

# Safe randomness: theory and practice

# 安全なランダムネスの理論と実践

# Kenji Rikitake

りきたけ けんじ

7-SEP-2018

Builderscon Tokyo 2018

Kyoseikan, Keio University

Yokohama City, Kanagawa,

Japan

@jj1bdx

Copyright ©2018 Kenji Rikitake.

This work is licensed under a

Creative Commons Attribution

4.0 International License.



In this talk I'm going to talk about  
**Randomness**

この発表ではランダムネスについて話します

What is randomness?  
... unpredictability

ランダムネスとは予測不能性のことです

Randomness is essential for  
secure operation

ランダムネスは安全な運用に不可欠です

# When randomness needed

ランダムネスが必要な時

- Password/key generation / パスワードや鍵の生成
- Timing obfuscation / 処理時間を隠す
- Using multiple resources equally but unpredictably / 複数の資源を同じように、しかし予測されないように使いたい

In algorithm, randomness is represented as:

# Random numbers

アルゴリズムでのランダムネスは  
**乱数**によって表現します

My works on random numbers  
乱数について何をやってきたか

# Software contribution to Erlang/OTP

- Improve the random number algorithms
- 亂数アルゴリズムの改善
- Erlang/OTP rand module
- SFMT for Erlang/OTP
- TinyMT calculation of 256M keys

# Bad algorithm example (JS V8)



# Legacy Erlang/OTP random module

- A 1980s algorithm called AS183
- Can be fully scanned in 8 hours
- Became a security issue - deprecated since OTP 19 (June 2016)
- 8時間で全数検索できてしまう
- セキュリティ問題になりOTPバージョン19（2016年6月）より非推奨

... And hardware contribution  
because software is not enough  
ソフトだけでは不十分なのでハードもやってます

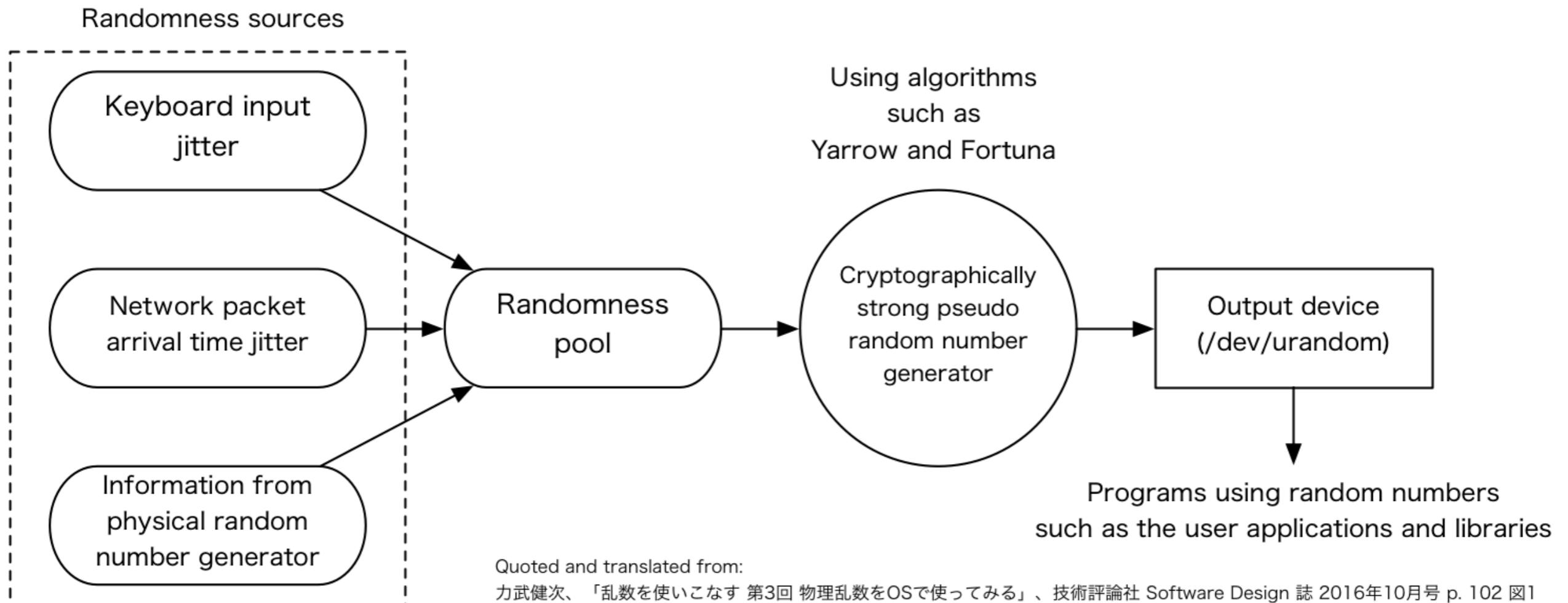
# Why hardware?

- Computers are *programmed* and *predictive* machines; finding randomness inside computers is *extremely difficult*
- コンピュータはプログラムされた通りに、予想通りに動く→コンピュータの中でランダムネスを見つけるのは非常に難しい

# Randomness sources in a computer

- CPU clock jitter / CPUクロックの揺れ
- Keyboard timing / キーボード打鍵のタイミング
- Network packet timing / パケットのタイミング
- Storage seeking timing / ストレージのタイミング
- ... Those sources are highly predictive
- ... これらのソースは予測可能

# Randomness processing flow



# Little randomness available in a system

システムの中からはランダムネスは少ししか得られない

## A result: only ~0.62bit/sec

- A dormant Linux server without attached keyboard
- `/proc/sys/kernel/random/entropy_avail`
- Bits of entropy (= randomness) in the system
- 258 bits / 415.6 seconds (~7 minutes)

Additional randomness needed  
追加のランダムネスが必要

# Why? Because:

Security depends on unpredictability

Secure operations consume randomness

Availability of randomness is limited

セキュリティは予測不能性に依存している

安全な処理はランダムネスを消費する

使えるランダムネスは有限

# Physical randomness source

物理的なランダムネス源

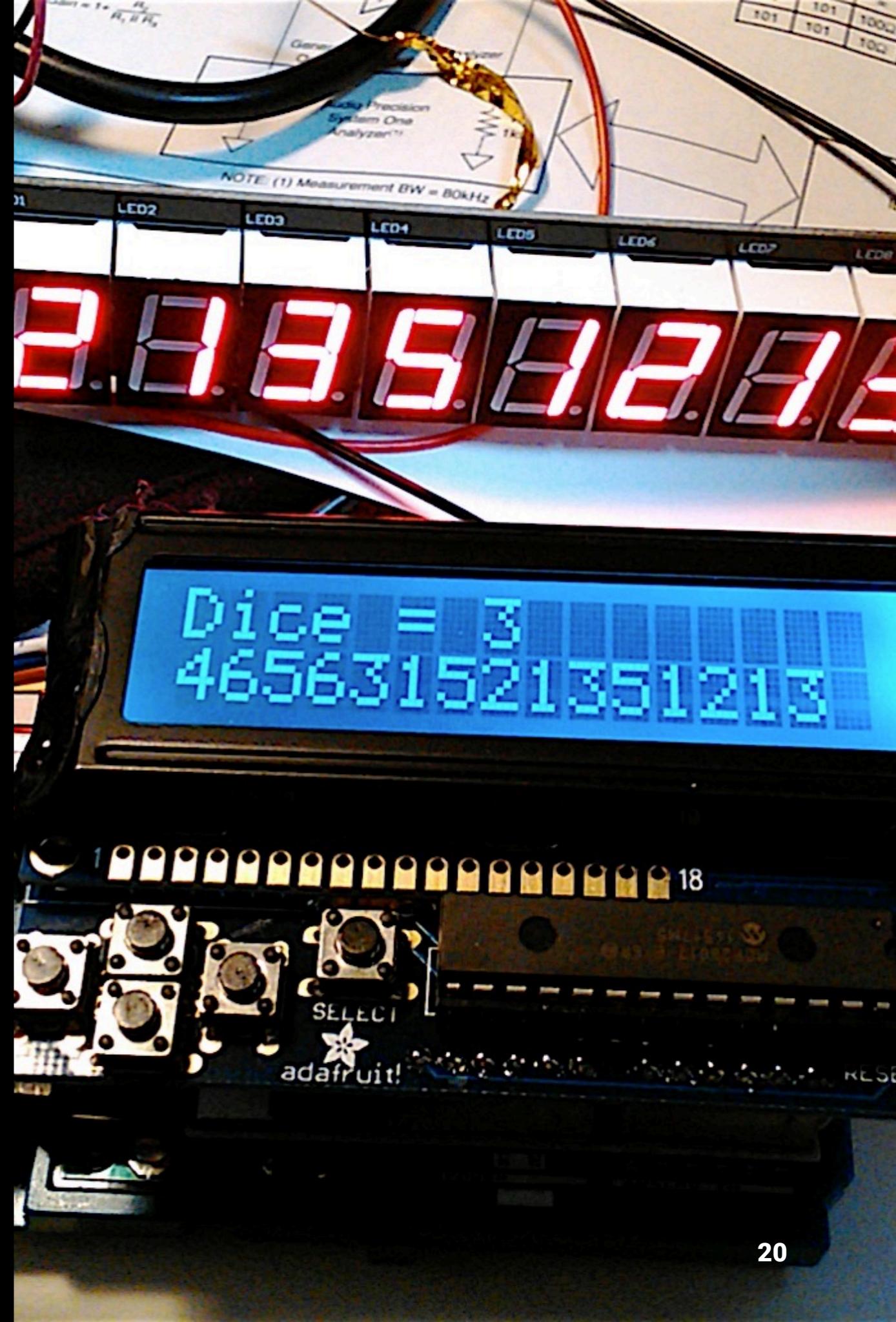
- Thermal noise / 热雑音
- Avalanche noise of semiconductor junctions / 半導体接合部のなだれ降伏雑音
- Timing jitter of oscillation circuits / 発振回路のタイミングの揺れ

# Physical random number generator with Arduino UNO

Displayed at Maker Faire Tokyo  
2016

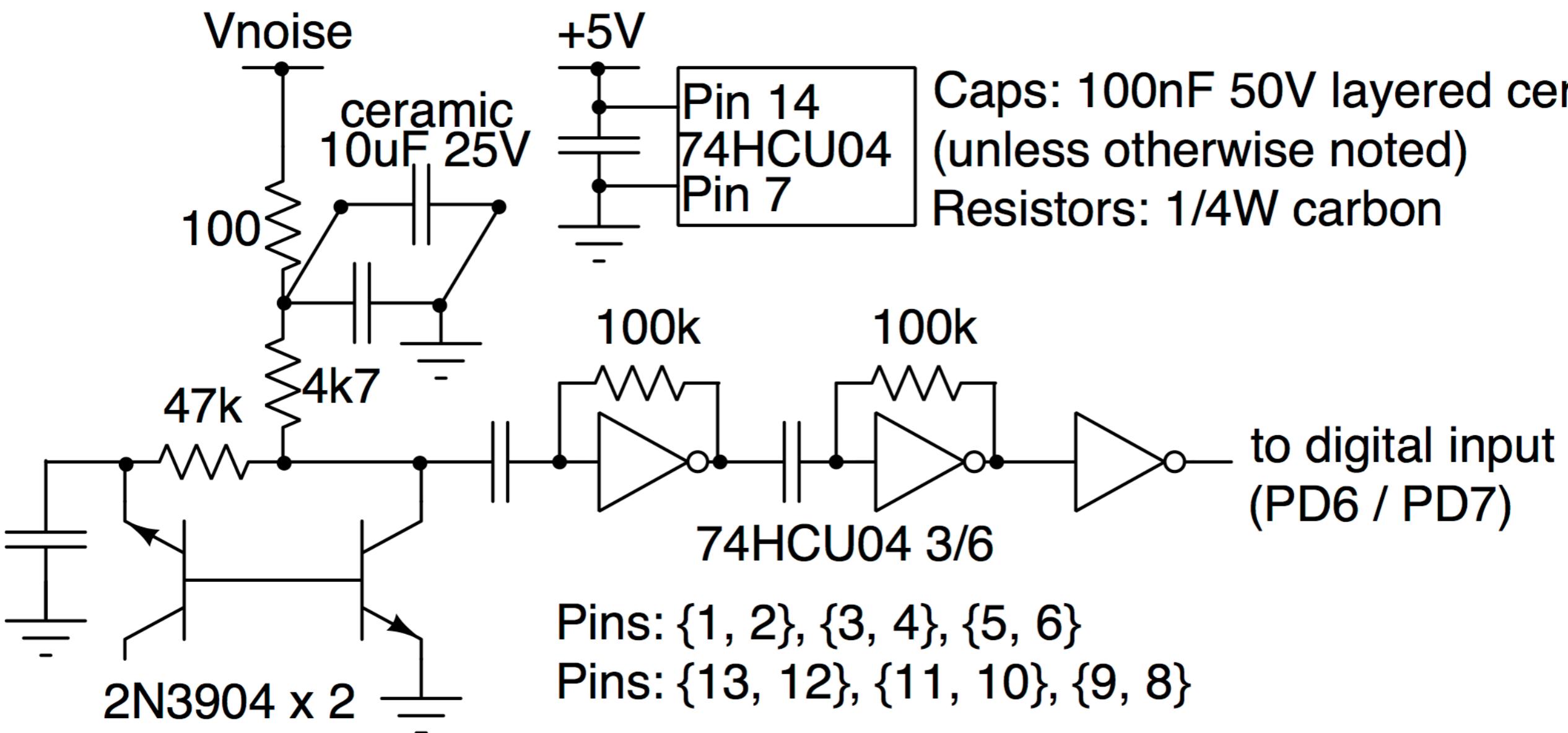
This implementation is working  
as a dice: generating numbers of  
1~6 / サイコロ同様に1から6ま  
での数字を生成する

Generating ~10kbytes/sec



avrhwrrng: Arduino 2009/UNO shield schematics  
for a hardware random number generator  
by Kenji Rikitake / v2rev1fix1 / 5-JUN-2016  
Licensed CC-BY-4.0

物理乱数発生のための  
回路例



(The above circuit only shows one of the two same necessary circuits)  
Change from v2rev1: add a 10uF cap (parallel to 100nF) for Vin ripple filter  
Vnoise must be  $\geq +12V$  (+9V didn't work)  
Vnoise can be supplied from Arduino Vin or a DC-DC converter

# Infinity Noise TRNG<sup>12</sup>

- Thermal noise based
- USD35/device
- Public domain, no patent
- No MCU on the device / デバイスはMCUを持たない
- ~40Kbytes/sec

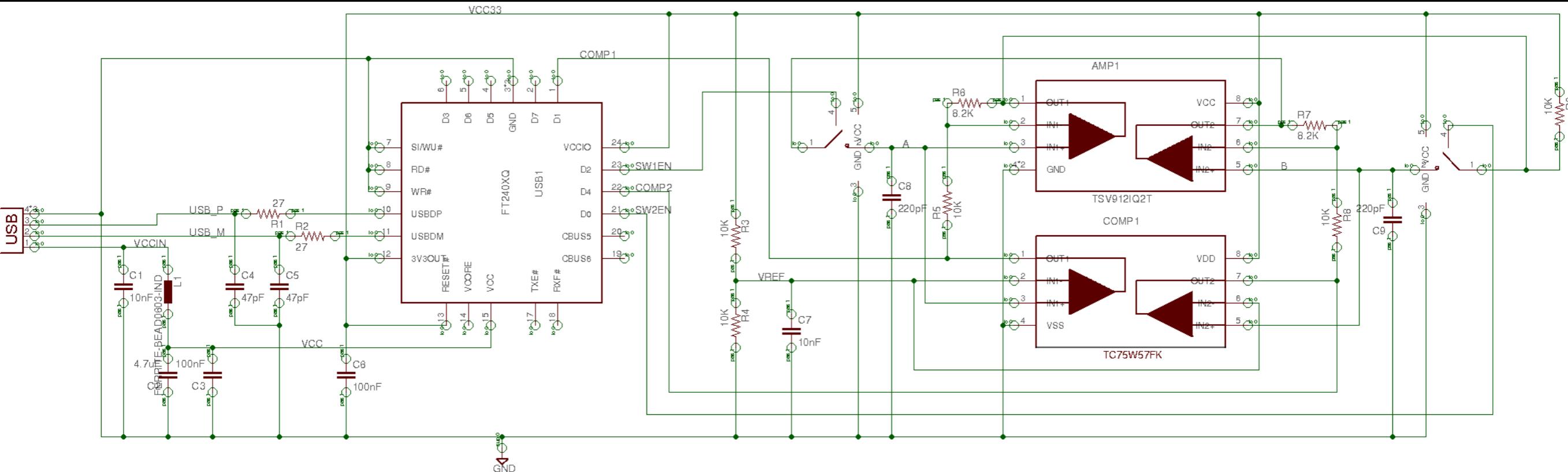


<sup>1</sup> <https://github.com/13-37-org/infnoise>

<sup>2</sup> Crowd Supply product page of Infinity Noise TRNG

# Infinity Noise TRNG schematics

# FTDI bitbang I/O controls the noise amplifier



# How to inject external randomness to the operating systems

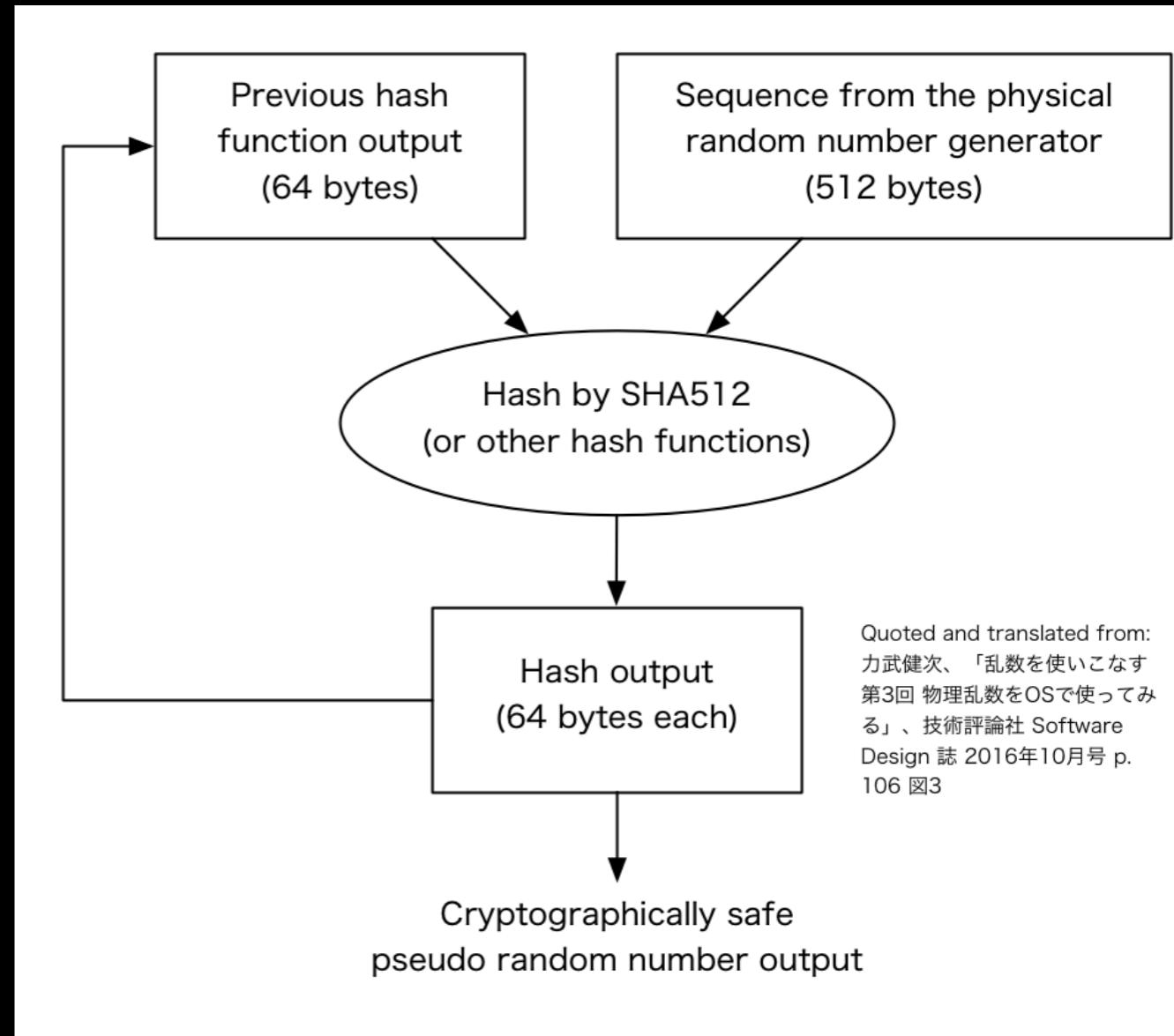
- Linux: random(4) ioctl() of RNDGETENTCNT, RNDADDENTROPY (User accessible)
- FreeBSD: random\_harvest(9) (Accessible from kernel modules only)<sup>3</sup>
- Other proprietary OSes: unable to find the same functions / その他の独自OSでは外部からランダムネスを注入できない

---

<sup>3</sup> My FreeBSD randomness injection device driver

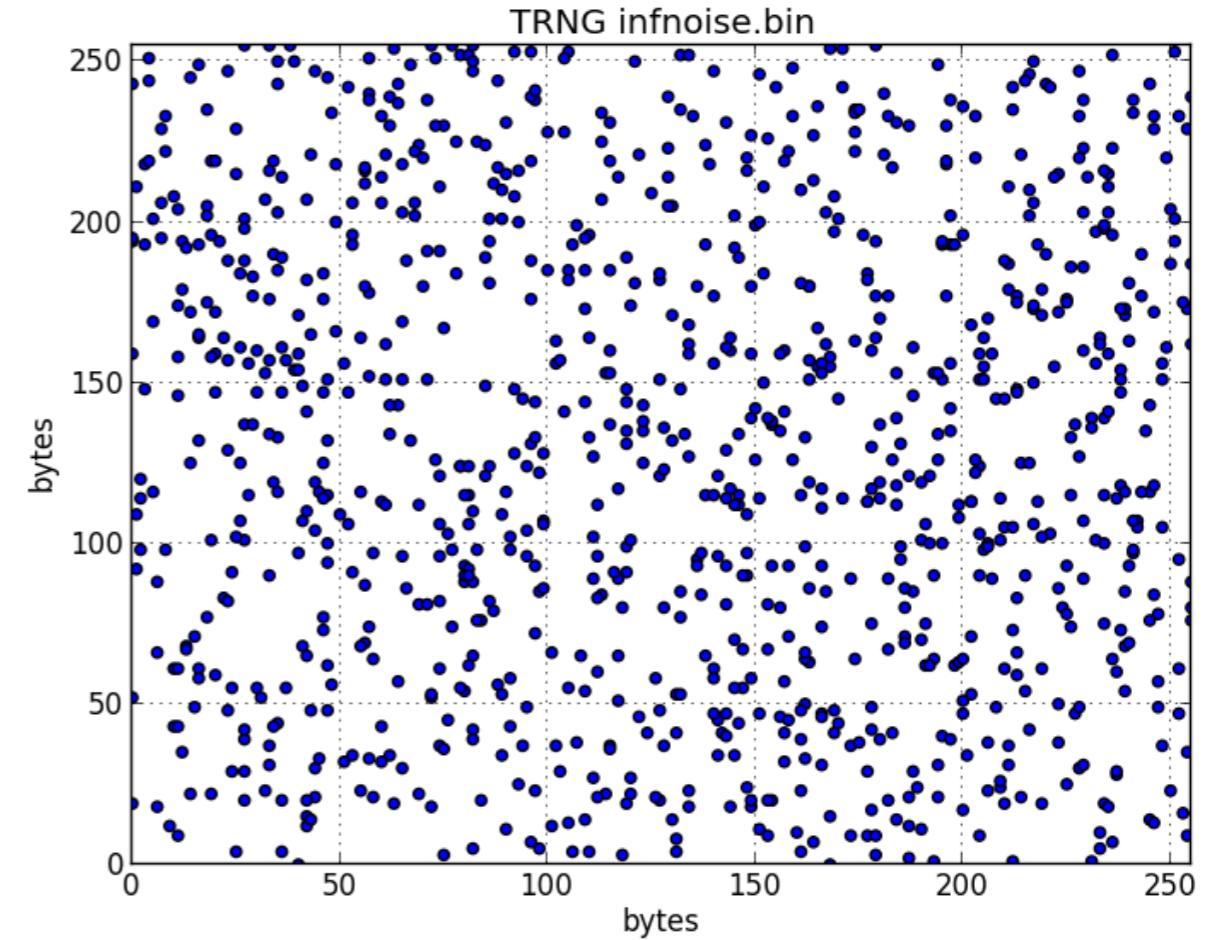
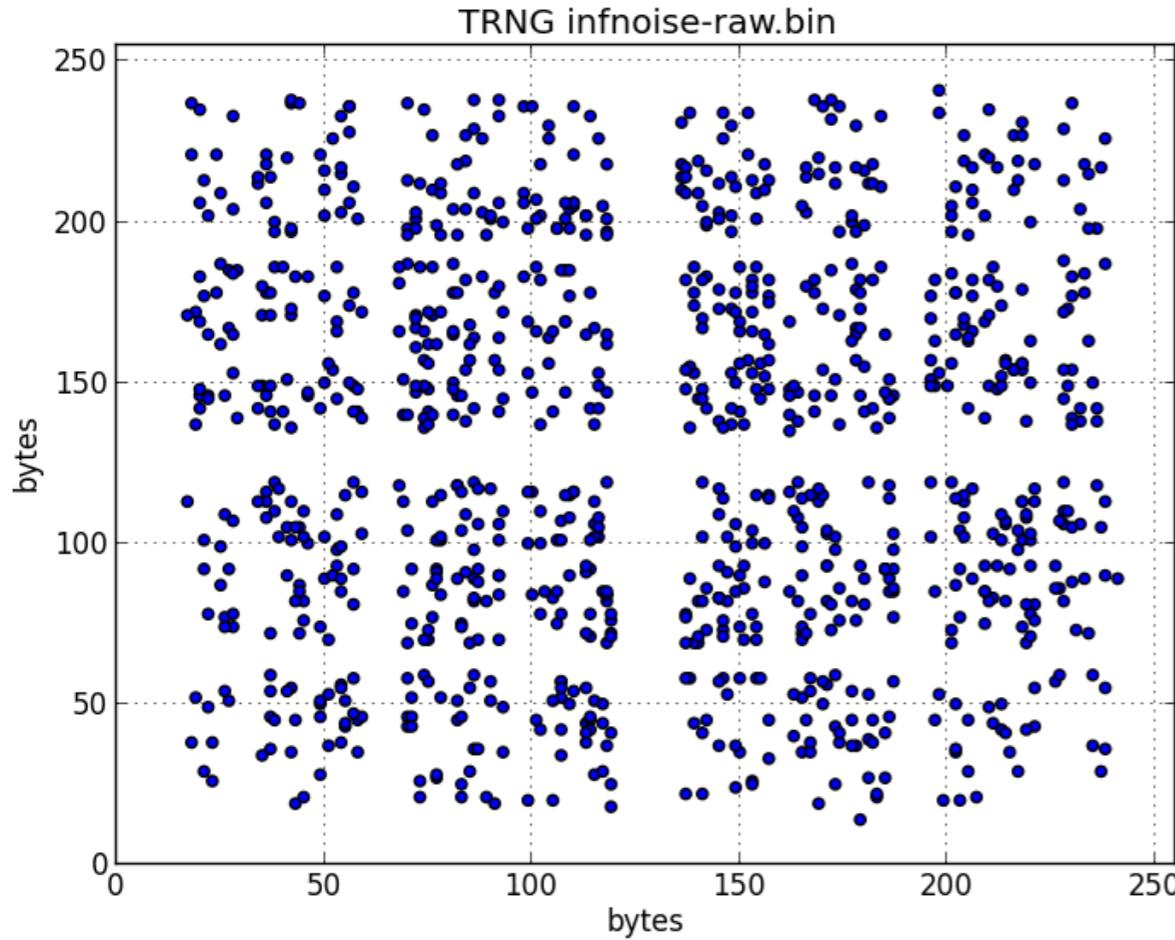
# Whitening for uniform distribution

- Cryptographically strong hash functions are used in the whitening
- Whitening is implemented in the driver or the post-processing software
- 暗号化ハッシュ関数を適用して出力の分布を一様化する処理（ホワイトニング）が必要



Quoted and translated from:  
力武健次、「乱数を使いこなす  
第3回 物理乱数をOSで使ってみ  
る」、技術評論社 Software  
Design 誌 2016年10月号 p.  
106 図3

# How whitening works on Infinity Noise TRNG



# How much randomness is enough?

- USD <100 generator: > ~10kbytes/sec, more than sufficient for an active server
- If you generate *a lot of* keys/passwords, consider dedicated generator of Mbps or Gbps class (they exist but expensive)
  - ハードウェア生成器があれば~10kバイト/秒以上（通常の運用には十分）
  - 本気で大量に鍵やパスワードを生成するなら専用の物理乱数生成器を導入すべし

# Summary/まとめ

- Good randomness is hard to obtain
- External physical random number generator is essential for secure operation
- Do not invent your own methods
- 良いランダムネスを得るのは難しい
- 安全な運用には外部の物理乱数装置が不可欠
- 自己流でやらない

# Other references

- <https://github.com/jj1wdx/avrhwrng/>
- **Fifteen Ways to Leave Your Random Module**  
(Erlang User Conference 2016)
- 疑似乱数の作り方・使い方 ゲームから情報セキュリティまで

# Thanks Questions?

Give the feedback please; use the QR code on your name card

フィードバックをおねがいします / ネームカードのQRコードを使ってください