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

ABSTRACT

The currently proposed RISC-V True Random Number Generator (TRNG) architecture breaks with previous ISA TRNG practice by splitting the Entropy Source (ES) component away from cryptographic PRNGs into a separate interface, and in its use of polling. We describe the interface, its use in cryptography, and offer additional discussion, background, and rationale for various aspects of it. This design is informed by lessons learned from earlier mainstream ISAs, recently introduced SP 800-90B and FIPS 140-3 entropy audit requirements, AIS 31 and Common Criteria, current and emerging cryptographic needs such as post-quantum cryptography, and the goal of supporting a wide variety of RISC-V implementations and applications. Many of the architectural choices are a result of quantitative observations about random number generators in secure microcontrollers, the Linux kernel, and cryptographic libraries. We further compare the architecture to some contemporary random number generators and describe a minimalistic TRNG reference implementation that uses the Entropy Source together with RISC-V AES instructions.

KEYWORDS

Entropy Source, RISC-V, Random, TRNG, FIPS 140-3, SP 800-90B

1 INTRODUCTION

The security of cryptographic systems is based on secret bits and keys. To prevent guessing, these bits need to be random, so they come from True Random Number Generators (TRNGs).

As a fundamental security function, the generation of random numbers is governed by numerous standards and technical requirements. This work describes an architecture and approach that can be taken by RISC-V [53] implementors to address these challenges.

RISC-V (https://riscv.org/) is a popular open source Instruction Set Architecture (ISA) that anyone can freely use. The minimalistic base instruction sets RV32I and RV64I (for 32- and 64-bit architectures) are often amended with extensions that provide features such as floating point arithmetic or bit manipulation.

Note: The RISC-V specifications are created by committees and task groups within RISC-V International. The TRNG architecture discussed in this work grew out of the efforts by individual members of the Cryptographic Extensions Task Group [48] and represents their personal opinions only; not their respective employers or RISC-V International. The architecture or particular instructions discussed have not been ratified at the time of writing; it is the purpose of this work to support that ratification process.

1.1 Standards and Terminology

A driving design goal for our architecture was for it to be easy to implement, yet compatible with current versions of FIPS 140-3 [38] and NIST SP 800-90B [49], significantly updated standards that are only coming into use in 2020. Naturally, the architecture should also support other RNG frameworks such as German AIS 20 / 31 [12, 25, 26] which is widely used in Common Criteria evaluations.

These standards set many of the technical requirements for the design, and we use their terminology if possible.

1.1.1 Entropy Source (ES). Physical sources of true randomness are called Entropy Sources (ES) [49]. They are built by sampling and processing data from a noise source (Section 4.1). Since these are directly based on natural phenomena and are subject to environmental conditions (which may be adversarial), they require features and sensors that monitor the "health" and quality of those sources. See Section 3.2 for a discussion about such security controls.

1.1.2 Conditioning. Raw physical randomness (noise) sources are rarely statistically perfect and some generate very large amounts of bits, which need to be "debiased" and reduced to a smaller number of bits. This process is called conditioning. A secure hash function is an example of a cryptographic conditioner. It is important to note that even though hashing may make the output look more random, it does not increase its entropy content.

Non-cryptographic conditioners and extractors such as von Neumann's "debiased coin tossing" [52] are easier to implement efficiently but may reduce entropy content much more than hashes. However, they are not based on computational hardness assumptions and are therefore inherently more future proof. See Section 4.4 for a more detailed discussion.

1.1.3 Pseudorandom Number Generator (PRNG). Pseudorandom Number Generators (PRNGs) use deterministic mathematical formulas to create a large amount of random numbers from a smaller amount of "seed" randomness. PRNGs are divided into cryptographic and non-cryptographic ones.

Non-cryptographic PRNGs, such as the linear-congruential generators found in many programming libraries, may generate statistically satisfactory random numbers but must never be used for cryptographic keying. This is because they are not designed to resist *cryptanalysis*; it is usually possible to take some output and mathematically derive the "seed" or the internal state of the PRNG from it. This is a security problem since knowledge of the state allows the attacker to compute future or past outputs.

1.1.4 Deterministic Random Bit Generator (DRBG). Cryptographic PRNGs are also known as Deterministic Random Bit Generators (DRBGs), a term used by SP 800-90A [7]. A strong cryptographic algorithm such as AES [34] or SHA-2/3 [35, 36] is used to produce

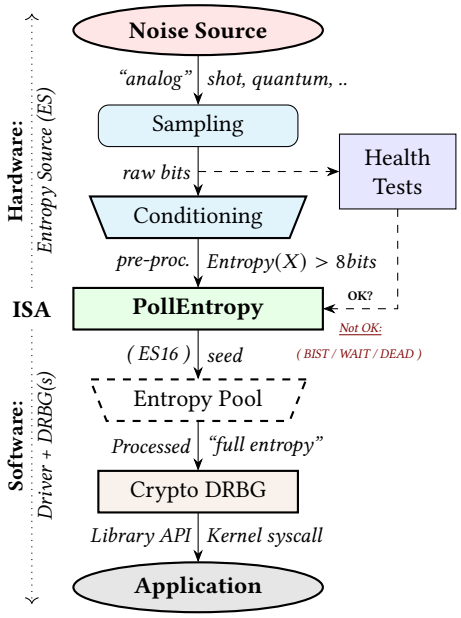

Figure 1: PollEntropy is an Entropy Source (ES) only, not a stateful random number generator. As a result, it can support arbitrary security levels; cryptographic (AES, SHA2/3) instructions can be used to construct high-speed DRBGs.

random bits from a seed. The secret seed material is acting as a cryptographic key; determining the seed from the DRBG output is as hard as breaking AES or a strong hash function. This also illustrates that the seed/key needs to be long enough and come from a trusted Entropy Source. The DRBG can be frequently refreshed or "reseeded."

1.2 RISC-V Target Considerations

One of the key features of RISC-V is that essentially the same instruction set can be used on a wide range of application platforms. We identify two broad targets for the TRNG ISA: Secure Microcontrollers and Linux Profile systems.

1.2.1 Secure Microcontrollers. Some RISC-V cores are being designed specifically for smart cards and other secure elements, where a hardware-based random number generator is the only viable source of keying material. These "security chip" targets have stringent engineering requirements in relation to RNG quality and certification, physical security, energy efficiency, and unit cost.

Configuration: Embedded-style CPUs may be permanently in machine mode [54], and therefore run only trusted firmware. They are likely to have an RV32 single-hart configuration.

API Interface: These targets are expected to interface and implement the TNRG subsystem via a cryptographic library or security-oriented runtime firmware.

1.2.2 General-purpose Linux. We expect Linux and BSD-style operating system kernels to dominate the mobile, desktop, and server application areas. Some of these targets also need to be hardened against invasive physical attacks. Energy efficiency is a concern for mobile devices. Additional entropy sources may be available.

Configuration: These higher-performance CPUs support privilege separation and memory management. They are more likely to be in RV64 multi-hart or even multiprocessor configuration.

API Interface: Generally, the TRNG interface will be via the operating system kernel which runs in privileged machine mode and has sole access to the hardware entropy source. The kernel processes input entropy via a multi-source random pool and makes it available to users via `/dev/[u]random` and `getrandom(2)`.

Table 1: PollEntropy output register contents

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

2 THE ENTROPY SOURCE INTERFACE

The proposed RISC-V TRNG ISA is primarily an Entropy Source (ES) interface. A valid implementation should satisfy properties that allow it to be used to *seed* standard and nonstandard cryptographic DRBGs of virtually any state size and security level.

The purpose of the baseline specification is to guarantee that a simple, device-independent driver component (e.g. in Linux kernel, embedded firmware, or a cryptographic library) can use the instruction to generate truly random bits. If an ES cannot be interfaced or conditioned in a way that meets the baseline criteria, it can be interfaced via regular IO interfaces and custom drivers instead. The delineation of various components is illustrated in Figure 1.

2.1 PollEntropy

The ISA-level interface consists of a single instruction, `PollEntropy` that returns a 32/64-bit value in a CPU register [48]. It is invoked in **Machine Mode** (which may be the only mode) as follows:

```
// Get randomness. imm=0 for baseline operation
pollentropy rd, imm
```

```
// C calling convention
int PollEntropy(0);
```

The instruction is **non-blocking** and returns immediately, either with two status bits `rd[31:30]=OPST` set to `ES16` (01), indicating successful polling, or with **no** entropy and one of three polling failure statuses `BIST` (00), `WAIT` (10), or `DEAD` (11), discussed below. See Tables 1 and 2 for further information.

The sixteen bits of randomness in `rd[15:0]=seed` polled with `ES16` status **must be cryptographically conditioned** before they can be used as (up to 8 bits of) keying material. See Section 2.2.

For the purposes of interoperable, baseline functionality, `imm = 0`. Nonzero values of `imm` are reserved for future use; the behavior described here may not apply when `imm` $\neq 0$.

Table 2: The four OPST states of the entropy source.

OPST	Status name and description
0 0	BIST indicates that Built-In Self-Test "on-demand" (BIST) statistical testing is being performed. In typical implementations, BIST will last only a few milliseconds, up to a few hundred. If the system returns temporarily to BIST from any other state, this signals a non-fatal (usually non-actionable) self-test alarm.
0 1	ES16 indicates success; the low bits rd[15:0] will have 16 bits of randomness which must be guaranteed to have at least 8 bits true entropy regardless of implementation. For example, 0x4000ABCD is a valid ES16 status output on RV32, with 0xABCD being the seed value.
1 0	WAIT means that a sufficient amount of entropy is not yet available. This is not an error condition and may (in fact) be more frequent than ES16, since true entropy sources may not have very high bandwidth. If polling in a loop, we suggest calling WFI (wait for interrupt) before the next poll.
1 1	DEAD is an unrecoverable self-test error. This may indicate a hardware fault, a security issue, or (extremely rarely) a type-1 statistical false positive in the continuous testing procedures. Implementations do not need to implement DEAD as it may not require an end-user notification; an immediate lock-down may be a more appropriate response in dedicated security devices.

2.1.1 Polling. Figure 2 illustrates operational state (OPST) transitions. The state is usually either WAIT or ES16. The baseline specification requires no additional signaling or interrupts to be supported by the hardware. However, the polling mechanism should be implemented in a way that allows even generic non-interrupt drivers to benefit from interrupts.

We specifically recommend against busy-loop polling on this instruction as it may have relatively low bandwidth. Even though no specific interrupt sequence is specified, it is required that the WFI (wait for interrupt) instruction is available and does not trap on systems that implement PollEntropy. The RISC-V ISA allows WFI to be implemented as a NOP. As a minimum requirement for portable drivers, a WAIT or BIST from PollEntropy should be followed by a WFI before another PollEntropy instruction is issued. There is no need to poll after a DEAD state.

To guarantee that no sensitive data is read twice and that different callers don't get correlated output, it is suggested that hardware implements "wipe-on-read" on the randomness pathway during each read (successful poll). For the same reasons, only complete and fully processed randomness words shall be made available via PollEntropy (no half-conditioned buffers or even full buffers in WAIT state – even if they are to be ignored by compliant callers).

2.1.2 Rationale and Discussion. An entropy source does not require a high-bandwidth interface; a single DRBG source initialization only requires 512 bits (256 bits of entropy) and the DRBG state can be shared by any number of callers. Once initiated, a DRBG requires new randomness only for the purposes of forward security.

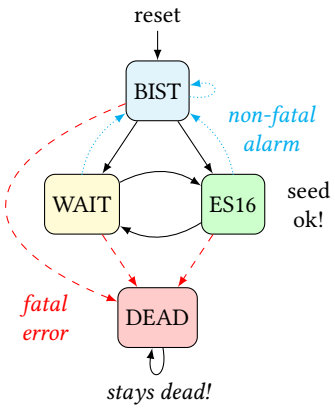

Figure 2: Normally the operational state alternates between WAIT (no data) and ES16, which means that 16 bits of randomness (seed) has been polled. BIST (Built-in Self-Test) only occurs after reset or to signal a non-fatal self-test alarm (if reached after WAIT or ES16). DEAD is an unrecoverable error state.

Without a polling-style mechanism the entropy source could hang for thousands of cycles under some circumstances. The WFI mechanism (at least potentially) allows energy-saving sleep on MCUs and context switching on a higher-end CPUs.

The reason for the particular OPST two-bit mechanism is to provide redundancy. The "fault" bit combinations 11 (and 00) are more likely for electrical reasons if feature discovery fails and the entropy source is actually not available (this has happened to AMD [46]).

The 16-bit bandwidth was a compromise motivated by the desire to provide redundancy in the return value, some protection against potential Power/EM leakage (further alleviated by the 2:1 cryptographic conditioning discussed in Section 2.2.1), and the desire to have all of the bits "in the same place" on both RV32 and RV64 architectures for programming convenience.

2.2 Entropy Source Requirements

Output SEED from PollEntropy is not necessarily fully conditioned randomness due to hardware limitations of smaller, low-powered implementations. However minimum requirements are defined. Therefore a caller should not use the output directly but poll twice the amount of required entropy, cryptographically condition (hash) it, and use that to seed a cryptographic DRBG.

2.2.1 Entropy Requirement. Each 16-bit output sample (SEED) must have more than 8 bits of independent, unpredictable entropy. This minimum requirement is satisfied if (in a NIST SP 800-90B [49] assessment) 128 bits of output entropy can be obtained from each 256-bit (16 × 16-bit) PollEntropy output sequence via a vetted cryptographic conditioning algorithm (see Section 3.1.5.1.2 in [49]).

Driver developers may make this conservative assumption but are not prohibited from using more than twice the number of seed bits relative to the desired resulting entropy.

Rationale: Rather than attempting to mathematically define the properties that the entropy source output must satisfy, we define that it should be good enough to pass an evaluation and certification when conditioned cryptographically ("perfectly") in ratio 2:1. This is our "safety margin" for non-cryptographic conditioners.

Note that the min-entropy assessment methodology in SP 800-90B [49] also has a safety margin in its confidence intervals, and therefore there must be consistently *more than* 8 bits of entropy per 16-bit word. In practice, we recommend the distribution to be significantly closer to uniform to satisfy possible additional use cases and AIS 20 / 31 [26] requirements (if those can't be met with a software conditioner).

Note that the usage of a vetted conditioner (such as SHA-2/3) was specified for technical reasons related to SP 800-90B itself; non-vetted conditioners may offer similar security.

The 128-bit output block size was selected because that is the output size of the CBC-MAC conditioner specified in [49] and also the smallest key size we expect to see in applications.

2.2.2 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. This requirement is satisfied if the construction of the physical source and sampling mechanism suggests nothing against the IID assumption and the IID tests in Section 5 of NIST SP 800-90B [49] are consistently passed.

Rationale: IID is an optional requirement in SP 800-90B [49] but it is needed to prevent information leakage between different entities that possibly share the same entropy source. It also significantly simplifies certification and vendor-independent driver development. The `PollEntropy` instruction itself can be later expanded to support non-IID sources (e.g. via a different immediate constant).

2.2.3 Secret State Size Requirement. A `PollEntropy` implementation can also output fully conditioned, perfectly distributed numbers. However, it is required that if a DRBG is used as a source, it must have an internal state with at least 256 bits of secret entropy (Example: a CTR_DRBG built from AES-128 is never sufficient). In general, any implementation of `PollEntropy` that limits the security strength shall not reduce it to less than 256 bits.

Rationale: DRBGs can be used to feed other DRBGs but of course that does not increase the absolute amount of entropy in the system. The entropy source must be able to support current and future security standards and applications.

The 256-bit level is quite arbitrary but maps to "Category 5" of Post-Quantum Cryptography standards (See section 4.A.5 "Security Strength Categories" in [37]) and TOP SECRET schemes in Suite B and the newer U.S. Government CNSA Suite [40].

3 MONITORING AND INFORMATION FLOWS

> *"The noise source state shall be protected from adversarial knowledge or influence to the greatest extent possible. The methods used for this shall be documented, including a description of the (conceptual) security boundary's role in protecting the noise source from adversarial observation or influence."*
>
> –Noise Source Requirements, NIST SP 800-90B [49].

Some of our earlier designs had many more states and possibly complex interaction mechanisms, which were simplified to the bare minimum that could still meet our requirements.

3.1 Oracles and Side Channels

Our approach is informed by the experience of designing and implementing cryptographic protocols. Some of the most devastating practical attacks against real-life cryptosystems have used inconsequential-looking additional information, such as padding error messages [6] or timing information [31]. In cryptography, such out-of-band information sources are called "oracles".

This also applies to the raw noise source. While most developers agree that access to the raw noise source would be *nice to have*, it is less clear why a generic driver would need it, or know what to do with it (the raw noise can literally be *anything*). Therefore raw source interface has been delegated to an optional vendor-specific debug interface outside the scope of the baseline specification.

The role of the RISC-V ISA implementation is to try to ensure that the hardware-software interface minimizes avenues for adversarial information flow; all status information that is unnecessary in normal operation should be eliminated. We specifically urge implementors against creating unnecessary information flows ("status oracles") via the custom bits or to allow the instruction to disable or affect the TNRG output in any significant way. All information flows and interaction mechanisms must be considered from an adversarial viewpoint and implemented only if they are truly necessary and their security impact can be fully understood.

For example, implementations of the polling interface may not be *constant time*. The polling interface can be modeled as a rejection sampler; such a timing oracle reveals information about source and the rejection criteria (distribution), but not the random output itself. Implementors must consider the significance of such flows.

3.2 Security Controls

The primary purpose of a cryptographic entropy source is to produce secret keying material. In almost all cases a hardware entropy source must implement appropriate *security controls* to guarantee unpredictability, prevent leakage, detect attacks, and to deny adversarial control over the entropy output or ts generation mechanism.

Many of the security controls built into the device are called "health checks". Health checks can take the form of integrity checks, start-up tests, and on-demand tests. These tests can be implemented in hardware or firmware; typically both. Several are mandated by standards such as NIST SP 800-90B [38]. The choice of appropriate health tests depends on the certification target, system architecture, the threat model, entropy source type, and other factors.

Health checks are not intended for hardware diagnostics but for detecting security issues – hence the default action should be aimed at damage control (prevent weak crypto keys from being generated). Additional "debug" mechanisms may be implemented if necessary, but then the device must be outside production use.

The statistical nature of some tests makes "type-1" false positives a possibility. Security architects will understand to use permanent or hard-to-recover "security-fuse" lockdowns only if the threshold of a test is such that the probability of false-positive is negligible over the entire device lifetime.

3.2.1 On-demand testing. A sequence of simple tests is invoked via resetting, rebooting, or powering-up the hardware (not an ISA signal). The implementation will simply return `BIST` during the initial start-up self-test period; in any case, the driver must wait for them to finish before starting cryptographic operations. Upon failure the entropy source will enter a no-output `DEAD` state.

Rationale: Interaction with hardware self-test mechanisms from the software side should be minimal; the term "on-demand" does not mean that the end-user or application program should be able to invoke them in the field (the term is a throwback to an age of discrete, non-autonomous crypto devices with human operators.)

3.2.2 Continuous checks. If an error is detected in continuous tests or environmental sensors, the entropy source will enter a no-output state. We define that a non-critical alarm is signaled if the entropy source returns to `BIST` state from live (`WAIT` or `ES16`) states. Such a `BIST` alarm should be latched until polled at least once. Critical failures will result in `DEAD` state immediately. A hardware-based continuous testing mechanism must not make statistical information externally available, and it must be zeroized periodically or upon demand via reset, power-up, or similar signal.

Rationale: Physical attacks can occur while the device is running. The design should avoid guiding such active attacks by revealing detailed status information. Upon detection of an attack the default action should be aimed at damage control – to prevent weak crypto keys from being generated.

See Section 4.3 for further discussion: There may be requirements for signaling of alarms; AIS 31 specifies "noise alarms" that can go off with non-negligible probability even if the device is functioning correctly; these can't be fatal but can be signaled with `BIST`.

The state of statistical runtime health checks (such as counters) is potentially correlated with some secret keying material, hence the zeroization requirement.

3.2.3 Fatal error states. Since the security of most cryptographic operations depends on the entropy source, a system-wide "default deny" security policy approach is appropriate for most entropy source failures. A hardware test failure should result in at least in `DEAD` state and possibly reset/halt. It's a show stopper: The entropy source (or its cryptographic client application) *must not* be allowed to run if its secure operation can't be guaranteed.

Rationale: These tests can complement other integrity and tamper resistance mechanisms (See Chapter 18 of [2] for examples).

Some hardware random generators are, by their physical construction, exposed to relatively non-adversarial environmental and manufacturing issues. However, even such "innocent" failure modes may indicate a *fault attack* [24] and therefore should be addressed as a system integrity failure rather than as a diagnostic issue.

4 IMPLEMENTATION STRATEGIES

As a general rule, RISC-V specifies the ISA only. We provide some additional requirements so that portable, vendor-independent middleware and kernel components can be created. The actual hardware implementation and certification is left to vendors and circuit designers; the discussion in this section is purely informational.

While we do not require entropy source implementations to be certified designs, we do expect that they behave in a compatible manner and do not create unnecessary security risks to users. Self-evaluation and testing following appropriate security standards is usually needed to achieve this. NIST has made its SP 800-90B[49] min-entropy estimation package freely available[1] and similar free tools are also available[2] for AIS 31 [26].

Entropy Flow. When considering implementation options and trade-offs one must look at the entire information flow since each step is interconnected.

(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).

Steps 1-4 (possibly 5) are considered to be part of the Entropy Source (ES) and provided by the `PollEntropy` instruction. Adding the software-side cryptographic steps 5-6 and control logic complements it into a True Random Number Generator (TRNG). This information flow is illustrated by Figure 1.

4.1 Noise Sources

The theory of random signals and electrical noise became well established in the post-World War II period [23, 43, 44]. We will give some examples of common noise sources that can be implemented in the processor itself (using standard cells).

4.1.1 Ring Oscillators. The most common entropy source type in production use today is based on "free running" ring oscillators and their timing jitter. Here, an odd number of inverters is connected into a loop from which noise source bits are sampled in relation to a reference clock [8]. The sampled bit sequence may be expected to be relatively uncorrelated (close to IID) if the sample rate is suitably low [26]. However further processing is usually required.

AMD [1], ARM [3], and IBM [28] are examples of ring oscillator TRNGs intended for high-security applications (see Section 5.1.)

There are related metastability-based generator designs such as Transition Effect Ring Oscillator (TERO) [51]. The differential/feedback Intel construction [18] is slightly different but also falls into the same general metastable oscillator-based category.

The main benefits of ring oscillators are: (1) They can be implemented with standard cell libraries without external components – and even on FPGAs [50], (2) there is an established theory for their behavior [8, 16, 17], and (3) ample precedent exists for testing and certifying them at the highest security levels.

[1] EntropyAssessment: https://github.com/usnistgov/SP800-90B_EntropyAssessment
[2] (In German) AIS 31-Implementierung in JAVA: https://www.bsi.bund.de/SharedDocs/Downloads/DE/BSI/Zertifizierung/Interpretationen/AIS_31_testsuit_zip

Ring oscillators also have well-known implementation pitfalls. Their output is sometimes highly dependent on temperature, which must be taken into account in testing and modeling. If the ring oscillator construction is parallelized, it is important that the number of stages and/or inverters in each chain is coprime to avoid entropy reduction due to harmonic "Huyghens synchronization" [5]. Such harmonics can also be inserted maliciously in a frequency injection attack, which can have devastating results [29]. Countermeasures are related to circuit design; environmental sensors, electrical filters, and usage of a differential oscillator may help.

4.1.2 Shot Noise. A category of random sources consisting of discrete events and modeled as a Poisson process is called "shot noise". There's a long-established precedent of certifying them; the AIS 31 document [26] itself offers reference designs based on noisy diodes. Shot noise sources are often more resistant to temperature changes than ring oscillators. Some of these generators can also be fully implemented with standard cells (The Rambus / Inside Secure generic TRNG IP [42] is described as a Shot Noise generator).

4.1.3 Other types of noise. It may be possible to certify more exotic noise sources and designs, although their stochastic model needs to be equally well understood and their CPU interfaces must be secure. See Section 5.3 for a discussion of Quantum entropy sources.

4.2 Samplers and GetNoise

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 since NIST SP 800-90B [49] urges implementors to protect the source in production devices. We are leaving access as a vendor-specific matter but we urge them to protect the raw source and to make it unavailable to casual users.

Rationale: Samplers can generate vast amounts of data. NIST SP 800-90B [49] defines a conceptual interface `GetNoise()` for the raw output and also anticipates that the actual interfaces "will depend on the entropy source deployed".

Building data paths to make the raw noise available through the ISA would be problematic as it is unclear how to "sample" possibly up to several gigabits of information per second in a way that is appropriately representative of its properties. SP 800-90B notes that it is permitted that such an interface be available only in "test mode" and that it is disabled when the source is operational.

> *"The vendor may use special methods (or devices, such as an oscilloscope) that require detailed knowledge of the source to collect raw data. The testing laboratory is required [...] to present a rationale why the data collections methods will not alter the statistical properties of the noise source or explain how to account for any change in the source's statistical characteristics [...]"*
>
> – FIPS 140 Implementation Guidance, 2019 [39]

4.3 Continuous Health Tests

If NIST SP 800-90B certification is required, the hardware should implement at least the health tests defined in Section 4.4 of [49]: repetition count test and adaptive proportion test.

Health monitoring requires some state information related to the noise source to be maintained. The tests should be designed in a way that a specific number of samples guarantees a state flush (no hung states). We suggest flush size $W \leq 1024$ to match with the NIST SP 800-90B required tests (See Section 4.4 in [49]). The state is also fully zeroized in a system reset.

Rationale: The two mandatory tests can be built with minimal circuitry. Full histograms are not required, only simple counter registers: repetition count, window count, and sample count. Repetition count is reset every time the output sample value changes; if the count reaches a certain cutoff limit, a noise alarm (`BIST`) or failure (`DEAD`) is signaled. Window counter is used to save every W'th output (typically $W \in 512, 1024$.) The frequency of this reference sample in the following window is counted; cutoff values are defined in the standard. We see that the structure of the mandatory tests is such that, if well implemented, no information is carried beyond a limit of W samples.

Section 4.5 of [49] explicitly permits additional developer-defined tests and several more were defined in early versions of FIPS 140-1 before being "crossed out". The choice of additional tests depends on the nature and implementation of the physical source.

Especially if a non-cryptographic a conditioner is used in hardware, it is possible that the AIS 31 [26] online tests are implemented by driver software. They can also be implemented in hardware. For some security profiles AIS 31 mandates that their tolerances are set in a way that the probability of an alarm is at least 10^{-6} yearly under "normal usage". Such requirements are problematic in modern applications since their probability is too high for critical systems[3]. There rarely is anything that can or should be done about a non-fatal alarm condition in an operator-free, autonomous system. However, AIS 31 allows the DRBG component to keep running despite a failure in its Entropy Source, so we suggest re-entering temporary `BIST` state (Section 3.2.2) to signal a non-fatal statistical error if such (non-actionable) signaling is necessary. Drivers and applications can react to this appropriately (or simply log it) but it will not directly affect the availability of the TRNG. A permanent error condition should result in `DEAD` state.

4.4 Non-cryptographic Conditioners

As noted in Section 1.1.2, physical randomness sources generally require a post-processing step called *conditioning* to meet the desired quality requirements, which are outlined in Section 2.2.

The approach taken in this interface is to allow a combination of non-cryptographic and cryptographic filtering to take place. The first stage (hardware) merely needs to be able to distill the entropy comfortably above 4 bits per byte (Sect. 2.2.1, Entropy) and to guarantee that the samples are independent (Sect. 2.2.2, IID).

- One may take a set of bits from a noise source and XOR them together to produce a less biased (and more independent) bit. If the source model is well understood, such a construction lends itself well to analysis and entropy estimation [14].
- The von Neumann extractor [52] looks at consecutive pairs of bits, rejects 00 and 11, and outputs 0 or 1 for 01 and 10, respectively. It will reduce the number of bits to less than 25% of original but the output is provably unbiased (assuming independence).

[3] Currently (2020) about 10^{10} secure elements are shipped yearly, many in critical applications and with TRNGs, according to https://www.eurosmart.com.

- Blum's extractor [11] can be used on sources whose behavior resembles n-state Markov chains. If its assumptions hold, it also removes dependencies, creating an IID source.
- Other linear and non-linear correctors such as those discussed by Dichtl and Lacharme [27].
- One may also implement a full cryptographic conditioner in the entropy source hardware, even though the software driver is required to implement one too.

Rationale: The main advantage of non-cryptographic filters is in their energy efficiency, relative simplicity, and amenability to mathematical analysis. If well designed, they can be evaluated in conjunction with a stochastic model of the noise source itself. They do not require computational hardness assumptions.

In some cases, an entropy source (and the circuit that implements it) may have a uniquely identifiable hardware "signature". This can be harmless or even useful (as random sources may exhibit PUF-like features) but highly undesirable in others (anonymized virtualized environments and enclaves). Such virtualized environments are probably better off just using `/dev/urandom` of the host rather than sharing the host's hardware-backed Entropy Source to the guest environment. Also note the source entropy requirement (Sect. 2.2.3, Secret State) when sharing such generators.

4.5 Cryptographic Conditioners

Cryptographic conditioners are always required on the software side of the PollEntropy ISA boundary. They may be also implemented on the hardware side if necessary. In any case, the PollEntropy output must always be compressed 2:1 (or more) before being used as keying material or considered "full entropy".

Examples of cryptographic conditioners include the random pool of the Linux operating system, secure hash functions (SHA-2/3, SHAKE [35, 36]), and the AES-based CBC-MAC construction of SP 800-90B [49].

In some constructions, such as the Linux RNG and SHA-3/SHAKE [36] based generators the cryptographic conditioning and output (DRBG) generation is provided by the same component.

Rationale: For many low-power targets constructions such as Intel's [30] and AMD's [1] hardware AES CBC-MAC conditioner would be too complex and expensive to implement solely to serve `PollEntropy`. On the other hand, simpler non-cryptographic conditioners may be too wasteful on input entropy if very high-quality random output is required – ARM TrustZone TRBG [3] outputs only 10Kbit/sec at 200 MHz. Hence a resource-saving compromise is made between hardware and software generation that allows an implementation to use the RISC-V cryptographic ISA.

4.6 The Final Random: DRBGs

All random bits reaching end users and applications must come from a cryptographic DRBG. These are generally implemented by the driver component in software. The RISC-V AES and SHA instruction set extensions should be used if available [48], since they offer additional security features such as timing attack resistance.

Currently recommended DRBGs are defined in NIST SP 800-90A (Rev 1) [7]: `CTR_DRBG`, `Hash_DRBG`, and `HMAC_DRBG`. Certification often requires known answer tests (KATs) for the symmetric components and the DRBG as a whole. These are significantly easier to implement in software than in hardware. In addition to the directly certifiable SP 800-90A DRBGs, a Linux-style random pool construction based on ChaCha20 [32] can be used, or an appropriate construction based on SHAKE256 [36].

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

5 CASE STUDIES

We discuss current mainstream TRNG ISAs and their implementations, anticipated cryptanalytic requirements, and our minimalistic reference design, Minidice.

5.1 Current Mainstream TRNG ISAs

TRNGs are available in many mainstream CPUs and mobile devices. This is by no means an exhaustive list.

5.1.1 Intel Secure Key. Intel's random number interface is known as "Intel Secure Key" [30] and has been available via the RDRAND instruction since Ivy Bridge (2012). A reseeding instruction RDSEED was added for Broadwell (2014). Internally the Intel solution is based on a self-oscillating feedback circuit [18], CBC-MAC conditioning and the CTR_DRBG [7] – both built from AES-128 .

5.1.2 AMD. AMD's interface is compatible with Intel's, but internally uses 16 ring oscillator chains as a noise source, CBC-MAC conditioning but a higher-security AES-256 version of CTR_DRBG. AMD additionally offers raw noise output via the TRNG_RAW register in its cryptographic coprocessor (CCP) [1].

5.1.3 ARM-based devices. The ARMv8.5-RNG ISA extension has instructions RNDR (Random Number) and RNDRRS (Reseed Random Number) that seem to work much like RDRAND and RDSEED [4]. These instructions are new and not very widely available.

More often ARM devices interface TRNGs via a bus (e.g. APB) instead of ISA. The TrustZone TRNG [3] is actually a non-ISA Entropy Source, built from ring oscillators [8] and a von Neumann debiaser [52] – without a DRBG or other cryptographic components. This makes the TRNG low-bandwidth but energy efficient.

5.2 A full DRBG in Hardware

> *"I am so glad I resisted pressure from Intel engineers to let /dev/random rely only on the RDRAND instruction. Relying solely on an implementation sealed inside a chip and which is impossible to audit is a BAD idea."*
>
> –Theodore Ts'o, author of the Linux RNG.[4]

Our proposal does not prevent RISC-V implementors from creating full DRBG implementations as custom instructions, just like Intel and ARM does. However, we can offer some reasons why that may not be as useful as one might think.

5.2.1 No black boxes. Cryptographers actually don't want to use hardware DRBGs directly as it would force them to blindly rely on hardware. This is of course much more of an issue for a Linux-profile system than to a security microcontroller where the hardware and firmware are likely to come from the same vendor.

[4] September 5, 2013 (after Snowden): https://news.ycombinator.com/item?id=6336505

If the DRBG is hardwired to the entropy source, and hardware is sourced from a third party, there is usually no easy way to verify that it is doing what it is supposed to be doing. As a source of secret keying material, an RNG is an obvious location for a potential cryptographic backdoor. It has a large potential for supply chain attacks such as hardware trojans [9] and other un-auditable backdoors in the style of NSA's `Dual_EC_DRBG` [13].

However, most operating system kernels and well-designed cryptographic libraries use (and welcome) CPU-sourced entropy source as one of the many ingredients to their "entropy soup". Hence the DRBGs are ultimately implemented in software anyway – possibly using cryptographic ISA instructions for speed. This approach eliminates a single point of vulnerability in entropy sourcing and allows a higher level of audit transparency.

5.2.2 Flexibility. Deterministic hardware DRBGs can become technically obsolete quickly and ISA updates are hard. This can happen due to a standards update or hardcoded design problems and limitations. Intel's RDRAND is designed around AES-128 with forced reseeding only every 512 invocations.

There is a simple attack that demonstrates the entropy bottleneck and forward-security problem; if two 128-bit output blocks are known, the secret key and counter can be recovered with 2^{128} classical effort. This in turn can be expanded to the entire segment of secret blocks, revealing up to $512 \times 128 = 65536$ bits of keying material with no additional effort. Intel's RDRAND generator can, therefore, create a security bottleneck in applications that are specified to support "256-bit" security.

For additional entropy (in case of unavailability of RDSEED), Intel recommends polling 1024×64-bit words out of the RDRAND to force a reseed flush and then reducing the DRBG output back to 128 bits (a process with 99.8% redundancy) [30]. Users were recommended to effectively "bypass" the large, expensive Intel DRBG component at cost of thousands and thousands of cycles only a few years after its introduction.

5.2.3 Resource sharing. High-throughput DRBG sharing can be tricky to implement, as the CrossTalk / SRBDS vulnerability shows [41]. This vulnerability causes the same random output bytes to be available simultaneously to multiple cores. The SRBDS mitigation serializes the entire DRBG operation by locking the (memory) bus for RDRAND calls and can have a serious performance impact [20]. Of course, such problems may also occur if an Entropy Source is shared rather than a DRBG. However, Entropy Source interfaces are not designed for throughput so more conservative design choices for the sharing mechanism can be made.

5.2.4 Area. In a small microcontroller-type RISC-V implementation it is difficult to justify the hardware area requirement of a full-sized AES or some other cryptographic algorithm just to provide cryptographic conditioning or a DRBG output. One would prefer to use that area for cryptographic instructions that actually increase the throughput of secure communications (TLS, IPSec), or disk encryption, *in addition* to DRBG output. Random number entropy is essentially never a performance bottleneck in cryptographic implementations so using a lot of transistors for RNG speed is not compatible with the quantitative approach usually associated with RISC CPU design.

5.3 Quantum vs Classical Random

A Quantum Random Number Generator (QRNG) is a TRNG whose source of randomness can be unambiguously identified to be a *specific* quantum phenomenon such as quantum state superposition, quantum state entanglement, Heisenberg uncertainty, quantum tunneling, spontaneous emission, or radioactive decay [21].

Direct quantum entropy is theoretically the best possible kind of entropy. A typical TRNG based on electronic noise is also largely based on quantum phenomena and is equally unpredictable - the difference is that the relative amount of quantum and classical physics involved is difficult to quantify for a classical TRNG.

QRNGs are designed in a way that allows the amount of quantum-origin entropy to be modeled and estimated. This distinction is important in the security model used by QKD (Quantum Key Distribution) security mechanisms which can be used to protect the physical layer (such as fiber optic cables) against interception by using quantum mechanical effects directly.

This security model means that many of the available QRNG devices do not use cryptographic conditioning and may fail cryptographic statistical requirements [19]. Hence they should perhaps be considered to be entropy sources instead.

Relatively little research has gone into QNRG implementation security, but many QRNG designs are arguably more susceptible to leakage than classical generators (such as ring oscillators) as they tend to employ external components and mixed materials.

> *"The NCSC believes that classical RNGs will continue to meet our needs for government and military applications for the foreseeable future."*
>
> – U.K. QRNG Guidance, March 2020 [33].

Post-Quantum Cryptography. The classical/quantum origin of randomness is not relevant in NIST Post-Quantum Cryptography (PQC) [37]. Recall that cryptography aims to protect the confidentiality and integrity of data itself and does not place any requirements on the physical communication channel (like QKD). Classical good-quality TRNGs are perfectly fine for generating the secret keys for PQC protocols that are hard for quantum computers to break, but implementable on classical computers. What matters in cryptography is that the secret keys have enough true randomness (entropy) and that they are generated and stored securely.

Of course one must avoid DRBGs that are based on problems that are easily solvable with quantum computers, such as factoring [47] in the case of Blum-Blum-Shub generator [10]. However most symmetric algorithms are less affected as the best quantum attacks are still exponential to key size [15].

As an example, the original Intel RNG [30], whose output generation is based on AES-128 can be attacked using Grover's algorithm with approximately square-root effort [22]. While even "64-bit" quantum security is extremely difficult to break, many applications specify a higher security requirement. NIST [37] defines AES-128 to be "Category 1" equivalent post-quantum security, while AES-256 is "Category 5" (highest). We avoid this possible future issue by exposing a more direct access to the entropy source, which can derive its security from information-theoretic assumptions only.

5.4 Minidice

Minidice is a minimalistic reference implementation of the RISC-V TRNG design (as discussed in this work) for the microcontroller profile (Section 1.2). This open source implementation is in Verilog and can be easily integrated into cores such as PicoRV32[5]. It can also be expanded to the Linux profile with relatively little work.

Minidice demonstrates how small a RISC-V TRNG can be – just a few hundred gate equivalents. Together with a 32-bit Scalar AES instruction set extension "SAES32" [45, 48], a device can be made self-sufficient in entropy with a very small area and energy cost. The quality of randomness is sufficient to support classical and post-quantum cryptographic key generation and other operations at the highest security levels.

The noise source of Minidice is based on a configurable ring oscillator. If multiple, parallel chains are defined, their output is XORed for the conditioner component. The conditioner uses a variant of Blum's [11] method to remove simple bias and correlation between consecutive output bits.

Due to the structure of the conditioner, output bits are generated at a variable rate, depending on the noise source. The 16-bit output word is continuously overwritten. Furthermore, the data pathways implement wipe-on-read (i.e. zeroization when polled).

Minidice implements the repetition count test and adaptive proportion test (Section 4.3) by default. An additional delay counter is used for a start-up / on-demand test. A reset clears all registers and puts the device into `BIST` state – after a period of successful continuous tests the implementation enters into normal operation or the `DEAD` state in case of failure.

The firmware driver component consists of a polling mechanism, AES-CBC type cryptographic conditioner and `CTR_DRBG` built from AES-256 using the SAES32 instructions. The firmware footprint of only a few kilobytes, including KAT tests and integrity tests.

Minidice can be synthesized to FPGAs that lack native hardware generators, such as Xilinx 7 Series and Lattice ECP5 FPGAs. It can serve as an experimentation target and as a freely available "real life" component will undoubtedly be subjected to many kinds of attacks, which will serve to refine its design.

It is our plan and hope that when this (or related) open source TRNG design is taken through various official certification processes, all of the specific documents and arguments from the designers and certification laboratory will be published. This in turn will ease the validation and certification of subsequent RISC-V `PollEntropy` and other open source TRNG implementations, which is the main goal of its authors.

6 CONCLUSIONS

The RISC-V Cryptographic Extensions Task Group is working to introduce True Random Number Generator (TRNG) support. The proposed instruction and the wider TRNG architecture is designed to allow compliance with FIPS 140-3 and related international standards such as AIS 31 at high assurance levels, while being extremely lightweight to implement.

The proposal differs from other contemporary TRNG ISAs in that it is based on a polling paradigm and focuses on providing Entropy Source (ES) functionality only. The interface can be used to instantiate a random number generator of virtually any strength if the `PollEntropy` implementation is stateless (non-virtual).

We described the ISA interface and the basic requirements it sets on the entropy. These definitions are needed so that interoperable drivers can be implemented – they simply set the minimum standards that can be expected from polled randomness.

We then discussed information flows, monitoring, and implementation aspects in more detail, diving into the rationale of various engineering decisions that are required.

We concluded with case studies that contain a brief overview of generators in current mainline ISAs, commentary on the impact of quantum threat on TRNG generation (not a problem for post-quantum cryptography), and a description of Minidice, an open source TRNG. Like the rest of this work, Minidice is aimed at helping RISC-V users and the open source community to have access to built-in, highly secure sources of randomness.

[5]PicoRV32 - A Size-Optimized RISC-V CPU https://github.com/cliffordwolf/picorv32

ACKNOWLEDGMENTS

We thank RISC-V Cryptographic Extensions Task Group members for their input, especially Andy Glew, Barry Spinney, Derek Atkins, Ken Dockser, and Nathan Menhorn. This work was supported in part by Innovate UK (R&D Project Ref.: 105747), and by EPSRC (Grant No.: EP/R012288/1, under the RISE programme.)

REFERENCES

[1] AMD. 2017. AMD Random Number Generator. AMD TechDocs. https://www.amd.com/system/files/TechDocs/amd-random-number-generator.pdf

[2] Ross J. Anderson. 2020. *Security engineering - a guide to building dependable distributed systems (3. ed.).* Wiley. https://www.cl.cam.ac.uk/~rja14/book.html

[3] ARM. 2017. ARM TrustZone True Random Number Generator: Technical Reference Manual. ARM 100976_0000_00_en (rev. r0p0). http://infocenter.arm.com/help/index.jsp?topic=/com.arm.doc.100976_0000_00_en

[4] ARM. 2020. Arm Architecture Registers: Armv8, for Armv8-A architecture profile. ARM DDI 0595 (ID033020). https://developer.arm.com/docs/ddi0595/g

[5] Per Bak. 1986. The Devil's Staircase. *Phys. Today* 39, 12 (December 1986), 38–45. https://doi.org/10.1063/1.881047

[6] Romain Bardou, Riccardo Focardi, Yusuke Kawamoto, Lorenzo Simionato, Graham Steel, and Joe-Kai Tsay. 2012. Efficient Padding Oracle Attacks on Cryptographic Hardware. In *Advances in Cryptology - CRYPTO 2012 - 32nd Annual Cryptology Conference, Santa Barbara, CA, USA, August 19-23, 2012. Proceedings (Lecture Notes in Computer Science)*, Reihaneh Safavi-Naini and Ran Canetti (Eds.), Vol. 7417. Springer, 608–625. https://doi.org/10.1007/978-3-642-32009-5_36

[7] Elaine Barker and John Kelsey. 2015. Recommendation for Random Number Generation Using Deterministic Random Bit Generators. NIST Special Publication SP 800-90A Revision 1. https://doi.org/10.6028/NIST.SP.800-90Ar1

[8] Mathieu Baudet, David Lubicz, Julien Micolod, and André Tassiaux. 2011. On the Security of Oscillator-Based Random Number Generators. *J. Cryptology* 24, 2 (2011), 398–425. https://doi.org/10.1007/s00145-010-9089-3

[9] Georg T. Becker, Francesco Regazzoni, Christof Paar, and Wayne P. Burleson. 2014. Stealthy dopant-level hardware Trojans: extended version. *J. Cryptographic Engineering* 4, 1 (2014), 19–31. https://doi.org/10.1007/s13389-013-0068-0

[10] Lenore Blum, Manuel Blum, and Mike Shub. 1986. A Simple Unpredictable Pseudo-Random Number Generator. *SIAM J. Comput.* 15, 2 (1986), 364–383. https://doi.org/10.1137/0215025

[11] Manuel Blum. 1986. Independent unbiased coin flips from a correlated biased source – A finite state Markov chain. *Combinatorica* 6, 2 (1986), 97–108. https://doi.org/10.1007/BF02579167

[12] BSI. 2013. Evaluation of random number generators. Version 0.10, BSI. https://www.bsi.bund.de/SharedDocs/Downloads/DE/BSI/Zertifizierung/Interpretationen/AIS_20_AIS_31_Evaluation_of_random_number_generators_e.html

[13] Stephen Checkoway, Jacob Maskiewicz, Christina Garman, Joshua Fried, Shaanan Cohney, Matthew Green, Nadia Heninger, Ralf-Philipp Weinmann, Eric Rescorla, and Hovav Shacham. 2016. A Systematic Analysis of the Juniper Dual EC Incident. In *Proceedings of the 2016 ACM SIGSAC Conference on Computer and Communications Security, Vienna, Austria, October 24-28, 2016*, Edgar R. Weippl, Stefan Katzenbeisser, Christopher Kruegel, Andrew C. Myers, and Shai Halevi (Eds.). ACM, 468–479. https://doi.org/10.1145/2976749.2978395

[14] Robert B. Davies. 2002. Exclusive OR (XOR) and hardware random number generators. Author-hosted manuscript. http://www.robertnz.net/pdf/xor2.pdf

[15] Lov K. Grover. 1996. A Fast Quantum Mechanical Algorithm for Database Search. In *Proceedings of the Twenty-eighth Annual ACM Symposium on Theory of Computing (STOC '96)*. ACM, 212–219. https://doi.org/10.1145/237814.237866

[16] Ali Hajimiri and Thomas H. Lee. 1998. A general theory of phase noise in electrical oscillators. *IEEE Journal of Solid-State Circuits* 33, 2 (1998), 179–194. https://doi.org/10.1109/4.658619

[17] Ali Hajimiri, Sotirios Limotyrakis, and Thomas H. Lee. 1999. Jitter and phase noise in ring oscillators. *IEEE Journal of Solid-State Circuits* 34, 6 (June 1999), 790–804. https://doi.org/10.1109/4.766813

[18] Mike Hamburg, Paul Kocher, and Mark E. Marson. 2012. Analysis of Intel's Ivy Bridge Digital Random Number Generator. Technical Report, Cryptography Research (Prepared for Intel).

[19] Darren Hurley-Smith and Julio César Hernández-Castro. 2020. Quantum Leap and Crash: Searching and Finding Bias in Quantum Random Number Generators. *ACM Transactions on Privacy and Security* 23, 3 (June 2020), 1–25. https://doi.org/10.1145/3403643

[20] Intel. 2020. SRBDS Mitigation Impact on Intel® Secure Key. Intel Developer Zone. https://software.intel.com/security-software-guidance/insights/srbds-mitigation-impact-intel-secure-key

[21] ITU. 2019. Quantum noise random number generator architecture. Recommendation ITU-T X.1702. https://www.itu.int/rec/T-REC-X.1702-201911-I/en

[22] Samuel Jaques, Michael Naehrig, Martin Roetteler, and Fernando Virdia. 2020. Implementing Grover Oracles for Quantum Key Search on AES and LowMC. In *Advances in Cryptology - EUROCRYPT 2020 - 39th Annual International Conference on the Theory and Applications of Cryptographic Techniques, Zagreb, Croatia, May 10-14, 2020, Proceedings, Part II (Lecture Notes in Computer Science)*, Anne Canteaut and Yuval Ishai (Eds.), Vol. 12106. Springer, 280–310. https://doi.org/10.1007/978-3-030-45724-2_10

[23] Wilbur B. Davenport Jr. and William L. Root. 1958. *An Introduction to the Theory of Random Signals and Noise.* McGraw-Hill. 401 pages. https://ieeexplore.ieee.org/servlet/opac?bknumber=5265617

[24] Dusko Karaklajic, Jörn-Marc Schmidt, and Ingrid Verbauwhede. 2013. Hardware Designer's Guide to Fault Attacks. *IEEE Trans. Very Large Scale Integr. Syst.* 21, 12 (2013), 2295–2306. https://doi.org/10.1109/TVLSI.2012.2231707

[25] Wolfgang Killmann and Werner Schindler. 2001. A Proposal for: Functionality classes and evaluation methodology for true (physical) random number generators. AIS 31, Version 3.1, English Translation, BSI. https://www.bsi.bund.de/SharedDocs/Downloads/DE/BSI/Zertifizierung/Interpretationen/AIS_31_Functionality_classes_evaluation_methodology_for_true_RNG_e.html

[26] Wolfgang Killmann and Werner Schindler. 2011. A Proposal for: Functionality classes for random number generators. AIS 20 / AIS 31, Version 2.0, English Translation, BSI. https://www.bsi.bund.de/SharedDocs/Downloads/DE/BSI/Zertifizierung/Interpretationen/AIS_31_Functionality_classes_for_random_number_generators_e.html

[27] Patrick Lacharme. 2008. Post-Processing Functions for a Biased Physical Random Number Generator. In *Fast Software Encryption, 15th International Workshop, FSE 2008, Lausanne, Switzerland, February 10-13, 2008, Revised Selected Papers (Lecture Notes in Computer Science)*, Kaisa Nyberg (Ed.), Vol. 5086. Springer, 334–342. https://doi.org/10.1007/978-3-540-71039-4_21

[28] John S. Liberty, Adrian Barrera, David W. Boerstler, Thomas B. Chadwick, Scott R. Cottier, H. Peter Hofstee, Julie A. Rosser, and Marty L. Tsai. 2013. True hardware random number generation implemented in the 32-nm SOI POWER7+ processor. *IBM J. Res. Dev.* 57, 6 (2013). https://doi.org/10.1147/JRD.2013.2279599

[29] A. Theodore Markettos and Simon W. Moore. 2009. The Frequency Injection Attack on Ring-Oscillator-Based True Random Number Generators. In *Cryptographic Hardware and Embedded Systems - CHES 2009, 11th International Workshop, Lausanne, Switzerland, September 6-9, 2009, Proceedings (Lecture Notes in Computer Science)*, Christophe Clavier and Kris Gaj (Eds.), Vol. 5747. Springer, 317–331. https://doi.org/10.1007/978-3-642-04138-9_23

[30] John P. Mechalas. 2018. Intel® Digital Random Number Generator (DRNG) Software Implementation Guide. Intel Technical Report, Version 2.1. https://software.intel.com/content/www/us/en/develop/articles/intel-digital-random-number-generator-drng-software-implementation-guide.html

[31] Daniel Moghimi, Berk Sunar, Thomas Eisenbarth, and Nadia Heninger. 2020. TPM-FAIL: TPM meets Timing and Lattice Attacks. In *29th USENIX Security Symposium (USENIX Security 20)*. USENIX Association, To appear. https://www.usenix.org/conference/usenixsecurity20/presentation/moghimi-tpm

[32] Stephan Müller. 2020. Documentation and Analysis of the Linux Random Number Generator, Version 3.6. Prepared for BSI by atsec information security GmbH. https://www.bsi.bund.de/SharedDocs/Downloads/EN/BSI/Publications/Studies/LinuxRNG/LinuxRNG_EN.pdf

[33] NCSC. 2020. Quantum security technologies. White paper, Version 1.0. National Cyber Security Centre (UK). https://www.ncsc.gov.uk/whitepaper/quantum-security-technologies

[34] NIST. 2001. Advanced Encryption Standard (AES). Federal Information Processing Standards Publication FIPS 197. https://doi.org/10.6028/NIST.FIPS.197

[35] NIST. 2015. Secure Hash Standard (SHS). Federal Information Processing Standards Publication FIPS 180-4. https://doi.org/10.6028/NIST.FIPS.180-4

[36] NIST. 2015. SHA-3 Standard: Permutation-Based Hash and Extendable-Output Functions. Federal Information Processing Standards Publication FIPS 202. https://doi.org/10.6028/NIST.FIPS.202

[37] NIST. 2016. Submission Requirements and Evaluation Criteria for the Post-Quantum Cryptography Standardization Process. Official Call for Proposals, National Institute for Standards and Technology. http://csrc.nist.gov/groups/ST/post-quantum-crypto/documents/call-for-proposals-final-dec-2016.pdf

[38] NIST. 2019. Security Requirements for Cryptographic Modules. Federal Information Processing Standards Publication FIPS 140-3. https://doi.org/10.6028/NIST.FIPS.140-3

[39] NIST and CCCS. 2019. Implementation Guidance for FIPS 140-2 and the Cryptographic Module Validation Program. CMVP Update. https://csrc.nist.gov/CSRC/media/Projects/Cryptographic-Module-Validation-Program/documents/fips140-2/FIPS1402IG.pdf

[40] NSA/CSS. 2015. Commercial National Security Algorithm Suite. https://apps.nsa.gov/iaarchive/programs/iad-initiatives/cnsa-suite.cfm

[41] Hany Ragab, Alyssa Milburn, Kaveh Razavi, Herbert Bos, and Cristiano Giuffrida. 2021. CrossTalk : Speculative Data Leaks Across Cores Are Real. In *IEEE Symposium on Security & Privacy 2021*. IEEE, To appear. https://download.vusec.net/papers/crosstalk_sp21.pdf

[42] Rambus. 2020. TRNG-IP-76 / EIP-76 Family of FIPS Approved True Random Generators. Commercial Cryptographic IP data sheet. https://www.rambus.com/security/crypto-accelerator-hardware-cores/basic-crypto-blocks/trng-ip-76/

[43] Stephen O. Rice. 1944. Mathematical analysis of random noise (Parts I-II). *The Bell System Technical Journal* 23, 3 (July 1944), 282–332. https://doi.org/10.1002/j.1538-7305.1944.tb00874.x

[44] Stephen O. Rice. 1945. Mathematical analysis of random noise (Parts III-IV)). *The Bell System Technical Journal* 24, 1 (January 1945), 46–156. https://doi.org/10.1002/j.1538-7305.1945.tb00453.x

[45] Markku-Juhani O. Saarinen. 2020. A Lightweight ISA Extension for AES and SM4. To appear in SECRISC-V 2020. arXiv:2002.07041. https://arxiv.org/abs/2002.07041

[46] Jim Salter. 2019. How a months-old AMD microcode bug destroyed my weekend. Ars Technica. https://arstechnica.com/gadgets/2019/10/how-a-months-old-amd-microcode-bug-destroyed-my-weekend/

[47] Peter W. Shor. 1994. Algorithms for quantum computation: Discrete logarithms and factoring. In *35th Annual Symposium on Foundations of Computer Science, Santa Fe, New Mexico, USA, 20-22 November 1994*. IEEE, 124–134. https://doi.org/10.1109/SFCS.1994.365700

[48] RISC-V Crypto TG. 2020. RISC-V Cryptography Extensions. Editor's location – to be integrated with main specifications. https://github.com/scarv/riscv-crypto

[49] Meltem Sönmez Turan, Elaine Barker, John Kelsey, Kerry A. McKay, Mary L. Baish, and Mike Boyle. 2018. Recommendation for the Entropy Sources Used for Random Bit Generation. NIST Special Publication SP 800-90B. https://doi.org/10.6028/NIST.SP.800-90B

[50] Boyan Valtchanov, Viktor Fischer, Alain Aubert, and Florent Bernard. 2010. Characterization of randomness sources in ring oscillator-based true random number generators in FPGAs. In *13th IEEE International Symposium on Design and Diagnostics of Electronic Circuits and Systems, DDECS 2010, Vienna, Austria, April 14-16, 2010*, Elena Gramatová, Zdenek Kotásek, Andreas Steininger, Heinrich Theodor Vierhaus, and Horst Zimmermann (Eds.). IEEE Computer Society, 48–53. https://doi.org/10.1109/DDECS.2010.5491819

[51] Michal Varchola and Milos Drutarovský. 2010. New High Entropy Element for FPGA Based True Random Number Generators. In *Cryptographic Hardware and Embedded Systems, CHES 2010, 12th International Workshop, Santa Barbara, CA, USA, August 17-20, 2010. Proceedings (Lecture Notes in Computer Science)*, Stefan Mangard and François-Xavier Standaert (Eds.), Vol. 6225. Springer, 351–365. https://doi.org/10.1007/978-3-642-15031-9_24

[52] John von Neumann. 1951. Various Techniques Used in Connection with Random Digits. In *Monte Carlo Method*, A. S. Householder, G. E. Forsythe, and H. H. Germond (Eds.). National Bureau of Standards Applied Mathematics Series, Vol. 12. US Government Printing Office, Washington, DC, Chapter 13, 36–38. https://mcnp.lanl.gov/pdf_files/nbs_vonneumann.pdf

[53] Andrew Waterman and Krste Asanović (Eds.). 2019. *The RISC-V Instruction Set Manual, Volume I: User-Level ISA.* RISC-V Foundation. https://riscv.org/specifications/ Document Version 20191213.

[54] Andrew Waterman and Krste Asanović (Eds.). 2019. *The RISC-V Instruction Set Manual, Volume II: Privileged Architecture.* RISC-V Foundation. https://riscv.org/specifications/ Document Version 20190608-Priv-MSU-Ratified.