's public key and they can share encrypted text file. After getting an encrypted file they will decrypt it with their own private key. Assume one is Alice and the other one is Bob. They both need private and public keys for RSA encryption. We can follow the following commands:

For Alice,

```
...
```

```
mkdir A
```

```
ls
```

```
cd A
```

```
openssl genrsa -out keypairA.pem 2048
```

```
...
```

For Bob

...

mkdir B

ls

cd B

openssl genrsa -out keypairB.pem 2048

...

Here, for the last command genrsa is generate an RSA key and it will save as “keypairA” and “keypairB”, which will contain private and public keys.

The screenshot shows two terminal windows side-by-side. Both windows have a title bar 'B -- zsh -- 132x44'. The left window (labeled A) shows the command 'openssl genrsa -out keypairB.pem 2048' being run, with the output indicating the generation of a 2048-bit RSA private key. The right window (labeled B) shows the command 'openssl genrsa -out keypairB.pem 2048' also being run, with the output indicating the generation of a 2048-bit RSA private key. Both windows show the command 'Generating RSA private key, 2048 bit long modulus' followed by a progress bar and the message 'e is 65537 (0x10001)'.

Now, we are going to separate our public key from keypairA and keypairB.

For Alice,

...

openssl rsa -in keypairA.pem -pubout -out publicA.pem

cat publicA.pem

...

\*\*Notice that after OpenSSL we do not generate the key, we are using this RSA key to separate the file. So we used RSA after OpenSSL.

For Bob,

...

openssl rsa -in keypairB.pem -pubout -out publicB.pem

cat publicB.pem

...

The screenshot shows two terminal windows side-by-side. Both windows have a title bar 'B -- zsh -- 132x44'. The left window (labeled A) shows the command 'openssl rsa -in keypairA.pem -pubout -out publicA.pem' being run, with the output indicating the creation of a PUBLIC KEY file. The right window (labeled B) shows the command 'openssl rsa -in keypairB.pem -pubout -out publicB.pem' being run, with the output indicating the creation of a PUBLIC KEY file. Both windows show the command 'writing RSA key' followed by a progress bar and the message 'e is 65537 (0x10001)'.

#### Task4, Exchange of encrypted data.

a. Encrypt a file (e.g., a text file) with an algorithm and key length of your choice.

To exchange data, we will need to create a link between Alice and Bob to access each other's public keys by

For Alice,

10

```
In -s /Users/muhfatalam/Downloads/Summer\ 2022/CSCI\ 360/Project\ 2\ -\openssl/B/publicB.pem
```

15

11

For Bob,

11

```
In -s /Users/muhfatalam/Downloads/Summer\ 2022/CSCI\ 360/Project\ 2\ -\ openssl/A/publicA.pem
```

15

18

```
muhfatalam@Muhfats-MBP:~ % ls
keypairA.pem    publicA.pem
muhfatalam@Muhfats-MBP:~ % ln -s /Users/muhfatalam/Downloads/Summer\ 2022/CSCI\ 360/Project\ 2\ \ openssl
/B/publicB.pem
muhfatalam@Muhfats-MBP:~ % ls
keypairA.pem    publicA.pem    publicB.pem
muhfatalam@Muhfats-MBP:~ %

[muhfatalam@Muhfats-MBP:~ % openssl genrsa -out keypairB.pem 2048
Generating RSA private key, 2048 bit long modulus
.....+**+
.....+**+
e is 65537 (0x10001)
muhfatalam@Muhfats-MBP:~ % ls
keypairB.pem
muhfatalam@Muhfats-MBP:~ % openssl rsa -in keypairB.pem -pubout -out publicB.pem
writing RSA key
muhfatalam@Muhfats-MBP:~ % ls
keypairB.pem    publicB.pem
muhfatalam@Muhfats-MBP:~ % cat publicB.pem
-----BEGIN PUBLIC KEY-----
MIIBIjANBgkqhkiG9w0BAQEFAACQABIAjCpKACQAw75GJomjye0zGL9EhJWS
KQq1z6ZP7XfUqg/F1wtAwFGQDj+w8x+XNQn2u562vewUBhwmoTQAXXXHnf
ngv1geXuANuAnME16hB5j3xrve87021WmJGj83oNYncG69d1xeCOVp5DU
V5bSDQg3uOr8/B2j3BYCcVZMttt688LRu10vgwOhr2fDz+o2=+QHAAJBMKH4
9Tz1yA8y5297qfYhngMequ7dr1gRgKvoxxrPK33SBPjhEVSiqKDS0LTHbaGh21
ywIDQAAB
-----END PUBLIC KEY-----
muhfatalam@Muhfats-MBP:~ % ln -s /Users/muhfatalam/Downloads/Summer\ 2022/CSCI\ 360/Project\ 2\ \ openssl/A/public
B/publicB.pem
muhfatalam@Muhfats-MBP:~ % ls
keypairB.pem    publicA.pem    publicB.pem
muhfatalam@Muhfats-MBP:~ % ]
```

Now for demonstration purposes, we will make an encrypted text file with bob's public key in the Alice directory. In the text file, we typed "Hi, My account number is 1234567890. And we all going to John Jay" and saved it as "textFile"

11

## nano textFile

cat textFile

11

Now we are going to encrypt our “textFile” with Bob’s public key “public.pem” and save it as “encryptedFile”.

11

```
openssl rsautl -encrypt -in textFile -out encrptedFile -inkey publicB.pem -pubin
```

|s

```
cat encryptedFile
```

11

```

muhfatalam@Muhfats-MBP A % nano textField
muhfatalam@Muhfats-MBP A % cat textField
Hi, My account number is 1234567890. And we all going to John Jay
muhfatalam@Muhfats-MBP A % openssl rsautl -encrypt -in textField -inkey publicB.pem -pu bin
muhfatalam@Muhfats-MBP A % ls
encryptedFile  keypairA.pem  publicA.pem  publicB.pem  textField
muhfatalam@Muhfats-MBP A % cat encryptedFile
/g9Pd X()/??n??m?n~?n
?E?
hn?=?d#?`k
???
??????@W?????zUR????,??F??uAw??h??<??+H?;???????@?8RQ?&?cq?QaU??^?yA?-G?V~-?S?c?{.Yj??ng?O????$.??
??U?u,?ay?Zt?74?
?C?<?U?K?e{?An?i?2??K?FD?6=?
?<<?7?V?
?hAi?SH??
muhfatalam@Muhfats-MBP A % 

```

**b. Exchange the file and the necessary credentials for decryption (i.e., algorithm, key) with your neighbor.**

Now, Bob, it is going to copy this encrypted file from Alice and save his directory with the name “received.”

...

```
cp /Users/muhfatalam/Downloads/Summer\ 2022/CSCI\ 360/Project\ 2\ -\openssl/A/encryptedFile received
```

...

**c. Decrypt the secret of your neighbor.**

For decryption, Bob will use his own private key and save it as “decryptedFile.”

...

```
openssl rsautl -decrypt -in received -out decryptedFile -inkey keypairB.pem
```

ls

```
cat decryptedFile
```

...

```

muhfatalam@Muhfats-MBP A % nano textField
muhfatalam@Muhfats-MBP A % cat textField
Hi, My account number is 1234567890. And we all going to John Jay
muhfatalam@Muhfats-MBP A % openssl rsautl -encrypt -in textField -inkey publicB.pem -pu bin
muhfatalam@Muhfats-MBP A % ls
encryptedFile  keypairA.pem  publicA.pem  publicB.pem  textField
muhfatalam@Muhfats-MBP A % cat encryptedFile
/g9Pd X()/??n??m?n~?n
?E?
hn?=?d#?`k
???
??????@W?????zUR????,??F??uAw??h??<??+H?;???????@?8RQ?&?cq?QaU??^?yA?-G?V~-?S?c?{.Yj??ng?O????$.??
??U?u,?ay?Zt?74?
?C?<?U?K?e{?An?i?2??K?FD?6=?
?<<?7?V?
?hAi?SH??
muhfatalam@Muhfats-MBP A % cp /Users/muhfatalam/Downloads/Summer\ 2022/CSCI\ 360/Project\ 2\ -\openssl/A/encryptedFile
usage: cp [-R [-H | -L | -P]] [-f] [-n] [-aclpsvx] source_file target_file
cp [-R [-H | -L | -P]] [-f] [-n] [-aclpsvx] source_file ... target_directory
muhfatalam@Muhfats-MBP A % ls
keypairB.pem  publicA.pem  publicB.pem
muhfatalam@Muhfats-MBP A % cp /Users/muhfatalam/Downloads/Summer\ 2022/CSCI\ 360/Project\ 2\ -\openssl/A/encryptedFile
received
muhfatalam@Muhfats-MBP A % ls
keypairB.pem  publicA.pem  publicB.pem  received
muhfatalam@Muhfats-MBP A % openssl rsautl -decrypt -in received -out decryptedFile -inkey keypairB.pem
muhfatalam@Muhfats-MBP A % ls
decryptedFile  keypairB.pem  publicA.pem  publicB.pem  received
muhfatalam@Muhfats-MBP A % cat decryptedFile
/g9Pd X()/??n??m?n~?n
?E?
hn?=?d#?`k
???
??????@W?????zUR????,??F??uAw??h??<??+H?;???????@?8RQ?&?cq?QaU??^?yA?-G?V~-?S?c?{.Yj??ng?O????$.??
4y?Zt?74?
?C?<?U?K?e{?An?i?2??K?FD?6=?
?<<?7?V?
?hAi?SH??
muhfatalam@Muhfats-MBP A % 

```

For security purposes, before Alice encrypts the file with Bob's public key, we can separate our private key with any kind of encryption methods like AES or DES from "keypairA.pem" to save it with "privateA.pem." Here we are using "AES-256-CBC". For passwords, we are using "password."

11

```
openssl rsa -in keypairA.pem -aes-256-cbc -out privateA.pem
```

11

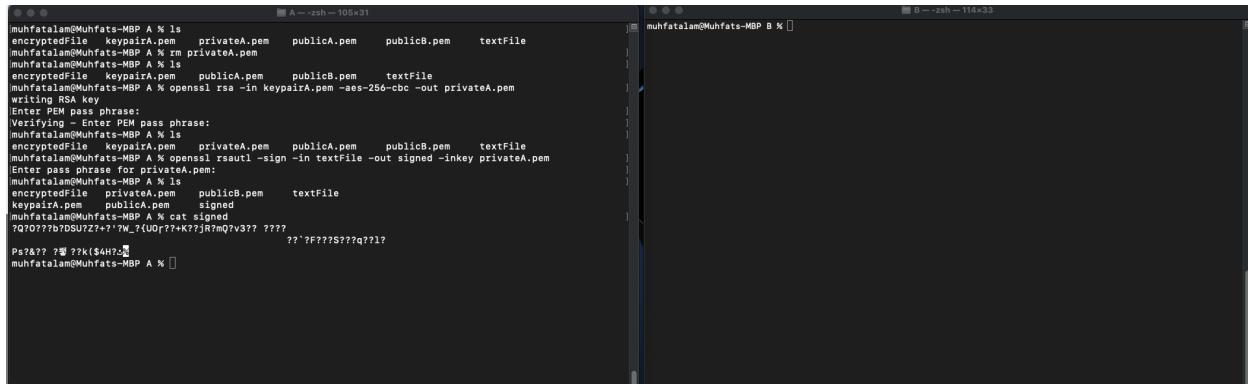
After verifying the password, we will sign our “textFile”, and save it as a “signed” name using the inkey “privateA.pem.”

11

```
openssl rsautl -sign -in textFile -out signed -inkey privateA.pem
```

11

From now every time, we used our private key we need to use password.



Now Bob needs to copy this file and save it as a “singed” name.

三

```
cp /Users/muhfatalam/Downloads/Summer\ 2022/CSCI\ 360/Project\ 2\ -\ openssl/A/signed  
signed
```

1c

15

After copying the file Bob needs to verify this signed file and save it as “signedFile” using the public key of Alice “publicA.pem.”

三

```
openssl rsautl -verify -in signed -out signedFile -inkey publicA.pem -pubin
```

1

Now we can open this file:

...

ls

cat signedFile

...

The screenshot shows two terminal windows side-by-side. The left window (A) is on a Mac OS X system (zsh shell) and demonstrates the process of generating keys, encrypting a file, signing it, and then decrypting and verifying the signed file. The right window (B) is on a Windows system (cmd shell) and shows the verification of the signed file using OpenSSL's RSAUTL command.

**Terminal A (Mac OS X):**

```
muhfatalam@Muhfats-MBP ~ % ls
encryptedFile keypairA.pem privateA.pem publicA.pem publicB.pem textField
muhfatalam@Muhfats-MBP ~ % rm privateA.pem
muhfatalam@Muhfats-MBP ~ % ls
encryptedFile keypairA.pem publicA.pem publicB.pem textField
muhfatalam@Muhfats-MBP ~ % openssl rsa -in keypairA.pem -aes-256-cbc -out privateA.pem
Enter PEM pass phrase:
Verifying - Enter PEM pass phrase:
muhfatalam@Muhfats-MBP ~ %
encryptedFile keypairA.pem privateA.pem publicA.pem publicB.pem textField
muhfatalam@Muhfats-MBP ~ % openssl rsautl -sign -in textField -out signed -inkey privateA.pem
Enter PEM pass phrase for privateA.pem:
muhfatalam@Muhfats-MBP ~ %
encryptedFile privateA.pem publicB.pem textField
keypairA.pem publicA.pem signed
muhfatalam@Muhfats-MBP ~ % cat signed
?Q?0????b?DSU?Z??/?_W_?d?U?r??+K?j?R?mQ?v3?? ?????
??`?F???S???q??1
Ps?&?? ?@ ??k{$_4H?z
muhfatalam@Muhfats-MBP ~ %
```

**Terminal B (Windows cmd):**

```
muhfatalam@Muhfats-MBP ~ % cp /Users/muhfatalam/Downloads/Summer\ 2022/OSCI\ 360/Project\ 2\ \openssl/A/signed signed
muhfatalam@Muhfats-MBP ~ %
ls
decryptedFile keypairA.pem publicA.pem publicB.pem received signed
muhfatalam@Muhfats-MBP ~ % openssl rsautl -verify -in signed -out signedFile -inkey publicA.pem -pubin
muhfatalam@Muhfats-MBP ~ %
decryptedFile keypairA.pem publicA.pem publicB.pem received signed signedfile
muhfatalam@Muhfats-MBP ~ % cat signedFile
Hi, My account number is 1234567890. And we all going to John Jay
muhfatalam@Muhfats-MBP ~ %
```

When we used any encryption method for RSA it will become more secure besides the public and private keys.