01. gRPC 실용사례 CPP

본 문서 에서는 CPP 에서 gRPC 를 실제 사용하기 위한 암호화 서버 및 Client 작성에 관하여 설명 한다. 사용된 암호화 모듈의 구현 코드는 공개 되지 않지만 CPP 에서 gRPC Server 및 Client 를 작성 하는 과정을 볼 수 있다.

1. Interface 를 위한 .proto 파일

실용사례 에서 공통으로 사용할 .proto 파일로 이름은 `xdb_grpc.proto` 로 작성 하였다.

```
syntax = "proto3";

// Option for Java.
option java_multiple_files = true;
option java_package = "com.hancomsechre.xecuredb.grpc";
option java_outer_classname = "XecuredbProto";

// Option for Ovjective-C.
option objc_class_prefix = "XDB";

package xdbproto;

// The Xecuredb service definition.
service Xecuredb {
  rpc Encrypt (EncRequest) returns (EncReply) {}
  rpc Decrypt (DecRequest) returns (DecReply) {}
  rpc CryptoHash (HashRequest) returns (HashReply) {}
}

// The request message containing the alias and plain text.
message EncRequest {
  string alias = 1;
  string plain = 2;
}

// The response message containing the err code and cipher text.
message EncReply {
  int32 err = 1;
  string msg = 2;
  string cipher = 3;
}

// The request message containing the alias and cipher text.
message DecRequest {
  string alias = 1;
  string cipher = 2;
}

// The response message containing the err code and plain text.
message DecReply {
  int32 err = 1;
  string msg = 2;
```

```
  string plain = 3;
}

// The request message containing the alogorithm id and plain text.
message HashRequest {
  int32 id = 1;
  string plain = 2;
}

// The response message containing the err code and hash string.
message HashReply {
  int32 err = 1;
  string msg = 2;
  string cipher = 3;
}
```

2. CPP Interface 파일 생성을 위한 protoc 사용법

```
protoc -I ../proto --cpp_out=. --plugin=protoc-gen-grpc=`which grpc_cpp_plugin`
xdb_grpc.proto
protoc -I ../proto --grpc_out=. --plugin=protoc-gen-grpc=`which grpc_cpp_plugin`
xdb_grpc.proto
```

- xdb_grpc.proto 파일은 ../proto 경로에 존재 한다.
- 이 명령은 Interface 파일이 생성 되는 과정을 보기 위한 예제 이며 CPP 에서는 Makefile 에서 자동 생성 하게 하였다.

3. gRPC 서버 프로그램 작성

```
#include <iostream>
#include <memory>
#include <string>
#include <stdio.h>
#include <stdlib.h>

#include <grpcpp/grpcpp.h>

#include "xdb_grpc.grpc.pb.h"
#include "XdbAdapter.h" // 암호화 함수 관련 Header 파일

using grpc::Server;
using grpc::ServerBuilder;
using grpc::ServerContext;
using grpc::Status;

using xdbproto::DecReply;
using xdbproto::DecRequest;
using xdbproto::EncReply;
using xdbproto::EncRequest;
using xdbproto::HashReply;
using xdbproto::HashRequest;
using xdbproto::Xecuredb;
```

```
char *iniFile = (char*) "XdbAdapter.ini";

class XecuredbImpl final : public Xecuredb::Service {
  // 암호화 함수
  Status Encrypt(ServerContext*    context,
                 const EncRequest* request,
                 EncReply*         reply) override
  {
    int nRes;
    char Buff[ 128 ];

    memset( Buff, 0x00, sizeof(Buff));
    // XdbAdapter.h 에 있는 실제 암호화 함수 호출
    // request 로 전달된 alias(), plain() 을 parameter 로 사용 하였다.
    // request->alias() 는 xdb_gprc.proto 의 EncRequest 에 의해 자동 으로
    // 생성 된다. get/set method 가 protoc 에 의해 자동으로 생성 된다.
    nRes = xdb_enc(iniFile, (char*) request->alias().c_str(),
                            (char*) request->plain().c_str(), Buff );

    // 암호화 결과 오류코드 Setting
    reply->set_err(nRes);
    // 암호화 결과 값 Setting
    reply->set_cipher(std::string(Buff));

    // 함수의 Return 값, 실패시 실패 값을 돌려 주면 될듯.
    return Status::OK;
  }

  // 복호화 함수 (구현 부분은 암호화 함수와 동일한 형식 이므로 설명 생략)
  Status Decrypt(ServerContext*    context,
                 const DecRequest* request,
                 DecReply*         reply) override
  {
    int nRes;
    char Buff[ 128 ];

    memset( Buff, 0x00, sizeof(Buff));
    nRes = xdb_dec(iniFile, (char*) request->alias().c_str(),
                            (char*) request->cipher().c_str(), Buff );

    reply->set_err(nRes);
    reply->set_plain(std::string(Buff));

    return Status::OK;
  }

  // Hash 함수 (구현 부분은 암호화 함수와 동일한 형식 이므로 설명 생략)
  Status CryptoHash(ServerContext*  context,
              const HashRequest*    request,
              HashReply*            reply) override
  {
    int  nRes;
    char Buff[ 128 ];

    memset( Buff, 0x00, sizeof(Buff));
    nRes = xdb_hash(request->id(), (char*) request->plain().c_str(), Buff );
    reply->set_err(nRes);
```

```
    reply->set_cipher( std::string(Buff) );

    return Status::OK;
  }
};

// 서버 실행을 위한 함수, 접근 가능 IP 와 PORT 를 지정 하여 서비스를 생성 한다.
// 아래의 코드를 보면 여러개의 리스너와, 서비스를 등록 하여 사용 가능 한 것으로 보인다.
void RunServer() {
  // 0.0.0.0:포트로 주지 않으면 외부에서 접속을 할 수 없다.
  std::string server_address("0.0.0.0:50052");
  XecuredbImpl service;

  ServerBuilder builder;

  // 리스너 추가
  builder.AddListeningPort(server_address, grpc::InsecureServerCredentials());

  // 서비스 추가
  builder.RegisterService(&service);

  // 서버 포인터 생성
  std::unique_ptr<Server> server(builder.BuildAndStart());
  std::cout << "Server listening on " << server_address << std::endl;

  // 서버 실행
  server->Wait();
}

int main(int argc, char** argv)
{
  RunServer();

  return 0;
}
```

4. gRPC 클라이언트 프로그램 생성

```
#include <iostream>
#include <memory>
#include <string>

#include <grpcpp/grpcpp.h>

#include "xdb_grpc.grpc.pb.h"

using grpc::Channel;
using grpc::ClientContext;
using grpc::Status;

using xdbproto::DecReply;
using xdbproto::DecRequest;
using xdbproto::EncReply;
using xdbproto::EncRequest;
```

```
using xdbproto::HashReply;
using xdbproto::HashRequest;
using xdbproto::Xecuredb;

class XecuredbClient {
 public:
  XecuredbClient(std::shared_ptr<Channel> channel)
      : stub_(Xecuredb::NewStub(channel)) {}

  // 암호화 함수 작성
  std::string Encrypt(const std::string& alias, const std::string& plain) {
    EncRequest request;
    request.set_alias(alias);
    request.set_plain(plain);

    EncReply reply;
    ClientContext context;

    Status status = stub_->Encrypt(&context, request, &reply);

    if (status.ok()) {
      return reply.cipher();
    } else {
      std::cout << status.error_code() << ": " << status.error_message()
                << std::endl;
      return "RPC failed";
    }
  }

  // 복호화 함수 작성
  std::string Decrypt(const std::string& alias, const std::string& cipher) {
    DecRequest request;
    request.set_alias(alias);
    request.set_cipher(cipher);

    DecReply reply;
    ClientContext context;

    Status status = stub_->Decrypt(&context, request, &reply);

    if (status.ok()) {
      return reply.plain();
    } else {
      std::cout << status.error_code() << ": " << status.error_message()
                << std::endl;
      return "RPC failed";
    }
  }

  // Hash 함수 작성.
  std::string CryptoHash(int algoid, const std::string& plain) {
    HashRequest request;
    request.set_id(algoid);
    request.set_plain(plain);

    HashReply reply;
    ClientContext context;
```

```
    Status status = stub_->CryptoHash(&context, request, &reply);

    if (status.ok()) {
      return reply.cipher();
    } else {
      std::cout << status.error_code() << ": " << status.error_message()
                << std::endl;
      return "RPC failed";
    }
  }

 private:
  std::unique_ptr<Xecuredb::Stub> stub_;
};

int main(int argc, char** argv) {

  // 서버 접속을 위한 Client 생성.
  XecuredbClient xecuredb( grpc::CreateChannel( "localhost:50052",

grpc::InsecureChannelCredentials()
                                                  )
                          );
  // 암호화 함수 호출
  std::string alias("normal");
  std::string plain("12345678901234567");
  std::string enc_str = xecuredb.Encrypt(alias, plain);
  std::cout << "Encrypt received: " << enc_str << std::endl;

  // 복호화 함수 호출
  std::string dec_str = xecuredb.Decrypt(alias, plain);
  std::cout << "Decrypt received: " << dec_str << std::endl;

  // Hash 함수 호출
  std::string hash_str = xecuredb.CryptoHash(6, plain);
  std::cout << "Hash received: " << hash_str << std::endl;

  return 0;
}
```

- Server 와 Client 에서 사용 하는 Class 가 다르다. `using grpc::` 부분 눈여겨 볼것.

5. 컴파일을 위한 Makefile 작성

```
HOST_SYSTEM = $(shell uname | cut -f 1 -d_)
SYSTEM ?= $(HOST_SYSTEM)
CXX = g++
CPPFLAGS += `pkg-config --cflags protobuf grpc`
CXXFLAGS += -std=c++11 -I./include
ifeq ($(SYSTEM),Darwin)
LDFLAGS += -L/usr/local/lib `pkg-config --libs protobuf grpc++ grpc`\
           -L./lib -lxdsp -lxdsp_api -lXdbAdapter \
           -lgrpc++_reflection\
           -ldl
```

```
else
LDFLAGS += -L/usr/local/lib `pkg-config --libs protobuf grpc++ grpc`\
           -L./lib -lxdsp -lxdsp_api -lXdbAdapter \
           -Wl,--no-as-needed -lgrpc++_reflection -Wl,--as-needed\
           -ldl
endif
PROTOC = protoc
GRPC_CPP_PLUGIN = grpc_cpp_plugin
GRPC_CPP_PLUGIN_PATH ?= `which $(GRPC_CPP_PLUGIN)`

PROTOS_PATH = ../proto

vpath %.proto $(PROTOS_PATH)

all: xdb_server xdb_client

xdb_server: xdb_grpc.pb.o xdb_grpc.grpc.pb.o xdb_server.o
    $(CXX) $^ $(LDFLAGS) -o $@

xdb_client: xdb_grpc.pb.o xdb_grpc.grpc.pb.o xdb_client.o
    $(CXX) $^ $(LDFLAGS) -o $@

.PRECIOUS: %.grpc.pb.cc
%.grpc.pb.cc: %.proto
    $(PROTOC) -I $(PROTOS_PATH) --grpc_out=. --plugin=protoc-gen-
grpc=$(GRPC_CPP_PLUGIN_PATH) $<

.PRECIOUS: %.pb.cc
%.pb.cc: %.proto
    $(PROTOC) -I $(PROTOS_PATH) --cpp_out=. $<

clean:
    rm -f *.o *.pb.cc *.pb.h xdb_server xdb_client
```

6. 테스트

6.1 Server 기동

```
./xdb_server
```

6.2 Client 기동

```
./xdb_clienr
```

실행 결과

```
Encrypt received: AAGcCKXHphJcgo5zYi2wAH6E7nLi8ImBPP/h1YoNUd7G9g==
Decrypt received: 12345678901234567
Hash received: l/QLiuPkOxGLtK+2I9PnaPBNm8L5E71kKW681TOGwaw=
```

7. Remark