Fundusze Europejskie Wiedza Edukacja Rozwój
Politechnika Wrocławska
Unia Europejska Europejski Fundusz Społeczny

„ZPR PWr – Zintegrowany Program Rozwoju Politechniki Wrocławskiej”

System Security I - Exercise Notes on PKI

Przemysław Kubiak

30.06.2019

Contents

Introduction

OpenSSL is a widely used library to implement TLS protocol. The TLS protocol is used for secure connection between client application and the service server, and is composed from two main stages:

- TLS handshake,
- an the TLS record protocol that is used to encrypt data sent between the client and the server.

During the TLS handshake

- both sides negotiate the algorithms and protocols used,
- the server authenticates itself against the client,
- optionaly if the server requests the client authenticates itself against the server,
- and both sides establish session keys for symmetric encryption and message authentication.

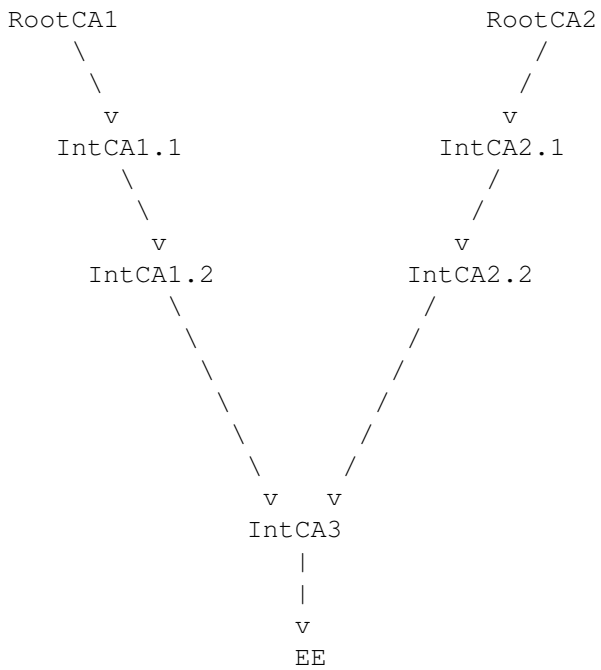
In this report we shall investigate how OpenSSL library helps in implementing some feature of client authentication in TLS handshake, namely the functionality of chosing by the client a certificate path that the server will accept.
For this purpose:

1. For a single client (End Entity) we shall build two certification paths.
2. We shall find (with the gnu debugger) what procedures in the OpenSSL library can be used by the client to compose from a given set of certificates the subset forming a chain that ends with a CA certificate that the server will trust.
3. Then we shall make a short implementation of the core functionality.

Chapter 1

Certificate paths generation with openssl

We are going to build the following hierarchy of Root CA, Intermediate CA, and End Entity (EE) certificates. For this purpose we shall follow the instructions from https://jamielinux.com/docs/openssl-certificate-authority/index.html This is a simple example but much more complicated hierarchy could be used in practice.
Later on all certificates from such hierarchy are to be stored in client's memory, and during the TLS handshake when the server says that it trusts e.g., RootCA2 certification authority, client's task is to find on the fly a path of certificates from the EE certificate to the RootCA2 certificate.



By building the above hierarchy we shall learn a few options of openssl commandline tools and solve some additional tasks related to X.509 certificates. **We shall create certificates for different cryptosystems: elliptic curve cryptosystems, RSA and DSA**.

1.1 RootCa1

1.1.1 Generate RootCa1 keypair – an elliptic curve key pair

What curve to choose?

```
....
  brainpoolP256r1: RFC 5639 curve over a 256 bit prime field
  brainpoolP256t1: RFC 5639 curve over a 256 bit prime field
  brainpoolP320r1: RFC 5639 curve over a 320 bit prime field
  brainpoolP320t1: RFC 5639 curve over a 320 bit prime field
  brainpoolP384r1: RFC 5639 curve over a 384 bit prime field
  brainpoolP384t1: RFC 5639 curve over a 384 bit prime field
  brainpoolP512r1: RFC 5639 curve over a 512 bit prime field
  brainpoolP512t1: RFC 5639 curve over a 512 bit prime field
```

Generate an EC private key, and output it to a file named RootCa1PrivateKey.pem:

```
openssl ecparam -name brainpoolP512r1 -genkey -out ./private/RootCa1PrivateKey.pem

chmod 400 private/RootCa1PrivateKey.pem
```

Extract the public key from the key pair, which can be used in a certificate:

```
openssl ec -in ./private/RootCa1PrivateKey.pem -pubout -out ./public/RootCa1PublicKey.pem

chmod 400 public/RootCa1PublicKey.pem

openssl ec -inform PEM -pubin -in ./public/RootCa1PublicKey.pem -text -noout


read EC key
Public-Key: (512 bit)
pub:
    04:6b:17:9d:e5:22:e3:d0:aa:ee:a3:01:4e:1d:77:
    76:ac:9f:c9:cd:02:a9:ed:0b:75:68:42:79:a3:14:
    b3:ba:78:ab:d5:f2:21:2c:f7:a6:8f:82:b2:dd:92:
    7e:a5:37:6c:ff:c7:86:72:61:b9:3c:79:9a:24:c3:
    02:74:ad:33:23:a6:93:7c:fb:ed:02:87:42:d3:ee:
    a2:bd:c3:7f:a9:54:af:77:6e:e8:d0:dc:0a:3f:95:
    9a:00:0d:c9:26:18:6c:ca:9e:1e:c2:7e:c3:95:60:
    80:d0:d4:ad:1e:1e:09:ca:40:eb:3f:fa:2e:b6:80:
    4b:e2:08:bb:f3:e7:ac:68:22
ASN1 OID: brainpoolP512r1
```

The ./public/RootCa1PublicKey.pem file decoded on https://lapo.it/asn1js/ :

```
SEQUENCE (2 elem)
  SEQUENCE (2 elem)
    OBJECT IDENTIFIER 1.2.840.10045.2.1 ecPublicKey (ANSI X9.62 public key type)
    OBJECT IDENTIFIER 1.3.36.3.3.2.8.1.1.13 brainpoolP512r1 (ECC Brainpool Standard Curves and Curve Generation)
  BIT STRING (1032 bit) 000001000110101100010111100111011110010100100010111000111101000010101...
```

Let consider the above public key format in more detail:

- What is an object identifier (OID)? - please refer to http://www.oid-info.com/#oid How to register OID? – cf. e.g., https://docs.microsoft.com/pl-pl/windows/desktop/AD/obtaining-an-object-identifier Polish body for registering new OIDs - see https://www.krio.pl/krio/krio,onas_inf_ogolne.xml
- The whole structure above corresponds to (see sect.2 of https://tools.ietf.org/html/rfc5480):

```
SubjectPublicKeyInfo  ::=  SEQUENCE  {
  algorithm         AlgorithmIdentifier,
  subjectPublicKey  BIT STRING
}


AlgorithmIdentifier  ::=  SEQUENCE  {
  algorithm   OBJECT IDENTIFIER,
  parameters  ANY DEFINED BY algorithm OPTIONAL
}
```

- The ”unrestricted” algorithm identifier is:

```
id-ecPublicKey OBJECT IDENTIFIER ::= {
   iso(1) member-body(2) us(840) ansi-X9-62(10045) keyType(2) 1 }
```

  It is unrestricted in the sense that indicates that the algorithms that can be used with the subject public key are unrestricted.

- The parameter for the above id-ecPublicKey AlgorithmIdentifier is as follows and MUST always be present:

```
ECParameters ::= CHOICE {
  namedCurve         OBJECT IDENTIFIER
  -- implicitCurve   NULL
  -- specifiedCurve  SpecifiedECDomain
}
```

  The CHOICE type has no (default) universal class tag, because the CHOICE type does not exists ‘as is’; it is only a collection of several types among which one of them is chosen to be encoded with its associated tag. That is why we see the named curve immediately after the ”unrestricted” algorithm identifier.

- the named curve is brainpoolP512r1,
- the OBJECT IDENTIFIER (OID) of brainpoolP512r1: see sect.4.1 of https://tools.ietf.org/html/rfc5639
- The bitstring represents uncompressed EC point: the first octet 0x04 indicates lack of compression - cf. sect.2.2 of https://tools.ietf.org/html/rfc5480.
- it is easy to check that the hexadecimal representation of the bitstring is

```
0000 0100  0110 1011  0001  0111 1001 1101 ....
0    4     6    B     1     7    9    d ...
```

1.1.2 Generate certificate for RootCa1

- The process takes place on EC so consult also https://knowledge.digicert.com/generalinformation/INFO1909.html
- For allowed hash functions see openssl-1.1.1a/crypto/ec/ec_pmeth.c - Ctr-F for EVP_PKEY_CTRL_MD:

```
case EVP_PKEY_CTRL_MD:
    if (EVP_MD_type((const EVP_MD *)p2) != NID_sha1 &&
        EVP_MD_type((const EVP_MD *)p2) != NID_ecdsa_with_SHA1 &&
        EVP_MD_type((const EVP_MD *)p2) != NID_sha224 &&
        EVP_MD_type((const EVP_MD *)p2) != NID_sha256 &&
        EVP_MD_type((const EVP_MD *)p2) != NID_sha384 &&
        EVP_MD_type((const EVP_MD *)p2) != NID_sha512) {
```

```
            ECerr(EC_F_PKEY_EC_CTRL, EC_R_INVALID_DIGEST_TYPE);
            return 0;
        }
```

As we see there is no support for ecdsa with sha3 in openssl-1.1.1a...

So generate the certificate:

```
openssl req -config openssl.cnf -key ./private/RootCa1PrivateKey.pem -new -x509 -days 3660 \
 -sha512 -extensions v3_ca -out certs/RootCa1Cert.pem
```

Note: it may be necessary to add the file index.txt.attr next to openssl.cnf, the file should contain a single line

```
unique\_subject = yes
```

1.1.3 Verify the root certificate

The appropriate command is

```
openssl x509 -noout -text -in certs/RootCa1Cert.pem
```

The output is:

```
Certificate:
    Data:
        Version: 3 (0x2)
        Serial Number:
            2f:f1:8b:b2:19:76:06:4a:cb:41:d7:ac:82:68:29:60:3f:f0:b1:75
        Signature Algorithm: ecdsa-with-SHA512
        Issuer: C = PL, ST = Lower Silesian Voivodeship, L = Wroclaw, O = Wroclaw University of Science and Technology,
        OU = WPPT/K2, CN = System Security I - RootCA1, emailAddress = przemyslaw.kubiak@pwr.edu.pl
        Validity
            Not Before: Feb 22 18:53:35 2019 GMT
            Not After : Mar  1 18:53:35 2029 GMT
        Subject: C = PL, ST = Lower Silesian Voivodeship, L = Wroclaw, O = Wroclaw University of Science and Technology,
        OU = WPPT/K2, CN = System Security I - RootCA1, emailAddress = przemyslaw.kubiak@pwr.edu.pl
        Subject Public Key Info:
            Public Key Algorithm: id-ecPublicKey
                Public-Key: (512 bit)
                pub:
                    04:6b:17:9d:e5:22:e3:d0:aa:ee:a3:01:4e:1d:77:
                    76:ac:9f:c9:cd:02:a9:ed:0b:75:68:42:79:a3:14:
                    b3:ba:78:ab:d5:f2:21:2c:f7:a6:8f:82:b2:dd:92:
                    7e:a5:37:6c:ff:c7:86:72:61:b9:3c:79:9a:24:c3:
                    02:74:ad:33:23:a6:93:7c:fb:ed:02:87:42:d3:ee:
                    a2:bd:c3:7f:a9:54:af:77:6e:e8:d0:dc:0a:3f:95:
                    9a:00:0d:c9:26:18:6c:ca:9e:1e:c2:7e:c3:95:60:
                    80:d0:d4:ad:1e:1e:09:ca:40:eb:3f:fa:2e:b6:80:
                    4b:e2:08:bb:f3:e7:ac:68:22
                ASN1 OID: brainpoolP512r1
        X509v3 extensions:
            X509v3 Subject Key Identifier:
                34:E5:64:87:E8:B7:74:DC:57:4A:07:AE:3D:69:CB:D4:62:19:FA:36
            X509v3 Authority Key Identifier:
                keyid:34:E5:64:87:E8:B7:74:DC:57:4A:07:AE:3D:69:CB:D4:62:19:FA:36

            X509v3 Basic Constraints: critical
                CA:TRUE
    Signature Algorithm: ecdsa-with-SHA512
         30:81:86:02:41:00:82:aa:15:52:0b:60:d5:a5:c4:4d:a0:83:
         fe:7a:ec:bb:c7:d7:5a:bc:29:16:6d:e7:08:30:60:fb:87:76:
         20:f1:dc:55:32:26:c1:2c:65:03:d5:86:c2:66:be:45:7f:39:
         9e:37:fd:b3:7e:86:f2:40:d3:4f:c2:64:16:14:6b:db:02:41:
         00:aa:6d:2f:ba:eb:26:67:e8:75:8d:e6:16:3e:19:7b:ca:72:
         21:76:07:82:f0:00:fa:98:14:9d:dc:95:8b:f4:e1:f2:29:e2:
         09:b3:43:61:91:d4:19:0b:dd:9c:50:8d:16:70:d0:fc:f8:57:
         ec:14:06:55:f0:ba:e8:de:f9:b2:d7
```

1.2 Intermediate IntCA1.1

Changes to the openssl.cnf - according to the online tutorial.

```
cd intermediate1
openssl ecparam -name brainpoolP384r1 -genkey -noout -out ./private/IntermediateCa1_1_PrivateKey.pem
chmod 400 ./private/IntermediateCa1_1_PrivateKey.pem
openssl ec -in ./private/IntermediateCa1_1_PrivateKey.pem -pubout -out ./public/IntermediateCa1_1_PublicKey.pem
chmod 400 ./public/IntermediateCa1_1_PublicKey.pem
openssl ec -inform PEM -pubin -in ./public/IntermediateCa1_1_PublicKey.pem -text -noout
```

```
read EC key
Public-Key: (384 bit)
pub:
    04:49:e5:8d:b6:18:8e:5a:4d:71:14:b0:dc:cd:7c:
    4e:e2:06:a4:bb:19:4e:3f:7d:ba:c1:fa:2a:67:73:
    ff:38:6c:1b:94:12:6a:2a:ee:fe:2a:e2:62:45:12:
    c3:f3:07:36:09:70:4b:b3:1b:73:7e:41:03:5d:e9:
    1c:14:7f:54:72:54:92:e7:4b:a4:3e:b4:d8:f4:97:
    f9:ae:12:77:19:49:87:92:20:9d:54:79:f7:4a:31:
    42:fd:25:97:38:d7:48
ASN1 OID: brainpoolP384r1


cd ..
openssl req -config intermediate1/openssl.cnf -new -sha384 \
     -key intermediate1/private/IntermediateCa1_1_PrivateKey.pem -out intermediate1/csr/IntermediateCa1_1_Csr.pem
```

Note that intermediate1/openssl.cnf belongs to IntCa1_1 entity, and there is no need to call it from RootCa1's directory. The line above could be replaced with the command executed from intermediate1 subdirectory:

```
openssl req -config openssl.cnf -new -sha384 \
     -key private/IntermediateCa1_1_PrivateKey.pem -out csr/IntermediateCa1_1_Csr.pem
```

Anyway assume that we are in RootCa1's directory. Execute:

```
openssl req -text -noout -verify -in intermediate1/csr/IntermediateCa1_1_Csr.pem

verify OK
Certificate Request:
    Data:
        Version: 1 (0x0)
        Subject: C = PL, ST = Lower Silesian Voivodeship, L = Wroclaw, O = Wroclaw University of Science and Technology,
        OU = WPPT/K2, CN = System Security I - IntermediateCA1.1, emailAddress = przemyslaw.kubiak@pwr.edu.pl
        Subject Public Key Info:
            Public Key Algorithm: id-ecPublicKey
                Public-Key: (384 bit)
                pub:
                    04:49:e5:8d:b6:18:8e:5a:4d:71:14:b0:dc:cd:7c:
                    4e:e2:06:a4:bb:19:4e:3f:7d:ba:c1:fa:2a:67:73:
                    ff:38:6c:1b:94:12:6a:2a:ee:fe:2a:e2:62:45:12:
                    c3:f3:07:36:09:70:4b:b3:1b:73:7e:41:03:5d:e9:
                    1c:14:7f:54:72:54:92:e7:4b:a4:3e:b4:d8:f4:97:
                    f9:ae:12:77:19:49:87:92:20:9d:54:79:f7:4a:31:
                    42:fd:25:97:38:d7:48
                ASN1 OID: brainpoolP384r1
        Attributes:
            a0:00
    Signature Algorithm: ecdsa-with-SHA384
         30:64:02:30:6e:e8:e7:1c:e2:19:21:bd:44:41:e4:47:ce:a4:
         d8:03:dd:c1:ad:a8:d5:ed:4c:72:96:ee:a9:b0:4a:9d:a9:12:
         5a:bb:da:07:c7:d2:6b:79:08:2f:d1:7f:fa:44:61:2f:02:30:
         34:98:63:32:7b:13:35:21:6c:ae:7e:54:ff:0e:f0:e0:bb:37:
         c2:20:c1:7d:33:da:ff:e1:34:f7:04:eb:ee:52:32:08:90:91:
         55:ad:07:be:1c:82:d7:fb:2d:d3:fa:9a
```

The ./intermediate1/csr/IntermediateCa1_1_Csr.pem file decoded on https://lapo.it/asn1js/

```
SEQUENCE (3 elem)
  SEQUENCE (4 elem)
    INTEGER 0
    SEQUENCE (7 elem)
      SET (1 elem)
        SEQUENCE (2 elem)
          OBJECT IDENTIFIER 2.5.4.6 countryName (X.520 DN component)
          PrintableString PL
      SET (1 elem)
        SEQUENCE (2 elem)
          OBJECT IDENTIFIER 2.5.4.8 stateOrProvinceName (X.520 DN component)
          UTF8String Lower Silesian Voivodeship
      SET (1 elem)
        SEQUENCE (2 elem)
          OBJECT IDENTIFIER 2.5.4.7 localityName (X.520 DN component)
          UTF8String Wroclaw
      SET (1 elem)
        SEQUENCE (2 elem)
          OBJECT IDENTIFIER 2.5.4.10 organizationName (X.520 DN component)
          UTF8String Wroclaw University of Science and Technology
      SET (1 elem)
        SEQUENCE (2 elem)
          OBJECT IDENTIFIER 2.5.4.11 organizationalUnitName (X.520 DN component)
          UTF8String WPPT/K2
      SET (1 elem)
        SEQUENCE (2 elem)
          OBJECT IDENTIFIER 2.5.4.3 commonName (X.520 DN component)
          UTF8String System Security I - IntermediateCA1.1
      SET (1 elem)
```

```
      SEQUENCE (2 elem)
        OBJECT IDENTIFIER 1.2.840.113549.1.9.1 emailAddress (PKCS #9. Deprecated, use an altName extension instead)
        IA5String przemyslaw.kubiak@pwr.edu.pl
    SEQUENCE (2 elem)
      SEQUENCE (2 elem)
        OBJECT IDENTIFIER 1.2.840.10045.2.1 ecPublicKey (ANSI X9.62 public key type)
        OBJECT IDENTIFIER 1.3.36.3.3.2.8.1.1.11 brainpoolP384r1 (ECC Brainpool Standard Curves and Curve Generation)
      BIT STRING (776 bit) 000001000100100111100101100011011011011000011000100011100101101001001...
    [0] (0 elem)
  SEQUENCE (1 elem)
    OBJECT IDENTIFIER 1.2.840.10045.4.3.3 ecdsaWithSHA384 (ANSI X9.62 ECDSA algorithm with SHA384)
  BIT STRING (1 elem)
    SEQUENCE (2 elem)
      INTEGER (383 bit) 17070576956900257073924755075725791783220623160719522589187058396222777...
      INTEGER (382 bit) 80951519664494481980154159821085220571915166262968725013780517401062 94...
```

The tailing

```
...
  BIT STRING (1 elem)
    SEQUENCE (2 elem)
      INTEGER (383 bit) 17070576956900257073924755075725791783220623160719522589187058396222777...
      INTEGER (382 bit) 80951519664494481980154159821085220571915166262968725013780517401062 94...
```

in hexadecimal representation equals:

```
03 67 00 30 64 02  30 6E E8 E7 1C E2 19 21
BD 44 41 E4 47 CE A4 D8  03 DD C1 AD A8 D5 ED 4C
72 96 EE A9 B0 4A 9D A9  12 5A BB DA 07 C7 D2 6B
79 08 2F D1 7F FA 44 61  2F 02 30 34 98 63 32 7B
13 35 21 6C AE 7E 54 FF  0E F0 E0 BB 37 C2 20 C1
7D 33 DA FF E1 34 F7 04  EB EE 52 32 08 90 91 55
AD 07 BE 1C 82 D7 FB 2D  D3 FA 9A
```

Exercise: Read the ASN.1 encoded value above (printed in hexadecimals).

```
03 67                                           - BIT STRING, length: 6*16+7=103 octets,
 00                                             - the last octet has 0 padding bits
 30 64                                          - SEQUENCE, length 64h = 100 octets
    02  30 6E E8 E7 1C E2 19 21                 - INTEGER of length 16*3 = 48 octets = 384 bits (r)
    BD 44 41 E4 47 CE A4 D8  03 DD C1 AD A8 D5 ED 4C
    72 96 EE A9 B0 4A 9D A9  12 5A BB DA 07 C7 D2 6B
    79 08 2F D1 7F FA 44 61  2F

    02 30 34 98 63 32 7B                        - INTEGER of length 16*3 = 48 octets = 384 bits (s)
    13 35 21 6C AE 7E 54 FF  0E F0 E0 BB 37 C2 20 C1
    7D 33 DA FF E1 34 F7 04  EB EE 52 32 08 90 91 55
    AD 07 BE 1C 82 D7 FB 2D  D3 FA 9A
```

Note that in X.509 formats, SET and SEQUENCE types are used in constructed form (that is 6th bit is set to 1). By setting 1 in 6th bit for SEQUENCE universal tag (0x10) you will get 0x30.
Exercise: How object identifiers are encoded? Consider e.g., 1.3.36.3.3.2.8.1.1.11 (brainpoolP384r1)
A very nice description of OIDs encoding is given at https://www.sysadmins.lv/blog-en/how-to-encode-object-identifier-to-an-asn1-der-encoded-string.aspx Hexadecimal representation of encoding of 1.3.36.3.3.2.8.1.1.11 equals 06 09 2B 24 03 03 02 08 01 01 0B, which means:

```
06        - tag OBJECT IDENTIFIER
   09     - length 9 octets
      2B  - the first two nodes of all OIDs are represented by a single octet (the first node * 40 + the second node),
                        here: 1*40+3 = 43 = 2*16+11 = 2Bh
         24 03 03 02 08 01 01 0B  - the remaining nodes are represented by separate octets
```

Exercise: Encode the object identifier 1.2.840.10045.2.1 (ecPublicKey).
Answer: 06 07 2A 86 48 CE 3D 02 01
The encoding of leading nodes 1.2 : 1*40 + 2 = 42 = 0x2A
node 840: 840/128 = 6,56.. hence the transitional byte value is 0x86. The next byte represents the difference 840-6*128 = 72 = 64+8 = 0x48
Node 10045: 10045/2048=4,904.. so the major transitional byte value is 0x8+0x4 = 0xC, 10045 - 4*2048= 1853, 1853/128 = 14,476.. so the minor transactional byte value is 14 = 0xE, 1853 - 14*128 = 61 = 0x3D, Finally node 10045 is encoded as 0xCE3D.
Nodes 2.1 are encoded as 0x0201
Exercise: Justify that the decoding is unambiguous.

Exercise: What if the octet value of a node is greater than 16383 (0xFF 0x7F)? Encode node value 230000.
Generate the certificate for IntCA1.1[1]:

```
openssl ca -config openssl.cnf -extensions v3_intermediate_ca  -notext -md sha384 -days 3648 \
          -in intermediate1/csr/IntermediateCa1_1_Csr.pem  -out intermediate1/certs/IntermediateCa1_1_Cert.pem
openssl x509 -noout -text -in intermediate1/certs/IntermediateCa1_1_Cert.pem

Certificate:
   Data:
       Version: 3 (0x2)
       Serial Number: 4660 (0x1234)
       Signature Algorithm: ecdsa-with-SHA384
       Issuer: C = PL, ST = Lower Silesian Voivodeship, L = Wroclaw, O = Wroclaw University of Science and Technology,
       OU = WPPT/K2, CN = System Security I - RootCA1, emailAddress = przemyslaw.kubiak@pwr.edu.pl
       Validity
           Not Before: Feb 23 23:55:31 2019 GMT
           Not After : Feb 18 23:55:31 2029 GMT
       Subject: C = PL, ST = Lower Silesian Voivodeship, O = Wroclaw University of Science and Technology,
       OU = WPPT/K2, CN = System Security I - IntermediateCA1.1, emailAddress = przemyslaw.kubiak@pwr.edu.pl
       Subject Public Key Info:
           Public Key Algorithm: id-ecPublicKey
               Public-Key: (384 bit)
               pub:
                   04:49:e5:8d:b6:18:8e:5a:4d:71:14:b0:dc:cd:7c:
                   4e:e2:06:a4:bb:19:4e:3f:7d:ba:c1:fa:2a:67:73:
                   ff:38:6c:1b:94:12:6a:2a:ee:fe:2a:e2:62:45:12:
                   c3:f3:07:36:09:70:4b:b3:1b:73:7e:41:03:5d:e9:
                   1c:14:7f:54:72:54:92:e7:4b:a4:3e:b4:d8:f4:97:
                   f9:ae:12:77:19:49:87:92:20:9d:54:79:f7:4a:31:
                   42:fd:25:97:38:d7:48
               ASN1 OID: brainpoolP384r1
       X509v3 extensions:
           X509v3 Subject Key Identifier:
               FF:8B:C3:8D:62:57:23:8D:9C:73:8A:0F:4D:DB:5B:A6:DB:4A:37:B0
           X509v3 Authority Key Identifier:
               keyid:34:E5:64:87:E8:B7:74:DC:57:4A:07:AE:3D:69:CB:D4:62:19:FA:36

           X509v3 Basic Constraints: critical
               CA:TRUE, pathlen:0
           X509v3 Key Usage: critical
               Digital Signature, Certificate Sign, CRL Sign
   Signature Algorithm: ecdsa-with-SHA384
        30:81:85:02:41:00:90:65:ae:65:f4:15:b4:14:67:ec:46:eb:
        3b:2a:e6:64:9f:e6:f9:5e:61:38:72:0b:16:5b:e0:dd:b2:49:
        37:54:8f:f7:b3:84:8d:91:91:a5:e3:59:31:bf:9a:fd:63:a6:
        1d:be:71:1a:aa:3d:d3:f6:3b:ee:de:bb:34:c7:9f:0b:02:40:
        72:31:87:c1:89:dd:70:cc:3c:91:5f:5d:87:09:27:ee:c9:17:
        46:09:df:32:6a:a6:88:de:a9:55:bd:d0:51:28:8a:6e:e8:f7:
        e3:32:6f:77:ae:4d:d7:c5:de:9a:79:d6:d3:72:1e:8c:8c:3f:
        4d:b9:20:d0:bb:97:54:3b:8c:4b
```

Exercise: compare issuer and subject with the previous certificate.
The certificate file decoded on https://lapo.it/asn1js/ Exercise: look at RFC 5280 and try to map certificate components between theese two decodings.

```
SEQUENCE (3 elem)
  SEQUENCE (8 elem)
    [0] (1 elem)
      INTEGER 2
    INTEGER 4660
    SEQUENCE (1 elem)
      OBJECT IDENTIFIER 1.2.840.10045.4.3.3 ecdsaWithSHA384 (ANSI X9.62 ECDSA algorithm with SHA384)
    SEQUENCE (7 elem)
      SET (1 elem)
        SEQUENCE (2 elem)
          OBJECT IDENTIFIER 2.5.4.6 countryName (X.520 DN component)
          PrintableString PL
      SET (1 elem)
        SEQUENCE (2 elem)
          OBJECT IDENTIFIER 2.5.4.8 stateOrProvinceName (X.520 DN component)
          UTF8String Lower Silesian Voivodeship
      SET (1 elem)
        SEQUENCE (2 elem)
          OBJECT IDENTIFIER 2.5.4.7 localityName (X.520 DN component)
          UTF8String Wroclaw
      SET (1 elem)
        SEQUENCE (2 elem)
          OBJECT IDENTIFIER 2.5.4.10 organizationName (X.520 DN component)
          UTF8String Wroclaw University of Science and Technology
      SET (1 elem)
        SEQUENCE (2 elem)
          OBJECT IDENTIFIER 2.5.4.11 organizationalUnitName (X.520 DN component)
          UTF8String WPPT/K2
      SET (1 elem)
```

[1] To generate the certificate for IntCA1.1 sha384 was used as a message diggest function. Why this choice made by the RootCA1 is not the best one?

```
        SEQUENCE (2 elem)
          OBJECT IDENTIFIER 2.5.4.3 commonName (X.520 DN component)
          UTF8String System Security I - RootCA1
      SET (1 elem)
        SEQUENCE (2 elem)
          OBJECT IDENTIFIER 1.2.840.113549.1.9.1 emailAddress (PKCS #9. Deprecated, use an altName extension instead)
          IA5String przemyslaw.kubiak@pwr.edu.pl
    SEQUENCE (2 elem)
      UTCTime 2019-02-23 23:55:31 UTC
      UTCTime 2029-02-18 23:55:31 UTC
    SEQUENCE (6 elem)
      SET (1 elem)
        SEQUENCE (2 elem)
          OBJECT IDENTIFIER 2.5.4.6 countryName (X.520 DN component)
          PrintableString PL
      SET (1 elem)
        SEQUENCE (2 elem)
          OBJECT IDENTIFIER 2.5.4.8 stateOrProvinceName (X.520 DN component)
          UTF8String Lower Silesian Voivodeship
      SET (1 elem)
        SEQUENCE (2 elem)
          OBJECT IDENTIFIER 2.5.4.10 organizationName (X.520 DN component)
          UTF8String Wroclaw University of Science and Technology
      SET (1 elem)
        SEQUENCE (2 elem)
          OBJECT IDENTIFIER 2.5.4.11 organizationalUnitName (X.520 DN component)
          UTF8String WPPT/K2
      SET (1 elem)
        SEQUENCE (2 elem)
          OBJECT IDENTIFIER 2.5.4.3 commonName (X.520 DN component)
          UTF8String System Security I - IntermediateCA1.1
      SET (1 elem)
        SEQUENCE (2 elem)
          OBJECT IDENTIFIER 1.2.840.113549.1.9.1 emailAddress (PKCS #9. Deprecated, use an altName extension instead)
          IA5String przemyslaw.kubiak@pwr.edu.pl
    SEQUENCE (2 elem)
      SEQUENCE (2 elem)
        OBJECT IDENTIFIER 1.2.840.10045.2.1 ecPublicKey (ANSI X9.62 public key type)
        OBJECT IDENTIFIER 1.3.36.3.3.2.8.1.1.11 brainpoolP384r1 (ECC Brainpool Standard Curves and Curve Generation)
      BIT STRING (776 bit) 000001000100100111100101100011011011011000011000100011100101101001001...
    [3] (1 elem)
      SEQUENCE (4 elem)
        SEQUENCE (2 elem)
          OBJECT IDENTIFIER 2.5.29.14 subjectKeyIdentifier (X.509 extension)
          OCTET STRING (1 elem)
            OCTET STRING (20 byte) FF8BC38D6257238D9C738A0F4DDB5BA6DB4A37B0
        SEQUENCE (2 elem)
          OBJECT IDENTIFIER 2.5.29.35 authorityKeyIdentifier (X.509 extension)
          OCTET STRING (1 elem)
            SEQUENCE (1 elem)
              [0] (20 byte) 34E56487E8B774DC574A07AE3D69CBD46219FA36
        SEQUENCE (3 elem)
          OBJECT IDENTIFIER 2.5.29.19 basicConstraints (X.509 extension)
          BOOLEAN true
          OCTET STRING (1 elem)
            SEQUENCE (2 elem)
              BOOLEAN true
              INTEGER 0
        SEQUENCE (3 elem)
          OBJECT IDENTIFIER 2.5.29.15 keyUsage (X.509 extension)
          BOOLEAN true
          OCTET STRING (1 elem)
            BIT STRING (7 bit) 1000011
  SEQUENCE (1 elem)
    OBJECT IDENTIFIER 1.2.840.10045.4.3.3 ecdsaWithSHA384 (ANSI X9.62 ECDSA algorithm with SHA384)
  BIT STRING (1 elem)
    SEQUENCE (2 elem)
      INTEGER (512 bit) 756269461224553453227354114121561455769388084803612501933729594148426 1...
      INTEGER (511 bit) 598079771934778520973843274195968657190520982068202847764633204311838 8...
```

Verify the certificate:

```
openssl verify -CAfile certs/RootCa1Cert.pem intermediate1/certs/IntermediateCa1_1_Cert.pem

intermediate1/certs/IntermediateCa1_1_Cert.pem: OK
```

1.3 Revoking the IntCA1.1 certificate

But something went wrong. What? Why IntermediateCa1_1_Cert.pem is useless for us? Answer: we have used default settings for the intermediate CA, and consequently got:

```
X509v3 Basic Constraints: critical
    CA:TRUE, pathlen:0
```

From https://www.openssl.org/docs/man1.0.2/man5/x509v3_config.html we learn that:

The pathlen parameter indicates the maximum number of CAs that can appear below this one in a chain. So if you have a CA with a pathlen of zero it can only be used to sign end user certificates and not further CAs.
So what should we do?

- revoke the certificate of IntCA1.1 (on the side of RootCA1),
- modify the openssl.cfg of the RootCA1
- generate a new cert for IntCA1.1 by RootCA1.

Why revoking?
From https://stackoverflow.com/questions/9496698/how-to-revoke-an-openssl-certificate-when-you-dont-have-the-certificate/9517132 we learn: ”Alternatively you can also change the config file to contain the line

```
unique_subject = no
```

to allow multiple certificates with the same common name. If you have published the original certificate, revoking the old one is however the preferable solution, even if you don’t run an OSCP server or provide CRLs.”
So let us revoke the IntCA1.1 cert. We are going to use the instructions from https://jamielinux.com/docs/openssl-certificate-authority/certificate-revocation-lists.html

```
cd root1/ca/
touch crlnumber
echo 1000 > crlnumber
openssl ca -config ./openssl.cnf -gencrl -out crl/RootCA1.crl.pem
openssl crl -in crl/RootCA1.crl.pem -noout -text


Certificate Revocation List (CRL):
        Version 2 (0x1)
        Signature Algorithm: ecdsa-with-SHA512
        Issuer: C = PL, ST = Lower Silesian Voivodeship, L = Wroclaw,
        O = Wroclaw University of Science and Technology, OU = WPPT/K2, CN = System Security I - RootCA1,
        emailAddress = przemyslaw.kubiak@pwr.edu.pl
        Last Update: Apr 17 15:17:51 2019 GMT
        Next Update: Apr 14 15:17:51 2029 GMT
        CRL extensions:
            X509v3 Authority Key Identifier:
                keyid:34:E5:64:87:E8:B7:74:DC:57:4A:07:AE:3D:69:CB:D4:62:19:FA:36

            X509v3 CRL Number:
                4096
No Revoked Certificates.
    Signature Algorithm: ecdsa-with-SHA512
         30:81:85:02:41:00:85:17:4d:ec:03:05:6d:43:82:9e:d1:cb:
         ef:0f:5e:53:f6:d6:94:72:d5:1b:e2:a3:de:e0:fa:28:b7:0c:
         1f:cc:5a:33:39:af:b8:c9:34:0f:89:c6:43:28:8e:92:24:43:
         8b:a8:a5:3e:4f:07:2a:6a:03:4d:c0:0b:dd:ca:4d:e1:02:40:
         52:e7:57:02:bd:65:b8:f5:07:64:34:0e:10:3e:c1:7f:da:d6:
         8f:db:5f:cb:84:f3:8a:59:fc:e0:57:52:36:72:c2:36:3d:05:
         65:b7:d3:f5:19:66:d5:03:b7:6c:0e:3b:c5:93:02:8f:f0:75:
         ae:90:18:15:c2:a7:7e:dc:43:8b
```

CRL Number $4096 = 4 * 2^{10} = 2^{12} = 0x1000$ so the file crlnumber contains the hexadecimal value of the number.
Now we must revoke the certificate of IntCA1.1:

```
openssl ca -config ./openssl.cnf  -revoke ./intermediate1/certs/IntermediateCa1_1_Cert.pem
```

To check the result we display the content of index.txt file:

```
cat index.txt

R       290218235531Z   190511152652Z   1234    unknown /C=PL/ST=Lower Silesian Voivodeship/
O=Wroclaw University of Science and Technology/OU=WPPT/K2/CN=System Security I - IntermediateCA1.1
/emailAddress=przemyslaw.kubiak@pwr.edu.pl
```

Letter R at the beginning of the line means that the certificate has been revoked. Let us check if CRL is automatically updated by openssl. Command

```
openssl crl -in crl/RootCA1.crl.pem -noout -text
```

shows CRL with ”No Revoked Certificates”, so update the CRL manually:

```
openssl ca -config ./openssl.cnf -gencrl -out crl/RootCA1.crl.pem
```

And the command

```
openssl crl -in crl/RootCA1.crl.pem -noout -text
```

shows the updated CRL:

```
Certificate Revocation List (CRL):
        Version 2 (0x1)
        Signature Algorithm: ecdsa-with-SHA512
        Issuer: C = PL, ST = Lower Silesian Voivodeship, L = Wroclaw, O = Wroclaw University of Science and Technology,
        OU = WPPT/K2, CN = System Security I - RootCA1, emailAddress = przemyslaw.kubiak@pwr.edu.pl
        Last Update: May 11 17:33:08 2019 GMT
        Next Update: May  8 17:33:08 2029 GMT
        CRL extensions:
            X509v3 Authority Key Identifier:
                keyid:34:E5:64:87:E8:B7:74:DC:57:4A:07:AE:3D:69:CB:D4:62:19:FA:36

            X509v3 CRL Number:
                4097
Revoked Certificates:
    Serial Number: 1234
        Revocation Date: May 11 15:26:52 2019 GMT
    Signature Algorithm: ecdsa-with-SHA512
         30:81:85:02:41:00:88:3e:9c:26:4f:2b:81:8c:b1:3d:d6:d9:
         62:06:72:e5:70:99:ad:3d:53:2b:4b:4f:91:6b:40:4b:8d:ae:
         c5:73:3f:6d:86:fd:52:51:d7:27:6f:36:2d:5f:27:c5:33:b2:
         4b:62:67:a5:52:f3:df:90:23:e1:d0:d8:cc:2d:9d:40:02:40:
         1c:f3:cf:f8:14:8b:23:3c:63:1f:4c:51:6c:fe:3b:fd:23:75:
         34:d1:25:77:56:c3:53:0c:01:9c:0b:d4:a1:a0:86:85:d1:ec:
         fb:9d:0f:56:ec:3e:f0:50:2d:1e:89:55:0c:63:79:09:bb:89:
         f9:bd:47:7d:16:9a:7f:a0:cb:98
```

See that value of the field ”X509v3 CRL Number” has been incremented. Note that in the ./crl subdirectory still only one CRL file is maintained by openssl, that is the file RootCA1.crl.pem (so we have literally an update, not generation of a second file with the fresh crl). Now its time to modify the file openssl.cfg of the RootCA1 - set pathlen:2 in the settings for the intermediate CA (certificate of the subordinate CA - i.e., IntCA1.1 - will contain that value, so below IntCA1.1 two layers of CAs will be allowed).

1.4 The second certificate of IntCA1.1

In the CRL above only serial number of the revoked certificate is included (hexadecimal 1234), with no information about the public key included in the revoked certificate. Hence let us try to utilize the same certificate request to obtain a new certificate for IntCA1.1. This may look suspicious (usually a certificate is revoked to prevent usage of the corresponding private key), but we do this for the sake of simplicity. However, in a real case a new keypair should be generated on the side of IntCA1.1, and the certificate should be requested for the new public key.

```
openssl ca -config openssl.cnf -extensions v3_intermediate_ca  -notext -md sha384 -days 3600 \
-in intermediate1/csr/IntermediateCa1_1_Csr.pem  -out intermediate1/certs/IntermediateCa1_1_Cert2.pem
```

The command

```
openssl x509 -noout -text -in intermediate1/certs/IntermediateCa1_1_Cert2.pem
```

```
Certificate:
    Data:
        Version: 3 (0x2)
        Serial Number: 4661 (0x1235)
        Signature Algorithm: ecdsa-with-SHA384
        Issuer: C = PL, ST = Lower Silesian Voivodeship, L = Wroclaw, O = Wroclaw University of Science and Technology,
        OU = WPPT/K2, CN = System Security I - RootCA1, emailAddress = przemyslaw.kubiak@pwr.edu.pl
        Validity
            Not Before: May 11 18:42:58 2019 GMT
            Not After : Mar 19 18:42:58 2029 GMT
        Subject: C = PL, ST = Lower Silesian Voivodeship, O = Wroclaw University of Science and Technology,
        OU = WPPT/K2, CN = System Security I - IntermediateCA1.1, emailAddress = przemyslaw.kubiak@pwr.edu.pl
        Subject Public Key Info:
            Public Key Algorithm: id-ecPublicKey
                Public-Key: (384 bit)
                pub:
                    04:49:e5:8d:b6:18:8e:5a:4d:71:14:b0:dc:cd:7c:
```

```
                    4e:e2:06:a4:bb:19:4e:3f:7d:ba:c1:fa:2a:67:73:
                    ff:38:6c:1b:94:12:6a:2a:ee:fe:2a:e2:62:45:12:
                    c3:f3:07:36:09:70:4b:b3:1b:73:7e:41:03:5d:e9:
                    1c:14:7f:54:72:54:92:e7:4b:a4:3e:b4:d8:f4:97:
                    f9:ae:12:77:19:49:87:92:20:9d:54:79:f7:4a:31:
                    42:fd:25:97:38:d7:48
                ASN1 OID: brainpoolP384r1
            X509v3 extensions:
                X509v3 Subject Key Identifier:
                    FF:8B:C3:8D:62:57:23:8D:9C:73:8A:0F:4D:DB:5B:A6:DB:4A:37:B0
                X509v3 Authority Key Identifier:
                    keyid:34:E5:64:87:E8:B7:74:DC:57:4A:07:AE:3D:69:CB:D4:62:19:FA:36

                X509v3 Basic Constraints: critical
                    CA:TRUE, pathlen:2
                X509v3 Key Usage: critical
                    Digital Signature, Certificate Sign, CRL Sign
    Signature Algorithm: ecdsa-with-SHA384
         30:81:84:02:40:5a:41:2b:4a:e9:b0:57:c1:40:ac:c6:2a:a6:
         d2:ad:00:2e:11:5e:a5:63:ee:d1:da:c9:14:9d:85:a7:ef:b5:
         b5:f9:62:27:9a:6c:79:d3:72:bb:fb:34:d9:48:ee:a2:56:a4:
         0e:e5:cd:98:d6:66:e3:b7:02:2f:00:83:b1:3b:f4:02:40:4a:
         7b:8a:90:fc:5f:d1:72:0a:00:e1:45:7c:7b:0f:9d:00:66:ee:
         e9:ab:f1:9e:86:1f:b0:a7:f7:ec:f2:15:5e:15:08:09:b5:34:
         f0:eb:63:df:ff:64:e0:97:7a:00:84:53:da:7c:15:fb:74:de:
         28:58:ff:92:74:0d:1d:30:f4
```

shows correct pathlen, and the serial number of the certificate equal to 0x1235. See that the new and the old certificate contain the same public key. Let continue with creation of the certificate hierarchy.

1.5 Intermediate IntCA1.2 certificate

```
cd intermediate1
mkdir intermediate2
```

Set pathlen:1 in section [v3_intermediate_ca] of the file intermediate1/openssl.cnf then build the structure of the files for IntCA1.2: follow the instructions for IntCA1.1 adjusting them to IntCA1.2 Changes to the ./intermediate1/intermediate2/openssl.cnf - similar to the ones from intermediate1/openssl.cnf

```
cd intermediate2
openssl ecparam -name brainpoolP384r1 -genkey -noout -out ./private/IntermediateCa1_2_PrivateKey.pem
chmod 400 ./private/IntermediateCa1_2_PrivateKey.pem
openssl ec -in ./private/IntermediateCa1_2_PrivateKey.pem -pubout -out ./public/IntermediateCa1_2_PublicKey.pem
chmod 400 ./public/IntermediateCa1_2_PublicKey.pem
openssl ec -inform PEM -pubin -in ./public/IntermediateCa1_2_PublicKey.pem -text -noout


read EC key
Public-Key: (384 bit)
pub:
    04:17:c0:ff:7c:1e:2b:81:6a:53:bd:de:1d:9b:f1:
    2a:96:ab:5d:d0:be:c6:b9:fc:9e:63:6f:ad:d8:8b:
    17:ed:f9:48:19:66:e2:6e:88:b9:e9:7c:3e:ab:04:
    98:45:85:cd:5b:65:ee:7c:01:c6:91:b4:4c:1c:8a:
    b8:9a:3d:ef:c8:20:ef:c4:fd:6b:81:57:b2:b8:7a:
    db:1f:9c:67:3f:f8:aa:31:97:52:d4:05:04:30:de:
    e2:89:3f:97:16:23:48
ASN1 OID: brainpoolP384r1

cd ..
openssl req -config intermediate2/openssl.cnf -new -sha384 -key intermediate2/private/IntermediateCa1_2_PrivateKey.pem \
 -out intermediate2/csr/IntermediateCa1_2_Csr.pem
openssl req -text -noout -verify -in intermediate2/csr/IntermediateCa1_2_Csr.pem

verify OK
Certificate Request:
    Data:
        Version: 1 (0x0)
        Subject: C = PL, ST = Lower Silesian Voivodeship, L = Wroclaw, O = Wroclaw University of Science and Technology,
        OU = WPPT/K2, CN = System Security I - IntermediateCA1.2, emailAddress = przemyslaw.kubiak@pwr.edu.pl
        Subject Public Key Info:
            Public Key Algorithm: id-ecPublicKey
                Public-Key: (384 bit)
                pub:
                    04:17:c0:ff:7c:1e:2b:81:6a:53:bd:de:1d:9b:f1:
                    2a:96:ab:5d:d0:be:c6:b9:fc:9e:63:6f:ad:d8:8b:
                    17:ed:f9:48:19:66:e2:6e:88:b9:e9:7c:3e:ab:04:
                    98:45:85:cd:5b:65:ee:7c:01:c6:91:b4:4c:1c:8a:
                    b8:9a:3d:ef:c8:20:ef:c4:fd:6b:81:57:b2:b8:7a:
                    db:1f:9c:67:3f:f8:aa:31:97:52:d4:05:04:30:de:
                    e2:89:3f:97:16:23:48
                ASN1 OID: brainpoolP384r1
        Attributes:
            a0:00
```

```
    Signature Algorithm: ecdsa-with-SHA384
         30:64:02:30:04:9d:b6:ea:e0:56:b3:51:ce:a0:af:68:4f:8a:
         13:76:7e:25:8c:23:60:0f:b9:d3:17:8f:fb:32:92:40:28:41:
         01:16:4f:75:fa:24:ff:06:ba:3a:6c:e3:b0:4f:69:a8:02:30:
         15:3f:17:cc:a0:9c:f7:4e:9c:b9:70:95:98:ce:70:e2:39:e3:
         e3:84:f8:ea:77:f2:c2:d0:fe:2d:69:89:7a:02:e7:34:64:ba:
         a8:8b:ac:92:50:7d:bf:1f:8c:8a:21:af
```

Generate the certificate for IntCA1.2:

```
openssl ca -config openssl.cnf -extensions v3_intermediate_ca  -notext -md sha384 -days 3600 \
 -in intermediate2/csr/IntermediateCa1_2_Csr.pem  -out intermediate2/certs/IntermediateCa1_2_Cert.pem
openssl x509 -noout -text -in intermediate2/certs/IntermediateCa1_2_Cert.pem
```

```
Certificate:
    Data:
        Version: 3 (0x2)
        Serial Number: 4096 (0x1000)
        Signature Algorithm: ecdsa-with-SHA384
        Issuer: C = PL, ST = Lower Silesian Voivodeship, O = Wroclaw University of Science and Technology,
        OU = WPPT/K2, CN = System Security I - IntermediateCA1.1, emailAddress = przemyslaw.kubiak@pwr.edu.pl
        Validity
            Not Before: May 11 21:37:57 2019 GMT
            Not After : Mar 19 21:37:57 2029 GMT
        Subject: C = PL, ST = Lower Silesian Voivodeship, L = Wroclaw, O = Wroclaw University of Science and Technology,
        OU = WPPT/K2, CN = System Security I - IntermediateCA1.2, emailAddress = przemyslaw.kubiak@pwr.edu.pl
        Subject Public Key Info:
            Public Key Algorithm: id-ecPublicKey
                Public-Key: (384 bit)
                pub:
                    04:17:c0:ff:7c:1e:2b:81:6a:53:bd:de:1d:9b:f1:
                    2a:96:ab:5d:d0:be:c6:b9:fc:9e:63:6f:ad:d8:8b:
                    17:ed:f9:48:19:66:e2:6e:88:b9:e9:7c:3e:ab:04:
                    98:45:85:cd:5b:65:ee:7c:01:c6:91:b4:4c:1c:8a:
                    b8:9a:3d:ef:c8:20:ef:c4:fd:6b:81:57:b2:b8:7a:
                    db:1f:9c:67:3f:f8:aa:31:97:52:d4:05:04:30:de:
                    e2:89:3f:97:16:23:48
                ASN1 OID: brainpoolP384r1
        X509v3 extensions:
            X509v3 Subject Key Identifier:
                74:B1:9E:C3:36:68:B3:66:80:77:31:71:09:CD:B2:46:50:44:8B:0C
            X509v3 Authority Key Identifier:
                keyid:FF:8B:C3:8D:62:57:23:8D:9C:73:8A:0F:4D:DB:5B:A6:DB:4A:37:B0

            X509v3 Basic Constraints: critical
                CA:TRUE, pathlen:1
            X509v3 Key Usage: critical
                Digital Signature, Certificate Sign, CRL Sign
    Signature Algorithm: ecdsa-with-SHA384
         30:65:02:30:19:4a:93:8c:d7:04:c2:20:3c:c0:5b:89:29:38:
         a7:28:16:54:6b:4b:f9:4d:fd:43:83:4c:1e:76:2a:bc:1c:1b:
         3a:93:3f:f0:41:91:2c:55:9d:b2:92:7f:98:4e:07:73:02:31:
         00:84:eb:69:e3:c4:11:b8:0c:e4:6c:e1:bd:ca:62:77:53:99:
         6b:e0:f4:51:46:9a:90:bf:a5:a9:9e:fb:66:33:16:b7:a4:7f:
         17:b3:e5:56:60:08:04:d9:d1:1e:8b:13:9e
```

Task: collect the Issuer, Subject, X509v3 Subject Key Identifier, X509v3 Authority Key Identifier from RootCA1, IntCA1.1, IntCA1.2 certificates.
RootCA1 (RootCa1Cert.pem):

- Issuer: C = PL, ST = Lower Silesian Voivodeship, L = Wroclaw, O = Wroclaw University of Science and Technology, OU = WPPT/K2, CN = System Security I - RootCA1, emailAddress = przemyslaw.kubiak@pwr.edu.pl
- Subject: C = PL, ST = Lower Silesian Voivodeship, L = Wroclaw, O = Wroclaw University of Science and Technology, OU = WPPT/K2, CN = System Security I - RootCA1, emailAddress = przemyslaw.kubiak@pwr.edu.pl
- X509v3 Subject Key Identifier: 34:E5:64:87:E8:B7:74:DC:57:4A:07:AE:3D:69:CB:D4:62:19:FA:36
- X509v3 Authority Key Identifier: keyid:34:E5:64:87:E8:B7:74:DC:57:4A:07:AE:3D:69:CB:D4:62:19:FA:36

IntCA1.1 (IntermediateCa1_1_Cert2.pem):

- Issuer: C = PL, ST = Lower Silesian Voivodeship, L = Wroclaw, O = Wroclaw University of Science and Technology, OU = WPPT/K2, CN = System Security I - RootCA1, emailAddress = przemyslaw.kubiak@pwr.edu.pl

- Subject: C = PL, ST = Lower Silesian Voivodeship, O = Wroclaw University of Science and Technology, OU = WPPT/K2, CN = System Security I - IntermediateCA1.1, emailAddress = przemyslaw.kubiak@pwr.edu.pl
- X509v3 Subject Key Identifier: FF:8B:C3:8D:62:57:23:8D:9C:73:8A:0F:4D:DB:5B:A6:DB:4A:37:B0
- X509v3 Authority Key Identifier: keyid:34:E5:64:87:E8:B7:74:DC:57:4A:07:AE:3D:69:CB:D4:62:19:FA:36

IntCA1.2 (IntermediateCa1_2_Cert.pem):

- Issuer: C = PL, ST = Lower Silesian Voivodeship, O = Wroclaw University of Science and Technology, OU = WPPT/K2, CN = System Security I - IntermediateCA1.1, emailAddress = przemyslaw.kubiak@pwr.edu.pl
- Subject: C = PL, ST = Lower Silesian Voivodeship, L = Wroclaw, O = Wroclaw University of Science and Technology, OU = WPPT/K2, CN = System Security I - IntermediateCA1.2, emailAddress = przemyslaw.kubiak@pwr.edu.pl
- X509v3 Subject Key Identifier: 74:B1:9E:C3:36:68:B3:66:80:77:31:71:09:CD:B2:46:50:44:8B:0C
- X509v3 Authority Key Identifier: keyid:FF:8B:C3:8D:62:57:23:8D:9C:73:8A:0F:4D:DB:5B:A6:DB:4A:37:B0

We see that the corresponding fields are the same, so the chain is being buil.

1.6 Intermediate IntCA1.3 – ED448 public key

Task: Check if openssl allows to generate keys on one of the curves: Ed25519, Ed448, x25519 (the curves are depicted in https://tools.ietf.org/html/rfc7748). If YES, generate IntCA1.3 certificate for one of the curves. Note that OIDs for those curves exist - see https://tools.ietf.org/id/draft-ietf-curdle-pkix-10.html
Let check ecparam

```
openssl ecparam -list_curves | grep 448
```

No entry..

```
openssl version

OpenSSL 1.1.1  11 Sep 2018
```

On https://www.openssl.org/docs/manmaster/man7/Ed448.html we find that with version 1.1.1 Ed25519, Ed448 are supported. But https://wiki.openssl.org/index.php/Command_Line_Elliptic_Curve_Operations explains that
”The only Elliptic Curve algorithms that OpenSSL currently supports are Elliptic Curve Diffie Hellman (ECDH) for key agreement and Elliptic Curve Digital Signature Algorithm (ECDSA) for signing/verifying. x25519, ed25519 and ed448 aren’t standard EC curves so you can’t use ecparams or ec subcommands to work with them. If you need to generate x25519 or ed25519 keys then see the genpkey subcommand. ”
On page https://www.openssl.org/docs/manmaster/man1/genpkey.html we find that ED448 private key is generated with the command

```
openssl genpkey -algorithm ED448 -out xkey.pem
```

additionaly we see that X448 curve can be used. Let generate the keypair of IntCA1.3 on the basis of 448 bit curve.

```
cd intermediate2
mkdir intermediate3
```

Set pathlen:0 in section [v3_intermediate_ca] of the file intermediate1/intermediate2/openssl.cnf then build the structure of the files for IntCA1.3: follow the instructions for IntCA1.2 adjusting them to IntCA1.3 Changes to the ./intermediate1/intermediate2/intermediate2/openssl.cnf - similar to the ones from intermediate1/intermediate2/openssl.cnf

```
cd intermediate3
openssl genpkey -algorithm X448 -out ./private/IntermediateCa1_3_PrivateKey.pem
chmod 400 ./private/IntermediateCa1_3_PrivateKey.pem
openssl ec -in ./private/IntermediateCa1_3_PrivateKey.pem -pubout -out ./public/IntermediateC
```

The last command produces an error

```
read EC key
unable to load Key
140212291310656:error:0608308E:digital envelope routines:EVP_PKEY_get0_EC_KEY:expecting a ec
```

So let try openssl pkey -in key.pem -pubout -out pubkey.pem

```
    openssl pkey -in ./private/IntermediateCa1_3_PrivateKey.pem -pubout -out ./public/IntermediateCa1_3_PublicKey.pem
```

And that works.

```
    chmod 400 ./public/IntermediateCa1_3_PublicKey.pem
    openssl ec -inform PEM -pubin -in ./public/IntermediateCa1_3_PublicKey.pem -text -noout
```

The last command produces similar error like above. Try

```
    openssl pkey -inform PEM -pubin -in ./public/IntermediateCa1_3_PublicKey.pem -text -noout

X448 Public-Key:
pub:
    1c:a4:20:df:0b:e3:61:cc:f7:68:ed:ac:17:c9:7e:
    ea:18:7e:5d:c9:7e:2d:68:49:78:88:73:19:8b:e3:
    6e:0b:82:13:9c:4f:42:af:aa:4a:34:04:ee:d3:09:
    be:95:3d:27:42:3a:4c:4e:3e:6b:12
```

The ./public/IntermediateCa1_3_PublicKey.pem file decoded on https://lapo.it/asn1js/

```
SEQUENCE (2 elem)
  SEQUENCE (1 elem)
    OBJECT IDENTIFIER 1.3.101.111 curveX448 (ECDH 448 key agreement algorithm)
  BIT STRING (448 bit) 000111001010010000100000110111110000101111100011011000011100110011111101...
```

For the command

```
    cd ..
    openssl req -config intermediate3/openssl.cnf -new
        -key intermediate3/private/IntermediateCa1_3_PrivateKey.pem -out intermediate3/csr/IntermediateCa1_3_Csr.pem
```

An error occured:

```
    139650743022656:error:0608D096:digital envelope routines:EVP_PKEY_sign_init:operation
      not supported for this keytype:../crypto/evp/pmeth_fn.c:40:
```

So let try with ED448 (we have some promising descryption https://www.openssl.org/docs/man1.1.1/man7/Ed448.html).

```
cd intermediate3
openssl genpkey -algorithm ED448 -out ./private/IntermediateCa1_3_PrivateKey2.pem
chmod 400 ./private/IntermediateCa1_3_PrivateKey2.pem
openssl pkey -in ./private/IntermediateCa1_3_PrivateKey2.pem -pubout -out ./public/IntermediateCa1_3_PublicKey2.pem
openssl pkey -inform PEM -pubin -in ./public/IntermediateCa1_3_PublicKey2.pem -text -noout

ED448 Public-Key:
pub:
    65:76:06:15:64:30:42:f9:2c:71:94:91:6a:81:0e:
    9b:0f:1e:9b:26:07:0c:39:d8:a9:fc:86:09:5f:d9:
    53:99:6e:51:08:74:6b:4f:30:81:5c:1e:bb:cd:04:
    64:4c:16:5c:66:31:e5:dc:73:ee:b5:80
```

The ./public/IntermediateCa1_3_PublicKey2.pem file decoded on https://lapo.it/asn1js/

```
SEQUENCE (2 elem)
  SEQUENCE (1 elem)
    OBJECT IDENTIFIER 1.3.101.113 curveEd448 (EdDSA 448 signature algorithm)
  BIT STRING (456 bit) 011001010111011000000110000101010110010000110000010000101111100100101 1...
```

Now execute

```
cd ..
openssl req -config intermediate3/openssl.cnf -new -key intermediate3/private/IntermediateCa1_3_PrivateKey2.pem -out intermediate3/csr/
```

And this ends with no error. So the source of the previous error seems to be the designation of X448 key: curveX448 (ECDH 448 key agreement algorithm)

```
openssl ca -config openssl.cnf -extensions v3_intermediate_ca  -notext -md sha384 -days 3580
        -in intermediate3/csr/IntermediateCa1_3_Csr2.pem  -out intermediate3/certs/IntermediateCa1_3_Cert.pem
```

Note that -md sha384 is the option for IntCa1_3 key.

```
openssl x509 -noout -text -in intermediate3/certs/IntermediateCa1_3_Cert.pem
```

```
Certificate:
    Data:
        Version: 3 (0x2)
        Serial Number: 4369 (0x1111)
        Signature Algorithm: ecdsa-with-SHA384
        Issuer: C = PL, ST = Lower Silesian Voivodeship, L = Wroclaw, O = Wroclaw University of Science and Technology,
        OU = WPPT/K2, CN = System Security I - IntermediateCA1.2, emailAddress = przemyslaw.kubiak@pwr.edu.pl
        Validity
            Not Before: May 12 12:35:40 2019 GMT
            Not After : Feb 28 12:35:40 2029 GMT
        Subject: C = PL, ST = Lower Silesian Voivodeship, L = Wroclaw, O = Wroclaw University of Science and Technology,
        OU = WPPT/K2, CN = System Security I - IntermediateCA1.3, emailAddress = przemyslaw.kubiak@pwr.edu.pl
        Subject Public Key Info:
            Public Key Algorithm: ED448
                ED448 Public-Key:
                pub:
                    65:76:06:15:64:30:42:f9:2c:71:94:91:6a:81:0e:
                    9b:0f:1e:9b:26:07:0c:39:d8:a9:fc:86:09:5f:d9:
                    53:99:6e:51:08:74:6b:4f:30:81:5c:1e:bb:cd:04:
                    64:4c:16:5c:66:31:e5:dc:73:ee:b5:80
        X509v3 extensions:
            X509v3 Subject Key Identifier:
                8F:D4:99:A9:38:CB:00:17:F1:F9:07:59:45:36:4A:3D:6D:54:79:B0
            X509v3 Authority Key Identifier:
                keyid:74:B1:9E:C3:36:68:B3:66:80:77:31:71:09:CD:B2:46:50:44:8B:0C

            X509v3 Basic Constraints: critical
                CA:TRUE, pathlen:0
            X509v3 Key Usage: critical
                Digital Signature, Certificate Sign, CRL Sign
    Signature Algorithm: ecdsa-with-SHA384
         30:64:02:30:06:aa:6a:e2:d6:c5:5b:1f:0c:93:c3:32:71:71:
         43:93:9e:db:4e:ec:7d:f2:bd:d6:cf:fb:1b:00:9b:4b:62:00:
         41:03:ec:48:21:08:2a:74:a2:67:48:18:76:00:39:0e:02:30:
         0e:9d:dd:1b:ba:f0:31:63:e3:1c:af:3b:e3:ad:df:96:e0:63:
         1c:b3:86:00:0b:8f:20:96:d8:98:48:6c:e4:61:d7:48:89:75:
         0c:d1:19:bc:af:d4:83:10:9d:69:a8:25
```

Task: Update the table containing Issuer, Subject, X509v3 Subject Key Identifier, X509v3 Authority Key Identifier for RootCA1, IntCA1.1, IntCA1.2 certificates, by the IntCA1.3 data.
RootCA1 (RootCa1Cert.pem):

- Issuer: C = PL, ST = Lower Silesian Voivodeship, L = Wroclaw, O = Wroclaw University of Science and Technology, OU = WPPT/K2, CN = System Security I - RootCA1, emailAddress = przemyslaw.kubiak@pwr.edu.pl
- Subject: C = PL, ST = Lower Silesian Voivodeship, L = Wroclaw, O = Wroclaw University of Science and Technology, OU = WPPT/K2, CN = System Security I - RootCA1, emailAddress = przemyslaw.kubiak@pwr.edu.pl
- X509v3 Subject Key Identifier: 34:E5:64:87:E8:B7:74:DC:57:4A:07:AE:3D:69:CB:D4:62:19:FA:36
- X509v3 Authority Key Identifier: keyid:34:E5:64:87:E8:B7:74:DC:57:4A:07:AE:3D:69:CB:D4:62:19:FA:36

IntCA1.1 (IntermediateCa1_1_Cert2.pem):

- Issuer: C = PL, ST = Lower Silesian Voivodeship, L = Wroclaw, O = Wroclaw University of Science and Technology, OU = WPPT/K2, CN = System Security I - RootCA1, emailAddress = przemyslaw.kubiak@pwr.edu.pl
- Subject: C = PL, ST = Lower Silesian Voivodeship, O = Wroclaw University of Science and Technology, OU = WPPT/K2, CN = System Security I - IntermediateCA1.1, emailAddress = przemyslaw.kubiak@pwr.edu.pl

- X509v3 Subject Key Identifier: FF:8B:C3:8D:62:57:23:8D:9C:73:8A:0F:4D:DB:5B:A6:DB:4A:37:B0
- X509v3 Authority Key Identifier: keyid:34:E5:64:87:E8:B7:74:DC:57:4A:07:AE:3D:69:CB:D4:62:19:FA:36

IntCA1.2 (IntermediateCa1_2_Cert.pem):

- Issuer: C = PL, ST = Lower Silesian Voivodeship, O = Wroclaw University of Science and Technology, OU = WPPT/K2, CN = System Security I - IntermediateCA1.1, emailAddress = przemyslaw.kubiak@pwr.edu.pl
- Subject: C = PL, ST = Lower Silesian Voivodeship, L = Wroclaw, O = Wroclaw University of Science and Technology, OU = WPPT/K2, CN = System Security I - IntermediateCA1.2, emailAddress = przemyslaw.kubiak@pwr.edu.pl
- X509v3 Subject Key Identifier: 74:B1:9E:C3:36:68:B3:66:80:77:31:71:09:CD:B2:46:50:44:8B:0C
- X509v3 Authority Key Identifier: keyid:FF:8B:C3:8D:62:57:23:8D:9C:73:8A:0F:4D:DB:5B:A6:DB:4A:37:B0

IntCA1.3 (IntermediateCa1_3_Cert.pem):

- Issuer: C = PL, ST = Lower Silesian Voivodeship, L = Wroclaw, O = Wroclaw University of Science and Technology, OU = WPPT/K2, CN = System Security I - IntermediateCA1.2, emailAddress = przemyslaw.kubiak@pwr.edu.pl
- Subject: C = PL, ST = Lower Silesian Voivodeship, L = Wroclaw, O = Wroclaw University of Science and Technology, OU = WPPT/K2, CN = System Security I - IntermediateCA1.3, emailAddress = przemyslaw.kubiak@pwr.edu.pl
- X509v3 Subject Key Identifier: 8F:D4:99:A9:38:CB:00:17:F1:F9:07:59:45:36:4A:3D:6D:54:79:B0
- X509v3 Authority Key Identifier: keyid:74:B1:9E:C3:36:68:B3:66:80:77:31:71:09:CD:B2:46:50:44:8B:0C

1.7 End entity certificate

```
cd intermediate3
mkdir endEntity
cd endEntity
mkdir private public csr

openssl genpkey -algorithm ED25519 -out ./private/EndEntity_PrivateKey.pem
chmod 400 ./private/EndEntity_PrivateKey.pem
openssl pkey -in ./private/EndEntity_PrivateKey.pem -pubout -out ./public/EndEntity_PublicKey.pem
chmod 400 ./public/EndEntity_PublicKey.pem
openssl pkey -inform PEM -pubin -in ./public/EndEntity_PublicKey.pem -text -noout

ED25519 Public-Key:
pub:
    d7:c6:26:eb:48:96:52:04:1a:76:07:cb:50:fd:0d:
    04:45:54:c1:c7:46:dc:0f:be:08:2d:61:cc:be:e2:
    06:bc
```

Put openssl.cnf to endEntity directory. Edit the cnf file to have the following content:

```
#-------------------------openssl.cnf begin------------------------------

[ req ]
distinguished_name        = req_distinguished_name
string_mask = utf8only
req_extensions = v3_req # The extensions to add to a certificate request

[ req_distinguished_name ]
countryName                         = Country Name (2 letter code)
countryName_default                 = PL
countryName_min                         = 2
countryName_max                         = 2
```

```
stateOrProvinceName                = State or Province Name (full name)
stateOrProvinceName_default        = Lower Silesian Voivodeship

localityName                        = Locality Name (eg, city)
localityName_default                = Wroclaw

0.organizationName                 = Organization Name (eg, company)
0.organizationName_default         = Wroclaw University of Science and Technology


organizationalUnitName                 = Organizational Unit Name (eg, section)
organizationalUnitName_default         = WPPT/K2

commonName                          = Common Name (e.g. server FQDN or YOUR name)
commonName_max                         = 64
commonName_default      = localhost

emailAddress                           = Email Address
emailAddress_max                = 64
emailAddress_default                   = przemyslaw.kubiak@pwr.edu.pl

[ v3_req ]

# Extensions to add to a certificate request

basicConstraints = CA:FALSE
keyUsage = nonRepudiation, digitalSignature, keyEncipherment
subjectAltName = IP:127.0.0.1

#-------------------------openssl.cnf end-------------------------------
```

Then execute commands:

```
openssl req -config openssl.cnf -new -sha256 -key private/EndEntity_PrivateKey.pem -out csr/EndEntity_Csr.pem
openssl req -text -noout -verify -in csr/EndEntity_Csr.pem

verify OK
Certificate Request:
    Data:
        Version: 1 (0x0)
        Subject: C = PL, ST = Lower Silesian Voivodeship, L = Wroclaw, O = Wroclaw University of Science and Technology,
        OU = WPPT/K2, CN = localhost, emailAddress = przemyslaw.kubiak@pwr.edu.pl
        Subject Public Key Info:
            Public Key Algorithm: ED25519
                ED25519 Public-Key:
                pub:
                    d7:c6:26:eb:48:96:52:04:1a:76:07:cb:50:fd:0d:
                    04:45:54:c1:c7:46:dc:0f:be:08:2d:61:cc:be:e2:
                    06:bc
        Attributes:
        Requested Extensions:
            X509v3 Basic Constraints:
                CA:FALSE
            X509v3 Key Usage:
                Digital Signature, Non Repudiation, Key Encipherment
            X509v3 Subject Alternative Name:
                IP Address:127.0.0.1
    Signature Algorithm: ED25519
         c0:23:65:2d:79:e8:88:a9:71:81:ae:3c:31:bb:46:cd:4a:df:
         f2:33:fc:e5:75:78:b6:7e:1a:bf:ed:33:f7:2a:60:44:c9:ad:
         04:7d:86:17:36:f2:17:ad:67:f4:66:5e:d9:e1:ac:c4:27:9b:
         c5:a5:41:ee:b5:78:d3:7f:ac:03


cd ..
openssl ca -config openssl.cnf -extensions usr_cert -notext -md sha512 -days 3560 \
       -in endEntity/csr/EndEntity_Csr.pem  -out endEntity/certs/EndEntity_Cert.pem
openssl x509 -noout -text -in endEntity/certs/EndEntity_Cert.pem

Certificate:
    Data:
        Version: 3 (0x2)
```

```
        Serial Number: 6670 (0x1a0e)
        Signature Algorithm: ED448
        Issuer: C = PL, ST = Lower Silesian Voivodeship, L = Wroclaw, O = Wroclaw University of Science and Technology,
        OU = WPPT/K2, CN = System Security I - IntermediateCA1.3, emailAddress = przemyslaw.kubiak@pwr.edu.pl
        Validity
            Not Before: May 12 19:06:35 2019 GMT
            Not After : Feb  8 19:06:35 2029 GMT
        Subject: C = PL, ST = Lower Silesian Voivodeship, L = Wroclaw, O = Wroclaw University of Science and Technology,
        OU = WPPT/K2, CN = localhost, emailAddress = przemyslaw.kubiak@pwr.edu.pl
        Subject Public Key Info:
            Public Key Algorithm: ED25519
                ED25519 Public-Key:
                pub:
                    d7:c6:26:eb:48:96:52:04:1a:76:07:cb:50:fd:0d:
                    04:45:54:c1:c7:46:dc:0f:be:08:2d:61:cc:be:e2:
                    06:bc
        X509v3 extensions:
            X509v3 Basic Constraints:
                CA:FALSE
            Netscape Cert Type:
                SSL Client, S/MIME
            Netscape Comment:
                OpenSSL Generated Client Certificate
            X509v3 Subject Key Identifier:
                C6:17:70:3E:D7:74:F2:65:EB:9A:EA:8E:E7:97:97:E5:91:06:97:4B
            X509v3 Authority Key Identifier:
                keyid:8F:D4:99:A9:38:CB:00:17:F1:F9:07:59:45:36:4A:3D:6D:54:79:B0

            X509v3 Key Usage: critical
                Digital Signature, Non Repudiation, Key Encipherment
            X509v3 Extended Key Usage:
                TLS Web Client Authentication, E-mail Protection
    Signature Algorithm: ED448
         c0:44:e7:40:f0:bc:b8:52:3d:66:e8:63:c6:6c:9e:52:77:5c:
         c4:42:39:32:ab:39:09:73:d8:d3:35:d6:4a:2d:a3:8e:3d:32:
         21:8d:6e:3f:ea:6c:39:75:2a:11:95:a7:95:b7:b1:e5:16:3b:
         bf:25:00:dc:18:0b:64:77:75:98:eb:de:4a:2d:82:38:77:ad:
         d4:d3:6a:50:07:ad:0d:ba:4d:01:6c:34:72:80:19:b7:ca:70:
         9f:e0:cd:30:bd:d1:76:aa:21:77:86:6b:97:16:00:70:d3:9e:
         d3:77:4d:02:11:00
```

Task: Verify the whole certificate chain.
Let first create a single file that contains the fragment of chain above the EndEntity certificate. For exemplary instruction see https://jamielinux.com/docs/openssl-certificate-authority/create-the-intermediate-pair.html#create-the-certificate-chain-file

```
cd root1/ca
cat intermediate1/intermediate2/intermediate3/certs/IntermediateCa1_3_Cert.pem \
intermediate1/intermediate2/certs/IntermediateCa1_2_Cert.pem \
intermediate1/certs/IntermediateCa1_1_Cert2.pem \
certs/RootCa1Cert.pem > intermediate1/intermediate2/intermediate3/certs/IntermediateCa1_3_Cert_ChainToRoot1.pem

openssl verify -CAfile intermediate1/intermediate2/intermediate3/certs/IntermediateCa1_3_Cert_ChainToRoot1.pem \
        intermediate1/intermediate2/intermediate3/endEntity/certs/EndEntity_Cert.pem

    intermediate1/intermediate2/intermediate3/endEntity/certs/EndEntity_Cert.pem: OK
```

So we have the correct chain between endEntity and Root1.
Task: Learn how to set serial number of the certificate.
Solution: Basically the value written in ”serial” file is hexadecimal format. On the website https://www.phildev.net/ssl/opensslconf.html we read:
”The serial number which the CA is currently at. You should not initialize this with a number! instead, use the -create_serial option”
and on the page https://www.phildev.net/ssl/creating_ca.html
”-create_serial is especially important. Many HOW-TOs will have you echo ”01” into the serial file thus starting the serial number at 1, and using 8-bit serial numbers instead of 128-bit serial numbers. This will generate a random 128-bit serial number to start with. The randomness helps to ensure that if you make a mistake and start over, you won’t overwrite existing serial numbers out there.”
So for the next chain of certificates we shall follow the hint above.

1.8 RootCa2 keypair and certificate – RSA keys

Copy the directory structure from root1, change the names in the openssl.cnf files and delete unnecessary files.

So we shall start with RootCa2PrivateKey.pem, however this time we shall generate RSA key, according to the tutorial from https://www.phildev.net/ssl/creating_ca.html

```
cd ca
openssl genrsa -out ./private/RootCa2PrivateKey.pem 4096

openssl req -config openssl.cnf -new -sha512 -key ./private/RootCa2PrivateKey.pem -out ./csr/RootCa2_Csr.pem
openssl req -text -noout -verify -in ./csr/RootCa2_Csr.pem


verify OK
Certificate Request:
    Data:
        Version: 1 (0x0)
        Subject: C = PL, ST = Lower Silesian Voivodeship, L = Wroclaw, O = Wroclaw University of Science and Technology,
        OU = WPPT/K2, CN = System Security I - RootCA2, emailAddress = przemyslaw.kubiak@pwr.edu.pl
        Subject Public Key Info:
            Public Key Algorithm: rsaEncryption
                RSA Public-Key: (4096 bit)
                Modulus:
                    00:af:2b:cb:45:53:36:03:91:b9:9e:93:b7:c8:0f:
                    0c:57:1b:fb:dc:10:aa:95:3f:fb:d9:a8:04:a4:20:
                    8b:bb:86:6c:96:52:aa:bd:54:12:e8:cf:6f:96:fb:
                    8c:d6:41:60:d2:03:95:12:d6:6c:dd:96:76:95:0a:
                    54:03:47:25:19:2c:bb:ae:86:fc:d8:cc:3a:85:35:
                    8f:25:88:17:f0:d5:ca:ff:24:28:39:70:c2:7f:5c:
                    25:04:02:f1:d4:5a:45:a9:1d:54:14:95:d0:a5:2a:
                    63:74:21:52:f3:95:03:73:1c:0a:4b:1a:ee:06:08:
                    b2:0f:57:8a:d7:ef:72:e1:6e:10:67:02:44:d7:dd:
                    a1:dd:ef:54:6e:6a:3d:c2:1b:69:c2:eb:40:7d:12:
                    2e:5f:80:45:79:ba:95:87:cb:67:9e:e7:53:78:7a:
                    00:2b:8e:b4:3e:94:80:00:cb:15:41:d5:ef:8b:77:
                    ae:c6:d8:ba:88:d3:80:5a:8a:73:3c:92:98:c3:53:
                    8b:dc:be:e5:cf:32:12:a1:9a:d3:2a:de:8b:1d:7f:
                    39:f5:2c:8f:5e:5d:c4:f4:94:09:18:42:2e:6b:49:
                    f1:f9:5a:d5:77:57:a1:b6:1b:88:b5:23:f6:ab:84:
                    fc:20:d7:99:1c:6b:ce:d5:36:48:be:44:91:5d:95:
                    05:de:ee:c5:92:7d:20:87:a5:7a:59:61:cb:b7:71:
                    c0:38:51:e6:06:19:ec:ba:cc:ac:85:3f:4d:d3:fc:
                    15:99:96:76:38:4c:4f:c8:eb:0b:04:16:b3:55:e5:
                    a0:4c:a4:31:b7:f4:e5:09:a2:c9:fd:89:a6:c9:e0:
                    73:a1:68:ab:97:a9:f6:60:a2:81:4e:08:e3:44:2d:
                    f0:50:22:0c:65:ad:cb:df:d9:c3:53:98:12:f3:3c:
                    f7:04:2f:a7:53:66:77:a2:3e:0c:24:2b:09:7b:06:
                    b2:06:eb:00:5b:13:4b:57:74:74:88:86:50:87:08:
                    71:6f:a6:a7:03:1f:b3:6f:1a:73:c3:6c:15:80:3c:
                    d8:9b:81:da:ae:87:87:b1:8b:41:01:9f:b9:b9:90:
                    dc:f9:a2:dc:60:db:48:01:79:5e:9c:0d:95:fb:68:
                    b0:13:9a:ec:a0:9e:5d:4e:43:1c:43:bf:a2:c0:2e:
                    d2:70:0a:bc:c5:2f:11:c1:0a:df:07:b1:b1:0b:e3:
                    09:2c:20:7f:97:db:87:d8:9d:bc:90:5a:0f:d6:88:
                    e8:c9:c0:77:87:9d:99:43:d9:c2:8d:13:06:99:56:
                    85:30:12:27:5f:eb:e7:74:17:a6:84:77:07:65:cf:
                    63:e0:47:0f:b7:50:2d:d9:01:c8:72:22:3e:67:54:
                    44:8a:f7
                Exponent: 65537 (0x10001)
        Attributes:
            a0:00
    Signature Algorithm: sha512WithRSAEncryption
         92:90:75:83:7e:0e:39:bd:78:8a:29:4f:f4:4f:3a:aa:59:44:
         01:a8:33:df:8b:4b:32:01:c6:f4:da:89:e1:0d:d2:a6:88:a6:
         ec:cb:64:fe:cc:84:ac:5b:81:7f:0b:e0:0a:50:6b:15:e7:06:
         29:54:df:8b:47:0d:b6:eb:86:ab:fd:1a:51:00:c0:c4:be:0f:
         8c:dc:6e:56:60:14:0b:58:61:6f:86:35:5f:6b:9a:5d:33:2b:
         eb:67:df:8e:a9:d3:b5:a3:18:14:18:61:e8:7a:db:06:99:11:
         38:8f:84:1b:3a:fa:05:7c:fa:e7:98:c3:cd:6f:ce:a8:a1:dc:
         2b:45:e1:44:78:da:64:f2:e8:07:e4:7d:c2:18:92:9c:86:58:
         02:29:55:ed:c2:2a:8d:68:cb:04:88:72:dc:7d:e9:41:77:4a:
         bb:81:01:48:10:4d:b9:77:69:d8:09:bf:2a:fe:85:66:59:d5:
         91:1b:7f:bd:af:79:7e:83:07:d3:ce:45:8c:85:cd:bd:41:b5:
         5a:9b:e9:ab:31:d7:f9:8a:bf:49:74:ef:cf:47:53:b6:cb:6a:
         c0:6c:de:89:d8:fd:41:8b:4a:53:fd:ce:5d:bf:3b:1c:b8:71:
         a7:cb:08:0d:d1:4c:c8:90:5c:87:80:1e:8f:0e:42:78:6a:46:
         b9:18:44:bb:a6:01:c0:ce:b4:5d:e6:07:05:f9:bb:29:ac:dd:
         57:3f:c7:fe:67:2c:ae:33:80:52:63:03:9f:8b:87:14:e1:94:
         53:20:fe:9a:15:c0:6b:e2:ae:16:fd:b7:d4:1d:e9:9f:5b:cb:
         22:d3:03:88:ff:b4:ad:45:e7:5b:75:92:70:19:18:85:b2:9a:
         5f:c4:9e:45:cd:f2:1d:eb:a4:f0:78:ab:b1:f2:60:bb:04:9f:
         c4:14:39:5c:2c:31:05:7a:ab:f8:54:f7:89:5b:6b:ff:90:35:
         85:4c:c1:92:07:28:dd:90:a6:3d:5c:e0:73:6c:75:88:e7:5d:
         46:c2:c6:43:6e:13:5d:d3:41:9a:33:9f:09:3e:20:04:79:b2:
         cd:e0:3f:7d:5d:37:11:b4:17:95:0e:30:b3:de:df:e0:49:60:
         a2:6a:4b:81:a1:5a:a2:86:3b:4b:9c:99:6d:21:4f:4b:ff:fa:
         ed:d0:d3:42:4f:0c:dd:4e:78:f8:84:0c:2f:34:42:f2:e2:dc:
         42:2c:2f:cd:14:80:c5:07:fd:b2:97:83:39:f5:09:36:0c:70:
         b9:6f:5b:69:06:d9:97:73:0a:e5:11:0a:84:03:92:76:c8:d7:
         40:df:94:fc:30:14:dd:e8:01:9a:cf:7a:c1:c8:61:1c:cf:1c:
         06:76:fe:39:f1:20:9f:b7

touch index.txt
openssl ca -create_serial -notext -out certs/RootCa2Cert.pem -days 3660 -keyfile ./private/RootCa2PrivateKey.pem \
        -selfsign -extensions v3_ca -config ./openssl.cnf -infiles ./csr/RootCa2_Csr.pem
```

Note that in the case of RootCa1 we have used ”openssl req” command to generate the certificate:

```
openssl req -config openssl.cnf -key ./private/RootCa1PrivateKey.pem -new -x509 -days 3660 \
 -sha512 -extensions v3_ca -out certs/RootCa1Cert.pem
```

and in result the selfsigned certificate of RootCa1 is not present in RootCa1’s newcerts directory, moreover is not listed in the index.txt database. What is more, RootCa1’s selfsigned certificate has random serial number, but the serial numbers of the certificates issued by RootCa1 to subordinate IntCA1.1 are not related to RootCa1’s serial number:

```
Certificate:
    Data:
        Version: 3 (0x2)
        Serial Number:
            2f:f1:8b:b2:19:76:06:4a:cb:41:d7:ac:82:68:29:60:3f:f0:b1:75
        Signature Algorithm: ecdsa-with-SHA512
        Issuer: C = PL, ST = Lower Silesian Voivodeship, L = Wroclaw, O = Wroclaw University of Science and Technology,
        OU = WPPT/K2, CN = System Security I - RootCA1, emailAddress = przemyslaw.kubiak@pwr.edu.pl
        Validity
            Not Before: Feb 22 18:53:35 2019 GMT
            Not After : Mar  1 18:53:35 2029 GMT
        Subject: C = PL, ST = Lower Silesian Voivodeship, L = Wroclaw, O = Wroclaw University of Science and Technology,
        OU = WPPT/K2, CN = System Security I - RootCA1, emailAddress = przemyslaw.kubiak@pwr.edu.pl
...

 Certificate:
    Data:
        Version: 3 (0x2)
        Serial Number: 4660 (0x1234)
        Signature Algorithm: ecdsa-with-SHA384
        Issuer: C = PL, ST = Lower Silesian Voivodeship, L = Wroclaw, O = Wroclaw University of Science and Technology,
        OU = WPPT/K2, CN = System Security I - RootCA1, emailAddress = przemyslaw.kubiak@pwr.edu.pl
        Validity
            Not Before: Feb 23 23:55:31 2019 GMT
            Not After : Feb 18 23:55:31 2029 GMT
        Subject: C = PL, ST = Lower Silesian Voivodeship, O = Wroclaw University of Science and Technology,
        OU = WPPT/K2, CN = System Security I - IntermediateCA1.1, emailAddress = przemyslaw.kubiak@pwr.edu.pl
...
```

On the other hand:

- The serial number of RootCa2’s selfsigned certificate is copied to the ”serial” file and will be incremented for the next certificates issued by RootCa2.
- The RootCa2’s selfsigned certificate is copied to newcerts subdirectory and is present the index.txt database

So the preffered method for issuing selfsigned certificates should be:

```
openssl ca -create_serial ... -selfsign ...
```

1.9 IntCA2.1 certificate – DSA key

Let IntCA2.1 use DSA keys to sign certificates.

```
cd intermediate1
openssl dsaparam -out private/dsaparam.pem 4096
openssl dsaparam -in private/dsaparam.pem -noout -text

    P:
        00:d8:06:53:b5:c2:d3:09:60:19:09:d7:a9:a1:73:
        be:f0:cf:21:22:03:72:a5:1c:53:f3:90:8c:9f:8c:
        ed:5c:36:37:74:65:a3:cb:5d:a9:a2:da:1d:cb:3a:
        35:0d:b5:57:07:7c:08:1d:2a:70:ef:71:22:30:1b:
        f7:19:90:48:bd:50:8d:2e:ab:07:a2:27:38:17:4c:
        f1:6f:15:04:5a:b0:d0:aa:6a:d2:f7:17:1a:ec:32:
        92:36:98:11:63:6e:4b:3e:5b:2a:be:ab:d6:9a:03:
        e8:20:cc:a8:cc:aa:3b:0a:f3:31:25:de:73:8c:22:
        7c:3c:d1:0e:24:9a:4c:91:cb:05:91:85:ca:78:bf:
        cd:bb:bb:69:ab:b4:a8:5b:27:1f:11:42:6b:a6:5e:
        72:1b:dc:db:14:60:8b:4a:6d:3f:00:e6:9b:ab:ac:
        38:84:a9:1d:e3:ea:79:cd:32:47:6a:b7:fc:1f:c9:
        32:9f:f0:4c:d5:4a:c5:5f:3f:ed:f0:0a:df:d2:a8:
        a5:ee:71:ec:44:64:52:de:1b:2b:8a:6f:88:54:f3:
        7b:30:f5:98:4c:12:13:29:48:37:1e:d9:d1:a9:83:
        97:76:b0:36:08:b2:42:2c:7d:29:50:1e:8b:e1:ed:
        d5:ce:bd:06:94:cd:cc:ba:f7:2f:03:93:57:f3:33:
```

```
        eb:37:6c:56:7c:38:a1:4d:90:43:0a:81:c8:55:3e:
        12:e2:d4:5f:fe:20:de:17:48:67:6c:e1:26:ab:1e:
        93:7f:47:8f:3d:8b:bd:f6:6b:d5:41:a4:cd:a5:1e:
        4b:c0:47:ee:a8:94:fe:37:42:82:75:7d:6c:1d:cb:
        07:bf:47:01:e5:8c:ee:c6:28:14:a0:0b:56:e5:2e:
        68:19:5d:73:8f:e5:e6:d5:30:eb:b5:77:cb:fa:21:
        bb:f5:b5:5a:1d:d1:62:9b:df:18:d3:1c:7a:10:14:
        77:25:ef:2b:a0:d3:47:11:48:0a:18:19:20:64:ea:
        52:bb:df:09:6f:c6:c9:e4:5b:f2:a2:26:f1:84:e5:
        88:06:94:12:8f:e8:c5:63:f3:98:42:2b:57:51:ea:
        1f:2c:6f:eb:bb:86:43:b3:d4:1b:a4:cd:e8:e8:54:
        bf:8f:37:f3:a1:50:96:09:32:49:bc:49:41:cb:4b:
        ad:87:51:08:ef:ff:99:8b:bb:43:aa:5e:de:48:f8:
        12:ca:52:5f:ba:54:3b:2d:62:49:80:90:f5:ec:35:
        e5:3e:d0:81:9c:b3:99:ae:5e:f9:a3:1c:5e:6c:25:
        29:c6:8d:d7:3e:c9:45:7d:fe:83:a4:ca:33:ad:62:
        55:24:36:84:3c:8c:42:7e:d2:37:c5:47:d6:bf:f1:
        22:33:0f
    Q:
        00:c3:f7:6c:88:25:d0:86:af:8a:f4:11:fa:84:b6:
        e7:24:1e:52:17:c7:32:67:77:8f:2e:6e:5a:d4:d1:
        82:b6:2d
    G:
        00:a0:7f:db:cc:7f:b4:38:d9:5a:d6:ce:b3:6e:55:
        01:91:61:75:fe:24:43:92:7e:0e:5b:ce:c6:fd:06:
        56:9f:0d:f9:db:b5:ef:ee:52:6e:54:67:14:41:f4:
        fe:3d:12:bd:71:fe:4c:7b:50:db:af:b5:ed:b6:e9:
        f7:14:b9:13:c1:8f:29:8a:58:3f:db:46:67:30:e5:
        8a:b1:5c:b5:ee:f2:8b:23:0c:27:4c:b7:dd:1d:2c:
        2f:ab:7b:f5:fa:f6:b0:9c:b4:26:b8:8a:3a:e2:15:
        1f:a4:44:78:3b:29:d7:92:6a:1e:10:8a:e6:bc:de:
        5e:1c:7b:30:5a:48:2d:00:62:7d:73:cd:d7:ba:b0:
        c8:f0:10:24:1b:f4:8a:69:49:4a:8d:74:4c:f8:3d:
        dc:56:02:74:18:2b:4c:79:dc:4f:7c:02:80:1a:3a:
        04:24:78:46:d0:26:f9:7f:66:70:4e:6e:8f:2b:0b:
        06:e8:ea:17:e3:b5:ce:21:1d:cd:05:f7:e2:72:7b:
        7a:b9:a6:3b:37:e5:33:8f:82:30:6f:21:eb:86:b4:
        35:fe:d2:76:15:0a:ea:7a:47:97:40:78:2e:da:9a:
        df:4c:e8:02:67:71:60:af:b9:ef:f7:60:ee:ff:fa:
        a3:f8:18:95:af:c3:3a:55:d1:90:bc:78:1f:74:da:
        b0:44:2b:51:b6:1e:d5:57:b7:1e:16:48:63:4d:c0:
        35:83:c1:83:02:4f:f7:3a:57:6e:88:a8:cd:36:e5:
        2d:97:cd:fa:09:22:6f:1e:de:b6:be:1a:d7:cb:48:
        67:cf:a9:67:4e:43:36:5a:d1:23:99:42:9c:8b:1d:
        be:e5:c1:13:41:33:54:90:93:13:52:53:66:1e:29:
        b3:8e:a9:6b:ce:58:42:7c:a7:e8:38:41:fc:a7:72:
        dd:70:83:c9:cc:19:82:e1:00:bf:3b:4d:5b:33:74:
        5a:a4:75:61:d0:3b:62:92:ac:2a:5e:41:3c:23:f2:
        79:d3:c0:7d:d7:f9:27:fe:fb:a6:37:37:14:0e:be:
        e8:e6:67:e4:26:81:01:fd:e0:33:67:34:6b:12:38:
        e1:dd:83:8c:cc:8e:bd:d0:fd:91:36:85:da:ea:62:
        85:5b:36:73:38:56:76:ad:11:f1:fa:56:39:a5:63:
        91:8d:a2:d8:93:69:80:e3:ee:1f:f7:81:d4:01:d4:
        33:79:4c:c4:dc:a3:ec:d1:54:7b:7a:5d:75:8a:0c:
        e5:96:7a:12:13:80:ab:07:5b:5a:80:ff:c3:eb:ce:
        f1:fa:d1:cd:89:3f:74:bb:fd:71:ac:14:f3:1d:91:
        6d:c0:63:e5:1f:02:c1:bd:ba:2e:ca:72:e5:8a:f4:
        53:a8:9a
```

$\mathrm{ord}G = p$, $G \in \mathbb{F}_p^*$.
The private/dsaparam.pem file decoded on https://lapo.it/asn1js/ :

```
SEQUENCE (3 elem)
  INTEGER (4096 bit) 881303946333794411096237240186040020288803043809321465228240903820222...
  INTEGER (256 bit) 886381657587777442358965212053165407066721545165125852183441166042148 5...
  INTEGER (4096 bit) 654780619378495611224783499548268735055337756174861251531568134165864...
```

which hexadecimally is:

```
30 82 04 2D
    02 82 02 01  00 D8 06 53 B5 C2 D3 09
    60 19 09 D7 A9 A1 73 BE  F0 CF 21 22 03 72 A5 1C
    53 F3 90 8C 9F 8C ED 5C  36 37 74 65 A3 CB 5D A9
    A2 DA 1D CB 3A 35 0D B5  57 07 7C 08 1D 2A 70 EF
    71 22 30 1B F7 19 90 48  BD 50 8D 2E AB 07 A2 27
    ... skipping 416 bytes ...
    FE 83 A4 CA 33 AD 62 55  24 36 84 3C 8C 42 7E D2
    37 C5 47 D6 BF F1 22 33  0F

    02 21  00 C3 F7 6C 88 25  D0 86 AF 8A F4 11 FA 84
    B6 E7 24 1E 52 17 C7 32  67 77 8F 2E 6E 5A D4 D1
    82 B6 2D

    02 82 02 01  00 A0 7F DB  CC 7F B4 38 D9 5A D6 CE
    B3 6E 55 01 91 61 75 FE  24 43 92 7E 0E 5B CE C6
    FD 06 56 9F 0D F9 DB B5  EF EE 52 6E 54 67 14 41
    F4 FE 3D 12 BD 71 FE 4C  7B 50 DB AF B5 ED B6 E9
    F7 14 B9 13 C1 8F 29 8A  58 3F DB 46 67 30 E5 8A
    B1 5C B5 EE
    ... skipping 416 bytes ...
```

```
    C0 63 E5 1F 02 C1 BD BA  2E CA 72 E5 8A F4 53 A8
    9A

openssl gendsa -out ./private/IntermediateCa2_1_PrivateKey.pem ./private/dsaparam.pem
chmod 400 ./private/IntermediateCa2_1_PrivateKey.pem

openssl req -config openssl.cnf -new -sha256 -key ./private/IntermediateCa2_1_PrivateKey.pem -out ./csr/IntermediateCa2_1_Csr.pem
openssl req -text -noout -verify -in ./csr/IntermediateCa2_1_Csr.pem

verify OK
Certificate Request:
    Data:
        Version: 1 (0x0)
        Subject: C = PL, ST = Lower Silesian Voivodeship, L = Wroclaw, O = Wroclaw University of Science and Technology,
        OU = WPPT/K2, CN = System Security I - IntermediateCA2.1, emailAddress = przemyslaw.kubiak@pwr.edu.pl
        Subject Public Key Info:
            Public Key Algorithm: dsaEncryption
                pub:
                    32:3b:68:30:0e:57:4d:5a:df:c6:7e:e8:f1:4e:50:
                    56:ca:55:2f:5d:75:a0:30:91:cb:48:d8:3d:2c:34:
                    2e:92:d4:18:64:66:73:d5:46:4e:5d:4b:d4:86:4e:
                    bb:60:f3:ca:03:3b:49:9a:6c:b6:6d:98:ed:5e:86:
                    4d:14:90:4d:3b:bd:c8:5b:38:c5:74:6f:8f:48:c3:
                    32:08:c3:bf:31:83:26:b7:60:5d:ea:c4:57:f6:d7:
                    9b:4d:0e:5d:9e:0f:f1:64:b8:f4:fe:e1:10:7e:e5:
                    4a:b7:61:ac:6a:0f:cb:77:37:3b:d6:a9:1f:39:a6:
                    8c:74:2c:5d:2a:ce:58:d8:39:6a:cb:89:52:fd:6b:
                    b4:2c:05:55:44:f9:43:c4:c5:9f:4e:e2:10:56:42:
                    1f:20:38:cc:65:44:21:7b:e6:06:21:5d:67:7e:1f:
                    ae:34:3f:6b:65:b5:6d:16:3e:f7:02:2d:b1:c3:4f:
                    2b:0b:70:c2:23:a4:ae:d4:59:f8:08:22:72:d0:48:
                    d8:7b:29:dc:48:87:c3:bb:c3:5a:6a:03:fb:46:53:
                    66:5e:b4:ba:ba:48:fe:99:df:2d:32:86:a3:16:c2:
                    32:e7:9a:8c:25:4c:02:19:d7:d6:6d:0f:b0:c9:a8:
                    b9:bc:fa:06:ef:b6:ad:e8:14:66:c3:85:fb:33:38:
                    7b:a4:8e:db:79:21:26:f8:54:53:2f:72:ca:44:bf:
                    f9:a9:f7:76:2b:e9:3e:0f:9f:0e:70:d9:31:82:ff:
                    f8:cc:62:b7:84:d1:35:11:a2:4d:3c:7d:c6:78:e9:
                    ff:10:bf:50:1b:9e:ff:e4:51:81:15:d0:bc:16:aa:
                    e6:00:c5:12:07:bd:32:43:07:0c:8f:44:70:6f:71:
                    6c:95:fe:91:44:b6:69:a3:5c:65:df:2e:46:0d:b2:
                    92:b4:0f:5c:a6:e8:17:95:2c:a8:b4:f4:56:0a:a7:
                    cb:d3:88:c7:fc:bd:49:a9:19:67:3f:de:ab:ba:92:
                    32:a3:8b:5c:1d:b8:55:e9:7f:02:dd:3d:54:09:fd:
                    95:d8:1d:50:32:e6:eb:77:e1:7e:94:b3:1f:58:41:
                    3c:89:72:dc:4e:b2:05:b8:3d:08:b3:d6:b0:37:f7:
                    86:ae:cb:f5:c9:a2:cb:98:fa:32:8b:33:8d:f2:c9:
                    4d:e4:f5:a7:8a:20:34:f5:39:a6:d9:4e:d2:c8:c6:
                    23:8b:c6:51:83:99:d4:69:03:69:0a:a5:75:cc:61:
                    78:46:55:ec:12:83:27:39:34:76:e6:d5:ad:1a:8d:
                    13:7f:50:05:12:78:aa:1d:17:7b:ff:06:65:46:92:
                    a8:98:a3:e1:c9:d7:e9:ea:38:1d:65:36:26:61:c3:
                    05:63
                P:
                    00:d8:06:53:b5:c2:d3:09:60:19:09:d7:a9:a1:73:
                    be:f0:cf:21:22:03:72:a5:1c:53:f3:90:8c:9f:8c:
                    ed:5c:36:37:74:65:a3:cb:5d:a9:a2:da:1d:cb:3a:
                    35:0d:b5:57:07:7c:08:1d:2a:70:ef:71:22:30:1b:
                    f7:19:90:48:bd:50:8d:2e:ab:07:a2:27:38:17:4c:
                    f1:6f:15:04:5a:b0:d0:aa:6a:d2:f7:17:1a:ec:32:
                    92:36:98:11:63:6e:4b:3e:5b:2a:be:ab:d6:9a:03:
                    e8:20:cc:a8:cc:aa:3b:0a:f3:31:25:de:73:8c:22:
                    7c:3c:d1:0e:24:9a:4c:91:cb:05:91:85:ca:78:bf:
                    cd:bb:bb:69:ab:b4:a8:5b:27:1f:11:42:6b:a6:5e:
                    72:1b:dc:db:14:60:8b:4a:6d:3f:00:e6:9b:ab:ac:
                    38:84:a9:1d:e3:ea:79:cd:32:47:6a:b7:fc:1f:c9:
                    32:9f:f0:4c:d5:4a:c5:5f:3f:ed:f0:0a:df:d2:a8:
                    a5:ee:71:ec:44:64:52:de:1b:2b:8a:6f:88:54:f3:
                    7b:30:f5:98:4c:12:13:29:48:37:1e:d9:d1:a9:83:
                    97:76:b0:36:08:b2:42:2c:7d:29:50:1e:8b:e1:ed:
                    d5:ce:bd:06:94:cd:cc:ba:f7:2f:03:93:57:f3:33:
                    eb:37:6c:56:7c:38:a1:4d:90:43:0a:81:c8:55:3e:
                    12:e2:d4:5f:fe:20:de:17:48:67:6c:e1:26:ab:1e:
                    93:7f:47:8f:3d:8b:bd:f6:6b:d5:41:a4:cd:a5:1e:
                    4b:c0:47:ee:a8:94:fe:37:42:82:75:7d:6c:1d:cb:
                    07:bf:47:01:e5:8c:ee:c6:28:14:a0:0b:56:e5:2e:
                    68:19:5d:73:8f:e5:e6:d5:30:eb:b5:77:cb:fa:21:
                    bb:f5:b5:5a:1d:d1:62:9b:df:18:d3:1c:7a:10:14:
                    77:25:ef:2b:a0:d3:47:11:48:0a:18:19:20:64:ea:
                    52:bb:df:09:6f:c6:c9:e4:5b:f2:a2:26:f1:84:e5:
                    88:06:94:12:8f:e8:c5:63:f3:98:42:2b:57:51:ea:
                    1f:2c:6f:eb:bb:86:43:b3:d4:1b:a4:cd:e8:e8:54:
                    bf:8f:37:f3:a1:50:96:09:32:49:bc:49:41:cb:4b:
                    ad:87:51:08:ef:ff:99:8b:bb:43:aa:5e:de:48:f8:
                    12:ca:52:5f:ba:54:3b:2d:62:49:80:90:f5:ec:35:
                    e5:3e:d0:81:9c:b3:99:ae:5e:f9:a3:1c:5e:6c:25:
                    29:c6:8d:d7:3e:c9:45:7d:fe:83:a4:ca:33:ad:62:
                    55:24:36:84:3c:8c:42:7e:d2:37:c5:47:d6:bf:f1:
                    22:33:0f
                Q:
                    00:c3:f7:6c:88:25:d0:86:af:8a:f4:11:fa:84:b6:
                    e7:24:1e:52:17:c7:32:67:77:8f:2e:6e:5a:d4:d1:
                    82:b6:2d
```

```
            G:
                00:a0:7f:db:cc:7f:b4:38:d9:5a:d6:ce:b3:6e:55:
                01:91:61:75:fe:24:43:92:7e:0e:5b:ce:c6:fd:06:
                56:9f:0d:f9:db:b5:ef:ee:52:6e:54:67:14:41:f4:
                fe:3d:12:bd:71:fe:4c:7b:50:db:af:b5:ed:b6:e9:
                f7:14:b9:13:c1:8f:29:8a:58:3f:db:46:67:30:e5:
                8a:b1:5c:b5:ee:f2:8b:23:0c:27:4c:b7:dd:1d:2c:
                2f:ab:7b:f5:fa:f6:b0:9c:b4:26:b8:8a:3a:e2:15:
                1f:a4:44:78:3b:29:d7:92:6a:1e:10:8a:e6:bc:de:
                5e:1c:7b:30:5a:48:2d:00:62:7d:73:cd:d7:ba:b0:
                c8:f0:10:24:1b:f4:8a:69:49:4a:8d:74:4c:f8:3d:
                dc:56:02:74:18:2b:4c:79:dc:4f:7c:02:80:1a:3a:
                04:24:78:46:d0:26:f9:7f:66:70:4e:6e:8f:2b:0b:
                06:e8:ea:17:e3:b5:ce:21:1d:cd:05:f7:e2:72:7b:
                7a:b9:a6:3b:37:e5:33:8f:82:30:6f:21:eb:86:b4:
                35:fe:d2:76:15:0a:ea:7a:47:97:40:78:2e:da:9a:
                df:4c:e8:02:67:71:60:af:b9:ef:f7:60:ee:ff:fa:
                a3:f8:18:95:af:c3:3a:55:d1:90:bc:78:1f:74:da:
                b0:44:2b:51:b6:1e:d5:57:b7:1e:16:48:63:4d:c0:
                35:83:c1:83:02:4f:f7:3a:57:6e:88:a8:cd:36:e5:
                2d:97:cd:fa:09:22:6f:1e:de:b6:be:1a:d7:cb:48:
                67:cf:a9:67:4e:43:36:5a:d1:23:99:42:9c:8b:1d:
                be:e5:c1:13:41:33:54:90:93:13:52:53:66:1e:29:
                b3:8e:a9:6b:ce:58:42:7c:a7:e8:38:41:fc:a7:72:
                dd:70:83:c9:cc:19:82:e1:00:bf:3b:4d:5b:33:74:
                5a:a4:75:61:d0:3b:62:92:ac:2a:5e:41:3c:23:f2:
                79:d3:c0:7d:d7:f9:27:fe:fb:a6:37:37:14:0e:be:
                e8:e6:67:e4:26:81:01:fd:e0:33:67:34:6b:12:38:
                e1:dd:83:8c:cc:8e:bd:d0:fd:91:36:85:da:ea:62:
                85:5b:36:73:38:56:76:ad:11:f1:fa:56:39:a5:63:
                91:8d:a2:d8:93:69:80:e3:ee:1f:f7:81:d4:01:d4:
                33:79:4c:c4:dc:a3:ec:d1:54:7b:7a:5d:75:8a:0c:
                e5:96:7a:12:13:80:ab:07:5b:5a:80:ff:c3:eb:ce:
                f1:fa:d1:cd:89:3f:74:bb:fd:71:ac:14:f3:1d:91:
                6d:c0:63:e5:1f:02:c1:bd:ba:2e:ca:72:e5:8a:f4:
                53:a8:9a
        Attributes:
            a0:00
    Signature Algorithm: dsa_with_SHA256
         r:
             00:b3:aa:b5:fa:f9:6f:d4:b0:65:5e:a6:f5:e8:e2:
             19:0d:80:ec:90:43:74:70:d1:8d:de:43:2f:ce:53:
             95:7b:44
         s:
             62:89:52:d7:50:78:35:c5:d1:f3:4a:c1:45:67:8c:
             2e:ef:dc:c0:b0:e2:d4:e1:d6:1c:3e:f0:0c:5d:13:
             56:33
```

Note that despite very long p signature value (r, s) is short. Generate the certificate:

```
cd ..
openssl ca -config openssl.cnf -extensions v3_intermediate_ca  -notext -days 3648 \
       -in intermediate1/csr/IntermediateCa2_1_Csr.pem  -out intermediate1/certs/IntermediateCa2_1_Cert.pem
openssl x509 -noout -text -in intermediate1/certs/IntermediateCa2_1_Cert.pem
```

```
Certificate:
    Data:
        Version: 3 (0x2)
        Serial Number:
            2e:e1:fe:6c:61:54:a6:fb:52:4f:70:a9:46:61:82:ba:47:18:e0:5c
        Signature Algorithm: sha512WithRSAEncryption
        Issuer: C = PL, ST = Lower Silesian Voivodeship, O = Wroclaw University of Science and Technology,
        OU = WPPT/K2, CN = System Security I - RootCA2, emailAddress = przemyslaw.kubiak@pwr.edu.pl
        Validity
            Not Before: May 17 13:15:32 2019 GMT
            Not After : May 12 13:15:32 2029 GMT
        Subject: C = PL, ST = Lower Silesian Voivodeship, O = Wroclaw University of Science and Technology,
        OU = WPPT/K2, CN = System Security I - IntermediateCA2.1, emailAddress = przemyslaw.kubiak@pwr.edu.pl
        Subject Public Key Info:
            Public Key Algorithm: dsaEncryption
                pub:
                    32:3b:68:30:0e:57:4d:5a:df:c6:7e:e8:f1:4e:50:
                    56:ca:55:2f:5d:75:a0:30:91:cb:48:d8:3d:2c:34:
                    2e:92:d4:18:64:66:73:d5:46:4e:5d:4b:d4:86:4e:
                    bb:60:f3:ca:03:3b:49:9a:6c:b6:6d:98:ed:5e:86:
                    4d:14:90:4d:3b:bd:c8:5b:38:c5:74:6f:8f:48:c3:
                    32:08:c3:bf:31:83:26:b7:60:5d:ea:c4:57:f6:d7:
                    9b:4d:0e:5d:9e:0f:f1:64:b8:f4:fe:e1:10:7e:e5:
                    4a:b7:61:ac:6a:0f:cb:77:37:3b:d6:a9:1f:39:a6:
                    8c:74:2c:5d:2a:ce:58:d8:39:6a:cb:89:52:fd:6b:
                    b4:2c:05:55:44:f9:43:c4:c5:9f:4e:e2:10:56:42:
                    1f:20:38:cc:65:44:21:7b:e6:06:21:5d:67:7e:1f:
                    ae:34:3f:6b:65:b5:6d:16:3e:f7:02:2d:b1:c3:4f:
                    2b:0b:70:c2:23:a4:ae:d4:59:f8:08:22:72:d0:48:
                    d8:7b:29:dc:48:87:c3:bb:c3:5a:6a:03:fb:46:53:
                    66:5e:b4:ba:ba:48:fe:99:df:2d:32:86:a3:16:c2:
                    32:e7:9a:8c:25:4c:02:19:d7:d6:6d:0f:b0:c9:a8:
                    b9:bc:fa:06:ef:b6:ad:e8:14:66:c3:85:fb:33:38:
                    7b:a4:8e:db:79:21:26:f8:54:53:2f:72:ca:44:bf:
                    f9:a9:f7:76:2b:e9:3e:0f:9f:0e:70:d9:31:82:ff:
                    f8:cc:62:b7:84:d1:35:11:a2:4d:3c:7d:c6:78:e9:
                    ff:10:bf:50:1b:9e:ff:e4:51:81:15:d0:bc:16:aa:
```

```
            e6:00:c5:12:07:bd:32:43:07:0c:8f:44:70:6f:71:
            6c:95:fe:91:44:b6:69:a3:5c:65:df:2e:46:0d:b2:
            92:b4:0f:5c:a6:e8:17:95:2c:a8:b4:f4:56:0a:a7:
            cb:d3:88:c7:fc:bd:49:a9:19:67:3f:de:ab:ba:92:
            32:a3:8b:5c:1d:b8:55:e9:7f:02:dd:3d:54:09:fd:
            95:d8:1d:50:32:e6:eb:77:e1:7e:94:b3:1f:58:41:
            3c:89:72:dc:4e:b2:05:b8:3d:08:b3:d6:b0:37:f7:
            86:ae:cb:f5:c9:a2:cb:98:fa:32:8b:33:8d:f2:c9:
            4d:e4:f5:a7:8a:20:34:f5:39:a6:d9:4e:d2:c8:c6:
            23:8b:c6:51:83:99:d4:69:03:69:0a:a5:75:cc:61:
            78:46:55:ec:12:83:27:39:34:76:e6:d5:ad:1a:8d:
            13:7f:50:05:12:78:aa:1d:17:7b:ff:06:65:46:92:
            a8:98:a3:e1:c9:d7:e9:ea:38:1d:65:36:26:61:c3:
            05:63
        P:
            00:d8:06:53:b5:c2:d3:09:60:19:09:d7:a9:a1:73:
            be:f0:cf:21:22:03:72:a5:1c:53:f3:90:8c:9f:8c:
            ed:5c:36:37:74:65:a3:cb:5d:a9:a2:da:1d:cb:3a:
            35:0d:b5:57:07:7c:08:1d:2a:70:ef:71:22:30:1b:
            f7:19:90:48:bd:50:8d:2e:ab:07:a2:27:38:17:4c:
            f1:6f:15:04:5a:b0:d0:aa:6a:d2:f7:17:1a:ec:32:
            92:36:98:11:63:6e:4b:3e:5b:2a:be:ab:d6:9a:03:
            e8:20:cc:a8:cc:aa:3b:0a:f3:31:25:de:73:8c:22:
            7c:3c:d1:0e:24:9a:4c:91:cb:05:91:85:ca:78:bf:
            cd:bb:bb:69:ab:b4:a8:5b:27:1f:11:42:6b:a6:5e:
            72:1b:dc:db:14:60:8b:4a:6d:3f:00:e6:9b:ab:ac:
            38:84:a9:1d:e3:ea:79:cd:32:47:6a:b7:fc:1f:c9:
            32:9f:f0:4c:d5:4a:c5:5f:3f:ed:f0:0a:df:d2:a8:
            a5:ee:71:ec:44:64:52:de:1b:2b:8a:6f:88:54:f3:
            7b:30:f5:98:4c:12:13:29:48:37:1e:d9:d1:a9:83:
            97:76:b0:36:08:b2:42:2c:7d:29:50:1e:8b:e1:ed:
            d5:ce:bd:06:94:cd:cc:ba:f7:2f:03:93:57:f3:33:
            eb:37:6c:56:7c:38:a1:4d:90:43:0a:81:c8:55:3e:
            12:e2:d4:5f:fe:20:de:17:48:67:6c:e1:26:ab:1e:
            93:7f:47:8f:3d:8b:bd:f6:6b:d5:41:a4:cd:a5:1e:
            4b:c0:47:ee:a8:94:fe:37:42:82:75:7d:6c:1d:cb:
            07:bf:47:01:e5:8c:ee:c6:28:14:a0:0b:56:e5:2e:
            68:19:5d:73:8f:e5:e6:d5:30:eb:b5:77:cb:fa:21:
            bb:f5:b5:5a:1d:d1:62:9b:df:18:d3:1c:7a:10:14:
            77:25:ef:2b:a0:d3:47:11:48:0a:18:19:20:64:ea:
            52:bb:df:09:6f:c6:c9:e4:5b:f2:a2:26:f1:84:e5:
            88:06:94:12:8f:e8:c5:63:f3:98:42:2b:57:51:ea:
            1f:2c:6f:eb:bb:86:43:b3:d4:1b:a4:cd:e8:e8:54:
            bf:8f:37:f3:a1:50:96:09:32:49:bc:49:41:cb:4b:
            ad:87:51:08:ef:ff:99:8b:bb:43:aa:5e:de:48:f8:
            12:ca:52:5f:ba:54:3b:2d:62:49:80:90:f5:ec:35:
            e5:3e:d0:81:9c:b3:99:ae:5e:f9:a3:1c:5e:6c:25:
            29:c6:8d:d7:3e:c9:45:7d:fe:83:a4:ca:33:ad:62:
            55:24:36:84:3c:8c:42:7e:d2:37:c5:47:d6:bf:f1:
            22:33:0f
        Q:
            00:c3:f7:6c:88:25:d0:86:af:8a:f4:11:fa:84:b6:
            e7:24:1e:52:17:c7:32:67:77:8f:2e:6e:5a:d4:d1:
            82:b6:2d
        G:
            00:a0:7f:db:cc:7f:b4:38:d9:5a:d6:ce:b3:6e:55:
            01:91:61:75:fe:24:43:92:7e:0e:5b:ce:c6:fd:06:
            56:9f:0d:f9:db:b5:ef:ee:52:6e:54:67:14:41:f4:
            fe:3d:12:bd:71:fe:4c:7b:50:db:af:b5:ed:b6:e9:
            f7:14:b9:13:c1:8f:29:8a:58:3f:db:46:67:30:e5:
            8a:b1:5c:b5:ee:f2:8b:23:0c:27:4c:b7:dd:1d:2c:
            2f:ab:7b:f5:fa:f6:b0:9c:b4:26:b8:8a:3a:e2:15:
            1f:a4:44:78:3b:29:d7:92:6a:1e:10:8a:e6:bc:de:
            5e:1c:7b:30:5a:48:2d:00:62:7d:73:cd:d7:ba:b0:
            c8:f0:10:24:1b:f4:8a:69:49:4a:8d:74:4c:f8:3d:
            dc:56:02:74:18:2b:4c:79:dc:4f:7c:02:80:1a:3a:
            04:24:78:46:d0:26:f9:7f:66:70:4e:6e:8f:2b:0b:
            06:e8:ea:17:e3:b5:ce:21:1d:cd:05:f7:e2:72:7b:
            7a:b9:a6:3b:37:e5:33:8f:82:30:6f:21:eb:86:b4:
            35:fe:d2:76:15:0a:ea:7a:47:97:40:78:2e:da:9a:
            df:4c:e8:02:67:71:60:af:b9:ef:f7:60:ee:ff:fa:
            a3:f8:18:95:af:c3:3a:55:d1:90:bc:78:1f:74:da:
            b0:44:2b:51:b6:1e:d5:57:b7:1e:16:48:63:4d:c0:
            35:83:c1:83:02:4f:f7:3a:57:6e:88:a8:cd:36:e5:
            2d:97:cd:fa:09:22:6f:1e:de:b6:be:1a:d7:cb:48:
            67:cf:a9:67:4e:43:36:5a:d1:23:99:42:9c:8b:1d:
            be:e5:c1:13:41:33:54:90:93:13:52:53:66:1e:29:
            b3:8e:a9:6b:ce:58:42:7c:a7:e8:38:41:fc:a7:72:
            dd:70:83:c9:cc:19:82:e1:00:bf:3b:4d:5b:33:74:
            5a:a4:75:61:d0:3b:62:92:ac:2a:5e:41:3c:23:f2:
            79:d3:c0:7d:d7:f9:27:fe:fb:a6:37:37:14:0e:be:
            e8:e6:67:e4:26:81:01:fd:e0:33:67:34:6b:12:38:
            e1:dd:83:8c:cc:8e:bd:d0:fd:91:36:85:da:ea:62:
            85:5b:36:73:38:56:76:ad:11:f1:fa:56:39:a5:63:
            91:8d:a2:d8:93:69:80:e3:ee:1f:f7:81:d4:01:d4:
            33:79:4c:c4:dc:a3:ec:d1:54:7b:7a:5d:75:8a:0c:
            e5:96:7a:12:13:80:ab:07:5b:5a:80:ff:c3:eb:ce:
            f1:fa:d1:cd:89:3f:74:bb:fd:71:ac:14:f3:1d:91:
            6d:c0:63:e5:1f:02:c1:bd:ba:2e:ca:72:e5:8a:f4:
            53:a8:9a
X509v3 extensions:
    X509v3 Subject Key Identifier:
        0F:C4:5E:94:25:23:D5:90:E1:B1:B7:11:60:2A:EA:B1:92:BB:08:95
```

```
            X509v3 Authority Key Identifier:
                keyid:7C:ED:0A:E8:55:B7:D2:7C:05:20:E4:7F:A7:2E:40:32:2F:30:55:07

            X509v3 Basic Constraints: critical
                CA:TRUE, pathlen:2
            X509v3 Key Usage: critical
                Digital Signature, Certificate Sign, CRL Sign
    Signature Algorithm: sha512WithRSAEncryption
         1a:6a:97:7c:02:2f:4a:91:5c:7c:50:6d:19:4e:52:db:8f:ab:
         c8:f0:f2:e6:a1:c3:38:a0:94:a2:8d:e6:3d:c5:d5:2a:0e:9b:
         ad:8a:6e:14:11:f9:fc:4d:14:ca:a5:35:3a:0a:27:e0:64:72:
         46:d8:9a:cc:20:34:55:c6:91:e7:47:7a:cf:c8:e6:c5:8c:f8:
         8a:08:6e:71:92:d0:1d:1a:f2:ef:c7:78:3d:e8:6b:c8:ce:32:
         7b:b5:49:ba:b7:9e:b7:dc:68:c8:4c:d3:f9:b7:d8:35:26:dd:
         4d:31:91:54:2a:58:25:77:b0:d7:b3:23:32:ba:83:ee:4b:05:
         27:ee:e3:18:4b:51:a6:da:2c:34:91:4d:2f:86:71:5c:66:2e:
         15:f8:0c:f3:1a:98:e5:9f:8b:6f:bf:bc:3f:1b:4b:00:29:7c:
         72:6e:d6:5f:e8:10:11:ff:dc:4b:e7:4a:e5:0e:72:08:b9:fb:
         5e:bf:2e:83:80:e0:96:ad:ad:bc:47:a3:03:7e:4b:2e:27:c7:
         1c:b7:00:12:f1:db:d7:62:00:19:9d:3e:9b:12:bb:a8:b1:2a:
         21:4e:d8:2d:d7:9a:cb:71:0b:0e:db:57:ad:f1:ce:6a:22:81:
         af:8b:7c:70:0d:39:be:d5:3f:b8:2c:81:e6:41:c9:7c:db:00:
         de:06:09:50:e7:f9:5d:ee:14:ca:14:f6:6b:c2:0f:6a:de:ce:
         19:44:99:39:42:62:7d:d2:9d:95:aa:c0:91:a3:37:81:fb:f1:
         e3:1f:96:9a:d7:42:ad:9e:48:a3:84:49:2b:15:4e:4d:ba:80:
         c9:b7:5b:ea:ef:25:75:e1:4f:49:a5:1b:e3:95:74:36:99:a5:
         81:4c:0b:37:e7:87:42:c5:55:e6:4a:20:0e:d2:7d:b2:2d:33:
         39:68:67:07:e5:90:db:9b:e0:60:c4:55:ed:4c:5c:0e:f5:20:
         97:d5:53:55:12:25:b9:cd:af:3f:44:f6:cf:b8:27:00:d2:bd:
         e6:49:32:67:c9:48:f8:88:46:de:23:f5:bb:77:d9:ba:cc:de:
         75:2c:90:66:a1:13:5f:3b:fd:b3:ba:c9:7a:64:d8:7c:56:89:
         f5:d4:7a:b1:63:6f:09:5d:eb:76:f0:8e:27:28:4b:b0:70:aa:
         09:06:b0:36:e0:42:6c:b0:64:6f:7b:84:b4:cb:7d:e6:45:0b:
         23:34:00:48:ca:a9:b5:43:bd:7b:aa:71:c6:48:9f:79:39:be:
         a3:cb:f4:ca:a3:93:41:f4:7e:b3:78:58:86:40:32:60:2b:49:
         c8:b2:41:b9:c5:b6:49:39:06:db:10:6f:8f:be:56:ad:67:66:
         f1:c4:3b:3a:22:ad:2f:7a
```

See that for the keylength of RootCA2 sha512 was used by default. Note that the serial number of the certificate issued by RootCA2 to IntCA2.1 is the incremented number that was randomly chosen for the self-signed RootCA2 certificate.

1.10 IntCA2.2 certificate

Execute the following sequence of commands:

```
cd intermediate2
openssl dsaparam -out private/dsaparam.pem 3072
openssl dsaparam -in private/dsaparam.pem -noout -text
openssl gendsa -out ./private/IntermediateCa2_2_PrivateKey.pem ./private/dsaparam.pem
chmod 400 ./private/IntermediateCa2_2_PrivateKey.pem

openssl req -config openssl.cnf -new -sha256 -key ./private/IntermediateCa2_2_PrivateKey.pem \
        -out ./csr/IntermediateCa2_2_Csr.pem
openssl req -text -noout -verify -in ./csr/IntermediateCa2_2_Csr.pem


verify OK
Certificate Request:
    Data:
        Version: 1 (0x0)
        Subject: C = PL, ST = Lower Silesian Voivodeship, L = Wroclaw, O = Wroclaw University of Science and Technology,
        OU = WPPT/K2, CN = System Security I - IntermediateCA2.2, emailAddress = przemyslaw.kubiak@pwr.edu.pl
        Subject Public Key Info:
            Public Key Algorithm: dsaEncryption
                pub:
                    0a:77:dd:a2:be:66:21:d8:6f:70:04:80:11:8a:f9:
                    0e:52:39:43:52:82:01:cd:ce:9f:3d:09:6c:d0:bb:
                    f6:c4:90:94:39:21:46:e5:e3:5c:9c:52:2e:13:9c:
                    46:7c:13:6a:a3:75:d4:6d:92:ba:81:06:2c:a5:27:
                    63:7b:cc:47:d7:86:99:d2:09:19:79:e4:4a:54:e2:
                    a5:63:01:a0:74:9b:c4:08:bf:95:8f:38:2f:9b:8e:
                    c5:d6:ed:61:d7:74:03:18:96:03:c3:ce:6e:6d:0d:
                    22:8d:d2:c8:ac:b0:86:21:dd:2c:ed:29:a0:6a:8b:
                    b3:e7:e0:e8:47:78:8c:97:17:04:30:60:27:f0:9f:
                    e6:51:f1:4d:44:5b:cf:63:8f:83:7c:0a:4c:f3:34:
                    da:67:4f:fc:6d:34:c4:04:e3:18:57:51:50:20:4c:
                    4e:a4:83:8b:5a:38:bf:5e:db:5f:ab:d0:e9:65:78:
                    8a:7b:68:2d:6e:aa:b8:fe:c4:1d:af:2c:31:9a:12:
                    04:09:88:ad:90:fc:f9:45:4b:34:9a:aa:ea:76:2a:
                    0f:bb:45:0c:76:29:57:c4:9d:df:22:92:44:c9:1f:
                    2b:4a:81:19:07:ff:26:95:ed:ef:8f:7e:41:67:9b:
                    9c:13:36:10:ce:63:57:c6:64:d6:7e:b8:f9:c8:d6:
                    1a:14:57:75:30:b0:53:df:b4:33:eb:8f:24:31:a0:
                    cf:27:31:92:e1:ca:3d:18:d1:6e:f3:8c:ad:49:81:
                    2a:1f:0a:c1:02:32:26:61:f7:18:71:86:2e:18:46:
```

```
                    04:c4:ed:e8:09:c6:59:be:21:a4:0b:62:19:ad:03:
                    a2:9b:5e:de:94:9b:4e:15:e1:ad:77:6f:77:99:48:
                    db:c4:db:9d:f6:22:f7:17:7f:c9:a7:2a:e9:3e:77:
                    b3:12:ea:24:82:cf:c2:be:4f:91:29:d3:98:fb:bc:
                    ef:82:17:1d:d0:a2:2d:bd:b5:14:da:47:ac:17:a5:
                    b9:ca:a4:df:f7:17:b9:50:74
                P:
                    00:ad:7c:19:14:79:11:21:6a:b0:bc:7c:86:e4:70:
                    0e:35:c0:7d:8f:1c:de:58:a5:6c:ae:3e:ce:67:b0:
                    31:bd:41:58:14:64:78:d2:fe:59:10:c2:75:fb:ad:
                    42:df:d1:8e:69:b2:08:9d:c5:cf:0e:d3:46:e9:f7:
                    d6:d5:65:29:07:d8:fd:91:9f:b9:ec:55:4f:57:23:
                    ab:20:a6:01:31:eb:c8:a9:5b:bb:d2:43:5c:bc:29:
                    ed:cd:44:8f:09:8e:c6:ab:5b:d2:cf:9c:51:c3:32:
                    be:da:f0:f8:c4:78:10:8f:32:3f:d2:5f:33:90:c6:
                    92:5a:b9:36:25:ed:ad:f1:27:c9:cc:82:59:6d:87:
                    ac:ca:4e:bd:64:76:f4:da:31:1d:5d:bb:94:21:21:
                    2a:bd:d8:99:16:12:6a:bf:f8:a1:9d:75:0e:75:40:
                    fa:28:00:ad:ab:95:24:f4:4d:34:7f:ae:d8:72:09:
                    56:9c:4c:12:3c:58:4a:91:3a:8f:6b:b2:76:17:e7:
                    1d:77:8e:f7:75:7c:c2:c3:93:da:2b:c9:03:38:bf:
                    13:df:2a:9e:31:c5:14:b4:ce:0c:5a:cc:99:64:5a:
                    9c:0a:35:a4:26:44:51:69:ae:17:34:0a:27:0b:83:
                    96:91:b4:1d:d1:42:78:a2:ef:e9:fd:39:7d:05:ae:
                    b4:fb:9a:4c:58:48:9d:48:ce:6a:b0:04:ce:c3:3d:
                    cf:b3:3e:d5:9c:60:d1:e3:03:dd:0d:4b:95:e4:88:
                    0a:dd:a9:14:37:83:c7:e6:f6:b0:b1:a2:3d:7a:a3:
                    4a:c1:09:bb:3e:da:85:49:16:c8:9e:b4:3b:d0:fb:
                    14:f9:9b:7a:1f:4d:84:12:fe:23:bd:99:82:17:cc:
                    bf:ca:24:72:6b:c8:5a:f5:e1:5b:1d:84:81:d2:37:
                    1e:37:ee:a5:fa:76:3c:1c:0b:bd:ae:b0:80:18:50:
                    49:a7:4e:b6:9e:3c:40:0e:2b:5b:26:55:b1:e1:7e:
                    f3:8a:5c:05:64:9d:7c:48:53:13
                Q:
                    00:8a:35:6f:81:9e:b8:5a:8e:97:2c:46:14:a6:2d:
                    3d:b3:25:06:a7:ef:fc:f0:3d:fe:ab:bf:e3:5b:17:
                    96:54:af
                G:
                    50:2c:ea:99:83:7f:04:45:71:8f:76:a2:1f:af:65:
                    c7:db:d4:67:18:bb:ff:77:3b:15:05:b0:5b:07:17:
                    82:43:7f:2b:9e:59:94:54:c2:69:9e:8f:73:91:53:
                    71:a4:c0:ff:bb:00:ff:02:5e:48:b9:79:0f:53:7a:
                    11:5c:b0:16:43:02:77:be:05:e9:6d:2e:74:d5:93:
                    f1:e8:fc:cc:82:23:59:bf:ea:6c:e5:e2:76:b9:6c:
                    0d:a8:92:4b:a0:33:ac:51:09:b3:bb:8f:e4:b6:ee:
                    e6:d3:52:29:84:6a:5c:c3:a5:ca:f9:90:0d:c1:c6:
                    3f:ea:3b:72:b5:e6:f2:dc:88:b4:8e:27:ef:dd:22:
                    1c:d7:6b:c5:e1:1a:74:2e:e3:c1:ab:4a:03:3c:e0:
                    4e:99:be:51:2e:a3:36:ad:67:c7:83:31:19:7a:49:
                    82:b1:14:b8:e3:e9:c3:bd:3e:0a:69:25:2e:36:9c:
                    9c:0c:5c:d3:c0:9b:63:c8:d7:40:cd:5f:89:31:7c:
                    f5:ef:c9:e1:e5:58:e0:73:fc:3c:eb:96:d8:58:32:
                    34:57:9c:97:ce:b2:6f:27:fd:95:65:fb:75:ac:60:
                    79:38:7f:ef:d8:b2:84:31:23:b7:6d:76:af:a4:0a:
                    db:95:a7:13:b6:d2:82:cf:3e:2f:21:dc:44:a1:83:
                    7a:fd:00:d3:58:ec:99:5d:0a:02:ea:9a:10:4e:8b:
                    c5:8b:05:98:85:60:99:ce:b7:51:8a:55:85:da:8a:
                    e0:88:67:aa:5b:2e:19:09:83:a7:fd:5c:27:44:b9:
                    93:51:5c:53:e9:4b:fe:c3:00:2c:33:5f:a5:4a:7f:
                    16:7a:5b:de:f3:2f:44:78:f2:22:7f:57:87:53:be:
                    8f:54:86:af:1d:a7:b6:d6:73:b4:0c:92:e8:8b:0e:
                    d4:df:be:a6:4e:0f:8a:78:60:e4:25:4c:49:28:39:
                    df:e4:ba:2b:e9:58:2c:48:6e:d9:11:94:12:23:2b:
                    e5:4a:3f:c3:74:08:70:33:56
        Attributes:
            a0:00
    Signature Algorithm: dsa_with_SHA256
         r:
             13:71:3e:9e:da:e2:1b:3c:b4:22:83:f5:4c:bd:50:
             7f:14:6b:bb:85:b5:d3:89:a0:97:0a:52:6f:0c:d0:
             f0:62
         s:
             30:5b:bd:e6:79:dd:82:ff:b1:4f:21:75:b6:41:2e:
             e9:4a:2d:d4:e2:8b:a6:98:3e:39:64:34:99:94:6b:
             c5:62


 cd ..
 touch index.txt
 openssl ca -create_serial -config openssl.cnf -extensions v3_intermediate_ca  -notext -days 3640 \
       -in intermediate2/csr/IntermediateCa2_2_Csr.pem  -out intermediate2/certs/IntermediateCa2_2_Cert.pem

Using configuration from openssl.cnf
Check that the request matches the signature
Signature ok
Certificate Details:
        Serial Number:
            4b:53:97:41:3b:ff:5c:01:d7:dd:87:bf:2b:cf:c0:f5:80:1f:92:be
        Validity
            Not Before: May 17 21:20:31 2019 GMT
            Not After : May  4 21:20:31 2029 GMT
        Subject:
            countryName               = PL
```

```
            stateOrProvinceName       = Lower Silesian Voivodeship
            localityName              = Wroclaw
            organizationName          = Wroclaw University of Science and Technology
            organizationalUnitName    = WPPT/K2
            commonName                = System Security I - IntermediateCA2.2
            emailAddress              = przemyslaw.kubiak@pwr.edu.pl
        X509v3 extensions:
            X509v3 Subject Key Identifier:
                79:67:E1:81:9E:B6:A9:71:E4:2C:A8:9C:A2:4D:06:F0:61:8A:9A:9D
            X509v3 Authority Key Identifier:
                keyid:0F:C4:5E:94:25:23:D5:90:E1:B1:B7:11:60:2A:EA:B1:92:BB:08:95

            X509v3 Basic Constraints: critical
                CA:TRUE, pathlen:1
            X509v3 Key Usage: critical
                Digital Signature, Certificate Sign, CRL Sign
Certificate is to be certified until May  4 21:20:31 2029 GMT (3640 days)
Sign the certificate? [y/n]:y
139868479132736:error:0D0DC0C6:asn1 encoding routines:ASN1_item_sign_ctx:digest and key type
not supported:../crypto/asn1/a_sign.c:185:
```

1.11 Revocation of IntCA2.2 certificate

Perhaps the problem lies in the not supported length of DSA key of IntCA2.1. (The hypothesis will turn to be wrong – you may jump directly to the Subsect.1.12.1 for the proper solution, but revocation could be a good task for the students, they may also try -verbose option for the openssl ca command above)
So what should we do?

- revoke the certificate of IntCA2.1 (on the side of RootCA2),
- generate a new keypair (a shorter one) and a new certificate request
- generate a new cert for IntCA2.1 by RootCA2.

Lets start with revocation of the certificate of IntCA2.1:

```
cd root2/ca/
echo 1A5D > crlnumber    # 0x1A5D = 6749
openssl ca -config ./openssl.cnf -gencrl -out crl/RootCa2.crl.pem   #create_serial option does not work for CRLs
openssl crl -in crl/RootCa2.crl.pem -noout -text


Certificate Revocation List (CRL):
        Version 2 (0x1)
        Signature Algorithm: sha512WithRSAEncryption
        Issuer: C = PL, ST = Lower Silesian Voivodeship, O = Wroclaw University of Science and Technology,
        OU = WPPT/K2, CN = System Security I - RootCA2, emailAddress = przemyslaw.kubiak@pwr.edu.pl
        Last Update: May 21 14:34:28 2019 GMT
        Next Update: May 18 14:34:28 2029 GMT
        CRL extensions:
            X509v3 Authority Key Identifier:
                keyid:7C:ED:0A:E8:55:B7:D2:7C:05:20:E4:7F:A7:2E:40:32:2F:30:55:07

            X509v3 CRL Number:
                6749
No Revoked Certificates.
    Signature Algorithm: sha512WithRSAEncryption
         30:38:12:72:99:3a:19:58:ed:ce:eb:c1:ac:d8:fe:4f:b8:30:
         84:4e:42:e9:2d:2d:8b:cf:75:f6:50:13:32:e6:c9:ed:66:2a:
         2a:5e:aa:6e:7e:98:11:e2:29:5e:32:b6:0d:c5:49:47:8f:e2:
         50:8f:35:a0:94:fe:3a:c2:1e:8a:be:fb:12:7d:05:62:d0:c3:
         07:3f:89:dd:fd:23:48:82:dd:1a:4c:d9:cf:af:66:e0:c8:a4:
         df:9b:c5:04:81:45:ea:74:e9:e2:5e:fa:76:54:71:3a:72:3b:
         72:fe:fa:6e:63:e3:1d:ec:83:52:fa:96:39:b0:c3:f6:65:b0:
         72:ba:e1:6f:43:6b:41:17:2a:dd:f2:67:8b:fd:fa:ad:65:77:
         4d:e0:59:e5:8a:23:ea:1c:31:3e:27:1a:e0:89:0e:96:cc:cf:
         6b:1c:6b:a7:10:2b:ce:55:a0:66:de:99:4a:25:01:bd:3b:d1:
         93:73:0a:ac:03:2d:a7:b9:d9:9a:fe:2b:78:d1:57:cb:98:30:
         e6:e8:a6:3a:e1:df:6e:c5:2f:68:73:c2:67:32:0f:14:a9:b3:
         45:6c:cf:df:2f:c6:3c:ed:db:f9:9d:6b:91:1b:62:81:55:34:
         6b:50:40:97:9e:cd:81:bb:52:13:07:8d:e1:f1:c6:5f:e7:8e:
         b6:cb:a4:4f:ce:c5:f3:40:32:58:04:cf:61:9b:e8:32:74:45:
         fc:26:f7:8a:f7:64:ae:cd:29:cb:0e:57:13:d2:b3:e3:8b:95:
         54:3e:4a:6f:60:e8:73:15:43:11:b5:aa:af:a5:d3:13:48:a9:
         5e:ea:32:0f:37:50:10:38:34:be:4f:fd:68:e3:78:9d:12:a8:
         15:61:6b:d4:cc:9a:9f:1e:ed:fc:23:ad:c0:bf:d8:be:61:a0:
         fd:cb:46:4c:1e:4d:e8:80:0b:5a:b2:71:15:9b:62:e2:89:cd:
         6c:39:a3:95:73:d1:11:d1:e3:9d:f4:59:ba:a9:c9:f3:ad:1c:
         c7:28:b7:88:9c:22:ce:7a:83:55:b5:02:b6:e8:39:e4:1a:20:
         e4:e1:db:8e:dc:53:86:51:4c:ce:36:7f:8b:0b:8b:06:53:ce:
         ea:07:f6:dd:5d:57:ff:0c:f0:ea:2a:75:d4:63:32:53:80:65:
```

```
         68:15:9e:46:ae:28:20:da:8a:39:b4:a4:46:4a:e0:8c:5d:27:
         b3:4c:e7:e4:0e:04:e3:2f:17:1e:c9:58:23:20:2c:75:39:0e:
         75:03:82:59:35:b6:0c:42:69:96:cd:2f:fb:92:06:19:4e:e1:
         3a:e2:98:ee:18:89:db:00:40:0b:80:4e:da:20:ac:d7:75:57:
         d4:3c:2c:fd:2b:02:cb:1a
```

Now we must revoke the certificate of IntCA2.1:

```
openssl ca -config ./openssl.cnf  -revoke ./intermediate1/certs/IntermediateCa2_1_Cert.pem
openssl ca -config ./openssl.cnf -gencrl -out crl/RootCa2.crl.pem
openssl crl -in crl/RootCa2.crl.pem -noout -text

Certificate Revocation List (CRL):
        Version 2 (0x1)
        Signature Algorithm: sha512WithRSAEncryption
        Issuer: C = PL, ST = Lower Silesian Voivodeship, O = Wroclaw University of Science and Technology,
        OU = WPPT/K2, CN = System Security I - RootCA2, emailAddress = przemyslaw.kubiak@pwr.edu.pl
        Last Update: May 21 14:45:50 2019 GMT
        Next Update: May 18 14:45:50 2029 GMT
        CRL extensions:
            X509v3 Authority Key Identifier:
                keyid:7C:ED:0A:E8:55:B7:D2:7C:05:20:E4:7F:A7:2E:40:32:2F:30:55:07

            X509v3 CRL Number:
                6750
Revoked Certificates:
    Serial Number: 2EE1FE6C6154A6FB524F70A9466182BA4718E05C
        Revocation Date: May 21 14:44:08 2019 GMT
    Signature Algorithm: sha512WithRSAEncryption
         31:91:1c:ff:e6:b8:7d:c1:e3:a0:70:ce:46:9b:57:08:49:89:
         5a:96:33:2c:75:dd:a0:07:1e:4e:de:0c:5a:e7:ef:c5:51:97:
         92:ef:4a:38:39:50:64:80:47:e7:7c:d7:9d:5b:2c:69:83:8b:
         17:df:93:d3:a6:f3:b9:7d:ee:4c:1b:c5:1d:87:09:19:36:9c:
         bc:66:0b:7c:d2:c7:3b:2e:eb:f3:38:bb:11:66:cc:03:3e:ab:
         71:e4:ba:be:ff:13:81:00:93:d5:b7:ab:d8:71:08:9a:1b:86:
         a8:37:c4:ff:f7:6e:69:e1:f4:09:68:f9:41:39:28:11:c2:c5:
         7b:7e:73:e6:ac:60:c9:b7:b3:f2:74:03:ff:e3:09:2d:f3:42:
         b6:7f:a2:ce:71:08:be:ad:88:0a:18:5f:41:77:ab:3f:6d:04:
         cc:2f:72:fd:26:2c:c6:86:46:96:d5:c4:97:85:69:2d:63:d5:
         4e:58:20:f3:57:43:04:e0:5e:47:f1:27:43:6b:bf:a6:52:d4:
         aa:48:8f:3e:d3:00:52:40:55:6b:09:e7:69:49:ee:84:36:57:
         1b:76:e0:04:fe:ee:9b:56:d3:dd:f6:b7:af:ae:37:6c:e9:59:
         ff:c5:f6:4e:9e:86:57:19:eb:f2:1e:35:ef:95:b4:64:6c:b9:
         b4:7b:3a:98:63:40:dd:66:4c:a4:7e:ed:12:2f:7a:47:cd:2c:
         d4:18:c3:c9:b5:cb:f6:a9:de:65:2b:41:0b:48:2c:68:aa:49:
         f6:b6:5e:26:dd:11:7b:a2:18:30:c9:e9:0b:45:89:0b:43:9e:
         fb:a7:a1:73:6f:f5:f8:ae:f9:54:9c:af:8e:97:eb:3e:16:e9:
         f4:da:91:7a:40:1d:98:33:37:4c:88:19:ed:2b:93:c7:01:fa:
         41:7f:c2:04:df:fb:2b:64:45:df:ba:a8:71:25:8b:a5:a3:37:
         bf:b8:15:19:a8:02:ac:c1:ca:bf:bb:0c:2f:8e:15:99:14:6b:
         48:28:2b:ad:f2:94:09:96:f7:3b:61:a0:1f:54:a1:e1:13:fc:
         2c:81:b1:d2:19:b7:6a:bd:eb:9f:fc:31:dc:4f:02:3c:f2:05:
         63:58:f1:03:35:a5:a4:fd:62:71:17:06:11:5b:66:5a:d7:a0:
         66:3b:94:e9:47:94:73:51:3c:f6:1b:2e:f3:7c:69:d5:1f:be:
         47:c4:42:5c:80:ff:30:53:09:84:0d:ac:52:d7:05:c6:3f:40:
         ea:60:10:1c:f7:42:11:fa:4c:8f:9e:23:03:4b:22:ab:58:84:
         87:f6:6d:61:ef:71:ba:dc:21:f8:54:66:01:31:27:7c:95:d6:
         c7:ff:61:7a:f3:b3:50:ca
```

1.12 Generate the second certificate of IntCA2.2

Now its time to generate a new keypair (a shorter one) and a new certificate request.

```
cd intermediate1
openssl dsaparam -out private/dsaparam.pem 3072
openssl dsaparam -in private/dsaparam.pem -noout -text

    P:
        00:f4:6d:1b:ed:21:f4:0d:50:88:c7:66:a1:46:d7:
        75:f7:b8:46:37:e2:cf:f9:83:50:a9:5e:53:6b:31:
        29:0d:60:4b:50:3f:ee:09:45:dc:6a:f2:88:ef:05:
        ee:24:d9:ed:de:95:d5:4a:f1:cb:1a:90:5e:67:42:
        4c:dc:eb:60:f1:fd:ea:43:d0:38:af:4a:96:d7:73:
        13:f1:50:05:63:31:bd:70:8e:7b:f0:f5:e0:a8:04:
        83:40:0f:e5:df:d6:03:66:3f:2d:64:5c:de:0d:74:
        47:76:dd:91:71:ef:c0:e2:56:ef:2a:aa:7a:d7:52:
        9d:d2:b1:d5:b3:27:d7:21:88:b9:e2:50:dd:a6:31:
        6b:54:a3:0a:2f:f5:ae:cc:6c:bc:19:62:b5:64:96:
        40:e0:69:93:1f:cf:28:33:d0:76:b5:88:c0:69:f7:
        88:b7:d1:ed:c3:e9:72:b7:40:b0:0d:a9:07:36:26:
        ef:27:51:81:28:39:24:2f:83:47:88:8d:a9:97:33:
        3d:c3:2e:1d:da:ae:b9:09:4a:5f:ac:92:5f:7c:96:
        da:9c:cf:d1:c8:84:4a:35:c5:eb:81:b6:23:1e:fd:
        1e:13:5d:ae:9d:8e:2a:e2:09:d9:76:94:4f:ae:8c:
        a4:ea:91:f6:c1:f2:9f:98:28:38:6c:a7:6e:0c:f6:
        4f:fb:29:be:c4:f8:7c:d2:f8:36:fa:32:a9:f1:91:
        68:47:2f:6c:39:d0:0d:85:ac:9b:ba:3d:e1:e8:42:
```

```
        06:03:2d:2e:19:ce:4f:42:ca:4f:ad:e8:1f:39:31:
        78:55:5a:e8:91:1c:ef:24:e7:fe:26:9c:4c:45:39:
        83:81:9f:f1:a8:17:f1:b9:7f:f6:5c:23:42:f2:57:
        83:88:b1:dc:89:12:c6:da:e9:6a:93:5a:7b:c0:8f:
        f2:13:e3:c5:83:e3:65:54:94:8c:71:fb:2c:ae:76:
        56:d0:f2:3a:fa:45:fe:c8:a7:e4:86:a6:48:13:ce:
        29:d1:42:f1:5e:3a:fc:50:0a:43
    Q:
        00:c4:94:1f:42:71:58:0b:02:36:87:6f:86:c8:e8:
        f2:b8:83:fa:69:f3:3b:47:5d:cb:7f:13:92:a8:29:
        2e:df:11
    G:
        41:83:c7:66:fc:71:d9:a6:f8:8c:ee:ff:83:dc:f2:
        87:e4:3c:00:6b:6d:73:13:aa:13:bb:ce:c2:23:6e:
        5b:e4:4a:7a:5b:f0:45:01:ea:76:29:9c:68:7c:3c:
        6f:15:f6:97:d9:da:52:71:f1:4c:2e:1f:fe:aa:d9:
        bf:16:52:e1:73:fc:e8:53:bb:8c:5d:a5:02:4f:f0:
        94:81:ac:dd:83:24:e6:f3:c8:02:52:34:4f:73:1b:
        cb:52:20:78:9d:86:61:a7:ea:d9:60:6d:21:00:9f:
        42:3f:60:03:8f:00:77:fc:72:5f:cb:4d:c3:0c:ac:
        73:de:e0:01:a6:67:87:ed:0f:e1:83:80:4e:1f:a0:
        c9:5f:62:c4:3b:67:80:42:c1:ed:a8:0d:e4:f1:4a:
        97:42:24:b6:3a:6f:c5:27:1e:a4:7b:bf:d5:b7:04:
        a3:24:17:d8:42:b6:30:c9:b0:c4:3d:9d:77:f9:62:
        47:6f:01:e9:df:8b:f4:34:06:7c:28:53:b1:d1:9f:
        6f:b4:00:a9:aa:b1:e6:d7:ec:d8:86:b9:a7:d3:43:
        74:a8:a0:6c:91:8e:d4:c0:58:9d:eb:94:60:61:c8:
        be:88:55:16:40:15:97:8d:4e:a9:2e:d1:ed:6c:11:
        0b:b1:81:35:69:e7:20:24:d4:55:13:21:0f:af:0a:
        d4:4f:f4:c2:74:d0:c9:60:08:2e:95:96:3c:da:89:
        cd:b7:42:51:13:d5:78:cb:72:e4:39:74:49:f7:38:
        07:5a:94:ea:69:65:cf:7d:cb:2e:61:11:4d:42:01:
        09:3c:95:05:a3:af:19:a0:31:fc:79:2a:c8:d9:70:
        15:03:56:1c:92:b4:28:30:66:0d:c4:4a:8e:84:29:
        57:20:67:04:24:ea:3a:51:22:c7:05:45:36:a1:42:
        ca:34:47:b9:9d:22:b2:2f:25:c6:49:27:21:8f:b5:
        74:8d:10:0a:49:78:a7:bd:d9:cc:f4:55:be:77:dd:
        f4:57:00:0a:17:a2:f2:5d:f8

openssl gendsa -out ./private/IntermediateCa2_1_PrivateKey2.pem ./private/dsaparam.pem
chmod 400 ./private/IntermediateCa2_1_PrivateKey2.pem
openssl req -config openssl.cnf -new -sha256 -key ./private/IntermediateCa2_1_PrivateKey2.pem \
    -out ./csr/IntermediateCa2_1_Csr2.pem
openssl req -text -noout -verify -in ./csr/IntermediateCa2_1_Csr2.pem
```

Change relevant lines in the ./openssl.cnf file (e.g., the private key path)
Finally generate a new cert for IntCA2.1 by RootCA2:

```
cd ..
openssl ca -config openssl.cnf -extensions v3_intermediate_ca  -notext -days 3645
   -in intermediate1/csr/IntermediateCa2_1_Csr2.pem  -out intermediate1/certs/IntermediateCa2_1_Cert2.pem \
openssl x509 -noout -text -in intermediate1/certs/IntermediateCa2_1_Cert2.pem

cd intermediate1
openssl ca -create_serial -config openssl.cnf -extensions v3_intermediate_ca  -notext -days 3640 \
   -in intermediate2/csr/IntermediateCa2_2_Csr.pem  -out intermediate2/certs/IntermediateCa2_2_Cert.pem

Using configuration from openssl.cnf
Check that the request matches the signature
Signature ok
Certificate Details:
        Serial Number:
            5b:e7:90:ae:28:2e:5a:61:00:d6:1d:de:6d:ff:5d:22:cf:fc:17:c2
        Validity
            Not Before: May 21 15:29:44 2019 GMT
            Not After : May  8 15:29:44 2029 GMT
        Subject:
            countryName               = PL
            stateOrProvinceName       = Lower Silesian Voivodeship
            localityName              = Wroclaw
            organizationName          = Wroclaw University of Science and Technology
            organizationalUnitName    = WPPT/K2
            commonName                = System Security I - IntermediateCA2.2
            emailAddress              = przemyslaw.kubiak@pwr.edu.pl
        X509v3 extensions:
            X509v3 Subject Key Identifier:
                79:67:E1:81:9E:B6:A9:71:E4:2C:A8:9C:A2:4D:06:F0:61:8A:9A:9D
            X509v3 Authority Key Identifier:
                keyid:70:C1:B0:91:47:E2:76:37:33:34:64:FB:1B:72:02:B7:8F:94:AF:66

            X509v3 Basic Constraints: critical
                CA:TRUE, pathlen:1
            X509v3 Key Usage: critical
                Digital Signature, Certificate Sign, CRL Sign
Certificate is to be certified until May  8 15:29:44 2029 GMT (3640 days)
Sign the certificate? [y/n]:y
140299443266624:error:0D0DC0C6:asn1 encoding routines:ASN1_item_sign_ctx:digest and key type
not supported:../crypto/asn1/a_sign.c:185:
```

1.12.1 Use a non-default hash function

The same problem... So perhaps the default message digest function is applied which is too short...
Add -md sha256 option...

```
openssl ca -create_serial -md sha256 -config openssl.cnf -extensions v3_intermediate_ca \
  -notext -days 3640 -in intermediate2/csr/IntermediateCa2_2_Csr.pem  -out intermediate2/certs/IntermediateCa2_2_Cert.pem
Using configuration from openssl.cnf
Check that the request matches the signature
Signature ok
Certificate Details:
        Serial Number:
            02:44:7b:43:16:30:34:cc:77:07:d2:ca:7b:f8:1e:27:8c:45:04:72
        Validity
            Not Before: May 21 15:54:06 2019 GMT
            Not After : May  8 15:54:06 2029 GMT
        Subject:
            countryName               = PL
            stateOrProvinceName       = Lower Silesian Voivodeship
            localityName              = Wroclaw
            organizationName          = Wroclaw University of Science and Technology
            organizationalUnitName    = WPPT/K2
            commonName                = System Security I - IntermediateCA2.2
            emailAddress              = przemyslaw.kubiak@pwr.edu.pl
        X509v3 extensions:
            X509v3 Subject Key Identifier:
                79:67:E1:81:9E:B6:A9:71:E4:2C:A8:9C:A2:4D:06:F0:61:8A:9A:9D
            X509v3 Authority Key Identifier:
                keyid:70:C1:B0:91:47:E2:76:37:33:34:64:FB:1B:72:02:B7:8F:94:AF:66

            X509v3 Basic Constraints: critical
                CA:TRUE, pathlen:1
            X509v3 Key Usage: critical
                Digital Signature, Certificate Sign, CRL Sign
Certificate is to be certified until May  8 15:54:06 2029 GMT (3640 days)
Sign the certificate? [y/n]:y


1 out of 1 certificate requests certified, commit? [y/n]y
Write out database with 1 new entries
Data Base Updated
```

1.13 The second certificate for IntCA3 public key

So two certificates will certify the same public key

```
cd intermediate2
touch index.txt
openssl ca -create_serial -md sha256 -config openssl.cnf -extensions v3_intermediate_ca  -notext \
    -days 3600 -in ../../../../root1/ca/intermediate1/intermediate2/intermediate3/csr/IntermediateCa1_3_Csr2.pem \
    -out ../../../../root1/ca/intermediate1/intermediate2/intermediate3/certs/IntermediateCa1_3_Cert2.pem
```

Now create the second certificate chain

```
cd root1/ca
cat intermediate1/intermediate2/intermediate3/certs/IntermediateCa1_3_Cert2.pem  \
    ../../root2/ca/intermediate1/intermediate2/certs/IntermediateCa2_2_Cert.pem \
    ../../root2/ca/intermediate1/certs/IntermediateCa2_1_Cert2.pem \
    ../../root2/ca/certs/RootCa2Cert.pem > intermediate1/intermediate2/intermediate3/certs/IntermediateCa1_3_Cert2_ChainToRoot2.pem

openssl verify -CAfile intermediate1/intermediate2/intermediate3/certs/IntermediateCa1_3_Cert2_ChainToRoot2.pem \
    intermediate1/intermediate2/intermediate3/endEntity/certs/EndEntity_Cert.pem

    intermediate1/intermediate2/intermediate3/endEntity/certs/EndEntity_Cert.pem: OK
```

Thus we have the second certificate chain for EndEntity_Cert.pem

Chapter 2

Tests of the default implementation of certificate verification functionality

Create openssl-working directory (aside to the root1, root2). Create subdirectories: client-certs, server-certs in the openssl-working directory:

```
mkdir client-certs server-certs
```

Copy all the certificates generated so far and not being revoked to the client-certs directory.

```
cd client-certs
ls -ll

-rw-rw-r-- 1 pkubiak pkubiak 1233 maj 12 21:07 EndEntity_Cert.pem
-rw-rw-r-- 1 pkubiak pkubiak 1257 maj 11 20:43 IntermediateCa1_1_Cert2.pem
-rw-rw-r-- 1 pkubiak pkubiak 1224 maj 11 23:38 IntermediateCa1_2_Cert.pem
-rw-rw-r-- 1 pkubiak pkubiak 1155 maj 21 18:12 IntermediateCa1_3_Cert2.pem
-rw-rw-r-- 1 pkubiak pkubiak 1172 maj 12 14:36 IntermediateCa1_3_Cert.pem
-rw-rw-r-- 1 pkubiak pkubiak 3268 maj 21 17:26 IntermediateCa2_1_Cert2.pem
-rw-rw-r-- 1 pkubiak pkubiak 2699 maj 21 17:54 IntermediateCa2_2_Cert.pem
-rw-rw-r-- 1 pkubiak pkubiak 1314 lut 22 19:53 RootCa1Cert.pem
-rw-rw-r-- 1 pkubiak pkubiak 2313 maj 21 18:28 RootCa2Cert.pem
```

Let evaluate openssl verify application. Let assume that untrusted is the whole chain from IntermediateCa1_3_Cert.pem to RootCa1Cert.pem inclusively, the trusted certificate is RootCa2Cert.pem and the certificate to be verified is EndEntity_Cert.pem. The verification should fail.

```
openssl verify -verbose -show_chain -check_ss_sig -untrusted IntermediateCa1_3_Cert_ChainToRoot1.pem \
    -trusted RootCa2Cert.pem EndEntity_Cert.pem
```

We got the following output:

```
C = PL, ST = Lower Silesian Voivodeship, L = Wroclaw, O = Wroclaw University of Science and Technology,
    OU = WPPT/K2, CN = System Security I - RootCA1, emailAddress = przemyslaw.kubiak@pwr.edu.pl
error 19 at 4 depth lookup: self signed certificate in certificate chain
error EndEntity_Cert.pem: verification failed
```

So try the following (we omit the check_ss_sig flag):

```
openssl verify -verbose -show_chain -untrusted IntermediateCa1_3_Cert_ChainToRoot1.pem \
    -trusted RootCa2Cert.pem EndEntity_Cert.pem
```

The output is:

```
C = PL, ST = Lower Silesian Voivodeship, L = Wroclaw, O = Wroclaw University of Science and Technology,
    OU = WPPT/K2, CN = System Security I - RootCA1, emailAddress = przemyslaw.kubiak@pwr.edu.pl
error 19 at 4 depth lookup: self signed certificate in certificate chain
error EndEntity_Cert.pem: verification failed
```

So use untrusted set without self-signed cert:

```
cat IntermediateCa1_3_Cert.pem IntermediateCa1_1_Cert2.pem IntermediateCa1_2_Cert.pem > untrusted1.pem
openssl verify -verbose -show_chain -check_ss_sig -untrusted untrusted1.pem -trusted RootCa2Cert.pem EndEntity_Cert.pem
```

We got the following output:

```
C = PL, ST = Lower Silesian Voivodeship, O = Wroclaw University of Science and Technology,
    OU = WPPT/K2, CN = System Security I - IntermediateCA1.1, emailAddress = przemyslaw.kubiak@pwr.edu.pl
error 20 at 3 depth lookup: unable to get local issuer certificate
error EndEntity_Cert.pem: verification failed
```

Check if trusted can also be a non-root certificate: e.g., a trusted certificate is the one issued by the trusted CA. Assume that we trust RootCA1, so the trusted certificate is IntermediateCa1_1_Cert2.pem

```
cat IntermediateCa1_3_Cert.pem IntermediateCa1_2_Cert.pem > untrusted1_no_IntCA1_1.pem
openssl verify -verbose -show_chain -check_ss_sig -untrusted untrusted1_no_IntCA1_1.pem \
    -trusted IntermediateCa1_1_Cert2.pem EndEntity_Cert.pem
```

The output is

```
C = PL, ST = Lower Silesian Voivodeship, O = Wroclaw University of Science and Technology,
    OU = WPPT/K2, CN = System Security I - IntermediateCA1.1, emailAddress = przemyslaw.kubiak@pwr.edu.pl
error 2 at 3 depth lookup: unable to get issuer certificate
error EndEntity_Cert.pem: verification failed
```

So try the following:

```
cat IntermediateCa1_1_Cert2.pem RootCa1Cert.pem > issued_by_Root1.pem
openssl verify -verbose -show_chain -check_ss_sig -untrusted untrusted1_no_IntCA1_1.pem \
    -trusted issued_by_Root1.pem EndEntity_Cert.pem
```

The output is:

```
EndEntity_Cert.pem: OK
Chain:
depth=0: C = PL, ST = Lower Silesian Voivodeship, L = Wroclaw, O = Wroclaw University of Science and Technology,
    OU = WPPT/K2, CN = localhost, emailAddress = przemyslaw.kubiak@pwr.edu.pl (untrusted)
depth=1: C = PL, ST = Lower Silesian Voivodeship, L = Wroclaw, O = Wroclaw University of Science and Technology,
    OU = WPPT/K2, CN = System Security I - IntermediateCA1.3, emailAddress = przemyslaw.kubiak@pwr.edu.pl (untrusted)
depth=2: C = PL, ST = Lower Silesian Voivodeship, L = Wroclaw, O = Wroclaw University of Science and Technology,
    OU = WPPT/K2, CN = System Security I - IntermediateCA1.2, emailAddress = przemyslaw.kubiak@pwr.edu.pl (untrusted)
depth=3: C = PL, ST = Lower Silesian Voivodeship, O = Wroclaw University of Science and Technology,
    OU = WPPT/K2, CN = System Security I - IntermediateCA1.1, emailAddress = przemyslaw.kubiak@pwr.edu.pl
depth=4: C = PL, ST = Lower Silesian Voivodeship, L = Wroclaw, O = Wroclaw University of Science and Technology,
    OU = WPPT/K2, CN = System Security I - RootCA1, emailAddress = przemyslaw.kubiak@pwr.edu.pl
```

We may also add -partial_chain option:

```
openssl verify -verbose -partial_chain -show_chain -check_ss_sig -untrusted untrusted1_no_IntCA1_1.pem \
    -trusted IntermediateCa1_1_Cert2.pem EndEntity_Cert.pem

EndEntity_Cert.pem: OK
Chain:
depth=0: C = PL, ST = Lower Silesian Voivodeship, L = Wroclaw, O = Wroclaw University of Science and Technology,
    OU = WPPT/K2, CN = localhost, emailAddress = przemyslaw.kubiak@pwr.edu.pl (untrusted)
depth=1: C = PL, ST = Lower Silesian Voivodeship, L = Wroclaw, O = Wroclaw University of Science and Technology,
    OU = WPPT/K2, CN = System Security I - IntermediateCA1.3, emailAddress = przemyslaw.kubiak@pwr.edu.pl (untrusted)
depth=2: C = PL, ST = Lower Silesian Voivodeship, L = Wroclaw, O = Wroclaw University of Science and Technology,
    OU = WPPT/K2, CN = System Security I - IntermediateCA1.2, emailAddress = przemyslaw.kubiak@pwr.edu.pl (untrusted)
depth=3: C = PL, ST = Lower Silesian Voivodeship, O = Wroclaw University of Science and Technology,
    OU = WPPT/K2, CN = System Security I - IntermediateCA1.1, emailAddress = przemyslaw.kubiak@pwr.edu.pl
```

So let us swich between RootCA1, RootCA2

```
cat IntermediateCa1_1_Cert2.pem IntermediateCa2_2_Cert.pem IntermediateCa1_3_Cert.pem IntermediateCa1_3_Cert2.pem \
    IntermediateCa1_2_Cert.pem IntermediateCa2_1_Cert2.pem > untrusted_all_intermediate.pem

openssl verify -verbose -show_chain -untrusted untrusted_all_intermediate.pem  \
    -trusted RootCa2Cert.pem EndEntity_Cert.pem
```

The output is:

```
C = PL, ST = Lower Silesian Voivodeship, O = Wroclaw University of Science and Technology,
    OU = WPPT/K2, CN = System Security I - IntermediateCA1.1, emailAddress = przemyslaw.kubiak@pwr.edu.pl
error 20 at 3 depth lookup: unable to get local issuer certificate
error EndEntity_Cert.pem: verification failed
```

Surprising! Only one path seems to be tried... Try -CAfile option instead of -trusted

```
openssl verify -verbose -show_chain -untrusted untrusted_all_intermediate.pem \
    -CAfile RootCa2Cert.pem EndEntity_Cert.pem


C = PL, ST = Lower Silesian Voivodeship, O = Wroclaw University of Science and Technology,
    OU = WPPT/K2, CN = System Security I - IntermediateCA1.1, emailAddress = przemyslaw.kubiak@pwr.edu.pl
error 20 at 3 depth lookup: unable to get local issuer certificate
error EndEntity_Cert.pem: verification failed


cat untrusted_all_intermediate.pem RootCa2Cert.pem > all_intermediate_plus_RootCa2Cert.pem
cat untrusted_all_intermediate.pem RootCa1Cert.pem > all_intermediate_plus_RootCa1Cert.pem
openssl verify -verbose -show_chain -untrusted RootCa1Cert.pem \
    -CAfile all_intermediate_plus_RootCa2Cert.pem EndEntity_Cert.pem
```

The output is:

```
C = PL, ST = Lower Silesian Voivodeship, O = Wroclaw University of Science and Technology,
    OU = WPPT/K2, CN = System Security I - IntermediateCA1.1, emailAddress = przemyslaw.kubiak@pwr.edu.pl
error 2 at 3 depth lookup: unable to get issuer certificate
error EndEntity_Cert.pem: verification failed
```

So apparently openssl tries only a single chain... We shall debug openssl library. But ealier let try the second RootCA:

```
openssl verify -verbose -show_chain -untrusted RootCa2Cert.pem \
    -CAfile all_intermediate_plus_RootCa1Cert.pem EndEntity_Cert.pem
```

And the output is:

```
EndEntity_Cert.pem: OK
Chain:
depth=0: C = PL, ST = Lower Silesian Voivodeship, L = Wroclaw, O = Wroclaw University of Science and Technology,
    OU = WPPT/K2, CN = localhost, emailAddress = przemyslaw.kubiak@pwr.edu.pl (untrusted)
depth=1: C = PL, ST = Lower Silesian Voivodeship, L = Wroclaw, O = Wroclaw University of Science and Technology,
    OU = WPPT/K2, CN = System Security I - IntermediateCA1.3, emailAddress = przemyslaw.kubiak@pwr.edu.pl
depth=2: C = PL, ST = Lower Silesian Voivodeship, L = Wroclaw, O = Wroclaw University of Science and Technology,
    OU = WPPT/K2, CN = System Security I - IntermediateCA1.2, emailAddress = przemyslaw.kubiak@pwr.edu.pl
depth=3: C = PL, ST = Lower Silesian Voivodeship, O = Wroclaw University of Science and Technology,
    OU = WPPT/K2, CN = System Security I - IntermediateCA1.1, emailAddress = przemyslaw.kubiak@pwr.edu.pl
depth=4: C = PL, ST = Lower Silesian Voivodeship, L = Wroclaw, O = Wroclaw University of Science and Technology,
    OU = WPPT/K2, CN = System Security I - RootCA1, emailAddress = przemyslaw.kubiak@pwr.edu.pl
```

Chapter 3

Finding the source of the problem – source code debugging

Task: Check why openssl fails our expectations and do not find the correct path. Use gdb for that purpose.

3.1 Basic options and the first look at the code

Downoad openssl sources to the openssl-working (in my case it is openssl-1.1.1b), configure and build it (but not install!) according to the instructions from https://medium.com/@amit.kulkarni/debugging-openssl-code-using-gdb-55451efe9428

```
cd openssl-1.1.1b
```

To be able to use our version of the libssl, libcrypto (build with debug symbols) let us change the LD_LIBRARY_PATH environmental variable:

```
export LD_LIBRARY_PATH=/SystemSecurity/openssl-working/openssl-1.1.1b:${LD_LIBRARY_PATH}
```

”export is bash shell BUILTINS commands, which means it is part of the shell. It marks an environment variables to be exported to child-processes.”
Run gdb:

```
gdb --args apps/openssl verify -verbose -show_chain -untrusted ../client-certs/untrusted_all_intermediate.pem \
    -trusted ../client-certs/RootCa2Cert.pem ../client-certs/EndEntity_Cert.pem
```

```
GNU gdb (Ubuntu 8.2-0ubuntu1) 8.2
Copyright (C) 2018 Free Software Foundation, Inc.
License GPLv3+: GNU GPL version 3 or later <http://gnu.org/licenses/gpl.html>
This is free software: you are free to change and redistribute it.
There is NO WARRANTY, to the extent permitted by law.
Type "show copying" and "show warranty" for details.
This GDB was configured as "x86_64-linux-gnu".
Type "show configuration" for configuration details.
For bug reporting instructions, please see:
<http://www.gnu.org/software/gdb/bugs/>.
Find the GDB manual and other documentation resources online at:
    <http://www.gnu.org/software/gdb/documentation/>.

For help, type "help".
Type "apropos word" to search for commands related to "word"...
Reading symbols from apps/openssl...done.

Below the output and the commands we use to trace the execution are mixed together. For gdb basics see e.g.,
https://www.tutorialspoint.com/gnu_debugger/gdb_debugging_programs.htm
or ftp://ftp.gnu.org/old-gnu/Manuals/gdb/html_chapter/gdb_9.html#SEC53
```

Put the breakpoint at the beginning of main:

```
(gdb) b main
Breakpoint 1 at 0x4e87d: file apps/openssl.c, line 121.
```

```
(gdb) run
Starting program: /SystemSecurity/openssl-working/openssl-1.1.1b/apps/openssl
    verify -verbose -show_chain -untrusted ../client-certs/untrusted_all_intermediate.pem \
    -trusted ../client-certs/RootCa2Cert.pem ../client-certs/EndEntity_Cert.pem
[Thread debugging using libthread_db enabled]
Using host libthread_db library "/lib/x86_64-linux-gnu/libthread_db.so.1".

Breakpoint 1, main (argc=9, argv=0x7fffffffdbd8) at apps/openssl.c:121
121     {
```

Use list command inside gdb:

```
(gdb) l main
116
117         return p;
118     }
119
120     int main(int argc, char *argv[])
121     {
122         FUNCTION f, *fp;
123         LHASH_OF(FUNCTION) *prog = NULL;
124         char **copied_argv = NULL;
125         char *p, *pname;
(gdb) n
123         LHASH_OF(FUNCTION) *prog = NULL;
(gdb)
....
```

Take a look on the sources of openssl main - line 167 and the later seem to be interesting:

```
    prog = prog_init();
    pname = opt_progname(argv[0]);

    /* first check the program name */
    f.name = pname;
    fp = lh_FUNCTION_retrieve(prog, &f);
    if (fp != NULL) {
        argv[0] = pname;
        ret = fp->func(argc, argv);
        goto end;
    }
```

So set a breakpoint at the beginning of the init function:

```
(gdb) b 167
Breakpoint 2 at 0x5555555a29d6: file apps/openssl.c, line 167.
(gdb) c
Continuing.

Breakpoint 2, main (argc=9, argv=0x7fffffffdbd8) at apps/openssl.c:167
167         prog = prog_init();
(gdb)
(gdb) l
162             ERR_print_errors(bio_err);
163             ret = 1;
164             goto end;
165         }
166
167         prog = prog_init();
168         pname = opt_progname(argv[0]);
169
170         /* first check the program name */
171         f.name = pname;
(gdb) n
168         pname = opt_progname(argv[0]);
(gdb)  p prog
$4 = (struct lhash_st_FUNCTION *) 0x555555644680
```

Let print the memory under the address $4 (below we print 32 characters..):

```
(gdb) x/32c $4
0x555555644680: 32 ' '  90 'Z'  100 'd' 85 'U'  85 'U'  85 'U'  0 '\000'        0 '\000'
0x555555644688: -90 '\246'      58 ':'  90 'Z'  85 'U'  85 'U'  85 'U'  0 '\000'        0 '\000'
0x555555644690: -40 '\330'      58 ':'  90 'Z'  85 'U'  85 'U'  85 'U'  0 '\000'        0 '\000'
0x555555644698: 77 'M'  0 '\000'        0 '\000'        0 '\000'        -128 '\200'     0 '\000'
                0 '\000'        0 '\000'
(gdb) n
171         f.name = pname;
(gdb) n
172         fp = lh_FUNCTION_retrieve(prog, &f);
(gdb) p pname
$6 = 0x555555627d60 <prog> "openssl"
```

Let print the memory under the address $6 (below we print 32 characters..):

```
(gdb) x/32c $6
0x555555627d60 <prog>:  111 'o' 112 'p' 101 'e' 110 'n' 115 's' 115 's' 108 'l' 0 '\000'
0x555555627d68 <prog+8>:        0 '\000'        0 '\000'        0 '\000'        0 '\000'
                                0 '\000'        0 '\000'        0 '\000'        0 '\000'
0x555555627d70 <prog+16>:       0 '\000'        0 '\000'        0 '\000'        0 '\000'
                                0 '\000'        0 '\000'        0 '\000'        0 '\000'
0x555555627d78 <prog+24>:       0 '\000'        0 '\000'        0 '\000'        0 '\000'
                                0 '\000'        0 '\000'        0 '\000'        0 '\000'
(gdb) n
173         if (fp != NULL) {
(gdb) p fp
$7 = (FUNCTION *) 0x0
(gdb) p f
$8 = {type = 4294967295, name = 0x555555627d60 <prog> "openssl", func = 0x7ffff7fd3628, help = 0x7ffff7ffd9f0 <_rtld_global+2448>}
```

Examine memory disasembling it at the address: f.func

```
(gdb) x/i f.func
   0x7ffff7fd3628:      mov    $0x12000000,%esp
```

And continue – the printout is self-explanatory:

```
(gdb) n
180         if (argc != 1) {
(gdb) n
181             argc--;
(gdb) n
182             argv++;
(gdb) n
183             ret = do_cmd(prog, argc, argv);
(gdb) s
do_cmd (prog=0x555555644680, argc=8, argv=0x7fffffffdbe0) at apps/openssl.c:545
545     {
(gdb) l
540         if (!one)
541             BIO_printf(bio_out, "\n\n");
542     }
543
544     static int do_cmd(LHASH_OF(FUNCTION) *prog, int argc, char *argv[])
545     {
546         FUNCTION f, *fp;
547
548         if (argc <= 0 || argv[0] == NULL)
549             return 0;
(gdb) s
548         if (argc <= 0 || argv[0] == NULL)
(gdb)
550         f.name = argv[0];
(gdb) n
551         fp = lh_FUNCTION_retrieve(prog, &f);
(gdb) p f.name
$12 = 0x7fffffffdfef "verify"
(gdb)
(gdb) p fp
$13 = (FUNCTION *) 0x5555556457c8
(gdb) n
552         if (fp == NULL) {
(gdb) p fp
$14 = (FUNCTION *) 0x55555561e720 <functions+1440>
(gdb) n
550         f.name = argv[0];
(gdb) n
551         fp = lh_FUNCTION_retrieve(prog, &f);
(gdb) p f.name
$12 = 0x7fffffffdfef "verify"
(gdb) p fp
$13 = (FUNCTION *) 0x5555556457c8
(gdb) n
552         if (fp == NULL) {
(gdb) p fp
$14 = (FUNCTION *) 0x55555561e720 <functions+1440>
(gdb) n
563         if (fp != NULL) {
(gdb) s
564             return fp->func(argc, argv);
(gdb) s
verify_main (argc=8, argv=0x7fffffffdbe0) at apps/verify.c:64
64      {
(gdb) l
59      #endif
60          {NULL}
61      };
62
63      int verify_main(int argc, char **argv)
64      {
65          ENGINE *e = NULL;
66          STACK_OF(X509) *untrusted = NULL, *trusted = NULL;
67          STACK_OF(X509_CRL) *crls = NULL;
68          X509_STORE *store = NULL;
```

To print the backtrace of the stack use the bt command:

```
(gdb) bt
#0  verify_main (argc=8, argv=0x7fffffffdbe0) at apps/verify.c:64
#1  0x00005555555a36e0 in do_cmd (prog=0x555555644680, argc=8, argv=0x7fffffffdbe0) at apps/openssl.c:564
#2  0x00005555555a2aa4 in main (argc=8, argv=0x7fffffffdbe0) at apps/openssl.c:183
(gdb) n
65          ENGINE *e = NULL;
(gdb) n
66          STACK_OF(X509) *untrusted = NULL, *trusted = NULL;
(gdb) n
67          STACK_OF(X509_CRL) *crls = NULL;
(gdb) n
68          X509_STORE *store = NULL;
(gdb) n
69          X509_VERIFY_PARAM *vpm = NULL;
(gdb) n
70          const char *prog, *CApath = NULL, *CAfile = NULL;
(gdb) n
71          int noCApath = 0, noCAfile = 0;
(gdb) n
72          int vpmtouched = 0, crl_download = 0, show_chain = 0, i = 0, ret = 1;
(gdb) n
75          if ((vpm = X509_VERIFY_PARAM_new()) == NULL)
(gdb) n
78          prog = opt_init(argc, argv, verify_options);
(gdb) p vpm
$15 = (X509_VERIFY_PARAM *) 0x555555646190
```

Now the consecutive parameters shall be analyzed:

```
(gdb) n
79          while ((o = opt_next()) != OPT_EOF) {
(gdb) n
80              switch (o) {
(gdb) p o
$18 = OPT_VERBOSE
(gdb) n
159                 v_verbose = 1;
(gdb) n
160                 break;
(gdb) n
79          while ((o = opt_next()) != OPT_EOF) {
(gdb) n
80              switch (o) {
(gdb) p o
$19 = OPT_SHOW_CHAIN
(gdb) n
152                 show_chain = 1;
(gdb) n
153                 break;
(gdb) n
79          while ((o = opt_next()) != OPT_EOF) {
(gdb) n
80              switch (o) {
(gdb) p o
$20 = OPT_UNTRUSTED
(gdb) n
124                 if (!load_certs(opt_arg(), &untrusted, FORMAT_PEM, NULL,
(gdb) p &untrusted
$21 = (struct stack_st_X509 **) 0x7fffffffd570
(gdb) n
127                 break;
(gdb) n
79          while ((o = opt_next()) != OPT_EOF) {
(gdb)
(gdb) n
80              switch (o) {
(gdb) p o
$22 = OPT_TRUSTED
(gdb)
(gdb) n
130                 noCAfile = 1;
(gdb) n
131                 noCApath = 1;
(gdb) n
132                 if (!load_certs(opt_arg(), &trusted, FORMAT_PEM, NULL,
(gdb) s
opt_arg () at apps/opt.c:760
760         return arg;
(gdb) p arg
$23 = 0x7fffffffe04e "../client-certs/RootCa2Cert.pem"
(gdb) n
761     }
(gdb) s
load_certs (file=0x7fffffffe04e "../client-certs/RootCa2Cert.pem", certs=0x7fffffffd578, format=32773, pass=0x0,
                desc=0x555555604f6b "trusted certificates") at apps/apps.c:967
967         return load_certs_crls(file, format, pass, desc, certs, NULL);
(gdb) s
load_certs_crls (file=0x7fffffffe04e "../client-certs/RootCa2Cert.pem", format=32773, pass=0x0,
                desc=0x555555604f6b "trusted certificates", pcerts=0x7fffffffd578, pcrls=0x0) at apps/apps.c:870
```

```
870     {
(gdb) n
873         STACK_OF(X509_INFO) *xis = NULL;
(gdb) n
876         int rv = 0;
(gdb) n
878         cb_data.password = pass;
(gdb) n
879         cb_data.prompt_info = file;
(gdb) n
881         if (format != FORMAT_PEM) {
(gdb) n
886         bio = bio_open_default(file, 'r', FORMAT_PEM);
(gdb) n
887         if (bio == NULL)
(gdb) n
890         xis = PEM_X509_INFO_read_bio(bio, NULL,
(gdb) n
894         BIO_free(bio);
(gdb) n
896         if (pcerts != NULL && *pcerts == NULL) {
(gdb) n
897             *pcerts = sk_X509_new_null();
(gdb) n
898             if (*pcerts == NULL)
(gdb) n
902         if (pcrls != NULL && *pcrls == NULL) {
...
(gdb) n
verify_main (argc=8, argv=0x7fffffffdbe0) at apps/verify.c:132
132                 if (!load_certs(opt_arg(), &trusted, FORMAT_PEM, NULL,
(gdb)
(gdb) n
135                 break;
(gdb) n
79          while ((o = opt_next()) != OPT_EOF) {
(gdb) n
163         argc = opt_num_rest();
(gdb) p o
$24 = OPT_EOF
(gdb)
(gdb) n
165         if (trusted != NULL && (CAfile || CApath)) {
(gdb) n
172         if ((store = setup_verify(CAfile, CApath, noCAfile, noCApath)) == NULL)
(gdb) l
167                         "%s: Cannot use -trusted with -CAfile or -CApath\n",
168                         prog);
169             goto end;
170         }
171
172         if ((store = setup_verify(CAfile, CApath, noCAfile, noCApath)) == NULL)
173             goto end;
174         X509_STORE_set_verify_cb(store, cb);
175
176         if (vpmtouched)
(gdb) s
setup_verify (CAfile=0x0, CApath=0x0, noCAfile=1, noCApath=1) at apps/apps.c:1225
1225        X509_STORE *store = X509_STORE_new();
```

3.2 Openssl data structures

What is `X509_STORE *store` ? Go to the root directory of the openssl sources and execute:

```
grep -nr  "X509_STORE" . | more
./test/x509_dup_cert_test.c:20:    X509_STORE_CTX *sctx = NULL;
./test/x509_dup_cert_test.c:21:    X509_STORE *store = NULL;
./test/x509_dup_cert_test.c:25:    if (TEST_ptr(store = X509_STORE_new())
./test/x509_dup_cert_test.c:26:        && TEST_ptr(lookup = X509_STORE_add_lookup(store, X509_LOOKUP_file())))
./test/x509_dup_cert_test.c:31:    X509_STORE_CTX_free(sctx);
./test/x509_dup_cert_test.c:32:    X509_STORE_free(store);
./test/sslapitest.c:5636:static int verify_cb(int preverify_ok, X509_STORE_CTX *x509_ctx)
./test/danetest.c:39:static void store_ctx_dane_init(X509_STORE_CTX *, SSL *);
./test/danetest.c:57:    X509_STORE_CTX *store_ctx = NULL;
./test/danetest.c:59:    X509_STORE *store = NULL;
./test/danetest.c:62:    int store_ctx_idx = SSL_get_ex_data_X509_STORE_CTX_idx();
./test/danetest.c:64:    if (!TEST_ptr(store_ctx = X509_STORE_CTX_new())
./test/danetest.c:68:            || !TEST_true(X509_STORE_CTX_init(store_ctx, store, cert, chain))
./test/danetest.c:69:            || !TEST_true(X509_STORE_CTX_set_ex_data(store_ctx, store_ctx_idx,
./test/danetest.c:73:    X509_STORE_CTX_set_default(store_ctx,
./test/danetest.c:75:    X509_VERIFY_PARAM_set1(X509_STORE_CTX_get0_param(store_ctx),
./test/danetest.c:80:        X509_STORE_CTX_set_verify_cb(store_ctx, SSL_get_verify_callback(ssl));
./test/danetest.c:86:    SSL_set_verify_result(ssl, X509_STORE_CTX_get_error(store_ctx));
./test/danetest.c:87:    X509_STORE_CTX_cleanup(store_ctx);
./test/danetest.c:90:    X509_STORE_CTX_free(store_ctx);
./test/danetest.c:427:static void store_ctx_dane_init(X509_STORE_CTX *store_ctx, SSL *ssl)
./test/danetest.c:429:    X509_STORE_CTX_set0_dane(store_ctx, SSL_get0_dane(ssl));
./test/crltest.c:235:    X509_STORE_CTX *ctx = X509_STORE_CTX_new();
./test/crltest.c:236:    X509_STORE *store = X509_STORE_new();
```

```
./test/crltest.c:250:        || !TEST_true(X509_STORE_CTX_init(ctx, store, leaf, NULL)))
./test/crltest.c:252:    X509_STORE_CTX_set0_trusted_stack(ctx, roots);
./test/crltest.c:253:    X509_STORE_CTX_set0_crls(ctx, crls);
./test/crltest.c:260:    X509_STORE_CTX_set0_param(ctx, param);
./test/crltest.c:265:                                    : X509_STORE_CTX_get_error(ctx);
./test/crltest.c:270:    X509_STORE_CTX_free(ctx);
./test/crltest.c:271:    X509_STORE_free(store);
./test/ossl_shim/ossl_shim.cc:207:static int VerifySucceed(X509_STORE_CTX *store_ctx, void *arg) {
./test/ossl_shim/ossl_shim.cc:211:static int VerifyFail(X509_STORE_CTX *store_ctx, void *arg) {
./test/ossl_shim/ossl_shim.cc:212:  X509_STORE_CTX_set_error(store_ctx, X509_V_ERR_APPLICATION_VERIFICATION);
./test/ssltest_old.c:86:static int verify_callback(int ok, X509_STORE_CTX *ctx);
./test/ssltest_old.c:87:static int app_verify_callback(X509_STORE_CTX *ctx, void *arg);
./test/ssltest_old.c:2769:static int verify_callback(int ok, X509_STORE_CTX *ctx)
./test/ssltest_old.c:2773:    s = X509_NAME_oneline(X509_get_subject_name(X509_STORE_CTX_get_current_cert(ctx)),
./test/ssltest_old.c:2777:            printf("depth=%d %s\n", X509_STORE_CTX_get_error_depth(ctx), buf);
./test/ssltest_old.c:2780:                    X509_STORE_CTX_get_error_depth(ctx),
./test/ssltest_old.c:2781:                    X509_STORE_CTX_get_error(ctx), buf);
./test/ssltest_old.c:2786:        int i = X509_STORE_CTX_get_error(ctx);
./test/ssltest_old.c:2804:static int app_verify_callback(X509_STORE_CTX *ctx, void *arg)
./test/ssltest_old.c:2811:        X509 *c = X509_STORE_CTX_get0_cert(ctx);
./test/ssltest_old.c:2821:                    X509_STORE_CTX_get_error_depth(ctx), buf);
....
```

Let try to modify the expression:

```
grep -nr  'X509_STORE[^_]' . | more
./test/x509_dup_cert_test.c:21:    X509_STORE *store = NULL;
./test/danetest.c:59:    X509_STORE *store = NULL;
./test/crltest.c:236:    X509_STORE *store = X509_STORE_new();
./test/verify_extra_test.c:98:    X509_STORE *store = NULL;
./CHANGES:982:     X509_STORE, X509_LOOKUP, and X509_LOOKUP_METHOD.  The unused type
./CHANGES:2659:     X509_STORE from X509_STORE_CTX.
./CHANGES:4289:     X509_STORE dependency on certificate verification and allow alternative
./CHANGES:4290:     lookup methods.  X509_STORE based implementations of these two callbacks.
./CHANGES:4293:  *) Allow multiple CRLs to exist in an X509_STORE with matching issuer names.
./CHANGES:4662:  *) Fix X509_STORE locking: Every 'objs' access requires a lock (to
./CHANGES:7630:     setting of purpose and trust fields. New X509_STORE trust and
./CHANGES:7637:     X509_STORE structure (such as flags for CRL checking and custom
./CHANGES:7642:     trust settings if they are not set in X509_STORE. This allows X509_STORE
./CHANGES:7650:     are set then the CRL is looked up in the X509_STORE structure and
./CHANGES:9759:     The LHASH 'certs' is X509_STORE has now been replaced
./CHANGES:9767:     with X509_STORE internally.
./CHANGES:9772:     The X509_STORE API doesn't directly support the retrieval
./CHANGES:9783:     callback. Although this currently uses an X509_STORE it
./CHANGES:9785:     to bypass the X509_STORE hackery necessary to make this
./CHANGES:12889:  *) Make sure the already existing X509_STORE->depth variable is initialized
./apps/ts.c:76:static X509_STORE *create_cert_store(const char *CApath, const char *CAfile,
./apps/ts.c:916:    /* Initialising the X509_STORE object. */
./apps/ts.c:937:static X509_STORE *create_cert_store(const char *CApath, const char *CAfile,
./apps/ts.c:940:    X509_STORE *cert_ctx = NULL;
./apps/smime.c:120:    X509_STORE *store = NULL;
./apps/pkcs12.c:33:static int get_cert_chain(X509 *cert, X509_STORE *store,
./apps/pkcs12.c:398:            X509_STORE *store;
./apps/pkcs12.c:742:static int get_cert_chain(X509 *cert, X509_STORE *store,
./apps/ocsp.c:219:    X509_STORE *store = NULL;
./apps/s_cb.c:1212:static int add_crls_store(X509_STORE *st, STACK_OF(X509_CRL) *crls)
./apps/s_cb.c:1225:    X509_STORE *st;
./apps/s_cb.c:1238:    X509_STORE *vfy = NULL, *ch = NULL;
./apps/crl.c:67:    X509_STORE *store = NULL;
./apps/x509.c:39:static int x509_certify(X509_STORE *ctx, const char *CAfile, const EVP_MD *digest,
./apps/x509.c:165:    X509_STORE *ctx = NULL;
./apps/x509.c:946:static int x509_certify(X509_STORE *ctx, const char *CAfile, const EVP_MD *digest,
./apps/cms.c:200:    X509_STORE *store = NULL;
./apps/apps.c:1223:X509_STORE *setup_verify(const char *CAfile, const char *CApath, int noCAfile, int noCApath)
./apps/apps.c:1225:    X509_STORE *store = X509_STORE_new();
./apps/apps.c:2081:void store_setup_crl_download(X509_STORE *st)
./apps/verify.c:22:static int check(X509_STORE *ctx, const char *file,
...
```

Still too many.. Modify the grep command even further:

```
grep -nr  'X509_STORE[^_]' --include \*.h | more

apps/apps.h:473:X509_STORE *setup_verify(const char *CAfile, const char *CApath,
apps/apps.h:572:void store_setup_crl_download(X509_STORE *st);
crypto/include/internal/x509_int.h:194:    X509_STORE *ctx;
crypto/x509/x509_lcl.h:97:    X509_STORE *store_ctx;      /* who owns us */
crypto/ts/ts_lcl.h:194:    X509_STORE *store;
include/openssl/x509_vfy.h:40:The X509_STORE holds the tables etc for verification stuff.
include/openssl/x509_vfy.h:42:The X509_STORE has X509_LOOKUPs for looking up certs.
include/openssl/x509_vfy.h:43:The X509_STORE then calls a function to actually verify the
include/openssl/x509_vfy.h:61:int X509_STORE_set_depth(X509_STORE *store, int depth);
include/openssl/x509_vfy.h:267:X509_STORE *X509_STORE_new(void);
include/openssl/x509_vfy.h:268:void X509_STORE_free(X509_STORE *v);
include/openssl/x509_vfy.h:269:int X509_STORE_lock(X509_STORE *ctx);
include/openssl/x509_vfy.h:270:int X509_STORE_unlock(X509_STORE *ctx);
include/openssl/x509_vfy.h:271:int X509_STORE_up_ref(X509_STORE *v);
include/openssl/x509_vfy.h:272:STACK_OF(X509_OBJECT) *X509_STORE_get0_objects(X509_STORE *v);
include/openssl/x509_vfy.h:276:int X509_STORE_set_flags(X509_STORE *ctx, unsigned long flags);
include/openssl/x509_vfy.h:277:int X509_STORE_set_purpose(X509_STORE *ctx, int purpose);
```

```
include/openssl/x509_vfy.h:278:int X509_STORE_set_trust(X509_STORE *ctx, int trust);
include/openssl/x509_vfy.h:279:int X509_STORE_set1_param(X509_STORE *ctx, X509_VERIFY_PARAM *pm);
include/openssl/x509_vfy.h:280:X509_VERIFY_PARAM *X509_STORE_get0_param(X509_STORE *ctx);
include/openssl/x509_vfy.h:282:void X509_STORE_set_verify(X509_STORE *ctx, X509_STORE_CTX_verify_fn verify);
include/openssl/x509_vfy.h:287:X509_STORE_CTX_verify_fn X509_STORE_get_verify(X509_STORE *ctx);
include/openssl/x509_vfy.h:288:void X509_STORE_set_verify_cb(X509_STORE *ctx,
...
```

In the file include/openssl/x509_vfy.h we find a comment:

```
/*-
SSL_CTX -> X509_STORE
                -> X509_LOOKUP
                        ->X509_LOOKUP_METHOD
                -> X509_LOOKUP
                        ->X509_LOOKUP_METHOD

SSL     -> X509_STORE_CTX
                ->X509_STORE

The X509_STORE holds the tables etc for verification stuff.
A X509_STORE_CTX is used while validating a single certificate.
The X509_STORE has X509_LOOKUPs for looking up certs.
The X509_STORE then calls a function to actually verify the
certificate chain.
*/
```

Let us check the header file included in the include/openssl/x509_vfy.h
In the x509_vfy.h file we find:

```
/*
 * Protect against recursion, x509.h and x509_vfy.h each include the other.
 */
# ifndef HEADER_X509_H
#  include <openssl/x509.h>
# endif
```

After examining the file openssl/x509.h still no definition found. However, definitions of types tend to end with ”;”

```
grep -nr  'X509_STORE;' --include \*.h | more

include/openssl/ossl_typ.h:127:typedef struct x509_store_st X509_STORE;
```

So we have it! After checking the header file we see that this is only enumeration of the typedefinitions based on the structures defined elsewhere. Let us cite a few:

```
...
typedef struct x509_st X509;
typedef struct X509_algor_st X509_ALGOR;
typedef struct X509_crl_st X509_CRL;
typedef struct x509_crl_method_st X509_CRL_METHOD;
typedef struct x509_revoked_st X509_REVOKED;
typedef struct X509_name_st X509_NAME;
typedef struct X509_pubkey_st X509_PUBKEY;
typedef struct x509_store_st X509_STORE;
typedef struct x509_store_ctx_st X509_STORE_CTX;

typedef struct x509_object_st X509_OBJECT;
typedef struct x509_lookup_st X509_LOOKUP;
typedef struct x509_lookup_method_st X509_LOOKUP_METHOD;
typedef struct X509_VERIFY_PARAM_st X509_VERIFY_PARAM;
....
```

Let start with

```
typedef struct x509_store_st X509_STORE;
typedef struct x509_store_ctx_st X509_STORE_CTX;

grep -nr  'x509_store_st' --include \*.h | more

crypto/x509/x509_lcl.h:105:struct x509_store_st {
include/openssl/ossl_typ.h:127:typedef struct x509_store_st X509_STORE;
ssl/ssl_locl.h:751:    struct x509_store_st /* X509_STORE */ *cert_store;
```

So let look at crypto/x509/x509_lcl.h:

```
/*
 * This is used to hold everything.  It is used for all certificate
 * validation.  Once we have a certificate chain, the 'verify' function is
 * then called to actually check the cert chain.
 */
```

```
struct x509_store_st {
    /* The following is a cache of trusted certs */
    int cache;                  /* if true, stash any hits */
    STACK_OF(X509_OBJECT) *objs; /* Cache of all objects */
    /* These are external lookup methods */
    STACK_OF(X509_LOOKUP) *get_cert_methods;
    X509_VERIFY_PARAM *param;
    /* Callbacks for various operations */
    /* called to verify a certificate */
    int (*verify) (X509_STORE_CTX *ctx);
    /* error callback */
    int (*verify_cb) (int ok, X509_STORE_CTX *ctx);
    /* get issuers cert from ctx */
    int (*get_issuer) (X509 **issuer, X509_STORE_CTX *ctx, X509 *x);
    /* check issued */
    int (*check_issued) (X509_STORE_CTX *ctx, X509 *x, X509 *issuer);
    /* Check revocation status of chain */
    int (*check_revocation) (X509_STORE_CTX *ctx);
    /* retrieve CRL */
    int (*get_crl) (X509_STORE_CTX *ctx, X509_CRL **crl, X509 *x);
    /* Check CRL validity */
    int (*check_crl) (X509_STORE_CTX *ctx, X509_CRL *crl);
    /* Check certificate against CRL */
    int (*cert_crl) (X509_STORE_CTX *ctx, X509_CRL *crl, X509 *x);
    /* Check policy status of the chain */
    int (*check_policy) (X509_STORE_CTX *ctx);
    STACK_OF(X509) *(*lookup_certs) (X509_STORE_CTX *ctx, X509_NAME *nm);
    STACK_OF(X509_CRL) *(*lookup_crls) (X509_STORE_CTX *ctx, X509_NAME *nm);
    int (*cleanup) (X509_STORE_CTX *ctx);
    CRYPTO_EX_DATA ex_data;
    CRYPTO_REF_COUNT references;
    CRYPTO_RWLOCK *lock;
};
```

So we have found the structure definition. The field:

```
X509_VERIFY_PARAM *param;
```

might be relevant (recall that the ’openssl verify’ contains a number of parameters). It is defined in the same file crypto/x509/x509_lcl.h:

```
/*
 * This structure holds all parameters associated with a verify operation by
 * including an X509_VERIFY_PARAM structure in related structures the
 * parameters used can be customized
 */

struct X509_VERIFY_PARAM_st {
    char *name;
    time_t check_time;          /* Time to use */
    uint32_t inh_flags;         /* Inheritance flags */
    unsigned long flags;        /* Various verify flags */
    int purpose;                /* purpose to check untrusted certificates */
    int trust;                  /* trust setting to check */
    int depth;                  /* Verify depth */
    int auth_level;             /* Security level for chain verification */
    STACK_OF(ASN1_OBJECT) *policies; /* Permissible policies */
    /* Peer identity details */
    STACK_OF(OPENSSL_STRING) *hosts; /* Set of acceptable names */
    unsigned int hostflags;     /* Flags to control matching features */
    char *peername;             /* Matching hostname in peer certificate */
    char *email;                /* If not NULL email address to match */
    size_t emaillen;
    unsigned char *ip;          /* If not NULL IP address to match */
    size_t iplen;               /* Length of IP address */
};
```

Now its turn of investigating

```
typedef struct x509_store_ctx_st X509_STORE_CTX;


grep -nr  'x509_store_ctx_st' --include \*.h | more

crypto/include/internal/x509_int.h:193:struct x509_store_ctx_st {      /* X509_STORE_CTX */
include/openssl/ossl_typ.h:128:typedef struct x509_store_ctx_st X509_STORE_CTX;
```

Now we check the line 193 in crypto/include/internal/x509_int.h (’internal’ may suggest that it is a structure of openssl used internally, perhaps being more prone to modifications in next releases):

```
/*
 * This is a used when verifying cert chains.  Since the gathering of the
 * cert chain can take some time (and have to be 'retried', this needs to be
 * kept and passed around.
 */
struct x509_store_ctx_st {      /* X509_STORE_CTX */
```

```
    X509_STORE *ctx;
    /* The following are set by the caller */
    /* The cert to check */
    X509 *cert;
    /* chain of X509s - untrusted - passed in */
    STACK_OF(X509) *untrusted;
    /* set of CRLs passed in */
    STACK_OF(X509_CRL) *crls;
    X509_VERIFY_PARAM *param;
    /* Other info for use with get_issuer() */
    void *other_ctx;
    /* Callbacks for various operations */
    /* called to verify a certificate */
    int (*verify) (X509_STORE_CTX *ctx);
    /* error callback */
    int (*verify_cb) (int ok, X509_STORE_CTX *ctx);
    /* get issuers cert from ctx */
    int (*get_issuer) (X509 **issuer, X509_STORE_CTX *ctx, X509 *x);
    /* check issued */
    int (*check_issued) (X509_STORE_CTX *ctx, X509 *x, X509 *issuer);
    /* Check revocation status of chain */
    int (*check_revocation) (X509_STORE_CTX *ctx);
    /* retrieve CRL */
    int (*get_crl) (X509_STORE_CTX *ctx, X509_CRL **crl, X509 *x);
    /* Check CRL validity */
    int (*check_crl) (X509_STORE_CTX *ctx, X509_CRL *crl);
    /* Check certificate against CRL */
    int (*cert_crl) (X509_STORE_CTX *ctx, X509_CRL *crl, X509 *x);
    /* Check policy status of the chain */
    int (*check_policy) (X509_STORE_CTX *ctx);
    STACK_OF(X509) *(*lookup_certs) (X509_STORE_CTX *ctx, X509_NAME *nm);
    STACK_OF(X509_CRL) *(*lookup_crls) (X509_STORE_CTX *ctx, X509_NAME *nm);
    int (*cleanup) (X509_STORE_CTX *ctx);
    /* The following is built up */
    /* if 0, rebuild chain */
    int valid;
    /* number of untrusted certs */
    int num_untrusted;
    /* chain of X509s - built up and trusted */
    STACK_OF(X509) *chain;
    /* Valid policy tree */
    X509_POLICY_TREE *tree;
    /* Require explicit policy value */
    int explicit_policy;
    /* When something goes wrong, this is why */
    int error_depth;
    int error;
    X509 *current_cert;
    /* cert currently being tested as valid issuer */
    X509 *current_issuer;
    /* current CRL */
    X509_CRL *current_crl;
    /* score of current CRL */
    int current_crl_score;
    /* Reason mask */
    unsigned int current_reasons;
    /* For CRL path validation: parent context */
    X509_STORE_CTX *parent;
    CRYPTO_EX_DATA ex_data;
    SSL_DANE *dane;
    /* signed via bare TA public key, rather than CA certificate */
    int bare_ta_signed;
};
```

We shall understand the meaning of the fields of the structure by returning to debugging the code.

3.3 Deaper in the code

If we have stopped the gdb previously we may set a breakpoit to the entry to a particular function:

```
gdb --args apps/openssl verify -verbose -show_chain -untrusted ../client-certs/untrusted_all_intermediate.pem \
    -trusted ../client-certs/RootCa2Cert.pem ../client-certs/EndEntity_Cert.pem
GNU gdb (Ubuntu 8.2-0ubuntu1) 8.2
Copyright (C) 2018 Free Software Foundation, Inc.
License GPLv3+: GNU GPL version 3 or later <http://gnu.org/licenses/gpl.html>
This is free software: you are free to change and redistribute it.
There is NO WARRANTY, to the extent permitted by law.
Type "show copying" and "show warranty" for details.
This GDB was configured as "x86_64-linux-gnu".
Type "show configuration" for configuration details.
For bug reporting instructions, please see:
<http://www.gnu.org/software/gdb/bugs/>.
Find the GDB manual and other documentation resources online at:
    <http://www.gnu.org/software/gdb/documentation/>.

For help, type "help".
Type "apropos word" to search for commands related to "word"...
Reading symbols from apps/openssl...done.
```

```
(gdb) b verify_main
Breakpoint 1 at 0x84630: file apps/verify.c, line 64.
```

an later on execute run command:

```
(gdb) r
Starting program: /SystemSecurity/openssl-working/openssl-1.1.1b/apps/openssl verify -verbose \
    -show_chain -untrusted ../client-certs/untrusted_all_intermediate.pem \
    -trusted ../client-certs/RootCa2Cert.pem ../client-certs/EndEntity_Cert.pem
[Thread debugging using libthread_db enabled]
Using host libthread_db library "/lib/x86_64-linux-gnu/libthread_db.so.1".

Breakpoint 1, verify_main (argc=8, argv=0x7fffffffdcd0) at apps/verify.c:64
64      {
(gdb) l
59      #endif
60          {NULL}
61      };
62
63      int verify_main(int argc, char **argv)
64      {
65          ENGINE *e = NULL;
66          STACK_OF(X509) *untrusted = NULL, *trusted = NULL;
67          STACK_OF(X509_CRL) *crls = NULL;
68          X509_STORE *store = NULL;
(gdb) l
69          X509_VERIFY_PARAM *vpm = NULL;
70          const char *prog, *CApath = NULL, *CAfile = NULL;
71          int noCApath = 0, noCAfile = 0;
72          int vpmtouched = 0, crl_download = 0, show_chain = 0, i = 0, ret = 1;
73          OPTION_CHOICE o;
74
75          if ((vpm = X509_VERIFY_PARAM_new()) == NULL)
76              goto end;
77
78          prog = opt_init(argc, argv, verify_options);
```

To return to exactly the same line we have stopped previously let set a second breakpoint and continue program execution:

```
(gdb) b apps/verify.c:172
Breakpoint 2 at 0x5555555d8ab9: file apps/verify.c, line 172.
(gdb) c
Continuing.

Breakpoint 2, verify_main (argc=1, argv=0x7fffffffdd08) at apps/verify.c:172
172         if ((store = setup_verify(CAfile, CApath, noCAfile, noCApath)) == NULL)
(gdb)
```

Let look at the stack of calls:

```
(gdb) bt
#0  verify_main (argc=1, argv=0x7fffffffdd08) at apps/verify.c:172
#1  0x00005555555a36e0 in do_cmd (prog=0x555555644680, argc=8, argv=0x7fffffffdcd0) at apps/openssl.c:564
#2  0x00005555555a2aa4 in main (argc=8, argv=0x7fffffffdcd0) at apps/openssl.c:183
(gdb)


(gdb) p noCAfile
$3 = 1
(gdb) p noCApath
$4 = 1
(gdb) s
setup_verify (CAfile=0x0, CApath=0x0, noCAfile=1, noCApath=1) at apps/apps.c:1225
1225        X509_STORE *store = X509_STORE_new();
(gdb)


(gdb) l
1220        BIO_printf(out, "\n};\n");
1221    }
1222
1223    X509_STORE *setup_verify(const char *CAfile, const char *CApath, int noCAfile, int noCApath)
1224    {
1225        X509_STORE *store = X509_STORE_new();
1226        X509_LOOKUP *lookup;
1227
1228        if (store == NULL)
1229            goto end;
(gdb) l
1230
1231        if (CAfile != NULL || !noCAfile) {
1232            lookup = X509_STORE_add_lookup(store, X509_LOOKUP_file());
1233            if (lookup == NULL)
1234                goto end;
1235            if (CAfile) {
1236                if (!X509_LOOKUP_load_file(lookup, CAfile, X509_FILETYPE_PEM)) {
1237                    BIO_printf(bio_err, "Error loading file %s\n", CAfile);
```

```
1238                    goto end;
1239                }
(gdb)

(gdb) s
X509_STORE_new () at crypto/x509/x509_lu.c:161
161     {
(gdb) l
156         }
157         return ret;
158     }
159
160     X509_STORE *X509_STORE_new(void)
161     {
162         X509_STORE *ret = OPENSSL_zalloc(sizeof(*ret));
163
164         if (ret == NULL) {
165             X509err(X509_F_X509_STORE_NEW, ERR_R_MALLOC_FAILURE);
(gdb)
```

Too low level, to continue to the end of the function use finish command (https://sourceware.org/gdb/current/onlinedocs/gdb/Continuing-and-Stepping.html):

```
Run till exit from #0  X509_STORE_new () at crypto/x509/x509_lu.c:161
0x00005555555dfd23 in setup_verify (CAfile=0x0, CApath=0x0, noCAfile=1, noCApath=1) at apps/apps.c:1225
1225        X509_STORE *store = X509_STORE_new();
Value returned is $5 = (X509_STORE *) 0x55555564b4d0
(gdb)


(gdb) n
1228        if (store == NULL)
(gdb) n
1231        if (CAfile != NULL || !noCAfile) {
(gdb) n
1245        if (CApath != NULL || !noCApath) {
(gdb) n
1259        ERR_clear_error();
(gdb) n
1260        return store;
(gdb) n
1264    }
(gdb) l
1259        ERR_clear_error();
1260        return store;
1261     end:
1262        X509_STORE_free(store);
1263        return NULL;
1264    }
1265
1266    #ifndef OPENSSL_NO_ENGINE
1267    /* Try to load an engine in a shareable library */
1268    static ENGINE *try_load_engine(const char *engine)
(gdb) n
verify_main (argc=1, argv=0x7fffffffdd08) at apps/verify.c:174
174         X509_STORE_set_verify_cb(store, cb);
(gdb) x/i cb
   0x5555555d8f71 <cb>: push   %rbp
(gdb) s
X509_STORE_set_verify_cb (ctx=0x55555564b4d0, verify_cb=0x5555555d8f71 <cb>) at crypto/x509/x509_lu.c:767
767         ctx->verify_cb = verify_cb;
(gdb) n
768     }
(gdb) n
verify_main (argc=1, argv=0x7fffffffdd08) at apps/verify.c:176
176         if (vpmtouched)
(gdb) p vpmtouched
$6 = 0
(gdb) n
179         ERR_clear_error();
(gdb) n
181         if (crl_download)
(gdb) n
184         ret = 0;
(gdb) n
185         if (argc < 1) {
(gdb) p argc
$7 = 1
```

3.4 Examining memory

From the source file apps/verify.c we see that we are near the end of the function verify_main. Let take a look on trusted and untrusted stacks.

```
(gdb) p trusted
$8 = (struct stack_st_X509 *) 0x5555556515e0
```

Since gdb allows to evaluate expressions, including functions, call:

```
(gdb) p sk_X509_num($8)
$9 = 1
(gdb) p untrusted
$10 = (struct stack_st_X509 *) 0x5555556528e0
(gdb) p sk_X509_num($10)
$11 = 6
(gdb)
```

Let take a look at one of the certificates. To do so we need to store some additional variable (a pointer), so we shall allocate memory for it (gdb is very flexible):

```
(gdb) call malloc(8)
$12 = (void *) 0x7ffff79f9e90
(gdb) set {unsigned char*}0x7ffff79f9e90 = NULL
```

Examine memory: 8 bytes in hexadecimal format:

```
(gdb) x/8xb 0x7ffff79f9e90
0x7ffff79f9e90: 0x00    0x00    0x00    0x00    0x00    0x00    0x00    0x00
```

So let us call some function:

```
(gdb) call i2d_X509(sk_X509_value(untrusted, 0), &{unsigned char*}0x7ffff79f9e90)
$16 = 886
```

What has happened?: sk_X509_value(untrusted, 0) - returns the pointer to X509 being index 0 element of the stack, What does the function int i2d_X509(X509 *x, unsigned char **out) ?
After https://www.openssl.org/docs/man1.0.2/man3/d2i_X509.html:
i2d_X509() encodes the structure pointed to by x into DER format. If out is not NULL is writes the DER encoded data to the buffer at *out, and increments it to point after the data just written. If the return value is negative an error occurred, otherwise it returns the length of the encoded data.
For OpenSSL 0.9.7 and later if *out is NULL memory will be allocated for a buffer and the encoded data written to it. In this case *out is not incremented and it points to the start of the data just written.
As we see some address is indeed put to the memory

```
(gdb) x/8xb 0x7ffff79f9e90
0x7ffff79f9e90: 0x20    0x84    0x64    0x55    0x55    0x55    0x00    0x00

(gdb) p {unsigned char*}0x7ffff79f9e90
$17 = (unsigned char *) 0x555555648420 "0\202\003r0\202\002\002\001\002\002\002\022\065\060
\n\006\b*\206H\316=\004\003\003\060\201\340\061\v0\t\006\003U\004\006\023\002PL1#0!\006\003U\004\
b\f\032Lower Silesian Voivodeship1\020\060\016\006\003U\004\a\f\aWroclaw1503\006\003U\004
\n\f,Wroclaw University of Science and Technology1\020\060\016\006\003U\004\v\f\aWPPT/K21$0\
"\006\003U\004\003\f\033System Security "...
```

Examine memory:

```
(gdb) dump binary memory tempMemDump.asn1 $17 $17+886
(gdb) shell
```

and on the console run: ”openssl asn1parse -inform DER -in tempMemDump.asn1” and then ”exit”

```
    0:d=0  hl=4 l= 882 cons: SEQUENCE
    4:d=1  hl=4 l= 727 cons: SEQUENCE
    8:d=2  hl=2 l=   3 cons: cont [ 0 ]
   10:d=3  hl=2 l=   1 prim: INTEGER           :02
   13:d=2  hl=2 l=   2 prim: INTEGER           :1235
   17:d=2  hl=2 l=  10 cons: SEQUENCE
   19:d=3  hl=2 l=   8 prim: OBJECT            :ecdsa-with-SHA384
   29:d=2  hl=3 l= 224 cons: SEQUENCE
   32:d=3  hl=2 l=  11 cons: SET
   34:d=4  hl=2 l=   9 cons: SEQUENCE
   36:d=5  hl=2 l=   3 prim: OBJECT            :countryName
   41:d=5  hl=2 l=   2 prim: PRINTABLESTRING   :PL
   45:d=3  hl=2 l=  35 cons: SET
   47:d=4  hl=2 l=  33 cons: SEQUENCE
   49:d=5  hl=2 l=   3 prim: OBJECT            :stateOrProvinceName
   54:d=5  hl=2 l=  26 prim: UTF8STRING        :Lower Silesian Voivodeship
   82:d=3  hl=2 l=  16 cons: SET
   84:d=4  hl=2 l=  14 cons: SEQUENCE
   86:d=5  hl=2 l=   3 prim: OBJECT            :localityName
   91:d=5  hl=2 l=   7 prim: UTF8STRING        :Wroclaw
  100:d=3  hl=2 l=  53 cons: SET
  102:d=4  hl=2 l=  51 cons: SEQUENCE
  104:d=5  hl=2 l=   3 prim: OBJECT            :organizationName
  109:d=5  hl=2 l=  44 prim: UTF8STRING        :Wroclaw University of Science and Technology
```

```
  155:d=3  hl=2 l=  16 cons: SET
  157:d=4  hl=2 l=  14 cons: SEQUENCE
  159:d=5  hl=2 l=   3 prim: OBJECT            :organizationalUnitName
  164:d=5  hl=2 l=   7 prim: UTF8STRING        :WPPT/K2
  173:d=3  hl=2 l=  36 cons: SET
  175:d=4  hl=2 l=  34 cons: SEQUENCE
  177:d=5  hl=2 l=   3 prim: OBJECT            :commonName
  182:d=5  hl=2 l=  27 prim: UTF8STRING        :System Security I - RootCA1
  211:d=3  hl=2 l=  43 cons: SET
  213:d=4  hl=2 l=  41 cons: SEQUENCE
  215:d=5  hl=2 l=   9 prim: OBJECT            :emailAddress
  226:d=5  hl=2 l=  28 prim: IA5STRING         :przemyslaw.kubiak@pwr.edu.pl
  256:d=2  hl=2 l=  30 cons: SEQUENCE
  258:d=3  hl=2 l=  13 prim: UTCTIME           :190511184258Z
  273:d=3  hl=2 l=  13 prim: UTCTIME           :290319184258Z
  288:d=2  hl=3 l= 216 cons: SEQUENCE
  291:d=3  hl=2 l=  11 cons: SET
  293:d=4  hl=2 l=   9 cons: SEQUENCE
  295:d=5  hl=2 l=   3 prim: OBJECT            :countryName
  300:d=5  hl=2 l=   2 prim: PRINTABLESTRING   :PL
  304:d=3  hl=2 l=  35 cons: SET
  306:d=4  hl=2 l=  33 cons: SEQUENCE
  308:d=5  hl=2 l=   3 prim: OBJECT            :stateOrProvinceName
  313:d=5  hl=2 l=  26 prim: UTF8STRING        :Lower Silesian Voivodeship
  341:d=3  hl=2 l=  53 cons: SET
  343:d=4  hl=2 l=  51 cons: SEQUENCE
  345:d=5  hl=2 l=   3 prim: OBJECT            :organizationName
  350:d=5  hl=2 l=  44 prim: UTF8STRING        :Wroclaw University of Science and Technology
  396:d=3  hl=2 l=  16 cons: SET
  398:d=4  hl=2 l=  14 cons: SEQUENCE
  400:d=5  hl=2 l=   3 prim: OBJECT            :organizationalUnitName
  405:d=5  hl=2 l=   7 prim: UTF8STRING        :WPPT/K2
  414:d=3  hl=2 l=  46 cons: SET
  416:d=4  hl=2 l=  44 cons: SEQUENCE
  418:d=5  hl=2 l=   3 prim: OBJECT            :commonName
  423:d=5  hl=2 l=  37 prim: UTF8STRING        :System Security I - IntermediateCA1.1
  462:d=3  hl=2 l=  43 cons: SET
  464:d=4  hl=2 l=  41 cons: SEQUENCE
  466:d=5  hl=2 l=   9 prim: OBJECT            :emailAddress
  477:d=5  hl=2 l=  28 prim: IA5STRING         :przemyslaw.kubiak@pwr.edu.pl
  507:d=2  hl=2 l= 122 cons: SEQUENCE
  509:d=3  hl=2 l=  20 cons: SEQUENCE
  511:d=4  hl=2 l=   7 prim: OBJECT            :id-ecPublicKey
  520:d=4  hl=2 l=   9 prim: OBJECT            :brainpoolP384r1
  531:d=3  hl=2 l=  98 prim: BIT STRING
  631:d=2  hl=2 l= 102 cons: cont [ 3 ]
  633:d=3  hl=2 l= 100 cons: SEQUENCE
  635:d=4  hl=2 l=  29 cons: SEQUENCE
  637:d=5  hl=2 l=   3 prim: OBJECT            :X509v3 Subject Key Identifier
  642:d=5  hl=2 l=  22 prim: OCTET STRING      [HEX DUMP]:0414FF8BC38D6257238D9C738A0F4DDB5BA6DB4A37B0
  666:d=4  hl=2 l=  31 cons: SEQUENCE
  668:d=5  hl=2 l=   3 prim: OBJECT            :X509v3 Authority Key Identifier
  673:d=5  hl=2 l=  24 prim: OCTET STRING      [HEX DUMP]:3016801434E56487E8B774DC574A07AE3D69CBD46219FA36
  699:d=4  hl=2 l=  18 cons: SEQUENCE
  701:d=5  hl=2 l=   3 prim: OBJECT            :X509v3 Basic Constraints
  706:d=5  hl=2 l=   1 prim: BOOLEAN           :255
  709:d=5  hl=2 l=   8 prim: OCTET STRING      [HEX DUMP]:30060101FF020102
  719:d=4  hl=2 l=  14 cons: SEQUENCE
  721:d=5  hl=2 l=   3 prim: OBJECT            :X509v3 Key Usage
  726:d=5  hl=2 l=   1 prim: BOOLEAN           :255
  729:d=5  hl=2 l=   4 prim: OCTET STRING      [HEX DUMP]:03020186
  735:d=1  hl=2 l=  10 cons: SEQUENCE
  737:d=2  hl=2 l=   8 prim: OBJECT            :ecdsa-with-SHA384
  747:d=1  hl=3 l= 136 prim: BIT STRING
```

Now its time to remove the file and free memory:

```
(gdb)shell

rm tempMemDump.asn1
exit

(gdb) call OPENSSL_free({unsigned char*}0x7ffff79f9e90)
(gdb) call free($12)
```

But we can utilize a simpler way to examine a certificate:

```
(gdb) p X509_get_subject_name(sk_X509_value(untrusted, 0))
$21 = (X509_NAME *) 0x555555648d00

(gdb) p X509_NAME_entry_count($21)
$22 = 6

(gdb) p X509_NAME_get_entry($21, 2)
$23 = (X509_NAME_ENTRY *) 0x555555649290

(gdb) p X509_NAME_ENTRY_get_data($23)
(gdb) $24 = (ASN1_STRING *) 0x555555649000

(gdb) p ASN1_STRING_data($24)
$33 = (unsigned char *) 0x555555648970 "Wroclaw University of Science and Technology"
```

The same effect:

```
(gdb) p ASN1_STRING_data(X509_NAME_ENTRY_get_data(X509_NAME_get_entry($21,2)))
$33 = (unsigned char *) 0x555555648970 "Wroclaw University of Science and Technology"
```

So it is easy to enumerate

```
(gdb) p ASN1_STRING_data(X509_NAME_ENTRY_get_data(X509_NAME_get_entry($21,0)))
$34 = (unsigned char *) 0x555555648d90 "PL"
(gdb) p ASN1_STRING_data(X509_NAME_ENTRY_get_data(X509_NAME_get_entry($21,1)))
$35 = (unsigned char *) 0x555555648d30 "Lower Silesian Voivodeship"
(gdb) p ASN1_STRING_data(X509_NAME_ENTRY_get_data(X509_NAME_get_entry($21,3)))
$36 = (unsigned char *) 0x5555556490c0 "WPPT/K2"
(gdb) p ASN1_STRING_data(X509_NAME_ENTRY_get_data(X509_NAME_get_entry($21,4)))
$37 = (unsigned char *) 0x5555556482b0 "System Security I - IntermediateCA1.1"
(gdb) p ASN1_STRING_data(X509_NAME_ENTRY_get_data(X509_NAME_get_entry($21,5)))
$38 = (unsigned char *) 0x555555648a50 "przemyslaw.kubiak@pwr.edu.pl"
```

Assuming that each certificate's DN has the same number of components (it is not the case in our example) we evaluate the following expressions:

```
(gdb) p ASN1_STRING_data(X509_NAME_ENTRY_get_data(X509_NAME_get_entry(X509_get_subject_name(sk_X509_value(untrusted, 1)),5)))
$40 = (unsigned char *) 0x55555564b6e0 "System Security I - IntermediateCA2.2"

(gdb) p ASN1_STRING_data(X509_NAME_ENTRY_get_data(X509_NAME_get_entry(X509_get_subject_name(sk_X509_value(untrusted, 2)),5)))
$41 = (unsigned char *) 0x55555564b2c0 "System Security I - IntermediateCA1.3"

(gdb) p ASN1_STRING_data(X509_NAME_ENTRY_get_data(X509_NAME_get_entry(X509_get_subject_name(sk_X509_value(untrusted, 3)),5)))
$42 = (unsigned char *) 0x55555564dd00 "System Security I - IntermediateCA1.3"

(gdb) p ASN1_STRING_data(X509_NAME_ENTRY_get_data(X509_NAME_get_entry(X509_get_subject_name(sk_X509_value(untrusted, 4)),5)))
$43 = (unsigned char *) 0x555555650610 "System Security I - IntermediateCA1.2"

(gdb) p ASN1_STRING_data(X509_NAME_ENTRY_get_data(X509_NAME_get_entry(X509_get_subject_name(sk_X509_value(untrusted, 5)),4)))
$46 = (unsigned char *) 0x555555652760 "System Security I - IntermediateCA2.1"
```

Since (untrusted, 2), (untrusted, 3) have the same subject, let us examine the issuer of those certificates:

```
(gdb) p ASN1_STRING_data(X509_NAME_ENTRY_get_data(X509_NAME_get_entry(X509_get_issuer_name(sk_X509_value(untrusted, 2)),5)))
$47 = (unsigned char *) 0x55555564b1c0 "System Security I - IntermediateCA1.2"

(gdb) p ASN1_STRING_data(X509_NAME_ENTRY_get_data(X509_NAME_get_entry(X509_get_issuer_name(sk_X509_value(untrusted, 3)),5)))
$48 = (unsigned char *) 0x55555564eee0 "System Security I - IntermediateCA2.2"
```

Ok, we have 6 different certificates – indeed all intermediate ones.

3.5 Make the diagnosis

Let us continue debugging – the output is self-explanatory:

```
(gdb) l
180
181         if (crl_download)
182             store_setup_crl_download(store);
183
184         ret = 0;
185         if (argc < 1) {
186             if (check(store, NULL, untrusted, trusted, crls, show_chain) != 1)
187                 ret = -1;
188         } else {
189             for (i = 0; i < argc; i++)
(gdb)

(gdb) n
189             for (i = 0; i < argc; i++)
(gdb) l
184         ret = 0;
185         if (argc < 1) {
186             if (check(store, NULL, untrusted, trusted, crls, show_chain) != 1)
187                 ret = -1;
188         } else {
189             for (i = 0; i < argc; i++)
190                 if (check(store, argv[i], untrusted, trusted, crls,
191                           show_chain) != 1)
192                     ret = -1;
193         }
(gdb) s
190                 if (check(store, argv[i], untrusted, trusted, crls,
(gdb) s
check (ctx=0x55555564b4d0, file=0x7fffffffe14e "../client-certs/EndEntity_Cert.pem", uchain=0x5555556528e0,
```

```
                                tchain=0x5555556515e0, crls=0x0, show_chain=1) at apps/verify.c:209
209         X509 *x = NULL;
(gdb) bt
#0  check (ctx=0x55555564b4d0, file=0x7fffffffe14e "../client-certs/EndEntity_Cert.pem", uchain=0x5555556528e0,
                                tchain=0x5555556515e0, crls=0x0, show_chain=1) at apps/verify.c:209
#1  0x00005555555d8bba in verify_main (argc=1, argv=0x7fffffffdd08) at apps/verify.c:190
#2  0x00005555555a36e0 in do_cmd (prog=0x555555644680, argc=8, argv=0x7fffffffdcd0) at apps/openssl.c:564
#3  0x00005555555a2aa4 in main (argc=8, argv=0x7fffffffdcd0) at apps/openssl.c:183
(gdb) l
204
205     static int check(X509_STORE *ctx, const char *file,
206                      STACK_OF(X509) *uchain, STACK_OF(X509) *tchain,
207                      STACK_OF(X509_CRL) *crls, int show_chain)
208     {
209         X509 *x = NULL;
210         int i = 0, ret = 0;
211         X509_STORE_CTX *csc;
212         STACK_OF(X509) *chain = NULL;
213         int num_untrusted;
(gdb) l
214
215         x = load_cert(file, FORMAT_PEM, "certificate file");
216         if (x == NULL)
217             goto end;
218
219         csc = X509_STORE_CTX_new();
220         if (csc == NULL) {
221             printf("error %s: X.509 store context allocation failed\n",
222                    (file == NULL) ? "stdin" : file);
223             goto end;
(gdb) n
210         int i = 0, ret = 0;
(gdb) n
212         STACK_OF(X509) *chain = NULL;
(gdb) n
215         x = load_cert(file, FORMAT_PEM, "certificate file");
(gdb) n
216         if (x == NULL)
(gdb) p x
$49 = (X509 *) 0x555555653320
(gdb) n
219         csc = X509_STORE_CTX_new();
(gdb) n
220         if (csc == NULL) {
(gdb) p csc
$50 = (X509_STORE_CTX *) 0x555555646640
(gdb) n
226         X509_STORE_set_flags(ctx, vflags);
(gdb) l
221             printf("error %s: X.509 store context allocation failed\n",
222                    (file == NULL) ? "stdin" : file);
223             goto end;
224         }
225
226         X509_STORE_set_flags(ctx, vflags);
227         if (!X509_STORE_CTX_init(csc, ctx, x, uchain)) {
228             X509_STORE_CTX_free(csc);
229             printf("error %s: X.509 store context initialization failed\n",
230                    (file == NULL) ? "stdin" : file);
(gdb) p vflags
$51 = 0
(gdb) s
X509_STORE_set_flags (ctx=0x55555564b4d0, flags=0) at crypto/x509/x509_lu.c:725
725         return X509_VERIFY_PARAM_set_flags(ctx->param, flags);
(gdb) s
X509_VERIFY_PARAM_set_flags (param=0x555555646210, flags=0) at crypto/x509/x509_vpm.c:272
272         param->flags |= flags;
(gdb) n
273         if (flags & X509_V_FLAG_POLICY_MASK)
(gdb) n
275         return 1;
(gdb) n
276     }
(gdb) n
X509_STORE_set_flags (ctx=0x55555564b4d0, flags=0) at crypto/x509/x509_lu.c:726
726     }
(gdb) n
check (ctx=0x55555564b4d0, file=0x7fffffffe14e "../client-certs/EndEntity_Cert.pem", uchain=0x5555556528e0,
                        tchain=0x5555556515e0, crls=0x0, show_chain=1) at apps/verify.c:227
227         if (!X509_STORE_CTX_init(csc, ctx, x, uchain)) {
(gdb) s
X509_STORE_CTX_init (ctx=0x555555646640, store=0x55555564b4d0, x509=0x555555653320,
                        chain=0x5555556528e0) at crypto/x509/x509_vfy.c:2202
2202        int ret = 1;
(gdb) l
2197    }
2198
2199    int X509_STORE_CTX_init(X509_STORE_CTX *ctx, X509_STORE *store, X509 *x509,
2200                            STACK_OF(X509) *chain)
2201    {
2202        int ret = 1;
2203
2204        ctx->ctx = store;
```

```
2205        ctx->cert = x509;
2206        ctx->untrusted = chain;
(gdb) l
2207        ctx->crls = NULL;
2208        ctx->num_untrusted = 0;
2209        ctx->other_ctx = NULL;
2210        ctx->valid = 0;
2211        ctx->chain = NULL;
2212        ctx->error = 0;
2213        ctx->explicit_policy = 0;
2214        ctx->error_depth = 0;
2215        ctx->current_cert = NULL;
2216        ctx->current_issuer = NULL;
(gdb) l
2217        ctx->current_crl = NULL;
2218        ctx->current_crl_score = 0;
2219        ctx->current_reasons = 0;
2220        ctx->tree = NULL;
2221        ctx->parent = NULL;
2222        ctx->dane = NULL;
2223        ctx->bare_ta_signed = 0;
2224        /* Zero ex_data to make sure we're cleanup-safe */
2225        memset(&ctx->ex_data, 0, sizeof(ctx->ex_data));
2226
(gdb) l
2227        /* store->cleanup is always 0 in OpenSSL, if set must be idempotent */
2228        if (store)
2229            ctx->cleanup = store->cleanup;
2230        else
2231            ctx->cleanup = 0;
2232
2233        if (store && store->check_issued)
2234            ctx->check_issued = store->check_issued;
2235        else
2236            ctx->check_issued = check_issued;
(gdb)

(gdb) finish
Run till exit from #0  X509_STORE_CTX_init (ctx=0x555555646640, store=0x55555564b4d0,
                          x509=0x555555653320, chain=0x5555556528e0) at crypto/x509/x509_vfy.c:2202
check (ctx=0x55555564b4d0, file=0x7fffffffe14e "../client-certs/EndEntity_Cert.pem", uchain=0x5555556528e0,
                           tchain=0x5555556515e0, crls=0x0, show_chain=1) at apps/verify.c:227
227         if (!X509_STORE_CTX_init(csc, ctx, x, uchain)) {
Value returned is $52 = 1
(gdb) n
233         if (tchain != NULL)
(gdb) n
234             X509_STORE_CTX_set0_trusted_stack(csc, tchain);
(gdb) s
X509_STORE_CTX_set0_trusted_stack (ctx=0x555555646640, sk=0x5555556515e0) at crypto/x509/x509_vfy.c:2343
2343        ctx->other_ctx = sk;
(gdb) n
2344        ctx->get_issuer = get_issuer_sk;
(gdb) p get_issuer
$53 = {int (X509 **, X509_STORE_CTX *, X509 *)} 0x7ffff7e30033 <get_issuer>
(gdb) n
2345        ctx->lookup_certs = lookup_certs_sk;
(gdb) n
2346    }
(gdb) n
check (ctx=0x55555564b4d0, file=0x7fffffffe14e "../client-certs/EndEntity_Cert.pem", uchain=0x5555556528e0,
                           tchain=0x5555556515e0, crls=0x0, show_chain=1) at apps/verify.c:235
235         if (crls != NULL)
(gdb) n
237         i = X509_verify_cert(csc);
(gdb) s
X509_verify_cert (ctx=0x555555646640) at crypto/x509/x509_vfy.c:255
255         SSL_DANE *dane = ctx->dane;
(gdb) bt
#0  X509_verify_cert (ctx=0x555555646640) at crypto/x509/x509_vfy.c:255
#1  0x00005555555d8dd1 in check (ctx=0x55555564b4d0, file=0x7fffffffe14e "../client-certs/EndEntity_Cert.pem",
                                 uchain=0x5555556528e0, tchain=0x5555556515e0, crls=0x0, show_chain=1) at apps/verify.c:237
#2  0x00005555555d8bba in verify_main (argc=1, argv=0x7fffffffdd08) at apps/verify.c:190
#3  0x00005555555a36e0 in do_cmd (prog=0x555555644680, argc=8, argv=0x7fffffffdcd0) at apps/openssl.c:564
#4  0x00005555555a2aa4 in main (argc=8, argv=0x7fffffffdcd0) at apps/openssl.c:183
(gdb) l
250         return ok;
251     }
252
253     int X509_verify_cert(X509_STORE_CTX *ctx)
254     {
255         SSL_DANE *dane = ctx->dane;
256         int ret;
257
258         if (ctx->cert == NULL) {
259             X509err(X509_F_X509_VERIFY_CERT, X509_R_NO_CERT_SET_FOR_US_TO_VERIFY);
(gdb) l
260             ctx->error = X509_V_ERR_INVALID_CALL;
261             return -1;
262         }
263
264         if (ctx->chain != NULL) {
265             /*
```

```
266              * This X509_STORE_CTX has already been used to verify a cert. We
267              * cannot do another one.
268              */
269             X509err(X509_F_X509_VERIFY_CERT, ERR_R_SHOULD_NOT_HAVE_BEEN_CALLED);
(gdb) p ctx->dane
$54 = (SSL_DANE *) 0x0
(gdb) p ctx->chain
$55 = (struct stack_st_X509 *) 0x0
(gdb) n
258         if (ctx->cert == NULL) {
(gdb) n
264         if (ctx->chain != NULL) {
(gdb) n
278         if (((ctx->chain = sk_X509_new_null()) == NULL) ||
(gdb) n
279             (!sk_X509_push(ctx->chain, ctx->cert))) {
(gdb) n
278         if (((ctx->chain = sk_X509_new_null()) == NULL) ||
(gdb) l
273
274         /*
275          * first we make sure the chain we are going to build is present and that
276          * the first entry is in place
277          */
278         if (((ctx->chain = sk_X509_new_null()) == NULL) ||
279             (!sk_X509_push(ctx->chain, ctx->cert))) {
280             X509err(X509_F_X509_VERIFY_CERT, ERR_R_MALLOC_FAILURE);
281             ctx->error = X509_V_ERR_OUT_OF_MEM;
282             return -1;
(gdb) l
283         }
284         X509_up_ref(ctx->cert);
285         ctx->num_untrusted = 1;
286
287         /* If the peer's public key is too weak, we can stop early. */
288         if (!check_key_level(ctx, ctx->cert) &&
289             !verify_cb_cert(ctx, ctx->cert, 0, X509_V_ERR_EE_KEY_TOO_SMALL))
290             return 0;
291
292         if (DANETLS_ENABLED(dane))
```

See that `ctx->chain` contains `ctx->cert`, and initially the chain is untrusted.

```
(gdb) p sk_X509_value(ctx->chain, 0)
$57 = (X509 *) 0x555555653320
(gdb) p ctx->cert
$58 = (X509 *) 0x555555653320
(gdb) whatis ctx
type = X509_STORE_CTX *
(gdb)
```

```
(gdb) s
284         X509_up_ref(ctx->cert);
(gdb) s
X509_up_ref (x=0x555555653320) at crypto/x509/x509_set.c:100
100     {
(gdb) l
95              return 0;
96          return X509_PUBKEY_set(&(x->cert_info.key), pkey);
97      }
98
99      int X509_up_ref(X509 *x)
100     {
101         int i;
102
103         if (CRYPTO_UP_REF(&x->references, &i, x->lock) <= 0)
104             return 0;
(gdb) l
105
106         REF_PRINT_COUNT("X509", x);
107         REF_ASSERT_ISNT(i < 2);
108         return ((i > 1) ? 1 : 0);
109     }
110
111     long X509_get_version(const X509 *x)
112     {
113         return ASN1_INTEGER_get(x->cert_info.version);
114     }
(gdb) p x->references
$60 = 1
(gdb) p x->lock
$61 = (CRYPTO_RWLOCK *) 0x555555654ef0
(gdb) n
103         if (CRYPTO_UP_REF(&x->references, &i, x->lock) <= 0)
(gdb) n
107         REF_ASSERT_ISNT(i < 2);
(gdb)
(gdb) p i
$62 = 2
(gdb)
```

```
(gdb) p x->references
$63 = 2
```

So it seems that the number of references to an object x has incremented

```
(gdb) n
108         return ((i > 1) ? 1 : 0);
(gdb) n
109     }
(gdb) n
X509_verify_cert (ctx=0x555555646640) at crypto/x509/x509_vfy.c:285
285         ctx->num_untrusted = 1;
(gdb) n
288         if (!check_key_level(ctx, ctx->cert) &&
(gdb) l
283         }
284         X509_up_ref(ctx->cert);
285         ctx->num_untrusted = 1;
286
287         /* If the peer's public key is too weak, we can stop early. */
288         if (!check_key_level(ctx, ctx->cert) &&
289             !verify_cb_cert(ctx, ctx->cert, 0, X509_V_ERR_EE_KEY_TOO_SMALL))
290             return 0;
291
292         if (DANETLS_ENABLED(dane))

(gdb) p ctx->other_ctx
$64 = (void *) 0x5555556515e0
(gdb) p ASN1_STRING_data(X509_NAME_ENTRY_get_data(X509_NAME_get_entry(X509_get_subject_name(sk_X509_value(ctx->other_ctx, 0)),4)))
$66 = (unsigned char *) 0x5555556507b0 "System Security I - RootCA2"
```

The field `ctx->other_ctx` contains the trusted stack. Interestingly, if we use trusted stack (`ctx->other_ctx`) some functions are set to the default ones (the default utilize `ctx->other_ctx`). See x509_vfy.c:

```
/*
 * Set alternative lookup method: just a STACK of trusted certificates. This
 * avoids X509_STORE nastiness where it isn't needed.
 */
void X509_STORE_CTX_set0_trusted_stack(X509_STORE_CTX *ctx, STACK_OF(X509) *sk)
{
    ctx->other_ctx = sk;
    ctx->get_issuer = get_issuer_sk;
    ctx->lookup_certs = lookup_certs_sk;
}
```

```
(gdb) bt
#0  X509_verify_cert (ctx=0x555555646640) at crypto/x509/x509_vfy.c:288
#1  0x00005555555d8dd1 in check (ctx=0x55555564b4d0, file=0x7fffffffe14e "../client-certs/EndEntity_Cert.pem",
                                 uchain=0x5555556528e0, tchain=0x5555556515e0, crls=0x0, show_chain=1) at apps/verify.c:237
#2  0x00005555555d8bba in verify_main (argc=1, argv=0x7fffffffdd08) at apps/verify.c:190
#3  0x00005555555a36e0 in do_cmd (prog=0x555555644680, argc=8, argv=0x7fffffffdcd0) at apps/openssl.c:564
#4  0x00005555555a2aa4 in main (argc=8, argv=0x7fffffffdcd0) at apps/openssl.c:183
(gdb) n
292         if (DANETLS_ENABLED(dane))
(gdb) n
295             ret = verify_chain(ctx);
(gdb) s
verify_chain (ctx=0x555555646640) at crypto/x509/x509_vfy.c:216
216         if ((ok = build_chain(ctx)) == 0 ||
(gdb) bt
#0  verify_chain (ctx=0x555555646640) at crypto/x509/x509_vfy.c:216
#1  0x00007ffff7e2afe4 in X509_verify_cert (ctx=0x555555646640) at crypto/x509/x509_vfy.c:295
#2  0x00005555555d8dd1 in check (ctx=0x55555564b4d0, file=0x7fffffffe14e "../client-certs/EndEntity_Cert.pem",
                                 uchain=0x5555556528e0, tchain=0x5555556515e0, crls=0x0, show_chain=1) at apps/verify.c:237
#3  0x00005555555d8bba in verify_main (argc=1, argv=0x7fffffffdd08) at apps/verify.c:190
#4  0x00005555555a36e0 in do_cmd (prog=0x555555644680, argc=8, argv=0x7fffffffdcd0) at apps/openssl.c:564
#5  0x00005555555a2aa4 in main (argc=8, argv=0x7fffffffdcd0) at apps/openssl.c:183
(gdb) s
build_chain (ctx=0x555555646640) at crypto/x509/x509_vfy.c:2859
2859    {
(gdb) bt
#0  build_chain (ctx=0x555555646640) at crypto/x509/x509_vfy.c:2859
#1  0x00007ffff7e2ac7a in verify_chain (ctx=0x555555646640) at crypto/x509/x509_vfy.c:216
#2  0x00007ffff7e2afe4 in X509_verify_cert (ctx=0x555555646640) at crypto/x509/x509_vfy.c:295
#3  0x00005555555d8dd1 in check (ctx=0x55555564b4d0, file=0x7fffffffe14e "../client-certs/EndEntity_Cert.pem",
                                 uchain=0x5555556528e0, tchain=0x5555556515e0, crls=0x0, show_chain=1) at apps/verify.c:237
#4  0x00005555555d8bba in verify_main (argc=1, argv=0x7fffffffdd08) at apps/verify.c:190
#5  0x00005555555a36e0 in do_cmd (prog=0x555555644680, argc=8, argv=0x7fffffffdcd0) at apps/openssl.c:564
#6  0x00005555555a2aa4 in main (argc=8, argv=0x7fffffffdcd0) at apps/openssl.c:183
(gdb)
```

```
2859    {
(gdb) l
2854
2855        return ok;
2856    }
2857
```

```
2858    static int build_chain(X509_STORE_CTX *ctx)
2859    {
2860        SSL_DANE *dane = ctx->dane;
2861        int num = sk_X509_num(ctx->chain);
2862        X509 *cert = sk_X509_value(ctx->chain, num - 1);
2863        int ss = cert_self_signed(cert);
(gdb) n
2860        SSL_DANE *dane = ctx->dane;
(gdb) n
2861        int num = sk_X509_num(ctx->chain);
(gdb) n
2862        X509 *cert = sk_X509_value(ctx->chain, num - 1);
(gdb) p dane
$1 = (SSL_DANE *) 0x0
(gdb) p num
$2 = 1
(gdb) n
2863        int ss = cert_self_signed(cert);
(gdb) p ASN1_STRING_data(X509_NAME_ENTRY_get_data(X509_NAME_get_entry(X509_get_issuer_name(cert,5))))
Too few arguments in function call.
(gdb) p ASN1_STRING_data(X509_NAME_ENTRY_get_data(X509_NAME_get_entry(X509_get_issuer_name(cert),5)))
$3 = (unsigned char *) 0x555555653920 "System Security I - IntermediateCA1.3"
(gdb) p ASN1_STRING_data(X509_NAME_ENTRY_get_data(X509_NAME_get_entry(X509_get_issuer_name(cert),4)))
$4 = (unsigned char *) 0x555555653850 "WPPT/K2"
(gdb) p ASN1_STRING_data(X509_NAME_ENTRY_get_data(X509_NAME_get_entry(X509_get_subject_name(cert),5)))
$5 = (unsigned char *) 0x555555656610 "localhost"
(gdb) l
2858    static int build_chain(X509_STORE_CTX *ctx)
2859    {
2860        SSL_DANE *dane = ctx->dane;
2861        int num = sk_X509_num(ctx->chain);
2862        X509 *cert = sk_X509_value(ctx->chain, num - 1);
2863        int ss = cert_self_signed(cert);
2864        STACK_OF(X509) *sktmp = NULL;
2865        unsigned int search;
2866        int may_trusted = 0;
2867        int may_alternate = 0;
(gdb) n
2864        STACK_OF(X509) *sktmp = NULL;
(gdb) p ss
$6 = 0
(gdb) n
2866        int may_trusted = 0;
(gdb) n
2867        int may_alternate = 0;
(gdb) n
2868        int trust = X509_TRUST_UNTRUSTED;
(gdb) n
2869        int alt_untrusted = 0;
(gdb) p trust
$7 = 3
(gdb) p X509_TRUST_UNTRUSTED                                //interesting!!
$8 = 3
(gdb) n
2871        int ok = 0;
(gdb) n
2875        if (!ossl_assert(num == 1 && ctx->num_untrusted == num))  {
(gdb) l
2870        int depth;
2871        int ok = 0;
2872        int i;
2873
2874        /* Our chain starts with a single untrusted element. */
2875        if (!ossl_assert(num == 1 && ctx->num_untrusted == num))  {
2876            X509err(X509_F_BUILD_CHAIN, ERR_R_INTERNAL_ERROR);
2877            ctx->error = X509_V_ERR_UNSPECIFIED;
2878            return 0;
2879        }
(gdb) p ctx->num_untrusted
$9 = 1
(gdb) n
2891        search = (ctx->untrusted != NULL) ? S_DOUNTRUSTED : 0;
(gdb) l
2886         * If we're doing DANE and not doing PKIX-TA/PKIX-EE, we never look in the
2887         * trust_store, otherwise we might look there first.  If not trusted-first,
2888         * and alternate chains are not disabled, try building an alternate chain
2889         * if no luck with untrusted first.
2890         */
2891        search = (ctx->untrusted != NULL) ? S_DOUNTRUSTED : 0;
2892        if (DANETLS_HAS_PKIX(dane) || !DANETLS_HAS_DANE(dane)) {
2893            if (search == 0 || ctx->param->flags & X509_V_FLAG_TRUSTED_FIRST)
2894                search |= S_DOTRUSTED;
2895            else if (!(ctx->param->flags & X509_V_FLAG_NO_ALT_CHAINS))
(gdb) p S_DOUNTRUSTED
$10 = 1
(gdb) p/x X509_V_FLAG_NO_ALT_CHAINS
$13 = 0x100000
(gdb) p/x X509_V_FLAG_TRUSTED_FIRST
$14 = 0x8000
(gdb) n
2892        if (DANETLS_HAS_PKIX(dane) || !DANETLS_HAS_DANE(dane)) {
(gdb) n
2893            if (search == 0 || ctx->param->flags & X509_V_FLAG_TRUSTED_FIRST)
```

```
(gdb) p search
$15 = 1
(gdb) p/x ctx->param->flags
$16 = 0x8000
(gdb) n
2894                search |= S_DOTRUSTED;
(gdb) n
2897            may_trusted = 1;
(gdb) n
2905        if (ctx->untrusted && (sktmp = sk_X509_dup(ctx->untrusted)) == NULL) {
(gdb) p search
$1 = 3
(gdb) p may_alternate
$2 = 0
(gdb) n
2921        if (DANETLS_ENABLED(dane) && dane->certs != NULL) {
(gdb) p ASN1_STRING_data(X509_NAME_ENTRY_get_data(X509_NAME_get_entry(X509_get_subject_name(sk_X509_value(ctx->untrusted, 0)),4)))
$3 = (unsigned char *) 0x5555556482b0 "System Security I - IntermediateCA1.1"
(gdb) p ASN1_STRING_data(X509_NAME_ENTRY_get_data(X509_NAME_get_entry(X509_get_subject_name(sk_X509_value(ctx->untrusted, 1)),5)))
$4 = (unsigned char *) 0x55555564b6e0 "System Security I - IntermediateCA2.2"
(gdb) p ASN1_STRING_data(X509_NAME_ENTRY_get_data(X509_NAME_get_entry(X509_get_subject_name(sk_X509_value(sktmp, 0)),4)))
$5 = (unsigned char *) 0x5555556482b0 "System Security I - IntermediateCA1.1"
```

See that the addresses $3 and $5 are the same - so sktmp is a kind of a shallow copy.

```
(gdb) p ASN1_STRING_data(X509_NAME_ENTRY_get_data(X509_NAME_get_entry(X509_get_subject_name(sk_X509_value(sktmp, 1)),5)))
$6 = (unsigned char *) 0x55555564b6e0 "System Security I - IntermediateCA2.2"
2941        if (ctx->param->depth > INT_MAX/2)
(gdb) p INT_MAX
$7 = 2147483647
(gdb) p ctx->param->depth
$8 = 100
(gdb) n
2949        depth = ctx->param->depth + 1;
(gdb) n
2951        while (search != 0) {
(gdb) display/x search
2: /x search = 0x3
(gdb) n
2952            X509 *x;
(gdb) n
2953            X509 *xtmp = NULL;
2: /x search = 0x3
(gdb) n
2969            if ((search & S_DOTRUSTED) != 0) {
2: /x search = 0x3
(gdb) p/x S_DOTRUSTED
$9 = 0x2
(gdb) n
2970                i = num = sk_X509_num(ctx->chain);
2: /x search = 0x3
(gdb) display num
3: num = 1
(gdb) n
2971                if ((search & S_DOALTERNATE) != 0) {
2: /x search = 0x3
3: num = 1
(gdb) p sk_X509_num(ctx->chain)
$11 = 1
(gdb) p/x S_DOALTERNATE
$2 = 0x4
(gdb) n
2989                x = sk_X509_value(ctx->chain, i-1);
1: /x search = 0x3
(gdb) n
2991                ok = (depth < num) ? 0 : get_issuer(&xtmp, ctx, x);
1: /x search = 0x3
(gdb) p ASN1_STRING_data(X509_NAME_ENTRY_get_data(X509_NAME_get_entry(X509_get_subject_name(x),5)))
$4 = (unsigned char *) 0x555555656610 "localhost"
```

We may remove search variable from being displayed:

```
(gdb) info display
Auto-display expressions now in effect:
Num Enb Expression
1:   y  /x search
(gdb) undisplay 1
(gdb) info display
There are no auto-display expressions now.
(gdb)
```

Recall that line 2991 is to be executed...

```
(gdb) p depth
$5 = 101
(gdb) p num
$6 = 1
(gdb) n
2993                if (ok < 0) {
```

```
(gdb) p ok
$8 = 0
(gdb) n
3000                if (ok > 0) {
(gdb) p ok
$8 = 0
(gdb) n
3111                if ((search & S_DOUNTRUSTED) == 0) {
(gdb) p/x S_DOUNTRUSTED
$9 = 0x1
(gdb) n
3129            if ((search & S_DOUNTRUSTED) != 0) {
(gdb) n
3130                num = sk_X509_num(ctx->chain);
(gdb) n
3131                if (!ossl_assert(num == ctx->num_untrusted)) {
(gdb) p num
$11 = 1
(gdb) n
3138                x = sk_X509_value(ctx->chain, num-1);
(gdb) n
3144                xtmp = (ss || depth < num) ? NULL : find_issuer(ctx, sktmp, x);
(gdb) p ASN1_STRING_data(X509_NAME_ENTRY_get_data(X509_NAME_get_entry(X509_get_subject_name(x),5)))
$12 = (unsigned char *) 0x555555656610 "localhost"
(gdb) n
3145                if (xtmp == NULL) {
(gdb) n
3153                (void) sk_X509_delete_ptr(sktmp, xtmp);
(gdb) p ASN1_STRING_data(X509_NAME_ENTRY_get_data(X509_NAME_get_entry(X509_get_subject_name(xtmp),5)))
$13 = (unsigned char *) 0x55555564b2c0 "System Security I - IntermediateCA1.3"
(gdb) p ASN1_STRING_data(X509_NAME_ENTRY_get_data(X509_NAME_get_entry(X509_get_issuer_name(xtmp),5)))
$14 = (unsigned char *) 0x55555564b1c0 "System Security I - IntermediateCA1.2"
```

We have found the EE issuer's certificate – the first one, which is included in the chain ending with RootCA1, which in turn is non-trusted.

```
(gdb) n
3155                if (!sk_X509_push(ctx->chain, xtmp)) {
(gdb) n
3163                X509_up_ref(x = xtmp);
(gdb) n
3164                ++ctx->num_untrusted;
```

We have extended the chain by the IntermediateCA1.3 certificate.

```
(gdb) n
3165                ss = cert_self_signed(xtmp);
(gdb) n
3170                switch (trust = check_dane_issuer(ctx, ctx->num_untrusted - 1)) {
(gdb) p ss
$15 = 0
(gdb) n
2951        while (search != 0) {
(gdb) p trust
$16 = 3
(gdb) p search
$17 = 3
```

And continue with the loop...

```
(gdb) n
2953            X509 *xtmp = NULL;
(gdb) n
2969            if ((search & S_DOTRUSTED) != 0) {
(gdb) n
2970                i = num = sk_X509_num(ctx->chain);
(gdb) n

Breakpoint 2, build_chain (ctx=0x555555646640) at crypto/x509/x509_vfy.c:2971
2971                if ((search & S_DOALTERNATE) != 0) {
(gdb) p i
$18 = 2
(gdb) p/x S_DOALTERNATE
$19 = 0x4
(gdb) n
2989                x = sk_X509_value(ctx->chain, i-1);
(gdb)
2991                ok = (depth < num) ? 0 : get_issuer(&xtmp, ctx, x);
(gdb) p ASN1_STRING_data(X509_NAME_ENTRY_get_data(X509_NAME_get_entry(X509_get_subject_name(x),5)))
$20 = (unsigned char *) 0x55555564b2c0 "System Security I - IntermediateCA1.3"
(gdb) n
2993                if (ok < 0) {
(gdb) n
3000                if (ok > 0) {
(gdb) n
3111                if ((search & S_DOUNTRUSTED) == 0) {
(gdb) n
3129            if ((search & S_DOUNTRUSTED) != 0) {
```

```
(gdb) n
3130                num = sk_X509_num(ctx->chain);
(gdb) n
3131                if (!ossl_assert(num == ctx->num_untrusted)) {
(gdb) n
3138                x = sk_X509_value(ctx->chain, num-1);
(gdb) n
3144                xtmp = (ss || depth < num) ? NULL : find_issuer(ctx, sktmp, x);
(gdb) n
3145                if (xtmp == NULL) {
(gdb) p ASN1_STRING_data(X509_NAME_ENTRY_get_data(X509_NAME_get_entry(X509_get_subject_name(xtmp),5)))
$21 = (unsigned char *) 0x555555650610 "System Security I - IntermediateCA1.2"
(gdb) n
3153                (void) sk_X509_delete_ptr(sktmp, xtmp);
(gdb) n
3155                if (!sk_X509_push(ctx->chain, xtmp)) {
(gdb) n
3163                X509_up_ref(x = xtmp);
(gdb) n
3164                ++ctx->num_untrusted;
(gdb) n
3165                ss = cert_self_signed(xtmp);
(gdb) n
3170                switch (trust = check_dane_issuer(ctx, ctx->num_untrusted - 1)) {
(gdb) n
2951        while (search != 0) {
(gdb) n
2953            X509 *xtmp = NULL;
(gdb)
```

Again, we continue with the loop. Short analysis of the source code of the build_chain function shows that the only possibility to build an alternative chain to the one found at first is to utilize trusted certificate set. Since neither IntermediateCA1.3 nor IntermediateCA2.2 is not trusted there is no way to find alternative chain with the given untrusted set and the build_chain function.

```
(gdb) c
Continuing.

Breakpoint 2, build_chain (ctx=0x555555646640) at crypto/x509/x509_vfy.c:2971
2971                if ((search & S_DOALTERNATE) != 0) {
```

Let us list and clear the breakpoints:

```
(gdb) b
Note: breakpoint 2 also set at pc 0x7ffff7e30416.
Breakpoint 5 at 0x7ffff7e30416: file crypto/x509/x509_vfy.c, line 2971.
(gdb) clear
Deleted breakpoints 2 5
(gdb) b
Breakpoint 6 at 0x7ffff7e30416: file crypto/x509/x509_vfy.c, line 2971.
(gdb)

(gdb) b
Breakpoint 6 at 0x7ffff7e30416: file crypto/x509/x509_vfy.c, line 2971.
(gdb) clear
Deleted breakpoint 6
(gdb) b
Breakpoint 7 at 0x7ffff7e30416: file crypto/x509/x509_vfy.c, line 2971.
(gdb) clear
Deleted breakpoint 7
(gdb) b
Breakpoint 8 at 0x7ffff7e30416: file crypto/x509/x509_vfy.c, line 2971.
(gdb) c
Continuing.

Breakpoint 4, build_chain (ctx=0x555555646640) at crypto/x509/x509_vfy.c:2993
2993                if (ok < 0) {
(gdb) clear
Deleted breakpoint 4
(gdb) c
Continuing.

Breakpoint 8, build_chain (ctx=0x555555646640) at crypto/x509/x509_vfy.c:2971
2971                if ((search & S_DOALTERNATE) != 0) {
(gdb) clear
Deleted breakpoint 8
(gdb) c
Continuing.
C = PL, ST = Lower Silesian Voivodeship, O = Wroclaw University of Science and Technology,
    OU = WPPT/K2, CN = System Security I - IntermediateCA1.1, emailAddress = przemyslaw.kubiak@pwr.edu.pl
error 20 at 3 depth lookup: unable to get local issuer certificate
error ../client-certs/EndEntity_Cert.pem: verification failed
[Inferior 1 (process 4218) exited with code 02]
(gdb)
```

As we have seen, we can have multiple breapoints on the same line.

```
(gdb) q
```

Chapter 4

Extending the default implementation

We need to write a piece of code assembling different untrusted certificate sets, each set containing an alternative chain at the top. Our changes are going to be only the proof of concept, not a completely functional code. It is convenient to put our changes to the openssl-1.1.1b/apps/verify.c source file: we shall modify the static check function.
Below declaration

```
static int check(X509_STORE *ctx, const char *file,
                 STACK_OF(X509) *uchain, STACK_OF(X509) *tchain,
                 STACK_OF(X509_CRL) *crls, int show_chain);
```

we declare a new function

```
static int check_with_permuted_untrusted(X509_STORE *ctx, X509 *x,
                 STACK_OF(X509) *uchain, STACK_OF(X509) *tchain,
                 STACK_OF(X509_CRL) *crls, int show_chain, const char *file);
```

Initially the functions look as follows:

```
static int check(X509_STORE *ctx, const char *file,
                 STACK_OF(X509) *uchain, STACK_OF(X509) *tchain,
                 STACK_OF(X509_CRL) *crls, int show_chain)
{
    X509 *x = NULL;
    int ret = 0;

    x = load_cert(file, FORMAT_PEM, "certificate file");
    if (x == NULL)
        goto end;

    ret = check_with_permuted_untrusted(ctx, x, uchain, tchain, crls, show_chain, file);

 end:
    X509_free(x);

    return ret;
}


static int check_with_permuted_untrusted(X509_STORE *ctx, X509 *x,
                 STACK_OF(X509) *uchain, STACK_OF(X509) *tchain,
                 STACK_OF(X509_CRL) *crls, int show_chain, const char *file)
{
    int i = 0, ret = 0;
    X509_STORE_CTX *csc;
    STACK_OF(X509) *chain = NULL;
    int num_untrusted;

    csc = X509_STORE_CTX_new();
    if (csc == NULL) {
        printf("error %s: X.509 store context allocation failed\n",
               (file == NULL) ? "stdin" : file);
        goto end;
    }

    X509_STORE_set_flags(ctx, vflags);
    if (!X509_STORE_CTX_init(csc, ctx, x, uchain)) {
        X509_STORE_CTX_free(csc);
        printf("error %s: X.509 store context initialization failed\n",
               (file == NULL) ? "stdin" : file);
        goto end;
    }
    if (tchain != NULL)
```

```
        X509_STORE_CTX_set0_trusted_stack(csc, tchain);
    if (crls != NULL)
        X509_STORE_CTX_set0_crls(csc, crls);
    i = X509_verify_cert(csc);
    if (i > 0 && X509_STORE_CTX_get_error(csc) == X509_V_OK) {
        printf("%s: OK\n", (file == NULL) ? "stdin" : file);
        ret = 1;
        if (show_chain) {
            int j;

            chain = X509_STORE_CTX_get1_chain(csc);
            num_untrusted = X509_STORE_CTX_get_num_untrusted(csc);
            printf("Chain:\n");
            for (j = 0; j < sk_X509_num(chain); j++) {
                X509 *cert = sk_X509_value(chain, j);
                printf("depth=%d: ", j);
                X509_NAME_print_ex_fp(stdout,
                                      X509_get_subject_name(cert),
                                      0, get_nameopt());
                if (j < num_untrusted)
                    printf(" (untrusted)");
                printf("\n");
            }
            sk_X509_pop_free(chain, X509_free);
        }
    } else {
        printf("error %s: verification failed\n", (file == NULL) ? "stdin" : file);
    }
    X509_STORE_CTX_free(csc);

 end:
    if (i <= 0)
        ERR_print_errors(bio_err);

    return ret;
}
```

As we see the check function has become a simple wrapper reading the end entity certificate from the file and calling the core check_with_permuted_untrusted function.
Let re-build the changed file:

```
cd openssl-1.1.1b
make
```

```
make depend && make _all
make[1]: Entering directory '/SystemSecurity/openssl-working/openssl-1.1.1b'
make[1]: Leaving directory '/SystemSecurity/openssl-working/openssl-1.1.1b'
make[1]: Entering directory '/SystemSecurity/openssl-working/openssl-1.1.1b'
gcc  -I. -Iinclude -Iapps -pthread -m64 -Wall -O0 -g -g3 -ggdb -gdwarf-4 -fno-inline -O0 -fno-omit-frame-pointer \
    -MMD -MF apps/verify.d.tmp -MT apps/verify.o -c -o apps/verify.o apps/verify.c
rm -f apps/openssl
${LDCMD:-gcc} -pthread -m64 -Wall -O0 -g -g3 -ggdb -gdwarf-4 -fno-inline -O0 -fno-omit-frame-pointer -L.   \
        -o apps/openssl apps/asn1pars.o apps/ca.o apps/ciphers.o apps/cms.o apps/crl.o apps/crl2p7.o apps/dgst.o \
        apps/dhparam.o apps/dsa.o apps/dsaparam.o apps/ec.o apps/ecparam.o apps/enc.o apps/engine.o apps/errstr.o \
        apps/gendsa.o apps/genpkey.o apps/genrsa.o apps/nseq.o apps/ocsp.o apps/openssl.o apps/passwd.o \
        apps/pkcs12.o apps/pkcs7.o apps/pkcs8.o apps/pkey.o apps/pkeyparam.o apps/pkeyutl.o apps/prime.o \
        apps/rand.o apps/rehash.o apps/req.o apps/rsa.o apps/rsautl.o apps/s_client.o apps/s_server.o apps/s_time.o \
        apps/sess_id.o apps/smime.o apps/speed.o apps/spkac.o apps/srp.o apps/storeutl.o apps/ts.o apps/verify.o \
        apps/version.o apps/x509.o \
         apps/libapps.a -lssl -lcrypto -ldl -pthread
make[1]: Leaving directory '/SystemSecurity/openssl-working/openssl-1.1.1b'
```

Let us call

```
apps/openssl verify -verbose -show_chain -untrusted ../client-certs/untrusted_all_intermediate.pem \
    -trusted ../client-certs/RootCa2Cert.pem ../client-certs/EndEntity_Cert.pem
```

We got the same error:

```
C = PL, ST = Lower Silesian Voivodeship, O = Wroclaw University of Science and Technology,
    OU = WPPT/K2, CN = System Security I - IntermediateCA1.1, emailAddress = przemyslaw.kubiak@pwr.edu.pl
error 20 at 3 depth lookup: unable to get local issuer certificate
error ../client-certs/EndEntity_Cert.pem: verification failed
```

Hence everything seems to be ok.
Now we add two new functions:

```
static int is_copy_permuted(X509 *x, STACK_OF(X509) *uchain_copy, int *chain_storage);
static void build_next_chain(X509 *chain_top, STACK_OF(X509) *uchain_copy,
                             int chain_depth, int *chain_storage, int *next_chain_found);
```

and change the function

```
static int check(X509_STORE *ctx, const char *file,
                 STACK_OF(X509) *uchain, STACK_OF(X509) *tchain,
                 STACK_OF(X509_CRL) *crls, int show_chain);
```

The code mentioned is depicted below:

```
static int check(X509_STORE *ctx, const char *file,
                 STACK_OF(X509) *uchain, STACK_OF(X509) *tchain,
                 STACK_OF(X509_CRL) *crls, int show_chain)
{
    X509 *x = NULL;
    STACK_OF(X509) *uchain_copy = NULL;
    int *chain_storage;

    int num = sk_X509_num(uchain);
    int ret = 0;
    int copy_permuted = 0;

    chain_storage = (int*) malloc((num+1)*sizeof(int));
    for (int i=0; i <= num ; i++)
        chain_storage[i] = -1;

    x = load_cert(file, FORMAT_PEM, "certificate file");
    if (x == NULL)
        goto end;

    do {
        uchain_copy = sk_X509_dup(uchain);

        copy_permuted = is_copy_permuted(x, uchain_copy, chain_storage);
        ret = check_with_permuted_untrusted(ctx, x, uchain_copy, tchain, crls, show_chain, file);

        sk_X509_free(uchain_copy);

    } while (!(ret > 0) && (copy_permuted));

end:
    free(chain_storage);
    X509_free(x);

    return ret;
}


static int is_copy_permuted(X509 *x, STACK_OF(X509) *uchain_copy, int *chain_storage)
{
    int  next_chain_found = 0;
    int  i, num;

    build_next_chain(x, uchain_copy, 0, chain_storage, &next_chain_found);
    if (next_chain_found) {
        //copy all chain elements from uchain_copy to chain stack - from 0 to chain_depth
        STACK_OF(X509) *chain = sk_X509_new_null();

        i = 0;
        while (chain_storage[i] > -1) {
            sk_X509_unshift(chain, sk_X509_value(uchain_copy, chain_storage[i]));
            i++;
        }

        //remove all copied elements from uchain_copy
        num = sk_X509_num(chain);
        for (i=0; i < num ; i++)
            sk_X509_delete_ptr(uchain_copy, sk_X509_value(chain, i));

        //move all elements from the top of the chain stack to the top of uchain_copy and remove the chain stack
        for (i=0 ; i < num ; i++)
            sk_X509_unshift(uchain_copy, sk_X509_shift(chain));
        sk_X509_free(chain);
    }

    return  next_chain_found;
}


static void build_next_chain(X509 *chain_top, STACK_OF(X509) *uchain_copy,
                             int chain_depth, int *chain_storage, int *next_chain_found)
{//this function does not cover all possible cases
    X509   *current;
    int    i = chain_storage[chain_depth];
    int    num = sk_X509_num(uchain_copy);

    if (chain_depth == num) {
        return;
    }

    if ((i > -1) && (chain_storage[chain_depth+1] > -1)) { //go to the top layer of the chain
        current = sk_X509_value(uchain_copy, i);
        build_next_chain(current, uchain_copy, chain_depth + 1, chain_storage, next_chain_found);
    }

    if ((i==-1) || !(*next_chain_found)) {
```

```
        while(++i < num) {
            current = sk_X509_value(uchain_copy, i);
            if ((X509_check_issued(current, chain_top)==X509_V_OK) && (current != chain_top)) {
                (*next_chain_found) = 1;
                chain_storage[chain_depth] = i;
                build_next_chain(current, uchain_copy, chain_depth + 1, chain_storage, next_chain_found);
                return;
            }
        }
        //unsuccessful search on this layer for the given chain_top
        chain_storage[chain_depth] = -1;
    }
}
```

Task: Check with gdb how the code works. The application output is:

```
apps/openssl verify -verbose -show_chain -untrusted ../client-certs/untrusted_all_intermediate.pem \
    -trusted ../client-certs/RootCa2Cert.pem ../client-certs/EndEntity_Cert.pem
C = PL, ST = Lower Silesian Voivodeship, O = Wroclaw University of Science and Technology,
    OU = WPPT/K2, CN = System Security I - IntermediateCA1.1, emailAddress = przemyslaw.kubiak@pwr.edu.pl
error 20 at 3 depth lookup: unable to get local issuer certificate
error ../client-certs/EndEntity_Cert.pem: verification failed
../client-certs/EndEntity_Cert.pem: OK
Chain:
depth=0: C = PL, ST = Lower Silesian Voivodeship, L = Wroclaw, O = Wroclaw University of Science and Technology,
    OU = WPPT/K2, CN = localhost, emailAddress = przemyslaw.kubiak@pwr.edu.pl (untrusted)
depth=1: C = PL, ST = Lower Silesian Voivodeship, L = Wroclaw, O = Wroclaw University of Science and Technology,
    OU = WPPT/K2, CN = System Security I - IntermediateCA1.3, emailAddress = przemyslaw.kubiak@pwr.edu.pl (untrusted)
depth=2: C = PL, ST = Lower Silesian Voivodeship, L = Wroclaw, O = Wroclaw University of Science and Technology,
    OU = WPPT/K2, CN = System Security I - IntermediateCA2.2, emailAddress = przemyslaw.kubiak@pwr.edu.pl (untrusted)
depth=3: C = PL, ST = Lower Silesian Voivodeship, O = Wroclaw University of Science and Technology,
    OU = WPPT/K2, CN = System Security I - IntermediateCA2.1, emailAddress = przemyslaw.kubiak@pwr.edu.pl (untrusted)
depth=4: C = PL, ST = Lower Silesian Voivodeship, O = Wroclaw University of Science and Technology, O
    U = WPPT/K2, CN = System Security I - RootCA2, emailAddress = przemyslaw.kubiak@pwr.edu.pl
```

As we see two permutations of the untrusted set of certificates are verified. The first permutation is not successful, but the second is.