Search Resources & Metadata

Resources require metadata for searching and later to support Attribute-Based Access Control for granting access to resources based upon a user's attributes

This is a problem for the project as the current PyOpenABE bindings produce single .cpabe files that do not match the format of the CLI tool. This is further complicated by the .cpabe files being produced in a base64 encoding that even when decoded is illegible and encrypted. Many efforts were made to identify and extract metadata from said ciphertext with mixed results.
In this case, the metadata of the resources refers to the metadata that is already embedded into the ciphertext, which allows for the proper decryption of the resources. This metadata includes the policy with which the resource was encrypted with (and thus, the policy must also resolve in order to allow decryption), the filename, file size, mime type, extension, author etc. Overall, it was determined that even the strongest methods were too circumstantial in execution to be reliable methods for extracting the relevant metadata directly from the ciphertext files. As such an alternative solution is required to extract and collect metadata from the encrypted resources, in order to allow for ABAC implementation and searching of the uploaded resources.

The below options were drafted and considered as possible solutions to this problem.

1. Each resource has a sister, auto-generated metadata file that it is directly linked to. This file contains all the relevant metadata in plaintext for simple processing. Metadata such as policy, title, author, mime type and file extension are examples of what might be stored in said meta files.

2. Alter the output files of the PyOpenABE encryption process such that the resulting file prefaces the ciphertext with plaintext metadata. This would require altering the decryption process as well, since PyOpenABE would not be able to understand files with plaintext prefaces. Additionally, this process would make the resulting encrypted files incompatible with other versions of PyOpenABE and the OpenABE C library.

3. Have the Resource Server store metadata of uploaded files directly into a DB. This would mean that any files stored would not have to have metadata sister files stored as well. However, since the metadata would only be stored in the DB on the Resource Server, said DB would be a risky asset to keep protected, since regular backups would be required in order to maintain the metadata dataset.

4. Combination of options 1. & 3., with slight alterations. Where each uploaded, encrypted file has a sister metadata file (as in 1.) but for simple, more efficient querying of resource metadata, during the upload process, the Resource Server also stores a copy of the metadata in a DB (similar to what is described in 3.).

The final decision for the project was for option 4. since it offers the most portability for the resources whilst also granting proper access to metadata in order for ABAC to be implemented, along with a Searching tool.

With this option, the upload process for the Resource Server will be altered to either receive metadata within the network request from the client alongside the uploaded file or alternatively, extract the metadata from the uploaded file during processing.
Either scenario will allow for the metadata to then be stored within a DB for easy access. This ensures that all file metadata can instantly be queried, without a laborious process of extracting metadata from the file tree. However, should an issue arise with the DB, then the metadata could be extracted from scratch from the metadata files in a single, long crawling process. Rebuilding the DB dataset.

Additionally, during upload, the encrypted files will be uniquely renamed to a GUID such as `f6340f64-efab-486f-a80b-f4ebc40a7199.cpabe`. This will ensure there are no conflicts with filenames and also guarantees that an end user cannot upload a malicious filename to affect the file system. When the metadata file is generated and created in tandem, it will be appropriately named `f6340f64-efab-486f-a80b-f4ebc40a7199.meta` to match the ciphertext file.
Thus within the DB, the file can properly be identified by its filename that the author gave it due to the pairing of the GUID to the original filename. Further, in the case of a DB reset, said GUID-to-filename pairings can be recreated from the `.meta` files.
This GUID naming scheme, also offers limited 'security through obscurity' since a file cannot be as easily identified without either querying the DB or reading all `.meta` files for filename data.