Building a Modern TRNG:
An Entropy Source Interface for RISC-V

Markku-Juhani O. Saarinen – PQShield Ltd., UK
mjos@pqshield.com

G. Richard Newell – Microchip Technology Inc., USA
richard.newell@microchip.com

Ben Marshall – University of Bristol, UK
ben.marshall@bristol.ac.uk

ASHES 2020 – 13 Nov 2020
Virtual Workshop / ACM CCS 2020

Talk Outline

I. **Intro: The RISC-V Crypto TG and Random ISE**

II. PollEntropy: The Entropy Source Interface

III. Noise Sources & Implementation Considerations

(Due to a presentation time constraint, I will be skimming/skipping over some slides.)

RISC-V: Growing, developing steadily

RISC-V is a *free* and *open* Instruction Set Architecture (ISA), whose development is organized by RISC-V International (`https://riscv.org/`). Industrial and academic support is very broad[1]; Silicon and IP cores are available from multiple vendors, etc.

[1] With the notable exception of Intel® and ARM®, owners of proprietary ISAs !

RV Crypto TG: Status in November 2020

- Anyone can create custom (“X”) instructions for RISC-V; but for interoperability (stock compilers, Linux kernel etc) we need standard (“Z”) extensions too.
- The RISC-V Cryptographic Task Group (“**Crypto TG**”) has been tasked with studying and proposing **Cryptographic Instruction Set Extensions** (ISEs).
- Crypto TG has been running for a few years, chaired by G. Richard Newell.
- A “TRNG” was one of the main requirements, in addition to common cryptographic operations that benefit from ISE support, such as AES.
- Late last year it was decided to add “scalar” (non-vector) crypto as well.
- The editorship was passed to Ben Marshall (U. Bristol). Many changes in 2020.
- **Many of the set goals of the Crypto TG have been met**, and the spec is starting to look complete. There are strong dependencies to “bitmanip” (Zb) and vector (RVV) extensions but we hope to start official review soon.

Draft specification is out

RISC-V Cryptographic Extension Proposals
Volume I: Scalar & Entropy Source Instructions

Editor: Ben Marshall
ben.marshall@bristol.ac.uk

Version 0.7.2 (November 12, 2020)

https://github.com/riscv/riscv-crypto/
git:master @ 9556a166927589af7ace76322ce78bd7a1df4213

Contributors to all versions of the spec in alphabetical order (please contact editors to suggest corrections):
Alexander Zeh, Andy Glew, Barry Spinney, Ben Marshall, Daniel Page, Derek Atkins, Ken Dockser, Markku-Juhani O. Saarinen, Nathan Menhorn, Richard Newell, Claire Wolf

Section 4, *"Entropy Source Extension"* and Appendix B, *"Entropy Source: Rationale and Discussion"* of the draft spec contain much of the same material as this paper.

Design goal: To be able to seed any DRBG

"Random Number Generation is too Important to be Left to Chance."

– Robert R. (Bob) Coveyou (1915 – 1996)

- A True Random Number Generator **(TRNG**) is required by virtually every cryptographic application for the *generation of secrets* such as keys and nonces.
- A **TRNG** is used to seed a secure Pseudo Random Number Generator (**PRNG**), also called a Deterministic Random Bit Generator (**DRBG**) by FIPS / NIST.
- DRBGs are built from a cryptographic algorithm such as an **AES** (a block cipher) or **SHA-2/3** (a hash algorithm). They are designed in a way that prevents their internal state ("seed") from being determined from random output.

Is FIPS Certification Required?

We **do not** require entropy source implementations to be FIPS/CC validated, but we expect that they are compatible and do not create risks to users.

- A key requirement was that the TRNG can be used in a way that complies with FIPS 140-3 (HSMs, U.S. Government systems) and AIS 20 / 31 which is widely used in Common Criteria Evaluations (mainly Smart Cards, Secure Elements).
- FIPS 140-3 was finally ratified in last year. Validations are just starting!
- Entropy Estimation and Compliance with SP 800-90B will become mandatory in November 2020 for FIPS 140. Before this RNG evaluations were different.
- Read *"Recommendation for the Entropy Sources Used for Random Bit Generation"* (NIST SP 800-90B, 2018) about entropy estimation. Suggest self-evaluation: `https://github.com/usnistgov/SP800-90B_EntropyAssessment`

TRNG ISA Targets

In RISC-V the same instruction set can be used on a wide range of application platforms. We identify two broad targets / scenarios for the TRNG ISA:

"Secure Element / MCU"

In smart cards and other secure elements. Stringent security, standard compliance requirements.

Configuration: Single-hart RV32. Embedded-style CPUs may be permanently in machine mode.

API: With a cryptographic library, runtime, or direct in an application.

"General Purpose Linux"

With Linux and BSD-style kernels. TRNG is a shared resource.

Configuration: RV64, multi-hart, supporting privilege separation and memory management.

API: OS has sole access to the hardware entropy source and "pool." DRBG for `/dev/[u]random` output.

Talk Outline

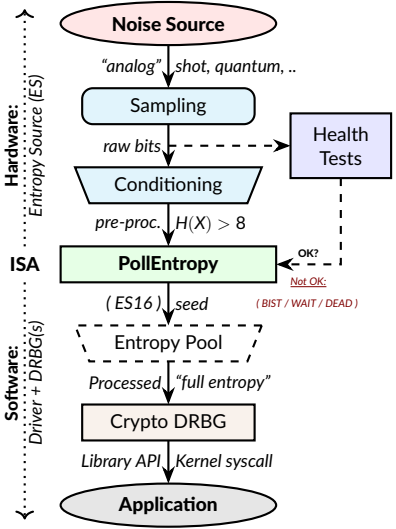
I Intro: The RISC-V Crypto TG and Random ISE

II **PollEntropy: The Entropy Source Interface**

III Noise Sources & Implementation Considerations

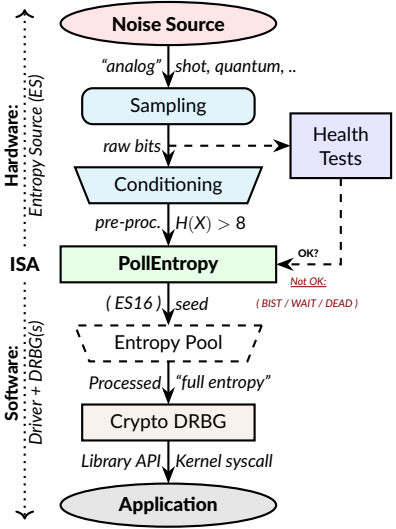
(Due to a presentation time constraint, I will be skimming/skipping over some slides.)

Entropy Source and The TRNG



- The current specification only covers the "Entropy Source" interface and leaves the DRBG part open.
- We already have AES and hash instructions; these can be used to generate fast streams of random bits, once initialized with a sufficient amount of entropy.
- If a DRBG is hard-wired into the noise source, it is difficult to audit. It is virtually impossible to know if such a source is secure and how secure it is.
- Hardware DRBGs may also have security limits; for example the Intel TRNG (RDRAND) should not be used directly if needed security level is >128 bits (e.g. 256-bit AES or Post-Quantum Cryptography).

TRNG Model



1. **A Noise Source** generates private, unpredictable signals from stable and well-understood physical random events.
2. **Sampling** digitizes the noise signal into a raw stream of bits. This raw data also needs to be protected by the design.
3. **Continuous health tests** ensure that the noise source and its environment meet their operational parameters.
4. **Non-cryptographic conditioners** remove much of the bias and correlation in input noise: Output entropy >4 bits/byte.
5. **Cryptographic conditioners** produce nearly full entropy output, completely indistinguishable from ideal random.
6. **DRBG** takes in $\geq$ 256 bits of seed entropy as keying material and uses a "one way" cryptographic process to rapidly generate bits on demand (without revealing the seed or the state).

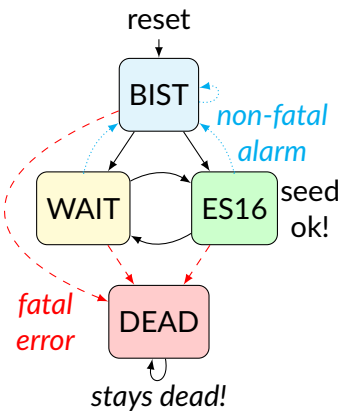
There is only one instruction in the baseline entropy source proposal:

```
pollentropy rd            // Poll randomness from entropy source.
(csrrs rd, mentropy, x0) // Encoding with m-mode "mentropy" CSR.
```

Bits	Name	Description
`rd[63:32]`	*Set to 0*	Bit 31 is zero-extended on RV64.
`rd[31:30]`	`OPST`	Status: 00 `BIST`, 01 `ES16`, 10 `WAIT`, 11 `DEAD`.
`rd[29:24]`	*reserved*	For future use by the RISC-V specification.
`rd[23:16]`	*custom*	Reserved for custom and experimental use.
`rd[15: 0]`	`SEED`	16 bits of randomness when `OPST=ES16`.

Generally, RISC-V specifies the ISA only. We specify minimal additional requirements so that portable, vendor-independent middleware and kernel components can be created.

High Status Bits



OPST	Status name and description
0 0	BIST indicates Built-In Self-Test (BIST) or a non-fatal (non-actionable) alarm.
0 1	ES16 indicates success. the low bits rd[15:0] will have 16 bits of randomness (which must have at least 8 bits of "true entropy" regardless of implementation).
1 0	WAIT means that a sufficient amount of entropy is not yet available. Not an error.
1 1	DEAD is an unrecoverable self-test error.

Polling and Post-Processing

Non-blocking: The instruction returns immediately, either with two status bits `rd[31:30]=OPST` set to `ES16` (01), indicating successful polling, or with **no** entropy.

Linux Kernel polls opportunistically but on-demand spinloop polling is also possible.

No interrupts needed: The state is usually either `WAIT` or `ES16`. There are no mandatory interrupts. However, it is required that the `WFI` (wait for interrupt) instruction is available when `pollentropy` is implemented (it can be a `NOP`).

For portable drivers in embedded systems, a `WAIT` or `BIST` from `pollentropy` should be followed by a `WFI` before another `pollentropy` instruction is issued.

Post-processing: The raw "`ES16`" output must be cryptographically conditioned (e.g. hashed) before it is used for keying etc. We recommend it being processed in at least $16 \times 16 = 256$-bit blocks (resulting in 128 bits of "full entropy" per block).

On Entropy Source Requirements

A PollEntropy implementation can always output fully conditioned, perfectly distributed random bits. However 2:1 cryptographic post-processing is still needed!

(One can also use more than twice the number of seed bits relative to key size.)

Entropy sources should meet at least one of three requirements:

1. **Virtual Entropy Sources** are fully seeded DRBGs with $\geq$ 256-bit security.
2. **SP 800-90B I.I.D. Entropy Sources** meet specific FIPS 140-3 requirements.
3. **PTG.2 Class RNGs** meet specific Common Criteria (AIS-31) requirements.

Virtual entropy sources can be used for enclaves, simulators, and virtual machines.

If a DRBG is used as a source, it must meet “Category 5” post-quantum security; Any implementation of pollentropy that limits the security strength shall not reduce it to less than AES-256 equivalent (“computationally bounded full entropy”.)

FIPS 140-3 / SP 800-90B I.I.D. Track

For FIPS 140-3, vendors should design their entropy sources and hardware conditioning components so that they can be submitted to the “I.I.D. track”.

§E1 **Entropy Requirement.** Each 16-bit output sample (`SEED`) must have more than 8 bits of randomness *(Technical definition is that a specific type of 2:1 post-processing step – a vetted conditioner – always achieves “full entropy”.)*

§E2 **I.I.D. Requirement.** The output must be *Independent and Identically Distributed* (IID), meaning that the output distribution does not change over time and that output words do not convey information about each other.

Both requirements must be satisfied (§E1 may appear looser than §E2). Full FIPS 140-3 validation imposes many additional requirements.

Simplified! These requirements have technical definitions in the SP 800-90B context.

CC / AIS 31 PTG.2 Class RNG (not in paper)

For alternative BSI AIS-31 / Common Criteria certification (or self-certification) vendors should target **PTG.2** Class RNG requirements. Entropy sources (SEED bits) are viewed as "internal random numbers" in AIS 31. A *stochastic model* is needed.

Note that PTG.2 does not preclude other certification levels – especially PTG.3 when combined with appropriate post-processing and DRBG on the software side.

§P1 ... simply present a terminology mapping to PTG.2.1 – PTG.2.7 requirements.

§P7 **[PTG.2.7]** Average Shannon entropy of internal random bits exceeds 0.997.

In PTG.2 validation the "loose I.I.D." SP 90B requirement of §E2 is not stated. But we recommend that all 16 SEED bits are considered for §P7 (not required by §E1).

PTG.2 has different methodology from SP 90B; no IID, need stochastic model, etc..

Security Controls (Tests)

Statistical tests and other security controls are mandated by standards, including NIST SP 800-90B. The choice of appropriate tests depends on the certification target, system architecture, the threat model, entropy source type, and other factors. They can be hardware or software, typically both.

§T1 **On-demand testing.** A sequence of simple tests is invoked via resetting, rebooting, or powering-up the hardware (not an ISA signal). Entropy source will be in `BIST` state. Software can do KATs on crypto etc.

§T2 **Continuous checks.** If an error is detected in continuous tests or environmental sensors, the entropy source will enter a no-output state. A non-fatal alarm can be signaled with `BIST`, latched until polled at least once.

§T3 **Fatal error states.** Since the security of cryptographic operations depends on the entropy source, a system-wide “default deny” security policy approach is appropriate for most entropy source failures.

GetNoise Test Interface (not in paper)

SP 800-90B validation requires access to the raw, unconditioned noise source. Access does not need to be via the ISA, but RISC-V provides a CSR location for it.

```
getnoise rd              // Raw noise source test interface.
(csrrs rd, mnoise, x0)   // Encoding with m-mode "mnoise" CSR.
```

Crypto ISE defines semantics for a single bit, `mnoise[31]`, named `NOISE_TEST`.

When `NOISE_TEST` is enabled, the “real” PollEntropy interface is disabled – `BIST` – for security reasons. The GetNoise test interface is not used by the Kernel and is not portable; is is vendor-specific and optional to implement.

Otherwise the contents and behavior of `mnoise` must be interpreted in the context of `mvendorid`, `marchid`, and `mimpid` identifiers, so GetNoise is effectively “custom”.

Talk Outline

I Intro: The RISC-V Crypto TG and Random ISE

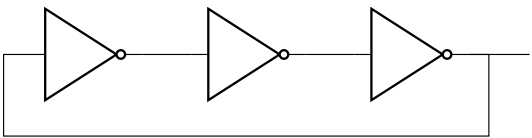
II PollEntropy: The Entropy Source Interface

III **Noise Sources & Implementation Considerations**

(Due to a presentation time constraint, I will be skimming/skipping over some slides.)

Most common: Ring Oscillators

- A *free running* **ring oscillator** has an odd number of inverters in a loop:



- Raw entropy is obtained by sampling a ring oscillator in relation to a reference clock. The oscillation“jitter” accumulates to random phase difference.
- Small: Few ASIC Standard Cell inverters / FPGA inverters (LUTs) in a chain.
- Randomness is derived from “unavoidable” Johnson-Nyquist (thermal) noise.
- Well-understood physical and stochastic models exist for entropy estimation.
- Digital post-processing to remove bias, condense entropy, and detect faults.

Noise Sources

- Standards require raw noise sources to be protected “to the greatest extent possible”. Such vendor-specific interfaces are delegated to a debug interface.
- In the paper we give some examples of common noise sources that can be implemented in the processor itself (using standard cells).
- There is a long precedent in validating free-running ring oscillators and related metastability-based designs. Intel, ARM, AMD use this general type of design in their CPUs and SoCs. Shot noise may be more resistant to temperature.
- Even post-quantum cryptography does **not** require “quantum” noise sources. Such “QRNGs” may be more tricky to implement and interface securely.

U.K. QRNG Guidance, March 2020

“The NCSC believes that classical RNGs will continue to meet our needs for government and military applications for the foreseeable future.”

Non-Cryptographic Conditioning

- It is necessary to verify that the noise source and sampler output matches with their stochastic models. We note that this is usually done in a laboratory setting to account for temperature and other environmental factors.
- To meet the entropy requirement (§E1) one typically needs to “condition” the raw noise in hardware. There are standard, light-weight methods for doing this.
- Reducing sample rate, XORing noise bits together, applying a von Neumann “coin flip” conditioner, or Blum’s Markov-model extractor help to increase the entropy of samples and to remove dependencies (§E2, IID).
- If well designed, non-cryptographic conditioners be evaluated in conjunction with a stochastic model of the noise source itself. They do not require computational hardness assumptions and are inherently “future proof.”

The Final Random: DRBGs

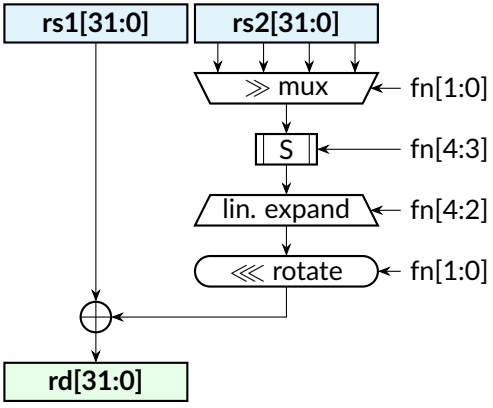
DRBG is a requirement: All random bits reaching end users and applications must come via a cryptographically secure DRBG. We recommend standard constructions from NIST SP 800-90A (Rev 1!): `CTR_DRBG`, `Hash_DRBG`, and `HMAC_DRBG`.

The RISC-V AES and SHA2/3 instruction set extensions should be used if available, since they offer additional security features such as timing attack resistance.

Linux random and other custom DRBGs. In addition to the SP 800-90A DRBGs, a Linux-style random pool construction based on ChaCha20 can be used, or an appropriate construction based on SHAKE256 .

Freedom to implement. These are just recommendations; programmers can adjust the usage of the CPU Entropy Source to meet future requirements.

Scalar 32/64-bit or Vector AES for Fast Output



Scalar AES has a very small hardware footprint.

```
aes32esi    rd, rs1, rs2, bs
aes32esmi   rd, rs1, rs2, bs
aes32dsi    rd, rs1, rs2, bs
aes32dsmi   rd, rs1, rs2, bs
```

- AES-256 based `CTR_DRBG` can produce random output from gathered entropy really fast, especially if implemented with RISC-V AES instructions.
- The scalar 32-bit AES has only one S-Box (<1000 GE for encrypt only) yet speeds up AES by 500%, is constant time.
- 64-bit and vector versions are even faster.

Conclusions

Real-World ISA Support for Random – Main Challenges:

- Accommodate SP 800-90B (FIPS 140-3) and PTG.2 (AIS 31, Common Criteria) .. yet allow a simple, common driver component and portable software!
- Allow implementation in embedded and high-performance systems alike.

RISC-V PollEntropy:

- M-mode polling entropy source interface, not a “random number generator”.
- Provides just seed entropy with “guaranteed properties” – small, future proof.
- Mandatory post-processing, DRBGs in software (AES/SHA/SM Extensions).

Thank You! Questions?