Entropy Source and DRNG Manager ...or /dev/random in User Space

Stephan Müller
<smueller@chronox.de>

Agenda

- ESDM Goals
- ESDM Design
- Initial Seeding Strategies
- Entropy Sources

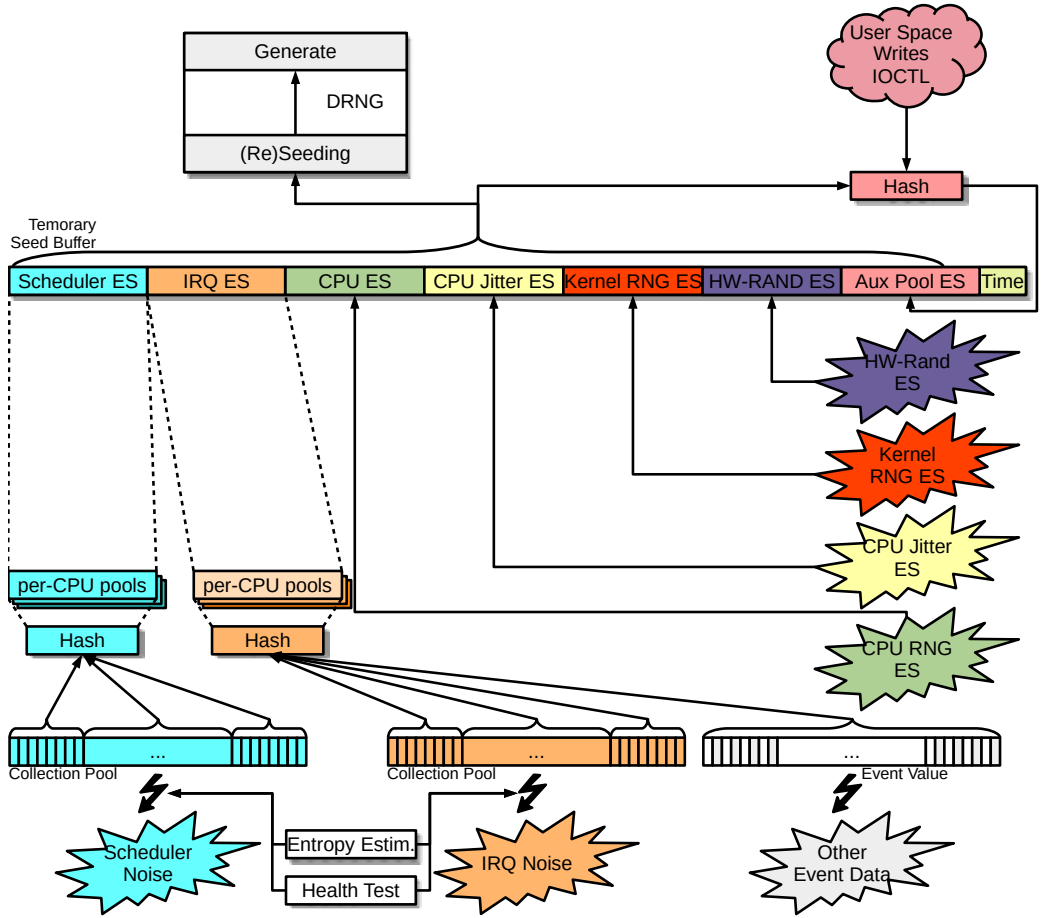
ESDM Goals

- Full user space implementation
  - No kernel changes needed – only if scheduler ES is requested
  - Independent of political games of developers
- API and ABI compliant drop-in replacement of Linux /dev/random, /dev/urandom and getrandom(2)
- Minimal dependencies on environment
  - Protobuf-c
  - FUSE / CUSE support
- Runs fully unprivileged
- Sole use of cryptography for data processing
- Flexible configuration supporting wide range of use cases
- Standards compliance: SP800-90A/B/C, AIS 20/31, FIPS IG 7.19 / D.K (use of DRBG as conditioner)

```
$ ps -efa | grep esdm
root         891       1  2 23:34 ?        00:00:00 /usr/local/bin/esdm-server -f
nobody       939     891  0 23:34 ?        00:00:00 /usr/local/bin/esdm-server -f
nobody      1050       1  0 23:34 ?        00:00:00 /usr/local/bin/esdm-cuse-random -f
nobody      1051       1  0 23:34 ?        00:00:00 /usr/local/bin/esdm-cuse-urandom -f
nobody      1054       1  0 23:34 ?        00:00:00 /usr/local/bin/esdm-proc --relabel -f -o allow_other /proc/sys/kernel/random
```

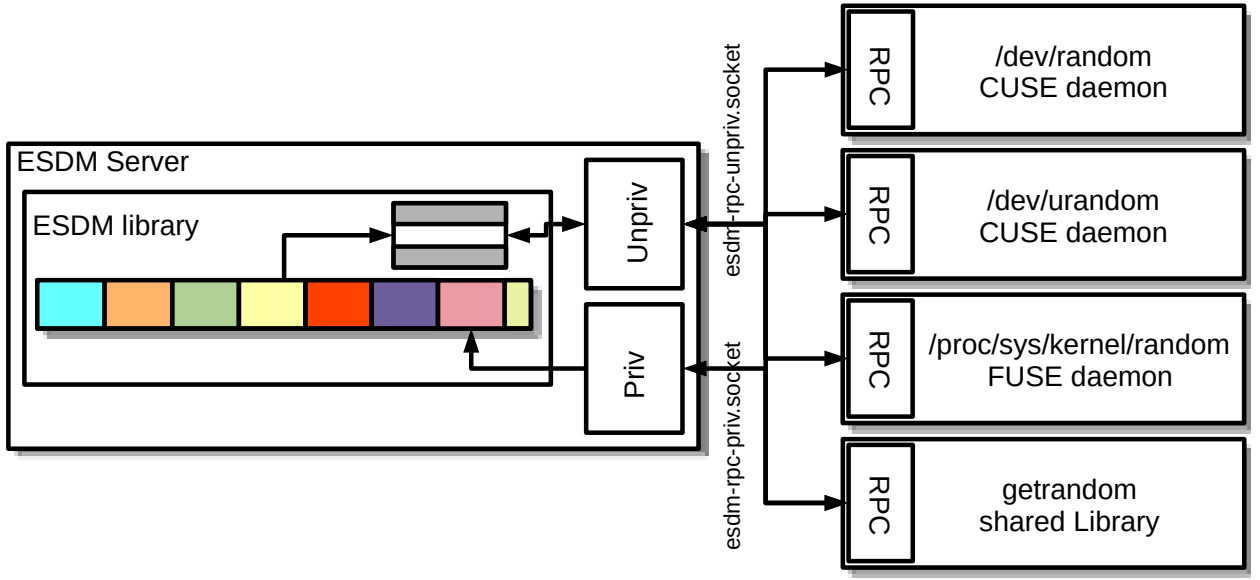
ESDM Design

- 7 Entropy Sources
  - 5 external
  - 2 internal
  - All ES treated equally
  - No domination by any ES – seeding triggered by initialization process or DRNG
- All ES can be selectively disabled at compile time
- ES data fed into DRNG
- DRNG accessible with APIs



ESDM Components

- ESDM Server
  - Using ESDM library
  - Offering RPC Interface
  - Protobuf-C RPC
  - Sensitive operations require privilege
- ESDM RPC library
  - API for other clients
- /dev/random CUSE
- /dev/urandom CUSE
- /proc/sys/kernel/random FUSE
- libesdm-getrandom.so



DRNG Output APIs

- Blocking APIs – deliver data only after fully initialized and fully seeded
  - When Linux compliant interfaces enabled:
    - /dev/random
    - getrandom() system call
- Prediction Resistance API – deliver data only after fully initialized and successful reseed returning at most data equal to the amount of entropy
  - ESDM RPC client API
  - Using /dev/random with O_SYNC
  - Using getrandom(2) with flag GRND_RANDOM
  - Compliant with:
    - FIPS IG 7.19 / D.K to use DRBG as conditioning component for seeding other DRBGs
    - German AIS 20/31 NTG.1 requirements
- Get seed: getrandom(2) with flag GRND_SEED to obtain data from entropy sources directly
- All other APIs deliver data without blocking until complete initialization
  - No guarantee of ESDM being fully initialized / seeded

DRNG Seeding

- Temporary seed buffer: concatenation of output from all ES
- Seeding during initialization: when 256 bits of entropy are available
- Seeding at runtime
  - After 2^{20} generate requests or 10 minutes
  - After forced reseed by user space
  - After new DRNG is loaded
  - At least 128 bits (SP800-90C mode: ESDM security strength) of total entropy must be available
  - 256 bits of entropy requested from each ES – ES may deliver less
  - Seed operation occurs when DRNG is requested to produce random bits
  - DRNG returns to not fully seeded when last seed with full entropy was > 2^{30} generate operations ago

```
ESDM (23:25:57) (esdm-server) Warning - Entropy Source [../esdm/esdm_es_sched.c:esdm_sched_initialize:140]: Disabling scheduler-based entropy source which is not present in kernel
ESDM (23:25:57) (esdm-server) Debug - Entropy Source [../esdm/esdm_es_mgr.c:esdm_es_mgr_initialize:397]: Initialize ES JitterRNG
ESDM (23:25:57) (esdm-server) Debug - Entropy Source [../esdm/esdm_es_jent.c:esdm_jent_initialize:53]: Jitter RNG working on current system
ESDM (23:25:57) (esdm-server) Debug - DRNG [../esdm/esdm_drng_mgr.c:esdm_drng_force_reseed:433]: force reseed of initial DRNG
ESDM (23:25:57) (esdm-server) Debug - DRNG [../esdm/esdm_drng_mgr.c:esdm_drng_seed_work_one:361]: reseed triggered by system events for DRNG on NUMA node 0
ESDM (23:25:57) (esdm-server) Debug - Entropy Source [../esdm/esdm_es_jent.c:esdm_jent_get:122]: obtained 256 bits of entropy from Jitter RNG noise source
ESDM (23:25:57) (esdm-server) Debug [../esdm/es_cpu/cpu_random_x86.h:cpu_es_x86_rdseed:98]: RDSEED support detected
ESDM (23:25:57) (esdm-server) Debug - Entropy Source [../esdm/esdm_es_cpu.c:esdm_cpu_multiplier:180]: Setting CPU ES multiplier to 1
ESDM (23:25:57) (esdm-server) Debug - Entropy Source [../esdm/esdm_es_cpu.c:esdm_cpu_get:230]: obtained 8 bits of entropy from CPU RNG entropy source
ESDM (23:25:57) (esdm-server) Debug - Entropy Source [../esdm/esdm_es_krng.c:esdm_krng_get:243]: obtained 128 bits of entropy from kernel RNG noise source
ESDM (23:25:57) (esdm-server) Debug - DRNG [../esdm/esdm_drng_mgr.c:esdm_drng_inject:280]: seeding regular DRNG with 272 bytes
ESDM (23:25:57) (esdm-server) Debug - DRNG [../esdm/esdm_drng_mgr.c:esdm_drng_inject:294]: regular DRNG stats since last seeding: 0 secs; generate calls: 0
ESDM (23:25:57) (esdm-server) Debug - DRNG [../esdm/esdm_drng_mgr.c:esdm_drng_inject:310]: regular DRNG fully seeded
ESDM (23:25:57) (esdm-server) Verbose - Entropy Source [../esdm/esdm_es_mgr.c:esdm_set_operational:250]: ESDM fully operational
ESDM (23:25:57) (esdm-server) Verbose - Entropy Source [../esdm/esdm_es_mgr.c:esdm_init_ops:355]: ESDM fully seeded with 392 bits of entropy
```

```
ESDM (22:00:20) (esdm-server) Debug - DRNG [../esdm/esdm_drng_mgr.c:esdm_drng_seed_work_one:361]: reseed triggered by system events for DRNG on NUMA node 1
ESDM (22:00:20) (esdm-server) Debug - Entropy Source [../esdm/esdm_es_jent.c:esdm_jent_get:122]: obtained 256 bits of entropy from Jitter RNG noise source
ESDM (22:00:20) (esdm-server) Debug - Entropy Source [../esdm/esdm_es_cpu.c:esdm_cpu_get:230]: obtained 8 bits of entropy from CPU RNG entropy source
ESDM (22:00:20) (esdm-server) Debug - Entropy Source [../esdm/esdm_es_krng.c:esdm_krng_get:243]: obtained 128 bits of entropy from kernel RNG noise source
ESDM (22:00:20) (esdm-server) Debug - Entropy Source [../esdm/esdm_es_aux.c:esdm_aux_get_pool:319]: obtained 0 bits by collecting 0 bits of entropy from aux pool, 0 bits of entropy remaining
ESDM (22:00:20) (esdm-server) Debug - DRNG [../esdm/esdm_drng_mgr.c:esdm_drng_inject:280]: seeding regular DRNG with 272 bytes
ESDM (22:00:20) (esdm-server) Debug - DRNG [../esdm/esdm_drng_mgr.c:esdm_drng_inject:294]: regular DRNG stats since last seeding: 0 secs; generate calls: 0
ESDM (22:00:20) (esdm-server) Debug - DRNG [../esdm/esdm_drng_mgr.c:esdm_drng_inject:310]: regular DRNG fully seeded
```

Initial Seeding Strategy I
Default Operation

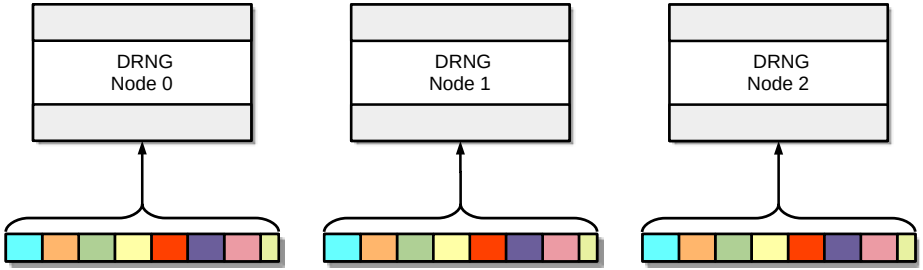
- DRNG is fully seeded with 256 bits of entropy
- Blocking interfaces released after DRNG is fully seeded
- Default applied
  - Either no specific seeding strategy compiled
  - Or specific seeding strategy is not enabled at boottime

Initial Seeding Strategy II
Entropy Source Oversampling

- Fully seeded step changed
- Compile time option
  - Function only enabled in FIPS mode
  - Function only enabled if message digest of conditioner >= 384 bits
- Final conditioning: s + 64 bit
- Initial DRNG seeding: every entropy source requested for s + 128 bits
  - Every ES alone could provide all required entropy
- All ES data concatenated into seed buffer
- Runtime debug mode: display of all processing steps
- SP800-90C compliance:
  - SP800-90A DRBG with 256-bit strength / SHA-512 vetted conditioning component
  - Complies with RBG2(NP) per default
  - Can be configured to provide RBG2(P)
- Can be used in parallel with seeding strategy III

```
# Option for: ESDM_OVERSAMPLE_ENTROPY_SOURCES
option('oversample_es', type: 'boolean', value: 'true',
       description:'''Oversampling of Entropy Sources.

If enabled, the ESDM oversamples the entropy sources compliant to SP800-90C.
The oversampling is applied in FIPS-enabled mode only.

The following oversampling is applied:
* Seed the DRNG with 128 more bits of entropy from the conditioning component.
* Inject 64 more bits of entropy from the noise source into the conditioning
  component.
''')
```

DRNG Management



- One DRNG per CPU up to maximum of 64
- Hash contexts on stack only
- Each DRNG initializes from entropy sources
- Sequential initialization of DRNG – first is CPU 0
- If one DRNG node is not yet fully seeded → use of DRNG(Node 0)
- Each DRNG instance managed independently
- To prevent reseed storm – reseed threshold different for each node
  - Node 0: 600 seconds
  - Node 1: 700 seconds
  - …
- Maximum node number is compile time option

```
# Option for: THREADING_MAX_THREADS
option('threading_max_threads', type: 'integer', min: 1, value: 64,
       description:'''Maximum number of concurrent threads supported.

This value can be set to any arbitrary number. Depending on the number
of threads, the required numbers of thread contexts are statically allocated.

The number of threads define:

* the number of concurrent DRNG instances that are maintained independent of
  each other - this value is limited by the number of found CPUs as it makes no
  sense to have more DRNG instances than CPUs that can execute them.

There is no other value that needs changing if the number of threads
shall be adjusted.

The value must not be lower than 1.
''')
```

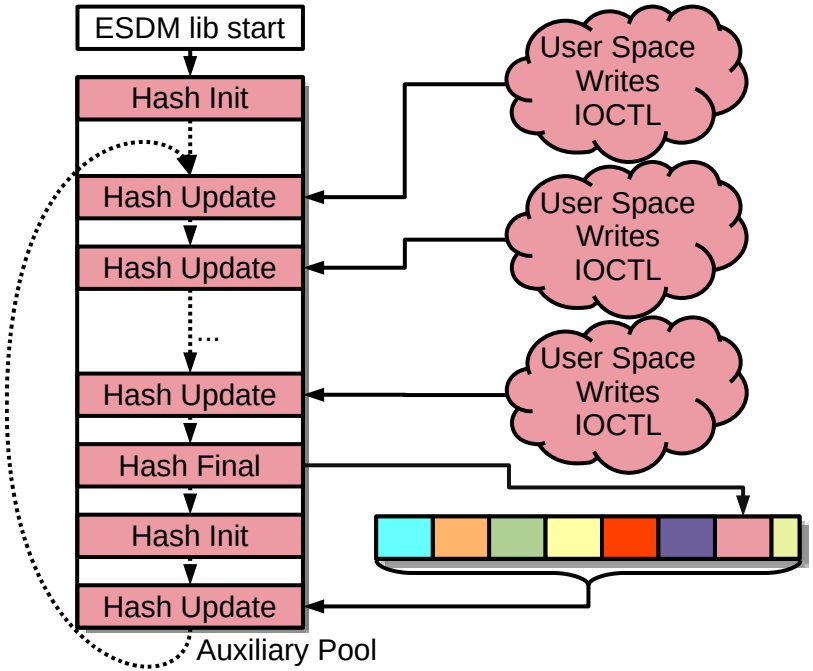
Data Processing Primitives

- Sole use of cryptographic mechanisms for data compression
- Cryptographic primitives compile-time switchable
  - DRNG, Conditioning hash
  - Built-in: SP800-90A Hash DRBG SHA2-512 / SHA2-512
  - Available:
    - ChaCha20 DRNG
    - SP800-90A Hash DRBG SHA3-512
    - SHA3-512 conditioning function
    - Well-defined API to allow other cryptographic primitive implementations
- Complete cryptographic primitive testing available
  - Full ACVP test harness available: https://github.com/smuellerDD/acvpparser
  - ChaCha20 DRNG userspace implementation: https://github.com/smuellerDD/leancrypto
- Other data processing primitives
  - Concatenation of data
  - Truncation of message digest to heuristic entropy value
- Entropy behavior of all data processing primitives based on fully understood and uncontended operations

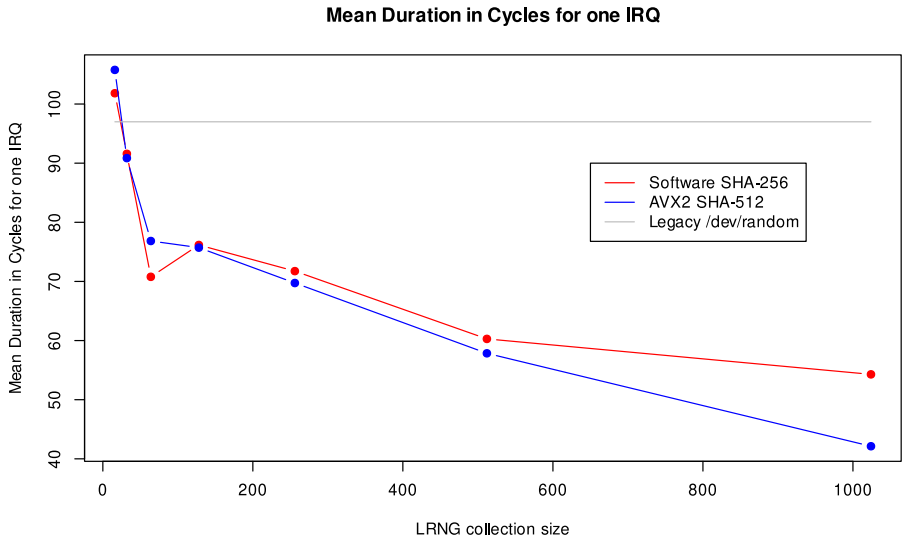
```
################################################################################
# Cryptographic backends configuration
################################################################################

# Option for: ESDM_DRNG_HASH_DRBG
option('drng_hash_drbg', type: 'feature', value: 'enabled',
       description:'''SP800-90A Hash Deterministic Random Number Generator.

This configuration enables an SP800-90A Hash DRBG with SHA-512 core
without prediction resistance.
''')

option('drng_chacha20', type: 'feature', value: 'disabled',
       description: 'ChaCha20-based Deterministic Random Number Generator.')

option('hash_sha512', type: 'feature', value: 'enabled',
       description: 'Enable SHA2-512 conditioning hash')

option('hash_sha3_512', type: 'feature', value: 'disabled',
       description: 'Enable SHA3-512 conditioning hash')
```

External Entropy Sources

- Use without additional conditioning – fast source
  - Jitter RNG
  - Kernel RNG
  - CPU (e.g. Intel RDSEED, POWER DARN, RISC-V CSRRW, IBM Z PRNO)
    - Proper oversampling as defined by specification
  - Data immediately available when ESDM requests it
- Additional conditioning – slow source
  - RNGDs
  - Arbitrary writers to /dev/random
  - All received data added to “auxiliary pool“ with hash update operation
  - Data “trickles in” over time
- Every entropy source has individual entropy estimate
  - Taken at face value – each ES requires its own entropy assessment



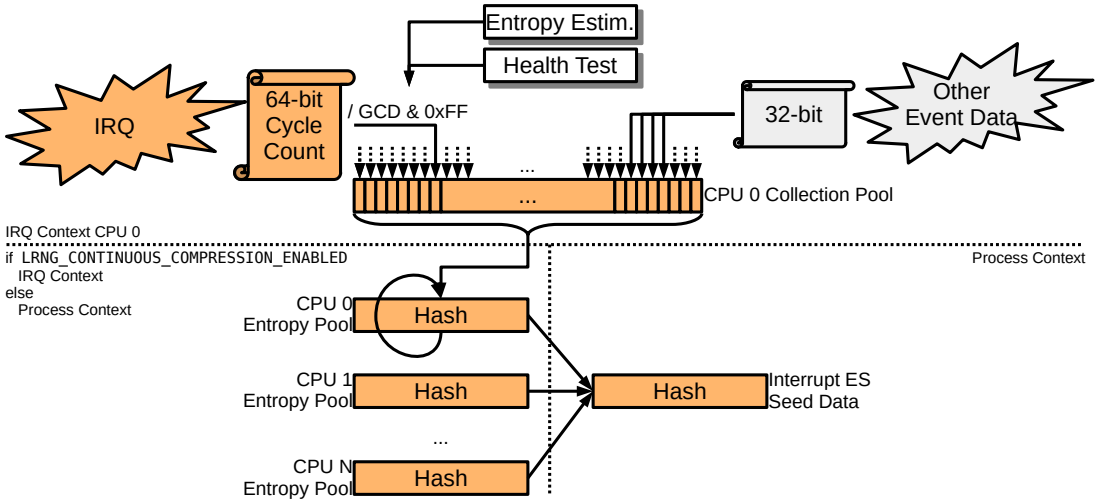
Internal ES: Interrupts

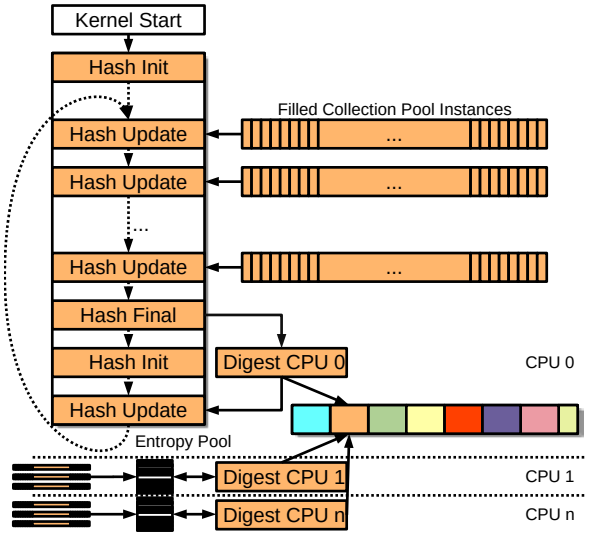


- Interrupt timing
  - All interrupts are treated as one entropy source
- Data collection executed in IRQ context
- Data compression executed partially in IRQ and process context
- Data compression is a hash update operation
- High performance: up to twice as fast as legacy /dev/random in IRQ context with LRNG_CONTINUOUS_COMPRESSION enabled
  - Even faster without continuous compression
- Kernel RNG lost acces to its primary IRQ ES→ ESDM reseeds kernel RNG periodically

Internal ES: IRQ Data Processing

- 8 LSB of time stamp divided by GCD concatenated into per-CPU collection pool
  - Entropy estimate
  - Health test
- 32 bits of other event data concatenated into per-CPU collection pool
- When array full → conditioned into per-CPU entropy pool
  - When entropy is required → conditioning of all entropy pools into one message digest
  - Addition of all per-CPU entropy estimates



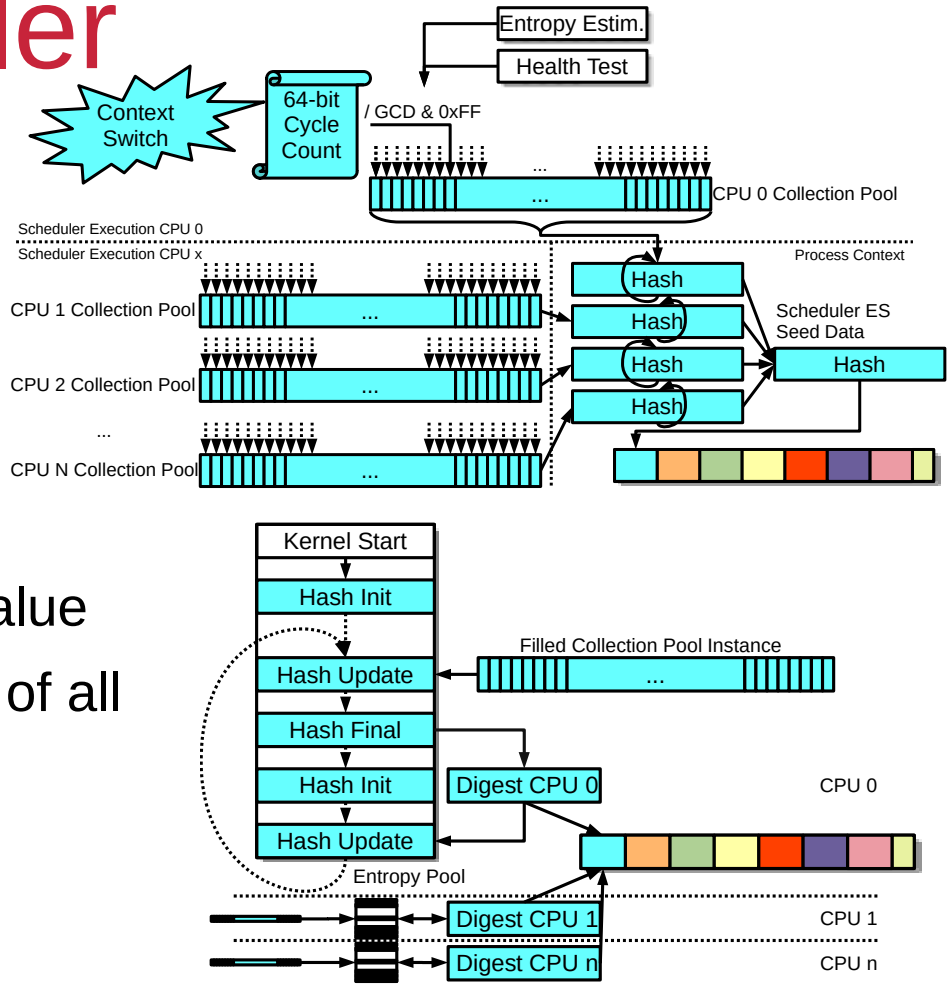


Internal ES: Scheduler Events

- Scheduler-based context switch timing
  - All context switches are treated as one entropy source
- Data collection executed in scheduler context
  - Collection: adding data into collection array → high-performance (couple of cycles)
- Data compression executed in process context during reseeding of DRNG
- Data compression is a hash operation
- Requires small kernel patch + kernel module
  - Code present in ESDM source tree

Internal ES: Scheduler Data Processing

- 8 LSB of time stamp divided by GCD concatenated into per-CPU collection pool
  - Entropy estimate
  - Health test
- When array full → overwriting of oldest value
- When entropy is required → conditioning of all entropy pools into one message digest
  - Addition of all per-CPU entropy estimates



Internal ES Testing Interfaces

- Testing code is compile time option
- Access via DebugFS
- Testing supports data collection at boot time and runtime:
  - Raw unprocessed entropy time stamps for Scheduler ES
  - Performance data for LRNG's Scheduler handler
- Full SP800-90B assessment documentation yet pending
- Raw entropy collection and analysis tools provided

Test System	Entropy of 1,000,000 Traces	Sufficient Entropy
ARMv7 rev 5	1.9344	Y
ARMv7 rev 5 (Freescale i.MX53)[22]	7.07088	Y
ARMv7 rev 5 (Freescale i.MX6 Ultralite)[23]	6.638399	Y
ARM 64 bit AppliedMicro X-Gene Mustang Board	5.599128	Y
Intel Atom Z530 – using GUI	3.38584	Y
Intel i7 7500U Skylake - 64-bit KVM environment	3.452064	Y
Intel i7 8565U Whiskey Lake – 64-bit KVM environment	7.400136	Y
Intel i7 8565U Whiskey Lake – 32-bit KVM environment	7.405704	Y
Intel i7 8565U Whiskey Lake	6.871	Y
Intel Xeon Gold 6234	4.434168	Y
IBM POWER 8 LE 8286-42A	6.830712	Y
IBM POWER 7 BE 8202-E4C	4.233912	Y
IBM System Z z13 (machine 2964)	4.366368	Y
IBM System Z z15 (machine 8561)	5.691832	Y
MIPS Atheros AR7241 rev 1[24]	7.157064	Y
MIPS Lantiq 34Kc V5.6[25]	7.032740	Y
Qualcomm IPQ4019 ARMv7[26]	6.638405	Y
SiFive HiFive Unmatched RISC-V U74	2.387470	Y

Internal ES Health Test

- Health test compile-time configurable
- Power-Up self tests
  - All cryptographic mechanisms
  - Time stamp management
- APT / RCT
- Time-Stamp Pattern detection: 1st/2nd/3rd discrete derivative of time ≠ 0
- Blocking interface: Wait until APT power-up testing complete
- Provides SP800-90B compliance of internal ES

General Testing

- Automated regression test suite covering the different options of ESDM
  - Meson test harness
- Applied testing frameworks
  - ASAN: address, thread, undefined
  - Valgrind memory leak detector
  - clang-scan
- Performance tests of DRNG
- Interface validation testing
- Compile test testing all options

```
# meson test -C build/
ninja: Entering directory `/home/sm/tmp/esdm-0.3.0/build'
ninja: no work to do.
 1/52 SHA256                                                 OK              0.01s
 2/52 SHA512                                                 OK              0.01s
 3/52 HMAC SHA256                                            OK              0.01s
 4/52 HMAC SHA512                                            OK              0.01s
 5/52 Hash DRBG SHA512                                       OK              0.00s
 6/52 IOCTL RNDADDENTROPY /dev/random                        OK              3.01s
 7/52 IOCTL RNDADDENTROPY /dev/urandom                       OK              3.01s
 8/52 IOCTL RNDCLEARPOOL /dev/random                         OK              3.11s
 9/52 IOCTL RNDCLEARPOOL /dev/urandom                        OK              3.11s
10/52 IOCTL RNDGETENTCNT /dev/random                         OK              3.01s
11/52 IOCTL RNDGETENTCNT /dev/urandom                        OK              3.01s
12/52 IOCTL RNDRESEEDCRNG /dev/random                        OK              3.02s
13/52 IOCTL RNDRESEEDCRNG /dev/urandom                       OK              3.01s
14/52 IOCTL RNDADDTOENTCNT /dev/random                       OK              3.01s
15/52 IOCTL RNDADDTOENTCNT /dev/urandom                      OK              3.01s
16/52 IOCTL get info /dev/random                             OK              3.01s
17/52 IOCTL get info /dev/urandom                            OK              3.01s
18/52 Poll writer FD /dev/random                             OK              5.02s
19/52 Poll writer FD /dev/urandom                            OK              5.02s
20/52 Read /dev/random                                       OK              3.18s
21/52 Read /dev/urandom                                      OK              3.23s
22/52 Write /dev/random                                      OK              3.45s
23/52 Write /dev/urandom                                     OK              3.45s
24/52 Read /dev/random - w/o ESDM server                     OK              3.55s
25/52 Read /dev/urandom - w/o ESDM server                    OK              3.56s
26/52 Write /dev/random - w/o ESDM server                    OK              3.56s
27/52 Write /dev/urandom - w/o ESDM server                   OK              3.56s
28/52 ES CPU                                                 OK              0.03s
29/52 ES Kernel                                              OK              0.03s
30/52 ES Scheduler                                           OK              0.02s
31/52 ESDM API call esdm_status                              OK              0.07s
32/52 ESDM API call esdm_version                             OK              0.09s
33/52 ESDM API call esdm_get_random_bytes_full               OK              0.20s
34/52 ESDM API call esdm_get_random_bytes                    OK              0.24s
35/52 ESDM API call esdm_get_random_bytes_min                OK              0.27s
36/52 ES Jitter RNG                                          OK              1.89s
37/52 ESDM DRNG manager max w/o reseed - 1 DRNG              OK              1.03s
38/52 ESDM DRNG manager max w/o reseed - 2 DRNG              OK              1.03s
39/52 ESDM seed entropy - all ES, no FIPS                    OK              1.07s
40/52 ESDM seed entropy - all ES, FIPS                       OK              1.09s
41/52 ESDM seed entropy - JENT ES, no FIPS                   OK              1.06s
42/52 ESDM seed entropy - JENT ES, FIPS                      OK              1.09s
43/52 ESDM seed entropy - CPU ES, no FIPS                    OK              1.07s
44/52 ESDM seed entropy - CPU ES, FIPS                       OK              1.09s
45/52 ESDM seed entropy - KRNG ES, no FIPS                   OK              1.06s
46/52 ESDM seed entropy - Sched ES, no FIPS                  SKIP            0.13s
 exit status 77
47/52 ESDM seed entropy - Sched ES, FIPS                     SKIP            0.12s
 exit status 77
48/52 System call getrandom                                  OK              0.55s
49/52 RPC call get_random_bytes_full_test                    OK              0.73s
50/52 RPC call get_random_bytes_min_test                     OK              0.80s
51/52 RPC call get_random_bytes_test                         OK              0.71s
52/52 RPC call status_test                                   OK              0.54s


Ok:                 50
Expected Fail:      0
Fail:               0
Unexpected Pass:    0
Skipped:            2
Timeout:            0
```

ESDM - Resources

- Code / Tests / Documentation: https://github.com/smuellerDD/esdm
- Testing conducted with
  - FIPS mode
  - Without FIPS mode
  - With SELinux enabled
  - Without SELinux

```
ESDM library version: 0.5.0
DRNG name: builtin SP800-90A Hash DRBG
ESDM security strength in bits: 256
Number of DRNG instances: 8
Standards compliance: NTG.1
ESDM minimally seeded: true
ESDM fully seeded: true
ESDM entropy level: 520
Entropy Source 0 properties:
 Name: Interrupt
 disabled - missing kernel support
Entropy Source 1 properties:
 Name: Scheduler
 disabled - missing kernel support
Entropy Source 2 properties:
 Name: JitterRNG
 Available entropy: 256
 Library version: 3040000
Entropy Source 3 properties:
 Name: CPU
 Hash for compressing data: N/A
 Available entropy: 8
 Data multiplier: 1
Entropy Source 4 properties:
 Name: KernelRNG
 Available entropy: 128
 Entropy Rate per 256 data bits: 128
Entropy Source 5 properties:
 Name: LinuxHWRand
 Available entropy: 128
 Entropy Rate per 256 data bits: 128
Entropy Source 6 properties:
 Name: Auxiliary
 Hash for operating entropy pool: builtin SHA-512
 Available entropy: 0
```

```
$ cat /proc/sys/kernel/random/esdm_type
ESDM library version: 0.5.0
DRNG name: builtin SP800-90A Hash DRBG
ESDM security strength in bits: 256
Number of DRNG instances: 19
Standards compliance: SP800-90C NTG.1
ESDM minimally seeded: true
ESDM fully seeded: true
ESDM entropy level: 1352
Entropy Source 0 properties:
 Name: Interrupt
 Hash for operating entropy pool: sha512
 Available entropy: 960
 per-CPU interrupt collection size: 1024
 Standards compliance: SP800-90B
 High-resolution timer: true
 Continuous compression: true
Entropy Source 1 properties:
 Name: Scheduler
 Hash for operating entropy pool: sha512
 Available entropy: 0
 per-CPU scheduler event collection size: 1024
 Standards compliance: SP800-90B
 High-resolution timer: true
Entropy Source 2 properties:
 Name: JitterRNG
 Available entropy: 256
 Library version: 3040000
Entropy Source 3 properties:
 Name: CPU
 Hash for compressing data: N/A
 Available entropy: 8
 Data multiplier: 1
Entropy Source 4 properties:
 Name: KernelRNG
 Available entropy: 0
 Entropy Rate per 256 data bits: 0
Entropy Source 5 properties:
 Name: LinuxHWRand
 Available entropy: 128
 Entropy Rate per 256 data bits: 128
Entropy Source 6 properties:
 Name: Auxiliary
 Hash for operating entropy pool: builtin SHA-512
 Available entropy: 0
```