Example C

Table of Contents

1. Library (libs/KalimaLib)

This library contains all the headers of the library, as well as two static archives lib_KalimaNodeLib.a and lib_KalimaNodeLib32.a. These archives will be used when creating the executable in the project (src). The headers will be used to call the methods that will be useful to us in our example.

2. Config file (etc/cfg)

```
LedgerName=KalimaLedger
NODE_NAME=Node Example
NotariesList=167.86.103.31:8080 5.189.168.49:8080 173.212.229.88:8080 62.171.153.36:8080 167.86.124.188:8080
FILES_PATH=log/
PRIVATE_KEY_FILE=RSA/private.pem
PUBLIC_KEY_FILE=RSA/public.pem
BLOCKCHAIN_PUBLIC_KEY_FILE=RSA/public.pem
SerialID=louisTest
```

Let's see the usefulness of the different configurations:

- LedgerName -> This is the name of the Ledger we are going to connect to. For the example it is not very useful.
- NODE_NAME -> This is the name of the node we create. It is also not very useful for our example.
- NotariesList -> This is the list of nodes to which we need to connect to communicatewith the blockchain tutorial. If you want to connect to another blockchain, just change the notaries here. Between each node, you have to put a space.
- FILES_PATH -> This is the directory in which you can find the log files. Din our example, the log folder will be created at launch in the directory
- KEY_FILES -> These are the paths of the different RSA encryption files. These files are necessary to communicate with the Blockchain. They will also be created automaticallyat the start of the project.
- SerialID -> This SerialID will serve as our identification with the blockchain. It must be authorized by the blockchain. This will probably be the only line you will have to modify to put the ID you want.

3. Main project (src)

a. Makefile

```
CC=gcc -pthread
CFLAGS=
LDFLAGS=-lcrypto -lm
EXEC= Exec

LIBRARY=lib_KalimaNodeLib.a

all: $(EXEC)

main.o: main.c
    $(CC) -o main.o -c main.c $(CFLAGS)

Exec: main.o
    cd ../libs/KalimaLib; if [ -d objects ]; then rm -rf objects; fi; mkdir objects; cd objects; ar -x ../$(LIBRARY)
    $(CC) -o main.run main.o ../libs/KalimaLib/objects/*.o $(LDFLAGS)
    cd ../libs/KalimaLib; if [ -d objects ]; then rm -rf objects; fi

clean:
    rm -rf *.o
    rm -rf main.run
    if [ -d log ]; then rm -rf log; fi
    cd ../libs/KalimaLib; if [ -d objects ]; then rm -rf objects; fi

mrproper: clean
    rm -rf $(EXEC)
```

The makefile allows us to create an object for our hand. Then, from this object and the archive of the library, we can create our executable "main.run".

Here, the default library used is "lib_KalimaNodeLib.a" which was compiled on a 64-bit linux architecture. It is also possible to replace with "lib_KalimaNodeLib32.a" to create the executable on a 32-bit architecture.

b. Project Main

```
Node *node = create_Node("../etc/cfg/config.txt", NULL);
printf("Config loaded\n");
Connect_to_Notaries(node, NULL);
sleep(2);
printf("GO!\n");
```

Here we create our node. By default, the node will use the "config.txt" file as the config file. Here it is placed in the etc/cfg folder (.. used to return to the previous folder).

When creating the node, we will also create a random deviceID that will be encrypted and writtenin the "DeviceID" file that will be created in the src folder. If this encrypted file already exists, it will not be recreated. The node will just decrypt the file and use the recovered deviceID. This deviceID, in connection with the SerialID, will allow the block chain to identify our node and allow us to write to it. An RSA folder will also be created containing a public key and a private key used to encrypt communications with the blockchain.

The "NULL" parameters that can be seen here correspond to the fact that we can allow the node to use LuaJIT scipts with a callback. Since in this example we do not use any, we must put NULL.

Finally, it will connect directly to the nodes written in the config file.

Next to the creation of the node, the example offers two choices to the user:

- Sending 10 messages by default

```
void send_10_messages(Node *node){
    for(int i=0 ; i<10 ; i++) {
        uint8_t body_size = get_int_len(95+i);
        char body[body_size];
        snprintf(body, body_size+1, "%d", 95+i);
        char key[13];
        snprintf(key, 13, "%s%d", "temperature", i);
        put_msg_with_ttl(node->clone, "/sensors", 8, key, 12, body, body_size, 10);
        printf("Sending value %s to key %s on address : /sensors\n", body, key);
        sleep(1);
    }
}
```

This choice sends 10 predefined messages to the blockchain on the address "/sensors". The 10 messages will have a key ranging from temperature0 to temperature9 and a value ranging from 95 to 104. Each message will remain in the blockchain for 10 seconds before being automatically deleted

- Fully configurable message

```
void send_modulable_message(Node *node){
    char temp;
    char choice[10] = {}, address[100] ={}, key[100] ={};
    scanf("%c", &temp);                                  //Clear buffer
    while(strncmp(choice, "a", 1) != 0 && strncmp(choice, "d", 1) != 0){
       printf("Do you want to add (a) or delete (d) ?\n");
       fgets(choice, sizeof(choice), stdin);

    }
    printf("Type the address you want to interact with :\n");
    fgets(address, sizeof(address), stdin);
    strtok(address, "\n");
    printf("Type the key of you choice :\n");
    fgets(key, sizeof(key), stdin);
    strtok(key, "\n");

    if(strncmp(choice, "a", 1) == 0){
        char msg[100];
        printf("Type the message of you choice :\n");
        fgets(msg, sizeof(msg), stdin);
        strtok(msg, "\n");
        put_msg_default(node->clone, address, strlen(address), key, strlen(key), msg, strlen(msg));
        printf("Sending value %s to key %s on address : %s\n", msg, key, address);
    }
    if(strncmp(choice, "d", 1) == 0){
        remove_msg(node->clone, address, strlen(address), key, strlen(key));
        printf("Deleting key %s on address : %s\n", key, address);
    }
}
```

Here we build ourselves the message of 0.

First of all, the user is offered to choose between adding or deleting a message. Then we retrieve the address, the key and the message from the user's terminal. Finally we send the message. Unlike the previous example, the message will remain here indefinitely.

4. Program Delivery

To run the example, simply open a terminal and move to the "src" folder.

Then simply type "make" to launch the makefile seen above. This command will create the executable file "myin.run" that will allow us to launch the program (as well as a main.o file that can be deleted, the DevID folder and the RSA folder).

You can also launch with the 32-bit library by typing "make LIBRARY= "lib_KalimaNodeLib32.a"" if you have not already modified the makefile.

```
e/src$ make
gcc -pthread -o main.o -c main.c
cd ../libs/KalimaLib; if [ -d objects ]; then rm -rf objects; fi; mkdir objects;
 cd objects; ar -x ../lib_KalimaNodeLib.a
gcc -pthread -o main.run main.o ../libs/KalimaLib/objects/*.o -lcrypto -lm
cd ../libs/KalimaLib; if [ -d objects ]; then rm -rf objects; fi
```

Finally, you must write "./main.run" to launch the program.

```
e/src$ ./main.run
Config loaded
GO!
commands available :
- 1 ==> Send 10 messages
- 2 ==> Send a modulable message
- 3 ==> Close
```

Once the program is launched, here's what should appear. Here we see that the config file is loaded for node creation. At this level, the node is created and connected to the blockchain. By typing "1", "2" or "3" you can run one of the options seen above.

Here's what we get with the first option:

```
Sending value 95 to key temperature0 on address : /sensors
Sending value 96 to key temperature1 on address : /sensors
Sending value 97 to key temperature2 on address : /sensors
Sending value 98 to key temperature3 on address : /sensors
Sending value 99 to key temperature4 on address : /sensors
Sending value 100 to key temperature5 on address : /sensors
Sending value 101 to key temperature6 on address : /sensors
Sending value 102 to key temperature7 on address : /sensors
Sending value 103 to key temperature8 on address : /sensors
Sending value 104 to key temperature9 on address : /sensors
```

And here's what we have with the second option:

```
Do you want to add (a) or delete (d) ?
a
Type the address you want to interact with :
/sensors
Type the key of you choice :
test
Type the message of you choice :
hello
Sending value hello to key test on address : /sensors
```

Here we decided to send the value "hello" for the key "test" on the address "/sensors"

The third option just serves to close the program.

5. Possible problems

If your message does not appear in the blockchain, here are some possible errors:

- Verify that the SerialID in the config file is the same as the one entered in the blockchain.
- If you delete a DeviceID or RSA file, the new calculated file will be different. It will therefore be necessary to redo the SerialID because the associated DeviceID will no longer be the same.
- If you have made changes on the hand that are not taken into account, make a "make clean" before redoing "make" so that all the changes are taken into account.
- If you have other problems, you can look in the logs when the program has problems and contact us.