OGC Testbed-16

Data Centric Security Engineering Report

Publication Date: 2021-02-26

Approval Date: 2021-02-25

Submission Date: 2021-11-18

Reference number of this document: OGC 20-021r2

Reference URL for this document: http://www.opengis.net/doc/PER/t16-D011

Category: OGC Public Engineering Report

Editor: Aleksandar Balaban

Title: OGC Testbed-16: Data Centric Security Engineering Report

OGC Public Engineering Report

Table of Contents

Chapter 1. Subject

The OGC Testbed-16 Data Centric Security Engineering Report (ER) continues the evaluation of a data-centric security (DCS) approach in a geospatial environment. In order to fully explore the potential of the DCS concept, this ER first specifies two advanced use case scenarios: Data Streaming and Offline Authorization for querying and consuming protected geospatial content. The ER then specifies the communication with a new architectural component called the Key Management Server (KMS) via an Application Programming Interface (API) created for this Testbed. The API was invoked to register keys used to encrypt data-centric protected content. Then clients called the same API to obtain those keys to perform the data verification/decryption.

The document evaluates options for structuring and encoding of containers and payloads capable of carrying the secured geospatial data sets. Previously utilized DCS container based on the tandem of formats NATO STANAG 4778 "Information on standard Metadata Binding" and NATO STANAG 4774 "Confidentiality Metadata Label Syntax" are alternatively encoded using JSON and JavaScript Object Signing and Encryption (JOSE) security standard. Thus, DCS architecture supports several content representations for enhanced interoperability. The determination of the best suited data representation occurs via Access HTTP header set by the client, which is one of the standard ways of negotiating a specific kind of resource. This header describes the preferred choice of the client. To support this mechanism, this engineering report proposes several new MIME types for geospatial, DCS specific content negotiation.

Chapter 2. Executive Summary

OGC members can derive business value from this ER in the following areas:

- Similarities between the DCS approach in the geospatial domain and the well-known commercial/enterprise Digital Rights Management (DRM) architecture for provision of protected multimedia contents. Also, the commercial geospatial content could be provided using this approach.
- Interoperability through the support for different encoding standards such as Extensible Markup Language (XML) and JSON, as well as the utilization of different container structures, such as STANAG 4774/8 and JOSE to support a variety of encoding standards and container formats.
- How to use the OGC API - Features Standard to enable client requests the DCS protected content encoded in a preferred way and how the OGC API - Features can be extended with content negotiation via additional media types.
- Common security context shared by all components. Bearer access tokens are issued by a common Authorization Server as defined in RFC 6750. The use of OAuth2 and OpenID Connect interfaces ensures interoperability.
- Data-centric Security (DCS) architecture which contains a dedicated Key management server or KMS.
- KMS API based on the OASIS Key Management Interoperability Protocol Specification 2.x provides interoperable solution and gives the strong protection for keys afforded by KMIP-compliant Servers.

The motivation for DCS is the possibility of preventing unauthorized access to systems storing sensitive data. Such systems could be increasingly popular cloud-based data storage solutions. When looking at drafting OGC standards such as OGC API - Features in a DCS scenario, standards need to include ways to classify the security requirements around data access. This classification (security label) can be performed through metadata fields as already evaluated in the OGC Testbed-15. A fundamental requirement for DCS is that the data is 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.

DCS protected data could be stored locally at the client location in order to be used within the validity period of time. As the data could pass through systems that do not belong to the data consumer nor the producer, the data must remain protected throughout all infrastructure that handles the geospatial data.

Another important aspect of the DCS is interoperability. In order to create, distribute, and consume the protected data set in an interoperable fashion, specifying the structures to encode the metadata, the protected contents, as well as the other related artefacts such as data access policies is very important.

The Testbed-16 findings show that it is possible to support DCS within an OGC API - Features implementation and have the API instance request the DCS protected content to be delivered in required encoding and DCS container format type. Storing the protected content on a mobile device locally and then decrypting and using the content offline and on demand during a possibly longer period of time is possible. Requesting the protected content online and having it delivered in a streamed fashion for a single consumption is also possible.

In support of the Testbed DCS experimentation, two scenarios were defined:

The first scenario anticipates immediate decryption and consumption of protected content. The key used for encryption is allowed to have lower strength of the encryption: Shorter key = less computational power required to encrypt/decrypt the content. The cypher (an algorithm for performing encryption or decryption) could also be simpler and therefore more efficient. The content owner wants to keep the full control over the protected content. The purpose of the encryption is to mitigate the risk of a possibly unsecured underlying network infrastructure. The content provider (DCS service) creates keys to encrypt requested content on a synchronous request/response basis. The key created by the DCS service gets registered to the KMS and could be retrieved only within its expiration time. The client is not supposed to permanently store encrypted data locally (on a desktop client). Even if the client would try to do that, due to the very short expiration time the referenced key cannot be obtained for such purpose.

The second scenario assumes the clients operate in an offline mode disconnected from the network where the DCS service is located. Protected data required for a "mission" are supposed to be downloaded and stored locally for later (field) use, possibly over a longer period of time. This scenario requires stronger security. As such, the keys are issued with an expiration time. The policies associated with issued keys contain additional geospatial assertions, which limit access rights based on user roles.

For both scenarios an OGC API - Features and DCS compatible service endpoint needs to be aware of the required encoding and the container format. Because one of the requirements for this Testbed activity was to provide support for several encoding and DCS container types, it is important to put the code for the required format in the request. A container format is a data structure which contains encrypted portions of sensitive data and associated metadata. To specify the required response formats several new media types are defined to be used as part of service requests. That includes encodings and standards such as XML, JSON, STANAG 4774/8 and JOSE.

A challenge, especially for the anticipated high grade of interoperability, was related to the absence of the support for either STANAG 4774/8 output format based on JSON encoding or any other structure/format besides the traditional XML encoding in the previous DCS architecture. The STANAG 4774/8 output format is a container format that contains signed and encrypted portions of sensitive data and associated security labels or the metadata. The Testbed participants specified and demonstrated implementations for multiple encoding options and documented recommendations regarding encodings and container formats (STANAG, JOSE).

Another design and implementation challenge was related to the Key Management Service. In the Testbed-16 architecture the KMS is responsible for creating, registering, invalidating and issuing cryptographic keys with selected strength and expiration time. The keys are used to perform cryptographic functions, including authentication, authorization and encryption. The KMS allows separation of DCS protected content from cryptographic material (keys) required for the consumption, which was not the case in Testbed-15. For Testbed-16 the new KMS components implement an OASIS API called KIMP designed to support the key management functions. In case of large data sets, where each entity is protected by a dedicated key, large key sets are required. In such situations, with many thousand keys required, key generation and retrieval takes too long. The slow responses from the KMS is caused by computational intensive key generation and might be mitigated by extending the API and optimize the data securing process.

Future testbeds should investigate following topics:

- Federated DCS architecture, which enable the collaboration (establishing of a required level of trust) among distinctive security domains.
- Additional media types for encoding and structuring of DCS protected binary data such as binary maps, tiles and coverages.
- Creation of new and update of already existing entities for DCS protected data sets.
- Standardized packaging distribution format(s) for all required artefacts such as policies, keys and protected data payload.
- Data-centric security for JP2 and GMLJP2 payloads.
- Standardize KMS

2.1. Document contributor contact points

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

Contacts

Name	Organization	Role
Aleksandar Balaban	m-click.aero	Editor
Andreas Matheus	Secure Dimensions	Contributor
Michael Leedahl	Maxar	Contributor
George Elphick	Helyx Secure Information Systems	Contributor
Marcus Alzona	keys	Contributor

2.2. Foreword

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.

Chapter 3. References

The following normative documents are referenced in this document.

- OGC API - Features [https://www.ogc.org/standards/ogcapi-features]
- GeoDRM RM [https://www.ogc.org/standards/as/geodrmrm] Geospatial DRM Reference Model (GeoDRM RM)
- OGC 06-121r9, OGC® Web Services Common Standard [https://portal.opengeospatial.org/files/?artifact_id=38867&version=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]
- NATO: "ADatP-4778" Metadata Binding Mechanism, edition A version 1, NSO, 2018. [https://nso.nato.int/nso/zPublic/ap/PROM/ADatP-4778%20EDA%20V1%20E.pdf]
- RFC 7946 - The Geo JSON Format [https://tools.ietf.org/html/rfc7946]
- RFC 7519 - JSON Web Token (JWT) [https://tools.ietf.org/html/rfc7519]
- IETF: The OAuth 2.0 Authorization Framework [https://tools.ietf.org/html/rfc6749]
- IETF: The OAuth 2.0 Authorization Framework: Bearer Token Usage [https://tools.ietf.org/html/rfc6750]
- OGC: GeoXACML 1.0, OGC Implementation Specification [http://portal.opengeospatial.org/files/?artifact_id=42734]
- OGC: GeoXACML3 - Core, OGC Discussion Paper [http://www.opengis.net/doc/DP/GEOXACML-CORE]
- OGC: GeoXACML3 - GML 3.2.1 Encoding Extension, OGC Discussion Paper [http://www.opengis.net/doc/DP/GEOXACML/GML3-Extension]
- OASIS: XACML 3, OASIS Standard [http://docs.oasis-open.org/xacml/3.0/xacml-3.0-core-spec-os-en.html]
- OGC 19-016r1, Testbed-15: Data Centric Security [http://docs.opengeospatial.org/per/19-016r1.html]
- OGC 18-026r1, Testbed-14: Security Engineering Report [https://docs.opengeospatial.org/per/18-026r1.html]
- OGC 17-021, Testbed-13: Security Engineering Report [http://docs.opengeospatial.org/per/17-021.html]
- OGC 16-040r1, Testbed-12: Aviation Security Engineering Report [http://docs.opengeospatial.org/per/16-040r1.html]
- OGC 12-139, OWS-9: SSI Security Rules Service Engineering Report [https://portal.opengeospatial.org/files/?artifact_id=51833]
- 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]
- PyKMIP: A Python implementation of the Key Management Interoperability Protocol (KMIP) [https://pykmip.readthedocs.io/en/latest/]

Chapter 4. Terms and definitions

For the purposes of this report, the definitions specified in Clause 4 of the OWS Common Implementation Standard OGC 06-121r9 [https://portal.opengeospatial.org/files/?artifact_id=38867&version=2] shall apply. In addition, the following terms and definitions apply.

AS	OAuth2 Authorization Server — a component that dispatches, validates manages bearer access tokens.
CRUD	In computer programming, create, read (aka retrieve), update, and delete are the four basic functions of persistent storage.
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.
GeoPDP	Geospatial Policy Decision Point — a component of a policy based system that uses a request, attributes about a request (including geospatial attributes) and a policy document to make an access decision to allow access to a resource. The GeoPDP implements the OGC GeoXACML implementation specification.
GeoPEP	Geospatial Policy Enforcement Point — a component of a geospatial aware policy based system that works with a GeoPDP to enforce access decision and perform obligations requested by the GeoPDP.
OGC API	A new OGC API Features Part 1 Core standard for a feature service application programming interface that provides access to feature collections and the items in them. This standard was formally known as WFS3 for Web Feature Service version 3.
LDProxy	LDProxy — An Open Source product by Interactive Instruments which provides most of the REST implementation specified in the OGC API - Features Standard.
PDP	Policy Decision Point — a component of a policy based system that uses a request, attributes about a request (including geospatial attributes) and a policy document to make an access decision to allow access to a resource. The PDP implements the OASIS XACML3 standard.
STANAG	In NATO, Standardization Agreement, defines processes, procedures, terms, and conditions for common military or technical procedures or equipment between the member countries of the alliance.

4.1. Abbreviated terms

ADR	Authorization Decision Request
DCS	Data Centric Security
DRM	Digital Rights Management
GeoPDP	Geospatial Policy Decision Point
GeoPEP	Geospatial Policy Enforcement Point
GeoXACML	Geospatial eXtensible Access Control Markup Language
JOSE	Javascript Object Signing and Encryption
JWT	JSON Web Token
KMS	Key Management Server
OAPIF	OGC API - Features
OGC	Open Geospatial Consortium
PDP	Policy Decision Point
PEP	Policy Enforcement Point
SAML	Security Assertion Markup Language
STANAG	Standardization Agreement
WFS3	Web Feature Service version 3 (Also known as OGC API Features)
XACML	eXtensible Access Control Markup Language
XML	eXtensible Markup Language
XSLT	eXtensible Stylesheet Language Template

Chapter 5. Overview

Chapter 6 introduces the problem of geospatial data centric security with respect to advanced use case scenarios derived from the digital rights management architecture.

Chapter 7 lists informal requirements and presents two advanced use case scenarios. These scenarios include content negotiation, retrieval, decryption, and the portrayal of data centric secured geospatial data sets on desktop and mobile devices. This chapter also depicts some aspects of Testbed-16’s solution/demonstration architecture.

Chapter 8 discusses payload encoding standards, data-centric container structures, and MIME media types. The chapter presents options for containers and encodings that utilize of XML, JSON, STANAG and/or JOSE standards. The chapter recommends new media type definitions required for content negotiation in "DCS aware" APIs.

Chapter 9 provides a summary of the main findings and explains the results in the implementation for the architecture used in Testbed-16. This section also lists the challenges which were tackled during the design and implementation process.

Chapter 10 considers interesting topics to be researched in future work.

Appendix A provides code snippets that illustrate the XML and JSON encodings as well as container structures based on STANAG and JOSE.

Appendix B explains the engineering aspects of the DCS components D120 (DCS Server) and D145 (Key Management Server).

Appendix C introduces the engineering aspects of the DCS component D146 (Key Management Server).

Appendix D presents Access Control Policies for DCS Server and Mobile Clients.

Appendix E presents DCS Roles Concept and Approach for Mobile Clients.

Chapter 6. Data Centric Security (DCS)

6.1. Introduction

Data-centric security (DCS) is an approach that underlines the *security of the data itself* rather than the *security of communication infrastructure* such as networks, servers, or applications. DCS further embeds security and usage policy within the content. The DCS related work previously conducted in the Testbed-15 [https://docs.ogc.org/per/19-016r1.html] explains the motivation for DCS in geospatial environment as "the response to the possibility that an unauthorized user, who intercepts network traffic or hacks systems storing sensitive information gains unauthorized data access".

Testbed-15 explored the DCS essentials and evaluated basis data centric protection (encryption), security labels (metadata), and access control based on security access policies. The policy enforcement considered temporal and spatial attributes assigned to requested data sets and service consumers. Testbed-16 adds the JSON encoding for responses and introduces a dedicated key management server component. Testbed-16 work further allows content negotiation via new, proposed media types. Other aspects of the DCS concept, such as management and tracking, might be subjects of the future work.

6.2. Key Concepts

A data-centric security model includes:

- **Discover:** The ability to inspect data storage areas to detect sensitive information.
- **Manage:** The ability to define access policies that will determine if certain data is accessible, editable, or blocked from specific users, or locations.
- **Protect:** The ability to defend against data loss or unauthorized use of data and prevent sensitive data from being sent to unauthorized users or locations.
- **Track:** The constant monitoring of data usage to identify meaningful deviations from normal behavior that would point to possible malicious intent.

According to established theoretical models DCS relies on the implementation of the following:

- Information (data) that is **self-describing and defending**, which means metadata that describes the information and the security of data does not depend on applications and infrastructure.
- Information that **remains protected as it moves** in and out of applications and storage systems, and changing business context.
- **Policies and controls** that account for business context (relevant use case scenarios)

These concepts should be considered as a key aspect of the whole information life cycle phases such as creation, processing, collaboration, storage, archive, search, and finally deletion.

In a DCS architecture the data sets are "labeled" with metadata. This is usually in form of a structured header having attributes which specify their security relevance and allow the application of security access policies on such data (RFC-7444 [https://tools.ietf.org/html/rfc7444]).

Security Labels provide a mechanism for controlling access to information in security environments. Data objects are labeled with a classification, such as "Confidential", "Secret", or "Top Secret". Data consumers are given a clearance, using the same scheme. The core model of security clearance (access control) is that someone accessing information has a security clearance, that controls what information can be accessed.

A common metadata format (DCS container structure) is the starting point from which more attributes could be integrated into the metadata. This includes for example, a creation and validity period, the data taxonomy, or the identity of the person assigning the classification. The data structure to hold this information should be designed in such a way that new attributes can easily be added. For example, the inclusion for post-release protection ensures the data can be released for a number of days, after which the data cannot be accessed. Cryptographic binding of the classification metadata to the data ensures integrity of the label and the data.

Document ITU-T X.841 (Security information objects for access control - Fig.5) provides general recommendation for DCS Access Control. Protected information is represented as a message with a cryptographically bounded (confidentiality) label. On the client-side policy enforcement makes data access decisions based on the label, user credentials (security level), and specific policy assertions about access granting. Comments in blue color added to the original figure represents the Testbed-16 DCS specific implementation details.

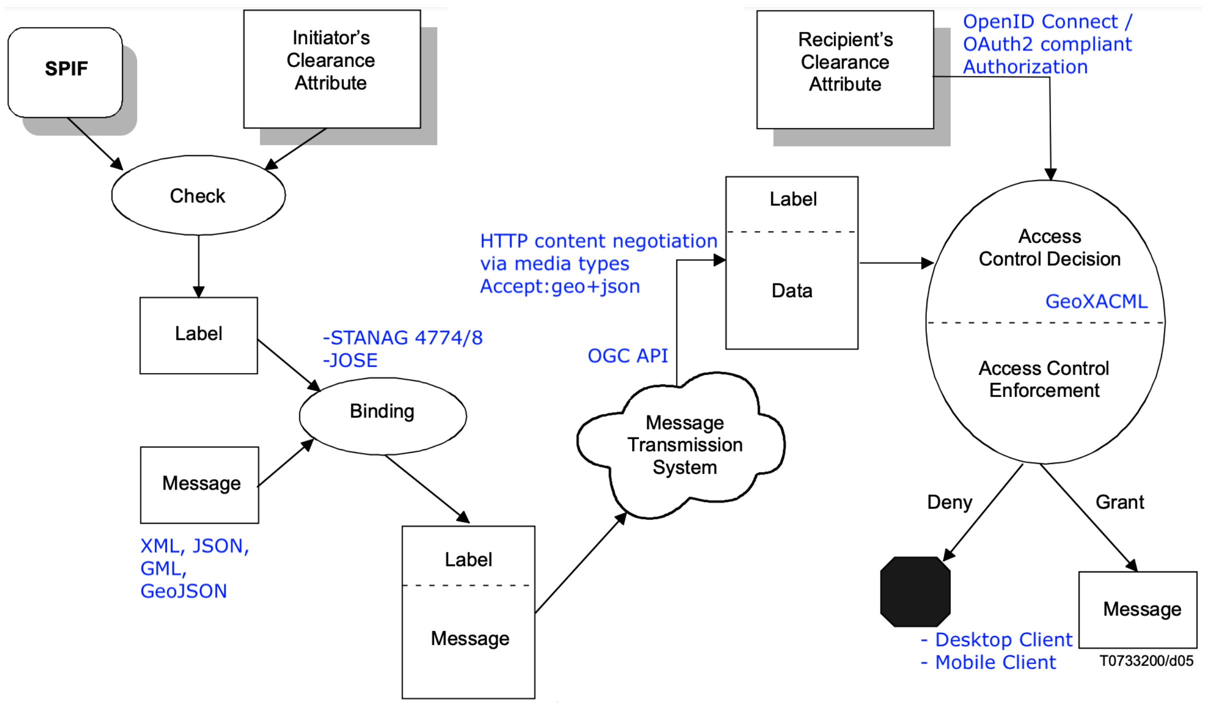

SPIF
Initiator's Clearance Attribute
Check
Label
-STANAG 4774/8
-JOSE
Binding
Message
XML, JSON, GML, GeoJSON
Label
Message
HTTP content negotiation via media types Accept:geo+json
OGC API
Message Transmission System
Label
Data
Recipient's Clearance Attribute
OpenID Connect / OAuth2 compliant Authorization
Access Control Decision
GeoXACML
Access Control Enforcement
Deny
Grant
Message
- Desktop Client
- Mobile Client
T0733200/d05

Figure 1. DCS Access Control

The most relevant part of a security (confidentiality) label is the classification. Many security label schemes (like those used in the previous Testbed-15 and in this one) use the following classifications:

- Unclassified
- Restricted
- Confidential
- Secret

* Top Secret

STANAG 4774/8 standards define the application of a confidentiality label. This is a structured representation of the sensitivity of a piece of information. Previous DCS work was primarily focused on exploring that aspect of DCS. A data originator security label example based on NATO STANAG 4774 is given in the listing below. Encoding is XML. Bound through cryptographic signature with an arbitrary data payload, for example a Geography Markup Language (GML) document, would label the document as "secret".

STANAG 4774 Confidentiality Label

```xml
<originatorConfidentialityLabel xmlns=
"urn:nato:stanag:4774:confidentialitymetadatalabel:1:0">
    <ConfidentialityInformation>
        <PolicyIdentifier>DCS_TB-16</PolicyIdentifier>
        <Classification>SECRET</Classification>
        <GenericValue>OGC</GenericValue>
    </ConfidentialityInformation>
</originatorConfidentialityLabel>
```

Chapter 7. Requirements, Scenarios and Architecture

This chapter describes the DCS architecture following the general concept of multiple views. This approach identifies architectural elements while illustrating and validating the architecture design. The contents of the chapter have views of logical, component, process and deployment. However, the focus of the chapter is on the logical components and their interactions. Appendices dedicated to the components provide an overview related to the practical deployment of logical system components (physical or deployment view).

7.1. Requirements

Although the task dedicated to the DCS did not mandate formal requirements in Testbed-16, this section lists informal requirements based on the OGC Testbed-16 Call for Participation (CFP) as well as the section Future Work [https://docs.ogc.org/per/19-016r1.html#FutureWork] from previous Testbed Engineering Reports.

Testbed-16 DCS Requirements

R01	DCS architecture will contain a key management service or KMS component, which shall be able to create, register, issue and invalidate keys used to protect the content in the context of DCS.
R02	KMS shall utilize standard or dedicated API to allow the communication with other components.
R03	DCS architecture shall provide the mechanism for content negotiation. Said differently, a client informs the content provider (DCS server) about the preferable encoding of the content and container format.
R04	The OGC API - Features shall provide data protection independent of the transport. Identities, tokens, keys, access rights, policies have to be supported by the API.
R05	DCS architecture shall keep support for XML/STANAG 4774/8 encoding for DCS payload containers.
R06	DCS architecture shall support JSON encoding for DCS payload containers.
R07	DCS architecture shall use well-established standards (JOSE) when implementing the JSON encoding for DCS containers.
R08	DCS architecture shall support encryption for meta-data in DCS containers.

7.2. Scenarios (Use Cases)

For this Testbed, the experiments advanced the DCS concept (with respect to the geospatial domain) to a high-level architecture similar to "Digital Rights Management" or DRM. DRM concepts allow for the creation, protection, and delivery/consumption of content in accordance with contracts put in place between the parties. Such contracts specify which content is available against predefined conditions, usually specified in a policy. DRM architectures make clear the distinctions between content author, owner, providers and consumers. Using a DRM platform, a client could order content. This content could be multimedia more relevant to this topic than geospatial data sets in GML or satellite imagery data. After receiving a payment or obtaining credentials elsewhere the service/client **streams and encodes/consumes the content on the fly** or **downloads and encodes/consumes the content later** (possibly several times during the key validity period). With inspiration from a DRM architecture, the Testbed experiments derived use case scenarios and specified component interactions.

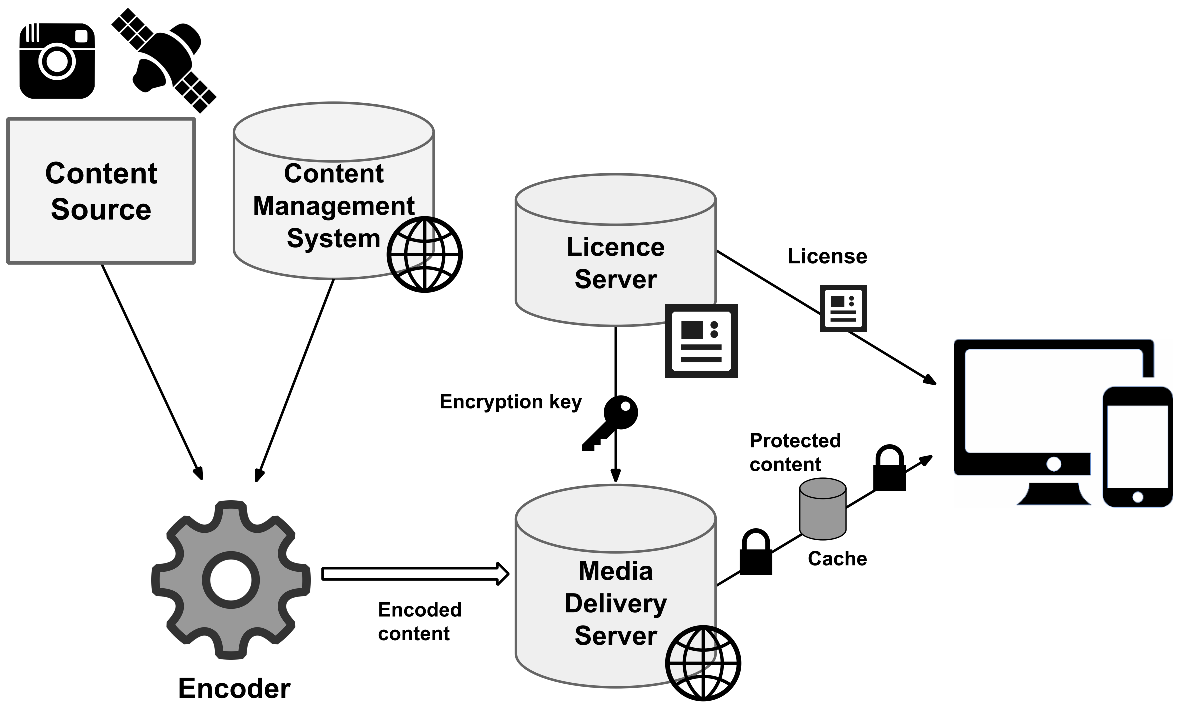


Figure 2. Digital Rights Management Architecture

An example of more detailed evaluation of DRM in the geospatial area could be seen in GeoDRM RM [https://www.ogc.org/standards/as/geodrmrm].

This motivation and the requirements put on DCS lead us to the following two use case scenarios:

- Online Streaming
- Offline Authorization

Because the DCS protected data sets leave the security perimeter of the data owner and are hosted in a cloud and/or distributed via third party channels, the important elements to deal with when applying the concepts of data-centric security are the strength of the encryption key (the length of the key) and cipher algorithm for performing encryption/decryption. These factors (strength and algorithm) relate to the differences in how these scenarios use the data. While the **Online Streaming** of protected content requires the decryption on the fly, in the case of **Offline Authorization** the client caches the encrypted content. The client decrypts the content when needed in the offline scenario.

7.2.1. Use Case 1 (Online Streaming)

For the Online Streaming use case, the client ensures the immediate decryption of the data. The client should not offer the ability to cache the encrypted data for use later as the keys are for a single use. In this case, the encryption key may be short for use with a simple cypher since the purpose of encryption is to overcome uncertain network security (e.g. multi-segmented network with unknown segment security or known low level protection). However, one should assume that intermediaries (bad actors) would be able to store encrypted content for later brute force decryption and that (bad) end users could re-distribute decrypted content.

Having said that, in this scenario the DCS server creates keys to encrypt the response content on a request/response basis. The DCS Server uploads the keys to the KMS. Clients retrieve these keys when they get the protected response data set and parse out the key identifications from the unprotected meta-data section of a DCS container. Key retrieval at KMS is only possible by meeting the criteria specified in the KMS policy (including the expiration time of the key). A criterion of the policy concerns the verification of the server's trust of a client. Achieving continuous protection of the data is only possible with trusted client applications that do support "viewing" of decrypted data only; no "save to disk" operation available.

7.2.2. Use Case 2 (Offline Authorization)

In this scenario the client operates disconnected from the network. In addition to the DRM architecture, the scenario derives motivation from the concept of Geo-information for Disaster Management. Rescue teams equipped with mobile equipment for navigation/communication and situational-awareness receive classified, encrypted geospatial data prior to deployment in disaster areas. The use of the data on the mobile device may span larger period of time. When meeting with first responders, critical information shall be shared with them without compromising sensitive information. Offline authorization requires stronger security partly because the KMS issue keys with longer expiration times. Clients or a service on the mobile device need to maintain policies against keys which may contain geospatial and temporal constraints on the use of the key.

This scenario contains assumptions that the data classification, and the actors that use the device vary. However, the data for all actors and classifications exist on the same device. This allows an incident commander to activate a role on the device and pass it out to an incident responder. Since the curation of data for the device happens prior to deployment in the field, the device needs to store the data, keys and key policies. The security problem increases because multiple users handle the device and not all the users have the same data needs or have clearance to see all the data.

To secure the data, keys and policies, the mobile device has a policy enforcement component PEP). This component maybe separate from or embedded into the GIS application. The PEP needs the ability for an authorized user to select a role from the policy to enforce. The security needs strong encryption and cypher algorithms to protect the data, keys and policies. The need for this strong security stance comes from the fact that the data may be on the device for a long timeframe. Additionally, multiple users may use the data several times before a user with proper clearance removes the data from the device.

The KMS that the curation workflow uses to create key, encrypt data, and sign content needs to support the strong keys and cypher algorithms. The curation tool, which may be on or separate from the mobile policy enforcement component, needs the ability to support the same STANAG

4774/8 derived formats that scenario one uses. In particular, the experiments in this testbed require that the participants use a JSON based encoding that was derived from the NATO STANAG 4774/8 standard.

7.3. DCS Architecture Components

The next figure depicts a high-level TB-16 DCS architecture with respect to the more general DRM architecture as depicted on Figure 2 and used as template:

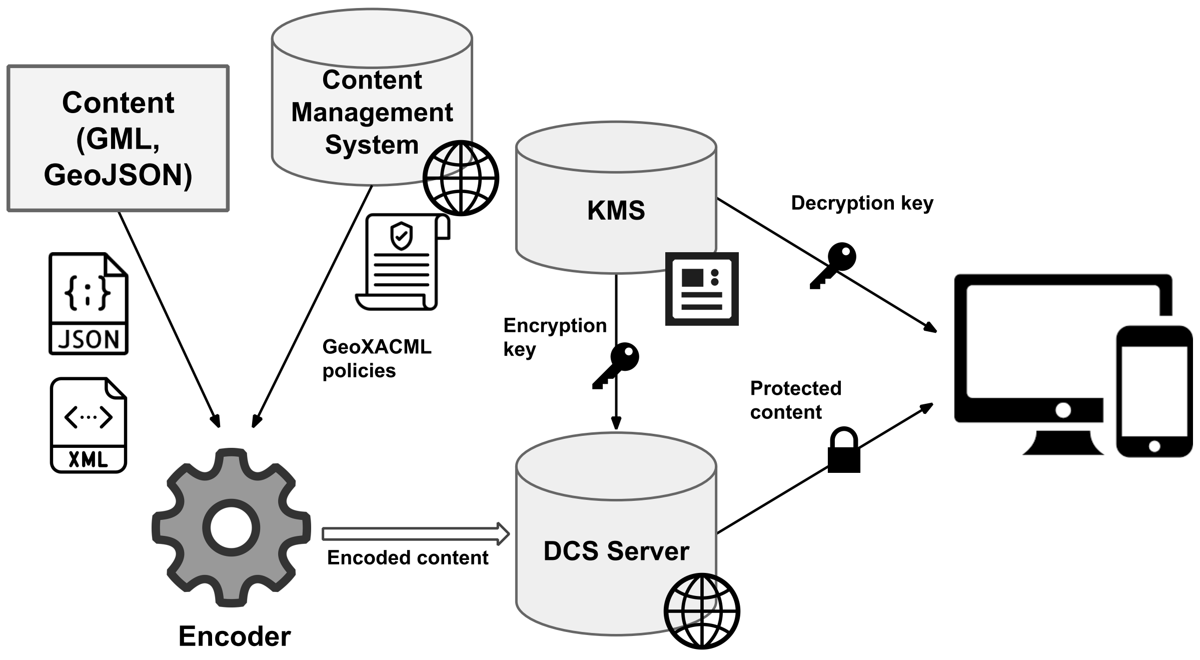


Figure 3. DCS in relation to DRM Architecture

The figure represents a logical view, which roughly mimics the general DRM blueprint. The implementations are different for desktop and mobile scenarios. In the desktop scenario a DCS server provides protected content on request via an OGC compatible API. An encoder component is implemented inside the DCS service and it is basically irrelevant for this testbed. Mobile scenario on the other side puts the focus on the policy enforcement, decoding and visualizing of protected content on mobile devices with respect to the user rights and policies associated with data sets. Therefore, the function of the encoder was performed by an external tool, which was used to prepare the content for TIE execution. Protected content resides in a cache on a mobile device. The mobile scenario demonstrates the interaction with KMS and content visualization on a mobile device after decryption was performed in accordance with user roles (security levels) and (GeoXACML) policies in the GeoPEP component.

The following components, listed with their essential features, are part of the Testbed-16 DCS architecture (in both scenarios):

DCS Server - Desktop (https://ogc.secure-dimensions.com/dcs)

- OGC API for test features, OAuth2 resource server
- Creates and registers DCS keys with KMS
- The following components build the DCS Server:
  - Geoserver with example data
  - ldproxy: This proxy to the Geoserver produces the OGC API Features on top of Geoserver
  - Policy enforcement point geoPEP, security proxy
  - Policy decision point geoPDP, GeoXACML 3 compliant

Authorization Component/Server (https://www.authenix.eu)

- OpenID Connect / OAuth2 compliant Authorization Server with federated login (Google, Facebook, eduGAIN, + OGC Portal IdP and Testbed IdP)
- The OGC Portal IdP containing logins for all OGC members

Testbed IdP (IdP) Component/Server - Desktop (https://ogc.secure-dimensions.com/simplesaml)

- The OGC Testbed IdP containing (fictitious) users from Testbed-15 with different clearance

Key Management Server (KMS) - Desktop (https://ogc.secure-dimensions.com/kms/api)

- REST API implemented in PHP, documented with OpenAPI 3.0

Key Management Server (KMS) - Mobile

- REST API implemented in Flask, documented with OpenAPI Doc 3.0
- Implements endpoints consistent with the OASIS Standard KMIP Client

DCS Client - Desktop (https://github.com/ogc-leedahl/QGIS/tree/OGC_Testbed_16)

- QGIS to interact with DCS Server and KMS
- Obtain key(s) from KMS
- Validate signature + decode encrypted content

DCS App / Client (Android) - Mobile (https://github.com/ogc-leedahl/QField/tree/Testbed16)

- QField Client
- Have a user feature flow for selecting features and using content
- Obtain features and key(s) from a Policy Enforcement Component
- Validate signature + decode encrypted content

Policy Enforcement Point (Android GeoPEP) - Mobile

- Has an administrative curation flow for creating encrypted contents
- Obtains content from some imagery source for curation
- Obtains encrypted and signed content from a KMS
- Serves curated encrypted features and encryption keys to a client

- Implementation of the policy enforcement component could be embedded in the mobile client or as a stand-alone component

DCS App (iOS) - Mobile

- Apple Mapkit-based
- Encrypted "DCS Features" Data pre-loaded onto mobile device
- Allows selection of "DCS Roles" as provided by the iOS PEP
- Displays feature content as decoded by the iOS PEP

Policy Enforcement Point (iOS PEP) - Mobile

- Module implemented within iOS DCS App
- User/Device-Specific "DCS Roles" Data pre-loaded onto mobile device
- Retrieves encryption keys for specified roles from KMS (caches for offline use)
- Allows DCS App to validate signature + decode encrypted content based on current DCS Role

7.3.1. Scenario 1, DCS Desktop/Client/Server

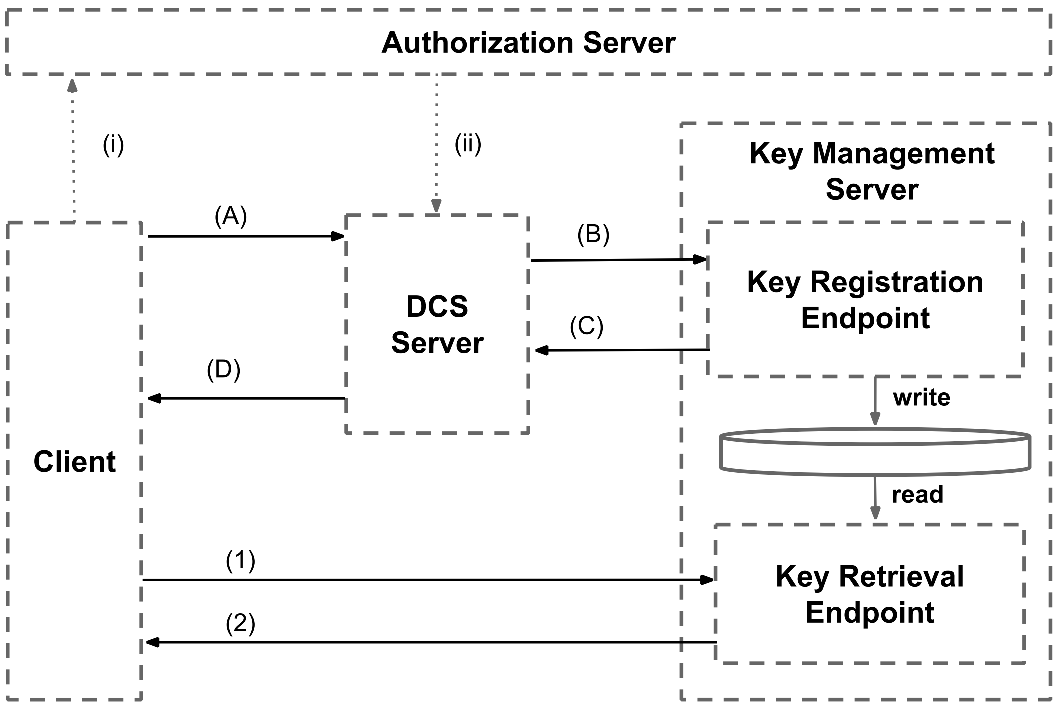


Figure 4. Testbed-16 DCS Desktop/Client/Server Components and their Interactions

Interactions between the desktop and server components of the architecture are:

Authorization

i. Client registers for OAuth2 Authorization Code Grant [https://tools.ietf.org/html/rfc6749#page-8]

ii. DCS Server is Resource Server (registered for OAuth Client Credentials Flow [https://tools.ietf.org/html/rfc6749#page-7])

Service Request

A. Feature request goes to an OGC API compatible DCS Server endpoint. The request contains access_token, key_challenge, challenge_method [https://tools.ietf.org/html/rfc7636]

B. The DCS client sends an OGC API - Features encoded request to the DCS Server including the access token and content type encoded in HTTP Access header for content negotiation. The access token from the request gets validated via the Authorization Server.

C. Based on the response from the backend feature data repository (OGC API), the DCS Server creates a cipher key per feature type classification (top_secret, secret, confidential, classified). The cipher keys differ in length and algorithm for each classification level. For each cipher key created, the DCS Server registers the key with the KMS.

D. Every key_id from the KMS response is included in the DCS container of choice (content negotiation). DCS server returns the response in the form of a DCS container of chosen encoding to the client.

Decryption Key Retrieval

1. The DCS client reads the response DCS container and extracts a list of key identifiers.
2. For each key_id the DCS client sends a request to the Key Management Server for obtaining the cipher key and decodes the payload.

Appendix Engineering Aspects for D120 and D145 provides very detailed sequence and explanation for desktop client use case (UC 1).

7.3.2. Scenario 2, DCS Mobile App/Client & Policy Enforcement Point

The DCS Mobile Scenario implementations differ slightly in their architectures and feature sets, allowing for the exploration of different distribution mechanisms.

7.3.2.1. QField / GeoPEP (Android Mobile App/Client)

The following figure shows the Android-based Testbed-16 DCS Mobile App Client/Server Components and their Interactions.

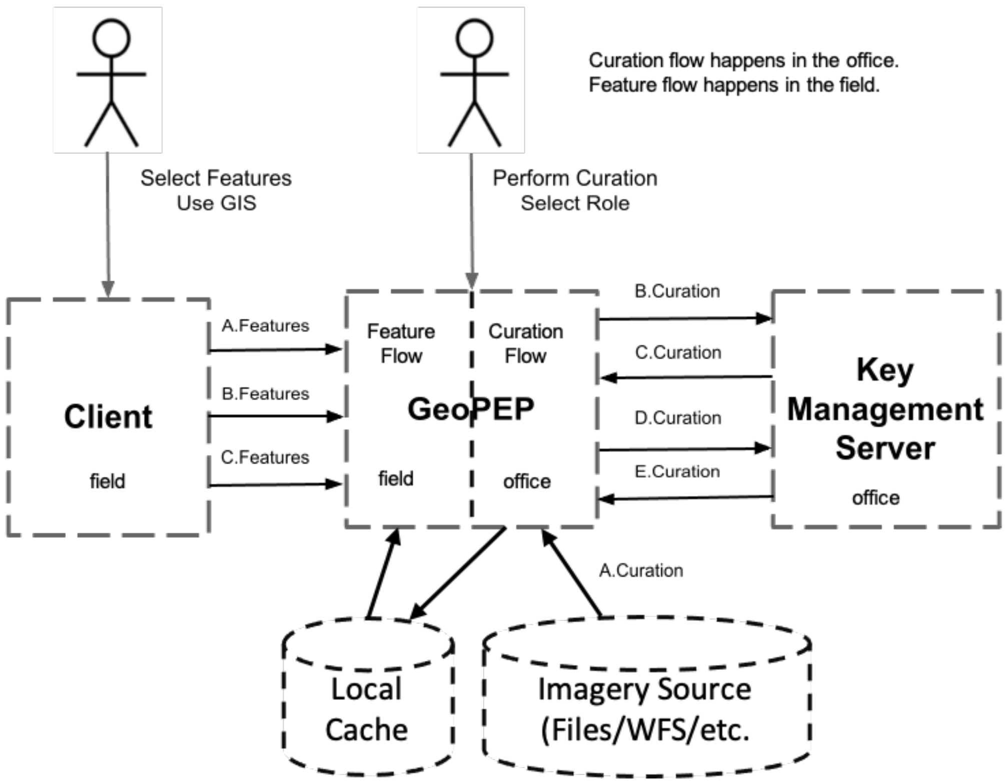


Figure 5. Testbed-16 DCS Mobile App Client/Server Components and their Interactions (Android)

Interactions between the mobile and server components of the architecture are:

i. **Curation Flow:** An administrative user or a user with proper clearance curates data and selects an active role to serve to a GIS user.
   a. A user using the GeoPEP defines roles, selects the features involved per role and defines rules for each role.
      i. The client fetches the data from files, a Web Feature Service or some other means depending on the implementation.
   b. The GeoPEP asks the KMS to create keys and encrypt data.
   c. The KMS creates the keys and encrypts the data and returns them to the GeoPEP.
   d. The GeoPEP asks the KMS to create signing keys and sign content.
   e. The KMS returns the keys and signed content to the GeoPEP.

ii. **Feature Flow:** A GIS user selects features to use in the GIS app and uses the data.
   a. A user using a client fetch a list of features to work with from the GeoPEP.
   b. The client fetches the features from the GeoPEP.
   c. The client fetches keys from the GeoPEP, validates the signature and decrypts the content.

7.3.2.2. MapKit / DCS Roles (iOS Mobile App)

The iOS Mobile Client architecture implements the DCS Roles concept, separating the scenario data into two categories - Feature Data ("DCS Data") and "DCS Roles". Please see Appendix E: Roles for full details.

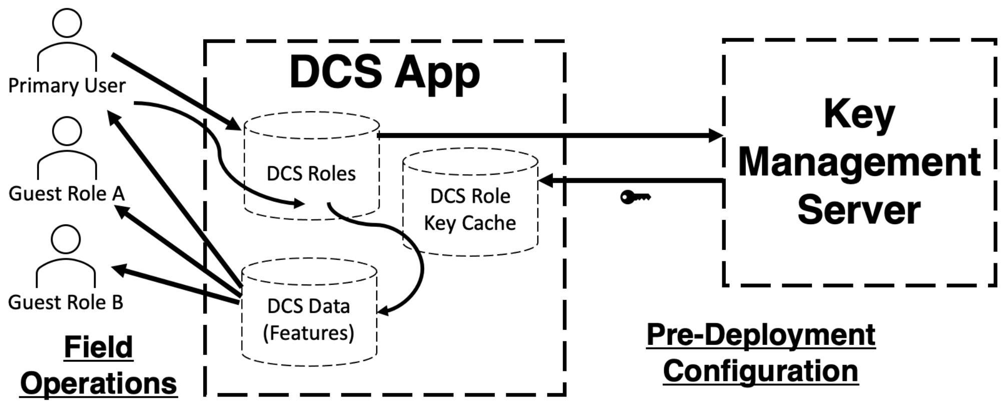


Figure 6. Testbed-16 DCS Mobile App Components, Roles, and their Interactions (iOS)

7.3.2.2.1. DCS Roles vs Users

Within this role-based mobile implementation, a user is the assigned user for the mobile device. That user has their personal security clearance loaded onto the device as a DCS Role. In addition to that personal DCS Role, per the scenario multiple generic DCS Roles representing generic security clearances for the categories of people the user may encounter in the field who the user may wish to share information.

7.3.2.2.2. DCS Roles vs DCS Data

Within this concept, each DCS Data item is to be restricted according to a specific Policy Identifier and a specific Classification, as specified within a DCS Data container (as described elsewhere in this document). Whereas each DCS Role could potentially specify access to multiple Classifications and multiple Contexts.

This allows the "filtering" of data displayed on the mobile device to show only DCS Data items that meet the restrictions of the current active DCS Role.

Furthermore, this allows for (requires) the DCS Data and (list of) DCS Roles to be distributed and installed separately on the mobile devices.

7.4. DCS Architecture Interactions

7.4.1. Desktop/Client/Server Interactions

Following diagram explains the workflows related to the communication with Authentication Server and KMS for both use cases:

- DCS Server creates key(s) for “immediate” use (Online Streaming UC)
  - keys are simple (symmetric and short)
  - expires_in as set by the DCS Server
- “expires_in” and “algorithm” as set by the client’s characteristics
  - grant_type=(implicit, authorization_code)
  - If scope=offline_access (possible for authorization_code_grant), key algorithm will be stronger but still symmetric (Offline Authorization UC)

- Key Registration (POST /kms/keys)
    - Requires scope=kms
    - Client (D120) has no KMS scope
    - DCS Server has KMS scope
- Client registers and uses OAuth2 Authorization Code Grant
- DCS Server is Resource Server (registered for OAuth Client Credentials Flow)
- OGC API Features + access_token + key_challenge + challenge_method

The workflow visualization provides the notation of UML sequence diagrams, which depict interactions among DCS architecture components for both use case scenarios. The diagram below explains the process of creation of DCS protected content on behalf of a client. The diagram also depicts the creation of cryptographic key material, registration on KMS server (with the return of key identifier) and encryption of data payload.

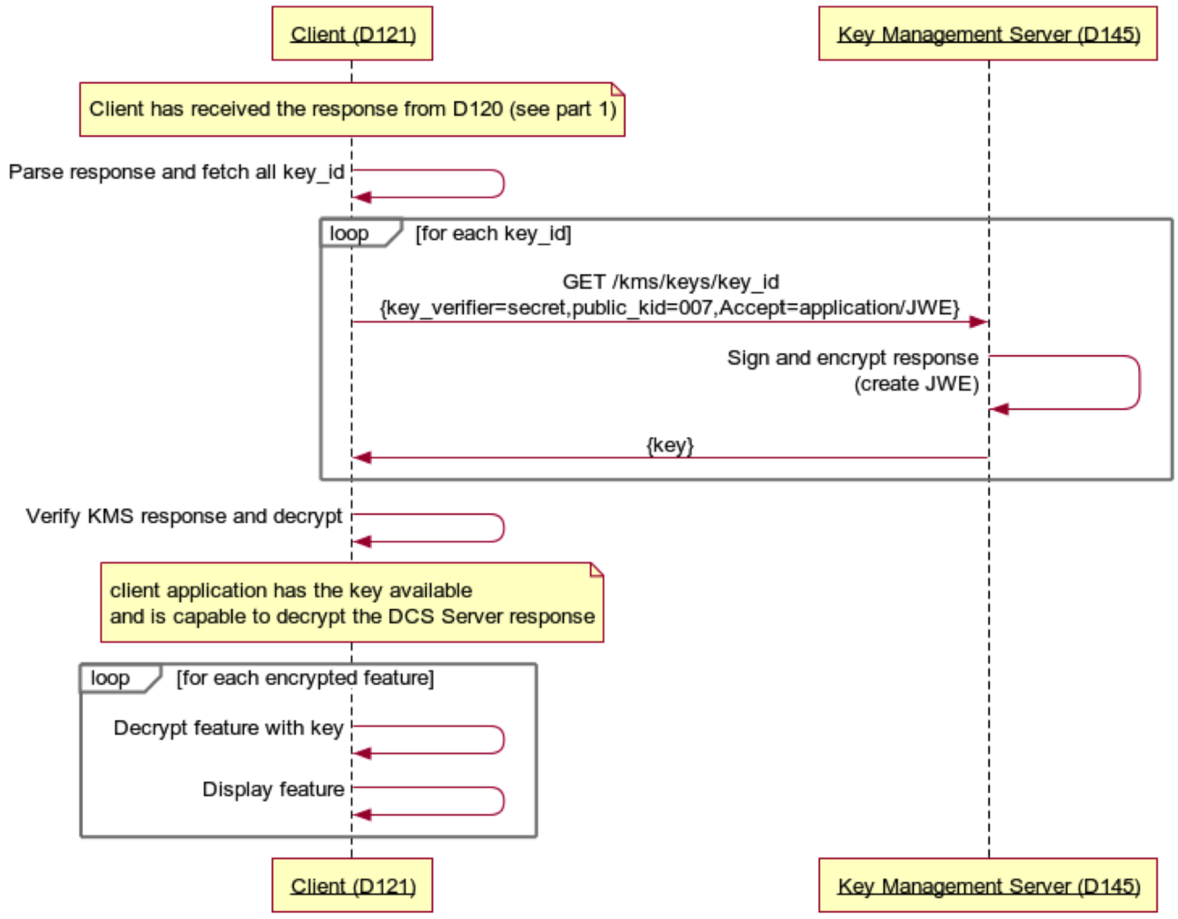

Client (D121)
Key Management Server (D145)
Client has received the response from D120 (see part 1)
Parse response and fetch all key_id
loop [for each key_id]
GET /kms/keys/key_id
{key_verifier=secret,public_kid=007,Accept=application/JWE}
Sign and encrypt response
(create JWE)
{key}
Verify KMS response and decrypt
client application has the key available
and is capable to decrypt the DCS Server response
loop [for each encrypted feature]
Decrypt feature with key
Display feature
Client (D121)
Key Management Server (D145)

Figure 7. TB-16 DCS component interactions 1

The next figure depicts the consumption of data centric protected information. The client first obtains and subsequently parses the protected content. The client, for every encrypted data segment, retrieves the key_id from the container's meta-data section and then uses that key to obtain the cryptographic key from KMS. Finally, the client decrypts and presents the protected content.

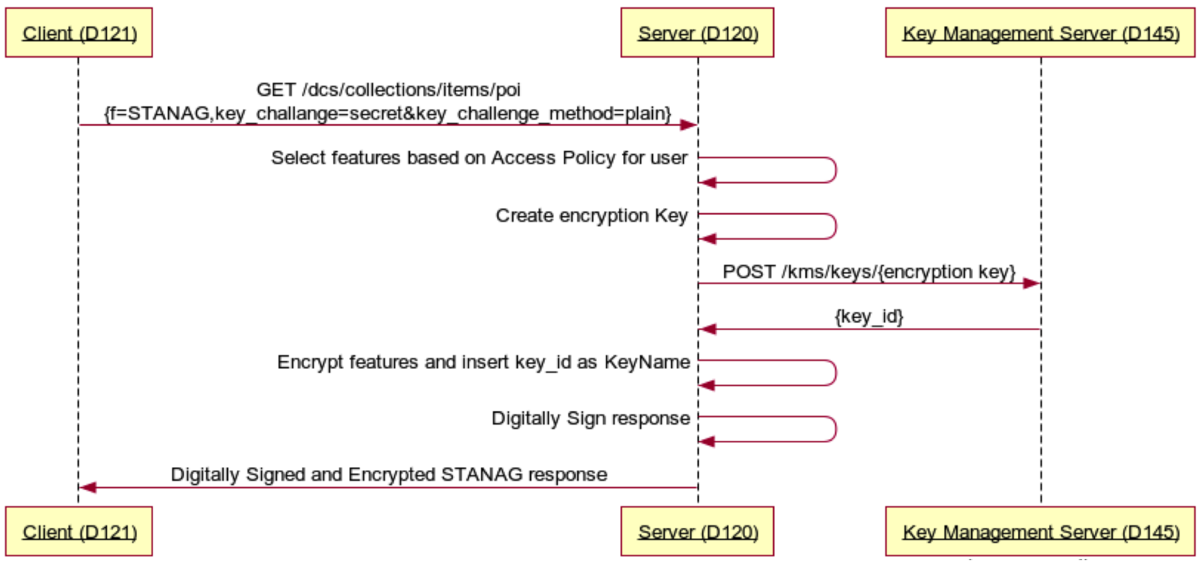

Client (D121)
Server (D120)
Key Management Server (D145)
GET /dcs/collections/items/poi {f=STANAG,key_challange=secret&key_challenge_method=plain}
Select features based on Access Policy for user
Create encryption Key
POST /kms/keys/{encryption key}
{key_id}
Encrypt features and insert key_id as KeyName
Digitally Sign response
Digitally Signed and Encrypted STANAG response
Client (D121)
Server (D120)
Key Management Server (D145)

Figure 8. TB-16 DCS component interactions 2

All implementation details could be seen in Engineering Aspects for D120 and D145

7.4.2. Mobile App/Server Interactions

The GeoPEP scenario for the mobile and KMS interactions involves the curation of offline data to present in the field to various users representing a variety of roles. To satisfy this requirement, an implementor may choose to implement the solution using GIS Software and a stand-alone GeoPEP for policy enforcement. The implementor may also choose to embed the GeoPEP inside the GIS Software. Regardless of the approach the interaction between the GIS component and GeoPEP component are similar.

The first step is to curate the data in the GeoPEP. This may be an offline process done with some curation tool, or it can be done in the GeoPEP. Regardless of where it is done the curation flow needs to define:

- Roles
- Which features or feature classes that the roles can use.
- Rules for what, when and where users may view features.

To facilitate the protection of the features the curation component can reach out to a KMS to:

- Create encryption keys
  - In this experiment the implementors are using symmetric keys to encrypt the feature inside the feature collection as was described in the desktop/server interactions above.
- Encrypt sensitive information about individual features.
  - The KMS creates keys and encrypts feature data that the KMS receives from a curation tool.
  - The curation tool creates a JSON Web Encryption (JWE) formatted response for each feature.
- Create signing keys.
  - In this experiment the implementors are using asymmetric keys to sign the feature collections.

- Sign the feature collection.
    - The curation tool stores signed feature collection in a Java Web Signature (JWS) structure.

The next flow of interactions is between the client and a GeoPEP component. The components may be separate or embedded in the same application.

- A user of the GIS client selects features to display from what is available to the role the user is assuming.
- The GIS component retrieves the features and keys from the GeoPEP component.
- The GIS component validates the signature, decrypts the data and displays it to the user.

7.4.2.1. DCS Mobile App and GeoPEP Server Interactions with Role Definitions

For both the iOS DCS App and the Android GeoPEP implementations, the application or GeoPEP limits the server interaction to the initial configuration of the application/component after loading or fetching the data. This is import to the scenarios this experiment defines as the scenarios start with the assumption that communications may be down for responders in the field. Thus, it is important to load the data before mobilizing in the field. Another import part of these interactions from the application/component is the use of rules for the specification of roles within software. Roles define rules for the encryption of the features and thus effect the interactions with the Key Management Service (KMS).

The mobile applications query the KMS for the encryption keys for the feature according to the specification of rules for roles in the application, caching these keys for offline (from the KMS/internal network) use. The iOS application bases the rules by roles and fetches keys according to the role specification (clearance, which may contain multiple classifications). The iOS application applies the appropriate key to each feature allowed by the role. The Android GeoPEP allows the user to specify a classification level, and an encryption strength for each feature class. The GeoPEP then fetches keys from the KMS for each feature according to the encryption strength specification of the feature class. The GeoPEP then defines roles and how the features map to them.

There are advantages and disadvantages to both approaches. Basing the encryption off of the role definition and choosing one key to represent a role/classification pairing requires fewer keys to encrypt/decrypt the features. Creating a new key for each feature provides more security but at the cost of needing more keys for encryption/decryption. Network latency and bandwidth considerations play into these decisions. In a time-critical response, creating separate keys may not be the best trade off since it can take hours to create the keys and encrypt the data on large datasets. However, if security is more important, because of the classification of the data, than using a single key per role/classification, that may not be the best security posture. Another factor for key implementations is the federation of features. If multiple authors collaborate to compose features, each author may sign/encrypt features in the collection differently. This may result in the use of different Key Management Services. This testbed only looked at a single agency scenario but future work may include federations.

Chapter 8. Data Encodings, DCS Containers and Media Types

8.1. Introduction

Information security can be applied either on the infrastructure (perimeter-security) or in a data-centric fashion. While infrastructure/transport oriented security standards are integrated in communication infrastructure like in TLS 1.3 [https://tools.ietf.org/html/rfc8446], DCS is fully independent from the underlying communication infrastructure. This is important because (in many security critical applications) the stakeholders cannot rely on the security provided by communication channels (like TLS). With other words, data owners need to remain in charge of data securing when they are distributed in cloud-based resource services. Thus, it appears reasonable to incorporate security concepts in the data sets, which urges expanding the existing data structures for additional elements to carry encrypted data and artefacts such as metadata, security labels, signatures, etc.

In protocols with application-layer intermediaries, channel-based security protocols protect messages from attackers between intermediaries, but not from the intermediaries themselves (not from, for example, malicious applications running on the platforms of intermediaries). These cases require object-based security technologies, which embed application data within a secure object that can be safely handled by untrusted entities as it was described for JSON encoding format (RFC 7165). Data-centric security advocates generation, storing and provision of security related metadata and content such as security tokens (digital identities), policies, keys, signatures, encrypted content and schemas.

In the geospatial domain, data centric security is applied on data encoded using geospatial domain specific grammars (schemas) such as GML or GeoJSON. GML is an XML encoding while the GeoJSON format is encoded in JSON. The previous work in Testbed-15 regarding the DCS was based on GML and the container structure required for DCS was encoded using STANAG 4774/8 and XML binding.

JSON (JavaScript Object Notation) is a well-known XML alternative and widely used data-interchange format. GeoJSON was created as a response to the popularity of JSON encoding format and as an alternative to previously established GML format based on XML and XML Schema. GeoJSON defines several types of JSON objects and the manner in which they are combined to represent data about geographic features, their properties, and their spatial extents. RFC 7946 [https://tools.ietf.org/html/rfc7946] is the current GeoJSON standard.

Additionally to new DCS container formats, new HTTP Accept header content types are proposed for content negotiation. The Accept request HTTP header advertises which content types (container formats) the client is able to understand, which is important for interoperability.

8.2. DCS Container

If the security is required to be applied in the data-centric fashion, an additional data structure needs to carry signature and encryption artefacts. This includes the metadata related to a data origin identity, access rights/policies, and optionally data structure/taxonomy. Two encoding options for data centric securing of (geospatial) content are available. These are XML-GML and GeoJSON based encoding standards with corresponding containers. Two set of standards, NATO STANAG 4774 [https://nso.nato.int/nso/zPublic/ap/PROM/ADatP-4774%20EDA%20V1%20E.pdf] and STANAG 4778 [https://nso.nato.int/nso/zPublic/ap/PROM/ADatP-4778%20EDA%20V1%20E.pdf] (short STANAG 4774/8) and JWT/JOSE were chosen to implement the required container structure. Additional DCS containers and encodings require new media types to describe all useful combinations of data/container encoding and applied security functions.

To enable data exchange and interoperability among NATO Member States, NATO STANAG 4774 and 4778 define a syntax (4774) for trusted security labels / markings and how these are cryptographically bound to data objects (4778) to ensure the integrity of data and the label. Trusted security labels include, for example, data on the creator, creation and expiration date. There are different profiles for REST, SMTP, or SOAP, XMPP or Office Open XML. JSON binding is not supported.

Beside standard security labels like classification, creator or creation and expiration date, in the geospatial domain additional metadata about the spatial scope of the data set is frequently relevant. For example, large, encrypted sets of geographic entities having metadata labels containing a bounding box allow access policies, which constrain data usage based on user location.

For JSON encoded data mostly used in interactions via RESTful APIs there is a family of standards designed to implement confidentiality and integrity in a data centric fashion. These standards are based on JSON Web Token (JWT) and also includes the JWE and JWS specifications, which is known as JOSE [https://tools.ietf.org/html/rfc7165]. JWT is basically seen as the root specification, which the JWE and JWS were derived from.

8.3. STANAG 4774/8 DCS Container

STANAG 4774/8 was used to implement the Testbed-15 DCS architecture with trusted security labels. Currently, the standard fully supports XML binding. Other representations, for example in JSON, would be possible (JSON is currently not supported). The full STANAG 4774/8 data structure is depicted in the following figure:

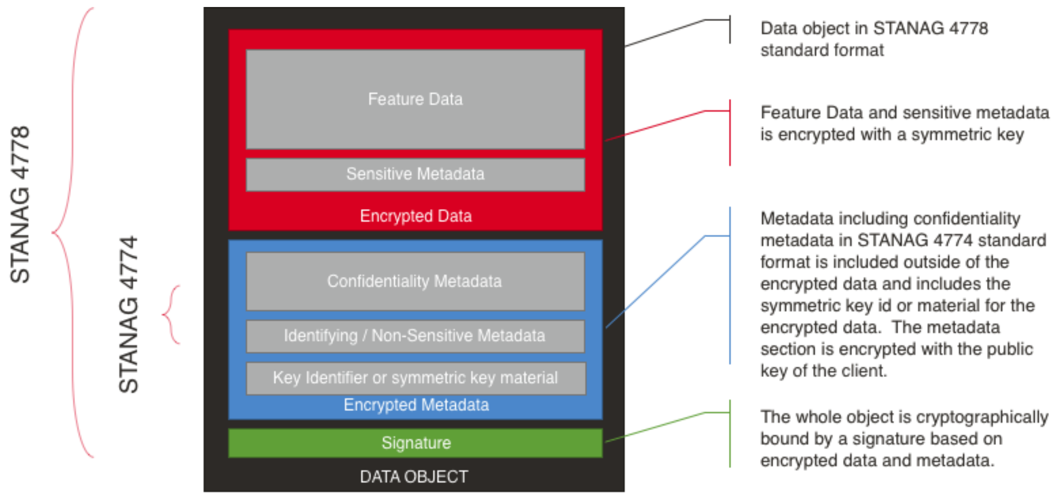

STANAG 4778
STANAG 4774
Feature Data
Sensitive Metadata
Encrypted Data
Confidentiality Metadata
Identifying / Non-Sensitive Metadata
Key Identifier or symmetric key material
Encrypted Metadata
Signature
DATA OBJECT
Data object in STANAG 4778 standard format
Feature Data and sensitive metadata is encrypted with a symmetric key
Metadata including confidentiality metadata in STANAG 4774 standard format is included outside of the encrypted data and includes the symmetric key id or material for the encrypted data. The metadata section is encrypted with the public key of the client.
The whole object is cryptographically bound by a signature based on encrypted data and metadata.

Figure 9. STANAG DCS Container Structure

The figure displays the container structure from testbed-15, which holds a key (symmetric) required for data decryption. The key is protected inside of the encrypted metadata section and can be extracted only if it were encrypted with the receiver’s public key, which means the encryption was performed using public cryptography (PKI). This approach has certain security limitations and inflexibility. In Testbed-16 that will be mitigated through a new component - the KMS responsible to issue decryption keys on request (for a given key_id encoded in a DCS container). Instead of encrypted keys, their identifiers (key_id) originating from the KMS will be placed in DCS containers.

8.4. STANAG 4774/8 DCS Container in JSON

Despite the fact that only XML binding specifications were provided for STANAG 4774/8 so far, there is another, (from the interoperability point of view) useful option - to introduce an equivalent binding in JSON for clients, services and APIs based on that technology. In such a scenario, JWS (encryption) and JWT (signature) standards (also called JOSE) are used instead of JSON-encoded STANAG 4778 content to ensure the integrity of the payload and security label, while STANAG 4774 will provide a blueprint for security label information and other meta-data of interests.

8.5. JOSE (JWS & JWE) based containers for JSON

In the JavaScript/JSON ecosystem the communication is secured on a data level by applying a set of standards such as JWT and JOSE (combination of encryption and signature via JWS and JWE). The standards provide a structure intended to capture the metadata and artefacts required for standard security functions such as confidentiality and integrity. The following table gives an overview of the security stack based on JWT and JOSE:

- JavaScript Object Singing and Encryption (JOSE)
  - JSON Web Signature (JWS)
    - A way of representing content secured with a digital signature (or MAC) using JSON data structures and base64url encoding
  - JSON Web Encryption (JWE)
    - Like JWS but for encrypted content
  - JSON Web Key (JWK)
    - JSON data structures representing cryptographic keys
- JSON Web Token
  - Defines the use of cryptographic algorithms and identifiers for JWS, JWE and JWK
  - A compact URL safe means to represent claims/attributes to be transferred between two parties
  - A JWT is a JWS and/or a JWE with JSON claims as a payload

A JSON security stack built around the standards listed above visualizes the relations between standards. While everything is encoded in JSON, JWE and JWS represent standard cryptographic operations related to encryption and signing while JWT deals with digital identities.

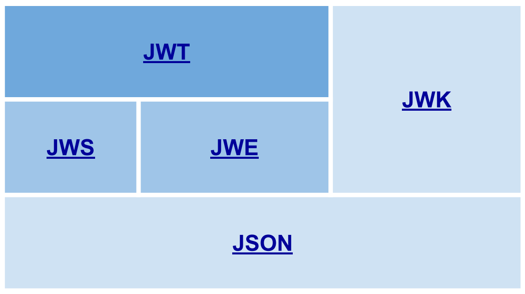


Figure 10. JSON security stack

When Object Signing and/or Encryption (JOSE) is used to protect a payload, the resulting structure (as depicted on the figure below) establishes a type of container holding both the payload and the metadata. While JWS requires a fairly simple format to ensure the integrity, JWE requires additional attributes to support the confidentiality through the encryption. The payload could be any geospatial content, for example GeoJSON. The figure represents the container encoding with comma separated sections. An alternative would be to use a full, slightly more complex JSON representation.

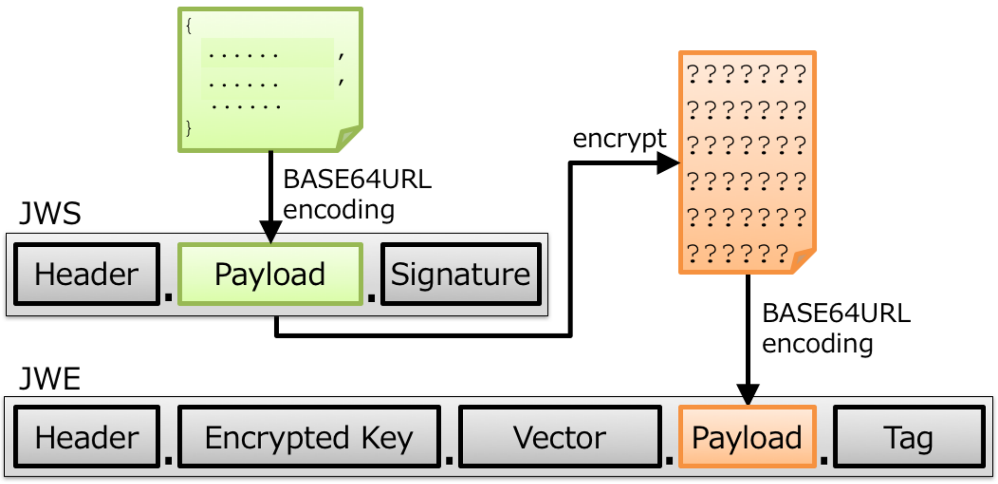


Figure 11. JOSE DCS container structure

For JWS the payload is first signed and enclosed in a data structure defined in RFC 7515. The structure has:

- Header
- Payload
- Signature

The header containing the signature metadata, the payload holds the base64 encoded protected content and the signature ensures integrity (or that header and payload are cryptographically bound to each other). The payload segment might also enclose the additional metadata information.

If the confidentiality of data is required, the plaintext data can be encrypted and wrapped up in a container structure based on JWE (RFC 7516). The container will have the structure according to the specification containing the following parts:

- Header

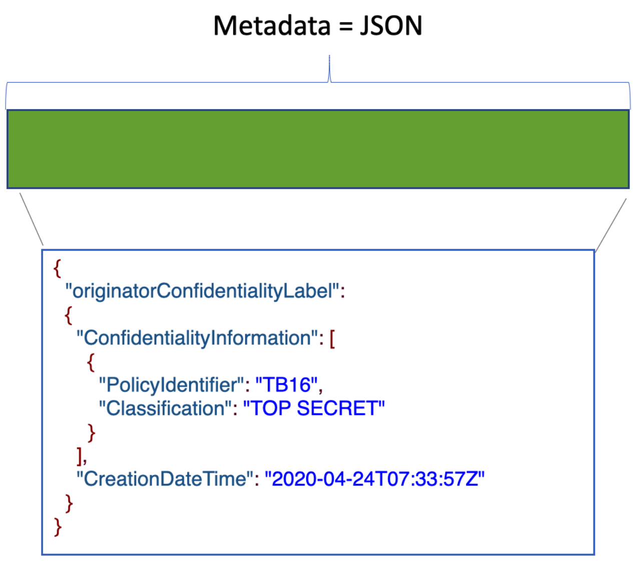
- Encrypted key
- Initialization vector
- Ciphertext
- Authentication tag

Ciphertext is the section where encrypted data payload (created out of original payload plaintext) is placed. Other segments such as initialization vector or authentication tag are populated according to the specification and in order to support data integrity.

The information to protect remains encoded following STANAG 4774/8 but in JSON (binding). The JOSE would provide additional cryptographic protection (ensuring integrity) for such a DCS container in JSON encoding. Following are the possible container forms or the combination of "JOSE for security implementation" and "STANAG to encode the meta-data":

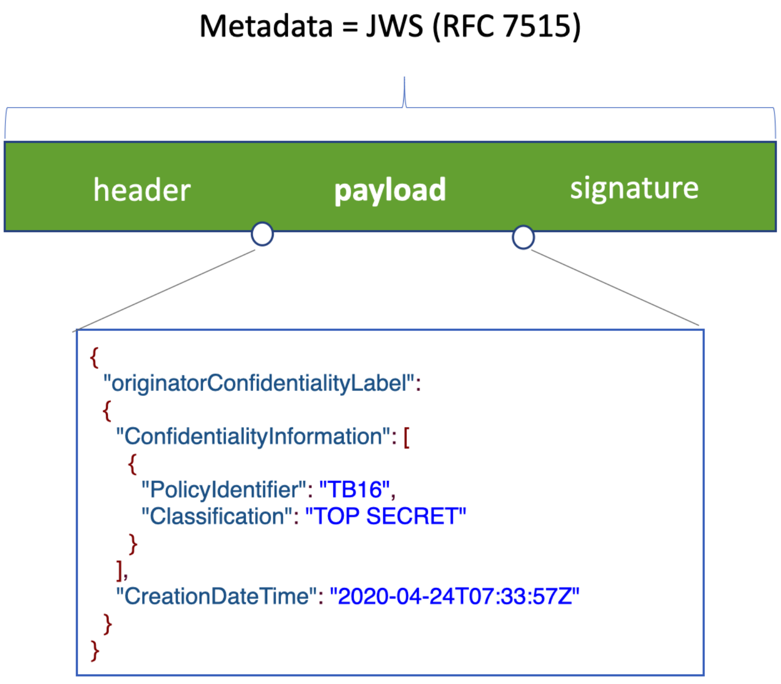
- Metadata = Plain JSON
- Metadata = JWS (RFC 7515)
- Metadata = JWE (RFC 7516)

The options are depicted in the following three figures. The green bar represents the overall information (data-centric container with all protection measures). Metadata as shown here is STANAG encoded in JSON:



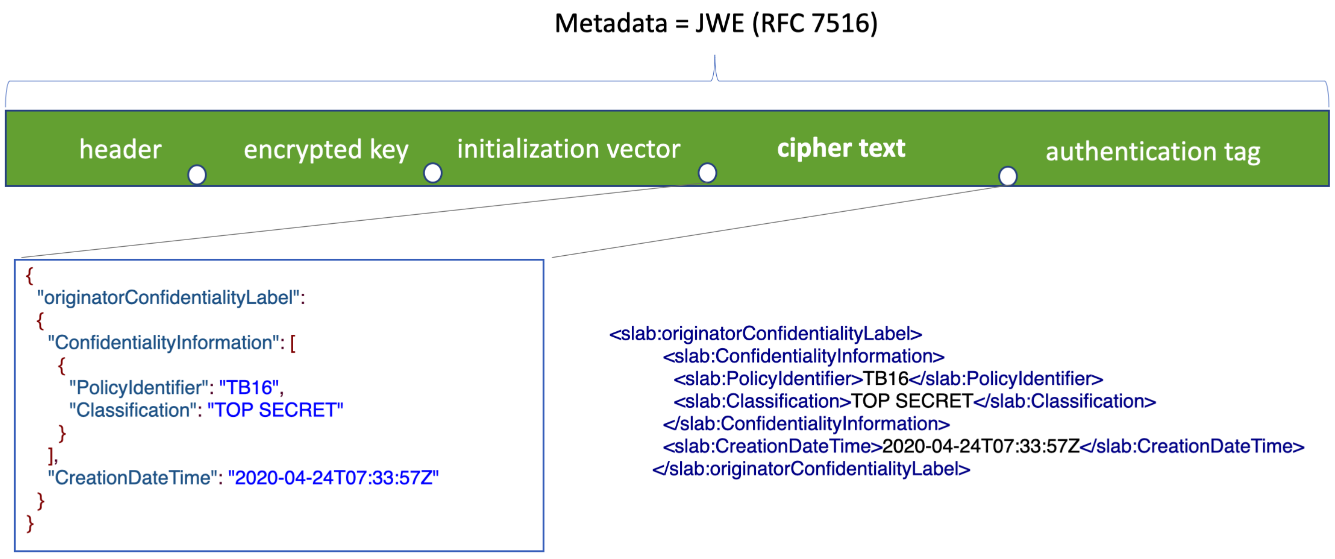
```
{
  "originatorConfidentialityLabel":
  {
    "ConfidentialityInformation": [
     {
       "PolicyIdentifier": "TB16",
       "Classification": "TOP SECRET"
     }
    ],
    "CreationDateTime": "2020-04-24T07:33:57Z"
  }
}
```

Figure 12. Plain STANAG 4774 metadata encoded in JSON



```
{
  "originatorConfidentialityLabel":
  {
    "ConfidentialityInformation": [
      {
        "PolicyIdentifier": "TB16",
        "Classification": "TOP SECRET"
      }
    ],
    "CreationDateTime": "2020-04-24T07:33:57Z"
  }
}
```

Figure 13. Signed STANAG 4774 metadata encoded in JSON in JWS container



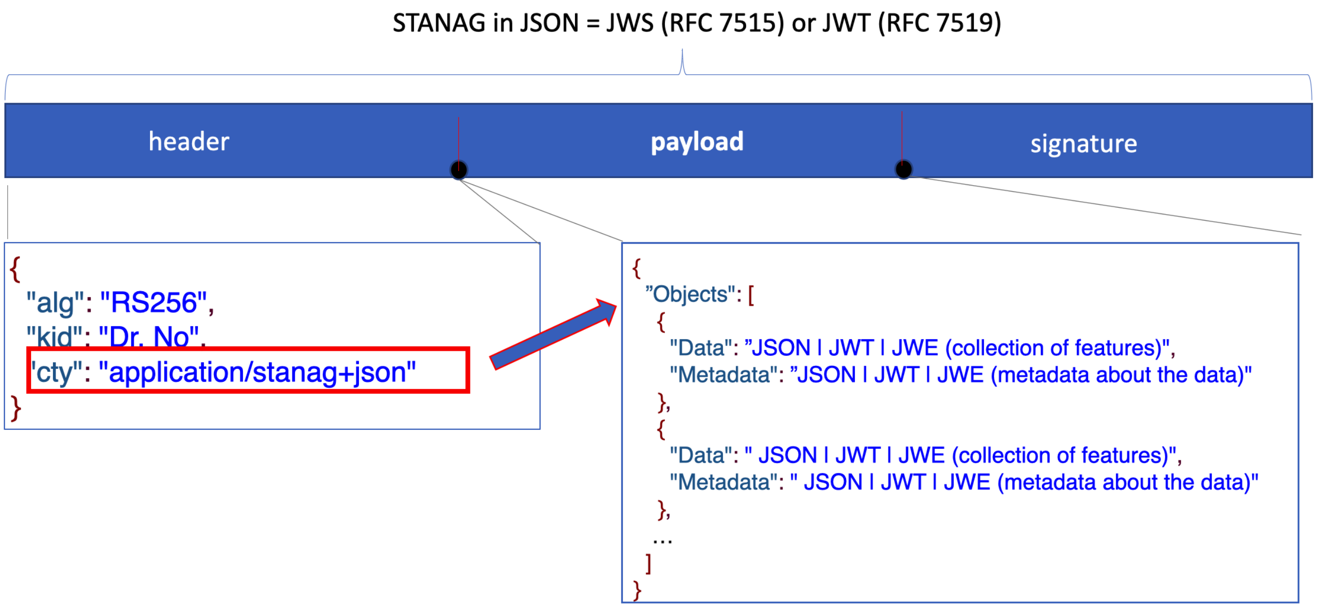
```
{
  "originatorConfidentialityLabel":
  {
    "ConfidentialityInformation": [
      {
        "PolicyIdentifier": "TB16",
        "Classification": "TOP SECRET"
      }
    ],
    "CreationDateTime": "2020-04-24T07:33:57Z"
  }
}
```

```
<slab:originatorConfidentialityLabel>
        <slab:ConfidentialityInformation>
          <slab:PolicyIdentifier>TB16</slab:PolicyIdentifier>
          <slab:Classification>TOP SECRET</slab:Classification>
        </slab:ConfidentialityInformation>
        <slab:CreationDateTime>2020-04-24T07:33:57Z</slab:CreationDateTime>
      </slab:originatorConfidentialityLabel>
```

Figure 14. Encrypted STANAG 4774 metadata encoded in JSON in JWE container

8.5.1. DCS Container based on JWS

The JWS container implements a signature mechanism to protect the integrity of payload and related meta-information and bind them together cryptographically. The container consists of three parts. The following figure depicts the structure. The payload section contains STANAG 4774 metadata and the information. The header section holds metadata about the signature algorithm and media type (context). For example, the attribute "ctx" (context) is defined as `application/stanag+json`. This represents a new media type used to identify the server response containing the mix of payload and STANAG metadata.

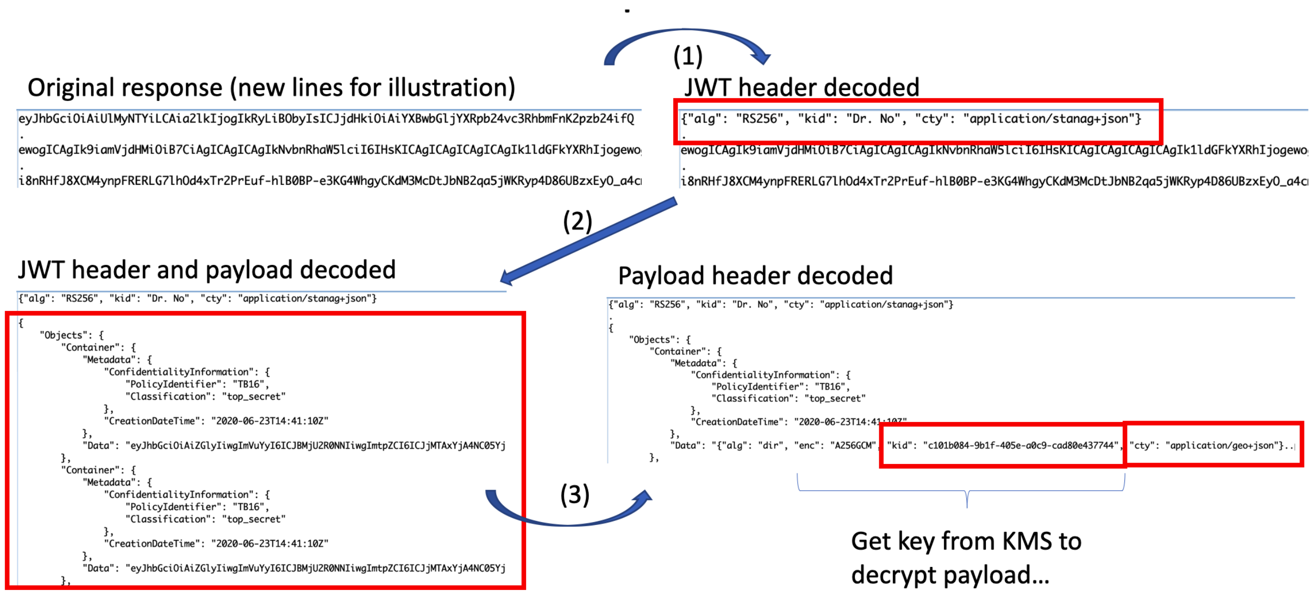


```
{
  "alg": "RS256",
  "kid": "Dr. No",
  "cty": "application/stanag+json"
}
```

```
{
  "Objects": [
    {
      "Data": "JSON I JWT I JWE (collection of features)",
      "Metadata": "JSON I JWT I JWE (metadata about the data)"
    },
    {
      "Data": " JSON I JWT I JWE (collection of features)",
      "Metadata": " JSON I JWT I JWE (metadata about the data)"
    },
    …
  ]
}
```

Figure 15. STANAG metadata in JWS container with signature

The workflow related to the DCS container based on JWS includes the integrity validation (signature verification), conversion back from Base64 format, and extracting the key identification values. These are used to retrieve the keys for decryption from the KMS server component:



Original response (new lines for illustration)

```
eyJhbGciOiAiUlMyNTYiLCAia2lkIjogIkRyLiBObyIsICJjdHkiOiAiYXBwbGljYXRpb24vc3RhbmFnK2pzb24ifQ
.
ewogICAgIk9iamVjdHMiOiB7CiAgICAgICAgIkNvbnRhaW5lciI6IHsKICAgICAgICAgICAgIk1ldGFkYXRhIjogewo
.
i8nRHfJ8XCM4ynpFRERLG7lhOd4xTr2PrEuf-hlB0BP-e3KG4WhgyCKdM3McDtJbNB2qa5jWKRyp4D86UBzxEyO_a4c
```

JWT header decoded

```
{"alg": "RS256", "kid": "Dr. No", "cty": "application/stanag+json"}
.
ewogICAgIk9iamVjdHMiOiB7CiAgICAgICAgIkNvbnRhaW5lciI6IHsKICAgICAgICAgICAgIk1ldGFkYXRhIjogewo
.
i8nRHfJ8XCM4ynpFRERLG7lhOd4xTr2PrEuf-hlB0BP-e3KG4WhgyCKdM3McDtJbNB2qa5jWKRyp4D86UBzxEyO_a4c
```

JWT header and payload decoded

```
{"alg": "RS256", "kid": "Dr. No", "cty": "application/stanag+json"}
{
    "Objects": {
        "Container": {
            "Metadata": {
                "ConfidentialityInformation": {
                    "PolicyIdentifier": "TB16",
                    "Classification": "top_secret"
                },
                "CreationDateTime": "2020-06-23T14:41:10Z"
            },
            "Data": "eyJhbGciOiAiZGlyIiwgImVuYyI6ICJBMjU2R0NNIiwgImtpZCI6ICJjMTAxYjA4NC05Yj
        },
        "Container": {
            "Metadata": {
                "ConfidentialityInformation": {
                    "PolicyIdentifier": "TB16",
                    "Classification": "top_secret"
                },
                "CreationDateTime": "2020-06-23T14:41:10Z"
            },
            "Data": "eyJhbGciOiAiZGlyIiwgImVuYyI6ICJBMjU2R0NNIiwgImtpZCI6ICJjMTAxYjA4NC05Yj
        },
```

Payload header decoded

```
{"alg": "RS256", "kid": "Dr. No", "cty": "application/stanag+json"}
.
{
    "Objects": {
        "Container": {
            "Metadata": {
                "ConfidentialityInformation": {
                    "PolicyIdentifier": "TB16",
                    "Classification": "top_secret"
                },
                "CreationDateTime": "2020-06-23T14:41:10Z"
            },
            "Data": "{"alg": "dir", "enc": "A256GCM", "kid": "c101b084-9b1f-405e-a0c9-cad80e437744", "cty": "application/geo+json"}..
        },
```

Figure 16. Container parsing/decryption workflow

8.5.2. Structure of Information as Metadata

Extending the metadata section of a DCS container for payload structure is useful. For example, the XML encoding is meant to include XML schema elements, which highlights the payload structure.

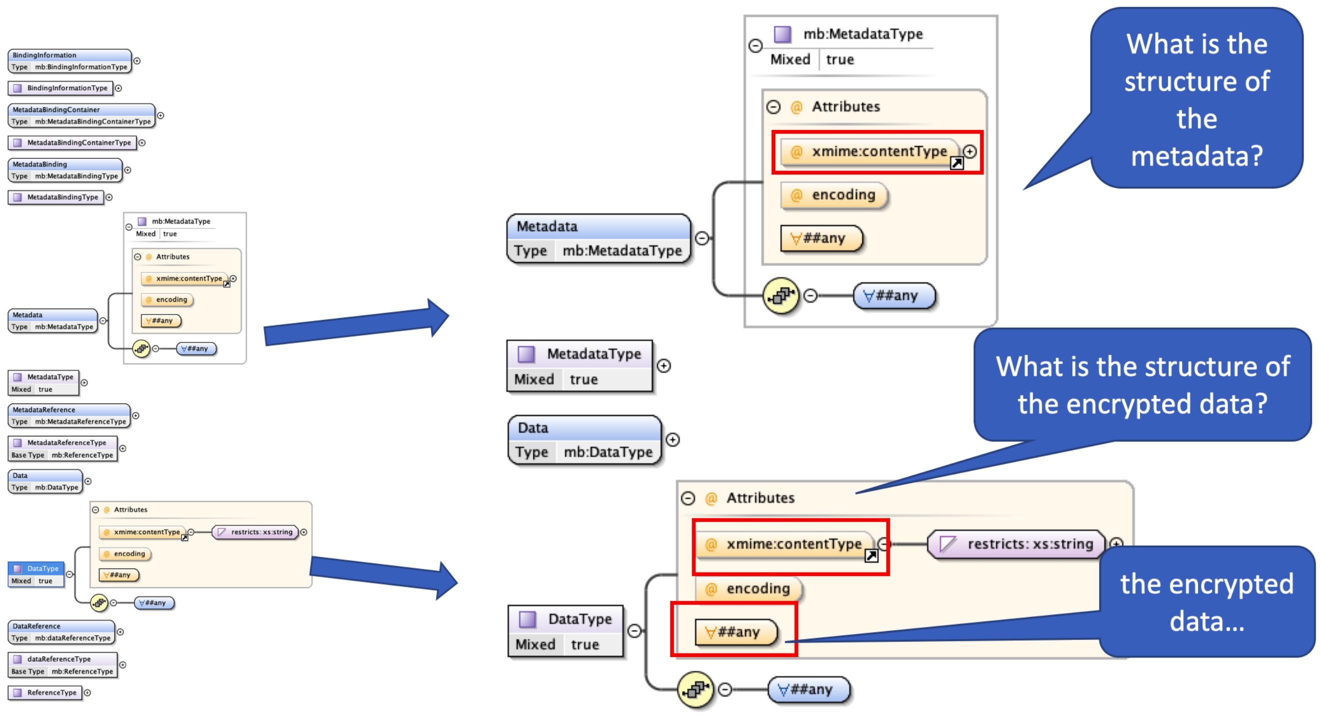
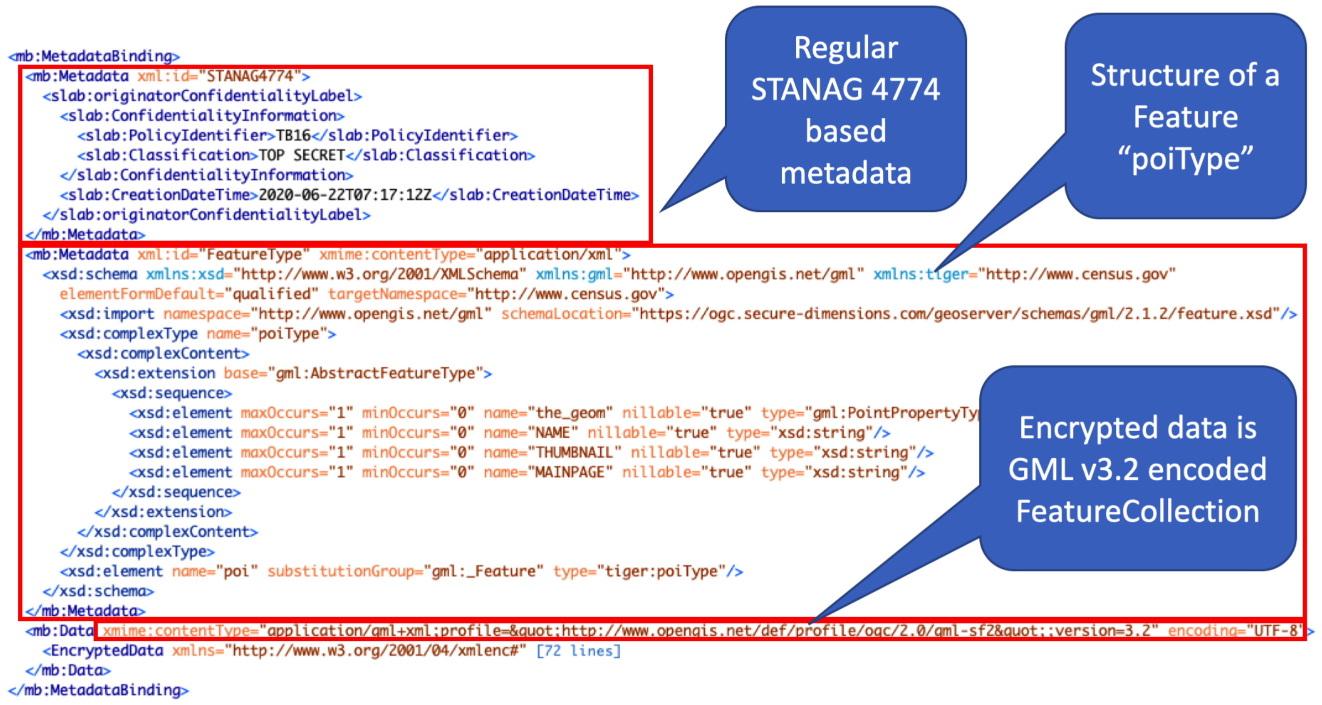


Figure 17. STANAG container encoded in XML

In the figure below the schema information for entity type "poiType" is framed in red in the middle. This enables the access decisions based on the structure/taxonomy of protected information. Of course, this approach could be extended to JSON encoding and corresponding JSON schemas.


Regular STANAG 4774 based metadata
Structure of a Feature "poiType"
Encrypted data is GML v3.2 encoded FeatureCollection

```xml
<mb:MetadataBinding>
  <mb:Metadata xml:id="STANAG4774">
    <slab:originatorConfidentialityLabel>
      <slab:ConfidentialityInformation>
        <slab:PolicyIdentifier>TB16</slab:PolicyIdentifier>
        <slab:Classification>TOP SECRET</slab:Classification>
      </slab:ConfidentialityInformation>
      <slab:CreationDateTime>2020-06-22T07:17:12Z</slab:CreationDateTime>
    </slab:originatorConfidentialityLabel>
  </mb:Metadata>
  <mb:Metadata xml:id="FeatureType" xmime:contentType="application/xml">
    <xsd:schema xmlns:xsd="http://www.w3.org/2001/XMLSchema" xmlns:gml="http://www.opengis.net/gml" xmlns:tiger="http://www.census.gov"
      elementFormDefault="qualified" targetNamespace="http://www.census.gov">
      <xsd:import namespace="http://www.opengis.net/gml" schemaLocation="https://ogc.secure-dimensions.com/geoserver/schemas/gml/2.1.2/feature.xsd"/>
      <xsd:complexType name="poiType">
        <xsd:complexContent>
          <xsd:extension base="gml:AbstractFeatureType">
            <xsd:sequence>
              <xsd:element maxOccurs="1" minOccurs="0" name="the_geom" nillable="true" type="gml:PointPropertyTy
              <xsd:element maxOccurs="1" minOccurs="0" name="NAME" nillable="true" type="xsd:string"/>
              <xsd:element maxOccurs="1" minOccurs="0" name="THUMBNAIL" nillable="true" type="xsd:string"/>
              <xsd:element maxOccurs="1" minOccurs="0" name="MAINPAGE" nillable="true" type="xsd:string"/>
            </xsd:sequence>
          </xsd:extension>
        </xsd:complexContent>
      </xsd:complexType>
      <xsd:element name="poi" substitutionGroup="gml:_Feature" type="tiger:poiType"/>
    </xsd:schema>
  </mb:Metadata>
  <mb:Data xmime:contentType="application/gml+xml;profile=&quot;http://www.opengis.net/def/profile/ogc/2.0/gml-sf2&quot;;version=3.2" encoding="UTF-8">
    <EncryptedData xmlns="http://www.w3.org/2001/04/xmlenc#" [72 lines]
  </mb:Data>
</mb:MetadataBinding>
```

Figure 18. STANAG XML with content schema type

8.5.3. DCS Container based on JWE

The DCS container structure based on JWE contains sections required to store encrypted content (ciphertext), metadata (header) and additional elements relevant for applied crypto algorithms. Encrypted key sections are left empty because the JWE key is not required here. Instead, the header's attribute "alg" signalizes "Direct Encryption with a Shared Symmetric Key". According to JWA [https://tools.ietf.org/html/rfc7518] in this case, the shared symmetric key is used directly as the Content Encryption Key (CEK) value for the "enc" algorithm. An empty octet sequence is used as the JWE Encrypted Key value. The "alg" (algorithm) Header Parameter value "dir" is used in this case. Attribute "kid" holds the key identification value for symmetric key retrieval from the KMS.

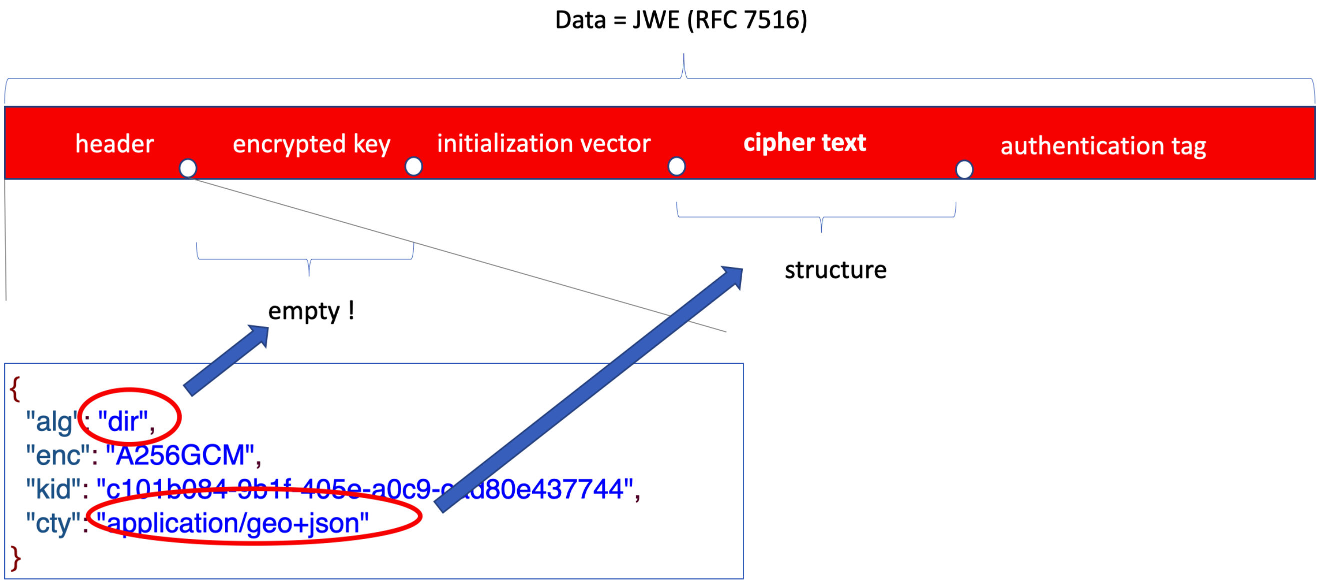


Figure 19. Encrypted payload enclosed in JSON container based on JWE

The basic structure for DCS container is defined by STANAG 4778 but can also be encoded in JSON:

- **XML:**
  - 1..* Metadata elements (encrypted or decrypted, contains key_id if encrypted).
  - 1 Data element (encrypted or decrypted, contains key_id if encrypted).
  - Signature (to keep the integrity of everything).
- **JSON:**
  - 1..* Metadata elements (encrypted or decrypted, contains key_id if encrypted).
  - 1 Data element (encrypted or decrypted, contains key_id if encrypted).
    - Signature (to keep the integrity of everything) OR,
    - Signature in JWS container OR,
    - Encryption in JWE container.

For the collection of entities, they can either be encrypted all together and put in the "Data" section of STANAG 4774/8 or they can each be encrypted separately in a dedicated STANAG object.

In other words, JWS provides signatures to ensure the integrity of DCS containers. JWS ensures confidentiality for DCS containers. The container does not contain keys required to decrypt the payload or the metadata. Instead, key identification attributes are populated, either inside of STANAG 4778 structure or put in the JOSE headers (attribute kid) and used to retrieve the key from KMS.

8.6. Media Types and profiles for DCS content negotiation

There are a variety of approaches to implementing the content negotiation as described in (RFC 7231 [https://tools.ietf.org/html/rfc7231#page-18]) for a RESTful API. Two common options are:

- Specify the content type in the URI (`/geojson/streets/42`).
- Specify the content type using a query parameter (`/streets/42?type=geojson`).

However, both of these non-HTTP content negotiation examples violate the rule saying that a REST API should not "include artificial extensions in URIs to indicate the format of a message's entity body". Instead, they should rely on the media type, as communicated through the Content-Type header, to determine how to process the body's content.

The proper way for content negotiation is to specify the content type of an HTTP response as a parameter of the media type in the HTTP request. This is the option that is selected here for DCS. This method avoids changing URIs and makes use of an existing HTTP header rather than creating a custom one. A Media Type describes the content of an HTTP request or response such that the service provider knows how to handle the request. In short, service requests use media types in order to notify the service (content provider with DCS) which combination of encoding and container structure is required.

When a client issues an HTTP request, it can indicate what media types the client would prefer to receive by using the Accept HTTP header. For example, GeoJSON has a (generic) media type "application/geo+json" for all resources. In order to support different encodings and DCS containers, media types need to be defined, such as `application/gml+stanag`.

How many different media types are needed to cover all useful type options for responses? The following list presents the encodings and container structures that appear to be useful in the context of this testbed:

- **Content Encodings:**
  1. XML
  2. GML
  3. GEO+JSON
  4. JSON
- **Container Types:**
  1. STANAG 4774/8
  2. STANAG 4774/8 in JWS
  3. STANAG 4774/8 in JWE

Not all possible combinations for encoding and content types as listed here make sense or are useful. First of all, the assumption is a STANAG 4774/8 container structure is taxonomy for DCS container structure. New supported encoding will be provided in JSON. When the content is encoded in JSON, using JWS and JWE "containers" for singing and/or encrypting appears to be the choice. JWS is used to ensure the integrity. Otherwise, XML encoded GML content is enclosed in the container based on the STANAG 4774/8 binding for XML. XML Signature (XML Signature defines an XML syntax for digital signatures) is used to ensure the integrity. The figure below provides an example depicting several options for content negotiation:

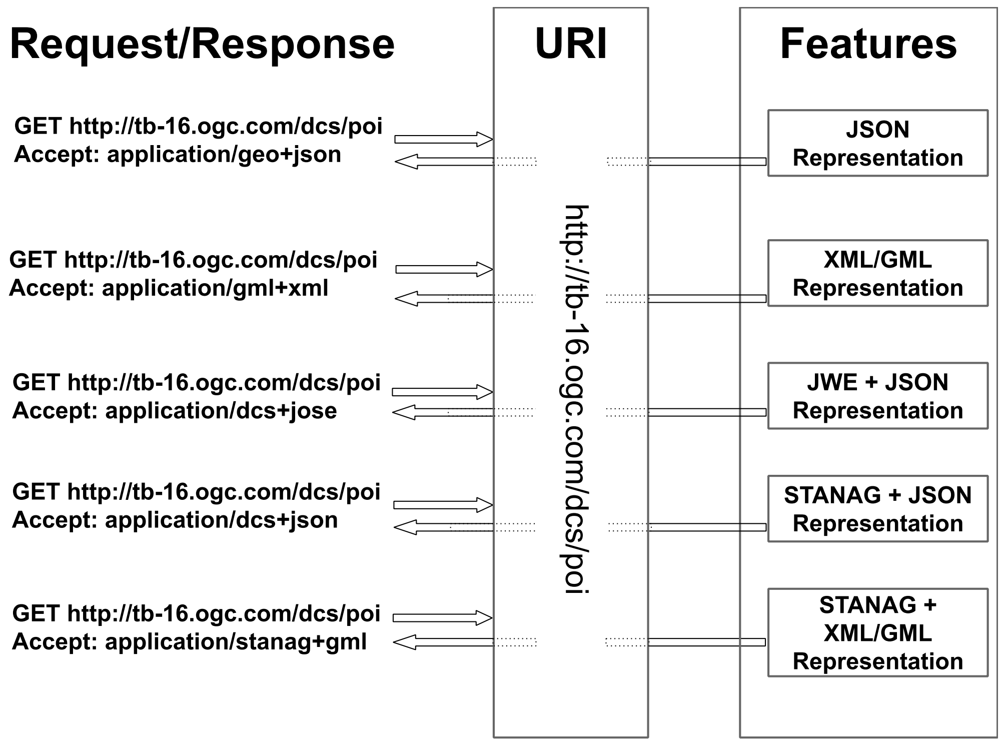

Request/Response
URI
Features
GET http://tb-16.ogc.com/dcs/poi
Accept: application/geo+json
GET http://tb-16.ogc.com/dcs/poi
Accept: application/gml+xml
GET http://tb-16.ogc.com/dcs/poi
Accept: application/dcs+jose
GET http://tb-16.ogc.com/dcs/poi
Accept: application/dcs+json
GET http://tb-16.ogc.com/dcs/poi
Accept: application/stanag+gml
http://tb-16.ogc.com/dcs/poi
JSON Representation
XML/GML Representation
JWE + JSON Representation
STANAG + JSON Representation
STANAG + XML/GML Representation

Figure 20. Content negotiation example for DCS aware GC API implementation

Another aspect of HTTP content negotiation, which introduces fine grained control over the response formats is related to the possible use of a Media Type parameter "profile" in HTTP header. As described in a W3C Working Draft Content Negotiation by Profile [https://www.w3.org/TR/dx-prof-conneg/], clients may negotiate for content provided by servers based on so called "data profiles" to which the content conforms. This is "distinct from negotiating by Media Type or Language: a profile may specify the content of information returned, which may be a subset of the information the responding server has about the requested resource, and may be structured in a specific way to meet interoperability requirements of a community of practice". An Application Profile bundles several specifications and possibly adds additional requirements on an implementation. Extra requirements can be interpreted either as additions or as constraints. For the Testbed-16 case, the media type could be used to specify required serialization such as XML, JSON, GML, GeoJSON, while profile parameter could be used to further differentiate between different DCS container options (STANAG 4774/8, JOSE).

Following profiles are useful to refine the content negotiation in DCS:

Table 1. Profiles

Profile	**Description**
http://www.opengis.net/spec/GML/3.X/req/dcs/stanag4778	Profile extension to **application/gml+xml** that supports STANAG 4778 notation for feature types such as <gmlce:SimplePolygon dcs=”stanag4778”>

Profile	Description
http://www.opengis.net/spec/GeoJSON/xx/req/dsc/jwt_jwe	Profile extension to **application/geo+json** that supports a JWT or JWE notation for feature types such as ”type”: “jwt” or “type”: “jwe” with a data element.
http://www.opengis.net/def/profile/ogc/2.0/gml-sf2	Profile extension to specify GML 3.2 SF-2 compliance.

Since the Accept header can contain multiple media types, clients can set alternative profiles and media types in the HTTP header for specific DCS protected geospatial content. Besides new media types to set this preference, the q parameter (relative quality factor) is used. The value of a q parameter can be from 0 to 1. 0 represents the least preferred while 1 is the most preferred choice.

HTTP GET with media types and profile parameters

```
GET /resource/a HTTP/1.1
Accept: application/dcs+geo;q=0.9;profile=ogc:sf:in:geojson, \

application/geo+dcs;q=0.7;profile=urn:nato:stanag:4778:bindinginformation:1:0:in:JSON
...

HTTP/1.1 200 OK
Content-Type: application/dcs+geo;profile=ogc:sf:in:geojson
```

Finally, DCS media types and profile parameters for content negotiation as proposed for XML and JSON encoded content are listed in these tables:

Table 2. XML related media types

Description	OGC API Parameter ‘f’	HTTP Accept / Content-Type Header
GML FC as defined by OGC	**xml or gml**	application/gml+xml; profile="http://www.opengis.net/def/profile/ogc/2.0/gml-sf2";version=3.2
GML FC where each feature is a STANAG 4778 data object	**gml+dcs**	application/gml+dcs; profile="http://www.opengis.net/def/profile/ogc/1.0/stanag#4778";
STANAG 4778 data object (container) containing an encrypted GML FC	**dcs+gml**	application/dcs+gml; profile="http://www.opengis.net/def/profile/ogc/2.0/gml-sf2";version=3.2

Table 3. JSON related media types

Description	OGC API Parameter ‘f’	HTTP Accept / Content-Type Header
Feature Collection in GeoJSON	**json or geo+json**	application/geo+json
Feature Collection in GeoJSON signed or encrypted	**jws or geo+jose**	application/geo+jose
STANAG 4778 in JSON encrypted or signed where data objects are GeoJSON encoded features	**dcs+geo**	application/dcs+geo;profile=ogc:sf:in:geojson
Feature Collection in GeoJSON where features are STANAG 4778 JSON encoded	**geo+dcs**	application/geo+dcs;profile=urn:nato:stanag:4778:bindinginformation:1:0:in:JSON

Testbed-16 demonstrated content negotiation and data retrieval with the following container/encoding variants:

- **STANAG+GML** returns the STANAG 4778 encoded and encrypted data in XML encoding. Each data object is a feature instance encoded in GML.
- **STANAG+JSON** returns the STANAG 4778 structure encoded in JSON. Each data element is an encrypted feature instance encoded in GeoJSON.
- **STANAG+JWS** returns the STANAG 4778 structure encoded in JSON with digital signature (JWT format). Each data element is an encrypted feature instance encoded in Geo+JSON.
- **GeoJSON+JWS** returns the digitally signed feature collection encoded in GeoJSON.

Examples for content types (HTTP GET) are and DCS containers are given in Appendix Engineering Aspects for D120 and D145 under Requesting encrypted data.

Chapter 9. Results

The Testbed participants were able to demonstrate that with a proper DCS security architecture put in place and having a KMS component responsible for key management, an implementation can satisfy the requirements for an extended data centric security model for both desktop and mobile clients. The following list summarizes the results achieved in this testbed:

- Effective data-centric security solution with standardized OGC API - Features + KMS + new HTTP Accept header media type and profile values for content negotiation.
- Protected data sets in different formats encoded using XML or JSON and by applying standards such as STANAG 4774/8 and/or JOSE to model DCS containers for metadata, cryptographic artefacts and payloads.
- As the ER evaluates above, a DRM architecture can be the motivation when implementing DCS for a geospatial domain because of similarities with DCS objectives.
- With the presentation sufficient rights, a server/client encrypts/decrypts the data on the fly as part of a synchronous API request.
- Client curation tools can encrypt data sets and stored them locally on a mobile device for decryption at a later time. The thought behind this is that data curation may take place much earlier than when the user intends to use the data. Clients may have a need to store the data over larger periods of time than an online desktop scenario would offer. This requires a set of policies to control when and how a user of a client may use the data, and the level of protection the client provides. The experiments confirm the curation of data can occur with differing protection levels and that clients can select user profiles to enforce in the field.
- Client implementations can model differing concepts of data classification and user/role-based clearances.

Performing the experiment did uncover some issues while attempting to implement the scenarios. The following is a list of the issues the experiments came across:

- The GIS client applications for the desktop and the Android mobile client rely heavily on third party open source libraries which increase the difficulty of implementing decryption of features.
- Mobile application threading and UI responsiveness requirements make implementing long-running synchronous tasks difficult.
- Large data sets create a burden on network bandwidth and take a long time to encrypt features.
- Encryption of large data items can timeout across network connections.
- Mobile Process / Power Management

9.1. Issue Explanations

9.1.1. Third Party Open Source Library implementations impede the implementation of decryption

Both QGIS and QField rely heavily on GDAL which is an open source tool for retrieving and manipulating geographic data formats. In this experiment the implementor had to make a choice about modifying GDAL or modifying the GIS application to perform the decryption of the data. Both QGIS and QField are open source projects as well. The burden on modifying multiple open source applications and libraries is an educational barrier to implementing customizations.

An implementor must learn about each open source project and weigh the consequences of where to make modifications. In this experiment the implementor choose to limit the modifications to QGIS and QField. However, modifying GDAL would have made the modifications to QGIS and QField easier to accomplish. However, modifying GDAL would take more thought as GDAL is used in many open source and proprietary software solutions. The need for specifying asynchronous key pairs, for signing content, and integrating with various Key Management Services could pose a real challenge for developers trying to use GDAL to do data centric security. This may also be something to consider in a future testbed.

9.1.2. Mobile Application and long-running synchronous operations

Mobile devices run an Event loop to process UI and environmental events. For example, the simple act of rotating a device triggers a series of events that destroy views and recreate new views. A long-running task may need to update UI elements that no longer exist. To increase the challenge a developer faces, many APIs for fetching file and making network requests are potentially long-running tasks in themselves and as such must be run asynchronously. This makes synchronous tasks hard to implement as they may require the fetching of multiple files or make multiple network requests.

In this experiment, the Android GeoPEP application needs the ability to communicate with a KMS to fetch encryption keys, HMAC verification keys, and encrypt data. The application must fetch the keys, then encrypt the data, and finally produce a JSON Web Encryption (JWE) data set. That is a synchronous activity as the encryption of content requires the creation of keys. However, the fetching of keys and the performing of the encryption, in this experiment, are done asynchronously by an API for HTTP requests. This means that the responses come back using a callback method.

To overcome the asynchronous fetching of data across the network, the application needs to implement some form of thread suspension to wait for results from one task before doing the next task. As a reader of this engineering report, you may ask why not just handle the next step in the callback from the network request. The implementor of the Android application had the same thought, however, the APIs for implementing network requests will not process further network requests until the previous ones have completed. This prevents you from making another network request in the thread of the callback, which is needed in the case of creating the encryption key and then calling the KMS to encrypt the content. Perhaps a different implementation of a Data Centric Security Key Management Service API vs. a standard Key Management Service API may provide an optimized solution to overcome some of these asynchronous issues of mobile applications.

9.1.3. Timeout Issues with Large Data Requests

During the development of the Android GeoPEP Application, the implementor ran across a timeout issue with encrypting the JSON Web Key (JWK) set. A large feature class containing thousands of features can create a key set that contains two keys for each feature (Encryption Key and HMAC key). Encrypting the key set for storage on the mobile devices may require large data structure. The application passes this data structure as the payload in a post request for a KMS to encrypt. This may result in the client connection timing out due to settings on the client/server presenting the client application with an error condition that is unrecoverable from the standpoint of completing the encryption task. A fallback position for the client maybe to do the encryption task itself; however, in this experiment that was not done.

9.1.4. Mobile Process / Power Management

Process and Power Management on mobile devices in general, and on iOS/iPhones/Apple Watches in particular, can add additional complications to the previously detailed issues. Apple aggressively controls processes running on devices to save power and memory resources, and will terminate processes which seem idle (perhaps waiting for a return call) or are utilizing too many resources. Because of this, an architectural decision was made for the iOS client to implement the PEP as a module within the iOS DCS App instead of a separate running service, as running a PEP web service in the background on the iPhone is problematic at best.

Chapter 10. Future Work

As identified by the Testbed-16 participants, an area of future work revolves around the use of federated security with basic DCS. Another area of future work stems from the importance participants place on the exploration of extending OGC APIs for DCS support. Future activities should also cover the packaging of protected data and other relevant artifacts such as policies. Consideration should further be given to the possible creation of a standardized KMS API to support DCS relevant functions. The next testbed may wish to examine additional media types for data centric protected binary data. The following list describes potential future activities:

- Full CRUD operations on “data centric” secured data sets, including creation of new entities and/or update of already existing entities (which implies the application of corresponding security functions).
- Federated security and DCS (data centric secured content transferred between different security domains and identities, keys etc. transformation/negotiation).
- Standardized KMS API for DCS.
- Packaging of data in DCS: Providing a standardized approach for packaging of additional data such as policies, keys etc. in the scope of DCS.
- Binary related Media Types: Geospatial payloads such as maps, tiles and coverages may also be secured on the data level including GeoTIFF, TIFF, JP2, GMLJP2, etc.
- DCS Roles and User Clearances vs Data Classification(s).

10.1. New features in DCS

The addition of a new feature to a feature collection has ramifications in a Data Centric Approach. Should the client or server perform the encryption? Which Key Management Service will handle the creation of keys and encryption of the content? Do the individual contributors sign the content? If the client encrypts/signs the content, how do servers support queries on the content? All these questions and perhaps more require some investigation and experimentation to determine how the creation of features fits the Data Centric Security Concept.

10.2. KMS for DCS

This activity could include the investigation of the specification of existing/new APIs for key management to enable standardization of CRUD for cryptographic keys within and outside of federated environments. Focusing on access control, an experiment could investigate how to control key creation and release depending on location of the user, the resource or both.

The protection of large data sets requiring keys for each item create situations producing/retrieving thousands of keys and encrypting thousands of items takes too long. The mitigation of the performance problem might involve tweaking a KMS API to support asynchronous key management and/or by optimizing the data securing process. For example, the API may support the bulk creation of keys. Perhaps a hybrid approach to DCS/KMS API could provide an API that is DCS aware and will make JWE and JWS data structures as part of the response. Such an API could provide enhancements over the classical approach to KMS APIs which is just concerned with key creation, encryption and signing as standalone activities that require the client to put together the DCS response.

10.3. Federated security and DCS

Federated security enables collaboration across multiple systems, networks, and organizations in different trust realms. On the other side, DCS enables securing the data sets independent of the communication and computation infrastructure. Future work in this area might investigate options for data centric secured information exchange in a federation environment with federated key management systems, identity servers, and other security infrastructure. Technologies such as Blockchain might be useful to establish a federated network of identity providers and services.

10.4. Packaging of data in the scope of DCS

In addition to current DCS approaches wrapping metadata keys and payloads into containers, future investigation may consider adding data such as policies, manifests, etc. to the package. This is important in mobile scenarios dealing with offline content.

10.5. Binary related Media Types

Future work should consider adding DCS approaches to binary data. The experiments in the testbeds to date focus on feature data types. However, in geographic realms, feature data is a small part of the overall amount of data available. Future work should investigate how to incorporate DCS container or to modify the existing formats to support DCS concepts. For example, JPEG 2000 (JP2) is an image compression standard and coding system. Perhaps the format can be expanded to support encryption and metadata for classification into JP2 payloads. JPEG 2000 images using the OGC GML in JPEG 2000 (GMLJP2) standard and other gridded coverage data holding geospatial content for imagery could add DCS concept for integrity and confidentiality.

Applications and professionals use many more binary data formats today in addition to the JPEG standard. Future work should consider the full spectrum of data types available. This would include GeoTIFF, TIFF, Mr. SID, etc. Future work should look at other binary packaging formats and how to extend them to support Data Centric Security concepts. GeoPackage is one example of a binary container format that future work could extend to include encryption, integrity and confidentiality.

Table 4. Suggestions for binary related Media Types

Description	OGC API Parameter ‘f’	HTTP Accept / Content-Type Header
Map	**jpeg or gif or …**	application/jpeg
JPEG200 with STANAG 4774 metadata	**jp2+dcs**	application/jp2+dcs; profile="http://www.opengis.net/def/profile/ogc/1.0/stanag#4774";

Description	OGC API Parameter ‘f’	HTTP Accept / Content-Type Header
STANAG 4778 in XML (each STANAG data object represents one JP2 image)	**dcs+jp2**	application/dcs+jp2; profile="http://www.opengis.net/spec/GMLJP2/2.0/req/core"

10.6. DCS Roles and User Clearances vs Data Classification(s)

Future work should consider further development of the DCS Roles concept, as detailed in Appendix E: Roles. The core thrust may proceed in two simultaneous directions.

First, future work should incorporate more specific examples / target sample data files of scenario classifications, clearances, and feature data.

Second (and in parallel), the Roles concept should be transformed to be more generic / less specific to the NATO clearance/classification concept (while still supporting it fully and robustly). This would be done so it could be applied to more generic commercial, industry and consumer domains, allowing for the expansion of DCS utilization, which in turn helps the original sponsors have greater long-term capabilities.

Chapter 11. Technology Integration Experiments (TIEs)

The TIEs for the Testbed-16 DCS task were grouped into multiple tests for each scenario. The TIEs are divided into sub-TIEs as follows:

- Online access of protected content on a desktop client.
- Offline use of protected content on mobile clients.

Testbed-16 primarily focused on the interaction with a newly introduced component KMS via a draft API. Further, the Testbed participants demonstrated the ability to request the response encoded using different container types.

The first step (which had already been evaluated in the previous testbed) included the data request sent to the DCS component. The request is accompanied by an access token. The token contains the security credentials for a login session and identifies the user.

Desktop and mobile clients communicate with the KMS retrieving cryptographic keys required to decrypt and consume (previously fetched) protected content (geospatial entities, signed or encrypted). Key retrieval is done via the draft KMS API by passing key identification strings extracted from the metadata section of protected content.

The resulting Data set was retrieved, the key_id values were extracted from the metadata section of the response container.

11.1. TIEs for Scenario One

This set of TIEs summarize the result when executing the implementation for scenario one with desktop client.

To determine the relevant TIEs, let’s take a look at the interactions between the components D121 (client), D120 (DCS Server) and D145 (Key Management Server). The figure below helps to identify the interactions.

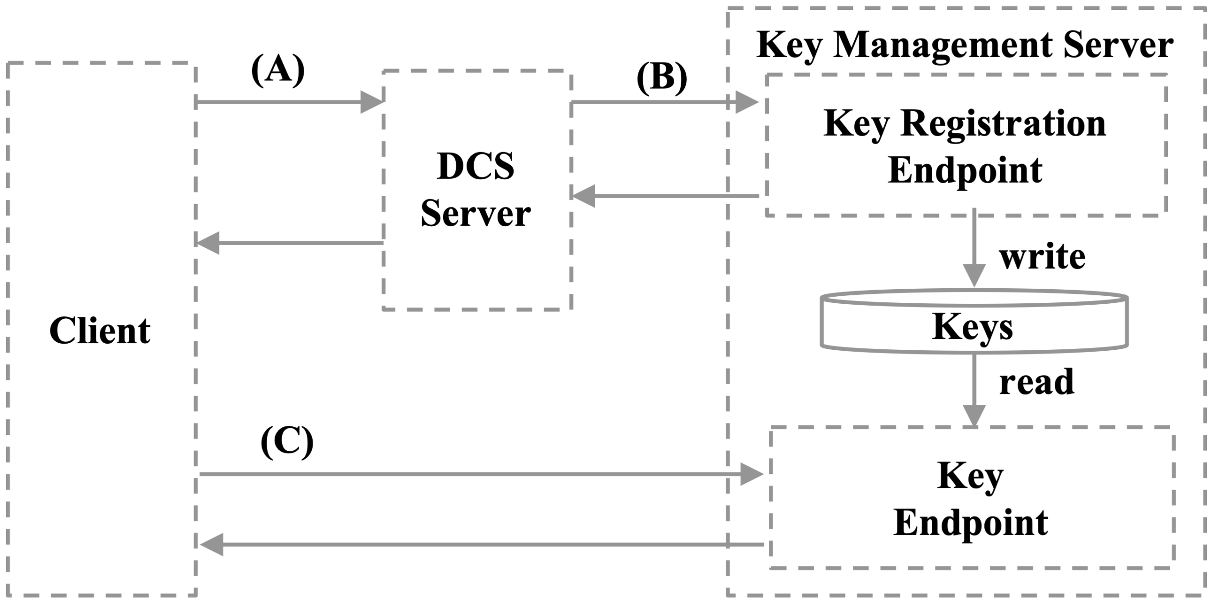


Figure 21. Abstract Protocol Flow between the TIE components

(A) Defines the interactions between D121 (the DCS client) and D120 (the DCS Server). (B) Defines the interactions between the DCS Server (D120) and the Key Management Server (D145) and finally between D121 and D145.

NOTE Detailed direct tests for the interfaces of DCS Server and Key Management Server via OpenAPI (and Curl) are outlined in annex Engineering Aspects for D120 and D145.

11.1.1. D120 / D121 TIE

The DCS client (D12) must send an OGC API - Features encoded request to the DCS Server (D120) including the `access token` as an HTTP Header `Authorization Bearer: 0476c745887f33cc43341375852df01e9b0fe2fe` and the `key_challenge` as well as the `key_challenge_method` query parameters. The client must also use one of the supported DCS media types to trigger DCS processing at the DCS Server.

Table 5. TIE Media Types

	DCS specific Media Type
1.	application/gml+dcs;profile="http://www.opengis.net/def/profile/ogc/1.0/stanag#4778"
2.	application/dcs+gml; profile="http://www.opengis.net/def/profile/ogc/2.0/gml-sf2";version=3.2
3.	application/geo+jose
4.	application/dcs+geo;profile=ogc:sf:in:geojson
5.	application/geo+dcs;profile=“http://www.opengis.net/def/profile/ogc/1.0/stanag#4778”

The `access token` represents the acting user. Jane, Bob, Alice and Joe are existing users belonging to the different security levels. A request will only return a TIE relevant response. This is if the user is in accordance with the security policies that fit their security levels.

Table 6. TIE Users

User	States	Roads	Landmarks	POIs
Jane	Yes	Yes	Yes	Yes
Bob	Yes	Yes	Yes	No
Alice	Yes	Yes	No	No
Joe	Yes	No	No	No

A successful TIE can be determined by the ability of the DCS client to decrypt (and display) the encrypted DCS response from the DCS server. When that is the case, the interactions (A), (B) and (C) must work as a whole: (A) Returns the encrypted response with DCS encoding; (B) Implicitly worked to register the cipher key(s) because the cipher keys referenced in the response for (A), could be (i) fetched via (C) and (ii) be used to decrypt the response.

The **successful** TIE was conducted via a QGIS DCS plugin requesting media type `application/dcs+geo`. The recording is available at https://www.youtube.com/watch?v=vcpayRQN6QI [https://www.youtube.com/watch?v=vcpayRQN6QI]

11.2. TIEs for Scenario Two

This set of TIEs summarize the result when executing the scenarios for mobile client implementation. The TIE results show interactions between two mobile clients with a policy enforcement module (Android [Maxar] and iOS [Keys]) and a Key Management Service [Helyx]. The experiment did not have time to perform TIEs for data interoperability between the two mobile clients.

The following chart shows the results of the mobile client interactions with the various KMS functions.

Table 7. Mobile Clients to KMS

Client Platform	KMS Function	Expected Result	Result Status
Android	POST: /key	key id	Passed
Android	GET: /key/{key_id}	JWK with key	Passed
Android	POST: /mac	HMAC of JWE	Passed
Android	POST: /encrypt	Cipher Text & IV	Passed
iOS	GET: /key/{key_id}	JWK with key	Passed
Configuration Tools (iOS)	POST: /key	key id	Passed
Configuration Tools (iOS)	GET: /key/{key_id}	JWK with key	Passed
Configuration Tools (iOS)	POST: /encrypt	Cipher Text & IV	Passed

11.2.1. Android Result Summary

The Android client contains a stand-alone Policy Enforcement app that stages data by fetching features from a Web Feature Service (WFS). With each feature the app fetches, the app creates an encryption key, and a HMAC key from the KMS. Then the app calls the KMS to calculate the HMAC and encrypt the feature data. The app stores the feature results in a file with a JWS DCS format. In addition, the app stores the keys returned by the KMS in a file containing a JWK set encrypted in a JWE. From the table above, the TIE shows that the Android Policy Enforcement app can create keys, fetch keys, calculate the HMAC and encrypt content.

11.2.2. iOS Result Summary

The iOS client architecture implements the DCS Roles concept, separating the scenario data into two categories - Feature Data ("DCS Data") and "DCS Roles" (see Appendix E: Roles for full details). Using external management tools and procedures, these two categories are bundled/packaged into a scenario-wide DCS Data package (common to all clients using this architectural concept), and into user/device specific DCS Role packages. Both bundles are encrypted leveraging interaction with the KMS, and are distributed to each scenario iOS device as part of organizational pre-deployment procedures.

The iOS client contains a built-in Policy Enforcement module. To complete pre-deployment configuration of each device, the iOS client's Policy Enforcement module queries the KMS, retrieving the keys authorized for each user role (for decryption of DCS Data), storing them into the module's DCS Role Key Cache for offline use by the client App.

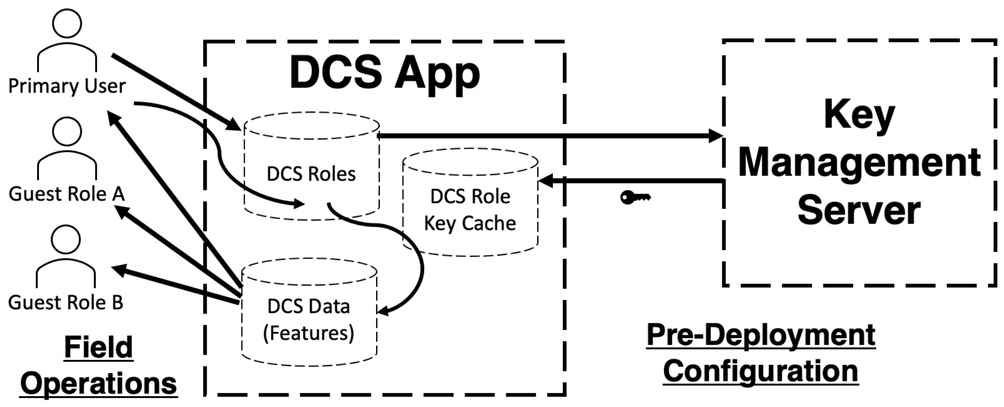


Figure 22. TIE iOS Mobile Policy Enforcement, Key Cache, and KMS

When visualizing data in the field, the iOS App uses the module to apply the appropriate keys to features allowed by the selected role.

Appendix A: Engineering Aspects for D120 and D145

This annex introduces the engineering aspects of the DCS components D120 (DCS Server) and D145 (DCS Key Management Server) implemented for Testbed-16 by Secure Dimensions.

During OGC Testbed-15, the DCS Server was implemented in its first version. In summary, the implementation supported NATO STANAG 4778 and 4774 response encoding. The XML encoded response was digitally signed and the data was encrypted. The encryption key was returned inline with the response. To protect the key, the public key of the user was used to encrypt the cipher key. This approach did not require any key management; the public key of the user(s) was manually distributed.

For Testbed-16, the requirements are (i) to support JSON encoded responses that allow to function like NATO STANAG 4778 and (ii) to return the cipher key with the response by reference, which requires to implement key management.

This section describes the architecture of the DCS Server, the DCS Key Management Server and their interactions. The emphasis for the architecture and the design of the Key Management Server in particular is to ensure protection of the cipher key - when created, registered and fetched for decryption.

A.1. Overview

The architecture illustrated in Figure 20 reflects the use case requirements for the desktop / server use case as outlined in the CFP: A desktop application (i.e. QGIS DCS plugin) can request NATO STANAG 4778 encoded responses where the data and the metadata is encrypted. For Testbed-16, the response structure is encoded in JSON mimicking NATO STANAG 4778 and 4774. The response contains an identifier for the cipher used to encrypt the meta- and data. When parsing the response in the DCS client, the cipher keys are fetched from the Key Management Server.

To ensure that a cipher key that is used by the DCS Server used to encrypt the meta- and data can only be obtained by a legitimate client / user, access tokens are used. The Authorization Server provides the capability for creating and verifying access tokens. The use of access tokens that are obtained by the DCS client and used with the DCS Server and Key Management Server ensure the sharing of a security context among all components.

Leveraging as many components from Testbed-15 as possible, the Testbed-16 architecture is comprised of only one new component: The Key Management Server. Albeit, the DCS Server is extended by processing JSON encoding.

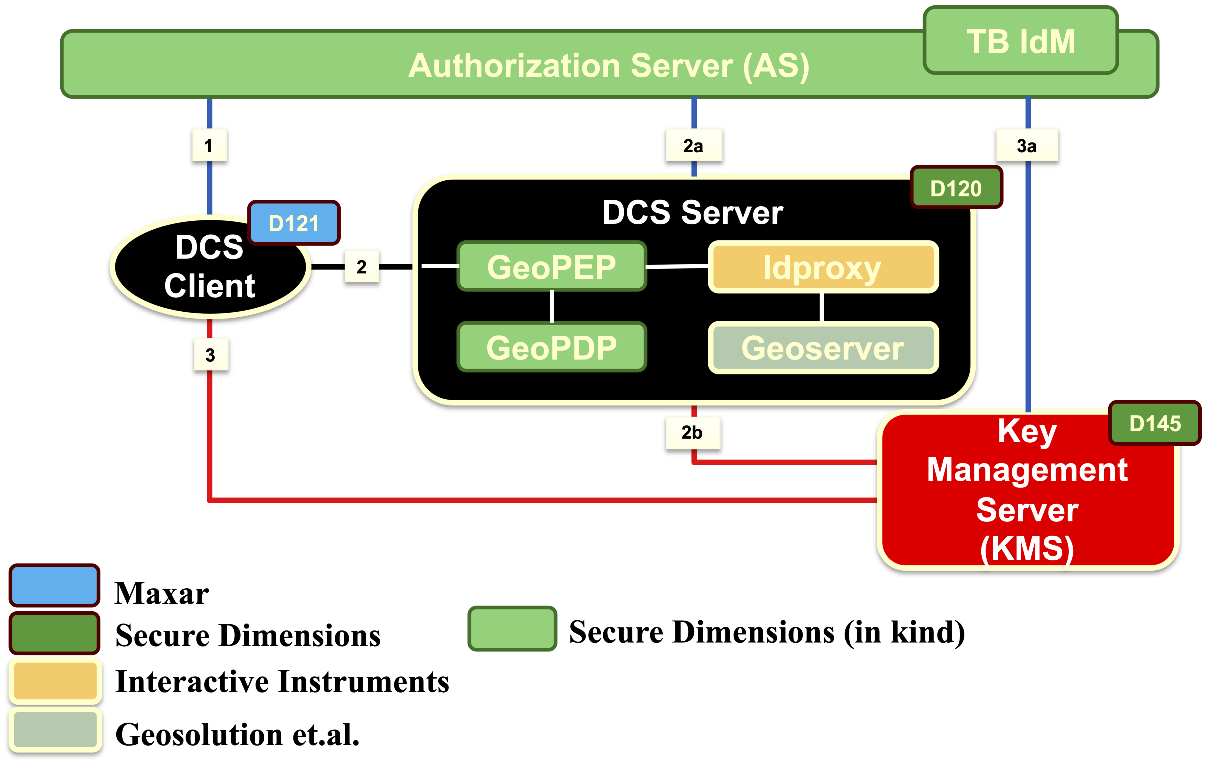


Figure 23. Component overview of DCS components to support the desktop / server use case

The following sequence of interactions explain the overall co-play of the components:

1. The DCS client requests an access token from the Authorization Server leveraging the OAuth2 / OpenId Connect protocol for authorization. During this interaction, the user must login to his Identity Provider. As a result, the DCS client receives an `access token` which is associated to the client and user, both identified with their UUID.

Access Token validation response example

```
{
  "access_token": "0476c745887f33cc43341375852df01e9b0fe2fe",
  "client_id": "019b7173-a9ed-7d9a-70d3-9502ad7c0575",
  "expires": 1602061225,
  "scope": "openid saml profile ogc",
  "username": "5a307c82-b440-3438-8aa7-b7437a83a4e0",
  "active": true
}
```

2. The DCS client sends an OGC API - Features encoded request to the DCS Server including the `access token` as an HTTP Header `Authorization Bearer: 0476c745887f33cc43341375852df01e9b0fe2fe` and the `key_challenge` as well as the `key_challenge_method` query parameters. The DCS Server extracts the `access token`, the `key_challenge` and the `key_challenge_method` from the request.

    a. The `access token` from the request gets validated via the Authorization Server. From the response, the DCS Server stores the `client_id` and the `username` for registering the cipher key(s) with the Key Management Server. Based on the response from the backend (OGC API Features), the DCS Server creates a cipher key per feature type classification (top_secret, secret, confidential, classified). The cipher keys differ in length and algorithm for each classification level.

b. For each cipher key created, the DCS Server registers the key with the Key Management server. The registration request includes the `key_challenge`, `key_challenge_method` and the `client_id` (aud) from (2). The request includes the HTTP Header `Authorization Bearer: 0476c745887f33cc43341375852df01e9b0fe2fe`.

Cipher key registration request example

```
{
  "alg": "http://www.w3.org/2009/xmlenc11#aes192-gcm",
  "kty": "oct",
  "iv": "",
  "k": "LiIeESRpwWngaJplPQxtsuT3xP5JtzJE",
  "key_challenge": "secret",
  "key_challenge_method": "plain",
  "audience": "019b7173-a9ed-7d9a-70d3-9502ad7c0575",
  "issuer": "af4f2285-979d-389a-892a-90aa9d776476"
}
```

The `key_id` from the response is included in the DCS Server response.

Cipher key registration response example

```
{
  "id": "859f22b4-1ce1-42c0-8668-aac789c79242",
  "issuer": "af4f2285-979d-389a-892a-90aa9d776476",
  "expires": 1602061971,
  "issued_at": 1602061941,
  "aud": "019b7173-a9ed-7d9a-70d3-9502ad7c0575",
  "sub": "5a307c82-b440-3438-8aa7-b7437a83a4e0"
}
```

3. The DCS client has parsed the response and extracted a list of key identifiers. For each `key_id` the DCS client sends request to the Key Management Server for obtaining the cipher key. The request includes the `access token` and the `key_verifier`.

   a. The Key Management Server verifies the `access token` with the Authorization Server and compares that the following conditions are met and returns the key information:

      - `client_id` associated with the `access token` matches the `aud` stored with the `key_id`
      - `username` associated with the `access token` matches the `sub` stored with the `key_id`
      - `key_verifier` matches the `key_challange` stored wit the `key_id` applying the `key_challange_method` stored with the `key_id` (as defined in RFC 7636)
      - current time in seconds is less than the `expires` stored with the `key_id` (the valid time frame for fetching keys is 30 seconds since creation)

Cipher key response from Key Management Server example

```
{
  "id": "859f22b4-1ce1-42c0-8668-aac789c79242",
  "kty": "RSA",
  "n":
"nhM1yyeJzcopJo79Cy_0jYbdhOL7XNzuYb2zi3HyTeQaNKwAzvt1c1MNMlm3Mt39kcB_mw5ehBZS1UZXGDWGV
2BH5WZhyvTufxONizUlb65M5NHRMIKbmeDEYgyegKke6aaNaOl4QfSI6sd7JH6Zq_RtFBb85evfm74poRuV_Jn
S7u8j-
kKrXUTgHNhwxHa8xuyz19o8506uWdDrYta53NYiuWdZ_So2Mzi3eK26o8rO3IX9Wk6nIWTYKmYetwYps0KOi7Q
8hiH1RknrLvnNFT-z7eK2SZ3jycZCbDmD15KAasm5HQAlP3tOWJvq9_w3HiZakHZlNDwGbgCT1l_1pQ",
  "e": "AQAB",
  "audience": "019b7173-a9ed-7d9a-70d3-9502ad7c0575",
  "sub": "5a307c82-b440-3438-8aa7-b7437a83a4e0"
}
```

With the cipher key received from the Key Management Server, the DCS client is able to decrypt the meta- and data received from the DCS Server in (2).

A.2. Deployment

The DCS client is implemented as a QGIS DCS plugin available from https://github.com/ogc-leedahl/QGIS/tree/OGC_Testbed_16

The following components build the DCS Server https://ogc.secure-dimensions.com/dcs

- Geoserver: This is a default Geoserver deployment 2.16.2 with example data loaded (Docker deplyoed).
- ldproxy: This proxy to the Geoserver produces the OGC API - Features on top of Geoserver (Docker deployed).
- geoPEP is the security proxy implemented as a Apache Web Server Module (httpd deployed).
- geoPDP is a GeoXACML 3 compliant service (Docker deployed).

The Key Management Server is a PHP application hosted on https://ogc.secure-dimensions.com/kms/api

A.3. Protecting the Cipher Keys

Applying encryption to achieve confidentiality does only make sense if it can be ensured that the cipher key can be protected against unauthorized disclosure. In Testbed-15, the protection of the cipher key was ensured by establishing a PKI for the users. The cipher key was encrypted with the public key of the user. The implication to this approach was that a cipher key is tied to a single user. It is not possible for software to process the encrypted data on behalf of the user. Or, for a user to pass the encrypted content to another user.

The more flexible key management in Testbed-16 demonstrates the concept where the cipher key is not included in a service response; rather the key identifier is included. This provides the ability to use and re-use cipher keys and modify the audience: which clients and users can fetch the key. But, the down side is that a Key Management Server must be designed that its flexible enough to support particular use cases but also ensure that a cipher key can only be obtained from an authorized user / application.

Studying the architecture from figure Figure 20 outlines that the DCS client is making an OGC API Features request where the response contains encrypted (meta)data and the keys must be obtained from the Key Management Server. This leads to a triangle of trust relationship: A security context must be exchanged between the components where the DCS Server is a kind of `man in the middle` between the client and key management server. So, the critical question is how to ensure that a key created by the DCS Server can only be fetched (and modified) by the original actor; the DCS Server is acting on behalf of the user / client when it comes to cipher key registration.

The base for protecting cipher keys, implemented into the Testbed-16 Key Management Server, is based on the concepts of RFC 7636 Proof Key for Code Exchange by OAuth Public Clients [https://tools.ietf.org/html/rfc7636]. The basic concept - as illustrated in the following figure - is that the client includes a `code_challenge` (either created by itself or provided by the user as a kind of private secret) with the request which is a hash of a private secret. When relevant in a later request, the client sends the private secret using the `code_verifier` parameter. The server can then compare the values sent first with the hash (assuming method S256 was used) of the plain sent with the current request.

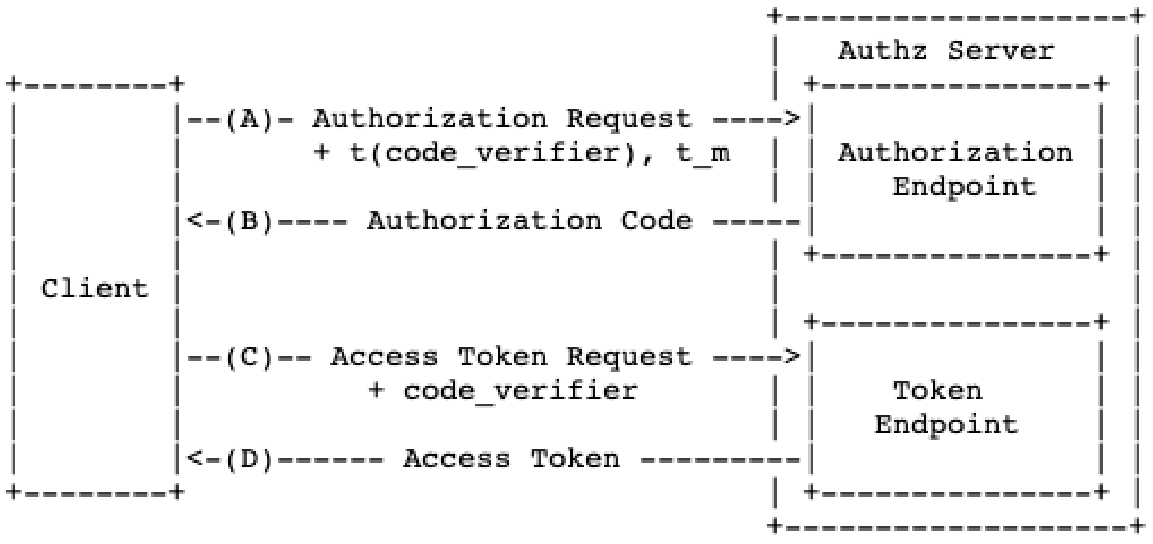

Authz Server
Client
--(A)- Authorization Request ---->
+ t(code_verifier), t_m
Authorization Endpoint
<-(B)---- Authorization Code -----
--(C)-- Access Token Request ---->
+ code_verifier
Token Endpoint
<-(D)------ Access Token ---------

Figure 24. Abstract Protocol Flow [RFC 7636, figure 2]

The adopted protocol for the Testbed-16 architecture is illustrated in the following figure.

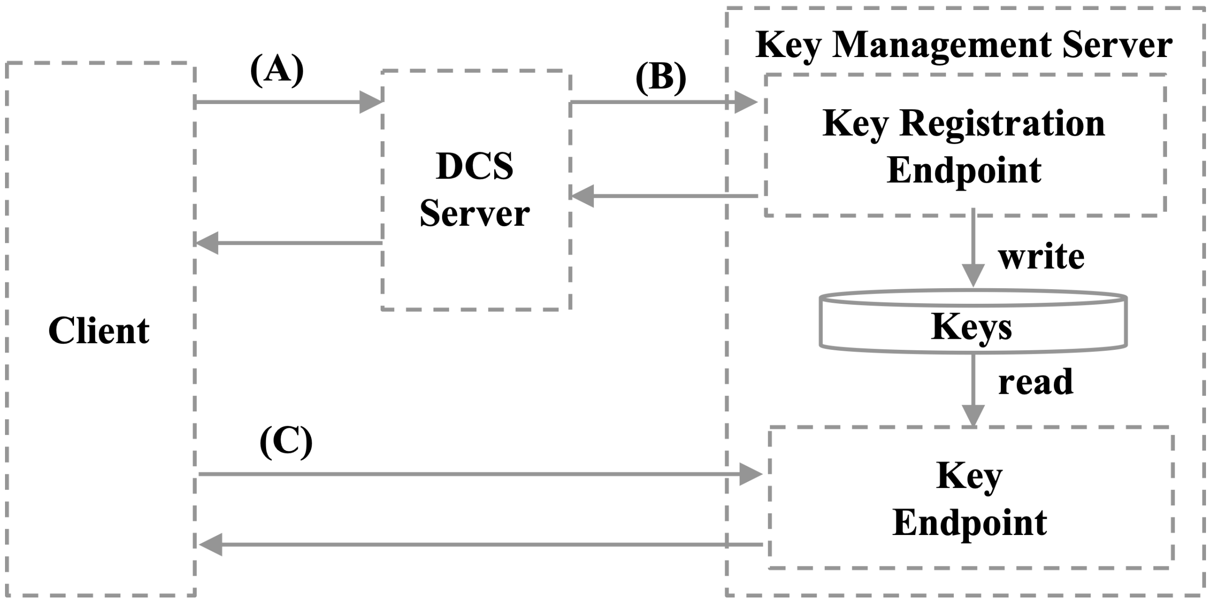


Figure 25. Abstract Protocol Flow from RFC 7636 adopted for Testbed-16

The use of the `key_challenge` parameter in request (A) allows the Client to request the cipher key from the Key Management Server adding the `key_verifier` parameter to the request (C).

The use of this protocol also enables the modification of the key by the original actor, as only the user / client is capable of adding the matching `key_verifier` to the request. The DCS Server or any other intermediary service is not able to provide or guess the private secret, as in the request (A) only the hash was submitted. It is therefore possible (but not implemented) that the user extends the audience for an encryption key that was created on his behalf (request). This would allow the user to share the received encrypted content with another user / client. The Key Management server would just need to implement the appropriate methods. Proof of authorization to modify an existing key can be based on the `key_verifier`.

However, the deletion of a cipher key is implemented, and that requires the caller to provide the `key_verifier` with the request to demonstrate ownership.

A.4. DCS Key Management Server

The implementation of the interfaces (API) and the functionalities for the Key Management Server is based on the requirements derived from the desktop / server use case. Two different interface categories exist:

- Managing cipher keys that can be used to encrypt/decrypt data and metadata
- Managing public keys that can be used to encrypt the response when fetching a cipher key.

All requests to the Key Management Server require a valid `access token` submitted via the HTTP header `Authorization` as described in RFC 6750.

A.4.1. Protecting Keys at Rest

The Key Management Server stores the keys in a simple database structure:

```
+--------+-------------+------+-----+---------+-------+
| Field  | Type        | Null | Key | Default | Extra |
+--------+-------------+------+-----+---------+-------+
| id     | varchar(36) | NO   | PRI | NULL    |       |
| type   | varchar(64) | YES  |     | NULL    |       |
| sub    | varchar(36) | YES  |     | NULL    |       |
| aud    | varchar(36) | YES  |     | NULL    |       |
| active | int(11)     | YES  |     | 1       |       |
| data   | text        | YES  |     | NULL    |       |
+--------+-------------+------+-----+---------+-------+
```

The `data` column contains the BASE64 encoded value of the encrypted cipher key encoded as JWK. The encryption for this implementation is configured to use the `AES-128-CBC` cipher with a secret key. The following is an example of the value stored in the `data` column:

```
Ixw37lqOcZZuXvIhXVMLUZIzV1siqprO3+oxT4kPxECRXZ9+6EaJGGQEYkDViSNXzAyghRcRcERai4bq7F40pB
eQXextZGnGi537NB7kBSYLCNUsm/y+6YgKw/GhtfjENEaFRHw7QoVaqyTuMQquB/hGS5mENyGmK16u0vToqfJ4
It1Ss7tPdFDFIHmjcxfWg0EePqa37z8cql+UFklDGwHYwizFKd5QV7kFrAlYofDz4acdoz2nwnFtzqTPLUh9tz
VzramlZZp2JDuFOL1XiGTEHhAwbHAKWAPeSHr7UDxbzzjrn54w7Ew5wo3eIwbqf9ZMmZ1qVyI+R8XU4L3njqpy
Gcq88UwnMzgHikKu1OTeCKUyxekejccXDEhSnqP/dW45eSOS2M0f7bExGBw1nBHhcgG/wcuzh5lDkNtDyXm6y7
nEGodPhn6uEHda4yukc9FpBxl+2ul7meapIv/dVCzgXywwE3JF4jRqQ7VS0PUGCyPWMlz3zdg7jvzMjIY6695V
oIYixOIs9G01nh5udA==
```

A.4.2. Managing Public Keys

To ensure the protection of a cipher key while in transit (sent from the Key Management Server to the client) the response can be encrypted. This requires that the client specifies a public key id (`public_kid` parameter) to be used for encrypting the response. In order to register a public key, the POST interface of the `/kms/jwks` path must be used. The `audience` restriction of the key is derived from the `access token` sent with the registration request.

Alternatively, the request can contain the `public_key` parameter including the JWK encoding of a public key.

A.4.3. Managing Cipher Keys

The current implementation allows to register (create), fetch (read) and delete a cipher key.

A.4.4. Create a Cipher Key

The Key Management Server supports to different ways to register a cipher key with the `/kms/keys` endpoint. Depending on the HTTP method, the caller must use HTTP `POST` or `PUT` to register a cipher key:

- A cipher can be registered using HTTP `POST` in two different ways: A JWK compliant key description is POSTed to the `/kms/keys` endpoint, (i) containing the `k` value, or (ii) not containing the `k` value. In the first case, the key is saved to the store and a `key_id` is created and returned to the caller. In the latter case, a k value and key_id is created and returned to the caller. The information about the client_id, username is stored with the key.

- A cipher can be created using HTTP PUT to the /kms/keys/{key_id} endpoint. The idempotent call basically returns a 201 (Created) on success or 409 (Conflict) in case the key_id does already exist but the content of the JWK description is different.

To GET a stored cipher key requires the following conditions to be true:

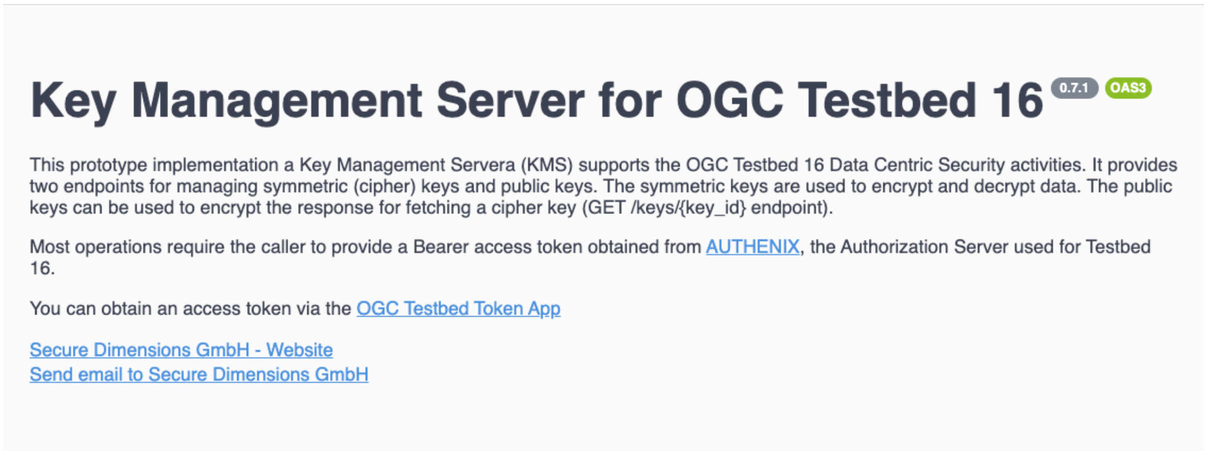
- client_id associated with the access token matches the aud stored with the key_id
- username associated with the access token matches the sub stored with the key_id
- current time in seconds is less than the expires stored with the key_id (the valid time frame or fetching keys is 30 seconds)

To GET the cipher key in a JWE format (encrypted JSON), the caller must specify the parameter public_kid or public_key with the request send to the /keys/{key_id} endpoint and set the HTTP Accept header to value application/jwe.

Even though the cipher key is protected with the access conditions above, it is the safest to simply delete the key once it got fetched by the client. To DELETE a cipher key, the caller must submit a valid access token and the key_verifier. The key_verifier proves that the caller is the owning entity that either directly or via the DCS server has registered the key. If the key_verifier matches the key_challange stored with the key_id applying the key_challange_method stored with the key to the key_verifier (as defined in RFC 7636), then the cipher key referenced by the key_id is deactivated from the store. The deactivation deletes the key data and marks the key_id inactive. The Key Management Server will respond to further requests for a deleted key_id with HTTP status code 410 (Gone).

A.4.5. OpenAPI

The endpoints of the Key Management Server are described in OpenAPI: https://ogc.secure-dimensions.com/kms/api


Key Management Server for OGC Testbed 16 0.7.1 OAS3

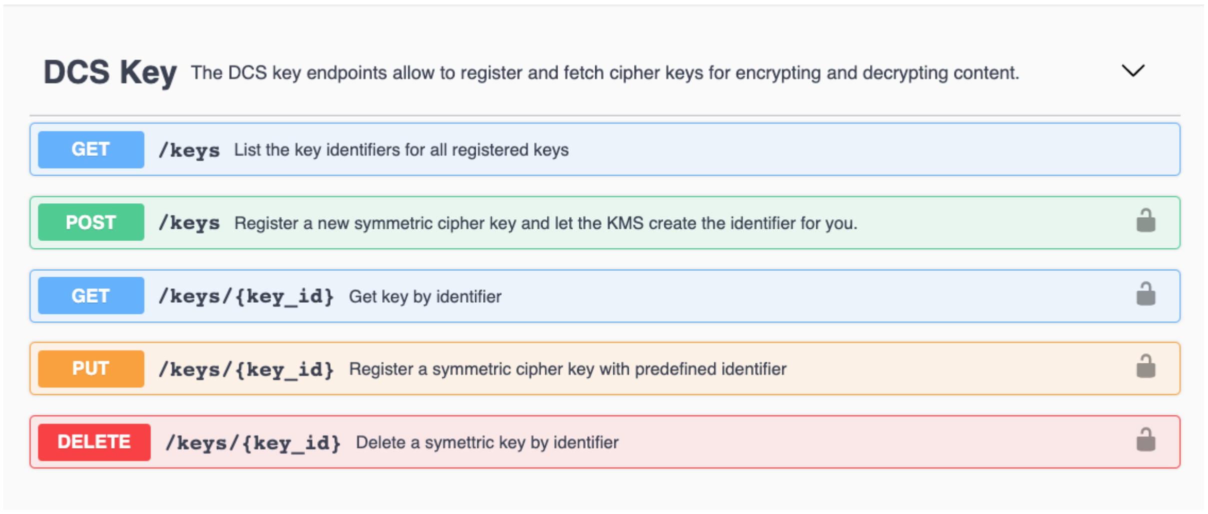
This prototype implementation a Key Management Servera (KMS) supports the OGC Testbed 16 Data Centric Security activities. It provides two endpoints for managing symmetric (cipher) keys and public keys. The symmetric keys are used to encrypt and decrypt data. The public keys can be used to encrypt the response for fetching a cipher key (GET /keys/{key_id} endpoint).

Most operations require the caller to provide a Bearer access token obtained from AUTHENIX, the Authorization Server used for Testbed 16.

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

Secure Dimensions GmbH - Website
Send email to Secure Dimensions GmbH

Figure 26. Key Management Server overview


DCS Key The DCS key endpoints allow to register and fetch cipher keys for encrypting and decrypting content.

GET /keys List the key identifiers for all registered keys

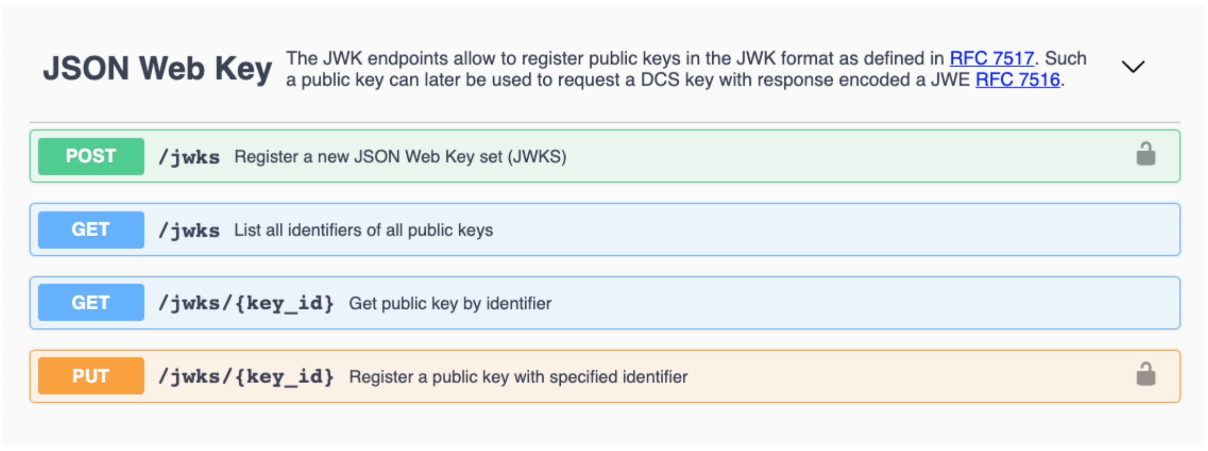
POST /keys Register a new symmetric cipher key and let the KMS create the identifier for you.

GET /keys/{key_id} Get key by identifier

PUT /keys/{key_id} Register a symmetric cipher key with predefined identifier

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

Figure 27. Key Management Server endpoints for managing cipher keys


JSON Web Key The JWK 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 DCS key with response encoded a JWE RFC 7516.

POST /jwks Register a new JSON Web Key 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 28. Key Management Server endpoints for managing public keys

A.4.6. Use Example

The use case for the Key Management Server is to support the encryption and decryption of data as outlined in figure Figure 22. In the desktop / server scenario, the DCS Server creates the cipher keys and encrypts the data. The DCS Server uses the KMS to register the cipher key, as illustrated in Figure 22, interaction (B). The key identifier received from the Key Management Server is inserted into the response to the client; the response to (C).

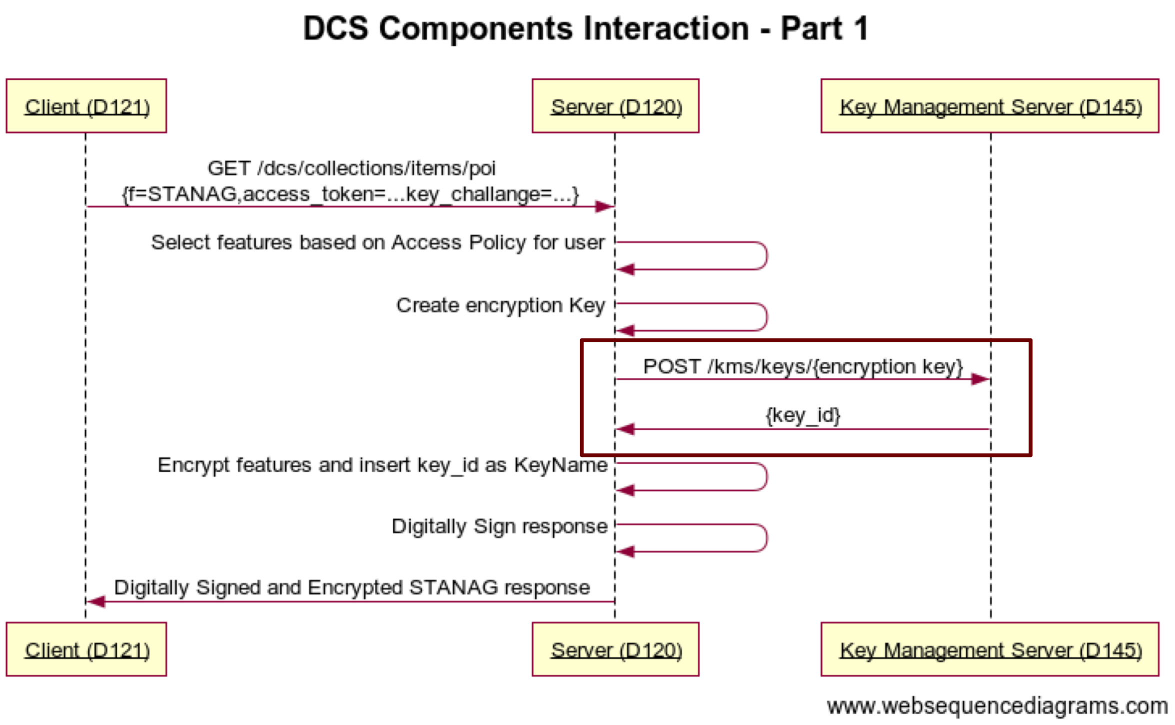
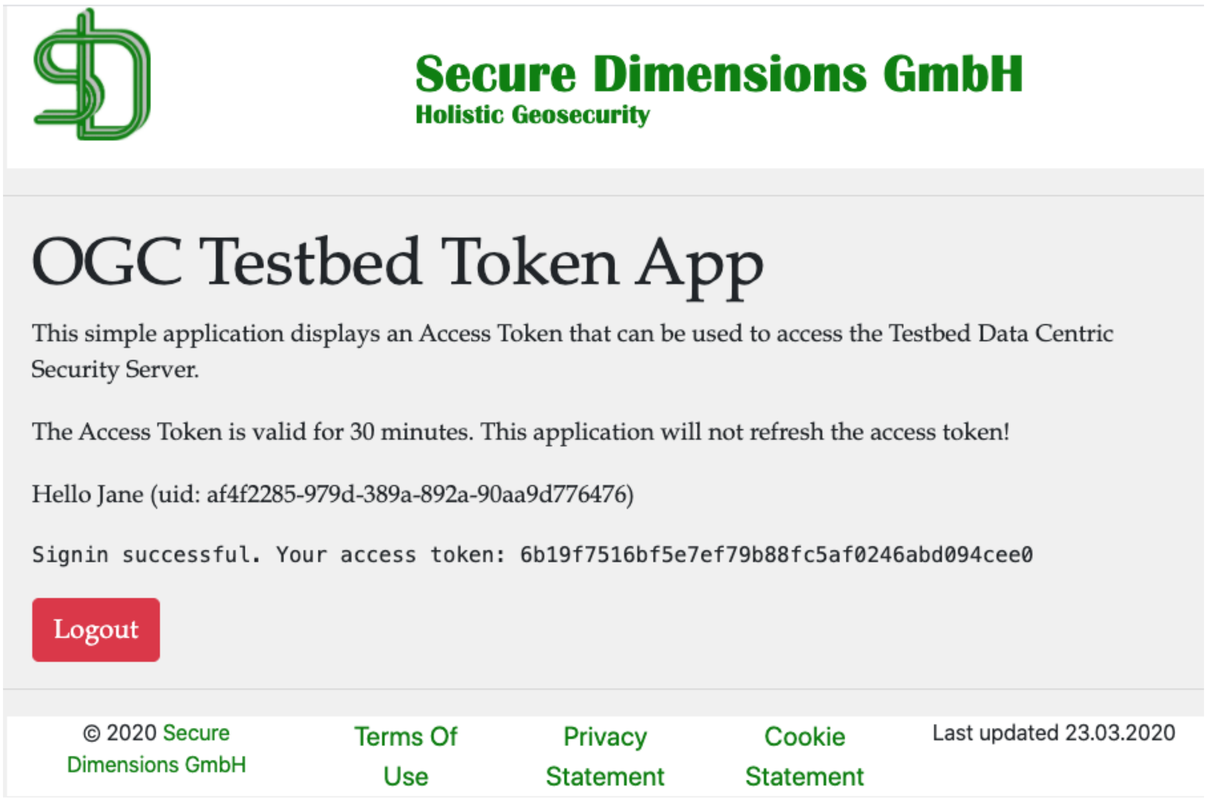

DCS Components Interaction - Part 1
Client (D121)
Server (D120)
Key Management Server (D145)
GET /dcs/collections/items/poi {f=STANAG,access_token=...key_challange=...}
Select features based on Access Policy for user
Create encryption Key
POST /kms/keys/{encryption key}
{key_id}
Encrypt features and insert key_id as KeyName
Digitally Sign response
Digitally Signed and Encrypted STANAG response
Client (D121)
Server (D120)
Key Management Server (D145)
www.websequencediagrams.com

Figure 29. Interactions between the client, DCS Server and the Key Management Server

The sequence diagram illustrates the round-trip interactions from the client to the DCS Server and the Key Management Server.

A.4.6.1. Managing a Cipher Key

Use the OGC Testbed Token App [https://ogc.secure-dimensions.com/dcs/token-app/] and login as user `jane` with password `secret` to visualize an access token (valid for 30 minutes).



OGC Testbed Token App

This simple application displays an Access Token that can be used to access the Testbed Data Centric Security Server.

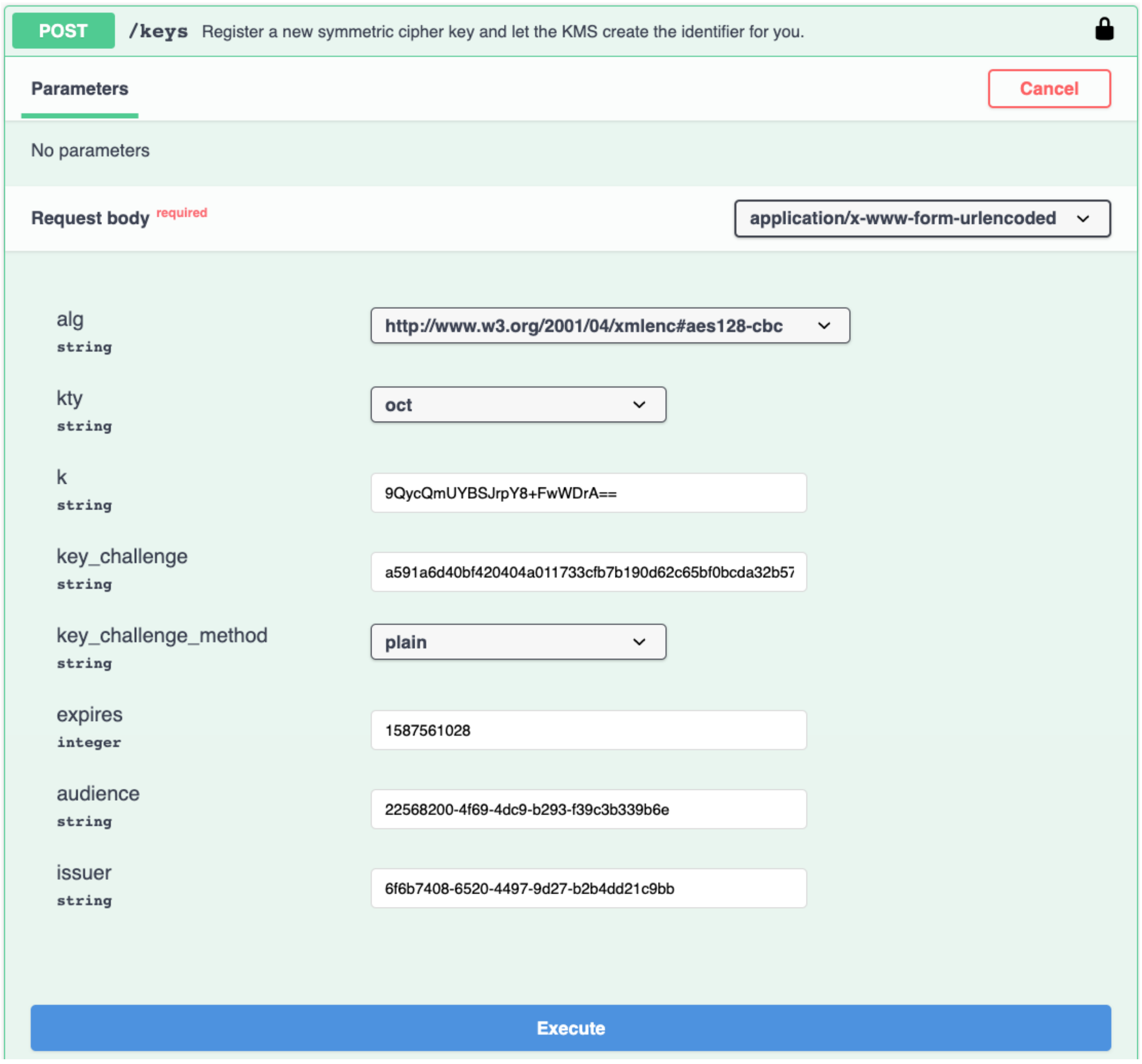
The Access Token is valid for 30 minutes. This application will not refresh the access token!

Hello Jane (uid: af4f2285-979d-389a-892a-90aa9d776476)

```
Signin successful. Your access token: 6b19f7516bf5e7ef79b88fc5af0246abd094cee0
```

Figure 30. OGC Testbed Token App displaying Jane's access token

Now use the Key Management Server OpenAPI to register a cipher key


POST /keys Register a new symmetric cipher key and let the KMS create the identifier for you.

Parameters

Cancel

No parameters

Request body required

application/x-www-form-urlencoded

alg string	http://www.w3.org/2001/04/xmlenc#aes128-cbc
kty string	oct
k string	9QycQmUYBSJrpY8+FwWDrA==
key_challenge string	a591a6d40bf420404a011733cfb7b190d62c65bf0bcda32b57
key_challenge_method string	plain
expires integer	1587561028
audience string	22568200-4f69-4dc9-b293-f39c3b339b6e
issuer string	6f6b7408-6520-4497-9d27-b2b4dd21c9bb

Figure 31. Key Management Server API to register a cipher key

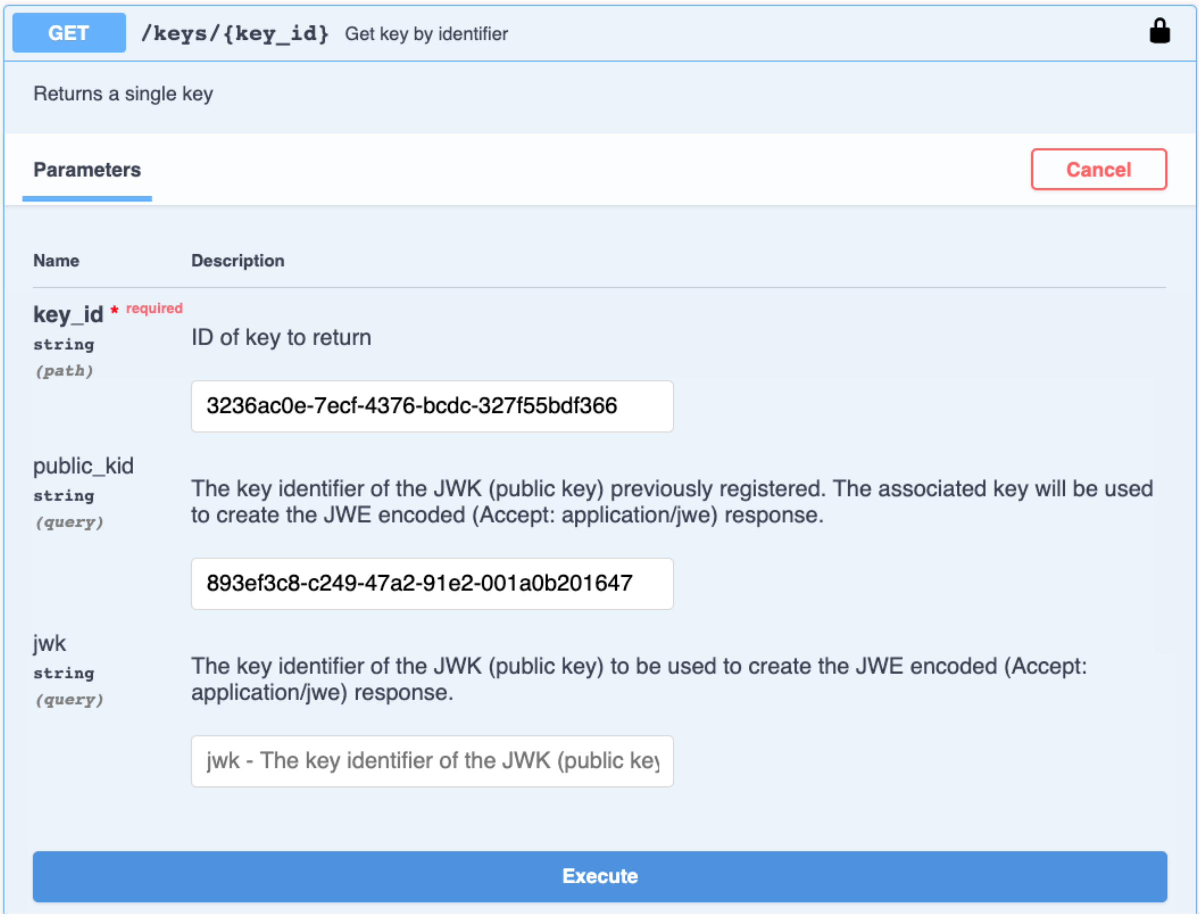
or send a CURL request like this:

```
curl -X POST "https://ogc.secure-dimensions.com/kms/keys" -H "accept:
application/json" -H "Authorization: Bearer 1a44f0f0db04876d86475d42597c6d653dd252b8"
-H "Content-Type: application/x-www-form-urlencoded" -d
"alg=http%3A%2F%2Fwww.w3.org%2F2001%2F04%2Fxmlenc%23aes128-
cbc&kty=oct&k=9QycQmUYBSJrpY8%2BFwWDrA%3D%3D&key_challenge=foobar&key_challenge_method
=plain&expires=2587561028&audience=019b7173-a9ed-7d9a-70d3-
9502ad7c0575&issuer=Andreas"
```

to get a response like this:

```
{
  "kid": "3236ac0e-7ecf-4376-bcdc-327f55bdf366",
  "alg": "http://www.w3.org/2001/04/xmlenc#aes128-cbc",
  "kty": "oct",
  "k": "DLCZvmi0vcTneZwkbudG_g",
  "issuer": "Andreas",
  "expires": 2587561028,
  "issued_at": 1602228077,
  "aud": "019b7173-a9ed-7d9a-70d3-9502ad7c0575",
  "sub": "af4f2285-979d-389a-892a-90aa9d776476"
}
```

With the access token and the `kid` you can use the Key Management Server OpenAPI to fetch the cipher key



/keys/{key_id} Get key by identifier

Returns a single key

Parameters

Name	Description
key_id * required string *(path)*	ID of key to return 3236ac0e-7ecf-4376-bcdc-327f55bdf366
public_kid string *(query)*	The key identifier of the JWK (public key) previously registered. The associated key will be used to create the JWE encoded (Accept: application/jwe) response. 893ef3c8-c249-47a2-91e2-001a0b201647
jwk string *(query)*	The key identifier of the JWK (public key) to be used to create the JWE encoded (Accept: application/jwe) response. jwk - The key identifier of the JWK (public key

Figure 32. Key Management Server API to fetch a cipher key

or send a CURL request like this:

```
curl -X GET "https://ogc.secure-dimensions.com/kms/keys/3236ac0e-7ecf-4376-bcdc-
327f55bdf366?public_kid=893ef3c8-c249-47a2-91e2-001a0b201647" -H "accept:
application/json" -H "Authorization: Bearer 1a44f0f0db04876d86475d42597c6d653dd252b8"
```

to get a response like this:

```
{
  "kid": "3236ac0e-7ecf-4376-bcdc-327f55bdf366",
  "alg": "http://www.w3.org/2001/04/xmlenc#aes128-cbc",
  "kty": "oct",
  "k": "DLCZvmi0vcTneZwkbudG_g",
  "issuer": "Andreas",
  "key_challenge": "foobar",
  "key_challenge_method": "plain",
  "expires": 2587561028,
  "issued_at": 1602228495,
  "aud": "019b7173-a9ed-7d9a-70d3-9502ad7c0575",
  "sub": "af4f2285-979d-389a-892a-90aa9d776476"
}
```

To receive the response encrypted (in `application/jose` or `application/jwe`) format, you must set the `Accept` header for the request accordingly and provide either a `public_kid ` of a previously registered public key or a JSON encoded JWK as value to the `public_key` parameter.

Request to receive encrypted response using `public_kid`

```
curl -X GET "https://ogc.secure-dimensions.com/kms/keys/3236ac0e-7ecf-4376-bcdc-
327f55bdf366?public_kid=820e2b52-c793-814a-8526-387ce0571fb4" -H "accept:
application/jwe" -H "Authorization: Bearer 1a44f0f0db04876d86475d42597c6d653dd252b8"
```

Request to receive encrypted response using `public_key`

```
curl -X GET "https://ogc.secure-dimensions.com/kms/keys/3236ac0e-7ecf-4376-bcdc-
327f55bdf366?jwk=%7B%20%20%20%22kid%22%3A%20%22820e2b52-c793-814a-8526-
387ce0571fb4%22%2C%20%20%20%22kty%22%3A%20%22RSA%22%2C%20%20%20%22n%22%3A%20%225MPCfUA
khGG6w76Cw2b7vzmyM-K4-
80bVn_aPMHHEBa4SQPfERmK_Q4L9fD6FD6krj_RU_DCYENmMo0ceZQymePdSmeSHgbrkyU9vXfvLDHNftGPgH0
xtQmc-gBWKMopRs6Svd13CCFaKn8P66iF25yVwmc13-
5WKGSLJV5oiDa3vOfiJKSqWnZAkejo2BaOSOl9R0qPjLt7z8B18LqTkNeOnsYigMIeAjis4CrXWVYfbIpryOLF
cGBC4gCHiF7tvP5YR3HtqDSmTNzK3xqSFNn_3PMRaGByV8yxcWDB3-
2lRr5JwznuZlm37r_RptgsU73AfhL1phFhYLdTQQ5kmQ%22%2C%20%20%20%22e%22%3A%20%22AQAB%22%2C%
20%20%20%22aud%22%3A%20%22019b7173-a9ed-7d9a-70d3-
9502ad7c0575%22%2C%20%20%20%22sub%22%3A%20%22af4f2285-979d-389a-892a-
90aa9d776476%22%20%7D" -H "accept: application/jwe" -H "Authorization: Bearer
1a44f0f0db04876d86475d42597c6d653dd252b8"
```

This is the encrypted response.

```
eyJhbGciOiJSU0ExXzUiLCJlbmMiOiJBMTI4Q0JDLUhTMjU2Iiwia2lkIjoiODIwZTJiNTItYzc5My04MTRhLT
g1MjYtMzg3Y2UwNTcxZmI0In0.Yg0j2cSVFvc2OQ5FUl0fxLyRVcA01FjObHY_P08VIir8StDSqe7RJMCcii1J
YPBCdht7O5k6Zay03eF63c6TTfWFEPFlkeaMVefQ7OTrd0Ls77bReCGSI93Uawx6Lcepwj9JsoDaT4r7YJb0vP
IyZfmQhcwdcysTQcw-BPURDq8v-Cji-
EmGO2GrH5EiNUdIvH8oaeuPbJZfsMofZrh02q6SN3uGW4AKahcMQ0p2DyRyQeVA15dLl13ohtba-aa6-
qKHqLk_CkURwHQpLDEVbdo_5tz8PQbUGnqH2mBWaWRRPfnLkFyslkDFhO2ZC-
hqfYGXIzv1f0xTW3h2o8nDgw.I7_S8Bb0BZ-SqLjdWt6_Cw.9GzczDxku_pI_-
upVSel4ZgGvwf0em_qTO3FZB05SO83eLewJWQS6QPmWBQgWFqzb6Pf2pbUNgtz-8Z5bVQmGJzIl7IFjM85-
pj5mlKWTH6Eb2OnbF5nbXmPMa789c0DnOO13Xi_SBt0Zs0AjlNtrXEasyjjCmN-
ghM09EhaLmF2YMLFYP5KephuDnyVINRMoyv1LeIMwMrYlHFU1l0a8lEIgKCs-
oKU54HulyGacLhajVUWYCKOCy78iBMeP1g19zrJnRmNKX5y88pS7a9OK8f62dMJLaA2kaSq-iDKiM1qO1lCYc-
CEmxLwvf-
8ixvQsODEFRRS0RN_HyusTFvViptigMOKxXUPdrXR8aJqGSUZ7EPicvqTcW7GhnMYOCQ1yAeejscAUBUeH3e7S
uX4Pi78nVfB309E3va2EprtUYU0tKtoUZ3RF1FFCBPgiVh8GUnEzImW-
gL4eaxv7JSgl1KR2ElG9N9NXBihL4eyKNSAY7Gm_qEiZGAEkJAQt5gP9gOemPDtFrQn0ioJVRkgybCYHvXNc2_
wGj-rsNSaFhO9ur6pR7vzKuhoswbMaLJgrmWrn_Rbm4UzTqja1MEUI_WO9-GCJizcdp7GVFm9qc0Od_2-7fL-
O9w5sZvIeHBd4kEUfEldWc2o9IQuhb1ZnZvk7r6PCi_tnrCDBY_JNqDkM0gSLrAlifoF5AfRBVZWILgf5dbcRG
51wwmbbyoQ6YcgEN6WOF5Sz-o0gls2i2fC-
n3bNOKXXNUp7dUXjPDIo5vuyfro2FhFv7sr19FOpupFmvKs8IlZ7f43Kl74luYw2fZqCV9iWCVFrt7bMPveUFw
XXPR66qQSJCcxBhPFYDpkCAkUzIYo1DF6KAhjd2Axu5n9rgpmeF81cCBbhcw0Z5USZDvNwyz6VEkCkZ8AuSPIs
1cGg8IQSosEU98V1mu38M3SBFbb5Z1h5CuCh_y2SZBv_4BthBgSJENpGTwJyOTU059h_NQ2m1U7GijiSnfTBse
zyxlQ7PhtvvSgFrc.as5ak-6CbI8N0qp74ElziA
```

Decoding the header of the JWE encoded response unveils the details of the encrypted content:

```
{
  "alg":"RSA1_5",
  "enc":"A128CBC-HS256",
  "kid":"820e2b52-c793-814a-8526-387ce0571fb4"
}
```

It is also possible to receive the response as JWT, where the digital signature can be verified with the JSON Web Key set published by the Key Management Server under https://ogc.secure-dimensions.com/kms/.well-known/jwks.json:

```
{
  "keys": [
    {
      "kid": "893ef3c8-c249-47a2-91e2-001a0b201647",
      "kty": "RSA",
      "n":
"nhM1yyeJzcopJo79Cy_0jYbdhOL7XNzuYb2zi3HyTeQaNKwAzvt1c1MNMlm3Mt39kcB_mw5ehBZS1UZXGDWGV
2BH5WZhyvTufxONizUlb65M5NHRMIKbmeDEYgyegKke6aaNaOl4QfSI6sd7JH6Zq_RtFBb85evfm74poRuV_Jn
S7u8j-
kKrXUTgHNhwxHa8xuyz19o8506uWdDrYta53NYiuWdZ_So2Mzi3eK26o8rO3IX9Wk6nIWTYKmYetwYps0KOi7Q
8hiH1RknrLvnNFT-z7eK2SZ3jycZCbDmD15KAasm5HQAlP3tOWJvq9_w3HiZakHZlNDwGbgCT1l_1pQ",
      "e": "AQAB"
     }
  ]
}
```

Key response a JWT

```
eyJ0eXAiOiJKV1QiLCJhbGciOiJSUzI1NiIsImtpZCI6Ijg5M2VmM2M4LWMyNDktNDdhMi05MWUyLTAwMWEwYj
IwMTY0NyJ9.eyJpc3MiOiJBbmRyZWFzIiwiYXVkIjoiMDE5YjcxNzMtYTllZC03ZDlhLTcwZDMtOTUwMmFkN2M
wNTc1IiwiaWF0IjoxNjAyMjI5ODg0LCJuYmYiOjE2MDIyMjk4ODQsImV4cCI6MjU4NzU2MTAyOCwidWlkIjoiM
zIzNmFjMGUtN2VjZi00Mzc2LWJjZGMtMzI3ZjU1YmRmMzY2IiwiYWxnIjoiaHR0cDpcL1wvd3d3LnczLm9yZ1w
vMjAwMVwvMDRcL3htbGVuYyNhZXMxMjgtY2JjIiwiayI6IkRMQ1p2bWkwdmNUbmVad2tidWRHX2NifQ.d0YkU9
OVcfaO-teRb8Vn9L4LFYqUyOLwnd5ktB7YV8xmaxlbGHADWkIbCMaQyWs7pllhKZa29XVK-2_ADy4tqAXLBNm-
MBKydP1JrN0-a3vJKvzDW17hMiC_-2UB65ngbpB-
c6FwVNBuaJa9ptDgtGdK6nz4X3kXZ4wZKfWAkcd1UDh9tHxLQLkOkmvWaCUdV5jIdeqWxG_eJe5F0cWK8PyhUn
DS9d4TCyBQHjRjtk8_XCLlESIbYzlUxbNKFMj04pIpdlkStUJG0m5ktkDE69u7WZBrsXFihDEJlf7YlD7FI_1D
IAOjvnbqkT1FcGkBkf9T-6t54dOOOzJxLdYobA
```

Decoding the header unveils details for the digital signature:

```
{
  "typ":"JWT",
  "alg":"RS256",
  "kid":"893ef3c8-c249-47a2-91e2-001a0b201647"
}
```

The verification of the response can be verified with `kid=893ef3c8-c249-47a2-91e2-001a0b201647` which is published by the Key Management Server.

A.4.6.2. Demonstrating cipher key protection

The cipher key with `kid=3236ac0e-7ecf-4376-bcdc-327f55bdf366` was registered by user Jane identified as `sub=af4f2285-979d-389a-892a-90aa9d776476`. The key can be used with a client identified by `client_id=019b7173-a9ed-7d9a-70d3-9502ad7c0575` (attribute `aud` in the key response).

Trying to fetch the cipher key with a different `client_id` or `sub` results in HTTP status code 403. To illustrate this behavior, use the OGC Testbed Token App [https://ogc.secure-dimensions.com/dcs/token-app/]

and login as user `bob` with password `secret`.

CURL request to fetch a cipher key using Bob's access token

```
curl -X GET "https://ogc.secure-dimensions.com/kms/keys/3236ac0e-7ecf-4376-bcdc-
327f55bdf366" -H "accept: application/json" -H "Authorization: Bearer
f4f225d8d7a44b1067cb55b7c48eabf08948e651"
```

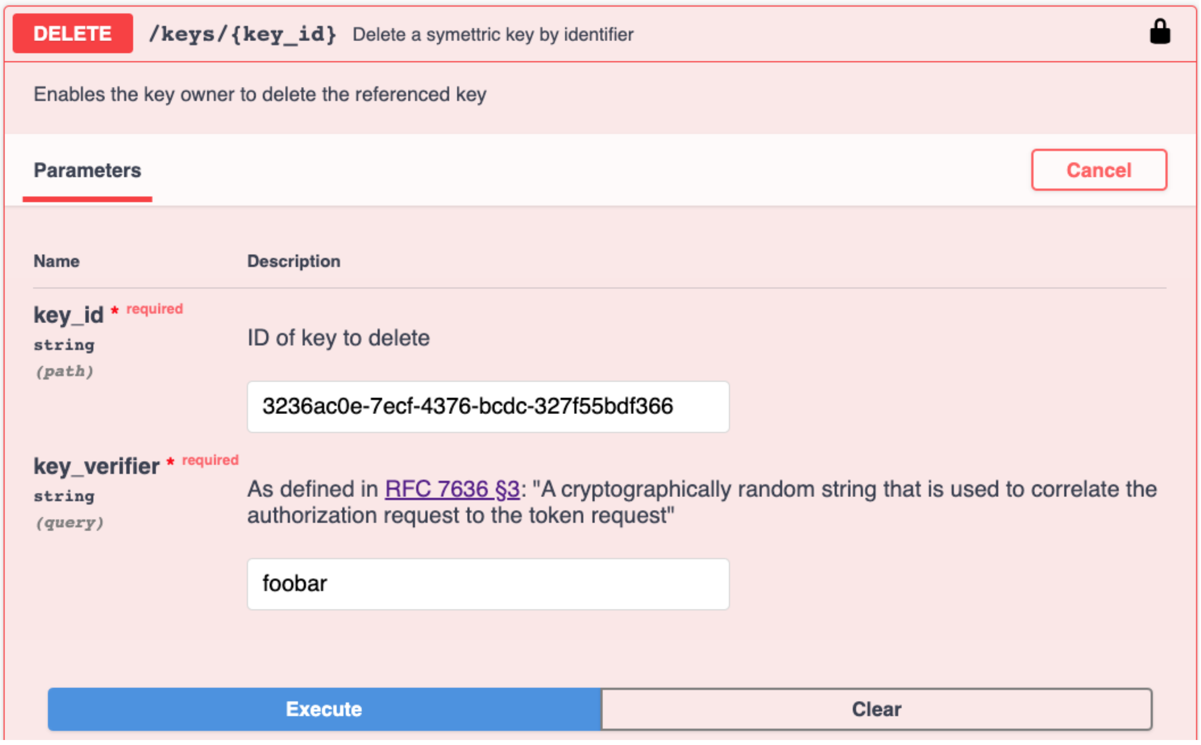
Response forbidden for user Bob

```
{
  "code": 403,
  "error": {
    "type": "INSUFFICIENT_PRIVILEGES",
    "description": "stealing a key?"
  }
}
```

To delete a cipher key, the request must also contain proof or ownership for the key. This proof is presented by the `key_verifier` value that matches the `key_challenge` processed by the `key_challenge_method` send with the registration request. The key registration above used the following values:

- `key_challenge=foobar`
- `key_challenge_method=plain`

The request can be made via OpenAPI like the figure shown next:



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

Enables the key owner to delete the referenced key

Parameters

Name	Description
key_id * required string (path)	ID of key to delete 3236ac0e-7ecf-4376-bcdc-327f55bdf366
key_verifier * required string (query)	As defined in RFC 7636 §3: "A cryptographically random string that is used to correlate the authorization request to the token request" foobar

Figure 33. Key Management Server API to fetch a cipher key

Alternatively, the request can be made using the following CURL request:

Request to delete a cipher key submitting correct `key_verifier`

```
curl -X DELETE "https://ogc.secure-dimensions.com/kms/keys/3236ac0e-7ecf-4376-bcdc-
327f55bdf366?key_verifier=foobar" -H "accept: */*" -H "Authorization: Bearer
0a8859836218963fe0b3588d7d5a9620bda2d100"
```

The deleting of a key only removes the key data in the database which makes the key unusable. The key identifier is kept but marked as inactive. This ensures that no further key can impersonate the same identifier. Therefore, submitting the same request results in HTTP status code 410 - GONE and not a 404 - NOT FOUND.

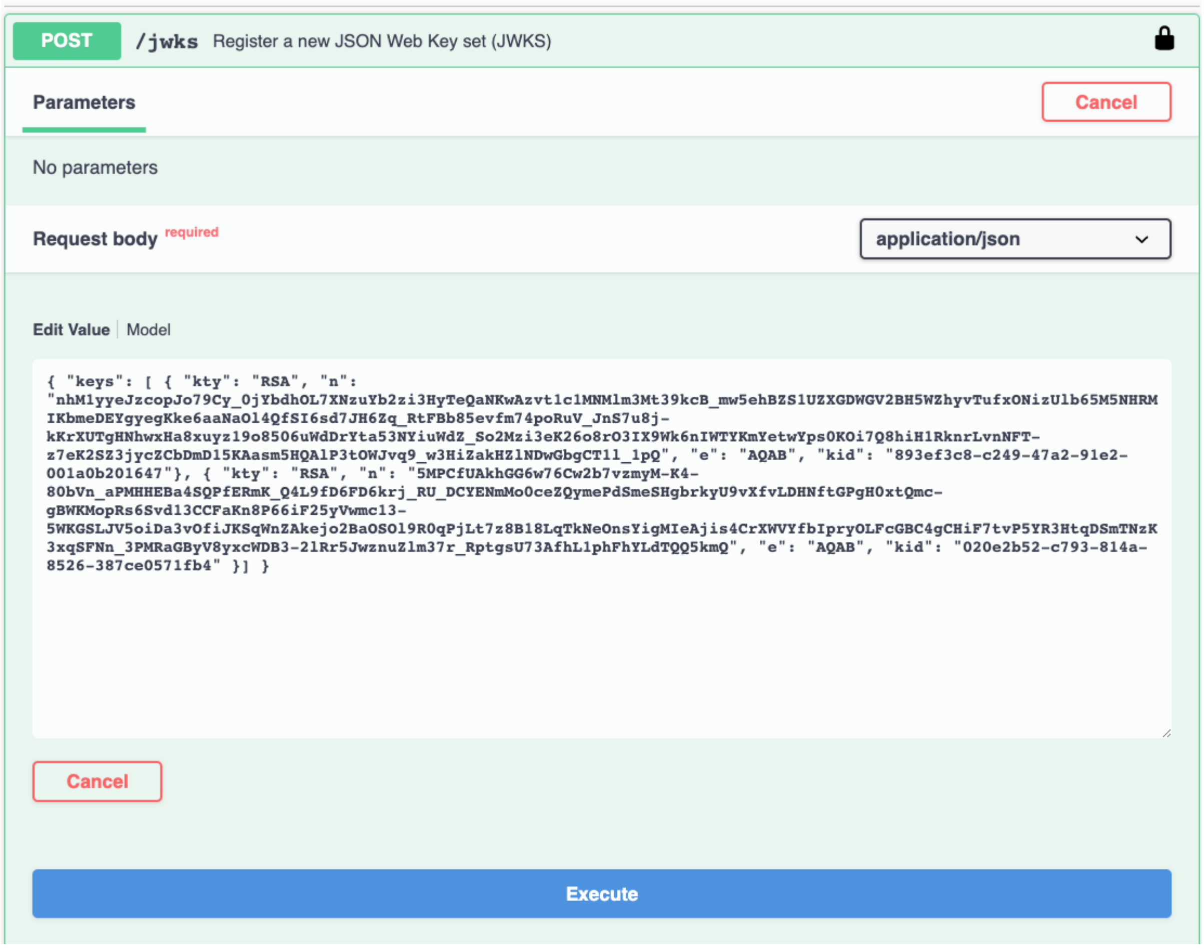
Sending a false `key_verifier` value results in a HTTP 403 status code:

Request to delete a cipher key submitting false `key_verifier`

```
curl -X DELETE "https://ogc.secure-dimensions.com/kms/keys/3236ac0e-7ecf-4376-bcdc-
327f55bdf366?key_verifier=a591a6d40bf420404a011733cfb7b190d62c65bf0bcda32b57b277d9ad9f
146e" -H "accept: */*" -H "Authorization: Bearer
0a8859836218963fe0b3588d7d5a9620bda2d100"
```

A.4.6.3. Managing a Public Key

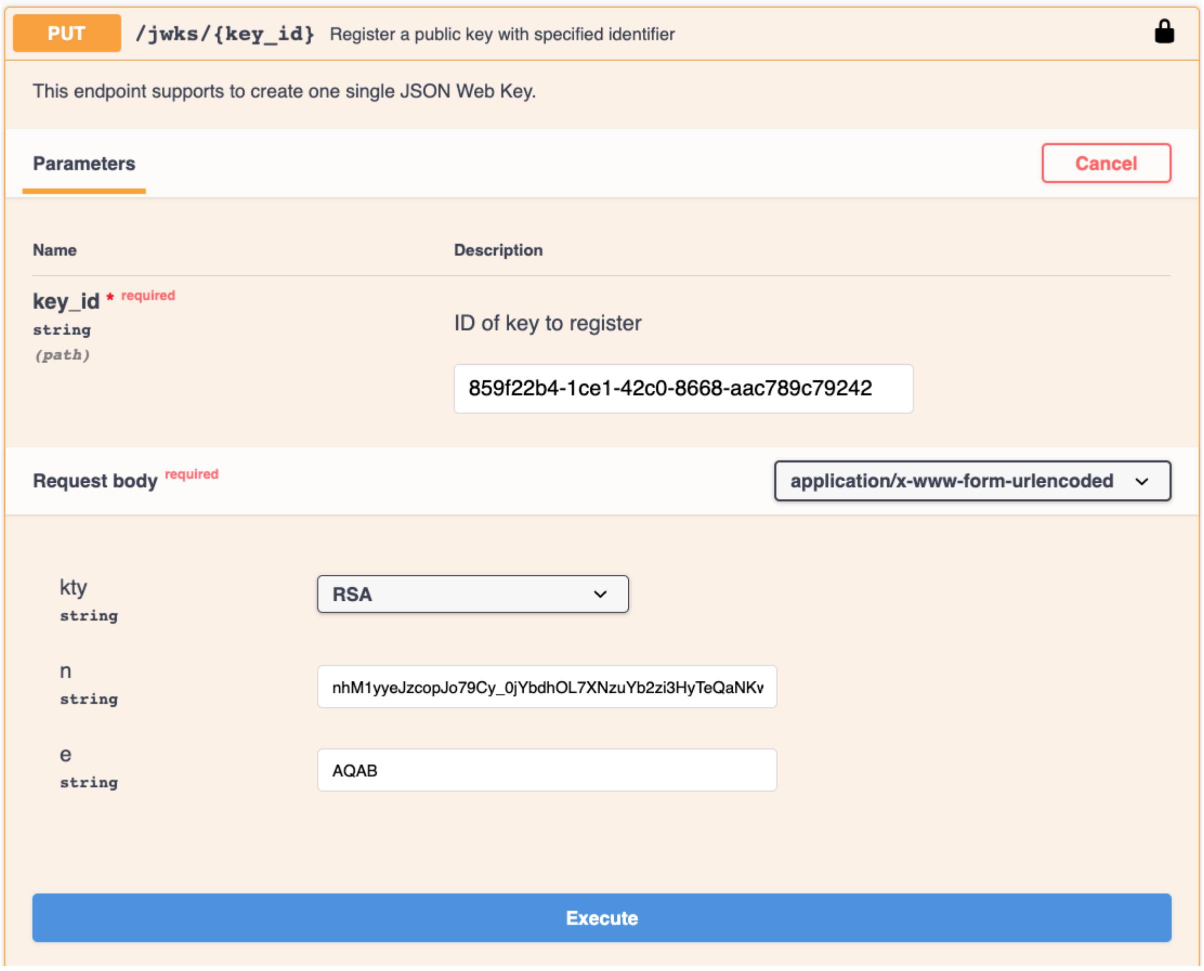
The JSON Web Key interface of the Key Management Server allows to register and obtain private keys. The registration endpoint `/jwks` accepts a JSON Web Key set via POST which allows the bulk registration of public keys. The `/jwks/{key_id}` allows the registration of a single public key via HTTP PUT. Both endpoints require the caller to provide a valid `access_token`.



```
{ "keys": [ { "kty": "RSA", "n":
"nhM1yyeJzcopJo79Cy_0jYbdhOL7XNzuYb2zi3HyTeQaNKwAzvt1c1MNMlm3Mt39kcB_mw5ehBZS1UZXGDWGV2BH5WZhyvTufxONizUlb65M5NHRM
IKbmeDEYgyegKke6aaNaOl4QfSI6sd7JH6Zq_RtFBb85evfm74poRuV_JnS7u8j-
kKrXUTgHNhwxHa8xuyz19o8506uWdDrYta53NYiuWdZ_So2Mzi3eK26o8rO3IX9Wk6nIWTYKmYetwYps0KOi7Q8hiH1RknrLvnNFT-
z7eK2SZ3jycZCbDmD15KAasm5HQAlP3tOWJvq9_w3HiZakHZlNDwGbgCT1l_1pQ", "e": "AQAB", "kid": "893ef3c8-c249-47a2-91e2-
001a0b201647"}, { "kty": "RSA", "n": "5MPCfUAkhGG6w76Cw2b7vzmyM-K4-
80bVn_aPMHHEBa4SQPfERmK_Q4L9fD6FD6krj_RU_DCYENmMo0ceZQymePdSmeSHgbrkyU9vXfvLDHNftGPgH0xtQmc-
gBWKMopRs6Svd13CCFaKn8P66iF25yVwmc13-
5WKGSLJV5oiDa3vOfiJKSqWnZAkejo2BaOSOl9R0qPjLt7z8B18LqTkNeOnsYigMIeAjis4CrXWVYfbIpryOLFcGBC4gCHiF7tvP5YR3HtqDSmTNzK
3xqSFNn_3PMRaGByV8yxcWDB3-2lRr5JwznuZlm37r_RptgsU73AfhL1phFhYLdTQQ5kmQ", "e": "AQAB", "kid": "020e2b52-c793-814a-
8526-387ce0571fb4" }] }
```

Figure 34. Key Management Server API to register a public key set in JWKS format

A public key can be fetched via HTTP GET via the open endpoint `/jwks/{key_id}`.


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

This endpoint supports to create one single JSON Web Key.

Parameters

Name	Description
key_id * required string (path)	ID of key to register: 859f22b4-1ce1-42c0-8668-aac789c79242

Request body required: application/x-www-form-urlencoded

kty string	RSA
n string	nhM1yyeJzcopJo79Cy_0jYbdhOL7XNzuYb2zi3HyTeQaNKv
e string	AQAB

Figure 35. Key Management Server API to register an individual public key in JWK format

For supporting the desktop / server use case, two entities can register public keys: (i) the DCS Server and (ii) the DCS client. The DCS Server can register a public key set to ensure that the response to a DCS key registration is encrypted. The registration of a JSON Web Key set for the client has the same purpose: Ask the Key Management Server to return the response of the GET cipher key request encrypted. An example of such an encrypted response is illustrated above.

The client has an alternative option to use a public key for encrypted cipher key responses: When using Dynamic Client Registration with the Authorization Server, the JWKS become available via the Authorization Server's `Token Introspection` endpoint. The public key set registration via the Authorization Server is implemented for the QGIS DCS plugin.

A.5. DCS Server

The Data Centric Security Server implemented for Testbed-16 is an extension to the Testbed-15 implementation:

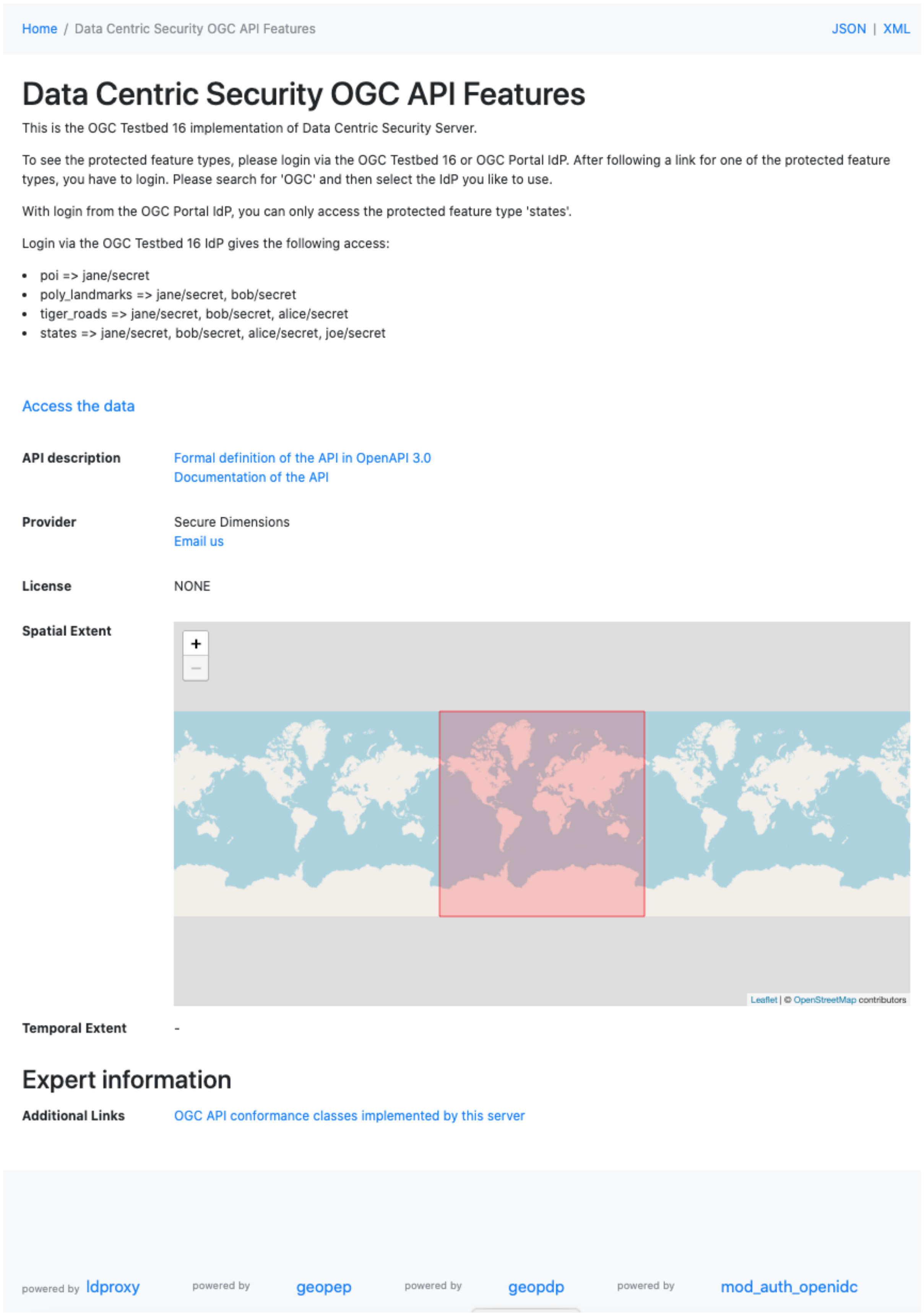
- JSON format including JWT and JWE was implemented to return NATO STANAG 4778 alike data structures. The client can now ask for encrypted data in NATO STANAG 4778 encoded as XML and NATO STANAG 4778 alike encoded as JSON, JWT or JWE.
- Cipher keys are included by reference (rather than inline as for Testbed-15)

As for Testbed-15, the OGC API Features is leveraged on a typical Geoserver data set. In order to demonstrate the ability that the cipher keys change with the classification level of the data objects, the following fictitious classification for the Geoserver standard data set is assumed:

- feature type `poi` is labeled TOP_SECRET
- feature type `poly_landmarks` is labeled SECRET
- feature type `tiger_roads` is labeled CONFIDENTIAL
- feature type `states` is labeled CLASSIFIED

To access the protected data (feature types) four different users are available with different clearance:

- user `jane` has clearance `TOP_SECRET`
- user `bob` has clearance `SECRET`
- user `alice` has clearance `CONFIDENTIAL`
- user `joe` has clearance `CLASSIFIED`


Home / Data Centric Security OGC API Features

JSON | XML

Data Centric Security OGC API Features

This is the OGC Testbed 16 implementation of Data Centric Security Server.

To see the protected feature types, please login via the OGC Testbed 16 or OGC Portal IdP. After following a link for one of the protected feature types, you have to login. Please search for 'OGC' and then select the IdP you like to use.

With login from the OGC Portal IdP, you can only access the protected feature type 'states'.

Login via the OGC Testbed 16 IdP gives the following access:

- poi => jane/secret
- poly_landmarks => jane/secret, bob/secret
- tiger_roads => jane/secret, bob/secret, alice/secret
- states => jane/secret, bob/secret, alice/secret, joe/secret

Access the data

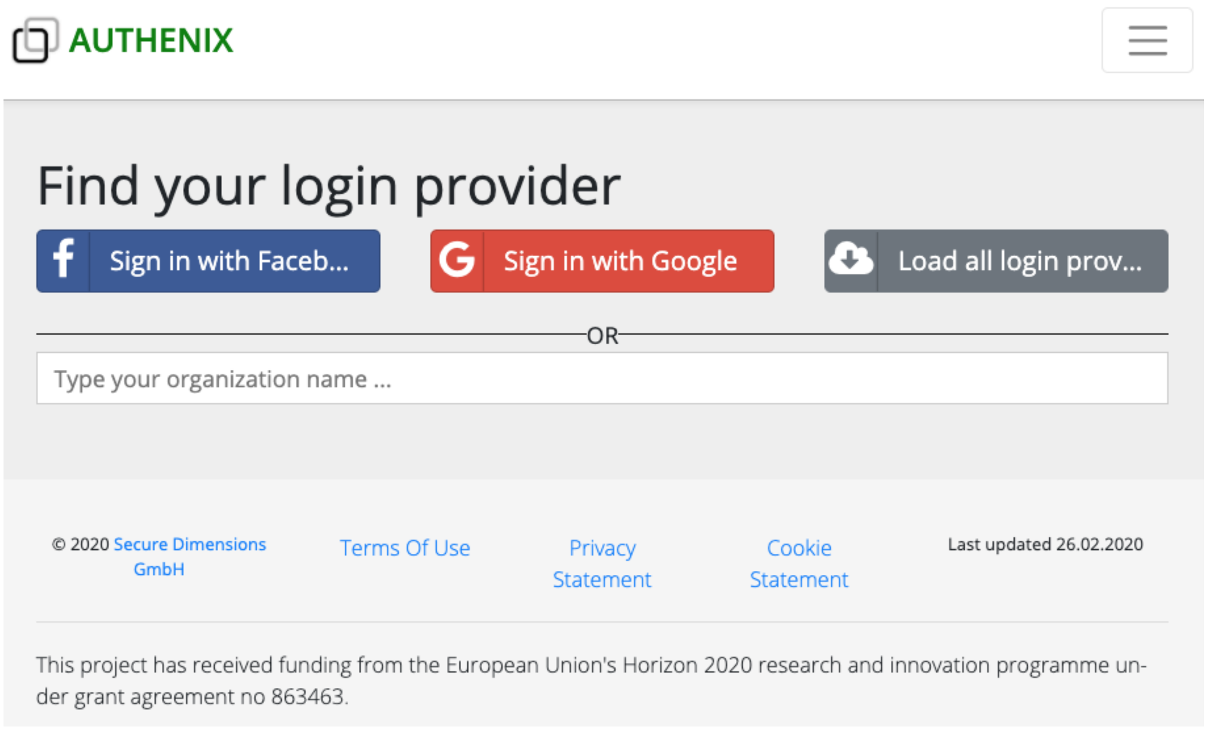
API description	Formal definition of the API in OpenAPI 3.0 Documentation of the API
Provider	Secure Dimensions Email us
License	NONE
Spatial Extent	
Temporal Extent	-

Expert information

Additional Links	OGC API conformance classes implemented by this server

Figure 36. Data Centric Security Server

When following Access the data [https://ogc.secure-dimensions.com/dcs/collections], the feature types from above are marked `(protected)` and a click on the feature type triggers the login via the Authorization Server (AUTHENIX [https://www.authenix.eu]).


AUTHENIX

Find your login provider

Sign in with Faceb... | Sign in with Google | Load all login prov...

OR

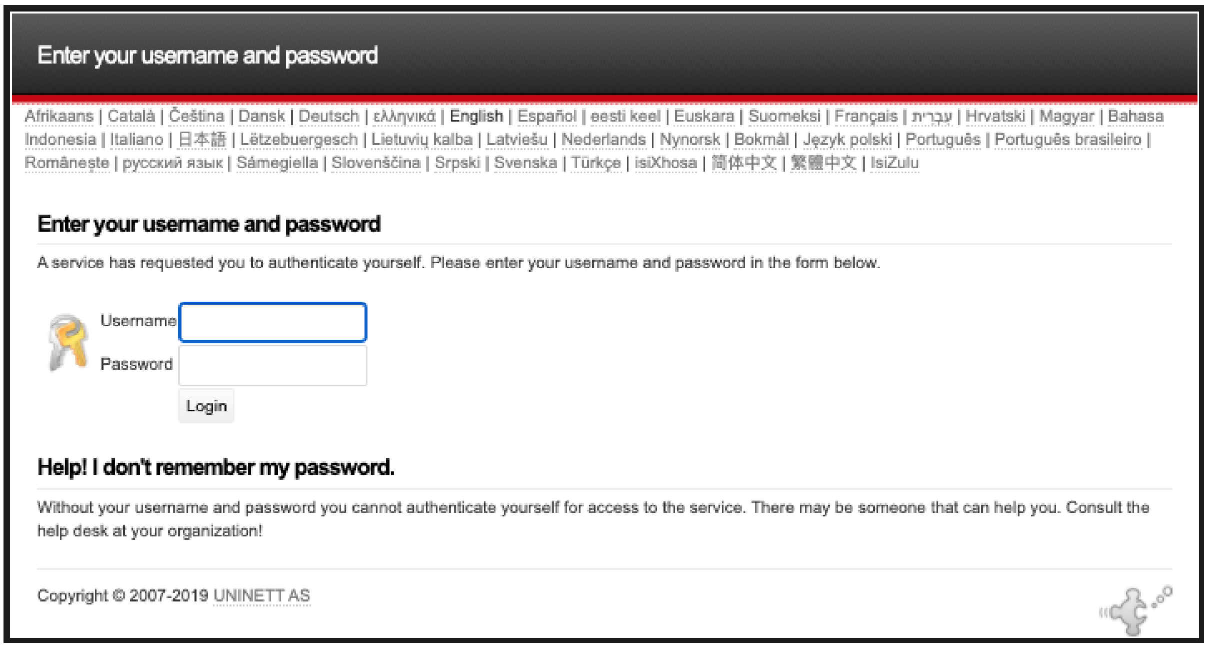
Type your organization name ...

© 2020 Secure Dimensions GmbH | Terms Of Use | Privacy Statement | Cookie Statement | Last updated 26.02.2020

This project has received funding from the European Union's Horizon 2020 research and innovation programme under grant agreement no 863463.

Figure 37. Authorization Server

After searching for the login organization `OGC` the login via the `OGC Testbed IdP` is required (login via another provider will not return encrypted responses).


Enter your username and password

Afrikaans | Català | Čeština | Dansk | Deutsch | ελληνικά | English | Español | eesti keel | Euskara | Suomeksi | Français | עִברִית | Hrvatski | Magyar | Bahasa Indonesia | Italiano | 日本語 | Lëtzebuergesch | Lietuvių kalba | Latviešu | Nederlands | Nynorsk | Bokmål | Język polski | Português | Português brasileiro | Românește | русский язык | Sámegiella | Slovenščina | Srpski | Svenska | Türkçe | isiXhosa | 简体中文 | 繁體中文 | IsiZulu

Enter your username and password

A service has requested you to authenticate yourself. Please enter your username and password in the form below.

Username

Password

Login

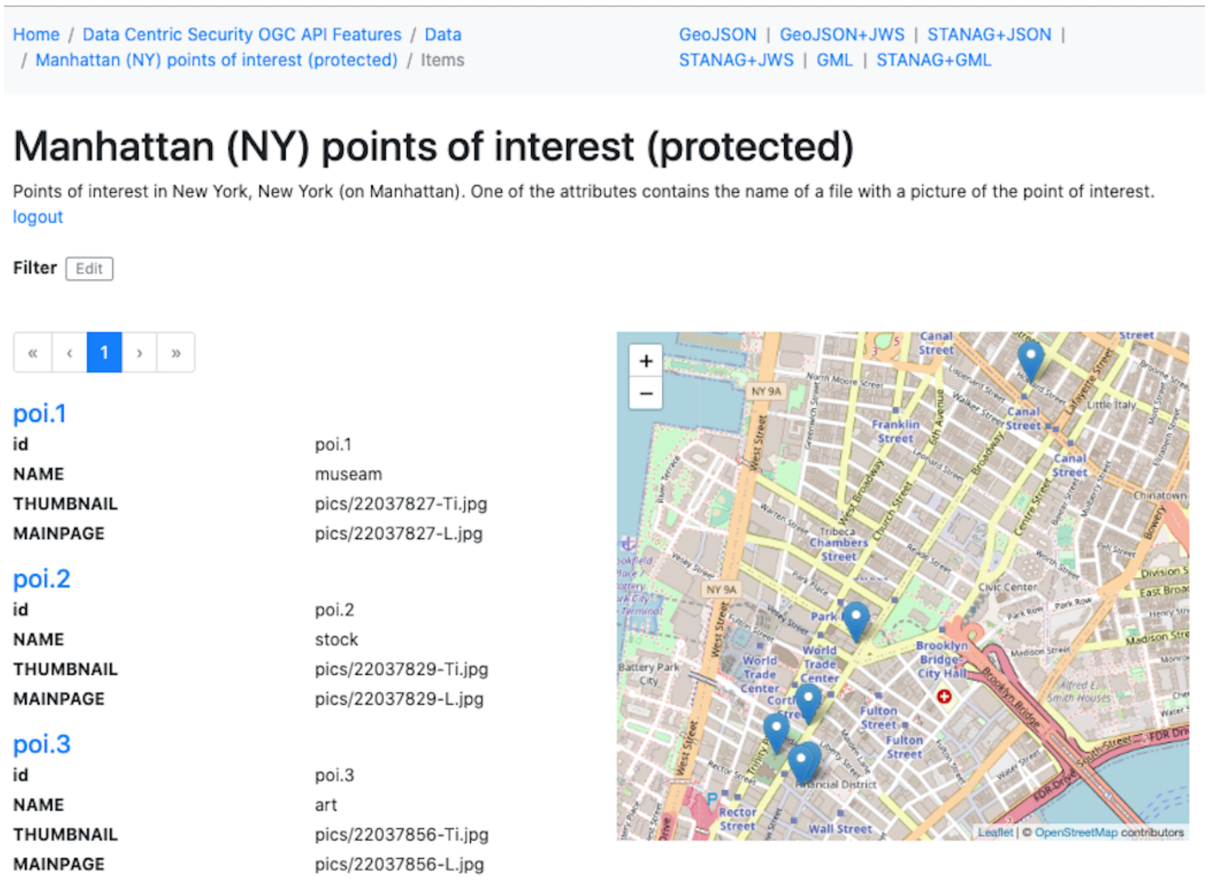
Help! I don't remember my password.

Without your username and password you cannot authenticate yourself for access to the service. There may be someone that can help you. Consult the help desk at your organization!

Copyright © 2007-2019 UNINETT AS

Figure 38. Login via OGC Testbed IdP

After a successful login with one of the users above and the password `secret`, the protected data is displayed in the preview mode. The links in the top right corner provide access to the NATO STANAG encrypted responses.


Home / Data Centric Security OGC API Features / Data / Manhattan (NY) points of interest (protected) / Items

GeoJSON | GeoJSON+JWS | STANAG+JSON | STANAG+JWS | GML | STANAG+GML

Manhattan (NY) points of interest (protected)

Points of interest in New York, New York (on Manhattan). One of the attributes contains the name of a file with a picture of the point of interest.
logout

Filter Edit

« ‹ 1 › »

poi.1

id	poi.1
NAME	museam
THUMBNAIL	pics/22037827-Ti.jpg
MAINPAGE	pics/22037827-L.jpg

poi.2

id	poi.2
NAME	stock
THUMBNAIL	pics/22037829-Ti.jpg
MAINPAGE	pics/22037829-L.jpg

poi.3

id	poi.3
NAME	art
THUMBNAIL	pics/22037856-Ti.jpg
MAINPAGE	pics/22037856-L.jpg

Figure 39. Preview of protected data `poi`

A.5.1. Requesting encrypted data

Requesting encrypted data from the DCS Server can simply be done by following the links in the top right corner:

- `STANAG+GML` returns the STANAG 4778 encoded and encrypted data in XML encoding. Each data object is a feature instance encoded in GML
- `STANAG+JSON` returns the STANAG 4778 alike structure encoded in JSON. Each data element is an encrypted feature instance encoded in GeoJSON.
- `STANAG+JWS` returns the STANAG 4778 alike structure encoded in JSON with digital signature (JWT format). Each data element is an encrypted feature instance encoded in Geo+JSON.
- `GeoJSON+JWS` returns the digitally signed feature collection encoded in GeoJSON

STANAG+GML response

```
<?xml version="1.0"?>
<mb:BindingInformation xmlns:xsi="https://www.w3.org/2001/XMLSchema-instance.xsd"
  xmlns:ds="http://www.w3.org/2000/09/xmldsig#" xmlns:xmime=
"http://www.w3.org/2005/05/xmlmime"
  xmlns:mb="urn:nato:stanag:4778:bindinginformation:1:0"
  xmlns:slab="urn:nato:stanag:4774:confidentialitymetadatalabel:1:0"
  xmlns:wsu="http://docs.oasis-open.org/wss/2004/01/oasis-200401-wss-wssecurity-
utility-1.0.xsd"
  xsi:schemaLocation="urn:nato:stanag:4778:bindinginformation:1:0 4778.xsd">
  <mb:MetadataBindingContainer xml:id="WFS">
```

```
    <mb:MetadataBinding>
      <mb:Metadata xml:id="STANAG4774">
        <EncryptedData xmlns="http://www.w3.org/2001/04/xmlenc#"
          Type="http://www.w3.org/2001/04/xmlenc#Element">
          <EncryptionMethod Algorithm="http://www.w3.org/2009/xmlenc11#aes256-gcm"/>
          <KeyInfo xmlns="http://www.w3.org/2000/09/xmldsig#">
            <KeyName>ed2ee63e-5442-4b6f-a44c-3961e78f5985</KeyName>
          </KeyInfo>
          <CipherData>

<CipherValue>AZL9PyzefgF8ScjO1gEn2SxcgtpF1Rv6r7Wxipvvz/Bz3kL9HPNpHBoolrqty3QY
X+hYnyYkRkA1xLsxIZwYG/wA6kJOXGRXZSJ2GFzNkjb3ct49bgeunRgW8lG16znw
u8IFgvQEIEheEkTjVZHT6s9ZoYZdMVE+9JejRPKNzAFKpeRFL6z2tifn767yKuHD
/KhxeoZoI9Ei9y5uQHSCzAs51981rIk2saEyyJt1HzW/gUAup65EapkK+i9MBWKb
4BBlvOlklhXx5bXyF/RWbuCbae4voLbWAfkbIk4PN6Gyzru5s1+brCzZEvnLgWUf
njp+nAD4Hr/AdXpqmLXj1JUx50kVM1LC1r4pnNARPNNbZ9VIWe4cmT2UA0JBuDBi
abG9jQP2pNaDMjfuUR5Nu6nNQ+yyEKSOWIQfMYHEz2mzJriokGwlCn6LeRR/vd/d
LlY6Q6Y=</CipherValue>
          </CipherData>
        </EncryptedData>
      </mb:Metadata>
      <mb:Metadata xml:id="FeatureType" xmime:contentType="application/xml">
        <EncryptedData xmlns="http://www.w3.org/2001/04/xmlenc#"
          Type="http://www.w3.org/2001/04/xmlenc#Element">
          <EncryptionMethod Algorithm="http://www.w3.org/2001/04/xmlenc#aes128-cbc"/>
          <KeyInfo xmlns="http://www.w3.org/2000/09/xmldsig#">
            <KeyName>3d2e5291-a723-49d0-8881-338ba564a2a4</KeyName>
          </KeyInfo>
          <CipherData>

<CipherValue>s5RIjtrqF4AbFWbNuhuMUkbF8gy0fhvVxVXVikGbk4LBVjdK9rRLxTbiobwbLfIs
VjNaFruAlI5avx7+zJHRguP8EO9JQvEhbF+8mC5Wv2zaokRUtogphF/yOaKMTqUK
ePF0htw442ZU6sbGGGmt1xkXtz0VLeENHYXS0syIr/b/JxhFrGNBp46bZXQuSAP9
AWCfeeOA6vSN7hjkFhtAKRLP9i+Sf1HTMv72El+NkhMTVm5AOWh7wIeTUcu8Z0gW
p+yM6K+6PEGoBTrSKkCnStMAGDO0k+B3zhcaTyP8rY8hdldVIqmgZSixuvVq6iQl
URE4K91ldoK4oDk4+wlrGxuE+VzKZ3N+hyYveKDvjqFxE+b14iXaxN/bvcf5o7zf
35mpOwLFVrQZKsydIxKRSFs2nRKxd8gWTk5QO0WbmQWGiMS9pBJ7WsZbH1B+gr3M
E8T4pDO2DFkxf8eB57FKH+4Yte1u0H4d3UWheoPjCPAQAT/TZZRiX3l2jqg6xdC5
ldldfoSha5HOkNYdrc3oi6eoqIX7SpawF0rVP+EB5yWaLdtp0XJ2vcLKhAY39ybw
Y8P+0K6TxuFvUspYFnsFZl80e34W66eSqX3RFzBUkT5KNNe4ieNDGvX4axlXENjH
+Q+LgrtoCFpqZiQ2XXsNudJducnGypfSiOfY2VrEUJMoj4+r7E42sTN5M/5UlTPy
0D0ntYg2NkKJ9mBqhKVVghj6CHwg2aq6Ttv3GftIvKKJO0SqsGdrNG2sEaI+Pc9i
DWObTpIaNna43kZ7dW0HzfjoZfDoHmycS/Q1lEkao6Y6OxszOh1srgBB3QPvuAG7
QQ0T/vQ2qpgYLhCnCVBSLOPE5oTtC73VIMMygZIzuGsQevbDe2sgHT9HhiD/4lfm
t7GYPjXogeR489EVdIdfAUUq2wzhmE2jugixOGwoRRXq93EQD3tyhn7ECxv5fs0m
65BibQvwhkHiAcZMXC6IpjFqltu24vpN0XPqkaJZWR6jrltWZ5/tL/RUsL2hwi0+
62EiFJzyaz8BTocuV5FupRwbjAwZ4mAqFtahqLIgXQDxTuD5KJay0uuzH8KY/kja
eU+uRuc2fdo3nTh+aLBUBq5qEHYY4Lg+srki6Q/oWRup4vw3ePa9QMrWgBQa3mXP
FIzQcEeyIEBOizi+ZaGugozf3G++5v8kFaamVTXnPSFt39oESSj0+dkGwP9oAbMl
iM9WID3l9yzozHsFGshfOrwix5bJ4V+ET5LAGk4yXiifyCmdT99Y1ZQKVYSkxyMz
MbRGi+WwDpYBP5p8tWjTg4V4wtoJLWTkyvlOFU/BOqPZRxnN6OwOsBUrbn5cGkHr
```

```
2jz8J9BRWtItE1USwVTw0g==</CipherValue>
          </CipherData>
        </EncryptedData>
      </mb:Metadata>
      <mb:Data>
        <EncryptedData xmlns="http://www.w3.org/2001/04/xmlenc#"
          Type="http://www.w3.org/2001/04/xmlenc#Element">
          <EncryptionMethod Algorithm="http://www.w3.org/2009/xmlenc11#aes256-gcm"/>
          <KeyInfo xmlns="http://www.w3.org/2000/09/xmldsig#">
            <KeyName>caea4408-5115-4405-ba4d-ff150cce0f6a</KeyName>
          </KeyInfo>
          <CipherData>

<CipherValue>Hq0a3yJK6ow4MGTzFL8D+o3/WEP4xZVwMtKY4i+nf7UHOWTWL1C/6GkmWkQ9lj+H
r/Aa46eL73yFwrfEwN3EOtZeP1iU0Bh0K31pfXEsjhrwyUfraRFwqIwGv9wp3t2+
d/8aciab9+6j6g6Naw5oUS/+cSSgIc8sTPP58PQGspZiD06GMDNf+Lv5EB9e5ELY
0rHNRcVDW9l7OyM2y+COuMM0eolpt/cDLyh5NEb0SJ9bvn+jX1cLh1loZ48amevq
0Y8LoA4ce7zMXzBsMfNjVASpO/O8iaxhAA2i1KFmLhUVS3JE6Hv6z+wVcj3H/wN1
fFyN9A+yo3vzp0KiFCx9Ofh0wMWWo2o/30Ybwpa2k9XIfSwoktSFLtcz5oPmwpQT
g+1ikyOAHAghnLRR53kTXsOb7GxFm8i1K/gie1fpSeRhPBJKFoQYbKr3atTevIzz
1eCwUXh8Xs4Gkp8joSe//fdJc09sf6LB4rN3rthFg7q/Xrp/Opve1KVaCUCrQYsY
kFoRJ/+5RztexfhXslEY1rfkBaUwpmGJ/8I0Z7/D7+FIQZLOKsvNBcDDqkbwITc1
bVt4sA8TrpC+yTHVOXNEWrC6EP2t1alzxJUlKdi4vwj12XKVqUIx1TkBp8byXi03
bf0tIy7R8n2fQELoEVAA/HJry7pN3dwlJQw2N5Nxl194FIZjG8/nTj6h3z/K0cRT
iLosnyRqAESNoPuf4qVIwsAu1LcPdyY6bb3rRMZBcdDv2/rM0QK/+iYpBLAkG6Wj
PlN6pXAECc7mwfiBfL8JsLbaA9UHt7rwL6ttUWsUH2q0lXAodXWiukJObd7zHQUW
WWHMu3zCXBtBK3TvNKVOwYJldEEVZPg2o8sQ/0pjPshUsQ5ySq046gZ4q5PpB3c/
7C2TrnK2w6GlALWlH/3l1y0VOFtzODbbFCRjyMHuOqEeYBQ/qa0sEy5gBi+ZLgS5
L8cSVy6Gi+dxPkQ7zUdQL4w6UKBx88PDpJAFkjrATi65dGp/UXNsTtb6Uj3AGuf3
KTVV61mpN9pVBxuASq7Qwe3n3z6eATgv03XOQ5crhBOb168CLyYG1cjuoaCNEGmc
kQQeFj9sCf0wSKx8/gc9Atq2CEAR2LEmSMoLzPW4dZyVdI8RxEC5/6eznWohjJUE
qTMsrMJYvhz7WFqY2I099nOsUgwE/RGGdZUXYxn2mNogWjqqbwIkkH5++WoW2lJ7
hTlmHLBx35xI4rmWexBAZjuMIG9SAVu9eNn4hhbbG2+wukpL+EicBPzKmk8QsNYl
8hOSXL02V7jXxrPIHoocmU9XPJpr7D6tYhAn25mFcXiN5wWynNjh8VicAUq4S3mr
S5LjjZ5POo7WNahcYABpWHT86Ykw/usBbtkDohwevF8FTpalDctb5JYGC2KlmoFs
Sn1Naxp5XQ==</CipherValue>
          </CipherData>
        </EncryptedData>
      </mb:Data>
    </mb:MetadataBinding>
  </mb:MetadataBindingContainer>
  <ds:Signature xmlns:ds="http://www.w3.org/2000/09/xmldsig#">
    <ds:SignedInfo>
      <ds:CanonicalizationMethod Algorithm="http://www.w3.org/TR/2001/REC-xml-c14n-
20010315"/>
      <ds:SignatureMethod Algorithm="http://www.w3.org/2000/09/xmldsig#rsa-sha1"/>
      <ds:Reference Id="id" URI="#WFS">
        <ds:Transforms>
          <ds:Transform Algorithm="http://www.w3.org/2000/09/xmldsig#enveloped-
signature"/>
        </ds:Transforms>
        <ds:DigestMethod Algorithm="http://www.w3.org/2000/09/xmldsig#sha1"/>
        <ds:DigestValue>ywlTa+g5URzREK68R9sVy9MVTII=</ds:DigestValue>
      </ds:Reference>
    </ds:SignedInfo>

<ds:SignatureValue>U7BjCkr8DIOm7CWVM7EMDCiol3b0Eavj/kQIJUfi4WA++nm1wH4FNTYLL/s1wAl1
jeLn7g2bchE602OjE7q8/e85OsZEz/LRmXpWJnBg8KlqzPm7y/8oszZTA5memnUa
6SuP70kvKRaTeA9f4j77QS3nlzvPO3krXsk+Sj5HZye3mF91Gc6M0z5TBDRk6z5j
wfrSBaLGkd/0wVE2czHC6ctnivVneVN9R83z4c1jHik3uJoAGSQsEHpVl17rU1ba
h6FfrV/2T604KKUSNQCX82U7q6izlkb8WSHK2IqEAAUc9++gscBH4tOedc7wWixO
WfIdYBUiKCxffVmchjvlxA==</ds:SignatureValue>
    <ds:KeyInfo>
      <ds:KeyName>Dr. No</ds:KeyName>
      <ds:X509Data>
        <ds:X509SubjectName>CN=Andreas Matheus,OU=Secure Dimensions GmbH,O=Secure
Dimensions GmbH,L=Munich,ST=Bavaria,C=DE</ds:X509SubjectName>

<ds:X509Certificate>MIIDuTCCAqGgAwIBAgIEYpLJdjANBgkqhkiG9w0BAQsFADCBjDELMAkGA1UEBhMC
REUxEDAOBgNVBAgTB0JhdmFyaWExDzANBgNVBAcTBk11bmljaDEfMB0GA1UEChMW
U2VjdXJlIERpbWVuc2lvbnMgR21iSDEfMB0GA1UECxMWU2VjdXJlIERpbWVuc2lv
bnMgR21iSDEYMBYGA1UEAxMPQW5kcmVhcyBNYXRoZXVzMB4XDTE1MTAyNTE0NDEw
MVoXDTE2MDEyMzE0NDEwMVowgYwxCzAJBgNVBAYTAkRFMRAwDgYDVQQIEwdCYXZh
cmlhMQ8wDQYDVQQHEwZNdW5pY2gxHzAdBgNVBAoTFlNlY3VyZSBEaW1lbnNpb25z
IEdtYkgxHzAdBgNVBAsTFlNlY3VyZSBEaW1lbnNpb25zIEdtYkgxGDAWBgNVBAMT
D0FuZHJlYXMgTWF0aGV1czCCASIwDQYJKoZIhvcNAQEBBQADggEPADCCAQoCggEB
AJBxrjwhMmOGnSKT4DLsOx+R+c4dN3gA74/03NdsxUdy2r6QB65AvF8Rm3YF5pJy
Hzdrlf43IObjOHK2yRn6p0tXpc5yYwBGd3tZMGTkyj4qhqqy/ug4LxYy4HYfCXE/
ec9UOTCDu7vfkbvmEfg8V0M2DfT6t5XnvFZmkUkSAi4L4vQ9PJthsFLyJXq2nNlh
tOMQeBWxcOzbog6EBAB7qaUyumlrrIojksHd9Tb4Om/BIp+JxcocRjGmSq7XoKZ1
GuXmWXSnrc877AnET/+Kbea4zqH+Oo44zP2G0XdCCMiKtL7nxqIAfwucp3SEGtqH
XGNv61RGsqihQbtlbhRkprcCAwEAAaMhMB8wHQYDVR0OBBYEFIVLBZDvNUo/OX9F
MKRLz7OFaUXXMA0GCSqGSIb3DQEBCwUAA4IBAQCA7FkGI0EOkJPr4yjCT8HxJvAd
lzNW539tl/SVYe4ducBm4J523G6POKvz6kVHbS30J2HiNd2FoQL9s2DMPN2ag9Q3
myzI8E9x8dowNKhaupmTJI/Ednegnp7pr/8/o612qBXTf00T4j8QP9mZxUreqC+x
TCV9GCO0XuIVpBM6sGbEiFfjg0xLs3HO7kBHla78WAb8EyZGv9aoHCsqoIE+A/L9
e++xrY09TN/wjJKrv665iRF3XG+WHj0lrUvzlPZzNHbLykqSo48DhDc/JmaadiqZ
cNFF8NBHOLzicsSo+GpeEnSJBKnCYwxStWJ+dFWoHQxwyHrkn+Om+EiQ6/2w</ds:X509Certificate>
      </ds:X509Data>
    </ds:KeyInfo>
  </ds:Signature>
</mb:BindingInformation>
```

The response is a NATO STANAG 4778 encoded XML instance document inside element `<mb:BindingInformation>`. The entire response is digitally signed (`<ds:Signature>` element). Each feature is included in the `<mb:Data>` element along with metadata. The response above contains two encrypted metadata elements where the `<mb:Metadata xml:id="STANAG4774">` element contains the NATO STANAG 4774 metadata and the `<mb:Metadata xml:id="FeatureType" xmime:contentType="application/xml">` element contains the XML Schema for the data structure of the included feature (DescribeFeatureType response with WFS 2.0). The data and the two metadata elements are encrypted with different cipher keys. The references in the response can be resolved via the Key Management Server (`/dcs/{key_id}`) endpoint as described above.

STANAG+JWS response

```
eyJhbGciOiAiUlMyNTYiLCAia2lkIjogIkRyLiBObyIsICJjdHkiOiAiYXBwbGljYXRpb24vc3RhbmFnK2pzb2
4ifQ.ewogICAgInR5cGUiOiAiU1RBTkFHNDc3OCIsCiAgICAidGltc3RhbXAiOiAiMjAyMC0xMC0xMlQwOToxM
TozM1oiLAogICAgIm51bWJlclJldHVybmVkIjogMSwKICAgICJudW1iZXJNYXRjaGVkIjogNiwKICAgICJsaW5
rcyI6IFsKICAgICAgICB7CiAgICAgICAgICAgICJocmVmIjogImh0dHBzOlwvXC9vZ2Muc2VjdXJlLWRpbWVuc
2lvbnMuY29tXC9kY3NcL2NvbGxlY3Rpb25zXC9wb2lcL2l0ZW1zP2xpbWl0PTEmZj1zdGFuYWcrandzJmtleV9
jaGFsbGVuZ2U9a2V5X2NoYWxsZW5nZV9tZXRob2Q9IiwKICAgICAgICAgICAgInJlbCI6ICJzZWxmIiwKICAgI
CAgICAgICAgInR5cGUiOiAiYXBwbGljYXRpb25cL3N0YW5hZytqd3MiLAogICAgICAgICAgICAidGl0bGUiOiA
iVGhpcyBkb2N1bWVudCIKICAgICAgICB9LAogICAgICAgIHsKICAgICAgICAgICAgImhyZWYiOiAiaHR0cHM6X
C9cL29nYy5zZWN1cmUtZGltZW5zaW9ucy5jb21cL2Rjc1wvY29sbGVjdGlvbnNcL3BvaVwvaXRlbXM_Zj1zdGF
uYWcrandzJmxpbWl0PTEmb2Zmc2V0PTEma2V5X2NoYWxsZW5nZT1rZXlfY2hhbGxlbmdlX21ldGhvZD0iLAogI
CAgICAgICAgICAicmVsIjogIm5leHQiLAogICAgICAgICAgICAidHlwZSI6ICJhcHBsaWNhdGlvblwvc3RhbmF
nK2p3cyIsCiAgICAgICAgICAgICJ0aXRsZSI6ICJOZXh0IHBhZ2UiCiAgICAgICAgfSwKICAgICAgICB7CiAgI
CAgICAgICAgICJyZWwiOiAiYWx0ZXJuYXRlIiwKICAgICAgICAgICAgImhyZWYiOiAiaHR0cHM6XC9cL29nYy5
zZWN1cmUtZGltZW5zaW9ucy5jb21cL2Rjc1wvY29sbGVjdGlvbnNcL3BvaVwvaXRlbXM_Zj1zdGFuYWcrandzJ
mxpbWl0PTEmIiwKICAgICAgICAgICAgInR5cGUiOiAiYXBwbGljYXRpb25cL2dlbytqc29uIiwKICAgICAgICA
gICAgInRpdGxlIjogIlRoaXMgZG9jdW1lbnQgYXMgR2VvSlNPTiIKICAgICAgICB9LAogICAgICAgIHsKICAgI
CAgICAgICAgInJlbCI6ICJhbHRlcm5hdGUiLAogICAgICAgICAgICAiaHJlZiI6ICJodHRwczpcL1wvb2djLnN
lY3VyZS1kaW1lbnNpb25zLmNvbVwvZGNzXC9jb2xsZWN0aW9uc1wvcG9pXC9pdGVtcz9mPXN0YW5hZytqd3Mmb
GltaXQ9MSYiLAogICAgICAgICAgICAidHlwZSI6ICJhcHBsaWNhdGlvblwvZ2VvK2p3cyIsCiAgICAgICAgICA
gICJ0aXRsZSI6ICJUaGlzIGRvY3VtZW50IGFzIGRpZ2l0YWxseSBzaWduZWQgR2VvSlNPTiIKICAgICAgICB9L
AogICAgICAgIHsKICAgICAgICAgICAgInJlbCI6ICJhbHRlcm5hdGUiLAogICAgICAgICAgICAiaHJlZiI6ICJ
odHRwczpcL1wvb2djLnNlY3VyZS1kaW1lbnNpb25zLmNvbVwvZGNzXC9jb2xsZWN0aW9uc1wvcG9pXC9pdGVtc
z9mPXN0YW5hZytqd3MmbGltaXQ9MSYiLAogICAgICAgICAgICAidHlwZSI6ICJhcHBsaWNhdGlvblwvc3RhbmF
nK2dtbCIsCiAgICAgICAgICAgICJ0aXRsZSI6ICJUaGlzIGRvY3VtZW50IGFzIFNUQU5BRyArIEdNTCIKICAgI
CAgICB9LAogICAgICAgIHsKICAgICAgICAgICAgInJlbCI6ICJhbHRlcm5hdGUiLAogICAgICAgICAgICAiaHJ
lZiI6ICJodHRwczpcL1wvb2djLnNlY3VyZS1kaW1lbnNpb25zLmNvbVwvZGNzXC9jb2xsZWN0aW9uc1wvcG9pX
C9pdGVtcz9mPXN0YW5hZytqd3MmbGltaXQ9MSYiLAogICAgICAgICAgICAidHlwZSI6ICJhcHBsaWNhdGlvblw
veG1sK2dtbDtjb250ZW50PWdtbDtwcm9maWxlPVwiaHR0cDpcL1wvd3d3Lm9wZW5naXMubmV0XC9kZWZcL3Byb
2ZpbGVcL29nY1wvMi4wXC9nbWwtc2YyXCI7dmVyc2lvbj0zLjJcIiIsCiAgICAgICAgICAgICJ0aXRsZSI6ICJ
UaGlzIGRvY3VtZW50IGFzIEdNTCIKICAgICAgICB9LAogICAgICAgIHsKICAgICAgICAgICAgInJlbCI6ICJhb
HRlcm5hdGUiLAogICAgICAgICAgICAiaHJlZiI6ICJodHRwczpcL1wvb2djLnNlY3VyZS1kaW1lbnNpb25zLmN
vbVwvZGNzXC9jb2xsZWN0aW9uc1wvcG9pXC9pdGVtcz9mPXN0YW5hZytqd3MmbGltaXQ9MSYiLAogICAgICAgI
CAgICAidHlwZSI6ICJ0ZXh0XC9odG1sIiwKICAgICAgICAgICAgInRpdGxlIjogIlRoaXMgZG9jdW1lbnQgYXM
gSFRNTCIKICAgICAgICB9CiAgICBdLAogICAgIm9iamVjdHMiOiBbCiAgICAgICAgewogICAgICAgICAgICAiT
WV0YWRhdGEiOiB7CiAgICAgICAgICAgICAgICAiQ29uZmlkZW50aWFsaXR5SW5mb3JtYXRpb24iOiB7CiAgICA
gICAgICAgICAgICAgICAgIlBvbGljeeUlkZW50aWZpZXIiOiAiVEIxNiIsCiAgICAgICAgICAgICAgICAgICAgI
kNsYXNzaWZpY2F0aW9uIjogInRvcF9zZWNyZXQiCiAgICAgICAgICAgICAgICB9LAogICAgICAgICAgICAgICA
gIkNyZWF0aW9uRGF0ZVRpbWUiOiAiMjAyMC0xMC0xMlQwOToxMTozM1oiCiAgICAgICAgICAgIH0sCiAgICAgI
CAgICAgICJEYXRhIjogImV5SmhiR2NpT2lBaVpHbHlJaXdnSW1WdVl5STZJQ0pCTWpVMlEwSkRMVWhUTlRFeUl
pd2dJbXRwWkNJNklDSmtPR1ptWTJNeU15MHdPVGRpTFRRMVltUXRZbU5sTVMweFpUYzFPRGxoWlRBMllUUWlmU
S4uazVEOW9XNmFDUGI5N0dxd1BsQm9BZy5LY3NWcnNVa1psYjBjakF5Yk44UVJjdWd4WlFPOEVDWFFsNW1HLWl
oMGNGZTRsSkNaUWo0bTFYR0dzV19VcENNVUXIyQ25LRkx2NHNjZWZBc3ZZU1NXNEw2eGdUeGxRSE9jdl9XYmFqd
3RxUWZzR210RkdhRWtwU0piSkhjTlZYRktKYm1iVV9YaTZ0TDdHYnJqWV9rOXZjUXV1TlVXd3hNa1J5U2l0X1o
4M2p2VnkzMGYzSWtpV1hBZ0xlcGQ0a0xpVUcyNTN2c2xWR1ktM2dneU15WnZkTU02TzV3MEtRWm9MUlFvRGxVb
TlDcmJsTFRnUHRnOF96LXhuVVlUdFhHbHdqVEN2OXBKUWYyWVNLem9GaDBXRUxYZjJDZGJhNmxxRWh5TWJnX1F
rTU9SRlNqRkVSbE5iSnUtVjRmemJ0UjVzVEFmTHR6TDVUU1M5cnliNENvYmc4M0hNS0lTSlRBeDEwZTJxRTJPW
EFqa2pCM3JGdlJsRmF0dWpPQ0dkd3VHQk9tc1ZOMVZtaFNyTDBUUDNHRkdZYXhNSlRBa2lLdUdCQ2x3NEJESUR
```

```
FLlRuY1J6YlVsQUx6TkNCQm5YRFpTdVdRU0JnSnkzREZGUlpRN2diUFNhZjgiCiAgICAgICAgfQogICAgXQp9C
g.o3GEWjM37ydZGYovsABi8E8ECFjBNDojJgeHF3Pp-kS4yX0IFdDQENUdwFt1QHKifkeQG4-
sdAo8HfrcWIPdsXnWNtEdu5NGuCpzBNu7HbXdbTdDeCe4xuEnEO-
5Dy23kGTuYJKkj6QeQYE9YV81whA2tqjukUzPt0DIqC-pS-RDBs0K63GsZRZUtZCuGbn-5GcqKPZ-
OCl3nTqt8PDbN8QcKz2UK1jHoHmgOfGQRdbfqiN1NkN56f8glpPl2QkvoFsPMAVScGmL56-UHtDQgGF3O7dr-
H2q2WTPhuRikLRp9F8DVZv4yA3SP72MH9mzP4kKxbWZvx5ZFQE-NNrrgw
```

Decoding the JWT header shows that the algorithm for the digital signature is `RS256` and the key name is `Dr. No`. The `cty` (content type) is expressed to be STANAG+JSON.

JWT header

```
{
  "alg": "RS256",
  "kid": "Dr. No",
  "cty": "application/stanag+json"
}
```

JWT payload

```
{
    "type": "STANAG4778",
    "timstamp": "2020-10-12T09:11:33Z",
    "numberReturned": 1,
    "numberMatched": 6,
    "links": [
        {
            "href": "https:\/\/ogc.secure-dimensions.com\/dcs\/collections\/poi
\/items?limit=1&f=stanag+jws&key_challenge=key_challenge_method=",
            "rel": "self",
            "type": "application\/stanag+jws",
            "title": "This document"
        },
        {
            "href": "https:\/\/ogc.secure-dimensions.com\/dcs\/collections\/poi
\/items?f=stanag+jws&limit=1&offset=1&key_challenge=key_challenge_method=",
            "rel": "next",
            "type": "application\/stanag+jws",
            "title": "Next page"
        },
        {
            "rel": "alternate",
            "href": "https:\/\/ogc.secure-dimensions.com\/dcs\/collections\/poi
\/items?f=stanag+jws&limit=1&",
            "type": "application\/geo+json",
            "title": "This document as GeoJSON"
        },
        {
            "rel": "alternate",
            "href": "https:\/\/ogc.secure-dimensions.com\/dcs\/collections\/poi
\/items?f=stanag+jws&limit=1&",
            "type": "application\/geo+jws",
            "title": "This document as digitally signed GeoJSON"
        },
        {
            "rel": "alternate",
            "href": "https:\/\/ogc.secure-dimensions.com\/dcs\/collections\/poi
\/items?f=stanag+jws&limit=1&",
            "type": "application\/stanag+gml",
            "title": "This document as STANAG + GML"
        },
        {
            "rel": "alternate",
            "href": "https:\/\/ogc.secure-dimensions.com\/dcs\/collections\/poi
\/items?f=stanag+jws&limit=1&",
            "type": "application\/xml+gml;content=gml;profile=\"http:\/
\/www.opengis.net\/def\/profile\/ogc\/2.0\/gml-sf2\";version=3.2\"",
            "title": "This document as GML"
        },
        {
            "rel": "alternate",
            "href": "https:\/\/ogc.secure-dimensions.com\/dcs\/collections\/poi
\/items?f=stanag+jws&limit=1&",
            "type": "text\/html",
            "title": "This document as HTML"
        }
    ],
    "objects": [
        {
            "Metadata": {
                "ConfidentialityInformation": {
                    "PolicyIdentifier": "TB16",
                    "Classification": "top_secret"
                },
                "CreationDateTime": "2020-10-12T09:11:33Z"
            },
            "Data":
"eyJhbGciOiAiZGlyIiwgImVuYyI6ICJBMjU2Q0JDLUhTNTEyIiwgImtpZCI6ICJkOGZmY2MyMy0wOTdiLTQ1Y
mQtYmNlMS0xZTc1ODlhZTA2YTQifQ..k5D9oW6aCPb97GqwPlBoAg.KcsVrsUkZlb0cjAybN8QRcugxZQO8ECX
Ql5mG-
ih0cFe4lJCZQj4m1XGGsW_UpCUQr2CnKFLv4scefAsvYSSW4L6xgTxlQHOcv_WbajwtqQfsGmtFGaEkpSJbJHc
NVXFKJbmbU_Xi6tL7GbrjY_k9vcQuuNUWwxMkRySit_Z83jvVy30f3IkiWXAgLepd4kLiUG253vslVGY-
3ggyMyZvdMM6O5w0KQZoLRQoDlUm9CrblLTgPtg8_z-
xnUYTtXGlwjTCv9pJQf2YSKzoFh0WELXf2Cdba6lqEhyMbg_QkMORFSjFERlNbJu-
V4fzbtR5sTAfLtzL5TSS9ryb4Cobg83HMKISJTAx10e2qE2OXAjkjB3rFvRlFatujOCGdwuGBOmsVN1VmhSrL0
TP3GFGYaxMJTAkiKuGBClw4BDIDE.TncRzbUlALzNCBBnXDZSuWQSBgJy3DFFRZQ7gbPSaf8"
        }
    ]
}
```

STANAG+JSON response

```
{
    "type": "STANAG4778",
    "timstamp": "2020-10-12T08:59:11Z",
    "numberReturned": 1,
    "numberMatched": 6,
    "links": [
        {
            "href": "https:\/\/ogc.secure-dimensions.com\/dcs\/collections\/poi
\/items?limit=1&f=stanag+json&key_challenge=key_challenge_method=",
            "rel": "self",
            "type": "application\/stanag+json",
            "title": "This document"
        },
        {
            "href": "https:\/\/ogc.secure-dimensions.com\/dcs\/collections\/poi
\/items?f=stanag+json&limit=1&offset=1&key_challenge=key_challenge_method=",
            "rel": "next",
            "type": "application\/stanag+json",
            "title": "Next page"
        },
        {
            "rel": "alternate",
            "href": "https:\/\/ogc.secure-dimensions.com\/dcs\/collections\/poi
\/items?f=stanag+json&limit=1&",
            "type": "application\/geo+json",
            "title": "This document as GeoJSON"
        },
        {
            "rel": "alternate",
            "href": "https:\/\/ogc.secure-dimensions.com\/dcs\/collections\/poi
\/items?f=stanag+json&limit=1&",
            "type": "application\/geo+jws",
            "title": "This document as digitally singed GeoJSON"
        },
        {
            "rel": "alternate",
            "href": "https:\/\/ogc.secure-dimensions.com\/dcs\/collections\/poi
\/items?f=stanag+json&limit=1&",
            "type": "application\/stanag+jws",
            "title": "This document as digitally signed STANAG in JSON"
        },
        {
            "rel": "alternate",
            "href": "https:\/\/ogc.secure-dimensions.com\/dcs\/collections\/poi
\/items?f=stanag+json&limit=1&",
            "type": "application\/stanag+gml",
            "title": "This document as STANAG + GML"
        },
        {
            "rel": "alternate",
            "href": "https:\/\/ogc.secure-dimensions.com\/dcs\/collections\/poi
\/items?f=stanag+json&limit=1&",
            "type": "application\/xml+gml;content=gml;profile=\"http:\/
\/www.opengis.net\/def\/profile\/ogc\/2.0\/gml-sf2\";version=3.2\"",
            "title": "This document as GML"
        },
        {
            "rel": "alternate",
            "href": "https:\/\/ogc.secure-dimensions.com\/dcs\/collections\/poi
\/items?f=stanag+json&limit=1&",
            "type": "text\/html",
            "title": "This document as HTML"
        }
    ],
    "objects": [
        {
            "Metadata": {
                "ConfidentialityInformation": {
                    "PolicyIdentifier": "TB16",
                    "Classification": "top_secret"
                },
                "CreationDateTime": "2020-10-12T08:59:11Z"
            },
            "Data":
"eyJhbGciOiAiZGlyIiwgImVuYyI6ICJBMjU2Q0JDLUhTNTEyIiwgImtpZCI6ICI4N2RjODE0Yi02N2E2LTQwY
jgtOWMzNi00NDZiMzkwZjk1YmIifQ..qrZPZESremlwevxM0XemgA.ru9PkdcEveUMPiyNVau6BM36tv2c1fEK
Nz0bMPseTf1djcsHdJ13zN8d3dRbZwC6hgcGkaGES6Qo8OjjQeq1rRWhGG_2FVb1ttJmJcKA4orBrsPEH96aP8
yz-0lACHAhKXIX_xw9efvpTXkpVyiubfGHVe6--wrC-IH_WOTSZW-tKOkAz8ud92oNZKLh4O1xT3RmVb0uW-
w_BNcOaYBVubEBw_nII6J9ZFW30haHR32vAdaRhESYSats8jicVRNFvc5-MlK-
t4MNs9OxqI7OJzZ5KpflAXuBg6v1tzI6yuIQrqazcpPdUGpkVZi4b5CfbFK1Kjyz040Ld7zKsJMIXgDxTYB7Sl
Pq4xRCCvaCyypzMAIlYbR5Uo-Aqlzkn_O8Qm1IDfA07iqJCInpcVmEdTPCniOuxugbWQYpqtRPL-
Q.izLY7AmM1osMOnwI7CU7Sr-VoxwgNdSATrdtwp7IUZo"
        }
    ]
}
```

The response includes a JSON data structure with an array of (data) objects where each element contains a Metadata and a Data element. The Metadata element above contains the unencrypted description mimicking the STANAG 4774 structure. The Data element is in JWE encrypted format using the compact serialization. Decoding the header shows:

```
{
  "alg": "dir",
  "enc": "A256CBC-HS512",
  "kid": "87dc814b-67a6-40b8-9c36-446b390f95bb"
}
```

The `alg=dir` denotes direct encoding of the data with a symmetric cipher key where the key is not included inline. This results in the two dots (..) separating the JWE header and initialization vector.

For inline cipher keys, the encrypted key would be included between those dots. The `enc=A256CBC-HS512` defines the cipher key algorithm (A256CBC) and the hashing algorithm (HS512) to compute the authentication tag. The actual key `kid=87dc814b-67a6-40b8-9c36-446b390f95bb` must be obtained from the Key Management Server.

GeoJSON+JWS response

```
eyJhbGciOiAiUlMyNTYiLCAia2lkIjogIkRyLiBObyIsICJjdHkiOiAiYXBwbGljYXRpb24vZ2VvK2pzb24ifQ
.ewogICAgInR5cGUiOiAiRmVhdHVyZUNvbGxlY3Rpb24iLAogICAgImxpbmtzIjogWwogICAgICAgIHsKICAgI
CAgICAgICAgImhyZWYiOiAiaHR0cHM6XC9cL29nYy5zZWN1cmUtZGltZW5zaW9ucy5jb21cL2Rjc1wvY29sbGV
jdGlvbnNcL3BvaVwvaXRlbXM_Zj1nZW8randzJmtleV9jaGFsbGVuZ2U9a2V5X2NoYWxsZW5nZV9tZXRob2Q9I
iwKICAgICAgICAgICAgInJlbCI6ICJzZWxmIiwKICAgICAgICAgICAgInR5cGUiOiAiYXBwbGljYXRpb25cL2d
lbytqd3MiLAogICAgICAgICAgICAidGl0bGUiOiAiVGhpcyBkb2N1bWVudCIKICAgICAgICB9LAogICAgICAgI
HsKICAgICAgICAgICAgInJlbCI6ICJhbHRlcm5hdGUiLAogICAgICAgICAgICAiaHJlZiI6ICJodHRwczpcL1w
vb2djLnNlY3VyZS1kaW1lbnNpb25zLmNvbVwvZGNzXC9jb2xsZWN0aW9uc1wvcG9pXC9pdGVtcz9mPWp3cyYiL
AogICAgICAgICAgICAidHlwZSI6ICJhcHBsaWNhdGlvblwvZ2VvK2pzb24iLAogICAgICAgICAgICAidGl0bGU
iOiAiVGhpcyBkb2N1bWVudCBhcyBHZW9KU09OIgogICAgICAgIH0sCiAgICAgICAgewogICAgICAgICAgICAic
mVsIjogImFsdGVybmF0ZSIsCiAgICAgICAgICAgICJocmVmIjogImh0dHBzOlwvXC9vZ2Muc2VjdXJlLWRpbWV
uc2lvbnMuY29tXC9kY3NcL2NvbGxlY3Rpb25zXC9wb2lcL2l0ZW1zP2Y9andzJiIsCiAgICAgICAgICAgICJ0e
XBlIjogImFwcGxpY2F0aW9uXC9zdGFuYWcranNvbiIsCiAgICAgICAgICAgICJ0aXRsZSI6ICJUaGlzIGRvY3V
tZW50IGFzIFNUQU5BRyArIEpTT04iCiAgICAgICAgfSwKICAgICAgICB7CiAgICAgICAgICAgICJyZWwiOiAiY
Wx0ZXJuYXRlIiwKICAgICAgICAgICAgImhyZWYiOiAiaHR0cHM6XC9cL29nYy5zZWN1cmUtZGltZW5zaW9ucy5
jb21cL2Rjc1wvY29sbGVjdGlvbnNcL3BvaVwvaXRlbXM_Zj1qd3MmIiwKICAgICAgICAgICAgInR5cGUiOiAiY
XBwbGljYXRpb25cL3N0YW5hZytqd3MiLAogICAgICAgICAgICAidGl0bGUiOiAiVGhpcyBkb2N1bWVudCBhcyB
kaWdpdGFsbHkgc2lnbmVkIFNUQU5BRyBpbiBKU09OIgogICAgICAgIH0sCiAgICAgICAgewogICAgICAgICAgI
CAicmVsIjogImFsdGVybmF0ZSIsCiAgICAgICAgICAgICJocmVmIjogImh0dHBzOlwvXC9vZ2Muc2VjdXJlLWR
pbWVuc2lvbnMuY29tXC9kY3NcL2NvbGxlY3Rpb25zXC9wb2lcL2l0ZW1zP2Y9andzJiIsCiAgICAgICAgICAgI
CJ0eXBlIjogImFwcGxpY2F0aW9uXC9zdGFuYWcrZ21sIiwKICAgICAgICAgICAgInRpdGxlIjogIlRoaXMgZG9
jdW1lbnQgYXMgU1RBTkFHICsgR01MIgogICAgICAgIH0sCiAgICAgICAgewogICAgICAgICAgICAicmVsIjogI
mFsdGVybmF0ZSIsCiAgICAgICAgICAgICJocmVmIjogImh0dHBzOlwvXC9vZ2Muc2VjdXJlLWRpbWVuc2lvbnM
uY29tXC9kY3NcL2NvbGxlY3Rpb25zXC9wb2lcL2l0ZW1zP2Y9andzJiIsCiAgICAgICAgICAgICJ0eXBlIjogI
mFwcGxpY2F0aW9uXC94bWwrZ21sO2NvbnRlbnQ9Z21sO3Byb2ZpbGU9XCJodHRwOlwvXC93d3cub3Blbmdpcy5
uZXRcL2RlZlwvcHJvZmlsZVwvb2djXC8yLjBcL2dtbC1zZjJcIjt2ZXJzaW9uPTMuMlwiIiwKICAgICAgICAgI
CAgInRpdGxlIjogIlRoaXMgZG9jdW1lbnQgYXMgR01MIgogICAgICAgIH0sCiAgICAgICAgewogICAgICAgICA
gICAicmVsIjogImFsdGVybmF0ZSIsCiAgICAgICAgICAgICJocmVmIjogImh0dHBzOlwvXC9vZ2Muc2VjdXJlL
WRpbWVuc2lvbnMuY29tXC9kY3NcL2NvbGxlY3Rpb25zXC9wb2lcL2l0ZW1zP2Y9andzJiIsCiAgICAgICAgICA
gICJ0eXBlIjogInRleHRcL2h0bWwiLAogICAgICAgICAgICAidGl0bGUiOiAiVGhpcyBkb2N1bWVudCBhcyBIV
E1MIgogICAgICAgIH0KICAgIF0sCiAgICAibnVtYmVyUmV0dXJuZWQiOiA2LAogICAgIm51bWJlck1hdGNoZWQ
iOiA2LAogICAgInRpbWVTdGFtcCI6ICIyMDIwLTEwLTEyVDA4OjM2OjQ4WiIsCiAgICAiZmVhdHVyZXMiOiBbC
iAgICAgICAgewogICAgICAgICAgICAidHlwZSI6ICJGZWF0dXJlIiwKICAgICAgICAgICAgImlkIjogInBvaS4
xIiwKICAgICAgICAgICAgImdlb21ldHJ5IjogewogICAgICAgICAgICAgICAgInR5cGUiOiAiUG9pbnQiLAogI
CAgICAgICAgICAgICAgImNvb3JkaW5hdGVzIjogWwogICAgICAgICAgICAgICAgICAgIC03NC4wMTA1LAogICA
gICAgICAgICAgICAgICAgIDQwLjcwNzYKICAgICAgICAgICAgICAgIF0KICAgICAgICAgICAgfSwKICAgICAgI
CAgICAgInByb3BlcnRpZXMiOiB7CiAgICAgICAgICAgICAgICAiTkFNRSI6ICJtdXNlYW0iLAogICAgICAgICA
gICAgICAgIlRIVU1CTkFJTCI6ICJwaWNzXC8yMjAzNzgyNy1UaS5qcGciLAogICAgICAgICAgICAgICAgIk1BS
U5QQUdFIjogInBpY3NcLzIyMDM3ODI3LUwuanBnIgogICAgICAgICAgICB9CiAgICAgICAgfSwKICAgICAgICB
7CiAgICAgICAgICAgICJ0eXBlIjogIkZlYXR1cmUiLAogICAgICAgICAgICAiaWQiOiAicG9pLjIiLAogICAgI
CAgICAgICAiZ2VvbWV0cnkiOiB7CiAgICAgICAgICAgICAgICAidHlwZSI6ICJQb2ludCIsCiAgICAgICAgICA
gICAgICAiY29vcmRpbmF0ZXMiOiBbCiAgICAgICAgICAgICAgICAgICAgLTc0LjAxMDgsCiAgICAgICAgICAgI
CAgICAgICAgNDAuNzA3NQogICAgICAgICAgICAgICAgXQogICAgICAgICAgICB9LAogICAgICAgICAgICAicHJ
vcGVydGllcyI6IHsKICAgICAgICAgICAgICAgICJOQU1FIjogInN0b2NrIiwKICAgICAgICAgICAgICAgICJUS
```

```
FVNQk5BSUwiOiAicGljc1wvMjIwMzc4MjktVGkuanBnIiwKICAgICAgICAgICAgICAgICJNQUlOUEFHRSI6ICJ
waWNzXC8yMjAzNzgyOS1MLmpwZyIKICAgICAgICAgICAgfQogICAgICAgIH0sCiAgICAgICAgewogICAgICAgI
CAgICAidHlwZSI6ICJGZWF0dXJlIiwKICAgICAgICAgICAgImlkIjogInBvaS4zIiwKICAgICAgICAgICAgImd
lb21ldHJ5IjogewogICAgICAgICAgICAgICAgInR5cGUiOiAiUG9pbnQiLAogICAgICAgICAgICAgICAgImNvb
3JkaW5hdGVzIjogWwogICAgICAgICAgICAgICAgICAgIC03NC4wMTA1LAogICAgICAgICAgICAgICAgICAgIDQ
wLjcwOTQKICAgICAgICAgICAgICAgIF0KICAgICAgICAgICAgfSwKICAgICAgICAgICAgInByb3BlcnRpZXMiO
iB7CiAgICAgICAgICAgICAgICAiTkFNRSI6ICJhcnQiLAogICAgICAgICAgICAgICAgIlRIVU1CTkFJTCI6ICJ
waWNzXC8yMjAzNzg1Ni1UaS5qcGciLAogICAgICAgICAgICAgICAgIk1BSU5QQUdFIjogInBpY3NcLzIyMDM3O
DU2LUwuanBnIgogICAgICAgICAgICB9CiAgICAgICAgfSwKICAgICAgICB7CiAgICAgICAgICAgICJ0eXBlIjo
gIkZlYXR1cmUiLAogICAgICAgICAgICAiaWQiOiAicG9pLjQiLAogICAgICAgICAgICAiZ2VvbWV0cnkiOiB7C
iAgICAgICAgICAgICAgICAidHlwZSI6ICJQb2ludCIsCiAgICAgICAgICAgICAgICAiY29vcmRpbmF0ZXMiOiB
bCiAgICAgICAgICAgICAgICAgICAgLTc0LjAwODYsCiAgICAgICAgICAgICAgICAgICAgNDAuNzExOQogICAgI
CAgICAgICAgXQogICAgICAgICAgICB9LAogICAgICAgICAgICAicHJvcGVydGllcyI6IHsKICAgICAgICAgICA
gICAgICJOQU1FIjogImxveCIsCiAgICAgICAgICAgICAgICAiVEhVTUJOQUlMIjogInBpY3NcLzIyMDM3O
Dg0LVRpLmpwZyIsCiAgICAgICAgICAgICAgICAiTUFJTlBBR0UiOiAicGljc1wvMjIwMzc4ODQtTC5qcGciCiA
gICAgICAgICAgIH0KICAgICAgICB9LAogICAgICAgIHsKICAgICAgICAgICAgInR5cGUiOiAiRmVhdHVyZSIsC
iAgICAgICAgICAgICJpZCI6ICJwb2kuNSIsCiAgICAgICAgICAgICJnZW9tZXRyeSI6IHsKICAgICAgICAgICA
gICAgICJ0eXBlIjogIlBvaW50IiwKICAgICAgICAgICAgICAgICJjb29yZGluYXRlcyI6IFsKICAgICAgICAgI
CAgICAgICAgICAtNzQuMDExOCwKICAgICAgICAgICAgICAgICAgICA0MC43MDg1CiAgICAgICAgICAgICAgICB
dCiAgICAgICAgICAgIH0sCiAgICAgICAgICAgICJwcm9wZXJ0aWVzIjogewogICAgICAgICAgICAgICAgIk5BT
UUiOiAiY2h1cmNoIiwKICAgICAgICAgICAgICAgICJUSFVNQk5BSUwiOiAicGljc1wvMjIwMzc4MzktVGkuanB
nIiwKICAgICAgICAgICAgICAgICJNQUlOUEFHRSI6ICJwaWNzXC8yMjAzNzgzOS1MLmpwZyIKICAgICAgICAgI
CAgfQogICAgICAgIH0sCiAgICAgICAgewogICAgICAgICAgICAidHlwZSI6ICJGZWF0dXJlIiwKICAgICAgICA
gICAgImlkIjogInBvaS42IiwKICAgICAgICAgICAgImdlb21ldHJ5IjogewogICAgICAgICAgICAgICAgInR5c
GUiOiAiUG9pbnQiLAogICAgICAgICAgICAgICAgImNvb3JkaW5hdGVzIjogWwogICAgICAgICAgICAgICAgICA
gIC03NC4wMDE1LAogICAgICAgICAgICAgICAgICAgIDQwLjcxOTkKICAgICAgICAgICAgICAgIF0KICAgICAgI
CAgICAgfSwKICAgICAgICAgICAgInByb3BlcnRpZXMiOiB7CiAgICAgICAgICAgICAgICAiTkFNRSI6ICJmaXJ
lIiwKICAgICAgICAgICAgICAgICJUSFVNQk5BSUwiOiAicGljc1wvMjg2NDA5ODQtVGkuanBnIiwKICAgICAgI
CAgICAgICAgICJNQUlOUEFHRSI6ICJwaWNzXC8yODY0MDk4NC1MLmpwZyIKICAgICAgICAgICAgfQogICAgICA
gIH0KICAgIF0KfQo.g_PI7SdkdZEwdDqn50p98FjPyf9wFdQm_v_W4qCGYB99EfzU_v7nAw0EyKmWsXNoLVe4F
-Jph8O4ULIjYmWo8YHLvjrlbnuNfUUe_BPxdHhS5mdDKDGpS_KBaXEiPsD06hQ_E0YpCDP0uNIT2nOKeAcB
-kKUBlGSi
-5xShfNUF0OgqHZG5BMhIrb7TMClFqOA2YzyKmygLPDdiSYLBnSo5kauvJtG7VOFXkiQK8eRJNe965rOahIUWw
4Pb1bo_RIh6WbpvY2ruThJJb_TiFsRhQm_epEXKu5fse2xgcG4Hq9X_3JNo9Xva9KI7IrWhD1nP9j2WBgj4-
kEC2OhXoUyg
```

Response Header decoded

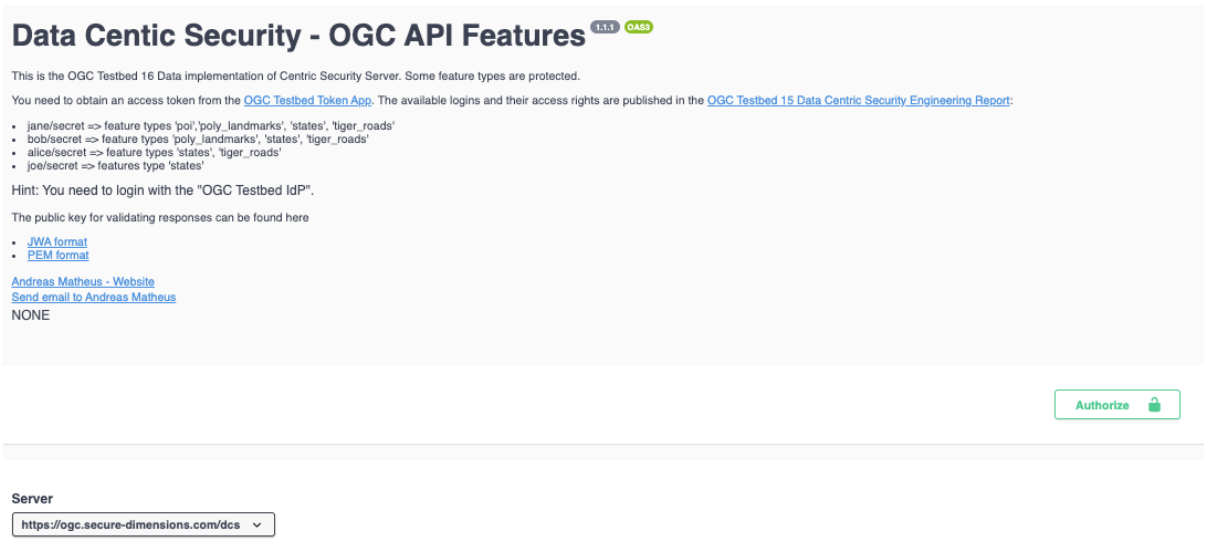
```
{
"alg": "RS256",
"kid": "Dr. No",
"cty": "application/geo+json"
}
```

Response Body decoded

```
{
    "type": "FeatureCollection",
    "links": [
        {
            "href": "https:\/\/ogc.secure-dimensions.com\/dcs\/collections\/poi
\/items?f=geo+jws&key_challenge=key_challenge_method=",
            "rel": "self",
            "type": "application\/geo+jws",
            "title": "This document"
        },
        {
            "rel": "alternate",
            "href": "https:\/\/ogc.secure-dimensions.com\/dcs\/collections\/poi
\/items?f=jws&",
            "type": "application\/geo+json",
            "title": "This document as GeoJSON"
        },
        {
            "rel": "alternate",
            "href": "https:\/\/ogc.secure-dimensions.com\/dcs\/collections\/poi
\/items?f=jws&",
            "type": "application\/stanag+json",
            "title": "This document as STANAG + JSON"
        },
        {
            "rel": "alternate",
            "href": "https:\/\/ogc.secure-dimensions.com\/dcs\/collections\/poi
\/items?f=jws&",
            "type": "application\/stanag+jws",
            "title": "This document as digitally signed STANAG in JSON"
        },
        {
            "rel": "alternate",
            "href": "https:\/\/ogc.secure-dimensions.com\/dcs\/collections\/poi
\/items?f=jws&",
            "type": "application\/stanag+gml",
            "title": "This document as STANAG + GML"
        },
        {
            "rel": "alternate",
            "href": "https:\/\/ogc.secure-dimensions.com\/dcs\/collections\/poi
\/items?f=jws&",
            "type": "application\/xml+gml;content=gml;profile=\"http:\/
\/www.opengis.net\/def\/profile\/ogc\/2.0\/gml-sf2\";version=3.2\"",
            "title": "This document as GML"
        },
        {
            "rel": "alternate",
            "href": "https:\/\/ogc.secure-dimensions.com\/dcs\/collections\/poi
\/items?f=jws&",
            "type": "text\/html",
            "title": "This document as HTML"
        }
    ],
    "numberReturned": 6,
    "numberMatched": 6,
    "timeStamp": "2020-10-12T08:36:48Z",
    "features": [
        {
            "type": "Feature",
            "id": "poi.1",
            "geometry": {
                "type": "Point",
                "coordinates": [
                    -74.0105,
                    40.7076
                ]
            },
            "properties": {
                "NAME": "museam",
                "THUMBNAIL": "pics\/22037827-Ti.jpg",
                "MAINPAGE": "pics\/22037827-L.jpg"
            }
        },
        {
            "type": "Feature",
            "id": "poi.2",
            "geometry": {
                "type": "Point",
                "coordinates": [
                    -74.0108,
                    40.7075
                ]
            },
            "properties": {
                "NAME": "stock",
                "THUMBNAIL": "pics\/22037829-Ti.jpg",
                "MAINPAGE": "pics\/22037829-L.jpg"
            }
        },
        {
            "type": "Feature",
            "id": "poi.3",
            "geometry": {
                "type": "Point",
                "coordinates": [
                    -74.0105,
                    40.7094
                ]
            },
            "properties": {
                "NAME": "art",
                "THUMBNAIL": "pics\/22037856-Ti.jpg",
                "MAINPAGE": "pics\/22037856-L.jpg"
            }
        },
        {
            "type": "Feature",
            "id": "poi.4",
            "geometry": {
                "type": "Point",
                "coordinates": [
                    -74.0086,
                    40.7119
                ]
            },
            "properties": {
                "NAME": "lox",
                "THUMBNAIL": "pics\/22037884-Ti.jpg",
                "MAINPAGE": "pics\/22037884-L.jpg"
            }
        },
        {
            "type": "Feature",
            "id": "poi.5",
            "geometry": {
                "type": "Point",
                "coordinates": [
                    -74.0118,
                    40.7085
                ]
            },
            "properties": {
                "NAME": "church",
                "THUMBNAIL": "pics\/22037839-Ti.jpg",
                "MAINPAGE": "pics\/22037839-L.jpg"
            }
        },
        {
            "type": "Feature",
            "id": "poi.6",
            "geometry": {
                "type": "Point",
                "coordinates": [
                    -74.0015,
                    40.7199
                ]
            },
            "properties": {
                "NAME": "fire",
                "THUMBNAIL": "pics\/28640984-Ti.jpg",
                "MAINPAGE": "pics\/28640984-L.jpg"
            }
        }
    ]
}
```

A.5.2. OpenAPI

The DCS Server’s API is described using OpenAPI v3: https://ogc.secure-dimensions.com/dcs/api/



Data Centic Security - OGC API Features 1.1.1 OAS3

This is the OGC Testbed 16 Data implementation of Centric Security Server. Some feature types are protected.

You need to obtain an access token from the OGC Testbed Token App. The available logins and their access rights are published in the OGC Testbed 15 Data Centric Security Engineering Report:

- jane/secret => feature types 'poi','poly_landmarks', 'states', 'tiger_roads'
- bob/secret => feature types 'poly_landmarks', 'states', 'tiger_roads'
- alice/secret => feature types 'states', 'tiger_roads'
- joe/secret => features type 'states'

Hint: You need to login with the "OGC Testbed IdP".

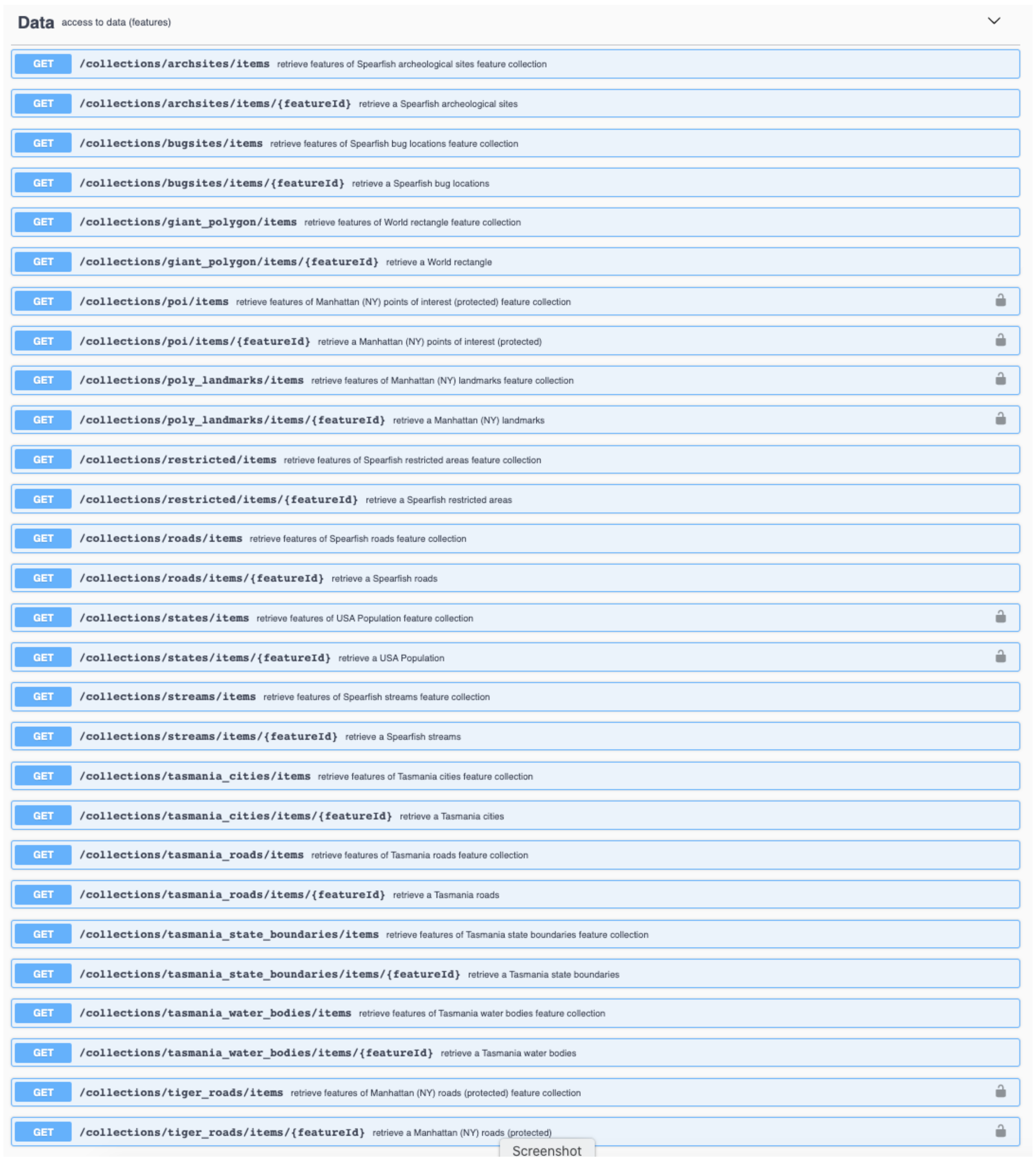
The public key for validating responses can be found here

- JWA format
- PEM format

Andreas Matheus - Website
Send email to Andreas Matheus
NONE

Figure 40. DCS Server described in OpenAPI

Some data endpoints are protected as illustrated below.



Data access to data (features)

GET /collections/archsites/items retrieve features of Spearfish archeological sites feature collection
GET /collections/archsites/items/{featureId} retrieve a Spearfish archeological sites
GET /collections/bugsites/items retrieve features of Spearfish bug locations feature collection
GET /collections/bugsites/items/{featureId} retrieve a Spearfish bug locations
GET /collections/giant_polygon/items retrieve features of World rectangle feature collection
GET /collections/giant_polygon/items/{featureId} retrieve a World rectangle
GET /collections/poi/items retrieve features of Manhattan (NY) points of interest (protected) feature collection
GET /collections/poi/items/{featureId} retrieve a Manhattan (NY) points of interest (protected)
GET /collections/poly_landmarks/items retrieve features of Manhattan (NY) landmarks feature collection
GET /collections/poly_landmarks/items/{featureId} retrieve a Manhattan (NY) landmarks
GET /collections/restricted/items retrieve features of Spearfish restricted areas feature collection
GET /collections/restricted/items/{featureId} retrieve a Spearfish restricted areas
GET /collections/roads/items retrieve features of Spearfish roads feature collection
GET /collections/roads/items/{featureId} retrieve a Spearfish roads
GET /collections/states/items retrieve features of USA Population feature collection
GET /collections/states/items/{featureId} retrieve a USA Population
GET /collections/streams/items retrieve features of Spearfish streams feature collection
GET /collections/streams/items/{featureId} retrieve a Spearfish streams
GET /collections/tasmania_cities/items retrieve features of Tasmania cities feature collection
GET /collections/tasmania_cities/items/{featureId} retrieve a Tasmania cities
GET /collections/tasmania_roads/items retrieve features of Tasmania roads feature collection
GET /collections/tasmania_roads/items/{featureId} retrieve a Tasmania roads
GET /collections/tasmania_state_boundaries/items retrieve features of Tasmania state boundaries feature collection
GET /collections/tasmania_state_boundaries/items/{featureId} retrieve a Tasmania state boundaries
GET /collections/tasmania_water_bodies/items retrieve features of Tasmania water bodies feature collection
GET /collections/tasmania_water_bodies/items/{featureId} retrieve a Tasmania water bodies
GET /collections/tiger_roads/items retrieve features of Manhattan (NY) roads (protected) feature collection
GET /collections/tiger_roads/items/{featureId} retrieve a Manhattan (NY) roads (protected)

Figure 41. DCS Server Data endpoints described in OpenAPI

Fetching features via the OGC API - Features endpoint for a protected feature type requires to provide a valid access token (via the lock) and to submit two additional parameters, not common to OGC API:

- `key_challenge` is the (optionally hashed) one-time secret that allows the client in later communication with the Key Management Server to prove ownership of the cipher keys that will be created and registered by the DCS for the request.
- `key_challenge_method` is the hashing method to be used to verify the key_challenge in later communication with the Key Management Server. When using `plain`, the `key_challenge` and `key_verifier` values are identical.

The following is an example request leveraging Curl:

```
curl -X GET "https://ogc.secure-
dimensions.com/dcs/collections/poi/items?key_challenge=secret&key_challenge_method=pla
in&limit=10&crs=http%3A%2F%2Fwww.opengis.net%2Fdef%2Fcrs%2FOGC%2F1.3%2FCRS84&bbox-
crs=http%3A%2F%2Fwww.opengis.net%2Fdef%2Fcrs%2FOGC%2F1.3%2FCRS84" -H "accept:
application/stanag+json" -H "Authorization: Bearer
adef6389a31d65cadbc024d8208777648c4cd6ed"
```

A.6. Conclusions

The DCS Server and the Key Management Server implemented for Testbed-16 demonstrate the ability to encrypt geospatial data and metadata separately as denoted in NATO STANAG 4778. The implementation illustrates the use of OGC API Features returning STANAG 4778 and 4774 encrypted data in XML and JSON encoding. Data is encrypted from a clear data source (Geoserver default data) with different cipher key strength, depending on the fictitious classification label of the feature type. The cipher keys created by the DCS Server can be obtained with the client application via the key identifier. Access control at the Key Management Server ensures that only a legitimate user / client combination can fetch the cipher key to decrypt the data or to delete (inactivate) a cipher key.

The Key Management Server supports encrypted responses to ensure data centric security at the highest level. This requires the use of public keys that can be registered by a user, client or the DCS server.

To ensure that all components (client, DCS server and Key Management Server) are able to share a common security context, Bearer access tokens are used from a common Authorization Server as defined in RFC 6750. The use of OAuth2 and OpenID Connect interfaces ensure an easy-to-use API as many SDKs exist in various programming languages.

Appendix B: Engineering Aspects for D146

This annex introduces the engineering aspects of the DCS component D146 (Key Management Server) implemented for Testbed-16 by Helyx Secure Information Systems Limited. In particular it describes the architecture of the Key Management Server (KMS) and its interactions with clients.

B.1. Overview

Helyx decided to take an existing standard for key management services as a starting point for the implementation of a KMS for DCS. Given the likely scenarios where DCS systems may be implemented, it was felt that basing the KMS on an implementation of the OASIS Key Management Interoperability Protocol (KMIP) Specification 2.x was of particular interest, given the strong protection for keys afforded by KMIP-compliant Servers. KMIP defines standard interfaces for both clients and servers and categorizes them in terms of basic and advanced cryptographic clients and servers. As this is a research task, a hardware KMIP server was not available to the team, so a software implementation in the form of PyKMIP Server was selected as the back-end key management server with a database.

As well as providing a Server module, PyKMIP also provides a Client module that simplifies the interactions with a KMIP Server. Client and Server communications are protected using mutually authenticated TLS. This client is used as part of a Python Flask application that uses the Connexion framework that handles HTTP requests based on OpenAPI Specification of the API described in YAML format. Connexion maps the endpoints to our underlying Python functions; this is preferable to other tools that generate the specification based on the underlying Python code. Connexion also validates requests and endpoint parameters automatically, based on the specification, and supports API versioning as well as providing a Web Swagger Console UI.

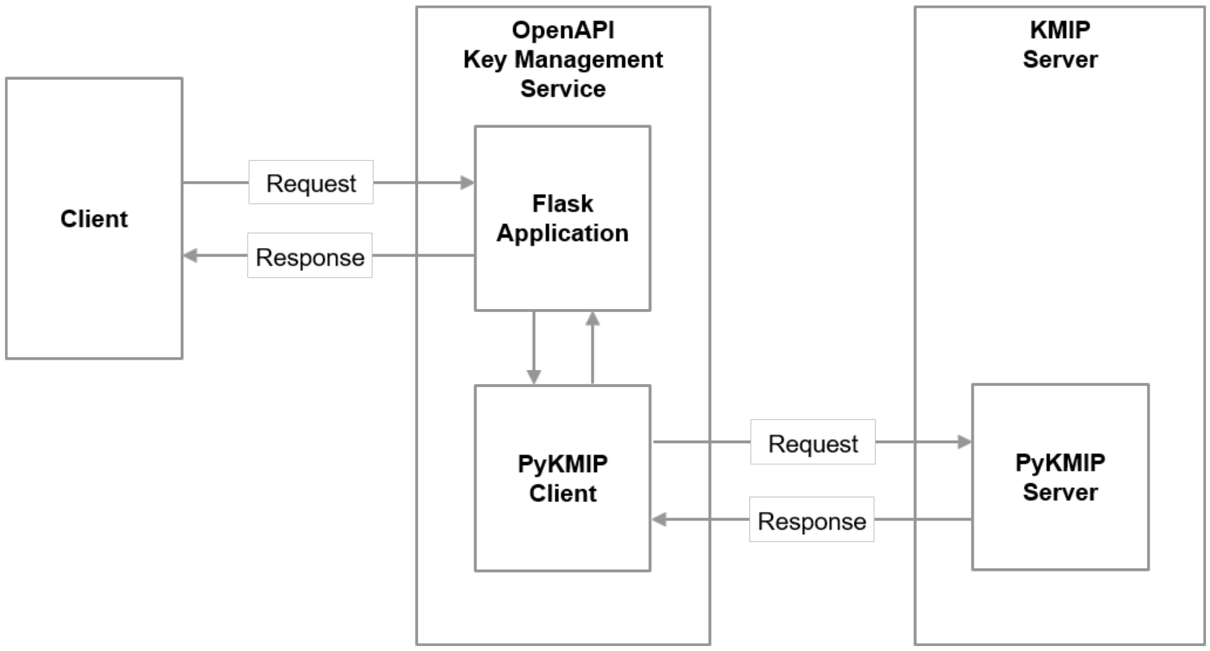


Figure 42. KMS architecture overview

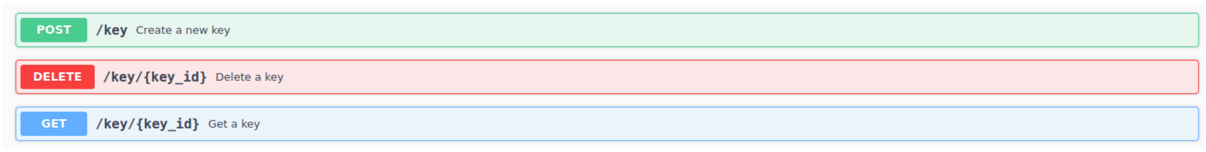
B.2. Key Management Server (KMS)

The implementation of the interface and functionality for the KMS is based on the requirements derived from the mobile / server use case. A number of different interface categories exist:

- Managing symmetric keys that can be used to encrypt/decrypt data and metadata
- Encrypt and MAC data and decrypt encrypted data
- Managing RSA key pairs that can be used to sign encrypted data.

B.2.1. Managing Symmetric Keys

The current implementation allows a client to create, read and delete a symmetric key.



POST /key Create a new key

DELETE /key/{key_id} Delete a key

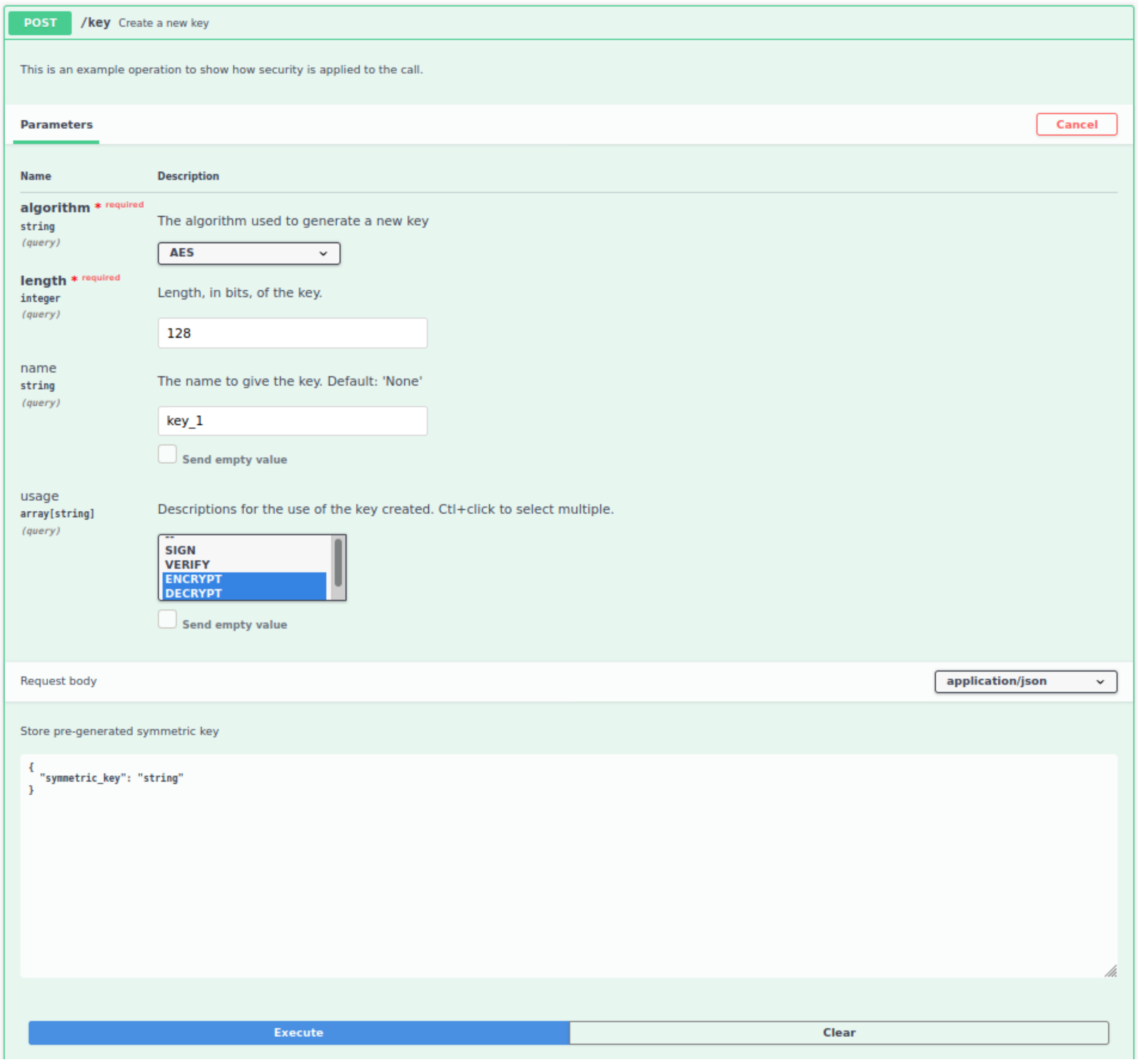
GET /key/{key_id} Get a key

Figure 43. Symmetric keys overview

B.2.1.1. OpenAPI Implementation

To create a symmetric key, the KMS requires an algorithm and an appropriate length for that key. The KMS also allows giving a name (an identifier) to the key being created and an intended usage for that key. If the user intends to use the other functionality of the KMS then a usage must be supplied in line with the expected use of the key.

The return value of this endpoint is a JWK containing only the key ID.



POST /key Create a new key

This is an example operation to show how security is applied to the call.

Parameters

Name	Description
algorithm * required string *(query)*	The algorithm used to generate a new key AES
length * required integer *(query)*	Length, in bits, of the key. 128
name string *(query)*	The name to give the key. Default: 'None' key_1 Send empty value
usage array[string] *(query)*	Descriptions for the use of the key created. Ctl+click to select multiple. -- SIGN VERIFY ENCRYPT DECRYPT Send empty value

Request body application/json

Store pre-generated symmetric key

```
{
  "symmetric_key": "string"
}
```

Figure 44. Create symmetric key

Alternatively send a cURL request similar to the following:

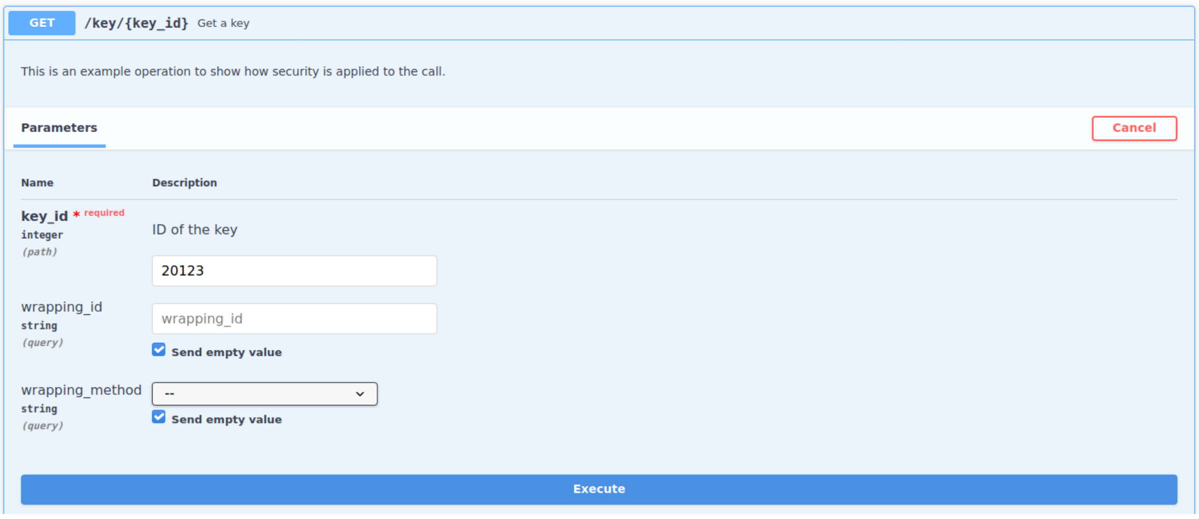
```
curl -X POST "https://kms.example.ogc.org/Helyx-
SIS/KMS/1.0.0/key?algorithm=AES&length=128&name=key_1&usage=ENCRYPT,DECRYPT" \
  -H  "accept: application/json" \
  -H  "Content-Type: application/json" \
  -d "{\"symmetric_key\":\"string\"}"
```

To elicit a response similar to the following:

```
{
  "kid": "20123"
}
```

B.2.1.2. Get key

The symmetric key’s ID is required in order to get the key as a JWK. Within that JWK there is a use parameter in line with RFC 7517.



GET /key/{key_id} Get a key

This is an example operation to show how security is applied to the call.

key_id * required integer (path) — ID of the key: 20123

wrapping_id string (query) — wrapping_id; Send empty value ✓

wrapping_method string (query) — --; Send empty value ✓

Figure 45. Get symmetric key

Alternatively send a cURL request similar to the following:

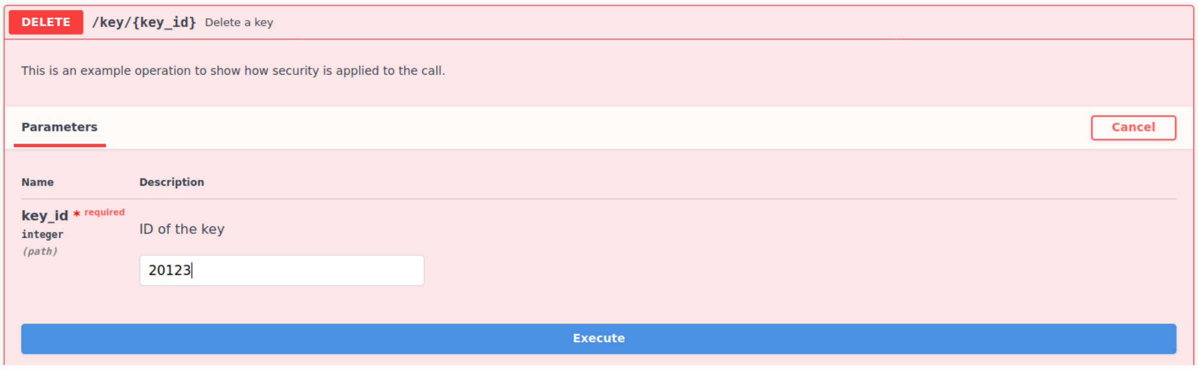
```
curl -X GET "https://kms.example.ogc.org/Helyx-
SIS/KMS/1.0.0/key/20123?wrapping_id=&wrapping_method=" \
  -H  "accept: application/json"
```

To elicit a response similar to the following:

```
{
  "k": "zdKgK3iVW42X-OrNbJ98PQ",
  "kid": "20123",
  "kty": "oct",
  "use": "enc"
}
```

B.2.1.3. Delete key

This only requires the key ID of the symmetric key to be deleted and will return an empty 204 response if successful.



DELETE /key/{key_id} Delete a key

This is an example operation to show how security is applied to the call.

key_id * required integer (path) — ID of the key: 20123

Figure 46. Delete symmetric key

Alternatively send a cURL request similar to the following:

```
curl -X DELETE "https://kms.example.ogc.org/Helyx-SIS/KMS/1.0.0/key/20123" \
  -H  "accept: */*"
```

B.2.2. Managing RSA Key Pairs

The current implementation allows a client to create an RSA key pair, as well as to read and delete their public and private key components.

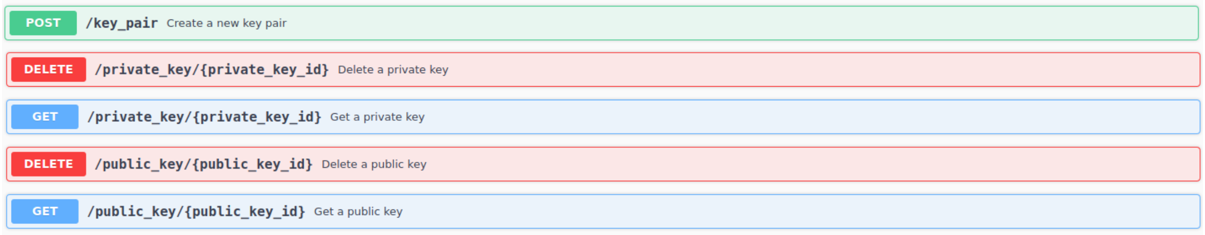
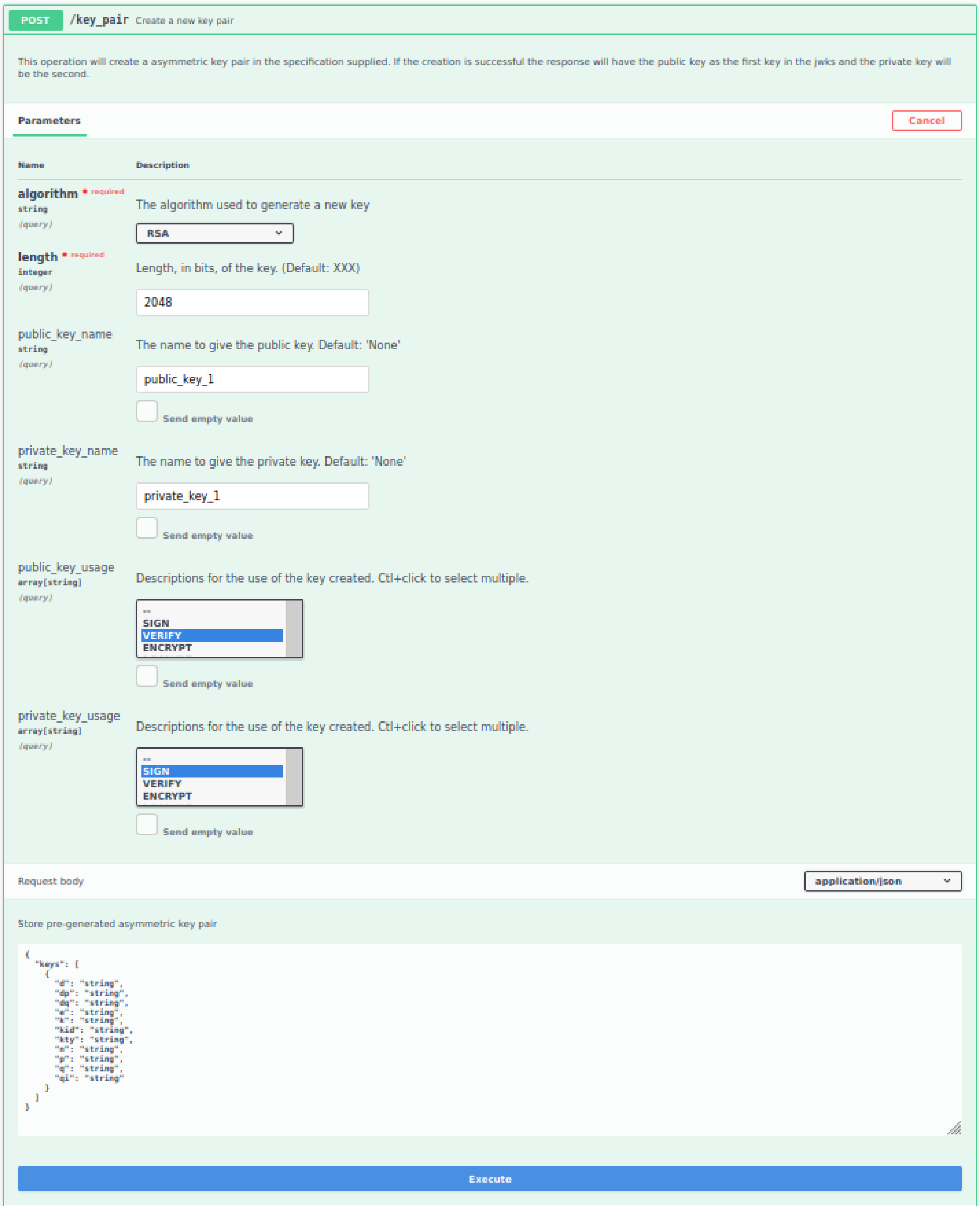

POST /key_pair Create a new key pair
DELETE /private_key/{private_key_id} Delete a private key
GET /private_key/{private_key_id} Get a private key
DELETE /public_key/{public_key_id} Delete a public key
GET /public_key/{public_key_id} Get a public key

Figure 47. Asymmetric keys overview

B.2.2.1. OpenAPI Implementation

Creating a key pair requires the algorithm and an appropriate length of the key. The server also allows names to be specified for each key and an intended usage for each key. If the user wants to use other functionality of the KMS, then the usage parameter should be supplied for the intended use of the key.

The KMS will return JWKS containing both keys created. The first key will be public key and the second key will be the private key.


POST /key_pair Create a new key pair

This operation will create a asymmetric key pair in the specification supplied. If the creation is successful the response will have the public key as the first key in the jwks and the private key will be the second.

Parameters — Cancel

Name	Description
algorithm * required string (query)	The algorithm used to generate a new key RSA
length * required integer (query)	Length, in bits, of the key. (Default: XXX) 2048
public_key_name string (query)	The name to give the public key. Default: 'None' public_key_1 Send empty value
private_key_name string (query)	The name to give the private key. Default: 'None' private_key_1 Send empty value
public_key_usage array[string] (query)	Descriptions for the use of the key created. Ctl+click to select multiple. -- SIGN [✓ VERIFY] ENCRYPT Send empty value
private_key_usage array[string] (query)	Descriptions for the use of the key created. Ctl+click to select multiple. -- [✓ SIGN] VERIFY ENCRYPT Send empty value

Request body — application/json

Store pre-generated asymmetric key pair

```
{
  "keys": [
    {
      "d": "string",
      "dp": "string",
      "dq": "string",
      "e": "string",
      "k": "string",
      "kid": "string",
      "kty": "string",
      "n": "string",
      "p": "string",
      "q": "string",
      "qi": "string"
    }
  ]
}
```

Execute

Figure 48. Create asymmetric key

Alternatively send a cURL request similar to the following:

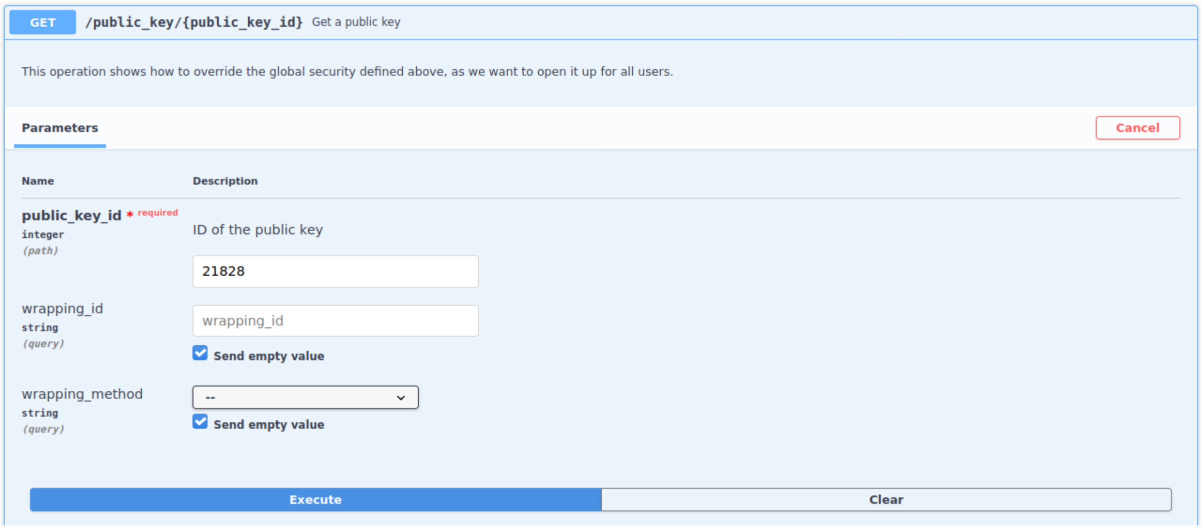
```
curl -X POST "https://kms.example.ogc.org/Helyx-
SIS/KMS/1.0.0/key_pair?algorithm=RSA&length=2048&public_key_name=public_key_1&private_
key_name=private_key_1&public_key_usage=VERIFY&private_key_usage=SIGN" \
  -H  "accept: application/json" \
  -H  "Content-Type: application/json" \
  -d
"{\"keys\":[{\"d\":\"string\",\"dp\":\"string\",\"dq\":\"string\",\"e\":\"string\",\"k
\":\"string\",\"kid\":\"string\",\"kty\":\"string\",\"n\":\"string\",\"p\":\"string\",
\"q\":\"string\",\"qi\":\"string\"}]}"
```

To elicit a response similar to the following:

```
{
  "keys": [
    {
      "kid": "21828"
    },
    {
      "kid": "21829"
    }
  ]
}
```

B.2.2.2. Get public key

The public key's ID is required in order to get the key as a JWK. Within that JWK there is a use parameter in line with RFC 7517.



/public_key/{public_key_id} Get a public key

This operation shows how to override the global security defined above, as we want to open it up for all users.

Name	Description
public_key_id * required integer (path)	ID of the public key: 21828
wrapping_id string (query)	wrapping_id; ☑ Send empty value
wrapping_method string (query)	--; ☑ Send empty value

Figure 49. Get public key

Alternatively send a cURL request similar to the following:

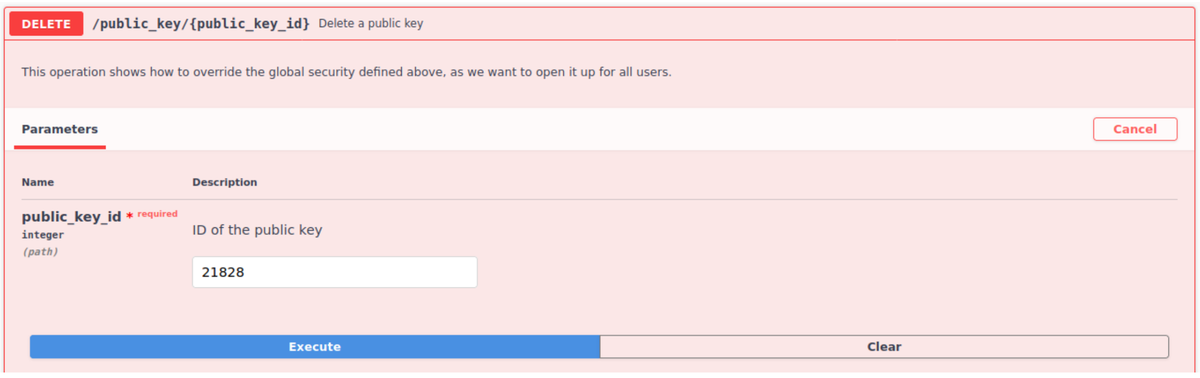
```
curl -X GET "https://kms.example.ogc.org/Helyx-
SIS/KMS/1.0.0/public_key/21828?wrapping_id=&wrapping_method=" \
  -H  "accept: application/json"
```

To elicit a response similar to the following:

```
{
  "e": "AQAB",
  "kid": "21828",
  "kty": "RSA",
  "n": "t6dZ02wgn19uZdgQT9qH5a3k_Bgch-4kmwtklYg8GYGEf29o2O0CQ-
oBR7OfgpyOasqpsEi3FIdEJ1rioVfepThtBnRy-mqiziJi6mrajfIxGeNSdGg-
q2IxMwH23Vh8icSjlZt90JvVP6GLKTNGjOkidZ6k5vbdExa-
n588y9hmHs6rpb1XyUbOsd7Uit_KHXkHMo_3DV52i0OyUw0cuIIuGWJeGE12CvRYBvoWbLgcI81ViduUz_PZom
WZ40D98J1-
BogZfxMEc4fY5li9B4Tx3W0CT54RnvpXv3RUW5aazVC6VtXJNE_EXGroeZKGa3m4yx2uwvgnwC65TivCxg",
  "use": "sig"
}
```

B.2.2.3. Delete public key

This only requires the key ID of the public key to be deleted and will return an empty 204 response if successful.



This operation shows how to override the global security defined above, as we want to open it up for all users.

public_key_id * required integer (path) — ID of the public key: 21828

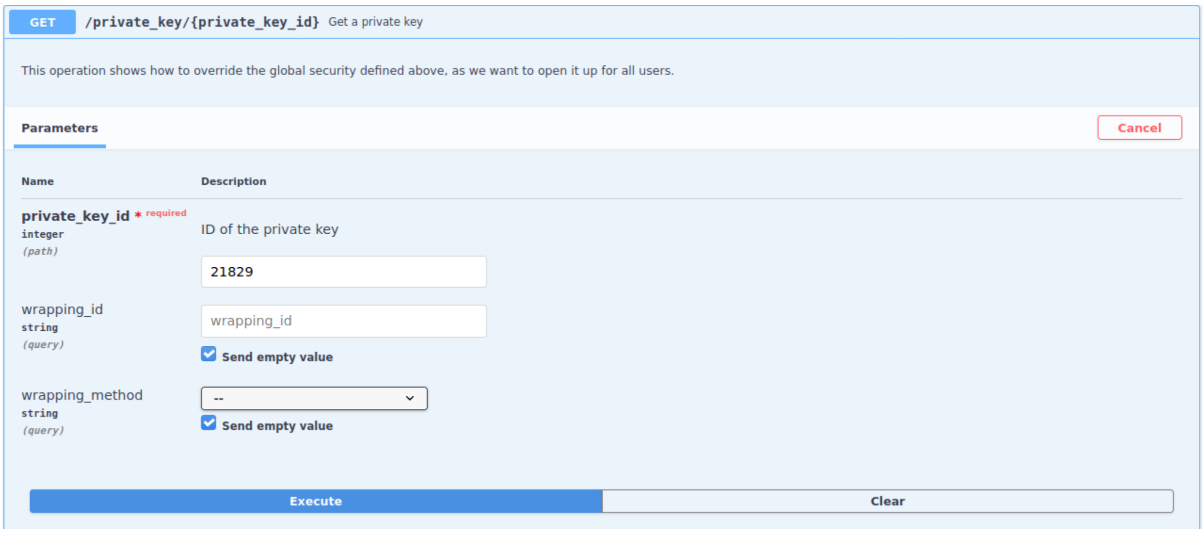
Figure 50. Delete public key

Alternatively send a cURL request similar to the following:

```
curl -X DELETE "https://kms.example.ogc.org/Helyx-SIS/KMS/1.0.0/public_key/21828" \
  -H  "accept: */*"
```

B.2.2.4. Get private key

The private key’s ID is required in order to get the key as a JWK. Within that JWK there is a use parameter in line with RFC 7517.


GET /private_key/{private_key_id} Get a private key

This operation shows how to override the global security defined above, as we want to open it up for all users.

Name	Description
private_key_id * required integer (path)	ID of the private key 21829
wrapping_id string (query)	wrapping_id ☑ Send empty value
wrapping_method string (query)	-- ☑ Send empty value

Figure 51. Get private key

Alternatively send a cURL request similar to the following:

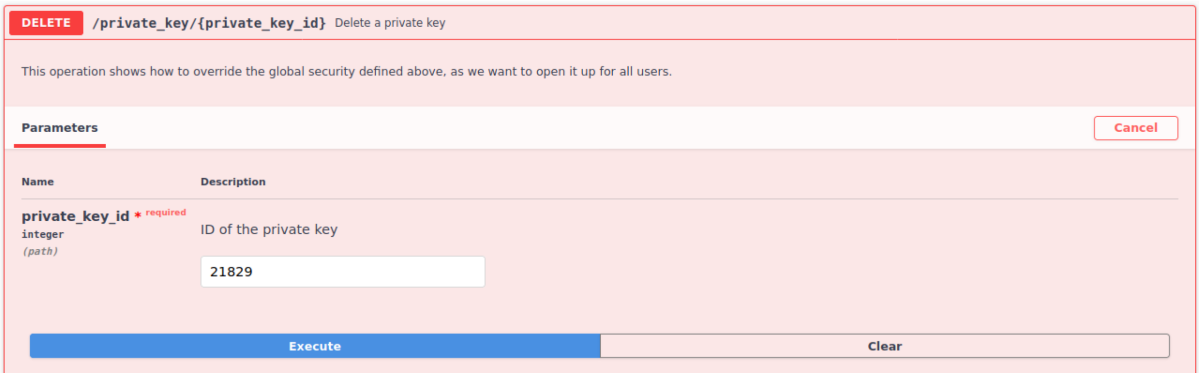
```
curl -X GET "https://kms.example.ogc.org/Helyx-
SIS/KMS/1.0.0/private_key/21829?wrapping_id=&wrapping_method=" \
  -H  "accept: application/json"
```

To elicit a response similar to the following:

```
{
  "d": "wUVuMlFqG5F8aJllswpRHy4JUtPrKDD8vU64zswZtBkrkEHC_hCFz2raG0d-
XvilaDc3L6_xn6_ynJGio8Z4Z0L_9WKRMPBABkA4V1Wvk_qtoTHMlYxxP7zSqw2lWvob0EP0l19_ak0dDtYkgN
g5EVQEnzYhBu7zltyi3DpYD2gt_SnJZa-EyuTUvFiVtvj9Qhjigz_0Q03O3xgXcf-
Io_sx6mpXRD2W9BuMW3K7ml6-7b0RO2Zt55GKf30b3OY0poKO4ZK4yuClAjPL0vshFUUXxHKLNXeGhE5feXl-
lFjewtO3Lky3q4ZToOfvrALQKHCvoaiBzSooz7W-Xt_8GQ",
  "dp":
"4VZ66NrGo8pcFVQbfavArGkwXxxy99rA1MQ810895I79pezQFsyNuWFwu33oHpSb3950YHdDEfbvj5MOqsvNk
UF4irwVVkh87D8q8rPLSfI9mj3F_gOwAlYv4ulHye6Upj_tLmRQ67shYC1vLqOSfMZ2Rx-
PCO5mqROFE21YHAU",
  "dq":
"qyFcBR4KmbGMtx6R6cwq1WvbCm7gYAzY7n9W37rMSbZr6mD7stb6whQBLEGjVBA9lYjXjKkkqcN8XQtZMaoYa
bcdIC2NyJbkTqmPQUyiSVeDtmYH1FNo35um42zU-XusLksZGfJv3iT-
OaBTIWveBt7KzPsLrCXy_YZplSSuj3U",
  "e": "AQAB",
  "kid": "21829",
  "kty": "RSA",
  "n": "t6dZ02wgn19uZdgQT9qH5a3k_Bgch-4kmwtklYg8GYGEf29o2O0CQ-
oBR7OfgpyOasqpsEi3FIdEJ1rioVfepThtBnRy-mqiziJi6mrajfIxGeNSdGg-
q2IxMwH23Vh8icSjlZt90JvVP6GLKTNGjOkidZ6k5vbdExa-
n588y9hmHs6rob1XyUbOsd7Uit_KHXkHMo_3DV52i0OyUw0cuIIuGWJeGE12CvRYBvoWbLgcI81ViduUz_PZom
WZ40D98J1-
BogZfxMEc4fY5li9B4Tx3W0CT54RnvpXv3RUW5aazVC6VtXJNE_EXGroeZKGa3m4yx2uwvgnwC65TivCxg",
  "p": "Vy6x5qgcuJEomw8Ivy1dC73sgZ3psvCapUjIgjyxD9iRvJc6Y7F4-eCwqND_U6w1g7t4gqfT4-
I0qchYYZ7zHrL3Q5Fq_h1iHOAsHklifFTNMW7pMJJKwZYkp0XuSVdEXjgkllWVL7nx79oSFSL6zdAOR4gIWBL7
jJz6XAcwz_o",
  "q": "obUY4JCTXXKlNw6XMZ2YgB8mIe3I7NL4Ke2I825T_gy6J-
7tp5aDt3ChxJ7Md6iUN5zH6PVGhSqVLAxwKTUbGj2A6yKFaFQGxAQoEDLZJw_Cc-uL8l-
s1cyTxaXPqLIq9ptC-8822v85QswumhDSl-JKaZXhY-M3VRcIgQY238o",
  "qi":
"lRpUf6mPGGf8GYh1clxqu8cfxgRzFFARIA_tWbWzDLFqluykkdyqFcpnRvAapt1b6o6ofbIddR1wQrrT6IHlu
s8gv5-O_6ZhlE2-hi2acj2Yn7dOvLRdmBaL-L7qzdYfzILDg8DCHrHfcZPQ76fIsKyFASx-
Smllthkam1HyFio",
  "use": "sig"
}
```

B.2.2.5. Delete private key

This only requires the key ID of the private key to be deleted and will return an empty 204 response if successful.


DELETE /private_key/{private_key_id} Delete a private key

This operation shows how to override the global security defined above, as we want to open it up for all users.

Name	Description
private_key_id * required integer (path)	ID of the private key 21829

Figure 52. Delete private key

Alternatively send a cURL request similar to the following:

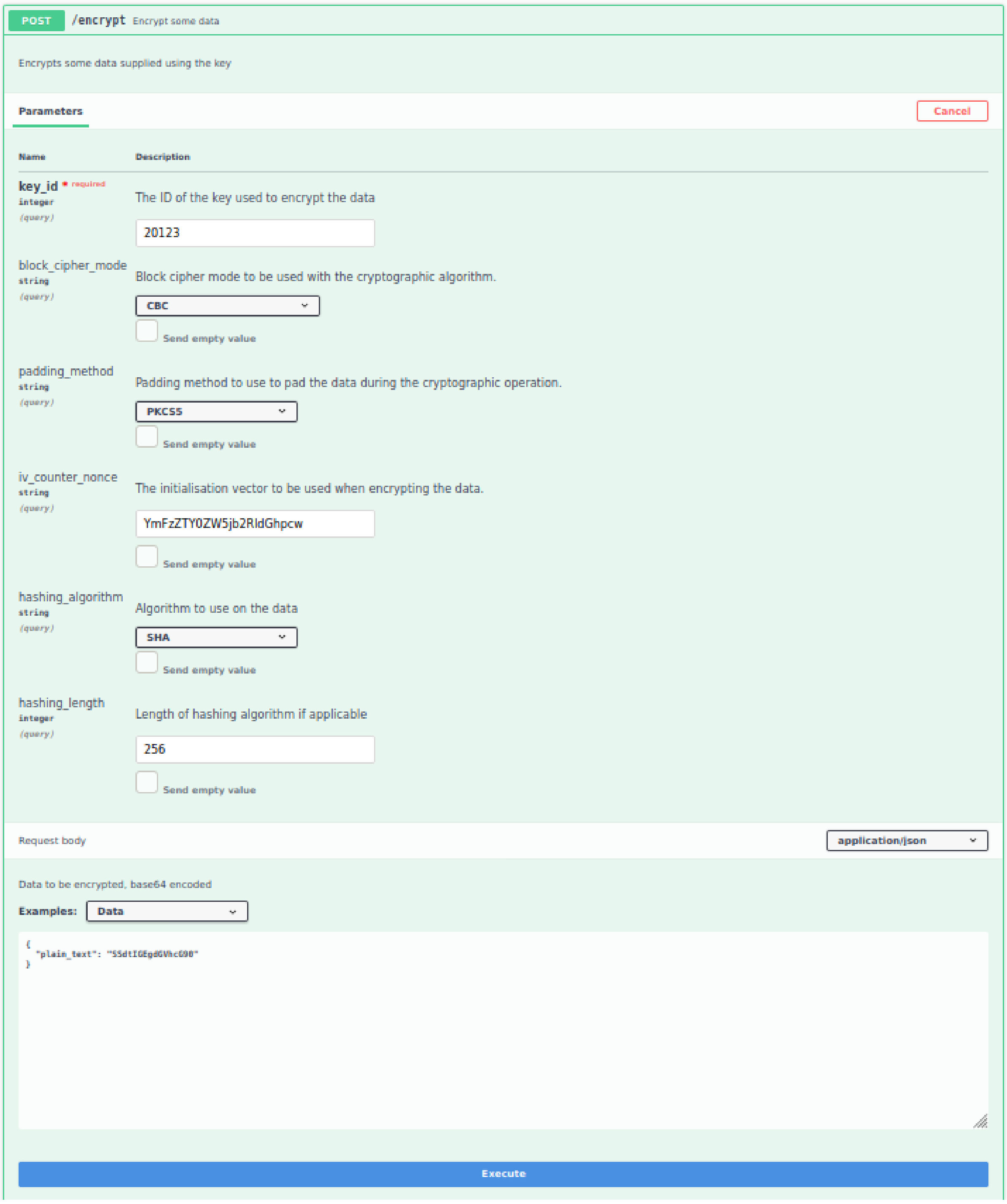
```
curl -X DELETE "https://kms.example.ogc.org/Helyx-SIS/KMS/1.0.0/private_key/21829" \
  -H  "accept: */*"
```

B.2.3. Other Functionality

B.2.3.1. Encrypt

The underlying PyKMIP server currently only supports symmetric key encryption. This symmetric key must have the encrypt usage mask associated to it.

The KMS allows for many different methods of encrypting data. These different parameters include the block cipher mode, the padding method and the hashing algorithm. This allows for the user to input their own initialization vector (IV) as long as it’s base64 encoded. If an IV is not supplied, the server will return one automatically generated, also base64 encoded.


POST /encrypt Encrypt some data

Encrypts some data supplied using the key

Parameters

Cancel

Name	Description
key_id * required integer (query)	The ID of the key used to encrypt the data 20123
block_cipher_mode string (query)	Block cipher mode to be used with the cryptographic algorithm. CBC Send empty value
padding_method string (query)	Padding method to use to pad the data during the cryptographic operation. PKCS5 Send empty value
iv_counter_nonce string (query)	The initialisation vector to be used when encrypting the data. YmFzZTY0ZW5jb2RldGhpcw Send empty value
hashing_algorithm string (query)	Algorithm to use on the data SHA Send empty value
hashing_length integer (query)	Length of hashing algorithm if applicable 256 Send empty value

Request body application/json

Data to be encrypted, base64 encoded

Examples: Data

```
{
  "plain_text": "SSdtIGEgdGVhcG90"
}
```

Execute

Figure 53. Encrypt data

Alternatively send a cURL request similar to the following:

```
curl -X POST "https://kms.example.ogc.org/Helyx-
SIS/KMS/1.0.0/encrypt?key_id=20123&block_cipher_mode=CBC&padding_method=PKCS5&iv_count
er_nonce=YmFzZTY0ZW5jb2RldGhpcw&hashing_algorithm=SHA&hashing_length=256" \
  -H  "accept: application/json" \
  -H  "Content-Type: application/json" \
  -d "{\"plain_text\":\"SSdtIGEgdGVhcG90\"}"
```

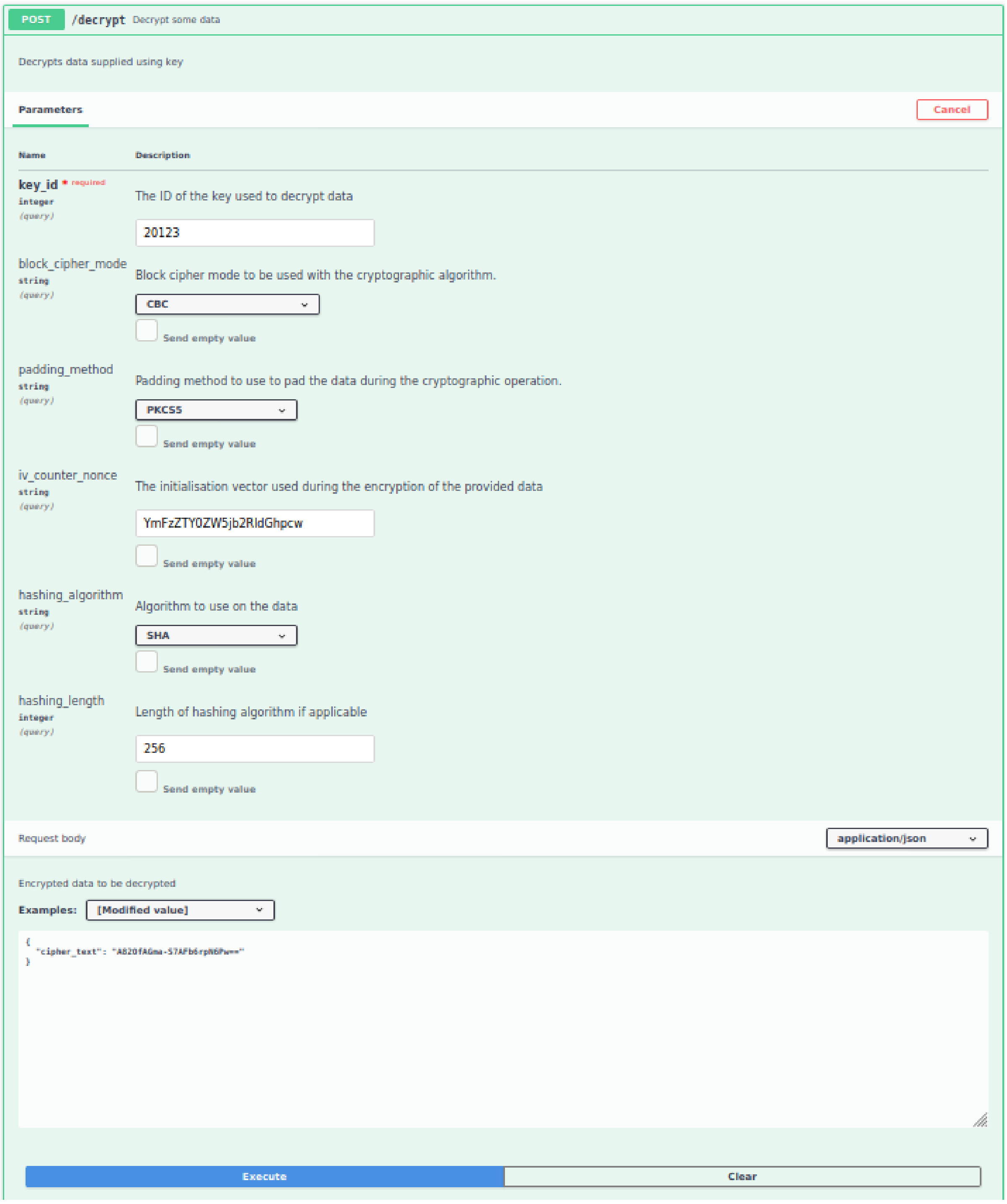
To elicit a response similar to the following:

```
{
  "cipher_text": "A82OfAGma-S7AFb6rpN6Pw"
}
```

B.2.3.2. Decrypt

The symmetric key that encrypted the data should also have the decrypt usage mask in order to allow for the decryption of the data.

If the same parameters are sent to the KMS as was inputted when encrypting the original piece of data, then the KMS will decrypt the cipher text to the original data. This also includes the IV, whether it was supplied by the user or created by the PyKMIP server.


POST /decrypt Decrypt some data

Decrypts data supplied using key

Parameters

Cancel

Name	Description
key_id * required integer (query)	The ID of the key used to decrypt data 20123
block_cipher_mode string (query)	Block cipher mode to be used with the cryptographic algorithm. CBC Send empty value
padding_method string (query)	Padding method to use to pad the data during the cryptographic operation. PKCS5 Send empty value
iv_counter_nonce string (query)	The initialisation vector used during the encryption of the provided data YmFzZTY0ZW5jb2RldGhpcw Send empty value
hashing_algorithm string (query)	Algorithm to use on the data SHA Send empty value
hashing_length integer (query)	Length of hashing algorithm if applicable 256 Send empty value

Request body application/json

Encrypted data to be decrypted

Examples: [Modified value]

```
{
  "cipher_text": "A820fAGma-S7AFb6rpN6Pw=="
}
```

Execute Clear

Figure 54. Decrypt data

Alternatively send a cURL request similar to the following:

```
curl -X POST "https://kms.example.ogc.org/Helyx-
SIS/KMS/1.0.0/decrypt?key_id=20123&block_cipher_mode=CBC&padding_method=PKCS5&iv_count
er_nonce=YmFzZTY0ZW5jb2RldGhpcw&hashing_algorithm=SHA&hashing_length=256" \
  -H  "accept: application/json" \
  -H  "Content-Type: application/json" \
  -d "{\"cipher_text\":\"A820fAGma-S7AFb6rpN6Pw==\"}"
```

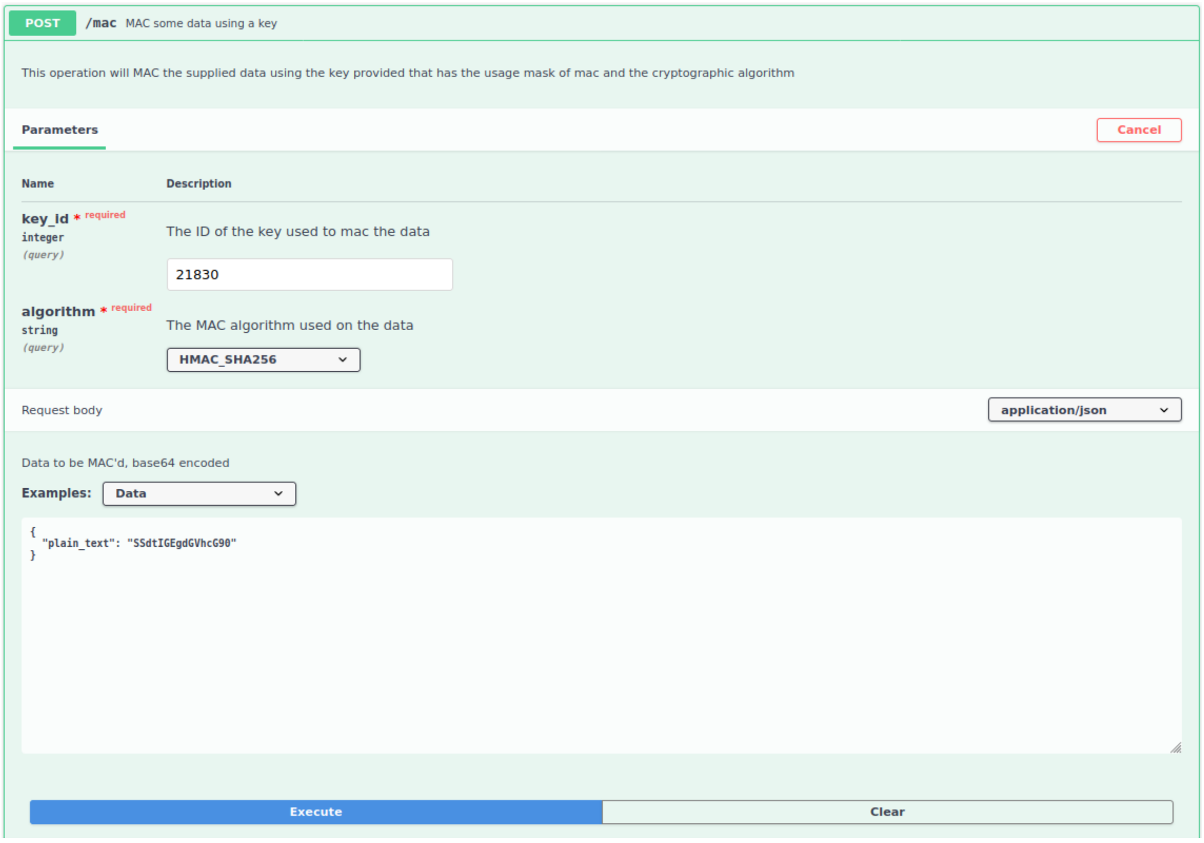
To elicit a response similar to the following:

```
{
  "plain_text": "SSdtIGEgdGVhcG90"
}
```

B.2.3.3. MAC

MACing data requires the key that is to be used have the MAC generate and MAC verify usage masks associated to them.

This function allows several different versions of the HMAC algorithm to be used on the data.



This operation will MAC the supplied data using the key provided that has the usage mask of mac and the cryptographic algorithm

Name	Description
key_id * required integer (query)	The ID of the key used to mac the data 21830
algorithm * required string (query)	The MAC algorithm used on the data HMAC_SHA256

Request body application/json

Data to be MAC'd, base64 encoded

Examples: Data

```
{
  "plain_text": "SSdtIGEgdGVhcG90"
}
```

Figure 55. MAC data

Alternatively send a cURL request similar to the following:

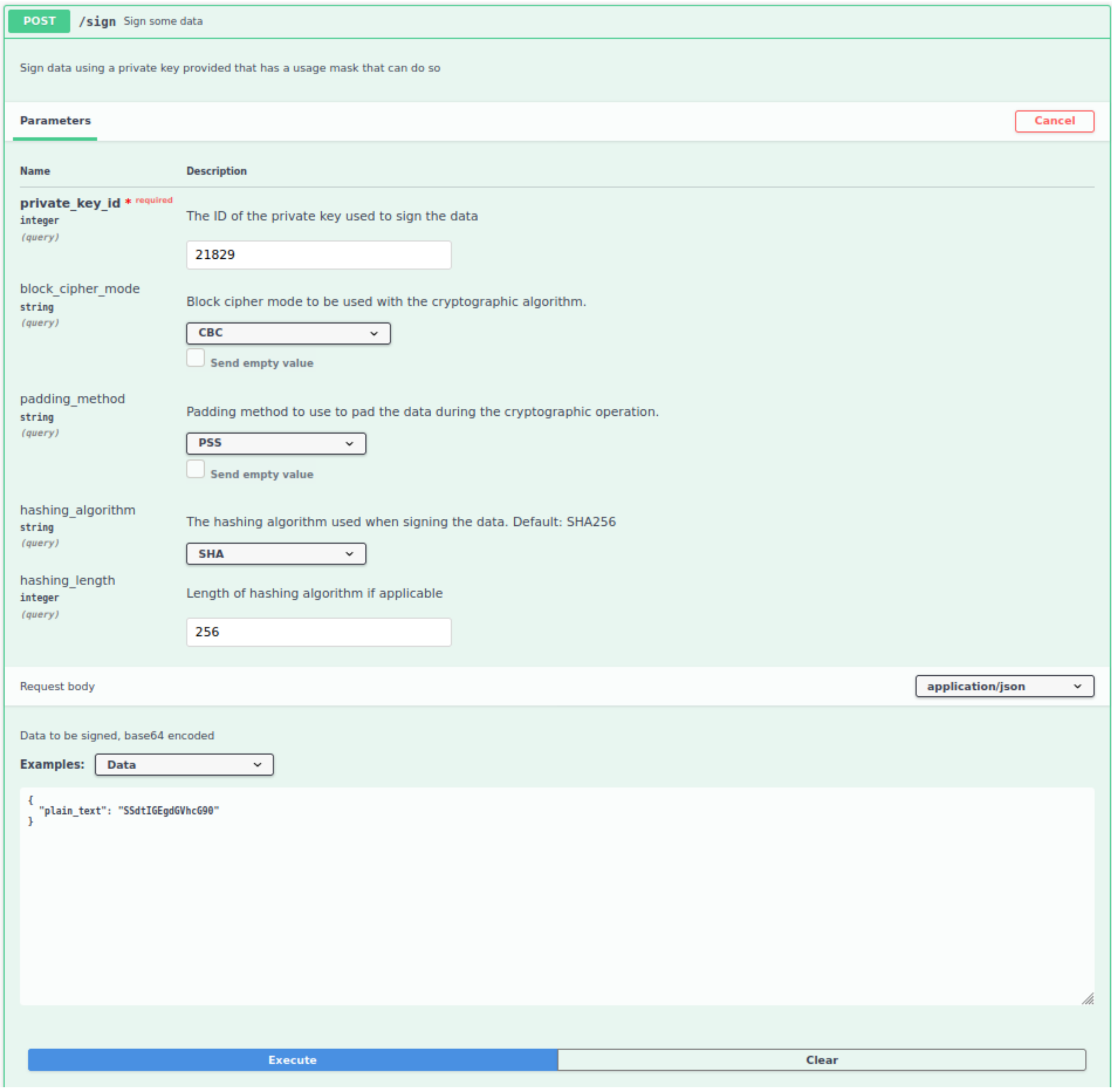
```
curl -X POST "https://kms.example.ogc.org/Helyx-
SIS/KMS/1.0.0/mac?key_id=21830&algorithm=HMAC_SHA256" \
  -H  "accept: application/json" \
  -H  "Content-Type: application/json" \
  -d "{\"plain_text\":\"SSdtIGEgdGVhcG90\"}"
```

To elicit a response similar to the following:

```
{
  "mac": "0ZnilHhU8iJ98VGEiJPPyqFQ2PDTo8SxT0CpP-q8gCw"
}
```

B.2.3.4. Sign

Signing data requires the private key that will sign them have the sign usage mask associated to it.


POST /sign Sign some data

Sign data using a private key provided that has a usage mask that can do so

Parameters

Name	Description
private_key_id * required integer (query)	The ID of the private key used to sign the data 21829
block_cipher_mode string (query)	Block cipher mode to be used with the cryptographic algorithm. CBC Send empty value
padding_method string (query)	Padding method to use to pad the data during the cryptographic operation. PSS Send empty value
hashing_algorithm string (query)	The hashing algorithm used when signing the data. Default: SHA256 SHA
hashing_length integer (query)	Length of hashing algorithm if applicable 256

Request body application/json

Data to be signed, base64 encoded

Examples: Data

```
{
  "plain_text": "SSdtIGEgdGVhcG90"
}
```

Figure 56. Sign data

Alternatively send a cURL request similar to the following:

```
curl -X POST "https://kms.example.ogc.org/Helyx-
SIS/KMS/1.0.0/sign?private_key_id=21829&block_cipher_mode=CBC&padding_method=PSS&hashi
ng_algorithm=SHA&hashing_length=256" \
  -H  "accept: application/json" \
  -H  "Content-Type: application/json" \
  -d "{\"plain_text\":\"SSdtIGEgdGVhcG90\"}"
```

To elicit a response similar to the following:

```
{
  "signature":
"qrLLOC6uDRhG37O0mGIJZBD6A7t6hP0VExCw0G14wXYC6gDvvwADdF1WlyuBxTmB_BiT69k-PP0Dcy_6-
dr4Q47o5636NCR6by1BjcuXAZH1yANqtTLy-Em92YkXTj2d42k8-
oc7Wf6rITFcUiRdXG5_MeZyAmI9I0nPqSzkjw51vZCOR772HKpobbbvXtkEJdsNAQmYqo7AQzb8hgxwS6fO4BC
PpCKnI9EhY6knU2doEYgWdKAnJMXvlez0jgs9VpyZOxtXcf9tle0lc0vzj1BT0k8o2q3-yyoGmDeW8-
UnkuFAa5u_UPjpfdLr4L8lQVRbrx_mx2RRdYxwSi5fzQ"
}
```

B.2.4. Docker-Compose Deployment

The Key Management Server is run on two docker containers deployed using docker-compose. The PyKMIP server is in one and the OpenAPI KMS is in the other.

KMIP relies on mutual TLS authentication to allow a client and server to communicate. In the Helyx implementation, for research and development purposes, Helyx bundled the client and server certificates within the container images. This is done as follows:

- Build the PyKMIP Server image:
  - The certificates for the PyKMIP server and the client are created
  - The PyKMIP image is built using these certificates and the PyKMIP package
  - This image also has the config file, the policy file and certificates bundled into it
  - The run command starts the PyKMIP server in Python
- Build the Key Management Server image:
  - Bundle the required certificates for the client and the public key for the server
  - Add an `.env` file to the image containing details of the PyKMIP server address and port: the default values are `KMS_SERVER_ADDRESS="pykmip-server"`` and `KMS_SERVER_PORT=5696`
  - Add the KMS Python source files and any required packages (including PyKMIP)
  - The run command starts the KMS Flask server in Python.

A docker compose file is used to execute the two connected containers:

```
version: '3'

services:
  swagger_server:
    image: kms:0.0.1
    ports:
      - "8080:8080"
    environment:
      - KMS_SERVER_ADDRESS=pykmip_server
      - KMS_SERVER_PORT=5696
  pykmip_server:
    image: pykmip:0.0.1
    ports:
      - "5696:5696"
```

B.3. Conclusions

The implementation of an OpenAPI compliant interface to interact with a KMIP Server was largely successful and provided a number of useful insights into how a KMS to support OGC APIs may be integrated into such a service in the future.

At the start of the Testbed, the architecture was intended to be close to mirroring the KMIP endpoints, in order to provide as much flexibility as possible to the client implementations. During the course of the Testbed, there was significant thought put into how a KMS might interact with a DCS client, including the standards used for encoding keys, encrypted data and signing data. These ideas were not known at the time the decision was made on what the API might look like. As a result, this implementation of the KMS is quite different to that implemented in D145.

Whilst it has great flexibility it also poses some challenges especially in the offline scenario: the interface is relatively chatty, requiring a number of separate calls to the API to create a key, encrypt data, MAC data, sign data and then retrieve a key; this is just for one data item. This is required to be repeated for as many data items that exist and undertaking this via a RESTful interface can take long time. There are a number of potential approaches for dealing with this, such as:

- Providing support for bulk operations
- Providing support for key creation, encryption, MACing and signing in a single operation.

In addition, it does not currently provide endpoints that allow a client to request a JWE or a JWS directly, which would be useful to the clients as they have been implemented with these standards in mind, which were not envisaged at the start of the Testbed. The KMS implements a content-type within the request of `application/json` though it may be that alternative content-types could be considered that modify the response according to their needs, for example `application/jose+json`.

A fundamental underpinning capability of KMIP-compliant servers is its support for KMIP operation policies that provide access controls over keys and operations on them. An operation policy is a set of permissions, indexed by object type and operation. For any KMIP object type and operation pair, the policy defines who is allowed to conduct the operation on the object type. In the current implementation, all operations are permitted by all users. However, if KMIP is to be used in the future, consideration needs to be given to whether and how these policies align with (Geo)XACML policies used to protect the data itself and how they might be kept up to date. This is additionally complicated by the fact that the Flask web service that provides the RESTful API uses its own mutual authentication with the KMIP Server; there is currently no capability to provide "on behalf of" operations. It is possible that when a client is registered to the KMS, it could also register a keypair with the KMS that is then also registered for authentication with the KMIP server and the KMS dynamically switches its authentication keys based on the client. Otherwise, it may also be possible for the KMS to provide its own authorization that could be linked to a PDP.

As PyKMIP is a software implementation that is not designed to be used in production, it does not provide all key variants and encryption options that are available in a commercial implementation. For example, it is not currently able to encrypt using an RSA key. Further development of the KMS may require changes to the PyKMIP implementation or workarounds to emulate the KMIP functionality within the KMS itself.

The KMS also has some of its own limitations due to the constraints of time and the focus on particular features required of the client. These include:

- Clients are currently unable to supply their own keys to the KMS for storage: they must create a new one
- Key wrapping is not currently supported i.e. One cannot wrap a key when getting a key and return it as a JWE.
- As described above, it does not provide endpoints that support JWE or JWS
- Integration of an authorization server using OAuth 2.0.

Appendix C: Access Control Policies for DCS Server and Mobile Clients

C.1. Overview

This annex introduces the engineering aspects of the policy decision and enforcement points and the policy documents used to specify the rules of access to DCS protected content.

Two implementations for desktop client and mobile clients uses different policy enforcement techniques for access control. In both cases we consider temporal and spatial aspects of access control as well as a role-based security classification.

C.2. GeoXACML Policies for DCS Server in Desktop Scenario

The DCS Server in the desktop/client scenario has the functionality to dynamically construct NATO STANAG compliant responses from an OGC API Features backend service using XML or JSON encodings. This DCS functionality is realized in two modules, as illustrated in Figure 20: The GeoPEP and the GeoPDP.

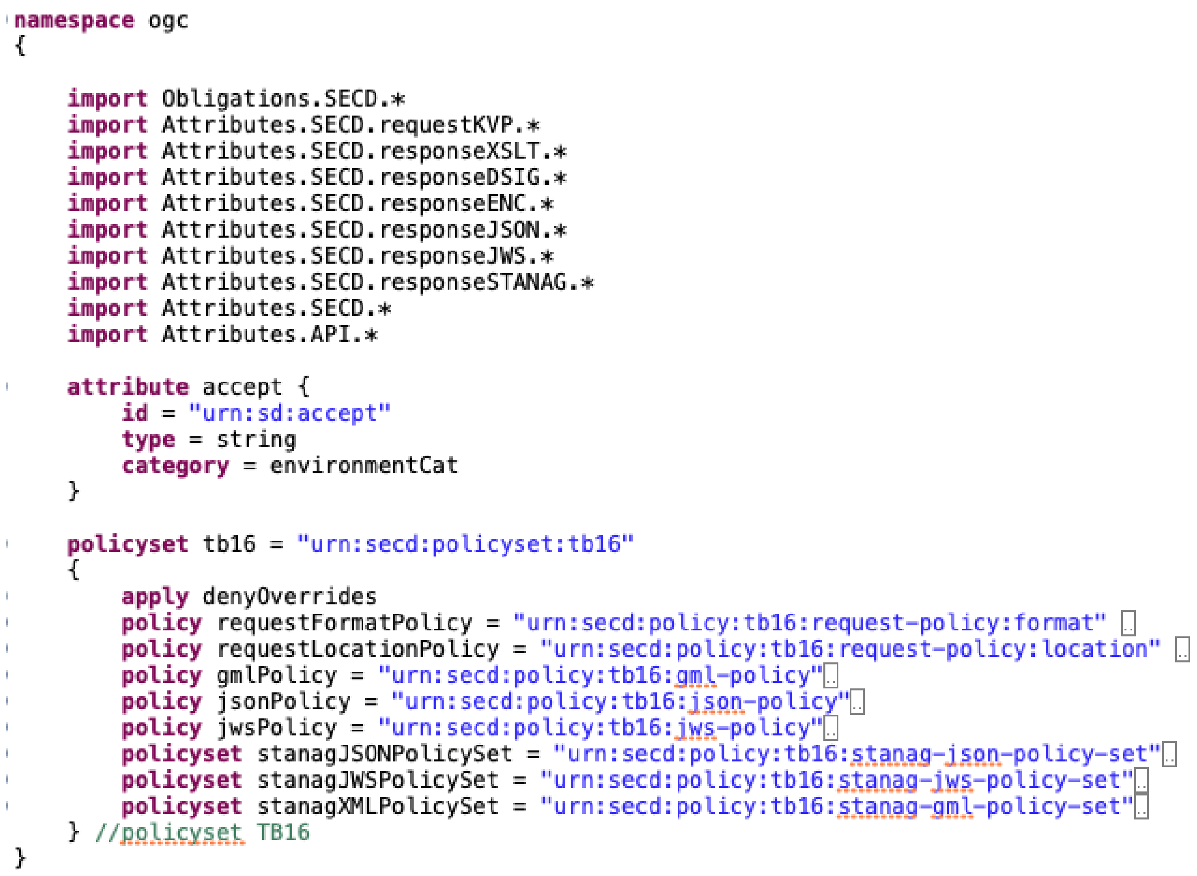
According to the "Data Flow Model" of the XACML 3 standard (Data-flow diagram - Figure 1)[http://docs.oasis-open.org/xacml/3.0/xacml-3.0-core-spec-os-en.html], the GeoPEP implements the PEP, the context handler and the obligation service. According to the (Figure 3 - Policy language model)[http://docs.oasis-open.org/xacml/3.0/xacml-3.0-core-spec-os-en.html], an Authorization Decision sent from the PDP to the PEP can contain a set of obligations (0..1 ObligationExpression). The PEP processes each obligation which can be used to activate specific processing. When it comes to control the GeoPEP for achieving the DCS processing goals for this testbed, specific obligations are leveraged:

- XML Encryption
- XML Digital Signatures
- XML XSLT
- JSON Rewrite
- JSON Encryption (JWE)
- JSON Signatures (JWS)

In addition to the data response obligations listed above, the GeoPEP can also be instrumented to modify the incoming request before sending it to the backend service. The relevant obligations to achieve that request rewriting are:

- HTTP GET query_string rewrite
- HTTP POST rewrite for www-x-form-encoded request bodies
- HTTP POST rewrite for XML encoded request bodies

To explain how the policy controls the GeoPEP processing for the desktop/client use case, the GeoXACML policy is illustrated using ALFA.



```
namespace ogc
{

    import Obligations.SECD.*
    import Attributes.SECD.requestKVP.*
    import Attributes.SECD.responseXSLT.*
    import Attributes.SECD.responseDSIG.*
    import Attributes.SECD.responseENC.*
    import Attributes.SECD.responseJSON.*
    import Attributes.SECD.responseJWS.*
    import Attributes.SECD.responseSTANAG.*
    import Attributes.SECD.*
    import Attributes.API.*

    attribute accept {
        id = "urn:sd:accept"
        type = string
        category = environmentCat
    }

    policyset tb16 = "urn:secd:policyset:tb16"
    {
        apply denyOverrides
        policy requestFormatPolicy = "urn:secd:policy:tb16:request-policy:format"
        policy requestLocationPolicy = "urn:secd:policy:tb16:request-policy:location"
        policy gmlPolicy = "urn:secd:policy:tb16:gml-policy"
        policy jsonPolicy = "urn:secd:policy:tb16:json-policy"
        policy jwsPolicy = "urn:secd:policy:tb16:jws-policy"
        policyset stanagJSONPolicySet = "urn:secd:policy:tb16:stanag-json-policy-set"
        policyset stanagJWSPolicySet = "urn:secd:policy:tb16:stanag-jws-policy-set"
        policyset stanagXMLPolicySet = "urn:secd:policy:tb16:stanag-gml-policy-set"
    } //policyset TB16
}
```

Figure 57. GeoXACML Policy Overview

As illustrated in Figure 54 the policy consists of a PolicySet and individual policies that apply to specific circumstances. The `urn:secd:policy:tb16:request-policy:format` policy evaluates decisions based on the request HTTP ACCEPT header. The `urn:secd:policy:tb16:request-policy:location` policy ensures that users in a particular location have elevated access. The `urn:secd:policy:tb16:*-policy` policies apply to particular data processing requirements.

The following listing illustrates the `location` part of the policy.

```
policy requestLocationPolicy = "urn:secd:policy:tb16:request-policy:location"
{
    apply denyOverrides
    rule requestLocationRule
    {
        permit
        target
            clause
                GeoXACML3.subject_location <=
"CRS=EPSG:4326;POLYGON((40.704586878965245 -74.0361785888672,40.76962180287486
-74.0361785888672,40.76962180287486 -73.94966125488283,40.704586878965245
-73.94966125488283,40.704586878965245 -74.0361785888672))":geometry
            and
                time <= "18:00:00Z":time
        on permit {
            obligation requestKVP {
                action = "insert"
                key = "bbox"
                value = "40.704586878965245,-74.0361785888672,40.76962180287486,-
73.94966125488283"
            }
            obligation requestKVP {
                action = "remove"
                key = "access_token"
            }
            obligation requestKVP {
                action = "remove"
                key = "subjectlocation"
            }
        }
    }
}
```

The following policy instruments the GeoPEP to transform backend service response into NATO STANAG 4778 compliant container format.

```
policyset stanagJSONPolicySet = "urn:secd:policy:tb16:stanag-json-policy-set"
{
    target
        clause
            f == "stanag+json" or
            accept == "application/stanag+json"
    apply denyOverrides
    featurePolicySet
    policy JSON {
        apply permitOverrides
        rule {
            permit
        }
        on permit {
            obligation responseJSON {
                content_type = "application/stanag+json"
            }
        }
    }
}
```

Important in the policy above is the involvement of the `responseJSON` obligation attached to the PERMIT response.

To transform GML backend response into STANAG 4778 including Digital Signature and Encryption (for the 'poi' feature type), the following policy is used.

```
policyset stanagXMLPolicySet = "urn:secd:policy:tb16:stanag-gml-policy-set"
{
    target
        clause
            f == "stanag" or
            f == "stanag+gml" or
            accept == "application/stanag+gml"
    apply permitOverrides
    policy poiPolicy = "urn:secd:policy:tb16:poi-policy"
    {
        target
            clause
                path == "/dcs/collections/poi/items"
        apply permitOverrides
        rule responseLocationRule
        {
            permit
            target
                clause
                    GeoXACML3.subject_location <=
"CRS=EPSG:4326;POLYGON((40.704586878965245 -74.0361785888672,40.76962180287486
-74.0361785888672,40.76962180287486 -73.94966125488283,40.704586878965245
-73.94966125488283,40.704586878965245 -74.0361785888672))":geometry
                    and
                    time <= "18:00:00Z":time
            on permit {
                obligation responseXSLT {
                    document = "..."
                    parameter = "unclassifiedFeatureType=states tiger_roads
poly_landmarks poi"
                }
            }

        }
        rule permitRule
        {
            permit
            target
                clause
                    subject_clearance == "top_secret"
                    or
                    subject_affiliation == "OGC Testbed-16"
            on permit {
                obligation responseXSLT {
                    document = "..."
                }
            }
            on permit {
                obligation responseENC {
                    xpath = "//*[local-name() =
'Metadata'][./@xml:id='FeatureType']/*"
                    responseENC.key_algorithm =
"http://www.w3.org/2009/xmlenc11#aes128-cbc"
                }
                obligation responseENC {
                    xpath = "//*[local-name() = 'Metadata'][./@xml:id='STANAG4774']/*"
                    responseENC.key_algorithm =
"http://www.w3.org/2009/xmlenc11#aes256-gcm"
                }
                obligation responseENC {
                    xpath = "//*[local-name() = 'Data']/*"
                    responseENC.key_algorithm =
"http://www.w3.org/2009/xmlenc11#aes256-gcm"
                }
            }
        }
        rule denyRule
        {
            deny
            target
                clause
```

```
                    subject_clearance == "secret"
                    or
                    subject_clearance == "classified"
                    or
                    subject_clearance == "unclassified"
          }
      }
      on permit {
          obligation responseDSIG {
              private_key_file = "/etc/pki/tls/private/testbed15.pem"
              private_key_name = "Dr. No"
              certificate_file = "/etc/pki/tls/certs/testbed15.crt"
              id_element_value = "#WFS"
              id_element_qname = "id"
          }
      }
  }
```

Examining the Obligations from the policy explain naturally how the GeoPEP converts XML backend to NATO STANAG 4778:

- use of `responseXSLT` obligation to transform GML into NATO STANAG 4778 structure
- use `responseENC` obligation to encrypt `Metadata`, `Data` sections
- use `responseDSIG` obligation to digitally sign the response

C.3. GeoXACML Policies for Mobile Scenarios

C.3.1. Use Case:

- Access to the features is only possible if user location is within an operational boundary:
  - POLYGON 40.8175 -74.0008, 40.753 -74.0008, 40.753 -73.9499, 40.8175 -73.9499, 40.8175 -74.0008
- Location of user is defined by attribute
  - “urn:sd:location” := GeoXACML geometry: CRS=EPSG:4326;Point(40.76 -74.0)
- Bell-La Padula Policy for classified feature types
  - Permit: clearance(user) >= classification(feature_type)
- Users have attribute clearance
  - “urn:sd:clearance” := {top_secret, secret, confidential, classified}
- Resources are features of feature_type
  - “urn:sd:feature-type” := {“poi”, “poly_landmarks”, “tiger_roads”, “states”}
- Classification per feature type
  - “poi” := “top_secret”
  - “poly_landmarks” := “secret”

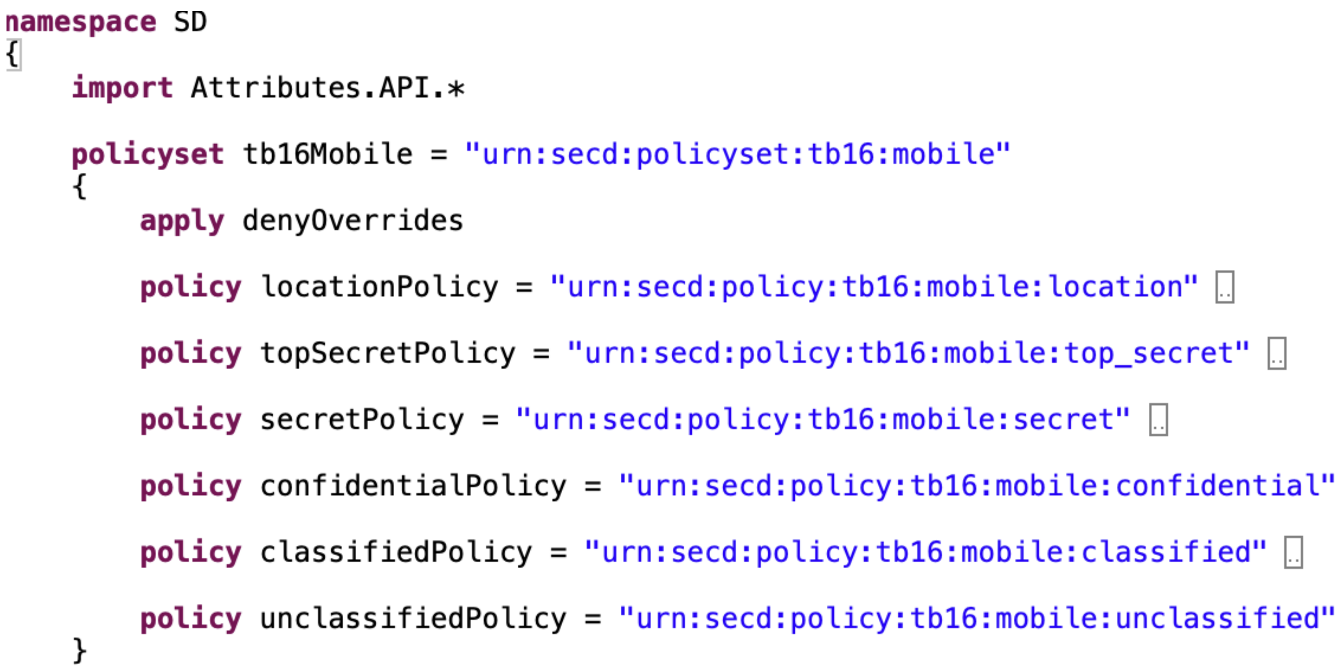
    - “tiger_roads” := “confidential”
    - “states” := “classified”
- Any other feature type is not classified

C.3.2. GeoXACML Policy in ALFA

ALFA, the Abbreviated Language For Authorization, is a pseudocode language used in the formulation of access-control policies.

Option (i)

Packaged a one inline PolicySet Produces one single Policy file

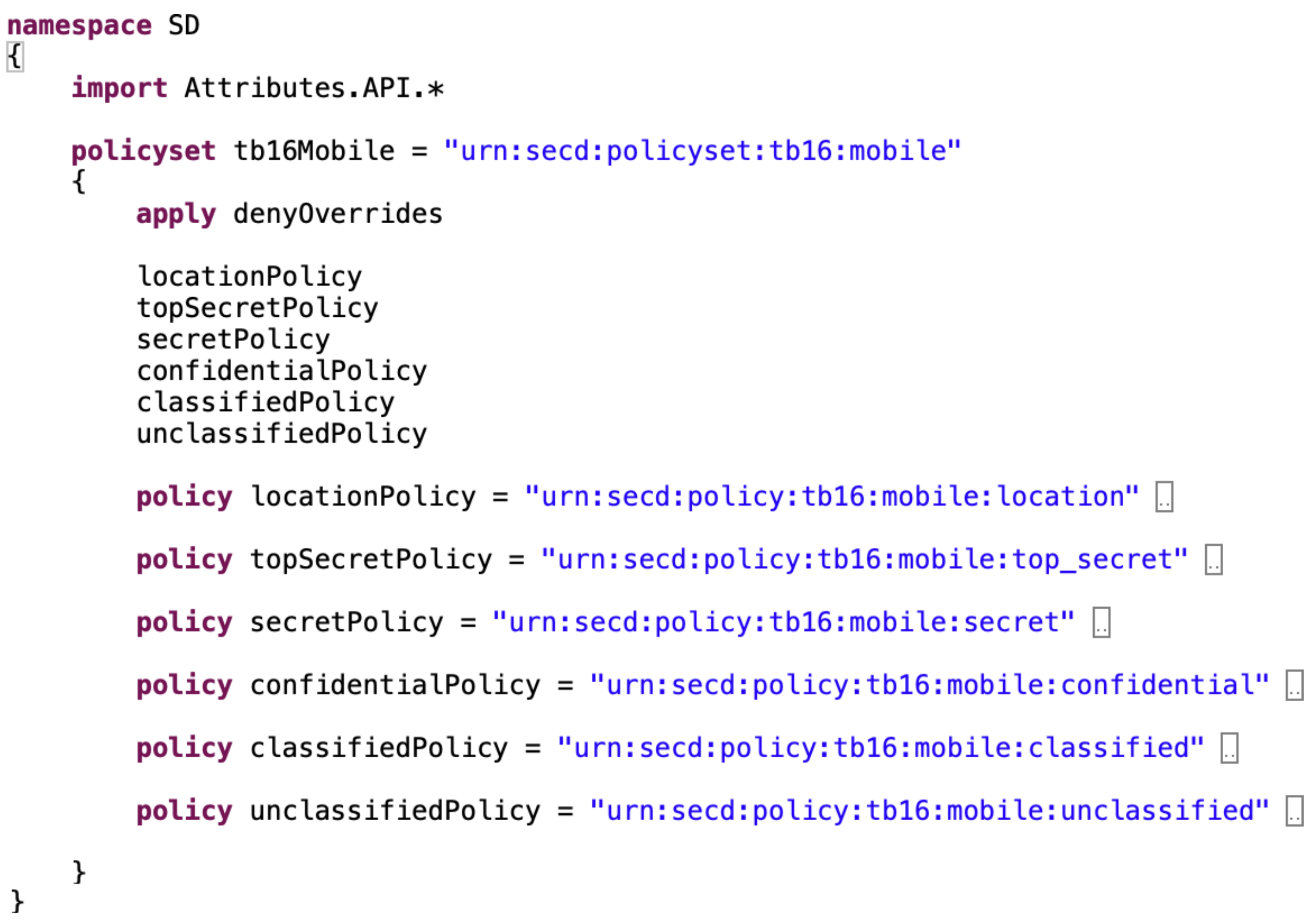


```
namespace SD
{
    import Attributes.API.*

    policyset tb16Mobile = "urn:secd:policyset:tb16:mobile"
    {
        apply denyOverrides

        policy locationPolicy = "urn:secd:policy:tb16:mobile:location"

        policy topSecretPolicy = "urn:secd:policy:tb16:mobile:top_secret"

        policy secretPolicy = "urn:secd:policy:tb16:mobile:secret"

        policy confidentialPolicy = "urn:secd:policy:tb16:mobile:confidential"

        policy classifiedPolicy = "urn:secd:policy:tb16:mobile:classified"

        policy unclassifiedPolicy = "urn:secd:policy:tb16:mobile:unclassified"
    }
```

Figure 58. Single Policy file

Option (ii)

Packaged as linked Policies Produces one file for the PolicySet and one file per each Policy



```
namespace SD
{
    import Attributes.API.*

    policyset tb16Mobile = "urn:secd:policyset:tb16:mobile"
    {
        apply denyOverrides

        locationPolicy
        topSecretPolicy
        secretPolicy
        confidentialPolicy
        classifiedPolicy
        unclassifiedPolicy

        policy locationPolicy = "urn:secd:policy:tb16:mobile:location"

        policy topSecretPolicy = "urn:secd:policy:tb16:mobile:top_secret"

        policy secretPolicy = "urn:secd:policy:tb16:mobile:secret"

        policy confidentialPolicy = "urn:secd:policy:tb16:mobile:confidential"

        policy classifiedPolicy = "urn:secd:policy:tb16:mobile:classified"

        policy unclassifiedPolicy = "urn:secd:policy:tb16:mobile:unclassified"

    }
}
```

Figure 59. Linked policies

C.3.3. Policy and Verification

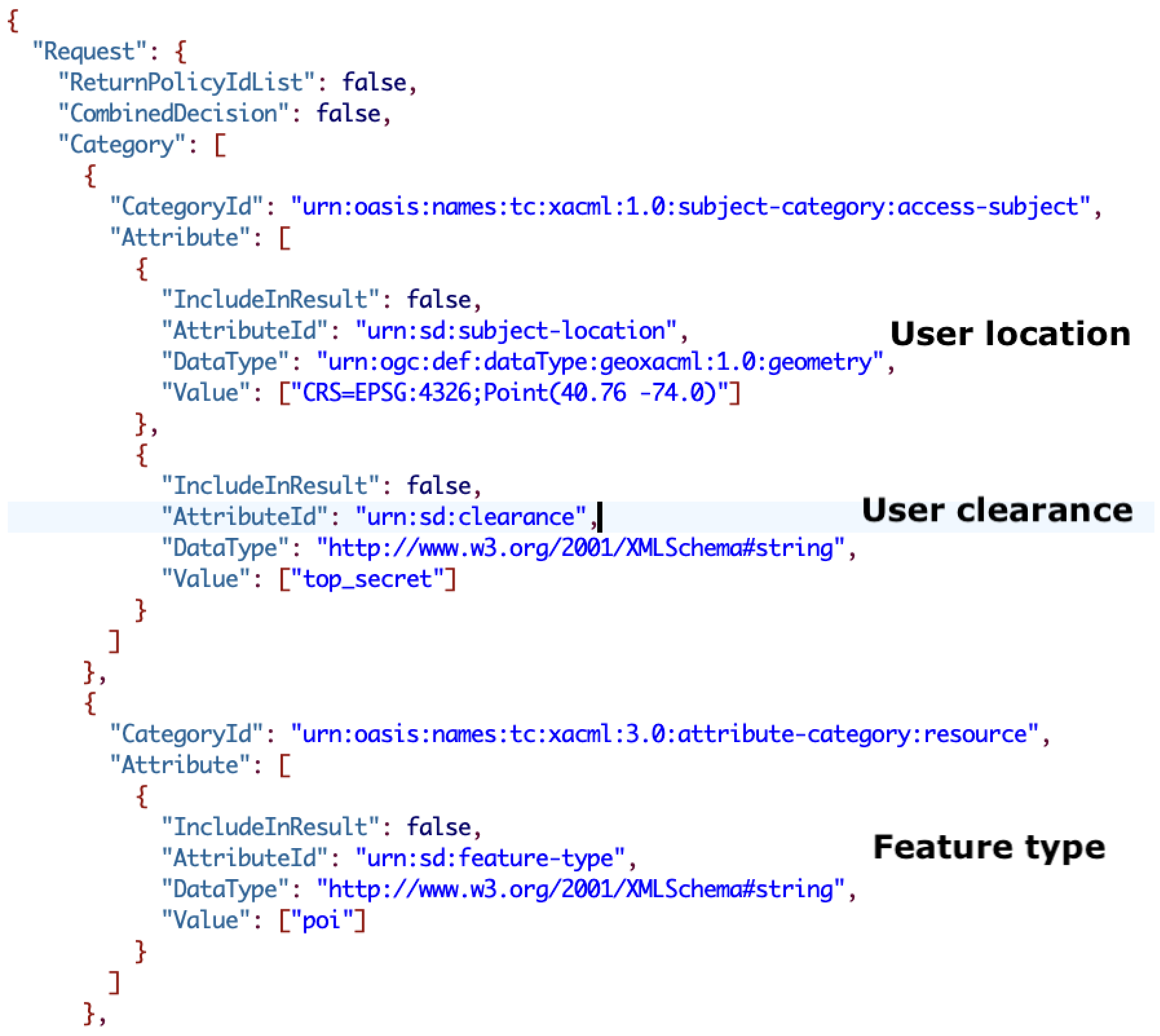
Endpoint

- https://ogc.secure-dimensions.com/geopdp-mobile

POST requests with Http header Content-Type

- “Content-Type: application/xacml+json”

Example ADR



```
{
  "Request": {
    "ReturnPolicyIdList": false,
    "CombinedDecision": false,
    "Category": [
      {
        "CategoryId": "urn:oasis:names:tc:xacml:1.0:subject-category:access-subject",
        "Attribute": [
          {
            "IncludeInResult": false,
            "AttributeId": "urn:sd:subject-location",
            "DataType": "urn:ogc:def:dataType:geoxacml:1.0:geometry",
            "Value": ["CRS=EPSG:4326;Point(40.76 -74.0)"]
          },
          {
            "IncludeInResult": false,
            "AttributeId": "urn:sd:clearance",
            "DataType": "http://www.w3.org/2001/XMLSchema#string",
            "Value": ["top_secret"]
          }
        ]
      },
      {
        "CategoryId": "urn:oasis:names:tc:xacml:3.0:attribute-category:resource",
        "Attribute": [
          {
            "IncludeInResult": false,
            "AttributeId": "urn:sd:feature-type",
            "DataType": "http://www.w3.org/2001/XMLSchema#string",
            "Value": ["poi"]
          }
        ]
      },
```

Figure 60. ADR example

C.3.4. Verification

Table 8. First Table

ADR	User Location	Decision
0	CRS=EPSG:4326;Point(40.76 -74.0)	Permit
1	CRS=EPSG:4326;Point(40.75 -74.0)	Deny

Table 9. Second Table

ADR	Clearance	Feture Type	Decision
10	top_secret	poi, poi, poly_landmarks archsites	Permit, Permit, Permit
20	secret secret	poly_landmarks, poly_landmarks tiger_roads	Permit, Permit
21	secret	poi, poi archsites	Deny, Deny
…			

C.3.5. ADR Example

```
{
    "Request" : {
        "ReturnPolicyIdList": false,
        "CombinedDecision": false,
        "Category": [
            {
                "CategoryId": "urn:oasis:names:tc:xacml:1.0:subject-category:access-
subject",
                "Attribute": [
                    {
                        "IncludeInResult": false,
                        "AttributeId": "urn:sd:subject-location",
                        "DataType": "urn:ogc:def:dataType:geoxacml:1.0:geometry",
                        "Value": ["CRS=EPSG:4326;Point(40.76 -74.0)"]
                    },
                    {
                        "IncludeInResult": false, "AttributeId": "urn:sd:clearance",
                        "DataType": "http://www.w3.org/2001/XMLSchema#string",
                        "Value": ["top_secret"]
                        }
                    ]
                },
            {
                "CategoryId": "urn:oasis:names:tc:xacml:3.0:attribute-
category:resource",
                "Attribute": [
                    {
                        "IncludeInResult": false, "AttributeId": "urn:sd:feature-type
",

                        "DataType": "http://www.w3.org/2001/XMLSchema#string",
                        "Value": ["poi"]
                    }
                ]
            }
        ]
    }
}
```

Appendix D: Data Centric Security Roles

D.1. Mobile Scenario

The Testbed-16 DCS scenario for the mobile client was defined as the following:

Cell-phone scenario

A Sargent in the U.S. National Guard has been deployed on a disaster recovery mission. He carries with him a smart phone which contains sensitive data. When meeting with first responders, how does he share critical information with them without compromising sensitive information? How does internet connectivity affect that scenario?

Hypothesis: Use of the Data Centric Security techniques developed in Testbed-16 could address this problem. All sensitive data is encapsulated in a Data Centric Security package. Security policies are defined using GeoXACML. A Policy Enforcement Point (PEP) applet only allows access that data allowed under the currently active security policy. Authorized users can set the active security policy.

End State: The Sargent selects the security policy appropriate for the intended audience. He can now access data on his smart phone without worries about exposing sensitive information.

Multiple mobile implementations were created to explore how to address the goals of this scenario, some of which include the concept of "DCS Roles".

D.2. DCS Roles

Within the Mobile Scenario, an individual DCS Role is either the clearance/security authorizations for a specific person, or a generic clearance for a group of people (as defined by an administrator) - a member of the "intended audience" with whom the primary user wishes to share critical information without exposing sensitive information.

D.2.1. DCS Roles vs Users

Within this role-based mobile implementation, a user is the specific assigned user for the mobile device. That user has their personal security clearance loaded onto the device as a DCS Role. In addition to that personal DCS Role, per the scenario multiple generic DCS Roles representing generic security clearances for the categories of people the user may encounter in the field who the user may wish to share information.

D.2.2. DCS Roles vs DCS Data

Within this concept, each DCS Data item is to be restricted according to a specific Policy Identifier and a specific Classification, as specified within a DCS Data container (as described elsewhere in this document). Whereas each DCS Role could potentially specify access to multiple Classifications and multiple Contexts.

This allows the "filtering" of data being displayed on the mobile device to show only DCS Data items that meet the restrictions of the current active DCS Role.

Furthermore, this allows for (requires) the DCS Data and (list of) DCS Roles to be distributed and installed separately on the mobile devices.

D.2.3. DCS Roles vs NATO STANAG 4774

The DCS Roles concept was informed by and adapted from NATO 4774 Appendix 2, which details that each clearance must have (at least) the following items:

- PolicyIdentifier (e.g. "NATO", "USA", "GBR")
- ClassificationList (e.g. "UNCLASSIFIED, RESTRICTED, CONFIDENTIAL, SECRET....")
- Context (the different categories of information/context that this clearance allows)

The following is an "example clearance for the United Kingdom, a founding member of NATO" from 4774, as well as a GeoJSON adaptation equivalent.

UK NATO 4774 Example

```
<sclr:ConfidentialityClearance xmlns:sclr=
"urn:nato:stanag:4774:confidentialityclearance:1:0" xmlns=
"urn:nato:stanag:4774:confidentialitymetadatalabel:1:0"> <PolicyIdentifier>
NATO</PolicyIdentifier> <sclr:ClassificationList>
<Classification>UNCLASSIFIED</Classification> <Classification>
RESTRICTED</Classification> <Classification>CONFIDENTIAL</Classification>
<Classification>SECRET</Classification> <Classification>TOP SECRET</Classification>
</sclr:ClassificationList>
<Category TagName="Context" Type="PERMISSIVE">
<GenericValue>NATO</GenericValue>
App 2-A4
Edition A Version 1
Annex A to Appendix 2 to ADatP-4774
<GenericValue>EAPC</GenericValue> <GenericValue>GEORGIA</GenericValue>
<GenericValue>ISAF</GenericValue> <GenericValue>KFOR</GenericValue> <GenericValue>
PFP</GenericValue> <GenericValue>RUSSIA</GenericValue> <GenericValue>
UKRAINE</GenericValue> <GenericValue>Releasable</GenericValue>
</Category>
<Category TagName="Releasable To" Type="PERMISSIVE">
<GenericValue>NATO</GenericValue> <GenericValue>EAPC</GenericValue> <GenericValue>
ISAF</GenericValue> <GenericValue>KFOR</GenericValue> <GenericValue>PFP</GenericValue>
<GenericValue>GBR</GenericValue>
</Category>
<Category TagName="Only" Type="PERMISSIVE">
<GenericValue>NATO</GenericValue>
<GenericValue>GBR</GenericValue>
</Category>
<Category TagName="Additional Sensitivity" Type="RESTRICTIVE">
<GenericValue>ATOMAL</GenericValue> <GenericValue>BOHEMIA</GenericValue>
<GenericValue>CRYPTO</GenericValue>
</Category> </sclr:ConfidentialityClearance>
```

UK NATO GeoJSON Adaptation Example

```
    {
        "name": "UK NATO",
        "id": "GBR",
        "PolicyIdentifier": "NATO",
        "ClassificationList": [
            "UNCLASSIFIED",
            "RESTRICTED",
            "CONFIDENTIAL",
            "SECRET",
            "TOP SECRET"
        ],
        "Context": [
            "NATO",
            "EAPC",
            "GEORGIA",
            "ISAF",
            "KFOR",
            "PFP",
            "RUSSIA",
            "UKRAINE",
            "Releasable"
        ],
        "Only": [
            "NATO",
            "GBR"
        ]
    },
```

D.3. DCS Mobile Client Role Switching

The following is an example implementation of the mobile client displaying DCS Role selection on a user device, as well as the filtering of different features by clearance as represented by DCS Role.

D.3.1. DCS Mobile Client Role Selection

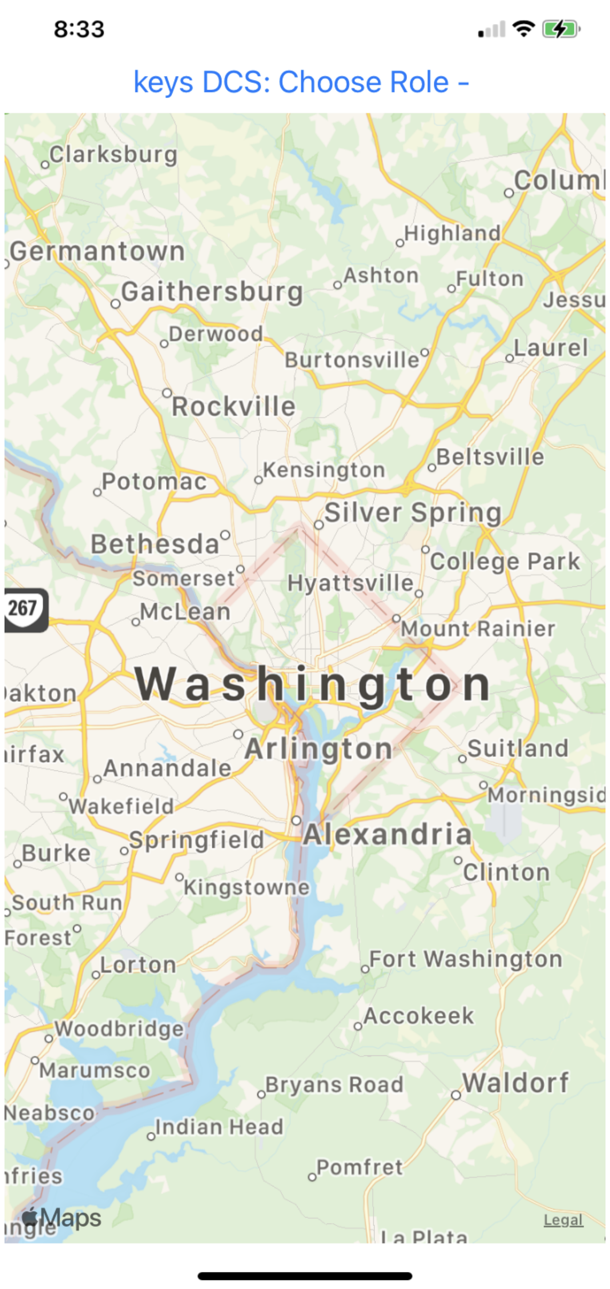
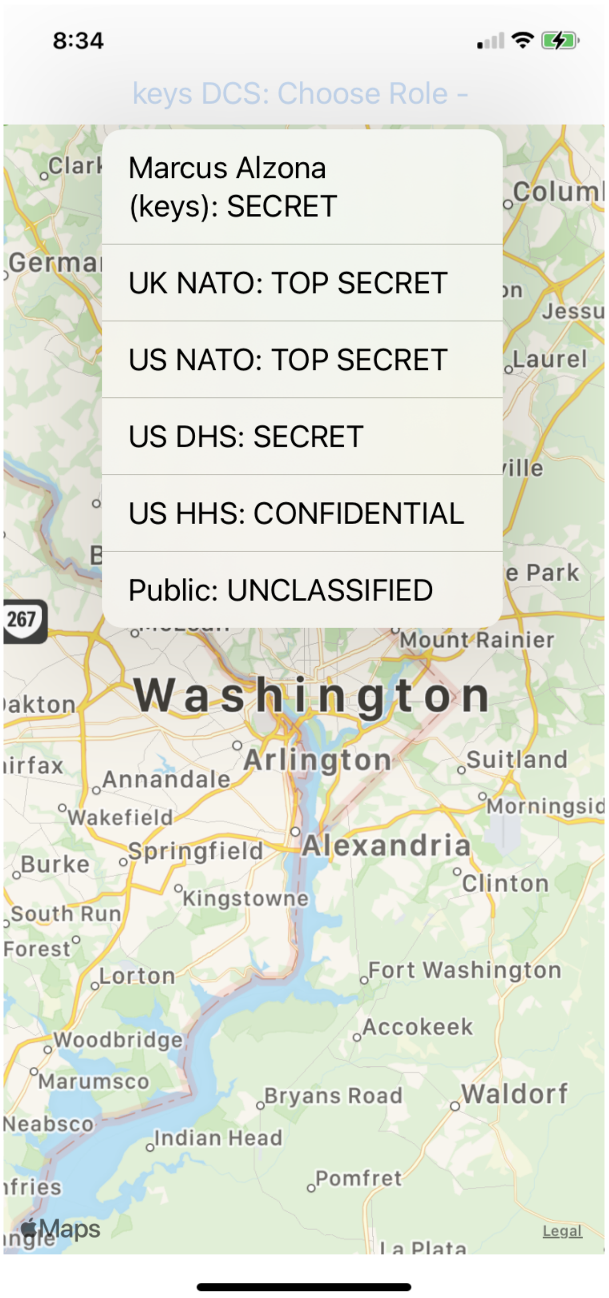


Figure 61. Choose Role



Marcus Alzona (keys): SECRET

UK NATO: TOP SECRET

US NATO: TOP SECRET

US DHS: SECRET

US HHS: CONFIDENTIAL

Public: UNCLASSIFIED

Figure 62. Choose Role Selection

D.3.2. DCS Mobile Client - National Geospatial Intelligence Agency

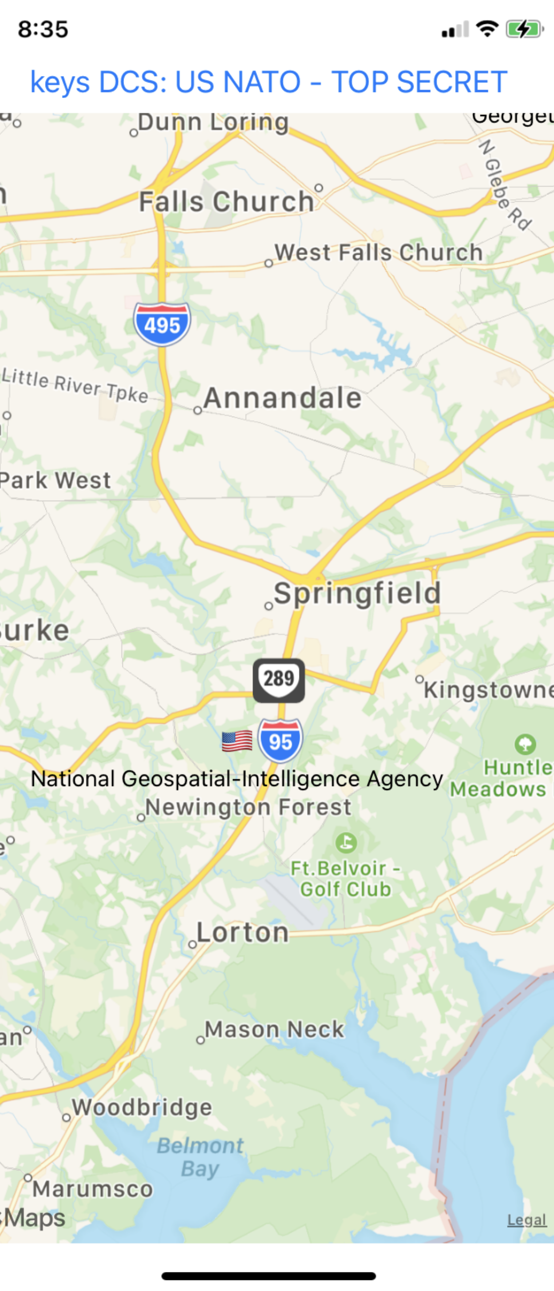


Figure 63. NGA - US NATO TS

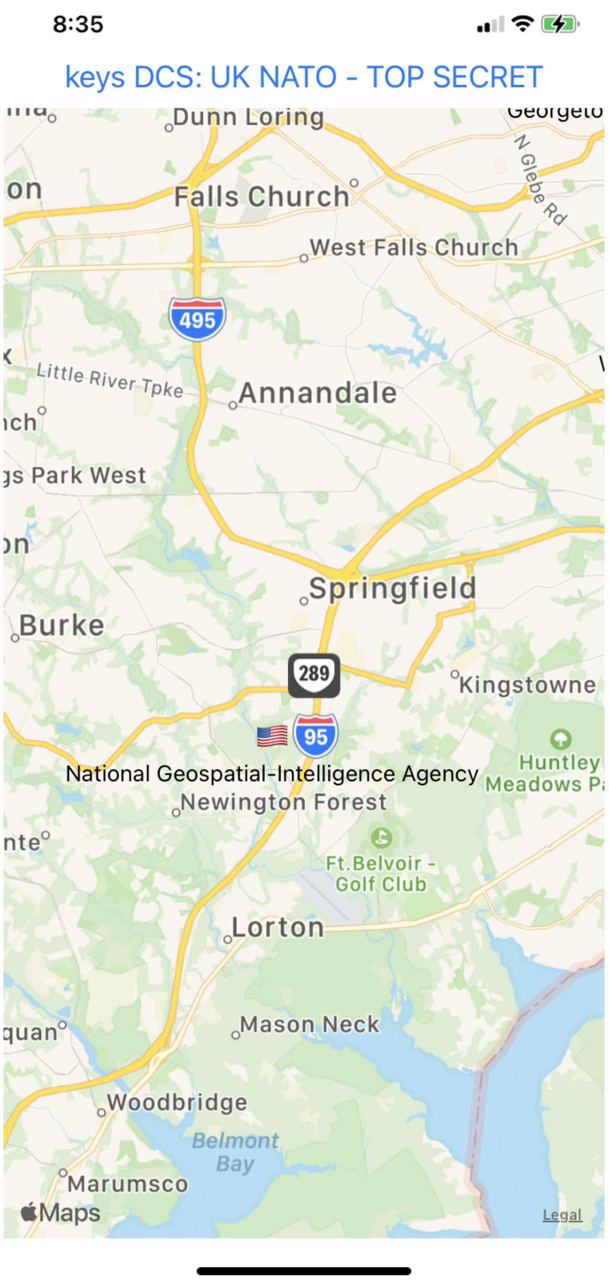


Figure 64. NGA - UK NATO TS

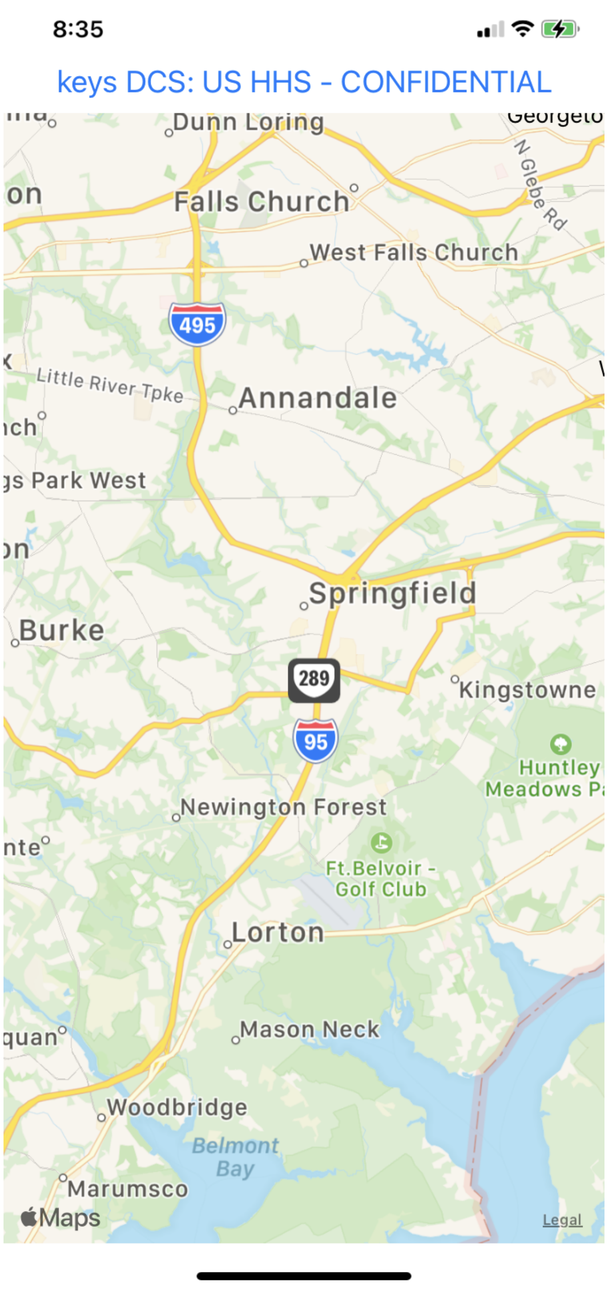


Figure 65. NGA - US HHS CON

D.3.3. DCS Mobile Client - United States Capitol Outbreak

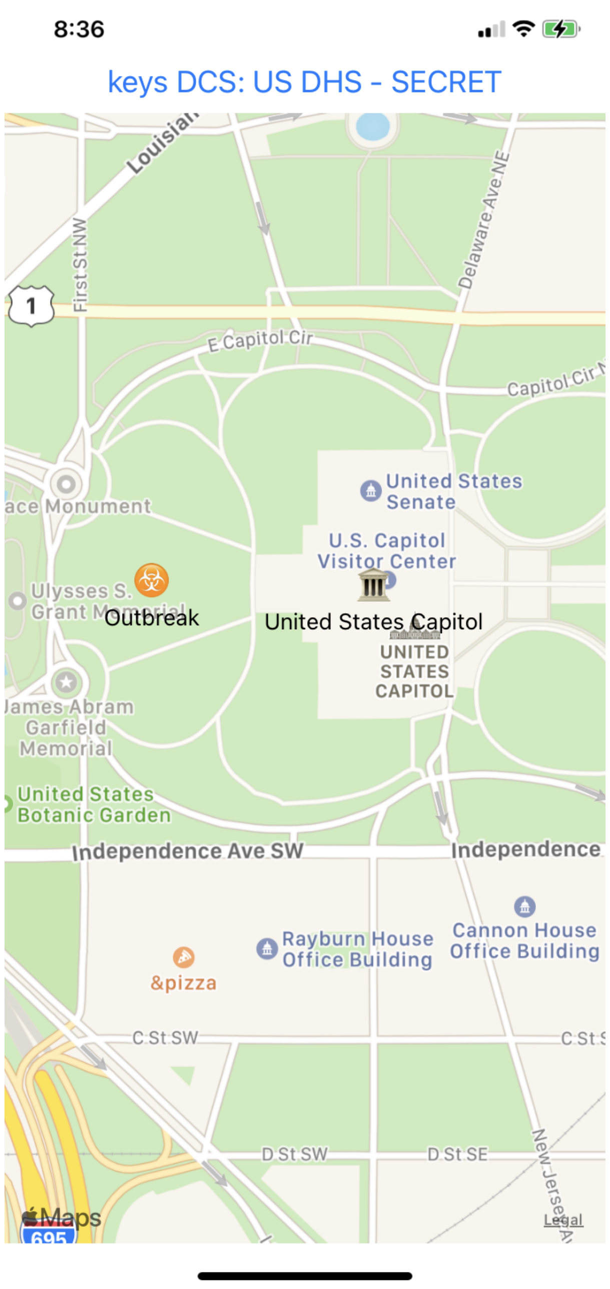


Figure 66. US Capitol - US DHS S

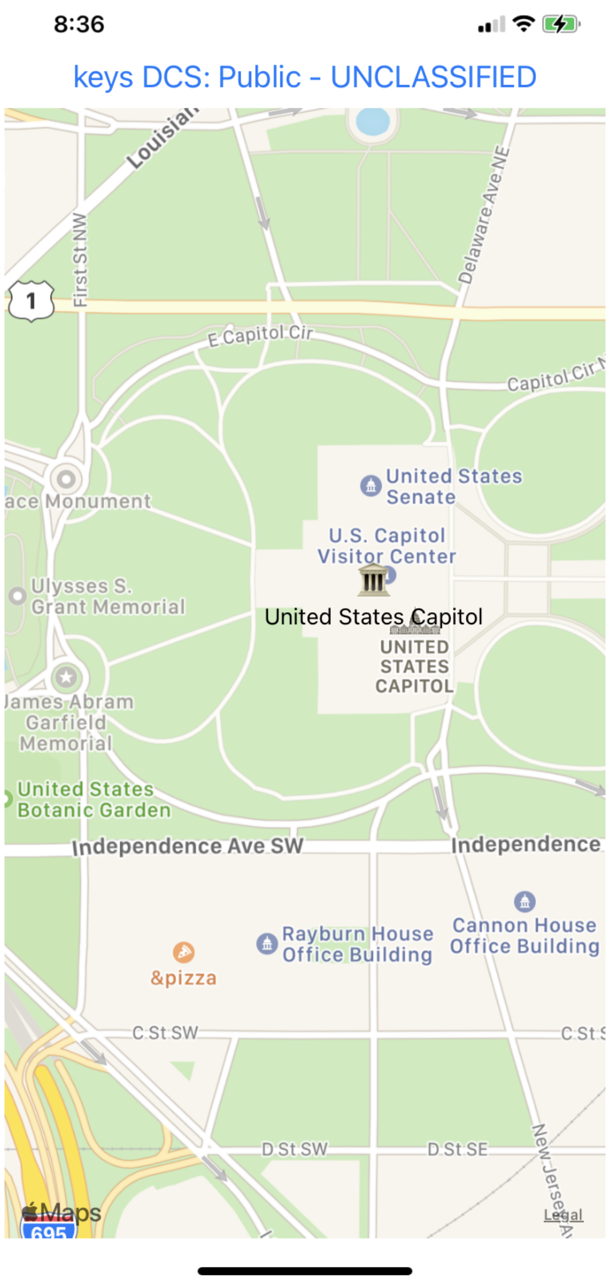


Figure 67. US Capitol - Public UNC

D.3.4. DCS Mobile Client Bethesda Walter Reed

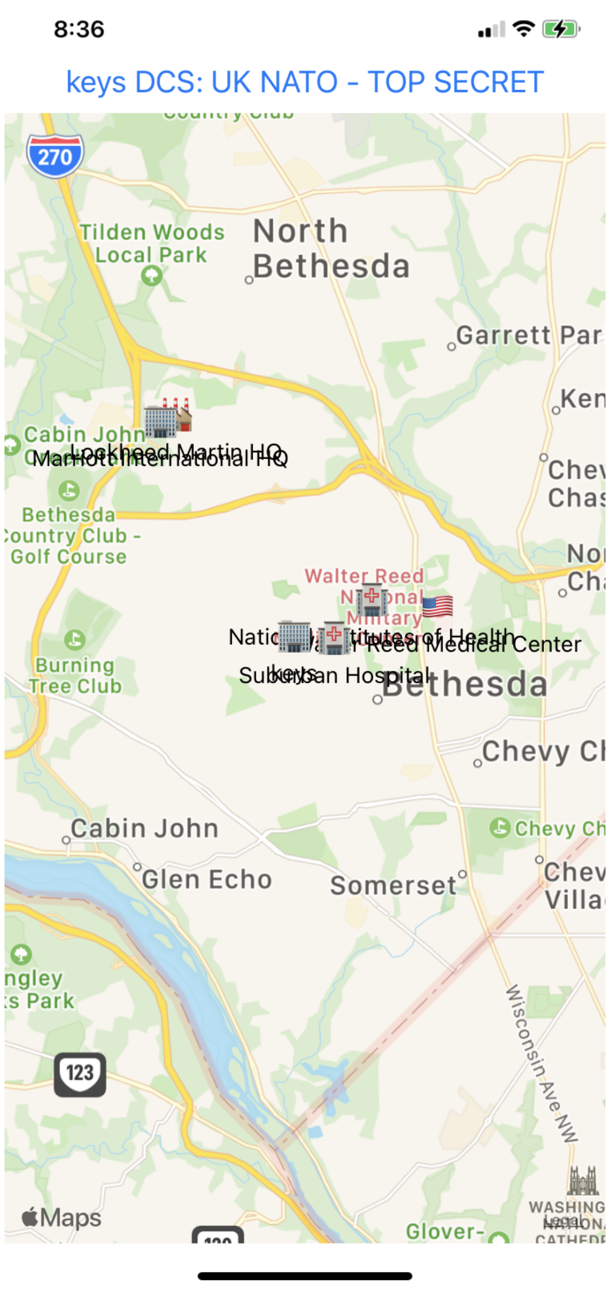


Figure 68. Bethesda Walter Reed - UK NATO TS

8:35

keys DCS: US HHS - CONFIDENTIAL

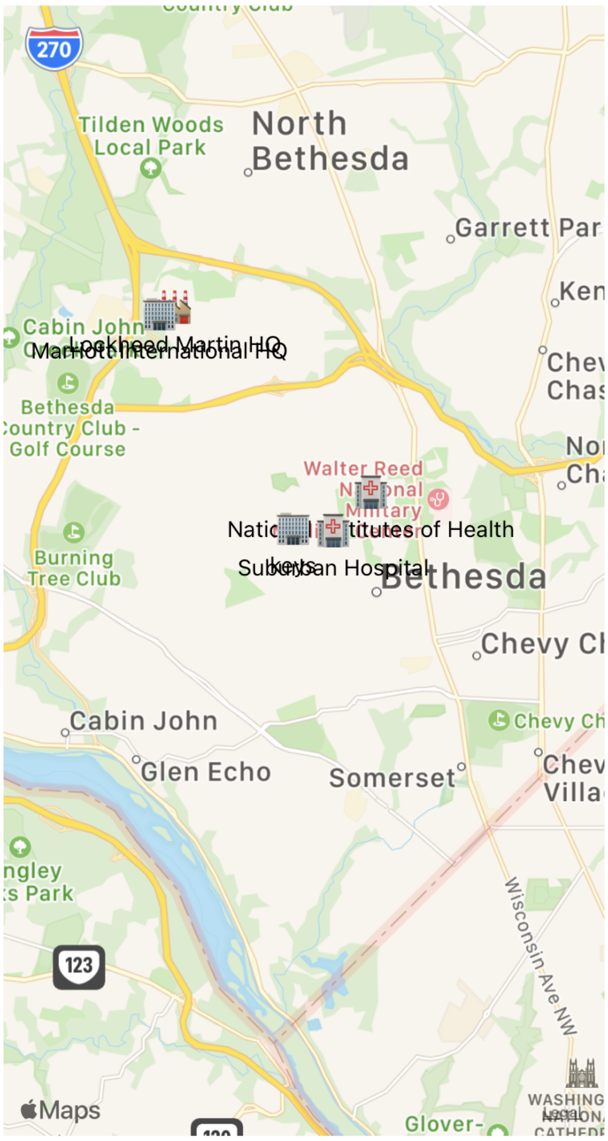


Figure 69. Bethesda Walter Reed - US HHS CON

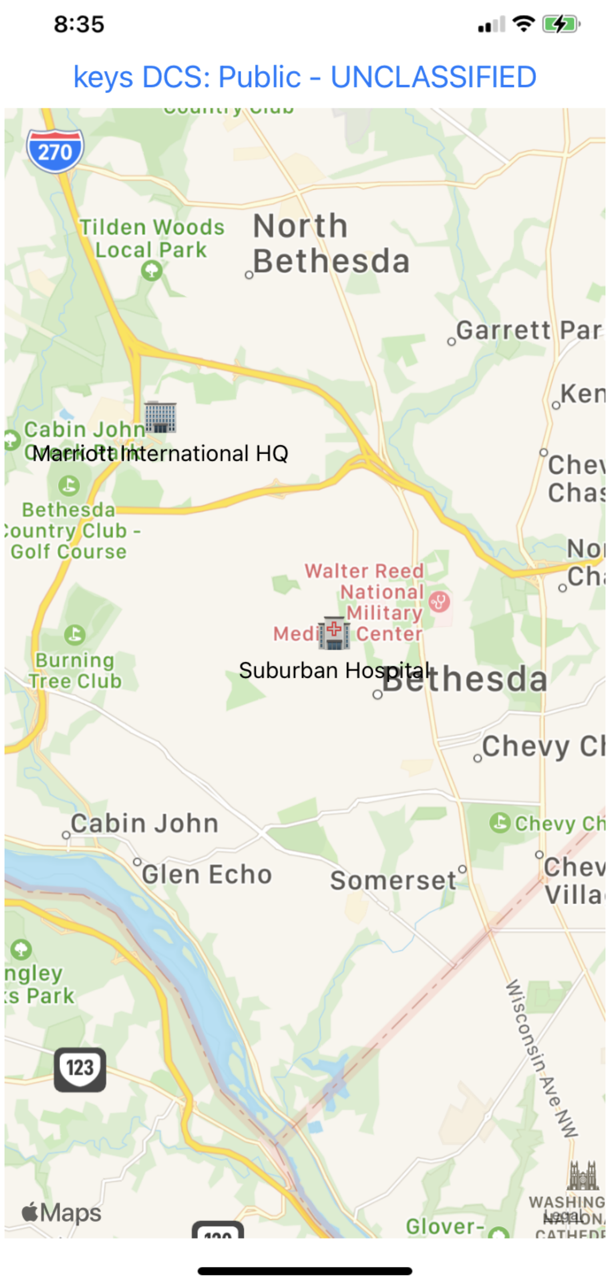


Figure 70. Bethesda Walter Reed - Public UNC

Appendix E: Revision History

This table presents the document revision history.

Table 10. Revision History

Date	Editor	Release	Primary clauses modified	Descriptions
April 25, 2020	A. Balaban	.1	all	Initial draft version
August 13, 2020	A. Balaban	.2	all	Second draft version
October 15, 2020	A. Matheus	.3	Appendix B	Contributed first draft
October 31, 2020	A. Balaban	.4	all	Incorporated comments for final draft
November 11, 2020	A. Matheus	.5	Appendix D	Contributed first draft
November 11, 2020	A. Matheus	.6	all	Proofreading
November 18, 2020	A. Balaban	1.0	all	Release version
November 20, 2020	A. Balaban	1.1	all	Release version with minor adjustments

Appendix F: Bibliography