



# How to use OpenSSL and the Internet PKI on Linux systems

A high-level overview of TLS/SSL and the OpenSSL tool, creating private keys and CSRs, and an introduction to the Internet PKI.

Posted: April 1, 2021 || [Jörg Kastning](#) (Sudoer)



Photo by [RODNAE Productions](#) from [Pexels](#)

This article is part two of three covering encryption concepts and the Internet public key infrastructure (PKI). The first article in this series [introduced symmetric and public key \(asymmetric\) encryption in cryptography](#). If you're not familiar with the basic concept of public-key encryption, you should read part one before you go ahead with this one.

In this part, I show you the basics of Transport Layer Security and Secure Socket Layer (TLS/SSL), how the Internet PKI works, and OpenSSL, the Swiss Army knife for TLS/SSL tasks. I cover how to use OpenSSL to create key-pairs and to generate a certificate signing request (CSR) to send to your certificate authority (CA) for signing. After that, I discuss some weaknesses of the Internet PKI you should be aware of.

**[ You might also enjoy: [Security advice for sysadmins: Own IT, Secure IT, Protect IT](#) ]**

## Preface

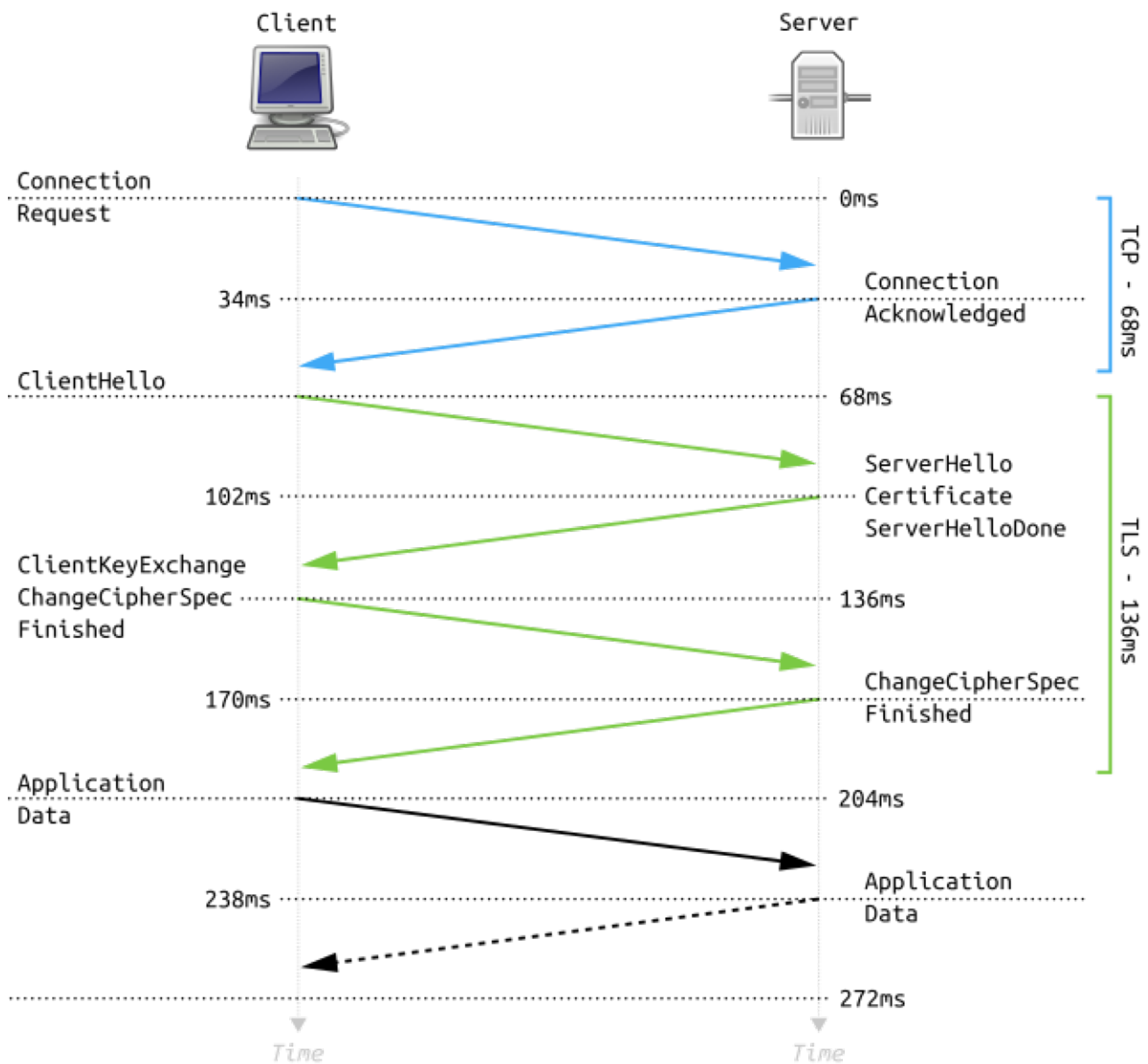
Assume that you're a sysadmin like me and one of your tasks is to manage a webserver. Because your users care about authenticity, integrity, and privacy, you'd like to secure your web application with some kind of encryption.\* You don't know in advance who's using your site, so symmetric encryption is off the table because of its key distribution problem. I use public-key encryption in the following sections instead.

## Linux security

- [What is security automation?](#)
- [Red Hat OpenShift Service on AWS security FAQ](#)
- [Enhance security with automation](#)
- [Implementing DevSecOps guide](#)
- [Red Hat CVE checker](#)

## TLS/SSL

The acronyms for *Transport Layer Security* and *Secure Socket Layer* are TLS and SSL. They are used interchangeably most of the time, and that's OK. While the old SSL protocol versions are deprecated, you'll usually find TLSv1.2 and TLSv1.3 on the web these days. TLS is used in HTTPS connections between some clients and some web servers. The following image shows a simple example of an HTTPS handshake.



Full TLS 1.2 Handshake with timing data (Source: [Full\\_TLS\\_1.2\\_Handshake.svg](#))

First, the well-known TCP handshake happens between client and server. Then the client starts the HTTPS handshake by sending

the **ClientHello**. In this step, the client transmits information about the server name it requests and the supported cipher suites. The server responds with the **ServerHello**, transmits a selected cipher suite, connection parameters, and sends information for calculating a symmetric key for the ongoing connection. Last but not least, it sends its certificate to authenticate itself to the client.

If you would like to get a thorough understanding of TLS and SSL, I recommend the book [Bulletproof SSL and TLS by Ivan Ristic](#).

Focus on the certificate the server has transmitted to the client. It contains the server's public key, which the client uses to encrypt data before sending it to the server. A trusted Certificate Authority (CA) signed the public key in the certificate. Today, every operating system and web browser comes with a store containing the public keys of many different trusted CAs. These public keys are then used to verify the signatures in server certificates like the one discussed here. This way, the client can check the server's authenticity and that it is the correct host the client wants to connect to.

As you can see, public-key encryption is used in this scenario for two tasks:

1. Verify the authenticity of the server
2. Use the server's public key to transmit encrypted data to the server

Be aware that public key encryption is used only to establish the HTTPS connections and calculate a symmetric session key used for further communication. That's because symmetric encryption is much faster than asymmetric.

## OpenSSL

So now the question is, how do you get the key-pair for the webserver? As stated earlier, [OpenSSL](#) is the Swiss Army knife for SSL and TLS tasks. Since its documentation has left some room for improvement, I suggest that you read the free book, [OpenSSL Cookbook by Ivan Ristic](#) to get all the details. My article focuses on creating a key-pair and a Certificate Signing Request (CSR) for a single domain name and a CSR that includes multiple domain names.

The CSR is needed to send the public key and other information about your domain to a CA for signing. The signed public key you'll get back is your certificate which you will install on your web server along with the corresponding private key.

### Creating a private key and a CSR for a single domain

Next, I'll give you an example of creating a private key (RSA 2048 bits) and CSR in one single step. You will enter all information necessary.

```
$ openssl req -newkey rsa:2048 -keyout example.com.key -out example.com.csr
Generating a RSA private key
.....+++++
.....
writing new private key to 'example.com.key'
Enter PEM pass phrase:
Verifying - Enter PEM pass phrase:
-----
You are about to be asked to enter information that will be incorporated
into your certificate request.
What you are about to enter is what is called a Distinguished Name or a DN.
There are quite a few fields but you can leave some blank
For some fields there will be a default value,
If you enter '.', the field will be left blank.
-----
Country Name (2 letter code) [AU]:DE
State or Province Name (full name) [Some-State]:EXAMPLE-STATE
Locality Name (eg, city) []:Example City
Organization Name (eg, company) [Internet Widgits Pty Ltd]:Example Ltd
Organizational Unit Name (eg, section) []:IT
Common Name (e.g. server FQDN or YOUR name) []:example.com
Email Address []:webmaster@example.com

Please enter the following 'extra' attributes
to be sent with your certificate request
A challenge password []:
An optional company name []:
```

You'll find both files in your current working directory. The passphrase you entered during the creation process protects your private key file. If you open them in some kind of text viewer, you'll only recognize that these are a private key and a CSR but won't find further plain text information:



```
$ cat example.com.{key,csr}
-----BEGIN ENCRYPTED PRIVATE KEY-----
MIIFHDBOBgkqhkiG9w0BBQ0wQTApBgkqhkiG9w0BBQwwHAQIaoQRP0LFgu4CAggA
MAwGCCqGSIB3DQIJBQAwFAYIKoZIhvcNAwcECFWvU11tJyJNBIIIEyCFv43Q7xe+F
Am5EtIJ02FtwbjXV9hQpl4Zb4JlG7vQvt7aH1/hQcPFYnjJJnlu8gP84tne1riSc
2YWxsZQ3sZLti0rVIqE0sCumJpoqS8AsAw/fuFIwRpXZenX2/snGwQSVN545BDGV
lzGnUph8lnrG9Kho5loiQYTAQWYUCwgLwe8fW0dXXWY6+UdKjJHaUYxHdkuMBvnQ
FJ+Iln4b4onLrGcFKJgNkJ/AShKW/m31ZvRXNX2QiKgl6u3zDTsE3l6rfVaPEJyx
g43guPil9nNrGS5rPt6XWEvtWSTfSFEVrMTBStr1BrUipfnN3gFfCmtNqoSNdkjk
pI9vsRekjzGZnVp763WGXmxx59/sXwgLnA7Br5K+7B1ECA9Z8iRzfb40mSrR8HVb
uYGRlq5jq/GgBTddFQRYUna0bufGRRBntIj0esYGbVWrg4dTnF8jncqvM+eXJtCv
Va0RrpTtuZbdRf3/5scjlFpvWQs705tbi777AIw9MySVpsgFY8eD8shKYNpQoRx5
kiILlkbabNK1EP0dnCBSCfHoK2ssZFKEgc0xG8HLM4iwVS/fg3002HS2jP74wPJ9
1sN81IvJ+W10J8MizkLJDc4bAER/yo907hMJ1ITxlUXWcr0ZwxilcQzmLLocFh46
uKQ8N9iAEnl1m2Ax98hbxttrHVVjGBBbrcIrrp/KfMAUcM6goUjz2PG9+XI5/6bkA
0g4qSNVotdPaYTnco0eRClzxv0llE8CBeVECql7+KN3VA0VM2Rtpb98QQZ+asKkY
fYf8byQLiCPED4NPkLeYKKL9CrMz+p7Zt3j/q5n0CLEbf/ZX8S6677m6+Hie9ArE
PU1fwpxonH9b4zvZnrSA9fFKW7zIWxq/tt885EjqzxvVUQKcCn2A3lXoXC3FoLaA
jmWkjue9vZAdn29C0E4+Ruk4oGfr/FipI4lZKH+sT1u86Gotjg2W3Zie8I+Na7v/
SJnA086L7cuG6PFHyt8tBYF7l3WV465GP5PW3bJc0TQnNRUoisMzEH0CRlwYqCHj
YIOeZWFCJeyR52Y8qI2yG/hgKMJQ4e/0vFy+y7l0dU6TSKZ96E3XEVjLPsHKCW0S
aEX1gLQdtoTQciwAnzrl9uu/xbyoW7CLTg7Hny8LgJZr+YQaILd1wljEX3Kflie
N7stTCj0eF8AH2LgPMhvJ9Zy60GdVw/oU9rJXhQD3Z6WpQkCrrfE4/LymuXGdmGI
bG1CpuHB8Vlij4om2Yn0ZebPiDi0LZ9cawanmZkm3YedPbYJbv1NW9koToLKbuW8
1ZDyzcpFj0aYBtut6RdvNYfZl0mJPd8jyKuGKRYnqDbeQ/Snk74BNkBiV+02ozGF
aGd1s45A94fFAtzf0bY0HZ4JucjspJK6Y2yphPKhPHQiW7vm1mlvRgY3WN6DINq4
+3Irdwpxky7hxPToh2aUjz1TLdruqR1dk+JSV1+SFypoBhPB1bx/zglh0ncttbeM
5/Khgu+uXp+ePf7g64lnPFjVIOvBega4CAMHQqffa5c2z3f9qKThIH1Igs5e7G0R
GHL57MVtvBjUtUyc8eFx4EzgmWVV7D/JzxVDpUBWeAnlyNe0oxjuf2tkZgb/+Gni
tlbRN/s0Yo8lnbEuplF9UQ==
-----END ENCRYPTED PRIVATE KEY-----
-----BEGIN CERTIFICATE REQUEST-----
MIIC4TCCAckCAQAwZsxCzAJBgNVBAYTAkRFMRywFAYDVQQIDA1FWEFNUEXFLVNU
QVRFRMRUwEwYDVQQHDAxFeGFtcGx1IENpdHkxWDASBgNVBAoMC0V4YW1wbGUgTHRk
MQswCQYDVQQQLDAJJVDEUMBIGA1UEAwwLZXhhbXBsZS5jb20xJDAiBgkqhkiG9w0B
CQEWFxd1Ym1hc3RlckBleGFtcGx1LmNvbTCCASIwDQYJKoZIhvcNAQEBBQADggEP
ADCCAQoCggEBAMAc6BwShsBW1lDcsHnF88m99fDiDESd9zq4VCq+wpcc6RjPmAf
7uG8ha+CSiyvr9fDgJfzMDpDUG5W2R+WLQhe+9IS8VFHqpT7ePwHlFt7h+0Rx3ke
us9ErY085jFHvGziuhuh4fGE0Ez69LT0ZujeeNad2GEuGQSRTWuloNVTyFy0hIb2
f6fAsGBGbW9uSjW8L4hIPpmUgLSLDZ9Ev6p9/MrNITeUwkIi+FaM3NYrf9Xx2MI2
/fr17ZTl2uGyW8uh24sd6vMoaw1arI+RI/mcxyvADP5qAtnDRhXCPgZBPI0sFpX4
MAVj99+VvxahSubB3yxca8DDFn8cbBfZMBUCAwEAaAAMA0GCSqGSIB3DQEBCwUA
A4IBAQAQDSBAnoTFFUphk7EnMRM34oFg4tzoGkV46eLLnUlu0cLmggzuw+BBg0/Zx
3vVjqu7wudr9bg+u0vNHc0UjuAmU5NVhwb4WHBZHw5Lba6WNKV LZYoC4ZzeU1MyW
TYh7076mZzmXVpppotcK5uU5ojDhiiaxkQuqWlSk3txdf/qJ/1F/pFnHRydT0hMf
JQBYo7FLIa+ZuHEysjoUmoo0arLJyuymlWYWFmzV/ds0r4ci9AtUfUUbt3JgcVhZ
+h0KQNf26D96c9C+5v5PnV2pVs4tIzY08HQKEmbWqKyTWTowID6Qi/dNqhDwt7B
UN4g4tNW1wIRPW76nHKGsgTRJFRq
-----END CERTIFICATE REQUEST-----
```

Well, that's not very useful for the human mind so in the next section, you'll find the commands to review your private key's parameters and the CSR in a more human-readable form.

## Reviewing the private key

**Attention:** Never share your private key(s) with anyone. They are yours and yours only. Keep them secret, safe, and sound. Follow your organization's policies for handling private keys, CSRs, and certificates. The security of your organization (and your job) may depend on it.

The following example shows you how a private key looks like. In normal operation, there is usually no need to print out a private key like that.

```
$ openssl pkey -text -noout -in example.com.key
Enter pass phrase for example.com.key:
RSA Private-Key: (2048 bit, 2 primes)
modulus:
 00:c0:1c:e8:1c:12:86:c0:56:8b:50:dc:b0:79:c5:
 f3:c9:bd:f5:f0:e2:0c:44:a9:0f:dc:ea:e1:50:aa:
 fb:0a:5c:73:a4:63:3e:60:1f:ee:e1:bc:85:af:82:
 4a:2c:af:af:d7:c3:80:97:f3:30:3a:43:50:6e:56:
 d9:1f:96:2d:08:5e:fb:d2:12:f1:51:47:aa:94:fb:
 78:fc:07:94:5b:7b:87:ed:11:c7:79:1e:ba:cf:44:
 ad:8d:3c:e6:31:47:bc:6c:e2:ba:1b:a1:e1:f1:84:
 d0:4c:fa:f4:b4:f4:66:e8:de:78:d6:9d:d8:61:2e:
 19:04:91:4d:6b:a5:a0:d5:53:c8:5c:b4:84:86:f6:
 7f:a7:c0:b0:60:46:6d:6f:6e:4a:35:bc:2f:88:48:
 3e:99:94:80:be:4b:0d:9f:44:bf:aa:7d:fc:ca:cd:
 21:37:94:c2:42:22:f8:56:8c:dc:d6:2b:7f:d5:f1:
 d8:c2:36:fd:fa:e5:ed:94:e5:da:e1:b2:5b:cb:a1:
 db:8b:1d:ea:f3:28:6b:0d:5a:ac:8f:91:23:f9:9c:
 c7:2b:c0:0c:fe:6a:02:d9:c3:46:15:c2:3e:06:41:
 3c:8d:2c:16:95:f8:30:05:63:f7:df:95:bf:16:a1:
 4a:e6:c1:df:2c:5c:6b:c0:c3:16:7f:1c:6c:17:d9:
 30:15
publicExponent: 65537 (0x10001)
privateExponent:
 3d:34:f8:7e:79:28:95:7e:fd:43:f6:0c:03:c0:1d:
 bb:d8:d9:d2:b5:32:53:6d:c9:b0:08:e5:60:5e:19:
 3d:63:d7:34:38:aa:56:d5:d5:b5:f5:ea:61:d0:90:
 f1:4b:c8:2a:66:0f:42:5a:28:b3:67:5e:e8:c8:a2:
 c4:7d:84:3b:76:87:a4:96:84:6b:f0:f9:58:1e:06:
 9d:c5:8c:6e:d9:1c:a5:5b:54:c2:32:18:32:91:1e:
 fa:30:bc:e6:56:84:a1:ec:5a:7f:13:44:79:3e:67:
 5f:1f:55:77:91:a6:77:ea:e9:74:f7:35:49:40:f6:
 97:8d:f3:ac:a8:48:65:ae:8e:84:34:16:d8:e8:7d:
 71:b2:30:29:df:fd:db:ff:8a:0f:68:af:d5:76:48:
 15:ec:78:14:51:1b:13:de:45:d0:73:7d:cd:32:9a:
 0c:08:3f:1e:2a:67:c4:db:51:b8:74:e2:1f:ba:a7:
 86:6c:9f:7f:9d:49:75:5a:54:47:3d:01:e6:64:83:
 7d:6b:d9:52:0a:d8:f3:a7:1a:28:61:f2:01:9b:32:
 e2:cf:99:2b:3a:35:f1:21:88:f8:37:c7:80:80:69:
 2e:2b:05:9b:00:c8:49:a9:08:a9:bf:d3:74:e0:0c:
 18:13:35:74:e9:03:d6:6b:6b:09:bd:b3:9c:b9:04:
 c1
prime1:
 00:e0:d6:d1:18:cd:60:b2:18:29:93:3f:7a:79:e3:
 9b:59:f6:f6:52:74:3d:65:df:92:8f:2c:dc:fb:06:
 ca:c1:e6:85:d7:2e:a8:1c:04:1d:07:e1:be:0a:64:
 e4:75:5f:44:71:6b:a7:16:94:46:81:e0:65:b3:09:
 3a:55:0e:a8:f0:d9:ee:9f:4e:b4:46:c9:9a:32:b7:
 f4:63:9f:b7:e2:54:8c:6f:8f:52:a1:98:07:34:04:
 da:56:38:02:f0:4b:54:48:c1:20:29:23:08:19:93:
 69:4e:7a:a5:ab:e4:8a:c3:f5:85:37:82:fa:73:ca:
 51:fa:f6:9d:4b:d9:ae:b7:eb
prime2:
 00:da:bc:fa:d3:56:b9:6d:82:73:95:9f:b7:42:42:
 4a:eb:5a:98:5c:9c:4d:f7:60:1b:46:1b:6b:87:a5:
 b3:8b:ce:a4:fe:2b:ee:65:89:5c:85:22:e0:71:62:
 cf:2a:36:ac:aa:1d:d3:66:29:c4:dc:dc:5a:d8:6d:
 ce:52:9a:ea:b4:12:c6:24:79:d2:50:08:27:e1:a5:
 3d:50:01:5a:fe:1e:86:c5:e1:d7:34:24:d3:00:84:
 fa:45:ab:20:69:17:9d:c7:16:8a:04:85:e3:4b:df:
 08:5d:71:63:74:d4:78:fd:ea:c5:9c:61:68:55:bc:
 f9:3b:0d:96:80:a1:45:b7:ff
exponent1:
 19:ce:4e:1d:8c:a5:06:8b:e3:69:b5:25:77:8b:fa:
 2e:af:3b:c2:66:f9:0d:12:46:1e:0c:c6:28:41:b5:
```

```
4f:e0:07:88:95:20:52:66:de:76:23:20:ce:cc:99:
b4:27:05:12:07:8b:1b:fa:54:c5:5f:5e:0c:d8:88:
4b:6e:ed:51:07:92:6d:d2:78:ba:8f:35:15:91:2b:
89:ab:b1:4d:56:ea:ef:7b:01:be:ee:56:15:50:61:
46:f8:12:ce:45:b7:1c:ca:9d:c8:5a:ee:f9:10:84:
4b:af:a4:08:40:a8:f4:a8:df:6b:69:ab:19:53:25:
69:aa:98:8f:36:22:e9:95
```

exponent2:

```
68:45:57:e8:30:da:39:da:ca:d0:93:8f:5f:75:8c:
93:3e:df:8c:9e:32:08:6f:76:f0:e7:97:4d:d5:6e:
8e:81:d6:63:17:7c:10:48:f7:a2:bb:aa:74:42:9e:
f0:c7:99:6d:8a:c4:40:5a:3e:ee:ed:d8:1c:7d:d3:
b1:cb:09:81:07:c6:0e:93:47:ef:40:c4:0a:2b:a3:
db:a4:99:c4:b3:b6:99:53:fc:2b:6e:36:6b:73:f8:
7e:07:82:6e:b6:84:4e:e8:6e:a8:93:4e:73:d7:80:
fc:52:56:0f:d6:4d:4a:f5:84:77:f4:73:31:13:e1:
57:06:36:2d:61:33:83:ef
```

coefficient:

```
00:99:64:d8:95:ad:72:3b:f0:a9:4d:d5:94:72:e3:
5e:bd:17:42:1a:65:c9:1b:bb:eb:e8:09:1a:f8:c6:
93:ee:9d:1d:39:b0:4f:35:70:a6:c3:0c:1f:45:fa:
4b:e3:07:d3:00:6d:20:c7:d9:07:7f:2f:11:25:81:
45:52:21:be:c6:39:a4:42:3d:a8:29:22:a4:80:69:
0f:b6:aa:39:c5:9d:35:81:8b:36:20:f4:93:0c:2c:
4e:25:95:e9:02:5e:ed:97:6d:84:70:97:11:78:26:
4f:f6:76:7b:f2:b9:3c:08:59:6c:44:50:16:f0:41:
51:16:0a:b4:74:92:f3:3c:2c
```

## Reviewing the CSR

The following example shows you how to review your CSR before submitting it to a CA for signing.

```
$ openssl req -text -noout -in example.com.csr
Certificate Request:
  Data:
    Version: 1 (0x0)
    Subject: C = DE, ST = EXAMPLE-STATE, L = Example City, O = Example Ltd, OU = IT, CN =
example.com, emailAddress = webmaster@example.com
    Subject Public Key Info:
      Public Key Algorithm: rsaEncryption
      RSA Public-Key: (2048 bit)
      Modulus:
        00:c0:1c:e8:1c:12:86:c0:56:8b:50:dc:b0:79:c5:
        f3:c9:bd:f5:f0:e2:0c:44:a9:0f:dc:ea:e1:50:aa:
        fb:0a:5c:73:a4:63:3e:60:1f:ee:e1:bc:85:af:82:
        4a:2c:af:af:d7:c3:80:97:f3:30:3a:43:50:6e:56:
        d9:1f:96:2d:08:5e:fb:d2:12:f1:51:47:aa:94:fb:
        78:fc:07:94:5b:7b:87:ed:11:c7:79:1e:ba:cf:44:
        ad:8d:3c:e6:31:47:bc:6c:e2:ba:1b:a1:e1:f1:84:
        d0:4c:fa:f4:b4:f4:66:e8:de:78:d6:9d:d8:61:2e:
        19:04:91:4d:6b:a5:a0:d5:53:c8:5c:b4:84:86:f6:
        7f:a7:c0:b0:60:46:6d:6f:6e:4a:35:bc:2f:88:48:
        3e:99:94:80:be:4b:0d:9f:44:bf:aa:7d:fc:ca:cd:
        21:37:94:c2:42:22:f8:56:8c:dc:d6:2b:7f:d5:f1:
        d8:c2:36:fd:fa:e5:ed:94:e5:da:e1:b2:5b:cb:a1:
        db:8b:1d:ea:f3:28:6b:0d:5a:ac:8f:91:23:f9:9c:
        c7:2b:c0:0c:fe:6a:02:d9:c3:46:15:c2:3e:06:41:
        3c:8d:2c:16:95:f8:30:05:63:f7:df:95:bf:16:a1:
        4a:e6:c1:df:2c:5c:6b:c0:c3:16:7f:1c:6c:17:d9:
        30:15
      Exponent: 65537 (0x10001)
    Attributes:
      a0:00
  Signature Algorithm: sha256WithRSAEncryption
    03:48:10:27:a1:31:45:52:98:64:ec:49:cc:44:cd:f8:a0:58:
    38:b7:3a:06:91:5e:3a:78:b2:e7:52:5b:b4:70:b9:a0:83:3b:
    b0:f8:10:60:d3:f6:71:de:f5:63:aa:ee:f0:b9:da:fd:6e:0f:
    ae:d2:f3:47:73:45:23:b8:09:94:e4:d5:61:c1:be:16:1c:16:
    47:c3:92:db:6b:a5:8d:29:52:f3:62:80:b8:67:37:94:d4:cc:
    96:4d:88:7b:3b:be:a6:67:39:97:56:9a:69:a2:d7:0a:e6:e5:
    39:a2:30:e1:8a:26:b1:91:0b:aa:5a:5b:24:de:dc:5d:7f:fa:
    89:ff:51:7f:a4:59:c7:47:27:53:d2:13:1f:25:00:58:a3:b1:
    4b:21:af:99:b8:71:32:b2:3a:14:9a:8a:34:6a:b2:c9:ca:ec:
    a6:95:66:16:14:cc:d5:fd:db:34:af:87:22:f4:0b:54:7d:45:
    1b:b7:72:60:71:58:59:fa:1d:0a:40:d7:f6:e8:3f:7a:73:d0:
    be:e6:fe:4f:9d:5d:a9:56:ce:2d:23:36:34:f0:74:0a:12:66:
    d6:a8:ac:93:59:3a:31:c0:80:fa:42:2f:dd:36:a8:43:c2:de:
    c1:50:de:20:e2:d3:56:d7:02:11:3d:6e:fa:9c:72:86:b2:04:
    d1:24:54:6a
```

## Creating a CSR including multiple domain names

There are situations where your application is reachable under different domain names like:

- example.com
- www.example.com
- app.example.com

To create a CSR that contains all three domain names you can use a configuration file. For example, use your favorite text editor to create a file with the following content:

```
[ req ]
default_bits = 2048
default_keyfile = test_privatekey.pem
distinguished_name = req_distinguished_name
encrypt_key = no
prompt = no
string_mask = nombstr
req_extensions = v3_req

[ v3_req ]
basicConstraints = CA:FALSE
keyUsage = digitalSignature, keyEncipherment, dataEncipherment
extendedKeyUsage = serverAuth, clientAuth
subjectAltName = DNS:example.com, DNS:www.example.com, DNS: app.example.com

[ req_distinguished_name ]
countryName = DE
stateOrProvinceName = EXAMPLE-STATE
localityName = Example City
0.organizationName = Example Ltd.
organizationalUnitName = IT
commonName = example.com
```

The first section **[ req ]** specifies that a private RSA key with 2048 bits is to be generated and stored as `test_privatekey.pem`. Also, the section contains information about finding the bits that you entered interactively in the earlier section of this article (in the section **[ req\_distinguished\_name ]**). In **[ v3\_req ]**, you'll find some constraints on **keyUsage** but more importantly, for this article, the parameter **subjectAltName** where the common name and all additional names are specified. Save it as `openssl.cnf` and run it with the following command to create a private key and CSR:

```
$ openssl req -batch -new -config openssl.cnf -out example.com.csr
Generating a RSA private key
.....+++++
.....+++++
writing new private key to 'test_privatekey.pem'
-----

$ ls
example.com.csr  openssl.cnf  test_privatekey.pem
```

Be aware in this example, a passphrase does not protect the private key. This comes in handy when you would like a webserver to be able to read it. But keep an eye on it and keep it safe.

The CSR is stored in `example.com.csr` and you can check the content as before:



```

$ openssl req -text -noout -in example.com.csr
Certificate Request:
  Data:
    Version: 1 (0x0)
    Subject: C = DE, ST = EXAMPLE-STATE, L = Example City, O = Example Ltd., OU = IT, CN =
example.com
    Subject Public Key Info:
      Public Key Algorithm: rsaEncryption
      RSA Public-Key: (2048 bit)
      Modulus:
        00:b4:ce:85:79:cc:74:f4:36:13:fe:c0:ff:57:6a:
        31:0f:68:80:17:eb:f4:0f:b5:3f:56:dd:34:c8:4d:
        a8:3f:f0:5f:fa:0a:f8:94:4d:3b:92:59:05:c2:c3:
        8f:54:40:5d:db:d7:d8:f8:e5:a7:f4:a4:d5:77:18:
        cc:dc:21:37:f1:72:23:28:16:f2:fe:03:67:a1:e3:
        22:3d:85:80:0b:dd:e2:44:95:97:5f:12:3d:ab:f2:
        8c:0d:71:c3:8a:76:4f:d1:5b:89:5b:14:c4:e3:33:
        2e:5f:5a:1d:46:e0:4b:7b:f0:cf:ba:a5:ab:35:7f:
        57:70:b0:97:0d:47:d8:3e:0c:ef:15:45:2d:c2:df:
        d7:3b:91:6f:1b:2e:96:e3:a5:3f:32:32:ad:67:57:
        5b:72:09:35:a8:bc:e6:78:07:c5:65:ea:f7:29:3d:
        77:2e:3c:6b:e8:94:49:2b:28:de:9c:11:92:bf:05:
        b8:ab:05:d1:51:78:82:13:58:f4:79:4f:ad:f4:ec:
        42:8d:74:12:22:48:c0:6f:87:95:45:2d:8e:6d:2f:
        78:ab:d9:41:c5:56:bb:ed:b9:5d:23:d9:6e:f9:80:
        88:3d:2b:f2:f9:a3:45:98:15:99:19:c0:65:22:df:
        8a:b2:e1:71:9e:7f:42:8d:6f:ce:9a:81:18:a4:22:
        b3:85
      Exponent: 65537 (0x10001)
    Attributes:
    Requested Extensions:
      X509v3 Basic Constraints:
        CA:FALSE
      X509v3 Key Usage:
        Digital Signature, Key Encipherment, Data Encipherment
      X509v3 Extended Key Usage:
        TLS Web Server Authentication, TLS Web Client Authentication
      X509v3 Subject Alternative Name:
        DNS:example.com, DNS:www.example.com, DNS:app.example.com
    Signature Algorithm: sha256WithRSAEncryption
      96:4b:00:08:e3:44:20:fe:b4:45:01:ac:e6:28:27:f9:22:e4:
      da:ea:99:45:d3:dd:44:04:b6:c8:29:2f:a9:5d:f2:bb:92:6e:
      86:a4:c6:22:fb:1b:c0:89:29:bc:00:b4:69:bc:cf:29:01:e7:
      b3:d2:b6:f5:cc:95:b1:d1:a2:f6:8d:b4:40:a4:94:4f:7f:1d:
      cc:aa:d4:9c:7e:6d:53:2d:eb:2a:7a:2f:0e:b8:c0:4d:13:71:
      19:dd:30:2c:d6:ed:03:a9:70:ff:90:71:82:6a:4e:64:93:1e:
      60:c4:f3:cc:16:07:81:47:63:e9:4a:f5:79:99:b8:5e:e0:e1:
      bb:68:8b:45:08:91:4a:d6:dc:70:e6:b4:75:11:77:a6:3d:54:
      64:55:50:9f:27:1d:fe:d2:96:19:7b:d2:10:a5:4b:ef:33:6e:
      9d:31:5b:a3:4e:89:11:b6:09:c6:07:d6:c9:8e:e7:88:c9:be:
      9a:78:3e:2c:82:76:08:20:e0:1a:e8:9e:fd:7a:b0:3c:6b:5c:
      60:a1:77:98:7c:dd:98:93:c6:d8:d3:f4:de:9a:8f:5a:f3:10:
      d1:df:18:a7:7d:bc:2c:50:4e:b9:bf:90:62:57:46:2f:39:b9:
      82:1d:5a:1c:bc:92:9f:5b:7e:e0:ac:7d:29:63:fb:04:e9:71:
      68:53:33:90

```

You should recognize the section **x509v3 Subject Alternative Name** containing all the domain names you need.

---

## Kubernetes and OpenShift

- [Kubernetes cheat sheet](#)
- [Kubernetes: Everything you need to](#)

[know](#)

- [Interactive course: Getting started with OpenShift](#)
- [Red Hat OpenShift and Kubernetes ... what's the difference?](#)
- [Interactive course: Deploy a cluster in Red Hat OpenShift Service on AWS \(ROSA\)](#)

Whether you need only one domain name or more in your certificate, you now know how to generate the necessary CSR. For an in-depth understanding, I suggest that you refer to the book I recommended above.

## The Internet PKI

Now that you have a CSR ready to get signed by a trusted CA, it's time to examine the Internet PKI. Here's a basic overview. For details, refer to [RFC 5280](#).

In the Internet PKI there are the following roles:

**Subscriber** - Someone who would like to provide a TLS/SSL secured service. So probably you and me.

**Certification Authority (CA)** - Verifies the identity of subscribers or their domains and issues certificates that could be installed on web servers. It also provides information about certificates that have been revoked.

**Relying Party** - This could be a web browser or some other kind of client which tries to validate the certificate sent by your web server. So-called trust anchors are used to verify the certificate. Trust anchors are certificates that are completely trusted and are kept in the browser's trusted CA store.

Usually, the following workflow is completed before a web browser verifies a certificate from your web server.

1. Generate private key with OpenSSL
2. Create CSR with OpenSSL
3. Submit CSR to CA for signing
4. Receive signed certificate from CA
5. Install private key and certificate on your web server
6. Your users/customers can start using your site/app

Remember what you already know about public-key encryption. You could use the private key to sign a message and use the corresponding public key to verify the signature. Something very similar happens when the CA signs your public key and issues your certificate.

The certificate itself is a data structure that includes some information about you or your organization. It contains the domain name, your public key, the name of the CA that issued the certificate, and the CA's signature. To give you an example, I'll show you the certificate that is currently in use on my personal blog and explain the most important sections of it (shortened for a better overview).

```
$ echo "" | openssl s_client -connect www.my-it-brain.de:443 | openssl x509 -text -noout
depth=2 0 = Digital Signature Trust Co., CN = DST Root CA X3
verify return:1
depth=1 C = US, 0 = Let's Encrypt, CN = R3
verify return:1
depth=0 CN = www.my-it-brain.de
verify return:1
DONE
Certificate:
    Data:
        Version: 3 (0x2)
        Serial Number:
            04:50:26:2d:14:91:0f:23:32:af:19:d8:38:a6:00:cf:b4:d6
        Signature Algorithm: sha256WithRSAEncryption
        Issuer: C = US, 0 = Let's Encrypt, CN = R3
    Validity
        Not Before: Feb 16 22:34:35 2021 GMT
        Not After : May 17 22:34:35 2021 GMT
    Subject: CN = www.my-it-brain.de
    Subject Public Key Info:
        Public Key Algorithm: rsaEncryption
            RSA Public-Key: (2048 bit)
            Modulus:
                [...]
            Exponent: 65537 (0x10001)
    X509v3 extensions:
    [...]

```

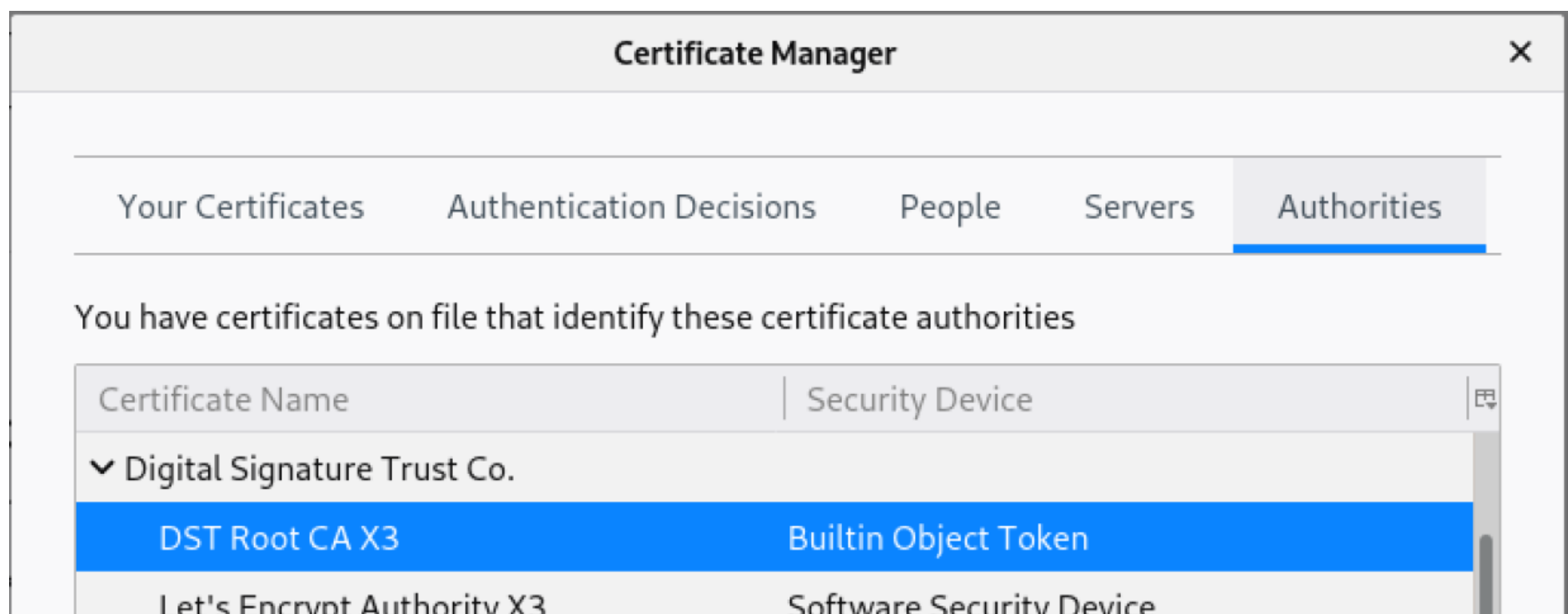
The part after "**Certificate:**" contains information like the serial number, the signature algorithm used to sign the public key, the name of the issuer, the timeframe in which this cert is valid, the domain name, and the public key. But look at the very first lines as they show the signing path:

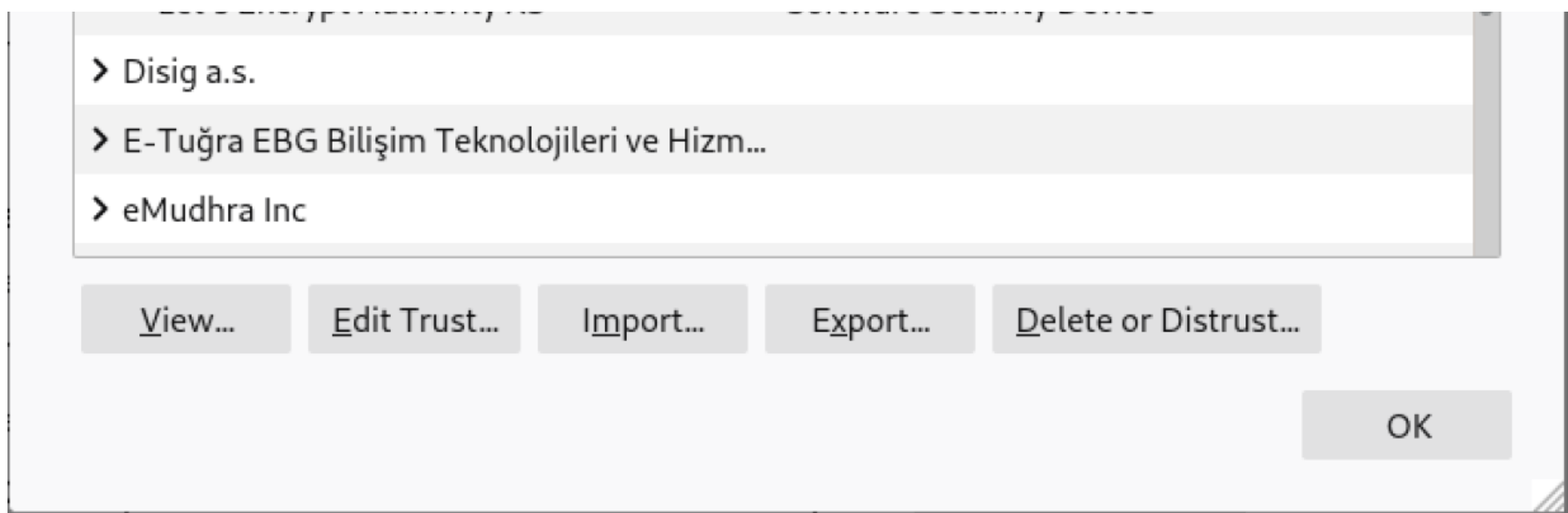
```
depth=2 0 = Digital Signature Trust Co., CN = DST Root CA X3
verify return:1
depth=1 C = US, 0 = Let's Encrypt, CN = R3
verify return:1
depth=0 CN = www.my-it-brain.de
verify return:1
DONE

```

What you see is the chain of trust. You could see that this certificate (the public key associated with the FQDN [www.my-it-brain.de](http://www.my-it-brain.de)) was signed by *Let's Encrypt R3* (R3 is the name of the certificate) which in turn was signed by Digital Signature Trust Co. DST Root CA X3 (here DST Root CA X3 is the name of the certificate).

But how does your browser know whether to trust one of these certificates? The DST Root CA X3 certificate is stored in your browser's trust anchor. See the following image, which shows my browser's trust anchor:





This entry tells your browser to trust all certificates that were signed by DST Root CA X3. A chain of trust is built up to my leaf certificate for [www.my-it-brain.de](http://www.my-it-brain.de). The following figure illustrates how the chain of trust works.

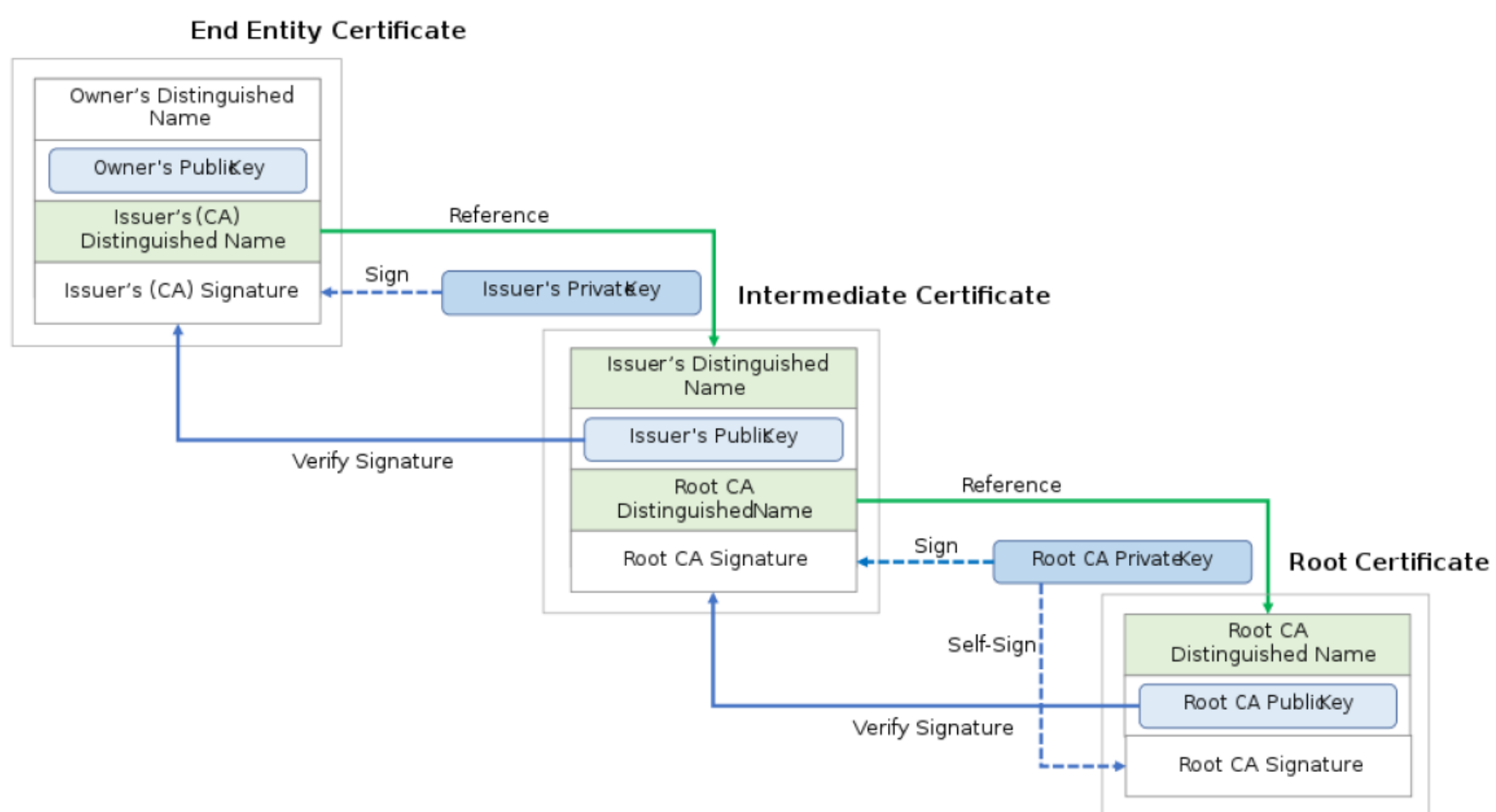


Image Source: [Yuhkih](#), [CC BY-SA 4.0](#).

Notice that between your end-entity certificate and a trust anchor in your browser's store there may be more than one intermediate certificate. It's important that your browser can verify each certificate along the chain to verify your leaf certificate. That's important to notice because if your browser misses one of the intermediate certificates, your certificate's verification will fail.

Usually, you'll get a full chain certificate from your CA, including your certificate and all intermediates involved, down to some root cert residing in your browser trust anchor. It's considered a best practice to configure your webserver to deliver the full chain to make sure your users have a nice experience visiting your site.

**[ Thinking about security? [Check out this free guide to boosting hybrid cloud security and protecting your business.](#) ]**

## Wrap up

In the first article of this series, you learned the basics of encryption concepts. In this article, you received a high-level overview about TLS/SSL and the OpenSSL tool, learning how to create private keys and CSRs, which you could send to a CA for signing. Also, the Internet PKI was introduced with the chain of trust, showing how the verification process works.

In the next article of this series, I'll look at some Internet PKI issues and what you can do about them.

\*Of course, there is another reason: Your website will receive a lower ranking in search results if not using TLS/SSL encryption.





## [Locking down sshd](#)

Learn how to keep your systems safe and prevent unauthorized access through SSH by following these simple suggestions.

Posted: July 19, 2019

Author: [Nathan Lager](#) (Sudoer, Red Hat)



This content is the author's employer or of Red Hat. The content published on this site are NOT, AND ARE NOT INTENDED TO BE, RED HAT DOCUMENTATION, SUPPORT, OR ADVICE.

## [A little SSH file copy magic at the command line](#)

You might not be aware that SSH is a magical tool with many different uses. Using it, you can copy files between systems without logging into them, as if by magic.

Posted: October 10, 2019

This content is the author's employer or of Red Hat. The content published on this site are NOT, AND ARE NOT INTENDED TO BE, RED HAT DOCUMENTATION, SUPPORT, OR ADVICE.

Author: [Ken Hess](#) (Sudoer alumni)

Red Hat and the Red Hat logo are trademarks of Red Hat, Inc., registered in the United States and other countries.



Ab  
Jo  
Ev  
Lo  
Co  
Re  
Div

[Cool Stuff Store](#)

[Red Hat Summit](#)

## [SSHFS: Mounting a remote file system over SSH](#)

Learn how to securely mount a remote file system from another server locally on your machine with SSHFS.

Posted: January 6, 2020

Author: [Chris Collins](#) (Sudoer alumni, Red Hat)

© 2023 Red Hat, Inc.

[Privacy statement](#)

[Terms of use](#)

[All policies and guidelines](#)

[Digital accessibility](#)

[Cookie preferences](#)



## Jörg Kastning

Jörg has been a Sysadmin for over ten years now. His fields of operation include Virtualization (VMware), Linux System Administration and Automation (RHEL), Firewalling (Forcepoint), and Loadbalancing (F5). [More about me](#)

# Try Red Hat Enterprise Linux

Download it at no charge from the Red Hat Developer program.

## Related Content



### [8 open source 'Easter eggs' to have fun with your Linux terminal](#)

Hunt these 8 hidden or surprising features to make your Linux experience more entertaining.

Posted: April 10, 2023

Author: [Ricardo Gerardi](#) ([Editorial Team](#), [Sudoer alumni](#), [Red Hat](#))



### [Troubleshooting Linux performance, building a golden image for your RHEL homelab, and more tips for sysadmins](#)

Check out Enable Sysadmin's top 10 articles from March 2023.

Posted: April 4, 2023

Author: [Vicki Walker](#) ([Editorial Team](#), [Red Hat](#))



### [Do advanced Linux disk usage diagnostics with this sysadmin tool](#)

Use topdiskconsumer to address disk space issues when you're unable to interrupt production.

Posted: March 31, 2023

Author: [Kimberly Lazarski](#) ([Red Hat](#))