1. How to deploy a node on a blockchain?

To deploy a node on a Kalima blockchain, you need to know the list of Master Nodes on which the node will have to connect.

In addition, the node must be authorized on the blockchain (paragraph on authorizations).

At a minimum, the node must contain a Clone. The clone will be in charge of connecting to the blockchain, establishing a secure connection with each Master Node and synchronizing the data on which it is authorized to establish its database in memory.

We can then add our profession. For this two choices are available to us:

- Using callbacks directly in the node source code
- Execution of smart contracts according to the rules of our choice ➔ However this option is only possible with the Java API, via javascript or python contracts, and soon via the C language API, via LUA scripts.

2. Permissions Management

Master Nodes govern the permissions of each Node. For this, blockchain administrators can define permission lists in PUB (Publish) and SUB (SUBSCRIBE).

For example, we can create a "demo" permission, which would be in SUB authorization the addresses /alarms/fire and /sensors, and in PUB authorization only the address /sensors. This permission would therefore allow nodes to be allowed on read and write sensor data, and on read-only fire alarms.

When a node is created, a unique identifier called deviceId is created. This deviceId will then identify the node on the blockchain. However, when the node first connects to the blockchain, the node is not yet known, so it is not allowed on the blockchain. We therefore initiate the identification of the node via a temporary authorization of an identifier called the serialId. The serialId will be used only during the first connection of the node on the blockchain, and will make it possible to make the link between the deviceId and an authorization.

3. Smart contracts

A Smart Contract is a computer program that automates the execution of a set of predefined instructions when prerequisites are met.

During its execution, all validation steps are recorded at the blockchain level. These records ensure the security of all data by preventing its modification or deletion.

As explained above, smart contracts can be executed by client nodes, thanks to the Java API (for javascript contracts see python) and soon thanks to the C API (for LUA contracts).

The Kalima Blockchain uses Smart Contracts developed by standard and open tools like JavaScript and Python, these open tools allow its simple interconnection with other blockchains, other databases and other applications.

In these Smart Contracts we can make the inference of AI thanks to models possibly created from the data collected preferably by Kalima, because they are immutable, secure and complete data.

Here we will describe the process of publishing contracts.

a. Contract storage

Contracts are stored on git directories. To modify a contract, simply push the changes on the git directory.

b. Contract deployment

Contracts are published to the nodes responsible for their executions is done through a devOps chain after each contract commit. The deployment is done in 4 steps:

- Contract encryption: The devOps chain will encrypt the contract in AES256 with randomly generated encryption keys.
- Signature calculation: A signature is then calculated on the encrypted contract
- Publication on the blockchain: The signature of the contract, as well as the keys to decrypt it are published on the blockchain. Thus, only the nodes authorized on this data will be able to execute smart contracts .
- Contract deployment: The devOps chain then deploys the encrypted contracts to the target devices.

c. Loading a contract

To load and execute a contract, a dedicated node must be installed on a machine. Once the contracts have been correctly deployed on the machine by the devOps chain, the node will detect the arrival or modification of contracts, and will be able to load the contract in 4 steps:

- Retrieving contract information: If the node is properly connected to the blockchain, and properly authorized, it will have already received the information from the new contract. He will therefore read this information to retrieve the signature of the contract, as well as the keys necessary for its decryption.
- Verification of the signature of the contract: The node will calculate the signature of the contract it has received, and verify that it corresponds to the one recorded in blockchain. If the signatures do not match, he will not do anything.
- Decryption: If the signatures match, he will try to decrypt the contract in memory, via the keys present in blockchain.
- Loading: If the decryption goes smoothly, it will load the script into memory, and will now be able to execute it.

Encryption as well as the signing of contracts prevent any attempt to fraudulently modify contracts.

d. Performance of a contract

Once the node has been able to successfully load a contract, it will be able to execute it according to the rules that have been defined. To execute a contract we must execute a function present in the script, we can pass it the parameters of our choice, and we can wait for a return of the script.