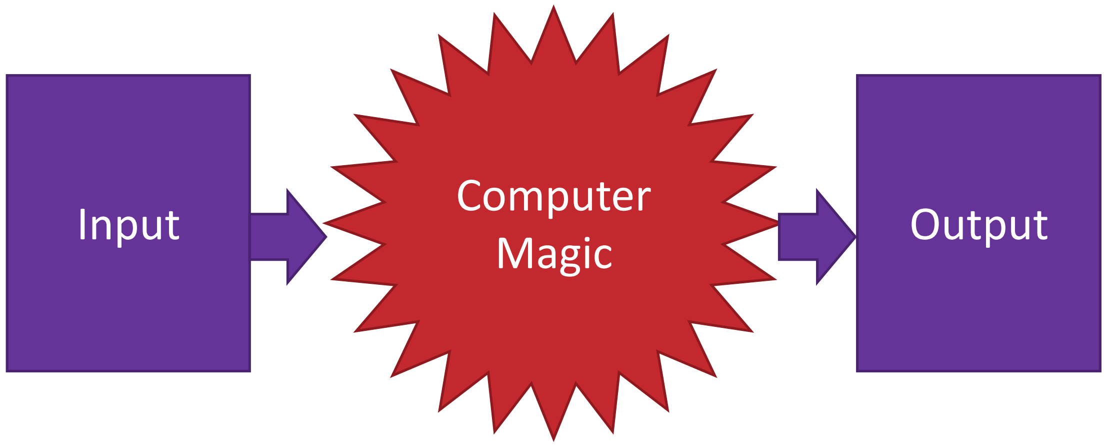
BISHOPFOX

YOU'RE DOING IOT RNG

DOING RNG

RANDOM NUMBERS

- Random numbers are very important to security
  - Encryption keys
  - Authentication tokens
  - Business logic



- **Computers are notoriously bad at making random numbers**
  - They inherently only do deterministic functions

- Hardware RNG
  - Makes entropy
  - Solves the problem… right?

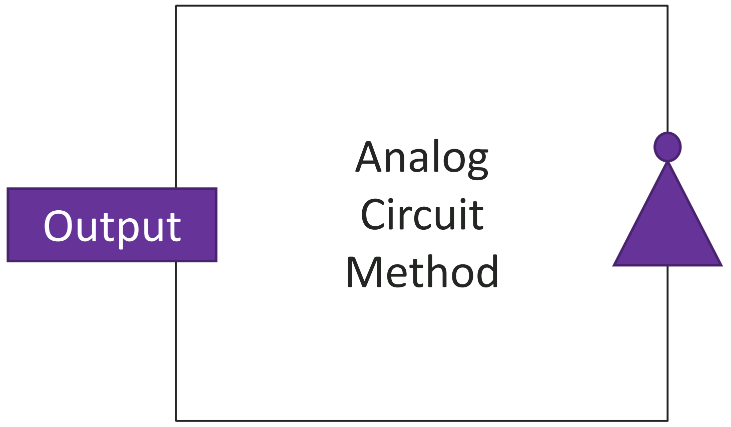
DOING RNG

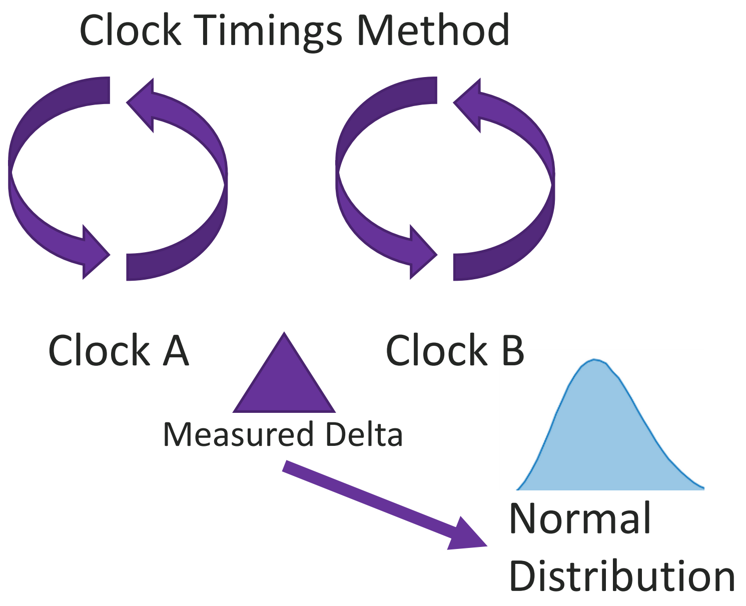
RANDOM NUMBERS

- PRNG (Pseudo RNG)
  - Cryptographically Secure
  - “Regular”
- “True” RNG (TRNG)
  - Terrible name
  - Hardware RNG

HARDWARE RNG DESIGN

- Black boxes
  - Very little information
  - Except the STM32, great info!
- Two common implementations:
  - Analog circuit
  - Clock timings
- Issues that come up
  - Running too fast
  - Accidental syncing





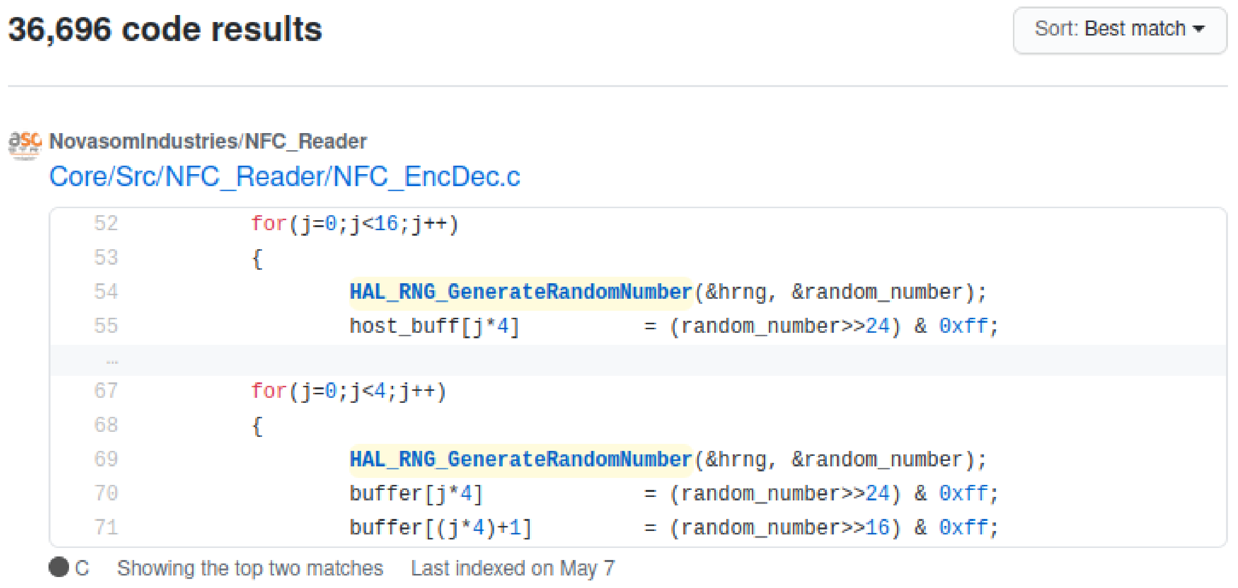
HOW IOT DOES RNG

- Most new IoT SoCs have a hardware RNG as of 2021
  - An entire hardware peripheral devoted to doing just this one thing
  - Surely it must be super secure
- No operating systems
  - IoT devices run C/C++ on bare metal
  - Call it like this:
  - `u8 hal_get_random_number(u32 *out_number);`
- Two parts we care about here:
  - Output variable (our random number)
  - Return code

NOBODY CHECKS ERROR CODES

(Two Examples)

FREERTOS


36,696 code results

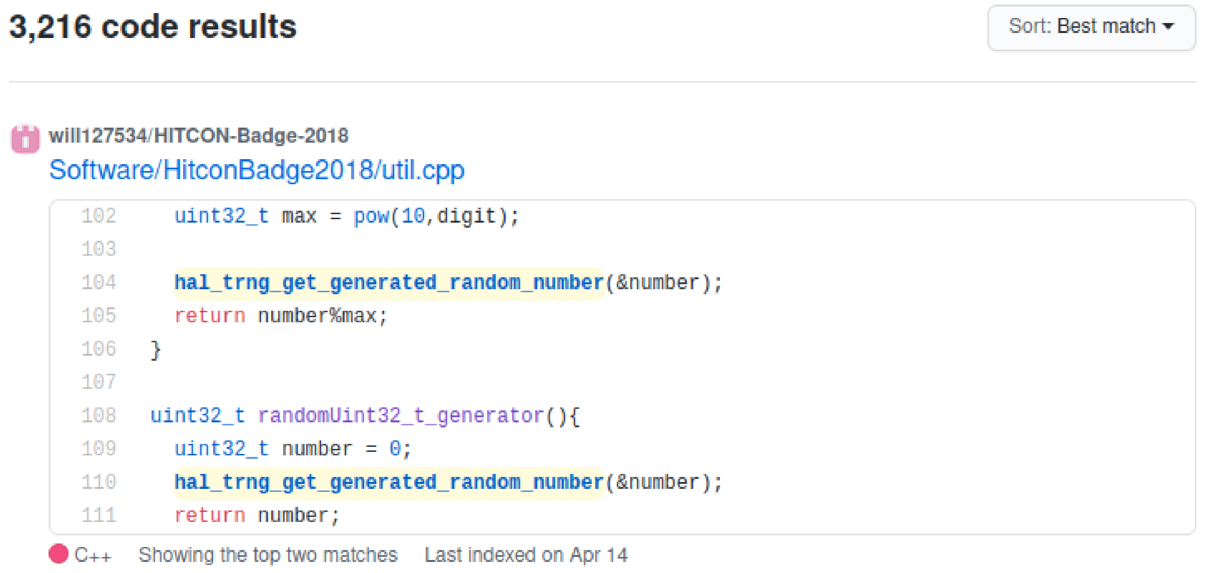
Sort: Best match

NovasomIndustries/NFC_Reader
Core/Src/NFC_Reader/NFC_EncDec.c

```
        for(j=0;j<16;j++)
        {
                HAL_RNG_GenerateRandomNumber(&hrng, &random_number);
                host_buff[j*4]          = (random_number>>24) & 0xff;
…
        for(j=0;j<4;j++)
        {
                HAL_RNG_GenerateRandomNumber(&hrng, &random_number);
                buffer[j*4]             = (random_number>>24) & 0xff;
                buffer[(j*4)+1]         = (random_number>>16) & 0xff;
```

● C   Showing the top two matches   Last indexed on May 7

MEDIATEK 7697


3,216 code results

Sort: Best match

will127534/HITCON-Badge-2018
Software/HitconBadge2018/util.cpp

```
  uint32_t max = pow(10,digit);

  hal_trng_get_generated_random_number(&number);
  return number%max;
}

uint32_t randomUint32_t_generator(){
  uint32_t number = 0;
  hal_trng_get_generated_random_number(&number);
  return number;
```

● C++   Showing the top two matches   Last indexed on Apr 14

PETRO’S LAW FORESHADOWING

WHAT’S THE WORST THAT COULD HAPPEN?

- What happens when the RNG call fails?

Undefined

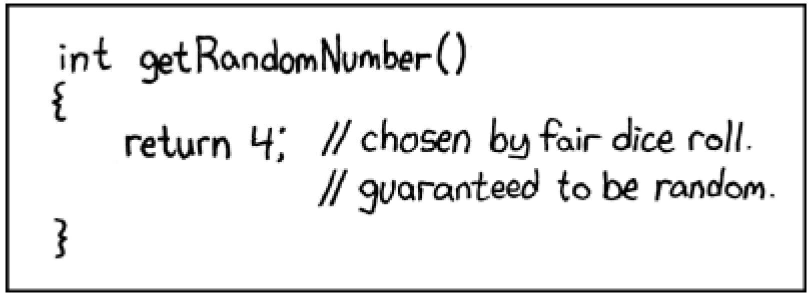
Behavior

FORESHADOWING

WHAT'S THE WORST THAT COULD HAPPEN?

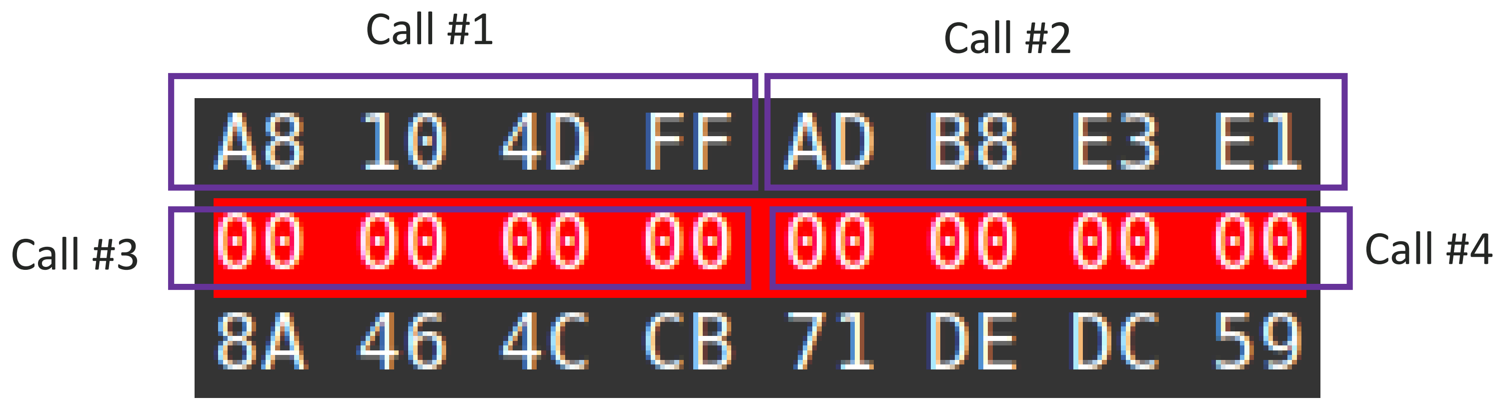
- Typically one of:
  - Partial entropy
  - The number 0

u8 hal_get_random_number(u32 ***out_number**);



```
int getRandomNumber()
{
    return 4;  // chosen by fair dice roll.
               // guaranteed to be random.
}
```

https://imgs.xkcd.com/comics/random_number.png



```
A8 10 4D FF AD B8 E3 E1
00 00 00 00 00 00 00 00
8A 46 4C CB 71 DE DC 59
```

PETRO’S LAW:
IT CAN ALWAYS BE WORSE

IT'S WORSE

WHAT'S THE WORST THAT COULD HAPPEN?

- Uninitialized memory

```
u32 random_number;
hal_get_random_number(&random_number);
// Sends over the network
packet_send(random_number);
```

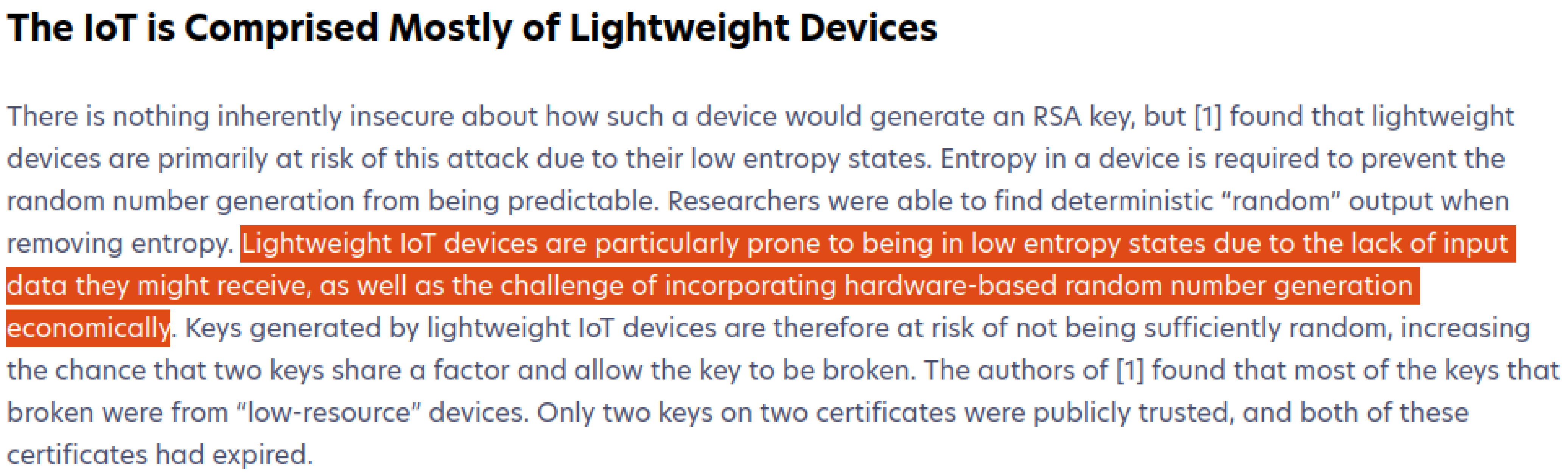
Arbitrary bytes from RAM!

REAL WORLD INSTANCES?

- RSA Certificate Vulnerability Factoring RSA Keys in the IoT Era
    - https://www.keyfactor.com/resources/factoring-rsa-keys-in-the-iot-era/

"435,000 weak certificates – 1 in 172 of the certificate we found on the Internet – are vulnerable to attack."

- Rationale:



The IoT is Comprised Mostly of Lightweight Devices

There is nothing inherently insecure about how such a device would generate an RSA key, but [1] found that lightweight devices are primarily at risk of this attack due to their low entropy states. Entropy in a device is required to prevent the random number generation from being predictable. Researchers were able to find deterministic "random" output when removing entropy. Lightweight IoT devices are particularly prone to being in low entropy states due to the lack of input data they might receive, as well as the challenge of incorporating hardware-based random number generation economically. Keys generated by lightweight IoT devices are therefore at risk of not being sufficiently random, increasing the chance that two keys share a factor and allow the key to be broken. The authors of [1] found that most of the keys that broken were from "low-resource" devices. Only two keys on two certificates were publicly trusted, and both of these certificates had expired.

DON'T BLAME THE USER

```
if(HAL_TRNG_STATUS_OK !=
hal_trng_get_generated_random_number(&random_number)) {
    //error handle
}
```

- **RANDOM NUMBERS ARE CRITICAL**

  You can't just "handle" the error and move forward without it

- **YOU'RE GIVEN TWO OPTIONS:**

  Spin-loop (using 100% CPU) indefinitely

  Quit out and kill the entire process

- **BOTH ARE UNACCEPTABLE**

  Will lead to buggy, broken, and useless devices

- **DEVELOPERS GO FOR OPTION THREE:**

  YOLO

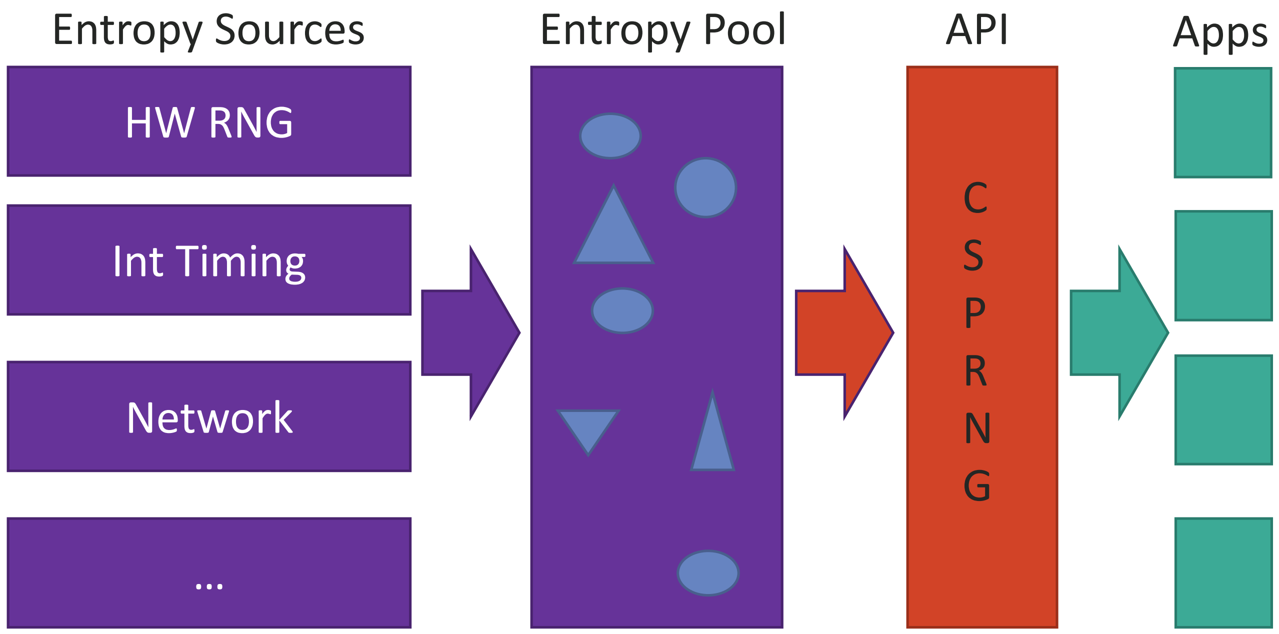
RNG IN IOT IS FUNDAMENTALLY BROKEN

IT IS NOT THE FAULT OF THE USER

DOING CSPRNG

THE RIGHT WAY TO RNG

- Cryptographically Secure Pseudorandom Number Generator (CSPRNG)
  - Never blocks execution
  - API calls never fail
  - Pools from many entropy sources
  - Always returns crypto-quality results
- This is how every major operating system works
  - Linux, MacOS, Windows, BSD, etc…
- ***This is how IoT should work too***



WHY YOU REALLY DO NEED A CSPRNG SUBSYSTEM

DOING IOT RNG WITHOUT KNOWING IT

USING HARDWARE RNG TO SEED AN INSECURE PRNG

- Nobody codes from scratch
  - In IoT, there's lots of reference and example code
  - When the reference code has vulnerabilities it propagates out

Contiki-NG: The OS for Next Generation IoT Devices

https://contiki-ng.readthedocs.io/en/release-v4.6/_api/arch_2cpu_2nrf52840_2dev_2random_8c_source.html

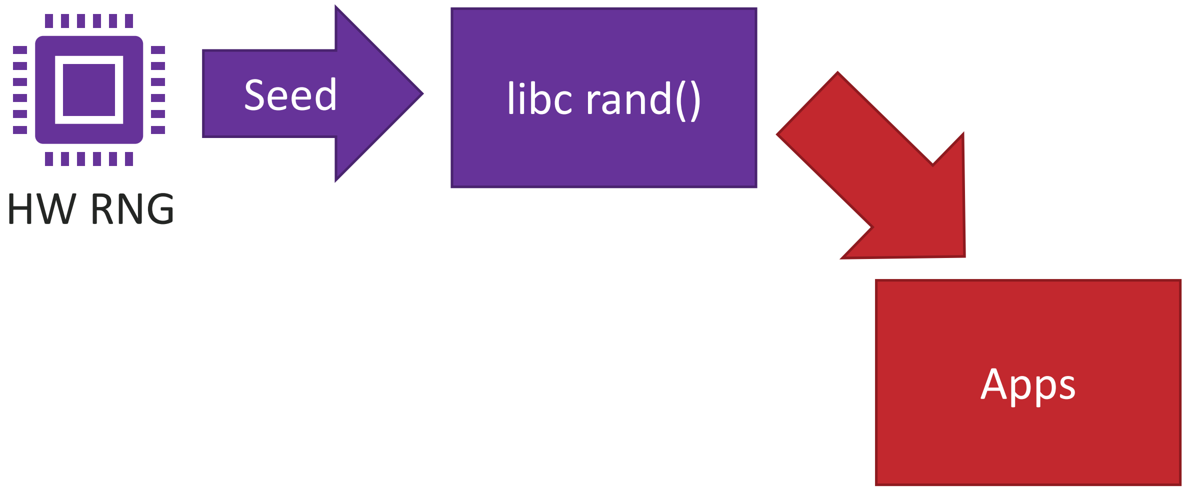
- Some IoT devices / Operating Systems use the hardware
  - But only to seed the ***<u>insecure</u>*** libc PRNG

Mediatek Linkit7697

https://github.com/MediaTek-Labs/Arduino-Add-On-for-LinkIt-SDK/blob/53acd43fbee57b034141068969fa643465dfd743/project/linkit7697_hdk/apps/wifi_demo/src/sys_init.c#L198

DOING IOT RNG WITHOUT KNOWING IT

USING HARDWARE RNG TO SEED AN INSECURE PRNG



- You're **not** really using the hardware
  - **Even though you think you are**

Contiki-NG: The OS for Next Generation IoT Devices

https://contiki-ng.readthedocs.io/en/release-v4.6/_api/arch_2cpu_2nrf52840_2dev_2random_8c_source.html

Mediatek Linkit7697

https://github.com/MediaTek-Labs/Arduino-Add-On-for-LinkIt-SDK/blob/53acd43fbee57b034141068969fa643465dfd743/project/linkit7697_hdk/apps/wifi_demo/src/sys_init.c#L198

DEMO

SORRY, BUG BOUNTY HUNTERS

EXPLOITABILITY

- Often very context-dependent
  - Not a simple CVE with exploit to apply
  - But still **very exploitable**
  - Just not canned

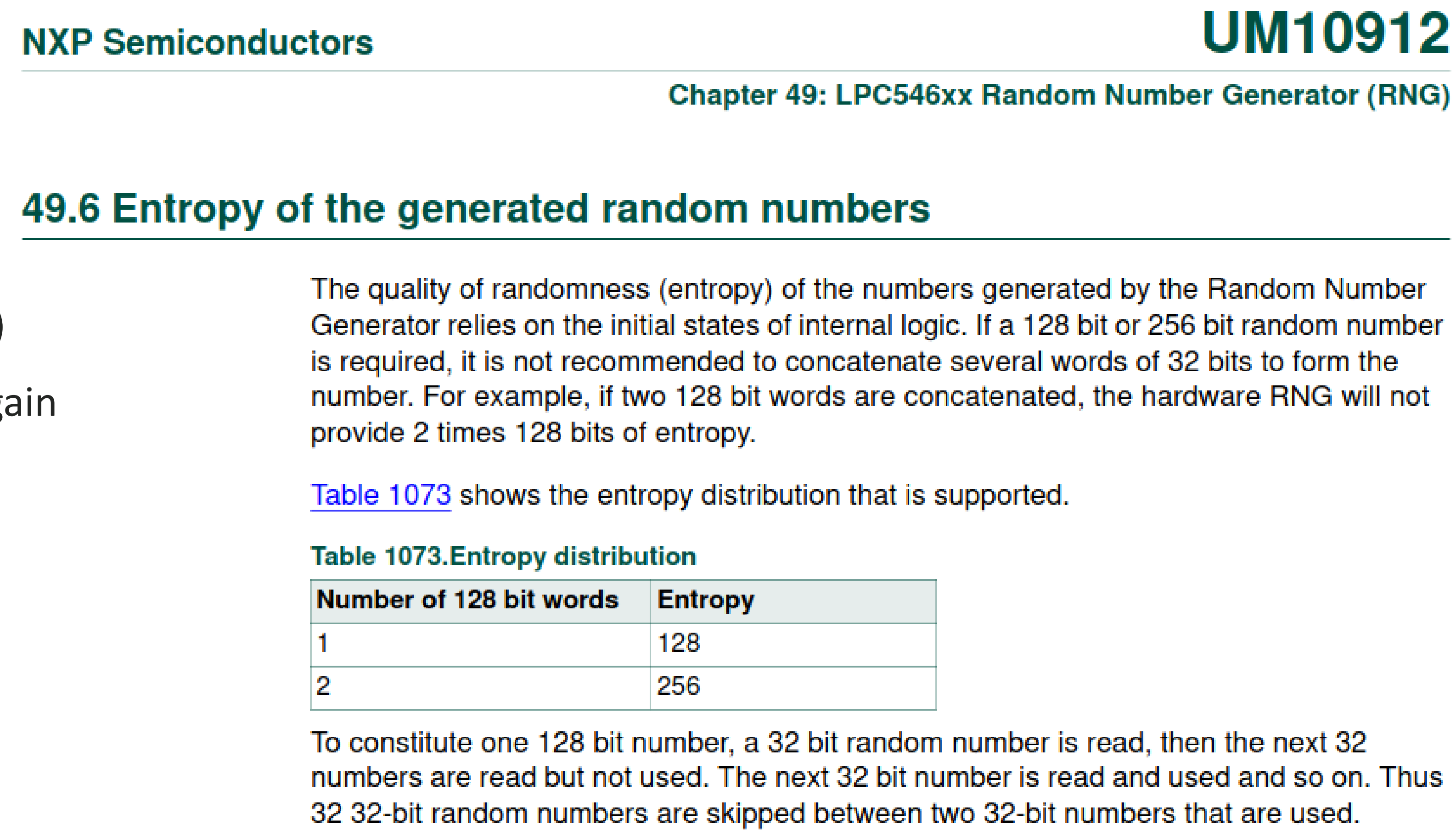
- Look at crypto keys, especially asymmetric keys
  - Very long (2048 bits or more)
  - Sensitive to low entropy

- Check out “The Mechanics of Compromising Low Entropy RSA Keys”
  - At this DEF CON
  - (we did not plan this)

EVEN WHEN IOT RNG FEELS RIGHT, YOU'RE STILL DOING IOT RNG

USAGE QUIRKS

- **Never** write your own code that interfaces with a hardware RNG
    - It's no different than crypto code
    - **You WILL do it wrong**
- Example: LPC 54628
    - Warning on page 1,106 (of 1,152)
    - Throw out 32 results, then use again



NXP Semiconductors **UM10912**

Chapter 49: LPC546xx Random Number Generator (RNG)

49.6 Entropy of the generated random numbers

The quality of randomness (entropy) of the numbers generated by the Random Number Generator relies on the initial states of internal logic. If a 128 bit or 256 bit random number is required, it is not recommended to concatenate several words of 32 bits to form the number. For example, if two 128 bit words are concatenated, the hardware RNG will not provide 2 times 128 bits of entropy.

Table 1073 shows the entropy distribution that is supported.

Table 1073.Entropy distribution

Number of 128 bit words	Entropy
1	128
2	256

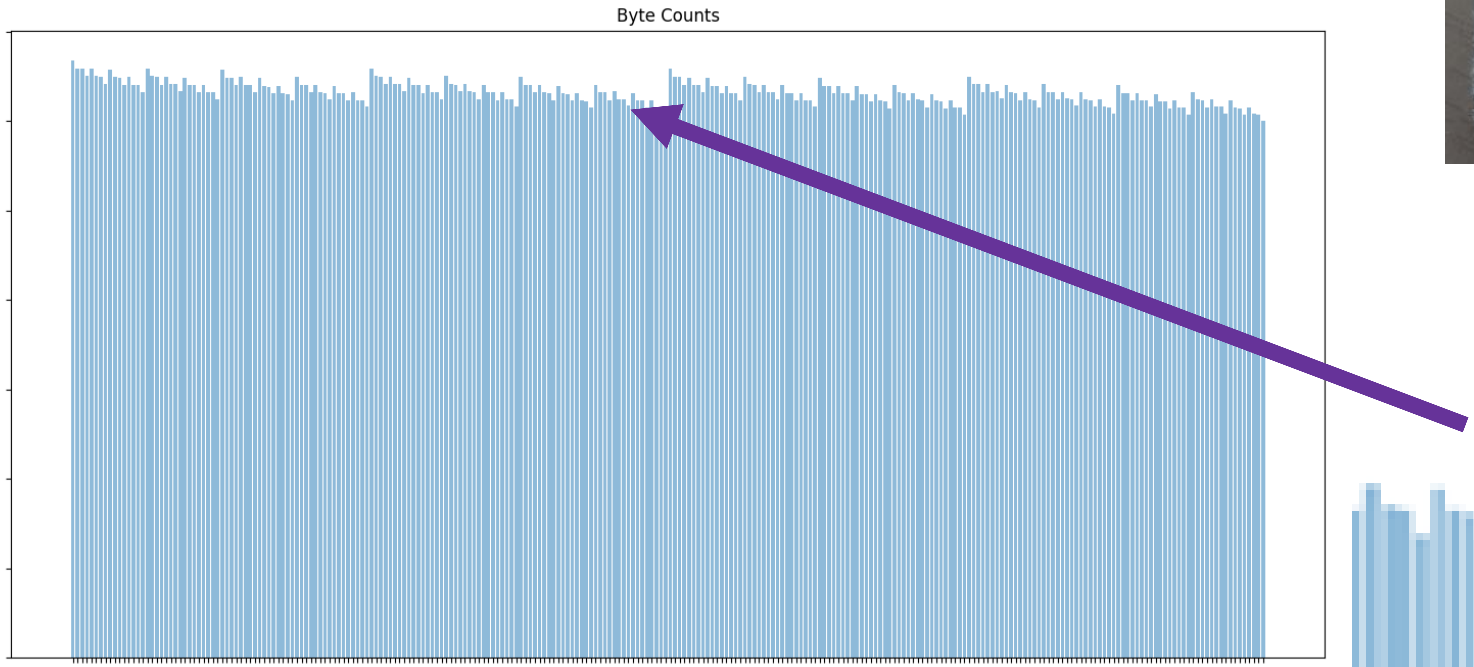
To constitute one 128 bit number, a 32 bit random number is read, then the next 32 numbers are read but not used. The next 32 bit number is read and used and so on. Thus 32 32-bit random numbers are skipped between two 32-bit numbers that are used.

THAT'S NOT SO RANDOM

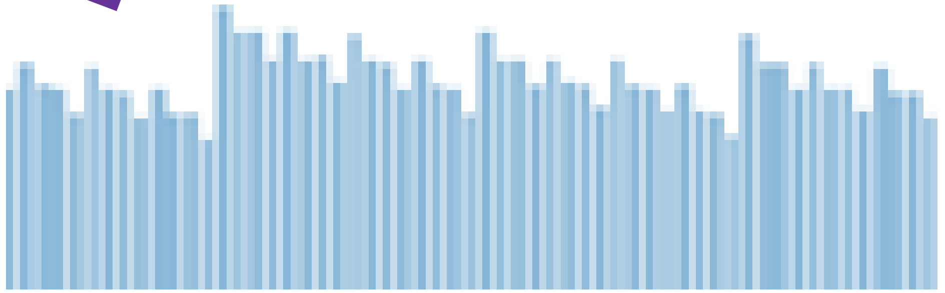
STATISTICAL ANALYSIS

- Mediatek 7697

LinkIt™ 7697
Module : wrtnode7
FCC ID:2ALR4-WRTNODE7
RX
TX
RST
PWR
USR
GND D+ D- 5V 3V3 RST GND P5 P4 P3 P2 P1 P0 GND
GND P17 P16 P15 P14 P13 P12 P11 P10 P9 P8 P7 P6 3V3
RTC_PWR



Histogram of the frequency of each byte 0 to 255

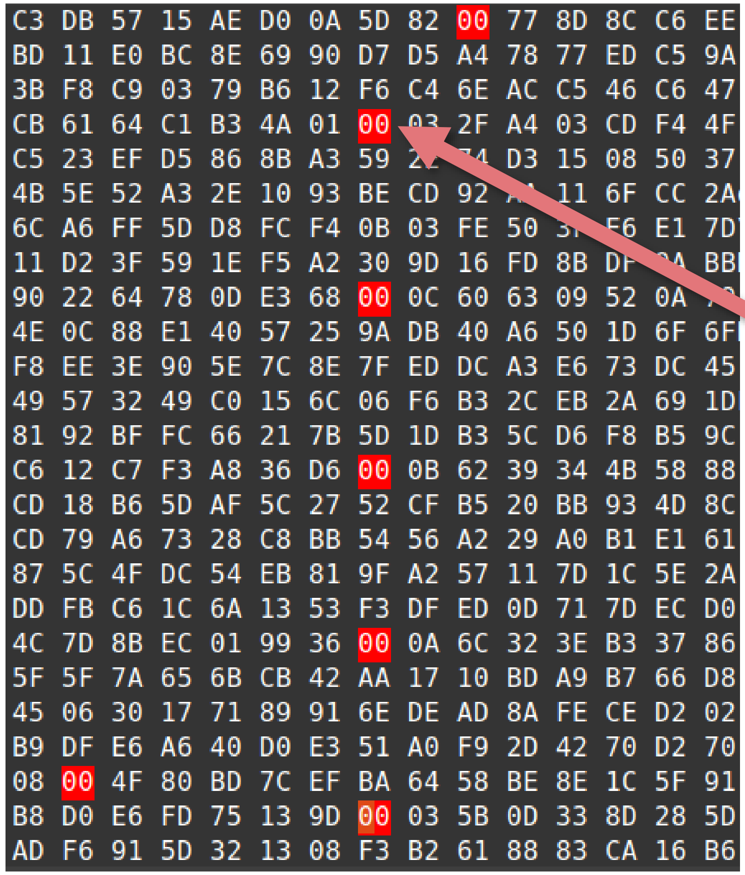


That's not very random…

NULL HYPOTHESIS

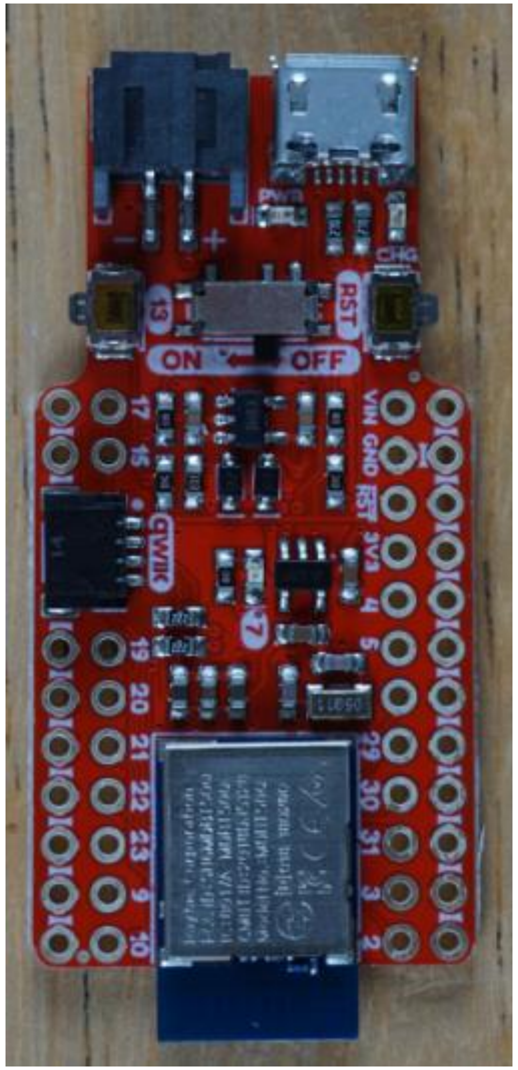
STATISTICAL ANALYSIS

- Nordic nrf52840



```
C3 DB 57 15 AE D0 0A 5D 82 00 77 8D 8C C6 EE
BD 11 E0 BC 8E 69 90 D7 D5 A4 78 77 ED C5 9A
3B F8 C9 03 79 B6 12 F6 C4 6E AC C5 46 C6 47
CB 61 64 C1 B3 4A 01 00 03 2F A4 03 CD F4 4F
C5 23 EF D5 86 8B A3 59 [illegible] 74 D3 15 08 50 37
4B 5E 52 A3 2E 10 93 BE CD 92 [illegible] 11 6F CC 2A
6C A6 FF 5D D8 FC F4 0B 03 FE 50 [illegible] F6 E1 7D
11 D2 3F 59 1E F5 A2 30 9D 16 FD 8B DF 0A BB
90 22 64 78 0D E3 68 00 0C 60 63 09 52 0A [illegible]
4E 0C 88 E1 40 57 25 9A DB 40 A6 50 1D 6F 6F
F8 EE 3E 90 5E 7C 8E 7F ED DC A3 E6 73 DC 45
49 57 32 49 C0 15 6C 06 F6 B3 2C EB 2A 69 1D
81 92 BF FC 66 21 7B 5D 1D B3 5C D6 F8 B5 9C
C6 12 C7 F3 A8 36 D6 00 0B 62 39 34 4B 58 88
CD 18 B6 5D AF 5C 27 52 CF B5 20 BB 93 4D 8C
CD 79 A6 73 28 C8 BB 54 56 A2 29 A0 B1 E1 61
87 5C 4F DC 54 EB 81 9F A2 57 11 7D 1C 5E 2A
DD FB C6 1C 6A 13 53 F3 DF ED 0D 71 7D EC D0
4C 7D 8B EC 01 99 36 00 0A 6C 32 3E B3 37 86
5F 5F 7A 65 6B CB 42 AA 17 10 BD A9 B7 66 D8
45 06 30 17 71 89 91 6E DE AD 8A FE CE D2 02
B9 DF E6 A6 40 D0 E3 51 A0 F9 2D 42 70 D2 70
08 00 4F 80 BD 7C EF BA 64 58 BE 8E 1C 5F 91
B8 D0 E6 FD 75 13 9D 00 03 5B 0D 33 8D 28 5D
AD F6 91 5D 32 13 08 F3 B2 61 88 83 CA 16 B6
```

Repeating 0x000 pattern
Every 0x50 bytes



STATISTICAL ANALYSIS

- STM32-L432KC

```
#=============================================================================#
#            dieharder version 3.31.1 Copyright 2003 Robert G. Brown          #
#=============================================================================#
rng_name|filename rands/second|file_input_raw|STM32L432randData.bin|1.72e+07  |
#=============================================================================#
        test_name   |ntup| tsamples |psamples|  p-value |Assessment
#=============================================================================#
rgb_minimum_distance|   0|     10000|    1000|0.00000000|  FAILED
```

CONCLUSIONS

YOU’RE DOING IOT RNG

CONCLUSIONS

- ✓ This affects the whole IoT industry
  - Not a single vendor or device
- ✓ The IoT needs a CSPRNG subsystem
  - This can’t be fixed by changing documentation and blaming users
- ✓ RNG code should be considered dangerous to write on your own
  - Just like crypto code
- ✓ Never use entropy directly from the hardware
  - You don’t know how strong or weak it may be

WHAT YOU CAN DO ABOUT IT

CONCLUSIONS

- ☑ Device Owners
  - Keep an eye out for updates
- ☑ IoT Device Developers
  - Use one of the emerging IoT Operating Systems
- ☑ IoT Device Manufacturers / IoT OS Developers
  - Implement CSPRNG subsystems, deprecate / disallow users from reading directly from the hardware
- ☑ Pen Testers
  - This will probably be a perennial finding for years to come

THANK YOU

- Dan “AltF4” Petro
- Allan Cecil (dwangoAC)