Randomness

Daniel J. Bernstein
University of Illinois at Chicago
Technische Universiteit Eindhoven

Tanja Lange
Technische Universiteit Eindhoven

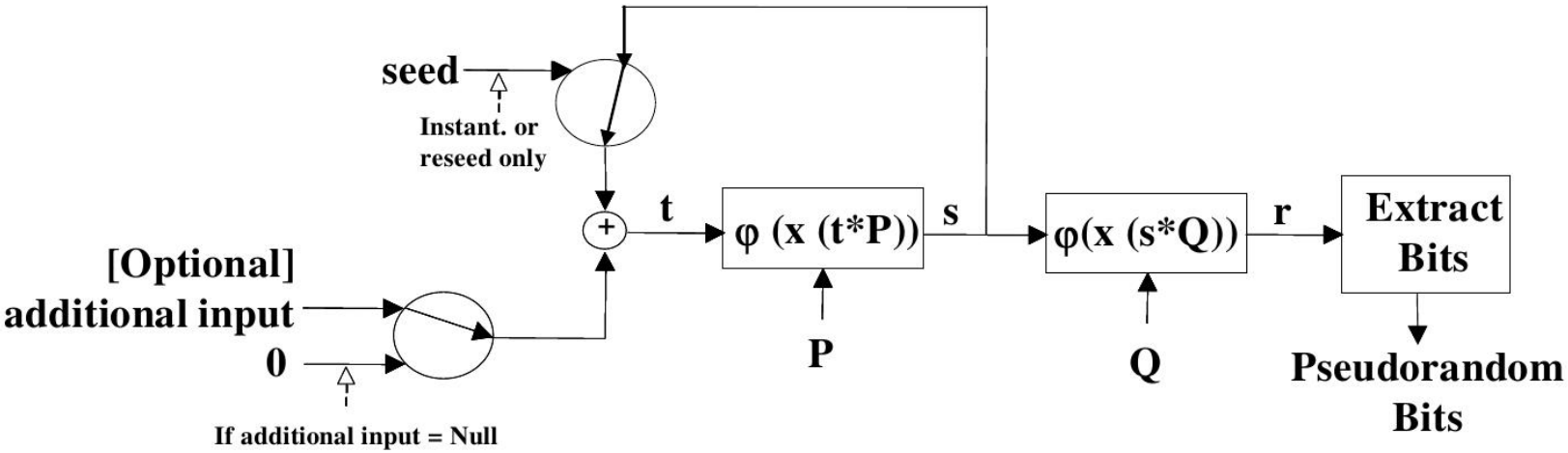
Preliminary analysis of some submissions to snakeoil.cr.yp.to

Daniel J. Bernstein
University of Illinois at Chicago
Technische Universiteit Eindhoven

Tanja Lange
Technische Universiteit Eindhoven

DUAL_EC RNG: history part I

Earliest public source (?) June 2004, draft of ANSI X9.82:



φ gives all but the top 16 bits

$\Rightarrow$ about 2^{15} points sQ match given string.

Claim:

Dual_EC_DRBG is based on the following hard problem, sometimes known as the "elliptic curve discrete logarithm problem" (ECDLP): given points P and Q on an elliptic curve of order n, find a such that $Q = aP$.

DUAL_EC RNG: common public history part II

- Gjøsteen (March 2006): Output sequence is biased.
  "While the practical impact of these results are modest, it is hard to see how these flaws would be acceptable in a pseudo-random bit generator based on symmetric cryptographic primitives. They should not be accepted in a generator based on number-theoretic assumptions."
- Brown (March 2006): "This proof makes essential use of Q being random." If d with $dQ = P$ is known then $dR_i = S_{i+1}$, concludes that then there is a distinguisher.
- Sidorenko & Schoenmakers (May 2006):
  Can amplify bias in output sequence is even more.
  NIST answer: Too late to change, already implemented.
- June 2006: SP800-90 gets published.
- Shumow & Ferguson (August 2007): Backdoor if d is known.

DUAL_EC RNG: common public history part II

- Gjøsteen (March 2006): Output sequence is biased.
  "While the practical impact of these results are modest, it is hard to see how these flaws would be acceptable in a pseudo-random bit generator based on symmetric cryptographic primitives. They should not be accepted in a generator based on number-theoretic assumptions."
- Brown (March 2006): "This proof makes essential use of Q being random." If d with $dQ = P$ is known then $dR_i = S_{i+1}$, concludes that then there is a distinguisher.
- Sidorenko & Schoenmakers (May 2006):
  Can amplify bias in output sequence is even more.
  NIST answer: Too late to change, already implemented.
- June 2006: SP800-90 gets published.
- Shumow & Ferguson (August 2007): Backdoor if d is known.
- Already June 2006 version of NIST standard has appendix about choosing points verifiably at random but states "To avoid using potentially weak points, the points specified in Appendix A.1 should be used."

September 2013: NSA Bullrun program

- (TS//SI//REL TO USA, FVEY) Insert vulnerabilities into commercial encryption systems, IT systems, networks, and endpoint communications devices used by targets.
- (TS//SI//REL TO USA, FVEY) Collect target network data and metadata via cooperative network carriers and/or increased control over core networks.
- (TS//SI//REL TO USA, FVEY) Leverage commercial capabilities to remotely deliver or receive information to and from target endpoints.
- (TS//SI//REL TO USA, FVEY) Exploit foreign trusted computing platforms and technologies.
- (TS//SI//REL TO USA, FVEY) Influence policies, standards and specification for commercial public key technologies.
- (TS//SI//REL TO USA, FVEY) Make specific and aggressive investments to facilitate the development of a robust exploitation capability against Next-Generation Wireless (NGW) communications.

September 2013: NSA Bullrun program

- (TS//SI//REL TO USA, FVEY) Insert vulnerabilities into commercial encryption systems, IT systems, networks, and endpoint communications devices used by targets.
- (TS//SI//REL TO USA, FVEY) Collect target network data and metadata via cooperative network carriers and/or increased control over core networks.
- (TS//SI//REL TO USA, FVEY) Leverage commercial capabilities to remotely deliver or receive information to and from target endpoints.
- (TS//SI//REL TO USA, FVEY) Exploit foreign trusted computing platforms and technologies.
- (TS//SI//REL TO USA, FVEY) Influence policies, standards and specification for commercial public key technologies.
- (TS//SI//REL TO USA, FVEY) Make specific and aggressive investments to facilitate the development of a robust exploitation capability against Next-Generation Wireless (NGW) communications.

Later NYT names Dual_EC_DRBG...

September 2013: NSA Bullrun program

- (TS//SI//REL TO USA, FVEY) Insert vulnerabilities into commercial encryption systems, IT systems, networks, and endpoint communications devices used by targets.
- (TS//SI//REL TO USA, FVEY) Collect target network data and metadata via cooperative network carriers and/or increased control over core networks.
- (TS//SI//REL TO USA, FVEY) Leverage commercial capabilities to remotely deliver or receive information to and from target endpoints.
- (TS//SI//REL TO USA, FVEY) Exploit foreign trusted computing platforms and technologies.
- (TS//SI//REL TO USA, FVEY) Influence policies, standards and specification for commercial public key technologies.
- (TS//SI//REL TO USA, FVEY) Make specific and aggressive investments to facilitate the development of a robust exploitation capability against Next-Generation Wireless (NGW) communications.

Later NYT names Dual_EC_DRBG. . . but surely nobody uses that piece of shit?!

September 2013: NSA Bullrun program

- (TS//SI//REL TO USA, FVEY) Insert vulnerabilities into commercial encryption systems, IT systems, networks, and endpoint communications devices used by targets.
- (TS//SI//REL TO USA, FVEY) Collect target network data and metadata via cooperative network carriers and/or increased control over core networks.
- (TS//SI//REL TO USA, FVEY) Leverage commercial capabilities to remotely deliver or receive information to and from target endpoints.
- (TS//SI//REL TO USA, FVEY) Exploit foreign trusted computing platforms and technologies.
- (TS//SI//REL TO USA, FVEY) Influence policies, standards and specification for commercial public key technologies.
- (TS//SI//REL TO USA, FVEY) Make specific and aggressive investments to facilitate the development of a robust exploitation capability against Next-Generation Wireless (NGW) communications.

Later NYT names Dual_EC_DRBG. . . but surely nobody uses that piece of shit?!

NIST's DRBG Validation List: RSA's BSAFE has Dual_EC_DRBG enabled and default.

September 2013: NSA Bullrun program

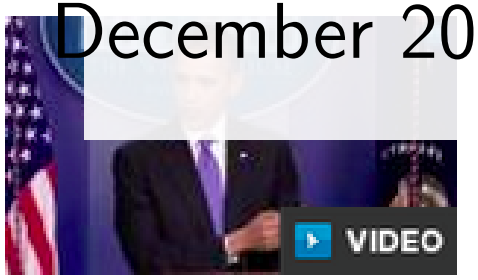
- (TS//SI//REL TO USA, FVEY) Insert vulnerabilities into commercial encryption systems, IT systems, networks, and endpoint communications devices used by targets.
- (TS//SI//REL TO USA, FVEY) Collect target network data and metadata via cooperative network carriers and/or increased control over core networks.
- (TS//SI//REL TO USA, FVEY) Leverage commercial capabilities to remotely deliver or receive information to and from target endpoints.
- (TS//SI//REL TO USA, FVEY) Exploit foreign trusted computing platforms and technologies.
- (TS//SI//REL TO USA, FVEY) Influence policies, standards and specification for commercial public key technologies.
- (TS//SI//REL TO USA, FVEY) Make specific and aggressive investments to facilitate the development of a robust exploitation capability against Next-Generation Wireless (NGW) communications.

Later NYT names Dual_EC_DRBG. . . but surely nobody uses that piece of shit?!

NIST's DRBG Validation List: RSA's BSAFE has Dual_EC_DRBG enabled and default.

NIST re-opens discussions on SP800.90; recommmends against using Dual_EC.
RSA suggests changing default in BSAFE.

RELATED VIDEO



Obama on surveillance: "There may be another way of skinning the cat"

December 2013

(Reuters) - As a key part of a campaign to embed encryption software that it could crack into widely used computer products, the U.S. National Security Agency arranged a secret $10 million contract with RSA, one of the most influential firms in the computer security industry, Reuters has learned.

Documents leaked by former NSA contractor Edward Snowden show that the NSA created and promulgated a flawed formula for generating random numbers to create a "back door" in encryption products, the New York Times reported in September. Reuters later reported that RSA became the most important distributor of that formula by rolling it into a software tool called Bsafe that is used to enhance security in personal computers and many other products.

Undisclosed until now was that RSA received $10 million in a deal that set the NSA formula as the preferred, or default, method for number generation in the BSafe software, according to two sources familiar with the contract. Although that sum might seem paltry, it represented more than a third of the revenue that the relevant division at RSA had taken in during the entire previous year, securities filings show.

December 22, 2013

Recent press coverage has asserted that RSA entered into a "secret contract" with the NSA to incorporate a known flawed random number generator into its BSAFE encryption libraries. We categorically deny this allegation.

We have worked with the NSA, both as a vendor and an active member of the security community. We have never kept this relationship a secret and in fact have openly publicized it. Our explicit goal has always been to strengthen commercial and government security.

Key points about our use of Dual EC DRBG in BSAFE are as follows:

- We made the decision to use Dual EC DRBG as the default in BSAFE toolkits in 2004, in the context of an industry-wide effort to develop newer, stronger methods of encryption. At that time, the NSA had a trusted role in the community-wide effort to strengthen, not weaken, encryption.
- This algorithm is only one of multiple choices available within BSAFE toolkits, and users have always been free to choose whichever one best suits their needs.
- We continued using the algorithm as an option within BSAFE toolkits as it gained acceptance as a NIST standard and because of its value in FIPS compliance. When concern surfaced around the algorithm in 2007, we continued to rely upon NIST as the arbiter of that discussion.
- When NIST issued new guidance recommending no further use of this algorithm in September 2013, we adhered to that guidance, communicated that recommendation to customers and discussed the change openly in the media.

RSA, as a security company, never divulges details of customer engagements, but we also categorically state that

How expensive is using the backdoor?

Rereading the standard:
"$x(A)$ is the x-coordinate of the point A on the curve, given in affine coordinates. An implementation may choose to represent points internally using other coordinate systems; for instance, when efficiency is a primary concern. In this case, a point shall be translated back to affine coordinates before $x()$ is applied."

How expensive is using the backdoor?

Rereading the standard:
"$x(A)$ is the x-coordinate of the point A on the curve, given in affine coordinates. An implementation may choose to represent points internally using other coordinate systems; for instance, when efficiency is a primary concern. In this case, a point shall be translated back to affine coordinates before $x()$ is applied."

Given $r_i = \varphi(x(s_i Q))$, $r_{i+1} = \varphi(x(s_{i+1} Q))$, and NSA backdoor $d = log_P(Q)$.

1. Expand r_i to candidate $Q_i = s_i Q$,
   [50% chance; if fail move on to next candidate]
2. compute candidate $P_{i+1} = dQ_i$ and $s_{i+1} = \varphi(x(P_{i+1}))$
3. check, $\varphi(x(s_{i+1} Q))$ against r_{i+1}.
   [if fail, goto 1.; else most likely done!]

In practice

Timings on Core i7 M620

missing	16 bits	24 bits	32 bits
1 core	20s	85m	15d4h

In practice

Timings on Core i7 M620

missing	16 bits	24 bits	32 bits
1 core	20s	85m	15d4h
64k cores			20s

In practice

Timings on Core i7 M620

missing	16 bits	24 bits	32 bits
1 core	20s	85m	15d4h
64k cores			20s

From the standard:

"For performance reasons, the value of outlen should be set to the maximum value as provided in Table 4."

Don't give us fewer bits!

US 20070189527A1

(19) **United States**

(12) **Patent Application Publication** (10) **Pub. No.: US 2007/0189527 A1**

Brown et al. (43) **Pub. Date: Aug. 16, 2007**

(54) **ELLIPTIC CURVE RANDOM NUMBER GENERATION**

(76) Inventors: **Daniel R. L. Brown**, Mississauga (CA); **Scott A. Vanstone**, Campbellville (CA)

Correspondence Address:
Blake, Cassels & Graydon LLP
Commerce Court West
P.O. Box 25
Toronto, ON M5L 1A9 (CA)

(21) Appl. No.: **11/336,814**

(22) Filed: **Jan. 23, 2006**

Related U.S. Application Data

(60) Provisional application No. 60/644,982, filed on Jan. 21, 2005.

Publication Classification

(51) **Int. Cl.**
H04L 9/00 (2006.01)

(52) **U.S. Cl.** **380/44**

(57) **ABSTRACT**

An elliptic curve random number generator avoids escrow keys by choosing a point Q on the elliptic curve as verifiably random. An arbitrary string is chosen and a hash of that string computed. The hash is then converted to a field element of the desired field, the field element regarded as the x-coordinate of a point Q on the elliptic curve and the x-coordinate is tested for validity on the desired elliptic curve. If valid, the x-coordinate is decompressed to the point Q, wherein the choice of which is the two points is also derived from the hash value. Intentional use of escrow keys can provide for back up functionality. The relationship between P and Q is used as an escrow key and stored by for a security domain. The administrator logs the output of the generator to reconstruct the random number with the escrow key.

Hat tip @nymble.

Snippets from the patent

can provide for back up functionality. The relationship between P and Q is used as an escrow key and stored by for a security domain. The administrator logs the output of the generator to reconstruct the random number with the escrow key.

accounts. A more seamless method may be applied for cryptographic applications. For example, in the SSL and TLS protocols, which are used for securing web (HTTP) traffic, a client and server perform a handshake in which their first actions are to exchange random values sent in the clear.

[**0054**] Many other protocols exchange such random values, often called nonces. If the escrow administrator observes these nonces, and keeps a log of them **508**, then later it may be able to determine the necessary r value. This

Details on Intel's RNG

Details on Intel's RNG

[7] D. J. Johnston, "Mircoarchitecture Specification (MAS) for PP-DRNG," Intel Corporation (unpublished), V1.4, 2009.

[8] C. E. Dike, "3 Gbps Binary RNG Entropy Source," Intel Corporation (unpublished), 2011.

[9] C. E. Dike and S. Gueron, "Digital Symmetric Random Number Generator Mathematics," Intel Corporation (unpublished), 2009.

(References from "Analysis of Intel's Ivy Bridge Digital Random Number Generator Prepared for Intel" by Mike Hamburg, Paul Kocher, and Mark E. Marson. Cryptography Research, Inc.)

Design (from CRI report)

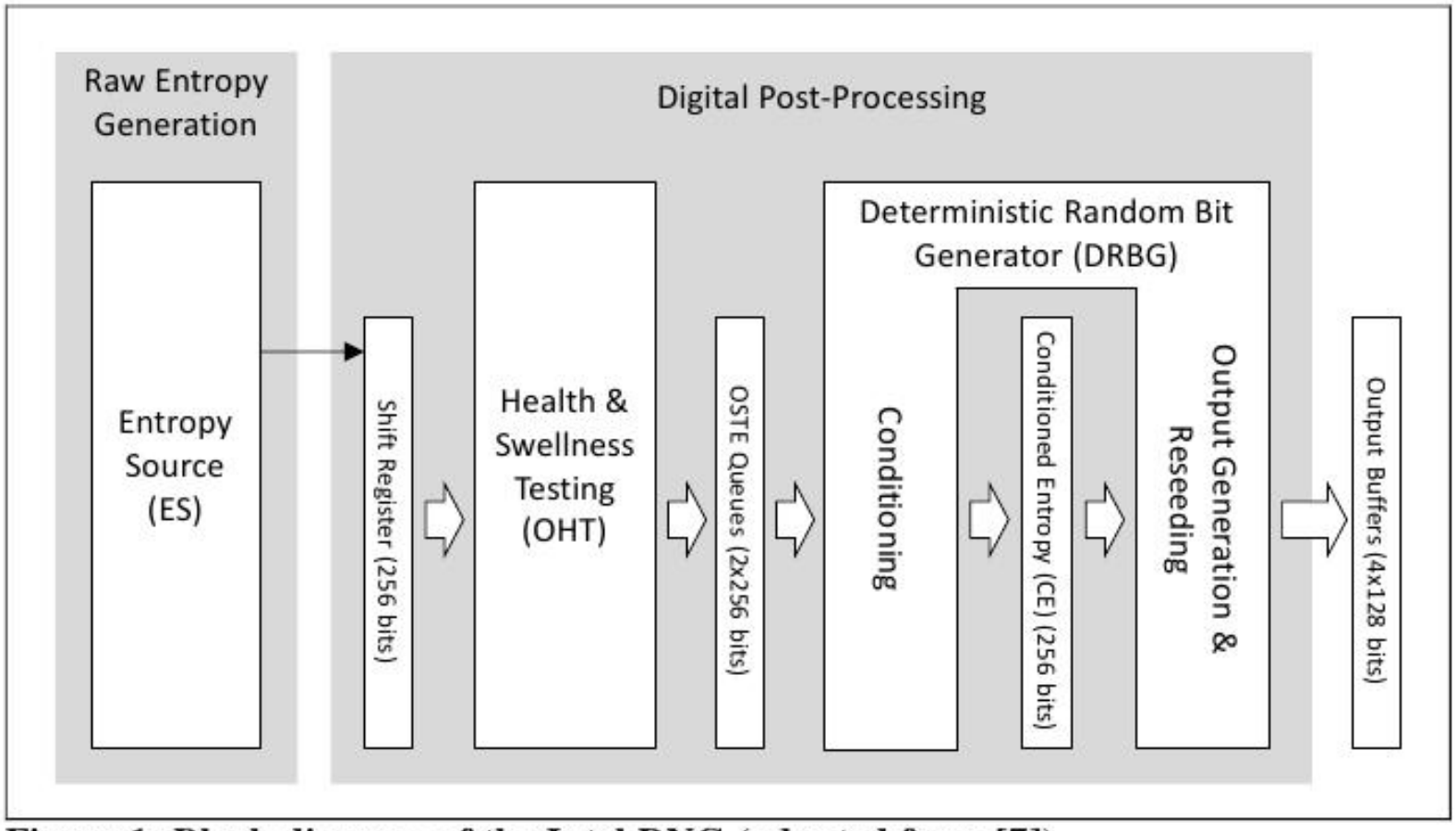


Figure 1: Block diagram of the Intel RNG (adapted from [7])

Entropy Source (from CRI report)

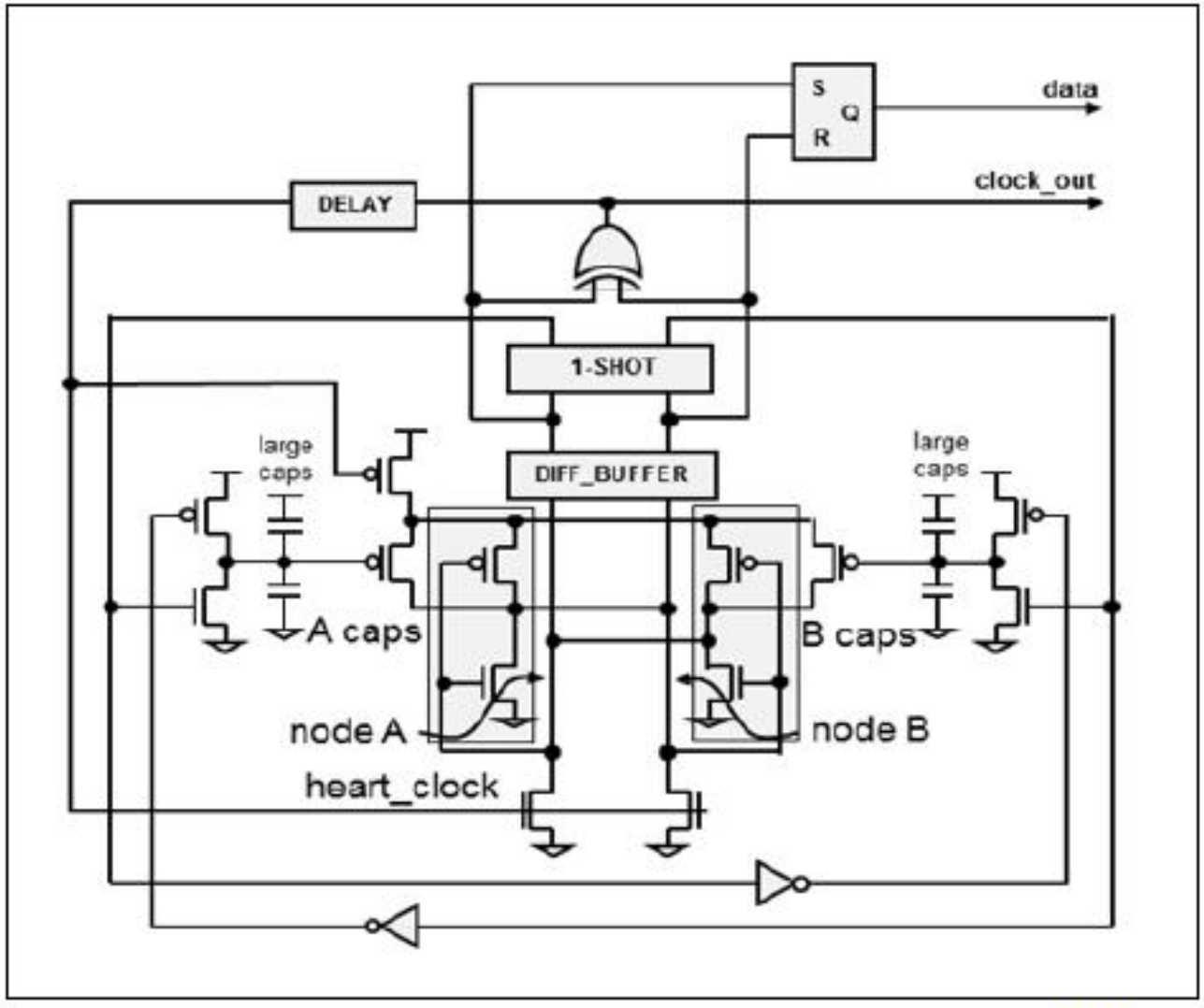


Figure 2: Entropy source for the Intel RNG (from [8])

Design (from CRI report)

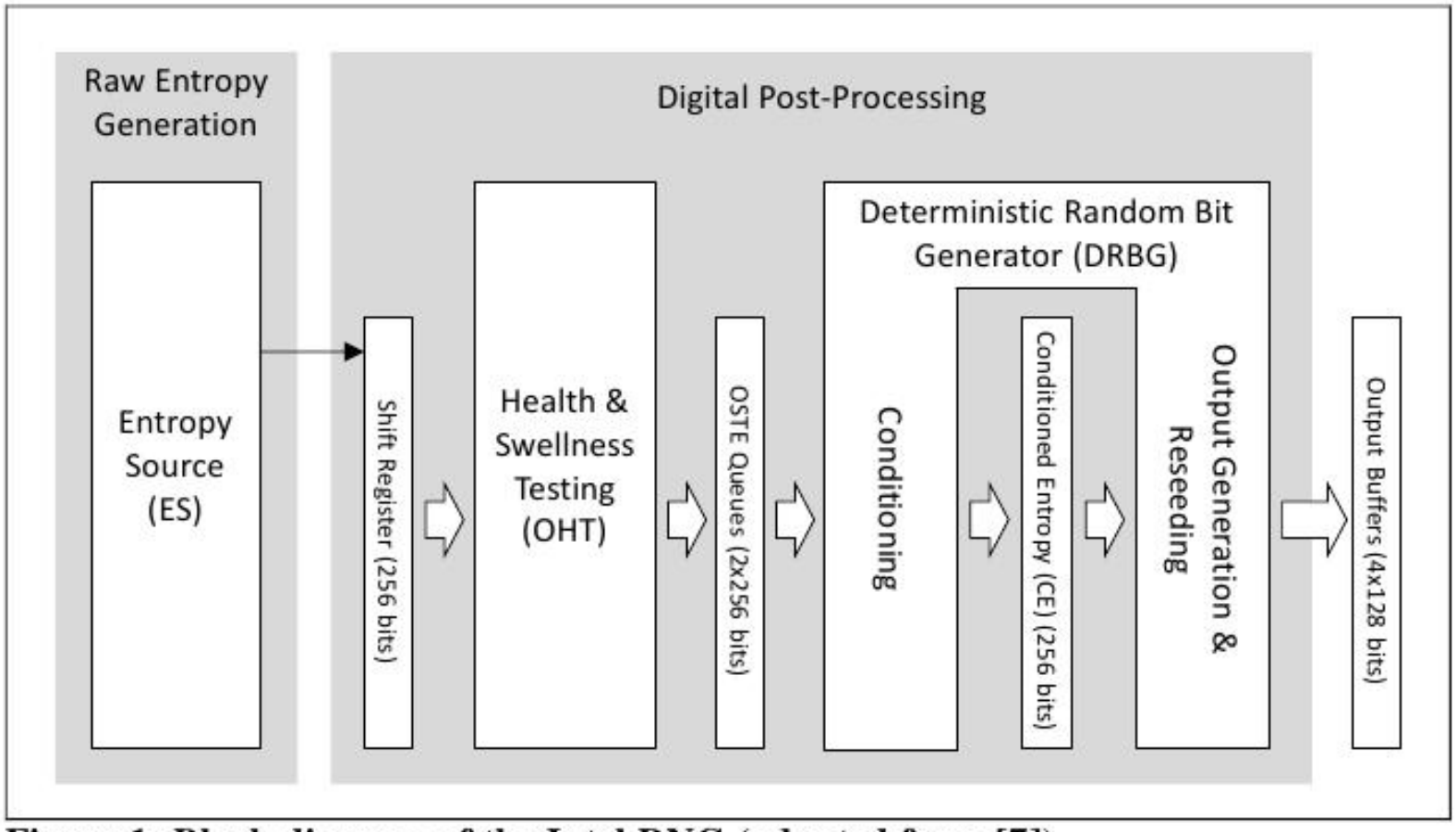


Figure 1: Block diagram of the Intel RNG (adapted from [7])

"It uses the counter mode CTR_DRBG construction as defined in [2], with AES-128 as the block cipher."

Intel assurances – David Johnston

I've examined my own RNG with electron microscopes and picoprobes. So I and a number of test engineers know full well that the design hasn't been subverted. For security critical systems, having multiple entropy sources is a good defense against a single source being subverted. But if an Intel processor were to be subverted, there are better things to attack, like the microcode or memory protection or caches. We put a lot of effort into keeping them secure, but as with any complex system it's impossible to know that you've avoided all possible errors, so maintaining the security of platforms is an ongoing battle. [..] But the implication at the top of this thread is that we were leaned on by the government to undermine our own security features. I know for a fact that I was not leant on by anyone to do that. X9.82 took my contributions and NIST is taking about half my contributions, but maybe they're slowly coming around to my way of thinking on online entropy testing. If I ultimately succeed in getting those specs to be sane, we better hope that I am sane.

Scary Paper of the Year: *Stealthy Dopant-Level Hardware Trojans*

by Becker, Regazzoni, Paar, and Burleson, CHES 2013

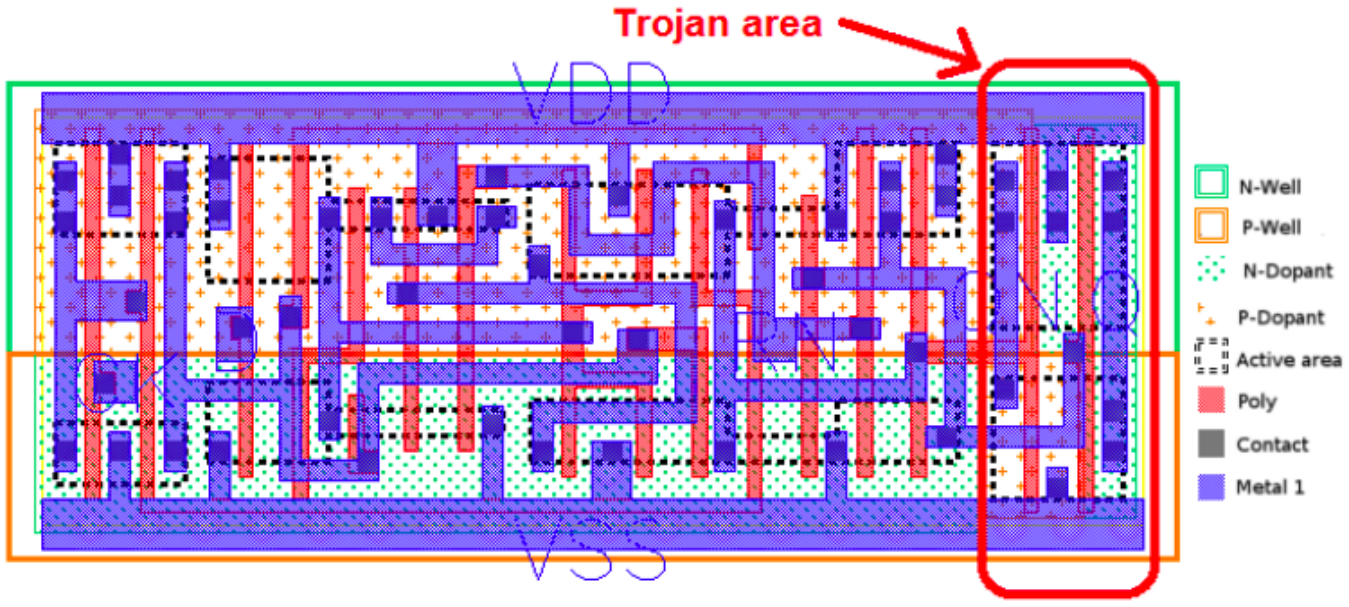


Fig. 2. Layout of the Trojan DFFR_X1 gate. The gate is only modified in the highlighted area by changing the dopant mask. The resulting Trojan gate has an output of $Q = V_{DD}$ and $QN = GND$.

Scary recommendations

CRI: “Because the Ivy Bridge RNG is implemented as an instruction in the CPU, it is much simpler to use than other hardware-based RNGs and avoids the need for additional software layers that could introduce bugs.”

Johnston: “Just use the output of the RDRAND instruction wherever you need a random number.” (github search for RDRAND has 33 609 code results)

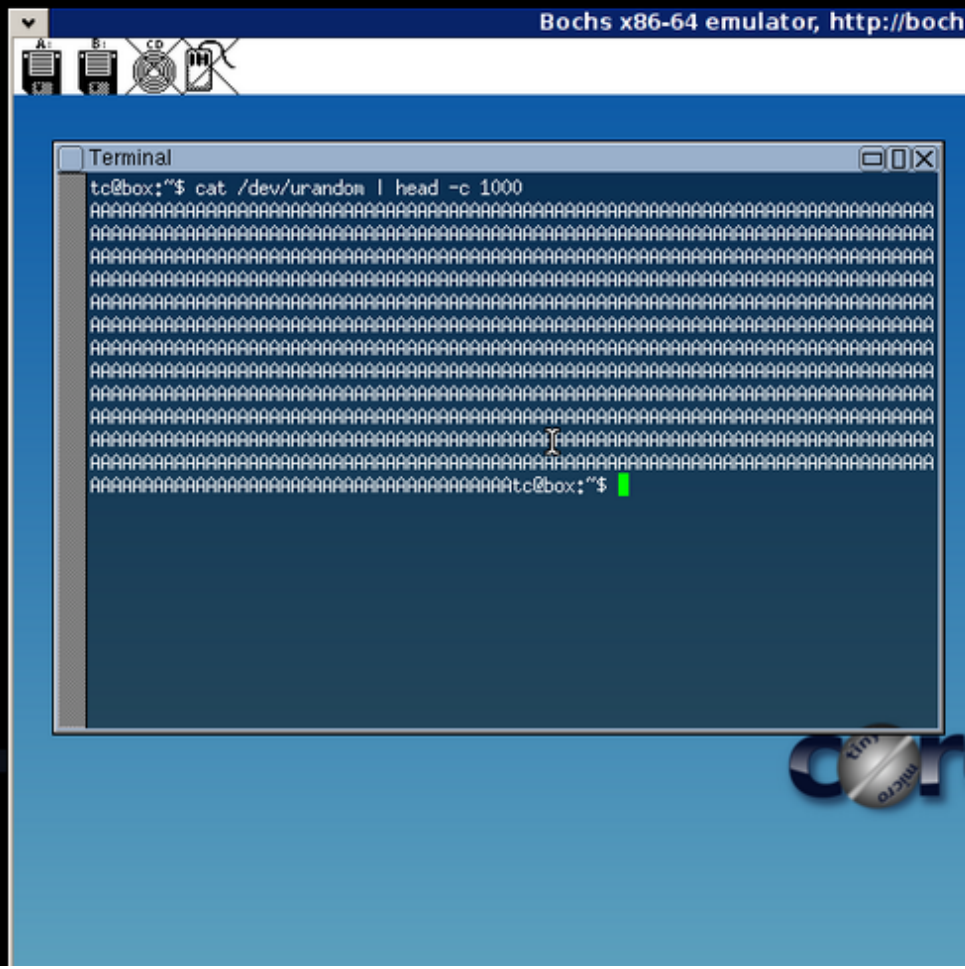
Intel manual 325462, June 2013, page 177:
"extremely rare cases" RDRAND "will return no data".
Also: "returning no data transitorily" because of "heavy load".
Recommendation to "retry for a limited number of iterations"; the subsequent explanation makes clear that this catches the "transitory" failures but not the "extremely rare" failures.
There is no quantification of "extremely rare".

Linux use of RDRAND

```
 -rw-r--r-- H. Peter Anvin     2012-07-27 22:26 random.c:
/*
* In case the hash function has some recognizable output
* pattern, we fold it in half. Thus, we always feed back
* twice as much data as we output.
*/
hash.w[0] ^= hash.w[3];
hash.w[1] ^= hash.w[4];
hash.w[2] ^= rol32(hash.w[2], 16);
/*
* If we have a architectural hardware random number
* generator, mix that in, too.
*/
for (i = 0; i < LONGS(EXTRACT_SIZE); i++) {
    unsigned long v;
    if (!arch_get_random_long(&v))
            break;
    hash.l[i] ^= v;
}
memcpy(out, &hash, EXTRACT_SIZE);
memset(&hash, 0, sizeof(hash));
```

RDRAND backdoor proof of concept – Taylor Hornby

```
40   }
41 #endif
42
43   Bit16u val_16 = 0;
44
45   if (HW_RANDOM_GENERATOR_READY) {
46     val_16 |= rand() & 0xff;  // hack using std C rand() function
47     val_16 <<= 8;
48     val_16 |= rand() & 0xff;
49
50     setEFlagsOSZAPC(EFlagsCFMask);
51   }
52   else {
53     setEFlagsOSZAPC(0);
54   }
55
56   BX_WRITE_16BIT_REG(i->dst(), val_16);
57
58   BX_NEXT_INSTR(i);
59 }
60
61 BX_INSF_TYPE BX_CPP_AttrRegparmN(1) BX_CPU_C::RDRAND_Ed(bxInstruction_c *i)
62 {
63   Bit32u val_32 = 0;
64
65   BX_INFO(("In RDRAND_Ed!"));
66
67   Bit32u edx = get_reg32(BX_32BIT_REG_EDX);
68
69   if (EIP > 0xc0387d20 && EIP < 0xc0387e19) {
70     BX_INFO(("Triggering backdoor!"));
71     Bit32u at_edx = read_virtual_dword(BX_SEG_REG_DS, edx);
72     val_32 = at_edx ^ 0x41414141;
73   }
74
75   setEFlagsOSZAPC(EFlagsCFMask);
76
77   BX_WRITE_32BIT_REGZ(i->dst(), val_32);
78
79   BX_NEXT_INSTR(i);
80 }
81
82 #if BX_SUPPORT_X86_64
83 BX_INSF_TYPE BX_CPP_AttrRegparmN(1) BX_CPU_C::RDRAND_Eq(bxInstruction_c *i)
```



"The way RDRAND is being used in kernels <= 3.12.3 allows it to cancel out the other entropy. See extract_buf()."
"if I make RDRAND return `[EDX] ^ 0x41414141`, /dev/urandom output will be all `'A'`." Full thread

Updated in Linux repository (Dec 2013); not yet shipping

```
/*
* If we have an architectural hardware random number
* generator, use it for SHA's initial vector
*/
sha_init(hash.w);
for (i = 0; i < LONGS(20); i++) {
      unsigned long v;
      if (!arch_get_random_long(&v))
            break;
      hash.l[i] = v;
}
/* Generate a hash across the pool,
* 16 words (512 bits) at a time */
spin_lock_irqsave(&r->lock, flags);
for (i = 0; i < r->poolinfo->poolwords; i += 16)
      sha_transform(hash.w, (__u8 *)(r->pool + i), workspace);
```

Would you like to audit this?

```
2013-12-17 21:16 Theodore Ts'o      o [dev] [origin/dev] random: use the architectural HWRNG for~
2013-12-06 21:28 Greg Price         o random: clarify bits/bytes in wakeup thresholds
2013-12-07 09:49 Greg Price         o random: entropy_bytes is actually bits
2013-12-05 19:32 Greg Price         o random: simplify accounting code
2013-12-05 19:19 Greg Price         o random: tighten bound on random_read_wakeup_thresh
2013-11-29 20:09 Greg Price         o random: forget lock in lockless accounting
2013-11-29 15:56 Greg Price         o random: simplify accounting logic
2013-11-29 15:50 Greg Price         o random: fix comment on "account"
2013-11-29 15:02 Greg Price         o random: simplify loop in random_read
2013-11-29 14:59 Greg Price         o random: fix description of get_random_bytes
2013-11-29 14:58 Greg Price         o random: fix comment on proc_do_uuid
2013-11-29 14:58 Greg Price         o random: fix typos / spelling errors in comments
2013-11-16 10:19 Linus Torvalds     M-| Merge tag 'random_for_linus' of git://git.kernel.org/pub~
2013-11-03 18:24 Theodore Ts'o      | o [random_for_linus] random: add debugging code to detect ~
2013-11-03 16:40 Theodore Ts'o      | o random: initialize the last_time field in struct timer_r~
2013-11-03 07:56 Theodore Ts'o      | o random: don't zap entropy count in rand_initialize()
2013-11-03 06:54 Theodore Ts'o      | o random: printk notifications for urandom pool initializa~
2013-11-03 00:15 Theodore Ts'o      | o random: make add_timer_randomness() fill the nonblocking~
2013-10-03 12:02 Theodore Ts'o      | o random: convert DEBUG_ENT to tracepoints
2013-10-03 01:08 Theodore Ts'o      | o random: push extra entropy to the output pools
2013-10-02 21:10 Theodore Ts'o      | o random: drop trickle mode
2013-09-22 16:04 Theodore Ts'o      | o random: adjust the generator polynomials in the mixing f~
2013-09-22 15:24 Theodore Ts'o      | o random: speed up the fast_mix function by a factor of fo~
2013-09-22 15:14 Theodore Ts'o      | o random: cap the rate which the /dev/urandom pool gets re~
2013-09-21 19:42 Theodore Ts'o      | o random: optimize the entropy_store structure
2013-09-12 14:27 Theodore Ts'o      | o random: optimize spinlock use in add_device_randomness()
2013-09-12 14:10 Theodore Ts'o      | o random: fix the tracepoint for get_random_bytes(_arch)
2013-09-10 23:16 H. Peter Anvin     | o random: account for entropy loss due to overwrites
2013-09-10 23:16 H. Peter Anvin     | o random: allow fractional bits to be tracked
2013-09-10 23:16 H. Peter Anvin     | o random: statically compute poolbitshift, poolbytes, pool~
2013-09-21 18:06 Theodore Ts'o      | o random: mix in architectural randomness earlier in extra~
2013-11-11 12:20 Hannes Frederic S~ o | random32: add prandom_reseed_late() and call when nonblo~
2013-10-10 12:31 Linus Torvalds     M-| Merge tag 'random_for_linus' of git://git.kernel.org/pub~
2013-09-21 13:58 Theodore Ts'o      | o random: allow architectures to optionally define random_~
2013-09-10 10:52 Theodore Ts'o      | o random: run random_int_secret_init() run after all late_~
2013-08-30 09:39 Martin Schwidefsky o | Remove GENERIC_HARDIRQ config option
```

What would we like to see?

- Cryptographers can help here!
- Easy part: Stream cipher generates randomness from seed.
  With big seed, safe to have output overwrite old seed.
- Hard part: Need comprehensible mechanism
  to securely merge entropy sources into seed.
- Some sources are bad. Is full hashing really necessary?
- Some sources are influenced or controlled by attacker.
  Is protection against malice possible?
- Maybe helpful:
  Some malicious sources have limited time and space.
  Concatenate independent hashes of several sources,
  apply many rounds of wide permutation, then truncate?