ALDABA

Interfaces de acceso a la infraestructura de Generación Cuántica de números aleatorios (QRNG)

Manuel Ángel Linares Franco

Director General

ALDABA

Una breve carta de presentación de nuestra empresa

Capital privado 100% gallego

Oficinas en A Coruña y Ourense

>20 años prestando servicio

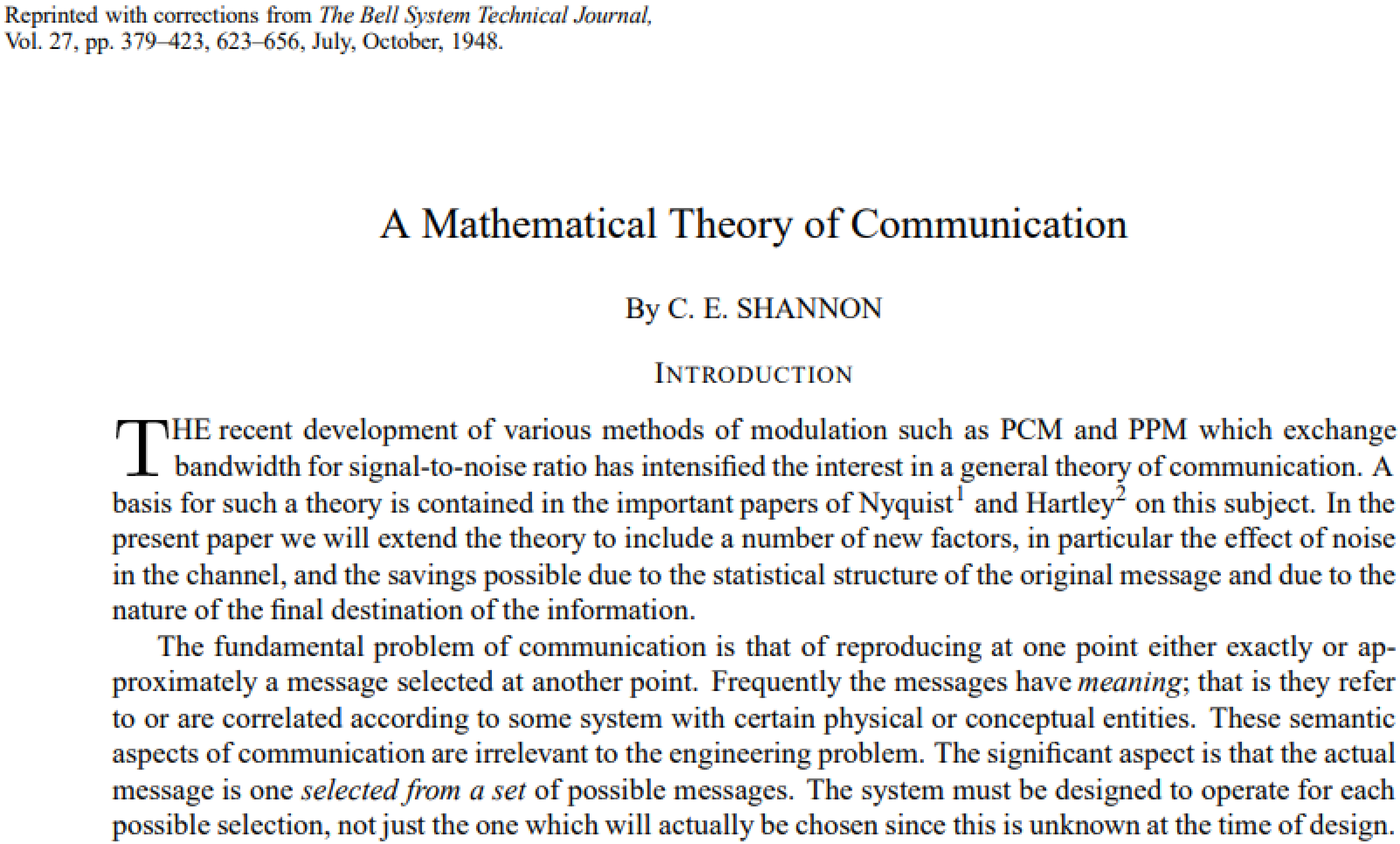
>200 profesionales formados en las universidades gallegas

Foco en aportar valor al tejido industrial. >99% de nuestra facturación proviene del sector privado

Aportamos soluciones en:

- Ingeniería software
- Ingeniería de datos y analítica
- Infraestructura de sistemas (datacenter/cloud) y seguridad
- Consultoría SAP

Random Number Generation



Reprinted with corrections from *The Bell System Technical Journal,*
Vol. 27, pp. 379–423, 623–656, July, October, 1948.

A Mathematical Theory of Communication

By C. E. SHANNON

INTRODUCTION

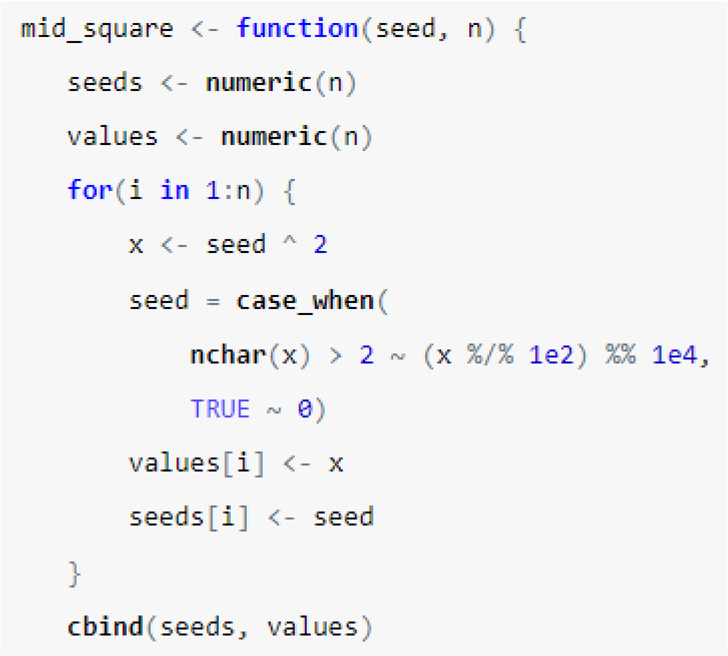
THE recent development of various methods of modulation such as PCM and PPM which exchange bandwidth for signal-to-noise ratio has intensified the interest in a general theory of communication. A basis for such a theory is contained in the important papers of Nyquist[1] and Hartley[2] on this subject. In the present paper we will extend the theory to include a number of new factors, in particular the effect of noise in the channel, and the savings possible due to the statistical structure of the original message and due to the nature of the final destination of the information.

The fundamental problem of communication is that of reproducing at one point either exactly or approximately a message selected at another point. Frequently the messages have *meaning*; that is they refer to or are correlated according to some system with certain physical or conceptual entities. These semantic aspects of communication are irrelevant to the engineering problem. The significant aspect is that the actual message is one *selected from a set* of possible messages. The system must be designed to operate for each possible selection, not just the one which will actually be chosen since this is unknown at the time of design.

$$\mathrm{H}(X) := -\sum_{x\in\mathcal{X}} p(x)\log p(x) = \mathbb{E}[-\log p(X)]$$

Random Number Generation

Von Neuman: parte media del cuadrado



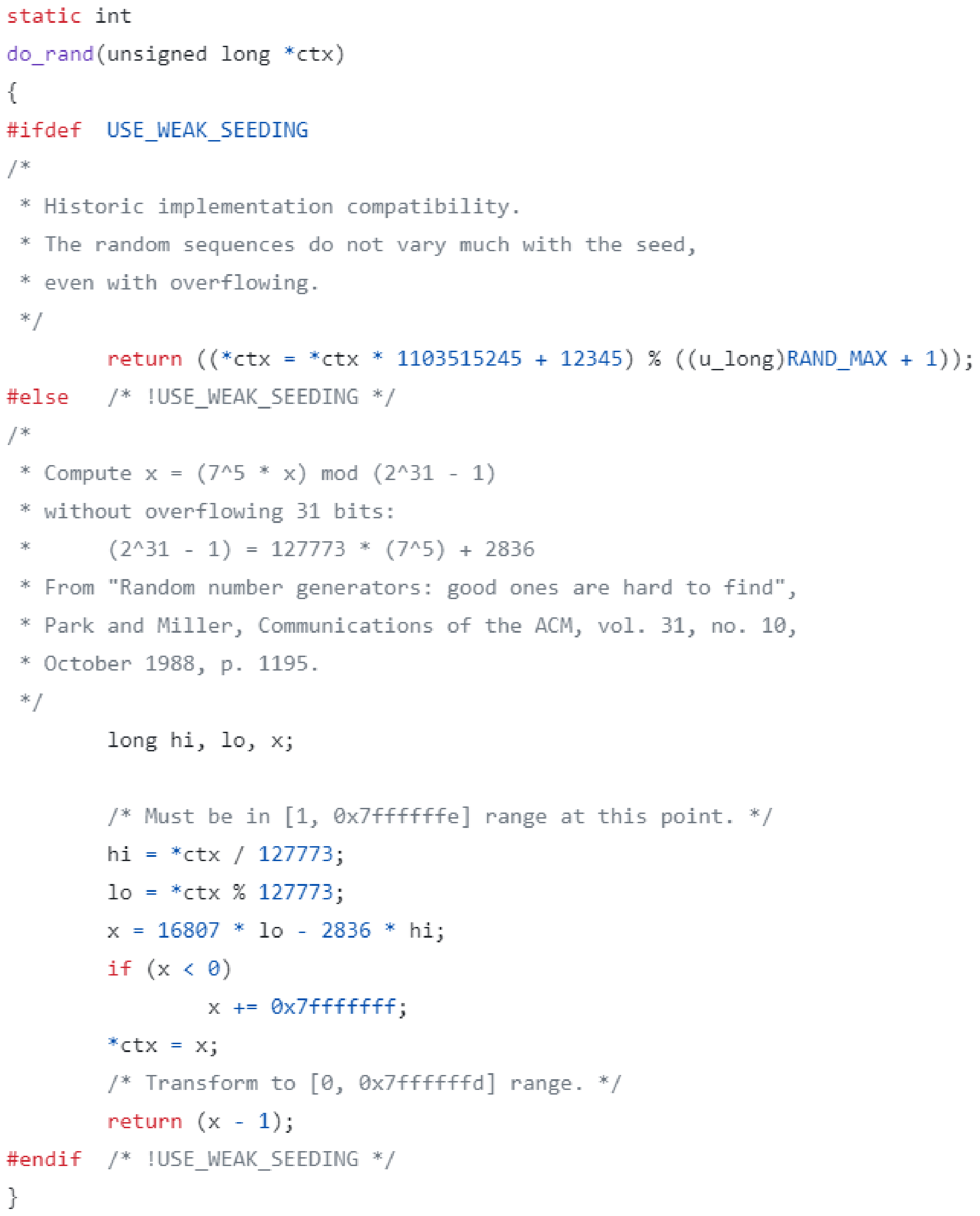
```r
mid_square <- function(seed, n) {
   seeds <- numeric(n)
   values <- numeric(n)
   for(i in 1:n) {
       x <- seed ^ 2
       seed = case_when(
           nchar(x) > 2 ~ (x %/% 1e2) %% 1e4,
           TRUE ~ 0)
       values[i] <- x
       seeds[i] <- seed
   }
   cbind(seeds, values)
```

RANDU

$$X_{n+1} = (2^{16} + 3)X_n mod(2^{31})$$

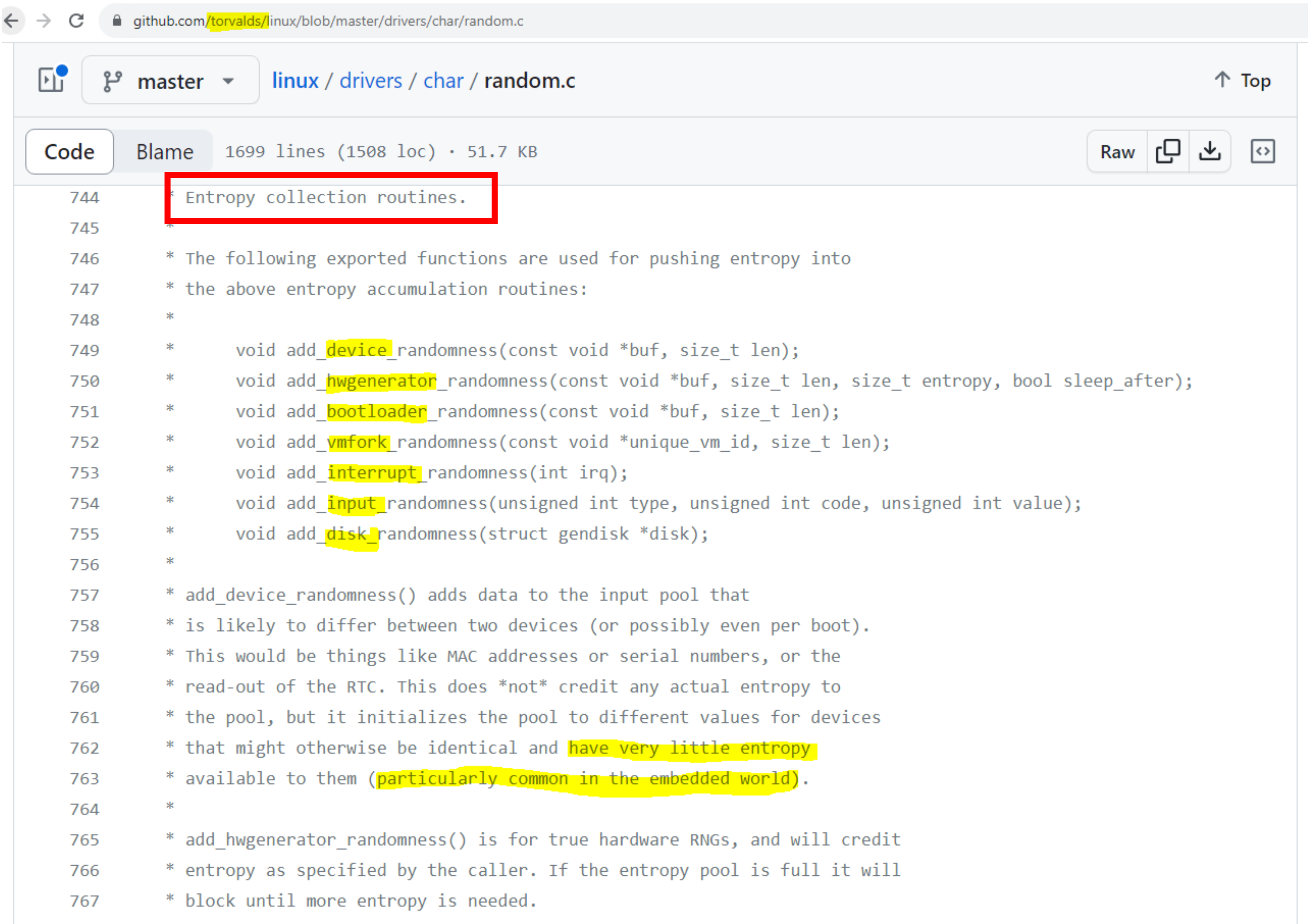
RAND (Mathlab, BSD)

$$X_{n+1} = (7^5)X_n mod(2^{31} - 1)$$



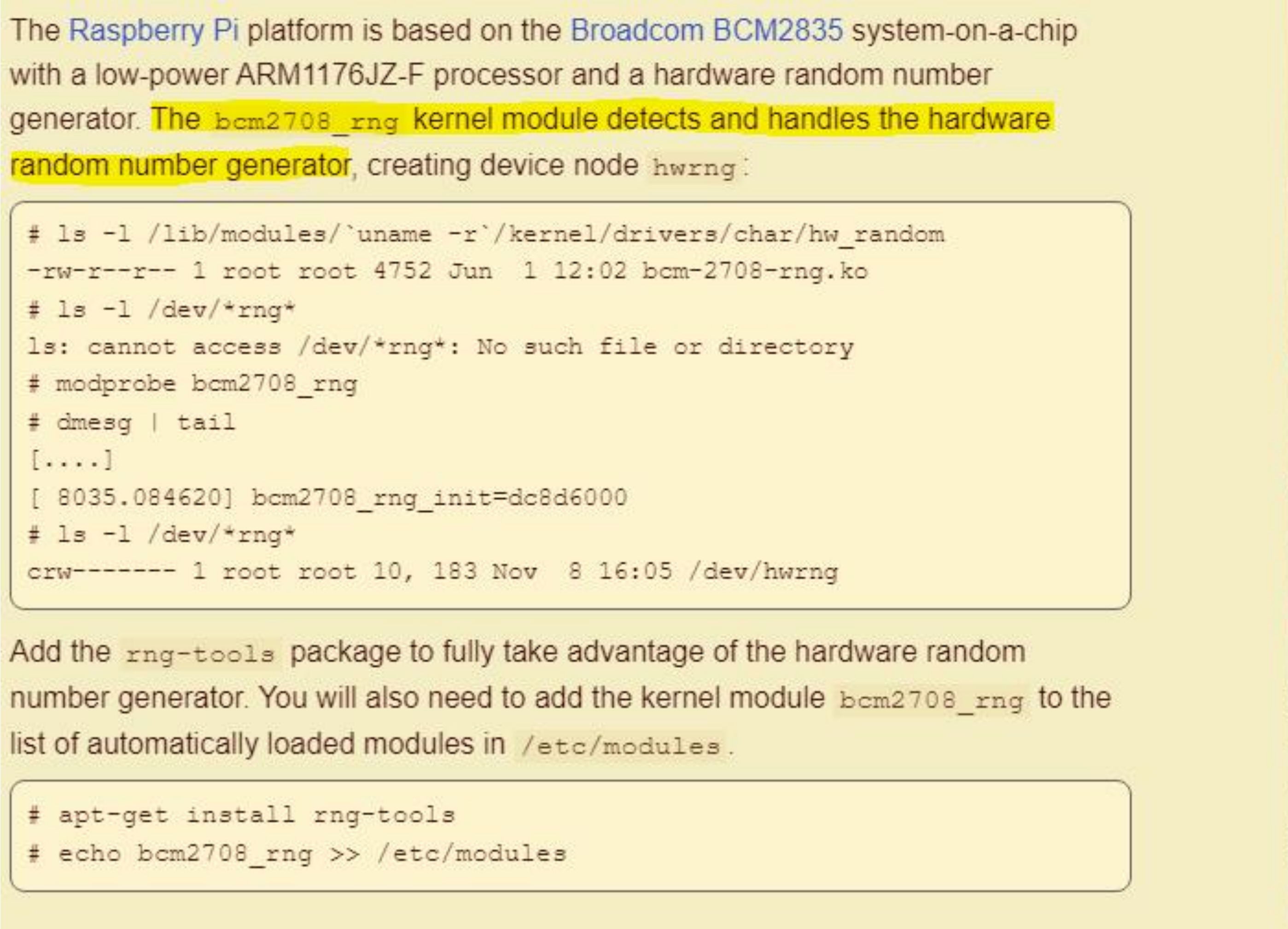
```c
static int
do_rand(unsigned long *ctx)
{
#ifdef  USE_WEAK_SEEDING
/*
 * Historic implementation compatibility.
 * The random sequences do not vary much with the seed,
 * even with overflowing.
 */
        return ((*ctx = *ctx * 1103515245 + 12345) % ((u_long)RAND_MAX + 1));
#else   /* !USE_WEAK_SEEDING */
/*
 * Compute x = (7^5 * x) mod (2^31 - 1)
 * without overflowing 31 bits:
 *      (2^31 - 1) = 127773 * (7^5) + 2836
 * From "Random number generators: good ones are hard to find",
 * Park and Miller, Communications of the ACM, vol. 31, no. 10,
 * October 1988, p. 1195.
 */
        long hi, lo, x;

        /* Must be in [1, 0x7ffffffe] range at this point. */
        hi = *ctx / 127773;
        lo = *ctx % 127773;
        x = 16807 * lo - 2836 * hi;
        if (x < 0)
                x += 0x7fffffff;
        *ctx = x;
        /* Transform to [0, 0x7ffffffd] range. */
        return (x - 1);
#endif  /* !USE_WEAK_SEEDING */
}
```

Random Number Generation


github.com/torvalds/linux/blob/master/drivers/char/random.c
master
linux / drivers / char / random.c
Top
Code
Blame
1699 lines (1508 loc) · 51.7 KB
Raw

```
 * Entropy collection routines.
 *
 * The following exported functions are used for pushing entropy into
 * the above entropy accumulation routines:
 *
 *      void add_device_randomness(const void *buf, size_t len);
 *      void add_hwgenerator_randomness(const void *buf, size_t len, size_t entropy, bool sleep_after);
 *      void add_bootloader_randomness(const void *buf, size_t len);
 *      void add_vmfork_randomness(const void *unique_vm_id, size_t len);
 *      void add_interrupt_randomness(int irq);
 *      void add_input_randomness(unsigned int type, unsigned int code, unsigned int value);
 *      void add_disk_randomness(struct gendisk *disk);
 *
 * add_device_randomness() adds data to the input pool that
 * is likely to differ between two devices (or possibly even per boot).
 * This would be things like MAC addresses or serial numbers, or the
 * read-out of the RTC. This does *not* credit any actual entropy to
 * the pool, but it initializes the pool to different values for devices
 * that might otherwise be identical and have very little entropy
 * available to them (particularly common in the embedded world).
 *
 * add_hwgenerator_randomness() is for true hardware RNGs, and will credit
 * entropy as specified by the caller. If the entropy pool is full it will
 * block until more entropy is needed.
```

Random Number Generation

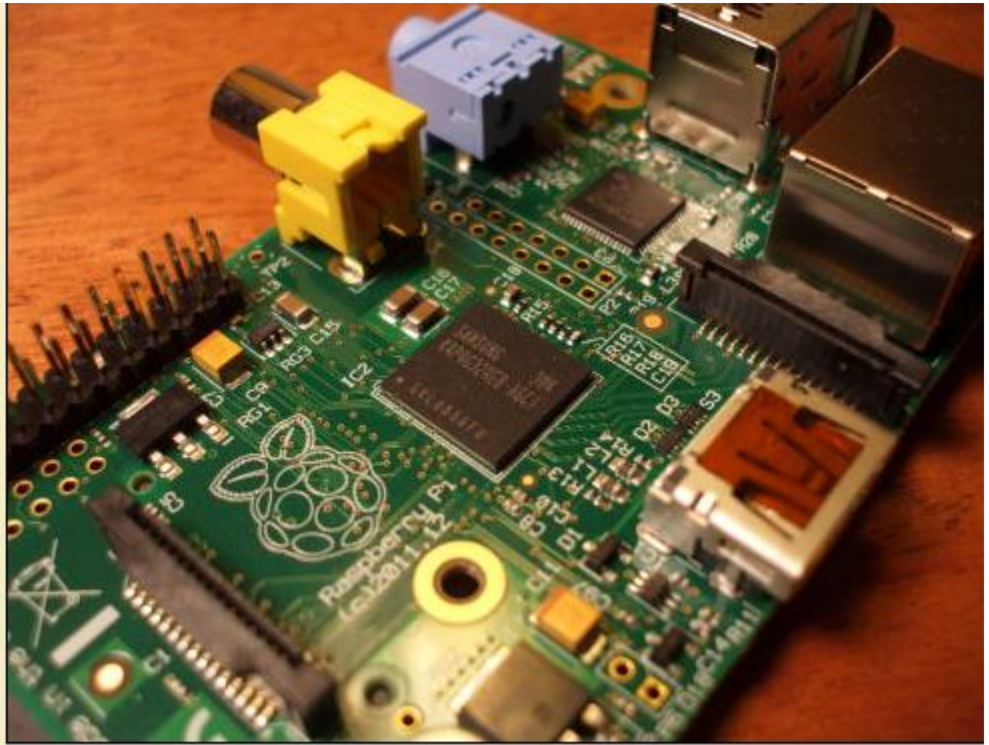


The Raspberry Pi platform is based on the Broadcom BCM2835 system-on-a-chip with a low-power ARM1176JZ-F processor and a hardware random number generator. The `bcm2708_rng` kernel module detects and handles the hardware random number generator, creating device node `hwrng`:

```
# ls -l /lib/modules/`uname -r`/kernel/drivers/char/hw_random
-rw-r--r-- 1 root root 4752 Jun  1 12:02 bcm-2708-rng.ko
# ls -l /dev/*rng*
ls: cannot access /dev/*rng*: No such file or directory
# modprobe bcm2708_rng
# dmesg | tail
[....]
[ 8035.084620] bcm2708_rng_init=dc8d6000
# ls -l /dev/*rng*
crw------- 1 root root 10, 183 Nov  8 16:05 /dev/hwrng
```

Add the `rng-tools` package to fully take advantage of the hardware random number generator. You will also need to add the kernel module `bcm2708_rng` to the list of automatically loaded modules in `/etc/modules`.

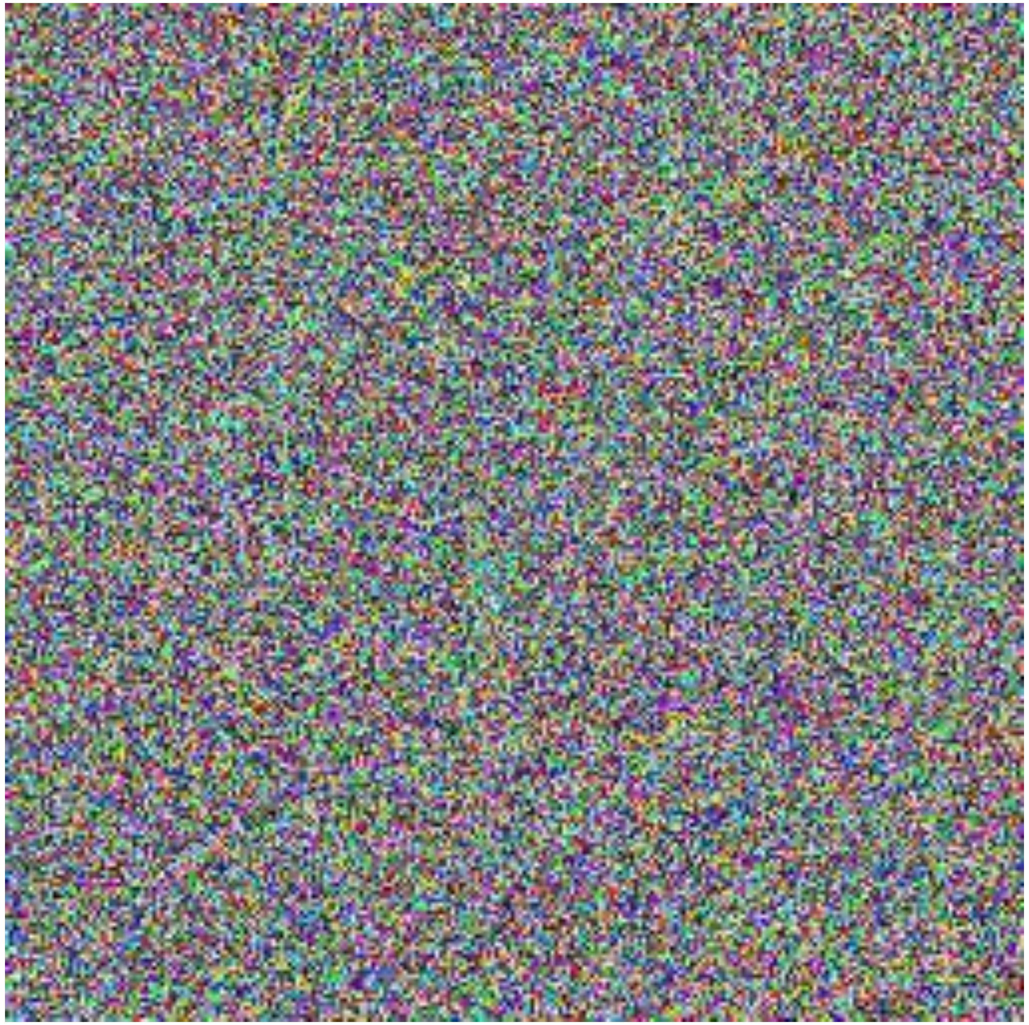
```
# apt-get install rng-tools
# echo bcm2708_rng >> /etc/modules
```



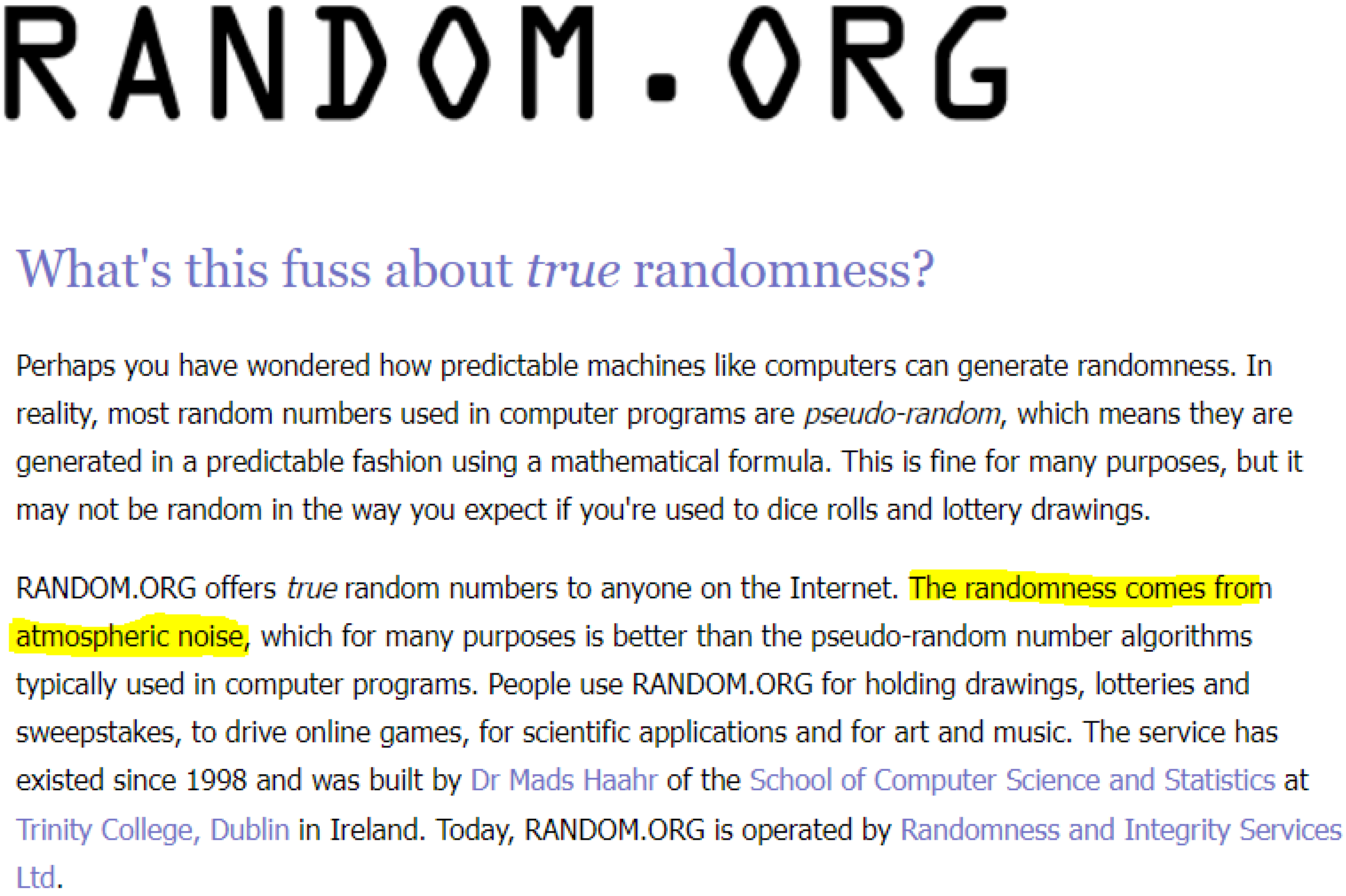
IC2 is the SoC and RAM. It's the large module (12.5×12.5 mm) in the center of the board, between the yellow RCA connector and the orange-topped HDMI connector, to the right of the "Raspberry Pi" logo. The Samsung SDRAM is stacked on top of the Broadcom BCM2835 SoC.
IC3 is the combined USB and Ethernet controller. It's the chip between the blue audio connector, the USB connector and the Ethernet connector.

Random Number Generation

```
sudo cat /dev/hwrng  | rawtoppm -rgb 256 256 | pnmtopng > random$(date +%Y%m%d%H%M%S).png
```



Random Number Generation



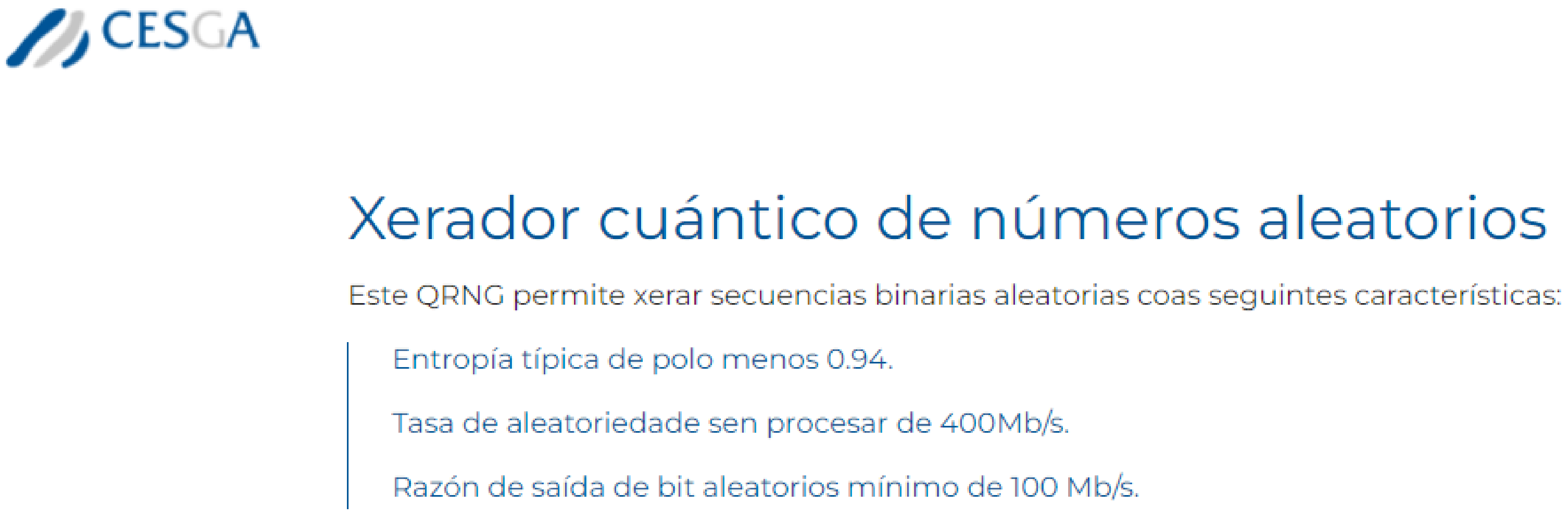
RANDOM.ORG

What's this fuss about *true* randomness?

Perhaps you have wondered how predictable machines like computers can generate randomness. In reality, most random numbers used in computer programs are *pseudo-random*, which means they are generated in a predictable fashion using a mathematical formula. This is fine for many purposes, but it may not be random in the way you expect if you're used to dice rolls and lottery drawings.

RANDOM.ORG offers *true* random numbers to anyone on the Internet. The randomness comes from atmospheric noise, which for many purposes is better than the pseudo-random number algorithms typically used in computer programs. People use RANDOM.ORG for holding drawings, lotteries and sweepstakes, to drive online games, for scientific applications and for art and music. The service has existed since 1998 and was built by Dr Mads Haahr of the School of Computer Science and Statistics at Trinity College, Dublin in Ireland. Today, RANDOM.ORG is operated by Randomness and Integrity Services Ltd.

Random Number Generation



Xerador cuántico de números aleatorios

Este QRNG permite xerar secuencias binarias aleatorias coas seguintes características:

> Entropía típica de polo menos 0.94.
>
> Tasa de aleatoriedade sen procesar de 400Mb/s.
>
> Razón de saída de bit aleatorios mínimo de 100 Mb/s.

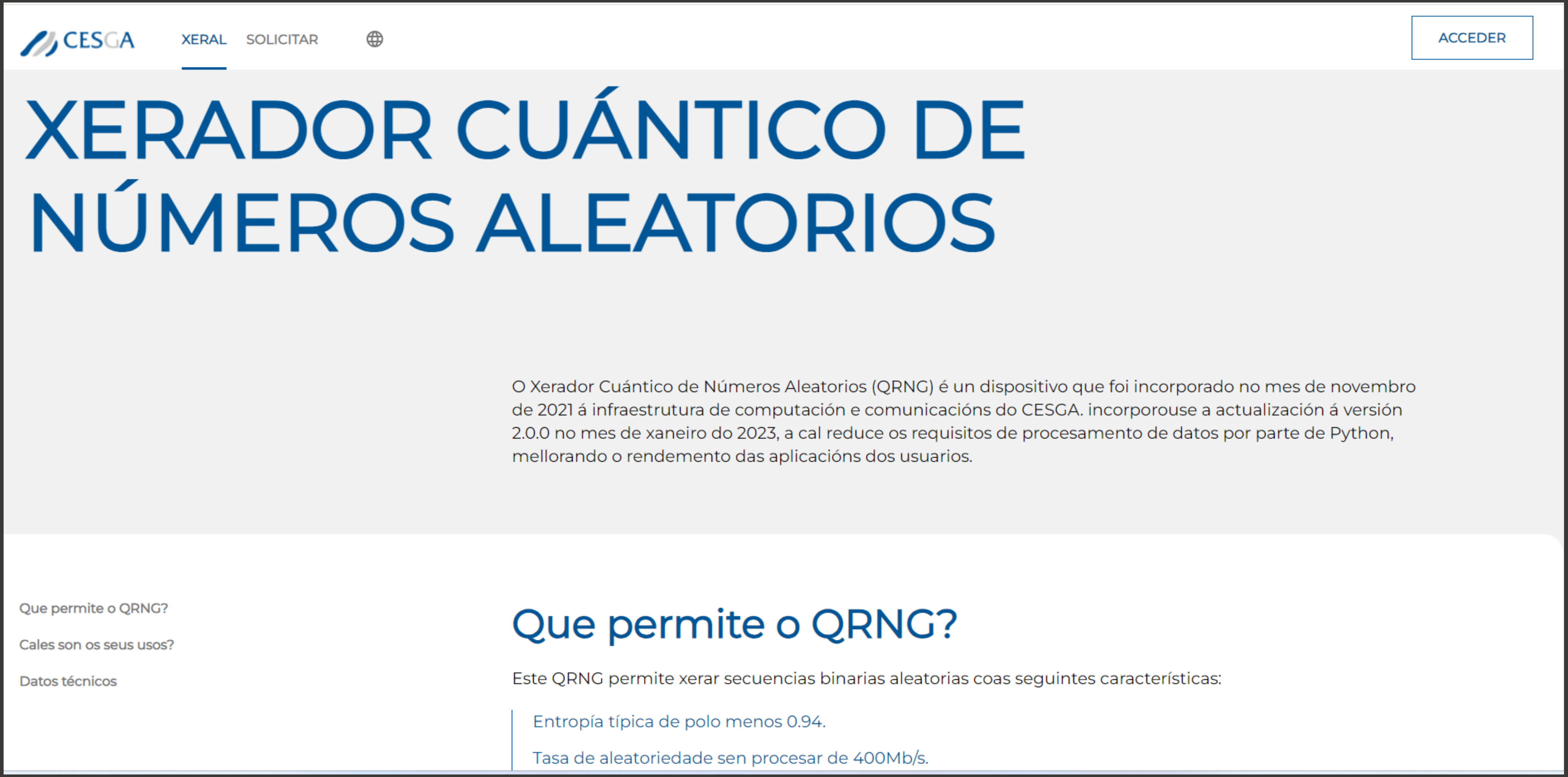
ALDABA

Interfaces de acceso (QRNG)

- Web pública & descarga de lote de números aleatorios
- API
- Librería Python
- Ejemplos de uso

ALDABA

Interfaces de acceso (QRNG): web pública



XERADOR CUÁNTICO DE NÚMEROS ALEATORIOS

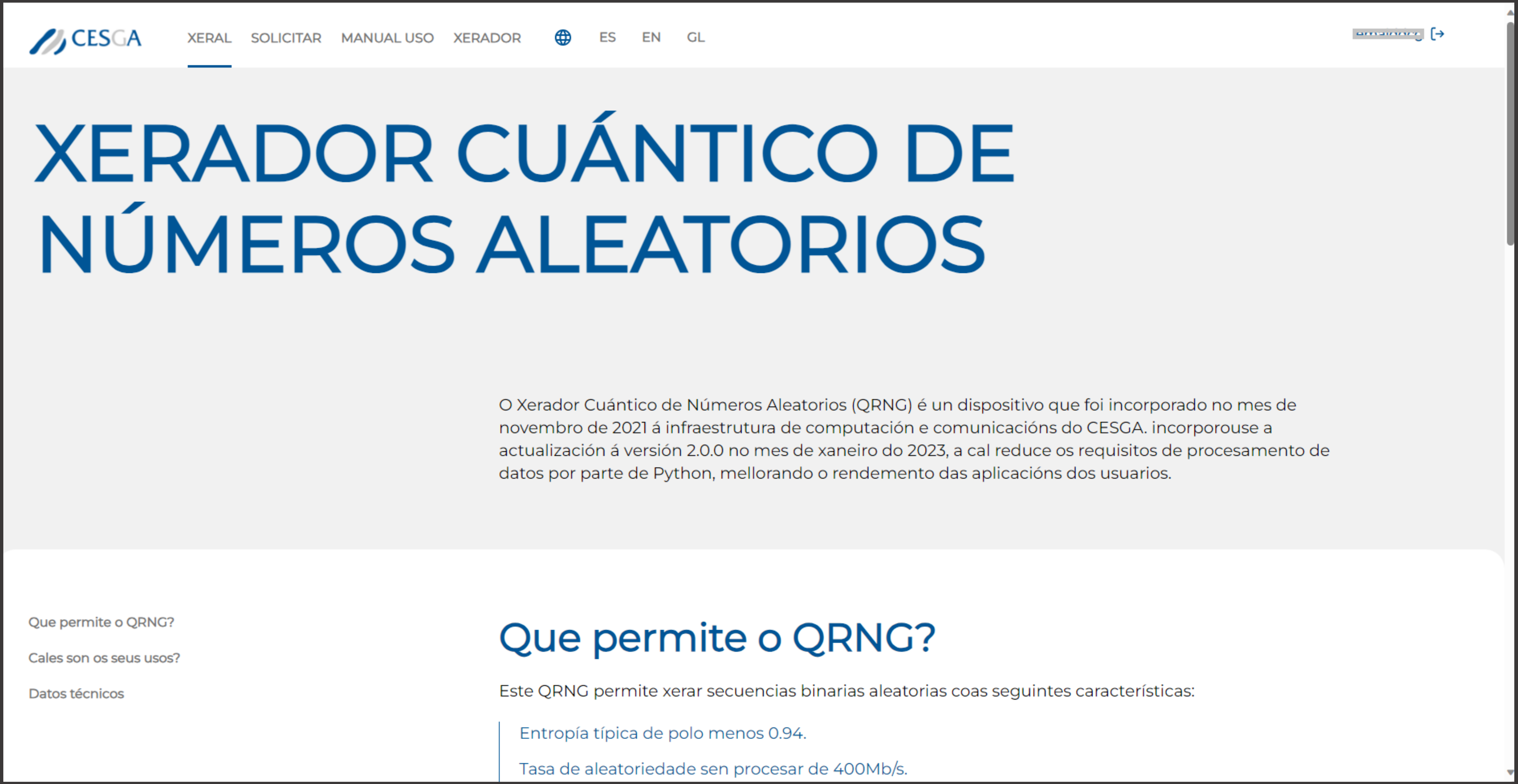
O Xerador Cuántico de Números Aleatorios (QRNG) é un dispositivo que foi incorporado no mes de novembro de 2021 á infraestrutura de computación e comunicacións do CESGA. incorporouse a actualización á versión 2.0.0 no mes de xaneiro do 2023, a cal reduce os requisitos de procesamento de datos por parte de Python, mellorando o rendemento das aplicacións dos usuarios.

Que permite o QRNG?

Este QRNG permite xerar secuencias binarias aleatorias coas seguintes características:

> Entropía típica de polo menos 0.94.
>
> Tasa de aleatoriedade sen procesar de 400Mb/s.

Interfaces de acceso (QRNG): web pública



XERADOR CUÁNTICO DE NÚMEROS ALEATORIOS

O Xerador Cuántico de Números Aleatorios (QRNG) é un dispositivo que foi incorporado no mes de novembro de 2021 á infraestrutura de computación e comunicacións do CESGA. incorporouse a actualización á versión 2.0.0 no mes de xaneiro do 2023, a cal reduce os requisitos de procesamento de datos por parte de Python, mellorando o rendemento das aplicacións dos usuarios.

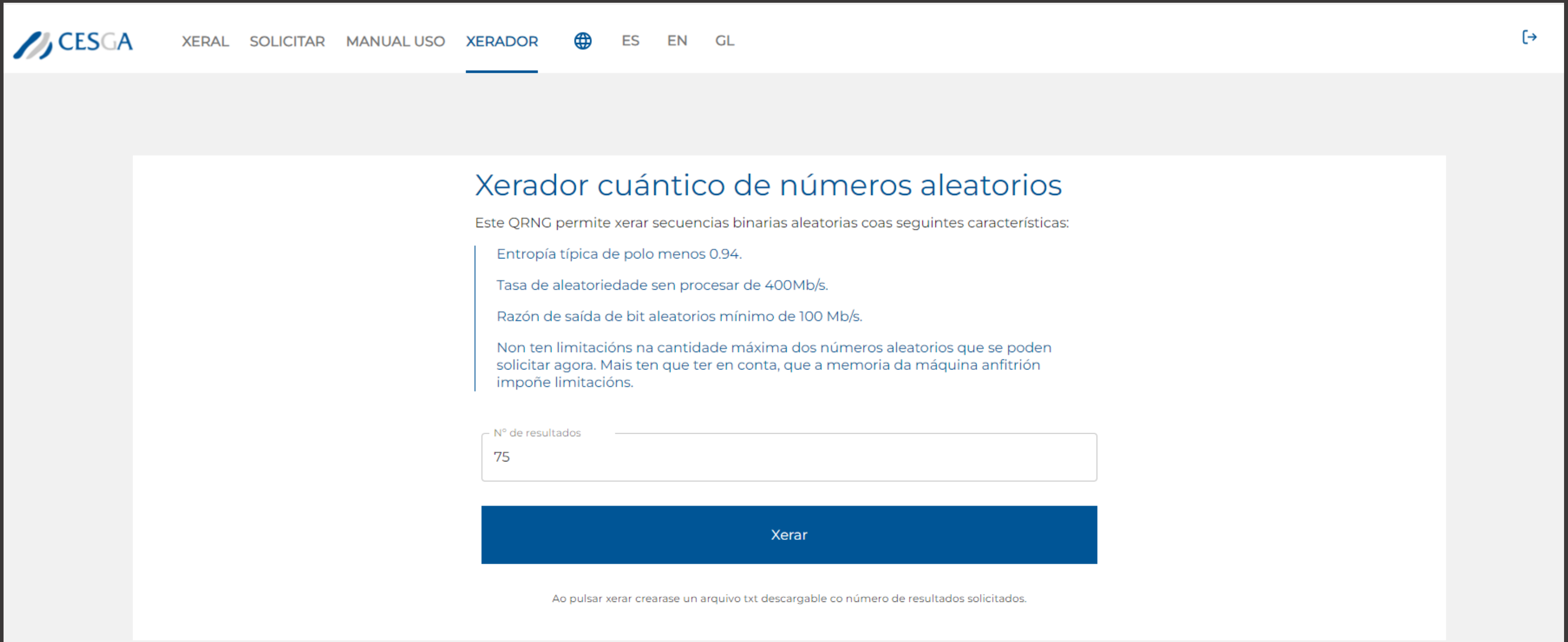
Que permite o QRNG?

Este QRNG permite xerar secuencias binarias aleatorias coas seguintes características:

> Entropía típica de polo menos 0.94.
>
> Tasa de aleatoriedade sen procesar de 400Mb/s.

ALDABA

Interfaces de acceso (QRNG): web pública



Xerador cuántico de números aleatorios

Este QRNG permite xerar secuencias binarias aleatorias coas seguintes características:

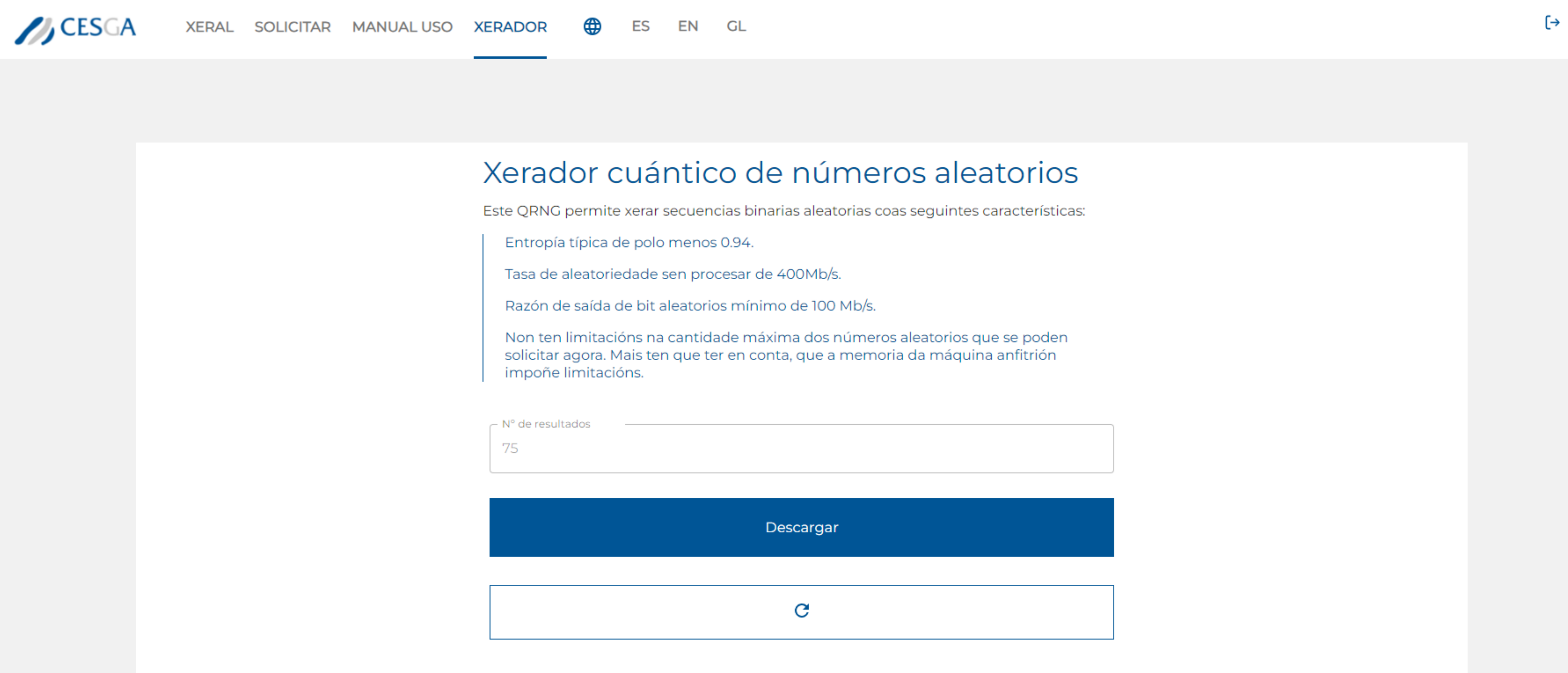
> Entropía típica de polo menos 0.94.
>
> Tasa de aleatoriedade sen procesar de 400Mb/s.
>
> Razón de saída de bit aleatorios mínimo de 100 Mb/s.
>
> Non ten limitacións na cantidade máxima dos números aleatorios que se poden solicitar agora. Mais ten que ter en conta, que a memoria da máquina anfitrión impoñe limitacións.

Nº de resultados: 75

Ao pulsar xerar crearase un arquivo txt descargable co número de resultados solicitados.

ALDABA

Interfaces de acceso (QRNG): web pública



Xerador cuántico de números aleatorios

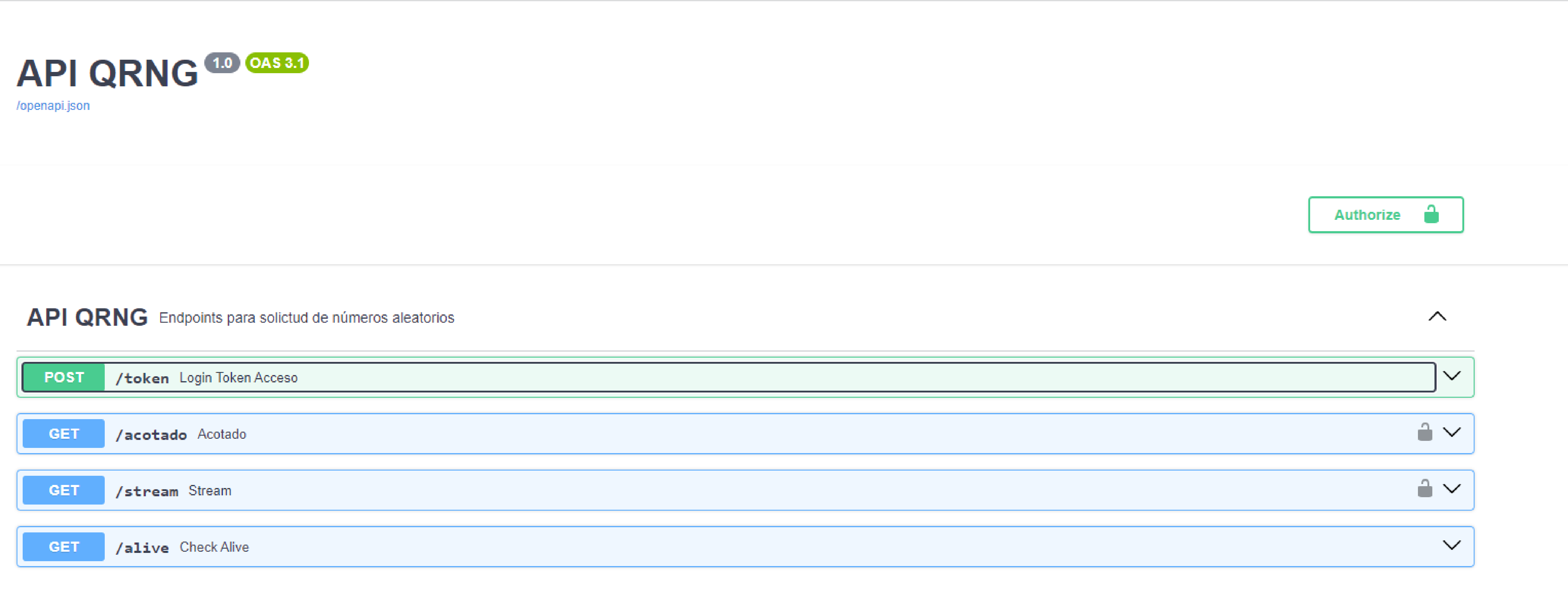
Este QRNG permite xerar secuencias binarias aleatorias coas seguintes características:

> Entropía típica de polo menos 0.94.
>
> Tasa de aleatoriedade sen procesar de 400Mb/s.
>
> Razón de saída de bit aleatorios mínimo de 100 Mb/s.
>
> Non ten limitacións na cantidade máxima dos números aleatorios que se poden solicitar agora. Mais ten que ter en conta, que a memoria da máquina anfitrión impoñe limitacións.

Nº de resultados: 75

ALDABA

Interfaces de acceso (QRNG): API



API QRNG 1.0 OAS 3.1

/openapi.json

API QRNG Endpoints para solictud de números aleatorios

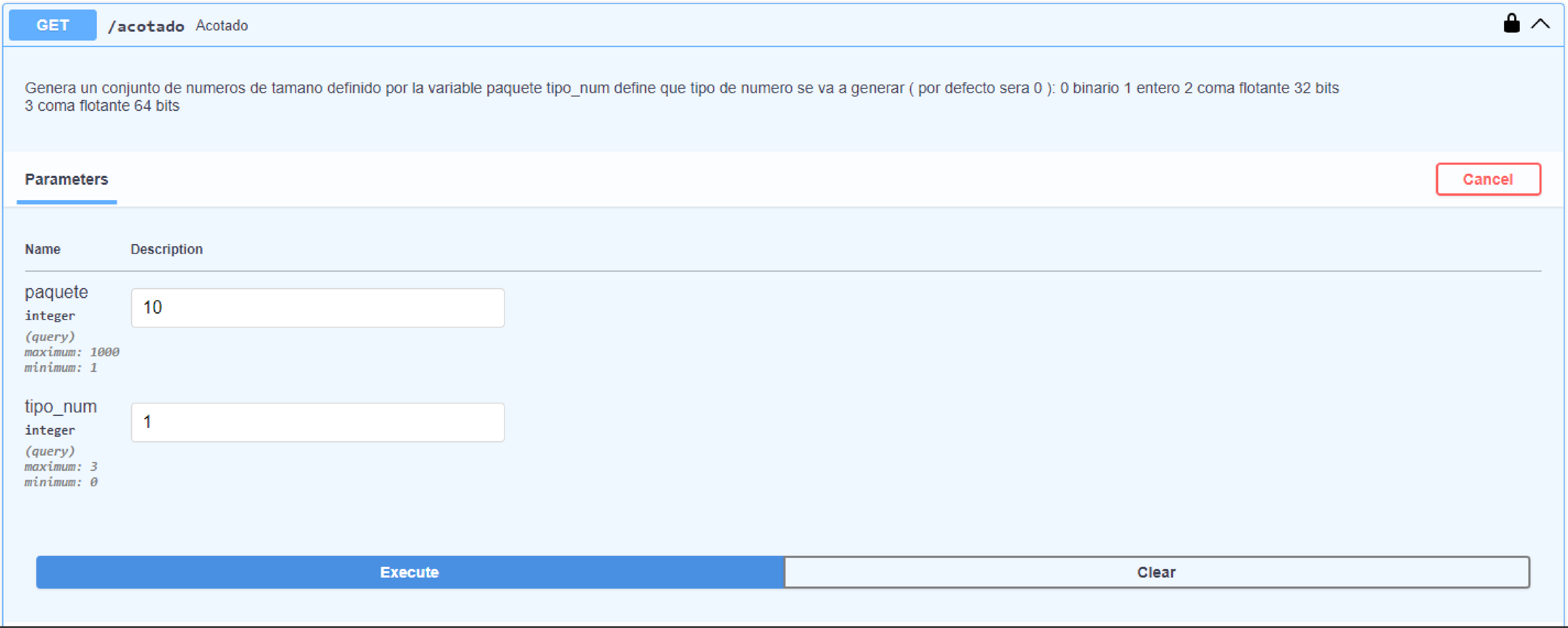
POST `/token` Login Token Acceso

GET `/acotado` Acotado

GET `/stream` Stream

GET `/alive` Check Alive

Interfaces de acceso (QRNG): API



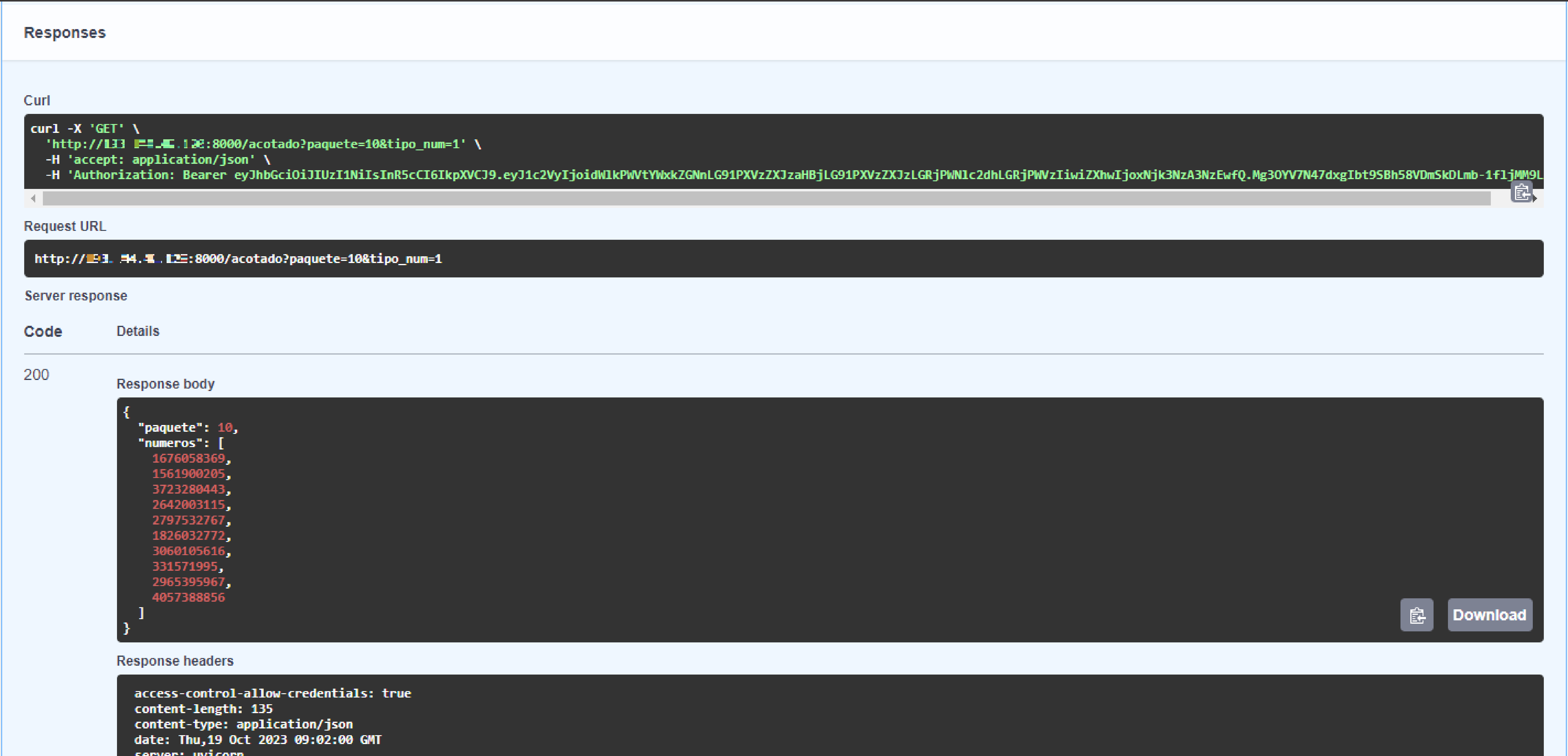
GET /acotado Acotado

Genera un conjunto de numeros de tamano definido por la variable paquete tipo_num define que tipo de numero se va a generar (por defecto sera 0): 0 binario 1 entero 2 coma flotante 32 bits 3 coma flotante 64 bits

Name	Description
paquete integer *(query)* *maximum: 1000* *minimum: 1*	10
tipo_num integer *(query)* *maximum: 3* *minimum: 0*	1

ALDABA

Interfaces de acceso (QRNG): API



Responses

Curl

```
curl -X 'GET' \
  'http://[redacted]:8000/acotado?paquete=10&tipo_num=1' \
  -H 'accept: application/json' \
  -H 'Authorization: Bearer eyJhbGciOiJIUzI1NiIsInR5cCI6IkpXVCJ9.eyJ1c2VyIjoidWlkPWVtYWxkZGNnLG91PXVzZXJzaHBjLG91PXVzZXJzLGRjPWNlc2dhLGRjPWVzIiwiZXhwIjoxNjk3NzA3NzEwfQ.Mg3OYV7N47dxgIbt9SBh58VDmSkDLmb-1fljMM9L
```

Request URL

```
http://[redacted]:8000/acotado?paquete=10&tipo_num=1
```

Server response

Code Details

200

Response body

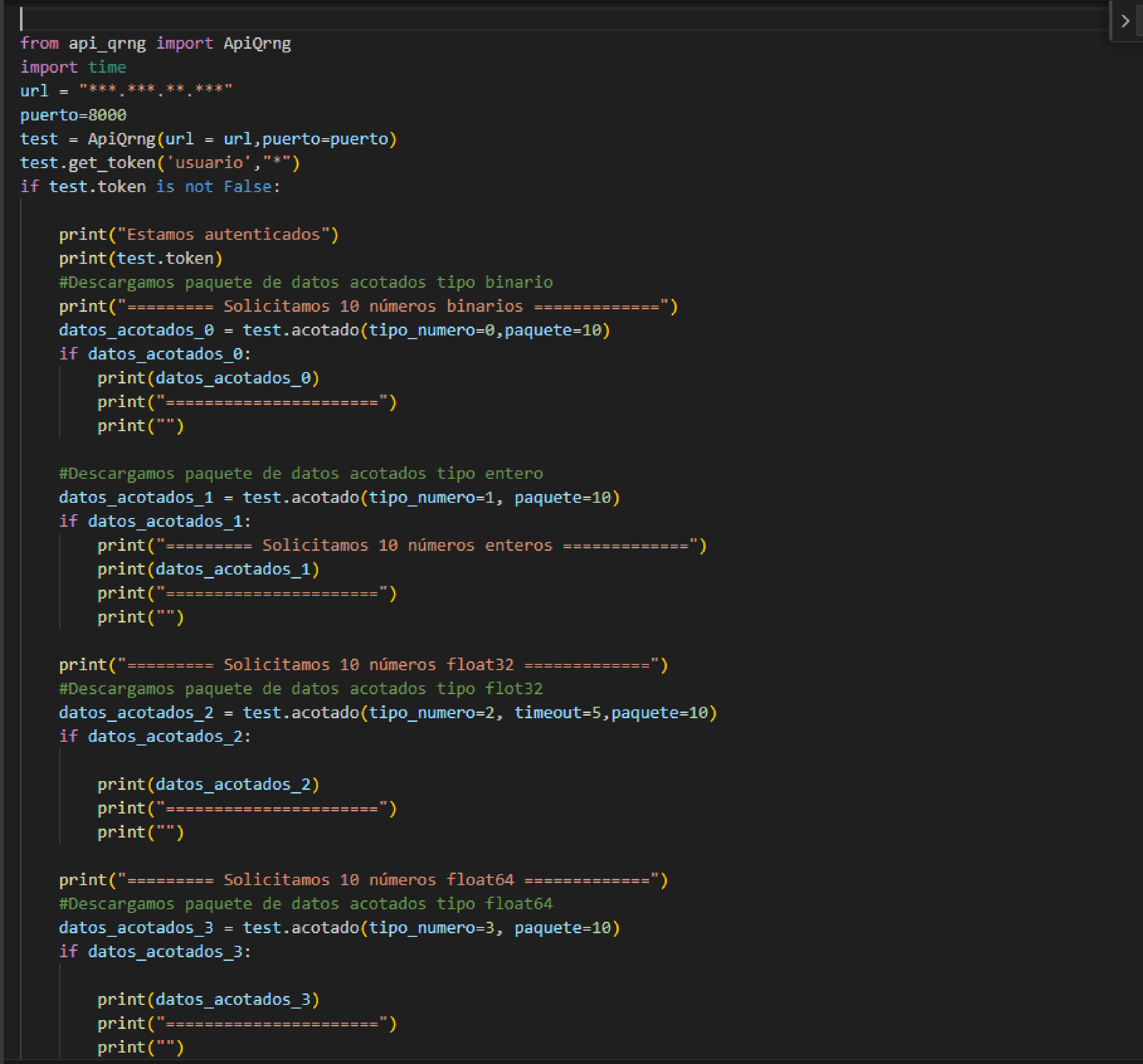
```
{
  "paquete": 10,
  "numeros": [
    1676058369,
    1561900205,
    3723280443,
    2642003115,
    2797532767,
    1826032772,
    3060105616,
    331571995,
    2965395967,
    4057388856
  ]
}
```

Response headers

```
 access-control-allow-credentials: true
 content-length: 135
 content-type: application/json
 date: Thu,19 Oct 2023 09:02:00 GMT
 server: uvicorn
```

ALDABA

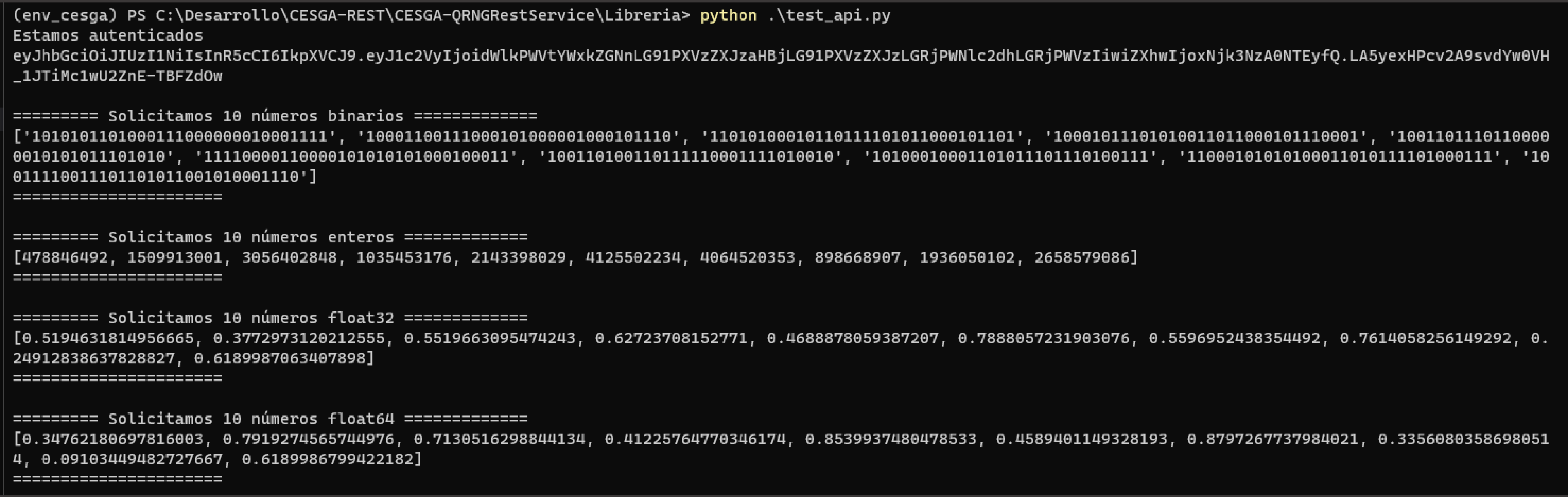
Interfaces de acceso (QRNG): Librería



```python
from api_qrng import ApiQrng
import time
url = "***.***.**.***"
puerto=8000
test = ApiQrng(url = url,puerto=puerto)
test.get_token('usuario',"*")
if test.token is not False:

    print("Estamos autenticados")
    print(test.token)
    #Descargamos paquete de datos acotados tipo binario
    print("========= Solicitamos 10 números binarios ============")
    datos_acotados_0 = test.acotado(tipo_numero=0,paquete=10)
    if datos_acotados_0:
        print(datos_acotados_0)
        print("======================")
        print("")

    #Descargamos paquete de datos acotados tipo entero
    datos_acotados_1 = test.acotado(tipo_numero=1, paquete=10)
    if datos_acotados_1:
        print("========= Solicitamos 10 números enteros ============")
        print(datos_acotados_1)
        print("======================")
        print("")

    print("========= Solicitamos 10 números float32 ============")
    #Descargamos paquete de datos acotados tipo flot32
    datos_acotados_2 = test.acotado(tipo_numero=2, timeout=5,paquete=10)
    if datos_acotados_2:

        print(datos_acotados_2)
        print("======================")
        print("")

    print("========= Solicitamos 10 números float64 ============")
    #Descargamos paquete de datos acotados tipo float64
    datos_acotados_3 = test.acotado(tipo_numero=3, paquete=10)
    if datos_acotados_3:

        print(datos_acotados_3)
        print("======================")
        print("")
```

ALDABA

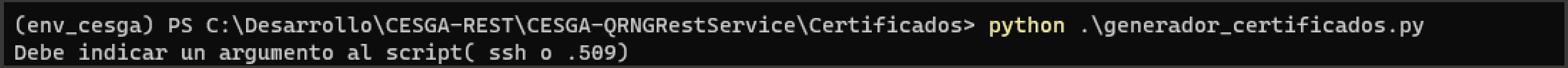
Interfaces de acceso (QRNG): Librería



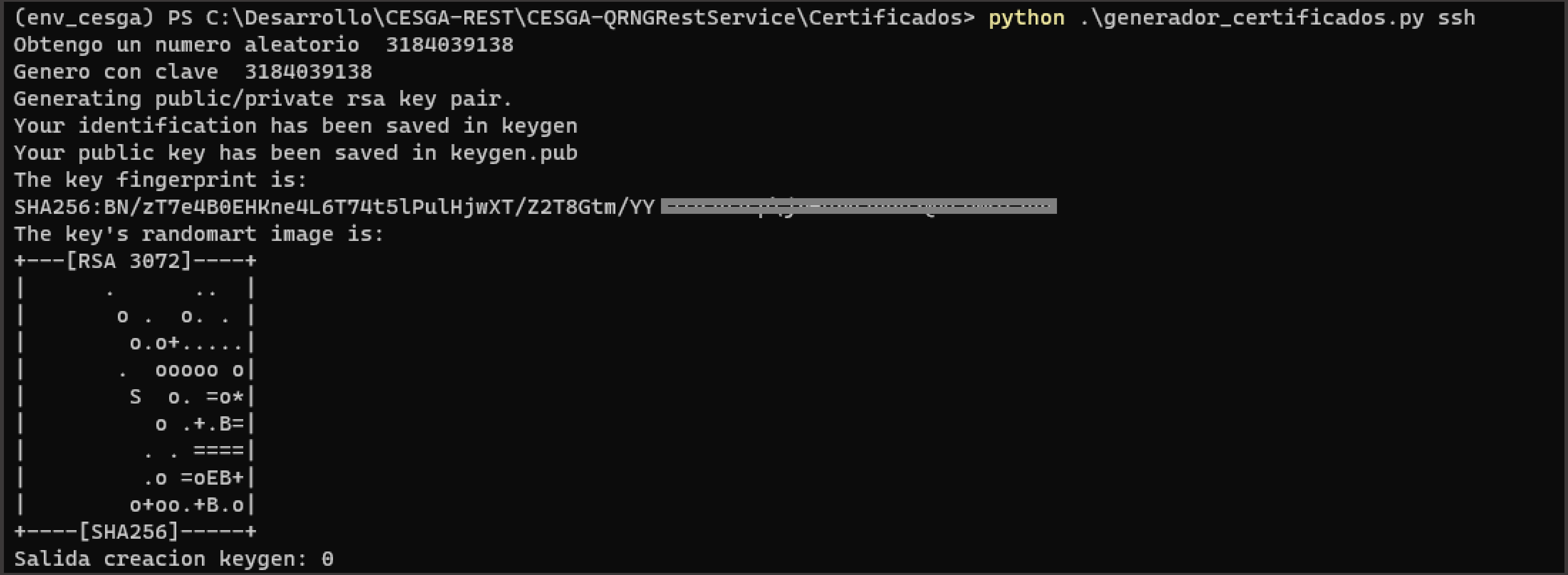
```
(env_cesga) PS C:\Desarrollo\CESGA-REST\CESGA-QRNGRestService\Libreria> python .\test_api.py
Estamos autenticados
eyJhbGciOiJIUzI1NiIsInR5cCI6IkpXVCJ9.eyJ1c2VyIjoidWlkPWVtYWxkZGNnLG91PXVzZXJzaHBjLG91PXVzZXJzLGRjPWNlc2dhLGRjPWVzIiwiZXhwIjoxNjk3NzA0NTEyfQ.LA5yexHPcv2A9svdYw0VH
_1JTiMc1wU2ZnE-TBFZdOw

========= Solicitamos 10 números binarios =============
['10101011010001110000000010001111', '10001100111000101000001000101110', '11010100010110111101011000101101', '10001011101010011011000101110001', '1001101110110000
0010101011101010', '11110000110000101010101000100011', '10011010011011111100011111010010', '10100010001101011101110100111', '11000101010100011010111101000111', '10
01111001110110101100101000111 0']
======================
```

ALDABA

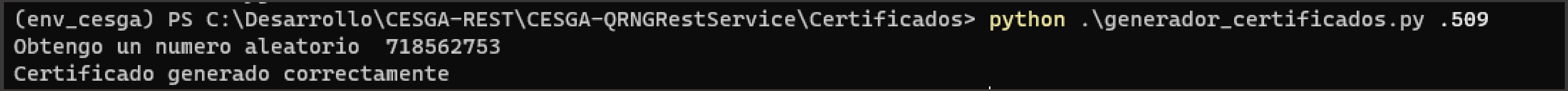
Interfaces de acceso (QRNG): Ejemplo de uso



```
(env_cesga) PS C:\Desarrollo\CESGA-REST\CESGA-QRNGRestService\Certificados> python .\generador_certificados.py
Debe indicar un argumento al script( ssh o .509)
```

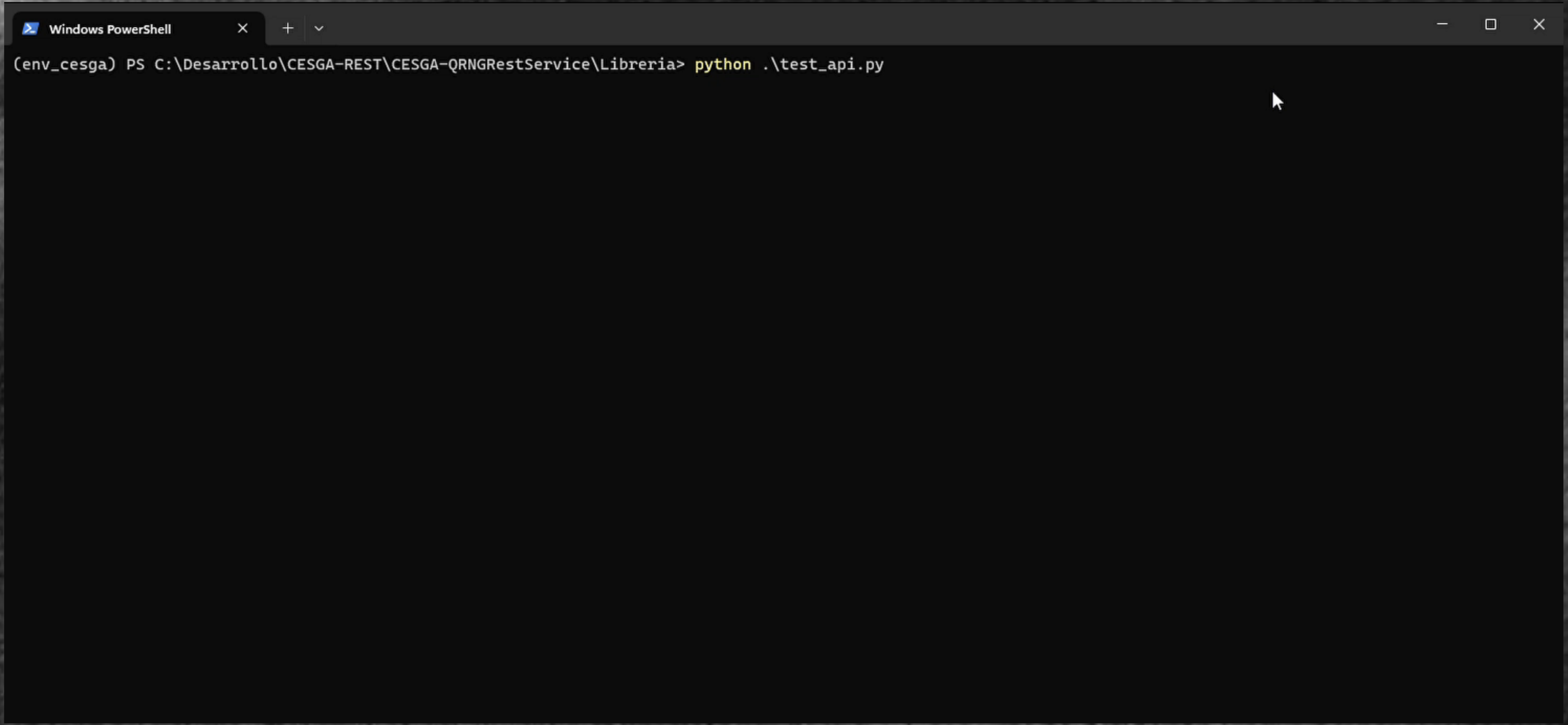


```
(env_cesga) PS C:\Desarrollo\CESGA-REST\CESGA-QRNGRestService\Certificados> python .\generador_certificados.py ssh
Obtengo un numero aleatorio  3184039138
Genero con clave  3184039138
Generating public/private rsa key pair.
Your identification has been saved in keygen
Your public key has been saved in keygen.pub
The key fingerprint is:
SHA256:BN/zT7e4B0EHKne4L6T74t5lPulHjwXT/Z2T8Gtm/YY [redacted]
The key's randomart image is:
+---[RSA 3072]----+
|      .      ..  |
|       o .  o. . |
|        o.o+.....|
|       .  ooooo o|
|        S  o. =o*|
|          o .+.B=|
|         . . ====|
|         .o =oEB+|
|        o+oo.+B.o|
+----[SHA256]-----+
Salida creacion keygen: 0
```



```
(env_cesga) PS C:\Desarrollo\CESGA-REST\CESGA-QRNGRestService\Certificados> python .\generador_certificados.py .509
Obtengo un numero aleatorio  718562753
Certificado generado correctamente
```

Gracias!



```
(env_cesga) PS C:\Desarrollo\CESGA-REST\CESGA-QRNGRestService\Libreria> python .\test_api.py
```

aldaba.es | info@aldaba.es | t. +34 981 90 20 60 | f. +34 981 90 20 05

A iniciativa do Polo de Tecnoloxías Cuánticas de Galicia conta con financiamento de:

Fondos REACT EU

AXENCIA GALEGA DE INNOVACIÓN

UNIÓN EUROPEA

Xacobeo 21·22

CESGA
Centro de Supercomputación de Galicia

Despregamento dunha infraestrutura baseada en tecnoloxías cuánticas da información que permita impulsar a I+D+i en Galicia.

Apoiar a transición cara a unha economía dixital.

Operación financiada pola Unión Europea, a través do FONDO EUROPEO DE DESENVOLVEMENTO REXIONAL (FEDER), como parte da resposta da Unión á pandemia da COVID-19.

PROGRAMA OPERATIVO
FEDER GALICIA
2014-2020

Unha maneira de facer Europa