

A Modern C++ KMIP Client Based on a Reusable Protocol Library



Oleksiy Lukin
Senior Software Engineer
MySQL team

About myself

- .1982-1988: Kharkiv Aviation Institute, Radio faculty
- .1988-1990 Soviet Army, aviation
- .1990-2010 **Chernihiv Polytech** + IT business, first Internet provider in the region.
- .2011: IT Business: Ethon GmbH in close co-operation with Phil Zimmerman and his team, *VOIP encryprion*; Infopulse: *Distributed encryped data storage system*; Samsung: *cross-platform transparent encryped file system*; FirstBridge: *Blockchain/cryptocurrency*; Eleks: *Retail system*, Qarkus *development*
- . 2024: **Percona, MySQL team**
- .**EasyCrypt** Open source Java library <https://github.com/alukin>

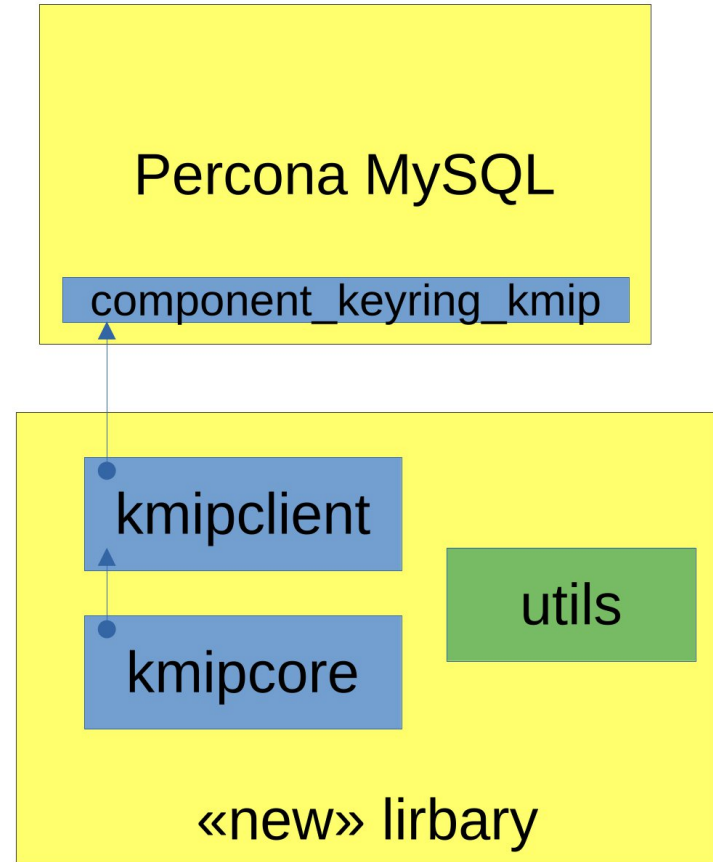
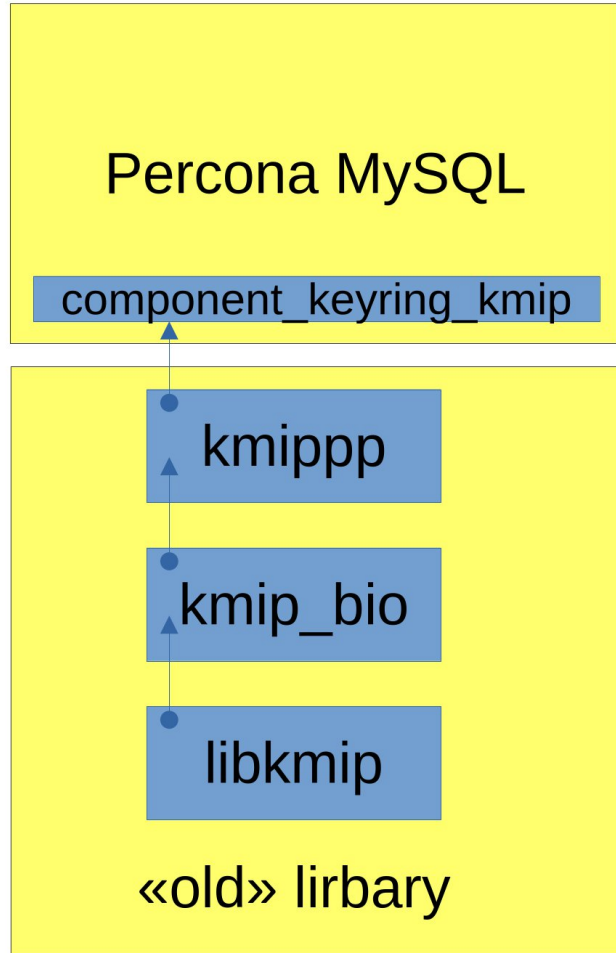
What is KMIP?

- The Key Management Interoperability Protocol (KMIP) provides a standardized interface between applications and key management systems (KMS).
- [OASIS specifications](#): Open standards for interoperability
- KMIP, developed by the OASIS KMIP Technical Committee, defines a standardized protocol for communication between key management clients and servers. Versions 1.4, 2.1, 3.0 is coming
- Keeping keys under a separate security and administrative boundary enables controlled access, key rotation, auditing, and lifecycle management independently of the systems storing encrypted data.
- Crypto Keys should never be placed alongside encrypted data!

Why another KMIP library?

- Just **one** open source C implementation: Percona's fork of old reasearch project "libkmip"
- Written in C, with traditional C data models
- Manual memory allocation/de-allocaion, separate alloc/free/parse funtions for each request/reponse type
- Minimal specification implementation, not so easy to extend (Activate-Deactivate as an example)
- Too costly in the support and development
- **Why new library:** Modern C++ with memory management and better abstraction; Strong specification conformance; Extensibility and maintainability; Clean and transparent data model; **Unit and integration tests**

Old C library vs new C++ library



Design goals

- Designed for extension: at the moment we do not have full KMIP secs implementation
- Clear protocol level and operations/transport level separation
- Easy to extend protocol level; Easy to use operational API
- Support for multiple request/response batch items: get keys with attributes, for example.
- **Testing and interoperability:** Specification-driven unit tests plus 5 real KMIP servers: PyKMIP, **Cosmian KMS**, HashiCorp Vault Enterprise, Fortanix DSM, Thales CipherTrust Manager. Every release is tested against all five.

Architecture

- **kmipcore**: Data model, KMIP objects, requests/responses, TTLV serialization/parsing. No networking. It could also form the basis of a KMIP server.
- **kmipclient**: Network abstraction (NetClient), connection pool, high-level client API, protocol-version selection and negotiation. OpenSSL based NetClient and connection pool implementations.
- **utils**: Keys and X.509 certificates reading/writing in standard formats, other useful utilites (work in progress)
- **Unit tests** for kmipcore, **integration tests** for kmipclient
- **Examples** for easy start

Kmipclient example: get key

```
NetClientOpenSSL net_client(host: argv[1], port: argv[2], clientCertificateFn: argv[3],
    clientKeyFn: argv[4], serverCaCertFn: argv[5]);
KmipClient client(&net_client);

try {
    std::string id = argv[6];
    auto key:unique_ptr<Key> = client.op_get_key(id);
    std::cout << "Key: 0x";
    print_hex(key->key->value());
    std::cout << "Attributes:" << std::endl;
    for (const auto &[attr_name:const string, attr_value:const string] :
        key->attributes().as_string_map()) {
        std::cout << "  " << attr_name << ": " << attr_value << std::endl;
    }
} catch (const std::exception &e) {
    std::cerr << "Can not get key with id:" << argv[6] << " Cause: " << e.what()
        << std::endl;
    return 1;
}; | Tab to review
return 0;
```

Another example: activate operation

```
NetClientOpenSSL net_client( host: argv[1], port: argv[2], clientCertificateFn: argv[3], clientKeyFn: argv[4],
serverCaCertFn: argv[5], timeout_ms: 200);
KmpClient client([&]net_client);
try {
    const auto opt_key:string = client.op_activate(argv[6]);
    std::cout << "Key with ID: " << argv[6] << " is activated." << std::endl;
    return 0;
} catch (const std::exception &e) {
    std::cerr << "Can not activate key with id:" << argv[6]
               << " Cause: " << e.what() << std::endl;
};

return -1;
```

op_activate implemenmtation

```
std::string KmipClient::op_activate(const std::string &id) const {
    auto request = make_request_message();
    const auto batch_item_id:uint32_t =
        request.add_batch_item(kmipcore::ActivateRequest(id));

    std::vector<uint8_t> response_bytes;
    io->do_exchange(
        request_bytes: request.serialize(), [&] response_bytes, request.getMaxResponseSize()
    );

    kmipcore::ResponseParser rf(response_bytes, request);
    return rf
        .getResponseByBatchItemId<kmipcore::ActivateResponseBatchItem>(
            batch_item_id
        ) // SimpleIdResponseBatchItem<18>
        .getUniqueIdentifier(); // const string &
}
```

Already in production

- Percona server for MySQL (component_keyring_kmip):
 - Releases 8.4.10, 8.4.11
 - Releases 9.7.1, 9.7.2
 - Release 26.10
 - MTR test suite for component_keyring_kmip with Cosmian KMS in Docker

Future development

- Move some code to “utils” submodule, clean up Key API, extend operation API (wip)
- Better support for X.509 (planned)
- Optimization of memory usage and performance (wip)
- Secure memory de-allocation (PR is in review)
- More complete specification implementation
- Users and Contributors are welcome:
<https://github.com/percona/kmipclient>

AI assisted development

- My first experience with AI: GitHub Copilot as the Clion plugin
- Some hand-written code was already present: API, examples, C++ bridge to libkmip
- Feed all spectications to the AI first!
- Explain architecture and goals to the AI, check how it understands, tell it to write docs. Read docs, fix inconistence and errors.
- Discuss the development plan with AI and write some kind of WBS
- Implement API first, take several review/update iteration
- Implement things step-by step, function by function, watch you back!
- Use other models to AI code review

Credits and Thanks

- **Konstatin Trushin**: for friendly help in my first steps with KMIP library and servers (Percona MongoDB team)
- **Yura Sorokin, Przemyslaw Skibinski**: for code reviews, suggestions and friendly support (Percona MySQL team)
- **Zsolt Parragi**: for discussions and great contributions (Percona PostgreSQL team)
- **Saikumar VS**: for extensive testing of the library and KMIP keyring components with 5 different KMIP-capable servers (Percona QA team)
- All Percona engineers and managers for support