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:

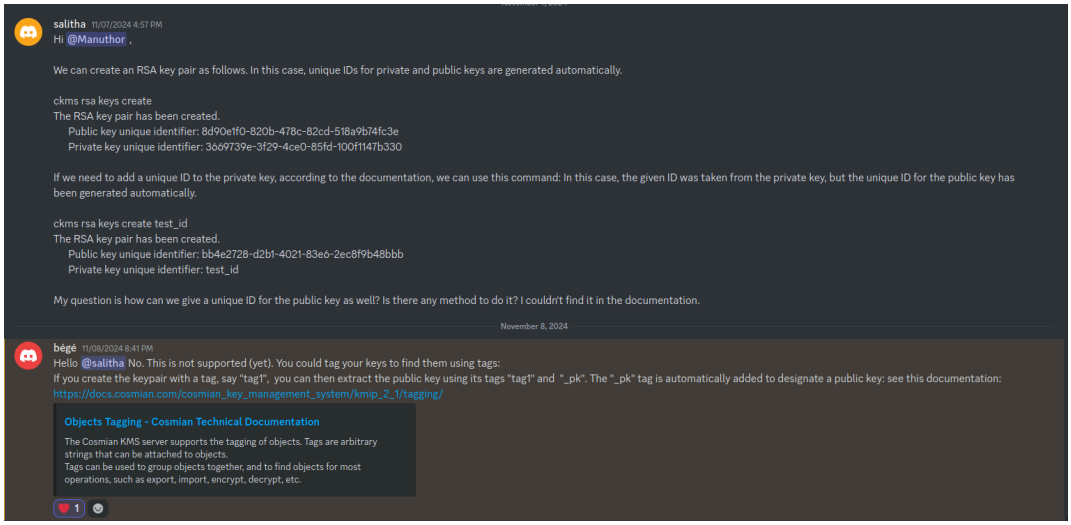
```
async fn handle_ttlv(
   kms: &KMS,
   ttlv: &TTLV,
   user: &str,
   database_params: Option<&ExtraDatabaseParams>,
) -> KResult<TTLV> {
   if ttlv.tag.as_str() == "Message" {
       let req = from_ttlv::<Message>(ttlv)?;
       let resp = kms.message(req, user, database_params).await?;
       Ok(to_ttlv(&resp)?)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

Update:

```rust
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>(ttlv)?;
        /*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, ttlv, 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.



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.

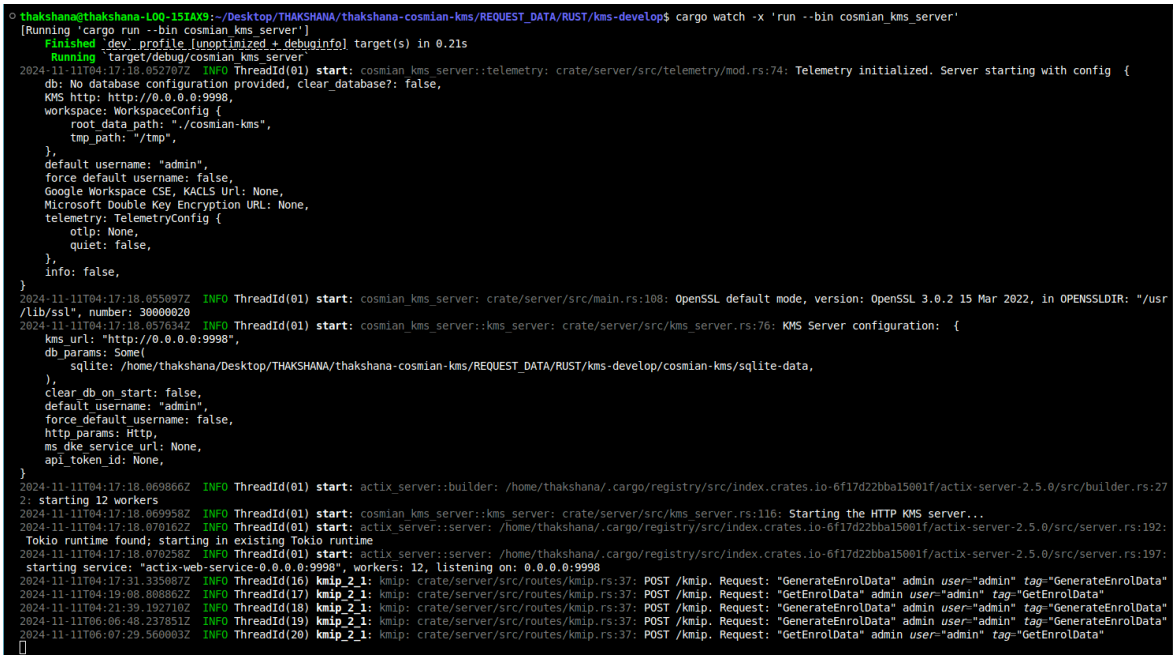
sudo apt update
sudo apt install curl
curl --proto '=https' --tlsv1.2 -sSf https://sh.rustup.rs | sh
source $HOME/.cargo/env
cargo --version

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.



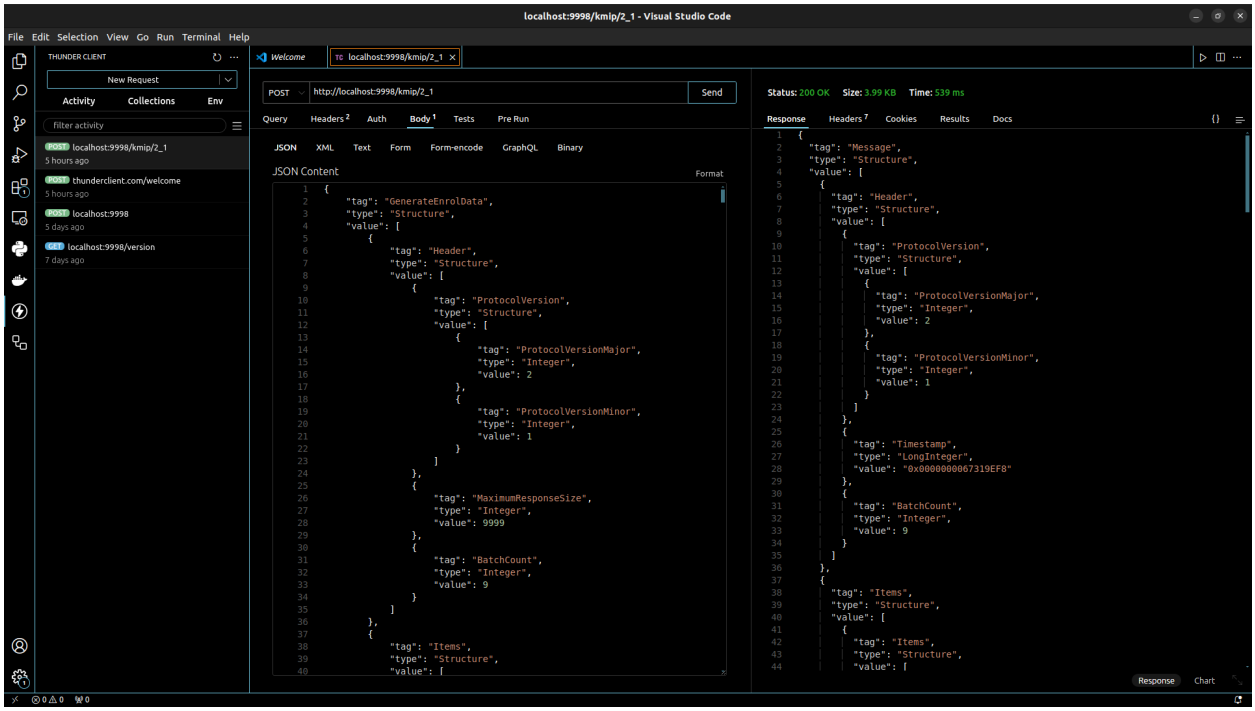
If you want to build a docker image and run it in a container:

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-