aitos.io

Feb. 2021

AITOS.19.70.100UM

1.0.1

BoAT Getting Started

BoAT Framework Overview

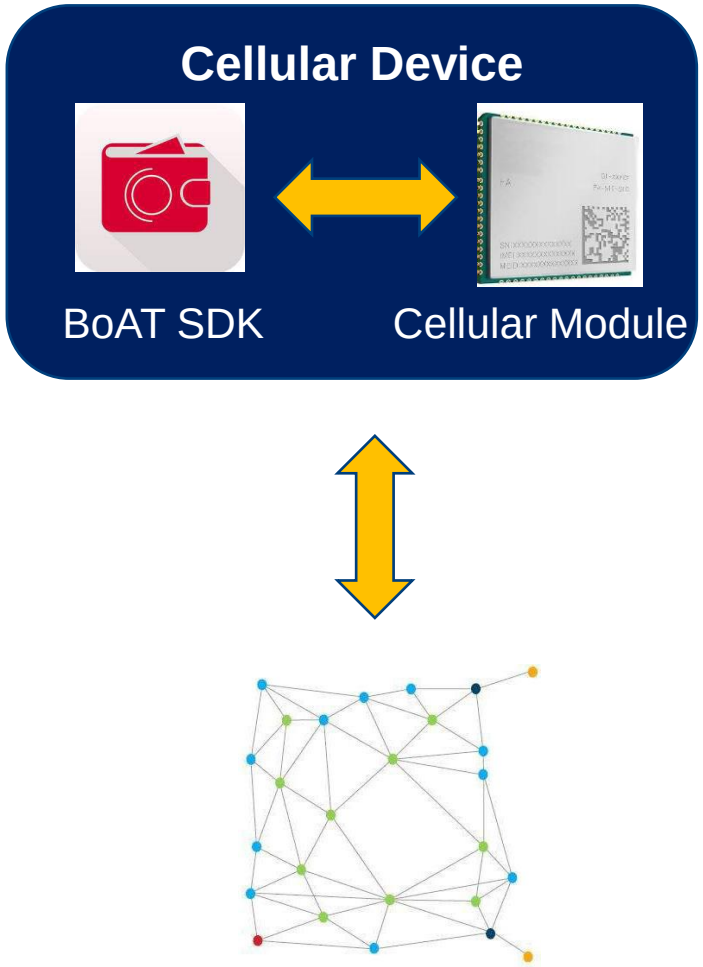
What is BoAT Framework

BoAT is a blockchain application framework for cellular modules.

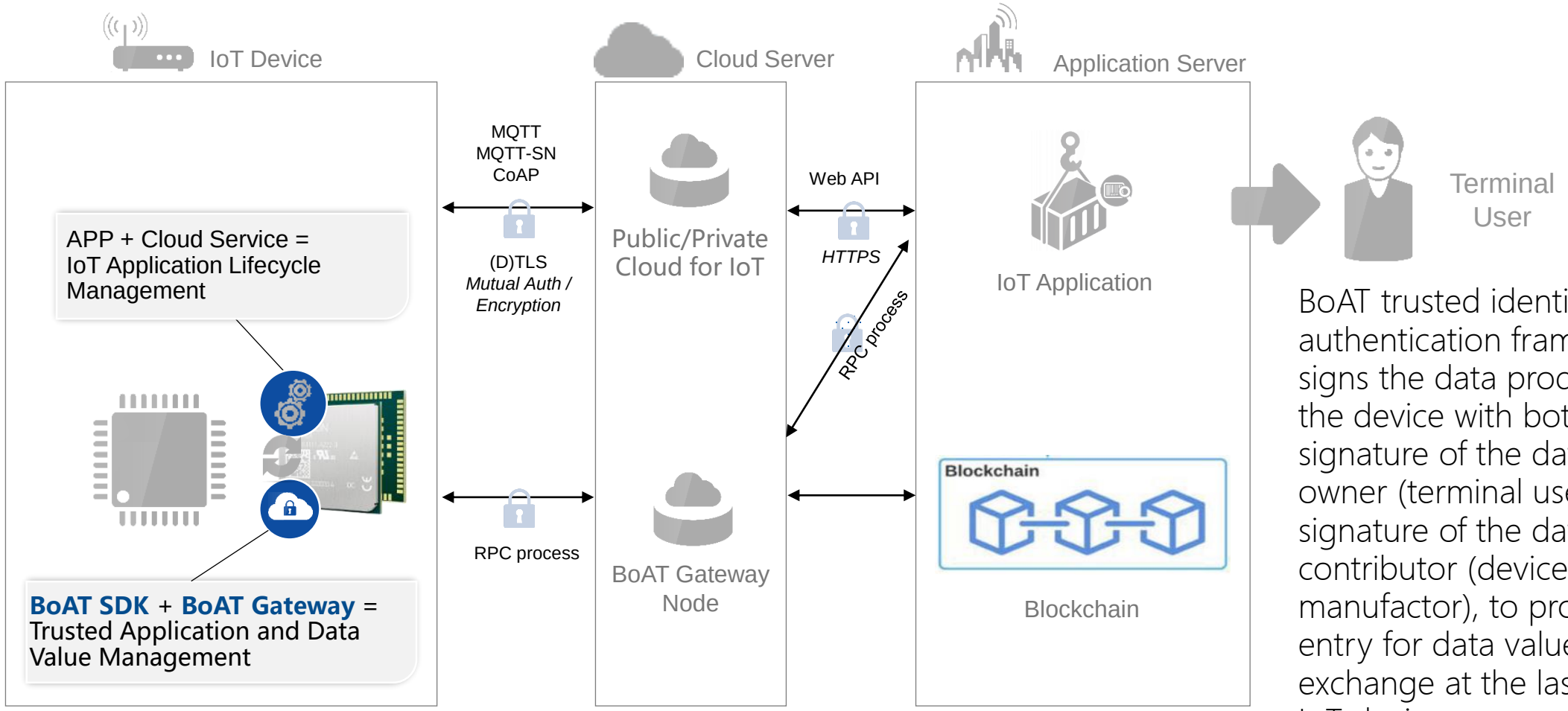
BoAT runs as an SDK for IoT applications in cellular module. It could generate unique identity (cryptography key) for the device, sign the IoT data with the key and invoke blockchain smart contract with the signed data. The blockchain stores the signed data in a decentralized way to ensure the data are tamper-resist during their circulation, which is essential for data value.

BoAT IoT SDK Features

- ✓ Sign data
- ✓ Send blockchain transactions
- ✓ Invoke blockchain smart contract
- ✓ Manage blockchain cryptography keys in the IoT device
- ✓ C and Java implementation

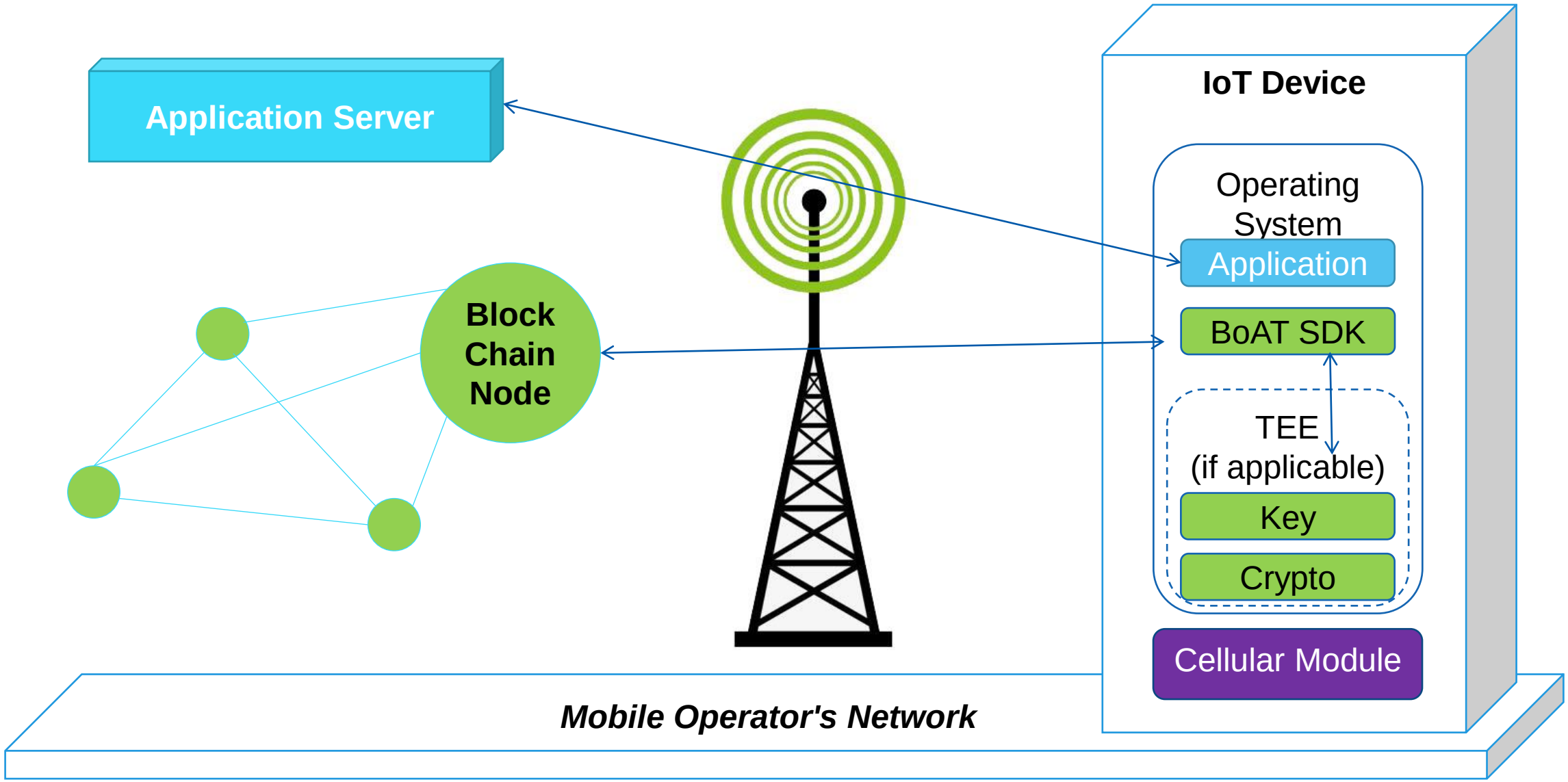


BoAT: Protect Data Value at Last Mile to IoT Device

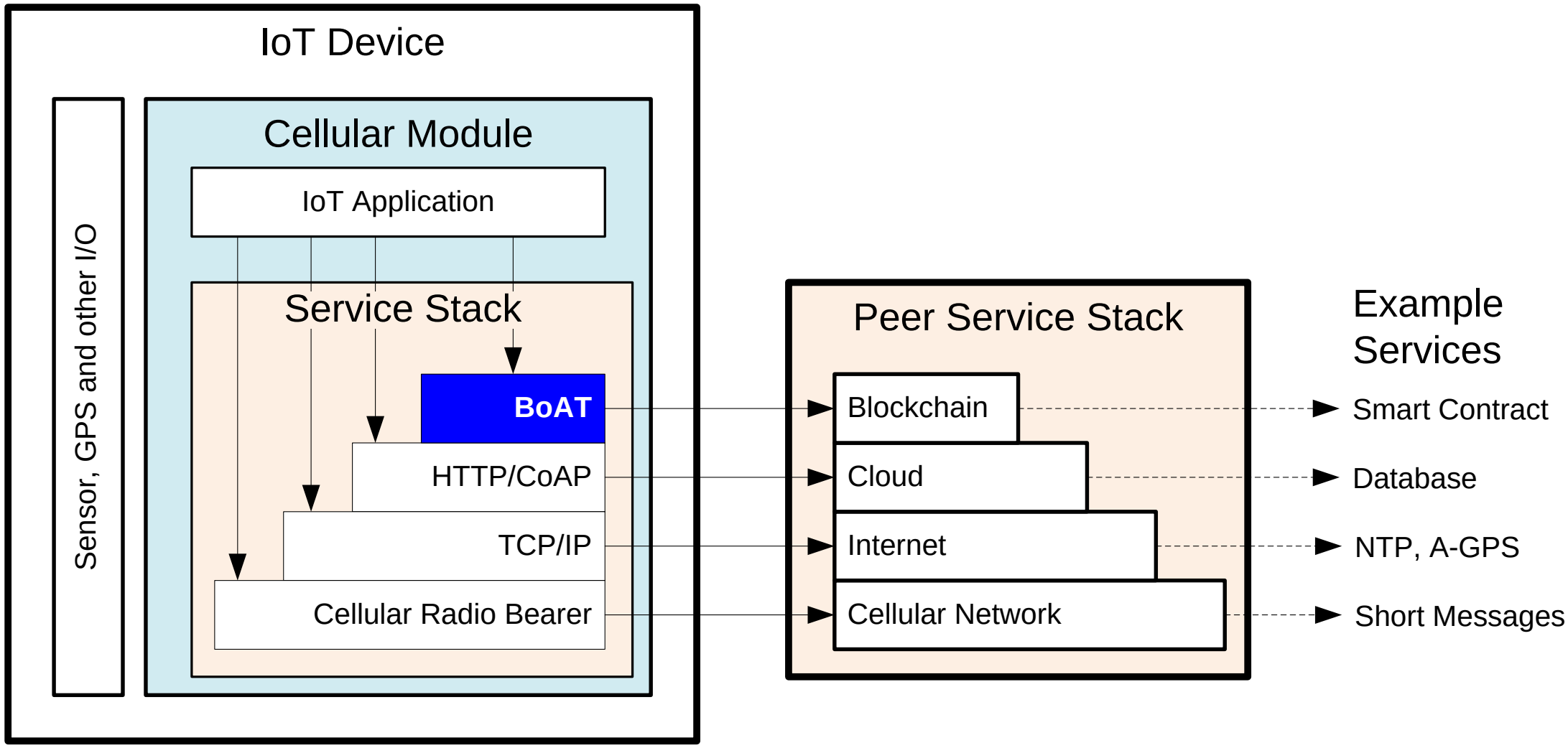


BoAT trusted identity authentication framework signs the data produced by the device with both signature of the data's owner (terminal user) and signature of the data contributor (device manufactor), to protect the entry for data value exchange at the last mile to IoT device.

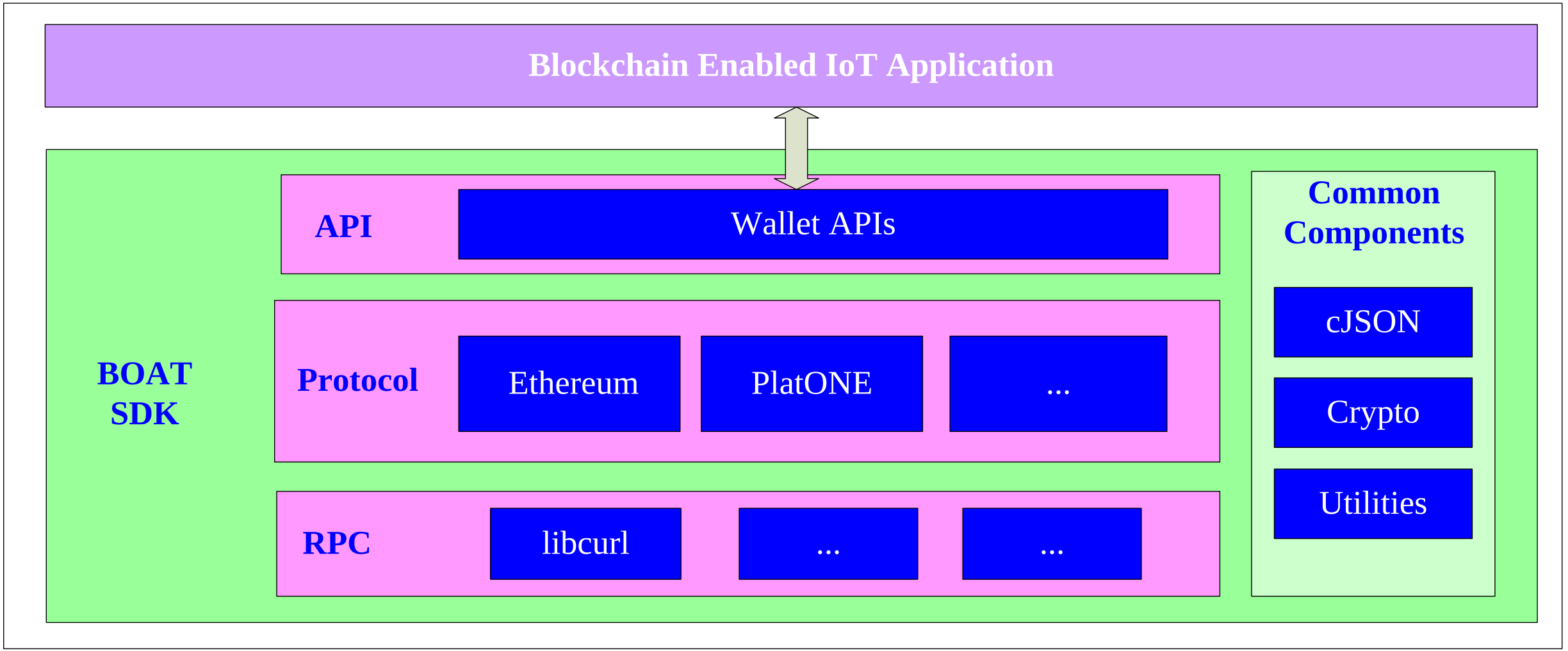
BoAT Framework and BoAT Elements



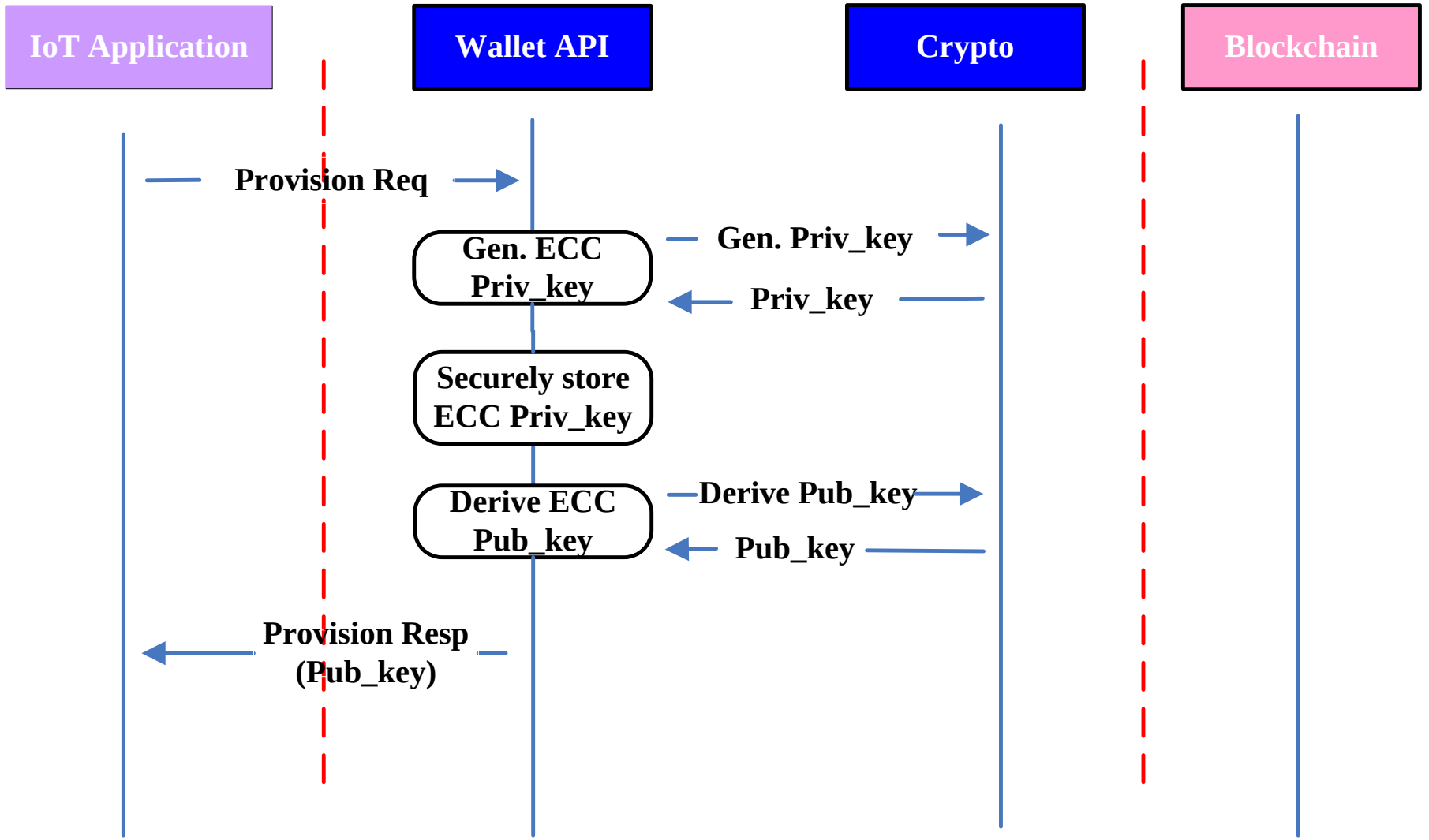
BoAT in System



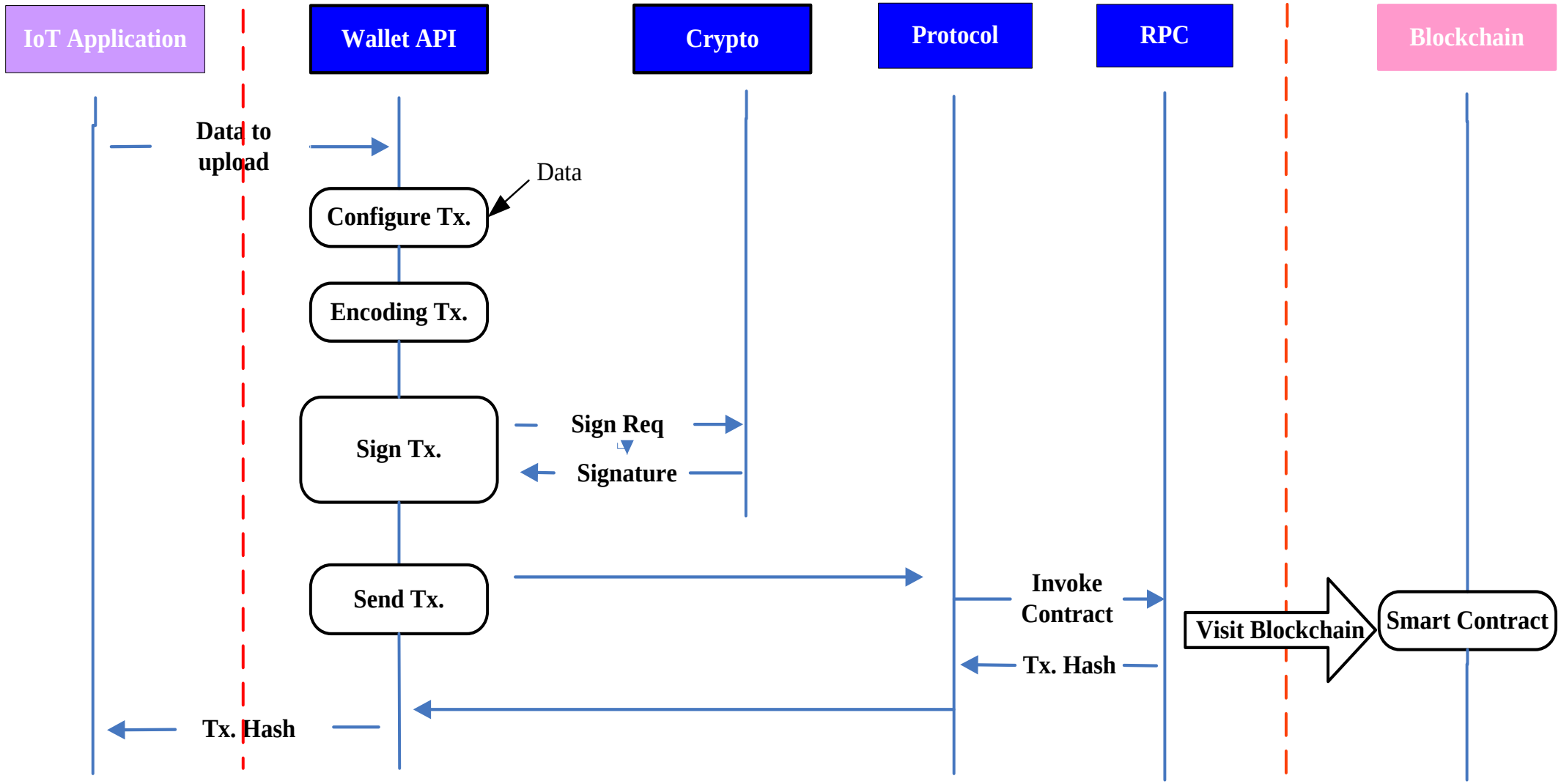
BoAT Software Architecture



Typical Procedure: Provision



Typical Procedure: Upload IoT data onto blockchain



BoAT IoT Framwork SDK (C language)

Characterics

BoAT IoT Framework SDK is a C-language blockchain client SDK compatible with Ethereum and PlatONE.

Dependencies

Demo BoAT IoT Framework SDK is tested with following dependencies. The dependent softwares with lower version probably work well but are not tested.

Host OS - linux or Cygwin
Compiler - gcc 9.3.0
Cross Compiler - arm-oe-linux-gnueabi-gcc 4.9.2
GNU Make - 4.3
Python 3.8.3 (Python 2.7 is also compatible)
libcurl - 7.55.1
openssl - 1.1.1d

BoAT IoT Framework SDK Source Directory

```
<SDKRoot>
|
+---build          | Directory to store object and executable files
+---contract       | Demo smart contract ABI and generated C interface file
+---demo           | Demo application
+---docs           | API reference manual
+---hwdep          | Hardware dependency
|   \---default    |     A default pure software version for development
+---include        | Header files for application to include
+---lib            | Lib files for application to link with
+---sdk            | SDK source
|   +---cJSON      |     An open source JSON parser
|   +---include    |     Header files for SDK internal use
|   +---protocol   |     Blockchain client protocol implementation
|   +---rlp        |     RLP encoder
|   +---rpc        |     Remote procedure call wrapper
|   +---utilities  |     Utility APIs
|   \---wallet     |     SDK entry API implementation
+---tests          | Test cases
\---tools          | Tools for generating C interface from contract ABI
```

NOTE: ./build and ./lib are created in building

Run the Demo

Blockchain Node Setup

To run demo you must have known the IP address of an Ethereum/PlatONE blockchain node and migrate demo contract to the blockchain network.

For Ethereum (public blockchain):
In practice it's a good idea to install Truffle and Ganache, which are the popular Ethereum development suite and blockchain node simulator. Ethernet testnet such as Ropsten is also a good idea.

NOTE: Ganache 1.x and ganache-cli don't “remember” the state once it's terminated. If you're using such versions, you have to migrate the contract every time you run Ganache.

To install Truffle and Ganache, visit: https://trufflframework.com

For PlatONE (consortium blockchain):
To deploy a PlatONE node, visit: https://github.com/PlatONEnterprise

Run the Demo

Contract Migration

Some of the demo cases require demo contracts being deployed on the blockchain network.
See comments in tests codes.

For Ethereum:

You can deploy the solidity contracts with Truffle.
See truffle documents for details.

For PlatONE:

You can deploy the WASM contracts with PlatONE ctool.
See PlatONE documents for details.

Demo contracts lie in ./contract

Run the Demo

Modify the Key and Address

Modify the blockchain node URL (IP address and port), private key, recipient address / contract address in tests code to the value as you deployed.

Build

Step 1:
Extract boatiotsdk source to anywhere that you have write permission. Open a terminal window and cd into the directory where you have extracted boatiotsdk.

Step 2:
Make sure dependency is satisfied.
Configure environment for cross-compiler if you are going to cross-compile boatiotsdk.

Run the demo

Build (cont.)

Step 3:
To build SDK library:
$make boatlibs

To build SDK demo:
$make demo

The generated demo locates at ./build/demo and libraries in ./lib.

Run

To run the demo, open a terminal window, change working directory to *boatiotsdk* and execute:
$./build/demo/boatdemo

The execution result will print in the terminal.

Using BoAT IoT Framework SDK Library in Your Code

Contract C Interface Generation

Smart contract is the code running on the blockchain virtual machine. Smart contract runs like remote API calls. Though the programming language of smart contract is not C, it has defined ABI (Application Binary Interface). Remote call to the contract must follow the ABI.

However manually applying the rule of ABI is quite complex for embedded C programmers. BoAT IoT Framework SDK provides some tools to generate C interface codes from the ABI. The generated C API can be called from other part within the C project. Though not all contract ABI can be converted to C interface due to lack of object-oriented programming capability, the tools could ease a lot of works.

The generation tools are written in Python and lie in ./tools.

Copy the ABI json file generated by truffle or ctool during contract compilation, to the corresponding directory in ./contract. The generation tool will be called against the ABI file during make. You can include generated head files (./contract/generated) in your C code to call the APIs.

Using BoAT IoT Framework SDK Library in Your Code

How to Call a Contract in Your C code

1. Call BoatWalletCreate() with appropriate configuration to create a wallet. The private key in the configuration must follow the actual key in your control. You can also generate a private key by calling BoatXXWalletGeneratePrivkey() if you know what you're doing.

2. Call BoatXXTxInit() with the wallet reference and other parameters to initialize a transaction object (even if it's a state-less call), where XX is the blockchain protocol name.

3. Call generated C interface API with the initialized transaction object and other arguments.

4. Check the return value of the C interface API. If it's not NULL, parse the string content as per contract prototype.

See tests codes for reference.

To manually organize a contract call, refer to the generated C API codes. Note that the interface definition is different for different blockchain protocols.

Using BoAT IoT Framework SDK Library in Your Code

Configure Your Makefile and C code

To use BoAT IoT Framework SDK in your own code, please following these steps:

Step 1:
Place SDK source somewhere in your project and build SDK libraries.

Step 2:
Modify Makefile of your project:
Add include file search path: <SDKRoot>/include
Add to link options all library files in <SDKRoot>/lib in sequence:
libboatcontract.a libboatwallet.a libboathwdep.a
Add to link options: -lcurl -lcrypto

Step 3:
Modify your C code:
Add: #include “boatiotsdk.h”
Follow instructions in “How to Call a Contract in Your C code”

BoAT IoT Framework SDK API

API	Description
BoatIotSdktInit	Initialize SDK
BoatIotSdkDeInit	De-initialize SDK
BoatWalletCreate	Create/Load a wallet
BoatWalletUnload	Unload a wallet
BoatWalletDelete	Delete a persistent wallet
BoatGetWalletByIndex	Get index of created wallet
BoatXXWalletGeneratePrivkey	Generate key pair
BoatXXTxSend	Send a transaction
BoatXXCallContractFunc	Call a state-less contract function
BoatXXTransfer	Transfer value

See *BoAT API Reference Manual* for details.

aitos.io 摩联科技
Contact: info@aitos.io
Github: https://github.com/aitos-io/BoAT-X-Framework