Annex 10A

(informative)

Example ONU authentication credentials

1.1 Introduction

As described in 11.2.2.1, the ONU has a built-in credential (the *Device Authentication Credential* or DAC) and can additionally be configured with an operator-provided credential (the *Network Authentication Credential* or NAC).

This annex provides example X.509 certificate credentials for each credential type along with some description of the required and optional elements. This content is informational only.

1.1.1 Software packages, versions, and environment

The examples presented in this annex rely on the following software packages:

- Python programming language, version 3.12.3, 64-bit (AMD64) (see https://www.python.org/)
- Python cryptographic library (module) '*cryptography'*, version 42.0.8 (see https://pypi.org/project/cryptography/)
- OpenSSL library, version 3.3.1 (see https://openssl.org/)
- Execution environment: Windows 10, 64 bit

The above software packages were used for illustrative purposes only. There is no requirement to rely on these software packages for any SIEPON-4 development or production workflows.

1.2 Device Authentication Credential (DAC)

1.2.1 Device Authentication Keypair (DAK)

The DAK requirements are specified in 11.2.2.1.1. The DAK may be generated as part of the DAC certificate generation, for example, by using the following code:

```
from cryptography.hazmat.backends import default_backend
from cryptography.hazmat.primitives import serialization
from cryptography.hazmat.primitives.asymmetric import ec

onu_private_key = ec.generate_private_key(ec.SECP384R1(), default_backend())
onu_public_key = onu_private_key.public_key()

onu_private_key_pem = onu_private_key.private_bytes(
    encoding=serialization.Encoding.PEM,
    format=serialization.PrivateFormat.PKCS8,
    encryption_algorithm=serialization.NoEncryption())

onu_public_key_pem = onu_public_key.public_bytes(
    encoding=serialization.Encoding.PEM,
    format=serialization.PublicFormat.SubjectPublicKeyInfo)
```

The above code produces a new unique private/public keypair on every execution. However, to achieve repeatability of generated DAC certificate values, the examples discussed in this annex rely on the following static key values generated by the above code:

```
onu_private_key_pem = """
-----BEGIN PRIVATE KEY-----
MIG2AgEAMBAGByqGSM49AgEGBSuBBAAiBIGeMIGbAgEBBDCw5yv+U9KtYCYUPUUJ
plNJe/L9/81XJljDYfnZV9i/o5R+0CKiYvhffSXqR21cXrehZANiAARgA9offJKP
2SqqC/6iY/WyOFU9X/wKuXxhweUwzDSD5GU3TqXEx3Wx72Xi4TPF4ofQb0WEESH1
NMQthEA1l9EFxaURBjaNgKnSgE4vvZXP3A3lIfZ9mgZeqoe4WjIEq/g=
-----END PRIVATE KEY-----
"""

onu_public_key_pem = """
-----BEGIN PUBLIC KEY-----
MHYwEAYHKoZIzj0CAQYFK4EEACIDYgAEYAPaH3ySj9kqqgv+omP1sjhVPV/8Crl8
YcHlMMw0g+RlN06lxMd1se9l4uEzxeKH0G9FhBEh9TTELYRANZfRBcWlEQY2jYCp
0oBOL72Vz9wN5SH2fZoGXqqHuFoyBKv4
-----END PUBLIC KEY-----
"""
```

1.2.2 DAC certificate generation

A DAC is an X.509 certificate. There is no requirement for the DAC to be generated by the ONU itself. The DAC may be created by the ONU manufacturer and is loaded into the ONU trust store as a PEM-encoded file. The following code illustrates a method to generate a DAC compliant with the requirements of 11.2.2.1.3:

```
from cryptography import x509
from cryptography.x509.oid import ObjectIdentifier
from cryptography.hazmat.backends import default_backend
from cryptography.hazmat.primitives import hashes, serialization
from datetime import datetime

################################################################################

def create_dac_cert(onu_id,
                    private_key_pem,
                    public_key_pem,
                    cert_serial_num,
                    country,
                    manufacturer_name,
                    model_name,
                    not_valid_before,
                    not_valid_after):

    private_key = serialization.load_pem_private_key(private_key_pem.encode(),
                                                     password = None,
                                                     backend = default_backend())

    public_key = serialization.load_pem_public_key(public_key_pem.encode())

    subject_name = x509.Name([
        x509.NameAttribute(x509.NameOID.COUNTRY_NAME, country),
        x509.NameAttribute(x509.NameOID.ORGANIZATION_NAME, manufacturer_name),
        x509.NameAttribute(x509.NameOID.ORGANIZATIONAL_UNIT_NAME, model_name),
        x509.NameAttribute(x509.NameOID.COMMON_NAME, f"SIEPON4_ONU_{onu_id}")])

    bc_extension = x509.BasicConstraints(ca=False, path_length=None)

    ski_extension = x509.SubjectKeyIdentifier.from_public_key(public_key)

    aki_extension = x509.AuthorityKeyIdentifier.from_issuer_subject_key_identifier(
                        x509.SubjectKeyIdentifier.from_public_key(public_key))

    # Create the SIEPON4 Credential Type extension field.
    # Credential type value 1 identifies the DAC (see 11.3.3.2)
```

```
    siepon4_cred_type_oid = ObjectIdentifier("1.3.111.2.1904.4.1.1")
    siepon4_cred_type_extension = x509.UnrecognizedExtension(siepon4_cred_type_oid,
                                                             value=b'\x01')

    dac_certificate = (
        x509.CertificateBuilder()
        .subject_name(subject_name)
        .issuer_name(subject_name)
        .public_key(public_key)
        .serial_number(cert_serial_num)
        .not_valid_before(not_valid_before)
        .not_valid_after(not_valid_after)
        .add_extension(bc_extension, critical=True)
        .add_extension(ski_extension, critical=False)
        .add_extension(aki_extension, critical=False)
        .add_extension(siepon4_cred_type_extension, critical=False)
        .sign(private_key, algorithm=hashes.SHA256())
    )
    return dac_certificate
```

1.2.3 Example DAC certificate

The following code illustrates DAC generation with a given set of example parameters values, including an example PEM-encoded value of the ONU's private/public keypair shown in 1.2.1. The ONU ID is assumed to equal 0x58-D0-8F-12-34-56. The DAC is then PEM-encoded and stored in a file named "*ONU_58D08F123456.dac.cert.pem*".

```
example_dac_certificate = create_dac_cert(
    onu_id = "58D08F123456",
    private_key_pem = onu_private_key_pem,
    public_key_pem = onu_public_key_pem,
    cert_serial_num = 159405573470659635927626028419554851843497597566,
    country = "US",
    manufacturer_name = "ACME Devices",
    model_name = "SuperONU 5000",
    not_valid_before = datetime.fromisoformat("2024-04-29 02:11:20+00:00"),
    not_valid_after = datetime.fromisoformat("2064-04-29 02:11:20+00:00")
)

with open("ONU_58D08F123456.dac.cert.pem", "wb") as file:
    file.write(example_dac_certificate.public_bytes(serialization.Encoding.PEM))
```

Executing the above code produces the following PEM-encoded X.509 certificate:

```
-----BEGIN CERTIFICATE-----
MIICXTCCAeSgAwIBAgIUG+v+PcLIS47ibUs8BuX0ybbRGn4wCgYIKoZIzj0EAwIw
XzELMAkGA1UEBhMCVVMxFTATBgNVBAoMDEFDTUUgRGV2aWNlczEWMBQGA1UECwwN
U3VwZXJPTlUgNTAwMDEhMB8GA1UEAwwYU0lFUE9ONF9PTlVfNThEMDhGMTIzNDU2
MCAXDTI0MDQyOTAyMTEyMFoYDzIwNjQwNDI5MDIxMTIwWjBfMQswCQYDVQQGEwJV
UzEVMBMGA1UECgwMQUNNRSBEZXZpY2VzMRYwFAYDVQQLDA1TdXBlck9OVSA1MDAw
MSEwHwYDVQQDDBhTSUVQT040X09OVV81OEQwOEYxMjM0NTYwdjAQBgcqhkjOPQIB
BgUrgQQAIgNiAARgA9offJKP2SqqC/6iY/WyOFU9X/wKuXxhweUwzDSD5GU3TqXE
x3Wx72Xi4TPF4ofQb0WEESH1NMQthEA1l9EFxaURBjaNgKnSgE4vvZXP3A3lIfZ9
mgZeqoe4WjIEq/ijXzBdMAwGA1UdEwEB/wQCMAAwHQYDVR0OBBYEFM4cvYPRsUY3
8oj0o3YZTFu6LwvPMB8GA1UdIwQYMBaAFM4cvYPRsUY38oj0o3YZTFu6LwvPMA0G
CCtvAo5wBAEBBAEBMAoGCCqGSM49BAMCA2cAMGQCMEgRBfgGcNvhWGEGjQmhD8Q+
mTKs+y5izA40iLqfookzM8ZTQMS6GgE27hZJWQ45HAIwSh9iP0gAJFxvAuAuWIiA
8IL9rI5zatIlx6bifXYzqO3XacmN+jDXEuiXgDO2Yks9
-----END CERTIFICATE-----
```

NOTE—The certificate data generated from this example code may differ from the above PEM-encoded certificate in two areas (shown on a darker background):

— For the ECDSA signature field, the ASN.1 encoding of *r* and *s* values requires prepending of a 0x00 octet if the first bit of *r* and/or *s* is 1. Thus, the total length of this example certificate may

vary between 609 and 611 octets. In PEM-encoded certificate, this variability results in 6th base64 digit taking one of three possible values: “T”, “j”, or “z”.

— The signature value (the last 139-142 base64 digits of the PEM-encoded certificate) does not match the PEM example since ECDSA signatures are seeded with a *nonce* – a one-time random value that makes every signature unique.

1.2.4 Parsed example X.509 DAC

Parsing of the generated DAC can be performed using the following command:

```
openssl x509 -in ONU_58D08F123456.dac.cert.pem -noout -text
```

which produces the example output below:

```
Certificate:
    Data:
        Version: 3 (0x2)
        Serial Number:
            1b:eb:fe:3d:c2:c8:4b:8e:e2:6d:4b:3c:06:e5:f4:c9:b6:d1:1a:7e
        Signature Algorithm: ecdsa-with-SHA256
        Issuer: C=US, O=ACME Devices, OU=SuperONU 5000, CN=SIEPON4_ONU_58D08F123456
        Validity
            Not Before: Apr 29 02:11:20 2024 GMT
            Not After : Apr 29 02:11:20 2064 GMT
        Subject: C=US, O=ACME Devices, OU=SuperONU 5000, CN=SIEPON4_ONU_58D08F123456
        Subject Public Key Info:
            Public Key Algorithm: id-ecPublicKey
                Public-Key: (384 bit)
                pub:
                    04:60:03:da:1f:7c:92:8f:d9:2a:aa:0b:fe:a2:63:
                    f5:b2:38:55:3d:5f:fc:0a:b9:7c:61:c1:e5:30:cc:
                    34:83:e4:65:37:4e:a5:c4:c7:75:b1:ef:65:e2:e1:
                    33:c5:e2:87:d0:6f:45:84:11:21:f5:34:c4:2d:84:
                    40:35:97:d1:05:c5:a5:11:06:36:8d:80:a9:d2:80:
                    4e:2f:bd:95:cf:dc:0d:e5:21:f6:7d:9a:06:5e:aa:
                    87:b8:5a:32:04:ab:f8
                ASN1 OID: secp384r1
                NIST CURVE: P-384
        X509v3 extensions:
            X509v3 Basic Constraints: critical
                CA:FALSE
            X509v3 Subject Key Identifier:
                CE:1C:BD:83:D1:B1:46:37:F2:88:F4:A3:76:19:4C:5B:BA:2F:0B:CF
            X509v3 Authority Key Identifier:
                CE:1C:BD:83:D1:B1:46:37:F2:88:F4:A3:76:19:4C:5B:BA:2F:0B:CF
            1.3.111.2.1904.4.1.1:
                .
    Signature Algorithm: ecdsa-with-SHA256
    Signature Value:
        30:64:02:30:48:11:05:f8:06:70:db:e1:58:61:06:8d:09:a1:
        0f:c4:3e:99:32:ac:fb:2e:62:cc:0e:34:88:ba:9f:a2:89:33:
        33:c6:53:40:c4:ba:1a:01:36:ee:16:49:59:0e:39:1c:02:30:
        4a:1f:62:3f:48:00:24:5c:6f:02:e0:2e:58:88:80:f0:82:fd:
        ac:8e:73:6a:d2:25:c7:a6:e2:7d:76:33:a8:ed:d7:69:c9:8d:
        fa:30:d7:12:e8:97:80:33:b6:62:4b:3d
```

Note that OID `1.3.111.2.1904.4.1.1` represents the *SIEPON4CredentialType* defined in 11.3.3.2 and the “.” represents the binary encoding of the type. In this case, the value is 0x01, indicating that this certificate is a DAC.

1.3 Network Authentication Credential (NAC)

The Network Authentication Credential (NAC) is an X.509 end-entity certificate containing the ONU's public key, operator-defined metadata, and a cryptographic signature used to verify the authenticity of the NAC (see 11.2.2.1.4).

The example presented in this subclause illustrates a X.509 certificate chain credential consisting of a sample operator-created NAC and a sample operator's Certificate Authority (CA) certificate. The NAC contains the same fields as the DAC, with the addition of a subscriber ID encoded into the X.509 *SubjectAlternativeName (SAN)* extension. (The operators may use other extensions to carry the subscriber ID or not include it in he NAC at all.)

Note that the creation of the NAC only requires access to the ONU's public key and metadata – all contained in the DAC. The ONU's private key is not required, since the NAC is signed by the network operator's Certificate Authority (CA).

1.3.1 NAC generation

The following code illustrates methods to generate a CA and a NAC certificates compliant with the requirements of 11.2.2.1.4, although it is noted that the generation of these certificates is not the responsibility of the OLT or the ONU devices.

```
from cryptography import x509
from cryptography.x509.oid import ObjectIdentifier, NameOID
from cryptography.hazmat.backends import default_backend
from cryptography.hazmat.primitives import serialization
from cryptography.hazmat.primitives.asymmetric import ec
from cryptography.hazmat.primitives import hashes
from datetime import datetime

################################################################################

def create_ca_cert(org_name,
                   common_name,
                   cert_serial_num,
                   ca_private_key_pem,
                   not_valid_before,
                   not_valid_after):

    ca_private_key = serialization.load_pem_private_key(ca_private_key_pem.encode(),
                                                        password=None,
                                                        backend=default_backend())
    ca_public_key = ca_private_key.public_key()

    ca_name = x509.Name([x509.NameAttribute(NameOID.ORGANIZATION_NAME, org_name),
                         x509.NameAttribute(NameOID.COMMON_NAME, common_name)])

    bc_extension = x509.BasicConstraints(ca=True, path_length=None)
    ski_extension = x509.SubjectKeyIdentifier.from_public_key(ca_public_key)

    ca_certificate = (
        x509.CertificateBuilder()
        .subject_name(ca_name)
        .issuer_name(ca_name)
        .public_key(ca_private_key.public_key())
        .serial_number(cert_serial_num)
        .not_valid_before(not_valid_before)
        .not_valid_after(not_valid_after)
        .add_extension(bc_extension, critical=True)
        .add_extension(ski_extension, critical=False)
        .sign(ca_private_key, hashes.SHA256(), default_backend())
    )

    return ca_certificate
```

```
################################################################################

def create_nac_cert(onu_id,
                    country,
                    issuer_name,
                    manufacturer_name,
                    model_name,
                    subscriber_id,
                    onu_public_key_pem,
                    cert_serial_num,
                    ca_private_key_pem,
                    not_valid_before,
                    not_valid_after):

    onu_public_key = serialization.load_pem_public_key(onu_public_key_pem.encode())
    ca_private_key = serialization.load_pem_private_key(ca_private_key_pem.encode(),
                                                        password=None,
                                                        backend=default_backend())
    subject_name = x509.Name([
        x509.NameAttribute(NameOID.COUNTRY_NAME, country),
        x509.NameAttribute(NameOID.ORGANIZATION_NAME, manufacturer_name),
        x509.NameAttribute(NameOID.ORGANIZATIONAL_UNIT_NAME, model_name),
        x509.NameAttribute(NameOID.COMMON_NAME, f"ONU_{onu_id}")])

    bc_extension = x509.BasicConstraints(ca=False, path_length=None)

    ski_extension = x509.SubjectKeyIdentifier.from_public_key(onu_public_key)

    aki_extension = x509.AuthorityKeyIdentifier.from_issuer_subject_key_identifier(
        x509.SubjectKeyIdentifier.from_public_key(ca_private_key.public_key()))

    san_extension = x509.SubjectAlternativeName([
        x509.UniformResourceIdentifier(f"urn:subscriber-id:{subscriber_id}")])

    # Create the SIEPON4 Credential Type extension field.
    # Credential type value 2 identifies the NAC (see 11.3.3.2)
    siepon4_cred_type_oid = ObjectIdentifier("1.3.111.2.1904.4.1.1")
    siepon4_cred_type_extension = x509.UnrecognizedExtension(siepon4_cred_type_oid,
                                                             value=b'\x02')
    nac_certificate = (
        x509.CertificateBuilder()
        .subject_name(subject_name)
        .issuer_name(issuer_name)
        .public_key(onu_public_key)
        .serial_number(cert_serial_num)
        .not_valid_before(not_valid_before)
        .not_valid_after(not_valid_after)
        .add_extension(bc_extension, critical=True)
        .add_extension(ski_extension, critical=False)
        .add_extension(aki_extension, critical=False)
        .add_extension(san_extension, critical=False)
        .add_extension(siepon4_cred_type_extension, critical=False)
        .sign(ca_private_key, hashes.SHA256(), default_backend())
    )
    return nac_certificate
```

1.3.2 Example NAC and intermediate certificates

The following code illustrates generation of an NAC and an intermediate certificate with a given set of example parameters values, including an example PEM-encoded value of the intermediate certificate private key and ONU's public key. The value of ONU's public key is assumed to have been extracted from ONU's DAC and its value is equal to `onu_public_key_pem` shown in 1.2.1. The NAC and the intermediate certificate are PEM-encoded and stored in a file named "*ONU_58D08F123456.nac.cert.pem*".

```
ca_private_key_pem = """
-----BEGIN PRIVATE KEY-----
```

```
MIGHAgEAMBMGByqGSM49AgEGCCqGSM49AwEHBG0wawIBAQQglSSQsWUVCOXT62uY
48RJ5deAVnHIzolj0GKwIwXJ8g6hRANCAATLg4YdH0Ba61zCIv0u60ALfbOBH6Ji
lKWoOBkAVRoKoOM2ScqCvcGjmCZQwFjVEmXyP0+9L/FnV3ptonQMJhJY
-----END PRIVATE KEY-----"""

# onu_public_key_pem = see 1.2.1

example_ca_certificate = create_ca_cert(
    org_name = "Example CA",
    common_name = "Example CA Certificate",
    cert_cerial_num = 0x6d17fc8ecdcd36c30d2c4c7e01a951ef,
    ca_private_key_pem = ca_private_key_pem,
    not_valid_before = datetime.fromisoformat("2022-01-07 00:00:05+00:00"),
    not_valid_after = datetime.fromisoformat("2032-01-07 00:00:05+00:00")
)

example_nac_certificate = create_nac_cert(
    onu_id = "58D08F123456",
    country = "US",
    issuer_name = example_ca_certificate.subject,
    onu_public_key_pem = onu_public_key_pem,
    cert_serial_num = 0x2de07ac8c6bc401cb15ee45106a4fd81,
    manufacturer_name = "ACME Devices",
    model_name = "SuperONU 5000",
    subscriber_id = "8765432-1A",
    ca_private_key_pem = ca_private_key_pem,
    not_valid_before = datetime.fromisoformat("2024-01-25 00:00:05"),
    not_valid_after = datetime.fromisoformat("2024-02-25 00:00:05")
)

with open("ONU_58D08F123456.nac.cert.pem", "wb") as file:
    file.write(example_nac_certificate.public_bytes(serialization.Encoding.PEM))
    file.write(example_ca_certificate.public_bytes(serialization.Encoding.PEM))
```

Executing the above code produces a PEM file containing two PEM-encoded X.509 certificate:

```
-----BEGIN CERTIFICATE-----
MIICMjCCAdigAwIBAgIQLeB6yMa8QByxXuRRBqT9gTAKBggqhkjOPQQDAjA2MRMw
EQYDVQQKDApFeGFtcGxlIENBMR8wHQYDVQQDDBZFeGFtcGxlIENBIENlcnRpZmlj
YXRlMB4XDTI0MDEyNTAwMDAwNVoXDTI0MDIyNTAwMDAwNVowVzELMAkGA1UEBhMC
VVMxFTATBgNVBAoMDEFDTUUgRGV2aWNlczEWMBQGA1UECwwNU3VwZXJPTlUgNTAw
MDEZMBcGA1UEAwwQT05VXzU4RDA4RjEyMzQ1NjB2MBAGByqGSM49AgEGBSuBBAAi
A2IABGAD2h98ko/ZKqoL/qJj9bI4VT1f/Aq5fGHB5TDMNIPkZTdOpcTHdbHvZeLh
M8Xih9BvRYQRIfU0xC2EQDWX0QXFpREGNo2AqdKATi+9lc/cDeUh9n2aBl6qh7ha
MgSr+KOBiTCBhjAMBgNVHRMBAf8EAjAAMB0GA1UdDgQWBBTOHL2D0bFGN/KI9KN2
GUxbui8LzzAfBgNVHSMEGDAWgBSE+5sKxs3oe5LxvNoW0x70MfYqSDAnBgNVHREE
IDAehhx1cm46c3Vic2NyaWJlci1pZDo4NzY1NDMyLTFBMA0GCCtvAo5wBAEBBAEC
MAoGCCqGSM49BAMCA0gAMEUCIQCEz+LMw6RaWnYAUFZkP4oNViM6E0ad/hBc3gC2
8Bz1GQIgGtT5cd5WZWosZoSZoGUIpttzL8vhyce2bAZHuoQLJ9g=
-----END CERTIFICATE-----
-----BEGIN CERTIFICATE-----
MIIBnDCCAUKgAwIBAgIQbRf8js3NNsMNLEx+AalR7zAKBggqhkjOPQQDAjA2MRMw
EQYDVQQKDApFeGFtcGxlIENBMR8wHQYDVQQDDBZFeGFtcGxlIENBIENlcnRpZmlj
YXRlMB4XDTIyMDEwNzAwMDAwNVoXDTMyMDEwNzAwMDAwNVowNjETMBEGA1UECgwK
RXhhbXBsZSBDQTEfMB0GA1UEAwwWRXhhbXBsZSBDQSBDZXJ0aWZpY2F0ZTBZMBMG
ByqGSM49AgEGCCqGSM49AwEHA0IABMuDhh0fQFrrXMIi/S7rQAt9s4EfomKUpag4
GQBVGgqg4zZJyoK9waOYJlDAWNUSZfI/T70v8WdXem2idAwmElijMjAwMA8GA1Ud
EwEB/wQFMAMBAf8wHQYDVR0OBBYEFIT7mwrGzeh7kvG82hbTHvQx9ipIMAoGCCqG
SM49BAMCA0gAMEUCIGCSh7c7XX75NybBBBSmPeyNQYUOc5vQJUrvIURQFYgwAiEA
iCMuEggIY8cS6UWHtWcoJg8Rzg9IwgaWwxRNzh4kSug=
-----END CERTIFICATE-----
```

NOTE—The certificate data generated from this example code may differ from the above PEM encoded certificates in two areas: the total certificate length and the certificate signature field. For additional details, see the note at the end of 1.2.3.

1.3.3 Parsed example X.509 NAC

Parsing of the generated NAC and the intermediate certificate can be performed using the following command:

```
openssl crl2pkcs7 -nocrl -certfile ONU_58D08F123456.nac.cert.pem | openssl pkcs7
-print_certs -noout -text
```

which produces the example output below:

```
Certificate:
    Data:
        Version: 3 (0x2)
        Serial Number:
            2d:e0:7a:c8:c6:bc:40:1c:b1:5e:e4:51:06:a4:fd:81
        Signature Algorithm: ecdsa-with-SHA256
        Issuer: O=Example CA, CN=Example CA Certificate
        Validity
            Not Before: Jan 25 00:00:05 2024 GMT
            Not After : Feb 25 00:00:05 2024 GMT
        Subject: C=US, O=ACME Devices, OU=SuperONU 5000, CN=ONU_58D08F123456
        Subject Public Key Info:
            Public Key Algorithm: id-ecPublicKey
                Public-Key: (384 bit)
                pub:
                    04:60:03:da:1f:7c:92:8f:d9:2a:aa:0b:fe:a2:63:
                    f5:b2:38:55:3d:5f:fc:0a:b9:7c:61:c1:e5:30:cc:
                    34:83:e4:65:37:4e:a5:c4:c7:75:b1:ef:65:e2:e1:
                    33:c5:e2:87:d0:6f:45:84:11:21:f5:34:c4:2d:84:
                    40:35:97:d1:05:c5:a5:11:06:36:8d:80:a9:d2:80:
                    4e:2f:bd:95:cf:dc:0d:e5:21:f6:7d:9a:06:5e:aa:
                    87:b8:5a:32:04:ab:f8
                ASN1 OID: secp384r1
                NIST CURVE: P-384
        X509v3 extensions:
            X509v3 Basic Constraints: critical
                CA:FALSE
            X509v3 Subject Key Identifier:
                CE:1C:BD:83:D1:B1:46:37:F2:88:F4:A3:76:19:4C:5B:BA:2F:0B:CF
            X509v3 Authority Key Identifier:
                84:FB:9B:0A:C6:CD:E8:7B:92:F1:BC:DA:16:D3:1E:F4:31:F6:2A:48
            X509v3 Subject Alternative Name:
                URI:urn:subscriber-id:8765432-1A
            1.3.111.2.1904.4.1.1:
                .
    Signature Algorithm: ecdsa-with-SHA256
    Signature Value:
        30:45:02:21:00:84:cf:e2:cc:c3:a4:5a:5a:76:00:50:56:64:
        3f:8a:0d:56:23:3a:13:46:9d:fe:10:5c:de:00:b6:f0:1c:f5:
        19:02:20:1a:d4:f9:71:de:56:65:6a:2c:66:84:99:a0:65:08:
        a6:db:73:2f:cb:e1:c9:c7:b6:6c:06:47:ba:84:0b:27:d8

Certificate:
    Data:
        Version: 3 (0x2)
        Serial Number:
            6d:17:fc:8e:cd:cd:36:c3:0d:2c:4c:7e:01:a9:51:ef
        Signature Algorithm: ecdsa-with-SHA256
        Issuer: O=Example CA, CN=Example CA Certificate
        Validity
            Not Before: Jan  7 00:00:05 2022 GMT
            Not After : Jan  7 00:00:05 2032 GMT
        Subject: O=Example CA, CN=Example CA Certificate
        Subject Public Key Info:
            Public Key Algorithm: id-ecPublicKey
                Public-Key: (256 bit)
                pub:
                    04:cb:83:86:1d:1f:40:5a:eb:5c:c2:22:fd:2e:eb:
                    40:0b:7d:b3:81:1f:a2:62:94:a5:a8:38:19:00:55:
```

```
                1a:0a:a0:e3:36:49:ca:82:bd:c1:a3:98:26:50:c0:
                58:d5:12:65:f2:3f:4f:bd:2f:f1:67:57:7a:6d:a2:
                74:0c:26:12:58
            ASN1 OID: prime256v1
            NIST CURVE: P-256
    X509v3 extensions:
        X509v3 Basic Constraints: critical
            CA:TRUE
        X509v3 Subject Key Identifier:
            84:FB:9B:0A:C6:CD:E8:7B:92:F1:BC:DA:16:D3:1E:F4:31:F6:2A:48
Signature Algorithm: ecdsa-with-SHA256
Signature Value:
    30:45:02:20:60:92:87:b7:3b:5d:7e:f9:37:26:c1:04:14:a6:
    3d:ec:8d:41:85:0e:73:9b:d0:25:4a:ef:21:44:50:15:88:30:
    02:21:00:88:23:2e:12:08:08:63:c7:12:e9:45:87:b5:67:28:
    26:0f:11:ce:0f:48:c2:06:96:c3:14:4d:ce:1e:24:4a:e8
```

Note that OID 1.3.111.2.1904.4.1.1 represents the *SIEPON4CredentialType* defined in 11.3.3.2 and the "." represents the binary encoding of the type. In this case, the value is 0x02, indicating that this certificate is a NAC.