Smart sensor data notarization with Algorand: AlgoIoT

v20231012-1

Table of Contents

[Brief 1](#_Toc147413808)

[Introduction 2](#_Toc147413809)

[Target device 4](#_Toc147413810)

[Software resources (SDKs etc.) 4](#_Toc147413811)

[Our software 5](#_Toc147413812)

[Setup 5](#_Toc147413813)

[Loop 5](#_Toc147413814)

[Conclusions 6](#_Toc147413815)

[References 6](#_Toc147413816)

# Brief

In some scenarios, smart sensors data could use a notarization of sort; an excellent way to accomplish this is through the Algorand blockchain.

In order for data notarization to be trusted, blockchain cryptographic operations should be performed on the sensor node itself; that is, the IoT device should host the wallet internally and directly submit the transaction to the blockchain.

This is easily accomplished if the sensor node is of Single-Board-Computer class (full-blown CPU, standard OS etc.); but many sensors are of microcontroller-level class (MCU), thus having access to far lower resources in terms of processing power, memory, firmware size, energy.

We propose a fully-contained yet lightweight Algorand client for the ESP32 microcontroller (easily adapted to other MCU platforms).

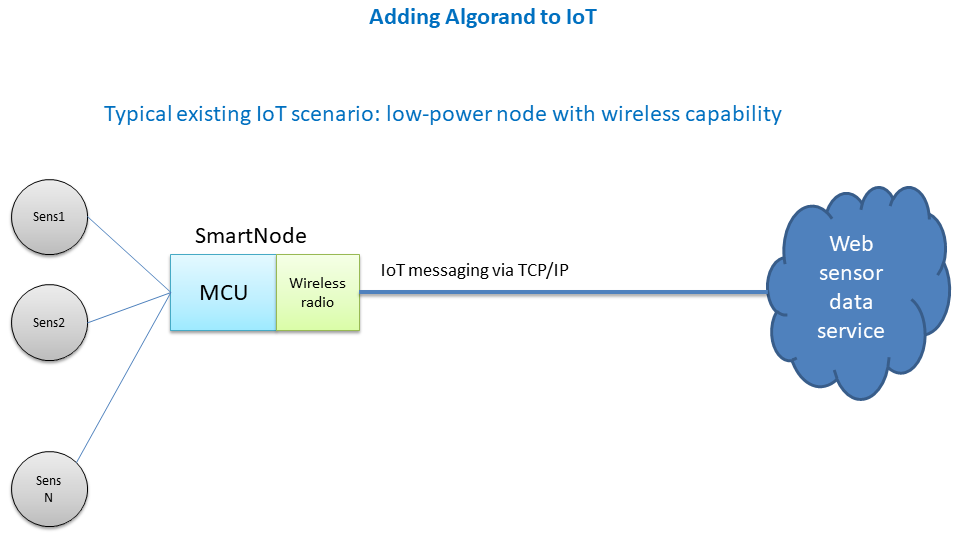
# Introduction

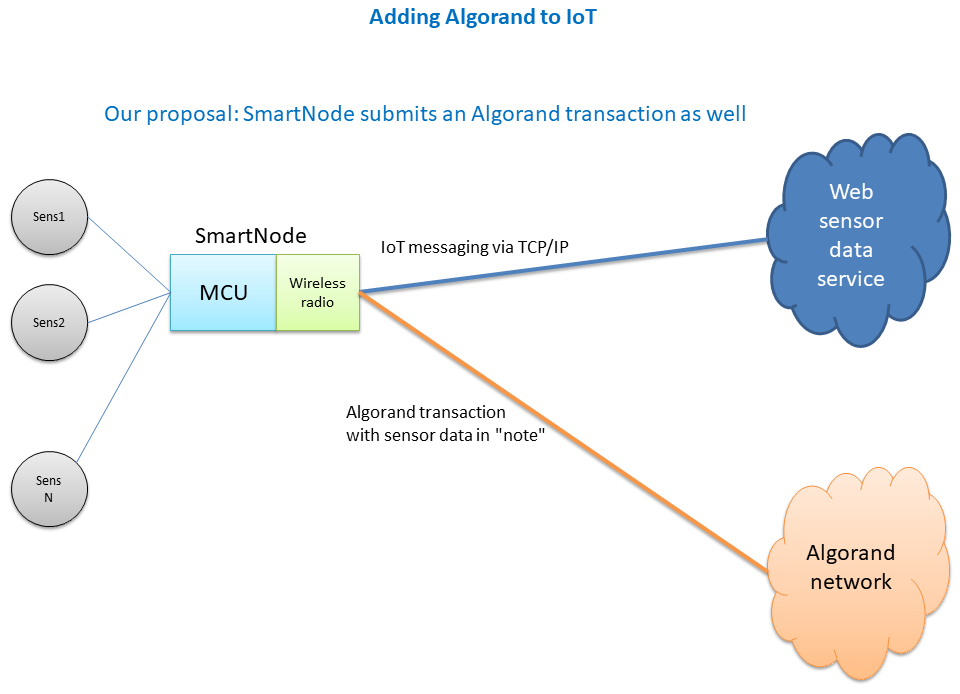
Smart sensors market is growing with double-digit yearly figures; the estimated Compound Annual Growth Rate is between 14% and 19% up to 2027, according to various sources.

“Smart sensors” is a broad term; we will focus on relatively small MCU-controlled, battery-operated units (possibly with energy harvesting capabilities) with wireless network access and the ability to connect to the cloud via a TCP/IP stack.

Our proposal is to add blockchain-based (Algorand-based, specifically) sensors data “notarization” to existing smart sensors infrastructures.

Since we are targeting lightweight devices (microcontroller-class), it is of paramount importance to keep the inevitable overhead, in terms of wakeup time, computing resources, memory footprint, firmware size, to a minimum. At the same time, in order to obtain the required level of trust, we are convinced that the Algorand transaction should originate from the smart sensor itself, without resorting to an intermediate service gateway. This means the MicroController Unit has to manage the Algorand account (keys) itself; it also means the Ed25519 signature has to be processed onboard and that the smart sensor node has to communicate with the Algorand “*algod*” service directly, via its REST API.





# Target device

For our Proof of Concept we selected the ubiquitous Expressif ESP32 microcontroller family.

It is the world’s most common WiFi-enabled MCU, first in worldwide market share since its introduction according to TSR reports1. While the exact number of units sold is not publicly available, the combined sales figures of ESP32 MCUs and the predecessor ESP8266 exceed 1 billion units since 20142.

Being the best seller in its class would be a compelling argument in itself; additionally, the ESP32 microcontroller boasts very good specifications.

For our project, we used the ESP32-PICO-D4 via UM TinyPICO3 development board:

* Dual-core 32 bit RISC CPU at 240 MHz (Xtensa LX6)
* 520 KB RAM
* 4 MB flash memory
* 2.4 GHz WiFi 802.11 b/g/n

Despite having selected the ESP32 as our target device, our code should be easy to adapt to different MCU platforms, especially those compatible with the Arduino development environment; the RAM footprint is about 49 KB, including the Arduino-ESP32 overhead.

# Software resources (SDKs etc.)

Software applications (firmware) for the ESP32 family of microcontrollers may be developed with the well-known Arduino IDE, thanks to the official open source Arduino-ESP32 “core” (tools + hardware abstraction layer)4.

This means we can leverage a large number of libraries available for the Arduino platform.

In particular, we use:

* WiFi5 and HTTPClient6 libraries to connect to Algorand services via the “Algonode” provider7
* ArduinoJson library8 to extract current transaction parameters from the REST query to “algod” and build the “note” field (in ARC-2 format)
* Arduino Cryptographic library9 to sign transaction data with Ed25519 EdDSA
* Base64 library10 to decode Algorand network hash and for debug purposes

Given our goal to minimize memory footprint and to keep firmware size and overhead to a minimum, we wrote all Algorand-specific code in C++ from scratch, keeping everything as bare-bone as possible; therefore, we did not use existing Algorand SDKs.

The basis for Algorand-based “notarization” is the Algorand Payment Transaction; we exploit the “node” field to write sensors data.

# Our software

Our PoC has the following structure, which conforms to the standard Arduino setup+loop sketch format:

## Setup

Initializations, in particular for WiFi client and MessagePack data structure, and conversions from mnemonic formats to binary buffers.

## Loop

Once the WiFi connection is established, we read sensors data via **readSensors()**.

Then we build the “*note*” transaction field in ARC-2 format, calling **buildARC2JNotesField()**.

We need to read current blockchain parameters “*last-round*” and “*min-fee*”, so we query “algod” service via REST; this is implemented in **getAlgorandTxParams()**.

At this point, we have everything we need to build the Algorand Payment Transaction.

First of all, we build the MessagePack11 data structure to be signed. Strict rules must be followed to obtain a MessagePack accepted by “algod”. This is carried on by **prepareTransactionMessagePack()**.

Next step is prefixing “TX” to the MessagePack and passing the resulting binary buffer to Ed25519::sign() (from Arduino Cryptographic Library). It returns a 64-byte signature. The function is **signMessagePackAddingPrefix()**.

At this point, we edit our MessagePack to add the signature (stripping the “TX” prefix in the process). This requires altering its structure, since we now have a root map with two entries (“sig” and “txn”: the latter is the previous root map). Implemented in **createSignedBinaryTransaction()**.

In the end we submit the new MessagePack to the “algod” service, via an appropriate HTTPS POST query to the AlgoNode Testnet endpoint: accomplished by **submitTransaction()**.

# Conclusions

For smart sensors data to be trusted, blockchain-based notarization, expecially leveraging Algorand flexibility and speed, is particularly interesting.

For this solution to be effective, data have to be processed as soon as possible, ideally at the sensing stage, to avoid integrity pitfalls.

We propose a fully working, minimal-footprint Proof of Concept which consists in a ESP32 WiFi-enabled microcontroller which submits Algorand payment transactions straight to the “algod” service (transaction sign process is done on-board); data to be “notarized” are embedded into the “Notes” field of the transaction, following the ARC-2 format12.

# References

[1] https://www.espressif.com/sites/default/files/financial/Espressif%20Systems%202023%20Q2%20%26%20Half-Year%20Report.pdf

[2] https://www.espressif.com/en/news/1\_Billion\_Chip\_Sales

[3] https://www.tinypico.com/

[4] https://github.com/espressif/arduino-esp32

[5] https://github.com/espressif/arduino-esp32/tree/master/libraries/WiFi

[6] https://github.com/espressif/arduino-esp32/tree/master/libraries/HTTPClient

[7] https://algonode.io/

[8] https://arduinojson.org/

[9] https://github.com/rweather/arduinolibs

[10] https://github.com/Densaugeo/base64\_arduino

[11] https://msgpack.org/index.html

[12] https://github.com/algorandfoundation/ARCs/blob/main/ARCs/arc-0002.md