

Generating Public and Private Keys

Generating RSA Keys	Generating DSA Keys:
Generate 2048 bit RSA Private Key saved as KEY1.pem <code>openssl genrsa -out KEY1.pem 2048</code>	Generate DSA Parameters File <code>openssl dsaparam -out DSA-PARAM.pem 1024</code>
Generate 4096 bit RSA Private Key, encrypted with AES128 <code>openssl genrsa -out KEY2.pem -aes128 4096</code>	Generate DSA Keys file with Parameters file <code>openssl gendsa -out DSA-KEY.pem DSA-PARAM.pem</code>
<ul style="list-style-type: none"> - Key size must be last argument of command - Omit <code>-out <FILE></code> argument to output to StdOut - Other encryption algorithms are also supported: <code>-aes128, -aes192, -aes256, -des3, -des</code> 	Generate DSA Parameters and Keys in one File <code>openssl dsaparam -genkey -out DSA-PARAM-KEY.pem 2048</code> See <i>Inspecting</i> section to view file contents.
Generating Elliptic Curve Keys:	
Generate EC Parameters file <code>openssl genpkey -genparam -algorithm EC -pkeyopt ec_paramgen_curve:secp384r1 -out EC-PARAM.pem</code>	
Generate EC Keys from Parameters file <code>openssl genpkey -paramfile EC-PARAM.pem -out EC-KEY.pem</code>	
Generate EC Keys directly <code>openssl genpkey -algorithm EC -pkeyopt ec_paramgen_curve:P-384 -out EC-KEY.pem</code>	
View supported Elliptic Curves <code>openssl ecparam -list_curves</code> Recommended Curves: prime256v1, secp384r1, secp521r1 (<i>identical to P-256, P-384, P-521</i>)	

Inspecting RSA, DSA, and Elliptic Curve Keys

Inspecting RSA Key Files	Inspecting any Key file using pkey utility
Converting an RSA Private Key into text <code>openssl rsa -in KEY.pem -noout -text</code>	Converting any Private Key file into text (RSA, DSA, or EC) <code>openssl pkey -in KEY.pem -noout -text</code>
Removing encryption from an RSA key file <code>openssl rsa -in ENCRYPTED-KEY.pem -out KEY.pem</code>	Extracting only Public Key as text from any Key file <code>openssl pkey -in KEY.pem -noout -text_pub</code>
Encrypting an RSA Key File <code>openssl rsa -in KEY.pem -aes128 -out ENCRYPTED-KEY.pem</code>	Extracting only Public Key in PEM format <code>openssl pkey -in KEY.pem -pubout</code> <i>pkey expects a Private Key file. Public Key file can be read with <code>-pubin</code></i>
Inspecting DSA Parameters and Keys	Check if RSA Key matches a CSR or Cert
Inspecting DSA Parameters file <code>openssl dsaparam -in DSA-PARAM.pem -text -noout</code>	Compare Modulus values to see if files match each other <code>openssl req -in CSR.pem -noout -modulus</code> <code>openssl x509 -in CERT.pem -noout -modulus</code> <code>openssl rsa -in KEY.pem -noout -modulus</code>
Inspecting DSA Private Key file <code>openssl dsa -in DSA-KEY.pem -text -noout</code>	
Inspecting EC Parameters and Keys	Check if EC Key matches a CSR or Cert
Inspecting Elliptic Curve (EC) Parameters file <code>openssl ecparam -in EC-PARAM.pem -text -noout</code>	Compare Public Key values to see if files match each other <code>openssl req -in EC-CSR.pem -noout -pubkey</code> <code>openssl x509 -in EC-CERT.pem -noout -pubkey</code> <code>openssl ec -in EC-KEY.pem -pubout</code>
Inspecting Elliptic Curve (EC) Private Key file <code>openssl ec -in EC-KEY.pem -text -noout</code>	

OpenSSL Cheat Sheet

Presented by
Practical Networking .net

Latest version of this cheat sheet and
training on how to use it are available here:
pracnet.net/openssl

Want to really understand SSL & TLS?
pracnet.net/tls

OpenSSL Cheat Sheet is provided for
free by Practical Networking .net

It is free to share with anyone
unmodified without restrictions.

License: CC BY-ND 4.0



Generating Certificate Signing Requests (CSRs) and Self-Signed Certificates

Generating CSRs:	Generating Self-Signed Certificates				
Generate CSR with <i>existing</i> Private Key file <code>openssl req -new -key KEY.pem -out CSR.pem</code>	Generate Certificate with <i>existing</i> Private Key file <code>openssl req -x509 -key KEY.pem -out CERT.pem</code>				
Generate CSR and <i>new</i> Private Key file <code>openssl req -new -newkey <alg:opt> -nodes -out CSR.pem</code>	Generate Certificate and <i>new</i> Private Key file <code>openssl req -x509 -newkey <alg:opt> -nodes -out CERT.pem</code>				
Notes / Options					
Commands above will prompt you for the Subject Distinguished Name (DN) attributes. Alternatively, you can specify them using <code>-subj</code> : Examples: <code>-subj "/CN=website.com"</code> <code>--or--</code> <code>-subj "/C=US/ST=Colorado/L=Denver/O=ACME Inc./CN=acme.com"</code>					
<code>-nodes</code> - Generate Key File with No DES encryption - Skips prompt for PEM Pass phrase <code>-<digest></code> - Sign CSR/Cert using <code><digest></code> hashing algorithm. View supported algorithms: <code>openssl list --digest-commands</code> <code>-config</code> - Specify config file with custom options. Default Config file: <code>openssl.cnf</code> in directory specified by <code>openssl version -d</code>					
The argument <code>-newkey <alg:opt></code> lets you create RSA , DSA , or EC Keys:					
<table> <tr> <td><code>-newkey 1024</code> - Generate 1024 bit RSA Keys (<i>legacy</i>)</td><td><code>-newkey dsa:DSA-PARAM.pem</code> - Generate DSA Keys using DSA Parameters</td></tr> <tr> <td><code>-newkey rsa:2048</code> - Generate 2048 bit RSA Keys</td><td><code>-newkey ec:EC-PARAM.pem</code> - Generate EC Keys using EC Parameters</td></tr> </table>		<code>-newkey 1024</code> - Generate 1024 bit RSA Keys (<i>legacy</i>)	<code>-newkey dsa:DSA-PARAM.pem</code> - Generate DSA Keys using DSA Parameters	<code>-newkey rsa:2048</code> - Generate 2048 bit RSA Keys	<code>-newkey ec:EC-PARAM.pem</code> - Generate EC Keys using EC Parameters
<code>-newkey 1024</code> - Generate 1024 bit RSA Keys (<i>legacy</i>)	<code>-newkey dsa:DSA-PARAM.pem</code> - Generate DSA Keys using DSA Parameters				
<code>-newkey rsa:2048</code> - Generate 2048 bit RSA Keys	<code>-newkey ec:EC-PARAM.pem</code> - Generate EC Keys using EC Parameters				
If <code>-key</code> or <code>-newkey</code> is not specified, a private key file will be automatically generated using directives specified in <code>openssl.cnf</code>					

Inspecting Certificate Signing Requests (CSRs) and Certificates

Viewing contents of Certs and CSRs	Extracting Specific Info from Certificates
Viewing x509 Certificate as human readable Text	Extract specific pieces of information from x509 Certificates
openssl x509 -in CERT.pem -noout -text	openssl x509 -in CERT.pem -noout -dates
Viewing Certificate Signing Request (CSR) contents as Text:	openssl x509 -in CERT.pem -noout -issuer -subject
openssl req -in CSR.pem -noout -text	Other items you can extract: -modulus -pubkey -ocsp_uri -ocspid -serial -startdate -enddate
Extracting x509 Certificate Extensions	
Extract specific Extension(s) from a certificate	
openssl x509 -in CERT.pem -noout -ext subjectAltName	
openssl x509 -in CERT.pem -noout -ext authorityInfoAccess,crlDistributionPoints	
Other extensions you can extract:	basicConstraints nameConstraints certificatePolicies keyUsage extendedKeyUsage subjectKeyIdentifier authorityKeyIdentifier
Extract all Extensions from a certificate	
openssl x509 -in CERT.pem -noout -text sed '/X509v3 extensions/,/Signature Algorithm:!/d'	

File Formats and Converting between formats (PEM, DER, PFX)

Check if file is PEM, DER, or PFX	PEM <==> DER
To check if file is PEM format <code>openssl x509 -in FILE</code>	Convert PEM Certificate file to DER <code>openssl x509 -in CERT.pem -outform DER -out CERT.der</code>
To check if file is DER format <code>openssl x509 -in FILE -inform DER</code>	Convert DER Certificate file to PEM <code>openssl x509 -in CERT.der -inform der -out CERT.pem</code>
To check if file is PFX format <code>openssl pkcs12 -in FILE -nodes</code>	PEM --> PFX
To check, or convert, PEM or DER Key Files use <code>openssl pkey</code> instead of <code>openssl x509</code> and same command arguments.	Convert PEM Certificate(s) to PFX <code>openssl pkcs12 -in CERTS.pem -nokeys -export -out CERTS.pfx</code> To include a key in PFX file use <code>-inkey KEY.pem</code> instead of <code>-nokeys</code>
PFX --> PEM	
To extract everything within a PFX file as a PEM file: <code>openssl pkcs12 -in FILE.pfx -out EVERYTHING.pem -nodes</code>	PFX files can contain Certificate(s), or Certificate(s) + one matching Key <code>-clcerts</code> - extract only end-entity certificate (client certificate) <code>-cacerts</code> - extract all but end-entity certificate <code>-nokeys</code> - extract only certificates
To extract only the Private Key from a PFX file as PEM: <code>openssl pkcs12 -in FILE.pfx -out KEY.pem -nodes -nocerts</code>	