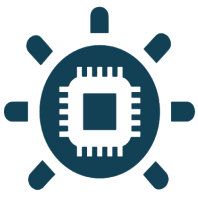
# OpenPAYGO

## EnAccess

## Open Source PAYGO Token

***Software Part***

***Documentation for the example implementation***



PaygOps

Last-Mile Management System

The EnAccess Foundation

# License

While projects financed by EnAccess usually use an MIT license for maximum openness, this project is using an Apache 2.0 license that adds the additional restriction over MIT license that changes made to the code have to be documented when used in other projects. We have chosen to do this to avoid having projects using this token system with modification that break compatibility with no mention that they are not compatible, hence leading to confusion.

# How to use the code provided:

## The Example C implementation for a Device:

The “Example_Device_Implementation_C” folder contains code intended to make it very easy to implement the system on a device. It is a working example that implements a device “simulator” and shows how the system would work on it. To use it on the real device, just replace the “simulator” functions (such as the BlinkRedLed function or the GetKeyPressed function) by the real function used in your device.

The folder contains:

- The “main.c” file that presents an example of a complete PAYG firmware that can be implemented on the device
- The “opaycode_system” folder, containing a device-neutral C implementation of the code system, with the code generation and code decoding functions.
- The “device_simulator” folder, containing “simulator” implementations of the functions that would be needed on a real device (for example the BlinkRedLed function only prints “The Red LED Blinked” on your computer screen instead of actually blinking an LED.

## The Python full system implantation and security tests:

The “Example_Full_System_Implementation_Python” folder contains code intended to showcase how the full system of a server and a device would work together to generate tokens and decode them. It provides a concrete scenario tests and allows to generate tokens of any value for any device (given that the device uses the same key as the server).

The server and device simulator both have a print_status() function that can be used to print their internal state, this should be very useful for debugging when implementing into the device. The whole code is made for use with Python 3 and the tests can be run as it is.

The folder contains:

A “shared.py” file containing functions shared between encoder and decoder

- A “encode_token.py” and “decode_token.py” containing respectively the function to generate tokens and to decode tokens.
- A “server_simulator” folder that contains an example implementation of a server
- A “device_simulator” folder that contains an example implementation of a device
- A “tools” folder with a utility to generate random keys
- An “example” folder that contains test scenarios
- A “security” folder that contains example attacks (see the “Security Audit” for context)

**Important Note:** A secret key was here chosen randomly for the example. This secret key ensures that someone knowing the algorithm cannot generate tokens for all your devices if he does not have that secret key. For more security in the actual implementation we should use a different key that you choose yourself and keep private for everybody except the person who compiles the firmware in the device and the person who sets up the server.

# How to quickly test that a device implementation is functional:

This test (scenario 1) allows to quickly assess whether an implementation seems to work or not and help in the debugging process, even if it is by no mean exhaustive. For more tests, you can check the test scenarios 2 and 3 in the full system implementation example.

1. Setup the device with the starting code 123456789 and the model as well as with the following secret key: **{0xa2, 0x9a, 0xb8, 0x2e, 0xdc, 0x5f, 0xbb, 0xc4, 0x1e, 0xc9, 0x53, 0xf, 0x6d, 0xac, 0x86, 0xb1}**. We assume that the device has a count of 0. **DO NOT USE THAT KEY IN PRODUCTION, THIS IS JUST AN EXAMPLE.**

2. Press '*' to start entering the token and enter the token "123456789" into the device, the Red LED should blink 10 times showing that the token is not valid. Invalid codes are the codes that take the longest to process by design.

3. Press '*' to start entering the token and enter the code "662486790", this should activate the device for 1 day (Add Time) and the Green LED should blink twice to show it is valid.

4. Press '*' to start entering the token and entering the code "662486790" again, this should not change the device activation and the Red LED should blink 10 times to show that the token has already been entered properly.

5. Press '*' to start entering the token and enter the code "927706818", this should activate the device for an additional 29 days days (Add Time) and the Green LED should blink twice to show it is valid.

6. Press '*' to start entering the token and enter the code "942433796", this should activate the device for 7 days (Set Time), removing 23 days from the current status, and the Green LED should blink twice to show it is valid.

7. Press '*' to start entering the token and enter the code "650975787", this should disable PAYG on the device (it should now be active forever) and the Green LED should blink 5 times to show it is valid.

8. Press '*' to start entering the token and enter the code "592185789", this should enable PAYG again on the device and set it to 0 days (not active). and the Green LED should blink 2 times to show it is valid.