1 APPENDIX SECOM Test Project StormGeo Report

1.1 Participant Testbed description

<description of the participants testbed>

- Overall “architecture sketch”
- Service interfaces implemented (refer to sub clauses in SECOM clause 5)
- Libraries used
- Data protection implemented (refer to sub clauses in SECOM clause 7)
- Transport security implemented (refer to sub clauses in SECOM clause 6)

1.2 Participant Test report

Describe outcome within each Test Case. Outcome is both observations during creation of the testbed and observations on test runs using the testbed.

1.2.1 Observations

Number	Observation	Consequence	Suggestion	Document reference
SGEO-001	Different cryptographic libraries use different binary encodings for signature.	Possibility to have invalid signatures only because for different libraries in use.	Define expected signature binary encoding in specification so users could choose libraries accordingly. Widely used encoding – DER.	
SGEO-002	Secom specification does not define rules regarding encoding.	Different encodings might lead to errors (ASCII, UTF-8)	Define the rules for encoding selection.	
SGEO-003	Without reading annex, these patterns seems to use terms that is associated with different communication protocols, such as messaging. Also word message is commonly used to describe whole http request/response (headers, body), but in this context, it is used to describe just	Exchange pattern values might be unclear and misleading.	Use clear naming and meaning for exchange patterns without collision between communication protocols as SECOM should be only REST over HTTP. Consider if those brings ant actual value to specification.	**Table 5-1 - Service interfaces overview,** **Message exchange patterns**

	the body (content) part. Also term 'Technical response' is questionable, if it is only http code without body (content) or some specific model to represent error state. It also requires review to check if correct Exchange patterns have been used as both upload endpoints are set as ONE_WAY, but by descriptions, REQUEST_CALLBACK would be more suitable because of ack operation. Wider discussion is needed to consider exchange patterns, need and value they bring to specification.			
SGEO-004	UploadObject name sounds like a method/ verb.	General naming consistency.	Rename to ObjectUpload or ObjectUploadModel which naturally sounds more like a noun than a verb (REST describes resource as noun).	**Figure 1 - Upload interface UML diagram**
SGEO-005	In case of success, same model as for errors is returned	Not consistent with HTTP and REST.	If success scenario is not supposed to return any data, 204(NoContent) status code should be returned.	**Figure 2 - Upload interface UML diagram**
SGEO-006	In HTTP and REST context endpoint is called as a service interface, which could suggest that it contains a set of operations.	General document readability.	Call it as more familiar name: endpoint/action/operation. Personal opinion: Object upload and object upload by link could logically be called object upload interface	
SGEO-007	Enums are passed as integers.	OpenApi contract contains integer values without actual enum meaning.	Pass enums as string, so OpenApi contract would contain actual enum values also as generated client would have full enum without additional mapping.	**Table 5-2 – Information input for Upload interface**

		Additional mapping is necessary to know, what number actually means.		
SGEO-008	Encoding column seems a mix of data type and its format (encoding). Mult column without reading its description seem to represent quantity, but actually represents if it must be passed or not.	Separation of technical agnostic and Rest design might confuse.	Separate Encoding to two columns as data type and specific format if it is optional. Data type could have finite list of possible values. Multi column could just use Mandatory and Optional values as it would make more sense.	**Table 5-3 – Information input for Upload interface**
SGEO-009	Bool fields does not follow common naming convention.	Not following general practises.	Name bool fields with prefix is. For example, IsDataEncrypted, IsDataCompressed, IsFromActiveSubscription. Name DataProtection is ambitious, as signing data could also mean data protection in place.	**Table 5-4 – Information input for Upload interface**
SGEO-010	compressionFlag is not mandatory.	Not clear what state is indicated when it is not passed and why it should not be passed at all as it is part of metadata.	Change property to mandatory so it would represent clear state. Or align it with dataProtection flag so it would be consistent that such fields might not be passed and that state would mean same thing.	**Table 5-5 – Information input for Upload interface**
SGEO-011	DigitalSignatureValue Object sounds ambigious when inside it has attribute DigitalSignature.	General naming rules.	Could be renamed to DigitalSignatureModel/DigitalSignatureObject and could have attribute Value, that would store actual signature value.	**Table 5-6 – Information input for Upload interface**
SGEO-012	Attribute digitalSignatureReference	General naming rules.	Completely remove field, as algorithm can be resolved from public certificate.	**Table 5-7 – Information input for Upload interface**
SGEO-013	Custom error model	Not using general practise.	Use rfc standard: https://tools.ietf.org/html/rfc7807	
SGEO-014	Error code as enum.	Poor extensibility.	Use string for error code as it would be easier to extend possible error codes without	**Table 5-8 – Enumerations for**

			breaking the contract. Consider if different implemetations might have different error codes or wider range of them (when would come back from VendorApi for example).	**Upload interface**
SGEO-015	Resource in singular form.	Not following common resource naming practise for REST	Use resources in plural form in url.	**Table 5-9 - REST implementation of Upload**
SGEO-016	Successful response is 200 with error response model to indicate success.	Not using appropriate HTTP code	If request was successful, it should response with 204 as no content is intended to be returned.	**Table 5-10 – HTTP Response codes and message in response object**
SGEO-017	Missing 401 http code.	Not using appropriate HTTP code	If user is not authenticated, it should return 401.	**Table 5-11 – HTTP Response codes and message in response object**
SGEO-018	Status code 500 is used to indicate bad request.	Not using appropriate HTTP code and good API practices	For errors that is caused by request validation and client can fix them by altering request, 400 (BadRequest) should be returned with error message describing how to fix it.	**Table 5-12 – HTTP Response codes and message in response object**
SGEO-019	Status code 403 is described incorrectly and reffers to No erros status code.	Might be confusing with 401.	Associate 403 status code with forbidden message also applying correct error code if needed.	**Table 5-13 – HTTP Response codes and message in response object**
SGEO-020	Upload and Upload link methods share same ExhangeMetadataObject	Same model is defined in two places, changes must also be applied	If ExchangeMetadaObject should be same in specification scope, then it would be logical to extract it to separate section and do not define it	**Table 5-14 - Information input for**

		in two places in document	fully each time it should be used with method.	**Upload Link interface**
SGEO-021	Set of HTTP codes differs from one, used in Upload method.	Lack of consistency. Not clear if http code shall not be returned or just missing.	Align Http codes in different operations. If some Http codes is more useful in whole service scope (same meaning through all operations), it could be extracted to one place and reused through whole documentation.	**Table 5-15 – HTTP Response codes and message in response object**
SGEO-021	Extensive use of prefix Envelope when model name already indicates that ir is an envelope.	Longer names increases payload size in Json.	Envelope prefix could be removed from envelope models fields naming.	**Table 16 – Information input for Upload interface**
SGEO-022	AcknowledgmentObject inconsistent naming. In some cases we are using abrevation ack, in somes cases full word acknowledgment.	Lack of consistency.	Decide between using either abbreviation or full term.	**Table 17 - Information input for Acknowledgment interface**
SGEO-023	Acknowledgment response is limited by NackTypeEnum.	It suppress the ability to correctly inform about error. It is poorly extendable and too generic.	As acknowledgment is just delayed actual HTTP response, it should inform about error in response with details what went wrong, so it would be possible to solve the issue. https://tools.ietf.org/html/rfc7807 is already mentioned in SGEO-013. It could be also user with acknowledgment model. For example, it could define a custom error model, which could contain: error code, documentation link, description. Lets keep in mind, that acknowledgment should inform caller about end application behaviour. That means, that errors that you want to send, should also be end application specific, so that user could solve the error and repeat the request. If behind the secom there will be all kind of end applications	**Table 18 - Information input for Acknowledgment interface**

			with their own possible errors, then we need to allow them to return original detailed error to the caller.
			
			Example:
			
			{
			"envelope": {
			"createdAt": "2021-04-28T13:25:49.788Z",
			"envelopeCertificate": "string",
			"envelopeRootCertificateThumbprint": "string",
			"transactionIdentifier": "3fa85f64-5717-4562-b3fc-2c963f66afa6",
			"ackType": 0,
			"isSuccess"= false,
			"errorDetails": {
			"code": "ECDIS1568",
			"documentationLink": https://ECDIS.COM/docs/errors/ECDIS1568,
			"detail": "Empty exchange set."
			},
			"envelopeSignatureTime": "2021-04-28T13:25:49.788Z"
			},
			"digitalSignature": "string"
			}
			
			Additionally we could add isSuccess field, so it would be clear if error occured, and then there is no need to have Error value for AckType.	

SGEO-024	Capability could be expressed through hypermedia	More self explanatory way for api to inform its user about possible actions.	REST design have one feature - HATEOAS, that is often forgotten or just ignored due the complexity it adds. In general HATEOAS allows not only to respond with data, but also provide next possible actions. The idea would be, to dynamically expose possible steps with each data type.	**Service interface - Capability**

			For this example, I will call data types as products and will keep example as minimal as possible (more info could be added). We start with endpoint, that would return all possible product types for Secom instance:
			
			Request
			GET: V1/products
			
			Response:
			[
			{
			"type": "RTZ",
			"links": [
			{
			"object": "POST: /v1/object"
			},
			{
			"object": "POST: /v1/object/link"
			},
			{
			"object": "GET: /v1/object"
			},
			{
			"object": "GET: /v1/object/link"
			},
			
			]
			}
			]
			
			In this way client would see, what possible actions could be performed with product type. Also it matters if we want to put product type to url ('POST: /v1/products/rtz/object' with this example, products would became root endpoint, from where you can see all other possible actions).
			In other endpoints, we could also add links with possible	

			further steps accordingly to access rights, that user posses. This idea is worth further elaboration, if Secom wants to expose its capability in dynamic way and it would be not a custom solution, but well known REST specification.	
SGEO-025	Get Summary endpoint defines a lot of query parameters with not fixed expected size. (Get interface exposes same risk)	Query parameters might exceed allowed url length.	It should be consider, either method should be changed to POST and resource changed to action, so query parameters would be sent through payload in request model. Or size limitation should be defined for string parameters such as geometry.	**Service interface - Get Summary**
SGEO-026	String and CharacterString encodings is mentioned.	Confusing definitions	Could misslead that it means different ways to represent strings. Encodings (or data types) should be alligned in document, so it would not cause confusion behind it actual meanings.	**Table 19 - REST implementation of Encryption Key**
SGEO-027	SECOM does not define identity structure to identify caller.	Unclear how identity should be resolved from client certificate.	When client certificate is received, which is from trusted CA it is unclear how actual identity of certificate holder should be resolved. If we state, that SECOM does not strictly define PKI structure or define valid PKI providers, then resolving actual identity might be interpretated differently in each SECOM instance, what might cause incompatabilities or might add manual procedures and delay to requests processing. From practical perspective, it would be helpful if SECOM either would define expected format and quantity of information that is necessary to be in certificate or either define such information retrieval and its structure through service discovery service. So it would be possible to identify actual	**General Description of SECOM**

			caller dynamically in unified manner (for example such information as company identifier or ship IMO or some other publicly accessible identifier of a caller).	

1.2.2 Test Case 1 Data protection (signing) of unclassified data

1.2.2.1 Results, observations, discussions

Based on questions in Test Cases

1.2.2.2 Test data and commands

1.2.2.3 Conclusions and recommendations to SECOM WG

1.2.3 Test Case n