TRABALHO INDIVIDUAL SOBRE OPENSSL

Fernando Paladini, Segurança em Computação (INE5429) 01/10/2016

1) Comando asn1parse

O comando ASN1PARSE da ferramenta OpenSSL tem como finalidade analisar (*parsear*, *varrer*) e extrair dados de estruturas descritas em ASN.1 (*Abstract Syntax Notation 1*). Dentre as opções disponíveis para esse comando estão:

-inform <formato>: o formato do arquivo de entrada, que pode ter como valor DER (binário) ou PEM (arquivo codificado em base64). O valor padrão é PEM.

-in <arquivo>: o único argumento é o arquivo de entrada (<arquivo>).

-out <arquivo>: o arquivo de saída onde os dados codificados em DER serão armazenados. Se esta opção não estiver presente, nenhuma saída não será gerada.

-noout: não gera uma saída.

-offset <numero>: o *offset* dado antes de começar a varrer (*parsear*) o arquivo. Por padrão é o começo do arquivo.

-length <numero>: número de bytes para varrer (*parsear*). Por padrão é até o final do arquivo.

-i: indenta a saída de acordo com a profundidade da estrutura dada como arquivo de entrada.

-dump: fazer o dump de dados em formato hexadecimal.

-strparse <offset>: varre (*parse*) os octetos do conteúdo do objeto ASN.1 a partir do <offset> fornecido. Essa opção pode ser usada múltiplas vezes.

-genstr <string>, -genconf <arquivo>: gera dados codificados em <string>, <arquivo> ou em ambos utilizando o formato ASN.1.
Exemplo de comando:

```
1 #
2 $ openssl asn1parse -inform PEM -in directbooking_co.crt -i
3     0:d=0  hl=4 l=1401 cons: SEQUENCE
4     4:d=1  hl=4 l=1121 cons:  SEQUENCE
```

```
 5     8:d=2  hl=2 l=   3 cons:   cont [ 0 ]
 6    10:d=3  hl=2 l=   1 prim:    INTEGER           :02
 7    13:d=2  hl=2 l=  17 prim:   INTEGER           :↩
        CAFBF1C9B17442D25C179688F808A2EC
 8    32:d=2  hl=2 l=  13 cons:   SEQUENCE
 9    34:d=3  hl=2 l=   9 prim:    OBJECT            :sha256WithRSAEncryption
10    45:d=3  hl=2 l=   0 prim:    NULL
11    47:d=2  hl=3 l= 144 cons:   SEQUENCE
12    50:d=3  hl=2 l=  11 cons:    SET
13    52:d=4  hl=2 l=   9 cons:     SEQUENCE
14    54:d=5  hl=2 l=   3 prim:      OBJECT            :countryName
15    59:d=5  hl=2 l=   2 prim:      PRINTABLESTRING   :GB
16    63:d=3  hl=2 l=  27 cons:    SET
17    65:d=4  hl=2 l=  25 cons:     SEQUENCE
18    67:d=5  hl=2 l=   3 prim:      OBJECT            :stateOrProvinceName
19    72:d=5  hl=2 l=  18 prim:      PRINTABLESTRING   :Greater Manchester
20    92:d=3  hl=2 l=  16 cons:    SET
21    94:d=4  hl=2 l=  14 cons:     SEQUENCE
22    96:d=5  hl=2 l=   3 prim:      OBJECT            :localityName
23   101:d=5  hl=2 l=   7 prim:      PRINTABLESTRING   :Salford
24   110:d=3  hl=2 l=  26 cons:    SET
25   112:d=4  hl=2 l=  24 cons:     SEQUENCE
26   114:d=5  hl=2 l=   3 prim:      OBJECT            :organizationName
27   119:d=5  hl=2 l=  17 prim:      PRINTABLESTRING   :COMODO CA Limited
28   138:d=3  hl=2 l=  54 cons:    SET
29   140:d=4  hl=2 l=  52 cons:     SEQUENCE
30   142:d=5  hl=2 l=   3 prim:      OBJECT            :commonName
31   147:d=5  hl=2 l=  45 prim:      PRINTABLESTRING   :COMODO RSA Domain ↩
       Validation Secure Server CA
32   194:d=2  hl=2 l=  30 cons:   SEQUENCE
33   196:d=3  hl=2 l=  13 prim:    UTCTIME           :160204000000Z
34   211:d=3  hl=2 l=  13 prim:    UTCTIME           :170203235959Z
35   226:d=2  hl=2 l=  97 cons:   SEQUENCE
36   228:d=3  hl=2 l=  33 cons:    SET
37   230:d=4  hl=2 l=  31 cons:     SEQUENCE
38   232:d=5  hl=2 l=   3 prim:      OBJECT            :↩
       organizationalUnitName
39   237:d=5  hl=2 l=  24 prim:      PRINTABLESTRING   :Domain Control ↩
       Validated
40   263:d=3  hl=2 l=  33 cons:    SET
41   265:d=4  hl=2 l=  31 cons:     SEQUENCE
42   267:d=5  hl=2 l=   3 prim:      OBJECT            :↩
       organizationalUnitName
43   272:d=5  hl=2 l=  24 prim:      PRINTABLESTRING   :PositiveSSL Multi-↩
       Domain
44   298:d=3  hl=2 l=  25 cons:    SET
45   300:d=4  hl=2 l=  23 cons:     SEQUENCE
```

```
46   302:d=5  hl=2 l=   3 prim:      OBJECT            :commonName
47   307:d=5  hl=2 l=  16 prim:      PRINTABLESTRING   :directbooking.co
48   325:d=2  hl=4 l= 290 cons:   SEQUENCE
49   329:d=3  hl=2 l=  13 cons:    SEQUENCE
50   331:d=4  hl=2 l=   9 prim:     OBJECT            :rsaEncryption
51   342:d=4  hl=2 l=   0 prim:     NULL
52   344:d=3  hl=4 l= 271 prim:    BIT STRING
53   619:d=2  hl=4 l= 506 cons:   cont [ 3 ]
54   623:d=3  hl=4 l= 502 cons:    SEQUENCE
55   627:d=4  hl=2 l=  31 cons:     SEQUENCE
56   629:d=5  hl=2 l=   3 prim:      OBJECT            :X509v3 Authority Key ↩
        Identifier
57   634:d=5  hl=2 l=  24 prim:      OCTET STRING      [HEX DUMP]:3016801490↩
        AF6A3A945A0BD890EA125673DF43B43A28DAE7
58   660:d=4  hl=2 l=  29 cons:     SEQUENCE
59   662:d=5  hl=2 l=   3 prim:      OBJECT            :X509v3 Subject Key ↩
        Identifier
60   667:d=5  hl=2 l=  22 prim:      OCTET STRING      [HEX DUMP]:04140↩
        C56EA09820200C68F0CFE499D592F02DE5055F1
61   691:d=4  hl=2 l=  14 cons:     SEQUENCE
62   693:d=5  hl=2 l=   3 prim:      OBJECT            :X509v3 Key Usage
63   698:d=5  hl=2 l=   1 prim:      BOOLEAN           :255
64   701:d=5  hl=2 l=   4 prim:      OCTET STRING      [HEX DUMP]:030205A0
65   707:d=4  hl=2 l=  12 cons:     SEQUENCE
66   709:d=5  hl=2 l=   3 prim:      OBJECT            :X509v3 Basic ↩
        Constraints
67   714:d=5  hl=2 l=   1 prim:      BOOLEAN           :255
68   717:d=5  hl=2 l=   2 prim:      OCTET STRING      [HEX DUMP]:3000
69   721:d=4  hl=2 l=  29 cons:     SEQUENCE
70   723:d=5  hl=2 l=   3 prim:      OBJECT            :X509v3 Extended Key ↩
        Usage
71   728:d=5  hl=2 l=  22 prim:      OCTET STRING      [HEX DUMP]:301406082↩
        B06010505070301060 82B06010505070302
72   752:d=4  hl=2 l=  79 cons:     SEQUENCE
73   754:d=5  hl=2 l=   3 prim:      OBJECT            :X509v3 Certificate ↩
        Policies
74   759:d=5  hl=2 l=  72 prim:      OCTET STRING      [HEX DUMP]:3046303↩
        A060B2B06010401B23101020207302B302906082B06010505070201161D68747470733A2F2F7365637

75   833:d=4  hl=2 l=  84 cons:     SEQUENCE
76   835:d=5  hl=2 l=   3 prim:      OBJECT            :X509v3 CRL ↩
        Distribution Points
77   840:d=5  hl=2 l=  77 prim:      OCTET STRING      [HEX DUMP]:304↩
        B3049A047A045864368747470 3A2F2F63726C2E636F6D6F646F63612E636F6D2F434F4D4F444F52534

78   919:d=4  hl=3 l= 133 cons:     SEQUENCE
79   922:d=5  hl=2 l=   8 prim:      OBJECT            :Authority Information↩
```

```
          Access
80    932:d=5  hl=2 l= 121 prim:      OCTET STRING      [HEX DUMP]:3077304↩
        F06082B06010505073002864368747470 3A2F2F6372742E636F6D6F646F63612E636F6D2F434F4D4F44

81   1055:d=4  hl=2 l=  72 cons:     SEQUENCE
82   1057:d=5  hl=2 l=   3 prim:      OBJECT            :X509v3 Subject ↩
       Alternative Name
83   1062:d=5  hl=2 l=  65 prim:      OCTET STRING      [HEX DUMP]:303↩
       F8210646972656374626F6F6B696E672E636F82146170702E646972656374626F6F6B696E672E636F821

84   1129:d=1  hl=2 l=  13 cons:  SEQUENCE
85   1131:d=2  hl=2 l=   9 prim:   OBJECT            :sha256WithRSAEncryption
86   1142:d=2  hl=2 l=   0 prim:   NULL
87   1144:d=1  hl=4 l= 257 prim:  BIT STRING
```

2) Comando ciphers

O comando CIPHERS da ferramenta OpenSSL tem como finalidade converter uma lista textual de cifradores OpenSSL para uma lista ordenada de preferências de cifradores OpenSSL.
-v: opção com *verbosidade*, ou seja, imprimindo na tela muito mais informações do que está acontecendo, incluindo descrição da versão do protocolo, autenticação, encriptação, etc.

-ssl3: lista os cifradores compatíves com o SSLv3.

-tls1: lista os cifradores compatíves com TLSv1, TLSv1.1 ou TLSv1.2.

-ssl2: lista apenas os cifradores compatíves com SSLv2.

<cipherlist>: uma lista de cifradores que será convertida para para uma lista de preferências de cifradores. Se nenhum argumento for passado, então a lista de cifradores padrão será utilizada. Essa lista consiste de strings separadas por dois pontos (:), virgulas ou espaço.
Exemplo de comando:

```
1 $ openssl ciphers -v ECDH+aRSA+HIGH
2 ECDHE-RSA-AES256-GCM-SHA384 TLSv1.2 Kx=ECDH     Au=RSA  Enc=AESGCM(256) ↩
     Mac=AEAD
3 ECDHE-RSA-AES256-SHA384 TLSv1.2 Kx=ECDH     Au=RSA  Enc=AES(256)  Mac=↩
     SHA384
4 ECDHE-RSA-AES256-SHA    SSLv3 Kx=ECDH     Au=RSA  Enc=AES(256)  Mac=SHA1
5 ECDHE-RSA-AES128-GCM-SHA256 TLSv1.2 Kx=ECDH     Au=RSA  Enc=AESGCM(128) ↩
     Mac=AEAD
6 ECDHE-RSA-AES128-SHA256 TLSv1.2 Kx=ECDH     Au=RSA  Enc=AES(128)  Mac=↩
     SHA256
```

```
7  ECDHE-RSA-AES128-SHA    SSLv3 Kx=ECDH     Au=RSA  Enc=AES(128)  Mac=SHA1
8  ECDHE-RSA-DES-CBC3-SHA  SSLv3 Kx=ECDH     Au=RSA  Enc=3DES(168) Mac=SHA1
```

3) Comando dgst

O comando DGST da ferramenta OpenSSL tem como objetivo gerar uma mensagem sintetizada de um arquivo de origem, além de também poder gerar e verificar assinaturas digitais.

-c: printa na saída padrão grupos de dois digitos separados por dois pontos. Essa opção apenas é relevante se um formato de saída em hexadecimal é usado.

-d: printa na saída padrão informações de debug do comando.

-hex: a saída será gerada como um dump em hexadecimal. Esse é o caso normal para uma síntese "normal", diferente de uma assinatura digital.

-binary: gera uma saída de síntese ou assinatura digital em binário.

-r: gera uma saída síntese no formato de "coreutils", que é utilizado em programas como sha1sum.

-non-fips-allow: permite o uso de síntese não FIPS quando o mode FIPS está ativado. Não tem efeito nenhum quando o modo FIPS não está ativado.

-out <arquivo>: arquivo onde a saída será escrita. Por padrão utiliza a saída padrão.

-sign <arquivo>: assina digital o arquivo utilizando a chave privada fornecida no argumento <arquivo>.

-keyform <arg>: especifica o formato da chave utiliza para assinar a síntese. Os formatos DER, PEM, P12 e ENGINE são suportados por essa opção.

-engine <id>: use o motor fornecido no argumento <id> para operações (incluindo armazenamento da private key). Esse motor não é usado como código para algoritmos de síntese, a menos que ele também esteja especificado no arquivo de configuração.

-sigopt nm:v: passa opções para o algoritmo de assinatura usando "assinar" ou "verificar". Os nomes e valores para essas opções são especificadas para cada algoritmo.

-passin arg: A fonte da senha da chave privada.

-verify <arquivo>: verificar a assinatura usando a chave pública contida em <arquivo>. A saída pode ser "Falha na verificação" ou ainda "Verificação OK".

-prverify <arquivo>: verifica a assinatura usando a chave privada contida em <arquivo>.

-signature <arquivo>: a assinatura a ser verificada.

-hmac <chave>: cria um MAC hasheado usando o argumento <chave>.

-mac <alg>: cria um (Message Authentication Code). O algoritmo de MAC mais popular é o HMAC, mas existem outros algoritmos de MAC que não são baseados em hash, como por exemplo o algoritmo gost-mac.

-macopt <nm:v>: passa argumentos para o algoritmo MAC especificado pela opção "-mac".

-rand <file(s)>: um arquivo ou vários arquivos contendo dados aleatórios.

-non-fips-allow: ativa o uso de algoritmos não FIPS, tais como MD5.

-fips-fingerprint: computa o HMAC usando a chave especifica para certas operações OpenSSL-FIPS.

file...: arquivo ou arquivos para serem usados de entrada. Se nenhum arquivo é especificado, então a entrada padrão é utilizada.

Exemplo de comando:

```
1 # Gera um hash MD5 para o arquivo cachorro.gif
2 $ openssl dgst -md5 cachorro.gif
3 MD5(cachorro.gif)= a303ed7ce439738c2ce9f0791d9799c1
4
5 # Gera um hash SHA1 para o arquivo cachorro.gif
6 $ openssl dgst -sha1 cachorro.gif
7 SHA1(cachorro.gif)= e5cba219bad315b7d0d6e0912a2d423ee9801611
```

4) Comando enc

O comando ENC da ferramenta OpenSSL tem como objetivo fornecer meios para encriptar ou descriptografar vários blocos ou cifradores stream baseado em senhas ou explicitamente fornecidos. Encriptação e descriptografia também podem ser feitas em Base64.

-in <arquivo>: o arquivo de entrada. Por padrão o valor é a entrada padrão.

-out <arquivo>: o arquivo de saída. Por padrão o valor é saída padrão.

-pass <arg>: a fonte da senha.

-salt: usa um salt nas rotinas de derivação de chaves. Este é o valor default.

-nosalt: não usa salt nas rotinas de derivação de chaves. Está opção NÃO deve ser usado exceto para testes ou compatibilidade com versões antigas do OpenSSL e SSLeay.

-e: encripta os dados de entrada. Esse é o padrão.

-d: descriptografa os dados de entrada.

-a: processa os dados em base64. Isso significa que se os dados estão codificados em base64.

-base64: sinônimo para "-a".

-A: se a opção "-a" estiver presente, então a base64 é processada em uma linha.

-k <senha>: a senha é derivada da chave. Esta opção é útil para versões antigas de OpenSSL.

-kfile <arquivo>: lê a primeira linha do arquivo fornecido em <arquivo> e o utiliza de senha para derivar a chave. Esta opção é útil para compatibilidade com versões antigas do OpenSSL.

-nosalt: não use salt.

-salt: use salt quando encriptando (este é o padrão).

-S <salt>: o salt a ser usado: precisa ser representado como uma string de digitos hexadecimais.

Exemplo de comando:

```
1 # Codificar arquivo em base 64
2 $ openssl enc -base64 -in meu_texto.txt -out minha_saida_base64.txt
3 $ cat minha_saida_base64.txt
4 SSBhbSBnb2luIHRvIGJlIGVuY29kZWQK
```

4) Comando passwd

O comando PASSWD da ferramenta OpenSSL tem como objetivo fornecer meios para encriptar ou descriptografar vários blocos ou cifradores stream baseado em senhas ou explicitamente fornecidos. Encriptação e descriptografia também podem ser feitas em Base64.

-help: printa uma mensagem de como utilizar.

-crypt: Usa o algoritmo de crypt.

-1: use o MD5 baseado no algoritmo de senhas BSD 1.

-apr1: usa o algoritmo apr1 (variante do algoritmo BSD da Apache.

-5 -6: usa o algoritmo SHA256/SHA512 definidos por Ulrich Drepper.

-salt <string>: usa o salt especificado. Quando está lendo uma senha do terminal, implica diretamente em -noverify.

-in <arquivo>: lê a senha do arquivo.

-stdin: lê senhas da entrada padrão.

-noverify: não verifica quando lendo uma senha do terminal.

-quiet: não gera avisos de saída quando as senhas usadas na linha de comando estão truncadas.

-table: na lista de saída, adiciona a senha em *cleartext* e uma caracter de TAB para cada hash de senha.

Exemplo de comando:

```
1 $ openssl passwd -1 -salt
2 Password:
3 $1$yoursalt$5WA5NN0quMJ62v5LCu8kj1
```

5) Comando rand

O comando RAND da ferramenta OpenSSL tem como objetivo fornecer números pseudo-aleatórios gerados a partir de um seed.

-help: printa na tela uma mensagem de como usar este comando.

-out <arquivo>: escreve a saída para um arquivo ao invés da saída padrão.

-rand <arquivo(s)>: usa arquivos especificados para usar de seed para o gerador de números pseudo-aleatório. Múltiplos arquivos podem ser especificados de acordo com o sistema operacional: ";" para Windows, "," para OpenVMS e ":" para todos os outros.

-base64: faz uma codificação de base64 da saída.

-hex: mostra a saída como uma string em hexadecimal.

Exemplo de comando:

```
1 # Gera 8 bits aleatorios e armazena no arquivo rand1.bin
2 $ openssl rand 8 -out rand1.bin
3 $ xxd rand1.bin
4 00000000: 52c4 c60a 9ab9 3279                    R.....2y
```

6) Comando speed

O comando SPEED da ferramenta OpenSSL tem como objetivo testar a performance de algoritmos criptográficos. Bom para realizar *benchmarks*.

-help: printa uma mensagem com instruções de uso para o comando.

-engine <id>: especificar um motor (de acordo com o seu ID único) fará com que o comando tente obter uma referência funcional para o motor especificado, portanto inicializando-o se for necessário.

-elapsed: mensura o tempo em tempo real ao invés de tempo da CPU. Pode ser útil para testar a velocidade em hardwares específicos.

-evp <algo>: usa o cifrador especificado ou o algoritmo de message digest pela interface EVP.

-decrypt: vai verificar o tempo de decriptação ao invés do tempo de encriptação. Affeta apenas o teste EVP.

[zero or more test algorithms]: se alguma opção é dada, então usa o teste de tempo para esses algoritmos. Do contrário, todos os algoritmos acima são utilizados.

Exemplo de comando:

```
# Faz um benchmark da velocidade dos algoritmos de encriptacao no meu computador.
$ openssl speed
Doing md4 for 3s on 16 size blocks: 13621592 md4's in 3.00s
Doing md4 for 3s on 64 size blocks: 10239804 md4's in 3.00s
Doing md4 for 3s on 256 size blocks: 5901982 md4's in 3.00s
Doing md4 for 3s on 1024 size blocks: 2186520 md4's in 3.00s
Doing md4 for 3s on 8192 size blocks: 319187 md4's in 3.00s
Doing md5 for 3s on 16 size blocks: 7905998 md5's in 3.00s
Doing md5 for 3s on 64 size blocks: 5532097 md5's in 3.00s
Doing md5 for 3s on 256 size blocks: 3151240 md5's in 3.00s
Doing md5 for 3s on 1024 size blocks: 1283545 md5's in 3.00s
Doing md5 for 3s on 8192 size blocks: 187701 md5's in 3.00s
Doing hmac(md5) for 3s on 16 size blocks: 8009845 hmac(md5)'s in 3.00s
Doing hmac(md5) for 3s on 64 size blocks: 6083095 hmac(md5)'s in 3.00s
Doing hmac(md5) for 3s on 256 size blocks: 3492758 hmac(md5)'s in 3.00s
Doing hmac(md5) for 3s on 1024 size blocks: 1281680 hmac(md5)'s in 3.00s
Doing hmac(md5) for 3s on 8192 size blocks: 186728 hmac(md5)'s in 2.99s
Doing sha1 for 3s on 16 size blocks: 11334892 sha1's in 3.00s
Doing sha1 for 3s on 64 size blocks: 7635323 sha1's in 3.00s
Doing sha1 for 3s on 256 size blocks: 4589719 sha1's in 3.00s
Doing sha1 for 3s on 1024 size blocks: 1699105 sha1's in 3.00s
Doing sha1 for 3s on 8192 size blocks: 245389 sha1's in 3.00s
Doing sha256 for 3s on 16 size blocks: 10155012 sha256's in 3.00s
Doing sha256 for 3s on 64 size blocks: 5905678 sha256's in 3.00s
Doing sha256 for 3s on 256 size blocks: 2952707 sha256's in 3.00s
Doing sha256 for 3s on 1024 size blocks: 869493 sha256's in 3.00s
Doing sha256 for 3s on 8192 size blocks: 109402 sha256's in 3.00s
Doing sha512 for 3s on 16 size blocks: 6731332 sha512's in 3.00s
Doing sha512 for 3s on 64 size blocks: 6870264 sha512's in 3.00s
Doing sha512 for 3s on 256 size blocks: 3305895 sha512's in 3.00s
Doing sha512 for 3s on 1024 size blocks: 1147902 sha512's in 3.00s
Doing sha512 for 3s on 8192 size blocks: 165633 sha512's in 3.00s
Doing whirlpool for 3s on 16 size blocks: 4335960 whirlpool's in 3.00s
Doing whirlpool for 3s on 64 size blocks: 2577480 whirlpool's in 3.00s
Doing whirlpool for 3s on 256 size blocks: 1069801 whirlpool's in 3.00s
Doing whirlpool for 3s on 1024 size blocks: 320088 whirlpool's in 3.00s
```

```
Doing whirlpool for 3s on 8192 size blocks: 41874 whirlpool's in 3.00s
Doing rmd160 for 3s on 16 size blocks: 6175293 rmd160's in 3.00s
Doing rmd160 for 3s on 64 size blocks: 3839459 rmd160's in 3.00s
Doing rmd160 for 3s on 256 size blocks: 1738074 rmd160's in 3.00s
Doing rmd160 for 3s on 1024 size blocks: 531218 rmd160's in 3.00s
Doing rmd160 for 3s on 8192 size blocks: 67192 rmd160's in 3.00s
Doing rc4 for 3s on 16 size blocks: 54556333 rc4's in 2.99s
Doing rc4 for 3s on 64 size blocks: 23391953 rc4's in 3.00s
Doing rc4 for 3s on 256 size blocks: 6939078 rc4's in 3.00s
Doing rc4 for 3s on 1024 size blocks: 1777970 rc4's in 3.00s
Doing rc4 for 3s on 8192 size blocks: 225977 rc4's in 3.00s
Doing des cbc for 3s on 16 size blocks: 10740581 des cbc's in 3.00s
Doing des cbc for 3s on 64 size blocks: 2758040 des cbc's in 3.00s
Doing des cbc for 3s on 256 size blocks: 680809 des cbc's in 3.00s
Doing des cbc for 3s on 1024 size blocks: 171407 des cbc's in 3.00s
Doing des cbc for 3s on 8192 size blocks: 21347 des cbc's in 3.00s
Doing des ede3 for 3s on 16 size blocks: 4067650 des ede3's in 3.00s
Doing des ede3 for 3s on 64 size blocks: 1021482 des ede3's in 3.00s
Doing des ede3 for 3s on 256 size blocks: 255168 des ede3's in 3.00s
Doing des ede3 for 3s on 1024 size blocks: 64499 des ede3's in 3.00s
Doing des ede3 for 3s on 8192 size blocks: 8071 des ede3's in 2.98s
Doing aes-128 cbc for 3s on 16 size blocks: 19556081 aes-128 cbc's in 3.00
s
Doing aes-128 cbc for 3s on 64 size blocks: 5268901 aes-128 cbc's in 2.98s
Doing aes-128 cbc for 3s on 256 size blocks: 1382784 aes-128 cbc's in 3.00
s
Doing aes-128 cbc for 3s on 1024 size blocks: 344760 aes-128 cbc's in 3.00
s
Doing aes-128 cbc for 3s on 8192 size blocks: 42878 aes-128 cbc's in 3.00s
Doing aes-192 cbc for 3s on 16 size blocks: 15276462 aes-192 cbc's in 3.00
s
Doing aes-192 cbc for 3s on 64 size blocks: 3987925 aes-192 cbc's in 3.00s
Doing aes-192 cbc for 3s on 256 size blocks: 1033564 aes-192 cbc's in 3.00
s
Doing aes-192 cbc for 3s on 1024 size blocks: 265124 aes-192 cbc's in 3.00
s
Doing aes-192 cbc for 3s on 8192 size blocks: 33892 aes-192 cbc's in 3.00s
Doing aes-256 cbc for 3s on 16 size blocks: 12975236 aes-256 cbc's in 3.00
s
Doing aes-256 cbc for 3s on 64 size blocks: 3177106 aes-256 cbc's in 2.99s
Doing aes-256 cbc for 3s on 256 size blocks: 892504 aes-256 cbc's in 3.00s
Doing aes-256 cbc for 3s on 1024 size blocks: 221622 aes-256 cbc's in 3.00
s
Doing aes-256 cbc for 3s on 8192 size blocks: 27547 aes-256 cbc's in 3.00s
Doing aes-128 ige for 3s on 16 size blocks: 18974989 aes-128 ige's in 2.99
s
Doing aes-128 ige for 3s on 64 size blocks: 4584223 aes-128 ige's in 3.00s
```

```
Doing aes-128 ige for 3s on 256 size blocks: 1165075 aes-128 ige's in 3.00s
Doing aes-128 ige for 3s on 1024 size blocks: 304980 aes-128 ige's in 2.98s
Doing aes-128 ige for 3s on 8192 size blocks: 35839 aes-128 ige's in 3.00s
Doing aes-192 ige for 3s on 16 size blocks: 16553145 aes-192 ige's in 3.00s
Doing aes-192 ige for 3s on 64 size blocks: 4080884 aes-192 ige's in 3.00s
Doing aes-192 ige for 3s on 256 size blocks: 1046068 aes-192 ige's in 2.99s
Doing aes-192 ige for 3s on 1024 size blocks: 222882 aes-192 ige's in 3.00s
Doing aes-192 ige for 3s on 8192 size blocks: 28026 aes-192 ige's in 3.00s
Doing aes-256 ige for 3s on 16 size blocks: 13182810 aes-256 ige's in 3.00s
Doing aes-256 ige for 3s on 64 size blocks: 3061727 aes-256 ige's in 3.00s
Doing aes-256 ige for 3s on 256 size blocks: 793723 aes-256 ige's in 2.98s
Doing aes-256 ige for 3s on 1024 size blocks: 209393 aes-256 ige's in 2.99s
Doing aes-256 ige for 3s on 8192 size blocks: 26010 aes-256 ige's in 3.00s
Doing ghash for 3s on 16 size blocks: 161509112 ghash's in 3.00s
Doing ghash for 3s on 64 size blocks: 132800014 ghash's in 3.00s
Doing ghash for 3s on 256 size blocks: 56504797 ghash's in 3.00s
Doing ghash for 3s on 1024 size blocks: 15203044 ghash's in 2.97s
Doing ghash for 3s on 8192 size blocks: 2244381 ghash's in 2.99s
Doing camellia-128 cbc for 3s on 16 size blocks: 13378774 camellia-128 cbc's in 3.00s
Doing camellia-128 cbc for 3s on 64 size blocks: 5224896 camellia-128 cbc's in 3.00s
Doing camellia-128 cbc for 3s on 256 size blocks: 1638421 camellia-128 cbc's in 3.00s
Doing camellia-128 cbc for 3s on 1024 size blocks: 417163 camellia-128 cbc's in 3.00s
Doing camellia-128 cbc for 3s on 8192 size blocks: 50035 camellia-128 cbc's in 3.00s
Doing camellia-192 cbc for 3s on 16 size blocks: 11808534 camellia-192 cbc's in 3.00s
Doing camellia-192 cbc for 3s on 64 size blocks: 4182872 camellia-192 cbc's in 3.00s
Doing camellia-192 cbc for 3s on 256 size blocks: 1166638 camellia-192 cbc's in 3.00s
Doing camellia-192 cbc for 3s on 1024 size blocks: 294774 camellia-192 cbc's in 3.00s
Doing camellia-192 cbc for 3s on 8192 size blocks: 36936 camellia-192 cbc's in 3.00s
Doing camellia-256 cbc for 3s on 16 size blocks: 12261399 camellia-256 cbc's in 3.00s
```

```
Doing camellia-256 cbc for 3s on 64 size blocks: 4120629 camellia-256 cbc's in 3.00s
Doing camellia-256 cbc for 3s on 256 size blocks: 1133325 camellia-256 cbc's in 3.00s
Doing camellia-256 cbc for 3s on 1024 size blocks: 308700 camellia-256 cbc's in 3.00s
Doing camellia-256 cbc for 3s on 8192 size blocks: 37282 camellia-256 cbc's in 3.00s
Doing seed cbc for 3s on 16 size blocks: 11320701 seed cbc's in 3.00s
Doing seed cbc for 3s on 64 size blocks: 2741992 seed cbc's in 3.00s
Doing seed cbc for 3s on 256 size blocks: 714561 seed cbc's in 3.00s
Doing seed cbc for 3s on 1024 size blocks: 184665 seed cbc's in 3.00s
Doing seed cbc for 3s on 8192 size blocks: 23129 seed cbc's in 3.00s
Doing rc2 cbc for 3s on 16 size blocks: 7110354 rc2 cbc's in 3.00s
Doing rc2 cbc for 3s on 64 size blocks: 1885029 rc2 cbc's in 3.00s
Doing rc2 cbc for 3s on 256 size blocks: 475165 rc2 cbc's in 3.00s
Doing rc2 cbc for 3s on 1024 size blocks: 114167 rc2 cbc's in 3.00s
Doing rc2 cbc for 3s on 8192 size blocks: 14968 rc2 cbc's in 3.00s
Doing blowfish cbc for 3s on 16 size blocks: 17303000 blowfish cbc's in 3.00s
Doing blowfish cbc for 3s on 64 size blocks: 4516158 blowfish cbc's in 3.00s
Doing blowfish cbc for 3s on 256 size blocks: 1141510 blowfish cbc's in 2.99s
Doing blowfish cbc for 3s on 1024 size blocks: 287649 blowfish cbc's in 3.00s
Doing blowfish cbc for 3s on 8192 size blocks: 35200 blowfish cbc's in 3.00s
Doing cast cbc for 3s on 16 size blocks: 15842021 cast cbc's in 2.99s
Doing cast cbc for 3s on 64 size blocks: 4300098 cast cbc's in 3.00s
Doing cast cbc for 3s on 256 size blocks: 1095230 cast cbc's in 3.00s
Doing cast cbc for 3s on 1024 size blocks: 277468 cast cbc's in 3.00s
Doing cast cbc for 3s on 8192 size blocks: 34482 cast cbc's in 3.00s
Doing 512 bit private rsa's for 10s: 158504 512 bit private RSA's in 10.00s
Doing 512 bit public rsa's for 10s: 2037684 512 bit public RSA's in 9.99s
Doing 1024 bit private rsa's for 10s: 56185 1024 bit private RSA's in 9.99s
Doing 1024 bit public rsa's for 10s: 852498 1024 bit public RSA's in 10.00s
Doing 2048 bit private rsa's for 10s: 11842 2048 bit private RSA's in 10.00s
Doing 2048 bit public rsa's for 10s: 274765 2048 bit public RSA's in 10.00s
Doing 4096 bit private rsa's for 10s: 1177 4096 bit private RSA's in 10.00s
Doing 4096 bit public rsa's for 10s: 70746 4096 bit public RSA's in 10.00s
```

```
Doing 512 bit sign dsa's for 10s: 133438 512 bit DSA signs in 9.99s
Doing 512 bit verify dsa's for 10s: 161504 512 bit DSA verify in 9.99s
Doing 1024 bit sign dsa's for 10s: 64123 1024 bit DSA signs in 10.00s
Doing 1024 bit verify dsa's for 10s: 68694 1024 bit DSA verify in 10.00s
Doing 2048 bit sign dsa's for 10s: 22561 2048 bit DSA signs in 10.00s
Doing 2048 bit verify dsa's for 10s: 19803 2048 bit DSA verify in 10.00s
Doing 160 bit sign ecdsa's for 10s: 130122 160 bit ECDSA signs in 10.00s
Doing 160 bit verify ecdsa's for 10s: 37711 160 bit ECDSA verify in 10.00s
Doing 192 bit sign ecdsa's for 10s: 116077 192 bit ECDSA signs in 10.00s
Doing 192 bit verify ecdsa's for 10s: 30791 192 bit ECDSA verify in 9.99s
Doing 224 bit sign ecdsa's for 10s: 121442 224 bit ECDSA signs in 10.00s
Doing 224 bit verify ecdsa's for 10s: 55749 224 bit ECDSA verify in 10.00s
Doing 256 bit sign ecdsa's for 10s: 180561 256 bit ECDSA signs in 10.00s
Doing 256 bit verify ecdsa's for 10s: 63484 256 bit ECDSA verify in 10.00s
Doing 384 bit sign ecdsa's for 10s: 37804 384 bit ECDSA signs in 10.00s
Doing 384 bit verify ecdsa's for 10s: 8663 384 bit ECDSA verify in 10.00s
Doing 521 bit sign ecdsa's for 10s: 20125 521 bit ECDSA signs in 10.00s
Doing 521 bit verify ecdsa's for 10s: 10814 521 bit ECDSA verify in 9.99s
Doing 163 bit sign ecdsa's for 10s: 42827 163 bit ECDSA signs in 9.96s
Doing 163 bit verify ecdsa's for 10s: 18169 163 bit ECDSA verify in 9.99s
Doing 233 bit sign ecdsa's for 10s: 22586 233 bit ECDSA signs in 9.99s
Doing 233 bit verify ecdsa's for 10s: 15181 233 bit ECDSA verify in 9.95s
Doing 283 bit sign ecdsa's for 10s: 15516 283 bit ECDSA signs in 9.97s
Doing 283 bit verify ecdsa's for 10s: 8420 283 bit ECDSA verify in 9.96s
Doing 409 bit sign ecdsa's for 10s: 6313 409 bit ECDSA signs in 10.00s
Doing 409 bit verify ecdsa's for 10s: 4646 409 bit ECDSA verify in 10.00s
Doing 571 bit sign ecdsa's for 10s: 2722 571 bit ECDSA signs in 10.00s
Doing 571 bit verify ecdsa's for 10s: 1966 571 bit ECDSA verify in 9.99s
Doing 163 bit sign ecdsa's for 10s: 40800 163 bit ECDSA signs in 9.98s
Doing 163 bit verify ecdsa's for 10s: 17618 163 bit ECDSA verify in 10.00s
Doing 233 bit sign ecdsa's for 10s: 23682 233 bit ECDSA signs in 10.00s
Doing 233 bit verify ecdsa's for 10s: 14473 233 bit ECDSA verify in 10.00s
Doing 283 bit sign ecdsa's for 10s: 15560 283 bit ECDSA signs in 10.00s
Doing 283 bit verify ecdsa's for 10s: 7915 283 bit ECDSA verify in 10.00s
Doing 409 bit sign ecdsa's for 10s: 6716 409 bit ECDSA signs in 9.95s
Doing 409 bit verify ecdsa's for 10s: 4906 409 bit ECDSA verify in 9.98s
Doing 571 bit sign ecdsa's for 10s: 2919 571 bit ECDSA signs in 9.99s
Doing 571 bit verify ecdsa's for 10s: 1829 571 bit ECDSA verify in 10.00s
Doing 160 bit  ecdh's for 10s: 47214 160-bit ECDH ops in 10.00s
Doing 192 bit  ecdh's for 10s: 39533 192-bit ECDH ops in 10.00s
Doing 224 bit  ecdh's for 10s: 82735 224-bit ECDH ops in 9.99s
Doing 256 bit  ecdh's for 10s: 97678 256-bit ECDH ops in 9.99s
Doing 384 bit  ecdh's for 10s: 12379 384-bit ECDH ops in 10.00s
................................
................................
```