Eluvio Content Fabric V2 Spec

Eluvio

2022

Definitions

Node A server which stores and serves parts.

Provider An individual or organization which owns, secures, and operates nodes.

Tenant An individual or organization which owns content.

Content A versioned set of data which is owned by a tenant.

Space A group of providers and tenants, where providers agree to run nodes that serve content owned by a tenant according to a common set of rules.

Part A part is a sequence of bytes stored in the space, referenced by its hash.

Content Object Version A collection of parts created by a tenant, referenced by its hash.

Content Object A collection of versions.

Library A 'folder' of content objects owned by a tenant with a permission structure what determines who within a tenancy is able to create, modify, delete content objects and content object versions.

KMS A tenant-owned server which holds keys for encrypting/decrypting content which the tenant stores in the space.

The following entities are defined by fixed length identifiers as follows:

Entity	Identifier	Substrate Type
Node	$\mathtt{ID}_{\mathtt{node}}$	10-byte array
Space	$\mathtt{ID}_{\mathtt{space}}$	10-byte array
Content Object	$\mathtt{ID}_{\mathtt{conq}}$	10-byte array
Content Object Version	$\mathtt{ID}_{\mathtt{version}}$	32-byte array
KMS	$\mathtt{ID}_{\mathtt{kms}}$	10-byte array
Library	$\mathtt{ID}_{\mathtt{lib}}$	unsigned 16-bit integer

1 Spaces

The space functions as the top level governance structure of the fabric that orchestrates how providers cooperate to serve tenant data. It is responsible for

- Creating providers
- Creating tenants
- Defining rules of the space which tenants/providers agree to abide by
- Reserving slashable bonds for both providers and tenants
- Governance, including
  - Admitting new tenants/providers
  - Removing misbehaving tenants/providers
  - Slashing tenants/providers
  - Changing rules

1.1 Space rules

Provider Bond An amount, $\texttt{Bond}_{\texttt{prov}}$, of currency each provider must lock up in order to participate within the space. Funds can be slashed from here if a provider misbehaves.

Tenant Bond An amount, $\texttt{Bond}_{\texttt{ten}}$, of currency each tenant must lock up in order to participate within the space. Funds can be slashed from here if a tenant misbehaves.

SLAs Specifications for availability requirements provider nodes must have.

Partition number The partitioning constant for part storage

2 Providers

A provider is an entity which owns nodes within a space. It is responsible for ensuring its nodes abide by the space's rules, risking it's bond if it misbehaves. Providers also have a permission structure which can associated levels of privilege with cryptographic keys.

2.1 Provider Permissions

Provider keys have the following permission levels, from most to least privileged

1. $\texttt{Perm}_{\texttt{root}}$ Root level
   - add/remove admins (effectively allows for admin key rotation)
2. $\texttt{Perm}_{\texttt{admin}}$ Admin level
   - add/remove nodes
   - bill tenants
3. $\texttt{Perm}_{\texttt{node}}$ Node level
   - Co-author versions with tenants
   - Mark itself as no longer pending
   - Participate in part networking

Keys with a higher level may set the permission level of any keys strictly below it. For example, a key with $\texttt{Perm}_{\texttt{root}}$ may give other keys $\texttt{Perm}_{\texttt{admin}}$, but $\texttt{Perm}_{\texttt{admin}}$ keys may not give other keys $\texttt{Perm}_{\texttt{admin}}$ or change the rights of a key with $\texttt{Perm}_{\texttt{admin}}$

2.2 Provider Blockchain Calls

In addition to setting permissions on keys, we have the following calls

CreateProvider($k_{\texttt{origin}}, \texttt{ID}_{\texttt{space}}, \texttt{ID}_{\texttt{prov}}$) Creates the provider

- Check governance to see whether origin can create a provider
- Creates $\texttt{ID}_{\texttt{prov}}$ and sets its space to $\texttt{ID}_{\texttt{space}}$
- Sets $k_{\texttt{origin}}$ as the root key ($k_{\texttt{root}}$) of $\texttt{ID}_{\texttt{prov}}$
- Sets $k_{\texttt{origin}}$ as a key for $\texttt{ID}_{\texttt{prov}}$ with level $\texttt{Perm}_{\texttt{root}}$
- Bonds $\texttt{Bond}_{\texttt{prov}}$ from $k_{\texttt{root}}$ to the space under $\texttt{ID}_{\texttt{prov}}$

AddNode($k_{\texttt{origin}}, \texttt{ID}_{\texttt{prov}}, \texttt{ID}_{\texttt{node}}, k_{\texttt{node}}, \texttt{Loc}_{\texttt{node}}$) adds a node

- Checks that $k_{\texttt{origin}}$ has permission $\texttt{Perm}_{\texttt{admin}}$ or above for $\texttt{ID}_{\texttt{prov}}$.
- Creates a node $\texttt{ID}_{\texttt{node}}$ with locator $\texttt{Loc}_{\texttt{node}}$

- Registers $k_{\texttt{node}}$ to $\texttt{ID}_{\texttt{prov}}$ with permission level $\texttt{Perm}_{\texttt{node}}$ [1]
- Marks the node as pending while it syncs up parts with the rest of the space

ConfirmNode($k_{\texttt{origin}}, \texttt{ID}_{\texttt{prov}}, \texttt{ID}_{\texttt{node}}$) marks a node as no longer pending

- Checks that $k_{\texttt{origin}}$ has permissions $\texttt{Perm}_{\texttt{node}}$ or above for $\texttt{ID}_{\texttt{prov}}$
- Sets $\texttt{ID}_{\texttt{node}}$ to no longer pending

RemoveNode($k_{\texttt{origin}}, \texttt{ID}_{\texttt{prov}}, \texttt{ID}_{\texttt{node}}$) removes a node

- Checks that $k_{\texttt{origin}}$ has permissions $\texttt{Perm}_{\texttt{admin}}$ or above for $\texttt{ID}_{\texttt{prov}}$
- Removes all $\texttt{ID}_{\texttt{node}}$ information from the space and provider

BillTenant TODO

[1] Should this error if the key already exists within the permissions scheme?

3 Tenants

A tenant is an owner and creator of content. They are responsible for providing a service available to providers' nodes which can manage keys and encrypt/decrypt content.

3.1 Tenant Permissions

Tenant keys have the following permission levels, from most to least privileged

1. $\texttt{Perm}_{\texttt{root}}$ can add/remove admins
2. $\texttt{Perm}_{\texttt{admin}}$ can add/remove kmses, create libraries, and add/remove users from libraries
3. $\texttt{Perm}_{\text{kms}}$ can co-author content object versions with provider nodes

3.2 Tenant Blockchain Calls

CreateTenant$(k_{\texttt{origin}} = k_{\texttt{root}}, \texttt{ID}_{\texttt{space}}, \texttt{ID}_{\texttt{tenant}})$ creates a tenancy

- Checks governance to see whether origin can create a tenant
- Creates $\texttt{ID}_{\texttt{tenant}}$ and sets its space to $\texttt{ID}_{\texttt{space}}$
- Sets $k_{\texttt{root}}$ as the creator of $\texttt{ID}_{\texttt{tenant}}$
- Sets $k_{\texttt{root}}$ as a key for $\texttt{ID}_{\texttt{tenant}}$ with level $\texttt{Perm}_{\texttt{root}}$
- Bonds $\texttt{Bond}_{\texttt{ten}}$ from $k_{\texttt{root}}$ to the space under $\texttt{ID}_{\texttt{tenant}}$

AddKMS$(k_{\texttt{origin}}, \texttt{ID}_{\texttt{tenant}}, \texttt{ID}_{\text{kms}}, k_{\text{kms}}, \texttt{Loc}_{\text{kms}})$ creates a kms

- Checks that $k_{\texttt{origin}}$ has permission $\texttt{Perm}_{\texttt{admin}}$ or above for $\texttt{ID}_{\texttt{tenant}}$.
- Creates a KMS $\texttt{ID}_{\text{kms}}$ within $\texttt{ID}_{\texttt{tenant}}$ with locator $\texttt{Loc}_{\text{kms}}$
- Registers k_{kms} to $\texttt{ID}_{\texttt{tenant}}$ with permission level $\texttt{Perm}_{\text{kms}}$

RemoveKMS$(k_{\texttt{origin}}, \texttt{ID}_{\texttt{tenant}}, \texttt{ID}_{\text{kms}})$ removes a node

- Checks that $k_{\texttt{origin}}$ has permissions $\texttt{Perm}_{\texttt{admin}}$ or above for $\texttt{ID}_{\texttt{tenant}}$
- Removes all $\texttt{ID}_{\text{kms}}$ information from the space and tenancy
- Removes k_{kms} from $\texttt{ID}_{\texttt{tenant}}$

TODO: Remove Tenant, Top up billing balance

4 Libraries

Libraries group keys together which may create/modify content within a specific context. They act as a way to separate the keys which manage tenant infrastructure (like KMSs) from keys which manage content. The goal of the library is to have different levels of keys and different rules for different groups of content. For example, one library could hold staging content which is not publicly accessible outside of library owners. Content could be created there and then moved to a production library which has greater visibility.

4.1 Library Rights

Any `user` in a library has some associated rights, $\texttt{Rights}_{\texttt{user}} \in \mathcal{R}$. The exact format of this structure is TDB (currently they're bitflags), but should at the very least be able to answer the following questions:

$\texttt{IsAdmin}(\texttt{Rights}_{\texttt{user}})$: Is the user allowed to add other users as non-admins

$\texttt{CanEdit}(\texttt{Rights}_{\texttt{user}})$: Is the user allowed to edit content

4.2 Library Blockchain Calls

CreateLibrary$(k_{\texttt{origin}}, \texttt{ID}_{\texttt{tenant}}, \texttt{ID}_{\texttt{lib}}, \text{name})$ creates a library

- Checks the origin has permission $\texttt{Perm}_{\texttt{admin}}$ within $\texttt{ID}_{\texttt{tenant}}$
- Creates a library with $\texttt{ID}_{\texttt{lib}}$ and the given name within $\texttt{ID}_{\texttt{tenant}}$
  - Note that for convenience, this call could also set $\texttt{Rights}_{\texttt{origin}}$ such that $\texttt{IsAdmin}(\texttt{Rights}_{\texttt{origin}})$ is true

SetRights$(k_{\texttt{origin}}, \texttt{ID}_{\texttt{tenant}}, \texttt{ID}_{\texttt{lib}}, k_{\texttt{target}}, \texttt{Rights}_{\texttt{target}})$ sets rights as a library admin

- Check that $\texttt{IsAdmin}(\texttt{Rights}_{\texttt{origin}})$
- Checks that $!\texttt{IsAdmin}(\texttt{Rights}_{\texttt{target}})$
- Sets the rights of $k_{\texttt{target}}$ in $\texttt{ID}_{\texttt{lib}}$ to $\texttt{Rights}_{\texttt{target}}$

TenantSetRights$(k_{\texttt{origin}}, \texttt{ID}_{\texttt{tenant}}, \texttt{ID}_{\texttt{lib}}, k_{\texttt{target}}, \texttt{Rights}_{\texttt{target}})$ sets rights as a tenant admin

- Check that $k_{\texttt{origin}}$ has $\texttt{Perm}_{\texttt{admin}}$ in $\texttt{ID}_{\texttt{tenant}}$
- Sets the rights of $k_{\texttt{target}}$ in $\texttt{ID}_{\texttt{lib}}$ to $\texttt{Rights}_{\texttt{target}}$

This also provides a way for other parts of the blockchain if a given key has edit access to a library.

5 Content Objects

Content objects are the main way tenants store and retrieve data, globally referenced by $(\texttt{ID}_{\texttt{tenant}}, \texttt{ID}_{\texttt{conq}})$. They are created by storing data in a node, who calls **CommitVersion** with a digest of the data. Once the verison is commited, other nodes in the space can retrieve the content object. Once a sufficient number of nodes retrieve copies of the content object, the original authoring node submits a **ConfirmVersion** which marks the commit as finalized.

In order to prevent nodes from creating arbitrary versions without permission of tenants, a version commit message (VCM) and signature $\mathsf{sig}_{\mathsf{VCM}}$ by a tenant must be provided in the **CommitVersion** call. A VCM contains the following: TODO: nicer formatting

```
VersionCreateMessage {
  originator: ProviderId,
  tenant_id: TenantId,
  content_object_id: ContentObjectId,
  tlp_size: compact uint,
  digest: VersionId,
  ts: u64,
}
```

5.1 Content Objects and Libraries

Libraries are the permission structure that describes who can create content objects and content object versions. Upon creation, an $\texttt{Option}\langle\texttt{ID}_{\texttt{lib}}\rangle$ field is specified. If that field is $\texttt{Some}(\texttt{ID}_{\texttt{lib}})$, then any key k for which $\texttt{CanEdit}(\texttt{Rights}_{\texttt{k}})$ can create/modify versions. Otherwise, if $\texttt{ID}_{\texttt{lib}}$ is $\texttt{None}$, then only keys with at least $\texttt{Perm}_{\texttt{admin}}$ in $\texttt{ID}_{\texttt{tenant}}$ can create/modify versions. This makes libraries an optional component: if you want to use them, set them up with the proper keys and use them. If you'd like to ignore them, you can just use your tenant admin keys for creating/modifying content.

It's helpful to picture libraries as a filesystem with one level of folders:

```
Content Objects
├── Content Object 1      ← Operated by tenant admins
├── Content Object 2      ← Operated by tenant admins
├── Library 1
│   ├── Content Object 3  ← Operated by Library 1 admins
│   └── Content Object 4  ← Operated by Library 1 admins
└── Library 2
    ├── Content Object 5  ← Operated by Library 2 admins
    └── Content Object 6  ← Operated by Library 2 admins
```

Content objects are universally referred to by $(\texttt{ID}_{\texttt{tenant}}, \texttt{ID}_{\texttt{conq}})$, hence they can be moved between libraries. A sample use case would be to have one library for staging and one for production.

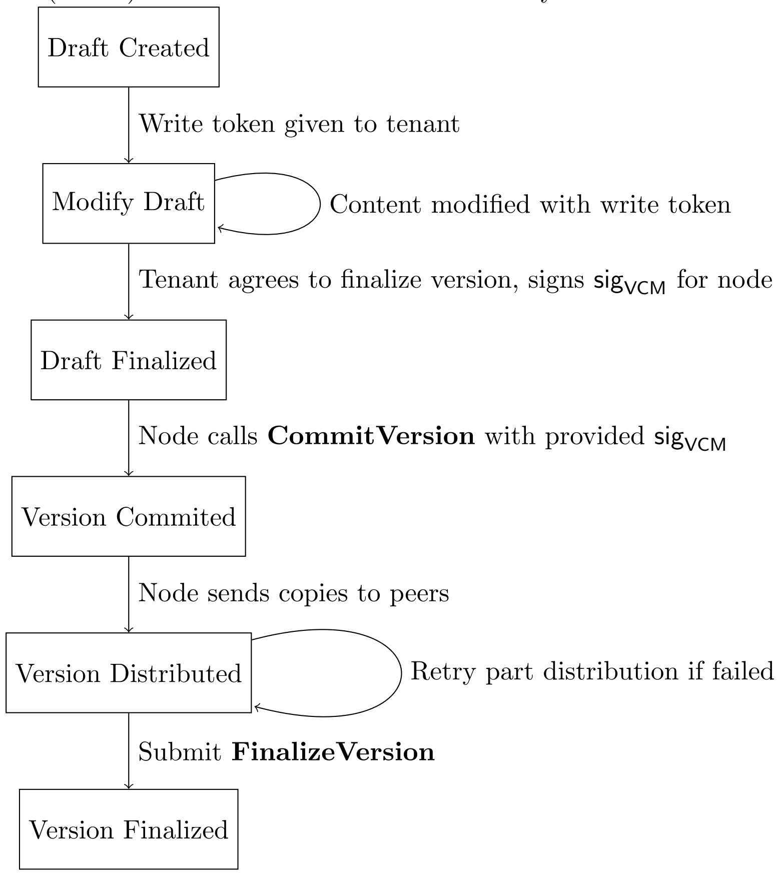
```
Content Objects
```

```
│
└─ Staging Library
│  └─ Staging Movie A ← Can move to production when finished
│  └─ Staging Movie B
└─ Production Library ← Things served to end users
   └─ Public Production Movie C
```

Loosely protected staging keys can be assigned to modify staging content only viewed internally in the tenancy. When the content is ready for production, a tenant admin (or similar role) can move the content from the staging library in to the production library, where keys are much more closely guarded.

5.2 Content Object Lifecycle

TODO(WILL): I'm sure there's more to this lifecycle


Draft Created
Write token given to tenant
Modify Draft
Content modified with write token
Tenant agrees to finalize version, signs $\mathsf{sig}_{\mathsf{VCM}}$ for node
Draft Finalized
Node calls **CommitVersion** with provided $\mathsf{sig}_{\mathsf{VCM}}$
Version Commited
Node sends copies to peers
Version Distributed
Retry part distribution if failed
Submit **FinalizeVersion**
Version Finalized

5.3 Content Types

TODO: Discuss

5.4 Content Object Blockchain Calls

CreateContentObject$(k_{\texttt{origin}}, \texttt{ID}_{\texttt{tenant}}, \texttt{ID}_{\texttt{conq}}, \texttt{Option}\langle\texttt{ID}_{\texttt{lib}}\rangle)$

- If $\texttt{Option}\langle\texttt{ID}_{\texttt{lib}}\rangle = \texttt{None}$, checks that $k_{\texttt{origin}}$ has at least $\texttt{Perm}_{\texttt{admin}}$ in $\texttt{ID}_{\texttt{tenant}}$
- Otherwise, if $\texttt{Option}\langle\texttt{ID}_{\texttt{lib}}\rangle = \texttt{Some}(\texttt{ID}_{\texttt{lib}})$, checks that $k_{\texttt{origin}}$ has $\texttt{CanEdit}(\texttt{Rights}_{\texttt{origin}})$
- Registers $\texttt{ID}_{\texttt{conq}}$ with its associated $\texttt{Option}\langle\texttt{ID}_{\texttt{lib}}\rangle$ to $\texttt{ID}_{\texttt{tenant}}$,

CommitVersion$(k_{\texttt{origin}}, \texttt{ID}_{\texttt{tenant}}, \texttt{ID}_{\texttt{conq}}, \mathsf{VCM}, k_{\texttt{signer}}, \mathsf{sig}_{k_{\texttt{signer}}, \mathsf{VCM}})$

- Checks that $k_{\texttt{origin}}$ has permission level $\texttt{Perm}_{\texttt{node}}$ within the $\texttt{ID}_{\texttt{prov}}$ specified in VCM.
- Retrieve the $\texttt{Option}\langle\texttt{ID}_{\texttt{lib}}\rangle$ for $(\texttt{ID}_{\texttt{tenant}}, \texttt{ID}_{\texttt{conq}})$.
  - If $\texttt{Option}\langle\texttt{ID}_{\texttt{lib}}\rangle = \texttt{None}$, checks that $k_{\texttt{signer}}$ has at least $\texttt{Perm}_{\texttt{admin}}$ in $\texttt{ID}_{\texttt{tenant}}$
  - Otherwise, if $\texttt{Option}\langle\texttt{ID}_{\texttt{lib}}\rangle = \texttt{Some}(\texttt{ID}_{\texttt{lib}})$, checks that $k_{\texttt{signer}}$ has $\texttt{CanEdit}(\texttt{Rights}_{\texttt{origin}})$
- Checks that $\mathsf{sig}_{k_{\texttt{signer}}}$ is a valid signature of VCM
- Stores the version $\texttt{ID}_{\texttt{version}}$ with the relevant metadata in VCM and a pending flag

FinalizeVersion$(k_{\texttt{origin}}, \texttt{ID}_{\texttt{prov}}, \texttt{ID}_{\texttt{tenant}}, \texttt{ID}_{\texttt{conq}}, \texttt{ID}_{\texttt{version}})$

- Checks that $k_{\texttt{origin}}$ has permission level $\texttt{Perm}_{\texttt{node}}$ within $\texttt{ID}_{\texttt{prov}}$.
- Checks that $\texttt{ID}_{\texttt{prov}}$ matches the $\texttt{ID}_{\texttt{prov}}$ stored in the metadata of $(\texttt{ID}_{\texttt{tenant}}, \texttt{ID}_{\texttt{conq}}, \texttt{ID}_{\texttt{version}})$
- Removes the pending flag from $(\texttt{ID}_{\texttt{tenant}}, \texttt{ID}_{\texttt{conq}}, \texttt{ID}_{\texttt{version}})$

6 Key Management Services (KMSs)

TODO

7 Part networking

TODO: @Serban