ndn-cxx - Feature #2766

CertificateBundle: design

04/23/2015 10:25 AM - Yingdi Yu

Status:	Closed	**Start date:**	
Priority:	Normal	**Due date:**	
Assignee:	Junxiao Shi	**% Done:**	100%
Category:	Security	**Estimated time:**	9.00 hours
Target version:	v0.8		

Description

Design the Key Bundle protocol to provide a way to retrieve a set of the certificates needed to authenticate a data packet within one RTT and assures that the certificates for the authentication can be retrieved for as long as the data publisher is online.

Related issues:

Blocks NLSR - Task #3950: Use CertificateBundleFetcher instead of DirectFetcher	**New**	**02/06/2017**

History

#1 - 04/23/2015 10:25 AM - Yingdi Yu

- Related to Feature #2451: New Abstraction for Identity Key Certificate added

#2 - 04/23/2015 10:26 AM - Junxiao Shi

- Tracker changed from Task to Feature

- Subject changed from Support key-bundle support in PIB to Support key-bundle in PIB

- Start date deleted (04/23/2015)

#3 - 12/14/2016 12:41 PM - Alex Afanasyev

- Subject changed from Support key-bundle in PIB to KeyBundle: design

- Description updated

#4 - 12/14/2016 12:42 PM - Alex Afanasyev

Preliminary design description: https://docs.google.com/document/d/1eMwwIUu4DMmL4UxbuKSsZAeWkOpOM6cBygzBNH5IYcA/edit?usp=sharing

#5 - 12/14/2016 12:42 PM - Alex Afanasyev

- Related to deleted (Feature #2451: New Abstraction for Identity Key Certificate)

#6 - 12/14/2016 12:49 PM - Alex Afanasyev

- Blocks Feature #3891: Integrate fetching certificates using KeyBundle into the Validator added

#7 - 12/14/2016 12:50 PM - Alex Afanasyev

- Blocks Feature #3892: Integrate publishing of KeyBundle into Face::setInterestFilter added

#8 - 12/14/2016 12:58 PM - Alex Afanasyev

- Assignee set to Manika Mittal

- Target version set to v0.6

#9 - 01/12/2017 08:59 PM - Alex Afanasyev

- Status changed from New to Feedback

- % Done changed from 0 to 80

#10 - 01/25/2017 12:44 PM - Junxiao Shi

Where's the spec? If it's already on Gerrit, post the link here.

#11 - 01/25/2017 12:53 PM - Alex Afanasyev

See the link in note 4

#12 - 01/25/2017 01:06 PM - Manika Mittal

I will update the specs to reflect the new validator interface.

#13 - 01/28/2017 12:39 PM - Junxiao Shi

I've briefly read the spec.

It's unclear how "list of certificates" is encoded. This needs to be formally declared.

Must the KeyBundle contain a complete certificate chain? Are certificates required to appear in a certain order?

How does a Validator know whether the requester has prepared a KeyBundle or it should retrieve individual certificates instead?

Why does the KeyBundle carry the trust anchor certificate? Trust anchor is already known by the validator and should be omitted. In WebPKI, the web server does not send root CA certificate in its certificate chain.

#14 - 02/06/2017 11:37 AM - Manika Mittal

I will add the packet spec on wiki here. I have updated the consumer side spec on the google doc.

The Bundle is expected to have the certificates in a certain order i.e. from the first certificate needed to validate the data packet itself, followed by the certificate needed to validate the first certificate, so on and so forth.

The validator first requests for a Certificate Bundle and if that interest times out or returns a Nack it will fetch individual certificates from the network.

Whether the bundle carries the trust anchor or not is undefined. The producer will create the bundle as per it's knowledge i.e. till it either reaches a self signed certificate or may be a loop or may be some const MAX_BUNDLE_SIZE.

#15 - 02/06/2017 01:56 PM - Manika Mittal

Packet format specs : https://redmine.named-data.net/projects/ndn-cxx/wiki/Certificate_Bundle_Packet_Format

#16 - 02/06/2017 03:23 PM - Muktadir Chowdhury

- Related to Task #3950: Use CertificateBundleFetcher instead of DirectFetcher added

#17 - 02/07/2017 09:52 PM - Junxiao Shi

> The Bundle is expected to have the certificates in a certain order i.e. from the first certificate needed to validate the data packet itself, followed by the certificate needed to validate the first certificate, so on and so forth.

This implies that a prover must have access to the trust schema in order to generate a bundle, because the trust schema defines which certificates are needed to verify a packet and in what order. This is a reasonable assumption.
A consequence is that the name of a bundle should indicate which trust schema is in use, because the same certificate may potentially be used by multiple applications with different trust schemas. Certificate_Bundle_Packet_Format rev1 includes <trust-model> component in a bundle name, but lacks an explanation on how the verifier can derive this name from the certificate name which is the only information available in KeyLocator.

> Whether the bundle carries the trust anchor or not is undefined.

This needs more explanation in the documents, because it's unobvious for a reader who knows PKI fairly well, as WebPKI recommends not to include the trust anchor (root CA certificate) in the certificate chain sent by a TLS sever.
I do see a valid use case for including the trust anchor (even if it's self-signed) in the bundle: the verifier's trust schema may only contain a digest of the trust anchor certificate instead of the full certificate. In this case, including the trust anchor would allow the verifier to see the public key.

> The producer will create the bundle as per it's knowledge i.e. till it either reaches a self signed certificate or may be a loop or may be some const MAX_BUNDLE_SIZE.

If the verifier encounters an incomplete bundle, i.e. it needs more certificates than what's available from the bundle, does the verifier attempt to retrieval additional certificates individually?

> the certificates present in the bundle are cached in the unverified cache and the validation process continues on the first certificate present in the bundle. Note it is assumed that the certificate bundle has its certificates is in a certain order where the first certificate is the certificate needed to validate the signature of the target data packet.

If the certificate bundle only serves as a data source to populate the unverified certificate cache, what's the purpose of the recommendation that certificates are sorted in a certain order?

> there is a version number because it’s possible that the Certificate Bundle is updated (in case some key in the chain is revoked)

How to ensure the verifier is retrieving the latest version, or at least a good enough version that does not contain a revoked/expired intermediate certificate, instead of an older version from a network cache that eventually causes the verificate to fail?

> Each segment will always have complete certificates.

What's the benefit of this requirement?
The obvious drawback is not being able to use the full size of a Data packet, and impossible to encapsulate the certificate that is close to the size limit because the outer name needs extra bytes.
Note that "if the first segment contains enough certificates then we don't retrieve the second segment" is not a valid reason for requiring segment to stop at certificate boundary, because the stream constructed by contents from segments can be processed in the same way as NFD UnixStreamTransport reads LpPackets from a stream socket.

#18 - 02/08/2017 02:07 PM - Manika Mittal

> A consequence is that the name of a bundle should indicate which trust schema is in use, because the same certificate may potentially be used by multiple applications with different trust schemas. Certificate_Bundle_Packet_Format rev1 includes <trust-model> component in a bundle name, but lacks an explanation on how the verifier can derive this name from the certificate name which is the only information available in KeyLocator.

For now we only support the trust models with a single hierarchy. I will append a number "00" as a component in the Bundle name to represent that and add that in the specs too.

> This needs more explanation in the documents, because it's unobvious for a reader who knows PKI fairly well, as WebPKI recommends not to include the trust anchor (root CA certificate) in the certificate chain sent by a TLS server.

I will add more explanation. Basically, if the trust anchor is agreed upon by both the producer and the consumer then we don't have to include it. But currently we don't know this. It's possible that a trust model as no fixed trust anchor. So it's undefined.

> If the verifier encounters an incomplete bundle, i.e. it needs more certificates than what's available from the bundle, does the verifier attempt to retrieval additional certificates individually?

Yes.

> If the certificate bundle only serves as a data source to populate the unverified certificate cache, what's the purpose of the recommendation that certificates are sorted in a certain order?

I would say it's better to keep it sorted because we don't want to fetch all the segments if we don't have to. So if the certificates are in random order in the bundle it might lead to some additional fetches which could have been avoided if the certificates were in order.

> How to ensure the verifier is retrieving the latest version, or at least a good enough version that does not contain a revoked/expired intermediate certificate, instead of an older version from a network cache that eventually causes the verification to fail?

Currently, we only rely on freshness period and must be fresh.

> What's the benefit of this requirement?

Actually, this is not a hard requirement. May be I can mention that in the spec that it is mostly for simplicity.

#19 - 02/08/2017 02:41 PM - Junxiao Shi

> For now we only support the trust models with a single hierarchy. I will append a number "00" as a component in the Bundle name to represent that and add that in the specs too.

Are you saying that each data-signing-certificate is usable with one trust schema? This is too much restriction and is unacceptable.

> It's possible that a trust model as no fixed trust anchor.

WebPKI has many root CAs, but the issuer of the highest level of intermediate CA is the trust anchor being used. Similarly, an NDN trust schema can have multiple trust anchors, and the bundle only needs to include up to the certificate directly issued by one of the trust anchors, but not the self-signed trust anchor. An exception is the verifier does not have the public key of the trust anchor, as described in note-17.

> If the certificate bundle only serves as a data source to populate the unverified certificate cache, what's the purpose of the recommendation that certificates are sorted in a certain order?

I would say it's better to keep it sorted because we don't want to fetch all the segments if we don't have to. So if the certificates are in random order in the bundle it might lead to some additional fetches which could have been avoided if the certificates were in order.

Agreed. This should be a SHOULD-level requirement rather than a MUST-level, and the rationale needs to explained in the document.

> How to ensure the verifier is retrieving the latest version, or at least a good enough version that does not contain a revoked/expired intermediate certificate, instead of an older version from a network cache that eventually causes the verification to fail?

Currently, we only rely on freshness period and must be fresh.

As you know, these are mostly useless. This should appear as a known limitation in the documents.

> Each segment will always have complete certificates.
> What's the benefit of this requirement?

Actually, this is not a hard requirement. May be I can mention that in the spec that it is mostly for simplicity.

Are you saying this is a MAY-level requirement? If so, verifier needs to be able to accept a bundle in which segment does not stop at certificate boundary, since a MAY-level requirement is truly optional at prover end.

#20 - 02/08/2017 02:54 PM - Alex Afanasyev

Junxiao Shi wrote:

> For now we only support the trust models with a single hierarchy. I will append a number "00" as a component in the Bundle name to represent that and add that in the specs too.

Are you saying that each data-signing-certificate is usable with one trust schema? This is too much restriction and is unacceptable.

I don't understand this comment. What Manika mentioned is that the current bundle can help with certificate retrieval for any trust schema that uses a single chain of certificates to verify a data packet (this is what single hierarchy is meant, it is not related in any way to name hierarchy or any restrictions on data/cert names). In principle, bundle can help Web-of-Trust models, but this will be defined/implemented, probably specific to an applications.

> It's possible that a trust model as no fixed trust anchor.

WebPKI has many root CAs, but the issuer of the highest level of intermediate CA is the trust anchor being used. Similarly, an NDN trust schema can have multiple trust anchors, and the bundle only needs to include up to the certificate directly issued by one of the trust anchors, but not the self-signed trust anchor. An exception is the verifier does not have the public key of the trust anchor, as described in note-17.

Bundle doesn't by itself impose trust model. It is just a combined set of certificates. It doesn't have knowledge of specific trust anchors, so whatever is included is included. Ultimately, it is Validator task to get proper certificates. Carrying them in a bundle is just a helper.

> > Each segment will always have complete certificates.
> > What's the benefit of this requirement?
>
> Actually, this is not a hard requirement. May be I can mention that in the spec that it is mostly for simplicity.
>
> Are you saying this is a MAY-level requirement? If so, verifier needs to be able to accept a bundle in which segment does not stop at certificate boundary, since a MAY-level requirement is truly optional at prover end.

As of now, it is MUST-level requirement. The goal for now to make implementation simple. This may change in the future and we may change the validator implementation, but I do not want to overcomplicate things for now.

#21 - 02/08/2017 03:42 PM - Junxiao Shi

> the current bundle can help with certificate retrieval for any trust schema that uses a single chain of certificates to verify a data packet

An example in WebPKI: *Let's Encrypt Authority X3* intermediate is issued by *ISRG Root X1* and cross-signed by *IdenTrust* root. Before *ISRG Root X1* is trusted by browsers, certificate chains will look like "example.com - *Let's Encrypt Authority X3 (issuer=IdenTrust)*". After *ISRG Root X1* becomes trusted by browsers, certificate chains will look like "example.com - *Let's Encrypt Authority X3 (issuer=ISRG)*". Note that *Let's Encrypt Authority X3 (issuer=IdenTrust)* and *Let's Encrypt Authority X3 (issuer=ISRG)* are two different certificates but they contain same public key.

Bringing this example to NDN trust schema: seeing example.com's certificate, the trust schema installed in Firefox 49 or below will derive *Let's Encrypt Authority X3 (issuer=IdenTrust)* as the issuer, while the trust schema installed in Firefox 50 or above will derive *Let's Encrypt Authority X3 (issuer=ISRG)* as the issuer.
How does the TLS server which intermediate should be sent in the bundle, if the bundle name does not contain an indicate on which trust schema is in use?

On the other hand, if we eliminate the requirement of sorting the certificates, this problem can be solved by including both *Let's Encrypt Authority X3 (issuer=IdenTrust)* and *Let's Encrypt Authority X3 (issuer=ISRG)* in the bundle, and verifier will take what they need from the unverified cache.

> bundle can help Web-of-Trust models, but this will be defined/implemented, probably specific to an applications.

In Web-of-Trust, there are multiple issuers for one certificate and all those issuers' certificates must be retrieved and verified. I don't think NDN trust schema can support such model, and thus it's not a problem specific to bundle.

> Bundle doesn't by itself impose trust model. It is just a combined set of certificates. It doesn't have knowledge of specific trust anchors, so whatever is included is included.

What would be the benefit of including a *self-signed certificate* (regardless of whether it is a trust anchor)? other than the situation in note-17.

> As of now, it is MUST-level requirement. The goal for now to make implementation simple. This may change in the future and we may change the validator implementation, but I do not want to overcomplicate things for now.

As indicated in note-17, there is an obvious drawback, and the so-called simplification is invalid because exactly same procedure is already implemented in UnixStreamTransport and can be abstracted into a reusable library routine.
A real simplification is: just use SegmentFetcher and retrieve all segments, and leave "stop retrieving after getting enough segments" for later optimization.

#22 - 02/08/2017 04:02 PM - Alex Afanasyev

Junxiao Shi wrote:

> > the current bundle can help with certificate retrieval for any trust schema that uses a single chain of certificates to verify a data packet
>
> An example in WebPKI: *Let's Encrypt Authority X3* intermediate is issued by *ISRG Root X1* and cross-signed by *IdenTrust* root. Before *ISRG Root X1* is trusted by browsers, certificate chains will look like "example.com - *Let's Encrypt Authority X3 (issuer=IdenTrust)*". After *ISRG Root X1* becomes trusted by browsers, certificate chains will look like "example.com - *Let's Encrypt Authority X3 (issuer=ISRG)*". Note that *Let's Encrypt Authority X3 (issuer=IdenTrust)* and *Let's Encrypt Authority X3 (issuer=ISRG)* are two different certificates but they contain same public key.
>
> Bringing this example to NDN trust schema: seeing example.com's certificate, the trust schema installed in Firefox 49 or below will derive *Let's Encrypt Authority X3 (issuer=IdenTrust)* as the issuer, while the trust schema installed in Firefox 50 or above will derive *Let's Encrypt Authority X3 (issuer=ISRG)* as the issuer.
> How does the TLS server which intermediate should be sent in the bundle, if the bundle name does not contain an indicate on which trust schema is in use?
>
> On the other hand, if we eliminate the requirement of sorting the certificates, this problem can be solved by including both *Let's Encrypt Authority X3 (issuer=IdenTrust)* and *Let's Encrypt Authority X3 (issuer=ISRG)* in the bundle, and verifier will take what they need from the unverified cache.

Why you bringing TLS here? You just effectively described a form of web-of-trust (multi-cert) model. This is not supported in our initial implementation of bundle fetcher nor bundle helper, but is irrelevant from the spec point of view. Bundle includes all certificates potentially needed, some may not be actually used. If use case of cert bundle will get to the point of specific trust model with specific trust anchors, then one free to construct a non-redundant bundle, but I don't know exactly where you going with it.

> > bundle can help Web-of-Trust models, but this will be defined/implemented, probably specific to an applications.
>
> In Web-of-Trust, there are multiple issuers for one certificate and all those issuers' certificates must be retrieved and verified. I don't think NDN trust schema can support such model, and thus it's not a problem specific to bundle.

Trust schema can support anything. You're referring a challenge in certificate retrieval (signature discovery). This is totally different question and

outside the scope for both current schema and cert bundle specs. We have been discussing a few proposals about this and there was one implementation (in ChronoChat) for that, but nothing is definitive yet.

> Bundle doesn't by itself impose trust model. It is just a combined set of certificates. It doesn't have knowledge of specific trust anchors, so whatever is included is included.

> What would be the benefit of including a *self-signed certificate* (regardless of whether it is a trust anchor)? other than the situation in note-17.

Ok. This is a different conversation. Self-signed cert != trust anchor both ways. Self-signed cert can be a trust anchor or not be a trust anchor. And trust anchor can be self-signed or can be a non-self signed.

To answer your latest questions. Yes, I don't immediately see a value of including self-signed cert regardless its role in trust schema.

> As of now, it is MUST-level requirement. The goal for now to make implementation simple. This may change in the future and we may change the validator implementation, but I do not want to overcomplicate things for now.

> As indicated in note-17, there is an obvious drawback, and the so-called simplification is invalid because exactly same procedure is already implemented in UnixStreamTransport and can be abstracted into a reusable library routine.
> A real simplification is: just use SegmentFetcher and retrieve all segments, and leave "stop retrieving after getting enough segments" for later optimization.

No. The point is to fetch only certs (bundle parts) that are actually needed. For first data packet under namespace, this could be all segments of the bundle, for others (in sub-branches), only one or two may be needed. That's the whole idea of not using the SegmentFetcher.

#23 - 02/08/2017 06:28 PM - Junxiao Shi

> You just effectively described a form of web-of-trust (multi-cert) model. This is not supported in our initial implementation of bundle fetcher nor bundle helper

Isn't it already supported, since the prover can put into the bundle more certificates than needed by trust schema?

> No. The point is to fetch only certs (bundle parts) that are actually needed. For first data packet under namespace, this could be all segments of the bundle, for others (in sub-branches), only one or two may be needed. That's the whole idea of not using the SegmentFetcher.

Yes, I figured that, but this is an optimization, not a requirement.
On the other hand, being able to include a certificate whose size is very close to Data packet size limit is a requirement for correct protocol operation.

#24 - 02/08/2017 06:30 PM - Alex Afanasyev

Junxiao Shi wrote:

> You just effectively described a form of web-of-trust (multi-cert) model. This is not supported in our initial implementation of bundle fetcher nor bundle helper

> Isn't it already supported, since the prover can put into the bundle more certificates than needed by trust schema?

Technically yes, it is just not a current focus. That's why I said not supported.

#25 - 01/02/2018 01:19 PM - Nicholas Gordon

- Related to deleted (Task #3950: Use CertificateBundleFetcher instead of DirectFetcher)

#26 - 01/02/2018 01:20 PM - Nicholas Gordon

- Blocks Task #3950: Use CertificateBundleFetcher instead of DirectFetcher added

#27 - 02/03/2018 08:58 PM - Davide Pesavento

- Target version changed from v0.6 to v0.7

#28 - 04/16/2018 08:44 PM - Davide Pesavento

- Blocks deleted (Feature #3891: Integrate fetching certificates using KeyBundle into the Validator)

#29 - 03/25/2019 02:53 PM - Junxiao Shi

- Subject changed from KeyBundle: design to CertificateBundle: design

- Status changed from Feedback to In Progress

- Assignee changed from Manika Mittal to Junxiao Shi

- % Done changed from 80 to 0

- Estimated time set to 9.00 h

Junxiao agreed on 20190325 call to write a spec of the certificate bundle.
We agreed that the "single RTT" goal of Certificate Bundle Packet Format is not always necessary, and initial implementation can derive bundle name from certificate name instead of Data packet name.

#30 - 04/02/2019 07:18 PM - Junxiao Shi

- Status changed from In Progress to Feedback

- % Done changed from 0 to 60

Certificate Bundle design

A certificate bundle collects multiple certificates necessary to validate a packet into a single segmented object.
When a publisher prepares a certificate bundle, it enables more efficient validation because the validator can retrieve the entire certificate chain in parallel.
Without a certificate bundle, the validator would have to wait until each certificate to arrive before knowing the name of its issuer, causing more round trips.

Packet Format

A certificate bundle is a segmented object that contains certificates.

For each Data packet:

- Name MUST contain 32=cert-bundle component.
- Name MUST end with a segment number.
- ContentType MUST be omitted.
- There is no signing requirement.
- FinalBlockId MUST be present in the last segment.

The payload of the segmented object is a sequence of CertificateV2 elements.
A certificate MAY span across multiple Data packets.

The certificates SHOULD be sorted in reverse order of the hierarchy: suppose *cert-A* is the issuer of *cert-B*, *cert-A* SHOULD come after *cert-B*.
This enables the validator to abort bundle retrieval as soon as it has received enough certificates in the lower part of the hierarchy, when it has locally cached the higher part of hierarchy.
The trust anchor SHOULD NOT be present in the certificate bundle, because the validator is expected to possess it.

Bundle Name Discovery

This section defines two common methods of finding certificate bundle name.
Application MAY define additional methods.

Name of a Data segment within a bundle is the bundle name plus a segment number component.

Deriving from Certificate Name

Every certificate SHOULD be published together with a certificate bundle.
For a certificate named /<SubjectName>/KEY/[KeyId]/[IssuerId]/[Version], its bundle SHOULD be named /<SubjectName>/KEY/[KeyId]/32=cert-bundle/[IssuerId]/[Version].

KeyLocator name normally ends at [KeyId].
Therefore, the 32=cert-bundle component is inserted after [KeyId], enabling a fetcher to derive bundle name from KeyLocator.

Provided in RDR Metadata

A producer MAY provide one or more certificate bundle names in the RDR metadata.
These certificate bundles SHOULD enable efficient validation of the RDR metadata packet itself, as well as all Data packets under the versioned name.

A bundle name is enclosed in an CertBundle TLV in RDR metadata's Content element:

```
<CertBundle>
  <Name>/subject/KEY/id/32=cert-bundle/issuer/35=%01</Name>
```

```
</CertBundle>
```

In the second example, two certificate bundle names are listed.
It is not required to include all certificates necessary to validate a packet in a single bundle.
The publisher MAY use multiple bundles to collectively provide certificates:

```
<CertBundle>
  <Name>/first/32=cert-bundle</Name>
</CertBundle>
<CertBundle>
  <Name>/second/32=cert-bundle/xx</Name>
</CertBundle>
```

The producer MAY indicate the number of segments in a certificate bundle to allow more efficient retrieval, using a SegmentNameComponent reflecting the last segment number:

```
<CertBundle>
  <Name>/subject/KEY/id/32=cert-bundle/issuer/35=%01</Name>
  <SegmentNameComponent>4</SegmentNameComponent>
</CertBundle>
```

CertBundle TLV-TYPE is taken from application range.
Unrecognized elements under CertBundle TLV follow TLV evolvability guidelines.

Low Level API

```
/** \brief Derive certificate bundle name from certificate name.
 */
Name
deriveCertBundleName(const Name& certName);
// Example: deriveCertBundleName("/subject/KEY/id/issuer/35=%01") == "/subject/KEY/id/32=cert-bundle/issuer/35
=%01"

/** \brief Derive certificate bundle name prefix from KeyLocator.
 */
Name
deriveCertBundlePrefixFromKeyLocator(const KeyLocator& kl);
// Example: deriveCertBundlePrefixFromKeyLocator(KeyLocator("/subject/KEY/id")) == "/subject/KEY/id/32=cert-bu
ndle"

/** \brief Encode a certificate bundle.
 *  \param certificates Certificates to be included in the bundle.
 *  \param bundleName name of bundle packets, excluding the segment number.
 *  \param keyChain KeyChain used to sign bundle packets.
 *  \param si SigningInfo used to sign bundle packets.
 */
std::vector<Data>
encodeCertBundle(const std::vector<Certificate>& certificates, const Name& bundleName,
                 KeyChain& keyChain, const SigningInfo& si);
// This function should internally use SegmentPublisher.

/** \brief Decode a certificate bundle.
 */
class CertBundleDecoder
{
public:
  signal::Signal<CertBundleDecoder, Certificate> certAvailable;

  /** \brief Append an arrived segment.
   *  \pre Prior segments, if any, has arrived.
   */
  void
  append(const Data& segment);
};
// This is a stream decoder: it can decode segments as they arrive in order, and does not need to
// wait for the entire certificate bundle to be retrieved. This allows a fetcher to stop retrieving
// later segments if it has collected enough certificate for validating a packet.

/** \brief Name and optionally segment count of a segmented object.
 */
struct SegmentedObjectRef
{
  Name prefix;
  optional<uint64_t> lastSegNum;
};
```

```
// extending existing type
class MetadataObject
{
public:
  const std::vector<SegmentedObjectRef>&
  getCertBundles() const;

  MetadataObject&
  setCertBundles(std::vector<SegmentedObjectRef> certBundles);
};
```

Instead of having these as static methods in publisher and fetcher APIs, this design opts to have separate low level APIs, so that logic and communication are separated. This separation improves modularity and testability.

Publisher API

```
/** \brief A certificate bundle being created and published.
 */
class CertBundleHandle
{
public:
  const Name&
  getBundleName();

  enum State {
    PENDING,     ///< bundle is being created
    READY,       ///< bundle is published
    FAILED,      ///< bundle creation has failed
    UNPUBLISHED, ///< bundle creation is canceled and bundle should be unpublished
  };

  /** \brief Get bundle creation and publishing state.
   */
  State
  getState() const;

  /** \brief Emitted when state has changed.
   */
  signal::Signal<X, State>& afterStateChange;
  // CertBundleHandle type must be copyable. Thus, this signal must live elsewhere
  // (e.g. an internal struct owned by CertBundleBuilder) and only a reference is stored.

  /** \brief Get last error during bundle creation.
   *  \pre getState() == FAILED
   */
  std::tuple<int, std::string>
  getError() const;

  /** \brief Get number of certificates included in the bundle.
   *  \pre getState() == READY
   */
  size_t
  countCerts() const;

  /** \brief Obtain bundle packets.
   *  \pre getState() == READY
   */
  const std::vector<Data>&
  get() const;

  /** \brief Cancel bundle creation and unpublish.
   */
  void
  cancel();
};

/** \brief Unpublish certificate bundle upon destruction.
 */
class ScopedCertBundleHandle;

/** \brief Create certificate bundles.
 */
class CertBundleBuilder
{
```

```
public:
  /** \brief Constructor.
   *  \param keyChain KeyChain used to sign bundle packets.
   *                  CertBundleBuilder can also collect certificates from its PIB.
   *  \param si SigningInfo used to sign bundle packets.
   */
  CertBundleBuilder(KeyChain& keyChain, const SigningInfo& si);

  /** \brief Enable collecting certificates from a CertificateStorage.
   */
  void
  setStorage(CertificateStorage& storage);

  /** \brief Enable retrieving certificates using a CertificateFetcher.
   *  \pre setStorage has been invoked.
   */
  void
  setFetcher(CertificateFetcher& fetcher);

  /** \brief Emitted when \c publish is invoked.
   *
   *  This allows an external component to know when a new certificate bundle is being
   *  created and hook onto its signals.
   */
  signal::Signal<CertBundleBuilder, CertBundleHandle> afterAdd;

  /** \brief Start preparing and publishing a certificate bundle.
   */
  CertBundleHandle
  add(const Name& certName, const Name& bundleName);

  CertBundleHandle
  add(const Name& certName)
  {
    return add(certName, deriveCertBundleName(certName));
  }
};
// This type builds certificate bundles starting from leaf certificate names passed to add() method.
// It can collect intermediate certificates from the local KeyChain, caches in a
// CertificateStorage, or the network via a fetcher. This type exposes prepared bundles via signals,
// but does not make them available for retrieval by itself. This design features a separation of
// responsibilities, with this type being responsible for building certificate bundles, while other
// types listed below are responsible for publishing them.

/** \brief Insert certificate bundles to an InMemoryStorage.
 */
class CertBundleImsInserter
{
public:
  /** \brief Constructor.
   *  \p wantDelete whether to delete bundle packets when bundle is unpublished.
   */
  CertBundleImsInserter(CertBundleBuilder& cbb, InMemoryStorage& ims, bool wantDelete = true);
};

/** \brief Serve certificate bundles on a face.
 */
class CertBundleProducer
{
public:
  /** \brief Constructor.
   *  \p wantRegisterPrefix if true, each bundle registers a prefix on the local forwarder;
   *                        otherwise, each bundle creates an InterestFilter only.
   */
  CertBundleProducer(CertBundleBuilder& cbb, Face& face, bool wantRegisterPrefix = true);

  Face&
  getFace();

  /** \brief Emitted when a certificate bundle has been published and is available for retrieval.
   */
  signal::Signal<CertBundleProducer, CertBundleHandle> afterPublish;
  // If prefix registration is necessary, this is emitted after prefix registration is completed.
};
```

```
/** \brief Insert certificate bundles to repo-ng.
 */
class CertBundleRepongInserter
{
public:
  /** \brief Constructor.
   *  \p wantDelete whether to delete bundle packets when bundle is unpublished.
   */
  CertBundleRepongInserter(CertBundleProducer& cbp, const Name& repoCommandPrefix,
                           bool wantDelete = false);
};
// This type should hook onto CertBundleProducer::afterPublish signal, and send a repo insertion
// command after a bundle is available for retrieval.
// If wantDelete is requested, this type should also send a repo deletion command when a bundle
// is unpublished.
```

Fetcher API

```
class CertBundleFetcher
{
public:
  struct Options
  {
    /// maximum number of certificate bundles allowed for each original packet
    int maxBundles = 1;
    /// InterestLifetime for bundle Interest
    time::milliseconds interestLifetime = 3000_ms;
    /// bundle fetching options
    SegmentFetcher::Options segmentFetcherOptions;
    /// whether to send Interest with NextHopFaceId when available
    bool wantDirectFetch = false;
    /// whether to send Interest without NextHopFaceId; implied true if wantDirectFetch is false
    /// or NextHopFaceId is unavailable
    bool wantNormalFetch = false;
  };
  // retransmitInterval is separate from interestLifetime, so that Interest is retransmitted before
  // PIT entry disappears, allowing strategy to make better decisions based on PIT entry.
  // This behavior should be implemented in DataFetcher, imported from ndncatchunks.

  CertBundleFetcher(unique_ptr<CertificateFetcher> inner, Face& face,
                    const Options& options = Options());

  /** \brief Indicate the certificate bundle at \p bundleName contains certificates necessary to
   *         validate packets under \p prefix .
   */
  void
  addBundle(const Name& prefix, const Name& bundleName, optional<uint64_t> lastSegNum);
  // Application should register bundle names from an RDR metadata packet with this function,
  // so that the fetcher would retrieve the specified bundles.
};
// Upon receiving a certificate request, this fetcher retrieves a certificate bundle registered by
// addBundle() if matched, or deriveCertBundleName(certName) otherwise. If a certificate bundle
// does not contain all necessary certificates for an original packet (i.e. ValidationState),
// another certificate request would arrive; this fetcher may fetch another bundle until the limit
// given in Options::maxBundles is reached, after which all requests are redirected to the inner
// fetcher.
//
// Retrieval of a single certificate bundle should be handled by SegmentFetcher. SegmentFetcher
// must be extended with:
// (1) a signal that is emitted a segment arrives in order and has been validated;
// (2) ability to send Interests with NextHopFaceId;
// (3) ability to set known last segment number.
// This fetcher hooks onto the segment in-order arrival signal and passes the segment to an internal
// CertBundleDecoder, and inserts decoded certificates into the CertificateStorage.
//
// If bundle retrieval fails (after SegmentFetcher has exhausted its retries), this fetcher passes
// the certificate request to the inner fetcher. The next certificate request may start a new
// bundle retrieval subject to Options::maxBundles limit, as described above.
```

#31 - 05/15/2019 02:52 PM - Junxiao Shi

- Blocks deleted (Feature #3892: Integrate publishing of KeyBundle into Face::setInterestFilter)

#32 - 08/27/2019 08:52 PM - Davide Pesavento

- Target version changed from v0.7 to v0.8

#33 - 09/24/2019 01:50 PM - Junxiao Shi

- Tags set to CertificateBundle

- Status changed from Feedback to Closed

- % Done changed from 60 to 100

Closing because there's no objection on this design.
Implementation tasks have been created: #5004#5005#5006#5007.

#34 - 05/18/2020 04:03 PM - Davide Pesavento

> ContentType MUST be omitted.

Why?

> The producer MAY indicate the number of segments in a certificate bundle [...] using a SegmentNameComponent reflecting the last segment number

Why not a FinalBlockId?

#35 - 05/18/2020 04:46 PM - Junxiao Shi

> > ContentType MUST be omitted.
>
> Why?

The name alone identifies the content type. There's no need for additional identification.
Allowing "blob" type doesn't convey additional meaning, but only wastes bytes.

> > The producer MAY indicate the number of segments in a certificate bundle [...] using a SegmentNameComponent reflecting the last segment number
>
> Why not a FinalBlockId?

The bundle spec is defined to adopt Naming Conventions rev2, so there's no need for wrapping into FinalBlockId.

#36 - 05/18/2020 05:20 PM - Davide Pesavento

Junxiao Shi wrote:

> > > ContentType MUST be omitted.
> >
> > Why?
>
> The name alone identifies the content type. There's no need for additional identification.
> Allowing "blob" type doesn't convey additional meaning, but only wastes bytes.

32=cert-bundle is 13 bytes, certificates are hundreds of bytes each... and you're worried about 3 bytes?

> > > The producer MAY indicate the number of segments in a certificate bundle [...] using a SegmentNameComponent reflecting the last segment number
> >
> > Why not a FinalBlockId?
>
> The bundle spec is defined to adopt Naming Conventions rev2, so there's no need for wrapping into FinalBlockId.

Still, FinalBlockId sounds more general to me.

#37 - 05/18/2020 05:28 PM - Junxiao Shi

> ContentType MUST be omitted.

Why?

> The name alone identifies the content type. There's no need for additional identification.
> Allowing "blob" type doesn't convey additional meaning, but only wastes bytes.

32=cert-bundle is 13 bytes, certificates are hundreds of bytes each... and you're worried about 3 bytes?

Not just that. Disallowing ContentType improves the uniqueness of encodings.
Yes I know non-uniqueness could come from many other places, but it's better to reduce such places than to introduce them.
And no, other older protocols that allow such ambiguity are not reasons for this protocol to make the same mistake.

> The producer MAY indicate the number of segments in a certificate bundle [...] using a SegmentNameComponent reflecting the last segment number

Why not a FinalBlockId?

> The bundle spec is defined to adopt Naming Conventions rev2, so there's no need for wrapping into FinalBlockId.

Still, FinalBlockId sounds more general to me.

No, this is an application layer protocol and there's no reason to copy from network layer.