Annex 10A

(informative)

Example ONU authentication credentials

10A.1 Introduction

As described in 11.3.3, 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 descrption of the required and optional elements. This content is informational only.

10A.2 DAC example 1

This example illustrates a basic X.509 DAC credential which meets the minimum requirements of 11.3.3.3 [The Device Authentication Credential].

10A.2.1 PEM-encoded X.509 credential – DAC example

```
-----BEGIN CERTIFICATE-----
MIICEDCCAbegAwIBAgIUB8XsFnLp1srJKz1OUQtFqMhsXWowCgYIKoZIzj0EAwIw
VzELMAkGA1UEBhMCVVMxFTATBgNVBAoMDEFDTUUgRGV2aWNlczEWMBQGA1UECwwN
U3VwZXJPTlUgNTAwMDEZMBcGA1UEAwwQT05VXzBBN0ZCNDlFMkNGMTAgFw0yNDA0
MjkwMjExMjBaGA8yMDY0MDQyOTAyMTEyMFowVzELMAkGA1UEBhMCVVMxFTATBgNV
BAoMDEFDTUUgRGV2aWNlczEWMBQGA1UECwwNU3VwZXJPTlUgNTAwMDEZMBcGA1UE
AwwQT05VXzBBN0ZCNDlFMkNGMTBZMBMGByqGSM49AgEGCCqGSM49AwEHA0IABMY/
OIoeAzAceUL9jOkZr3KqtDrwZDkPE7LxagHqLB0EGsXG5IhSyHGLZ6SVECNP6AZj
VG3w0qM17IGJZP2SkgCjXzBdMAwGA1UdEwEB/wQCMAAwHQYDVR0OBBYEFA/vr1e7
a4r5msFgBS+hzzbuorw8MB8GA1UdIwQYMBaAFA/vr1e7a4r5msFgBS+hzzbuorw8
MA0GCCtvAo5wBAEBBAEBMAoGCCqGSM49BAMCA0cAMEQCIEYAXb/xhhgwQtezgsMQ
FcSO6FNlVp4Qv164FeJpuKH/AiBdlHm0Lvhtf5auDSBA81jDZFRPUKwsCBb2U3ho
FPTNMw==
-----END CERTIFICATE-----
```

10A.2.2 Parsed X.509 credential – DAC example

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

```
openssl x509 -in ONU_0A7FB49E2CF1.dac.cert.pem -noout -text
```

which produces the example output below:

```
Certificate:
    Data:
        Version: 3 (0x2)
        Serial Number:
            07:c5:ec:16:72:e9:d6:ca:c9:2b:3d:4e:51:0b:45:a8:c8:6c:5d:6a
        Signature Algorithm: ecdsa-with-SHA256
        Issuer: C=US, O=ACME Devices, OU=SuperONU 5000, CN=ONU_0A7FB49E2CF1
        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=ONU_0A7FB49E2CF1
        Subject Public Key Info:
            Public Key Algorithm: id-ecPublicKey
                Public-Key: (256 bit)
                pub:
```

```
                    04:c6:3f:38:8a:1e:03:30:1c:79:42:fd:8c:e9:19:
                    af:72:aa:b4:3a:f0:64:39:0f:13:b2:f1:6a:01:ea:
                    2c:1d:04:1a:c5:c6:e4:88:52:c8:71:8b:67:a4:95:
                    10:23:4f:e8:06:63:54:6d:f0:d2:a3:35:ec:81:89:
                    64:fd:92:92:00
                ASN1 OID: prime256v1
                NIST CURVE: P-256
        X509v3 extensions:
            X509v3 Basic Constraints: critical
                CA:FALSE
            X509v3 Subject Key Identifier:
                0F:EF:AF:57:BB:6B:8A:F9:9A:C1:60:05:2F:A1:CF:36:EE:A2:BC:3C
            X509v3 Authority Key Identifier:
                0F:EF:AF:57:BB:6B:8A:F9:9A:C1:60:05:2F:A1:CF:36:EE:A2:BC:3C
            1.3.111.2.1904.4.1.1:
                .
    Signature Algorithm: ecdsa-with-SHA256
    Signature Value:
        30:44:02:20:46:00:5d:bf:f1:86:18:30:42:d7:b3:82:c3:10:
        15:c4:8e:e8:53:65:56:9e:10:bf:5e:b8:15:e2:69:b8:a1:ff:
        02:20:5d:94:79:b4:2e:f8:6d:7f:96:ae:0d:20:40:f3:58:c3:
        64:54:4f:50:ac:2c:08:16:f6:53:78:68:14:f4:cd:33
```

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, representing this certificate as a DAC. Once the OID is registered and recognized, OpenSSL will be able to parse the *SIEPON4CredentialType* and produce human-readable output.

10A.2.3 Sample Python 3 code – DAC example

```
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 cryptography.hazmat.primitives.asymmetric import ec
from datetime import datetime

siepon4_cred_type_val = {"DAC": b'\x01', "NAC": b'\x02'}

def create_dac_cert(onu_id, country, manufacturer_name, model_name,
                    device_key_id, creation_time, validity_years):
    device_pubkey = device_key_id.public_key()

    # Build the subject name
    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"ONU_{onu_id}"),
    ])

    # Create the SIEPON4CredentialType extension field
    siepon4_credential_id_ext \
        = x509.UnrecognizedExtension(oid=ObjectIdentifier("1.3.111.2.1904.4.1.1"),
                                     value=siepon4_cred_type_val['DAC'])
    dac_certificate = (
        x509.CertificateBuilder()
        .subject_name(subject_name)
        .issuer_name(subject_name)
        .public_key(device_pubkey)
        .serial_number(x509.random_serial_number())
        .not_valid_before(creation_time)
        .not_valid_after(creation_time.replace(year=creation_time.year
                                               + validity_years))
        .add_extension(x509.BasicConstraints(ca=False, path_length=None),
                       critical=True)
        .add_extension(x509.SubjectKeyIdentifier.from_public_key(device_pubkey),
                       critical=False)
        .add_extension(x509.AuthorityKeyIdentifier.from_issuer_subject_key_identifier(
```

```
                            x509.SubjectKeyIdentifier.from_public_key(device_pubkey)),
                        critical=False)
            .add_extension(siepon4_credential_id_ext, critical=False)
    )
    # Note: Signing of the DAC on the ONU must be performed using the TPM
    signed_dac_cert = dac_certificate.sign(private_key=device_key_id,
                                           algorithm=hashes.SHA256(),
                                           backend=default_backend())
    return signed_dac_cert


device_key = ec.generate_private_key(ec.SECP256R1(), default_backend())
# Note: Per FIPS 140-2, private keys must reside only in protected memory

device_certificate = create_dac_cert(
    onu_id="0A7FB49E2CF1",
    country="US",
    manufacturer_name="ACME Devices",
    model_name="SuperONU 5000",
    device_key_id=device_key,
    creation_time=datetime.fromisoformat("2024-04-29 02:11:20+00:00"),
    validity_years=40
)

with open("ONU_0A7FB49E2CF1.dac.cert.pem", "wb") as file:
    file.write(device_certificate.public_bytes(serialization.Encoding.PEM))
```

10A.3 NAC example

This example 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) extenstion.

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).

10A.3.1 PEM-encoded X.509 credential – NAC example

```
-----BEGIN CERTIFICATE-----
MIICMDCCAdWgAwIBAgIUfpr+sXjo7F+mo7V/Rwn7GQg8/H8wCgYIKoZIzj0EAwIw
NjETMBEGA1UECgwKRXhhbXBsZSBDQTEfMB0GA1UEAwwWRXhhbXBsZSBDQSBDZXJ0
aWZpY2F0ZTAeFw0yNDAxMDcwMDAwMDVaFw0yNDAyMDcwMDAwMDVaMFcxCzAJBgNV
BAYTAlVTMRUwEwYDVQQKDAxBQ01FIERldmljZXMxFjAUBgNVBAsMDVN1cGVyT05V
IDUwMDAxGTAXBgNVBAMMEE9OV8wQTdGQjQ5RTJDRjIwWTATBgcqhkjOPQIBBggq
hkjOPQMBBwNCAAS5DqGoPMiDZfGGEDW+1EKwfYnFlo6CowLK1Q68cKXmZwlHkxQb
vr8AMuRUHYesS8OtH4ox51kET/etprskwL3uo4GfMIGcMAwGA1UdEwEB/wQCMAAw
HQYDVR0OBBYEFPkZE5BFs0h4nvTBGOcKaQtwlgu1MB8GA1UdIwQYMBaAFP47ADU9
cVeKSTM+A+J+SFb+G5fcMD0GA1UdEQQ2MDSGMnVybjpvaWQ6MS4zLjYuMS40LjEu
MS4xLjEuMS4xLjE7Q3VzdElEOiA4NzY1NDMyLTFBMA0GCCtvAo5wBAEBBAECMAoG
CCqGSM49BAMCA0kAMEYCIQDkOdRbTDSarkspSACyOK0nD8KDMJCC78tzdbBtRzF7
DgIhAK4batXl10O0BeP7GUC63LX4heQKyebiaeUkyVMeZomV
-----END CERTIFICATE-----
-----BEGIN CERTIFICATE-----
MIIBoDCCAUagAwIBAgIUGk9lg8uex0nG+KEW7PMo3UrNpMEwCgYIKoZIzj0EAwIw
NjETMBEGA1UECgwKRXhhbXBsZSBDQTEfMB0GA1UEAwwWRXhhbXBsZSBDQSBDZXJ0
aWZpY2F0ZTAeFw0yMjAxMDcwMDAwMDVaFw0zMjAxMDcwMDAwMDVaMDYxEzARBgNV
BAoMCkV4YW1wbGUgQ0ExHzAdBgNVBAMMFkV4YW1wbGUgQ0EgQ2VydGlmaWNhdGUw
WTATBgcqhkjOPQIBBggqhkjOPQMBBwNCAATPvjNr+VCQNmsB5AmEph8o3apQg/5e
z2w3oV0RbgTdRPBJP7BFlDNaeWNI0EkTVy47EztPyxSeOW8yXjbv6UtYozIwMDAP
BgNVHRMBAf8EBTADAQH/MB0GA1UdDgQWBBT+OwA1PXFXikkzPgPifkhW/huX3DAK
BggqhkjOPQQDAgNIADBFAiBa3/SeF13AdB4rK1uAwTqL2rBVj9PqwUTTPlkjtcNG
jwIhAIokUSq4KctFm0b67m2TT9z1Jx6pms8fH+Nmr7Q+PeG4
-----END CERTIFICATE-----
```

10A.3.2 Parsed X.509 credential – NAC example

The file created by the sample code produces a X.509 certificate chain consisting of the operator-created NAC and the operator's Certificate Authority certificate (CAC).

Parsing of the generated certificate chain file, containing the NAC and CAC can be performed using the command:

```
while openssl x509 -noout -text; do :; done < ONU_0A7FB49E2CF2.nac.cert.pem
```

which produces the example output above (with the exception of the public key and DAC signature).

```
Certificate:
    Data:
        Version: 3 (0x2)
        Serial Number:
            7e:9a:fe:b1:78:e8:ec:5f:a6:a3:b5:7f:47:09:fb:19:08:3c:fc:7f
        Signature Algorithm: ecdsa-with-SHA256
        Issuer: O=Example CA, CN=Example CA Certificate
        Validity
            Not Before: Jan  7 00:00:05 2024 GMT
            Not After : Feb  7 00:00:05 2024 GMT
        Subject: C=US, O=ACME Devices, OU=SuperONU 5000, CN=ONU_0A7FB49E2CF2
        Subject Public Key Info:
            Public Key Algorithm: id-ecPublicKey
                Public-Key: (256 bit)
                pub:
                    04:b9:0e:a1:a8:3c:c8:83:65:f1:86:10:35:be:d4:
                    42:b0:7d:89:c5:96:8e:82:a3:02:ca:d5:0e:bc:70:
                    a5:e6:67:09:47:93:14:1b:be:bf:00:32:e4:54:1d:
                    87:ac:4b:c3:ad:1f:8a:31:e7:59:04:4f:f7:ad:a6:
                    bb:24:c0:bd:ee
                ASN1 OID: prime256v1
                NIST CURVE: P-256
        X509v3 extensions:
            X509v3 Basic Constraints: critical
                CA:FALSE
            X509v3 Subject Key Identifier:
                F9:19:13:90:45:B3:48:78:9E:F4:C1:18:E7:0A:69:0B:70:96:0B:B5
            X509v3 Authority Key Identifier:
                FE:3B:00:35:3D:71:57:8A:49:33:3E:03:E2:7E:48:56:FE:1B:97:DC
            X509v3 Subject Alternative Name:
                URI:urn:oid:1.3.6.1.4.1.1.1.1.1.1.1;CustID: 8765432-1A
            1.3.111.2.1904.4.1.1:
                .
    Signature Algorithm: ecdsa-with-SHA256
    Signature Value:
        30:46:02:21:00:e4:39:d4:5b:4c:34:9a:ae:4b:29:48:00:b2:
        38:ad:27:0f:c2:83:30:90:82:ef:cb:73:75:b0:6d:47:31:7b:
        0e:02:21:00:ae:1b:6a:d5:e5:d7:43:b4:05:e3:fb:19:40:ba:
        dc:b5:f8:85:e4:0a:c9:e6:e2:69:e5:24:c9:53:1e:66:89:95

Certificate:
    Data:
        Version: 3 (0x2)
        Serial Number:
            1a:4f:65:83:cb:9e:c7:49:c6:f8:a1:16:ec:f3:28:dd:4a:cd:a4:c1
        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:cf:be:33:6b:f9:50:90:36:6b:01:e4:09:84:a6:
                    1f:28:dd:aa:50:83:fe:5e:cf:6c:37:a1:5d:11:6e:
                    04:dd:44:f0:49:3f:b0:45:94:33:5a:79:63:48:d0:
                    49:13:57:2e:3b:13:3b:4f:cb:14:9e:39:6f:32:5e:
                    36:ef:e9:4b:58
                ASN1 OID: prime256v1
                NIST CURVE: P-256
        X509v3 extensions:
            X509v3 Basic Constraints: critical
                CA:TRUE
            X509v3 Subject Key Identifier:
                FE:3B:00:35:3D:71:57:8A:49:33:3E:03:E2:7E:48:56:FE:1B:97:DC
    Signature Algorithm: ecdsa-with-SHA256
    Signature Value:
        30:45:02:20:5a:df:f4:9e:17:5d:c0:74:1e:2b:2b:5b:80:c1:
        3a:8b:da:b0:55:8f:d3:ea:c1:44:d3:3e:59:23:b5:c3:46:8f:
        02:21:00:8a:24:51:2a:b8:29:cb:45:9b:46:fa:ee:6d:93:4f:
        dc:f5:27:1e:a9:9a:cf:1f:1f:e3:66:af:b4:3e:3d:e1:b8
```

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, representing this certificate as a NAC. Once the OID is registered and recognized, OpenSSL will be able to parse the *SIEPON4CredentialType* and produce human-readable output.

10A.3.3 Sample Python 3 code – NAC example

```
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

siepon4_cred_type_val = {"DAC": b'\x01', "NAC": b'\x02'}

def create_ca(creation_time, validity_years):
    ca_private_key = ec.generate_private_key(ec.SECP256R1(), default_backend())

    ca_name = x509.Name([
        x509.NameAttribute(NameOID.ORGANIZATION_NAME, "Example CA"),
        x509.NameAttribute(NameOID.COMMON_NAME, "Example CA Certificate"),
    ])

    # Create the SKI (Subject Key Identifier) extension
    ski_extension = x509.SubjectKeyIdentifier.from_public_key(
                            ca_private_key.public_key())
    ca_cert = (
        x509.CertificateBuilder()
        .subject_name(ca_name)
        .issuer_name(ca_name)
        .public_key(ca_private_key.public_key())
        .serial_number(x509.random_serial_number())
        .not_valid_before(creation_time)
        .not_valid_after(creation_time.replace(year=creation_time.year
                                               + validity_years))
        .add_extension(
            x509.BasicConstraints(ca=True, path_length=None),
            critical=True
        )
        .add_extension(ski_extension, critical=False)  # Add the SKI extension
```

```
        .sign(ca_private_key, hashes.SHA256(), default_backend())
    )

    return ca_private_key, ca_cert


def create_nac_cert(onu_id, country, manufacturer_name, model_name, subscriber_id,
                    device_pubkey, creation_time, ca_key, ca_cert, validity_months):
    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}"),
    ])

    san_extension = x509.SubjectAlternativeName([
        x509.UniformResourceIdentifier(f"urn:oid:1.3.6.1.4.1.1.1.1.1.1.1;CustID:
{subscriber_id}")
    ])

    ski_extension = x509.SubjectKeyIdentifier.from_public_key(device_pubkey)

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

    # Create the SIEPON4CredentialType extension field
    siepon4_credential_id_ext = \
        x509.UnrecognizedExtension(oid=ObjectIdentifier("1.3.111.2.1904.4.1.1"),
                                   value=siepon4_cred_type_val['NAC'])
    nac_cert = (
        x509.CertificateBuilder()
        .subject_name(subject_name)
        .issuer_name(ca_cert.subject)
        .public_key(device_pubkey)
        .serial_number(x509.random_serial_number())
        .not_valid_before(creation_time)
        .not_valid_after(creation_time.replace(month=creation_time.month +
validity_months))
        .add_extension(
            x509.BasicConstraints(ca=False, path_length=None),
            critical=True
        )
        .add_extension(ski_extension, critical=False)
        .add_extension(aki_extension, critical=False)
        .add_extension(san_extension, critical=False)
        .add_extension(siepon4_credential_id_ext, critical=False)
        .sign(ca_key, hashes.SHA256(), default_backend())
    )

    return nac_cert


ca_key, ca_cert = create_ca(datetime.fromisoformat("2022-01-07 00:00:05+00:00"), 10)

device_pubkey_pem="""
-----BEGIN PUBLIC KEY-----
MFkwEwYHKoZIzj0CAQYIKoZIzj0DAQcDQgAEuQ6hqDzIg2XxhhA1vtRCsH2JxZaO
gqMCytUOvHCl5mcJR5MUG76/ADLkVB2HrEvDrR+KMedZBE/3raa7JMC97g==
-----END PUBLIC KEY-----
""".encode()
device_pubkey=serialization.load_pem_public_key(device_pubkey_pem)

device_certificate = create_nac_cert(
    onu_id="0A7FB49E2CF2",
    country="US",
    device_pubkey=device_pubkey,
    manufacturer_name="ACME Devices",
    model_name="SuperONU 5000",
    subscriber_id="8765432-1A",
    creation_time=datetime.fromisoformat("2024-01-07 00:00:05"),
```

```
    ca_key=ca_key,
    ca_cert=ca_cert,
    validity_months=1
)

with open("ONU_0A7FB49E2CF2.nac.cert.pem", "wb") as file:
    file.write(device_certificate.public_bytes(serialization.Encoding.PEM))
    file.write(ca_cert.public_bytes(serialization.Encoding.PEM))
```