Content Access Control and Best Practices

Last Updated October 2022

Summary: This document is intended for Tenants, Publishers, and Developers interested to learn the access control and authorization mechanisms for securing content in the Content Fabric, and how the capabilities apply to typical streaming distribution cases. The last section summarizes best practices for content security.

Introduction to Access Control in the Content Fabric

Each content object that exists in the Content Fabric has an associated access control policy, which is fully controlled by the content owner. All fabric functions and key services functions evaluate this policy in order to provide any API or key services. From this perspective, access to a content object is entirely in the hands of the content owner. The access control is a function of the Base Access Control Policies of the object and any applicable Authorization Policies that have been configured on the object. We describe both in the following sections. Details of the Content Security Protocol implementation including cyphers, algorithms, etc. can be found here.

Base Access Control Policies

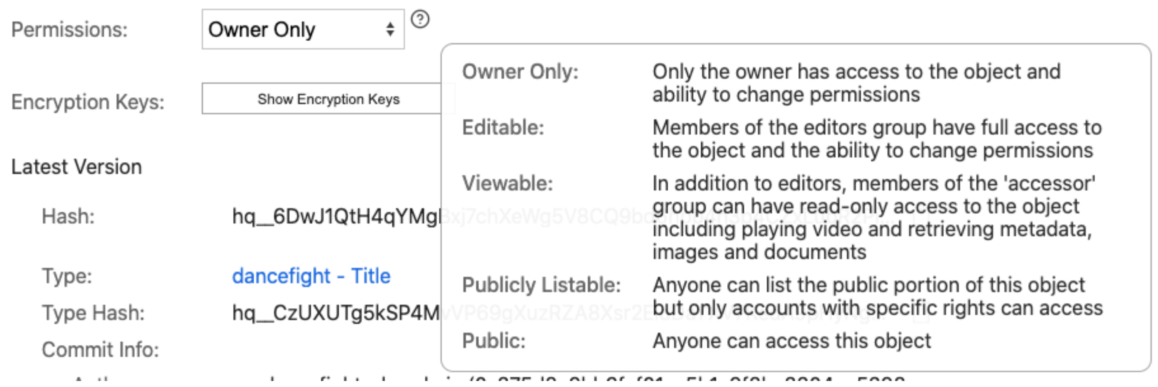
Content Objects in the Fabric are governed by a foundational access control policy based on a user/group-capability model that refers to the capability of a user or group of users to update ("write") to the object, change its permissions, or to access the object ("read" its metadata or its dynamic offerings via bitcode representations).

Each user or group is defined by the public blockchain address of the entity, and users may be added to groups by addresses that are delegated as group managers. Access rights are separated into the ability to a) update the object and its permissions "Editors", b) the ability to read private metadata of the object and access its dynamic offerings "Accessors", and c) the ability to read public metadata of the object (default for any user). These primitive capability groups are administered in the base smart contracts for the object.

A privileged owner of a Content Object can through the APIs or the Fabric browser set any of the following access policies for a Content Object:

- **Owner only access:** Only the owner of the object (that is, publisher) can access the object or change its permissions. No one but the owner can read the object.

- **Editable access:** Members with Edit permissions have full access to the object including the ability to write to the object or to change its permissions, and the ability to read its public and private metadata, any file output (e.g. images, docs, etc.), and its dynamic representations ("offerings") such as playable video, whether public or not.
- **Viewable access**: Members with Accessor permissions have full read-only access to the object including retrieving its public and private metadata, any file output (e.g. images, docs, etc.), and its dynamic representations ("offerings") such as playable video, whether public or not.
- **Publicly listable**: Anyone has read-only access to the object to list the public metadata of the object and its playable offerings. Only those with "Accessor" level permissions can list the non-public metadata and offerings, and only those with "Edit" permissions can write to the object or change its permissions.
- **Public:** Anyone has read-only access to the object (both public and private parts). Only those with "Edit" write can write to the object or change its permissions.



Permissions: Owner Only

Encryption Keys: Show Encryption Keys

Latest Version

Hash: hq__6DWJ1QtH4qYMg

Type: dancefight - Title

Type Hash: hq__CzUXUTg5kSP4M

Commit Info:

Owner Only:	Only the owner has access to the object and ability to change permissions
Editable:	Members of the editors group have full access to the object and the ability to change permissions
Viewable:	In addition to editors, members of the 'accessor' group can have read-only access to the object including playing video and retrieving metadata, images and documents
Publicly Listable:	Anyone can list the public portion of this object but only accounts with specific rights can access
Public:	Anyone can access this object

Extensible Authorization Policies

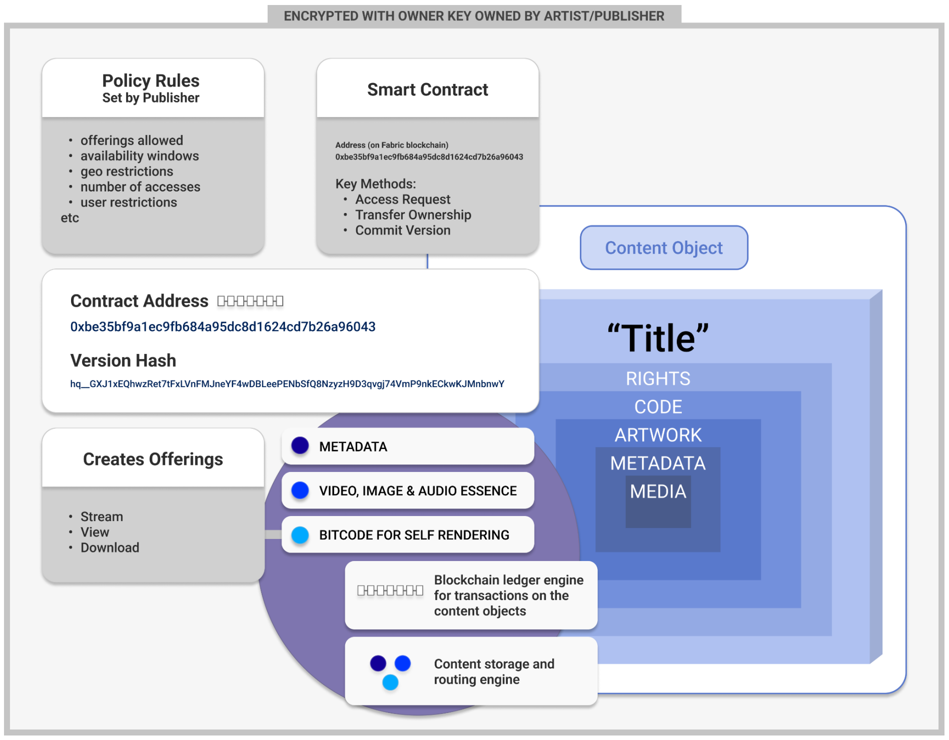
On top of the base access control policies the Fabric provides authorization policies as a flexible means to implement business rules that control access to content objects. (Details of the Authorization Policies can be found here.) The policy language offers a powerful expression syntax that can combine user data, content metadata, and API call information in order to allow or deny access to a given resource.

Additionally, a single policy can manage groups of related content objects, and a single content object can be tied to multiple policies. Policies can apply to access (read/write) to any data path within the content object or to any run-time 'offering' from the object. The policy implementation can also be tied to run-time bitcode that operates on content objects, separating the decision logic (what policy) from the implementation of the bitcode.

Policies are created by configuring a “policy object” (a type of content object). This process requires that the user have “Edit level” access rights on the policy object as well as on the content objects that should be governed by that policy object. In order to authorize against a policy, the client application forms a client-signed token containing a reference to the ID of the policy content object and the group membership or other attestation information (such as cross chain token balances, ec.), and signs the token. The token’s signature is verified on the Fabric side and the delegation policy is verified including its signature, and the policy is evaluated across all relevant paths (metadata, bitcode, links to other objects, etc.).

Group Administration and Bootstrapping

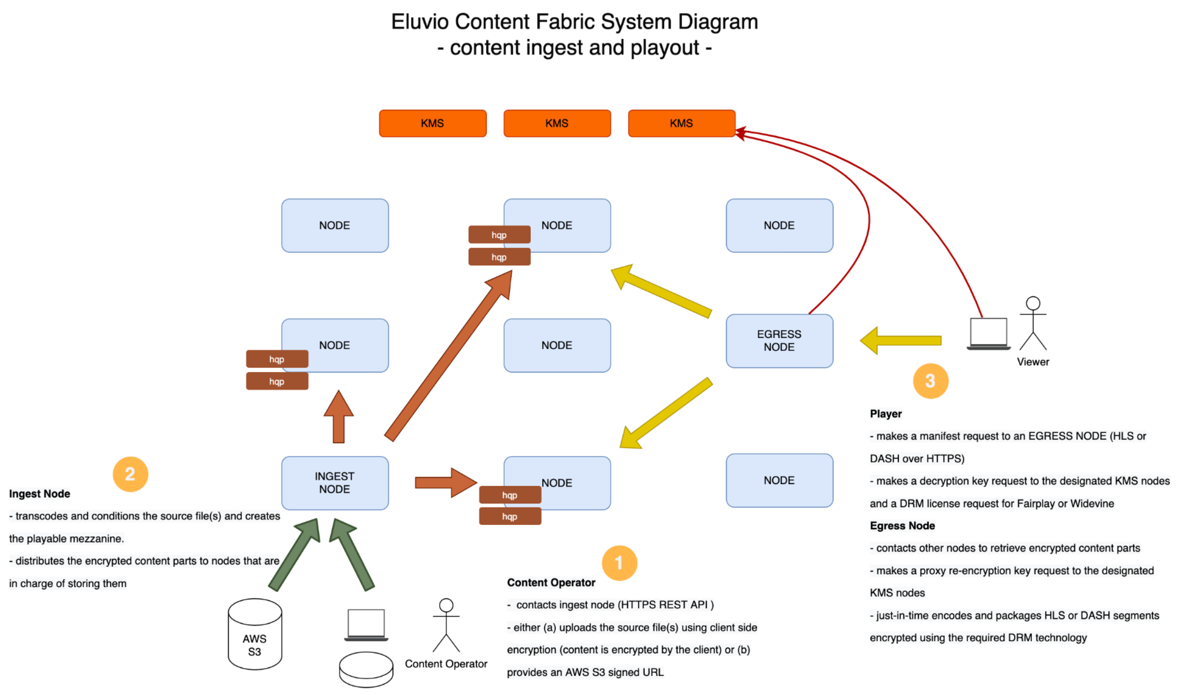
Content Object access control and authorization can be administered by a Group as well as by individual users. By convention, each Tenant in the Content Fabric is created with a Tenant Admin Group and one funded member address. The Tenant Admin group has permission to create additional groups and create content types. For other groups, individual addresses may be designated as Managers with the ability to add and remove and grant permissions to other group members.



ENCRYPTED WITH OWNER KEY OWNED BY ARTIST/PUBLISHER
Policy Rules
Set by Publisher
offerings allowed
availability windows
geo restrictions
number of accesses
user restrictions
etc
Smart Contract
Address (on Fabric blockchain)
0xbe35bf9a1ec9fb684a95dc8d1624cd7b26a96043
Key Methods:
Access Request
Transfer Ownership
Commit Version
Content Object
Contract Address
0xbe35bf9a1ec9fb684a95dc8d1624cd7b26a96043
Version Hash
hq__GXJ1xEQhwzRet7tFxLVnFMJneYF4wDBLeePENbSfQ8NzyzH9D3qvgj74VmP9nkECkwKJMnbnwY
“Title”
RIGHTS
CODE
ARTWORK
METADATA
MEDIA
Creates Offerings
Stream
View
Download
METADATA
VIDEO, IMAGE & AUDIO ESSENCE
BITCODE FOR SELF RENDERING
Blockchain ledger engine for transactions on the content objects
Content storage and routing engine

Access and Authorization Control in a Typical Content Streaming Distribution

The following block diagram shows a basic and typical system flow from content ingest, through encoding and encrypted publishing to the Content Fabric, and transmission through to client applications, such as players. The relevant link and API security methods are shown.



Ingest Node
- transcodes and conditions the source file(s) and creates the playable mezzanine.
- distributes the encrypted content parts to nodes that are in charge of storing them

Content Operator
- contacts ingest node (HTTPS REST API)
- either (a) uploads the source file(s) using client side encryption (content is encrypted by the client) or (b) provides an AWS S3 signed URL

Player
- makes a manifest request to an EGRESS NODE (HLS or DASH over HTTPS)
- makes a decryption key request to the designated KMS nodes and a DRM license request for Fairplay or Widevine

Egress Node
- contacts other nodes to retrieve encrypted content parts
- makes a proxy re-encryption key request to the designated KMS nodes
- just-in-time encodes and packages HLS or DASH segments encrypted using the required DRM technology

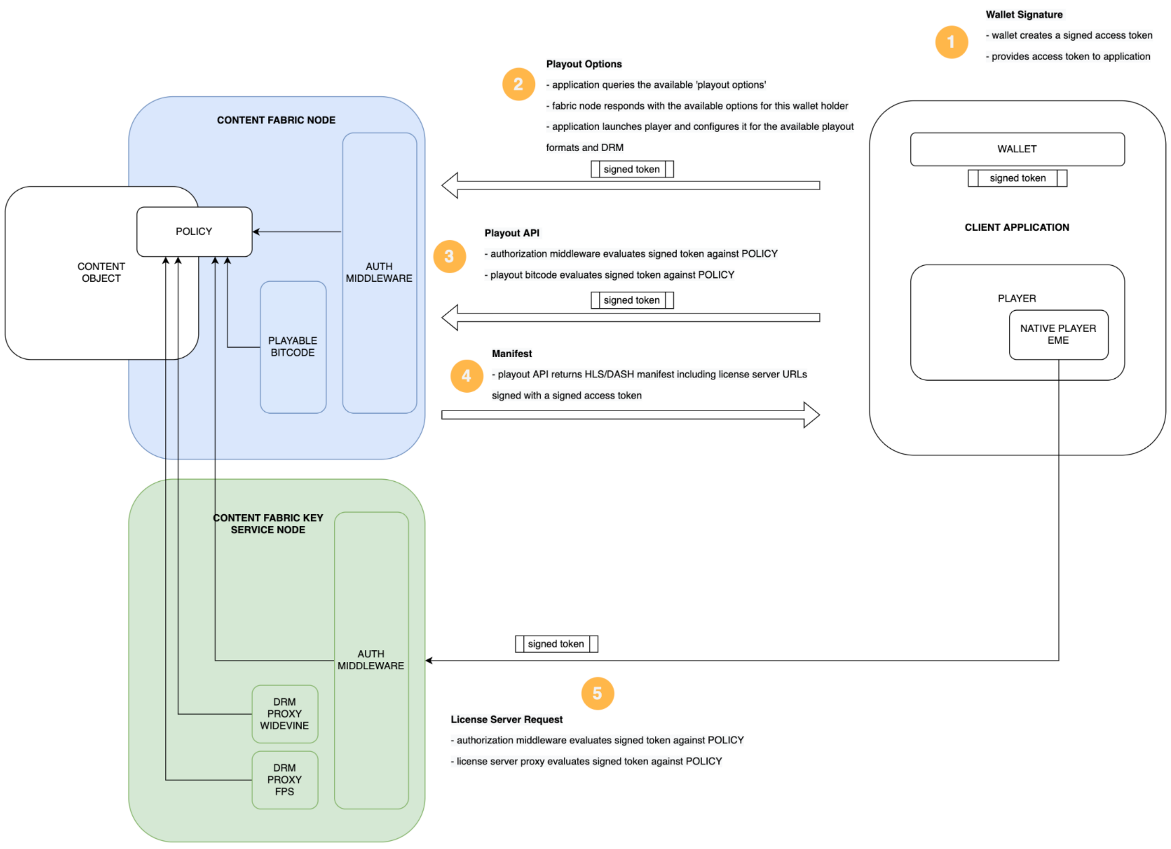
For context on the various entities involved, the Content Fabric platform is developed and maintained by Eluvio, Inc. The Eluvio Live and Media Wallet applications are open source and developed and maintained by Eluvio, Inc. The elv-player-js application is optional; content publishers may implement their own players and applications, but many choose to use the elv-player-js as an embeddable player. Its codebase is open source and developed and maintained by Eluvio, Inc, and uses open source Java Script native players (hls.js, dash reference player).

In order to access and stream content, the client application implements the following API flow:

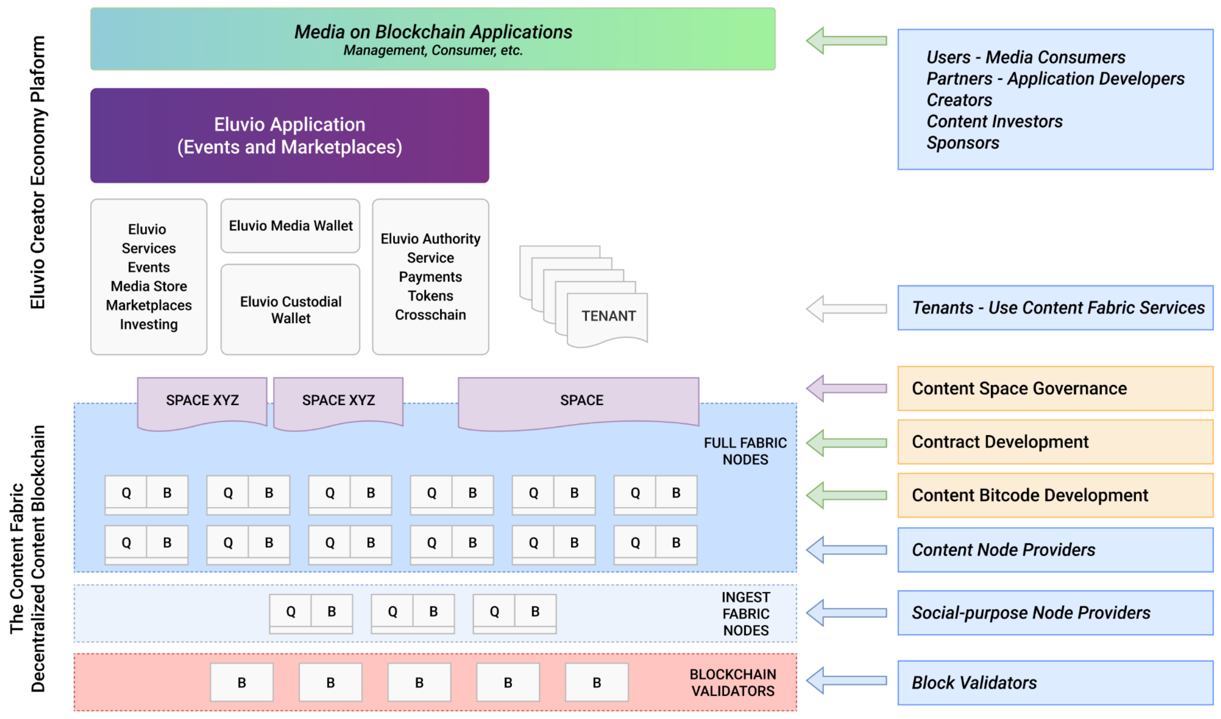
1) Application requests wallet to create a ‘client-signed access token’
2) Application calls Content Fabric API ‘playout options’ - this will return the available playout options and formats (including DRM) for this wallet holder
3) Application initializes the player using the appropriate configuration for the available playout format and DRM

4) Player requests HLS/DASH manifest (and includes the client-signed token with the request)
    a) Content Fabric node has two security layers - authorization middleware evaluates the content policy before invoking the bitcode layer
    b) Playout bitcode layer evaluates the content policy and uses the resulting authorization context for each content request
5) The manifest returns the license server URLs (FPS / Widevine)
6) The player engages the browser EME (Encrypted Media Extensions) or device native player (Android or iOS) - the EME or native player engage the device CDM which makes a requests to the license server, including the 'client-signed access token'
    a) The Content Fabric KMS node has two security layers - the authorization middleware evaluates the content object policy before invoking the DRM module
    b) Each DRM module evaluates the content policy and uses the resulting authorization context for accessing key and DRM server resources.

Application and Player API and Security Flow


Wallet Signature
1
- wallet creates a signed access token
- provides access token to application
Playout Options
2
- application queries the available 'playout options'
- fabric node responds with the available options for this wallet holder
- application launches player and configures it for the available playout formats and DRM
signed token
CONTENT FABRIC NODE
CONTENT OBJECT
POLICY
AUTH MIDDLEWARE
PLAYABLE BITCODE
WALLET
signed token
CLIENT APPLICATION
Playout API
3
- authorization middleware evaluates signed token against POLICY
- playout bitcode evaluates signed token against POLICY
signed token
PLAYER
NATIVE PLAYER EME
Manifest
4
- playout API returns HLS/DASH manifest including license server URLs signed with a signed access token
CONTENT FABRIC KEY SERVICE NODE
AUTH MIDDLEWARE
DRM PROXY WIDEVINE
DRM PROXY FPS
signed token
5
License Server Request
- authorization middleware evaluates signed token against POLICY
- license server proxy evaluates signed token against POLICY

Best Practices for Content and Administrative Security


Eluvio Creator Economy Plaform
Media on Blockchain Applications
Management, Consumer, etc.
Eluvio Application (Events and Marketplaces)
Eluvio Services Events Media Store Marketplaces Investing
Eluvio Media Wallet
Eluvio Custodial Wallet
Eluvio Authority Service Payments Tokens Crosschain
TENANT
Users - Media Consumers
Partners - Application Developers
Creators
Content Investors
Sponsors
Tenants - Use Content Fabric Services
The Content Fabric Decentralized Content Blockchain
SPACE XYZ
SPACE XYZ
SPACE
FULL FABRIC NODES
Q
B
INGEST FABRIC NODES
BLOCKCHAIN VALIDATORS
Content Space Governance
Contract Development
Content Bitcode Development
Content Node Providers
Social-purpose Node Providers
Block Validators

The diagram above shows the overall system architecture for most Media on Blockchain Applications. These work through an Eluvio Application API layer including the Eluvio Authority Service for administrative operations like on-chain asset creation/NFT minting, marketplace administration, payments, etc. and may also utilize the Content Fabric functions directly such as for publishing or content object administration. There are a few key best practices for keeping content and a Tenancy secure. We highlight the most important here.

- **Secure private keys for the Content Owner and user accounts with Editor or Accessor level permissions -** The blockchain private keys or corresponding mnemonics for users that are Owners of content objects, or have Editor or Accessor privileges should be secured in secure offline storage. Corresponding public addresses can be rotated out by simply revoking access of the public address to a corresponding object or membership in a group.

- **Secure a Tenant Admin private key in offline storage, not used -** Tenant Admin addresses have the ability to create and manage groups and operate the business level authority service APIs.
    - The corresponding private keys should be secured such that at least one tenant admin key is not used for day-to-day work and kept in offline storage. This key can be used to revoke the permissions of other addresses if needed.

    - Similarly secure keys for supplementary tenant worker addresses such as the addresses used to mint ("Minter" and "Mint Helper" keys) just as you secure the working Tenant Admin keys.

- **Use least privileges principles in setting the object permissions** - Objects that are especially sensitive should be secured with Editor only permissions and with as few addresses as possible having the Editor only rights. Objects that back digital assets like NFTs with permissioned access should have no more than Publicly Listable permissions. Public permissions should be reserved only for objects for which any of the object metadata is safe to be read by anyone.

- **Use strong DRM for secure content streaming -** For broadest compatibility with legacy security models, the Fabric supports options for content to be streamed using not only strong DRM such as Apple Fairplay and Google Widevine DRM, but also AES128 and Sample AES encryption. For greatest content protection content objects should be configured with the strong DRM options only.

- **API Best Practices** - Create separate addresses and keys for API access and assign these permissions just as you assign permissions to end-user addresses. Utilize the expiration timestamp in client signed and fabric authorization tokens to limit the lifetime of authorization to the minimum period for your application's use case. See for example https://eluv-io.github.io/elv-client-js/ElvClient.html#CreateSignedToken 'duration'

- **Content Ingest -** Master content can be ingested to the Fabric directly from cloud storage in addition to from external file storage. If using cloud storage use the securely signed URL option rather than providing cloud-based private keys. Create separate keys for highly secure content and use Owner-only or limited Editor access permission policies.

For further information on best practices or for any additional questions, please contact your Technical Account Manager or email support@eluv.io .