Symmetric Searchable Encryption component

API specification version 0.6.0

Component name: SSE – Symmetric Searchable Encryption

Component deployment name: sse

Changelog

Version	Date	Pages	Author	Modification
0.1	26/2/2020	2	Hai-Van Dang	Initial release with Javascript APIs for upload and search data
0.2	15/4/2020	4	Hai-Van Dang	Update response of uploadData function Add specification for updateData function
0.3	5/5/2020	6	Hai-Van Dang	Add specification for deleteData function, uploadKeyG function, change parameter names of previous functions
0.4	7/8/2020	9	Hai-Van Dang	Add functions encryptBlob, encryptUploadBlob, encryptUploadSearchableBlob, decryptBlob, downloadDecryptBlob, decryptSaveBlob
0.5	26/11/2020	10	Hai-Van Dang	Change APIs to support multiple keys
0.6	15/12/2020	13	Hai-Van Dang	Update functions encryptUploadBlob, encryptUploadSearchableBlob, downloadDecryptBlob to support multiple keys Add functions encryptProgressUploadBlob, encryptProgressUploadSearchableBlob, downloadProgressDecryptBlob to support large files (tested up to 800MB)

Table of Contents

Terminology

Terminology/ Abbreviation	Explanation
End-user	User who uploads/ searches data
SSE server	Server which stores encrypted data
Trusted Authority	Server which stores metadata necessary for upload/ search encrypted data

Introduction

This API specification covers APIs relevant to uploading, search, and update data, which are implemented in Javascript. Data upload is the process that a user chooses to send data, i.e. Json object, to SSE server in cloud. Data search is the process when a user wishes to search for the stored encrypted data in SSE server by providing a Json object by template. Data update is the process when a user wishes to update values of the whole or part of a stored Json object.

The following section describes specification of the Javascript library functionalities which supports the above processes.

API description

Upload data: function

uploadData(data,file_id,pwd1,pwd2,keyid)

This API allows a user to encrypt a Json object, then send its ciphertext to SSE server. It currently supports the following Json object format:

- JSON objects are surrounded by curly braces {}.
- JSON objects are written in key/value pairs.
- Keys must be strings, and values can be string, number. It does not support array and object type for values.
- Keys and values do not contain the vertical slash symbol, i.e. "|" (Because the vertical slash is used as string denominator in the implementation).
- Keys and values are case-sensitive
- Keys and values are separated by a colon.
- Each key/value pair is separated by a comma.

Parameters:

Name	Type	Description
data	Json object	Data to be uploaded, which is a JSON object
file_id	String	File identifier, which must be unique string
pwd1	String	Passphrase which is used to compute a shared symmetric key of users and TA
pwd2	string	Passphrase which is used to compute symmetric encryption key for users. This passphrase is needed to upload/ search/ update/ delete data
keyid	string	Key identification, which identifies a unique pair of (pwd1,pwd2)

Hash computation of pwd1 and pwd2 are different. Therefore, client can use the same value for pwd1 and pwd2 while TA still does not learn anything about their values and symmetric encryption key.

Example:

```
uploadData({firstname: "David", lastname: "White", age:
25},"id1","pwd1","pwd2","keyid1")
```

Response

Returned type	Description
Boolean value	True if uploaded successfully False if failed to upload data

Search data: **function** search(data,pwd1,pwd2,keyid)

Search for encrypted data in SSE server by providing a search content. SSE server will return encrypted files which contain the searched keyword.
The search content is a Json object, which follows the following format:

- JSON objects are surrounded by curly braces {}.
- JSON objects are written in one key/value pair.
- Key is "keyword", and value is a combined string of an attribute and its value separated by the vertical slash symbol, i.e. "|". For instance, if a user wishes to search for "firstname=David", the value will be "firstname|David".
- Key and value are case-sensitive

Example:

```
{
    "keyword": "firstname|David"
}
```

Parameters:

Name	Type	Description
data	Json	Searched data, which is a Json object
pwd1	String	Passphrase which is used to compute a shared symmetric key of users and TA
pwd2	String	Passphrase which is used to compute symmetric encryption key for users. This passphrase is needed to upload/ search/ update/ delete data
keyid	string	Key identification, which identifies a unique pair of (pwd1,pwd2)

Example:
`search({"keyword": "firstname|David"}, "pwd1", "pwd2","keyid1")` to search for firstname=David over data uploaded with passphrases identify by "keyid1".

Response

Returned type	Description
Json object	Json object contains the number of found objects, and their content. {count: <number of found objects>, objects: <Array of Json objects, which contain decrypted data>}

Update data: **function** updateData(data,file_id,pwd1,pwd2,keyid)

This API allows a user to update the whole or part of a Json object which is identified by its file_id. It currently supports the following Json object format:

- JSON objects are surrounded by curly braces {}.
- JSON objects are written in key/value pairs.
- Keys must be strings, and values are arrays of size two. The first item of the array is the current value of the corresponding key, and the second item is the update value. The two items are separated by comma.
- Keys and values do not contain the vertical slash symbol, i.e. "|" (Because the vertical slash is used as string denominator in the implementation).
- Keys and values are case-sensitive
- Keys and values are separated by a colon.
- Each key/value pair is separated by a comma.

Example:

```
{
    "firstname":["David","Peter"],
    "lastname":["White","Yellow"]
}
```

Parameters:

Name	Type	Description
data	Json	Update data, which is a Json object
file_id	String	File identifier, which must be unique string
pwd1	String	Passphrase which is used to compute a shared symmetric key of users and TA
pwd2	String	Passphrase which is used to compute symmetric encryption key for users. This passphrase is needed to upload/ search/ update/ delete data
keyid	String	Key identification, which identifies a unique pair of (pwd1,pwd2)

Example:

```
updateData("firstname":["David","Peter"],"lastname":["White","Yellow"],"id1
", "pwd1", "pwd2","keyid1")
```

requests to update firstname from "David" into "Peter", lastname from "White" to "Yellow" of the Json object with "id1" which has been uploaded with passphrases identify by "keyid1".

Response

Returned type	Description
Boolean value	True if updated successfully False if failed to update

Delete data: **function** deleteData(file_id,pwd1,pwd2,keyid)

This API allows a user to delete a Json object which is identified by its file_id.

Parameters:

Name	Type	Description
file_id	String	File identifier, which must be unique string
pwd1	String	Passphrase which is used to compute a shared symmetric key of users and TA
pwd2	String	Passphrase which is used to compute symmetric encryption key for users. This passphrase is needed to upload/ search/ update/ delete data
keyid	String	Key identification, which identifies a unique pair of (pwd1,pwd2)

Example:
`updateData("id1", "pwd1", "pwd2","keyid1")`
requests to delete the Json object with "id1" which has been uploaded with passphrases identify by "keyid1".

Response

Returned type	Description
Boolean value	True if deleted successfully False if the provided file_id does not exist

Upload shared key: **function** uploadKeyG(pwd1)

This API allows a user to upload a shared key to Trusted Authority.

Parameters:

Name	Type	Description
pwd1	String	Passphrase which is used to compute a shared symmetric key of users and TA
keyid	string	Key identification, which identifies a pair of (pwd1,pwd2)

Response

Returned type	Description
Boolean value	True

Encrypt blob data (<=10MB): `function encryptBlob(blobData,ftype,pwd)`

This API allows a user to encrypt blob data (<=10MB) using symmetric key.

Parameters:

Name	Type	Description
blobData	blob	Binary Large Object
ftype	string	File type
pwd	String	Passphrase which is used to compute symmetric encryption key for users. This key is then used to encrypt blob data.

Response

Returned type	Description
Promise	A promise to return encrypted blob when the encryption is completed

Encrypt and send blob data (<=10MB): `function encryptUploadBlob(blob,fname, pwd,keyid)`

Assuming that a storage server has been set up with Minio. This API allows a user to encrypt blob data (<=10MB) using symmetric key, then upload it to the storage server.

Parameters:

Name	Type	Description
blob	blob	Binary Large Object
fname	string	Filename
pwd	String	Passphrase which is used to compute symmetric encryption key for users. This key is then used to encrypt blob data.
keyid	String	Key identification

Response

Returned type	Description
Promise	A promise to complete encryption and sending data

Example:
https://gitlab.com/asclepios-project/sseclient/-/blob/master/sse/static/js/main.js#L295

Encrypt blob (<=10MB) and its metadata, and send to server: `function` encryptUploadSearchableBlob(blob,fname,jsonObj,file_id, pwd1, pwd2,keyid)

Assuming that a storage server has been set up with Minio. This API allows a user to encrypt blob data (<=10MB) using symmetric encryption, and its metadata using SSE. After that, it uploads both ciphertext to servers.

Please note that, the function will add filename of blob data to its metadata. This allows user to search over encrypted metadata, which returns found metadata and filename. The user then can use filename to retrieve blob data.

Parameters:

Name	Type	Description
blob	blob	Binary Large Object
fname	string	Filename
jsonObj	json	Metadata
file_id	String	Unique file id
pwd1	String	Passphrase which is used to compute a shared symmetric key of users and TA
pwd2	string	Passphrase which is used to compute symmetric encryption key for users. This key is then used to encrypt blob data and metadata.
keyid	string	Key identification, which identifies the unique pair (pwd1,pwd2)

Response

Returned type	Description
Promise	A promise to complete encryption and sending data

Example:
https://gitlab.com/asclepios-project/sseclient/-/blob/master/sse/static/js/main.js#L328

Decrypt blob data (<=10MB): `function decryptBlob(blobCipher,ftype,pwd)`

This API allows a user to decrypt blob (<=10MB) ciphertext using symmetric key

Parameters:

Name	Type	Description
blobCipher	blob	Binary Large Object, which is ciphertext
ftype	string	filetype
pwd	String	Passphrase which is used to compute symmetric encryption key for users. This key is then used to decrypt blob data.

Response

Returned type	Description
Promise	A promise to complete decryption

Download, decrypt and save blob data: `function downloadDecryptBlob(fname,pwd,keyid)`

This API allows a user to download a blob (<=10MB) ciphertext, then decrypt it using symmetric key and save it as a file.

Parameters:

Name	Type	Description
fname	String	File name (with filetype)
pwd	String	Passphrase which is used to compute symmetric encryption key for users. This key is then used to decrypt blob data.
keyid	String	Key identification

Response
The decrypted blob is saved as a file.

Example:
https://gitlab.com/asclepios-project/sseclient/-/blob/master/sse/static/js/main.js#L255

Decrypt and save blob data (<=10MB): `function` decryptSaveBlob(blob,fname,pwd)

This API allows a user to decrypt blob (<=10MB) ciphertext using symmetric key, then save it as a file.

Parameters:

Name	Type	Description
blob	blob	Binary Large Object, which is ciphertext
fname	string	Filename (filename contains filetype)
pwd	String	Passphrase which is used to compute symmetric encryption key for users. This key is then used to decrypt blob data.

Response

Returned type	Description
Promise	A promise to decrypt and save file to disk

Progressively encrypt and upload a large blob (a promise): `function` encryptProgressBlob(blob,fname,ftype, pwd, keyid)

This API promises a user to encrypt a large blob (tested up to 800MB) using symmetric key as multiple chunks of ciphertext, then upload them to the storage server (Minio server).

Technical approach to encrypt a large blob: A large blob is divided into chunks, size of which is configured as sseConfig.chunk_size. Each chunk is encrypted, and grouped into bulk for uploading as a ciphertext part. The number of chunks grouped in a bulk is configured as sseConfig.no_chunks_per_upload. As a result, in the storage server (Minio server), there are multiple ciphertext parts numbering in sequence, and a meta data file which tells the number of ciphertext parts.

Parameters:

Name	Type	Description

blob	blob	Binary Large Object, which is ciphertext
fname	string	Filename (with filetype)
ftype	string	File type
pwd	String	Passphrase which is used to compute symmetric encryption key for users. This key is then used to encrypt blob data.
keyid	String	Key identification

Response

Returned type	Description
Promise	A promise to encrypt and upload to the storage server

Progressively encrypt and upload large blob (a wrapper function): `function` encryptProgressUploadBlob(blob,fname,pwd,keyid)

This API allows a user to encrypt large blob data (tested up to 800MB) and upload to the storage server (Minio server). This is a wrapper of the function *encryptionProgressBlob*.

Parameters:

Name	Type	Description
blob	blob	Binary Large Object
fname	string	File name
pwd	string	Passphrase which is used to compute symmetric encryption key for users. This key is then used to encrypt blob data.
keyid	string	Key identification

Response

Returned type	Description
message	“Completed encrypting blob. Now send data to server” or an error message

Example:

Progressively encrypt a large blob with its searchable metadata, and upload: `function` encryptProgressUploadSearchableBlob(blob,fname,jsonObj,file_id, pwd1, pwd2,keyid)

Assuming that a storage server has been set up with Minio. This API allows a user to progressively encrypt blob data (tested up to 800MB) using symmetric encryption, and its metadata using SSE. After that, it uploads both ciphertext chunks and ciphertext of metadata to the server.

Please note that, the function will add filename of blob data to its metadata. This allows user to search over encrypted metadata, which returns found metadata and filename. The user then can use filename to retrieve blob data.

Parameters:

Name	Type	Description
blob	blob	Binary Large Object
fname	string	Filename
jsonObj	json	Metadata
file_id	String	Unique file id
pwd1	String	Passphrase which is used to compute a shared symmetric key of users and TA
pwd2	string	Passphrase which is used to compute symmetric encryption key for users. This key is then used to encrypt blob data and metadata.
keyid	string	Key identification, which identifies the unique pair (pwd1,pwd2)

Response

Returned type	Description
Promise	A promise to complete encryption and sending data

Example

Progressive download and decrypt large blob: `function` downloadProgressDecryptBlob(fname,pwd,keyid)

This API allows a user to progressively decrypt a blob (tested up to 800MB) ciphertext using symmetric key, then save it as multiple plaintext files.

Technical approach to decrypt a large blob: Assuming that there exist multiple chunks of ciphertext of the large blob in the storage server. This function downloads each chunk, decrypts, and save as a plaintext part. Finally, the function creates a script which can be used to merge multiple plaintext parts into the whole plaintext. As a result, there are multiple plaintext chunks, and a script file. The user needs to run the script defined in the script file to merge and create the plaintext.

Parameters:

Name	Type	Description
fname	string	Filename (filename contains filetype)
pwd	String	Passphrase which is used to compute symmetric encryption key for users. This key is then used to decrypt blob data.
keyid	String	Key identification

Example