Document Identifier: DSP2059

Date: 2021-12-02

Version: 1.1.0

Redfish Certificate Management White Paper

Supersedes: 1.0.0

Document Class: Informational

Document Status: Published

Document Language: en-US

CONTENTS

1 Foreword

The Redfish Certificate Management White Paper was prepared by the Redfish Forum of the DMTF.

DMTF is a not-for-profit association of industry members dedicated to promoting enterprise and systems management and interoperability. For information about the DMTF, see http://www.dmtf.org.

1.1 Acknowledgments

The DMTF acknowledges the following individuals for their contributions to this document:

- Chris Davenport - Hewlett Packard Enterprise
- Michael Raineri - Dell Inc.
- Peter Svensson - Ericsson AB

2 Introduction

Redfish is an evolving hardware management standard that is designed to be flexible, extensible, and interoperable. Redfish contains a data model that is used to describe certificates for devices, services, and other resources. It also provides interface for clients to manage their certificates. This document helps implementers and clients understand the Redfish certificate data model as well as the common workflows clients might require to manage certificates.

3 Model explanation

If a Redfish service supports certificate management, the service root resource will contain the `CertificateService` property. It will also support the `Certificates` property in various resources to show where certificates can be installed.

Within the `CertificateService` resource, a client will find the available actions for managing certificates, such as generating signing requests, or doing atomic replacement of certificates. A client will also find the `CertificateLocations` resource, which is responsible for providing an inventory of all certificates the service is managing.

The following sections detail how these things are reported by a Redfish service.

3.1 Certificate service

The certificate service is represented by the `CertificateService` resource. It contains a set of actions for service level management of certificates. There are currently two actions defined for this:

- `GenerateCSR` action (Certificate Signing Request)
- `ReplaceCertificate` action

The `CertificateService` resource also contains a link to the `CertificateLocations` resource via the `CertificateLocations` property.

Example `CertificateService` resource:


```
{
    "@odata.id": "/redfish/v1/CertificateService",
    "@odata.type": "#CertificateService.v1_0_0.CertificateService",
    "Id": "CertificateService",
    "Name": "Certificate Service",
    "Actions": {
        "#CertificateService.GenerateCSR": {
            "target": "/redfish/v1/CertificateService/Actions/CertificateService.GenerateCSR"
        },
        "#CertificateService.ReplaceCertificate": {
            "target": "/redfish/v1/CertificateService/Actions/CertificateService.ReplaceCertificate"
        }
    },
    "CertificateLocations": {
        "@odata.id": "/redfish/v1/CertificateService/CertificateLocations"
    }
```

```
}
```

3.1.1 GenerateCSR action

The `GenerateCSR` (Certificate Signing Request) action is used to create a signing request for a new certificate. The response of the action contains the CSR as a Privacy Enhanced Mail (PEM) encoded string. The CSR is then given to a CA (Certificate Authority) by the client, which then produces a signed certificate.

The parameters for the action specify many of the common fields that are put into the end certificate. The action also allows for the client to specify the type of key-pair that is generated for the certificate. The private portion of the key-pair is expected to be retained by the service and used with the certificate that is installed after the CA has signed it. It's possible one or more clients will invoke this action multiple times before any certificates are signed and installed. It's recommended that implementations retain sufficient information from each of the requests, such as the CSR string that was generated, so that it's possible to map a given certificate to a CSR that was produced by this action.

Parameter	X.509 attribute	Description
`CommonName`	`commonName`	The fully qualified domain name of the component that is being secured.
`AlternativeNames`	`subjectAltName`	Additional hostnames of the component that is being secured.
`Organization`	`organizationName`	The name of the organization making the request.
`OrganizationalUnit`	`organizationalUnitName`	The name of the unit or division of the organization making the request.
`City`	`localityName`	The city or locality of the organization making the request.
`State`	`stateOrProvinceName`	The state, province, or region of the organization making the request.
`Country`	`countryName`	The country of the organization making the request.
`Email`	`emailAddress`	The email address of the contact within the organization making the request.
`KeyUsage`	`keyUsage` and `extKeyUsage`	The usage of the key contained in the certificate.
`Surname`	`surname`	The surname of the user making the request.
`GivenName`	`givenName`	The given name of the user making the request.
`Initials`	`initials`	The initials of the user making the request.
`ChallengePassword`	`challengePassword`	The challenge password to be applied to the certificate for revocation requests.
`UnstructuredName`	`unstructuredName`	The unstructured name of the subject.
`ContactPerson`	`name`	The name of the user making the request.

Parameter	X.509 attribute	Description
`KeyPairAlgorithm`	N/A	The type of key-pair for use with signing algorithms.
`KeyBitLength`	N/A	The length of the key in bits, if needed based on the value of the `KeyPairAlgorithm` parameter.
`KeyCurveId`	N/A	The curve ID to be used with the key, if needed based on the value of the `KeyPairAlgorithm` parameter.
`CertificateCollection`	N/A	A link to the `CertificateCollection` resource where the certificate will be installed.

The `KeyPairAlgorithm` and `KeyCurveId` are both strings where the values are based on the contents of the Trusted Computing Group (TCG) Algorithm Registry. The `TPM_ALG_ID` and `TPM_ECC_CURVE` tables respectively contain the set of possible values for the two parameters. Services are not required to support the entire contents of the table, however, the following are recommended to be supported.

Recommended `TPM_ALG_ID` values:

Value	Description
`TPM_ALG_RSA`	The RSA algorithm.
`TPM_ALG_ECDSA`	Signature algorithm using elliptic curve cryptography (ECC).

Recommended `TPM_ECC_CURVE` values:

Value	Description
`TPM_ECC_NIST_P256`	The NIST P-256 curve.
`TPM_ECC_NIST_P384`	The NIST P-384 curve.
`TPM_ECC_NIST_P521`	The NIST P-521 curve.

A sample usage of this action is shown in the Generate a CSR section.

3.1.2 ReplaceCertificate action

The `ReplaceCertificate` action is used for cases where an atomic deletion of a `Certificate` resource followed by a creation of a new `Certificate` resource is required. For example, if an HTTPS service has a single certificate and a user would like to replace it, performing a delete operation on the old certificate first would create a window where the HTTPS service does not have a certificate, which the service might not be able to handle.

The request body for the `ReplaceCertificate` action contains the same fields required when creating a new `Certificate` resource. `CertificateType` contains the format of the certificate, and `CertificateString` contains

the string encoding of the certificate based on its format. In addition, the client also provides the URI of the certificate to be replaced using the `CertificateUri` parameter. The response from the action uses the `Location` HTTP header to show where the new certificate was placed. Depending on the implementation, the `Location` header in the response might contain the same URI specified in the `CertificateUri` parameter, but if they are not the same, it's expected that the URI referenced by the `CertificateUri` parameter is no longer available at this point.

A sample usage of this action is shown in Replace a certificate section.

3.2 Certificates

Devices, services, and other resources that support certificates have a `Certificates` property that references the `CertificateCollection` resource for the respective resource. This is done so that it's possible to associate one or more certificates with a particular usage. For example, the `CertificateCollection` resource found off of the `HTTPS` property within the `ManagerNetworkProtocol` resource is used to represent the certificates for the HTTPS service for a manager. Over time, resources that have use cases for managing certificates will have `CertificateCollection` resources added to them.

Users are able to add and remove members from the `CertificateCollection` resource like with any other type of resource collection. A user can submit a POST operation to the resource collection to install a new certificate, and a user can perform a DELETE operation to a `Certificate` resource to remove a certificate from the resource collection. When removing the last member of the `CertificateCollection` resource, the service might automatically install a default certificate. This could happen for cases where at least one certificate is always required to be present, such as with an HTTPS service.

Since there are different formats for certificates, the `CertificateCollection` resource uses the `@Redfish.SupportedCertificates` annotation to show which formats it will accept. The following example shows a `CertificateCollection` resource that only supports PEM style certificates.


```
{
    "@odata.id": "/redfish/v1/Managers/BMC/NetworkProtocol/HTTPS/Certificates",
    "@odata.type": "#CertificateCollection.CertificateCollection",
    "Name": "Certificate Collection",
    "Members@odata.count": 1,
    "Members": [
        {
            "@odata.id": "/redfish/v1/Managers/BMC/NetworkProtocol/HTTPS/Certificates/1"
        }
    ],
    "@Redfish.SupportedCertificates": [
        "PEM"
    ]
}
```

Within the `Certificate` resource, there are two properties that are provided by the user when installing a new certificate:

- `CertificateType` describes the format of the certificate.
- `CertificateString` is the string encoding of the certificate based on the value of `CertificateType`.

If there is ever a portion of the certificate that contains private information, such as a private key, a service is not allowed to provide it when the certificate is being read by a client.

The remaining properties of the `Certificate` resource contain decodings of attributes found in the certificate itself. An example of this is shown below.

```
{
    "@odata.id": "/redfish/v1/Managers/BMC/NetworkProtocol/HTTPS/Certificates/1",
    "@odata.type": "#Certificate.v1_1_0.Certificate",
    "Id": "1",
    "Name": "HTTPS Certificate",
    "CertificateString": "-----BEGIN CERTIFICATE-----\n...\n-----END CERTIFICATE-----",
    "CertificateType": "PEM",
    "Issuer": {
        "Country": "US",
        "State": "Oregon",
        "City": "Portland",
        "Organization": "Contoso",
        "OrganizationalUnit": "ABC",
        "CommonName": "manager.contoso.org"
    },
    "Subject": {
        "Country": "US",
        "State": "Oregon",
        "City": "Portland",
        "Organization": "Contoso",
        "OrganizationalUnit": "ABC",
        "CommonName": "manager.contoso.org"
    },
    "ValidNotBefore": "2018-09-07T13:22:05Z",
    "ValidNotAfter": "2019-09-07T13:22:05Z",
    "KeyUsage": [
        "KeyEncipherment",
        "ServerAuthentication"
    ]
}
```

3.2.1 Certificate usage in the data model

As of this publication, the following types of certificates can be found in the data model.

3.2.1.1 Web service certificates

The `ManagerNetworkProtocol` resource contains a `Certificates` property within the `HTTPS` property. The certificates in this collection are for the web service for the manager. In most cases, this collection will contain the certificate for the Redfish service itself. Because of this, the following semantics are recommended to be supported by services:

- There should always be at least one member in this collection. Having an empty collection would break HTTPS since a certificate is required for TLS.
- When a client performs a DELETE operation on a certificate in this collection and it's the only certificate, a new default certificate should be produced for this collection. Depending on the implementation, the new certificate might not appear until the next manager reset.
- The `ReplaceCertificate` action should be supported in order to install a new certificate. The previous recommendations make it difficult to support discrete DELETE and POST operations for removing and adding a new certificate.

3.2.1.2 Remote server certificates

The following table describes the resources that contain remote server certificates. In some cases, there is an additional property to control whether or not the service verifies the certificate received with the certificates in its collection as part of connecting to the remote server. In the absence of a verification property, the service might perform its own checks on the certificate from the remote service, such as verifying it's signed by a trusted CA.

Resource	Collection property	Verify property	Description
`AccountService`	`LDAP/Certificates`, `ActiveDirectory/Certificates`, `TACACSplus/Certificates`		Used to verify that the service is communicating with the correct authentication server.
`ComputerSystem`	`Boot/Certificates`		Used to verify that UEFI is communicating with the correct server when performing an HTTPS boot.
`ComputerSystem`	`KeyManagement/KMIPCertificates`		Used to verify that system is communicating with the correct KMIP server.
`EventDestination`	`Certificates`	`VerifyCertificate`	Used to verify that the service is communicating with the correct event listener prior to transmitting the event.
`ExternalAccountProvider`	`Certificates`		Used to verify that the service is communicating with the correct authentication server.

Resource	Collection property	Verify property	Description
`UpdateService`	`RemoteServerCertificates`	`VerifyRemoteServerCertificate`	Used to verify that the service is communicating with the correct image server when retrieving an image specified by the `ImageURI` parameter in the `SimpleUpdate` action.
`VirtualMedia`	`Certificates`	`VerityCertificate`	Used to verify that the service is communicating with the correct image server when retrieving a virtual media image.

3.2.1.3 Client certificates provided to remote servers

The following table describes the resources that contain client certificates that are provided to remote servers when the service performs an outgoing connection to a remote server. Remote servers can use the provided certificate to verify the service, such as during TLS handshaking.

Resource	Collection property
`EventDestination`	`ClientCertificates`
`UpdateService`	`ClientCertificates`
`VirtualMedia`	`ClientCertificates`

3.2.1.4 Device identity certificates

The following table describes the resources that contain device identity certificates. Clients can use these certificates to verify the identity of devices and for attestation purposes.

Resource	Collection property
`Chassis`	`Certificates`
`ComputerSystem`	`Certificates`
`Drive`	`Certificates`
`Manager`	`Certificates`
`Memory`	`Certificates`
`NetworkAdapter`	`Certificates`
`Processor`	`Certificates`
`Storage`	`StorageControllers/{Index}/Certificates`

Resource	Collection property
`StorageController`	`Certificates`
`Switch`	`Certificates`

3.2.1.5 UEFI Secure Boot certificates and signatures

The `SecureBootDatabase` resource contains a `Certificates` property and a `Signatures` property for managing the certificates and signatures in each of the databases. These certificates and signatures are used to control what UEFI drivers and boot images are allowed to run on the system. Each instance of this resource represents a particular UEFI Secure Boot database. UEFI defines the following databases.

Name	Description	Notes
`PK`	Platform Key. Used to show who can modify the `KEK` database.	Contains one certificate at most, and no signatures. When no certificate is installed, UEFI Secure Boot cannot be enabled.
`KEK`	Key Exchange Key database. Used to show who can modify the `db`, `dbx`, `dbt`, and `dbr` databases.	Contains any number of certificates, and no signatures.
`db`	Authorized Signature database. Contains a set of allowed certificates, hashes, and signatures for UEFI drivers and boot images.	Contains any number of certificates and signatures.
`dbx`	Forbidden Signature database. Contains a set of disallowed certificates, hashes, and signatures for UEFI drivers and boot images.	Contains any number of certificates and signatures.
`dbt`	Authorized Timestamp Signature database. Contains a set of certificates, hashes, and signatures from timestamp authorities for controlling when certificates and signatures in the `dbx` database are valid based on time.	Contains any number of certificates and signatures.
`dbr`	Authorized Recovery Signature database. Contains a set of allowed certificates, hashes, and signatures for recovery images.	Contains any number of certificates and signatures.

Each of the above databases have read-only default copy with `Default` appended to the name, such as `dbDefault`. Performing the `ResetKeys` action found in the `SecureBoot` resource with the `ResetKeysType` parameter set to `ResetAllKeysToDefault` will restore the contents of the databases to their defaults.

3.3 Certificate locations

The `CertificateLocations` resource is contains a set of links to all of the certificates managed by the service. This type of resource is to be used for security administrators who may need perform auditing of the service. The following is an example `CertificateLocations` resource that shows the service is managing three certificates.

```
{
    "@odata.id": "/redfish/v1/CertificateService/CertificateLocations",
    "@odata.type": "#CertificateLocations.v1_0_0.CertificateLocations",
    "Id": "CertificateLocations",
    "Name": "Certificate Locations",
    "Links": {
        "Certificates": [
            {
                "@odata.id": "/redfish/v1/Managers/BMC/NetworkProtocol/HTTPS/Certificates/1"
            },
            {
                "@odata.id": "/redfish/v1/AccountService/ExternalAccountProviders/LDAP/Certificates/1"
            }
        ]
    }
}
```

4 Client workflows

The following section contains possible workflows that clients may follow for managing certificates.

4.1 Common operations

The following section shows common operations used in various client workflows.

4.1.1 Generating a CSR

The following two examples show a client performing a `GenerateCSR` action and receiving a response. In the request, the client is specifying a new signing request using an RSA 4096-bit key-pair, and that the end certificate will be installed for the HTTPS service for the `Manager` resource named `BMC`. The response contains the PEM encoded signing request in the property `CSRString`.

Example generate CSR request:

```
POST /redfish/v1/CertificateService/Actions/CertificateService.GenerateCSR HTTP/1.1
Content-Type: application/json
Content-Length: <computed-length>

{
    "Country": "US",
    "State": "Oregon",
    "City": "Portland",
    "Organization": "Contoso",
    "OrganizationalUnit": "ABC",
    "CommonName": "manager.contoso.org",
    "AlternativeNames": [
        "manager.contoso.com",
        "manager.contoso.us"
    ],
    "Email": "admin@contoso.org",
    "KeyPairAlgorithm": "TPM_ALG_RSA",
    "KeyBitLength": 4096,
    "KeyUsage": [
        "KeyEncipherment",
        "ServerAuthentication"
    ],
    "CertificateCollection": {
        "@odata.id": "/redfish/v1/Managers/BMC/NetworkProtocol/HTTPS/Certificates"
    }
}
```

Example generate CSR response:

```
HTTP/1.1 200 OK
Content-Type: application/json
Content-Length: <computed-length>

{
    "CSRString": "-----BEGIN CERTIFICATE REQUEST-----...-----END CERTIFICATE REQUEST-----",
    "CertificateCollection": {
        "@odata.id": "/redfish/v1/Managers/BMC/NetworkProtocol/HTTPS/Certificates"
    }
}
```

4.1.2 Replace a certificate

The following two examples show a client performing a `ReplaceCertificate` action and receiving a response. The request specifies that it's replacing the HTTPS certificate with the `Id` of `1` for the `Manager` resource called `BMC` with a new PEM encoded certificate. The `Location` header in the response shows that the new certificate was assigned the `Id` of `2`.

Example replace certificate request:

```
POST /redfish/v1/CertificateService/Actions/CertificateService.ReplaceCertificate HTTP/1.1
Content-Type: application/json
Content-Length: <computed-length>

{
    "CertificateUri": {
        "@odata.id": "/redfish/v1/Managers/BMC/NetworkProtocol/HTTPS/Certificates/1"
    },
    "CertificateString": "-----BEGIN CERTIFICATE-----\n...\n-----END CERTIFICATE-----",
    "CertificateType": "PEM"
}
```

Example replace certificate response:

```
HTTP/1.1 204 No Content
Content-Type: application/json
Content-Length: <computed-length>
Location: /redfish/v1/Managers/BMC/NetworkProtocol/HTTPS/Certificates/2
```

4.1.3 Replace a certificate and provide a private key

There are cases where a private key may need to be provided with the certificate. This could happen if the signing request for the certificate originated from another entity, so the key-pair was not generated by the service. In these cases, the private key can be concatenated with the certificate itself.

The following two examples show a client performing a `ReplaceCertificate` action and receiving a response. The request specifies that it's replacing the HTTPS certificate with the `Id` of `1` for the `Manager` resource called `BMC` with a new PEM encoded certificate. It's also including a private key portion in `CertificateString`. The `Location` header in the response shows that the new certificate was assigned the `Id` of `2`.

Example replace certificate request:

```
POST /redfish/v1/CertificateService/Actions/CertificateService.ReplaceCertificate HTTP/1.1
Content-Type: application/json
Content-Length: <computed-length>

{
    "CertificateUri": {
        "@odata.id": "/redfish/v1/Managers/BMC/NetworkProtocol/HTTPS/Certificates/1"
    },
    "CertificateString": "-----BEGIN C...RTIFICATE-----\n-----BEGIN P...-END PRIVATE KEY-----",
    "CertificateType": "PEM"
}
```

Example replace certificate response:

```
HTTP/1.1 204 No Content
Content-Type: application/json
Content-Length: <computed-length>
Location: /redfish/v1/Managers/BMC/NetworkProtocol/HTTPS/Certificates/2
```

4.1.4 Install a certificate

The following two examples show a client installing a certificate to the `CertificateCollection` resource located at `/redfish/v1/Managers/BMC/NetworkProtocol/HTTPS/Certificates` and receiving a response. The `Location` header in the response shows that the new certificate was assigned the `Id` of `1`.

Example certificate installation request:

```
POST /redfish/v1/Managers/BMC/NetworkProtocol/HTTPS/Certificates HTTP/1.1
```

```
Content-Type: application/json
Content-Length: <computed-length>

{
    "CertificateString": "-----BEGIN CERTIFICATE-----\n...\n-----END CERTIFICATE-----",
    "CertificateType": "PEM"
}
```

Example certificate installation response:

```
HTTP/1.1 204 No Content
Content-Type: application/json
Content-Length: <computed-length>
Location: /redfish/v1/Managers/BMC/NetworkProtocol/HTTPS/Certificates/1
```

4.1.5 Install a certificate and provide a private key

There are cases where a private key may need to be provided with the certificate. This could happen if the signing request for the certificate originated from another entity, so the key-pair was not generated by the service. In these cases, the private key can be concatenated with the certificate itself.

The following two examples show a client installing a certificate to the `CertificateCollection` resource located at `/redfish/v1/Managers/BMC/NetworkProtocol/HTTPS/Certificates` and receiving a response. It's also including a private key portion in `CertificateString`. The `Location` header in the response shows that the new certificate was assigned the `Id` of `1`.

Example certificate installation request:

```
POST /redfish/v1/Managers/BMC/NetworkProtocol/HTTPS/Certificates HTTP/1.1
Content-Type: application/json
Content-Length: <computed-length>

{
    "CertificateString": "-----BEGIN C...RTIFICATE-----\n-----BEGIN P...-END PRIVATE KEY-----",
    "CertificateType": "PEM"
}
```

Example certificate installation response:

```
HTTP/1.1 204 No Content
Content-Type: application/json
```

```
Content-Length: <computed-length>
Location: /redfish/v1/Managers/BMC/NetworkProtocol/HTTPS/Certificates/1
```

4.1.6 Remove a certificate

The following example shows a client removing a certificate located at `/redfish/v1/Managers/BMC/NetworkProtocol/HTTPS/Certificates/1`.

Example certificate removal request:

```
DELETE /redfish/v1/Managers/BMC/NetworkProtocol/HTTPS/Certificates/1 HTTP/1.1
```

4.2 Generate and install a new certificate

In this workflow, a client is using the service to generate a new Certificate Signing Request (CSR), and installing a new certificate on a given service or device.

1. Locate the appropriate `CertificateCollection` resource where the new certificate will be installed.
   - For example, if installing a new HTTPS certificate for a manager:
     - Identify the appropriate `Manager` resource in the `ManagerCollection` resource.
     - Navigate to the manager's `ManagerNetworkProtocol` resource.
     - Navigate to the `CertificateCollection` resource found in the `HTTPS` object.
2. Perform the `GenerateCSR` action as shown in the Generate a CSR section.
   - The service will generate a new key-pair and retain the private key.
3. The value of `CSRString` in the response is taken to a Certificate Authority (CA).
   - The CA uses this string to produce a signed certificate.
4. Install the new certificate.
   - If the installation is to replace an existing certificate in the `CertificateCollection` resource:
     - Perform the `ReplaceCertificate` action as shown in the Replace a certificate section.
     - In the action request, specify the certificate to replace as found in the first step.
   - If the installation is to add a certificate to the `CertificateCollection` resource:
     - Perform a `POST` to the `CertificateCollection` resource from the first step, as shown in the Install a certificate section.

4.3 Install a new certificate that was not generated by the service

In this workflow, a client is installing a new certificate that has been generated by an external source. This could be a

common certificate provided by a client's IT group that needs to be installed on multiple systems. Depending on how the service will be using the certificate, the private key may need to be provided as part of the installation.

1. Locate the appropriate `CertificateCollection` resource where the new certificate will be installed.
   - For example, if installing a new HTTPS certificate for a manager:
     - Identify the appropriate `Manager` resource in the `ManagerCollection` resource.
     - Navigate to the manager's `ManagerNetworkProtocol` resource.
     - Navigate to the `CertificateCollection` resource found in the `HTTPS` object.
2. Install the new certificate.
   - If the installation is to replace an existing certificate in the `CertificateCollection` resource, and the service does not require the use of the private key:
     - Perform the `ReplaceCertificate` action as shown in the Replace a certificate section.
     - In the action request, specify the certificate to replace as found in the first step.
   - If the installation is to replace an existing certificate in the `CertificateCollection` resource, and the service does require the use of the private key:
     - Perform the `ReplaceCertificate` action as shown in the Replace a certificate and provide a private key section.
     - In the action request, specify the certificate to replace as found in the first step.
     - Also in the action request, the private key of the certificate will need to be provided.
   - If the installation is to add a certificate to the `CertificateCollection` resource, and the service does not require the use of the private key:
     - Perform a `POST` to the `CertificateCollection` resource from the first step, as shown in the Install a certificate section.
   - If the installation is to add a certificate to the `CertificateCollection` resource, and the service does requires the use of the private key:
     - Perform a `POST` to the `CertificateCollection` resource from the first step, as shown in the Install a certificate and provide a private key section.
     - In the POST request, the private key of the certificate will need to be provided.

4.4 Possible side effects when installing certificates

In some cases, multiple `CertificateCollection` resources may share a common set of certificates internal to the service. For example, a service might implement a common set of certificates for LDAP and Active Directory. This would make it so that when one is disabled and the other is enabled, the same certificates already installed previously would still be available. In these cases, installing a certificate in one `CertificateCollection` resource will result in the `Certificate` resources appearing in two different collections, and deleting a `Certificate` resource found in one collection will cause it to be removed from both collections. This does not mean that two certificates are tracked in the service, but rather one certificate can be found in two different collections.

4.5 Removing certificates

In this workflow, a client is removing a certificate from the service. This could be done as part of a decommissioning process for a system where certificates need to be removed first, or other types of maintenance activities.

1. Locate the appropriate `Certificate` resource that will be removed.
    - For example, if removing a certificate for a `ExternalAccountProvider` resource:
        - Identify the appropriate `ExternalAccountProvider` resource in the `ExternalAccountProviderCollection` resource.
        - Navigate to the `CertificateCollection` resource found in the `ExternalAccountProvider` resource.
        - Identify the `Certificate` resource to remove in the `CertificateCollection` resource.
2. Perform a `DELETE` on the `Certificate` resource from the first step, as shown in the Remove a certificate section.

Removing the last `Certificate` resource from a `CertificateCollection` resource might result in different outcomes, depending on the design of the service, or the type of certificate being removed.

- In cases where an empty collection is not valid, such as with an HTTPS service:
    - The operation might fail, which would require steps to be taken to replace the certificate with a default certificate.
    - The operation might result in a new default certificate to populate the collection.
- In other cases, the collection will become empty.

5 Appendix A: References

- RFC2985, PKCS #9: Selected Object Classes and Attribute Types Version 2.0: https://tools.ietf.org/html/rfc2985
- RFC5280, Internet X.509 Public Key Infrastructure Certificate and Certificate Revocation List (CRL) Profile: https://tools.ietf.org/html/rfc5280
- Trusted Computing Group Algorithm Registry: https://trustedcomputinggroup.org/resource/tcg-algorithm-registry/
- `CertificateService` schema: http://redfish.dmtf.org/schemas/v1/CertificateService_v1.xml
- `CertificateLocations` schema: http://redfish.dmtf.org/schemas/v1/CertificateLocations_v1.xml
- `Certificate` schema: http://redfish.dmtf.org/schemas/v1/Certificate_v1.xml

6 Appendix B: Change log

Version	Date	Description
1.1.0	2021-12-02	Updated **Certificate usage in the data model** to better describe how different certificate collections are used.
		Updated **Certificate usage in the data model** to include all certificate locations defined up through DSP8010 2021.3.
1.0.0	2020-06-11	Initial release.