**Requirements - Multi Cloud encryption of data at rest**

This document lists out the requirements related to the data being encrypted at rest, in SODA, for the following backend object storage types

* 1. AWS S3 <https://aws.amazon.com/s3/>
  2. GCP <https://cloud.google.com/storage/docs/json_api/v1/objects>
  3. Azure Blob storage <https://docs.microsoft.com/en-us/azure/storage/blobs/storage-blobs-introduction>
  4. HWS <https://intl.huaweicloud.com/en-us/product/obs.html>
  5. CEPH <https://docs.ceph.com/docs/mimic/radosgw/>
  6. YIG <https://github.com/journeymidnight/yig>

Today, most of the cloud object storage providers that are S3 compatible, chiefly support 3 types of encryption on the server side, of client data at rest (give or take a few exceptions to this subset among different storage providers, listed in the matrix below)

1. Encryption with server managed keys, which we abbreviate to **SSE\_SMK**
2. Encryption with customer provided keys, abbreviated to **SSE\_CPK**
3. Encryption with KMS (example of a KMS [here](https://docs.aws.amazon.com/kms/latest/developerguide/overview.html)) provided keys, abbreviated to **SSE\_KMS**. In this option too, the key is provided by the customer, but stored in a key vault instead of at the client side

|  |  |  |  |
| --- | --- | --- | --- |
|  | **SSE\_SMK** | **SSE\_CPK** | **SSE\_KMS** |
| AWS S3 | ✅ | ✅ | ✅ |
| GCP | ✅ | ✅ | ✅ |
| Azure | ✅ |  | ✅ |
| HWS | ✅ |  | ✅ |
| CEPH S3 |  | ✅ | ✅ |
| YIG | ✅ | ✅ |  |

**Some observations and a proposal/premise after initial analysis**

1. SSE\_SMK



* 1. The encryption is handled on the server, with symmetric key AES 256 cipher used for encryption
  2. Since it is [symmetric key encryption](https://www.cryptomathic.com/news-events/blog/symmetric-key-encryption-why-where-and-how-its-used-in-banking), the same key is used to encrypt and decrypt the data
  3. The key itself is also further encrypted, decrypted and used as needed, on the server side, without any explicit effort on the user side
  4. On most providers, this is enabled by default on a bucket
  5. A typical upload sequence would work like
     1. Upload object
     2. Use Key to encrypt data (NOT the metadata)
     3. Encrypt and store encrypted key alongside encrypted data
  6. Further, some providers also support bucket level policies on encrypted buckets like
     1. Do not allow non-encrypted objects
     2. Do not allow PUT requests that do NOT request encryption for the object

and so on…

1. SSE\_CPK



* 1. User sends the key to be used for encryption, during upload
  2. During a retrieve, the same key needs to be sent
  3. A typical upload sequence would be like
     1. Upload data
     2. Retrieve encryption algorithm, key and checksum from https header
     3. Use key and algorithm to encrypt data
     4. Store encrypted data
  4. The key is managed on the client side
     1. encrypted
     2. stored in a secure secure manner as long as needed
     3. sent over https to server

1. SSE\_KMS



* 1. User uses the KMS to manage the master keys
  2. During an upload, the KMS and the specific key id of the key in the KMS can be specified to be used for encryption
  3. Typically, the cloud provider supports their own KMS/some specific KMS, no interoperability between the KMS’es
  4. With this, the client side management of the keys is not needed, since the KMS is used

**Proposal**

SODA to support the SSE\_SMK method in the DAITO version, with more variants being added later

Why ?

1. No vendor specific KMS dependencies
2. SMK is more secure, supported by all backend providers, except CEPH S3 (it says SMK is supported for testing only <https://docs.ceph.com/docs/master/radosgw/encryption/>)
3. In order for CEPH to work with SMK, SODA will use a single master key with the SSE\_CPK mode of CEPH. To the end user, it will look like SSE\_SMK. We are digressing into some design here, which is inevitable in this case, imho. This is shown below



1. For the other providers, the SSE\_SMK approach will work with none/minimal complexity from SODA side and user side