Solving The Platform Entropy Problem Phase 2

George Cox

Security Architect

Intel Corporation

Session Introduction

Previously, I have described Intel's Digital Random Number Generator (DRNG) and its SW interface (the RdRand instruction)

Today, I will describe an incremental addition to Intel's DRNG and its added SW interface (the RdSeed instruction)

DRNG Introduction

Much of computer security is based upon the use of cryptography.

Cryptography is based upon two things:

- Good algorithms (e.g., AES) and
- Good keys (e.g., good random numbers).

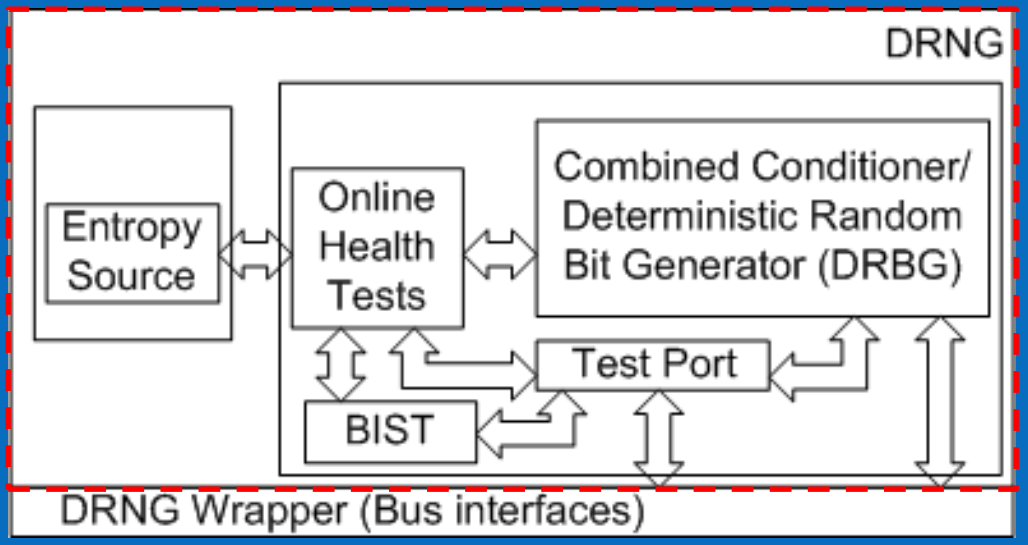
Historically, computing platforms broadly have not had high quality/high performance entropy (or random numbers).

By enabling ALL future Intel platforms with our RdRand/RdSeed instructions (supported by our underlying Digital Random Number Generator (DRNG)), we:

- Provide a fundamental solution to the long standing "Platform Entropy Problem" and
- Give Intel's customers the "common brand promise" of high quality/high performance entropy (or random numbers) everywhere across ALL Intel products in which it is embedded.

=> the "Platform Entropy Problem" just "goes away" on future Intel-based platforms.

The DRNG



A reusable IP module that provides each embedding with an autonomous/self contained, high quality/high performance, "complete" DRNG

Provides "common brand promise" of high quality, high performance entropy across ALL Intel products

Composed of

- An all-digital Entropy Source (NRBG), runtime entropy quality measurement via Online Health Test (OHT),
- Conditioning (via AES CBC-MAC mode) and DRBGing (via AES CTR mode) post processing and
- Built In Self Test (BIST) and Test Port

"Standards" compliant (NIST SP 800-90) and FIPS 140-2/3 Level 2 certifiable as such and

Designed for ease of testability, debug, and validation in HVM and in end user platforms

- Comprehensive Built In Self Test (BIST) and
- Test Port (and associated tools) for full pre/post-silicon debug flexibility

Red dotted line is the DRNG's FIPS boundary

Standards And Support For Them

There will soon be two NIST RNG standards SP800-90 A and B/C (with different required semantics) => two instructions (with different required semantics).

- RdRand and RdSeed are for two different purposes.
- RdRand serves as a NIST SP 800-90 A compliant DRBG/PRNG – whose output can be used directly (without need for a SW PRNG to post process its outputs) - Think of RdRand as being used to replace 128-bit cryptographic SW PRNGs
- RdSeed serves as a NIST SP 800-90 B/C compliant ENRBG – whose output is expected to be used for seeding SW PRNGs to post process its outputs - Think of RdSeed as providing keying material for seeding greater than 128-bit cryptographic SW PRNGs.

At a high level RDSEED produces greater prediction resistant entropy at a lower rate than RDRAND.

DRNG's DRBG and ENRBG designed to be NIST SP800-90 A and B/C compliant respectively and FIPS 140-2 certifiable as such

RdRand/RdSeed Delivery

Due to participation in the standards effort and corresponding good architecture work, we can deliver both sets of semantics from basically the same design => minimal incremental cost and ZERO incremental performance penalty to parallel RdRands

We can be the first to offer direct HW/instruction product support for both standards

RdRand was first delivered in Ivy Bridge/Silvermont generation processors

RdSeed is being added to Broadwell/Goldmont generation processors

RdRand versus RdSeed Semantics

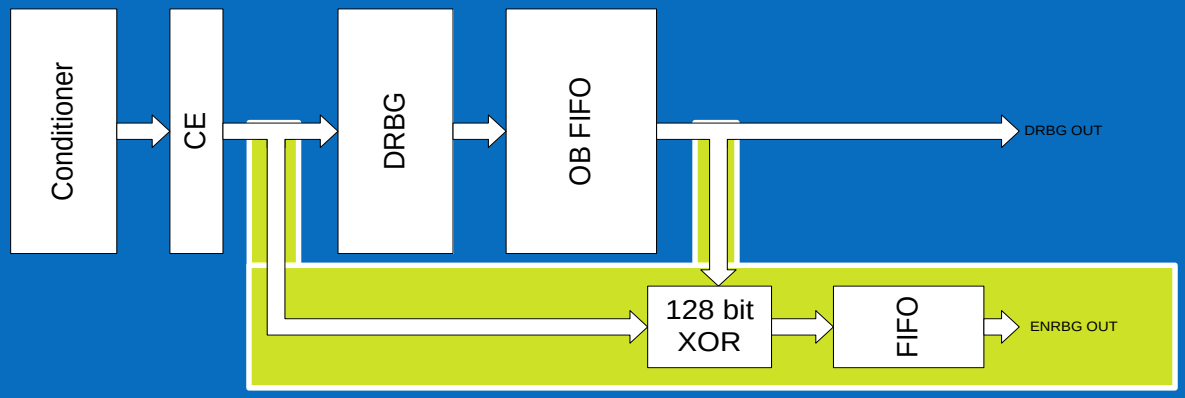
NIST SP 800-90 A compliant DRBGs/PRNGs have different (lesser) "prediction resistance" requirements than NIST SP 800-90 B/C compliant ENRBGs.

- Multiplicative vs. Additive resistance when combining values
  - 128 bits RdRand + 128 bits RdRand = $O(2^{129})$ prediction resistance
  - 128 bits RdSeed + 128 bits RdSeed = $O(2^{256})$ prediction resistance
- Hence RdSeed outputs are suitable for seeding arbitrary width SW PRNGs

DRBG continuous performance - 800 MBytes/s, 6.4 Gbit/s

ENRBG continuous performance - 92.8 MBytes/s, 742.4 Mbit/s

ENRBG XOR Construct



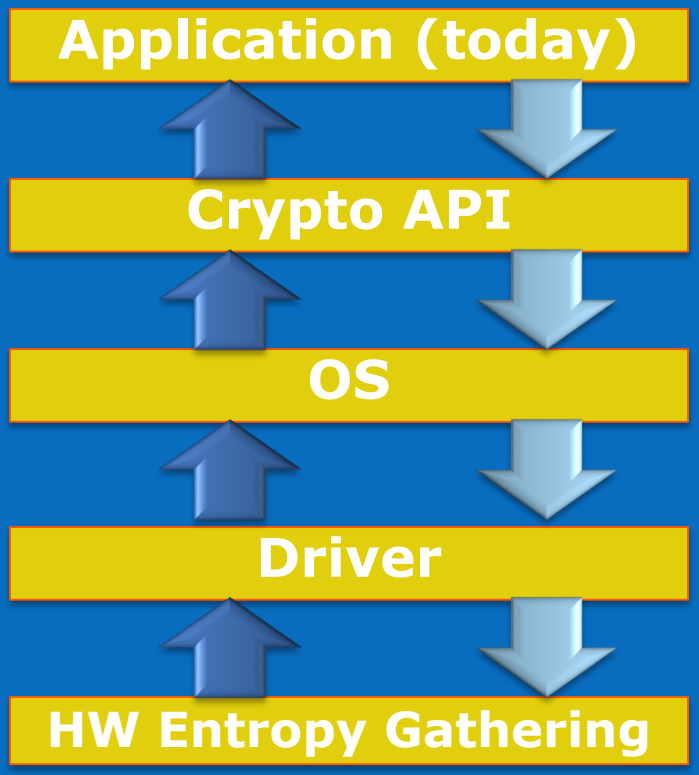
RDSEED Instruction

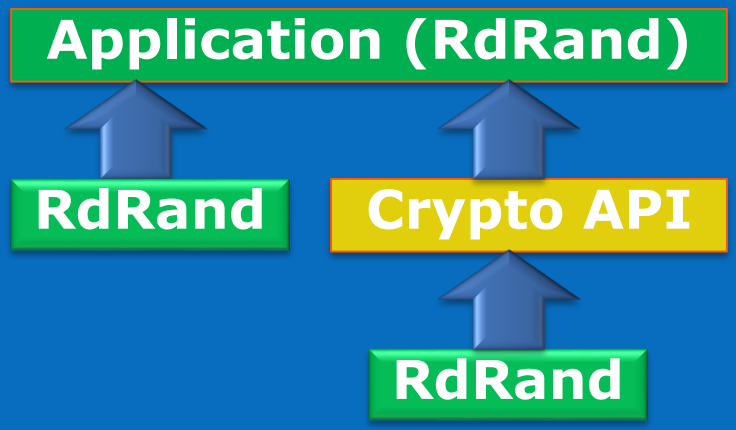
RDSEED (*0F C7 /7*) introduced on Broadwell/Goldmont

- Separate CPUID bit indicates availability
- Documented in SDM on intel.com
- CF=1 implies entropy delivered
- CF=0 implies no entropy available
- VT-x execution control for RDSEED
- Secondary based VT-x exit control for “RDSEED exiting”

Performance

Direct access to random numbers through RdRand bypasses OS, driver, and associated overhead





On-chip entropy source - no off-chip bus or I/O delays

Latency comparable to software PRNGs

Highly scalable

Summary

We described the “platform entropy problem” and showed how we are in the process of making this long standing security problem “just go away” - Randomness anywhere anytime!

We are succeeding in getting our DRNG and associated RdRand/RdSeed SW interfaces ***deployed*** across ALL Intel HW products!

Collaborate with us on getting the resultant high quality/high performance entropy widely ***used***, wherever needed, in your SW products

Acknowledgements

The original RdRand/DRNG architecture team - Ernie Brickell, James Coke, George Cox, Charles Dike, Martin Dixon, Steve Fischer, Ed Gamsaragan, Shay Gueron, Howard Herbert, DJ Johnston, Greg Piper, Guna Thuraisingham, Jesse Walker

The DRNG design/implementation team - George Cox, Charles Dike, and DJ Johnston

The integration teams across > 40 product embeddings

The product enabling teams working with OSVs/ISVs

Backup

NEW ISA Instructions: RDSEED

Feature Definition
The RDSEED instruction, companion to the RDRAND instruction, completes our solution of the "platform entropy problem"
Where is the feature on the platform?
In the Broadwell client/server processors.
What does it do? How will this feature work? ie How will it be activated?
RDSEED provides a high quality/high performance Entropy Source for (re)seeding SW PRNGs (e.g., Microsoft, Linux, OpenSSL, RSA)
Requirements/Dependencies
Given an instruction interface, RDSEED avoids any OS or library enabling dependencies and can be directly used by any SW at any protection level or processor state.
BOM Cost/Saving
RDSEED uses a small extension to the existing underlying Digital Random Number Generator (DRNG) (used to support RDRAND)
Is it an improvement over an existing feature/prev. gen?
RDSEED (along with RDRAND) fills out our standards compliant (e.g., NIST SP800-90 A, B, and C) HW-based Random Number Generator portfolio
Why is it useful for an end user? Why do we need to put it on the platform?
Give Intel's customers the "common brand promise" of high quality/high performance entropy (or random numbers) everywhere across ALL Intel products in which it is embedded.
Competition/Are there competing technologies that is/are similar or better than this feature?
There are not any competitive features in other company's products

Design Wins/TTM plans
Widely anticipated (e.g., Microsoft and Linux) – will follow the RDRAND enabling approach
Early feedback (OEM/research)
Will be used widely as soon as available
Target OEM/Geo
Everywhere
Availability/Is it part of GBB?
Basic broad security enabling - Not SKUed
Segment Positioning
Will be in ALL future Intel large/small core processors
CMG asks
N/A at this point

Key Marketing Message

Much of computer security is based upon the use of cryptography. Cryptography is based upon two things:

- Good algorithms (e.g., AES) and
- Good keys (e.g., good random numbers).

Historically, computing platforms broadly have not had high quality/high performance random numbers.

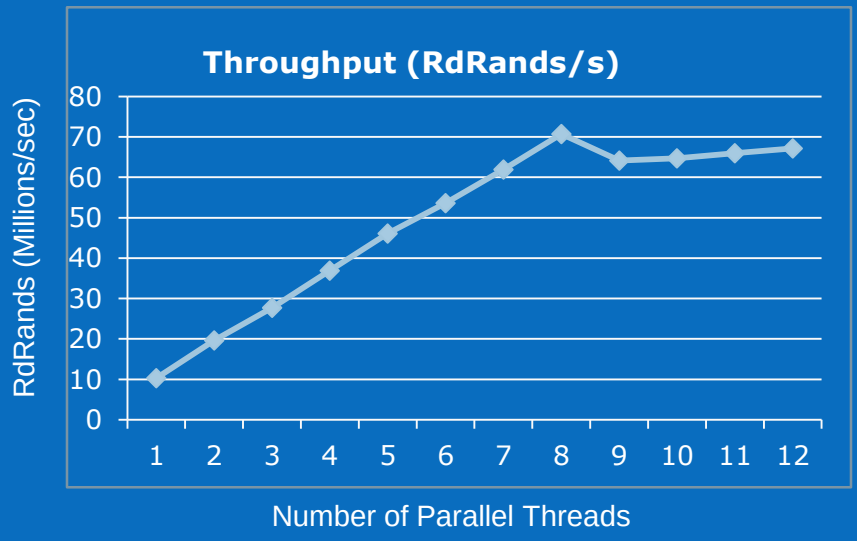
By enabling ALL future Intel platforms with our RDSEED (and RDRAND) instructions (supported by our underlying Digital Random Number Generator (DRNG)), we:

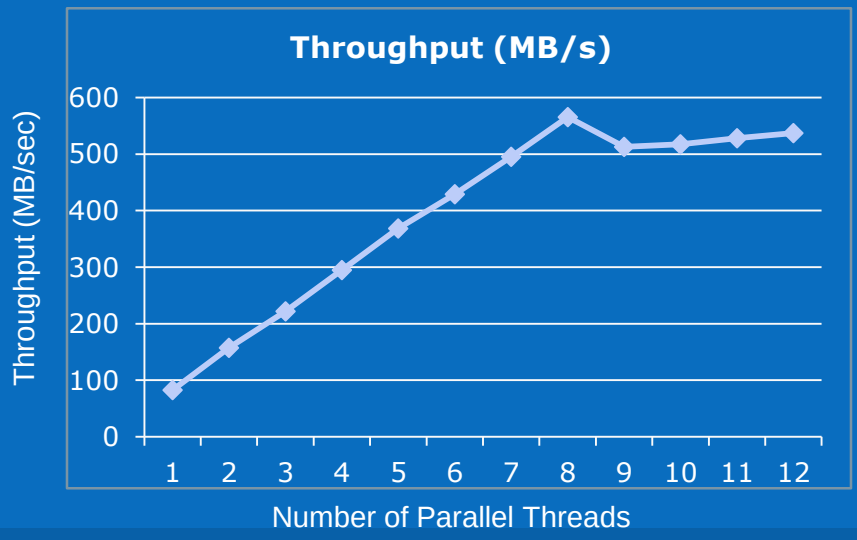
- Give Intel's customers the "common brand promise" of high quality/high performance entropy (or random numbers) everywhere across ALL Intel products in which it is embedded and Makes the long standing "Platform Entropy Problem" just "goes away" on future Intel-based platforms.

Measured Throughput

Preliminary data from pre-production Ivy Bridge sample[1]

- Up to 70 million RdRand invocations per second
- 500+ Million Bytes of random data per second
- Throughput ceiling is insensitive to number of contending parallel threads
  - Steady state maintained at peak performance





[1]Data taken from Intel® processor codename Ivy Bridge early engineering sample board. Quad core, 4 GB memory, hyper-threading enabled. Software: LINUX* Fedora 14, gcc version 4.6.0 (experimental) with RdRand support, test uses pthreads kernel API

Response Time and Reseeding Frequency

Preliminary data from pre-production Ivy Bridge sample[1]

RdRand Response Time

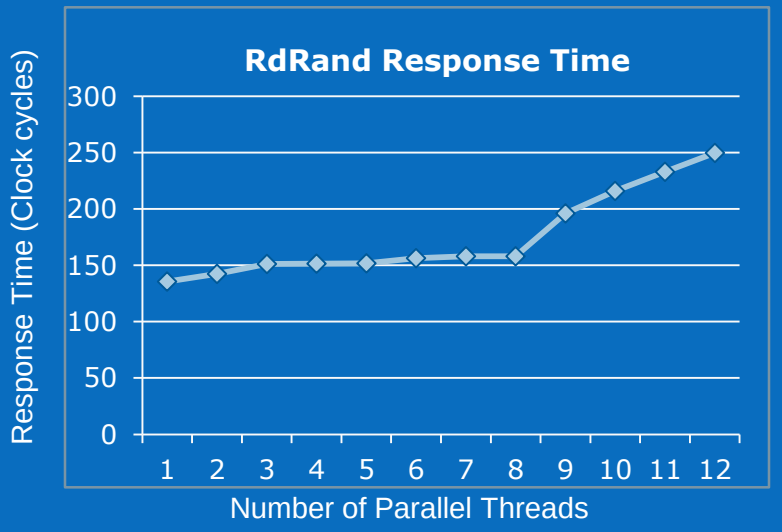
~150 clocks per invocation

(Note: Varies with CPU clock frequency since constraint is shared data path from DRNG to cores.)

Little contention until 8 threads

- (or 4 threads on 2 core chip)

Simple linear increase as additional threads are added



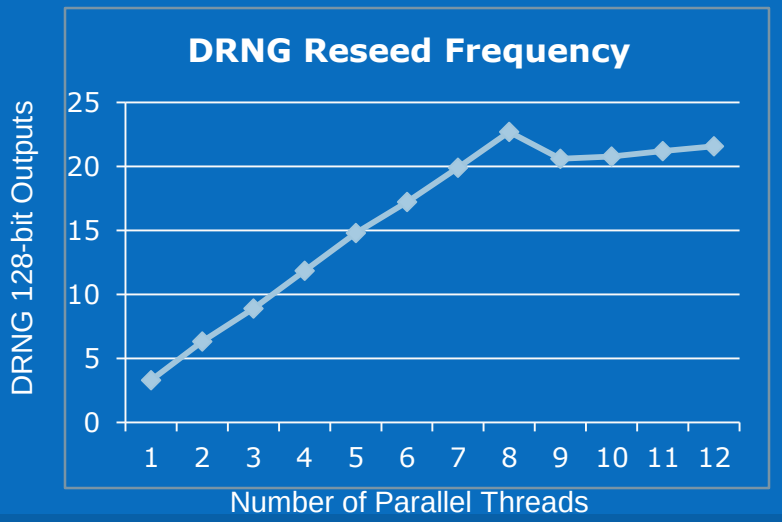
DRNG Reseed Frequency

Single thread worst case: Reseeds every 4 RdRand invocations

Multiple thread worst case: Reseeds every 23 RdRand invocations

At slower invocation rate, can expect reseed before every 2 RdRand calls

- ❑ NIST SP 800-90 recommends ≤ 2^{48}



[1]Data taken from Intel® processor codename Ivy Bridge early engineering sample board. Quad core, 4 GB memory, hyper-threading enabled. Software: LINUX* Fedora 14, gcc version 4.6.0 (experimental) with RdRand support, test uses pthreads kernel API