

How to generate a self-signed SSL certificate using OpenSSL? [closed]

Asked 11 years, 11 months ago Modified 3 months ago Viewed 3.0m times



1957



Closed. This question is [not about programming or software development](#). It is not currently accepting answers.



This question does not appear to be about [a specific programming problem, a software algorithm, or software tools primarily used by programmers](#). If you believe the question would be on-topic on [another Stack Exchange site](#), you can leave a comment to explain where the question may be able to be answered.

Closed 1 year ago.

The community reviewed whether to reopen this question 1 year ago and left it closed:

Original close reason(s) were not resolved

[Improve this question](#)

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](#)

Share Improve this question

Follow

edited May 25, 2021 at 6:57



Jesse Nickles




1,639 1 20 28

asked Apr 16, 2012 at 14:14



micheleamarcon

23.8k 17 53 69

-
- 63 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, 2012 at 14:17
-
- 64 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, 2012 at 14:20
-
- 21 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, 2015 at 22:01 
-
- 6 In addition to [@jww](#)'s comment. Per may 2017 Chrome doesn't accept certs w/o (empty) 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, 2017 at 7:35 
-
- 11 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 – [DisappointedByUnaccountableMod](#) Mar 27, 2018 at 12:13 
-
- 6 The Let's Encrypt site is not `.com` but `.org` – [Yu Jiaao](#) Oct 18, 2019 at 0:12
-

23 Answers

Sorted by:

Highest score (default)





You can do that in one command:

3133



```
# interactive
openssl req -x509 -newkey rsa:4096 -keyout key.pem -out cert.pem -sha256 -days
365

# non-interactive and 10 years expiration
openssl req -x509 -newkey rsa:4096 -keyout key.pem -out cert.pem -sha256 -days
3650 -nodes -subj "/C=XX/ST=StateName/L=CityName/O=CompanyName
/OU=CompanySectionName/CN=CommonNameOrHostname"
```



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 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).

Share Improve this answer

Follow

edited Nov 22, 2023 at 9:20



StackOfZtuff

2,729 1 29 28

answered Apr 16, 2012 at 15:04



Diego Woitasen

33.8k 1 18 20

28 For anyone who's interested, here is [the documentation](#), if you want to verify anything yourself. – user456814 Apr 15, 2014 at 17:34

32 How does signing with a 3rd-party provide more security? – James Mills Jul 11, 2014 at 3:14

326 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, 2015 at 18:13

74 @JamesMills I mean, think about it -- if a shady looking guy with "free candy" written on the side of his van invites you to come inside, you're totally going to think twice and be on guard about it -- but if someone you trust -- like *really* trust -- is all like, "naw man, he's legit" you're going to be all about that free candy. – BrainSlugs83 Dec 18, 2015 at 23:39

89 Remember to use `-sha256` to generate SHA-256-based certificate. – Gea-Suan Lin Jan 25, 2016 at 6:13



748



As of 2023 with OpenSSL $\geq 1.1.1$, the following command serves all your needs, including [Subject Alternate Name \(SAN\)](#):

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

If you prefer ECC over RSA, you can specify different crypto parameters:

```
openssl req -x509 -newkey ec -pkeyopt ec_paramgen_curve:secp384r1 -days 3650 \
  -nodes -keyout example.com.key -out example.com.crt -subj "/CN=example.com" \
  -addext "subjectAltName=DNS:example.com,DNS:*.example.com,IP:10.0.0.1"
```

On old systems with OpenSSL $\leq 1.1.0$, such as Debian ≤ 9 or CentOS ≤ 7 , a longer version of this command needs to be used:

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

Each of the above commands creates a certificate that is

- valid for the domain `example.com` (SAN),
- also valid for the wildcard domain `*.example.com` (SAN),
- also valid for the IP address `10.0.0.1` (SAN),
- relatively strong (as of 2023) and
- valid for `3650` days (~10 years).

The following files are generated:

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

All information is provided at the command line. There is **no interactive input** that annoys you. There are **no config files** you have to mess around with. All necessary steps are executed by a **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

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 2023 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: See also

- [Provide subjectAltName to openssl directly on command line](#)
- [How to add multiple email addresses to an SSL certificate via the command line?](#)
- [More information about MSYS_NO_PATHCONV](#)

Share Improve this answer

edited Jun 22, 2023 at 8:58

answered Dec 28, 2016 at 17:30

Follow



vog

24.3k 11 60 78

-
- 5 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 - [yozniak](#) Dec 23, 2018 at 14:12
-
- 5 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. - [yozniak](#) Dec 23, 2018 at 14:15
-
- 2 @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, 2019 at 20:18
-
- 7 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, 2019 at 5:56
-
- 2 As of Aug-31/2020 I can vouch this works!!! Thanks a lot! I really would like to see a reference that explains in simple terms why this is evolving at such pace. Part of me wonders if it's just because the idea of creating self signed certs is counter productive to the big tech cos. What is going to be needed in 10 or 20 years time? It's madness, and it's a testament of that the amount of activity this kind of questions on openssl generates. - [Alex. S.](#) Sep 1, 2020 at 2:51



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

662



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
# IP.1       = 127.0.0.1
# IP.2       = ::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

deprecated (but not prohibited). // 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

```

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.

Share Improve this answer

Follow

edited Oct 7, 2021 at 7:34



Community Bot

1 1

answered Jan 13, 2015 at 21:12



jww

99.8k

94

422

912



454



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.

Share Improve this answer

Follow

edited Dec 28, 2018 at 18:14




Peter Mortensen

31k 22 108 132

answered Apr 13, 2014 at 1:48

user456814

- 3 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, 2016 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](#) – [Nate Anderson](#) Jun 1, 2017 at 15:30



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

184



- 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
```

Share Improve this answer

Follow

edited Dec 28, 2018 at 18:46



[Peter Mortensen](#)



31k 22 108 132

answered Aug 13, 2015 at 9:44



[Mike N](#)

6,555 4 24 21

- 7 I needed a dev certificate for github.com/molnarg/node-http2 and this answer is just the best.
– Capaj Nov 8, 2015 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, 2015 at 12:33
- 1 Tks, works great to create a self signed certificate on FreeBSD 10 OpenLDAP 2.4 with TLS
– Thiago Pereira Oct 3, 2016 at 20:34
- 5 What about the key.pem file? – quikchange Nov 17, 2016 at 10:24
- 1 You don't need to go through all those steps. You can put everything together in a single, simple command. See: stackoverflow.com/a/41366949/19163 – vog Jan 12, 2018 at 9:23 



90



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>

Share Improve this answer

Follow

edited Dec 12, 2017 at 18:11



vcsjones


140k 33 294 287

answered May 9, 2017 at 2:37



rymo

3,374 2 36 41

-
- 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, 2017 at 18:25 
-
- 1 @Kyopaxa you're right - that parameter is redundant with line 3 of the cnf file; updated. – [rymo](#) May 10, 2017 at 21:23
-
- 2 Solid way. Thanks. I'd suggest adding `-sha256`. – [cherouvim](#) Jul 27, 2017 at 10:13
-
- 6 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, 2018 at 11:02
-
- 3 Looks like this option is called `-addext` now. – [Alexandr Zarubkin](#) Nov 20, 2018 at 14:10
-



76



I would recommend to add the **-sha256** parameter, to use the SHA-2 hash algorithm, because 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](https://google.com/blog/2018/02/26/ssl-sha256/).

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?](https://paulkiss.net/2018/05/01/why-is-it-fine-for-certificates-above-the-end-entity-certificate-to-be-sha-1-based/).

Share Improve this answer

Follow

edited Dec 28, 2018 at 18:16



[Peter Mortensen](#)

31k 22 108 132

answered Oct 20, 2014 at 9:52



[Maris B.](#)

2,382 4 22 36

-
- 2 If it's a self signed key, it's going to generate browser errors anyway, so this doesn't really matter – [Mark](#) Dec 16, 2014 at 13:43
-
- 33 @Mark, it matters, because SHA-2 is more secure – [Maris B.](#) Dec 17, 2014 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, 2014 at 8:33
-
- 2 "World-class encryption * zero authentication = zero security" gerv.net/security/self-signed-certs – [x-yuri](#) Mar 29, 2018 at 9:03
-
- 4 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/...](http://security.stackexchange.com/questions/91913/) – [Andrew Henle](#) May 8, 2018 at 18:55
-



29

I can't comment so I add a separate answer. I tried to create a self-signed certificate for NGINX and it was easy, but when I wanted to add it to Chrome white list I had a problem. And my solution was to create a Root certificate and signed a child certificate by it.



So step by step. Create file **config_ssl_ca.cnf** Notice, config file has an option **basicConstraints=CA:true** which means that this certificate is supposed to be root.



This is a good practice, because you create it once and can reuse.

```
[ req ]
default_bits = 2048

prompt = no
distinguished_name=req_distinguished_name
req_extensions = v3_req

[ req_distinguished_name ]
countryName=UA
stateOrProvinceName=root region
localityName=root city
organizationName=Market(localhost)
organizationalUnitName=roote department
commonName=market.localhost
emailAddress=root_email@root.localhost

[ alternate_names ]
DNS.1      = market.localhost
DNS.2      = www.market.localhost
DNS.3      = mail.market.localhost
DNS.4      = ftp.market.localhost
DNS.5      = *.market.localhost

[ v3_req ]
keyUsage=digitalSignature
basicConstraints=CA:true
subjectKeyIdentifier = hash
subjectAltName = @alternate_names
```

Next config file for your child certificate will be call **config_ssl.cnf**.

```
[ req ]
default_bits = 2048

prompt = no
distinguished_name=req_distinguished_name
req_extensions = v3_req

[ req_distinguished_name ]
countryName=UA
stateOrProvinceName=Kyiv region
localityName=Kyiv
organizationName=market place
organizationalUnitName=market place department
commonName=market.localhost
emailAddress=email@market.localhost

[ alternate_names ]
.  -  ..
```

```
DNS.1      = market.localhost
DNS.2      = www.market.localhost
DNS.3      = mail.market.localhost
DNS.4      = ftp.market.localhost
DNS.5      = *.market.localhost
```

```
[ v3_req ]
keyUsage=digitalSignature
basicConstraints=CA:false
subjectAltName = @alternate_names
subjectKeyIdentifier = hash
```

The first step - create Root key and certificate

```
openssl genrsa -out ca.key 2048
openssl req -new -x509 -key ca.key -out ca.crt -days 365 -config
config_ssl_ca.cnf
```

The second step creates child key and file CSR - Certificate Signing Request. Because the idea is to sign the child certificate by root and get a correct certificate

```
openssl genrsa -out market.key 2048
openssl req -new -sha256 -key market.key -config config_ssl.cnf -out market.csr
```

Open Linux terminal and do this command

```
echo 00 > ca.srl
touch index.txt
```

The **ca.srl** text file containing the next serial number to use in hex. Mandatory. This file must be present and contain a valid serial number.

Last Step, create one more config file and call it **config_ca.cnf**

```
# we use 'ca' as the default section because we're usign the ca command
[ ca ]
default_ca = my_ca

[ my_ca ]
# a text file containing the next serial number to use in hex. Mandatory.
# This file must be present and contain a valid serial number.
serial = ./ca.srl

# the text database file to use. Mandatory. This file must be present though
# initially it will be empty.
database = ./index.txt

# specifies the directory where new certificates will be placed. Mandatory.
new_certs_dir = ./

# the file containing the CA certificate. Mandatory
certificate = ./ca.crt

# the file contaning the CA private key. Mandatory
private_key = ./ca.key
```



```
# the message digest algorithm. Remember to not use MD5
default_md = sha256

# for how many days will the signed certificate be valid
default_days = 365

# a section with a set of variables corresponding to DN fields
policy = my_policy

# MOST IMPORTANT PART OF THIS CONFIG
copy_extensions = copy

[ my_policy ]
# if the value is "match" then the field value must match the same field in the
# CA certificate. If the value is "supplied" then it must be present.
# Optional means it may be present. Any fields not mentioned are silently
```

You may ask, why so difficult, why we must create one more config to sign child certificate by root. The answer is simple because child certificate must have a SAN block - Subject Alternative Names. If we sign the child certificate by "openssl x509" utils, the Root certificate will delete the SAN field in child certificate. So we use "openssl ca" instead of "openssl x509" to avoid the deleting of the SAN field. We create a new config file and tell it to copy all extended fields **copy_extensions = copy**.

```
openssl ca -config config_ca.cnf -out market.crt -in market.csr
```

The program asks you 2 questions:

1. Sign the certificate? Say "Y"
2. 1 out of 1 certificate requests certified, commit? Say "Y"

In terminal you can see a sentence with the word "Database", it means file index.txt which you create by the command "touch". It will contain all information by all certificates you create by "openssl ca" util. To check the certificate valid use:

```
openssl rsa -in market.key -check
```

If you want to see what inside in CRT:

```
openssl x509 -in market.crt -text -noout
```

If you want to see what inside in CSR:

```
openssl req -in market.csr -noout -text
```

Share Improve this answer

Follow

edited May 19, 2021 at 18:25



Doug

3,593 3 21 18

answered Jan 21, 2020 at 7:23

mrkiri mrkiri

user 941 13 11

5 Although, this process looks complicated, this is exactly what we need for .dev domain, as this



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

24



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 and 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
```

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.

Share Improve this answer

Follow

edited Dec 28, 2018 at 18:59



Peter Mortensen

31k 22 108 132

answered May 13, 2017 at 20:21



Drakes

23.4k 3 53 94

Works on macOS High Sierra and Chrome 58 – [Saqib Omer](#) Dec 23, 2017 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, 2018 at 16:56

@DJ2 I would set `BASE_DOMAIN="localhost"` – [Drakes](#) Apr 16, 2018 at 23:11



2017 one-liner:

11



```
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.

Share Improve this answer

Follow

edited Dec 28, 2018 at 19:02



Peter Mortensen

31k 22 108 132

answered Sep 20, 2017 at 16:27



joemillervi

1,037 1 8 18

-
- 2 For Linux users you'll need to change that path for the config. e.g. on current Ubuntu /etc/ssl/openssl.cnf works – [declension](#) Sep 27, 2018 at 10:53
-
- 1 For a one-liner that doesn't require you to specify the openssl.cnf location, see: stackoverflow.com/a/41366949/19163 – [vog](#) Nov 26, 2018 at 9:56
-



Generate a key without password and certificate for 10 years, the short way:

10

```
openssl req -x509 -nodes -new -keyout server.key -out server.crt -days 3650  
-subj "/C=/ST=/L=/O=/OU=web/CN=www.server.com"
```



for the flag `-subj` | `-subject` **empty values are permitted** `-subj "/C=/ST=/L=/O=/OU=web/CN=www.server.com"` , but you can sets more details as you like:



- C - Country Name (2 letter code)
- ST - State
- L - Locality Name (eg, city)
- O - Organization Name
- OU - Organizational Unit Name
- CN - Common Name - **required!**

Share Improve this answer

edited May 29, 2022 at 13:58

answered Dec 27, 2021 at 13:35

Follow



Maoz Zadok

5,340 3 37 46



9

openssl allows to generate self-signed certificate by a single command (-newkey instructs to generate a private key and -x509 instructs to issue a self-signed certificate instead of a signing request)::



```
openssl req -x509 -newkey rsa:4096 \
-keyout my.key -passout pass:123456 -out my.crt \
-days 365 \
-subj /CN=localhost/O=home/C=US/emailAddress=me@mail.internal \
-addext "subjectAltName = DNS:localhost,DNS:web.internal,email:me@mail.internal" \
-addext keyUsage=digitalSignature -addext extendedKeyUsage=serverAuth
```

You can generate a private key and construct a self-signing certificate in separate steps::

```
openssl genrsa -out my.key -passout pass:123456 2048

openssl req -x509 \
-key my.key -passin pass:123456 -out my.csr \
-days 3650 \
-subj /CN=localhost/O=home/C=US/emailAddress=me@mail.internal \
-addext "subjectAltName = DNS:localhost,DNS:web.internal,email:me@mail.internal" \
-addext keyUsage=digitalSignature -addext extendedKeyUsage=serverAuth
```

Review the resulting certificate::

```
openssl x509 -text -noout -in my.crt
```

Java keytool creates PKCS#12 store::

```
keytool -genkeypair -keystore my.p12 -alias master \
-storetype pkcs12 -keyalg RSA -keysize 2048 -validity 3650 \
-storepass 123456 \
-dname "CN=localhost,O=home,C=US" \
-ext 'san=dns:localhost,dns:web.internal,email:me@mail.internal'
```

To export the self-signed certificate::

```
keytool -exportcert -keystore my.p12 -file my.crt \
-alias master -rfc -storepass 123456
```

Review the resulting certificate::

```
keytool -printcert -file my.crt
```

certtool from GnuTLS doesn't allow passing different attributes from CLI. I don't like to mess with config files ((



gavenkoa

47k

23

256

321

- 1 I cannot stress that enough!!!!!! extendedKeyUsage = serverAuth, clientAuth is what got me the button "Proceed to localhost (unsafe)" – JohnnyJS Feb 28, 2022 at 10:48



One-liner version 2017:

8

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"))
```

Edit: added prepending Slash to 'subj' option for Ubuntu.

Share Improve this answer

Follow

edited Oct 7, 2019 at 16:16



Community Bot

1 1

answered Nov 28, 2017 at 10:11



user327843

497

6

17



7



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 apparmor's 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 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](#).

Share Improve this answer

edited Dec 28, 2018 at 18:51

community wiki

Follow

3 revs, 3 users 80%

ThorSummoner

-
- 1 -1; this is largely tangential to the question asked, and also does a bad job of making clear where its quotes are from. – [Mark Amery](#) Sep 29, 2019 at 17:12

This shows provisioning CA, Server/Client certs signed by the CA, configure them for reading by mysqld on a host with apparmor. It exemplifies a rather useless case of hosting the ca, server, and client on the same machine, and dangerously exposing that ca's authority to the mysqld process. This setup doesn't really make sense other than to test ssl configuration in a test environment. For operating an internal CA, I would recommend the gnutls toolchain over openssl help.ubuntu.com/community/GnuTLS and having a good understanding of tls before working around the mysqld+apparmor case – [ThorSummoner](#) Sep 29, 2019 at 20:57



7



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 <<](#)

Share Improve this answer

Follow

edited Dec 28, 2018 at 19:01



Peter Mortensen

31k 22 108 132

answered Sep 11, 2017 at 15:14



Okezie

5,072 3 26 27

-
- 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, 2018 at 10:38

[vog's answer](#). Linked, because usernames are neither unique nor immutable. "vog" could change to "scoogie" at any point in time. – [jpaugh](#) Jan 17, 2022 at 3:45



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

6

```
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](#)

Share Improve this answer

Follow

edited Dec 28, 2018 at 18:45



Peter Mortensen

31k 22 108 132

answered Apr 2, 2015 at 6:15



nneko

800 9 11



Quick command line: Minimal Version

6

"I want a self signed certificate, in pfx form, for www.example.com with minimal fuss":



```
openssl req -x509 -sha256 -days 365 -nodes -out cert.crt -keyout cert.key -subj  
"/CN=www.example.com"
```

```
openssl pkcs12 -export -out cert.pfx -inkey cert.key -in cert.crt
```



Share Improve this answer

Follow

edited Nov 23, 2022 at 10:31



Marin Bînzari

5,278 2 27 44

answered Sep 8, 2022 at 13:06



Richard

108k 21 207 267

-subj should be -subj – [milahu](#) Nov 10, 2022 at 20:06



5



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 second line is:

```
dns_digitalocean_token =  
0000111122223333444455556666777788889999aaaabbbbccccdddeeeeffff
```

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 apply 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
```

Share Improve this answer Follow

answered Feb 25, 2019 at 21:52



[Peter Jirak Eldritch](#)

773 3 8 15



After much of going around, playing with various solutions, still I faced the problem that issuing a self-signed certificate for localhost, gave me error

4

ERR_CERT_INVALID



When expanding the details, chrome said:



You cannot visit localhost right now because the website sent scrambled credentials...

And the only ugly way to get through is to type (directly in this screen, without seeing any cursor for the text) :

(type in the keyboard) ***thisisunsafe***

Which let me proceed.

Until I found `extendedKeyUsage = serverAuth, clientAuth`

TL;DR

1. `openssl genrsa -out localhost.key 2048`
2. `openssl req -key localhost.key -new -out localhost.csr`
3. (click enter on everything and just fill in the common name (CN) with localhost or your other FQDN.
4. put the following in a file named `v3.ext` (edit whatever you need):

```
subjectKeyIdentifier    = hash
authorityKeyIdentifier  = keyid:always,issuer:always
basicConstraints        = CA:TRUE
keyUsage                 = digitalSignature, nonRepudiation, keyEncipherment,
dataEncipherment, keyAgreement, keyCertSign
extendedKeyUsage         = serverAuth, clientAuth
subjectAltName           = DNS:localhost, DNS:localhost.localdomain
issuerAltName            = issuer:copy
```

5. `openssl x509 -req -in localhost.csr -signkey localhost.key -out localhost.pem`
`-days 3650 -sha256 -extfile v3.ext`

And voilà! You can visit the website, expand "Advanced" and click "Proceed to localhost (unsafe)".

Share Improve this answer

edited May 26, 2022 at 10:52

answered Feb 28, 2022 at 11:15

Follow



JohnnyJS

1,422 12 24

Maybe some smart fellow would be able to make all of this a nice one-liner... – [JohnnyJS](#) Feb 28, 2022 at 11:18

what is the `v3.ext` file in your last command? – [Daniel Klimuntowski](#) May 26, 2022 at 7:34

1 Edited the answer. look at point 4. – [JohnnyJS](#) May 26, 2022 at 10:52



this worked for me

2

```
openssl req -x509 -nodes -subj '/CN=localhost' -newkey rsa:4096 -keyout
./sslcert/key.pem -out ./sslcert/cert.pem -days 365
```



server.js



```
var fs = require('fs');
var path = require('path');
var http = require('http');
var https = require('https');
var compression = require('compression');
var express = require('express');
var app = express();

app.use(compression());
app.use(express.static(__dirname + '/www'));

app.get('/*', function(req,res) {
  res.sendFile(path.join(__dirname+'www/index.html'));
});

// your express configuration here

var httpServer = http.createServer(app);
var credentials = {
  key: fs.readFileSync('./sslcert/key.pem', 'utf8'),
  cert: fs.readFileSync('./sslcert/cert.pem', 'utf8')
};
var httpsServer = https.createServer(credentials, app);

httpServer.listen(8080);
httpsServer.listen(8443);

console.log(`RUNNING ON http://127.0.0.1:8080`);
console.log(`RUNNING ON http://127.0.0.1:8443`);
```

Share Improve this answer Follow

answered Dec 5, 2020 at 6:16



[Leonardo Pineda](#)

1,020 9 10



This very simple Python app that creates a self-signed certificate. Code:

-1



```
from OpenSSL import crypto, SSL
from secrets import randbelow
print("Please know, if you make a mistake, you must restart the program.")
def cert_gen(
    emailAddress=input("Enter Email Address: "),
    commonName=input("Enter Common Name: "),
    countryName=input("Enter Country Name (2 characters): "),
    localityName=input("Enter Locality Name: "),
    stateOrProvinceName=input("Enter State of Province Name: "),
    organizationName=input("Enter Organization Name: "),
    organizationUnitName=input("Enter Organization Unit Name: "),
    serialNumber=randbelow(1000000),
    validityStartInSeconds=0,
    validityEndInSeconds=10*365*24*60*60,
    KEY_FILE = "private.key",
    CERT_FILE="selfsigned.crt"):
    #can look at generated file using openssl:
    #openssl x509 -inform pem -in selfsigned.crt -noout -text
    # create a key pair
    k = crypto.PKey()
    k.generate_key(crypto.TYPE_RSA, 4096)
    # create a self-signed cert
    cert = crypto.X509()
    cert.get_subject().C = countryName
    cert.get_subject().ST = stateOrProvinceName
    cert.get_subject().L = localityName
    cert.get_subject().O = organizationName
    cert.get_subject().OU = organizationUnitName
    cert.get_subject().CN = commonName
    cert.get_subject().emailAddress = emailAddress
    cert.set_serial_number(serialNumber)
    cert.gmtime_adj_notBefore(0)
    cert.gmtime_adj_notAfter(validityEndInSeconds)
    cert.set_issuer(cert.get_subject())
    cert.set_pubkey(k)
    cert.sign(k, 'sha512')
    with open(CERT_FILE, "wt") as f:
        f.write(crypto.dump_certificate(crypto.FILETYPE_PEM,
```

However, you still get the "certificate is not trusted" error. This is because of a few reasons:

1. It is self-signed/not verified (a verified certificate would need a CA (Certificate Authority), like Let's Encrypt to be trusted on all devices).
2. It is not trusted on your machine. ([this](#) answer shows how you can make Windows trust your certificate).

Share Improve this answer Follow

answered May 23, 2022 at 4:56



Fighter178

383 2 12



If you want to generate self signed certificates using open ssl - here is a script we have generated which can be used as is.

-1



```
#!/bin/bash
```

```
subj='//SKIP=skip/C=IN/ST=Country/L=City/O=MyCompany/OU=Technology'
```

```
red='\033[31m'          # red
```

```
yellow='\033[33m'        # yellow
```

```
green='\033[32m'          # green
```

```
blue='\033[34m'           # Blue
```

```
purple='\033[35m'         # Purple
```

```
cyan='\033[36m'           # Cyan
```

```
white='\033[37m'          # White
```

```
gencerts(){
```

```
certname=$1
```

```
pkname=$2
```

```
alias=$3
```

```
$(openssl genrsa -out $pkname'pem.pem' 4096)
```

```
$(openssl req -new -sha256 -key $pkname'pem.pem' -out $certname'csr.csr' -subj  
$subj)
```

```
$(openssl x509 -req -sha256 -days 3650 -in $certname'csr.csr' -signkey  
$pkname'pem.pem' -out $certname'.crt')
```

```
$(openssl pkcs12 -export -out $pkname'.p12' -name $alias -inkey $pkname'pem.pem'  
-in $certname'.crt')
```

```
}
```

```
verify(){
```

```
pkname=$1
```

```
keytool -v -list -storetype pkcs12 -keystore $pkname'.p12'
```

```
}
```

```
echo -e "${purple}WELCOME TO KEY PAIR GENERATOR"
```

```
echo -e "${yellow} Please enter the name of the certificate required: "
```

```
read certname
```

```
echo -e "${green}Please enter the name of the Private Key p12 file required: "
```

```
read pkname
```

```
echo -e "${cyan}Please enter the ALIAS of the Private Key p12 file : "
```

```
read pkalias
```

```
echo -e "${white}Please wait while we generate your Key Pair"
```

- Do let me know if any improvements can be made to the script.

Share Improve this answer Follow

answered Oct 28, 2022 at 17:17



Akshay Patel

77 1



You don't need to use `openssl`'s bad user interface at all! Try [mkcert](#).

-4

```
$ brew install mkcert nss  
[...]
```



```
$ mkcert -install
```

Created a new local CA 🌟

The local CA is now installed in the system trust store! ⚡

The local CA is now installed in the Firefox trust store (requires browser restart)! 🐱



```
$ mkcert example.com "*.example.com" example.test localhost 127.0.0.1 ::1
```

Created a new certificate valid for the following names 📄

- "example.com"
- "*.example.com"
- "example.test"
- "localhost"
- "127.0.0.1"
- "::1"

The certificate is at `./example.com+5.pem` and the key at `./example.com+5-key.pem` ✅

Share Improve this answer

edited Sep 28, 2022 at 11:01

answered Aug 22, 2022 at 17:40

Follow



iono

2,643

1

29

38

- 1 why the downvote?? this is the best easy solution instead of 1000line to create a local ssl cert which at the end chrome does not accept even when we put it in windows cert store! mkcert works perfectly and does everything except adding `192.168.1.100 mysite.com` to `C:\Windows\System32\drivers\etc\HOSTS` – [Badr Elmers](#) Jun 30, 2023 at 14:23
- 1 Also it's worth pointing out that `mkcert` has over 40,000 stars on GitHub - this is a very popular tool! – [iono](#) Jul 2, 2023 at 6:38
- 1 Yes, I was surprised by the amount of GitHub stars. It is clear that he solves a real problem, otherwise he would not have got all these stars – [Badr Elmers](#) Jul 4, 2023 at 9:57



Highly active question. Earn 10 reputation (not counting the [association bonus](#)) in order to answer this question. The reputation requirement helps protect this question from spam and non-answer activity.