OneRNG

An open solution to a core security performance issue

Multicore World 2015

Jim Cheetham

OneRNG.info
Moonbase Otago

Jim Cheetham

Information Security Office, University of Otago, NZ

<jim.cheetham@otago.ac.nz>

Security, Unix/Linux, Networks

Operations & Architecture

Contents

Information Security Primer

- What do we mean by "InfoSec"?
- The CIA Triad

Definitions of "InfoSec"

- ***CIA Triad*: "Confidentiality, Integrity, and Availability"**
- *OECD 9 Principles*: "Awareness, Responsibility, Response, Ethics, Democracy, Risk Assessment, Security Design and Implementation, Security Management, and Reassessment"
- *Parkerian Hexad*: "Confidentiality, Possession, Integrity, Authenticity, Availability, and Utility"
- *IAS Octave*: "Confidentiality, Integrity, Availability, Accountability, Auditability, Authenticity/Trustworthiness, Non-repudiation and Privacy"

Confidentiality

- A cornerstone of Privacy
- Data at rest is protected
- Data in transit is protected
- Access controls within a system

Integrity

- Multiple copies of stored data
- Checksums
- Access controls & audit

Availability

- Capacity planning
- Early error detection
- Redundancy
- Disaster Recovery

Risk Assessment

- Confidentiality failures
  Who else has a copy of our data?
- Integrity failures
  What if our data has been corrupted/altered/lost?
- Availability failures
  Failure to record valid observations?

Early Timeline

- 1976 US Copyright Act - copyright became automatic
  - Software "value" changed, and started to protect itself against users
- 1986 Cuckoo's Egg - Cliff Stoll caught the KGB attacking US systems
- 1988 Morris Worm - demonstrating vulnerabilities in established code

Infosec in Computer Operations

- Systems are now much more defensive
- More 'initial conditions' are being set randomly to prevent predictive attacks
- More data is encrypted by default

Defensive uses of Random

- Address Space Layout Randomisation - ASLR
- Initial TCP Sequence numbers
- Session Identifiers

Examples of Encryption

- Transport Layer Security - TLS / HTTPS
- ssh & RDP - remote access
- Whole Disk Encryption
- Encrypted Malware

Encryption

Encryption prevents an unauthorised system from understanding any data that they have found

- **Symmetric Encryption**
  A single key is used both to lock and unlock the data
- **Public Key Encryption**
  Two related keys are needed; one locks, the other unlocks

Encryption Keys

- Keys need to be kept secret
- Keys need to be complex to prevent them being guessed
- Keys need to be *unpredictable*

Unpredictablity

- Described with two related concepts – *Randomness* & *Entropy*
- Randomness is a lack of pattern in a sequence of events
- Entropy is the amount of information in a 'message', or 'surprisal' value

Sources of Randomness

- The Real Non-Deterministic World - True Random
  - QM events, e.g. Radioactive Decay
  - Environmental Noise
- Algorithms with complex output
  - Pseudo-Random - PRNG
  - Distributed Random Bit Generator - DRBG
  - Cryptographically Secure - CSPRNG

Algorithmic Complexity

- Algorithms can produce uniformly distributed sequences that seem to be Random
- But the algorithms need initial conditions to be set - the 'seed'
- Using the same seed results in an identical sequence
- Therefore any run from a given seed value is not Random?

Seed Values

- Computers have limited sources for True Random data
- Environmental Noise: slow & not universally applicable
  - Keyboard/Mouse timings
  - Disk access timings
  - Network activity timings
- True Random data is needed to select seed values for PRNGs

Statistics & Modelling

This is not my expert area …

- LLN - the Law of Large Numbers
- Monte Carlo methods; simulations often specify a PRNG to be used, for portability
- Markov processes

PRNG/DRBG

"Anyone who considers arithmetical methods of producing random digits is, of course, in a state of sin." — John von Neumann, 1951

PRNG/DRBG

- Problems
  - Periodicity
  - Output Artifacts
  - Unprovable
- Benefits
  - Repeatable
  - Fast

Repeatable

Use known seeds when testing your simulation code

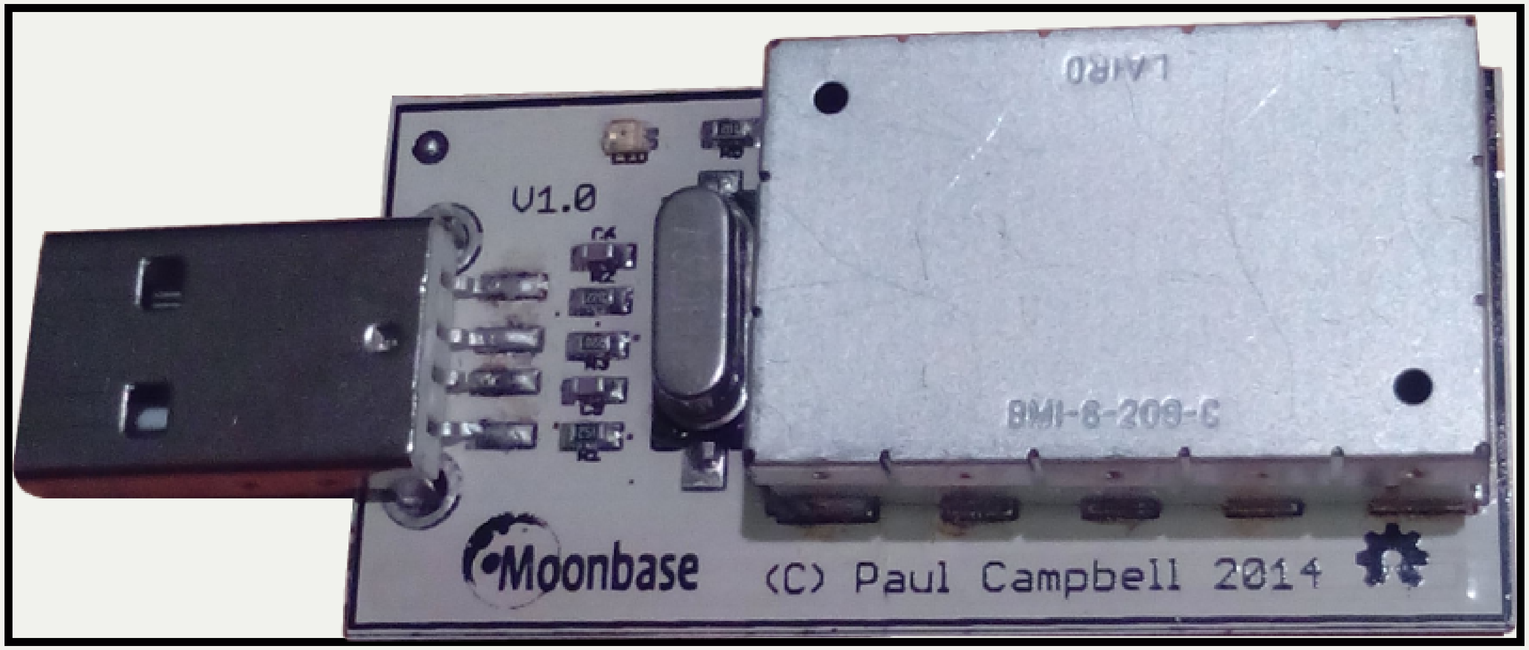
Use True Random seeds when conducting your research

What is OneRNG?

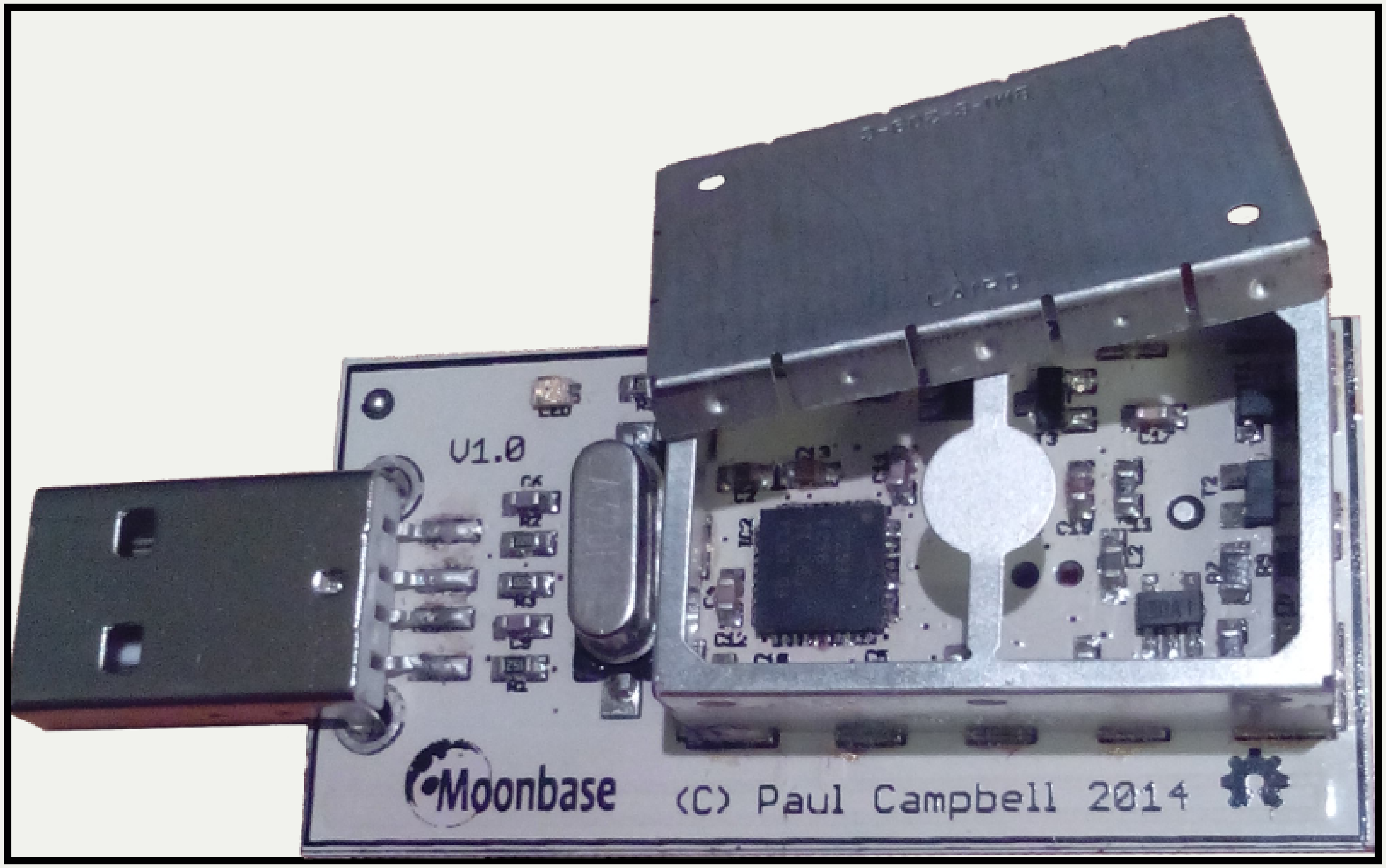
It is a small USB-connected device

It speeds up & increases your computer's ability
to provide you with high-quality random numbers

It has a Tin-Foil Hat



It has a Tin-Foil Hat



Entropy Collector

By default, OneRNG measures *unpredictable* physical events
from an **avalanche diode** circuit
and returns the results — but this is a bit too raw to be used directly

Environmental Noise

You can also enable the RF monitor to get another source of entropy data with a higher quality — but at the cost of a little paranoia

Whitened data

This raw data is then whitened through a CRC16 function which makes it good enough to be fed into your system

There is an AES hardware module available, but it is not used by the default firmware (trust issues)

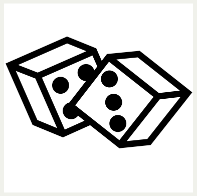
Keep topping up the pool

There's a 7.5KB pool of data that is kept full

New data is mixed in over the old data *all* the time

A LED on the board tells you when the pool is full and warns you when it is getting empty

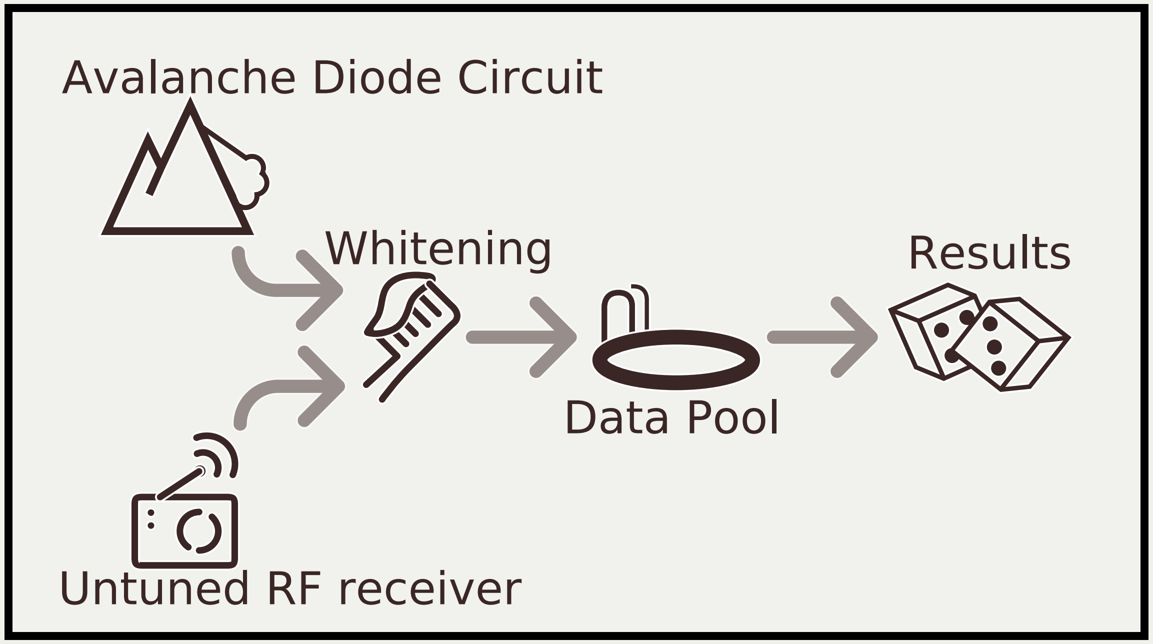
Output



The output from OneRNG should be used as an Entropy input to your existing systems (preferably OS)

We provide ~7.5 bits of entropy per byte of data

The flow of data



The user controls the choice of sources

The user controls the use of Whitening

The user controls where the results go

OneRNG for Crypto

OneRNG was originally designed to help end-users improve their online privacy

Current demands for seed values outstrip the available Entropy

Online privacy advocates should be quick to distrust …

How does OneRNG help?

Being an Open Hardware and Open Source solution is a start

But the OneRNG is also designed to be ***verifiable***

No Trust Needed

You should not trust this device — you do not *need* to trust this device

You should ***VERIFY*** that what you physically hold is what you need to have

How to verify the hardware

1. Remove the Tin-Foil Hat (ours, not yours)
2. Identify the components
3. Identify the connections, top and bottom
4. Research the one complex component, the CC2531 chip
5. See how much storage it has (256KB or 128KB)

How to verify the software

1. Plug the device into a machine
2. Dump the firmware into a file
3. Confirm the data is the full 256KB
4. Validate the GPG signature
5. Confirm the data is ‘uncompressible’

Scripts on your server should do steps 2,3 & 4 on startup

What does verification tell you?

- The hardware you received conforms with the description
- The firmware that is used matches the source
- Tampering should be detectable

KICKSTARTER

To get volume production set up, we used Kickstarter with an NZD**$10,000** target over 45 days

- 50% funded after 2 days
- 100% funded after 6 days
- 200% funded after 18 days
- 485% funded at close : NZD**$48,551**

KICKSTARTER

Rewards Options

We offered combinations of:

- NZD $50 - a single OneRNG from bulk manufacturing
- NZD $90 - a device programmer unit
- NZD $110 - a single hand-built unit

Current Status

As of mid February 2015:

- Units will be manufactured in Shenzen, China
- The programming & test devices are made in NZ
- Firmware is cryptographically signed by a machine in NZ before being written to the manufactured devices
- Waiting for Chinese New Year festivities to complete!
- Next, Moonbase Otago opens an online store to sell OneRNG

Diversion: Why does /dev/random block?

The NSA-designed SHA-1 was not fully "trusted"

The blocking behaviour is a defence against this untrusted DRBG

---

*Ted T'so, 2015 "… the paranoiacs were *right* that the NSA had introduced a back-door into a crypto algorithm which they gifted to the civilian world. It just turned out to be DUAL-EC instead of SHA-1."*

Just use `/dev/urandom` :-)

What's the difference between /dev/random and /dev/urandom?

These are fundamentally the same :-)

They will behave identically with sufficient entropy

Use `/dev/urandom`

Use `/dev/urandom`

Use `/dev/urandom`

If you already know better, you don't need me to tell you what to do

How does OneRNG help /dev/random?

Even though you should *use* `/dev/urandom`

OneRNG helps to avoid the need to block

Therefore systems that use /dev/random *run faster*?

(Catalyst are simlating workloads to quantify this assertion)

The more the merrier

A small quantity of good entropy is enough to improve everything

But the more sources of entropy you have, the better off you are

As well as OneRNG, please add other sources

Complements to OneRNG

- SW: **rtl-entropy** from SDR - https://github.com/pwarren/rtl-entropy (Paul Warren, LCA2014)
- HW: **USBtrng** - http://altusmetrum.org/USBtrng/ (Bdale Garbee/Keith Packard, DebConf 2014)
- SW: **Turbid** from audio I/O - http://www.av8n.com/turbid/ (John Denker, 2002-date)
- HW: **NeuG** ("noisy") - http://git.gniibe.org/gitweb/?p=gnuk/neug.git (Yutaka Niibe, DebConf 2014)

Q & A

http://onerng.info/

Credits

Presentation resources :-

Software - reveal.js, hosted on Github

Icons used are from The Noun Project, http://creativecommons.org/licenses/by/3.0/us/ licensed

Arrow and Bent Arrow by Thomas Le Bas, Dice by Weston Terrill, Toothbrush, Radio by Joe Harrison, Avalanche by Louis Dawson, Swimming Pool by Sitara Shah, Clock by Nick Green, Guy Fawkes by Christopher T. Howlett, Pac-Man by Luigi Di Capua, Surveillance by Luis Prado, Audit by Miroslav Koša, Skydiving by Jual Pablo Bravo, CPU by iconsmind.com, Certificate by Alex Auda Samora, Incognito by Alen Krummenacher, Layers by Cornelius Danger, Search by Melvin Salas, Infographic by Rob Gill, Seed Packet by Anton Gajdosik (Public Domain)