June 2021

AUTHENTICATED ISOLATION IN DEDUPLICATED STORAGE

Sandip Agarwala

Shravan Gaonkar

Sandeep Kumar

AUTHENTICATED ISOLATION IN DEDUPLICATED STORAGE

AUTHORS:
Sandip Agarwala
Shravan Gaonkar
Sandeep Kumar

ABSTRACT

Multi-tenancy, security, and deduplication are important features for distributed storage systems for either on premise or on cloud deployments. However, implementing these features can be difficult, as the features often do not play well together when building a performant system. Presented herein is a technique to provide a seamless, transparent deduplication system that incorporates all of these features without inter-dependence and without impacting performance.

DETAILED DESCRIPTION

Most primary and backup storage systems provided for distributed and cloud-based systems implement deduplication of content for space efficiency. In addition, certain compliance requirement or features might not allow for deduplicated data to comingle between different parties.

For example, in some instances customers may ask that their data is not cross-shared with others. This could be an example of a compliance requirement. In another example, data that is stored encrypted may not be allowed to be deduplicated with unencrypted data. If the keys for an encrypted data store are revoked, then data shared with an unencrypted data store would now be inaccessible, which may cause an unavailability issue. To address deduplication for these types of scenarios, a typical approach would be to instantiate multiple deduplication engines to handle each of the isolation requests. However, there is an overhead cost involved for managing multiple isolation requirements

Consider for an example, that a deduplication database typically maps the hash (e.g., Secure Hash Algorithm 256-bit (SHA256)) of an existing content to its location. As shown

in the Figure 1 below, a deduplication database could be distributed such that any node may query for an existing element. The uniqueness of the content is usually computed using a hash (e.g., SHA256) and used as the key.

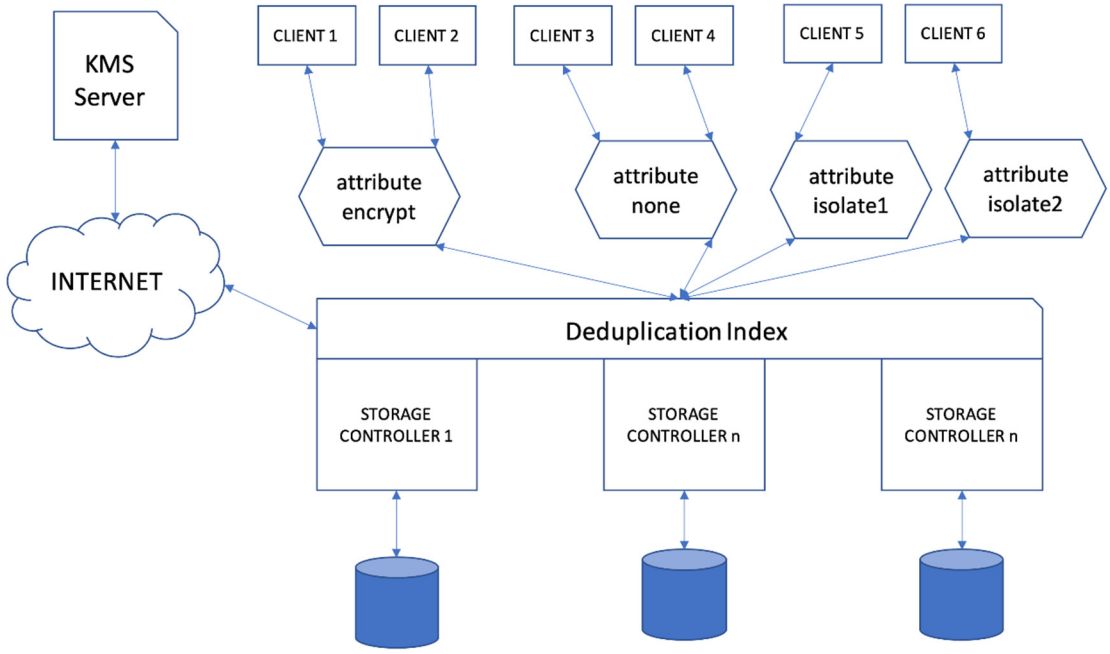


Figure 1: Example Multi-Tenant Deduplication System

Consider an example in which three tenants or clients (say, clients 5, 1, and 2) may generate content 'a' that is to be written such that the 256 hash of the content would be as shown in TABLE 1, below, which would be deduplicated into one block because the hash for each client is identical.

TABLE 1: Example Content Hash for Multiple Clients

Cl.	Cont.	Hash(256)
5	'a'	CA978112CA1BBDCAFAC231B39A23DC4DA786EFF8147C4E72B9807785AFEE48BB
1	'a'	CA978112CA1BBDCAFAC231B39A23DC4DA786EFF8147C4E72B9807785AFEE48BB
2	'a'	CA978112CA1BBDCAFAC231B39A23DC4DA786EFF8147C4E72B9807785AFEE48BB

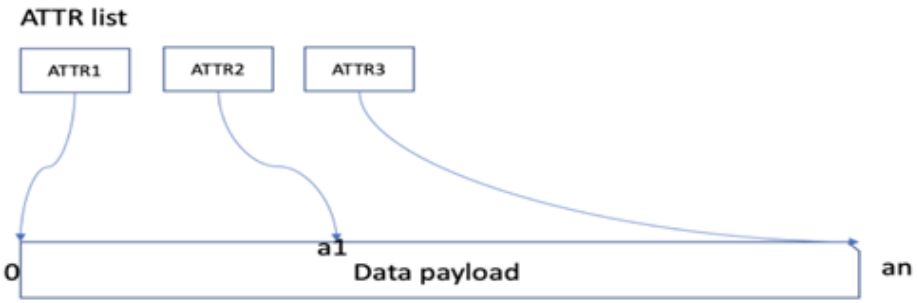
In another example, consider that clients 1, 5, and 3 for the system of Figure 1 seek to write a block with content 'a'. For this example, when inserting into the deduplication index, the SHA generated for all three clients would be the same Hash(a) (as shown in TABLE 1, above). However, it is desirable to avoid such an outcome as this would cause

the data to be deduplicated the same across all clients, which may allow an adversary to use client to peek into the data of client 1 and client 5.

However, if the clients want to avoid such an outcome and demand that their data is not shared with others in the large system, one simple approach would be to just encrypt the content with a client's key, but such an approach will incur an additional penalty of encryption and decryption for every block that is written into the system.

Presented herein is a technique through which an isolation property or attribute can be appended or embedded within a deduplication key itself as an attribute, thereby salting the hash. The isolation property can be agnostic to a given deduplication index engine. In this manner, multi-tenancy can be supported without having to develop a deduplication engine that is aware of different requirements. Furthermore, in some instances the attribute can be secured using a client's encryption key so that the attribute cannot be tampered with while generating hashes for deduplication. As discussed in further detail below, this salting can be implemented through an attribute framework in order to enable isolation at a minimized cost.

For the salting technique, a hash with a salt could be generated for each client that may be serviced within a system by inserting an attribute string to any location of a data payload, as shown in Figure 2, below.




```
ctx = ctx_init()
SHA256(ctx, attr1); SHA256(ctx,data[0..a1]);
SHA256(ctx, attr2); SHA256(ctx,data[a1+1..an]);
SHA256(ctx, attr3);
```

Figure 2: Example Details for Salting Data Payload

The salt for each attribute can be generated when a client is instantiated and the attributes can protected using the same key management server and encrypted. Similar to data that is encrypted and protected at rest, these attributes are a secret, known only to the storage system. The attributes can also specify how and where they should be applied. The only requirement is that the attribute salting is applied uniformly for all data payloads coming into and going out of the system.

For example, consider a single attribute that defines the isolation parameters. The decrypted salt after a client is authenticated could be strings as shown below in which encryption is applied for client 1 and client 2 ("encrypt"), no encryption is provided for clients 3 and 4, and so on. For each write (here, just "a") that comes in for a particular client, the attribute is salted as follows, generating different hashes for the deduplication engine:

Client 1, 2 --> Hash(a + "encrypt") -->H1

Client 3, 4 --> Hash(a) --> H

Client 5 --> Hash(a + "isolate1") -->H3

Client 6 --> Hash(a + "isolate2") -->H3

It should be noted that the probability of a collision for SHA256 is 1^128, which is very unlikely to occur and, thus, provides a large space within which the system may operate.

With the transformation involving the above example, the deduplication index will have 4 keys and is agnostic to the properties embedded into the keys. Thus, a scaled out deduplication engine can be provided, while being agnostic. As for the example provided, a table of salts can generated for each hash, as shown in TABLE 2, below, by appending the attribute to the end of the content.

TABLE 2: Example Content Hash with an Attribute Added to the End of the Content

Cl.	Cont.	Attribute	Hash(256)
1,2	'a'	encrypt	F2D400BF6A98C0682833156C985137D5F1C2BA6B1B631EB554A6565F96DAF7D0
3,4	'a'		CA978112CA1BBDCAFAC231B39A23DC4DA786EFF8147C4E72B9807785AFEE48BB
5	'a'	isolate1	690944280EA581066682516191597 3B1F959E183C7BEC5C02E317AD57F1C8CC7
6	'a'	isolate2	9E508632DE8754533942AAD2DAC930240704899468383C118588BA3339CE12C6

In one instance, the salting technique could be extended to encompass hierarchical policies using attributes in which attribute policies could be hierarchical in nature, thereby allowing clients to inherit certain common attributes. In one example, for instance, policies may be provided to allow clients 1 and 2 to deduplicate against clients 3 and 4 but no clients may be allowed to deduplicate data from clients 1 and 2. Such an attributes could be presented as:

Client 1, 2 {None, Read, "encrypt," Write}
Client 3, 4 {None, Read/Write}

Various advantages may be realized using the salting technique of this proposal. One advantage, for example, may be the ability to provide isolation between tenants without expensive encryption and decryption or techniques such as convergent encryption (e.g., clients 1 and 2 can share the content, while clients 3 and 4 can share content amongst themselves). Another advantage is that the salting technique presented herein may save central processing unit (CPU) resources by merely involving an additional hashing of the salting attribute. As another advantage, the tenancy requirement can be encoded in the salting technique, making it transparent to the user in which each block is aware of an isolation parameter. Thus, administration policy errors that may occur at a future date will not cause some blocks that were not previously shared to become shared to a new client.