

Cosmian KMS Source Code Modification - 001

Implement the REST API to Request EnrolData to the KMS/Cosmian

Approach (Methodology)

Cosmian comes up with two methods to handle the client's request.

- **Operation:** Execute a single operation at once
- **Message:** Execute multiple operations at once (Bulk mode)

Selection for the implementation: Message

Code Modifications

Modification 01: Requests Handling Inside the API Endpoint at the First Stage

File directory: kms-develop/crate/server/src/routes/kmip.rs

Made a change to `handle_ttlv` function in `kmip.rs` to handle the specific request.

Source code:

```
async fn handle_ttlv(
    kms: &KMS,
    ttlv: &TTLV,
    user: &str,
    database_params: Option<&ExtraDatabaseParams>,
) -> KResult<TTLV> {
    if ttlv.tag.as_str() == "Message" {
        let req = from_ttlv::<Message>(ttlvl)?;
        let resp = kms.message(req, user, database_params).await?;
        Ok(to_ttlv(&resp)?)
    } else {
        let operation = dispatch(kms, ttlv, user, database_params).await?;
        Ok(to_ttlv(&operation)?)
    }
}
```

Update:

```
async fn handle_ttlv(  
    kms: &KMS,  
    ttlv: &TTLV,  
    user: &str,  
    database_params: Option<&ExtraDatabaseParams>,  
) -> KResult<TTLV> {  
    if [  
        "Message",  
        "GenerateEnrolData",  
        "ReEnrolData",  
        "GetEnrolData",  
    ]  
    .contains(&ttlv.tag.as_str())  
    {  
        /*Note: Deserialize */  
        let req = from_ttlv:::<Message>(ttlvl)?;  
        /*Note: Process the request */  
        let resp = kms.message(req, user, database_params).await?;  
        /*Note: Serialize the response */  
        Ok(to_ttlv(&resp)?)  
    } else {  
        /*Note: Receive the Returning Response */  
        let operation = dispatch(kms, ttlvl, user, database_params).await?;  
        /*Note: Serialize */  
        Ok(to_ttlv(&operation)?)  
    }  
}
```

Modification 02: Public Key UID Generation

File directory: kms-develop/crate/server/src/core/operations/create_key_pair.rs

There is no built-in method to give a specific unique ID to a public key when key pair generation.

The screenshot shows a Discord chat interface with two messages. The first message is from user **salitha** (11/07/2024 4:57 PM) to **@Manuthor**. It discusses creating an RSA key pair and shows the output of the `ckms rsa keys create` command, which generates unique IDs for both public and private keys. It then shows the output of `ckms rsa keys create test_id`, where the private key has a specific ID but the public key ID is not explicitly set. The second message is from user **bébé** (11/08/2024 8:41 PM) to **@salitha**, stating that this functionality is not supported yet and suggesting the use of tags to identify keys. A link to the Cosmian Technical Documentation is provided, specifically the 'Objects Tagging' section, which explains that tags can be used to group objects and find them for operations like export, import, encrypt, and decrypt.

salitha 11/07/2024 4:57 PM
Hi **@Manuthor** ,

We can create an RSA key pair as follows. In this case, unique IDs for private and public keys are generated automatically.

```
ckms rsa keys create  
The RSA key pair has been created.  
Public key unique identifier: 8d90e1f0-820b-478c-82cd-518a9b74fc3e  
Private key unique identifier: 36d9739e-3f29-4ce0-85fd-100f1147b330
```

If we need to add a unique ID to the private key, according to the documentation, we can use this command: In this case, the given ID was taken from the private key, but the unique ID for the public key has been generated automatically.

```
ckms rsa keys create test_id  
The RSA key pair has been created.  
Public key unique identifier: bb4e2728-d2b1-4021-83e0-2ec8f9b48bb  
Private key unique identifier: test_id
```

My question is how can we give a unique ID for the public key as well? Is there any method to do it? I couldn't find it in the documentation.

bébé 11/08/2024 8:41 PM
Hello **@salitha** No. This is not supported (yet). You could tag your keys to find them using tags:
If you create the keypair with a tag, say "tag1", you can then extract the public key using its tags "tag1" and "_pk". The "_pk" tag is automatically added to designate a public key: see this documentation: https://docs.cosmian.com/cosmian_key_management_system/kmip_2_1/tagging/

Objects Tagging - Cosmian Technical Documentation
The Cosmian KMS server supports the tagging of objects. Tags are arbitrary strings that can be attached to objects.
Tags can be used to group objects together, and to find objects for most operations, such as export, import, encrypt, decrypt, etc.

There is a change in key pair generation by adding the functionality to give a specific unique ID with a pk suffix to the private key unique ID. For example, if we have a private key with the unique ID " test key" then the public key unique ID will be "testkey_pk". It will be beneficial when we generate multiple cryptographic objects same time and certify a public key since we have a pre-defined Unique ID.

Source code: (line no. 58)

```
// generate uids and create the key pair and tags
let sk_uid = request
    .private_key_attributes
    .as_ref() // Convert Option to Option reference
    .and_then(|attrs| attrs.unique_identifier.as_ref()) // Safely access unique_identifier
    .map_or_else(
        || Uuid::new_v4().to_string(),
        std::string::ToString::to_string,
    );
let pk_uid = Uuid::new_v4().to_string();
let (key_pair, sk_tags, pk_tags) = generate_key_pair_and_tags(request, &sk_uid, &pk_uid)?;
```

Update:

```
// generate uids and create the key pair and tags
let sk_uid = request
    .private_key_attributes
    .as_ref() // Convert Option to Option reference
    .and_then(|attrs| attrs.unique_identifier.as_ref()) // Safely access unique_identifier
    .map_or_else(
        || Uuid::new_v4().to_string(),
        std::string::ToString::to_string,
    );
// let pk_uid = Uuid::new_v4().to_string();
let pk_uid = format!("{}", sk_uid, "_pk");
let (key_pair, sk_tags, pk_tags) = generate_key_pair_and_tags(request, &sk_uid, &pk_uid)?;
```

JSON Requests Handling with TTLV Messaging Protocol

Three new message requests in TTLV format have been created for the "GenerateEnrolData" "ReEnrolData" and "GetEnrolData" requests. (JSON files attached.)

Code Compilation

Requirement 01: Rust

Run the Actix web application. Specifically, you will need:

- **Rust:** Install it using rustup if it's not already installed.
- **Cargo:** This comes with the Rust installation.

```
sudo apt update
sudo apt install curl
curl --proto '=https' --tlsv1.2 -sSf https://sh.rustup.rs | sh
source $HOME/.cargo/env
cargo --version
```

Requirement 02: cargo-watch (Recommended for Testing Purposes)

To automatically reload and apply changes to your Actix web application in real-time while developing, you can use a tool called cargo-watch. (Recommended for testing purposes)

```
cargo install cargo-watch
cargo watch -x 'run --bin cosmian_kms_server'
```

If you need to run in normal mode (without automatically reloading), the following command can be used.

```
cargo run --bin cosmian_kms_server
```

The server will be exposed to the default port 9998 of the local PC.

```
thakshana@thakshana-L00-151X9:~/Desktop/THAKSHANA/thakshana-cosmian-kms/REQUEST_DATA/RUST/kms-develop$ cargo watch -x 'run --bin cosmian_kms_server'
[Running 'cargo run --bin cosmian_kms_server']
Finished dev profile (unoptimized + debuginfo) target(s) in 0.21s
Running target/debug/cosmian_kms_server
2024-11-11T04:17:18.052707Z INFO ThreadId(01) start: cosmian_kms_server:telemetry: crate/server/src/telemetry/mod.rs:74: Telemetry initialized. Server starting with config {
  db: No database configuration provided, clear database?: false,
  kms_url: http://0.0.0.0:9998,
  workspace: WorkspaceConfig {
    root_data_path: "/cosmian-kms",
    tmp_path: "/tmp",
  },
  default_username: "admin",
  force_default_username: false,
  Google Workspace CSE, KACLS Url: None,
  Microsoft Double Key Encryption URL: None,
  telemetry: TelemetryConfig {
    otlp: None,
    quiet: false,
  },
  info: false,
}
2024-11-11T04:17:18.055977Z INFO ThreadId(01) start: cosmian_kms_server: crate/server/src/main.rs:108: OpenSSL default mode, version: OpenSSL 3.0.2 15 Mar 2022, in OPENSSLDIR: "/usr
/lib/ssl", number: 30000020
2024-11-11T04:17:18.057634Z INFO ThreadId(01) start: cosmian_kms_server:kms_server: crate/server/src/kms_server.rs:76: KMS Server configuration: {
  kms_url: "http://0.0.0.0:9998",
  db_params: Some(
    sqlite: /home/thakshana/Desktop/THAKSHANA/thakshana-cosmian-kms/REQUEST_DATA/RUST/kms-develop/cosmian-kms/sqlite-data,
  ),
  clear_db_on_start: false,
  default_username: "admin",
  force_default_username: false,
  http_params: Http,
  ms_dke_service_url: None,
  api_token_id: None,
}
2024-11-11T04:17:18.069866Z INFO ThreadId(01) start: actix_server:builder: /home/thakshana/.cargo/registry/src/index.crates.io-6f17d22bba15001f/actix-server-2.5.0/src/builder.rs:27
2: starting 12 workers
2024-11-11T04:17:18.069938Z INFO ThreadId(01) start: cosmian_kms_server:kms_server: crate/server/src/kms_server.rs:116: Starting the HTTP KMS server...
2024-11-11T04:17:18.070102Z INFO ThreadId(01) start: actix_server:server: /home/thakshana/.cargo/registry/src/index.crates.io-6f17d22bba15001f/actix-server-2.5.0/src/server.rs:192:
Tokio runtime found; starting in existing Tokio runtime
2024-11-11T04:17:18.070258Z INFO ThreadId(01) start: actix_server:server: /home/thakshana/.cargo/registry/src/index.crates.io-6f17d22bba15001f/actix-server-2.5.0/src/server.rs:197:
starting service: "actix-web-service-0.0.0:9998", workers: 12, listening on: 0.0.0.0:9998
2024-11-11T04:17:31.335087Z INFO ThreadId(16) kmip_2_1: kmip: crate/server/src/routes/kmip.rs:37: POST /kmip. Request: "GenerateEnrolData" admin user="admin" tag="GenerateEnrolData"
2024-11-11T04:19:08.808802Z INFO ThreadId(17) kmip_2_1: kmip: crate/server/src/routes/kmip.rs:37: POST /kmip. Request: "GetEnrolData" admin user="admin" tag="GetEnrolData"
2024-11-11T04:21:39.192710Z INFO ThreadId(18) kmip_2_1: kmip: crate/server/src/routes/kmip.rs:37: POST /kmip. Request: "GenerateEnrolData" admin user="admin" tag="GenerateEnrolData"
2024-11-11T06:06:48.237851Z INFO ThreadId(19) kmip_2_1: kmip: crate/server/src/routes/kmip.rs:37: POST /kmip. Request: "GenerateEnrolData" admin user="admin" tag="GenerateEnrolData"
2024-11-11T06:07:29.560003Z INFO ThreadId(20) kmip_2_1: kmip: crate/server/src/routes/kmip.rs:37: POST /kmip. Request: "GetEnrolData" admin user="admin" tag="GetEnrolData"
```

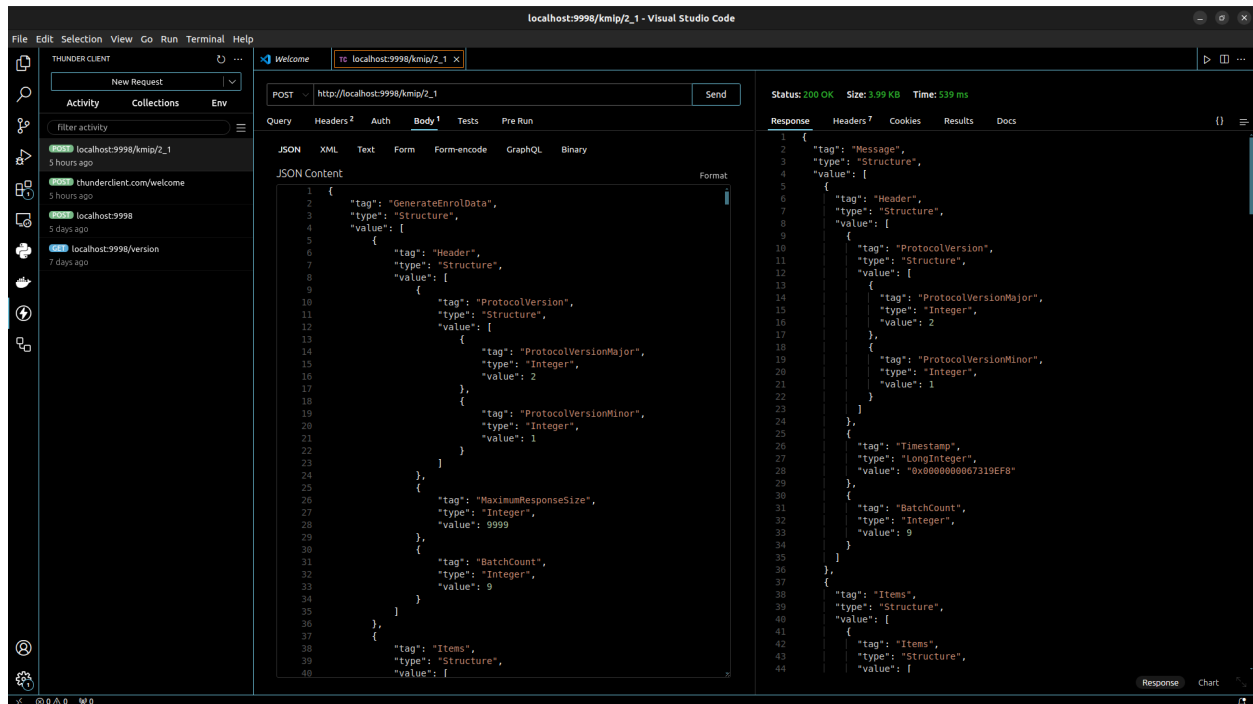
If you want to build a docker image and run it in a container:

```
docker build -t cosmian_kms .
docker run -d -p 9998:9998 --name kms cosmian_kms
```

```
docker start kms
docker stop kms
```

Testing

We can test the API with the specific JSON TTLV POST requests by sending them to the http://localhost:9998/kmip/2_1 endpoint. As an HTTP client, Thunder Client VS Code extension or any other API tester can be used. (Ex: Postman)



Notes

- Each JSON request payload has been converted to align with the TTLV messaging protocol.
- Only the JSON request payloads for the "GenerateEnrolData" and "GetEnrolData" have been generated and the JSON request payload for the "ReEnrolData" should be defined according to the requirement.
- The response JSON data are also attached and we have to manipulate that JSON data for the utilization of the next stage.

-End of the Document-