Symmetric Searchable Encryption component

API specification version 0.8.1

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)
0.7	11/3/2021	11	Hai-Van Dang	Delete the functions encryptUploadBlob, encryptUploadSearchableBlob, downloadDecryptBlob Update all other functions (except uploadKeyG) with “iskey” parameter Describe the format of the inputted passphrases/ keys. Update uploadKeyG to accept only a key with the correct format and size Add uploadSSEkeys, getSSEkeys
0.8	16/3/2021	12	Hai-Van Dang	Update search() function to support complex queries with AND/OR
0.8.1	22/3/2021	12	Hai-Van Dang	Update search() function to allow retrieving only file ids (jsonId) when searching

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,verK,encK,keyid,iskey=false)

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
verK	Hex string or string	Key (hex string) or passphrase (arbitrary string) for key generation. The generated key will be shared with SSE TA.
encK	Hex string or string	Key (hex string) or passphrase (arbitrary string) for key generation. The generated key is used to encrypt data at SSE client.
keyid	string	Key identification, which identifies a unique pair of (verK,encK)
iskey	boolean	false (default) if verK, encK are passphrases, true if they are keys

The user needs to use different values for verK and encK, to avoid the SSE TA learns about encK.

Example:

```
uploadData({firstname: "David", lastname: "White", age:
25},"id1","pwd1","pwd2","keyid1")
uploadData({firstname: "David", lastname: "White", age: 25},"id1","
358610db4b113a5763111164e391b5ab2696577f44407f92dfb55581b76b34ce","
ad68f3d6b434b48773f60220c1e48d974d15004c4348efee7cb7b111468da909","keyid1",
true) (in this example, key size of verK and encK are 256 bits; therefore,
their hex string contain 64 hex characters)
```

Response

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

Search data: **function** search(data,verK,encK,keyid,iskey=false,isfe=false)

Search for encrypted data in SSE server by providing a search content. SSE server will return encrypted files which contain the searched keyword or return the list of found file ids (jsonId).

The search content is a Json object, which follows the following format:

- JSON objects are surrounded by curly braces {}.
- Keys and values are case-sensitive
- The 1st key is "keyword", and its value is an array of string. Each string is a combination of an attribute and its value separated by the vertical slash symbol, i.e. "|". For instance, if a user wishes to search based on the two criteria "job=doctor" and "gender=female", the value will be ["job|doctor","gender|female "].
- The 2nd key is “condition”, and its value defines the search condition. The symbol + represents OR, while * represents AND. The number 1,2,… represents the index of the search criterion defined in the list of “keyword”.
- In case of search for a single keyword, the 2nd key/value “condition” is omitted.

Example 1:

```
{
        "keyword": ["gender|female","job|doctor","job|nurse"],
        "condition": "2+(1*3)"
}
```

The number 1 represents the 1st search criterion (gender=female), number 2 be (job=doctor), and number 3 be (job=nurse). Therefore, the search condition is: job=doctor OR (gender=female AND job=nurse).

Example 2:

```
{
        "keyword": "job|doctor"
}
or
{
        "keyword": ["job|doctor"]
}
```

request to search for a single keyword: job=doctor.

Parameters:

Name	Type	Description
data	Json	Searched data, which is a Json object
verK	hex string or string	Key (hex string) or passphrase (arbitrary string) for key generation. The generated key will be shared with SSE TA.
encK	hex string or string	Key (hex string) or passphrase (arbitrary string) for key generation. The generated key is used to encrypt data at SSE client.
keyid	string	Key identification, which identifies a unique pair of (verK,encK)
iskey	boolean	false (default) if verK, encK are passphrases, true if they are keys
isfe	boolean	false (default) if requesting to retrieve data content, true if requesting to retrieve only file ids (jsonId)

Example:

```
search({"keyword": "firstname|David"}, "pwd1", "pwd2","keyid1") to search for firstname=David over data uploaded with passphrases identified by "keyid1".
search({
      "keyword": ["gender|female","job|doctor","job|nurse"],
      "condition": "2+(1*3)"
},"358610db4b113a5763111164e391b5ab2696577f44407f92dfb55581b76b34ce","ad68f3d6b434b48773f60220c1e48d974d15004c4348efee7cb7b111468da909","keyid1",true)
```

Response

Returned type	Description
Json object	If isfe=false, 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>} If isfe=true, Json object contains the number of found objects, and the list of found file ids (jsonId)

Update data: function updateData(data,file_id,verK,encK,keyid,iskey=false)

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
verK	Hex string or String	Key (hex string) or passphrase (arbitrary string) for key generation. The generated key will be shared with SSE TA.
encK	Hex string or String	Key (hex string) or passphrase (arbitrary string) for key generation. The generated key is used to encrypt data at SSE client.
keyid	String	Key identification, which identifies a unique pair of (verK,encK)
iskey	boolean	false (default) if verK, encK are passphrases, true if they are keys

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 identified by "keyid1".

```
updateData("firstname":["David","Peter"],"lastname":["White","Yellow"],"id1
","358610db4b113a5763111164e391b5ab2696577f44407f92dfb55581b76b34ce","ad68f
3d6b434b48773f60220c1e48d974d15004c4348efee7cb7b111468da909","keyid1",true)
```

Response

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

Delete data: function deleteData(file_id,verK,encK,keyid,iskey=false)

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
verK	Hex string or String	Key (hex string) or passphrase (arbitrary string) for key generation. The generated key will be shared with SSE TA.
encK	Hex string or String	Key (hex string) or passphrase (arbitrary string) for key generation. The generated key is used to encrypt data at SSE client.
keyid	String	Key identification, which identifies a unique pair of (verK,encK)
iskey	boolean	false (default) if verK, encK are passphrases, true if they are keys

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

```
updateData("id1",
","358610db4b113a5763111164e391b5ab2696577f44407f92dfb55581b76b34ce","ad68f
3d6b434b48773f60220c1e48d974d15004c4348efee7cb7b111468da909","keyid1",true)
```

Response

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

Upload shared key: **function** uploadKeyG(verK,keyid)

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

Parameters:

Name	Type	Description
verK	Hex string	Key (hex string). The key will be shared with SSE TA.
keyid	string	Key identification, which identifies a pair of (verK,encK)

Response

Returned type	Description
Boolean value	True

Generate a key from a passphrase: `function computeKey(pwdphrase,ista=false){`

This API allows a user to generate a key from a passphrase.

Parameters:

Name	Type	Description
pwdphrase	string	Passphrase
ista	Boolean	true if the key will be shared with SSE TA, false if the key will be used for encryption at SSE client

Response

Returned type	Description
Hex string	Key

Progressively encrypt and upload a large blob (a promise): function encryptProgressBlob(blob,fname,ftype, encK, keyid, iskey=false)

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
encK	Hex string or String	Key (hex string) or passphrase (arbitrary string) for key generation. The generated key is used to encrypt data at SSE client.

keyid	String	Key identification
iskey	boolean	false (default) if encK is a passphrase, true if it is a key

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,encK,keyid,iskey=false)

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
encK	Hex string or String	Key (hex string) or passphrase (arbitrary string) for key generation. The generated key is used to encrypt data at SSE client.
keyid	string	Key identification
iskey	boolean	false (default) if encK is a passphrase, true if it is a key

Response

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

Example:
https://gitlab.com/asclepios-project/sseclient/-/blob/master/sse/static/js/main.js#L215

Progressively encrypt a large blob with its searchable metadata, and upload: function encryptProgressUploadSearchableBlob(blob,fname,jsonObj,file_id, verK, encK,keyid,iskey=false)

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
verK	Hex string or String	Key (hex string) or passphrase (arbitrary string) for key generation. The generated key will be shared with SSE TA.
encK	Hex string or String	Key (hex string) or passphrase (arbitrary string) for key generation. The generated key is used to encrypt data at SSE client.
keyid	string	Key identification, which identifies the unique pair (verK,encK)
iskey	boolean	false (default) if encK is a passphrase, true if it is a key

Response

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

Example:

https://gitlab.com/asclepios-project/sseclient/-/blob/master/sse/static/js/main.js#L265

Progressive download and decrypt large blob: `function` downloadProgressDecryptBlob(fname,encK,keyid,iskey=false)

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)
encK	Hex string or String	Key (hex string) or passphrase (arbitrary string) for key generation. The generated key is used to encrypt data at SSE client.
keyid	String	Key identification
iskey	boolean	false (default) if encK is a passphrase, true if it is a key

Example:
https://gitlab.com/asclepios-project/sseclient/-/blob/master/sse/static/js/main.js#L167

Encrypt and upload SSE keys to KeyTray using CP-ABE service: `function` uploadSSEkeys(verkey,enckey,token){

This API allows a user to encrypt SSE keys with CP-ABE and upload them to KeyTray. The encryption and uploading service are served by the CP-ABE server.

Parameters:

Name	Type	Description
verkey	string	Verification key
enckey	string	Encryption key
Token	String	Access token

Response

Returned type	Description
string	Key identification

Download and decrypt SSE keys from the KeyTray: **function** getSSEkeys(keyid,username,token)

This API allows a user to download SSE keys from KeyTray, and decrypt their CP-ABE ciphertext. This is done by using the service of CP-ABE server.

Parameters:

Name	Type	Description
keyid	string	Key identification
username	String	User name
token	String	Access token

Response

Returned type	Description
Json	Pair of keys: verKey and encKey