How to create a self-signed certificate with OpenSSL

Asked 7 years, 3 months ago Active 18 days ago Viewed 1.3m times

1110

732

I'm adding HTTPS support to an embedded Linux device. I have tried to generate a self-signed certificate with these steps:

```
openssl req -new > cert.csr
openssl rsa -in privkey.pem -out key.pem
openssl x509 -in cert.csr -out cert.pem -req -signkey key.pem -days 1001
cat key.pem>>cert.pem
```

This works, but I get some errors with, for example, Google Chrome:

> This is probably not the site you are looking for!
> The site's security certificate is not trusted!

Am I missing something? Is this the correct way to build a self-signed certificate?

ssl openssl certificate ssl-certificate x509certificate

edited Dec 28 '18 at 17:44

Peter Mortensen
14.3k ● 19 ● 88 ● 116

asked Apr 16 '12 at 14:14

michelemarcon
7,859 ● 14 ● 45 ● 64

protected by mkobit Apr 12 '18 at 20:22

This question is protected to prevent "thanks!", "me too!", or spam answers by new users. To answer it, you must have earned at least 10 reputation on this site (the association bonus does not count).

34 Self-signed certificates are considered insecure for the Internet. Firefox will treat the site as having an invalid certificate, while Chrome will act as if the connection was plain HTTP. More details: gerv.net/security/self-signed-certs – user1202136 Apr 16 '12 at 14:17

27 You need to import your CA certificate into your browsers and tell the browsers you trust the certificate -or- get it signed by one of the big money-for-nothing organizations that are already trusted by the browsers -or- ignore the warning and click past it. I like the last option myself. – trojanfoe Apr 16 '12 at 14:20

10 You should not use the "stock" OpenSSL settings like that. That's because you cannot place DNS names in the Subject Alternate Name (SAN). You need to provide a configuration file with an `alternate_names` section and pass it with the `-config` option. Also, placing a DNS name in the Common Name (CN) is deprecated (but not prohibited) by both the IETF and the CA/Browser Forums. Any DNS name in the CN must also be present in the SAN. There's no way to avoid using the SAN. See answer below. – jww Jan 13 '15 at 22:01

4 In addition to @jww 's comment. Per may 2017 Chrome doesn't accept certs w/o (emtpy) SAN's anymore: "The certificate for this site does not contain a Subject Alternative Name extension

containing a domain name or IP address." – GerardJP May 25 '17 at 7:35

5 These days, as long as your webserver is accessible by its FQDN on port 80 over the internet, you can use LetsEncrypt and get free full CA certs (valid for 90 days, renewal can be automated) that won't give any browser warnings/messages. www.letsencrypt.com – barny Mar 27 '18 at 12:13

14 Answers

1839

You can do that in one command:

```
openssl req -x509 -newkey rsa:4096 -keyout key.pem -out cert.pem -days 365
```

You can also add `-nodes` (short for `no DES`) if you don't want to protect your private key with a passphrase. Otherwise it will prompt you for "at least a 4 character" password.

The `days` parameter (365) you can replace with any number to affect the expiration date. It will then prompt you for things like "Country Name", but you can just hit Enter and accept the defaults.

Add `-subj '/CN=localhost'` to suppress questions about the contents of the certificate (replace `localhost` with your desired domain).

Self-signed certificates are not validated with any third party unless you import them to the browsers previously. If you need more security, you should use a certificate signed by a certificate authority (CA).

edited Dec 28 '18 at 18:13 — Peter Mortensen — 14.3k ● 19 ● 88 ● 116

answered Apr 16 '12 at 15:04 — Diego Woitasen — 20.3k ● 1 ● 13 ● 18

5 For anyone who's interested, here is the documentation, if you want to verify anything yourself. – user456814 Apr 15 '14 at 17:34

11 How does signing with a 3rd-party provide more security? – James Mills Jul 11 '14 at 3:14

5 @Rob - placing a DNS name (like localhost) in the CN is deprecated by both the IETF and CA/B Forums. While its deprecated, its currently not prohibited (that's coming next for the CA/B). All names go in the Subject Alternate Name. If a certificate has a DNS name in the CN, then it must also be present in the SAN. Otherwise, the certificate will fail to validate under browsers and other user agents that follow the CA/B. There's no way to avoid using the SAN. – jww Jan 13 '15 at 21:15

156 For anyone else using this in automation, here's all of the common parameters for the subject: `-subj "/C=US/ST=Oregon/L=Portland/O=Company Name/OU=Org/CN=www.example.com"` – Alex S Jun 5 '15 at 18:13

58 Remember to use `-sha256` to generate SHA-256-based certificate. – Gea-Suan Lin Jan 25 '16 at 6:13

> Am I missing something? Is this the correct way to build a self-signed certificate?

It's easy to create a self-signed certificate. You just use the `openssl req` command. It can be tricky to create one that can be consumed by the largest selection of clients, like browsers and command line tools.

It's difficult because the browsers have their own set of requirements, and they are more restrictive than the IETF. The requirements used by browsers are documented at the CA/Browser Forums (see references below). The restrictions arise in two key areas: (1) trust anchors, and (2) DNS names.

Modern browsers (like the warez we're using in 2014/2015) want a certificate that chains back to a trust anchor, and they want DNS names to be presented in particular ways in the certificate. And browsers are actively moving against self-signed server certificates.

Some browsers don't exactly make it easy to import a self-signed server certificate. In fact, you can't with some browsers, like Android's browser. So the complete solution is to become your own authority.

In the absence of becoming your own authority, you have to get the DNS names right to give the certificate the greatest chance of success. But I would encourage you to become your own authority. It's easy to become your own authority, and it will sidestep all the trust issues (who better to trust than yourself?).

> This is probably not the site you are looking for!
> The site's security certificate is not trusted!

This is because browsers use a predefined list of trust anchors to validate server certificates. A self-signed certificate does not chain back to a trusted anchor.

The best way to avoid this is:

1. Create your own authority (i.e., become a CA)
2. Create a certificate signing request (CSR) for the server
3. Sign the server's CSR with your CA key
4. Install the server certificate on the server
5. Install the CA certificate on the client

Step 1 - *Create your own authority* just means to create a self-signed certificate with `CA: true` and proper key usage. That means the *Subject* and *Issuer* are the same entity, CA is set to true in *Basic Constraints* (it should also be marked as critical), key usage is `keyCertSign` and `crlSign` (if you are using CRLs), and the *Subject Key Identifier* (SKI) is the same as the *Authority Key Identifier* (AKI).

To become your own certificate authority, see *How do you sign a certificate signing request with your certification authority? on Stack Overflow. Then, import your CA into the Trust Store used by the browser.

Steps 2 - 4 are roughly what you do now for a public facing server when you enlist the services of a CA like Startcom or CAcert. Steps 1 and 5 allows you to avoid the third-party authority, and act

as your own authority (who better to trust than yourself?).

The next best way to avoid the browser warning is to trust the server's certificate. But some browsers, like Android's default browser, do not let you do it. So it will never work on the platform.

The issue of browsers (and other similar user agents) *not* trusting self-signed certificates is going to be a big problem in the Internet of Things (IoT). For example, what is going to happen when you connect to your thermostat or refrigerator to program it? The answer is, nothing good as far as the user experience is concerned.

The W3C's WebAppSec Working Group is starting to look at the issue. See, for example, Proposal: Marking HTTP As Non-Secure.

> How to create a self-signed certificate with OpenSSL

The commands below and the configuration file create a self-signed certificate (it also shows you how to create a signing request). They differ from other answers in one respect: the DNS names used for the self signed certificate are in the *Subject Alternate Name (SAN)*, and not the *Common Name (CN)*.

The DNS names are placed in the SAN through the configuration file with the line `subjectAltName = @alternate_names` (there's no way to do it through the command line). Then there's an `alternate_names` section in the configuration file (you should tune this to suit your taste):

```
[ alternate_names ]

DNS.1       = example.com
DNS.2       = www.example.com
DNS.3       = mail.example.com
DNS.4       = ftp.example.com

# Add these if you need them. But usually you don't want them or
#   need them in production. You may need them for development.
# DNS.5       = localhost
# DNS.6       = localhost.localdomain
# DNS.7       = 127.0.0.1

# IPv6 localhost
# DNS.8     = ::1
```

It's important to put DNS name in the SAN and not the CN, because *both* the IETF and the CA/Browser Forums specify the practice. They also specify that DNS names in the CN are deprecated (but not prohibited). *If* you put a DNS name in the CN, then it *must* be included in the SAN under the CA/B policies. So you can't avoid using the Subject Alternate Name.

If you don't do put DNS names in the SAN, then the certificate will fail to validate under a browser and other user agents which follow the CA/Browser Forum guidelines.

Related: browsers follow the CA/Browser Forum policies; and not the IETF policies. That's one of the reasons a certificate created with OpenSSL (which generally follows the IETF) sometimes does not validate under a browser (browsers follow the CA/B). They are different standards, they have different issuing policies and different validation requirements.

Create a self signed certificate (notice the addition of `-x509` option):

```
openssl req -config example-com.conf -new -x509 -sha256 -newkey rsa:2048 -nodes \
    -keyout example-com.key.pem -days 365 -out example-com.cert.pem
```

Create a signing request (notice the lack of `-x509` option):

```
openssl req -config example-com.conf -new -sha256 -newkey rsa:2048 -nodes \
    -keyout example-com.key.pem -days 365 -out example-com.req.pem
```

Print a self-signed certificate:

```
openssl x509 -in example-com.cert.pem -text -noout
```

Print a signing request:

```
openssl req -in example-com.req.pem -text -noout
```

Configuration file (passed via `-config` option)

```
[ req ]
default_bits        = 2048
default_keyfile     = server-key.pem
distinguished_name  = subject
req_extensions      = req_ext
x509_extensions     = x509_ext
string_mask         = utf8only

# The Subject DN can be formed using X501 or RFC 4514 (see RFC 4519 for a description).
#   Its sort of a mashup. For example, RFC 4514 does not provide emailAddress.
[ subject ]
countryName         = Country Name (2 letter code)
countryName_default     = US

stateOrProvinceName     = State or Province Name (full name)
stateOrProvinceName_default = NY

localityName            = Locality Name (eg, city)
localityName_default        = New York

organizationName         = Organization Name (eg, company)
organizationName_default    = Example, LLC

# Use a friendly name here because it's presented to the user. The server's DNS
#   names are placed in Subject Alternate Names. Plus, DNS names here is deprecated
#   by both IETF and CA/Browser Forums. If you place a DNS name here, then you
#   must include the DNS name in the SAN too (otherwise, Chrome and others that
#   strictly follow the CA/Browser Baseline Requirements will fail).
commonName          = Common Name (e.g. server FQDN or YOUR name)
commonName_default      = Example Company

emailAddress            = Email Address
emailAddress_default        = test@example.com

# Section x509_ext is used when generating a self-signed certificate. I.e., openssl req
-x509 ...
[ x509_ext ]

subjectKeyIdentifier        = hash
authorityKeyIdentifier    = keyid,issuer

# You only need digitalSignature below. *If* you don't allow
#   RSA Key transport (i.e., you use ephemeral cipher suites), then
#   omit keyEncipherment because that's key transport.
```

```
basicConstraints        = CA:FALSE
keyUsage            = digitalSignature, keyEncipherment
subjectAltName          = @alternate_names
nsComment           = "OpenSSL Generated Certificate"

# RFC 5280, Section 4.2.1.12 makes EKU optional
#   CA/Browser Baseline Requirements, Appendix (B)(3)(G) makes me confused
#   In either case, you probably only need serverAuth.
# extendedKeyUsage    = serverAuth, clientAuth

# Section req_ext is used when generating a certificate signing request. I.e., openssl
req ...
[ req_ext ]

subjectKeyIdentifier        = hash

basicConstraints        = CA:FALSE
keyUsage            = digitalSignature, keyEncipherment
subjectAltName          = @alternate_names
nsComment           = "OpenSSL Generated Certificate"

# RFC 5280, Section 4.2.1.12 makes EKU optional
#   CA/Browser Baseline Requirements, Appendix (B)(3)(G) makes me confused
#   In either case, you probably only need serverAuth.
# extendedKeyUsage    = serverAuth, clientAuth

[ alternate_names ]

DNS.1       = example.com
DNS.2       = www.example.com
DNS.3       = mail.example.com
DNS.4       = ftp.example.com

# Add these if you need them. But usually you don't want them or
#   need them in production. You may need them for development.
# DNS.5       = localhost
# DNS.6       = localhost.localdomain
# DNS.7       = 127.0.0.1

# IPv6 localhost
# DNS.8     = ::1
```

You may need to do the following for Chrome. Otherwise Chrome may complain a *Common Name* is invalid (`ERR_CERT_COMMON_NAME_INVALID`). I'm not sure what the relationship is between an IP address in the SAN and a CN in this instance.

```
# IPv4 localhost
# IP.1       = 127.0.0.1

# IPv6 localhost
# IP.2     = ::1
```

There are other rules concerning the handling of DNS names in X.509/PKIX certificates. Refer to these documents for the rules:

- RFC 5280, Internet X.509 Public Key Infrastructure Certificate and Certificate Revocation List (CRL) Profile
- RFC 6125, Representation and Verification of Domain-Based Application Service Identity within Internet Public Key Infrastructure Using X.509 (PKIX) Certificates in the Context of Transport Layer Security (TLS)
- RFC 6797, Appendix A, HTTP Strict Transport Security (HSTS)
- RFC 7469, Public Key Pinning Extension for HTTP

- CA/Browser Forum Baseline Requirements
- CA/Browser Forum Extended Validation Guidelines

RFC 6797 and RFC 7469 are listed, because they are more restrictive than the other RFCs and CA/B documents. RFCs 6797 and 7469 *do not* allow an IP address, either.

edited Dec 28 '18 at 18:43

Peter Mortensen
14.3k ● 19 ● 88 ● 116

answered Jan 13 '15 at 21:12

jww
56.8k ● 43 ● 253 ● 543

4 Is it possible to use wildcards in the `alternate_names` section? Particularly sub-sub domains. I have a question referencing this answer here: serverfault.com/questions/711596/… – LeonardChallis Aug 12 '15 at 10:02

2 I've just replied to his specific question. I think doesn't make sense to add this long security description when the answer was so simple – Diego Woitasen Feb 21 '16 at 1:44

12 @diegows - your answer is not complete or correct. The reason it is not correct is discussed in the long post you don't want to read :) – jww Feb 21 '16 at 4:42

1 Thanks! I found your post very helpful. FYI I was recently playing with vault and found it insisted on IP.x 127.0.0.1 rather than DNS.x 127... I didn't check if this is in the standard or not. – Chomeh Aug 22 '16 at 0:32

2 Thank you @jww. You said, *"1. Create your own authority (i.e, become a CA)"*, then said, *"5. Install the CA certificate on the client"*. If the root key became compromised, a malicious person could sign a cert for any domain with that key, and if they trick you into going to their website, they can now do a man-in-the-middle attack. Is there a way to create the root CA such that it can only sign intermediary CAs and not certificates? Then you can protect your intermediary CA with a name constraint. – Robin Zimmermann Mar 30 '17 at 18:41

|

388

Here are the options described in @diegows's answer, described in more detail, from the documentation:

```
openssl req -x509 -newkey rsa:2048 -keyout key.pem -out cert.pem -days XXX
```

```
req
```

PKCS#10 certificate request and certificate generating utility.

```
-x509
```

this option outputs a self signed certificate instead of a certificate request. This is typically used to generate a test certificate or a self signed root CA.

```
-newkey arg
```

this option creates a new certificate request and a new private key. The argument takes one of several forms. **rsa:nbits**, where **nbits** is the number of bits, generates an RSA key **nbits** in size.

```
-keyout filename
```

this gives the filename to write the newly created private key to.

```
-out filename
```

This specifies the output filename to write to or standard output by default.

```
-days n
```

when the **-x509** option is being used this specifies the number of days to certify the certificate for. The default is 30 days.

```
-nodes
```

if this option is specified then if a private key is created it will not be encrypted.

The documentation is actually more detailed than the above; I just summarized it here.

edited Dec 28 '18 at 18:14

Peter Mortensen
14.3k • 19 • 88 • 116

answered Apr 13 '14 at 1:48

user456814

What is XXX in your command? – Nicolas S.Xu Oct 15 '16 at 14:59

2 The `XXX` in the original command should be replaced with the 'number of days to certify the certificate for'. The default is 30 days. For example, `-days XXX` becomes `-days 365` if you want your cert to be valid for 365 days. See the docs for more. – Nathan Jones Oct 19 '16 at 21:11

Thanks for adding the documentation. This IBM link on creating a self-signed certificate using command which seems identical to this answer – The Red Pea Jun 1 '17 at 15:30

228

As of 2019, the following command serves all your needs, including SAN:

```
openssl req -x509 -newkey rsa:4096 -sha256 -days 3650 -nodes \
  -keyout example.key -out example.crt -extensions san -config \
  <(echo "[req]";
    echo distinguished_name=req;
    echo "[san]";
    echo subjectAltName=DNS:example.com,DNS:example.net,IP:10.0.0.1
    ) \
  -subj /CN=example.com
```

In OpenSSL ≥ 1.1.1, this can be shortened to:

```
openssl req -x509 -newkey rsa:4096 -sha256 -days 3650 -nodes \
  -keyout example.key -out example.crt -subj /CN=example.com \
  -addext subjectAltName=DNS:example.com,DNS:example.net,IP:10.0.0.1
```

It creates a certificate that is

- valid for the domains `example.com` and `example.net` (SAN),
- also valid for the IP address `10.0.0.1` (SAN),

- relatively strong (as of 2019) and
- valid for `3650` days (~10 years).

It creates the following files:

- Private key: `example.key`
- Certificate: `example.crt`

All information is provided at the command line. There is no annoying interactive input. There is no messing around with config files. All necessary steps are executed by this single OpenSSL invocation: from private key generation up to the self-signed certificate.

Remark #1: Crypto parameters

Since the certificate is self-signed and needs to be accepted by users manually, it doesn't make sense to use a short expiration or weak cryptography.

In the future, you might want to use more than `4096` bits for the RSA key and a hash algorithm stronger than `sha256`, but as of 2019 these are sane values. They are sufficiently strong while being supported by all modern browsers.

Remark #2: Parameter "`-nodes`"

Theoretically you could leave out the `-nodes` parameter (which means "no DES encryption"), in which case `example.key` would be encrypted with a password. However, this is almost never useful for a server installation, because you would either have to store the password on the server as well, or you'd have to enter it manually on each reboot.

Remark #3: MinGW

On Windows in the MinGW bash, you should prefix the command with `MSYS_NO_PATHCONV=1`:

```
MSYS_NO_PATHCONV=1 openssl ...
```

Alternatively, run the command in the plain `cmd.exe` Windows command prompt.

Remark #4: See also

- Provide subjectAltName to openssl directly on command line
- How to add multiple email adresses to an SSL certificate via the command line?
- More information about MSYS_NO_PATHCONV

edited Jul 14 at 15:05

answered Dec 28 '16 at 17:30

vog
11.9k ● 6 ● 40 ● 60

I tried to use the oneliner #2 (modern) on windows in mingw64, and I faced a bug with -subj parameter. ` $ openssl req -x509 -newkey rsa:4096 -sha256 -days 3650 -nodes -keyout localhost.key -out localhost.crt -subj '/CN=localhost' -addext subjectAltName=DNS:localhost,IP:127.0.0.1 Generating a RSA private key [...] writing new private key to 'localhost.key' ----- name is expected to be in the format /type0=value0/type1=value1/type2=... where characters may be escaped by \. This name is not in that format: 'C:/Program Files/Git/CN=localhost' problems making Certificate Request ` – Yuriy Pozniak Dec 23 '18 at 14:12

1 I couldn't figure out what exactly was to blame in the arg /CN=localhost expanding to C:/Program Files/Git/CN=localhost , so I just ran the whole command in plain cmd.exe and it worked just fine. Just in case someone is struggling with this one. – Yuriy Pozniak Dec 23 '18 at 14:15

If `-newkey rsa:4096` is omitted, OpenSSL defaults to 2048-bit RSA which sounds enough. Also I think `-sha256` is the default for 1.1.1 or above. – Franklin Yu Jan 11 at 23:15

1 @FranklinYu Are you sure that rsa:2048 will be enough in 10 years from now? Because that's the validity period. As explained, it doesn't make sense to use short expiration or weak crypto. Most 2048-bit RSA keys have a validity period of 1-3 years at most. Regarding OpenSSL 1.1.1, I'm still leaving sha256 in there, so it's more explicit and obvious to change if you want a stronger hash. – vog Jan 12 at 20:18

If you're using git bash on windows, like @YuriyPozniak, you will get the error he listed where `/CN=localhost` is being expanded to `C:/Progra Files/Git/CN=localhost`. If you add an extra /, then the expansion won't occur. `//CN=localhost` – Dave Ferguson Jun 25 at 5:56

128

I can't comment, so I will put this as a separate answer. I found a few issues with the accepted one-liner answer:

- The one-liner includes a passphrase in the key.
- The one-liner uses SHA-1 which in many browsers throws warnings in console.

Here is a simplified version that removes the passphrase, ups the security to suppress warnings and includes a suggestion in comments to pass in -subj to remove the full question list:

```
openssl genrsa -out server.key 2048
openssl rsa -in server.key -out server.key
openssl req -sha256 -new -key server.key -out server.csr -subj '/CN=localhost'
openssl x509 -req -sha256 -days 365 -in server.csr -signkey server.key -out server.crt
```

Replace 'localhost' with whatever domain you require. You will need to run the first two commands one by one as OpenSSL will prompt for a passphrase.

To combine the two into a .pem file:

```
cat server.crt server.key > cert.pem
```

edited Dec 28 '18 at 18:46

Peter Mortensen
14.3k ● 19 ● 88 ● 116

answered Aug 13 '15 at 9:44

Mike N
2,954 ● 2 ● 18 ● 17

6 I needed a dev certificate for github.com/molnarg/node-http2 and this answer is just the best. – Capaj Nov 8 '15 at 11:09

1 To combine the certificate and the key in a single file: `cat server.crt server.key >foo-cert.pem`. Works with the example in `openssl-1.0.2d/demos/ssl/` – 18446744073709551615 Nov 9 '15 at 12:33

The cert I generated this way is still using SHA1. – user169771 Jan 7 '16 at 14:58

1 Tks, works great to create a self signed certificate on `FreeBSD 10` `OpenLDAP 2.4` with `TLS` – Thiago Pereira Oct 3 '16 at 20:34

2 What about the key.pem file? – quikchange Nov 17 '16 at 10:24

72

Modern browsers now throw a security error for otherwise well-formed self-signed certificates if they are missing a SAN (Subject Alternate Name). ***OpenSSL does not provide a command-line way to specify this***, so many developers' tutorials and bookmarks are suddenly outdated.

The quickest way to get running again is a short, stand-alone conf file:

1. Create an OpenSSL config file (example: `req.cnf`)

```
[req]
distinguished_name = req_distinguished_name
x509_extensions = v3_req
prompt = no
[req_distinguished_name]
C = US
ST = VA
L = SomeCity
O = MyCompany
OU = MyDivision
CN = www.company.com
[v3_req]
keyUsage = critical, digitalSignature, keyAgreement
extendedKeyUsage = serverAuth
subjectAltName = @alt_names
[alt_names]
DNS.1 = www.company.com
DNS.2 = company.com
DNS.3 = company.net
```

2. Create the certificate referencing this config file

```
openssl req -x509 -nodes -days 730 -newkey rsa:2048 \
 -keyout cert.key -out cert.pem -config req.cnf -sha256
```

Example config from https://support.citrix.com/article/CTX135602

edited Dec 12 '17 at 18:11

vcsjones
109k ● 23 ● 255 ● 259

answered May 9 '17 at 2:37

rymo
2,558 ● 2 ● 25 ● 37

1 It worked for me after removing the last parameter -extensions 'v3_req' which was causing an error. Using OpenSSL for windows. Finally, I manage to fix this issue! Thanks. – CGodo May 10 '17 at 18:25

1 @Kyopaxa you're right - that parameter is redundant with line 3 of the cnf file; updated. – rymo May 10 '17 at 21:23

2 Solid way. Thanks. I'd suggest adding `-sha256`. – cherouvim Jul 27 '17 at 10:13

5 You can now specify the SAN on the command line with `-extension 'subjectAltName = DNS:dom.ain, DNS:oth.er'` see github.com/openssl/openssl/pull/4986 – Alexandre DuBreuil Jan 23 '18 at 11:02

2 Looks like this option is called `-addext` now. – Alexandr Zarubkin Nov 20 '18 at 14:10

I would recommend to add the **-sha256** parameter, to use the SHA-2 hash algorithm, because

65

major browsers are considering to show "SHA-1 certificates" as not secure.

The same command line from the accepted answer - @diegows with added -sha256

> openssl req -x509 **-sha256** -newkey rsa:2048 -keyout key.pem -out cert.pem -days XXX

More information in Google Security blog.

Update May 2018. As many noted in the comments that using SHA-2 does not add any security to a self-signed certificate. But I still recommend using it as a good habit of not using outdated / insecure cryptographic hash functions. Full explanation is available in *Why is it fine for certificates above the end-entity certificate to be SHA-1 based?*.

edited Dec 28 '18 at 18:16

Peter Mortensen
14.3k ● 19 ● 88 ● 116

answered Oct 20 '14 at 9:52

Maris B.
1,616 ● 3 ● 16 ● 27

1 If it's a self signed key, it's going to generate browser errors anyway, so this doesn't really matter – Mark Dec 16 '14 at 13:43

28 @Mark, it matters, because SHA-2 is more secure – Maris B. Dec 17 '14 at 15:38

1 Opening the certificate in windows after renaming the cert.pem to cert.cer says the fingerprint algorithm still is Sha1, but the signature hash algorithm is sha256. – sinned Dec 19 '14 at 8:33

2 "World-class encryption * zero authentication = zero security" gerv.net/security/self-signed-certs – x-yuri Mar 29 '18 at 9:03

3 Note that the signature algorithm used on a self-signed certificate is irrelevant in deciding whether it's trustworthy or not. Root CA certs are self-signed. and as of May 2018, there are still many active root CA certificates that are SHA-1 signed. Because it doesn't matter if a certificate trusts itself, nor how that certificate verifies that trust. You either trust the root/self-signed cert for *who* it says it is, or you don't. See security.stackexchange.com/questions/91913/… – Andrew Henle May 8 '18 at 18:55

|

18

This is the script I use on local boxes to set the SAN (subjectAltName) in self-signed certificates.

This script takes the domain name (example.com) and generates the SAN for *.example.com and example.com in the same certificate. The sections below are commented. Name the script (e.g. `generate-ssl.sh`) and give it executable permissions. The files will be written to the same directory as the script.

Chrome 58 an onward requires SAN to be set in self-signed certificates.

```
#!/usr/bin/env bash

# Set the TLD domain we want to use
BASE_DOMAIN="example.com"

# Days for the cert to live
DAYS=1095

# A blank passphrase
PASSPHRASE=""

# Generated configuration file
CONFIG_FILE="config.txt"
```

```
cat > $CONFIG_FILE <<-EOF
[req]
default_bits = 2048
prompt = no
default_md = sha256
x509_extensions = v3_req
distinguished_name = dn

[dn]
C = CA
ST = BC
L = Vancouver
O = Example Corp
OU = Testing Domain
emailAddress = webmaster@$BASE_DOMAIN
CN = $BASE_DOMAIN

[v3_req]
subjectAltName = @alt_names

[alt_names]
DNS.1 = *.$BASE_DOMAIN
DNS.2 = $BASE_DOMAIN
EOF

# The file name can be anything
FILE_NAME="$BASE_DOMAIN"

# Remove previous keys
echo "Removing existing certs like $FILE_NAME.*"
chmod 770 $FILE_NAME.*
rm $FILE_NAME.*

echo "Generating certs for $BASE_DOMAIN"

# Generate our Private Key, CSR and Certificate
# Use SHA-2 as SHA-1 is unsupported from Jan 1, 2017

openssl req -new -x509 -newkey rsa:2048 -sha256 -nodes -keyout "$FILE_NAME.key" -days
$DAYS -out "$FILE_NAME.crt" -passin pass:$PASSPHRASE -config "$CONFIG_FILE"

# OPTIONAL - write an info to see the details of the generated crt
openssl x509 -noout -fingerprint -text < "$FILE_NAME.crt" > "$FILE_NAME.info"

# Protect the key
chmod 400 "$FILE_NAME.key"
```

This script also writes an information file, so you can inspect the new certificate and verify the SAN is set properly.

```
                ...
                28:dd:b8:1e:34:b5:b1:44:1a:60:6d:e3:3c:5a:c4:
                da:3d
            Exponent: 65537 (0x10001)
    X509v3 extensions:
        X509v3 Subject Alternative Name:
            DNS:*.example.com, DNS:example.com
Signature Algorithm: sha256WithRSAEncryption
     3b:35:5a:d6:9e:92:4f:fc:f4:f4:87:78:cd:c7:8d:cd:8c:cc:
     ...
```

If you are using Apache, then you can reference the above certificate in your configuration file like so:

```
<VirtualHost _default_:443>
    ServerName example.com
    ServerAlias www.example.com
    DocumentRoot /var/www/htdocs

    SSLEngine on
    SSLCertificateFile path/to/your/example.com.crt
    SSLCertificateKeyFile path/to/your/example.com.key
</VirtualHost>
```

Remember to restart your Apache (or Nginx, or IIS) server for the new certificate to take effect.

edited Dec 28 '18 at 18:59

Peter Mortensen
14.3k ● 19 ● 88 ● 116

answered May 13 '17 at 20:21

Drakes
17.9k ● 3 ● 36 ● 69

Works on macOS High Siera and Chrome 58 – Saqib Omer Dec 23 '17 at 11:58

I'm still not sure how the CN affects the overall setup? I'm attempting to run this as `localhost` or `127.0.0.1:port#` what would be the corresponding `CN` for something like this. – DJ2 Apr 16 '18 at 16:56

@DJ2 I would set BASE_DOMAIN="localhost" – Drakes Apr 16 '18 at 23:11

9

2017 one-liner:

```
openssl req \
-newkey rsa:2048 \
-x509 \
-nodes \
-keyout server.pem \
-new \
-out server.pem \
-subj /CN=localhost \
-reqexts SAN \
-extensions SAN \
-config <(cat /System/Library/OpenSSL/openssl.cnf \
    <(printf '[SAN]\nsubjectAltName=DNS:localhost')) \
-sha256 \
-days 3650
```

This also works in Chrome 57, as it provides the SAN, without having another configuration file. It was taken from an answer here.

This creates a single .pem file that contains both the private key and cert. You can move them to separate .pem files if needed.

edited Dec 28 '18 at 19:02

Peter Mortensen
14.3k ● 19 ● 88 ● 116

answered Sep 20 '17 at 16:27

joemillervi
479 ● 4 ● 16

2 For Linux users you'll need to change that path for the config. e.g. on current Ubuntu `/etc/ssl/openssl.conf` works – declension Sep 27 '18 at 10:53

For a one-liner that doesn't require you to specify the openssl.cnf location, see: stackoverflow.com/a/41366949/19163 – vog Nov 26 '18 at 9:56

6

One liner FTW. I like to keep it simple. Why not use one command that contains ALL the arguments needed? This is how I like it - this creates an x509 certificate and its PEM key:

```
openssl req -x509 \
 -nodes -days 365 -newkey rsa:4096 \
 -keyout self.key.pem \
 -out self-x509.crt \
 -subj "/C=US/ST=WA/L=Seattle/CN=example.com/emailAddress=someEmail@gmail.com"
```

That single command contains all the answers you would normally provide for the certificate details. This way you can set the parameters and run the command, get your output - then go for coffee.

>> More here <<

edited Dec 28 '18 at 19:01

Peter Mortensen
14.3k ● 19 ● 88 ● 116

answered Sep 11 '17 at 15:14

OkezieE
3,830 ● 2 ● 20 ● 23

1 All the arguments except for SANs... @vog's answer covers that as well (and predate this) (This has a more complete "Subject" field filled in though...) (Not a big fan of the one year expiry either) – Gert van den Berg Dec 18 '18 at 10:38

6

You have the general procedure correct. The syntax for the command is below.

```
openssl req -new -key {private key file} -out {output file}
```

However, the warnings are displayed, because the browser was not able to verify the identify by validating the certificate with a known Certificate Authority (CA).

As this is a self-signed certificate there is no CA and you can safely ignore the warning and proceed. Should you want to get a real certificate that will be recognizable by anyone on the public Internet then the procedure is below.

1. Generate a private key
2. Use that private key to create a CSR file
3. Submit CSR to CA (Verisign or others, etc.)
4. Install received cert from CA on web server
5. Add other certs to authentication chain depending on the type cert

I have more details about this in a post at *Securing the Connection: Creating a Security Certificate with OpenSSL*

edited Dec 28 '18 at 18:45

Peter Mortensen
14.3k ● 19 ● 88 ● 116

answered Apr 2 '15 at 6:15

nneko
583 ● 5 ● 10

One-liner version 2017:

4

CentOS:

```
openssl req -x509 -nodes -sha256 -newkey rsa:2048 \
-keyout localhost.key -out localhost.crt \
-days 3650 \
-subj "CN=localhost" \
-reqexts SAN -extensions SAN \
-config <(cat /etc/pki/tls/openssl.cnf <(printf
"\n[SAN]\nsubjectAltName=IP:127.0.0.1,DNS:localhost"))
```

Ubuntu:

```
openssl req -x509 -nodes -sha256 -newkey rsa:2048 \
-keyout localhost.key -out localhost.crt \
-days 3650 \
-subj "CN=localhost" \
-reqexts SAN -extensions SAN \
-config <(cat /etc/ssl/openssl.cnf <(printf
"\n[SAN]\nsubjectAltName=IP:127.0.0.1,DNS:localhost"))
```

edited Dec 28 '18 at 19:03

Peter Mortensen

14.3k ● 19 ● 88 ● 116

answered Nov 28 '17 at 10:11

user327843

312 ● 1 ● 14

3

Generate keys

I am using `/etc/mysql` for cert storage because `/etc/apparmor.d/usr.sbin.mysqld` contains `/etc/mysql/*.pem r`.

```
sudo su -
cd /etc/mysql
openssl genrsa -out ca-key.pem 2048;
openssl req -new -x509 -nodes -days 1000 -key ca-key.pem -out ca-cert.pem;
openssl req -newkey rsa:2048 -days 1000 -nodes -keyout server-key.pem -out server-
req.pem;
openssl x509 -req -in server-req.pem -days 1000 -CA ca-cert.pem -CAkey ca-key.pem -
set_serial 01 -out server-cert.pem;
openssl req -newkey rsa:2048 -days 1000 -nodes -keyout client-key.pem -out client-
req.pem;
openssl x509 -req -in client-req.pem -days 1000 -CA ca-cert.pem -CAkey ca-key.pem -
set_serial 01 -out client-cert.pem;
```

Add configuration

`/etc/mysql/my.cnf`

```
[client]
ssl-ca=/etc/mysql/ca-cert.pem
ssl-cert=/etc/mysql/client-cert.pem
ssl-key=/etc/mysql/client-key.pem

[mysqld]
ssl-ca=/etc/mysql/ca-cert.pem
ssl-cert=/etc/mysql/server-cert.pem
ssl-key=/etc/mysql/server-key.pem
```

On my setup, Ubuntu server logged to: `/var/log/mysql/error.log`

Follow up notes:

- `SSL error: Unable to get certificate from '...'`

  MySQL might be denied read access to your certificate file if it is not in apparmors configuration. As mentioned in the previous steps^, save all our certificates as `.pem` files in the `/etc/mysql/` directory which is approved by default by apparmor (or modify your apparmor/SELinux to allow access to wherever you stored them.)

- `SSL error: Unable to get private key`

  Your MySQL server version may not support the default `rsa:2048` format

  Convert generated `rsa:2048` to plain `rsa` with:

  ```
  openssl rsa -in server-key.pem -out server-key.pem
  openssl rsa -in client-key.pem -out client-key.pem
  ```

- Check if local server supports SSL:

  ```
  mysql -u root -p
  mysql> show variables like "%ssl%";
  +---------------+----------------------------+
  | Variable_name | Value                      |
  +---------------+----------------------------+
  | have_openssl  | YES                        |
  | have_ssl      | YES                        |
  | ssl_ca        | /etc/mysql/ca-cert.pem     |
  | ssl_capath    |                            |
  | ssl_cert      | /etc/mysql/server-cert.pem |
  | ssl_cipher    |                            |
  | ssl_key       | /etc/mysql/server-key.pem  |
  +---------------+----------------------------+
  ```

- Verifying a connection to the database is SSL encrypted:

  ### Verifying connection

  When logged in to the MySQL instance, you can issue the query:

  ```
  show status like 'Ssl_cipher';
  ```

  If your connection is not encrypted, the result will be blank:

  ```
  mysql> show status like 'Ssl_cipher';
  +---------------+-------+
  | Variable_name | Value |
  +---------------+-------+
  | Ssl_cipher    |       |
  +---------------+-------+
  1 row in set (0.00 sec)
  ```

  Otherwise, it would show a non-zero length string for the cypher in use:

  ```
  mysql> show status like 'Ssl_cipher';
  +---------------+--------------------+
  ```

```
| Variable_name | Value              |
+---------------+--------------------+
| Ssl_cipher    | DHE-RSA-AES256-SHA |
+---------------+--------------------+
1 row in set (0.00 sec)
```

- Require ssl for specific user's connection ('require ssl'):

  - SSL

  Tells the server to permit only SSL-encrypted connections for the account.

  ```
  GRANT ALL PRIVILEGES ON test.* TO 'root'@'localhost'
    REQUIRE SSL;
  ```

  To connect, the client must specify the --ssl-ca option to authenticate the server certificate, and may additionally specify the --ssl-key and --ssl-cert options. If neither --ssl-ca option nor --ssl-capath option is specified, the client does not authenticate the server certificate.

Alternate link: Lengthy tutorial in *Secure PHP Connections to MySQL with SSL*.

edited Dec 28 '18 at 18:51

community wiki
3 revs, 3 users 80%
ThorSummoner

3

As has been discussed in detail, self-signed certificates are not trusted for the Internet. You can add your self-signed certificate to many but not all browsers. Alternatively you can become your own certificate authority.

The primary reason one does not want to get a signed certificate from a certificate authority is cost -- Symantec charges between $995 - $1,999 per year for certificates -- just for a certificate intended for internal network, Symantec charges $399 per year. That cost is easy to justify if you are processing credit card payments or work for the profit center of a highly profitable company. It is more than many can afford for a personal project one is creating on the internet, or for a non-profit running on a minimal budget, or if one works in a cost center of an organization -- cost centers always try to do more with less.

An alternative is to use certbot (see about certbot). Certbot is an easy-to-use automatic client that fetches and deploys SSL/TLS certificates for your web server.

If you setup certbot, you can enable it to create and maintain a certificate for you issued by the Let's Encrypt certificate authority.

I did this over the weekend for my organization. I installed the required packages for certbot on my server (Ubuntu 16.04) and then ran the command necessary to setup and enable certbot. One likely needs a DNS plugin for certbot - we are presently using DigitalOcean though may be migrating to another service soon.

Note that some of the instructions were not quite right and took a little poking and time with Google to figure out. This took a fair amount of my time the first time but now I think I could do it in minutes.

For DigitalOcean, one area I struggled was when I was prompted to input the path to your DigitalOcean credentials INI file. What the script is referring to is the Applications & API page and the Tokens/Key tab on that page. You need to have or generate a personal access token (read and write) for DigitalOcean's API -- this is a 65 character hexadecimal string. This string then needs to be put into a file on the webserver from which you are running certbot. That file can have a comment as its first line (comments start with #). The seccond line is:

```
dns_digitalocean_token =
0000111122223333444455556666777788889999aaaabbbbccccddddeeeeffff
```

Once I figured out how to set up a read+write token for DigitalOcean's API, it was pretty easy to use certbot to setup a wildcard certificate. Note that one does not have to setup a wildcard certificate, one may instead specify each domain and sub-domain that one wants the certificate to appply to. It was the wildcard certificate that required the credentials INI file that contained the personal access token from DigitalOcean.

Note that public key certificates (also known as identity certificates or SSL certificates) expire and require renewal. Thus you will need to renew your certificate on a periodic (reoccurring) basis. The certbot documentation covers renewing certificates.

My plan is to write a script to use the openssl command to get my certificate's expiration date and to trigger renewal when it is 30 days or less until it expires. I will then add this script to cron and run it once per day.

Here is the command to read your certificate's expiration date:

```
root@prod-host:~# /usr/bin/openssl x509 -enddate -noout -in path-to-certificate-pem-file
notAfter=May 25 19:24:12 2019 GMT
```

answered Feb 25 at 21:52

Peter Jirak Eldritch

80 • 6