Cloudy with a Chance of c13y

Cipher 2022 hackathon presentation

https://github.com/abukosek/cloudy

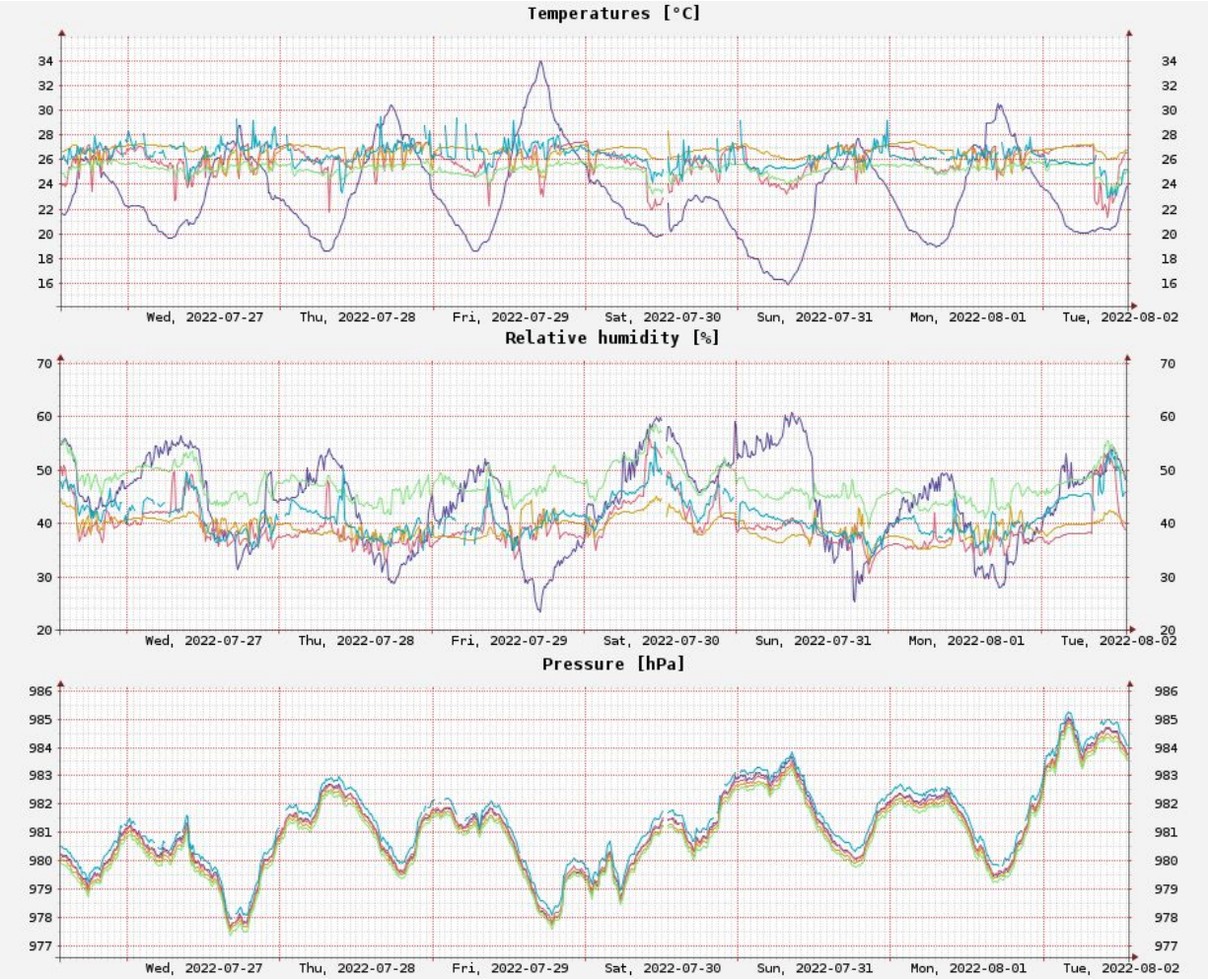
Andrej Bukošek, Matevž Jekovec

Goal

To create a confidential platform for collecting sensor measurements and allow queries on aggregated data only.

Why confidential?
Accurate sensor readings can leak important business secrets or personal habits.

Why blockchain?
Vendor-free, public accessibility. Not much data, block-time lag is not an issue.



Example 1 - Naughty Owner

Tenant is complaining to the apartment owner that it's too hot inside. The owner could place remote thermometers (and other sensors) and regularly check the temperature. BUT, this is a potential intrusion into privacy (e.g. measuring humidity/CO2 levels in the master bedroom every minute).

Cloudy solution™ stores confidential accurate sensors data and allows the owner to check the maximum temperature on a specific day only via smart contract.

Example 2 - Radioactive Puppies

The residents of Krško Nuclear power plant are concerned about the radioactivity in the vicinity. Recently, a highly fluorescent puppy was spotted in the night playing inside the orchard near the plant. The nuclear safety agency claims they perform measurements regularly and there was no leak in the past. The plant owners are strictly against precise monitoring because that would compromise their highly efficient and secret power extraction process.

Cloudy solution™ enables real-time accurate sensor data to be stored, but the general audience can only access the maximum radioactivity levels in the area for the past week.

Example 3 - Sneaky Farmer

Once upon a time, Johnny the baker had dozen of hens. No one knew how he came up with two dozen eggs each day, so the sneaky neighbor - farmer by profession - decided to call the poultry inspector. The inspector wanted to know, whether the hens are stressed because of the heat or they lack water, but Johnny didn’t want him to place a thermometer or other sensors inside the coop, because that would jeopardize his special egg production recipe.

Cloudy solution™ allows accurate temperature and water consumption readings, but only a ratio between those are reported to the inspector and he can rest assured that the hens are not too stressed while the rest remains secret.

Architecture

- Hardware:
  - ESP8266 (running NodeMCU) with sensors:
    - BME280 - temperature, relative humidity, pressure
    - MH-Z19B - CO2 concentration
    - BH1750 - illuminance
- Server and CLI (Go):
  - collects readings from sensors over a TLS connection
  - submits the batch to Cipher every X minutes
- Cipher WASM contract (Rust):
  - stores measurements into ConfidentialMap `(sensor ID, measurement type) -> Vec<(timestamp, value)>`
  - answers measurements queries (min, max, avg) depending on the granularity initially set for the sensor (=level of privacy)

Lessons learned

- Don’t use a garbage-collected language like Lua on a microcontroller with 30kB of available memory.
- Append-only data structure for storing measurements is tricky on confidential blockchain - changing one cell in the array means encrypting a complete array which is $$$. We decided to store sensor measurements in buckets for which time span is user-defined when they register a sensor (e.g. 1 bucket = 10 minutes). This way encrypting the array (vector) is cheap enough for writes and accessing it during queries is still cheap enough. The ConfidentialMap from the previous slide is in fact `(sensor ID, measurement type, bucket) -> Vec<(timestamp, value)>`
- Rewriting rust types to golang used by cbor is cumbersome.
- Gas estimation does not work for `oasis upload`, `oasis instantiate` and also inside go client contract calls. Since you cannot figure out what was the exact gas used for some execution, we did binary search of the optimal value and then extrapolate this to the (in our case) request size - number of measurements.
- Debugging smart contracts is PITA. We fought this by replicating e2e tests as rust’s unit tests and tried to figure out crashes there. Ideally, one would have a local cipher instance adding eprintln! lines in WASM contract and then checking node.log for messages. Also, logs are useful when running the contract over time.

TODO

- Hardware:
  - Client authentication for the sensors (this was only skipped because the poor ESP8266 boards were already running out of memory with just the basic TLS communication).
  - Rewrite the ESP8266 code in C.
  - Finish adding support for the geiger counter sensor.
- Cipher WASM contract:
  - Proper ACL. Currently, only the contract owner can register new sensors, define buckets/privacy granularity and submit measurements. Queries can be performed by anyone.
- Web and mobile app