Security not by chance:
the AltusMetrum hardware true random number generator

Tom Marble Informatique, Inc.

DebConf 14
Portland, USA

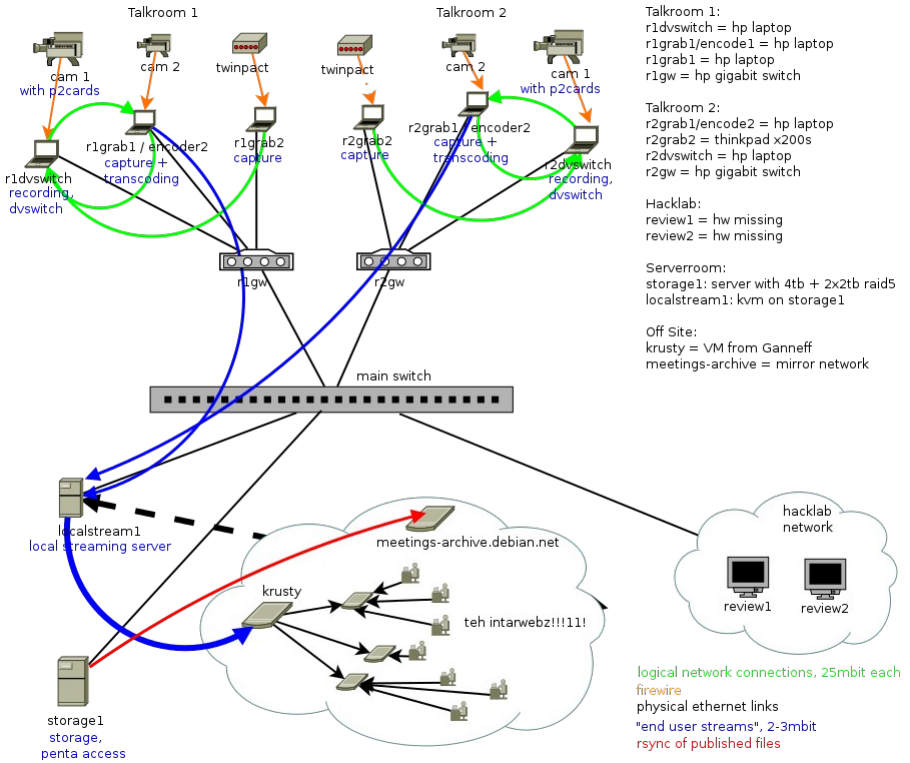
debian

USB TRNG

- Why do we care about randomness?
- Risks specifically for GNU/Linux (and /dev/random vs. /dev/urandom)
- AltusMetrum design for USB TRNG
- Measuring quality
- Current status & next steps
- Questions

Debian Video & IRC

- IRC /join #debconf-room329
- Live video stream



Why do we care about RNG?

- Generation of encryption keys (GPG)
- Synthesizing one time session keys (https)
- In kernel sources of randomness (sequence numbers)

PRNG vs. TRNG

- Psuedo random number generators
  - fast (good for driving software tests)
  - deterministic (predictable)
- Hardware (true) RNG
  - based on entropy of a physical process
  - acquiring quality entropy is slow (/dev/random will block)

Security risks

- Public key cryptography (RSA) is usually only used to encrypt a *session* key
- The session key is used with a symmetric cipher (AES) to encrypt traffic
- An attacker doesn't have to break the private key if they can guess the session key

Greg Kroah-Hartman

Shared publicly - Mar 7, 2014

A good read.

Myths about /dev/urandom

2uo.de

Thomas Hühn

http://www.2uo.de/myths-about-urandom/

+141 45

23 comments

Bhaskar Chowdhury Mar 7, 2014

Thanks a bunch Greg! for sharing :) indeed a good read..

The fine print...

Hühn says

What about entropy running low?

It doesn't matter.

The underlying cryptographic building blocks are designed such that an attacker cannot predict the outcome, as long as there was enough randomness (a.k.a. entropy) in the beginning. A usual lower limit for “enough” may be 256 bits. No more.

Does this occur in the wild?

Nadia Heninger (*et al*) Aug 2012:
Mining Your Ps and Qs: Detection of Widespread Weak Keys in Network Devices

“Every software package we examined relies on /dev/urandom to generate cryptographic keys; however, we find that Linux’s random number generator (RNG) can exhibit a boot-time entropy hole that causes urandom to produce deterministic output under conditions likely to occur in headless and embedded devices.”

	Our TLS Scan		Our SSH Scans	
Number of live hosts	12,828,613	(100.00%)	10,216,363	(100.00%)
... using repeated keys	7,770,232	(60.50%)	6,642,222	(65.00%)
... using vulnerable repeated keys	714,243	(5.57%)	981,166	(9.60%)
... using default certificates or default keys	670,391	(5.23%)		
... using low-entropy repeated keys	43,852	(0.34%)		
... using RSA keys we could factor	64,081	(0.50%)	2,459	(0.03%)
... using DSA keys we could compromise			105,728	(1.03%)
... using Debian weak keys	4,147	(0.03%)	53,141	(0.52%)
... using 512-bit RSA keys	123,038	(0.96%)	8,459	(0.08%)
... identified as a vulnerable device model	985,031	(7.68%)	1,070,522	(10.48%)
... model using low-entropy repeated keys	314,640	(2.45%)		

Table 2: **Summary of vulnerabilities** — We analyzed our TLS and SSH scan results to measure the population of hosts exhibiting several entropy-related vulnerabilities. These include use of repeated keys, use of RSA keys that were factorable due to repeated primes, and use of DSA keys that were compromised by repeated signature randomness. Under the theory that vulnerable repeated keys were generated by embedded or headless devices with defective designs, we also report the number of hosts that we identified as these device models. Many of these hosts may be at risk even though we did not specifically observe repeats of their keys.

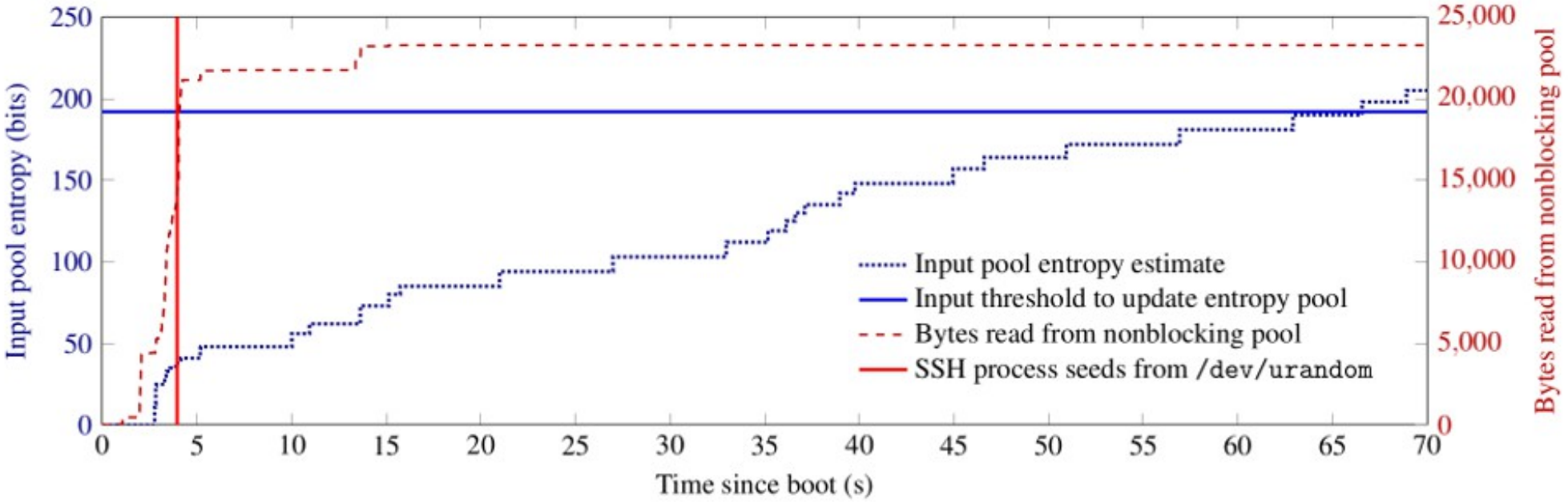


Figure 5: **Linux urandom boot-time entropy hole** — We instrumented an Ubuntu Server 10.04 system to record its estimate of the entropy contained in the Input entropy pool during a typical boot. Linux does not mix Input pool entropy into the Nonblocking pool that supplies `/dev/urandom` until the Input pool exceeds a threshold of 192 bits (blue horizontal line), which occurs here at 66 seconds post-boot. For comparison, we show the cumulative number of bytes generated from the Nonblocking entropy pool; the vertical red line marks the time when OpenSSH seeds its internal PRNG by reading from `urandom`, *well before* this facility is ready for secure use.

Linux entropy

- The Linux estimate of entropy is deemed to be conservative
- https://eprint.iacr.org/2006/086.pdf
- G UTTERMAN , Z., P INKAS , B., AND R EINMAN , T. Analysis of the Linux random number generator. In Proc. 2006 IEEE Symposium on Security and Privacy (May 2006), pp. 371–385.

Matthew Green
@matthew_d_green

Following

"How can we make the RNGs [secure] -- how can we test them?" - Edward Snowden asking the question none of us can answer.

Reply Retweeted Favorite More

RETWEETS	FAVORITES
32	14

5:24 AM - 11 Mar 2014

http://blog.cryptographyengineering.com/2014/03/how-do-you-know-if-rng-is-working.html

Reply to @matthew_d_green

Would you like some entropy?

- The challenge with existing TRNGs
  - Expensive
  - Out of stock
  - Closed designs

SimTec Entropy Key

- http://www.entropykey.co.uk/
- *Please note that there is a very long waiting period for Entropy Keys at the moment. We currently have no stock and do not have a date for when we will have more.*

NeuG

- http://www.gniibe.org/memo/development/gnuk/rng/neug.html

画像とボタン

画像と情報共有のボタンを付けたらと教えてもらったので付けてみます。

rtl-entropy

- Introduced at LCA 2014
- DVB-T dongle used for SDR
- http://www.rtl-sdr.com/true-random-numbers-rtl-entropy/

yes

Let's build one!

… because we can!

Altus
Metrum

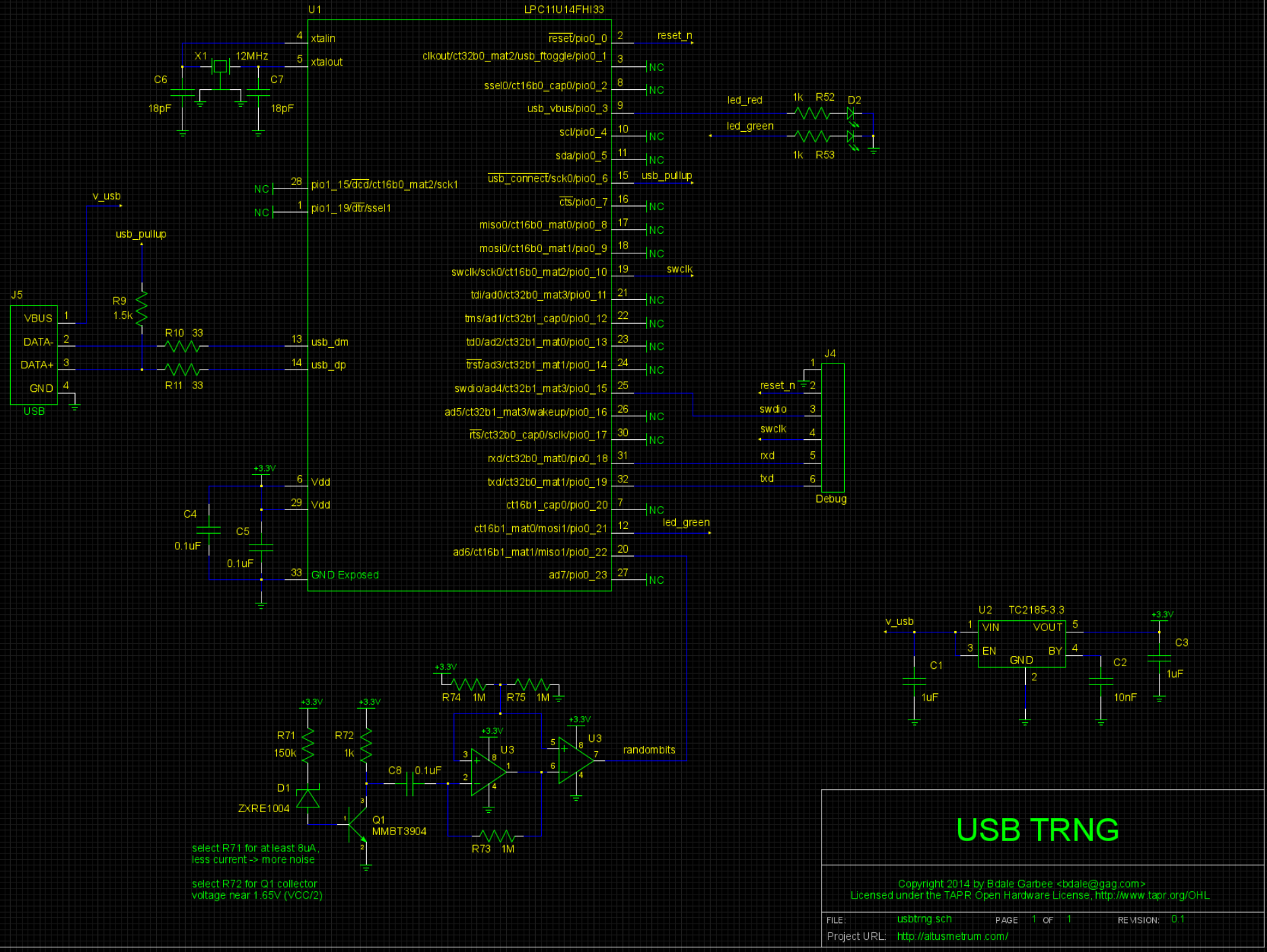
AltusMetrum

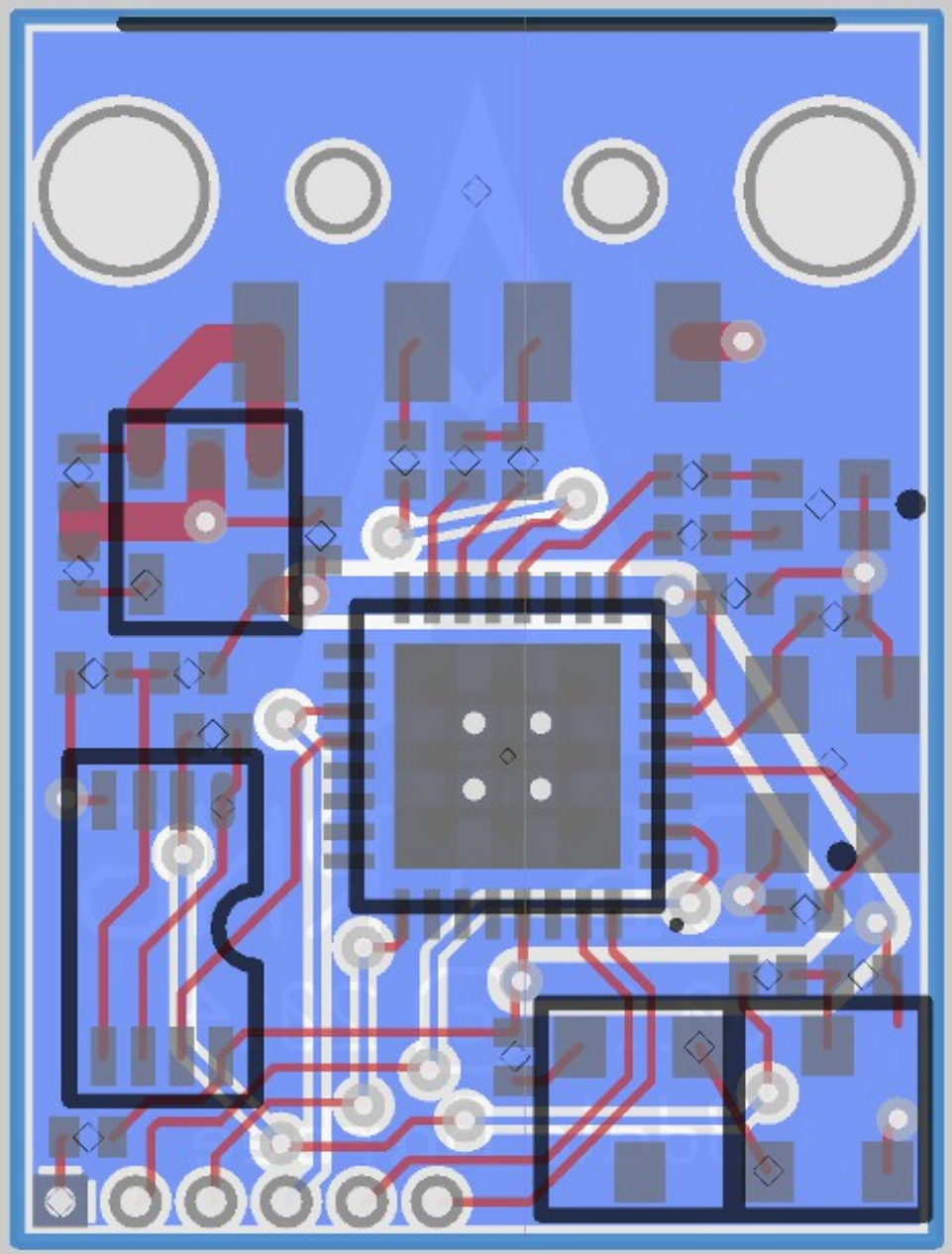
- Over 15 open hardware designs
  - STM32L (ARM)
  - CC1111 (8051)
  - ATmega32U4 (AVR)
- AltOS — real time OS
  - Multi-tasking
  - Non-preemptive
  - Sleep/wakeup scheduling
  - Mutexes
  - Timers

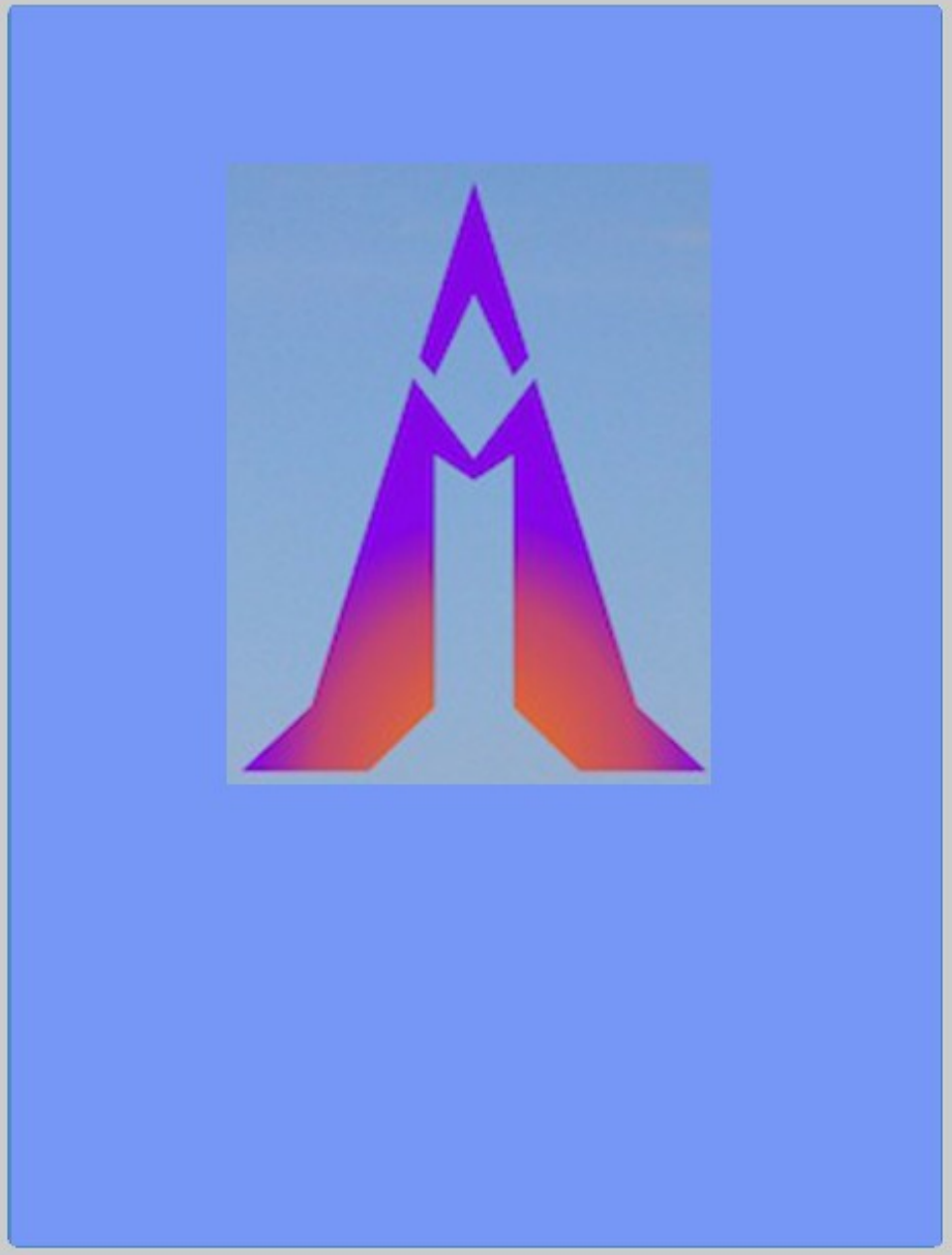

XVIII
ARGONIA, KANSAS

USB TRNG Design

- NXP Cortex M0
- Bootloader reflashable via USB
- Entropy sources based on
  - band gap voltage ref. (zener diode)
  - noise amplification (transistor)
  - dual op-amp w/rail-to-rail outputs


U1
LPC11U14FHI33
X1
12MHz
C6
18pF
C7
18pF
xtalin
xtalout
reset/pio0_0
reset_n
clkout/ct32b0_mat2/usb_ftoggle/pio0_1
ssel0/ct16b0_cap0/pio0_2
usb_vbus/pio0_3
scl/pio0_4
sda/pio0_5
usb_connect/sck0/pio0_6
usb_pullup
cts/pio0_7
miso0/ct16b0_mat0/pio0_8
mosi0/ct16b0_mat1/pio0_9
swclk/sck0/ct16b0_mat2/pio0_10
swclk
tdi/ad0/ct32b0_mat3/pio0_11
tms/ad1/ct32b1_cap0/pio0_12
tdo/ad2/ct32b1_mat0/pio0_13
trst/ad3/ct32b1_mat1/pio0_14
swdio/ad4/ct32b1_mat3/pio0_15
ad5/ct32b1_mat3/wakeup/pio0_16
rts/ct32b0_cap0/sclk/pio0_17
rxd/ct32b0_mat0/pio0_18
txd/ct32b0_mat1/pio0_19
ct16b1_cap0/pio0_20
ct16b1_mat0/mosi1/pio0_21
ad6/ct16b1_mat1/miso1/pio0_22
ad7/pio0_23
NC
pio1_15/dcd/ct16b0_mat2/sck1
pio1_19/dtr/ssel1
GND Exposed
led_red
led_green
1k
R52
D2
1k
R53
v_usb
usb_pullup
R9
1.5k
J5
VBUS
DATA-
DATA+
GND
USB
R10
33
R11
33
usb_dm
usb_dp
J4
reset_n
swdio
swclk
rxd
txd
Debug
+3.3V
C4
0.1uF
C5
0.1uF
Vdd
Vdd
R74
1M
R75
1M
R71
150k
R72
1k
D1
ZXRE1004
Q1
MMBT3904
C8
0.1uF
U3
U3
R73
1M
randombits
select R71 for at least 8uA, less current -> more noise
select R72 for Q1 collector voltage near 1.65V (VCC/2)
U2
TC2185-3.3
VIN
VOUT
EN
BY
GND
C1
1uF
C2
10nF
C3
1uF
USB TRNG
Copyright 2014 by Bdale Garbee <bdale@gag.com>
Licensed under the TAPR Open Hardware License, http://www.tapr.org/OHL
FILE: usbtrng.sch
PAGE 1 OF 1
REVISION: 0.1
Project URL: http://altusmetrum.com/






USB TRNG
v0.1 (c) 2014
Bdale Garbee

FLOSS + OSHW

- hw : TAPR OHL 1.0
- sw : GPLv2
- Faciliates community collaboration
- Enables independent implementation
  - Is Bart Massey in the room?

Quality Assessment

- FIPS 140-2 *Continuous random number generator test* (on device)
- Evaluation with accepted test suites (e.g. TestU01)
- Note: Green's observation low level testing is required (conditioning may not be enough)

How can we use USB TRNG?

- Can be connected to the GNU/Linux Entropy Key Daemon (ekeyd) which can provide entropy directly to the kernel pool
- Serving via the EGD protocol

Current Status

- Hardware: early prototype stage
- Software: design stage
- Testing: Use existing test suites at the most raw level (pre-conditioning)
- Participate!
  http://altusmetrum.org/USBtrng/
  IRC OFTC #altusmetrum
  http://lists.gag.com

Next Steps

- Alpha test prototype hardware
- Develop code path to ekeyd/EGD
- Document (triage) potential attack vectors (e.g. firmware upgrade, USB snooping).
- Learn about **krngd**
- Learn from **tfheen Mon, 02 Nov 2009 - Distributing entropy**
  - How much entropy is often needed
  - EGD memory leaks
  - Client auto-reconnection