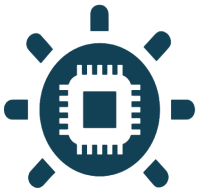
OpenPAYGO

EnAccess

Open Source PAYGO Token

Token System General Documentation



PaygOps

Last-Mile Management System

The EnAccess Foundation

Index

Description of the business need

This project targets manufacturer that want to add a Pay As You Go (PAYGO) feature to their product. They can use the hardware examples to add a hardware control feature (e.g. a switch to control the load) and use this token system as a secure way to remotely control the activation of the devices via a token.

The token system consists of a way of generating tokens containing a certain number of days to be activated on a unit (which will be generated on a server), and a way to verify that a token is valid and extract the number of days to be activated (which will be done in the device).

The token generation part of the system can then be integrated with a software platform such as PaygOps or any other software platform. The token validation part can be integrated in virtually any device.

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.

General requirements for the use case

- The activation token should be able to contain 2 years worth of activation time (and should be able to use more tokens after that period to add more time, up to at least 10 years).
- The activation tokens should be able to both add and remove days from a system (presenting advantages over most currently existing solution).
- The tokens should also be able to enable or disable the PAYGO mode at will
- The token system should be extensible to add certain commands required by some devices (but that do not need standardisation, e.g. “Enable High Power mode”).
- The code should be reasonably secure; trying to type the same code twice, taking a code from a neighbour or typing codes randomly should not succeed (or extremely rarely).
- The token generator (the server), will be given some starting data for each device that are defined by the manufacturer. This includes a key (that can be shared for a whole

batch or not) as well as a serial number and a “starting code” for each device. A specific format for this data based on CSV shall be specified to ensure interoperability.

- The token verification (the unit) will know its “starting code” and the key and will extract the number of days to be activated from the code
- The codes should ideally be 9 digits longs to make successful entry by end user more likely
- For devices with limited space and without the options for infrared remotes, having the ability to enter codes using only digits 1 to 4 should be possible (at the expense of accepting a longer 15 digit code).

Target minimum hardware requirements for the token verification part

- An MCU with at least 8KB of AVAILABLE flash (after the rest of the firmware functions are implemented). It is also possible to use a separate dedicated MCU that will communicate with your main MCU but this entirely depends on your design requirements and choices of MCU.
- A keypad input method (either Infrared Receiver and Infrared remote, an actual membrane keypad, or a USB port to which a USB keyboard can be connected). It should preferably have at least 12 buttons, but 10 buttons or even 4 buttons can be sufficient.
- A hardware Real Time Clock (RTC) that is accurate enough to drift by a non- noticeable time over 12 months (usually less than 6h would be good). However, there is no need for the RTC to be configured with the actual real time, it is only required that the timestamp obtained from the RTC keeps going up constantly by 1 every second with minimal drift.
- An EEPROM or other memory on which permanent data can be stored (such as the serial number, device keys, PAYG status, etc.). Flash can be used as long as proper care is taken to make sure the lifetime of the used flash blocks is sufficient
- A UART or other communication interface that allows the “starting code” and key to be set in factory. Alternatively, some other way to have different firmware for each unit can be used (e.g. factory flashed MCUs with procedure to set different values in flash for each device).
- If the device can be disconnected from the battery, also make sure that:
  - The RTC can be powered by a backup battery to not be reset and keep running when the battery is disconnected
  - There are backup registers with at least 96 bits that are kept by the backup battery when the power is disconnected (but lost if the backup battery is not

present and the RTC stops running). This is used for the storage of key values that are frequently changing.

Example implementation provided

An example implementation both on the server side and on the device side is provided with this documentation. It should allow you to just copy and paste most of the code into the firmware of your device and only adapt parts of the codes (which are purposely placeholders using Unix libraries to allow simulation on a computer) to match the actual hardware of your device (mainly the code to read the keypad presses, blink the LEDs and to access the RTC and memory). For more details, see below after the description of the solution.

Simplified description of the solution

The code system is based on codes that contain a unique “Activation Value” that the device for which it was generated can find. The code also contains a “count” that increments by one each time a code is generated on the server.

The decoder makes sure that the code entered has a count that is higher than its current count but only up to 30 higher for security reasons. This number can be decreased significantly to improve security, although there is a compromise in terms of usability in case the clients loses enough intermediary codes that the new code is then considered invalid. This ensures that entering a code already used will not add more days again. The decoder then extracts the activation value from the code and returns it to the rest of the system.

Encoding of the value

The value is encoded in the last 3 digits of the code by masking the value in the “starting code”. This is done by adding the value to the last 3 digits of the “starting code”, if the resulting value is over 999, then 1000 is subtracted from the value. For example, if the code is “123456**789**” and the value to be encoded is 50, the resulting code with value encoded would be “123456**839**”.
The decoding is done by subtracting the “starting code” base (the last 3 digit) to the base (last 3 digits) of the received code. For example, if the received code is 123456829, the base is 829. Subtracting 789 from 829 gives us 40 which is the value. If the value obtained is negative, then 1000 needs to be added.

Decoding the code on the device

1. Decode the value form the input code.

2. Generate the “base code” by adding encoding that value into the starting code following the steps in the “encoding of the value” section
3. Pass the “base code” X times through the “code generation function”, X being the last count + 30, at each iteration:
   3.1. Replace the “code base” in the resulting code with the “encoded code base”. For example, if the resulting code from step 3 is “234567**890**”, then the final code would be “234567**839**” (if the value to be encoded is 50 and the base 789 like in the example above)
   3.2. If the current X is strictly higher than the last count, we compare the resulting code to the input code, if they match. If it matches, then the code is valid and we return the value. If not we continue iterating
4. If the end of the loop is reached and no match was found, the code was invalid (either already used or properly invalid).

Description of the code generation function

The function only takes a 32 bits integer no bigger than 999999999 as a parameter, and returns a similar integer.

Computational steps for code generation:

1. Copy the input integer twice (big endianness) turning the 32bits (4 bytes) into 8 bytes
2. Generate the SipHash-2-4 hash of the those 8 bytes with the key
3. Split the resulting hash into 2 parts of 32 bits
4. XOR the two parts together
5. Remove the top 2 Most Significant Bits from the result of the XORing above (leaving the 30 LSBits)
6. If the resulting value is over 999999999, subtract 73741825 to the value, this leaves 29.5 effective bits of entropy

Description of the use of the decoded values

Standard use

- The activation code can contain values from 0 to 999
- Values from 0 to 995 represent number of days of activation, any of those values enables the PAYG mode
- The value 998 signify the disabling of PAYG mode
- The value of 999 is used for counter synchronisation

Customisable use for extension

- The code can be extended to 12 or 15 digits, with respectively 6 and 9 digits of value to be stored, this can be used for special data and settings for the devices. In that case the hash is computed on all the 6 or 9 digits of value. Any length between 7 and 15 digits could technically used for customised value use (with 9 digit reserved for activation purpose), although length that are multiples of 3 are recommended for readability purposes.

- The values of any code of length other than 9 digits is not covered by this standard and up to individual device manufacturers to use as desired.

- The values 996 and 997 are reserved for future extensions

Description of the “Add Time” and “Set Time” modes

Use of the modes

The token system supports two modes for the use of the values between 0 and 995;

- **Add Time** is used to add time to the current time present on the device (e.g. if the device has 3 days left and you do Add Time with 2 days, the device will now have 5 days left).
- **Set Time** is used to set the time on the device to a specific value (e.g. if the device has 3 days left and you do Set Time with 2 days, the device will now have 2 days left)

Limitations of Add Time

The Add Time mode is limited in the fact that ALL the tokens must be entered in the right order for the client to have the proper number of days, this is an issue in the following scenarios:

- If the client pays twice in a row and enter first the second token receive, he won’t be able to use the days of the first token received and will therefore be activated for less days than expected (e.g. if he buys a 7 days token and then a 1 day token, and enters only the second one, he will have only 1 day of activation and the first token will be rejected).

- Tokens are occasionally lost by clients, and trying to enter all the tokens to see which one was the last one entered is time consuming.

The Add Time mode also prevents the removal of days, if a client is given too many days by mistake, there is no way to remove the days from the device.

Limitations of Set Time

The Set Time mode can intuitively feels like it is the “best” mode to use since it allows both adding and removing days and ensure that the client has the right number of days, however there is an issue with important consequences; the time at which the token is actually entered in the device can usually not be controlled (as the client does it himself). In some cases, this means that if the client has still time left on his system and waits to enter a new token he receives with extra time, he could end up having more time than expected.

For example: If a client buys 10 days of activation and enters the token into his device, but at day 5 buys another 7 days of activation, he will get a token with 12 days. In theory he should be active for 17 days from the first day if he did enter the token when he receives it, which would be fine. However, if he waits until the end of the first 10 days (when his device turns off) to enter the second token, he will then have his device activate for 12 more days, meaning that his device will still be active 22 days after the first day, instead of 17 days.

Recommendation for combined use of Add Time and Set Time

What we recommend doing for handling activation as smoothly as possible, is to use a combination of Add Time and Set Time. In particular, we recommend that:

- For general situation where time is only added to the device, use Add Time

- If time is removed from a device, use Set Time to set the new correct time. We would also recommend keeping to use set time until the new activation time is over the time previous to the removal, this would guarantee that the client cannot add days without first removing the ones that should have been removed. For example: If the client was mistakenly given 28 days instead of 18 days on the 1st of the month, and that we removed 10 days, but he then pays for 7 days, Set Time should be used. And any further token generated before the 28th of the month should also use Set Time in case the client did not enter the other token to remove the days.

- In case of issues where a code was missed, or the client has less days than expected, use Set Time to make sure the client has the correct number of days.

- If the time to be given exceeds the overall time that can be given in a single token (e.g. >995 days). Several Add Time codes can be generated and sent to the client without issues (whereas several Set Time code would not work).

Time Divider and Activation Time

In some situations, the time granularity of 1 day for activation tokens is insufficient and the provided 995 days is not necessary. This could for example be the case for small devices that could be paid with sometimes half-days promotions.

To be able to cater to those specific scenarios, the devices can have a Time Divider constant used to increase granularity (at the expense of the maximum time that can be added with one code). For example, with a Time Divider value of 4, each unit in the value decoded from the token will correspond to ¼ day of activation (6 hours), hence an example value of 22 would correspond to 5 days and 12 hours of activation time. In that case, the maximum time that could be activated would be 248 days and 18 hours which is still sufficient in most applications.

Any value of the time divider between 1 and 255 should be acceptable, allowing granularity down to 5 minutes for some edge-case applications. However, we recommend using values no larger than 16 for practical applications as activation times of up to 2 months are extremely common in most applications and granularity smaller than 1h30 is not usually needed.

The Time Divider should be specified in the Device Setup parameters (as part of the CSV sheet). In case it is not specified, it will be assumed that is 1, meaning that it is assumed that the device will add 1 day of activation for each unit of value.

Restricted Digit Set Mode

Some devices do not have a full set of buttons to enter all digits between 0 and 9, instead they often have only 4 or 5 buttons. For those cases, OPAYGO provides a “Restricted Digit Set” mode, that needs to be set in both the device and server parameters. In that case, the server converts the original token to a longer token using only digits between 1 and 4.

The encoding is done by taking the original token represented as an unsigned integer and iterating through its bit 2 by 2, converting them to a digit between 0 and 3 and adding 1 for easier readability. The decoding is done by reading each digit individually and concatenating their value represented on 2 bits to end up with a large integer representing the original token. For example, the token 662 486 790 gives us the following bits 10 01 11 01 11 11 00 10 11 11 11 00

00 01 10. Those, taken 2 by 2 gives us the following sequence of numbers between 1 and 4: 324 244 134 441 123.

The difference in length is represented in the table below.

Regular token length	Restricted Digit token length
9	15
12	20
15	25

Counter Synchronisation feature

In some situations, the count kept on a server might get widely out of sync with a device, for example if a user mistakenly generate a lot of tokens for a unit and does not enter them in the device, or if up-to-date counter values are partially lost due to a server failure.

The counter synchronisation feature exist for those situations. If the value encoded in the token is 999, then the device should attempt to decode tokens with counts from 0 and up to 100 above the last count. Tha*t is, the device will pass the “base code” X times through the “code generation function”, X being the last count + 100, and at each iteration will replace the code base and compare with the current token reset count. If the token matches and the count is either 0 or between last count - 30 and last count + 100, it will set the internal token count to that count.

The support for this token synchronisation feature is optional, and the values for the number of tokens below and above the last count are indicative and so is the support for the counter synchronisation token with a count value of 0 (also called “reset token”).

An alternative solution to solve those issues is to directly adjust the token on the server, either by trying different values until the tokens are accepted by the device, or by reading out the count of the device directly via some mechanism (UART, display, etc.). Those solutions are often preferred as they present less security compromise, but they might not be feasible in terms of cost or complexity.

Security Considerations

Waiting period for invalid tokens

To ensure a good level of security, it is recommended that a waiting period is implemented in case of invalid token entry in order to make brute force attacks more difficult. We recommend starting with 1 minute wait, and doubling it for each subsequent invalid token entry going up to 512 minutes (~8 hours) after 9 consequent invalid tries.

This cap should be low enough that if a mistake (i.e. child playing with the keypad) happens, the wait is not so long as to prevent the use of the product, but is long enough that brute force attacks become very difficult (almost 1 year to try 1000 different tokens). We also recommend making sure that this waiting period is kept if the device is turned off and on again.

Key Security

It is highly recommended to use different keys for every device when the assembly processes make it possible. Great care should also be taken to keep those keys secure (e.g. avoiding sending files with all the device keys in non-secure ways) and to select them with a proper random process.

Counter synchronisation tokens

Device manufacturers should be mindful that while this feature is convenient, it affects the level of security of the device by having a larger range of valid tokens, making their bruteforce easier (while still unlikely).

Moreover, counter synchronisation tokens with a count value of 0 (also called “reset tokens”) by essence will work at any point in time in the lifetime of the device. So although they can be useful in some situations (regular counter sync not working or units needing with a lost count requiring refurbishment). Those reset tokens should never be given to a client, as it will allow them to just reuse any code by entering them into the device, then entering the reset token and entering the code again. Instead, they should only be generated and used by technician during very specific maintenance procedures.

Hardware Security

Device manufacturers should not neglect the risk of hardware attacks, particularly targeting lines that can be easily modified from low to high (or vice-versa) to enable a product.

Security Audit

For more information about security risks and mitigation, see the “Security Audit” document.

Description of the Device Setup parameters

Parameter	Description
Serial Number	Defined by the manufacturer, not actually used inside the device but required as a single unique identifier for PAYG platforms.
Starting Code	Defined by the manufacturer, should be random for each device and 9 digits long.
Key	Defined by the manufacturer, can be different for each device but does not have to be, it can be unique for a manufacturer or a distributor for convenience.
Time Divider (Optional)	Defining the division factor (from 1 day) to be applied to the activation value. Acceptables values are 1 to 255. If not specified, the default value should be 1.
Restricted Digit Mode (Optional)	If set to 1 the tokens generated will only use digits between 1 and 4, if set to 0 or not set, it will use the regular set of digit between 0 and 9.
Count (Optional)	If set, the count of the device will be initialized with that value, the default value is 0. Using another random value between 1 and 10 can help improve security.
Test Code (Optional)	It is not technically an OpenPAYGO valid token but will be used by the software platform to avoid clash (see note below)

Code Values Table

Code Value	Usage
0-995	Activation value from 0 to 995 (automatically enables PAYG)
996	RESERVED
997	RESERVED
998	Disable PAYG
999	Counter synchronisation

Format for setup parameters communication

To ensure smooth communication between the device manufacturers and the software provider or in between software providers when migrating, we recommend following a particular template. It is available in the file “example_device_list. csv”.

It is important that the CSV be encoded using actual comma separator “,” and a carriage return for line separation. It is also important that the first line (the header) is kept and that all the columns appear even if they are empty (the default value as defined above will then be used).

Important consideration on Serial Numbers

While serial numbers are not actually used inside the device or by the OpenPAYGO Token system, they are required as a single unique identifier for PAYG platforms. For that reason we highly recommend the following measures:

1. **Print the serial numbers very visibly on the devices:** both Agents and Clients will often need to find them for usage with the PAYG platform (e.g. as payment reference)

2. **Include a prefix related to your brand in the serial number:** this will prevent different products from different manufacturers having the same serial number. For example, if

your brand is “SuperLight” you can use the prefix “SLT” prefix and have serial numbers that look like this: SLT30000123 hence making serial number clashes a lot less likely.

3. **Be sure to use the same format of serial numbers everywhere:** make sure that the serial number format that you use as printed on your device, on the box, on the CSV list, etc. always have the same format to avoid confusion. For example avoid printing “30000123” but then having “SLT30000123” in the CSV to avoid confusing clients.

Test codes

In some cases, in particular during testing, manufacturers might find it useful to have a “test code” that activate the device for a short period of time (enough to test the device, but not enough to be useful to an end user, typically 30 seconds). It is often useful that these test codes are shared by a whole batch of devices to facilitate testing. We recommend for security purposes to limit the use of those codes, for example preventing the use of the test code more than 5 times every 60 minutes.

In any case, those test codes do not have the same constraints as the regular codes. We recommend to directly check for them in the code and bypass the OpenPAYGO Token logic entirely for those test codes. To avoid clash with OpenPAYGO Token, we still recommend that this test code be provided in the CSV spreadsheet to the software provider, so that if by chance an actual token was the same as the test code, the count could be increased to generate an alternate token.