Design Document

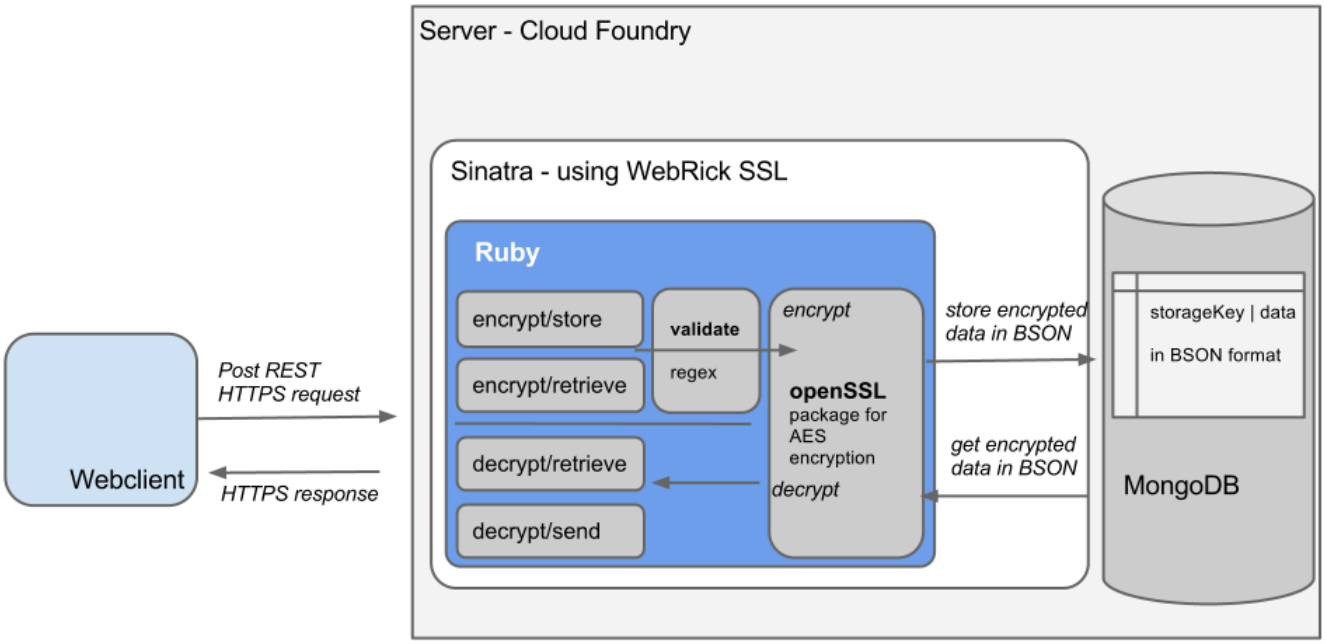
CMPE297

Data Encryption Service

Jesse Jackson, Marcel Kutsch

GitHub: *https://github.com/kutschm/DES.git*

Architecture Diagram



The architectural picture above highlights all major components in our Data Encryption Service. The server part consists of a Sinatra web server configured to use SSL for secure data connections. This server can will also be deployed on the Cloud Foundry platform. The web-server hosts 4 RESTful web-services implemented in RUBY, two for encryption and two for decryption. They are meant to be used as pairs as illustrated in the use case description below. Two internal modules will be deployed that handle input validation as well as the actual data encryption / decryption services. The web-services furthermore persist data in a MongoDB collection in BSON format. This collection associates server computed storageKeys with encrypted user data. The encryption is performed using a user provided symmetrical key, which is not stored on the server therefore only the user him/herself can decrypt the data under the assumption he/she provides the correct passphrase. The symmetric key cipher used is AES.

Use case: **blob/store - blob/read**

- The client calls the https://jmdes.com/blob/store web-service and using a JSON document as payload containing the encryption passphrase, the data to be encrypted and an optional validation regex.
- The web-service first verifies the plain-text data against the validation-regex.
- Once validated, the webservice encrypts the data using the given passphrase
- It then generates a unique storageKey and stores the encrypted data as BSON in the MongoDB collection.
- Finally the web-service returns a JSON document containing the storage key.
- This storage key can then be used for an ensuing web-service call to https://jmdes.com/blob/read. along with the passphrase to get the plain text back from the server.

Use case: **blob/retrieve - blob/send**

- The client calls the https://jmdes.com/blob/retrieve web-service and using a JSON document as payload containing the encryption passphrase, the data to be encrypted and an optional validation regex.
- The web-service first verifies the plain-text data against the validation-regex.
- Once validated, the webservice encrypts the data using the given passphrase
- Finally the web-service returns a JSON document containing the encrypted data. The data is not stored in MongoDB.
- In an ensuing web-service call to https://jmdes.com/blob/send along with the passphrase, a client can ask the server to decrypt the previously retrieved encrypted data. The web-service here will return the decrypted data.

All services involved in these two use-cases can be operated in batch mode, i.e. multiple json records can be added to the single document passed as input to the web-service. All of these will be processed within one web-service call.

Data Encryption Service Rest - API

POST blob/store

Allow a consumer to encrypt sensitive data and store in the secure data store system

It is strongly recommended you use HTTPS for all OAuth authorization steps.

Resource URL:

https://jmdes.com/blob/store

Parameters:

IN - *JSON array containing a single record with the followng keys / value pairs*

passPhrase required	A UTF8 string that represent the encryption passphrase used by the encryption service
blob required	A UTF8 string representation of the data optionally in base64 encoding to be encrypted and stored
validationRegex optional	A regex in UTF8 string representation that is validated against the plain text input text.

OUT- *JSON array containing a single record each with the followng keys / value pairs*

storageKey	A UTF8 string that represent a unique identifier of the store encrypted data. Only if returnCode = 200
returnCode	200 - OK (API call successful) 500 - Error
returnMessage	A UTF8 string that contains an error message in case returnCode > 200

Example Request

POST https://jmdes.com/blob/store
POST Data Format JSON

```
[{    passPhrase:       "secretKey",
      blob:             "text-to-encrypt",
      validationRegex:  "^\d{3}-\d{2}-\d{4}$"
}...]
```

Result Format JSON

```
[{    storageKey:       “uniqueStorageID”,
      returnCode:       200,
      returnMessage:    “”
}...]
```

POST blob/retrieve

Allow a consumer to encrypt sensitive data and retrieve the encrypted data without storing it in the system
It is strongly recommended you use HTTPS for all OAuth authorization steps.

Resource URL:

https://jmdes.com/blobl/retrieve

IN - *JSON array containing a single record with the followng keys / value pairs*

passPhrase required	A UTF8 string that represent the encryption passphrase used by the encryption service
blob required	A UTF8 string representation of the data optionally in base64 encoding to be encrypted and stored
validationRegex optional	A regex in UTF8 string representation that is validated against the plain text input text.

OUT - *JSON array containing a single record each with the followng keys / value pairs*

encryptedBlob	A UTF8 string that contains the encrypted text Only if returnCode = 0
returnCode	200 - OK 500 - Error
returnMessage	A UTF8 string that contains an error message in case returnCode > 200

Example Request

POST https://jmdes.com/blob/retrieve
POST Data Format JSON

```
[{
      passPhrase:       “secretKey”,
      blob:             “text-to-encrypt”,
      validationRegex:  “^\d{3}-\d{2}-\d{4}$”
}...]
```

Result Format JSON

```
[{
      encryptedBlob:    “encryptedBlob”,
      returnCode:       200,
      returnMessage:    “”
}...]
```

POST blob/read

Allow a consumer to retrieve records from the encrypted data store individually the web service returns them decrypted in plain text UTF8 back to the user.
It is strongly recommended you use HTTPS for all OAuth authorization steps.

Resource URL:

https://jmdes.com/blob/read

Parameters:

IN - *JSON array containing records with the followng keys / value pairs*

passPhrase required	A UTF8 string that represent the encryption passphrase used by the encryption service
storageKey required	A UTF8 string that represent a unique identifier of the store encrypted data.

OUT - *JSON array containing records each with the followng keys / value pairs*

storageKey	A UTF8 string representing the unique identifier of an encrypted string returned by an encrypt/store call
blob	A UTF8 string that contains the decrypted plain text of the encrypted text contained in the data store. Only returned if returnCode = 0
returnCode	200 - OK 500 - Error
returnMessage	A UTF8 string that contains an error message in case returnCode > 2000

Example Request

POST https://jmdes.com/blob/read

POST Data Format JSON

```
[{
      passPhrase:       “secretKey”,
      storageKey:       “uniqueStorageID”
}...]
```

Result Format JSON

```
[{
      storageKey:       “uniqueStorageID”
      blob:             “original_text”,
      returnCode:       200,
      returnMessage:    “”
}...]
```

POST blob/send

Allow a consumer to retrieve multiple encrypted records as a batch and decrypt them back to a plain text. It is strongly recommended you use HTTPS for all OAuth authorization steps.

Resource URL:

https://jmdes.com/blob/send

Parameters:

IN - JSON array of records each containing the followng keys / value pairs

passhrase required	A UTF8 string that represent the encryption passphrase used by the encryption service
encryptedBlob	A UTF8 string that contains the encrypted text

OUT - JSON array of records each containing the followng keys

blob	A UTF8 string that contains the decrypted plain text of the encrypted text. Only returned if returnCode =200
returnCode	200 - OK 500 - Error
returnMessage	A UTF8 string that contains an error message in case returnCode > 200

Example Request

POST https://jmdes.com/blob/send

POST Data Format JSON

```
[ {
      passphrase:       “secretKey”,
      encryptedBlob:    “encryptedBlob”,
} ... ]
```

Result Format JSON

```
[ {
      blob:             “original_text”,
      returnCode:       200,
      returnMessage:    “”
} ... ]
```

Samples:

Sample store / read

```
POST blob/store {"passPhrase":"phrase","blob":"text-to-
encrypt","validationRegex":"foo”}
Result:
{"returnCode":200,"storageKey":"508786227506a406e4000001","returnMessage":""}

POST blob/read
{"passPhrase":"phrase","storageKey":"508786227506a406e4000001"}
Result:
{"returnCode":200,"storageKey":"508786227506a406e4000001","returnMessage":"",
"blob":"text to encrypt"}
```

Sample retrieve / send

```
POST blob/retrieve {"passPhrase":"phrase","blob":"text to
encrypt","validationRegex":"foo"}
Result:
{"returnCode":200,"returnMessage":"","encryptedBlob":"Encrypted-text
to encrypt"}
done... blob/retrieve

POST blob/send
{"passPhrase":"phrase","validationRegex":"foo","encryptedBlob":"Encryp
ted-text to encrypt"}
Result: {"returnCode":200,"returnMessage":"","blob":"text to encrypt"}
done... blob/send
```