OGC® DOCUMENT: 21-020R1

External identifier of this OGC® document: http://www.opengis.net/doc/PER/t17-D007

Open Geospatial Consortium

TESTBED-17: DATA CENTRIC SECURITY ER

ENGINEERING REPORT

PUBLISHED

Submission Date: 2021-11-19
Approval Date: 2021-12-17
Publication Date: 2022-01-21
Editor: Aleksandar Balaban, Andreas Matheus

CONTENTS

LIST OF TABLES

LIST OF FIGURES

I ABSTRACT

This OGC Testbed-17 Engineering Report (ER) documents the enhancement of applying Data Centric Security (DCS) to OGC API Features, OGC API Maps (draft), and OGC API Tiles (draft).

As organizations move to the cloud, it is important to incorporate DCS into the design of the new cloud infrastructure, enabling the use of cloud computing, even for sensitive geospatial data sets. The ER documents the applicability of Zero Trust through a Data Centric security approach (DCS) when applied to vector and binary geospatial data sets (Maps, Tiles, GeoPackage containers) and OGC APIs.

The defined architecture extends the typical Zero Trust Domain component by introducing a Key Management System (KMS) to support key registration and the management of access conditions for key retrieval. The prototype implementations (DCS Client, DCS Server and KMS) demonstrate how to request encrypted geospatial data as JSON for encrypted vector data, HTTP Multipart for encrypted map data or GeoPackage with encrypted content; how to obtain decryption key(s) and how to decrypt and display the protected data in a mobile application on Android.

II EXECUTIVE SUMMARY

Data Centric Security (DCS) is an approach to apply security directly to data, independent from security features provided by a network, servers or applications. For Data Centric Security in the geospatial domain, proof-of-concept implementations were developed through work in OGC Testbed-15 and Testbed-16. Initially Extensible Markup Language (XML) based standards to label and protect geospatial feature data in Testbed-15 according to the NATO STANAG 4774 and 4778 specifications, the work expanded into JavaScript Object Notation (JSON) based structures during Testbed-16.

For Testbed-17, the primary goal of the DCS task was to apply Data Centric Security in the context of OGC API Standards that enable the delivery of binary data representations such as images and GeoPackage.

The motivation for applying Data Centric Security is the ability to protect sensitive data independently from controlling the access to systems storing and transferring such data. An example of such systems is increasingly popular cloud-based data storage or content delivery platforms. A fundamental requirement is that the data sets are always protected, until an authorized actor makes use of the data. Additional requirements include the need for representation of the source of the information, as well as an assurance that the information has not been tampered with. When looking at drafting OGC Standards such as an OGC API in a Data Centric Security scenario, standards need to include ways to classify the security requirements around data access. Data Centric Security protected data could be stored locally at the client location in order to be used within the validity period of time.

OGC members can derive business value from applying Data Centric Security in the following areas:

- Similarities between the Data Centric Security approach in the geospatial domain and the commercial/enterprise Digital Rights Management (DRM) architecture improves marketing perspectives: Distributing protected geospatial binary content (satellite imagery, sensitive content structured as a GeoPackge container) is similar to DRM.
- Flexible control over the decryption via a fit-for-purpose Key Management System.
- Interoperability through the support for different encoding standards such as XML and JSON. Work in Testbed-17 demonstrated the interoperability for DCS protected binary geospatial data sets and a way to protect and distribute packaged content.
- Leverage a common Zero Trust Domain architecture whereby a common security context is shared by all components based on Bearer Access Tokens as defined in The OAuth 2.0 Authorization Framework: Bearer Token Usage.
- Use GeoPackage with (individually) encrypted data for OGC API — Features or — Tiles that offer flexible decryption of content as necessary. This is an advantage compared to the standard approach of encrypting entire GeoPackage files, as these need to be decrypted as a whole before content can be extracted. The introduced approach allows for example mobile applications to just decrypt those tiles or features that are to be rendered on the display. This saves memory, increases performance and thereby extends battery lifetime.

The Testbed-17 DCS work envelope is defined by the results from Testbed-16 and the ambition to establish DCS for OGC API responses containing binary data. In particular, the following questions were considered:

- How can an implementation of the draft OGC API — Maps specification be used to return an encrypted image of a map request?
- How can an implementation of the OGC API — Features standard be used to return a GeoPackage container that contains encrypted features?
- How can an implementation of the draft OGC API — Tiles specification be used to return a GeoPackage container that contains encrypted tiles?
- Which metadata fields in the response allow a client application to successfully decrypt the “blob” of data?
- How can one be assured that the decryption of the encrypted responses can only be carried out with trusted applications and legitimate users?
- What are the implications of processing encrypted data regarding storage, processing times for encryption and decryption, as well as CPU burden?

The Testbed-17 findings demonstrate the ability to support Data Centric Security applied to binary geospatial data sets. A mobile client application running on the Android operating system was implemented to request and decrypt protected binary content, such as encrypted images (OGC API — Maps (draft)), encrypted features (OGC API — Features) as well as GeoPackages with encrypted content based on OGC API — Features and OGC API — Tiles (draft).

In support of the experimentation in this Testbed, several Technology Integration Experiments (TIEs)/scenarios were defined:

- The first scenario depicted the exchange of DCS-protected content from an OGC API Maps implementation.
- The second scenario depicted the exchange of DCS-protected content from an OGC API Features implementation.
- The third scenario depicted the exchange of DCS-protected GeoPackages containing encrypted features or map tiles requested via implementations of OGC API — Features or OGC API — Tiles.

This ER introduces GeoPackage encryption extensions for storing Features or Tiles in Annex A.

Future OGC Innovation Program activities should extend and implement access control based on spatio-temporal policies to complement the Testbed-17 Data Centric Security architecture towards a Zero Trust Domain architecture. In particular, further activities could demonstrate how to implement the Data Centric Security architecture (including access control) in a Federated Security environment as outlined in OGC Testbed-16: Federated Security. Another future activity should focus on the standardization of a Key Management System API meeting the specific requirements for key management with Data Centric Security enabled geospatial data.

III KEYWORDS

The following are keywords to be used by search engines and document catalogues.

ogcdoc, OGC document, Data-Centric Security

IV PREFACE

Attention is drawn to the possibility that some of the elements of this document may be the subject of patent rights. The Open Geospatial Consortium shall not be held responsible for identifying any or all such patent rights.

Recipients of this document are requested to submit, with their comments, notification of any relevant patent claims or other intellectual property rights of which they may be aware that might be infringed by any implementation of the standard set forth in this document, and to provide supporting documentation.

V SECURITY CONSIDERATIONS

When applying Data Centric Security, a false assumption might be drawn that **encryption of data** is the compulsory solution to everything. This is not true. Why? It is certainly possible to tamper encrypted data so that it still decrypts, but would produce fraudulent information in the “clear” (after being decrypted). Encryption provides confidentiality, as the encrypted data can only be transformed into the “clear” by applications or users that are in possession of the decryption key (and support for the cipher algorithm in use).

To completely trust encrypted data, assurance about integrity and most likely authenticity must be implemented. Even though the implementation of integrity also relies on encryption, it focuses on encrypting the hash value of the data (either before or after it was encrypted) and not the data itself. Whereas encryption is usually based on a shared secret (as symmetric key), digital signatures require the use of asymmetric keys (a public key shared with all relevant entities and a private key under sole possession of the owning user). Extending integrity to authenticity is quite simple, as it only requires that a certificate is created (by a trusted Certificate Authority) that bundles the public key with the identity information of the entity that owns (and is in single possession of) the associated private key.

Testbeds 15, 16 and 17 have successfully demonstrated how confidentiality can be applied to geospatial data, returned via OGC APIs. The focus was to encrypt responses:

- OGC API Maps for DCS returns a HTTP Multipart response where the second part represents the encrypted binary data of the map image
- OGC API Features for DCS returns a JSON container containing a collection of encrypted features encoded in GeoJSON or a GeoPackage where each feature entry is encrypted
- OGC API Tiles for DCS returns a GeoPackage where each tile data is encrypted

The Testbed 17 result `GeoPackage Encryption Extension` provides — as the name suggests — an approach to ensure confidentiality of the data stored in a GeoPackage. There is **no** guarantee about the integrity of the GeoPakage or even its authenticity. To ensure integrity of a GeoPackage, and thereby making it possible to detect tampering, it would be required to calculate the hash value for the entire GeoPackage and then create a digital signature. This digital signature would then have to be stored with some trusted entity, like a Content Delivery Network (CDN), that allows to retrieve the digital signature of a hash value. A similar process is in place for the distribution of binary software packages: JavaScript download with integrity check or Linux package distribution leverages SHA256 or SHA384 hash value verification.

VI SUBMITTING ORGANIZATIONS

The following organizations submitted this Document to the Open Geospatial Consortium (OGC):

- m-click.aero GmbH
- Secure Dimensions GmbH

VII SUBMITTERS

All questions regarding this document should be directed to the editor or the contributors:

NAME	ORGANIZATION	ROLE
Aleksandar Balaban	m-click.aero GmbH	Editor
Andreas Matheus	Secure Dimensions GmbH	Editor
Adam Parsons	Compusult Ltd.	Contributor

1

SCOPE

1 SCOPE

This OGC Engineering Report (ER) is deliverable D007 of the OGC Testbed-17 activity (Data Centric Security Across OGC APIs) performed as an OGC Innovation Program activity.

This document expands on the results in the area of geospatial Data Centric Security from OGC Testbed-16 by evaluating options to request binary geospatial data formats via implementations of the draft OGC API — Maps, and OGC API — Tiles, and the approved OGC API — Features standard. In particular, this ER discusses and outlines the approaches for extending the OGC GeoPackage Encoding Standard to support encrypted data (features or map tiles) plus adequate metadata for enabling client applications to perform the integrity checking, decryption and visualization. Regarding the use of image data formats, this ER also describes how the draft OGC API Maps could be extended to return encrypted map images as a HTTP Multipart response.

Another addition in the anticipated usage scenarios (and therefore in TIEs) is that encrypted content delivered as a GeoPackage container remains in the client's (local) storage and can be decrypted and displayed multiple times. It is also possible to share locally stored GeoPackage content with other users and applications by adopting access conditions to the (decryption) key via the Key Management System.

When it comes to identifying an architecture to solve a particular problem or challenge, it is wise to not start from scratch if working solutions exist. In that sense, the Testbed-17 architecture differs only slightly from the Testbed-16 architecture.

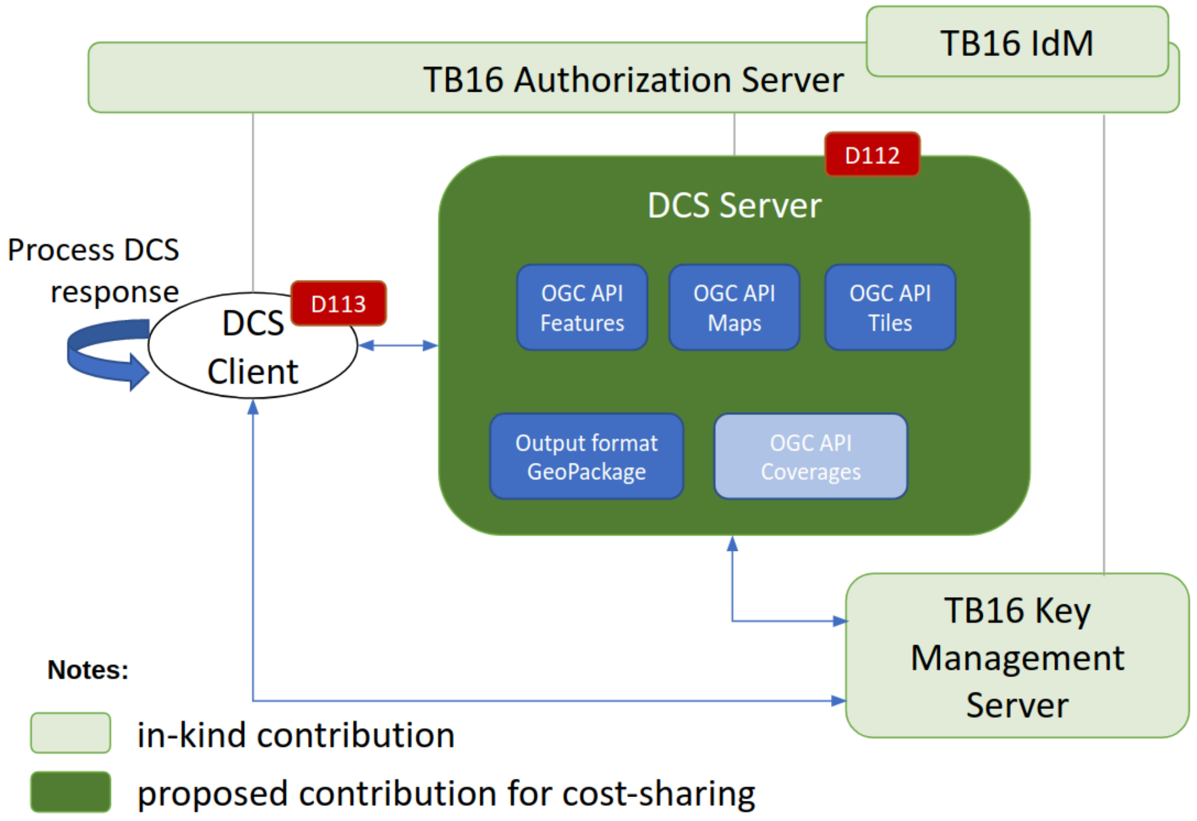


Figure 1 — Testbed-17 Architecture

The Testbed-17 DCS architecture consists of the following components:

Authorization Server (AS)

This component remains unchanged from Testbed-16: It acts as an OAuth2 / OpenID Connect compliant Authorization Server. The DCS Client application requests access tokens from this component after the user has logged in. The DCS Server and Key Management Server use the Authorization Server’s Token Introspection endpoint to validate access token.

Identity Management (IdM)

The Identity Management component supports social login from Facebook, Google and the OGC, as well as approximately 2800 universities. When logging in via the OGC portal IdP, a user’s active OGC projects become available as OpenID Connect claim `ogc-is-member-of` which can be used to undertake access control. This component is tightly integrated with the AS component.

Key Management Server (KMS)

This component ensures that (de/en)cryption keys can be created, registered and obtained from applications and services by presenting an access token. The KMS also supports the modification of access conditions of registered keys by owning users.

DCS Server

This component is implemented as a plugin to GeoServer v2.20 that supports to request encrypted responses for the OGC's Maps, Features and Tiles APIs. In order to execute the plugin, the standard OGC APIs must be extended with additional request parameters. On behalf of a user, for each request, the DCS Server generates and registers (de/en)cryption keys with the KMS.

NOTE 1: The DCS Server's implementation changed from a generic OGC API Proxy to an extension plugin (module) to GeoServer. This tight integration is necessary to accommodate a high performance processing when creating GeoPackage containers with encrypted content.

DCS Client

This component was implemented as an Android mobile application that supports the DCS protocol offered by the DCS Server to request, decrypt and visualize encrypted responses of various encoding, as documented in the TIE section (See Clause 6).

NOTE 2: The DCS Client changed from a QGIS plugin to an Android mobile application.

2

TERMS, DEFINITIONS AND ABBREVIATED TERMS

2 TERMS, DEFINITIONS AND ABBREVIATED TERMS

This document uses the terms defined in OGC Policy Directive 49, which is based on the ISO/IEC Directives, Part 2, Rules for the structure and drafting of International Standards. In particular, the word “shall” (not “must”) is the verb form used to indicate a requirement to be strictly followed to conform to this document and OGC documents do not use the equivalent phrases in the ISO/IEC Directives, Part 2.

This document also uses terms defined in the OGC Standard for Modular specifications (OGC 08-131r3), also known as the ‘ModSpec’. The definitions of terms such as standard, specification, requirement, and conformance test are provided in the ModSpec.

For the purposes of this document, the following additional terms and definitions apply.

2.1. Terms and definitions

2.1.1. Access token

A token that can be provided as part of a service request that grants access to the service being invoked on. This is part of the OpenID Connect and OAuth 2.0 specification.

2.1.2. Assertion

Information about a user. This usually pertains to a JWT authentication response that provided identity metadata about an authenticated user.

2.1.3. Authentication

The process of identifying and validating a user.

2.1.4. **Authorization**

The process of granting access to a user.

2.1.5. **Credentials**

Credentials are pieces of data used to verify the identity of a user.

2.1.6. **Client**

Clients are applications that want to request identity information or an access token so that they can securely invoke DCS secured services.

2.1.7. **DCAP**

Data centric audit and protection, term used by Gartner to describe an approach to information security that combines data security and audit with discovery, classification, policy controls, user and role-based access, and real-time data and user activity monitoring to help automate data security and regulatory compliance.

2.1.8. **Identity token**

A token that provides identity information about the user. Part of the OpenID Connect specification.

2.1.9. **Identity provider**

An identity provider (IdP) is a service that can authenticate a user.

2.1.10. **Federated identity**

Federated identity defines linking and using the identities that a user has across several security domains/realms and their identity providers.

2.1.11. **OGC API**

The OGC API family of standards is a suite of open standards for Web APIs that offer modular building blocks that make it easy for anyone to publish or consume geospatial data on the web.

2.1.12. **Realms**

A realm contains a set of users and their attributes (credentials, roles). A user belongs to a realm. Realms are isolated from one another and only manage and authenticate the users that they control.

2.1.13. **Roles**

Roles identify a type or category of user. Admin, user, manager, and employee are all typical roles that may exist in an organization. Applications often assign access and permissions to specific roles rather than individual users.

2.1.14. **Users**

Users are entities that are able to log into a system. They can have attributes and have specific roles assigned to them.

2.2. Abbreviated terms

AS	Authorization Server
CDN	Content Delivery Network
DCAP	Data centric audit and protection
DCS	Data Centric Security
DEK	Data Encryption Key
DRM	Digital Rights Management
GeoXACML	Geospatial eXtensible Access Control Markup Language
IdM	Identity Management
JOSE	Javascript Object Signing and Encryption
JSON	JavaScript Object Notation
JWT	JSON Web Token
KEK	Key Encryption Key
KID	Key ID
KMS	Key Management Server
LBAC	Label-based access control
OGC	Open Geospatial Consortium
SAML	Security Assertion Markup Language
STANAG	NATO Standardization Agreement
TIE	Technology Integration Experiment
XACML	eXtensible Access Control Markup Language
XML	Extensible Markup Language
ZTA	Zero Trust (Architecture)

3

INTRODUCTION

3 INTRODUCTION

The primary goal of the Testbed-17 Data Centric Security (DCS) task was to apply DCS in the context of OGC API Standards that deliver binary data representations such as images and GeoPackage. The motivation for applying Data Centric Security was the ability to protect sensitive data independently from controlling the access to systems storing and transferring such data.

The following questions were considered in the Testbed-17 Data Centric Security (DCS) task:

- How can an implementation of the draft OGC API — Maps specification be used to return an encrypted image of a map request?
- How can an implementation of the OGC API — Features standard be used to return a GeoPackage container that contains encrypted features?
- How can an implementation of the draft OGC API — Tiles specification be used to return a GeoPackage container that contains encrypted tiles?
- Which metadata fields in the response allow a client application to successfully decrypt the “blob” of data?
- How can one be assured that the decryption of the encrypted responses can only be carried out with trusted applications and legitimate users?
- What are the implications of processing encrypted data regarding storage, processing times for encryption and decryption, as well as CPU burden?

4 RESULTS

4 RESULTS

This chapter summarizes the key findings regarding DCS data formats and implications to standardization when applying DCS to OGC APIs and the GeoPackage encoding. It further lists issues and security implications when using the GeoPackage encoding.

4.1. Binary data formats for DCS

Depending on the OGC API Standard, different packaging and response encoding make sense. In particular the OGC API Features does only support one binary response encoding that was tested within this Testbed: GeoPackage. The capabilities of the OGC API Features limit request features for only one `collection_id`. Therefore, a GeoPackage with encrypted content only contains one DCS related table. However, the GeoPackage Encryption Extension for Features defined in the Annex A does support storing features of different types into different tables. This can be illustrated when requesting a DCS GeoPackage via the WFS 2.0 interface.

NOTE: The use of WFS 2.0 for requesting DCS GeoPackage was not an official part of the work and therefore not included in the TIE.

For the OGC API Features standard, it is possible to apply the DCS data format as an extension to XML and JSON. Both approaches were already implemented and documented during Testbed-16. To summarize, the STANAG 4778 XML encoding is supported where the content is GML encoded and JSON structured responses include JWE representations of GeoJSON encoded Features. Please see OGC Testbed-16: Data Centric Security Engineering Report for more details.

For the OGC API Maps and Tiles draft standards that support the return of binary image data (media type `image/*`), the data could be encrypted by the DCS Server before returning the response to the client. In this Testbed, the equivalent to DRM online streaming, where the application and server negotiate an encryption key to be used for streaming encrypted content, was not implemented. Instead, the more generic approach was implemented, where the encryption key is managed by a KMS.

In this case, the client must obviously receive information about the DEK, aka key metadata with the encrypted binary data. For conveying the encrypted image data and the DEK metadata for the OGC API Maps, the HTTP Multipart response format is used. More details are illustrated in Figure 4.

4.2. DCS GeoPackage Extension

A GeoPackage is the SQLite container and the OGC GeoPackage Encoding Standard governs the rules and requirements of content stored in a GeoPackage container. The GeoPackage standard defines the schema for a GeoPackage, including table definitions, integrity assertions, format limitations, and content constraints. Defined by the Open Geospatial Consortium (OGC)[2] membership with the backing of the US military[3] and published in 2014, GeoPackage has seen wide support from various government, commercial, and open source organizations.

The GeoPackage standard describes a set of conventions for storing the following within a SQLite database:

- Vector features
- Tile matrix sets of imagery and raster maps at various scales
- Attributes (non-spatial data)
- Extensions

This Testbed-17 Engineering Report defines two GeoPackage extensions to store encrypted data:

- GeoPackage Encryption Extension to store Features
- GeoPackage Encryption Extension to store Tiles

See Annex A for more details.

4.2.1. General Containerized Format

Inspired by previous work using XML structures defined in STANAG 4778 and 4774, previous Testbeds already defined a general containerized format based on JSON encoding: OGC Testbed-16: Data Centric Security Engineering Report.

Testbed-17 continued this response format for the OGC API Features via the media type `application/dcs+geo`.

4.3. Implications for OGC API Standardization Development and Implementation

For applying the concepts of Data Centric Security as a building block to OGC (Features, Maps, Tiles, etc.) APIs being able to easily extend the protocol with additional parameters is important.

Of related importance is that implementations of OGC APIs — that have potential wider use beyond just the “traditional” GIS Community — support flexible processing of additional request parameters and support returning “custom” response formats.

When applying DCS relevant parameters to the OGC APIs leveraging a POST request (www-url-form-encoded) as a secure alternative for the HTTP GET must be possible. This is in particular important if Bearer Access Tokens cannot be added to the HTTP Authorization Header or, when a user requests encrypted content but does not want to disclose the “layers” or “feature types” within the HTTP GET request!

IMPORTANT

OGC APIs and their implementations must support the declaration and processing of extensions that define new building block with specific request parameters.

IMPORTANT

OGC APIs and their implementations must support a security building block classed “HTTP Bearer Token” (RFC 6750).

IMPORTANT

OGC APIs and their implementations must support HTTP POST as a secured alternative to HTTP GET. This is in particular important when sending sensitive parameters to the server.

4.4. Discovered Issues with GeoPackage as response format

When applying encryption to data, the processing burden must obviously be considered. When extending APIs (such as the OGC APIs) that are designed to quickly return short to medium size responses (up to some MBs), the potentially large size, as well as long time required to DCS-protect returning GeoPackage container can turn out to become a kind of Denial-Of-Service attack.

Most OGC API Standards define synchronous RESTful web services, which means that the service returns a HTTP 200 OK success status and the response on the same socket pair as the one used to receive the request. This interaction pattern is acceptable, if the requested response format is streaming-ready. A streaming-ready response format is ZIP. The server can open a Zipped output stream and start immediately to push data down to the client. In the case of GeoPackage, this is **not** possible as a GeoPackage container is a binary file format that must first be completely created (on the server’s storage) before the stream towards the client can start.

For large GeoPackages, or for those where the creation takes a long time due to encryption, different issues may arise:

- The user client might disconnect from the socket after the TCP standard timeout. The DCS client was not aware of the requirement to set an extra-long (endless) timeout before making the request. Unfortunately, the service — meanwhile creating the GeoPackage — remains unaware of client disconnection, finishes package generation and eventually starts an attempt to deliver the data to the client. Therefore, the service might generate a large GeoPackage and waste CPU and memory by processing a request for which there is no longer a receiver.
- The creation of the (temporary) GeoPackage on the Server produces `no space left on device` error. In this case, the user would receive an error but most likely retry. This is a dangerous situation, as the server's storage might become spammed with long running GeoPackage processing tasks, which were started on behalf of perhaps disconnected clients.
- Inattentive or malicious users might produce a sequence of requests by click — cancel — repeat. The issue arises that all requests but the last are lost, as the cancel most likely disconnected the TCP socket.

These situations can quickly turn into a denial-of-service (DoS) attack, if resource quotas that limit resource consumption (per service request) aren't put in place.

In this testbed, the participants created GeoPackage sizes of 12GB for the default GeoServer data set (Tiger Roads) when requesting TileMatrixSet 0-21. The processing time on a medium size server took approximately 6 hours to create the container. Without a resource quota, the server resources are quickly not available.

A possible solution for problems mentioned above would be to utilize an asynchronous service design allowing clients to submit a GeoPackage processing request, disconnect and wait for the service to generate the required package (a long time running backend task). In order to prevent multiple service calls caused by malicious or unintentional repeated data requests, clients should further be limited on how many of such tasks they are allowed to submit at a time. Finally, The OGC API — Processes standard may fit right to implement such interaction in an OGC compliant way. OGC API — Processes defines an API that supports both synchronous and asynchronous execution of computing processes (and the retrieval of metadata describing their purpose and functionality).

5

IMPLEMENTATION ASPECTS

5 IMPLEMENTATION ASPECTS

The Testbed-17 DCS architecture extends the Testbed 15 and 16 architectures by supporting encryption of binary responses for the OGC API Features, Maps (draft), and Tiles (draft) standards. This feature is implemented in the DCS Server. The DCS client is a mobile application running on Android.

NOTE: The aim of this section is to provide a detailed Overview about all aspects of design and component development.

5.1. Architecture Overview

The overall architecture supporting the DCS goal is very similar to the one from Testbed-16 (see OGC Testbed-16: Data Centric Security Engineering Report for more details). The following figure illustrates the overall architecture.

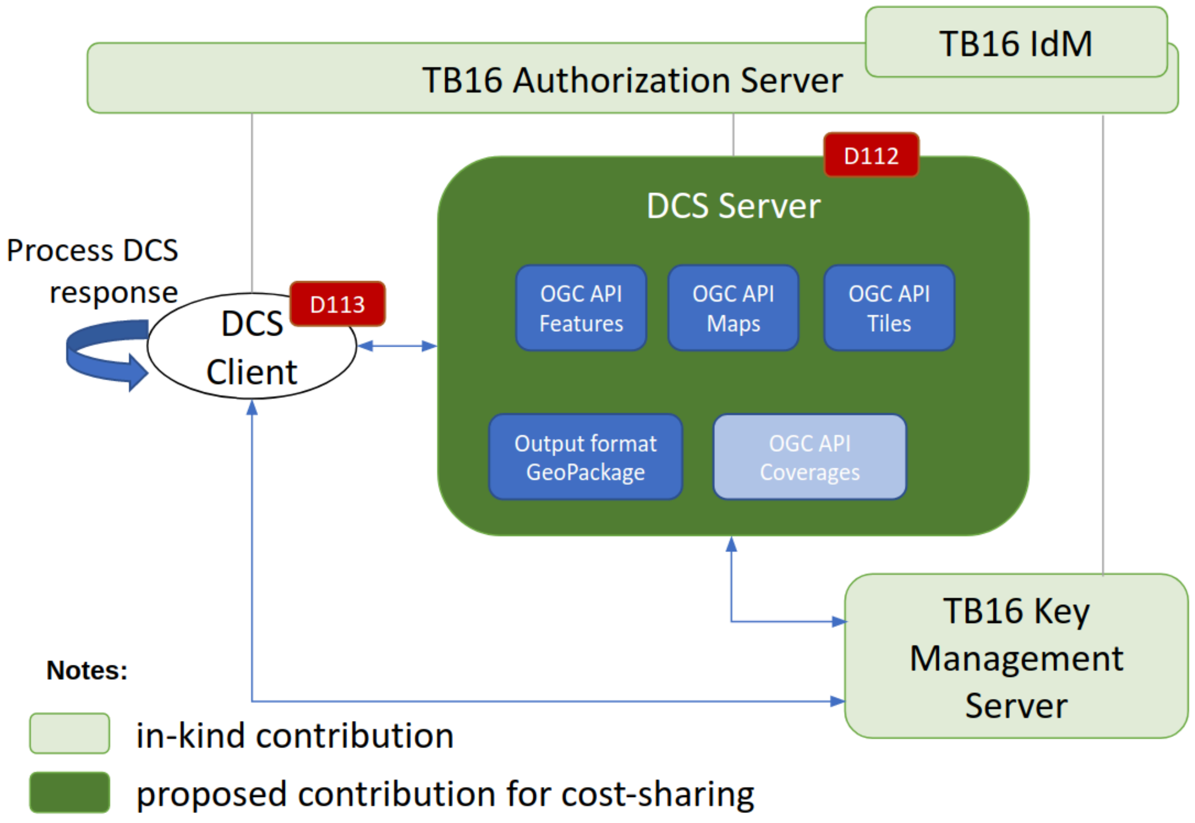


Figure 2 — Testbed-17 Architecture

The Testbed-17 DCS architecture is inspired by the Zero Trust definition as outlined in the OGC Testbed-16: Federated Security Engineering Report: Each request to a DCS enriched OGC API and the KMS requires authentication. This is achieved through the DCS Client that requires an acting user to login first. This results in a Bearer Access Token that the DCS Client requests from the AS. The Bearer Access Token must be used for all requests to the DCS Server and the Key Management Server. Based on the user's identity that is linked to the Access Token, the DCS Server and KMS apply access control based on the user's access rights.

Before making a request to the DCS Server, the user has to provide a PIN or secret to the DCS Client. This PIN is added to the request and used as a proof of ownership to the Data Encryption Key (DEK), generated by the DCS Server. The PIN is required to manage the access conditions for the DEK with the KMS at a later stage. Upon a successful request, the DCS Server returns the DCS specific response. In all cases, the encrypted data in the response is based on a cipher key that was created by the DCS Server and registered with the KMS. Therefore, to decrypt the response, the client first parses the response to find the DEK metadata; in particular the `kid` (key identifier) and the `kurl` the URL to fetch the key from the KMS. Depending on the actual response format, this information is at different places but is always in a standardized format, leveraging the JSON Web Key specification (RFC 7517). Before undertaking the actual decryption of the response, the client interacts with the KMS and fetches the DEK. In case the KMS returns "forbidden", the acting user should verify the access conditions for the key, in particular the expiration time. Amendment to the conditions and in particular to the expiration time can be done via the KMS admin pages. To change the access conditions for a DEK, the user has to provide the PIN.

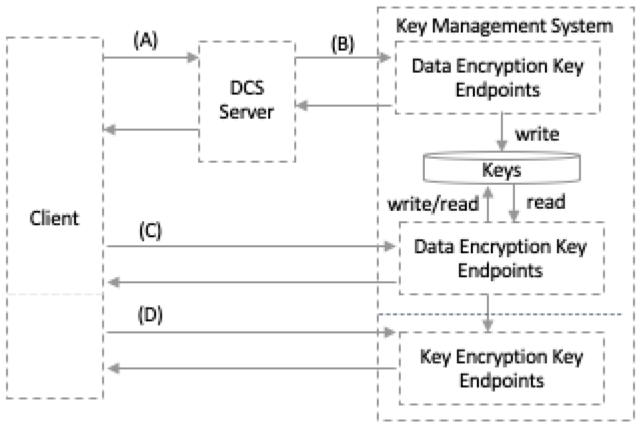

Key Management System
(A)
(B)
DCS Server
Data Encryption Key Endpoints
write
Keys
write/read
read
Client
(C)
Data Encryption Key Endpoints
(D)
Key Encryption Key Endpoints

Figure 3 — Testbed-17 DCS Architecture Interactions Overview

As illustrated in Figure 3, the interactions can be categorized as follows:

Category (A)

Interactions from the client to the server are based on an extension to the parameters defined for the OGC APIs:

- `access_token` parameter (mandatory) contains the value of the Bearer Access Token
- `key_challenge` parameter (mandatory) that contains the (hashed) PIN

- `key_challenge_method` parameter (mandatory) is either `plain` or `S256`. When using `S256`, the value of the `key_challenge` must be hashed according to Proof Key for Code Exchange by OAuth Public Clients, Sect. 4.2.

Responses from the server to the client contain the encrypted data.

Category (B)

The server interacts with the KMS to register the DEK that was generated to encrypt the content for the current request.

The response from the KMS contains the DEK identifier (`kid`).

Category (C)

Interactions between the client and the KMS to fetch a DEK for decrypting the actual response from the server or a GeoPackage that was loaded from (local) storage. The request parameters are

- `access_token` (mandatory) as defined above
- `kid` (mandatory) as part of the path (`/dek/{kid}`)
- `kek_kid` (optional) contains the key identifier for a registered public key that is to be used by the KMS to return the DEK encrypted

The response from the KMS to the client is the JWK encoding of the DEK, including the key's secret. When the request contains the `kek_kid` parameter, the response is a JWE where the payload is the JWK encoding of the key. (The referenced public key is used for key wrapping)

Category (D)

In case that the client wants to request encrypted key material defined by interactions of category (C), the client can use the KMS KEK (Key Encryption Key) API to register a public key in JWK format. The key identifier received in the response can be used as illustrated above in category (C).

5.2. DCS Server

The DCS Server implementation by Secure Dimensions is realized as a GeoServer Plugin, available from Github: GeoServer DCS Plugin. For the OGC Testbed-17 initiative, the implementation demonstrates the ability to apply Data Centric Security to geospatial data, requested via implementations of the OGC's Features, Tiles (draft) and Maps (draft) APIs. In order to support the DCS feature, the OGC API requests are extended with additional query parameters:

- `access_token` (mandatory) as defined above

- `kid` (mandatory) as part of the path (`/dek/{kid}`)
- `kek_kid` (optional) contains the key identifier for a registered public key that is to be used by the KMS to return the DEK encrypted

The request parameter `f` determines the response format based on DCS specific media types, valid in the context of this Testbed. The DCS Server implementation supports different response formats specific to an OGC API. The response format for OGC API Maps is a HTTP Multipart response where the first part contains the key metadata in JSON encoding and the encrypted image is an octet-stream provided as part two. For the OGC API — Features standard, two different response formats are supported: (i) encrypted features in JSON container and (ii) GeoPackage container holding encrypted features. For the OGC API — Tiles draft specification, only GeoPackages holding encrypted tiles are supported.

Table 1 — OGC Testbed-17 DCS Media Types

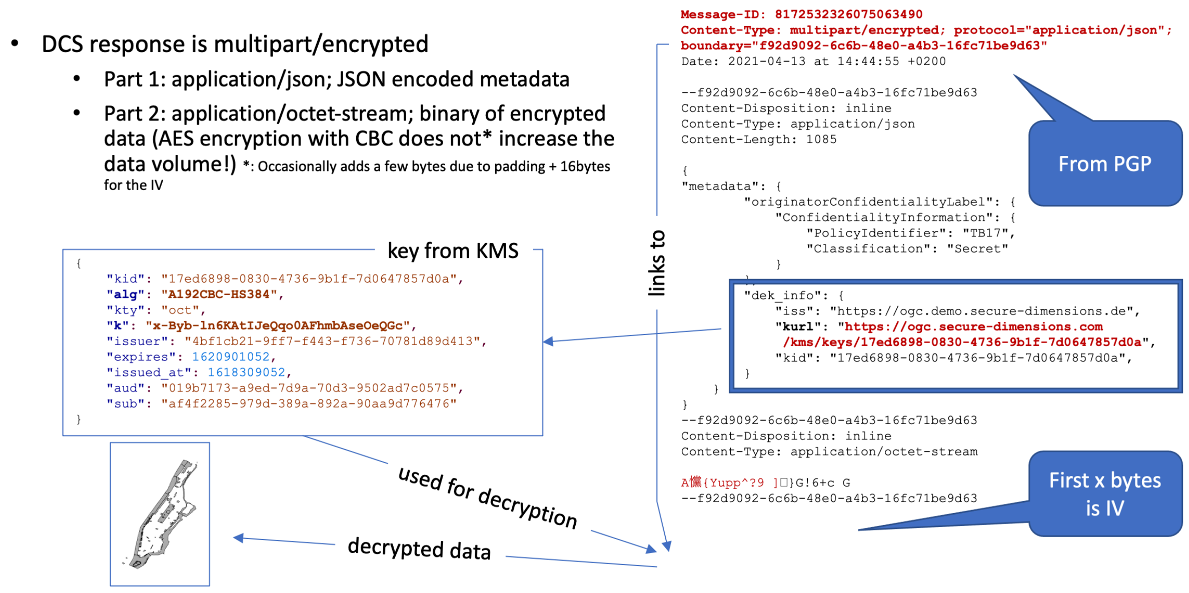
OGC-API	DCS MEDIA TYPE	DESCRIPTION
Features	application/dcs+geo	DCS container structure encrypted features, metadata as JSON
Features	application/dcs+geo;profile=metaSign	DCS container structure encrypted features, metadata as JWS
Features	application/dcs+geo;profile=metaEncrypt	DCS container structure encrypted features, metadata as JWE
Features	application/gpkg+dcs	GeoPackage with encrypted features
Tiles	application/gpkg+dcs	GeoPackage with encrypted tiled coverages
Maps	application/dcs+{png,jpeg,...}	Multipart response with encrypted map tiles

The actual GeoPackage extensions for encrypted Tiles and Features are described in section “GeoPackage Data Centric Security Extensions” (Annex A).

5.2.1. OGC API Maps Response Format

The OGC API Maps implementation returns an image in the format requested. Typical formats are PNG or JPEG. The DCS response format can be requested via the `f` parameter using the DCS specific media type `application/dcs+{<image format>}.

The structure of the response as a HTTP Multipart is illustrated in the following figure.



- DCS response is multipart/encrypted
  - Part 1: application/json; JSON encoded metadata
  - Part 2: application/octet-stream; binary of encrypted data (AES encryption with CBC does not* increase the data volume!) *: Occasionally adds a few bytes due to padding + 16bytes for the IV

```
{
    "kid": "17ed6898-0830-4736-9b1f-7d0647857d0a",
    "alg": "A192CBC-HS384",
    "kty": "oct",
    "k": "x-Byb-ln6KAtIJeQqo0AFhmbAseOeQGc",
    "issuer": "4bf1cb21-9ff7-f443-f736-70781d89d413",
    "expires": 1620901052,
    "issued_at": 1618309052,
    "aud": "019b7173-a9ed-7d9a-70d3-9502ad7c0575",
    "sub": "af4f2285-979d-389a-892a-90aa9d776476"
}
```

```
Message-ID: 8172532326075063490
Content-Type: multipart/encrypted; protocol="application/json";
boundary="f92d9092-6c6b-48e0-a4b3-16fc71be9d63"
Date: 2021-04-13 at 14:44:55 +0200

--f92d9092-6c6b-48e0-a4b3-16fc71be9d63
Content-Disposition: inline
Content-Type: application/json
Content-Length: 1085

{
"metadata": {
        "originatorConfidentialityLabel": {
            "ConfidentialityInformation": {
                "PolicyIdentifier": "TB17",
                "Classification": "Secret"
            }
        },
        "dek_info": {
            "iss": "https://ogc.demo.secure-dimensions.de",
            "kurl": "https://ogc.secure-dimensions.com
             /kms/keys/17ed6898-0830-4736-9b1f-7d0647857d0a",
            "kid": "17ed6898-0830-4736-9b1f-7d0647857d0a",
        }
    }
}
--f92d9092-6c6b-48e0-a4b3-16fc71be9d63
Content-Disposition: inline
Content-Type: application/octet-stream

A慵{Yupp^?9 ]□}G!6+c G
--f92d9092-6c6b-48e0-a4b3-16fc71be9d63
```

Figure 4 — DCS Testbed-17 OGC API Maps Interactions Overview

As illustrated in Figure 4, the response uses content type `multipart/encrypted; protocol=application/json`. This is a standard way to 'alert' the client about the encodings of the different parts. The response header further contains the `boundary` identifier of the multi part message (`f92d9092-6c6b-48e0-a4b3-16fc71be9d63` in the example above).

The following is an example first part:

```
{
    "metadata": {
        "originator_confidentiality_label": {
            "confidentiality_information": {
                "policy_identifier": "TB17",
                "classification": "Confidential"
            }
        },
        "data_producer": {
            "origin": "Not NGA",
            "date": "2021-10-01T14:03:08.409Z"
        },
        "data_description": {
            "type": "Feature",
            "properties": {
                "name": "tiger:tiger_roads",
                "content_type": "application/geo+json"
            },
            "geometry": {
                "type": "Polygon",
                "coordinates": [
                    [
                        [
                            -74.02722,
                            40.684221
                        ],
                        [
                            -73.907005,
                            40.684221
                        ],
                        [
                            -73.907005,
                            40.878178
                        ],
                        [
                            -74.02722,
                            40.878178
                        ],
                        [
                            -74.02722,
                            40.684221
                        ]
                    ]
                ]
            }
        }
    },
    "dek_info":
 "eyJqa3UiOiJodHRwczpcL1wvb2djLnNlY3VyZS1kaW1lbnNpb25zLmNvbVwvZGNzXC8ud2VsbC1rbm93blwvan
eyJzdWIiOiI5NWQzODI3ZC1lZmVmLTMwNTItYTJkZi00NmQxYzdjZTc3YTUiLCJhdWQiOiIwMTliNzE3My1hOWVk
T5BeNmsSLPSfZniVziJe_
7tobLBaA7wxP1wqKGliHq37JbLFMzlqLXmAnSDtSqNjQUtNycRXgkGKADbKUkBsht6o7GrI24Ox3h41F_
HGTVCnxIwr1AlQtA0xEK9QqP-CYjOiRITuHlBDZis6AjF-NCxLdXuplZ5OY9R_
Y3uaduoV4Klczy4xBXOFI2iA8qA8O1wCUUcYpaNm8-EalS5Jx0ve_xOT_
wVquC6Q2Gyz9LF0gjGQ01mhEmw6ZAOWLtiW6xphK34gakp1_oJXeqPWFPo2mDtlCi3DXe2yLhTrY_
uA0_7QAAGPOK9gskHotTZ9FJ-UffKPbs-TkPVEHTapQQ"
}
```

Part 1 of the Multipart response

In clear JSON encoding, the client can determine information about the confidentiality labelling, the data producer and the actual data structure. In particular, the boundary of the map images is presented which allows a GIS client to associate the image with the correct location on the rendered display.

For decryption, the client must use the information from the `dek_info` element. This is a JWT in compact decoding. After decoding, the following information becomes available:

```
{
  "jku": "https://ogc.secure-dimensions.com/dcs/.well-known/jwks.json",
  "kid": "Dr. No",
  "alg": "RS256"
}
```

Header of the JWT defining the public key used for signature

```
{
  "sub": "95d3827d-efef-3052-a2df-46d1c7ce77a5",
  "aud": "019b7173-a9ed-7d9a-70d3-9502ad7c0575",
  "kurl": "https://ogc.secure-dimensions.com/kms/keys/15517e95-c764-4602-9e09-
7aaad104d480",
  "kid": "15517e95-c764-4602-9e09-7aaad104d480",
  "iss": "https://ogc.secure-dimensions.com",
  "exp": 1633098986372,
  "alg": "A128GCM",
  "iat": 1633096986372
}
```

Payload of the JWT describing the key's metadata

Important information for the client is the `kid` or the `kurl` which can be used to fetch the actual decryption key from the KMS. The elements `sub` and `aud` can be used to verify that the key is actually designated to the acting user (identified by the `sub`) and the application itself (identified by the `aud`). Additional information when the key was created (`iat`) by whom (`iss`) and expiration (`exp`) can be used to test applicability and fitness of the key.

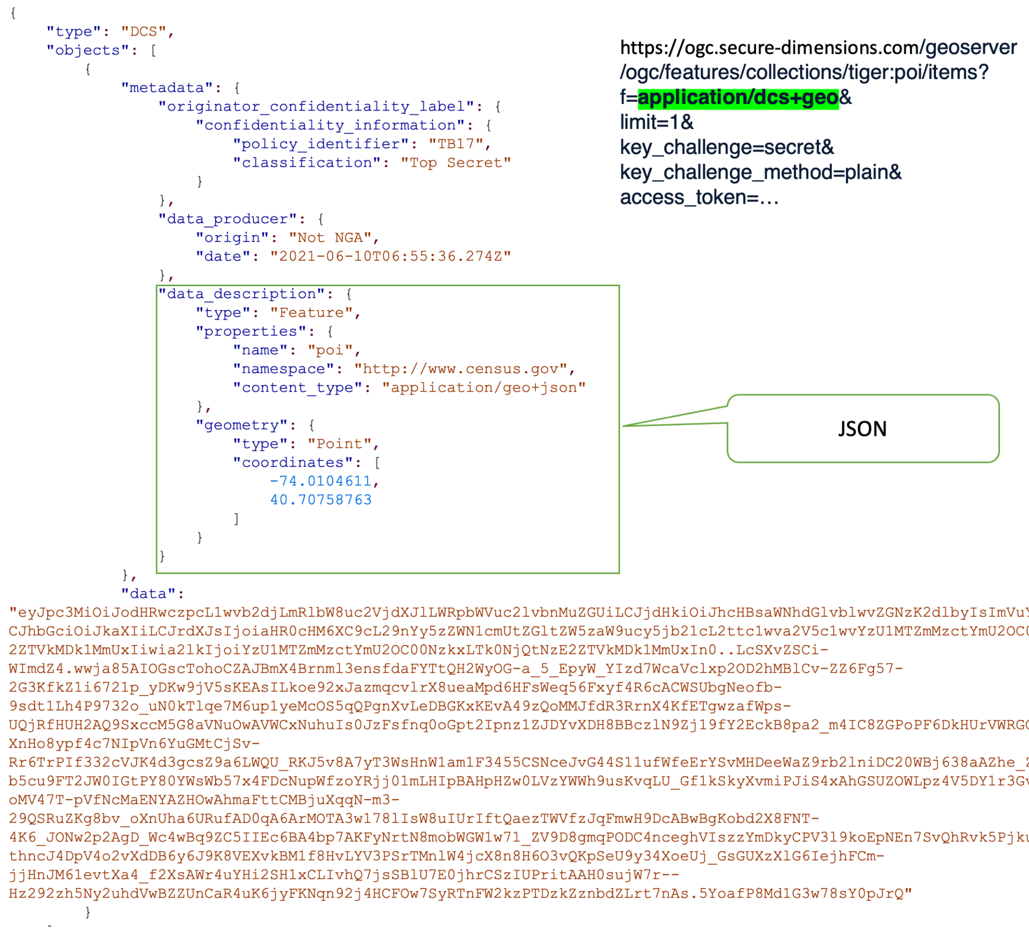
The actual key that corresponds to the `kid = 15517e95-c764-4602-9e09-7aaad104d480` is illustrated in the following code listing:

```
{
  "kid": "15517e95-c764-4602-9e09-7aaad104d480",
  "alg": "A128GCM",
  "kty": "oct",
  "k": "4rE7puBVzdMeXpv1Z3eyHQ",
  "issuer": "4bf1cb21-9ff7-f443-f736-70781d89d413",
  "expires": 1633098786,
  "issued_at": 1633096988,
  "aud": "019b7173-a9ed-7d9a-70d3-9502ad7c0575",
  "sub": "95d3827d-efef-3052-a2df-46d1c7ce77a5"
}
```

Decryption Key in JWK encoding

5.2.2. OGC API — Features JSON Container

The OGC API — Features service implementation returns a JSON encoded container with encrypted features plus relevant metadata. Different format values trigger different encoding for the metadata; the data is always encrypted. All response structures are described in the Data Centric Security ER from Testbed 16.

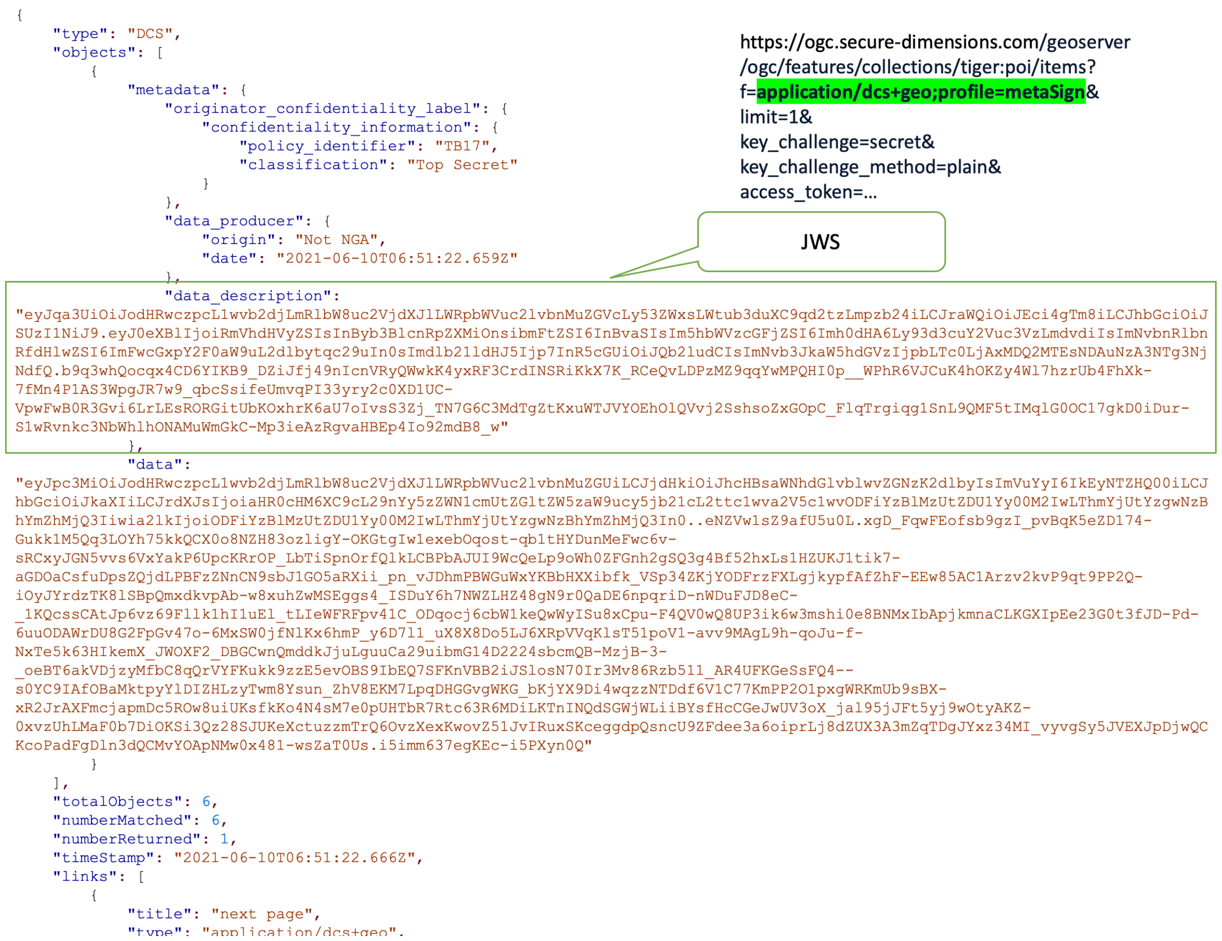


```
{
    "type": "DCS",
    "objects": [
        {
            "metadata": {
                "originator_confidentiality_label": {
                    "confidentiality_information": {
                        "policy_identifier": "TB17",
                        "classification": "Top Secret"
                    }
                },
                "data_producer": {
                    "origin": "Not NGA",
                    "date": "2021-06-10T06:55:36.274Z"
                },
                "data_description": {
                    "type": "Feature",
                    "properties": {
                        "name": "poi",
                        "namespace": "http://www.census.gov",
                        "content_type": "application/geo+json"
                    },
                    "geometry": {
                        "type": "Point",
                        "coordinates": [
                            -74.0104611,
                            40.70758763
                        ]
                    }
                }
            },
            "data":
"eyJpc3MiOiJodHRwczpcL1wvb2djLmRlbW8uc2VjdXJlLWRpbWVuc2lvbnMuZGUiLCJjdHkiOiJhcHBsaWNhdGlvblwvZGNzK2dlbyIsImVuYyI6IkEyNTZHQ00iL
CJhbGciOiJkaXIiLCJrdXJsIjoiaHR0cHM6XC9cL29nYy5zZWN1cmUtZGltZW5zaW9ucy5jb21cL2ttc1wva2V5c1wvYzU1MTZmMzctYmU2OC00NzkxLTk0NjQtNzE
2ZTVkMDk1MmUxIiwia2lkIjoiYzU1MTZmMzctYmU2OC00NzkxLTk0NjQtNzE2ZTVkMDk1MmUxIn0..LcSXvZSCi-
WImdZ4.wwja85AIOGscTohoCZAJBmX4Brnml3ensfdaFYTtQH2WyOG-a_5_EpyW_YIzd7WcaVclxp2OD2hMBlCv-ZZ6Fg57-
2G3KfkZ1i6721p_yDKw9jV5sKEAsILkoe92xJazmqcvlrX8ueaMpd6HFsWeq56Fxyf4R6cACWSUbgNeofb-
9sdt1Lh4P9732o_uN0kTlqe7M6up1yeMcOS5qQPgnXvLeDBGKxKEvA49zQoMMJfdR3RrnX4KfETgwzafWps-
UQjRfHUH2AQ9SxccM5G8aVNuOwAVWCxNuhuIs0JzFsfnq0oGpt2Ipnz1ZJDYvXDH8BBczlN9Zj19fY2EckB8pa2_m4IC8ZGPoPF6DkHUrVWRGGP0H_798pUXmT3_R8
XnHo8ypf4c7NIpVn6YuGMtCjSv-
Rr6TrPIf332cVJK4d3gcsZ9a6LWQU_RKJ5v8A7yT3WsHnW1am1F3455CSNceJvG44S11ufWfeErYSvMHDeeWaZ9rb2lniDC20WBj638aAZhe_ZDZprNADls2sgtaUw
b5cu9FT2JW0IGtPY80YWsWb57x4FDcNupWfzoYRjj01mLHIpBAHpHZw0LVzYWWh9usKvqLU_Gf1kSkyXvmiPJiS4xAhGSUZOWLpz4V5DY1r3GvnZHEBKAH4eNarc8K
oMV47T-pVfNcMaENYAZHOwAhmaFttCMBjuXqqN-m3-
29QSRuZKg8bv_oXnUha6URufAD0qA6ArMOTA3w178lIsW8uIUrIftQaezTWVfzJqFmwH9DcABwBgKobd2X8FNT-
4K6_JONw2p2AgD_Wc4wBq9ZC5IIEc6BA4bp7AKFyNrtN8mobWGW1w71_ZV9D8gmqPODC4nceghVIszzYmDkyCPV3l9koEpNEn7SvQhRvk5PjkuhVFWPQOBYGENMBHc
thncJ4DpV4o2vXdDB6y6J9K8VEXvkBM1f8HvLYV3PSrTMnlW4jcX8n8H6O3vQKpSeU9y34XoeUj_GsGUXzXlG6IejhFCm-
jjHnJM61evtXa4_f2XsAWr4uYHi2SH1xCLIvhQ7jsSBlU7E0jhrCSzIUPritAAH0sujW7r--
Hz292zh5Ny2uhdVwBZZUnCaR4uK6jyFKNqn92j4HCFOw7SyRTnFW2kzPTDzkZznbdZLrt7nAs.5YoafP8Md1G3w78sY0pJrQ"
        }
```

https://ogc.secure-dimensions.com/geoserver
/ogc/features/collections/tiger:poi/items?
f=application/dcs+geo&
limit=1&
key_challenge=secret&
key_challenge_method=plain&
access_token=…

Figure 5 — OGC API Features returning encrypted features collection with metadata as JSON

For a response including digitally signed metadata, the `f` parameter must be set to `application/dcs+geo`.

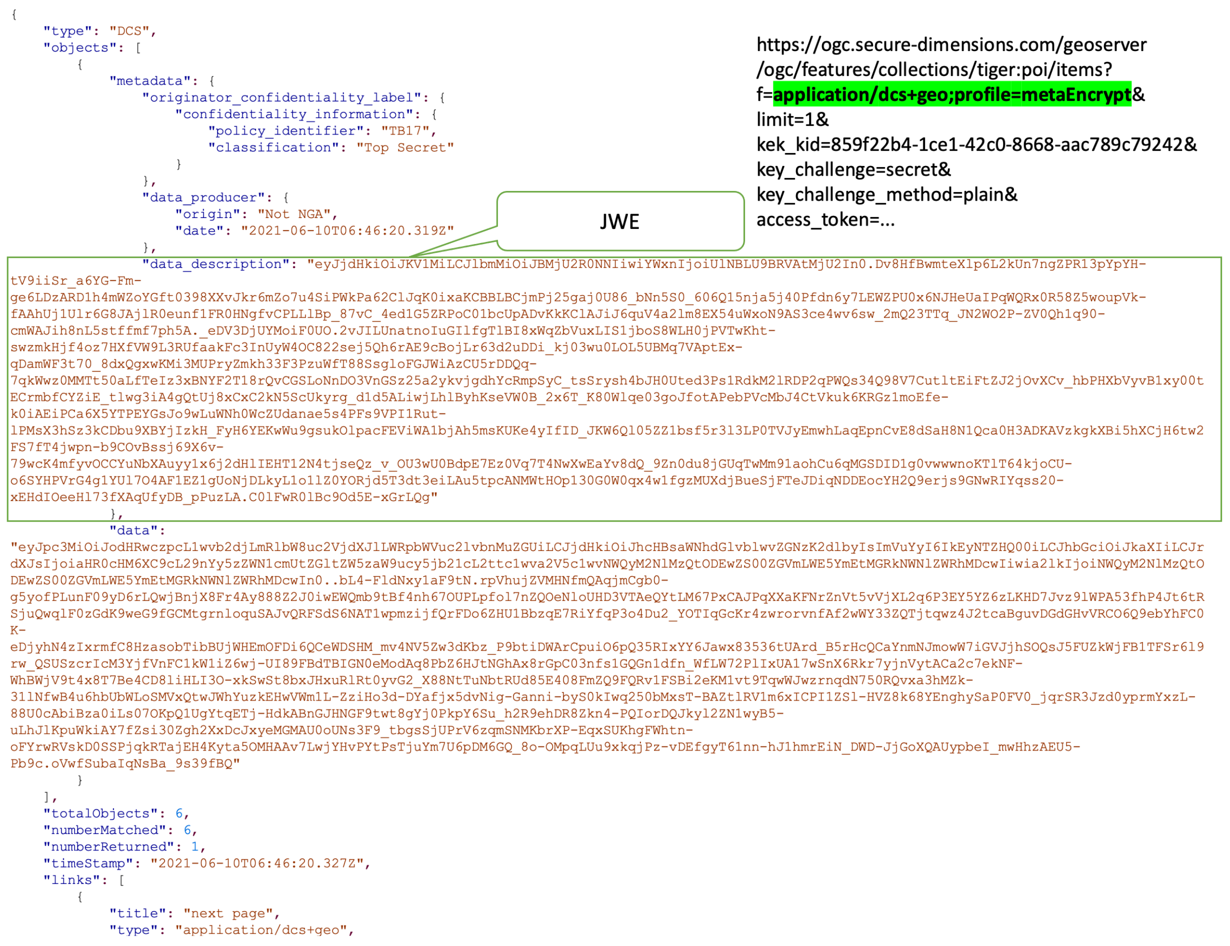


```
https://ogc.secure-dimensions.com/geoserver
/ogc/features/collections/tiger:poi/items?
f=application/dcs+geo;profile=metaSign&
limit=1&
key_challenge=secret&
key_challenge_method=plain&
access_token=...
```

```
{
    "type": "DCS",
    "objects": [
        {
            "metadata": {
                "originator_confidentiality_label": {
                    "confidentiality_information": {
                        "policy_identifier": "TB17",
                        "classification": "Top Secret"
                    }
                },
                "data_producer": {
                    "origin": "Not NGA",
                    "date": "2021-06-10T06:51:22.659Z"
                },
                "data_description":
"eyJqa3UiOiJodHRwczpcL1wvb2djLmRlbW8uc2VjdXJlLWRpbWVuc2lvbnMuZGVcLy53ZWxsLWtub3duXC9qd2tzLmpzb24iLCJraWQiOiJEci4gTm8iLCJhbGciOiJ
SUzI1NiJ9.eyJ0eXBlIjoiRmVhdHVyZSIsInByb3BlcnRpZXMiOnsibmFtZSI6InBvaSIsIm5hbWVzcGFjZSI6Imh0dHA6Ly93d3cuY2Vuc3VzLmdvdiIsImNvbnRlbn
RfdHlwZSI6ImFwcGxpY2F0aW9uL2dlbytqc29uIn0sImdlb21ldHJ5Ijp7InR5cGUiOiJQb2ludCIsImNvb3JkaW5hdGVzIjpbLTc0LjAxMDQ2MTEsNDAuNzA3NTg3Nj
NdfQ.b9q3whQocqx4CD6YIKB9_DZiJfj49nIcnVRyQWwkK4yxRF3CrdINSRiKkX7K_RCeQvLDPzMZ9qqYwMPQHI0p__WPhR6VJCuK4hOKZy4Wl7hzrUb4FhXk-
7fMn4P1AS3WpgJR7w9_qbcSsifeUmvqPI33yry2c0XD1UC-
VpwFwB0R3Gvi6LrLEsRORGitUbKOxhrK6aU7oIvsS3Zj_TN7G6C3MdTgZtKxuWTJVYOEhOlQVvj2SshsoZxGOpC_FlqTrgiqg1SnL9QMF5tIMqlG0OC17gkD0iDur-
S1wRvnkc3NbWhlhONAMuWmGkC-Mp3ieAzRgvaHBEp4Io92mdB8_w"
            },
            "data":
"eyJpc3MiOiJodHRwczpcL1wvb2djLmRlbW8uc2VjdXJlLWRpbWVuc2lvbnMuZGUiLCJjdHkiOiJhcHBsaWNhdGlvblwvZGNzK2dlbyIsImVuYyI6IkEyNTZHQ00iLCJ
hbGciOiJkaXIiLCJrdXJsIjoiaHR0cHM6XC9cL29nYy5zZWN1cmUtZGltZW5zaW9ucy5jb21cL2ttc1wva2V5c1wvODFiYzBlMzUtZDU1Yy00M2IwLThmYjUtYzgwNzB
hYmZhMjQ3Iiwia2lkIjoiODFiYzBlMzUtZDU1Yy00M2IwLThmYjUtYzgwNzBhYmZhMjQ3In0..eNZVw1sZ9afU5u0L.xgD_FqwFEofsb9gzI_pvBqK5eZD174-
Gukk1M5Qq3LOYh75kkQCX0o8NZH83ozligY-OKGtgIw1exebOqost-qb1tHYDunMeFwc6v-
sRCxyJGN5vvs6VxYakP6UpcKRrOP_LbTiSpnOrfQlkLCBPbAJUI9WcQeLp9oWh0ZFGnh2gSQ3g4Bf52hxLs1HZUKJ1tik7-
aGDOaCsfuDpsZQjdLPBFzZNnCN9sbJ1GO5aRXii_pn_vJDhmPBWGuWxYKBbHXXibfk_VSp34ZKjYODFrzFXLgjkypfAfZhF-EEw85AC1Arzv2kvP9qt9PP2Q-
iOyJYrdzTK8lSBpQmxdkvpAb-w8xuhZwMSEggs4_ISDuY6h7NWZLHZ48gN9r0QaDE6npqriD-nWDuFJD8eC-
_1KQcssCAtJp6vz69Fllk1hI1uEl_tLIeWFRFpv41C_ODqocj6cbW1keQwWyISu8xCpu-F4QV0wQ8UP3ik6w3mshi0e8BNMxIbApjkmnaCLKGXIpEe23G0t3fJD-Pd-
6uuODAWrDU8G2FpGv47o-6MxSW0jfNlKx6hmP_y6D7l1_uX8X8Do5LJ6XRpVVqKlsT51poV1-avv9MAgL9h-qoJu-f-
NxTe5k63HIkemX_JWOXF2_DBGCwnQmddkJjuLguuCa29uibmG14D2224sbcmQB-MzjB-3-
_oeBT6akVDjzyMfbC8qQrVYFKukk9zzE5evOBS9IbEQ7SFKnVBB2iJSlosN70Ir3Mv86Rzb511_AR4UFKGeSsFQ4--
s0YC9IAfOBaMktpyYlDIZHLzyTwm8Ysun_ZhV8EKM7LpqDHGGvgWKG_bKjYX9Di4wqzzNTDdf6V1C77KmPP2O1pxgWRKmUb9sBX-
xR2JrAXFmcjapmDc5ROw8uiUKsfkKo4N4sM7e0pUHTbR7Rtc63R6MDiLKTnINQdSGWjWLiiBYsfHcCGeJwUV3oX_jal95jJFt5yj9wOtyAKZ-
0xvzUhLMaF0b7DiOKSi3Qz28SJUKeXctuzzmTrQ6OvzXexKwovZ51JvIRuxSKceggdpQsncU9ZFdee3a6oiprLj8dZUX3A3mZqTDgJYxz34MI_vyvgSy5JVEXJpDjwQC
KcoPadFgDln3dQCMvYOApNMw0x481-wsZaT0Us.i5imm637egKEc-i5PXyn0Q"
        }
    ],
    "totalObjects": 6,
    "numberMatched": 6,
    "numberReturned": 1,
    "timeStamp": "2021-06-10T06:51:22.666Z",
    "links": [
        {
            "title": "next page",
            "type": "application/dcs+geo",
```

Figure 6 — OGC API Features returning encrypted features collection with metadata as JWS

For a response including digitally signed metadata, the `f` parameter must be set to `application/dcs+geo;profile=metaSign`.



```
https://ogc.secure-dimensions.com/geoserver
/ogc/features/collections/tiger:poi/items?
f=application/dcs+geo;profile=metaEncrypt&
limit=1&
kek_kid=859f22b4-1ce1-42c0-8668-aac789c79242&
key_challenge=secret&
key_challenge_method=plain&
access_token=...
```

```
{
    "type": "DCS",
    "objects": [
        {
            "metadata": {
                "originator_confidentiality_label": {
                    "confidentiality_information": {
                        "policy_identifier": "TB17",
                        "classification": "Top Secret"
                    }
                },
                "data_producer": {
                    "origin": "Not NGA",
                    "date": "2021-06-10T06:46:20.319Z"
                },
                "data_description": "eyJjdHkiOiJKV1MiLCJlbmMiOiJBMjU2R0NNIiwiYWxnIjoiUlNBLU9BRVAtMjU2In0.Dv8HfBwmteXlp6L2kUn7ngZPR13pYpYH-
tV9iiSr_a6YG-Fm-
ge6LDzARD1h4mWZoYGft0398XXvJkr6mZo7u4SiPWkPa62ClJqK0ixaKCBBLBCjmPj25gaj0U86_bNn5S0_606Q15nja5j40Pfdn6y7LEWZPU0x6NJHeUaIPqWQRx0R58Z5woupVk-
fAAhUj1Ulr6G8JAjlR0eunf1FR0HNgfvCPLLlBp_87vC_4ed1G5ZRPoC01bcUpADvKkKClAJiJ6quV4a2lm8EX54uWxoN9AS3ce4wv6sw_2mQ23TTq_JN2WO2P-ZV0Qh1q90-
cmWAJih8nL5stffmf7ph5A._eDV3DjUYMoiF0UO.2vJILUnatnoIuGIlfgTlBI8xWqZbVuxLIS1jboS8WLH0jPVTwKht-
swzmkHjf4oz7HXfVW9L3RUfaakFc3InUyW4OC822sej5Qh6rAE9cBojLr63d2uDDi_kj03wu0LOL5UBMq7VAptEx-
qDamWF3t70_8dxQgxwKMi3MUPryZmkh33F3PzuWfT88SsgloFGJWiAzCU5rDDQq-
7qkWwz0MMTt50aLfTeIz3xBNYF2T18rQvCGSLoNnDO3VnGSz25a2ykvjgdhYcRmpSyC_tsSrysh4bJH0Uted3Ps1RdkM2lRDP2qPWQs34Q98V7CutltEiFtZJ2jOvXCv_hbPHXbVyvB1xy00t
ECrmbfCYZiE_tlwg3iA4gQtUj8xCxC2kN5ScUkyrg_d1d5ALiwjLhlByhKseVW0B_2x6T_K80Wlqe03goJfotAPebPVcMbJ4CtVkuk6KRGz1moEfe-
k0iAEiPCa6X5YTPEYGsJo9wLuWNh0WcZUdanae5s4PFs9VPI1Rut-
lPMsX3hSz3kCDbu9XBYjIzkH_FyH6YEKwWu9gsukOlpacFEViWA1bjAh5msKUKe4yIfID_JKW6Q105ZZ1bsf5r313LP0TVJyEmwhLaqEpnCvE8dSaH8N1Qca0H3ADKAVzkgkXBi5hXCjH6tw2
FS7fT4jwpn-b9COvBssj69X6v-
79wcK4mfyvOCCYuNbXAuyy1x6j2dHlIEHT12N4tjseQz_v_OU3wU0BdpE7Ez0Vq7T4NwXwEaYv8dQ_9Zn0du8jGUqTwMm91aohCu6qMGSDID1g0vwwwnoKTlT64kjoCU-
o6SYHPVrG4g1YUl7O4AF1EZ1gUoNjDLkyL1o1lZ0YORjd5T3dt3eiLAu5tpcANMWtHOp130G0W0qx4w1fgzMUXdjBueSjFTeJDiqNDDEocYH2Q9erjs9GNwRIYqss20-
xEHdIOeeHl73fXAqUfyDB_pPuzLA.C0lFwR0lBc9Od5E-xGrLQg"
            },
            "data":
"eyJpc3MiOiJodHRwczpcL1wvb2djLmRlbW8uc2VjdXJlLWRpbWVuc2lvbnMuZGUiLCJjdHkiOiJhcHBsaWNhdGlvblwvZGNzK2dlbyIsImVuYyI6IkEyNTZHQ00iLCJhbGciOiJkaXIiLCJr
dXJsIjoiaHR0cHM6XC9cL29nYy5zZWN1cmUtZGltZW5zaW9ucy5jb21cL2ttc1wva2V5c1wvNWQyM2NlMzQtODEwZS00ZGVmLWE5YmEtMGRkNWNlZWRhMDcwIiwia2lkIjoiNWQyM2NlMzQtO
DEwZS00ZGVmLWE5YmEtMGRkNWNlZWRhMDcwIn0..bL4-FldNxy1aF9tN.rpVhujZVMHNfmQAqjmCgb0-
g5yofPLunF09yD6rLQwjBnjX8Fr4Ay888Z2J0iwEWQmb9tBf4nh67OUPLpfol7nZQOeNloUHD3VTAeQYtLM67PxCAJPqXXaKFNrZnVt5vVjXL2q6P3EY5YZ6zLKHD7Jvz9lWPA53fhP4Jt6tR
SjuQwqlF0zGdK9weG9fGCMtgrnloquSAJvQRFSdS6NAT1wpmzijfQrFDo6ZHU1BbzqE7RiYfqP3o4Du2_YOTIqGcKr4zwrorvnfAf2wWY33ZQTjtqwz4J2tcaBguvDGdGHvVRCO6Q9ebYhFC0
K-
eDjyhN4zIxrmfC8HzasobTibBUjWHEmOFDi6QCeWDSHM_mv4NV5Zw3dKbz_P9btiDWArCpuiO6pQ35RIxYY6Jawx83536tUArd_B5rHcQCaYnmNJmowW7iGVJjhSOQsJ5FUZkWjFB1TFSr6l9
rw_QSUSzcrIcM3YjfVnFC1kW1iZ6wj-UI89FBdTBIGN0eModAq8PbZ6HJtNGhAx8rGpC03nfs1GQGn1dfn_WfLW72PlIxUA17wSnX6Rkr7yjnVytACa2c7ekNF-
WhBWjV9t4x8T7Be4CD8liHLI3O-xkSwSt8bxJHxuRlRt0yvG2_X88NtTuNbtRUd85E408FmZQ9FQRv1FSBi2eKM1vt9TqwWJwzrnqdN750RQvxa3hMZk-
31lNfwB4u6hbUbWLoSMVxQtwJWhYuzkEHwVWm1L-ZziHo3d-DYafjx5dvNig-Ganni-byS0kIwq250bMxsT-BAZtlRV1m6xICPI1ZSl-HVZ8k68YEnghySaP0FV0_jqrSR3Jzd0yprmYxzL-
88U0cAbiBza0iLs07OKpQ1UgYtqETj-HdkABnGJHNGF9twt8gYj0PkpY6Su_h2R9ehDR8Zkn4-PQIorDQJkyl2ZN1wyB5-
uLhJlKpuWkiAY7fZsi30Zgh2XxDcJxyeMGMAU0oUNs3F9_tbgsSjUPrV6zqmSNMKbrXP-EqxSUKhgFWhtn-
oFYrwRVskD0SSPjqkRTajEH4Kyta5OMHAAv7LwjYHvPYtPsTjuYm7U6pDM6GQ_8o-OMpqLUu9xkqjPz-vDEfgyT61nn-hJ1hmrEiN_DWD-JjGoXQAUypbeI_mwHhzAEU5-
Pb9c.oVwfSubaIqNsBa_9s39fBQ"
        }
    ],
    "totalObjects": 6,
    "numberMatched": 6,
    "numberReturned": 1,
    "timeStamp": "2021-06-10T06:46:20.327Z",
    "links": [
        {
            "title": "next page",
            "type": "application/dcs+geo",
```

Figure 7 — OGC API Features returning encrypted features collection with metadata as JWE

For a response including digitally signed metadata, the `f` parameter must be set to `application/dcs+geo;profile=metaEncrypt`. The DCS server returns the encrypted metadata as JWE when the request includes the `kek_kid` which refers to the public key of the user, registered with the KMS. The DCS Server uses that public key to encrypt the cipher key used to encrypt the metadata. This is visible from the JWE header:

```
{
  "cty": "JWS",
  "enc": "A256GCM",
  "alg": "RSA-OAEP-256"
}
```

5.2.3. OGC API — Features GeoPackage Response

The OGC API — Features implementation returns a GeoPackage with encrypted features as described in Annex A.1. This response type can be requested via the `f` parameter set to `application/gpkg+dcs`.

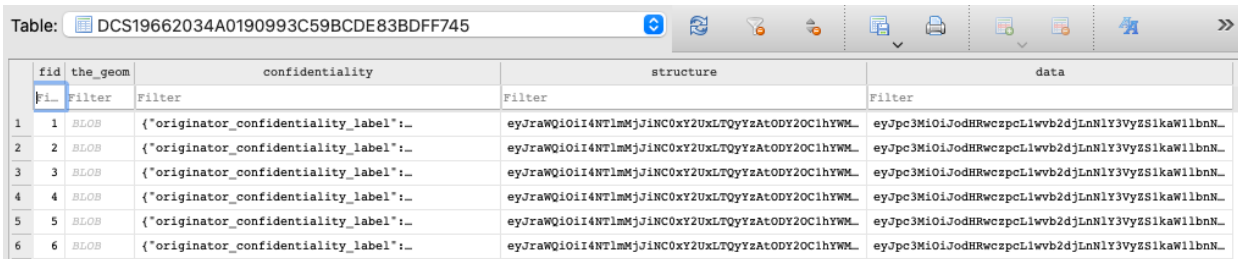


Table: DCS19662034A0190993C59BCDE83BDFF745

	fid	the_geom	confidentiality	structure	data
	Fi…	Filter	Filter	Filter	Filter
1	1	BLOB	{"originator_confidentiality_label":…	eyJraWQiOiI4NTlmMjJiNC0xY2UxLTQyYzAtODY2OC1hYWM…	eyJpc3MiOiJodHRwczpcL1wvb2djLnNlY3VyZS1kaW1lbnN…
2	2	BLOB	{"originator_confidentiality_label":…	eyJraWQiOiI4NTlmMjJiNC0xY2UxLTQyYzAtODY2OC1hYWM…	eyJpc3MiOiJodHRwczpcL1wvb2djLnNlY3VyZS1kaW1lbnN…
3	3	BLOB	{"originator_confidentiality_label":…	eyJraWQiOiI4NTlmMjJiNC0xY2UxLTQyYzAtODY2OC1hYWM…	eyJpc3MiOiJodHRwczpcL1wvb2djLnNlY3VyZS1kaW1lbnN…
4	4	BLOB	{"originator_confidentiality_label":…	eyJraWQiOiI4NTlmMjJiNC0xY2UxLTQyYzAtODY2OC1hYWM…	eyJpc3MiOiJodHRwczpcL1wvb2djLnNlY3VyZS1kaW1lbnN…
5	5	BLOB	{"originator_confidentiality_label":…	eyJraWQiOiI4NTlmMjJiNC0xY2UxLTQyYzAtODY2OC1hYWM…	eyJpc3MiOiJodHRwczpcL1wvb2djLnNlY3VyZS1kaW1lbnN…
6	6	BLOB	{"originator_confidentiality_label":…	eyJraWQiOiI4NTlmMjJiNC0xY2UxLTQyYzAtODY2OC1hYWM…	eyJpc3MiOiJodHRwczpcL1wvb2djLnNlY3VyZS1kaW1lbnN…

Figure 8 — OGC API Features returning GeoPackage with encrypted features plus metadata

5.2.4. OGC API — Tiles GeoPackage Response

The OGC API — Tiles implementation returns a GeoPackage with encrypted tiles as described in Annex A.2. This response type can be requested via the `f` parameter set to `application/gpkg+dcs`.

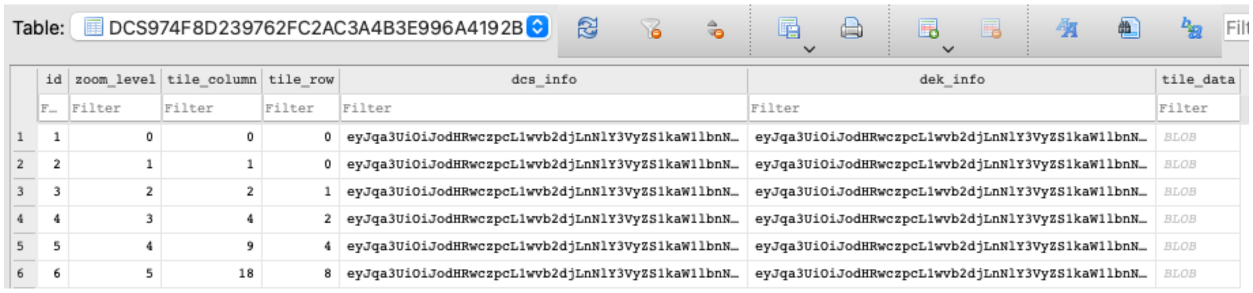


Table: DCS974F8D239762FC2AC3A4B3E996A4192B

	id	zoom_level	tile_column	tile_row	dcs_info	dek_info	tile_data
	F…	Filter	Filter	Filter	Filter	Filter	Filter
1	1	0	0	0	eyJqa3UiOiJodHRwczpcL1wvb2djLnNlY3VyZS1kaW1lbnN…	eyJqa3UiOiJodHRwczpcL1wvb2djLnNlY3VyZS1kaW1lbnN…	BLOB
2	2	1	1	0	eyJqa3UiOiJodHRwczpcL1wvb2djLnNlY3VyZS1kaW1lbnN…	eyJqa3UiOiJodHRwczpcL1wvb2djLnNlY3VyZS1kaW1lbnN…	BLOB
3	3	2	2	1	eyJqa3UiOiJodHRwczpcL1wvb2djLnNlY3VyZS1kaW1lbnN…	eyJqa3UiOiJodHRwczpcL1wvb2djLnNlY3VyZS1kaW1lbnN…	BLOB
4	4	3	4	2	eyJqa3UiOiJodHRwczpcL1wvb2djLnNlY3VyZS1kaW1lbnN…	eyJqa3UiOiJodHRwczpcL1wvb2djLnNlY3VyZS1kaW1lbnN…	BLOB
5	5	4	9	4	eyJqa3UiOiJodHRwczpcL1wvb2djLnNlY3VyZS1kaW1lbnN…	eyJqa3UiOiJodHRwczpcL1wvb2djLnNlY3VyZS1kaW1lbnN…	BLOB
6	6	5	18	8	eyJqa3UiOiJodHRwczpcL1wvb2djLnNlY3VyZS1kaW1lbnN…	eyJqa3UiOiJodHRwczpcL1wvb2djLnNlY3VyZS1kaW1lbnN…	BLOB

Figure 9 — OGC API Tiles returning GeoPackage with encrypted tiles plus metadata

5.3. DCS Client

The DCS Client implementation for the OGC Testbed 17 demonstrates the ability to request and ingest encrypted geospatial content using OGC APIs.

The actual Compusult implementation is realized using the Compusult GO Mobile Android application, which provides a standards-based geospatial and mapping solution for data discovery and visualization.

- Supports Android 5.0 (API Level 21) and above
- Supports many approved and draft OGC API standards (Maps, Tiles, Features, Styles, Images)
- Supports GeoPackage 1.3.0 and below
- Supports OpenID based authentication
- Supports The Key Management System (KMS) operations

- Supports decoding, verifying and decrypting JOSE and JWT content using Nimbus JOSE + JWT and Bouncy Castle
- Supports decoding and decrypting multi-part binary content using JCA and apache-mim4j

5.3.1. Client Registration

The DCS Client must first register its application with the AUTHENIX service to access the OGC APIs and KMS services. A global private/public key pair is created on the first load of the application and stored in the database for future client requests and decryption.

The client performs a registration request https://www.authenix.eu/oauth/register with the request body below.

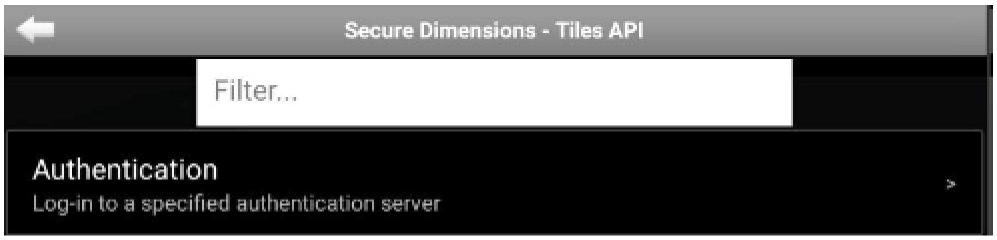
```
{
  "redirect_uris": [
    "http:\/\/www.gomobile_dcs.com\/oauth2redirect"
  ],
  "grant_types": [
    "authorization_code",
    "refresh_token"
  ],
  "response_types": [
    "code",
    "id_token",
    "code id_token"
  ],
  "client_name": "Compusult DCS Application - Mobile",
  "logo_uri": "https:\/\/www.compusult.com\/image\/layout_set_logo?img_id=
12133&t=1615550967456",
  "scope": "openid profile offline_access",
  "contacts": [
    "aparsons@compusult.net"
  ],
  "tos_uri": "http:\/\/127.0.0.1:4711\/TermsOfUse.php",
  "policy_uri": "http:\/\/127.0.0.1:4711\/PrivacyStatement.php",
  "software_id": "4ef5dcfd-22a7-4947-a636-4a46a1cf8d43",
  "software_version": "1.0",
  "post_logout_redirect_uris": [
    "https:\/\/www.example.com\/oauth2logout"
  ]
}
```

The server responds with a registration response including the client_id required for authorization. See below for an example of the server response.

```
{
  "client_id": "8d90bc42-4401-5f2a-9054-cb407a876ad8",
  "client_id_issued_at": 1629899374,
  "client_name": "Compusult DCS Application - Mobile",
  "client_secret": "...",
  "client_secret_expires_at": 1629928174,
  "contacts": [
    "aparsons@compusult.net"
  ],
  "grant_types": [
    "authorization_code",
    "refresh_token"
  ],
  "jwks": null,
  "post_logout_redirect_uris": [
    "https:\/\/www.example.com\/oauth2logout"
  ],
  "redirect_uris": [
    "http:\/\/www.gomobile_dcs.com\/oauth2redirect"
  ],
  "response_types": [
    "code",
    "id_token",
    "code id_token"
  ],
  "scope": "openid profile offline_access",
  "token_endpoint_auth_method": "client_secret_basic"
}
```

5.3.2. Client Authorization

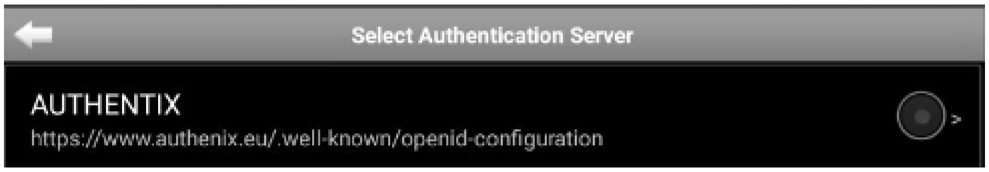
The DCS client will require the user to authorize against any services that are known to contain Data Centric Security content. For OGC APIs compatible services this is determined by inspecting the format types available for request. For GeoPackages this is determined by checking the registered extensions. If a service/geopackage is determined to require authentication, a user is required to login.



Authentication
Log-in to a specified authentication server

Figure 10 — DCS Client — Authentication

Before authenticating the user is required to select the server they wish to authenticate against. The OpenID configuration is parsed and used to determine how to construct the appropriate authorization and token requests.



AUTHENTIX
https://www.authenix.eu/.well-known/openid-configuration

Figure 11 — DCS Client — Server Selection

After selecting the server, the user provides an authorization request including the registered client_id as well as the user specified code_challenge. An example authorization request will take the form of :

https://www.authenix.eu/oauth/authorize?scope=openid+profile+offline_access&response_type=code+id_token&redirect_uri=http%3A%2F%2Fwww.gomobile_

dcs.com%2Foauth2redirect&state=xyz&code_challenge_method=S256&nonce=
123&client_id=8d90bc42-4401-5f2a-9054-cb407a876ad8&code_challenge=
14L0XjF8uRH5IPrEm8RfiPlZilPTFKwxvZqtD9NLxs0

The Client is then asked to select a provider, and provide credentials to authorize the user.

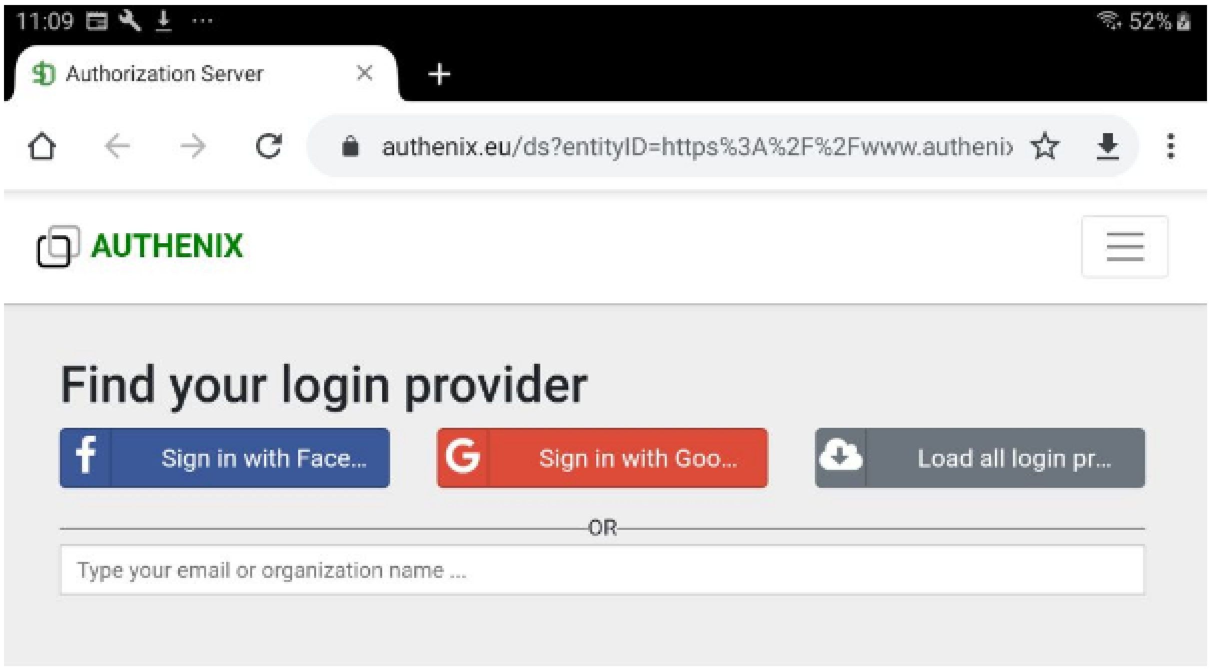

11:09
Authorization Server
authenix.eu/ds?entityID=https%3A%2F%2Fwww.authenix
AUTHENIX
Find your login provider
Sign in with Face...
Sign in with Goo...
Load all login pr...
OR
Type your email or organization name ...

Figure 12 — DCS Client — Provider Selection

After providing the credentials an authorization response is returned containing the authorization code.

http://www.gomobile_dcs.com/oauth2redirect?code=
bdc1634ddb2dd598327da3537064781948fd4ef0&state=xyz&id_token=
eyJ0eXAiOiJKV1QiLCJhbGciOiJSUzI1NiIsImtpZCI6IkFTUHVibGljS2V5In0.
eyJpc3MiOiJodHRwczpcL1wvd3d3LmF1dGhlbml4LmV1Iiwic3ViIjoiYzBmYjJlZWEtMTg1Ni0zNjczLTgwZmM
gV44NNYQmdTI9FU6XwewecwzqfTbA3Pud_oRzFvI9cxnCyzQ5Kx5K7-
utnJ5AdmsCzhxkuWyjz3nT9p6PfkywL9akdst58qhSjfNFAinnhIHfnHl_
GUvm3vBNXhTxR7tJxzhrm5vycdGnHx2bLej_Awk-wRLncGyWGBn_
kRJdePncYO78kBK3A4becIMGXkX8VDTO2Xx3vsrmmEx7NIam9RIN2s4P_
nGpgtYfGw5dX1qv2nFP6D2d9JbgoCsqqcfLCs0arktRZepSE4pAwbmzhKNd1rRCnIscNZsw3nOmNrIbXGkC
ZfROP0zLRWq4JSrORX6uCFxfzyBhlPw

The authorization redirect URL is handled by the app, and prompts the user to allow the application to ingest the response. Note the user is asked to select the appropriate app one time, and all subsequent requests afterwards are automatically redirected.

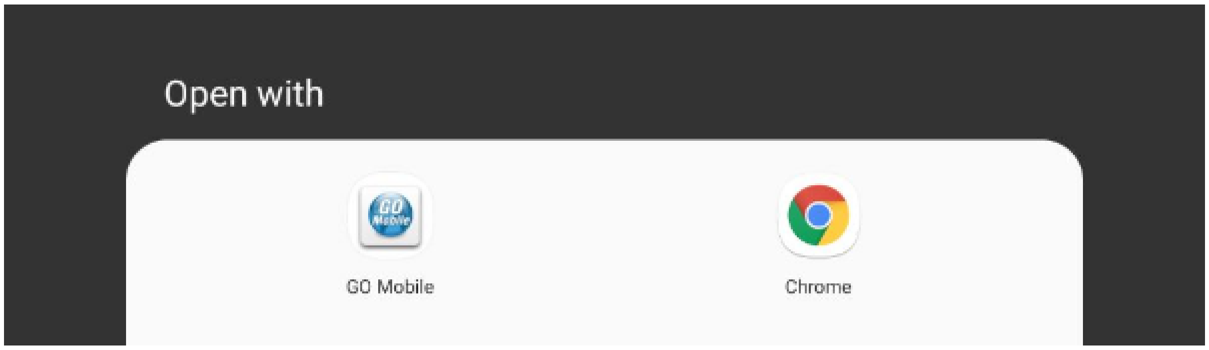


Figure 13 — DCS Client — Redirect Selection

After selecting the app to handle redirection, the application retrieves the authorization code and uses this code to retrieve a valid token for use for the following url : https://www.authenix.eu/oauth/token. Basic Auth request headers are added to the request using the client_secret supplied by the user.

```
code=bdc1634ddb2dd598327da3537064781948fd4ef0&grant_type=authorization_
code&redirect_uri=http%3A%2F%2Fwww.gomobile_dcs.com%2Foauth2redirect&code_
verifier=Hlp9DyvxEm0XPxnsb3Mjuut77nI7oHwO57nEuaejxsMwgdKBqeKV6BYdq-lEd2tk_
htpdj67u2otTShpws20sw
```

The server responds with a valid token response. The response contains the access_token required to be used with the KMS and OGC APIs.

```
{
  "access_token": "741e31abc7c6ec5444036b94fca79796b0925e09",
  "expires_in": 1800,
  "token_type": "bearer",
  "scope": "openid profile offline_access",
  "refresh_token": "c0ea4ca04432f47af977374a94dc8dc6e8b41f31"
}
```

If the request was successful, the client is then re-directed and updates the Authorization button to indicate authorization has been successful and that the client can now be used to request Data Centric Security content.

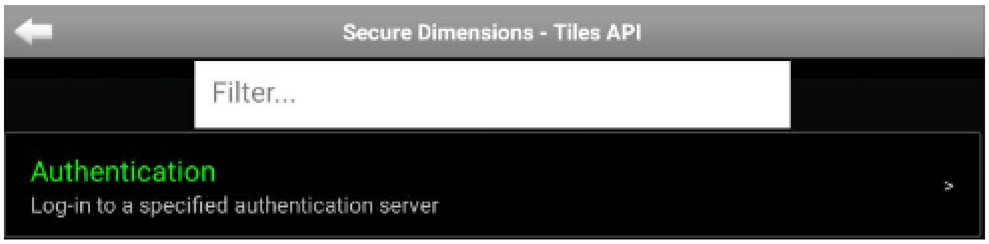

Secure Dimensions - Tiles API
Filter...
Authentication
Log-in to a specified authentication server
>

Figure 14 — DCS Client — Authentication Success

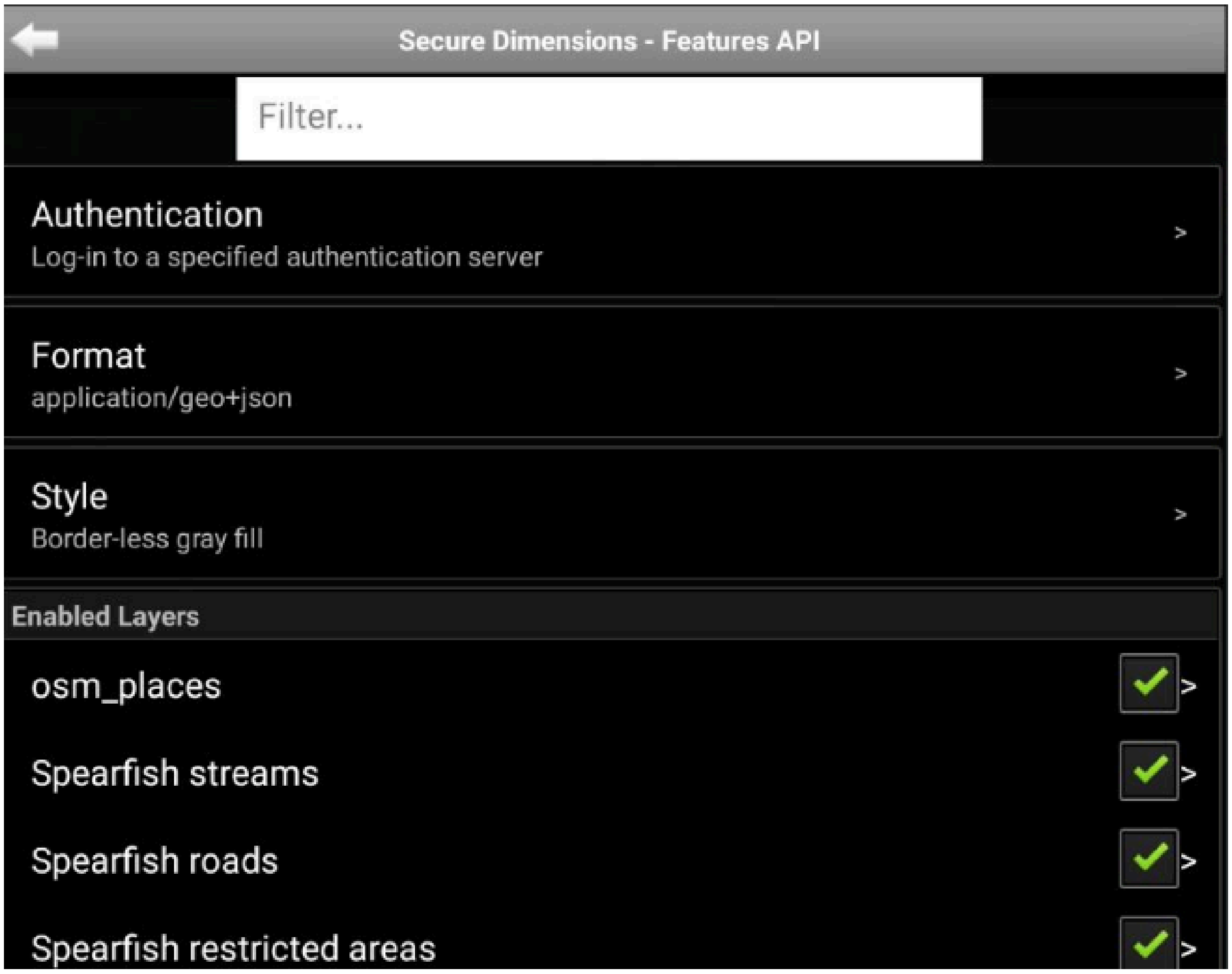
5.3.3. OGC APIs

The DCS client supports ingesting and requesting OGC API services including Tiles, Maps, Features, Styles and Images. Given an OpenAPI definition file the client auto-detects the capabilities of the server and provides the user with options to change formats and styles, as well as downloading GeoPackage content on a service and layer level. For this testbed the following services were utilized:

- OGC API — Features
- OGC API — Maps
- OGC API — Tiles

5.3.3.1. OGC API — Features

The DCS client reads the OpenAPI definition for an OGC API — Features service and through successive calls to /collections/{collectionId} produces a service with all of its required metadata for requesting content, including available styles and formats.


Secure Dimensions - Features API

Filter...

Authentication
Log-in to a specified authentication server

Format
application/geo+json

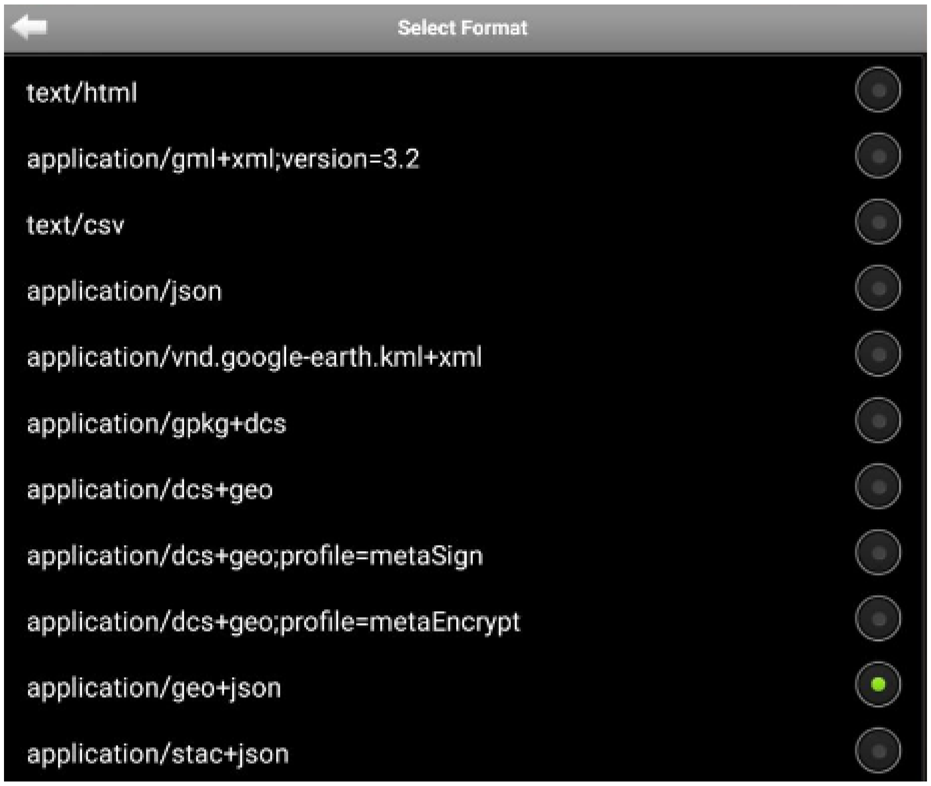
Style
Border-less gray fill

Enabled Layers

osm_places ✓
Spearfish streams ✓
Spearfish roads ✓
Spearfish restricted areas ✓

Figure 15 — DCS Client for OGC API — Features implementation

To request Data Centric Security content a user must select one of the available DCS formats. After selecting one of the appropriate formats, the client ensures the user is authorized and adds the appropriate request parameters/headers to request the encrypted content.



text/html
application/gml+xml;version=3.2
text/csv
application/json
application/vnd.google-earth.kml+xml
application/gpkg+dcs
application/dcs+geo
application/dcs+geo;profile=metaSign
application/dcs+geo;profile=metaEncrypt
[✓ application/geo+json]
application/stac+json

Figure 16 — DCS Client — OGC API Feature Formats

After receiving the content the client determines the `data` content-type by decoding and decrypting the `data_description`.

```
  "data_description":
 "eyJqa3UiOiJodHRwczpcL1wvb2djLnNlY3VyZS1kaW1lbnNpb25zLmNvbVwvZGNzXC8ud2VsbC1rbm93blwvan
eyJ0eXBlIjoiRmVhdHVyZSIsInByb3BlcnRpZXMiOnsibmFtZSI6InBvbHlfbGFuZG1hcmtzIiwibmFtZXNwYWNl
fi4XeIgcMKurVsew3IewJpSXIH4n8TKs3SJpMsFCUQ0yteH6VSeT1qGNz4_
PGADbWyCxOh4LZy8WB7tIR95PyIOix_D0cxS6YPI47k_eBDQ11WznyrHRs80iItXMCVxRr9o_
eHevPVARts63fxg-x-esw9clhU4CexQEjPAOsT9ETAVjRgidSweS0Sva5INJ53BsYrhDiaR5at-
dzuNI0BEoz-
S8bCUF9KFgVBjnkA5mYbWNS7KNGd6htXoJoujdGaA1TfucRLLGlUJib6iL3wdnY5cplelyC9JO3MVnJjk4tsOFeh
Eub3fnY_VXA2Q"
```

After decoding the content, the header is used to retrieve the JWK and decrypt the content.

```
{
  "jku": "https:\/\/ogc.secure-dimensions.com\/dcs\/.well-known\/jwks.json",
  "kid": "Dr. No",
  "alg": "RS256"
}
```

The result is a GeoJSON feature which describes the features extents, content-type and other application specific metadata.

```
{
  "type": "Feature",
  "properties": {
    "name": "poly_landmarks",
    "namespace": "http://www.census.gov",
    "content_type": "application/geo+json"
  },
  "geometry": {
    "type": "MultiPolygon",
    "coordinates": [
      [
        [
          [
            40.87822,
            -73.926377
          ],
          [
            40.87042,
            -73.932477
          ],
          [
            40.874699,
            -73.938081
          ],
          [
            40.882078,
            -73.933406
          ],
          [
            40.87901,
            -73.924598
          ],
          [
            40.878978,
            -73.924506
          ],
          [
            40.87822,
            -73.926377
          ]
        ]
      ]
    ]
  }
}
```

After determining the content-type of the encrypted content a similar approach is taken to decrypt the `data` in the response. Here the header contains a KMS request that allows the user to reach out to the KMS service and retrieves the appropriate key given the user credentials.

```
{
  "iss": "https:\/\/ogc.secure-dimensions.com",
  "cty": "application\/geo+json",
  "alg": "RS256",
  "jku": "https:\/\/ogc.secure-dimensions.com\/dcs\/.well-known\/jwks.json"
}
```

After the decryption the client converts the GeoJSON features and renders the content on the map.

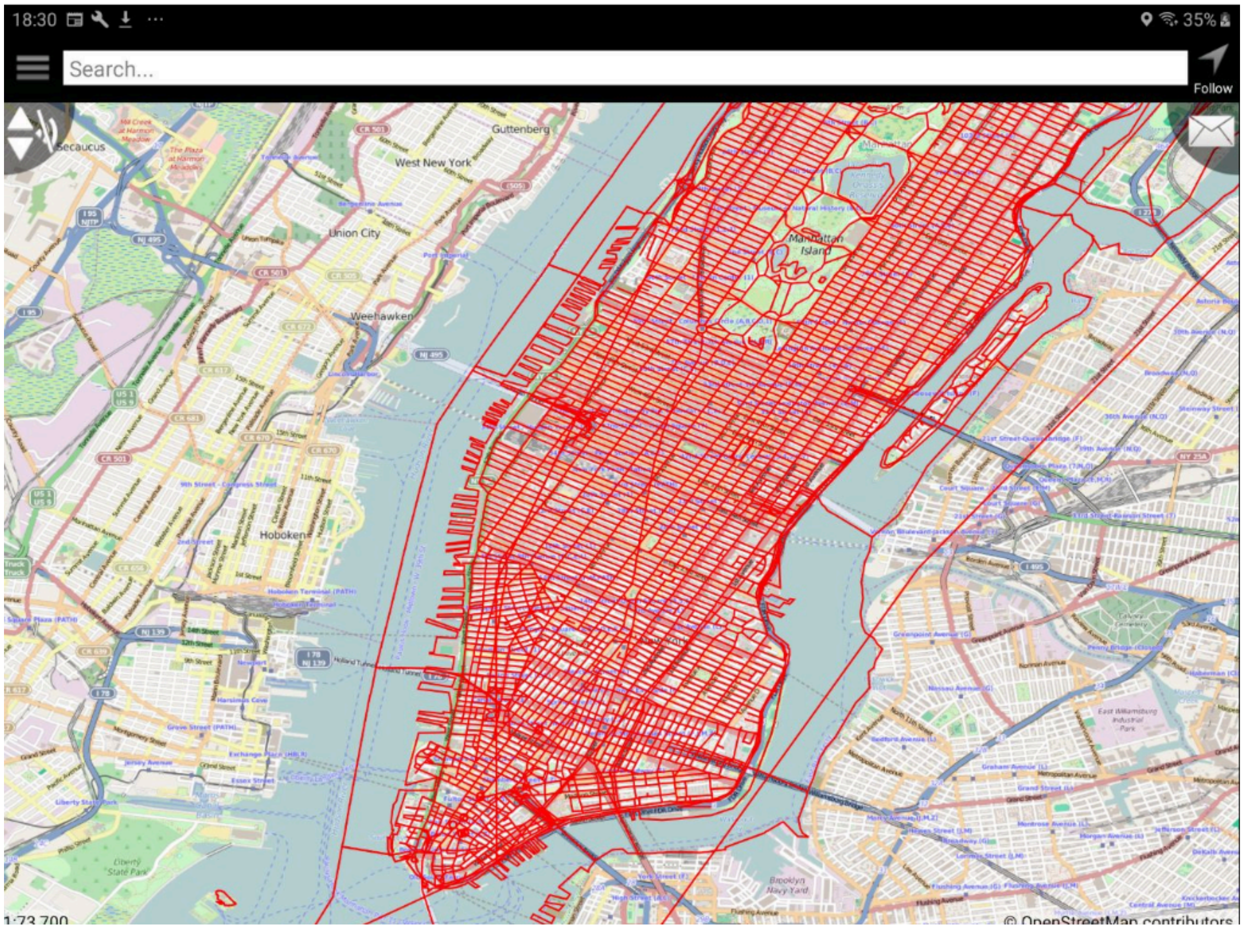
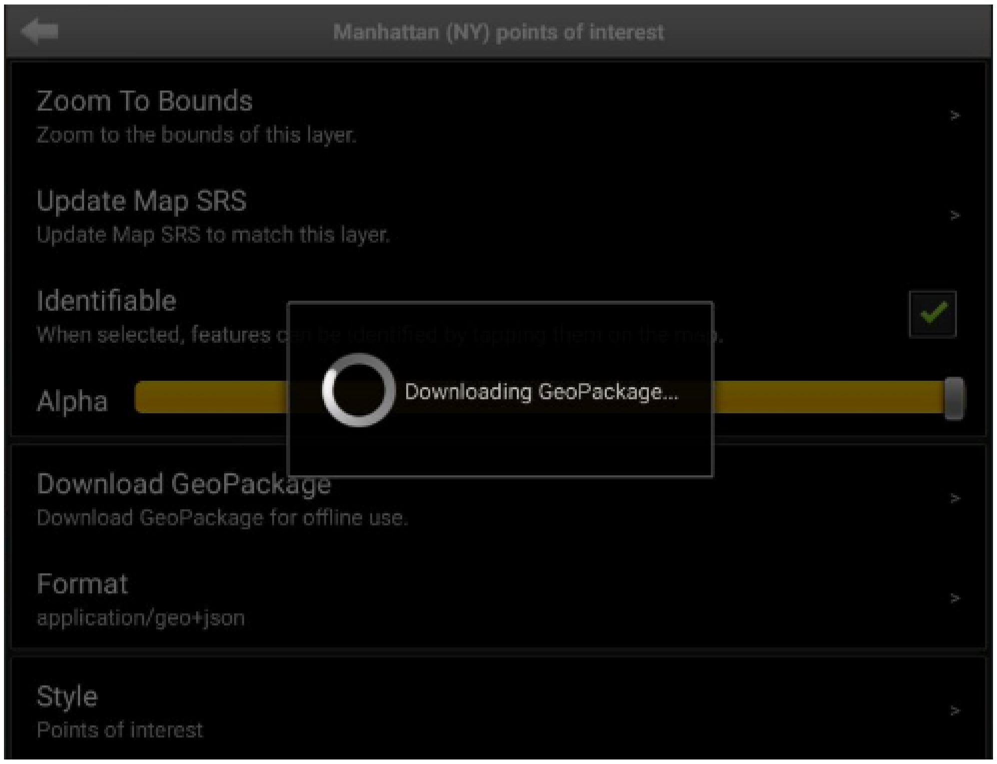


Figure 17 — DCS Client — OGC API Features — Manhattan (NY) Roads

5.3.3.1.1. GeoPackage from OGC API — Features

The DCS client checks the available formats from the OGC API — Features service and determines if it supports requesting encrypted GeoPackages. If it finds the format of `application/dcs+gpkg`, an option is made available for the user to download a GeoPackage.

```
{
  "href": "https://ogc.secure-dimensions.com/geoserver/ogc/features/
collections/tiger:giant_polygon/items?f=application%2Fgpkg%2Bdcs",
  "rel": "items",
  "type": "application/gpkg+dcs",
  "title": "tiger:giant_polygon items as application/gpkg+dcs"
}
```


Manhattan (NY) points of interest

Zoom To Bounds
Zoom to the bounds of this layer.

Update Map SRS
Update Map SRS to match this layer.

Identifiable
When selected, features can be identified by tapping them on the map.

Alpha

Downloading GeoPackage...

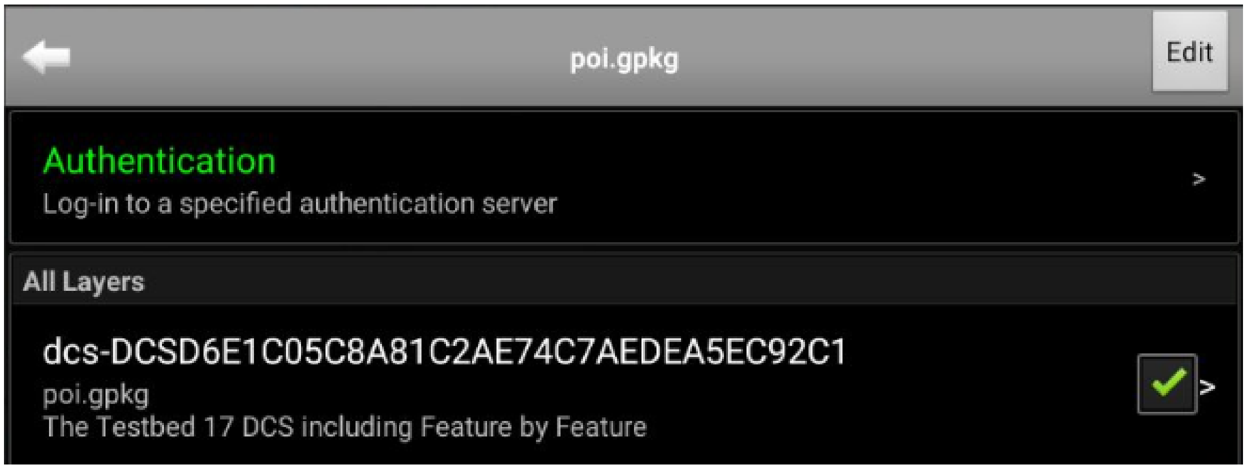
Download GeoPackage
Download GeoPackage for offline use.

Format
application/geo+json

Style
Points of interest

Figure 18 — DCS Client — OGC API Features — GeoPackage Download

Once downloaded the GeoPackage metadata is read from `gpkg_contents` and the GeoPackage service is generated. The `gpkg_extensions` table is then read to determine if the GeoPackage is a Data Centric Security compliant Feature GeoPackage based on the extension_name `sd_dcs_features`. The encrypted format of the content can also be determined by reading the `gpkg_data_columns` table. See the GeoPackage DCS Features Extension for more details.


poi.gpkg

Edit

Authentication
Log-in to a specified authentication server

All Layers

dcs-DCSD6E1C05C8A81C2AE74C7AEDEA5EC92C1
poi.gpkg
The Testbed 17 DCS including Feature by Feature

Figure 19 — DCS Client — OGC API Features — GeoPackage Service

The client then requests the feature content based on the map extents like any other GeoPackage Simple Feature layer. The `structure` column must first be decrypted to determine the content-type and the extents of the feature along with other metadata.

```
eyJraWQiOiI3MzBiMWJkOC0xNDE0LTQ2ZjAtOGRjMy02NzEzZjdlODZhZjgiLCJjdHkiOiJKV0UiLCJlbmMiOiJB
gxAXPp36uoNBC9WiTghy1sEgSG9iKwyYtaRGDunItQq5KmFKaOAUb06cBdnbOb2CvY3c-
sAVPT-21VZ6F1fmjG1uUV_sLmRxdjiW7XZKs1T3hc-Zn9Oy4Ak-3iOTukdtv883j0q_
fNMatTbdxWJh3tGRzTHMAwf_
lVNmqo0fAYKilIr6160AN6QxF97gGVFqHKfvXzY2l7thJV9poQtGXi8_e31iAJEyj38m_
XN91nXhuqocJxe4mWPifZuRa5B2LpVhw3BRSjLAVJ0mIZesKlUoQpM0Gn0HlaVGE4UGavize0KrLcKq34kWmt787
rAHfq9vlpL0b3rQn87J8pOUw.M_S4vt6kGIapv8SY.
AUx5Tnq5s3fs8qbkjFGU0Zqg6qUKl2CjVfctOMrHE3O4szdiKlPDlMsO2mLWqiZPcyHdXVjIOGZ5769YHnMyQp8t
WggaqUiYq8lFWeuNb2mjjylA1okuYMM0GZfEh6F0vqYIf2wLpw7mw28FMcfoO6v-
sYThF0yDglUV3i0X0bDlU18uhgH1HNGyNsepEBg_
Mj9i6hr9pcWeyb22iTNuKj7useXC6KRHrgzyWu8UeqJkQLFxGLo_kZuutjw3H-6_
CRpPXAETb5MubJVCo05jISx1btnJu5UX8S8L5LjcXwBCpAKVL_eHOF_OnKersgnN-
6H6FygtamS6RPkg_
b9Fshtwny6LxsnaDNxG3tXMyMaB02qGBwaOMkTHijCY7tJWyvlR9kZI8jsxccTeWxncVGJ128ge2H3MR40V8rof2
Eb1u9u7OCnLd-
2EyKWfdbWPcHseiEhNb6u6RfHqC2xcLfnnA7ZTJIZH0tDDwhLASODhSvFfaolIkxda-
3fd6IbaDh6S0r_IdH_Zlt6LObo6A7ESstFKUHiExT5YAKmM6FCg_t_BGt8jvVnxL9N-
Xe94p77bnoPUDWYJv-
y1yYRq4ExeJDuejv1uBbBnG0vLDxa7Hm8vwrNLcJVNeyUVtkwolEGdMZ1cg97geoKo7scQ85kJiZy6CBXaBcVl7O
uIbDxUB7yCTpuA6uKA-lbFiXSk2bEokGPdInTO8liLYWCoYqhn3NujJOnJ-
RYuMdQ8yvOYWJGHKZFHD9Lbr1MnGPPhVFq0ZovOdvbLFTsEuWxSKJaM-9LfBPmFyuoEE2OZRN3dDtu_
cBe7_KS7qriO3sltCPSK1-00Wehcxx4bXFCDC2arnCAMx0rMAtYJPhaL_
p6ibkGPrApJWxs529MefgGi5WRe3ImOWOGEPHJDsXSiHIaGJudzE57QhQQp6fQTOZR-
H8fLiuy_krC1ykJkeuGJX_Cpnr_nNnz2SHuiSr_Hve2piKI1JGzilU_
PcIyv8fguiCk_XIjcDhioclcYSR99XYJ_scCSuntALoVMbMRVvI7dXHEKTPjOR-
mZ6WVYPyClBe7OFd0EK6fdtyIDvNoaRX0DW2U8aeYE-1_fOg5F2EvO-wsEd7BgJ100C15ik_
AsOYJjlHbkLhL5B246Z2A8dIillSE913XMKrEWcYf25SKVp1KEzDFFx9v193hZOZn4OB95DgVuqe23PWBzsexlwL
3ubTT3yYZBIInUOUFeKQWQ
```

The client then requests the encryption key from KMS using the information available in the header.

```
{
  "kid": "730b1bd8-1414-46f0-8dc3-6713f7e86af8",
  "cty": "JWE",
  "enc": "A256GCM",
  "alg": "RSA-OAEP-256"
}
```

After retrieving the key, the structure is decrypted and verified to ensure the user has permission to consume the content. Using the `content_type` property the client determines what type of data to inspect in the encrypted `data` column.

```
{
  "type": "Feature",
  "properties": {
    "name": "DCSD6E1C05C8A81C2AE74C7AEDEA5EC92C1",
    "namespace": "tiger",
    "namespace_uri": "http://www.census.gov",
    "content_type": "application/geo+json",
    "attributes": [
      "the_geom",
      "NAME",
      "THUMBNAIL",
      "MAINPAGE"
    ]
  },
  "geometry": {
    "type": "Polygon",
    "coordinates": [
      [
        [
          40.70754684,
          -74.01083751
        ],
        [
          40.70754684,
          -74.01083751
        ],
        [
          40.70754684,
          -74.01083751
        ],
        [
          40.70754684,
          -74.01083751
        ],
        [
          40.70754684,
          -74.01083751
        ]
      ]
    ]
  }
}
```

A similar process is followed to decode, decrypt and verify the `data` content containing the GeoJSON features. After decrypting the content, the GeoJSON is rendered on the map.

```
{
  "type": "Feature",
  "geometry": {
    "type": "Point",
    "coordinates": [
      40.7075,
      -74.0108
    ]
  },
  "properties": {
    "NAME": "stock",
    "THUMBNAIL": "pics\/22037829-Ti.jpg",
    "MAINPAGE": "pics\/22037829-L.jpg"
  },
  "id": "poi.2"
}
```

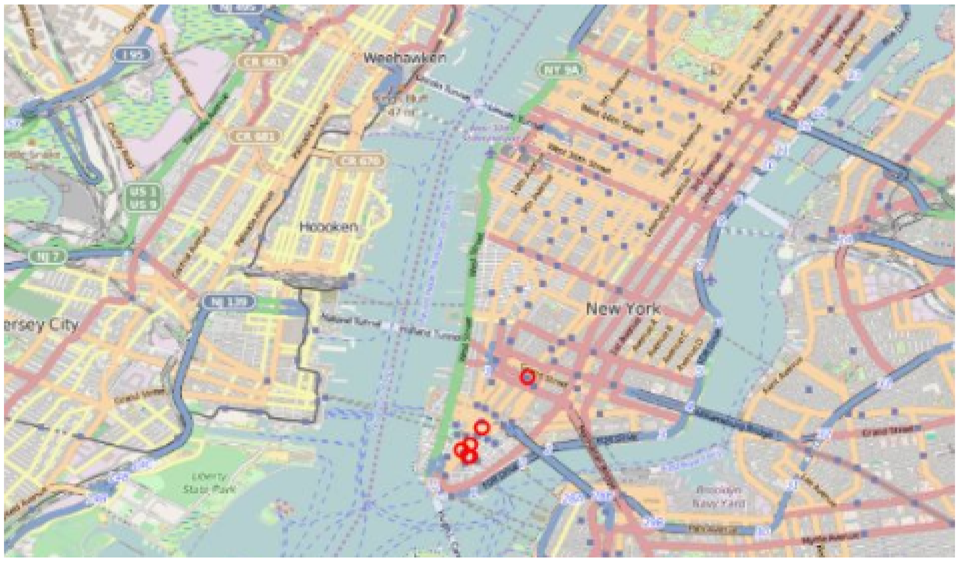
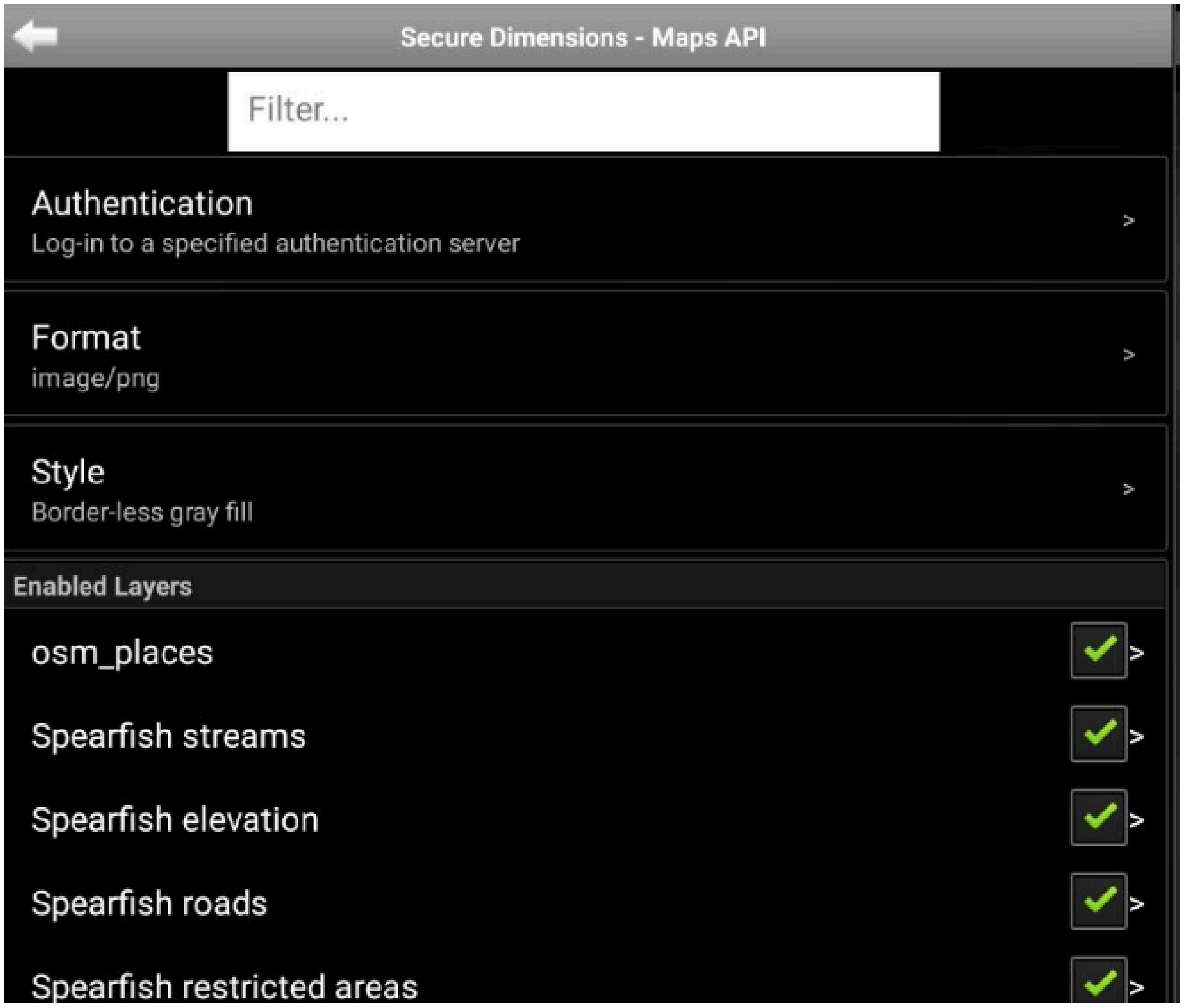


Figure 20 — DCS Client — OGC API Features — GeoPackage — Points of Interest

5.3.3.2. OGC API — Maps

The DCS client reads the OpenAPI definition for an OGC API Maps service and through successive calls to /collections/{collectionId} and /collections/{collectionId}/styles produces a service with all of its required metadata for requesting content, including available styles and formats.


Secure Dimensions - Maps API

Filter...

Authentication
Log-in to a specified authentication server

Format
image/png

Style
Border-less gray fill

Enabled Layers

osm_places ✓

Spearfish streams ✓

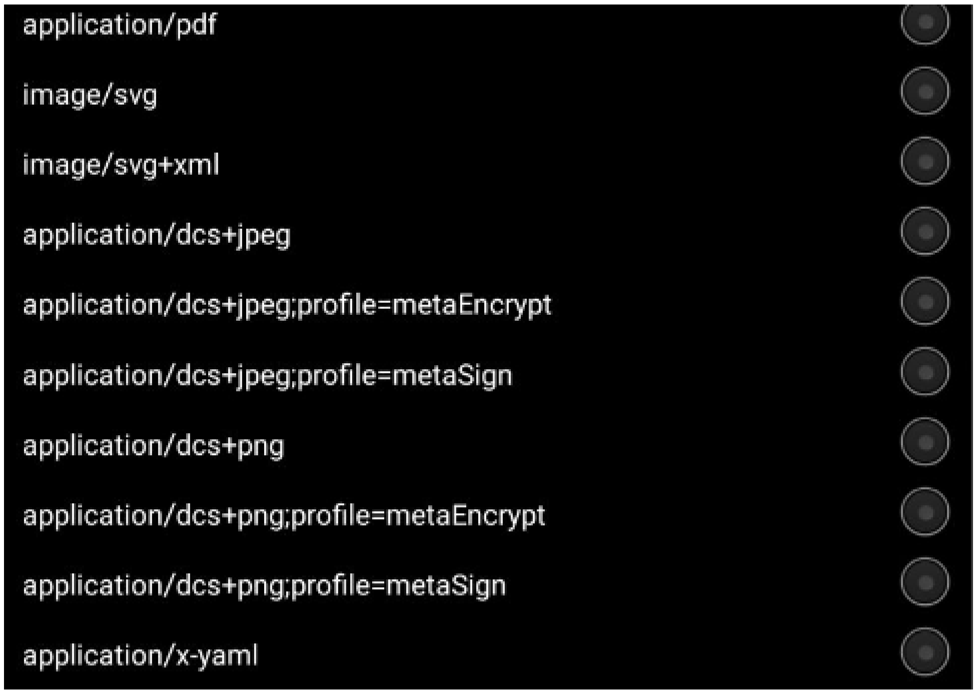
Spearfish elevation ✓

Spearfish roads ✓

Spearfish restricted areas ✓

Figure 21 — DCS Client — OGC API Maps Service

To request Data Centric Security content a user must select one of the available DCS formats. After selecting one of the appropriate formats, the client ensures the user is authorized and adds the appropriate request parameters/headers to request the encrypted content.



application/pdf

image/svg

image/svg+xml

application/dcs+jpeg

application/dcs+jpeg;profile=metaEncrypt

application/dcs+jpeg;profile=metaSign

application/dcs+png

application/dcs+png;profile=metaEncrypt

application/dcs+png;profile=metaSign

application/x-yaml

Figure 22 — DCS Client — OGC API Maps Formats

After receiving the content the client decodes and decrypts multi-part binary content based on its format type. External keys are retrieved from the KMS. These along with the applications stored private key are used to decrypt the content and metadata and display the images on the map.

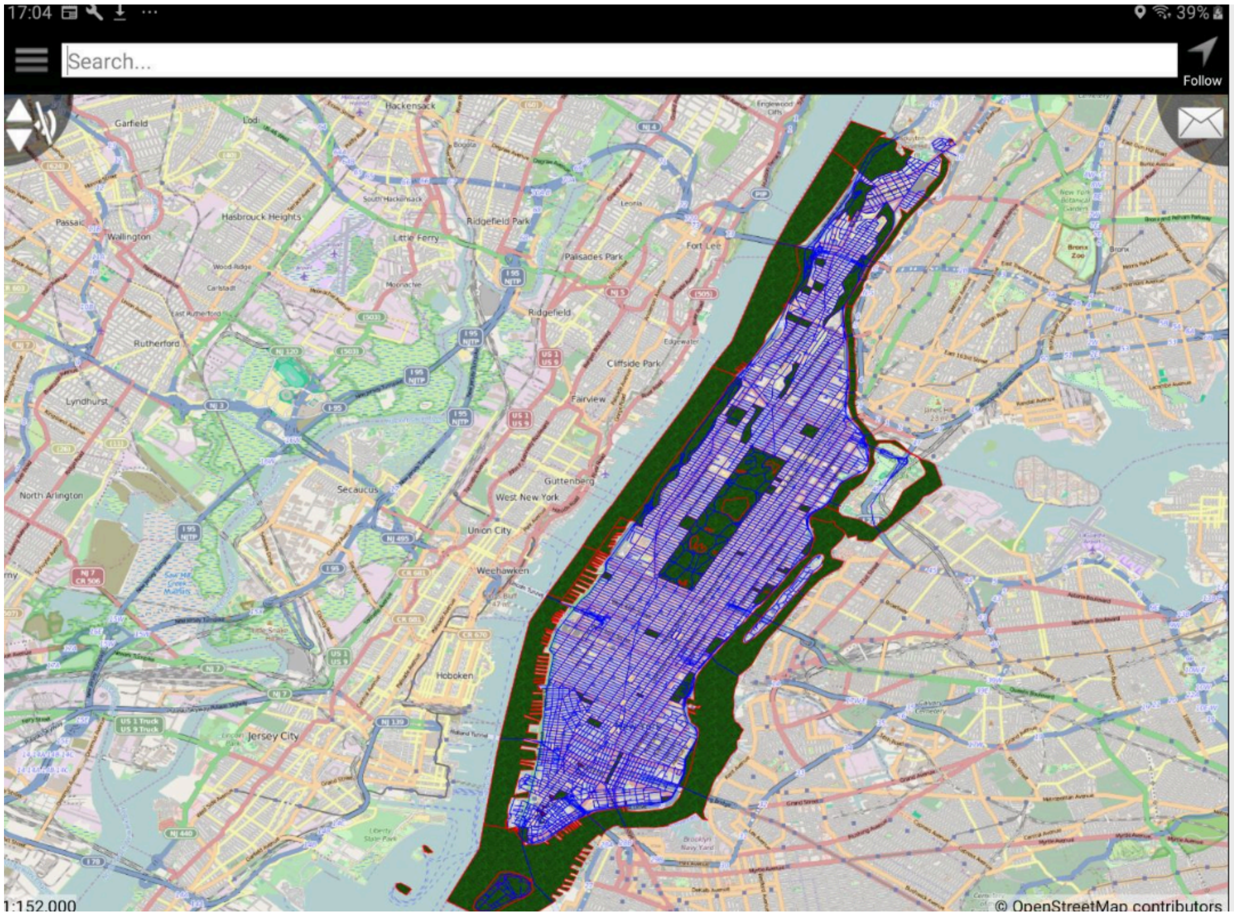
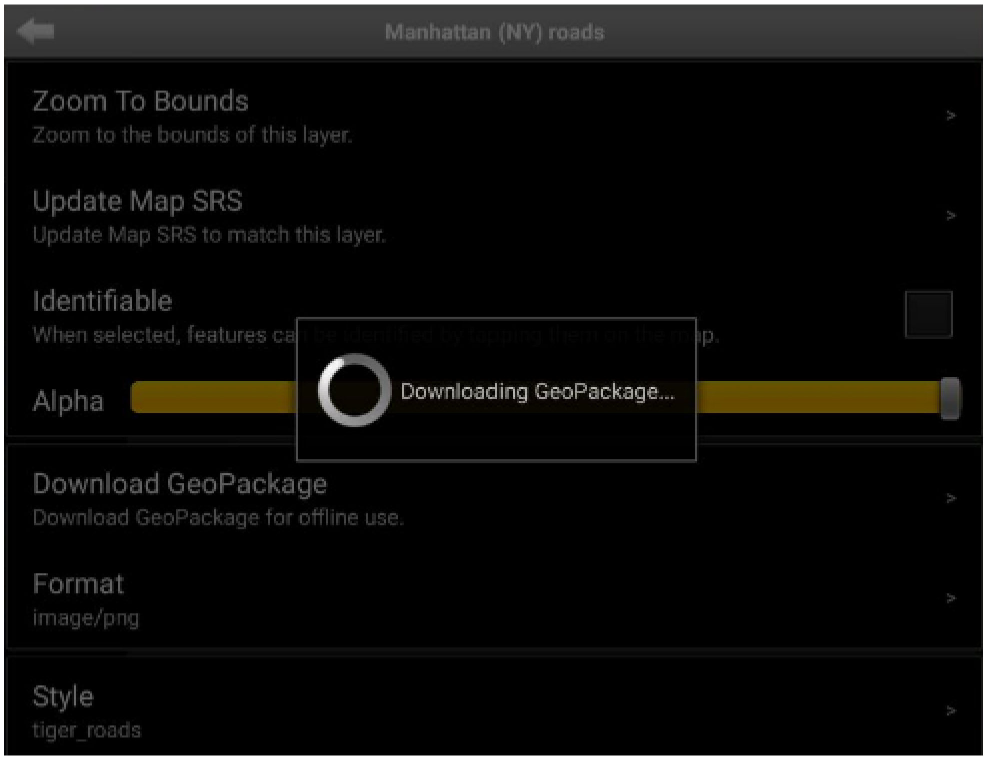


Figure 23 — DCS Client — OGC API Maps — Manhattan (NY) Roads/Landmarks

5.3.4. OGC API Tiles — GeoPackage

The DCS client checks the available formats from the OGC API — Tiles service and determines if it supports requesting encrypted GeoPackages. If it finds the format `tilematrixset-dcs` or `tilematrix-dcs` an option is made available for the user to download a GeoPackage.

```
{
  "href": "https://ogc.secure-dimensions.com/geoserver/ogc/tiles/collections/
tiger:giant_polygon/map/giant_polygon/tiles/EPSG:4326?f-tile=image/png&f=
application/gpkg%2Bdcs&multiTileType=tiles",
  "rel": "tilematrixset-dcs",
  "type": "image/png",
  "title": "tiger:giant_polygon tile matrix set as DCS GeoPackage"
},
{
  "href": "https://ogc.secure-dimensions.com/geoserver/ogc/tiles/collections/
tiger:giant_polygon/map/giant_polygon/tiles/EPSG:4326/{tileMatrix}?f-tile=
image/png&f=application/gpkg%2Bdcs&multiTileType=tiles",
  "rel": "tilematrix-dcs",
  "type": "image/png",
  "title": "tiger:giant_polygon tile matrix {tileMatrix} as DCS GeoPackage"
}
```


Manhattan (NY) roads

Zoom To Bounds
Zoom to the bounds of this layer.

Update Map SRS
Update Map SRS to match this layer.

Identifiable
When selected, features can be identified by tapping them on the map.

Alpha

Downloading GeoPackage...

Download GeoPackage
Download GeoPackage for offline use.

Format
image/png

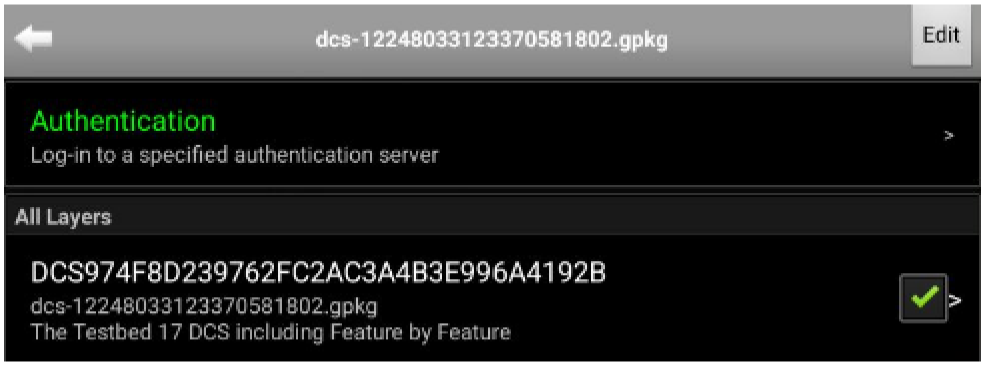
Style
tiger_roads

Figure 24 — DCS Client — OGC API Tiles GeoPackage Download

After selecting the layer to download and choosing to download a GeoPackage the client generates a request using the users access token and selected format type and projection.

https://ogc.secure-dimensions.com/geoserver/ogc/tiles/collections/tiger:poi/map/poi/tiles/EPSG:4326?f-tile=image/png&f=application/gpkg%2Bdcs&multiTileType=tiles&key_challenge=secret&key_challenge_method=plain&access_token=998abda1e387e99003c8a9ff5a314d1a51381b72&kek_kid=730b1bd8-1414-46f0-8dc3-6713f7e86af8

Once downloaded, the GeoPackage metadata is read from `gpkg_contents` and the GeoPackage service is generated. The `gpkg_extensions` table is then read to determine if the GeoPackage is a Data Centric Security compliant Feature GeoPackage based on the extension_name `sd_dcs_tiles`. The encrypted format of the content can also be determined by reading the `gpkg_data_columns` table. See the GeoPackage DCS Tiles Extension for more details.


dcs-12248033123370581802.gpkg

Authentication
Log-in to a specified authentication server

All Layers

DCS974F8D239762FC2AC3A4B3E996A4192B
dcs-12248033123370581802.gpkg
The Testbed 17 DCS including Feature by Feature

Figure 25 — DCS Client — OGC Tiles API — GeoPackage Service

The client then requests the feature content based on the map extents like any other GeoPackage Tiled layer. The result is encoded metadata `dcs_info`, encrypted key information `dek_info` and the encrypted tiled images `tile_data`.

First `dcs_info` is decoded, and the header is used to retrieve the key. The parameter `jku` is used to retrieve the keyset and load it.

```
{
  "keys": [
    {
      "kty": "RSA",
      "e": "AQAB",
      "use": "sig",
      "kid": "Dr. No",
      "n": "kHGuPCEyY4adIpPgMuw7H5H5zh03eADvj_
Tc12zFR3LavpAHrkC8XxGbdgXmknIfN2uV_jcg5uM4crbJGfqnS1elznJjAEZ3e1kwZOTKPiqGqrL-
6DgvFjLgdh8JcT95z1Q5MIO7u9-Ru-YR-
DxXQzYN9Pq3lee8VmaRSRICLgvi9D08m2GwUvIlerac2WG04xB4FbFw7NuiDoQEAHuppTK6aWusiiOSwd31Nvg6b
4pt5rjOof46jjjM_YbRd0IIyIq0vufGogB_C5yndIQa2odcY2_rVEayqKFBu2VuFGSmtw"
    }
  ]
}
```

The key defined in the header is then used from the response to decode the JWT content.

```
{
  "originator_confidentiality_label": {
    "confidentiality_information": {
      "policy_identifier": "TB17",
      "classification": "Top Secret"
    }
  },
  "data_producer": {
    "origin": "Not NGA",
    "date": "2021-08-25T18:38:23.957Z"
  },
  "tile_information": {
    "zoom_level": 14,
    "tile_column": 9647,
    "tile_row": 4486
  },
  "data_information": {
    "hash_alg": "SHA-384",
    "hash_value":
 "c5c48da9f073c0a0e477d5d7bced25c81e5dc0258b670d9450b33f96275a9f66fea6c81b1ef3475695144e
```

```
  }
}
```

A similar process is followed to decode the JWS `dek_info` using the key defined in the header to decode the payload resulting in the decryption key required for the `data`.

```
{
  "sub": "c0fb2eea-1856-3673-80fc-79d0e1efdfdb",
  "aud": "8d90bc42-4401-5f2a-9054-cb407a876ad8",
  "kurl": "https:\/\/ogc.secure-dimensions.com\/kms\/keys\/c2e5dc56-b14b-4f28-
8e72-a78779271744",
  "kid": "c2e5dc56-b14b-4f28-8e72-a78779271744",
  "iss": "https:\/\/ogc.secure-dimensions.com",
  "exp": 1629918,
  "alg": "A256GCM",
  "iat": 1629916703949
}
```

The decryption key is then used in conjunction with the `Java Cryptography Architecture` framework to decrypt the image. The first 16 bytes are read from the `data` column and used as the IvParameterSpec input to the cipher. The result are decrypted tiles based on the format requested by the client.

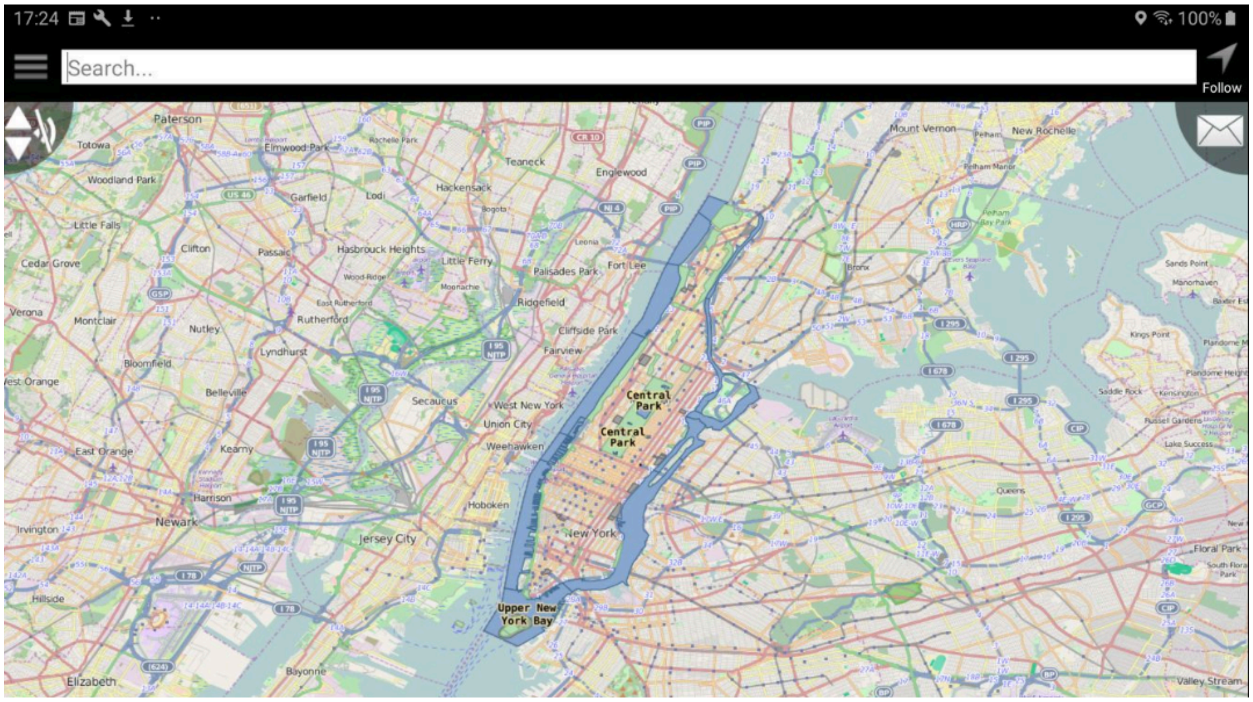
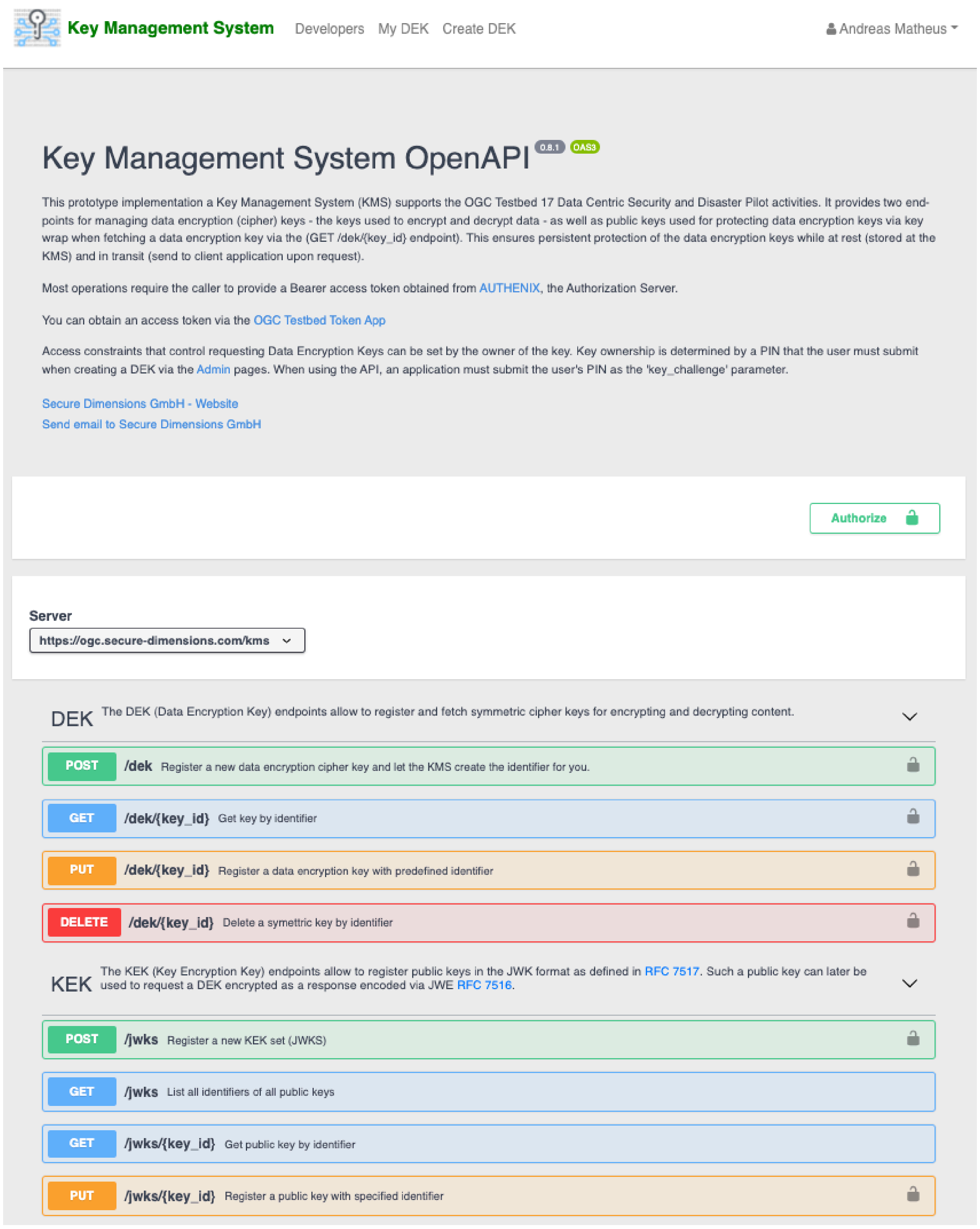


Figure 26 — DCS Client — OGC Tiles API — Geopackage Landmarks (NY)

5.4. Key Management System

The Key Management System (KMS) for Testbed-17 provides an API that offers the registration and fetching of Data Encryption Keys (DEK). The KMS supports key creation and key owners to specify access conditions for each of their DEK via portal pages, accessible after login. In addition, the KMS also offers an API for the registration of Key Encryption Keys (KEK) in order to encrypt responses that contain a DEK.



Key Management System OpenAPI 0.8.1 OAS3

This prototype implementation a Key Management System (KMS) supports the OGC Testbed 17 Data Centric Security and Disaster Pilot activities. It provides two end-points for managing data encryption (cipher) keys - the keys used to encrypt and decrypt data - as well as public keys used for protecting data encryption keys via key wrap when fetching a data encryption key via the (GET /dek/{key_id} endpoint). This ensures persistent protection of the data encryption keys while at rest (stored at the KMS) and in transit (send to client application upon request).

Most operations require the caller to provide a Bearer access token obtained from AUTHENIX, the Authorization Server.

You can obtain an access token via the OGC Testbed Token App

Access constraints that control requesting Data Encryption Keys can be set by the owner of the key. Key ownership is determined by a PIN that the user must submit when creating a DEK via the Admin pages. When using the API, an application must submit the user's PIN as the 'key_challenge' parameter.

Secure Dimensions GmbH - Website
Send email to Secure Dimensions GmbH

Server
https://ogc.secure-dimensions.com/kms

DEK The DEK (Data Encryption Key) endpoints allow to register and fetch symmetric cipher keys for encrypting and decrypting content.

POST /dek Register a new data encryption cipher key and let the KMS create the identifier for you.

GET /dek/{key_id} Get key by identifier

PUT /dek/{key_id} Register a data encryption key with predefined identifier

DELETE /dek/{key_id} Delete a symettric key by identifier

KEK The KEK (Key Encryption Key) endpoints allow to register public keys in the JWK format as defined in RFC 7517. Such a public key can later be used to request a DEK encrypted as a response encoded via JWE RFC 7516.

POST /jwks Register a new KEK set (JWKS)

GET /jwks List all identifiers of all public keys

GET /jwks/{key_id} Get public key by identifier

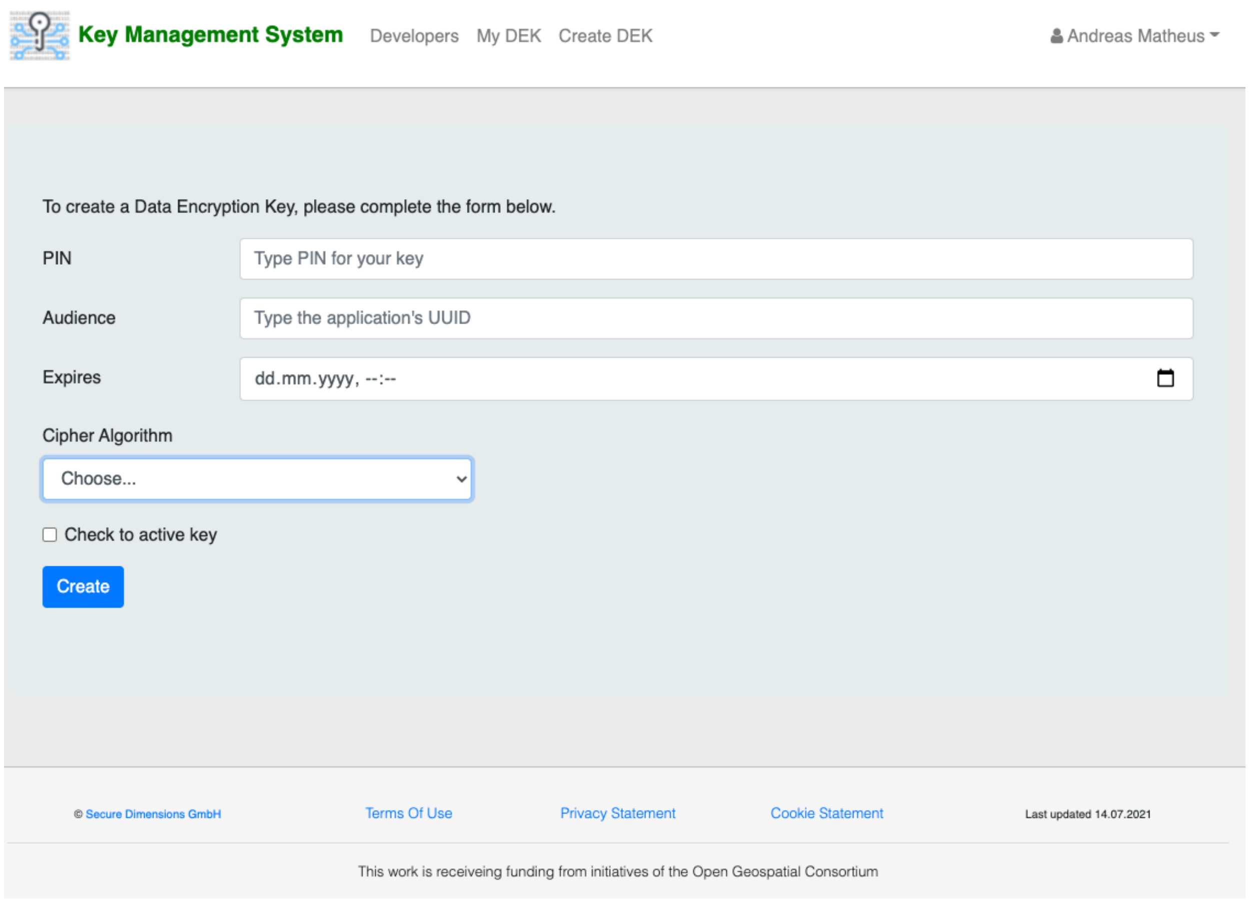
PUT /jwks/{key_id} Register a public key with specified identifier

Figure 27 — KMS API in OpenAPI

5.5. Data Encryption Key Creation

A keys' lifetime begins with its creation. This can take place in an encryption service like the DCS Server, offline or via the KMS admin interface. After the time of creation, no data has been ciphered yet and so the usage of the key is `encryption`. At the stage of encryption, the key might not be available for decryption purposes.

The following figure shows the KMS portal page for creating a Data Encryption Key.



To create a Data Encryption Key, please complete the form below.

PIN	Type PIN for your key
Audience	Type the application's UUID
Expires	dd.mm.yyyy, --:--

Cipher Algorithm

Choose...

☐ Check to active key

Create

© Secure Dimensions GmbH   Terms Of Use   Privacy Statement   Cookie Statement   Last updated 14.07.2021

This work is receiveing funding from initiatives of the Open Geospatial Consortium

Figure 28 — KMS Input form for Creating a Data Encryption Key

When creating a DEK via the KMS, the user must provide different types of information:

- `PIN` or secret that is required when adopting the access conditions of the key via the KMS Admin Portal.
- `Audience` is the UUID of the application that gets immediate access the key (if the `Check to activate key` option is selected). Additional applications can be added via the KMS Admin Portal). In the case where the DCS client shall be used for creating encrypted content based on the created key, the DCS client's UUID must be provided. If AUTHENIX is used as the Authorization Server, the UUID of the application can be found here.

- `Expires` determines the initial expiration time of the key. The expiration time can be changed via the KMS Admin Portal.
- `Cipher Algorithm` is a list of symmetric ciphers as illustrated in Figure 29. The top of the list is used with JWK and the list identified with URIs is used specifically for XML Encryption.
- `Check to activate key` option allows to make the key available immediately to the application specified above and until the expiration time.

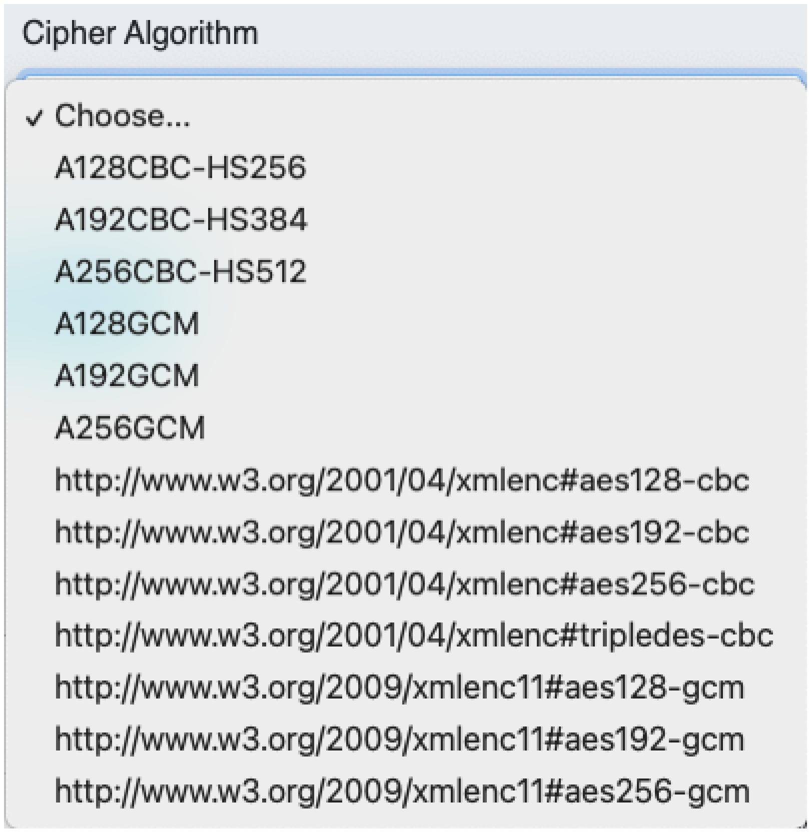

Cipher Algorithm
✓ Choose...
A128CBC-HS256
A192CBC-HS384
A256CBC-HS512
A128GCM
A192GCM
A256GCM
http://www.w3.org/2001/04/xmlenc#aes128-cbc
http://www.w3.org/2001/04/xmlenc#aes192-cbc
http://www.w3.org/2001/04/xmlenc#aes256-cbc
http://www.w3.org/2001/04/xmlenc#tripledes-cbc
http://www.w3.org/2009/xmlenc11#aes128-gcm
http://www.w3.org/2009/xmlenc11#aes192-gcm
http://www.w3.org/2009/xmlenc11#aes256-gcm

Figure 29 — KMS Ciphers for Data Encryption Key

In case a key was created offline or created by the DCS Server, the key must be registered with the KMS as no protocol exists that would allow the direct exchange of the DEK between the DCS client and the DCS Server.

5.6. Data Encryption Key Registration

The KMS offers an API for the (automated) DEK registration. The KMS API supports different options:

- the DEK can be uploaded via HTTP `POST` and the KMS created the key's identifier (`kid` in JWK terminology) via the DEK/addKey operation
- the client application creates a UUID based identifier for the DEK and uploads the key via HTTP `PUT` to a URI including the key's identifier via the DEK/addKEyById operation. In case that another key with that UUID already exists, the response is HTTP Conflict.

The HTTP response to the `POST` and `PUT` operation will contain the key identifier(s) that can be used with the `GET` operation to fetch the key details including the key's secret.

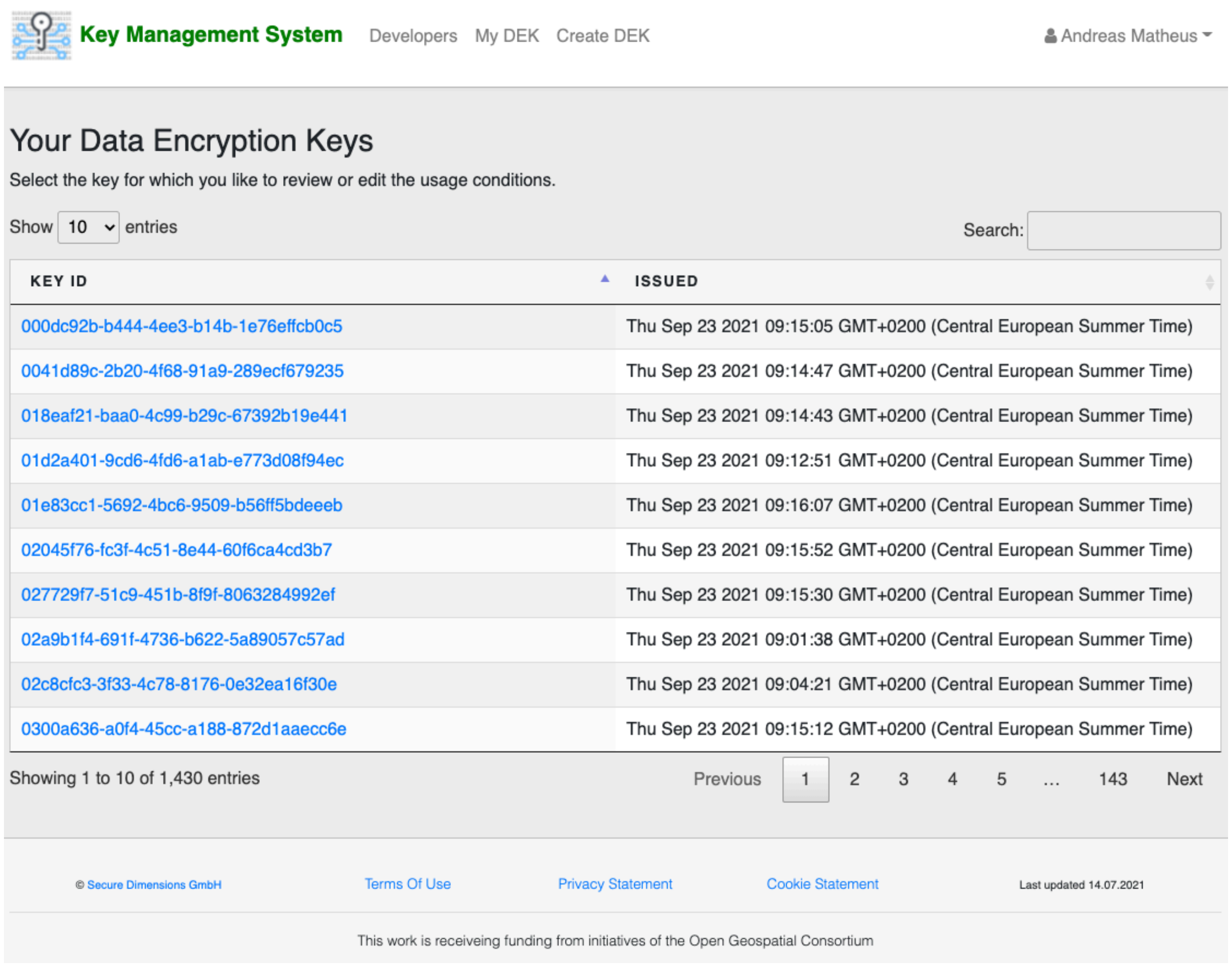
5.7. Data Encryption Key Deletion

A DEK can be deleted from the KMS via the DEK/delKeyById operation. Compliant to the NIST 800-57 recommendation and to ensure that key identifiers are unique, the HTTP `DELETE` option does delete the key's secret and deactivates the key so that is can no longer be fetched via the KMS API (the KMS will return a HTTP `GONE` if the key is requested after the deletion). Even though the key could still be reactivated via the Admin Portal, it is useless as the secret (`k`) value is set to `NULL`.

5.8. Data Encryption Key Access Management

Once a DEK is registered with the KMS, the owner of the key can modify the access conditions after login. Key ownership is identified via the PIN that a user submitted when creating a DEK via the portal pages or the DCS client submitted with the request for encrypted content via the `key_challenge` parameter.

The user must login to the KMS for viewing and adjusting access conditions to their DEK(s). Once logged in, the KMS displays a list of all keys accessible to the user (see Figure 30).



Your Data Encryption Keys

Select the key for which you like to review or edit the usage conditions.

Show 10 entries Search:

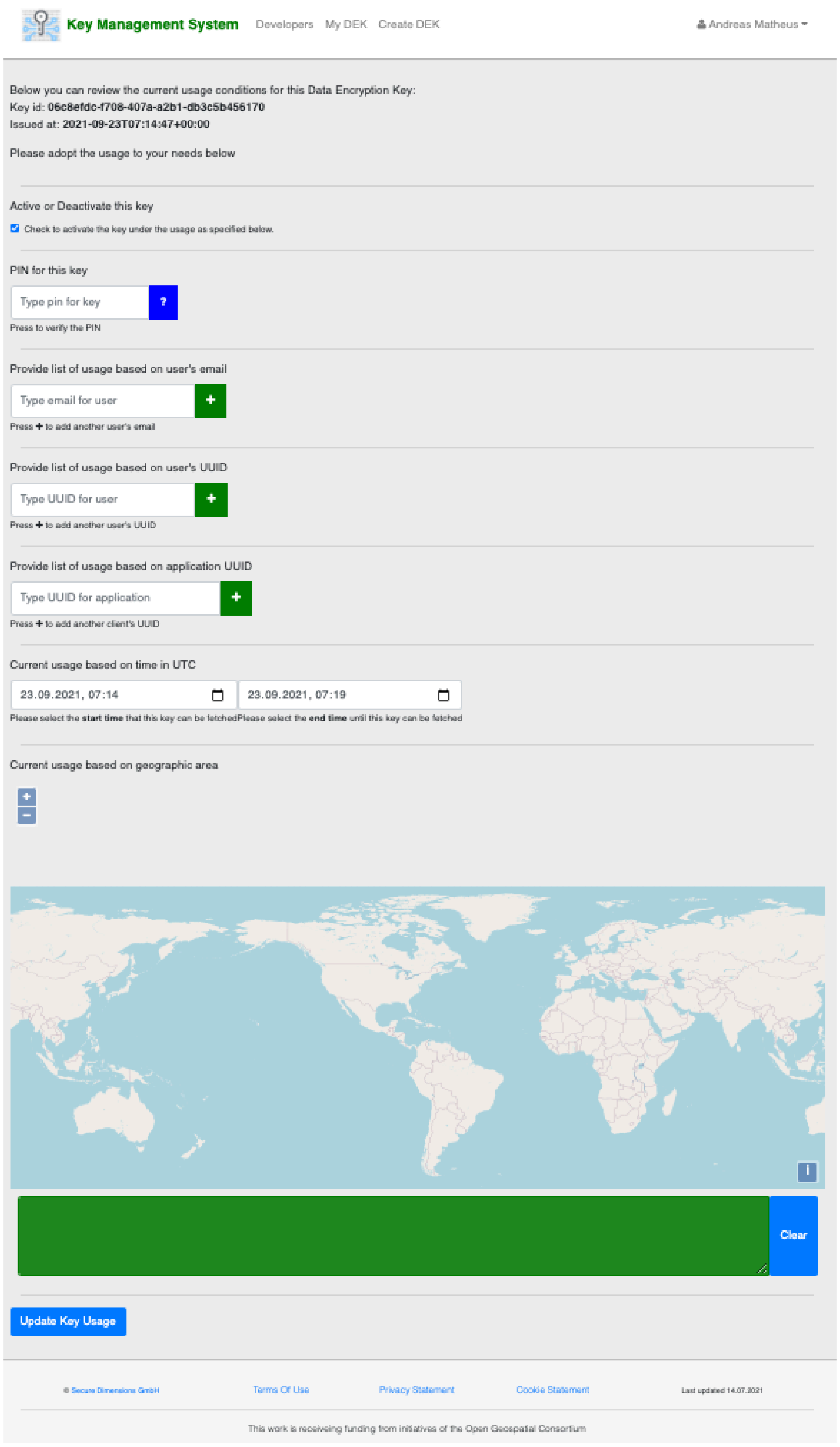
KEY ID	ISSUED
000dc92b-b444-4ee3-b14b-1e76effcb0c5	Thu Sep 23 2021 09:15:05 GMT+0200 (Central European Summer Time)
0041d89c-2b20-4f68-91a9-289ecf679235	Thu Sep 23 2021 09:14:47 GMT+0200 (Central European Summer Time)
018eaf21-baa0-4c99-b29c-67392b19e441	Thu Sep 23 2021 09:14:43 GMT+0200 (Central European Summer Time)
01d2a401-9cd6-4fd6-a1ab-e773d08f94ec	Thu Sep 23 2021 09:12:51 GMT+0200 (Central European Summer Time)
01e83cc1-5692-4bc6-9509-b56ff5bdeeeb	Thu Sep 23 2021 09:16:07 GMT+0200 (Central European Summer Time)
02045f76-fc3f-4c51-8e44-60f6ca4cd3b7	Thu Sep 23 2021 09:15:52 GMT+0200 (Central European Summer Time)
027729f7-51c9-451b-8f9f-8063284992ef	Thu Sep 23 2021 09:15:30 GMT+0200 (Central European Summer Time)
02a9b1f4-691f-4736-b622-5a89057c57ad	Thu Sep 23 2021 09:01:38 GMT+0200 (Central European Summer Time)
02c8cfc3-3f33-4c78-8176-0e32ea16f30e	Thu Sep 23 2021 09:04:21 GMT+0200 (Central European Summer Time)
0300a636-a0f4-45cc-a188-872d1aaecc6e	Thu Sep 23 2021 09:15:12 GMT+0200 (Central European Summer Time)

Showing 1 to 10 of 1,430 entries

This work is receiveing funding from initiatives of the Open Geospatial Consortium

Figure 30 — KMS Showing list of “my” keys

The user can open the access management page by clicking on the link for a particular key. As illustrated in Figure 31, the use has different options to constraint access.



Below you can review the current usage conditions for this Data Encryption Key:
Key id: **06c8efdc-f708-407a-a2b1-db3c5b456170**
Issued at: **2021-09-23T07:14:47+00:00**

Please adopt the usage to your needs below

Active or Deactivate this key

☑ Check to activate the key under the usage as specified below.

PIN for this key

Type pin for key ?

Press to verify the PIN

Provide list of usage based on user's email

Type email for user +

Press + to add another user's email

Provide list of usage based on user's UUID

Type UUID for user +

Press + to add another user's UUID

Provide list of usage based on application UUID

Type UUID for application +

Press + to add another client's UUID

Current usage based on time in UTC

23.09.2021, 07:14 | 23.09.2021, 07:19

Please select the **start time** that this key can be fetchedPlease select the **end time** until this key can be fetched

Current usage based on geographic area

This work is receiving funding from initiatives of the Open Geospatial Consortium

Figure 31 — KMS Access Management Page

- `PIN`: In order to change access conditions for the key, the user must provide the `PIN`. The blue `?` can be clicked to check if the PIN is correct. If so, the icon will change to a green check-mark; if incorrect, the icon will change to a red X.

- `activate`: The `activate` option controls whether the key can be requested via the DEK/getKeyById operation at all. If unset, the key cannot be fetched from the KMS, regardless of the conditions specified. This is a kind of emergency override to stop release of the key.

- `shared via email`: The next option allows to make the key available to users based on their email address(es). Please note that the email address of a user depends on the login provider chosen when logging in via AUTHENIX. If the `OGC IdP` is used for login, then the email address is equivalent to the email address shown in the OGC Portal; if logged in via Google, the user's Gmail address must be used for sharing.

- `shared via user Id`: In case that the key shall be shared with user(s) but the users do not want to provide their email address(es), the user can provide their UUID. The UUID is displayed once logged into AUTHENIX. The user's identifier UUID is displayed at the bottom of the page. It is the value for the `Sub` attribute name. This honors the privacy of the users as coined in European Union's General Data Protection Regulation (GDPR).

- `shared via application Id`: Any application that needs to obtain a DEK from the KMS must first be registered with AUTHENIX and the `client_id` UUID from the registration result must be put into the `application UUID` box. Any application that is currently registered with AUTHENIX can be found following the Operators link. For example, the OGC Token App used for various demonstrations has UUID `019b7173-a9ed-7d9a-70d3-9502ad7c0575` and the DCS client that supports decryption of GeoPackages with Encryption Extension has UUID `8d90bc42-4401-5f2a-9054-cb407a876ad8`.

- `temporal conditions`: Next, the user can specify the time window of access. The `from` time can be set to the future which is an important feature if encrypted data is packaged on some mobile devices and they are shipped to their final destination. While the devices are in transit, the associated decryption key cannot be accessed.

- `spatial conditions`: In addition to the temporal conditions, also geospatial conditions can be added. The selection of multiple polygons allows access to be constrained to particular areas. This ensures for example that mobile devices can only obtain the decryption key if within the 'green' area.

The registration and fetching of DEKs require that the request contains a Bearer access token issued by AUTHENIX. From the Bearer access token, the KMS determines the acting user (`sub`) and the application that issued the API request (`aud`). These two pieces of information are essential key access conditions.

6

TECHNOLOGY INTEGRATION EXPERIMENTS (TIES)

6 TECHNOLOGY INTEGRATION EXPERIMENTS (TIES)

The TIEs for the Testbed-17 DCS task were grouped into multiple tests for each of two scenarios. Testbed-17 primarily focused on applying DCS to binary geospatial data (maps and tiles) and geospatial data sets in GeoPackage container format (tiles and features). The TIEs are divided into sub-TIEs as follows:

- Provisioning and portrayal of DCS protected geospatial binary content.
- Provisioning and portrayal of DCS protected layers of geospatial vector and binary content in GeoPackage format.

The TIEs are executed between the DCS Server and the DCS Client, both implemented within Testbed-17.

6.1. TIE Setup

Common to all tests is that the DCS Server provides the DCS capabilities “on top” of regular OGC API requests by requiring the calling application to submit additional parameters:

- `OAuth2 Bearer Token` as specified in RFC 6750. The bearer token identifies the acting user.
- `key_challenge` as specified for `code_challenge` in RFC 7636
- `key_challenge_method` as specified for `code_challenge_method` in RFC 7636
- `kek_kid` or `kek_uri` as specified in Annex Clause 5.2: A private RSA key previously registered with the KMS.

6.1.1. Client prepares to send request to DCS Server

For Testbed-17, the DCS Client is implemented as a Mobile Application. This is a short summary of the interactions between the client application and the DCS Server:

- Mobile application uses the OAuth2 `Authorization Code Flow` to obtain a Bearer Token from the Authorization Server AUTHENIX.
- Mobile application creates private key for acting user and registers the RSA asymmetric key with the KMS.

- Mobile application asks the user for a `password, pin or secret` to protect the Data Encryption Key. This user input is used by the application as value for `key_challenge` parameter. It is recommended that the application uses `key_challenge_method=S256`.
- Mobile application sends OGC API request to DCS Server accompanied by the specific parameters outlined above. Parameter `f` controls the response format.

The client application must undertake the following steps (simplified summary) to decrypt the response from the DCS Server:

- The DCS Server response contains data and metadata encrypted with different DEKs (Data Encryption Keys). The keys are not included in the response. The client application must process the response to find the key identifier and/or the key URL to fetch the DEKs from the KMS.
- Mobile client starts interaction with the KMS retrieving cryptographic keys required to decrypt previously fetched protected content (binary geospatial data and GeoPackages).
- The resulting data set was retrieved, and the key_id values were extracted from the metadata section of the response.
- Key retrieval is done via the KMS component implemented in Testbed-16. The interaction with its API occurs by passing key identification strings extracted from the metadata section of protected content as well as a `Bearer Token` identifying the acting user and client application.
- Mobile client decrypts the payload and renders the content.

6.1.2. DCS Media Types and OGC APIs

NOTE: A future standardization should fix the media types and have them registered with the appropriate authority — i.e. IANA.

Testbed-17, as an extension to Testbed-16, allows fetching different kinds of DCS content from different OGC API implementations:

Table 2 — OGC API DCS Media Types

OGC-API	DCS MEDIA TYPE	DESCRIPTION	URL EXAMPLE
Features	application/dcs+geo	DCS container structure encrypted features, metadata as JSON	URL_1a
Features	application/dcs+geo;profile=metaSign	DCS container structure encrypted	URL_1b

OGC-API	DCS MEDIA TYPE	DESCRIPTION	URL EXAMPLE
		features, metadata as JWS	
Features	application/dcs+geo;profile=metaEncrypt	DCS container structure encrypted features, metadata as JWE	URL_1c
Features	application/gpkg+dcs	GeoPackage with encrypted features	URL_2
Tiles	application/gpkg+dcs	GeoPackage with encrypted tiles	URL_3a and URL_3b
Maps	application/dcs+{png,jpeg,...}	MultiPart response with encrypted map	URL_4

URL_1a	`https://ogc.secure-dimensions.com/geoserver/ogc/features/collections/tiger:poi/items?access_token=<your_access_token>&key_challenge=secret&key_challenge_method=plain&f=application%2Fdcs%2Bgeo&limit=1`
URL_1b	`https://ogc.secure-dimensions.com/geoserver/ogc/features/collections/tiger:poi/items?access_token=<your_access_token>&key_challenge=secret&key_challenge_method=plain&f=application%2Fdcs%2Bgeo;profile=metaSign&limit=1`
URL_1c	`https://ogc.secure-dimensions.com/geoserver/ogc/features/collections/tiger:poi/items?access_token=<your_access_token>&key_challenge=secret&key_challenge_method=plain&f=application%2Fdcs%2Bgeo;profile=MetaEncrypt&kek_kid=859f22b4-1ce1-42c0-8668-aac789c79242&limit=1`
URL_2	`https://ogc.secure-dimensions.com/geoserver/ogc/features/collections/tiger:poi/items?access_token=<your_access_token>&key_challenge=secret&key_challenge_method=plain&kek_kid=859f22b4-1ce1-42c0-8668-aac789c79242&f=application%2Fgpkg%2Bdcs`
URL_3a	Loading all tiles of a TileMatrix `https://ogc.secure-dimensions.com/geoserver/ogc/tiles/collections/tiger:tiger_roads/tiles/EPSG:4326/EPSG:4326:1?f-tile=image/png&multiTileType=tiles&f=application/gpkg%2Bdcs&key_challenge=secret&key_challenge_method=plain&kek_kid=859f22b4-1ce1-42c0-8668-aac789c79242&access_token=<your_access_token&gt;`

Large size and processing time

Please use this option with care! Encrypting all tiles for a TileMatrix might take a very long processing time and produce large responses!!! For example, changing the URL below to process TileMatrix EPSG:4326:21 produces a 12GB response with processing time of some 6 hours!!!

URL_3b Loading all tiles of a TileMatrixSet `https://ogc.secure-dimensions.com/geoserver/ogc/tiles/collections/tiger:poi/map/point/tiles/EPSG:4326?f-tile=image/png&multiTileType=tiles&f=application/gpkg%2Bdcs&key_challenge=secret&key_challenge_method=plain&access_token=<your_access_token>&kek_kid=859f22b4-1ce1-42c0-8668-aac789c79242`

Large size and processing time

Please use this option with care! Encrypting all tiles for a TileMatrixSet might take a very long processing time and produce large responses!!! E.g. for the example URL below, the f=application/zip produces a 7.5MB file that is returned after some seconds. Using the f=application/gpkg+dcs produces a 92MB response in some 30 seconds.

URL_4 `https://ogc.secure-dimensions.com/geoserver/ogc/maps/collections/tiger:tiger_roads/styles/tiger_roads/map?key_challenge=secret&key_challenge_method=plain&access_token=<your_access_token>&transparent=true&f=application/dcs%2Bpng&width=641&height=768&crs=EPSG%3A4326&width=641&height=768&bbox=-74.01012361049652%2C40.70959210395813%2C-74.00324642658234%2C40.71783185005188`

Use valid Access Token

Please use the Token App to obtain a valid access token and replace the `<access_token>` in the URL accordingly.

6.2. DCS Server Tests

This section introduces tests for validating manually that the DCS Server responses are as expected.

6.2.1. OGC API Features

The DCS Server provides two different response formats, recognized by the `f` parameter of the request: `application/dcs+geo` and `application/gpkg+dcs`.

6.2.1.1. Media Type application/dcs+geo

The `application/dcs+geo` media type returns a JSON encoded DCS container format that was developed during Testbed 16.

The expected response is a JSON encoded DCS container comprised of a `metadata` and a `data` element. The `data` element contains the encrypted feature in JWE encoding with compact serialization. The `metadata` element is JSON encoded.

```
{
    "type": "DCS",
    "objects": [
        {
            "metadata": {
                "originator_confidentiality_label": {
                    "confidentiality_information": {
                        "policy_identifier": "TB17",
                        "classification": "Top Secret"
                    }
                },
                "data_producer": {
                    "origin": "Not NGA",
                    "date": "2021-08-31T10:22:37.537Z"
                },
                "data_description": {
                    "type": "Feature",
                    "properties": {
                        "name": "poi",
                        "namespace": "http://www.census.gov",
                        "content_type": "application/geo+json"
                    },
                    "geometry": {
                        "type": "Point",
                        "coordinates": [
                            -74.0104611,
                            40.70758763
                        ]
                    }
                }
            },
            "data":
 "eyJpc3MiOiJodHRwczpcL1wvb2djLnNlY3VyZS1kaW1lbnNpb25zLmNvbSIsImN0eSI6ImFwcGxpY2F0aW9uXC
.LONNO9mf_oqXGEvY.l3axe9IMILYX-mvIy-3MTUYh_PT-
9KAwAdjXOlFsP9DiDdhbEeaOWOiJiDF30PM6rHw4uA7to-
HYCdusIqNjKU4fQkEKDLxJnE47jlmFS4zWQoIbnWvZpcIr4HD1hmIsa8dsMrccNqifwn_
YpQpSEtQh8H1-4k_GIt7M9lBvQUYVsRcauAJhOXaMMB7L-47_
ZiqRWAW08Bpe3wA49FjbSzbs0oUGgYl6d9O6eG3c5eZEE2tFvQlmfJLLvEi6bMu_
65adbq4KEIBC7UuXHQgi7v-JLzhNIh_5CXDlpqQcT3xz6wno81r-
EnLrqSjNDsLxwLd5Z3Mi3-ZzsDIBiHGG0TlfWFIqKtl6G-
sPdr0DHkpVXYZTh3FOVgejwckXITo3vlaDdVcTUtXvAvxNRdtnUgWEItOOy3vCRggyrFfFlD9YMq5JzjOilCU1JM
m7_riM_p_XOkmrvCCgNbm3LJJADoXc_33S0e-xKHcwZ08PDgqK2H23RHBdGyIgh-
yvcohHBeqc3hQMKcSPMkYb6nPIRVMY6Vk6fUvyYjBZUyP7CMk09DfLjSTBd1qfbjyVQpwgVxdhdHLEiGVJqzK3sa
ziJH4UaPSNmnqDzYhuIBwqOGaM74wIIastKFjvOYnh67XdTKVfFRM7DUqZIjfkH3Yye6YT5fa8Flqzi6xdYhEWP7
QLi6KFoh4cLfUU-Pc_r01HcGI0OGrVs4Xqgv4CP9atlAeGELWIHQnQITpUTioi3Fq6-
AeFuBDcXlD_6k3bVX-nSRHIPqAsMgTpiwM0GZ0LFu2u22Lyq804MJ3MEcA-S6u4vj9ZO40QRbi3-
DwIcSnwYLlCJN20e4p3V_
h25JHOFzgmkBTVNyh2gdVzwHFUW2Km5jrAicJeCWUNp6bR0SflMmhFNVpuiZ6m8cI5qQAPqCsGTIEP334QCrxzbV
RUdS26x5saAv2Zi8APFyn9IrDgo_LXFWXOeyH_yMXfScnU-BcMs0Wc4uh-dHdS7r4.
9NLNpMuwkd2MUINu4VBF2w"
        }
    ],
    "totalObjects": 6,
    "numberMatched": 6,
    "numberReturned": 1,
    "timeStamp": "2021-08-31T10:22:37.541Z",
    "links": [
        {
            "title": "next page",
            "type": "application/dcs+geo",
            "rel": "next",
            "href": "https://ogc.secure-dimensions.com/geoserver/
ogc/features/collections/tiger%3Apoi/items?access_token=
a7b05c5df39c47bb94148a1cdcf081876da54f50&key_challenge=secret&key_
challenge_method=plain&f=application%2Fdcs%2Bgeo&limit=1&bustCache=0.
7411873339750994&startIndex=1"
        }
    ]
}
```

Expected response for URL_1a

The JWE in the `data` element must contain the DEK's `kid`.

```
{
  "iss": "https://ogc.secure-dimensions.com",
  "cty": "application/dcs+geo",
  "enc": "A256GCM",
  "alg": "dir",
  "kurl": "https://ogc.secure-dimensions.com/kms/keys/4c98fa0a-a212-4d6a-8ced-
d8cb5d6b2e7e",
  "kid": "4c98fa0a-a212-4d6a-8ced-d8cb5d6b2e7e"
}
```

Base 64 decoded JWE header

6.2.1.2. Media Type application/gpkg+dcs;profile=metaSign

The expected response is a JSON encoded DCS container comprised of a `metadata` and a `data` element. The `data` element contains the encrypted feature in JWE encoding with compact serialization. The `data_description` element of the metadata element is JWS encoded with compact serialization.

```
{
    "type": "DCS",
    "objects": [
        {
            "metadata": {
                "originator_confidentiality_label": {
                    "confidentiality_information": {
                        "policy_identifier": "TB17",
                        "classification": "Top Secret"
                    }
                },
                "data_producer": {
                    "origin": "Not NGA",
                    "date": "2021-08-31T10:33:42.957Z"
                },
                "data_description":
 "eyJqa3UiOiJodHRwczpcL1wvb2djLnNlY3VyZS1kaW1lbnNpb25zLmNvbVwvZGNzXC8ud2VsbC1rbm93blwvan
eyJ0eXBlIjoiRmVhdHVyZSIsInByb3BlcnRpZXMiOnsibmFtZSI6InBvaSIsIm5hbWVzcGFjZSI6Imh0dHA6Ly93
eQKyi9iy54OdUE6UghkAn5_51SYoELW2PaSnmECu7Kcb-gejQpd5KA5n1UplDqSJK-tq-
xwOv1JqKI6mACPoD0DBZws296e1cRUMTv4vQt54BGJarJ3CCip7YMCbyYB1dUCRJi5Mr-
wgw6JAdJpVBMCMmn4fIzvGLo3FxhzghoRsAEvte5vGOEBnG-K4SDrqvDIVDAlCHr_
uibPNV3SnlZQZhY_
R0iCjGiP2X5uOj9FXL3JFgDuteHoyLoiIfhH5yYosmJygAOgJeQbBKHZ3TR97sire4cq84Y51MP9eCd7-
9wyoRzQISXbiUF22O0cmqtp-eX3NfiL3tam8RrA5LQ"
            },
            "data":
 "eyJpc3MiOiJodHRwczpcL1wvb2djLnNlY3VyZS1kaW1lbnNpb25zLmNvbSIsImN0eSI6ImFwcGxpY2F0aW9uXC
.yulhMbph5H97cOg1.cGRL7hAq1_-oyZvJ-erDcqtjluo-lgO6XcqHwLYVVFx-
0HNd2fYmF4chihmUX2bgkMc_cNJ5FaiTuQXhQVkMmlfAyw2mRK6vAkEJO0X9jCQytbycR_
lmG7hgiQF1MtXZEocyP73g748D-
E38GLkJkEJHO0jKe7nOeBynHJjkN2zQD0lqznj4d5tMCpUxh71S13Bz3UDIGxdY-
2QOZw9v7hJCnpRBvGKeMNZamgcK21LXtBBN3pfvsApZBsUvpW8ELNjyQlcZGSVD_
QLwXWvoVXDKlcSep2pEltz5YVwgnhXasmBqjr-eq9rPU-
4Aah9Q5F9CFFKGTpZtAS85lLqeNCBYAZUXkmYpX31cXZhbRTLuJfww3Sm8KAZGj4NxVV_
aEPbNiQM7XaZMSRXPqN8tV2h0htU1Vs7XgZGnHRnoNB8gQHgynmi7gGlM49K35lWpP0pn_
AKTS2lTRyCj9jkOoFZfxwv5G512vLzxtg5untsvyhUFHGOtF8vPPicG-
hSTroO7qqbVbvPmVN5sZ1skuhyM18Axr1-
GGwVb1QHyjH8I2QO7JBBmvxQ2led7nZPs6GbEkbv8y9YS1jY8A4DtTE767irKNvYEOBYM2ZeE3QMRQIw_
d2LKm5XVOeRMIiXVOIr42MaAuo1f6QuE61ARmUg8oc_e9GBkIicMeoHLRu__
VGUnoFjND6eCAuVo6gyz0fcDdTtCY9oQxVMgqy-xjEP4SEyOoj63d5rObUKdAgmFn9A6H5FF_
97s5_u1KlXK8vBO0-mcpqFwgBF-UrDshoswTSVTjAx52Z_1Vc_-kUnUHBLz_
WHfA9qaC2ljvy9XSWuFAWfvODuhSeDycJW6_VWJq8ueTCWwDjE2wD4oi6dsJ1jNSaHzxvykerR-
5Hd1_B1xfD3hg7RpCyw07zKnbr2X8PZBRK51D0vLU-KyLRbQXMfW-j6RbLCtOS_9_
wcGxZmsjOsQyy9_2dONzx5ojxyiUsLjsOgqtB0gnx8M_mZL-QS34fwSs5MT-IZNdm4IkO60j-iw-
IQGBEjtO8YhThzKfilOikLpTM7KdAydlzFvlKcYqNTbAE8HbXQSxTKZ2fNi8DhVzucRgTTvOlQ3GpeDYPtwo2YLN
KW5Q9vv8rprFTY.cEe9OC7lP3qlvc_q3WZAaQ"
        }
    ],
    "totalObjects": 6,
    "numberMatched": 6,
    "numberReturned": 1,
    "timeStamp": "2021-08-31T10:33:42.961Z",
    "links": [
        {
            "title": "next page",
            "type": "application/dcs+geo",
            "rel": "next",
            "href": "https://ogc.secure-dimensions.com/geoserver/
ogc/features/collections/tiger%3Apoi/items?access_token=
a7b05c5df39c47bb94148a1cdcf081876da54f50&key_challenge=secret&key_challenge_
method=plain&f=application%2Fdcs%2Bgeo%3Bprofile%3DmetaSign&limit=1&bustCache=
0.7067005135391409&startIndex=1"
        }
    ]
}
```

Expected Response for URL_1b

The JWE in the `data` element must contain the DEK's `kid`.

```
{
  "iss": "https://ogc.secure-dimensions.com",
  "cty": "application/dcs+geo",
  "enc": "A256GCM",
  "alg": "dir",
  "kurl": "https://ogc.secure-dimensions.com/kms/keys/2844436e-c36a-44fb-84c3-
43f02989ee7e",
  "kid": "2844436e-c36a-44fb-84c3-43f02989ee7e"
}
```

Base 64 decoded JWE header

The JWS in the `data_description` element must specify the public key reference (`jku`) of the issuer's public key and the `kid`.

```
{
  "jku": "https://ogc.secure-dimensions.com/dcs/.well-known/jwks.json",
  "kid": "Dr. No",
  "alg": "RS256"
}
```

Base 64 decoded JWS header

6.2.1.3. Media Type application/gpkg+dcs;profile=metaEncrypt

The expected response is a JSON encoded DCS container comprised of a `metadata` and a `data` element. The `data` element contains the encrypted feature in JWE encoding with compact serialization. The `data_description` element of the metadata element is JWE encoded with compact serialization.

```
{
    "type": "DCS",
    "objects": [
        {
            "metadata": {
                "originator_confidentiality_label": {
                    "confidentiality_information": {
                        "policy_identifier": "TB17",
                        "classification": "Top Secret"
                    }
                },
                "data_producer": {
                    "origin": "Not NGA",
                    "date": "2021-08-31T10:43:16.005Z"
                },
                "data_description":
 "eyJraWQiOiI4NTlmMjJiNC0xY2UxLTQyYzAtODY2OC1hYWM3ODljNzkyNDIiLCJjdHkiOiJKV0UiLCJlbmMiOi
Xyyi-3P38QPvtTS5z7gQ_II_Br88uXHWLzn5zoCYFEYXljBp73v3syBUvpRRh6N_
ihZLlH8rAI4aJ34pEDS9vn4ufdsp7wbJdSXojT0d-YA1-LZ3VtvUiz5VeGyLKD5njqaGoGm_xXc-
hM8L4PTbIlgTul_
h2OWbyAEGgeRIGpcS5dqLEjb8sA8E0bZF4hsugGKOCvv0nyciWOVzH1VzSlepUa1DerqY7uKtLCwdO7QeUnjZQUG
y8Ui6ZyEn_NQDvPymilJK5kGfNFGO807GHIjJToTFOCD0X_Huew4x8pHCKcj5orMMfUr-nmTaL6Hw.
kX3q83LN7MVAw4oq.hacw3EBp0mY3rCfoyBR5bWCAW_YsA_IujwIMZwPhpyfHrCeTufZWU-
rgv1kPsT10ZmQZ67RuZN7S-ci3B7d9Mv2hr74TUOpzgrKMsaT4ow4jOiE-
rAJj48dOb1JkVEd9QbpDPpZH5xO4H4br0AKsO6kQh68Wa_
wQOvlsRbAWu99ImuUpZ0XswoAulgmDPdOOBHPrBbEDurH4o_
s7ih84KsZa5X8OOzDetnJOfzQsTVesvRMi3VltfX-
Bkyape1dM8Dxddzzf7429oHWVxEnsJ2QQBJq1hexYSnVeHNk04KqN_
BrzvDrYeR2B0Yin6oxEAFbxlFsBjc10kNoB1rGk-9DcirmeLI9Pn9HlTgl_Dz-
yOMX5P9h4tjEN7NnlU7YS1hU7YPJjIpKKgoff608E8KTDI0pyBnF8lkGnug4zt3mY7Qw1KMUPydvfe09lOJpXAiy
92i4XwaHXdNnx3NQSjjD1VK4RoLUgBn4iIBmEMPPILuCc8Sg_
rv5qnNhaMpLkyU5hMvzFcaLH9nZRGWO-iM7-uZ_rfA5oBPSpLYeROtjUQ3FIClGr0zFnuo-
psphlpoqo3kDw5ZjB6QgrOOLQZmDc-
yADGVDMXwDlucCWmIwOmEzDW1f84htkSdhdePH9SKYC1b54khXiUD4621txd8nJ7FZfuM2dgn_
nHETYYJs36hYbwunXRnHia7vlCdZxxOefN7uE_
WFVZ1suGC7v06ajXei2NSyV2ty6FVV72o2L5DcnglJ7rVMOxLkuz8oesx363m_
b8eAm0pK_f3RdBVQ66bIF_EIUGYQ9udQMEJ0SgQO19pDGBMWKTMhQ5qXm-6t0Sa92qb6u_
JIuchKBCcvn6RjFe0JV_H7-JaNkRfWV6PG9EgcyxOf30ZjoCYm66rJ-JWiMM9IiF03g4UA_SP9B-
t8AUzbL0UUIK5Tg.bdy026MSYkNWFKIEAB74bw"
            },
            "data":
 "eyJpc3MiOiJodHRwczpcL1wvb2djLnNlY3VyZS1kaW1lbnNpb25zLmNvbSIsImN0eSI6ImFwcGxpY2F0aW9uXC
.x4HVYSDjm6CqAVCd.xoaVF6G7aK4jMGkw9YFW_osVDlnB92eEBgqVVdsnwGE1-
aHeVBas-xtnlptpN2pT2u59eG5Y9TZ9HerhlnClKkJyf3WVApkJXdGGbedsEefjIuaiJf-
yYe6wYHXr-ZBq_ANHVRvbAru2pO2aZj1quvLhuG3YGt-pCX0IZ7R9c9E5l-
MYxkU5jZtAJ8gQIHnkc7ZRrZls-VdwGrT7ogq0CVG4S_VG6vYwSh_
iRdbY17n749DgISxSWh-n3sQ8o_Xp3OveeYrJjdxXYOE3ThLbRJHkgzZ-
57VH7xL0kk3QJoGrJ0g8m6vrRcVY5EnUAXkUDvjtNfcRBnoFMf6BGDg5jFPfM3iMy3k1n8yXWKQ_
fKsIdSjEkqEkE-YM-9pqH8qYI7ShC_wnlkyBaZI38cXA1FaUxj3D3O3uTEXmO7tyrBgMX_
1R6NEQDyVRBZntOMhp0xyBprTxmG7823Nz1hgRQDd6rqqPPULqWIaq7NMXjc5nEyaMSkgU4cPgyRa7i3rPjdbNzt
k57D8tnJmsdEICV0VAXPukUCPh3hHtYnuvC0StFhcNJ9xmeI4hwDYxFPnvFUoIhUPJcv9jUXsNYk4QiX6b9dytLH
1ncMgJoI5sT3wzu6sdHM9-RBt2VXU2QFhRnKWLhSK_t1Zg8VcY29c-
dHdSuTq9KQuzAtpCG0HRLr27ZZO3SLLUL1BftRLSu-
LxufthohDmg5gaB4WapgF2poevGDyzpPTYZ3GVuENWkTQdamCI1Wgu2lrKifgbaLTyWJpxx-
nzyRNce9W7UWhRqGSd_
diEqznEbgFfGJCqZOPpxm6jN7BvfWHqDI9F2t8G7GCrLAl55lh9Q8NOWvxFLiCCrc4m6IJQqjNQNWefLnjN7TLAj
SA4Xy9hz50KJPNIsEdGzotW_3a76CXd_C6hjs2uThersj5G3Kpt4Y-
fgJv7g4oD5L8YiXK0264lr_Qy3lZbMUugRgO-MWWImbZ_i8D_
JOYKIbn4Uoihyg5QHBh9t4QRRRYMDRLVdycpQOwVeNU9spSUtG0xW0.g-nV2GyDYTNzTH0qLuBEQQ"
        }
    ],
    "totalObjects": 6,
    "numberMatched": 6,
    "numberReturned": 1,
    "timeStamp": "2021-08-31T10:43:16.009Z",
    "links": [
        {
            "title": "next page",
            "type": "application/dcs+geo",
            "rel": "next",
            "href": "https://ogc.secure-dimensions.com/geoserver/
ogc/features/collections/tiger%3Apoi/items?access_token=
a7b05c5df39c47bb94148a1cdcf081876da54f50&key_challenge=secret&key_challenge_
method=plain&f=application%2Fdcs%2Bgeo%3Bprofile%3DmetaEncrypt&kek_
kid=859f22b4-1ce1-42c0-8668-aac789c79242&limit=1&bustCache=0.
5590496976910175&startIndex=1"
        }
    ]
}
```

Expected Response for URL_1c

The JWE in the `data` element must contain the DEK's `kid`.

```
{
  "iss": "https://ogc.secure-dimensions.com",
  "cty": "application/dcs+geo",
  "enc": "A256GCM",
  "alg": "dir",
  "kurl": "https://ogc.secure-dimensions.com/kms/keys/93df27c6-5174-4b1d-b652-
704a42d4df31",
  "kid": "93df27c6-5174-4b1d-b652-704a42d4df31"
}
```

Base 64 decoded JWE header

The JWS in the `data_description` element must contain the KEK's `kid`. The value of the `kid` must be equivalent to the value of the request parameter `kek_id`. The private key, associated to the public key, must be used to decrypt the content.

```
{
  "kid": "859f22b4-1ce1-42c0-8668-aac789c79242",
  "cty": "JWE",
  "enc": "A256GCM",
  "alg": "RSA-OAEP-256"
}
```

Base 64 decoded JWS header

6.2.1.4. Media Type application/gpkg+dcs

The `application/gpkg+dcs` media type returns a GeoPackage extension that stores encrypted features as described in Annex A — GeoPackage DCS Features Extension.

6.2.2. OGC API Tiles

The DCS Server provides one response format, recognized by the `f` parameter of the request: `application/gpkg+dcs`. The actual image format of the tile is controlled via the `f-tile` parameter. Valid values for the `f-tile` parameter depend on the actual data deployment; most often `image/png` and `image/jpeg` are supported.

NOTE: For simplification, the TIEs and tests use the `image/png` format only.

6.2.2.1. Media Type application/gpkg+dcs

The `application/gpkg+dcs` media type in combination with `image/png` returns a GeoPackage container that contains all requested tiles honoring the GeoPackage extension defined in Annex Annex A — GeoPackage DCS Tiles Extension.

6.2.3. OGC API Maps

The DCS Server provides one response format, recognized by the `f` parameter of the request: `application/dcs+png` or `application/dcs+jpeg`.

NOTE: For simplification, the TIEs and tests use the `PNG` format only.

6.2.3.1. Media Type application/dcs+png

The `application/dcs+png` media type must return a HTTP Multipart response, where the first part contains the JSON encoded metadata and the second part the encrypted image as octet-stream. The response `content-type` must be `multipart/encrypted` denoting as a profile the `protocol` format that provides the metadata for the encrypted binary data.

```
Message-ID: 35d58964-dd70-4c99-b163-d4de161a48a1
Content-Type: multipart/encrypted; protocol="application/json"; boundary=
ce5b9207-6e4b-4ba0-8632-9964aea0b927
Date: 2021-08-31T12:41:33+0100

--ce5b9207-6e4b-4ba0-8632-9964aea0b927
Content-Disposition: inline
Content-Type: application/json
Content-Length: 1420

{"metadata":{"originator_confidentiality_label":{"confidentiality_information":
{"policy_identifier":"TB17","classification":"Confidential"}},"data_
producer":{"origin":"Not NGA","date":"2021-08-31T11:41:33.062Z"},"data_
description":{"type":"Feature","properties":{"name":"tiger:tiger_
roads","content_type":"application/geo+json"},"geometry":{"type":
"Polygon","coordinates":[[[-74.02722,40.684221],[-73.907005,40.684221],[-73.
907005,40.878178],[-74.02722,40.878178],[-74.02722,40.684221]]]}}},"dek_info":
"eyJqa3UiOiJodHRwczpcL1wvb2djLnNlY3VyZS1kaW1lbnNpb25zLmNvbVwvZGNzXC8ud2VsbC1rbm93blwvand
eyJzdWIiOiJmZjEwNDVjMi1hNmRlLTMxYWQtOGViMi0yYmUxMDRmZTI3ZWEiLCJhdWQiOiIwMTliNzE3My1hOWVk
gFENFhsCc61ZbX4phIiUFXtMp7fWJigILbzUyfaSZaPfA1k-
CuuHX975Ot4I3dLxvOLxerOlxIfUNnKJ9jHcZZUpxnZ5X0nyeYKt563ipKYnmCxptGz4C85dFskOvvRYHW1e4EHz
JP6lIofq_NeDklhYhq_
LSQnFwYzSFnowAQZqlRswR6aUGeGFgPG2APX3J1Otx9vLJcQh0UxQG6O2gFxGyX54sv1z5mlqb3fUyjm5uJxJqfR
--ce5b9207-6e4b-4ba0-8632-9964aea0b927
Content-Disposition: inline
Content-Type: application/octet-stream

t######e#####u##3
--ce5b9207-6e4b-4ba0-8632-9964aea0b927
```

Expected Response for URL_1c

The metadata returned as the first part must include a `dek_info` element that contains a JWT defining the issuer of the response as well as DEK information.

```
{
  "jku": "https://ogc.secure-dimensions.com/dcs/.well-known/jwks.json",
  "kid": "Dr. No",
  "alg": "RS256"
}
```

Base 64 decoded JWT header

```
{
  "sub": "ff1045c2-a6de-31ad-8eb2-2be104fe27ea",
  "aud": "019b7173-a9ed-7d9a-70d3-9502ad7c0575",
  "kurl": "https://ogc.secure-dimensions.com/kms/keys/a2e43d1f-3761-452b-b55a-
9f5f68304afe",
  "kid": "a2e43d1f-3761-452b-b55a-9f5f68304afe",
  "iss": "https://ogc.secure-dimensions.com",
  "exp": 1630411,
  "alg": "A128GCM",
  "iat": 1630410091822
}
```

Base 64 decoded JWT payload

The second part must use `Content-Disposition: inline` and `Content-Type: application/octet-stream`. Using the `kid=a2e43d1f-3761-452b-b55a-9f5f68304afe` must allow to decrypt part two of the response.

6.3. Provisioning and portrayal of DCS protected geospatial binary content

The DCS enablement requires an extension to a standard OGC API request with additional query parameters. Depending on the API and the response format, different additional parameters are mandatory.

The following table illustrates which parameters are required:

Table 3 — DCS parameters (m=mandatory, o=optional)

OGC-API	F	ACCESS_ TOKEN	KEY_ CHALLENGE	KEY_ CHALLENGE_ METHOD	KEK_ ID XOR KEK_ URI
Features	application/dcs+geo	m	m	m	o
Features	application/dcs+geo;profile=metaSign	m	m	m	o
Features	application/dcs+geo;profile=metaEncrypt	m	m	m	m
Features	application/gpkg+dcs	m	m	m	m
Maps	application/dcs+png	m	m	m	o
Maps	application/dcs+jpeg	m	m	m	o
Tiles	application/gpkg+dcs	m	m	m	m

6.4. TIE Results

TIEs are conducted between the DCS Client Application (Mobile App) and the DCS Server. A successful TIE requires that the DCS Client and the DCS Server interact with the Authorization Server (AUTHENIX) and the Key Management System (KMS).

6.4.1. DCS Client TIE

As illustrated in an earlier section, the DCS Client Application must obtain a valid OAuth2 Bearer Token from the Authorization Server. Depending on the requested response format (profile=metaEncrypt), the Client Application must register the user's public key (once) with the KMS. Because a public key (for this Testbed) does not expire, the Client Application can re-use a previously registered public key, but still the one-time registration is required.

Therefore, the TIE for the DCS Client is split into these individual TIEs:

- Interaction with the Authorization Server
- Interaction with the Key Management System
  - for registration of user's public key
  - for fetching DEK(s) to decrypt the response
- Interaction with the DCS Server

A final TIE determines whether the Client Application is capable of decrypting the response from the DCS Server and is able to render and display the geospatial content.

Table 4 — DCS Client TIEs

	STEP	STATUS
1.	Interaction with the Authorization Server	✓
	Interaction with the Key Management System	
2a.	for registration of user's public key	✓
2b.	for fetching DEK(s) to decrypt the response	✓
3.	Interaction with the DCS Server	✓

6.4.2. DCS Server TIE

The DCS Server must accept an OGC API DCS request as defined above. A valid request must produce the expected result as outlined above. In order to produce the response, the DCS Server must interact with the Authorization Server and the Key Management System.

Therefore, the TIE for the DCS Server can be separated into these individual TIEs:

- Interaction with the Authorization Server (for the purpose of validating Bearer Tokens — aka `access_token`)
- Interacting with the KMS to register generated DEKs

- Interacting with the KMS to fetch public key (determined by the `kek_kid` or `kek_uri` parameter)

Table 5 — DCS Server TIEs

	STEP	STATUS
1.	Interaction with the Authorization Server	✓
	Interaction with the Key Management System	
2a.	for fetching the user’s public key	✓
2b.	for registration of the DEK(s) used for encryption	✓

6.5. TIE Summary

A successful TIE can be determined by the ability of the DCS Client to query, decrypt (and display) the encrypted DCS response from the DCS Server. When that is the case, all outlined interactions must have worked as a whole.

Table 6 — TIE Summary for DCS Client and Server

OGC-API	F	TIE RESULT
Features	application/dcs+geo	✓
Features	application/dcs+geo;profile=metaSign	✓
Features	application/dcs+geo;profile=metaEncrypt	✓
Features	application/gpkg+dcs	✓
Maps	application/dcs+png	✓
Maps	application/dcs+jpeg	✓
Tiles	application/gpkg+dcs	✓

The **successful** TIEs were conducted between DCS Client Application and DCS Server.

7

TOWARDS DATA CENTRIC SECURITY AND FEDERATED SECURITY

7 TOWARDS DATA CENTRIC SECURITY AND FEDERATED SECURITY

NOTE: The goal of this section is to analyze the gaps between the DCS architecture and implementation solution developed in Testbed-17 towards its use in Federated Security.

During OGC Testbed 15-17 Data Centric Security activities, an architecture was developed that supports adding security when requesting geospatial data using one or more OGC APIs. The components of the architecture, as illustrated in Figure 1, require having access to a common (shared) security context. For the implemented solution, the exchange of the security context is coupled with the Bearer Access Token, created with a single Authorization Server. For a single security domain, as defined by the Authorization Server, this is common practice. Different major solutions in mainstream IT function exactly like this: Resources are protected by Security Proxies that check OAuth2 tokens for validity, user and audience conditions. With the extension of OAuth2 based on OpenID Connect, hybrid flows are used that also allow exchanging personal information as part of the security context. However, these solutions do **not** allow, for example, a user that has authenticated with Google to access Facebook resources. The user must authenticate with Facebook which will then give the application a Facebook token.

With Federated Security, users from different security domains are able to act on resources provided by operators from different security domains with applications from different security domains. As outlined in the Federated Security ER, different approaches exist to establish trust between applications, services and actors from different security domains.

This section analyses gaps between the existing DCS architecture and functioning in a Federated Security setup.

7.1. Overview

So far, the DCS solutions developed in Testbed 15-17 were demonstrated inside of an individual "Administrative Domain" (AD) that essentially comprised of:

- An Authorization Server that connects to an Identity Provider (IdP).
- A (Cloud) Service Provider (SP).
- A (mobile) client application.
- A user authenticated within the AD.

The relevant context of Federated Security is described in different (complex) publications:

- https://nvlpubs.nist.gov/nistpubs/SpecialPublications/NIST.SP.500-332.pdf

- http://docs.opengeospatial.org/per/20-027.html

The target architecture for supporting DCS with Federated Security is based on the Zero Trust Domain definition, where, in contrast to perimeter-based security, each interaction requires authentication and is subject to access control; literally trust nobody — verify all (users). With such a target architecture, where all interactions between a user-controlled application and secured services require authentication, there needs to be a solution for how to trust users that are authenticated in another domain. This implies the first topic to be analyzed: How can the existing DCS architecture be extended to support federated authentication?

As the Zero Trust can be coined to be data centric — each data flow is subject to being controlled — the question arises of how to establish such a control mechanism for encrypted data. With encrypted data, the typical access control approach no longer works, as the encrypted data must be treated as a BLOB or a black box. With data in the clear, the data itself can be used to derive fine grained access control, as it is possible with OASIS's XACML or OGC's GeoXACML. This challenge can be identified as the second important topic to be analyzed: How can access to encrypted (geospatial) data be controlled towards users authenticated in their own domains, as well as authenticated in another (federated) domain?

The overall question to analyze can be summarized as follows: "Is it actually possible to extend a single-domain DCS architecture to function equivalent in a Federated Security environment? Or, must limitations / constraints be defined that reduce the flexibility of the architecture with transiting towards federated support?

7.2. Implications to Authentication

The existing DCS architecture causes all authentication to be performed in one single Authorization Server. Even though the Authorization Server in place (AUTHENIX) is based on federated identity management — it allows users to login with accounts from multiple domains — the Bearer Access Token linked to the user is validated with AUTHENIX. Also, the personal data that is linked to the access token can only be fetched from AUTHENIX. Extending the architecture from a single domain to multiple domains implies that either all domains trust the authentication to be done by AUTHENIX or that each domain operates their own Authentication Service.

At this point, the option that one Authorization Server operates as the only trusted authentication service can be ignored. So, assuming that each (Zero Trust) domain will operate their own Authorization Server, what implications to the remaining architecture exist? Each service in a Zero Trust Domain environment that provides access to data must be able to validate the (claimed) identity of the acting user.

One typical solution to extend authentication from one single to a federated infrastructure can be based on digitally signed assertions. These assertions can be either authentication, authorization or other security context statements. The OASIS Security Assertion Markup Language (SAML) is, for example, the backbone of federated identity management for eduGAIN, the federation of many academic institutions around the globe. In a nutshell, the setup of federated identity management requires to roll-out a PKI where typically Certificates vouch for the identity of an assertion. The assertion is trusted by the receiving party, if the digital signature can be verified to be from one of the trusted domains. Each domain operating an Identity Provider must exchange their X.509 certificate in advance. Applying such a concept to the existing DCS solution, the use of Bearer access tokens could be changed to Digitally Signed access tokens. The use of JWT, as described in JSON Web Token (JWT) is a possibility. Establishing a trust federation of Authorization Servers and what it all needs is described in OpenID Connect Federation 1.0 — draft 17. Comparing the approach with SAML, it seems that the principles are somehow identical, but the protocol is more modern: JSON instead of XML.

Another interesting effort is the one regarding W3C Secure Payment Confirmation where the objective is *"... to scale authentication across merchants, to be used within a wide range of authentication protocols, and to produce cryptographic evidence that the user has confirmed transaction details."*

Any extension from one single authentication source to a federated variation has direct implications to access control.

7.3. Implications to Access Control

When users reside in a single security domain, the authentication process can ensure that all user attributes become available that are required to undertake access control. However, when extending authentication into a federation, the access control decision, derived locally, might lack required user attributes. The cause of that could be that either the required attribute is not available at the other authentication service or that the attribute is simply not released to another domain. An example for the first case can be found when using the Testbed-17 Authorization Server and the login via the OGC Portal IdP. An authenticated user from OGC will have the OGC related claim `ogc-is-member-of` which is part of access control in Testbed-17. This claim lists all the user's active OGC projects (DWG, SWG and project membership). When the same user uses Facebook, Google, or a University login, this particular claim is not available. Examples for withheld attributes can be found in the intelligence but also eduGAIN domain: user attributes that are considered sensitive (e.g. military ranking, personal information stored at a University) may not be released to other domains.

The other implication / complication to access control is the harmonization of access rights and their interoperable exchange. Based on the fact that in each Zero Trust Domain access control is done in isolation from other domains, how can it be ensured that one user gets access granted to a composite of services as required. As an example, a disaster management first responder requires access to information that is provided by services from different (Zero Trust) domains. There must be some (pseudonymize) unique identifier of the user that allows binding access rights in the local domain. This access control management objective has implication to authentication: All users — regardless of origin domain — must be identified via a globally unique identifier (unique and persistent across all acting authorization servers).

Assuming that each user can be uniquely identified, it is still possible that the user gets an HTTP status code 403 meaning access to the requested resource was forbidden. As the user is stuck in security, this message may not be particularly helpful. The resulting challenge is how to help the user or the application to "understand" what the issue is. This opens the challenge to know how much information can be exposed securely; remember: a Zero Trust Domain means “don’t trust anybody”. This implies that there won’t be any information available to help the user, as they are not trusted (and could be an adversary). But still, there must be some mechanism or protocol that helps to “bootstrap” a solution, assuming there was a legitimate request issued in the first place. Further analysis is required towards the question: How to determine the trustworthiness of a user in particular cross domain.

Another daring challenge is the exchange of access conditions among administrators of different domains for the purpose of harmonizing the access rights of users. The assumption is that the actual access control realization in each domain is done specifically, honoring different facets specific to the domain. But how can access conditions be exchanged with other domain admins to ensure that access requirements can be managed? Further analysis should determine how far the use of Access Control Markup Languages like XACML or GeoXACML help to (automatically) harmonize access conditions across domains.

Extending access control to encrypted data has implications for Key Management.

7.4. Implications for Key Management

During OGC Testbed 16, a Key Management System (KMS) was developed to protect the data decryption keys used to encrypt geospatial data. The functionality of the KMS was inspired by the NIST Special Publication 800-57 Part 1, Revision 5, Recommendation for Key Management: Part 1 – General publication. In particular the key’s lifecycle and the separation of data encryption and key encryption keys was implemented. A Data Encryption Key (DEK) is the actual cipher key that scrambles the data. The possession of DEK enables decrypting the actual data. A Key Encryption Key (KEK) is a cipher key that is used to encrypt a DEK for storage or transit. Typically, a DEK is a symmetric key (shared secret) and a KEK is an asymmetric key (private — public key pair). In that regard, the use of the keys is different and so are implemented options for constraining access to the keys. Regardless if DEK or KEK, the key registration requires the user to authenticate. The registration of the KEK is limited to public keys. The assumption is that the matching private key is securely stored with the owning user. The fetching of the public KEK is open to all users as encryption based on the public key can only be decrypted by the associated private key. Further, per definition, their key is only in permission of the user. However, to prevent misuse in case the private key is leaked, the KMS supports the owner of the KEK deleting the key.

The KMS allows the owning user to apply fine grained access control to a DEK, as illustrated in Clause 5.8. However, the access control is not tied to the actual content that originally existed before encryption. For example, constraining access to encrypted information based on `feature_type` or `classification` is not possible via the KMS. Further analysis is required to determine how far the encryption process itself can reflect the original data’s characteristics. For example, the Testbed-17 DCS Server uses different cipher algorithms for data that is (artificially marked as CLASSIFIED, SECRET, TOP SECRET). However, the KMS does not constrain access based on the key algorithm. So, what seems to be missing is metadata on the usage of the key and which characteristics of the encrypted data exist that should be considered for access control to the DEK.

What are the implications imposed by federated authentication? The Testbed KMS relies on authentication from the single Authorization Server. Sharing of encrypted data, based on sharing access to the associated DEK can be based on the user's email or unique identifier. But what is the user's email or identifier in a federated setup? How does the KMS know which Authorization Server to contact if users from multiple domains are considered?

Assuming that each Zero Trust Domain wants to operate their own DC Server, what implications to the protocol between the DC Server and the KMS exist? What about the protocol between the DCS Client application and the KMS?

7.5. Implications to the DCS Server

The Testbed-17 DCS Server offers requesting encrypted (geospatial) data hosted on a GeoServer implementation of the OGC Maps, Features and Tiles APIs. Regarding the encryption key, the DCS generates a DEK on behalf of the user. However, different options are possible:

a) The user requests encrypted content but leaves it to the DCS Server to generate the encryption key (DEK) on behalf of the user;

b) The user generates the encryption key (DEK), registers it with the KMS and submits the key ID with the request.

c) The user generates the encryption key and submits it with the request to the DCS.

For the first option, where the DCS Server generates the DEK on behalf of the user, the question is which KMS to use for key registration. To enable the DCS Server to register a key on the user's behalf requires that the DCS server has sufficient access rights on the KMS. In order to refrain from complex access condition management across domains, perhaps the DCS Server should always register the DEK with the own KMS — the KMS operated in the same security domain.

The second option implies that the DCS Server can resolve the actual encryption key from the KMS used by the user. When not constraining which KMS that user could have registered the key, the DCS Server needs sufficient access to fetch the key. But, would a KMS release a symmetric key to a DCS Server? Assuming that a DEK is a symmetric key, it could also be used to decrypt data. So, a copy of the key stored at the KMS or exposed via an attack might jeopardize all data previously encrypted with that key. Perhaps a DEK never crosses security domains? Perhaps a DEK only can only be fetched by trusted applications? If so, how could applications verified to be trusted?

Regarding the third option, the DEK must be protected while in transit (the request is sent via the network). Even though leveraging HTTPS does protect information sent through the TLS channel, it leaves the URL in the clear. Therefore, submitting the DEK with a HTTP GET is "unwise", even when the key is protected by a KEK. This brings an implication to OGC API standardization: OGC API Standards should support HTTP POST as an alternative to HTTP GET. Even though the use of HTTP POST over TLS would secure the DEK while in transit, but leveraging the transport level security mechanism is still against the philosophy of DCS: Apply security to the data — to the DEK in this case. Investigation of options applicable to protect the DEK when being submitted with the OGC API request is recommended. Is it possible / sufficient if the DEK is encrypted based on the public key of the DCS Server?

Would it be “wise” that the DCS Server accepts the (encrypted) DEK with the request? If so, would the DCS Server need to validate that the DEK actually belongs to the acting user? If not, a man in the middle attack might have replaced the encryption key with the own. It should be analyzed how a DEK’s origin (issuer and owner) can be verified in a federated environment.

7.6. Implications on the DCS Client

The DCS Client for Testbed-17 implements the OAuth2 / OpenID Connect hybrid flow to obtain a Bearer access token from the single Authorization Server for the acting user. This access token is sent with each request to the DCS Server and KMS.

The implications of extending the single domain solution to a federated “mash” of Authorization Servers would impact the client application. The application would need to know which Authorization Server to contact for login. This complexity is currently proxied by AUTHENIX as it offers the login provider discovery (IdP Discovery). Would it make sense to implement such a complex and trust intense protocol as the IdP discovery into a client application? It is possible / recommended in the first place to think of Web-based, desktop, mobile and server-side applications.

When leveraging multiple Authorization Servers, how would the client application know which access token needs to send to which service endpoint? The proper matching would require that the client application knows which service accepts tokens from what Authorization Server. Is it wise to disclose all that information to the client application? Is this feasible?

What other approaches exist to lift the burden of access token and Authorization Server / Resource Server linking management off the client application?

For decrypting data, the client application must get access to the KMS. How can a key owner associate access conditions to the client application; it might have different identities with different Authorization Servers. How to not over-complicate access condition management? The more difficult the access condition management to keys get, the more difficult the validation of the effective conditions becomes.

7.7. Implications to Security

The transition from a single source solution to a solution that works in a federated environment potentially introduces protocol vulnerabilities. How can vulnerabilities in protocols be identified to prevent or mitigate successfully attacks?

Potential security hazards must be considered when defining an interoperable protocol for a federated security DCS solution.

8

FUTURE WORK

8 FUTURE WORK

The following topics — more or less closely related to DCS — represent recommendations for future work.

8.1. Streaming-Ready Container Format

The GeoPackage container is a capable data storage format, but should not be created at the general user's request. Further, when making a GeoPackage container available as an output format for a service on the Internet, establishing a resource quota is recommended.

Alternative encodings like ZIP are considered streaming-ready and do not produce a storage burden on the server. Future investigation into how far the use of a ZIP container format can be used as an alternative to GeoPackage should be considered. Perhaps the SQLite 3 ZIP extension can be used as a start. When doing this investigation, it should also be evaluated whether other alternative storage structures allow to create, store and distribute large amounts of multi-part, encrypted geospatial data in a more efficient way.

8.2. OGC APIs and Asynchronous Responses

The current protocol for most (but not all) OGC API Standards requires that a `HTTP GET` request returns the resource as part of the response body. This can be characterized as synchronous request and response — the client and server are maintaining a TCP channel to exchange the request and response. In situations, where the production of the response takes a very long time or the size of the container is extraordinarily large (e.g. in Testbed-17 production of an encrypted GeoPackage for all Tiles from zoom level zero to twenty one took approximately 6 hours and required 21GB of storage). By the time the response was ready to be streamed to the client, the TCP socket was already closed. For Testbed-17, the DCS Server pushed the produced GeoPackage to a Content Delivery Network (CDN), where the client could fetch the response for the request at any later time.

An important response alternative could be that an API implementation reports to the client `response will be ready in 6 hours - please go 'here (URL)' to fetch it`. OGC API – Processes supports such asynchronous capability. Future work should investigate how other OGC API Standards could inform the client about an alternative location of the response. If an API implementation supports the persistent storage of previously requested resources (on a CDN), there could be a 'smart cache' at the API that redirects requests for previously created resources to their 'persistent' URL right away. Such a smart cache could reduce network and processing burden and improve response times.

8.3. Authenticity and Integrity of GeoPackage

How can a GeoPackage container be made tamper-resistant or how far can Integrity be applied to a GeoPackage container (perhaps as an extension) that enables the consumer to verify that the actual copy has not been tampered with since its production. In addition to (or as an extension of) Integrity, it should be evaluated how authenticity could be applied to a GeoPackage. It seems strange that a DCS extension to GeoPackage supports storing encrypted (rows of) data but that the consumer cannot verify the origin and integrity of the GeoPackage container.

8.4. Standardization of a GeoPackage Encryption Extension to OGC APIs

The results from Testbeds 15, 16 and 17 indicate that there is a common set of requirements that exists and defines a “DCS” Building Block for OGC APIs. However, in order to achieve an interoperable solution, the overall results from these Testbeds should be collected and brought forward as an Engineering Report, written as a Draft Implementation Standard. The OGC #17-007 OGC Web Services Security standard emerged in such a way.

8.5. Direct Encryption to prevent unauthorized disclosure in Web-Applications

When the client provides the Data Encryption Key (DEK) directly to the DCS Server, it is possible to ‘simply’ encrypt the response with the provided key. This simplistic scenario is equivalent to DRM direct streaming and would apply to protect data from being captured by Web-Applications, like JavaScript based clients. In the content of (one-page) Web-Applications it should be investigated in how far the encrypted data can be protected for being captured in its original format to prevent unauthorized disclosure.

8.6. Encrypted GeoTIFF and GMLJP2

Testbed-17 participants evaluated the options for packaging encrypted geospatial data into a GeoPackage container by defining suitable extensions (for Features and Tiles). The applicability of similar approaches for GeoTIFF and GMLJP2 should also be investigated along with the implications regarding the data and processing burden. If a solution exists, a standardization approach should be identified.

8.7. Standardization of a Key Management System API

For Testbed-16 and Testbed-17, a Key Management System amended the DCS components. The main responsibility of the KMS was to create a symmetric cipher key, its registration if created offline, and the administration of access conditions. The implementation from Secure Dimensions, based on NIST 800-57, was proven to do a “pretty good” job. However, the API is not standardized. In order to ensure the productive use of Data Centric Security, the standardization of the KMS API is paramount.

8.8. Sharing of a security context across administrative domains

For Testbed-17, a security context is exchanged with Bearer Access Tokens that are released and validated at one central Authorization Server. The Authorization server acts as the **single** trusted 3rd party to the DCS Server and DCS Client.

8.8.1. Federation of Authorization Servers

The first opportunity towards a federation of Zero Trust Domains could be based on establishing trust between Authorization Servers of each administrative domain. This architecture follows a recommendation from Figure 12 of the OGC Testbed-16: Federated Security Engineering Report.

8.8.2. Distributed Ledgers

Future work should evaluate the option to utilize a distributed ledger or blockchain technology to establish exchange of a security context **without** one single trusted party. With such an approach, perhaps trust for service catalogs, roles and attributes, policies in federation environments could be established.

8.9. Harmonization of Access Policies

Any Zero Trust Domain comprises of Authentication and Access Control, as illustrated in Figure 16 of the OGC Testbed-16: Federated Security Engineering Report. Even though the enforcement of access conditions must not rely on any standard, the exchange of access policies to facilitate policy harmonization across domains in a federated architecture should. Also, self-contained decision ready information GeoPackages with encrypted content should include the access policy that expresses — in a standardized fashion — the conditions for an application to decrypt and render the encrypted content. Further work should evaluate the latest work of the GeoXACML 3.0 standardization (see GeoXACML SWG for more details).

A

ANNEX A (INFORMATIVE) GEOPACKAGE DATA CENTRIC SECURITY EXTENSIONS

A ANNEX A (INFORMATIVE) GEOPACKAGE DATA CENTRIC SECURITY EXTENSIONS

The Data Centric Security (DCS) work within OGC Testbed 17 results in two GeoPackage extensions: the first extension for storing encrypted Features and the second extension for storing encrypted Tiles. Either extension defines how encrypted data (features or tiles) and the associated decryption key as well as related metadata information can be stored.

The client implementation for reading and decrypting features and tiles was implemented by Compusult. The DCS GeoPackage Server was implemented by Secure Dimensions as a Geoserver plugin. The implementation allows to create a DCS GeoPackage leveraging OGC API Features and Tiles as well as OGC Web Feature Service 2.0.

A.1. GeoPackage DCS Features Extension

WARNING

This subsection is a Testbed result and may change radically.

A.1.1. Extension Title

DCS-Features

A.1.2. Introduction

This extension provides a mechanism for storing encrypted Features encoded in GeoJSON as JWE in a GeoPackage.

A.1.3. Extension Author

Secure Dimensions GmbH, in collaboration with the participants of OGC Testbed-17, and the OGC Disaster Pilot 2021.

A.1.4. Extension Name or Template

`sd_dcs_features` (will become `gpkg_dcs_features` if adopted by the OGC)

A.1.5. Extension Type

New requirement dependent on GeoPackage Core (Clause 1).

A.1.6. Applicability

This extension allows to store Simple Features, encoded in GeoJSON, encrypted as JWE as defined in RFC 7516 — JSON Web Encryption (JWE). This extension does not constraint how the data is encrypted. It is outside the scope of this specification how the cipher key information is protected and securely communicated to the client application that is supposed to decrypt the data.

NOTE: For Testbed 17, a working prototype was implemented that is described in more detail in the Testbed 17 Data Centric Security ER.

A.1.7. Scope

read-write

A.1.8. Specification

A.1.8.1. `gpkg_extensions`

To use this extension, add the following rows to this table.

Table A.1 — gpkg_extensions Table Rows

TABLE_NAME	COLUMN_NAME	EXTENSION_NAME	DEFINITION	SCOPE
`gpkgext_contents`	`data_type`	`sd_dcs_features`	*a reference to this file*	`read-write`
`gpkgext_dcs_layers`	null	`sd_dcs_features`	*a reference to this file*	`read-write`

NOTE: The values in the `definition` column SHOULD refer in some human-readable way to this extension specification. If the extension is adopted by the OGC, it will gain the “gpkg_” prefix (replacing “sd_”) and get a different definition permalink.

A.1.8.2. New Table Definitions

Following are definitions of the tables for this extension.

As with other GeoPackage tables, this extension takes no position on how either of these tables are to be used by a client.

A.1.8.2.1. `sd_dcs_features_<identifier>`

This table contains encrypted features. The identifier is a string, uniquely representing the encrypted features stored in this table. The identifier may be an obfuscation of the actual feature type in cases where naming the table after the feature type might disclose sensitive information.

NOTE 1: For the Testbed 17 implementation, the identifier is a md5 hash of the feature type.

The columns of this table are:

- `fid` is a primary key
- `the_geom` is the bounding box of the encrypted feature or null in case it is withheld for sensitivity reasons
- `confidentiality` is a JSON Object describing the confidentiality labelling and additional information about the encrypted feature. The actual structure of the JSON is outside the scope of this specification
- `structure` is a GeoJSON based description of the structure of the feature type.
- `data` is the encrypted feature using compact JWE encoding

NOTE 2: === The confidentiality and structure columns may contain JSON, JWS or JWE. The exact type is specified via the mime-type column in the `gpkg_data_columns` ===

A.1.8.2.2. `gpkg_data_columns`

This table contains the information about the data type for the columns in the `sd_dcs_features_<identifier>` table.

To use this extension, add the following rows to this table.

Row 1: Definition of the `data` column

- `table_name` has the value `sd_dcs_features_<identifier>`
- `column_name` `data`

- `name` has the value `sd_dcs_features_<identifier>-data`
- `title` has value `Encrypted Feature Data`
- `description` description, e.g. `The encrypted data of the feature using JWE compact encoding with symmetric cipher`
- `mime_type` has value `application/jose`
- `constraint_name` has the value `null`

Row 2: Definition of the `structure` column

- `table_name` has the value `sd_dcs_features_<identifier>`
- `column_name` `structure`
- `name` has the value `sd_dcs_features_<identifier>-structure`
- `title` has value `Feature Type Data`
- `description` description, e.g. `The feature type description represented as JSON, JWS or JWE`
- `mime_type` has value `application/json` or `application/jose` to represent JWE or JWS encoded data
- `constraint_name` has the value `null`

Row 3: Definition of the `confidentiality` column

- `table_name` has the value `sd_dcs_features_<identifier>`
- `column_name` `confidentiality`
- `name` has the value `sd_dcs_features_<identifier>-confidentiality`
- `title` has value `The confidentiality labelling and producer information`
- `description` description, e.g. `The confidentiality labelling and producer information encoded as JSON or JWS`
- `mime_type` has value `application/json` or `application/jose`
- `constraint_name` has the value `null`

NOTE: No explicit media type exists for JWS or JWE encoded data. IANA media type "application/jose" must be used and from counting the dots, an application can defer whether the structure refers to JWS (2 dots) or JWE (4 dots). Some applications use the value "JWE" and "JWS" to determine between JWE and JWS encodings. For Testbed 17, the IANA media type "application/jose" was used.

A.2. GeoPackage DCS Tiles Extension

> ***WARNING***
>
> *This subsection is a Testbed result and may change radically.*

A.2.1. Extension Title

DCS-Tiles

A.2.2. Introduction

This extension provides a mechanism for storing encrypted Tiles and associated metadata such as integrity and confidentiality labelling in a GeoPackage.

A.2.3. Extension Author

Secure Dimensions GmbH, in collaboration with the participants of OGC Testbed-17, and the OGC Disaster Pilot 2021.

A.2.4. Extension Name or Template

`sd_dcs_tiles` (will become `gpkg_dcs_tiles` if adopted by the OGC)

A.2.5. Extension Type

New requirement dependent on GeoPackage Core (Clause 1).

A.2.6. Applicability

This extension allows to store encrypted binary image data (e.g. Tile) as a BLOB. This extension does not constraint how the data is encrypted. It is outside the scope of this specification how the cipher key information is protected and securely communicated to the client application that is supposed to decrypt the data.

This extension also defines the mechanism how to store cipher key information such that an application can decrypt the data.

To ensure tamper resistance of the encrypted tile data, an integrity check is stored that relates the data with the mandatory meta information zoom_level, tile_column, tile_row.

NOTE: For Testbed 17, a working prototype was implemented that is described in more detail in the Testbed 17 Data Centric Security ER.

A.2.7. Scope

read-write

A.2.8. Specification

A.2.8.1. `gpkg_extensions`

To use this extension, add the following rows to this table.

Table A.2 — gpkg_extensions Table Rows

TABLE_NAME	COLUMN_NAME	EXTENSION_NAME	DEFINITION	SCOPE
`gpkgext_contents`	`data_type`	`sd_dcs_tiles`	*a reference to this file*	`read-write`

NOTE: The values in the `definition` column SHOULD refer in some human-readable way to this extension specification. If the extension is adopted by the OGC, it will gain the “gpkg_” prefix (replacing “sd_”) and get a different definition permalink.

A.2.8.2. New Table Definitions

Following are definitions of the tables for this extension. As with other GeoPackage tables, this extension takes no position on how either of these tables are to be used by a client.

A.2.8.2.1. `sd_dcs_tiles_<identifier>`

This table contains encrypted tiles. The identifier is a string, uniquely representing the encrypted tile layer stored in this table. The identifier may be an obfuscation of the actual feature type in cases where naming the table after the feature type might disclose sensitive information.

NOTE: For the Testbed 17 implementation, the identifier is a md5 hash of the feature type.

The columns of this table are:

- `id` is a primary key
- `zoom_level` as specified by GeoPackage

- `tile_column` as specified by GeoPackage
- `tile_row` as specified by GeoPackage
- `dcs_info` is a JSON Object describing the confidentiality labelling and additional information about the encrypted feature. The actual structure of the JSON is outside the scope of this specification
- `dek_info` is a JWT defining the `kid` used for encryption. Also, the JWK payload contains additional information about the key origin such as issuer, creating application and user.
- `tile_data` is the encrypted tile using symmetric key encryption with the key described in `dek_info` column.

A.2.8.2.2. `gpkg_data_columns`

This table contains the information about the data type for the columns in the `sd_dcs_tiles_<identifier>` table.

To use this extension, add the following rows to this table.

Row 1: Definition of the `tile_data` column

- `table_name` has the value `sd_dcs_tiles_<identifier>`
- `column_name` `tile_data`
- `name` has the value `sd_dcs_tiles_<identifier>-tile_data`
- `title` has value `Encrypted Tile Data`
- `description` description, e.g. `The encrypted data of the tile using symmetric cipher`
- `mime_type` has value `application/octet-stream`
- `constraint_name` has the value `null`

Row 2: Definition of the `dcs_info` column

- `table_name` has the value `sd_dcs_tiles_<identifer>`
- `column_name` `dcs_info`
- `name` has the value `sd_dcs_tiles_<identifer>-dcs_info`
- `title` has value `DCS Metadata`
- `description` description, e.g. `The tile description`

- `mime_type` has value `application/jose` to represent JWE or JWS encoded data
- `constraint_name` has the value `null`

Row 3: Definition of the `dek_info` column

- `table_name` has the value `sd_dcs_tiles_<identifer>`
- `column_name` `dek_info`
- `name` has the value `sd_dcs_tiles_<identifer>-dek_info`
- `title` has value `The information about the encryption key as JWT`
- `description` description, e.g. `The key can be fetched via kid or kurl`
- `mime_type` has value `application/jose`
- `constraint_name` has the value `null`

NOTE: No explicit media type exists for JWS or JWE encoded data. IANA media type "application/jose" must be used and from counting the dots, an application can defer whether the structure refers to JWS (2 dots) or JWE (4 dots). Some applications use the value "JWE" and "JWS" to determine between JWE and JWS encodings. For Testbed 17, the IANA media type "application/jose" was used.

B

ANNEX B (INFORMATIVE) REVISION HISTORY

B ANNEX B (INFORMATIVE) REVISION HISTORY

DATE	RELEASE	AUTHOR	PRIMARY CLAUSES MODIFIED	DESCRIPTION
2021-05-18	0.1	A. Balaban	all	initial version
2021-07-22	0.2	A. Matheus	Annex A	initial version
2021-08-31	0.3	A. Balaban, A. Matheus	TIEs	initial version
2021-10-04	0.4	A.Matheus	Annex B	initial version
2021-10-11	0.5	A.Matheus	Sections 4, 5, 7	revision and additional content
2021-10-20	0.6	A.Matheus	All sections	streamlining content and adopting template
2021-11-05	0.7	A.Parsons	Section 5,	Client aspects
2021-11-05	0.8	G.Buehler	All sections	Re-align with template
2021-11-15	0.9	A. Balaban, A. Matheus	All sections	Incorperating editorial comments

BIBLIOGRAPHY

BIBLIOGRAPHY

1. OGC API — Features, http://docs.opengeospatial.org/is/17-069r3/17-069r3.html

2. NATO: "ADatP-4774" Confidentiality Metadata Label Syntax, edition A version 1, NSO, 2017. https://nso.nato.int/nso/zPublic/ap/PROM/ADatP-4774%20EDA%20V1%20E.pdf

3. Metadata Binding Mechanism, edition A version 1, NSO, 2018. https://nso.nato.int/nso/zPublic/ap/PROM/ADatP-4778%20EDA%20V1%20E.pdf

4. H. Butler, M. Daly, A. Doyle, S. Gillies, S. Hagen, T. Schaub: IETF RFC 7946, *The GeoJSON Format*. Internet Engineering Task Force, Fremont, CA (2016). https://raw.githubusercontent.com/relaton/relaton-data-ietf/master/data/reference.RFC.7946.xml

5. M. Jones, J. Bradley, N. Sakimura: IETF RFC 7519, *JSON Web Token (JWT)*. Internet Engineering Task Force, Fremont, CA (2015). https://raw.githubusercontent.com/relaton/relaton-data-ietf/master/data/reference.RFC.7519.xml

6. D. Hardt: IETF RFC 6749, *The OAuth 2.0 Authorization Framework*. Internet Engineering Task Force, Fremont, CA (2012). https://raw.githubusercontent.com/relaton/relaton-data-ietf/master/data/reference.RFC.6749.xml

7. M. Jones, D. Hardt: IETF RFC 6750, *The OAuth 2.0 Authorization Framework: Bearer Token Usage*. Internet Engineering Task Force, Fremont, CA (2012). https://raw.githubusercontent.com/relaton/relaton-data-ietf/master/data/reference.RFC.6750.xml

8. OGC GML in JPEG 2000 (GMLJP2) Encoding Standard. http://docs.opengeospatial.org/is/08-085r8/08-085r8.html

9. OGC: GeoXACML 1.0, OGC Implementation Specification. http://portal.opengeospatial.org/files/11-017

10. OGC 19-016r1, Testbed-15: Data Centric Security Engineering Report. http://docs.opengeospatial.org/per/19-016r1.html

11. OGC 20-021r2, Testbed-16: Data Centric Security Engineering Report. https://docs.ogc.org/per/20-021r2.html

12. OGC 20-027, Testbed-16: Federated Security Engineering Report. http://docs.opengeospatial.org/per/20-027.html#OIDC_Fed_Spec

13. OGC GeoPackage Encoding Standard. https://www.geopackage.org/spec130/index.html

14. OWS-9: SSI Security Rules Service Engineering Report. https://portal.opengeospatial.org/files/?artifact_id=51833

15. OASIS Key Management Interoperability Protocol Specification Version 2.1. https://docs.oasis-open.org/kmip/kmip-spec/v2.1/cs01/kmip-spec-v2.1-cs01.html

16. A Python implementation of the Key Management Interoperability Protocol (KMIP). https://pykmip.readthedocs.io/en/latest/

17. OGC API — Maps — Part 1: Core. http://docs.ogc.org/DRAFTS/20-058.html

18. OGC API — Tiles — Part 1: Core. http://docs.ogc.org/DRAFTS/20-057.html