A Provable-Security Analysis of Intel's Secure Key RNG

Thomas Shrimpton and R. Seth Terashima

Dept. of Computer Science, Portland State University
{teshrim,seth}@cs.pdx.edu

Abstract. We provide the first provable-security analysis of the Intel Secure Key hardware RNG (ISK-RNG), versions of which have appeared in Intel processors since late 2011. To model the ISK-RNG, we generalize the PRNG-with-inputs primitive, introduced Dodis et al. introduced at CCS'13 for their /dev/[u]random analysis. The concrete security bounds we uncover tell a mixed story. We find that ISK-RNG lacks backward-security altogether, and that the forward-security bound for the "truly random" bits fetched by the `RDSEED` instruction is potentially worrisome. On the other hand, we are able to prove stronger forward-security bounds for the pseudorandom bits fetched by the `RDRAND` instruction. En route to these results, our main technical efforts focus on the way in which ISK-RNG employs CBCMAC as an entropy extractor.

Keywords: random number generator, entropy extraction, provable security

Table of Contents

1 Introduction

In late 2011, Intel began production Ivy Bridge architecture processors, which introduced a new pseudorandom number generator (PRNG), fully implemented in hardware. Access to this PRNG is through the `RDRAND` instruction (pronounced "read rand"), and benchmarks demonstrate a throughput of over 500 MB/s on a quad-core Ivy Bridge processor [8]. The forthcoming Broadwell architecture will also support an additional instruction, `RDSEED` ("read seed"), which delivers true random bits, as opposed to cryptographically pseudorandom ones. Both `RDRAND` and `RDSEED` fall under the Intel Secure Key umbrella, so we will refer to the new hardware as the ISK-RNG [9].

The ISK-RNG has received a third-party lab evaluation [7], commissioned by Intel, but has yet to receive an academic, provable-security treatment along the lines of that given the `/dev/random` and `/dev/urandom` software RNGs by a line of papers [6,1,5]. We provide such a treatment.

Our abstract model for the ISK-RNG is that of a PRNG-with-input (PWI), established by Barak and Halevi [1] and extended by Dodis et al. [5]. To better capture important design features of the ISK-RNG we make several improvements to the PWI abstraction, which have significant knock-on effects for the associated security notions. Our results establish the security of the ISK-RNG relative to these notions. Doing so requires careful analysis of the security of the underlying components. Our findings are mixed, suggesting that `RDSEED` may not be as secure as one might hope, but with stronger results for `RDRAND`.

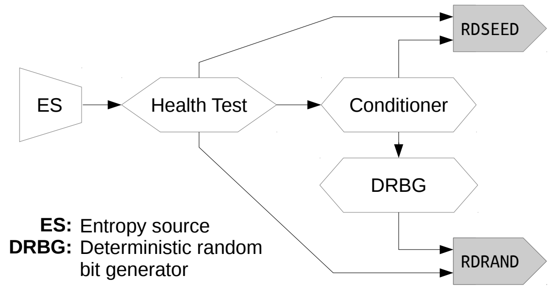

Fig. 1. Overview of Intel's hardware PRNG.

The ISK-RNG architecture. At a high-level, the ISK-RNG consists of four main components, as shown in Figure 1. At the heart is the hardware *entropy source*, which provides entropic bits at 3GHz to a 256-bit raw-sample buffer. This buffer is subjected to a battery of *health tests*, whose job is to infer information about the quality of bits provided by the source and, more generally, about the quality of `RDRAND` and `RDSEED` outputs. The buffer is then passed to a *conditioner* (i.e. an entropy extractor), whose job is to turn the raw, high min-entropy bits provided by the source into bitstrings that are close to uniform. These uniform bitstrings are then expanded by a deterministic PRNG, providing a high-speed source of computationally uniform bits. Calls to the `RDRAND` instruction will read from these bits, whereas calls to `RDSEED` will read from the conditioner output.

Modeling the ISK-RNG. The first step in our analysis of the ISK-RNG is to develop an appropriate model of it. We take as our starting point the the *pseudorandom number generator with input* (PWI) primitive, formalized by Dodis, Pointcheval, Ruhault, Vergniaud and Wichs [5] (hereafter DPRVW) as a model for `/dev/[u]random`. At a high level, a PWI surfaces three algorithms: one to initialize the internal state of primitive, one that turns the current state into (a new state and) an output for use by calling applications, and one that updates the state as a function of an *externally* provided input. Exposing an external input captures the practical situation in which PRNG outputs may depend up upon external sources of (assumed) entropy.

One contribution of this paper is to generalize the PWI abstraction in ways that better capture the ISK-RNG. These include allowances for:

1. Non-uniform PWI state: Real-world PRNGs like ISK-RNG have internal state that is is not properly modeled as being uniformly random; for example, because it contains counters or fixed strings. DPRVW made progress in abandoning the restrictions of prior works, whose security notions mandate that the internal state be indistinguishable from a uniformly random one. However, they do not go far enough, as they still assume the PWI is *initialized* with a uniform random state. This leads to problems: one can construct a PWI that is provably "forward-secure" in the sense defined by DPRVW, but is clearly not forward-secure in the commonly accepted sense. Conversely, there are PWIs that seemingly should be considered forward-secure, but are not under the DPRVW definition. We provide examples in Appendix A.

2. Realistic initialization: Relatedly, the current PWI formalism cannot model setup procedures, such as those used by the ISK-RNG.

3. Multiple external interfaces: the ISK-RNG API exposes two interfaces that depend on the same state, namely `RDRAND` and `RDSEED`. Hence the model should reflect this. We note that ISK-RNG isn't unique in this regard: `/dev/random` and `/dev/urandom` also share some state [11], and so generalizing the PWI model to allow multiple interfaces is a necessary prerequisite to establishing any positive results there.

4. Blocking behavior: the ISK-RNG and other real-world RNGs (e.g. /dev/random) block when their tests suggest that they have not accumulated sufficient entropy to generate random numbers securely. Capturing this behavior allows us to ensure that this mechanism cannot inadvertently compromise security. We briefly discuss the related issue of availability in Appendix D.

To deal with non-uniform state, we will introduce an analytical tool called a *masking function*. Loosely speaking, a masking function M is a tool for specifying what the "ideal" version $M(S)$ of any given PWI state S would be. This allows us to give general results about PWI security (e.g. what can be achieved when the state is ideal), yet admits per-scheme specification of what "ideal" means. We define masking functions, and incorporate them into the security notion given by DPRVW in such a way that their results can be quickly lifted to our setting. Masking functions also allow us to frame an appropriate definition for secure initialization.

Summary of security findings. We consider security of the ISK-RNG relative to four PWI-security notions, adopted (with modifications) from DPRVW, each of which considers the entropy source as potentially adversarial: *resilience*, the apparent randomness of `RDRAND` and `RDSEED` outputs; *forward security*, the apparent randomness of previous `RDRAND` and `RDSEED` outputs once the PWI state is revealed; *backward security*, the apparent randomness of future `RDRAND` and `RDSEED` outputs from a corrupted PWI state; and *robustness*, the apparent randomness of `RDRAND` and `RDSEED` outputs when state observation and corruption may happen at arbitrary times.

We prove ISK-RNG's forward security under assumptions well-motiviated by the hardware. The concrete bounds, however, are somewhat disappointing. The limiting factor here is the entropy extractor, which computes CBCMAC over the entropic bits. Even in the random permtuation model, the best known results for CBCMAC in this role can only guarantee its outputs are a statistical distance of 2^{-64} from uniform. This would be fine for a modest number of invocations, but a hybrid factor quickly pulls it away from a reasonable security level (e.g., 2^{-40} or 2^{-30}).

As a result, we can prove stronger bounds when RDSEED (which directly reads extractor outputs) is not required to return random bits. The interpretation here is that RDRAND (alone) delivers pseudorandom bits with a strong security bound even if an adversary has access to both RDRAND and RDSEED. This latter result also has the benefit of applying to Ivy Bridge chips, where RDSEED is unavailable.

However, ISK-RNG lacks backwards security, and hence lacks robustness, as well. This results from a combination of its power-saving and output-buffering features: outputs are buffered to reduce latency, and the entropy source is turned off when the buffers are full. As a result, buffered outputs and future DRBG seeds can be predicted following a state compromise — even if the system would otherwise have had time to gather fresh entropy and overwrite stagnant buffers. That being said, the fact that ISK-RNG is implemented in hardware makes this particular attack vector appear unlikely. Moreover, we prove that although such attackers will inevitably succeed in predicting a non-trivial amount of future outputs, ISK-RNG eventually recovers from such compromises.

Analyzing the ISK-RNG entropy extractor. Two core technical results of the paper are concerned with analyzing the ISK-RNG entropy extractor, which employs CBC-MAC over AES-128, using the fixed string $\mathsf{AES}_{\mathbf{0}}(1)$ as the AES key. Although Intel documents [13] appeal to a CRYPTO'02 paper by Dodis, Gennaro, Håstad and Krawcyzk [4] for support, this direct appeal is not well founded. There are significant technical obstacles to overcome before these CBC-MAC results can be applied. For example, because extractor-dependent state is maintained across extractions (including state revealed to the adversary by RDSEED), a crucial "seed independence" assumption is violated. We discuss and resolve these issues in Section 4.

Discussion of the attack model. The DPRVW syntax and security notions, which we take as our starting point, assume a strongly adversarial operating environment. They treat the entropy source as adversarial (although not pathologically bad), and allow attackers to observe, even corrupt, the full internal state of the PWI. One might argue that these choices are inappropriate in the case of ISK-RNG. After all, the entire RNG is implemented in 22-32nm hardware, so direct observation of the internal state should require the use of expensive and highly technical equipment, e.g. a state of the art scanning/tunnelling electron microscope. In addition, Intel material [13] states that the entire RNG shuts down when it detects that the physical operating environment (e.g. temperature) has moved outside of an allowed range.

We are sympathetic to this argument, but still find value in adopting the strong attack model. Even if the entropy source is beyond attacker influence, treating it as adversarial allows us emphasize the minimal assumptions required of it. Moreover, the model allows us to explore the limits of ISK-RNG's security, providing analysis of less pessimistic settings (i.e. resilience security) as a byproduct.

Summary of contributions. We make important refinements to the DPRVW PWI model, bringing it more in line with real-world constructions. Specifically, we generalize the model to include PWIs with non-uniform state, explicitly model initialization procedures (permitting us to formulate appropriate security notions), and allow for both multiple interfaces and blocking. We then map the ISK-RNG into the new model, and give the first provable-security analysis its properties. From a technical perspective, our analysis requires a careful consideration of ISK-RNG's conditioner, so as to avoid pitfalls that prevent straight-forward application of existing results from the entropy-extraction literature. Our analysis exposes ISK-RNG's lack of backward security, and provides a nuanced discussion of ISK-RNG's forward security.

2 Preliminaries

Notation. We denote the set of all n-bit strings as $\{0,1\}^n$, and the set of all (finite) binary strings as $\{0,1\}^*$. Given $x, y \in \{0,1\}^*$, both xy and $x \,\|\, y$ denote their concatenation, and $|x|$ is the length of x. If $|x| = |y|$, $x \oplus y$ is the bitwise XOR of x and y. The symbol ε denotes the empty string. The set $\mathrm{Perm}\,(n)$ denotes the set of permutations on $\{0,1\}^n$.

When S is a finite set, we assume that it is equipped with the uniform distribution unless otherwise specified. For any distribution $\mathcal{S}$, the notation $X \xleftarrow{\$} \mathcal{S}$ indicates X is a random variable sampled from $\mathcal{S}$. Similarly, if F is a randomized algorithm, $X \xleftarrow{\$} \mathsf{F}(x_1, \ldots, x_n)$ means that X is sampled from the distribution induced by providing F with the indicated arguments. An adversary A is a randomized algorithm, and we adopt the shorthand $A \Rightarrow y$ to mean that when its execution halts, it outputs y. When an algorithm P is provided oracle (black-box, unit-time) access to an algorithm Q, we write P^Q.

Entropy and Sources. If X and X' are random variables, then the statistical distance between them is $\Delta(X, X') = \frac{1}{2}\sum_x |\Pr[\, X = x\,] - \Pr[\, X' = x\,]|$, where the sum is over the union of the supports of X and X'. The min-entropy of X is $\mathbf{H}_\infty(X) = -\max_x (\log \Pr[\, X = x\,])$, and the worst-case min-entropy of X conditioned X' is $\mathbf{H}_\infty(X \mid X') = -\log\left(\max_{x,x'} \Pr[\, X = x \mid X' = x'\,]\right)$. When X is a random variable and $\mathcal{E}$ is some event, we denote by $X|_{\mathcal{E}}$ the random variable X conditioned on $\mathcal{E}$; i.e., for any x in the support of X, $\Pr[\, X|_{\mathcal{E}} = x\,] = \Pr[\, X = x \mid \mathcal{E}\,]$.

An *entropy source* $\mathcal{D}$ is a randomized algorithm that, on input a state string $\sigma \in \{0,1\}^*$, samples a tuple $(\sigma', I, \gamma, z) \in \{0,1\}^* \times \{0,1\}^p \times \mathbb{R}_{\geq 0} \times \{0,1\}^*$. Let $(\sigma_i, I_i, \gamma_i, z_i) \xleftarrow{\$} \mathcal{D}(\sigma_{i-1})$ be a sequence of samples, where $\sigma_0 = \varepsilon$, and $i = 1, \ldots, q_{\mathcal{D}}$ for some integer $q_{\mathcal{D}}$. We say that entropy source $\mathcal{D}$ is *legitimate* if $H_\infty(I_j \mid (I_i, z_i, \gamma_i)_{i \neq j}) \geq \gamma_j$. In this paper, we assume all entropy sources are legitimate.

In this definition, $\sigma, \sigma' \in \{0,1\}^*$ represent the current and new states for $\mathcal{D}$, respectively. The string $I \in \{0,1\}^p$ is what will be to be fed as input to the PWI, and should provide fresh entropy. The quantity $\gamma \in \mathbb{R}_{\geq 0}$ is an estimate for the amount of entropy contained in I. We note that γ is strictly a convenient book-keeping device in our PWI model, and is not intended to reflect an actual output of the entropy source being modeled. Our security notions will formalize attacker capabilities of interest, but we also allow for side-information (about I) that an attacker might obtain through means not explicit in the notions model (e.g. timing or power side-channels). This side information will be captured in the string z.

Cryptographic building blocks. A blockcipher is a function $E : \{0,1\}^\kappa \times \{0,1\}^n \to \{0,1\}^n$ such that for each key $K \in \{0,1\}^\kappa$, $E(K, \cdot)$, written $E_K(\cdot)$, is a permutation on $\{0,1\}^n$. Given $\mathsf{IV} \in \{0,1\}^n$, $K \in \{0,1\}^\kappa$, and $X_i \in \{0,1\}^n$ for $i \in [0..\nu]$, define

$$\mathsf{CTR}_K^{\mathsf{IV}}(X_0 \cdots X_\nu) = (X_0 \oplus E_K(\mathsf{IV})) \,\|\, \cdots \,\|\, (X_\nu \oplus E_K(\mathsf{IV} + \nu)).$$

(We define the $+$ operator on $\{0,1\}^n$ as addition modulo 2^n on the unsigned integers encoded by the operands.) Further define $\mathsf{CBCMAC}_K^{\mathsf{IV}}(X_0 \cdots X_\nu) = \mathsf{CBCMAC}_K^{E_K(\mathsf{IV} \oplus X_0)}(X_1 \cdots X_\nu)$, and $\mathsf{CBCMAC}_K^{\mathsf{IV}}(\varepsilon) = \mathsf{IV}$. Describing the standard CBCMAC algorithm in this manner simplifies descriptions of programs that compute CBCMAC online. We omit an explicit IV from the notation when $\mathsf{IV} = 0^n$. In this paper, the implicit blockcipher E will always be AES-128 ($\kappa = n = 128$).

The pseudorandom-permutation (PRP) advantage of an adversary A attacking a blockcipher $E : \{0,1\}^\ell \times \{0,1\}^n \to \{0,1\}^n$ is defined as

$$\mathbf{Adv}_E^{\mathrm{prp}}(A) = \Pr\left[\, K \xleftarrow{\$} \{0,1\}^\ell :\ A^{E_K} \Rightarrow 1 \,\right] - \Pr\left[\, \pi \xleftarrow{\$} \mathrm{Perm}(n) :\ A^\pi \Rightarrow 1 \,\right].$$

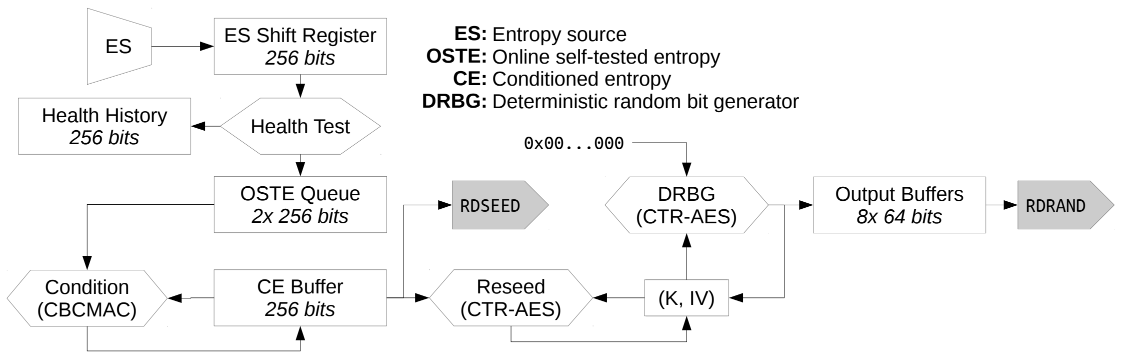

Fig. 2. Block diagram for Intel's `RDRAND` implementation. The CBCMAC computation uses AES-128 with the fixed key $K' = \mathsf{AES}_{\mathbf{0}}(1)$. The DRBG runs AES-128 in counter mode to produce $\{0,1\}^{128 \cdot 3}$ bits of output; the first 256 bits are used to update the key K and IV; the final 128 bits are sent to the output buffer, which is read by the `RDRAND` instruction.

3 The ISK-RNG architecture

This section describes the design of the ISK-RNG. Unless otherwise noted, this information comes from the CRI report [7]. The design can be divided roughly into three phases: entropy generation, entropy extraction, and expansion. Raw bits from the generation phase are fed into an entropy extractor, which is tasked with turning biased or correlated bits into uniform random strings. The expansion step uses these strings to seed a deterministic PRNG, which can produce cryptographically pseudorandom outputs at high speeds.

The design is shown in Figure 2. In this figure, rectangular boxes indicate we consider part of the ISK-RNG state, hexagons indicate procedures that read and modify the state, and the shaded arrows indicate assembly instructions that allow (unprivileged) processes to read from designated buffers.

Entropy generation, Health Tests, and "Swellness". The hardware entropy source (labeled ES) is a dual differential jamb latch with feedback; thermal noise resolves a latch formed by two cross-coupled inverters, generating a random bit before the system is reset. Bits from the entropy source are written into a 256-bit shift register.

Every 256 writes, the contents of the register are subjected to a series of health tests. These count how many times certain specified bit strings appear, and verifies that the resulting counts are within normal limits. For example, the string `010` may occur between 9 and 57 times, inclusive. If the current ES register fails one of the tests, that 256-bit source-sample is flagged as unhealthy. (For reference, a uniformly random 256-bit string would be flagged as unhealthy approximately 1% of the time.) We refer interested readers to the CRI report [7] for a more detailed description; for our purposes, it suffices to say there is some fixed set $\mathcal{H} \subseteq \{0,1\}^{256}$ of strings that pass the health tests. The health-history register tracks how many of the last 256 samples passed the health test. This is a first-in first-out buffer, where a 1-bit means that a sample was deemed healthy, and a 0-bit mean that a sample was deemed unhealthy. The global health of the ISK-RNG is captured by a property call *swellness*.

Definition 1 (Swell ISK-RNG). *The ISK-RNG is said to be* swell *if at least 128 of the last 256 samples were healthy, i.e. if the health-history register contains at least 128 1s.* □

Whether or not the current sample passes the health test, it is appended to the Online Self-Tested Entropy (OSTE) queue, and it is the OSTE queue that provides input to the extraction phase.

Extraction. Strings in the OSTE queue are not assumed to be uniformly random. Instead, each 256-bit entry is assumed to have a certain amount of min-entropy. The CBCMAC construction, over AES with key $K' = \mathsf{AES}_{\mathbf{0}}(1)$ [10], is employed as an entropy extractor, in order to turn strings in the OSTE queue into two 128-bit *conditioned entropy* (CE) strings. These are held in the CE buffer, which is initially all zeros, and are used to service `RDSEED` instructions and to reseed the DRBG. An important property of the CE buffer is its *availability*.

Definition 2 (Available CE). *The CE buffer is* available *if (1) the ISK-RNG is swell, and (2) since the last* `RDSEED` *call or the last DRBG reseeding (whichever was most recent), both 128-bit halves (*CE_0 *and* CE_1*) have been updated using* m *healthy OSTE values. For Ivy Bridge chips,* $m = 2$*; for Broadwell chips,* $m = 3$ *[10].* □

When the CE buffer is not available, the hardware will replenish the OSTE buffers with fresh entropy and feed them into a running CBCMAC calculation until a sufficient number of healthy samples have been conditioned. So if at some point $\mathsf{CE}_0 = X$ and then the CE buffer is used to service a `RDSEED` instruction (making the CE buffer unavailable), the hardware will collect entropy strings $I_1, I_2, I_3, \ldots \in \{0,1\}^{256}$ and reassign $\mathsf{CE}_0 \leftarrow \mathsf{CBCMAC}_{\mathbf{0}}(X I_1 I_2 I_3 \cdots)$ online until there exist $i_1 < i_2 < \cdots < i_m$ such that $I_{i_j} \in \mathcal{H}$ for $j \in [1..m]$ *and* the ISK-RNG is swell. Then the processes will repeat for CE_1.

The particulars of the way CBCMAC is used in the ISK-RNG extractor, and the notions of swellness and availability, will play a large part in Section 4.

Expansion. To reseed the DRBG, the contents of CE_0 and CE_1 are used to generate a key and IV (respectively) for counter mode encryption over AES. This reseeding process only happens when the CE buffer is available. It takes the current key and IV, (K, IV), and updates them by computing $K \parallel \mathsf{IV} \leftarrow \mathsf{CTR}_K^{\mathsf{IV}}(\mathsf{CE}_1 \parallel \mathsf{CE}_2)$. Initially, $K = \mathsf{IV} = 0^{128}$. However, using CTR with this non-random key is not a problem as long as the CE buffer is (close to) uniformly random: since the CE buffer is XORed into the CTR keystream, it can act as a one-time pad.

A pseudorandom value R is generated by computing $R \parallel K \parallel \mathsf{IV} \leftarrow \mathsf{CTR}_K^{\mathsf{IV}}(0^{3 \cdot 128})$. (Note that this process also irreversibly updates K and IV, which helps provide forward security.) The RNG writes R to an output buffer, which is read by `RDRAND`. The FIFO output buffer [8] can contain up to eight 64-bit values. ISK-RNG allows a maximum of 511 64-bit values to be generated between reseeding operations; after this, it will only return 0s and will clear the carry bit to signal an error.

Setup. When the ISK-RNG powers on, the ISK-RNG performs a series of known-answer, built-in self-tests. Then the conditioned entropy (CE) buffer is cleared and the deterministic random bit generator (DRBG) is reseeded four times [10]. Each reseeding operation requires reconditioning the CE buffer until it is available. Finally, the system populates the eight output buffers using the DRBG.

Standards compliance. Intel states [12] that ISK-RNG is compliant with NIST's SP800-90B & C draft standards. Whereas `RDRAND` can provide bit strings with "only" a 128-bit security level (since it uses AES-128 in CTR mode), `RDSEED` has no such limitation.

4 Analysis of the ISK-RNG extractor

As we will see, the PWI-security results for the ISK-RNG are not as strong as one might hope. Much of this is due to weak concrete bounds on the entropy extractor. Let us explain.

Previous CBC-MAC results are not (directly) helpful. A paper by Dodis, Gennaro, Håstad, Krawcyzk and Rabin [4] analyzes the security of CBC-MAC as an entropy extractor, and their results are cited by Intel documents [13] to support the ISK-RNG design. Because generic PRFs-as-entropy-extractors results [3] are too weak to be useful, the analysis of [4] takes place in the random permutation model. That is, instead of considering CBC-MAC over a blockcipher with a random key, they consider CBC-MAC over a random permutation. This model is a heuristic: even, say, AES equipped with a random key would not be a random permutation. In fact, CBC-MAC within the ISK-RNG uses AES with a fixed key $K' = \mathsf{AES}_0(1)$ (on all chips). This fact may strike one as alarming. But we believe that a "nothing-up-my-sleeve" value for the extractor seed is a reasonable choice. (Generating the seed from the entropy source would be highly suspect from a theoretical perspective, because one requires that the extractor seed be "independent" of the entropy distribution.)

Anyway, our primary goal here is to identify what we *can* say about ISK, even if we're forced to use a heuristic model. Dodis et al.[4] provide the following theorem:

Theorem 1 (CBCMAC entropy extractor [4]). *Fix positive integers k and L. Let $I \in \{0,1\}^{Lk}$ be a random variable, $R \xleftarrow{\$} \{0,1\}^k$ be a uniform random string, and let $\pi \xleftarrow{\$} Perm(k)$ be a random permutation. Then $\Delta((\pi, R), (\pi, \mathsf{CBCMAC}_\pi(I)) \le \frac{1}{2}\sqrt{2^{k-\mathbf{H}_\infty(I)} + \frac{\mathcal{O}(L^2)}{2^k}}$.*

Unfortunately, one cannot simply apply this theorem to the CBC-MAC-based extractor used in ISK-RNG, without attending to the following two significant obstacles:

(1) As we noted in Section 3, the CBCMAC-based extractor uses its own previous output as the first block of its next input. Consequently, the CBCMAC inputs are not independent of the seed. This is necessary for leftover-hash-lemma style results like Theorem 1, and furthermore prevents us from employing a black-box hybrid argument to lift the results to the multiple-query setting.

(2) The $\mathcal{O}(L^2)$ term is particularly problematic. It contributes a $\mathcal{O}(L/2^{k/2})$ term to the final result.[1] We note that this is signficantly worse than the familiar $\mathcal{O}(L^2/2^k)$ "birthday bound" — although the two both become vacuous when $L \approx 2^{k/2}$, the former violates a desired security level $\epsilon \ll 1$ much sooner (hidden constants being equal). The weak bound is exaccerbated by the fact that L may grow very quickly in the ISK-RNG during periods of time when the CE buffer is not available.

Analyzing the CBC-MAC extractor. In this section we present results that allow us to overcome these hurdles, bringing Theorem 1 into scope. In particular, the main technical result of this section is the following theorem. Loosely, it says that we can still obtain a hybrid-like bound, even though a black-box hybrid argument isn't possible. Moreover, we can avoid "runaway" input strings (resulting in large L) by, in effect, truncating them.

[1] A set of slides published by Intel [13] claims a much stronger result based on Theorem 1. However, in addition to failing to account for point (1) above, the difference appears to stem from a mistake in translating notation. Specifically, the above theorem from [4] writes the second term under the radical as $K \cdot \epsilon(L, K)$, where $\epsilon(L, K) = \mathcal{O}(L^2/K^2)$ and in our notation $K = 2^k$. The Intel slides, however, appear to have mistranscribed this term as $L \cdot \epsilon(L, K)$ (in their notation, $L = b$ and $K = 2^n$). Since $L \ll K$ for values of interest, Intel's claim significantly overestimates the concrete security bound.

Theorem 2. *Fix positive integers L, q and k. For $i \in [1..q]$, let $I_i \in \{0,1\}^*$ be random variables with length divisible by k, and sample $R_i \xleftarrow{\$} \{0,1\}^k$. Fix $\pi \xleftarrow{\$} \mathit{Perm}(k)$. Define I_i^{L} and I_i^{R} to be the unique strings such that $\left|I_i^{\mathsf{L}}\right| = \min\{|I_i|, Lk\}$ and $I_i = I_i^{\mathsf{L}} I_i^{\mathsf{R}}$. Let $C_i = \mathsf{CBCMAC}_\pi(C_{i-1} \,\|\, I_i)$, where $C_0 = 0^k$. Then:*

$$\Delta((\pi, C_1, \dots, C_q), (\pi, R_1, \dots, R_q)) \leq \frac{1}{2} \sum_{i=1}^{q} \sqrt{2^{k - \mathbf{H}_\infty\left(I_i^{\mathsf{L}} \mid (I_j)_{j>i}, I_i^{\mathsf{R}}\right)} + \frac{\mathcal{O}\left((L+1)^2\right)}{2^k}}.$$

Proof. Setting $R_0 = 0^k$, define:

$$\delta_i = \Delta((\pi, R_1, \dots, R_{i-1}, C_i^i, C_{i+1}^i, \dots, C_q^i), (\pi, R_1, \dots, R_{i-1}, R_i, C_{i+1}^i, \dots, C_q^i)).$$

where $C_j^i = \mathsf{CBCMAC}_\pi(R_{i-1} \,\|\, I_i)$ if $j = i$, and $C_j^i = \mathsf{CBCMAC}_\pi(C_{j-1}^i \,\|\, I_j)$ if $j > i$. As a consequence, we have $\Delta((\pi, C_1, \dots, C_q), (\pi, R_1, \dots, R_q)) \leq \sum_{i=1}^q \delta_i$. Since each R_i is independent and uniform and C_j^i can be computed from π, R_{i-1}, and $I_{>i} = (I_i^{\mathsf{R}}, I_{i+1}, I_{i+2}, \dots, I_q)$ whenever $j > i$,

$$\begin{aligned} \delta_i &\leq \Delta((\pi, R_{i-1}, R_i, I_{>i}), (\pi, R_{i-1}, C_i^i, I_{>i})) \\ &\leq \sum_{r,s} \Delta((\pi, r, R_i, s), (\pi, r, C_i^i\big|_{(R_{i-1}, I_{>i}) = (r,s)}, s)) \cdot \Pr[\, R_{i-1} = r \,] \Pr[\, I_{>i} = s \,]. \end{aligned}$$

Let $\mathcal{P}$ be the event $(R_{i-1} = r) \wedge (I_{>i} = s)$; then $C_i^i\big|_{\mathcal{P}} = \mathsf{CBCMAC}_\pi(r \,\|\, I_i^{\mathsf{L}}\big|_{\mathcal{P}} \,\|\, c^{\mathsf{R}})$, where c^{R} is the first component of s, corresponding to the (now fixed) value of I_i^{R}. Note that $I_i|_{\mathcal{P}}$ is distributed identically to $I_i|_{I_{>i}=s}$. Define $R_i' = \mathsf{CBCMAC}_\pi(r \,\|\, I_i^{\mathsf{L}}\big|_{\mathcal{P}})$. This now brings the results of Theorem 1 into scope:

$$\Delta((\pi, r, R_i, s), (\pi, r, R_i', s)) \leq \frac{1}{2} \sqrt{2^{k - \mathbf{H}_\infty\left(I_i^{\mathsf{L}} \mid I_{>i}\right)} + \frac{\mathcal{O}((L+1)^2)}{2^k}}.$$

Now, $C_i^i\big|_{\mathcal{P}} = \mathsf{CBCMAC}_\pi^{R'}(c^{\mathsf{R}})$. However, given that c^{R} is fixed, $\tau_{\pi, c^{\mathsf{R}}}(\cdot) = \mathsf{CBCMAC}_\pi^{(\cdot)}(c^{\mathsf{R}})$ is a permutation for every possible π. Therefore since R_i is uniformly distributed,

$$\begin{aligned} \Delta((\pi, r, R_i, s), (\pi, r, C_i^i\big|_{\mathcal{P}}, s)) &= \frac{1}{2} \sum_{(\rho,x)} \left|\Pr[\, (\pi, R_i) = (\rho, x) \,] - \Pr\left[\, (\pi, C_i^i\big|_{\mathcal{P}}) = (\rho, x) \,\right]\right| \\ &= \frac{1}{2} \sum_{(\rho,x)} \left| \frac{1}{2^k} - \Pr\left[\, C_i^i\big|_{\mathcal{P}} = x \mid \pi = \rho \,\right] \right| \Pr[\, \pi = \rho \,] \\ &= \frac{1}{2} \sum_{(\rho,x)} \left| \frac{1}{2^k} - \Pr\left[\, R' = \tau_{\rho, c^{\mathsf{R}}}^{-1}(x) \mid \pi = \rho \,\right] \right| \Pr[\, \pi = \rho \,] \\ &= \frac{1}{2} \sum_{(\rho,x)} \left|\Pr[\, (\pi, R_i) = (\rho, x) \,] - \Pr\left[\, (\pi, R_i') = (\rho, x) \,\right]\right| \\ &= \Delta((\pi, r, R_i, s), (\pi, r, R_i', s)), \end{aligned}$$

with summations over $(\rho, x) \in \mathrm{Perm}(k) \times \{0,1\}^k$. This completes the proof. □

It remains to show that, with high probability, the (potentially) truncated extractor input contains sufficient min-entropy. Note that making reasonable min-entropy assumptions regarding the entropy source is not sufficient to conclude this; for example, the approximate 1% false-positive rate of the health tests on uniformly random 256-bit strings implies that there are at least 2^{249} *unhealthy* strings. Therefore the entropy source could produce *only* unhealthy samples, resulting in unbounded L, and still have high min-entropy. In order to avoid such pathological behavior, we will later (in Section 7.2) need to introduce additional assumptions regarding the rate at which the entropy source produces healthy samples. Ultimately, we will choose L such that we have a high probability of never needing more that $L/2$ samples, but such that $L/2^{k/2}$ is small, as this term will dominate Theorem 2.

5 Modeling the ISK-RNG as a PWI

Building upon DPRVW, here we define the syntax of a PWI. We give the syntax first, and then discuss what it captures, pointing out where our definition differs from DPRVW.

Definition 3 (PWI). *Let n, p, and ℓ be non-negative integers, and let* $\mathrm{IFace}, \mathrm{Seed}, \mathrm{State}$ *be the non-empty sets. A* PRNG with input *(PWI) with interface set* IFace*, seed space* Seed*, and state space* State *is a tuple of deterministic algorithms* $\mathcal{G} = (\mathsf{setup}, \mathsf{refresh}, \mathsf{next}, \mathsf{tick})$*, where*

- setup *takes no input, and generates an initial PWI state* $S_0 \in \mathrm{State}$*. Although* setup *itself is deterministic, it may be provided oracle access to an entropy source* $\mathcal{D}$*, in which case its output* S_0 *will be a random variable determined by the coins of* $\mathcal{D}$*.*
- $\mathsf{refresh} : \mathrm{Seed} \times \mathrm{State} \times \{0,1\}^p \to \mathrm{State}$ *takes a seed* $\mathsf{seed} \in \mathrm{Seed}$*, the current PWI state* $S \in \mathrm{State}$*, and string* $I \in \{0,1\}^p$ *as input, and a returns new state.*
- $\mathsf{next} : \mathrm{Seed} \times \mathrm{IFace} \times \mathrm{State} \to \mathrm{State} \times (\{0,1\}^\ell \cup \{\perp\})$ *takes a seed, the current state, and an interface label* $m \in \mathrm{IFace}$*, and returns a new state, and either ℓ-bit output value or a distinguished, non-string symbol* $\perp$*.*
- $\mathsf{tick} : \mathrm{Seed} \times \mathrm{State} \to \mathrm{State}$ *takes a seed and the current state as input, and returns a new state.*

$\square$

We will typically omit explicit mention of the the seed argument to $\mathsf{refresh}$, next and tick, unless it is needed for clarity

The setup algorithm captures the initialization of the PWI, in particular its internal state. Unlike DPRVW, whose syntax requires setup to generate the PWI seed, we view the seed as something generated externally and provided to the PWI. Permitting an explicit setup procedure is necessary to correctly model ISK-RNG and, more generally, allows us to formulate an appropriate security definition for PWI initialization.

The $\mathsf{refresh}$ algorithm captures the incorporation of new entropy into the PWI state. Like DPRVW, we treat the entropy source as external. This provides a clean and general way to model the source as untrusted to provide consistent, high-entropy outputs.

Our next algorithm captures the interface exposed to (potentially adversarial) parties that request PWI outputs. By embellishing the DPRVW syntax for next with the interface set $\mathsf{interface}$, we model APIs that expose multiple functionalities that access PWI state. This is certainly the case for the ISK-RNG, via the `RDRAND` and `RDSEED` instructions. We also explicitly allow next to return $\perp$, capturing blocking behavior.

The tick algorithm is entirely new, and requires some explanation. In the security notions formalized by DPRVW, the passage of "time" is implicitly driven by adversarial queries. (This is

Variable	Bits	Description
ESSR	256	Entropy source shift register
window	8	Counts new bits in the ESSR
OSTE_1	256	Online self-tested entropy buffers
OSTE_2	256	
CE_1	128	Conditioned entropy buffers
CE_2	128	
ptr	1	Tracks CE buffer to condition next
health	256	Tracks health of last 256 ES samples
K	128	DRBG key (For AES-CTR)
IV	128	DRBG IV (For AES-CTR)
$\mathsf{out}_{1,\dots,8}$	512	Eight 64-bit output buffers
$\mathsf{outcount}$	≥ 4	Counts number of full output buffers
count	≥ 9	Counts DRBG calls since reseeding
CEfull	1	Set if CE buffers are available
block	1	Set if reseed has priority over `RDSEED`

Table 1. Table describing the state of Intel's HWRNG.

typical for security notions, in general.) But real PRNGs like the ISK-RNG may have behaviors that update the state in ways that are not cleanly captured by an execution model that is driven by entropy-input events (refresh calls), or output-request events (next calls). The tick algorithm handles this, while allowing our upcoming security notions to retain the tradition of being driven by adversarial queries: the adversary will be allowed to "query" the tick oracle, causing one unit of time to pass and state changes to occur.

Mapping the ISK-RNG to the PWI syntax. We now turn our attention to mapping the ISK-RNG specification into the PWI model. Table 1 summarizes the state that our model tracks. Figure 3 provides our model for the PWI setup, refresh, next, and tick oracles. Two additional procedures, DRBG and reseed, are used internally.

6 PWI Security

We now turn our attention to security notions for PWIs. The basic notions are those of DPRVW, with a few notable alterations. However, to handle issues of non-uniform state and (more) realistic initialization procedures, we introduce a new technical tool – masking functions. We will see that our formalization of masking functions simultaneously provides a clean way to deal with the issues we just mentioned, and allows us to leverage the definitional results from DPRVW.

6.1 Basic notions

Here we define four PWI-security notions, in the game-playing framework [2]. In each there is a (potentially adversarial) entropy source $\mathcal{D}$, and an adversary A. The latter is provided access to the oracles detailed in Figure 4 (top), and what distinguishes the four notions are restrictions applied to the queries of the adversary A. In particular, we consider the following games:

Oracle setup(ES):

```
for $i = 1, 2, 3, 4$ do
    $S.\mathsf{CE}_0 \leftarrow \mathsf{CBCMAC}_{K'}(S.\mathsf{CE}_0)$
    while $S.\mathsf{ptr} = 0$ do
        $I \xleftarrow{\$} \mathsf{ES}$
        $S \leftarrow \mathsf{refresh}(S, I)$
    $S.\mathsf{CE}_1 \leftarrow \mathsf{CBCMAC}_{K'}(S.\mathsf{CE}_1)$
    while $S.\mathsf{ptr} = 1$ do
        $I \xleftarrow{\$} \mathsf{ES}$
        $S \leftarrow \mathsf{refresh}(S, I)$
    $S \leftarrow \mathsf{reseed}(S)$
for $i = 1, 3, 5, 7$ do
    $(S, R) \leftarrow \mathsf{DRBG}(S)$
    $S.\mathsf{out}_i \parallel S.\mathsf{out}_{i+1} \leftarrow R$
$S.\mathsf{outcount} \leftarrow 8$
return $S$
```

Oracle DRBG(S):

```
$S.\mathsf{IV} \leftarrow S.\mathsf{IV} + 1$
$R \leftarrow \mathsf{CTR}_K^V(0^{128})$
if $S.\mathsf{CEfull}$ then
    $S \leftarrow \mathsf{reseed}(S)$
else if $S.\mathsf{count} < 512$
    $S.K \parallel S.V \leftarrow \mathsf{CTR}_{S.K}^{S.V+1}(0^{256})$
    $S.\mathsf{count} \leftarrow S.\mathsf{count} + 1$
else
    return $(S, \perp)$
return $(S, R)$
```

Oracle tick(S):

```
if $S.\mathsf{CEfull}$ and $S.\mathsf{count} > 0$ then
    $S \leftarrow \mathsf{reseed}(S)$
    return $S$
if $S.\mathsf{count} < 512$ and $S.\mathsf{outcount} < 8$ then
    $S.\mathsf{outcount} \leftarrow S.\mathsf{outcount} + 1$
    $(S, R) = \mathsf{DRBG}(S)$
    $S.\mathsf{out}_{\mathsf{outcount}} \leftarrow R$
    return $S$
return $S$
```

Oracle refresh(S, I):

```
$S.\mathsf{ESSR} \leftarrow \mathsf{shift}(S.\mathsf{ESSR}, I)$
$S.\mathsf{window} \leftarrow S.\mathsf{window} + 1 \bmod 256$
if $S.\mathsf{window} = 0$ then
    $S.\mathsf{health} \leftarrow \mathsf{shift}(S.\mathsf{health}, \mathsf{isHealthy}(S.\mathsf{ESSR}))$
    $S.\mathsf{OSTE}_2 \leftarrow S.\mathsf{OSTE}_1$
    $S.\mathsf{OSTE}_1 \leftarrow S.\mathsf{ESSR}$
    $i \leftarrow S.\mathsf{ptr}$
    $I_j^i \leftarrow I_j^i \parallel S.\mathsf{OSTE}_2$ // Record-keeping for proofs
    $S.\mathsf{CE}_i \leftarrow \mathsf{CBCMAC}_{K'}^{S.\mathsf{CE}_i}(\mathsf{OSTE}_2)$
    if $\mathsf{sum}(S.\mathsf{health}) \geq 128$ and $\mathsf{isHealthy}(\mathsf{OSTE}_2)$ then
        $S.\mathsf{samples} \leftarrow S.\mathsf{samples} + 1$
    if $S.\mathsf{samples} = m$ then
        $S.\mathsf{samples} \leftarrow 0$
        if $S.\mathsf{ptr} = 0$ then
            $S.\mathsf{ptr} \leftarrow 1$
        else
            $S.\mathsf{ptr} \leftarrow 0$; $S.\mathsf{CEfull} \leftarrow 1$
            $C_j^0 \parallel C_j^1 \leftarrow S.\mathsf{CE}$ // Record-keeping for proofs
            $j \leftarrow j + 1$; // Record-keeping for proofs
return $S$
```

Oracle reseed(S):

```
$S.K \parallel S.V \leftarrow \mathsf{CTR}_K^{V+1}(S.\mathsf{CE})$
$S.\mathsf{CE}_0 \leftarrow \mathsf{CBCMAC}_{K'}(S.\mathsf{CE}_0)$
$S.\mathsf{CE}_1 \leftarrow \mathsf{CBCMAC}_{K'}(S.\mathsf{CE}_1)$
$S.\mathsf{count} \leftarrow 0$; $S.\mathsf{CEfull} \leftarrow 0$; $S.\mathsf{ptr} \leftarrow 0$; $S.\mathsf{block} \leftarrow 0$
return $S$
```

Oracle next(interface, S):

```
if interface = RDRAND then
    if $S.\mathsf{outcount} = 0$ then return $(S, \perp)$
    $R \leftarrow \mathsf{LSB}_{64}(S.\mathsf{out}_1)$
    for $i = 1, \ldots, 7$ do
        $S.\mathsf{out}_i \leftarrow S.\mathsf{out}_{i+1}$
    $S.\mathsf{outcount} \leftarrow S.\mathsf{outcount} - 1$
    return $(S, R)$
else if interface = RDSEED
    if $S.\mathsf{CEfull} = 0$ or ($S.\mathsf{block} = 1$ and $S.\mathsf{count} > 0$) then
        return $(S, \perp)$
    $R \leftarrow S.\mathsf{CE}_0 \parallel S.\mathsf{CE}_1$
    $S.\mathsf{CEfull} \leftarrow 0$; $S.\mathsf{ptr} \leftarrow 0$
    $S.\mathsf{CE}_0 \leftarrow \mathsf{CBCMAC}_{K'}(S.\mathsf{CE}_0)$
    $S.\mathsf{CE}_1 \leftarrow \mathsf{CBCMAC}_{K'}(S.\mathsf{CE}_1)$
    $S.\mathsf{block} \leftarrow 1$
    return $(S, R)$
```

Fig. 3. The above oracles describe the behavior of ISK-RNG from within the PWI model. See Table 1 for a description of the state variables $S.*$. All bits are initially zero. For Ivy Bridge chips, $m = 2$, and for Broadwell chips $m = 3$. The key $K' = \mathsf{AES}_{\mathbf{0}}(1)$ is fixed across all chips.

```
Oracle $\mathcal{D}$-refresh:
    $(\sigma, I, \gamma, z) \xleftarrow{\$} \mathcal{D}(\sigma)$
    $S \leftarrow \mathsf{refresh}(S, I)$
    $c \leftarrow c + \gamma$
    if $c \geq \gamma^*$ then
        corrupt $\leftarrow$ false
    return $(\gamma, z)$
```

```
Oracle next-ror$(m)$:
    if corrupt then
        return $\perp$
    $(S, R_0) \leftarrow \mathsf{next}(m, S)$
    if $R_0 = \perp$ then
        $R_1 \leftarrow \perp$
    else
        $R_1 \xleftarrow{\$} \{0,1\}^\ell$
    return $R_b$
```

```
Oracle get-next$(m)$:
    $(S, R) \leftarrow \mathsf{next}(m, S)$
    if corrupt then
        $c \leftarrow 0$
    return $R$
```

```
Oracle wait:
    $S \leftarrow \mathsf{tick}(S)$
    return $\varepsilon$
```

```
Oracle get-state:
    $c \leftarrow 0$
    corrupt $\leftarrow$ true
    return $S$
```

```
Oracle set-state$(S^*)$:
    $c \leftarrow 0$
    corrupt $\leftarrow$ true
    $S \leftarrow S^*$
```

```
Procedure initialize:
    $\sigma \leftarrow 0$; seed $\xleftarrow{\$}$ Seed; $i \leftarrow 0$
    $S \leftarrow \mathsf{setup}^{\mathsf{ES}}$
    $c \leftarrow n$; corrupt $\leftarrow$ false
    $b \xleftarrow{\$} \{0,1\}$
    return (seed, $(\gamma_j, z_j)_{j=1}^{i}$)
```

```
Oracle ES:
    $i \leftarrow i + 1$
    $(\sigma, I, \gamma_i, z_i) \xleftarrow{\$} \mathcal{D}(\sigma)$
    return $I$
```

```
Procedure finalize$(b)$:
    if $b = b^*$ then
        return 1
    else
        return 0
```

Fig. 4. Top: Oracles for the PWI security games. **Bottom:** the shared intialize and finalize procedures for the PWI security games. Recall that the output of initialize is provided to adversary A as input, and the output of finalize is the output of the game.

Robustness (ROB): no restrictions on queries.

Forward security (FWD): no queries to set-state are allowed; and a single query to get-state is allowed, and this must be the final query.

Backward security (BWD): no queries to get-state are allowed; a single query to set-state is allowed, and this must be the first query.

Resilience (RES): no queries to get-state or set-state are allowed.

See DPRVW for additional discussion. We note that all games share common initialize and finalize procedures, shown in Figure 4 (bottom). Thus, the robustness-advantage of A in attacking $\mathcal{G}$ is defined to be $\mathbf{Adv}^{\mathrm{rob}}_{\mathcal{G},\mathcal{D}}(A) = 2\Pr[\,\mathsf{ROB}_{\mathcal{G},\mathcal{D}}(A) = 1\,] - 1$. The forward security, backward security, and resilience advantages $\mathbf{Adv}^{\mathrm{fwd}}_{\mathcal{G},\mathcal{D}}(A)$, $\mathbf{Adv}^{\mathrm{bwd}}_{\mathcal{G},\mathcal{D}}(A)$, and $\mathbf{Adv}^{\mathrm{res}}_{\mathcal{G},\mathcal{D}}(A)$ are similarly defined. It is clear that robustness implies forwards and backwards security, and both of these independently imply resilience.

We note that, because the PRNG cannot reasonably be expected to produce random-looking outputs without sufficient entropy or with a known or corrupted state, the various security experiments track (1) a boolean variable corrupt and (2) a value γ measuring the total entropy that has been fed into the PRNG since corrupt was last set. These serve as book-keeping devices to prevent trivial wins. The corrupt flag is cleared whenever γ exceeds some specified threshold γ^*.

6.2 Masking functions and Updated Notions

As noted earlier, the DPRVW security definitions assume the PWI state is initially uniformly random. However, this does not realistically model the behavior of real-world PWIs, notably ISK-RNG, which do not attempt to reach a pseudorandom state; for example, they may maintain counters. (Indeed one can construct PWIs that would be secure when starting from a uniformly random state, but that are insecure in practice because they are unable to ever reach a point where their state becomes pseudorandom; see Appendix A.) Yet, clearly, some portion of the PWI state must be unpredictable to an attacker, as otherwise one cannot expect PWI outputs to look random.

To better capture real-world characteristics of PWI state, we introduce the idea of a *masking function*. A masking function M over state space State $\mathcal{G}$ is a randomized algorithm from State to itself. As an example, if states consist of a counter c, a fixed identifier id, and an buffer B of (supposedly) entropic bits, then $M(c, \mathrm{id}, B)$ might be defined to return (c, id, B') where B' is sampled by M according so some distribution.

A masked state is meant to capture whatever is a "good" state of a PWI, i.e. after it has accumulated a sufficient amount of externally provided entropy. Informally, for any state S, we want that (1) a PWI with state $M(S)$ should produce pseudorandom outputs, and (2) after the PWI has gathered sufficient entropy, its state S should be indistinguishable from $M(S)$.

To the second point, the initial PWI state S_0 is of particular importance. In the following definition, we characterize masking functions M such that the initial S_0 and $M(S_0)$ are indistinguishable.

Definition 4 (Honest-initialization masking functions). Let $\mathcal{D}$ be an entropy source and let $\mathcal{G} = (\mathsf{setup}, \mathsf{refresh}, \mathsf{next})$ be a PWI with state space State, and let M : State $\to$ State be a masking function. Let (seed, Z) be the random variable returned by running the $\mathsf{initiailize}()$ procedure of Figure 4 (using $\mathcal{G}$ and $\mathcal{D}$), and let S_0 be the state produced by this procedure. Given an adversary A, define $\mathbf{Adv}^{\mathrm{init}}_{\mathcal{G},\mathcal{D},M}(A) = \Pr[\,A(S_0, \mathsf{seed}, Z) \Rightarrow 1\,] - \Pr[\,A(M(S_0), \mathsf{seed}, Z) \Rightarrow 1\,]$. If $\mathbf{Adv}^{\mathrm{init}}_{\mathcal{G},\mathcal{D},M}(A) \le \epsilon$ for any adversary A running in time t, then M is a $(\mathcal{G}, \mathcal{D}, t, \epsilon)$*-honest-initialization* masking function. $\square$

Note that the above definition is made with respect to a specific $\mathcal{D}$. The assumptions required of $\mathcal{D}$ (e.g., that it will provide a certain amount of entropy within a specified number of queries) will depend on the PWI in question, but should be as weak as possible.

We now defining "bootstrapped" versions of the PWI security goals, which always begin from a masked state. This will allow us to reason about security when the PWI starts from an "ideal" state, i.e. what we expect after an secure initialization of the system.

Definition 5 (Bootstrapped security). *Let $\mathcal{G}$ be a PWI and M be a masking function. For $x \in \{\mathrm{fwd}, \mathrm{bwd}, \mathrm{res}, \mathrm{rob}\}$, let $\mathbf{Adv}^{x/M}_{\mathcal{G},\mathcal{D}}(A)$ be defined as $\mathbf{Adv}^{x}_{\mathcal{G},\mathcal{D}}(A)$, except with line 02 of the* $\mathsf{initialize}$ *procedure (Fig. 4) changed, to execute instead $S' \xleftarrow{\$} \mathsf{setup}^{\mathsf{ES}}; S \xleftarrow{\$} M(S')$.* $\square$

6.3 PWI-Security Theorems

Bootstrapped security notions are useful, because they allow the analysis to begin with an idealized state. However, this comes at a cost: we need to ensure that the masking function is honest in the sense that it accurately reflects the result of running the setup procedure. The following theorem states the intuitive result that, if the masking function is secure (and honest), then security when the PWI begins in a masked state $M(S)$ implies security when the PWI begins in state S. We omit the simple proof, which follows from a standard reduction argument.

Theorem 3. *Let $\mathcal{G}$ be a PWI, $\mathcal{D}$ be an entropy source, and M be a masking function. Suppose M is a $(\mathcal{G}, \mathcal{D}, t, \epsilon)$-honest initialization mask. Then for any $x \in \{\mathrm{fwd}, \mathrm{bwd}, \mathrm{res}, \mathrm{rob}\}$ there exists some* *adversary $B(\cdot)$ such that for any adversary A, $\mathbf{Adv}^{x}_{\mathcal{G},\mathcal{D}}(A) \leq \mathbf{Adv}^{x/M}_{\mathcal{G},\mathcal{D}}(B(A)) + \epsilon$. Further, if it takes time t' to compute M, and A makes q queries and runs in time t, then $B(A)$ makes q queries and runs in time $\mathcal{O}(t) + t'$.*

For a second general result, we revisit a nice theorem by DPRVW and adapt it to our model. The theorem states that if a PWI possesses two weaker security properties — roughly, the ability to randomize a corrupted state after harvesting sufficient entropy and the ability to keep its state pseudorandom in the presence of adversarial entropy — then it is robust. These definitions, however, again assume that a state "should" appear uniformly random. We present modified definitions that instead use masking functions, and prove an analogous theorem. While the transition involves a couple subtleties — in particular, we require an idempotence property of the masking function — the proof is essential identical to the one in [5]; therefore we make an informal statement here and defer the formal treatment to Appendix B.

Theorem 4 (Informal). *Let $\mathcal{G}$ be a PWI. Suppose there exists a mask M such that: (1) When starting from an arbitrary initial state S of the adversary's choosing, the final PWI state S' is indistinguishable from $M(S')$ provided the PWI obtains sufficient entropy; (2) When starting from an initial state $M(S)$ (for adversarially chosen S), the final PWI state S' is indistinguishable from $M(S')$, even if the adversary controls the intervening entropy input strings; (3) $\mathcal{G}$ produces pseudorandom outputs when in a masked state. Then $\mathcal{G}$ is robust.*

7 Security of the ISK-RNG as a PWI

We are now positioned to analyze the security of ISK-RNG. To begin, we demonstrate some simple attacks that violate both forwards and backwards security (hence robustness, too). Next, we show that by placing a few additional restrictions on adversaries — restrictions that are well-motivated by the hardware — we can recover forward security. As we said in our introduction, the concrete security bounds we prove are not as strong as one might hope, due to some limitations of CBCMAC's effectiveness as an entropy extractor in the ISK-RNG. However, we are able to prove somewhat better results when legitimate parties use only the `RDRAND` interface, even when attackers also have access to `RDSEED`. This means that, e.g., a hostile process can't use its access to `RDSEED` to learn information about `RDRAND` return values used by a would-be victim; the result also implies stronger results for Ivy Bridge chips, where `RDSEED` is not available.

For the remainder of Section 7, we fix the following constants: $p = 1$ is the length of each entropy input; $k = 128$ is the length of each CBCMAC input block (since ISK-RNG uses AES); $\mathrm{IFace} = \{\texttt{RDSEED}, \texttt{RDRAND}\}$ are the ISK-RNG interfaces; $m = 2, 3$ is the number of healthy samples required by Ivy Bridge and Broadwell, respectively, before the CE buffer is available; and $\ell = 64$ is the length of the PWI outputs. Although `RDRAND` also allows programs to request 16 or 32 bits, this is implemented by fetching then truncating a 64-bit output, and similarly with `RDSEED` [10]. Therefore we assume without loss of generality that the adversary only requests the full 64 bits.

Recall that in the PWI model, the entropy source leaks information γ about each input string. We assume that every 256th such string (each one a single bit, $p = 1$) leaks the health of the corresponding 256 bit string (as determined by the online health test). Hence the adversary will always know the health of the OSTE buffers and the value of the health buffer. This is not simply a convenience: because the CE buffer is not available until it has been reconditioned with m healthy samples, `RDSEED` may leak health information through a timing side channel.

When the CE buffer is available, it can be used to reseed the DRBG or to service a `RDSEED` instruction. Priority is given to whichever was not last used [10]. However, because the PWI model cannot describe pending `RDSEED` instructions, the adversary must explicitly use its wait oracle to yield when it has priority: a wait invocation uses the CE to reseed, while a `RDSEED` invocation returns its contents.

The adversary's wait oracle also allows us to account for the fact that updating the eight 64-bit output buffers is not an atomic operation. By using the tick function (invoked by wait) to only fill one at a time, we conservatively allow the adversary to control if a reseeding operation intervenes. Note that tick will reseed rather than fill an output buffer if reseeding is desired ($S.\mathsf{count} > 0$) and possible ($S.\mathsf{CEfull} = 1$). This reflects the priorities of the hardware [10].

In order to save power, the entropy source goes to sleep if all the output buffers are full, the CE buffer is available, and no `RDRAND` instructions have been processed since the last reseed [10]. The PWI model, however, requires that we continue to provide $\mathcal{D}$-$\mathsf{refresh}$ access to the adversary. Our decision to leak health information to the adversary allows us to avoid any problems here: the adversary knows when the entropy source sleeps, so we can restrict the adversary to not make $\mathcal{D}$-$\mathsf{refresh}$ calls when it does.

To make this power-saving hardware constraint "work" with the PWI model, we assume that each healthy 256-bit block produced by the entropy source contains at least γ bits of min-entropy. Formally, define $(\sigma_i, b_i, \gamma_i, z_i) = \mathcal{D}(\sigma_{i-1})$ for $i \geq 1$ (where $\sigma_0 = 0$), and let $I_i = b_{256i} b_{256i+1} \cdots b_{256i+255}$. We assume there exists $\gamma > 0$ such that $\mathbf{H}_\infty\left(I_i \mid (\sigma_j, I_j, \gamma_j, z_j)_{j \neq i}, I_i \in \mathcal{H}\right) \geq \gamma$, and require that $\sum_{j=256i}^{256i+255} \gamma_i \geq \gamma$ whenever $I_i \in \mathcal{H}$. We set $\gamma^* = m\gamma$ to demand, in effect, that ISK-RNG delivers on its implicit promise that m healthy entropy samples are sufficient. At the end of this section, we will draw from the CRI report's analysis to find reasonable estimates for γ and discuss the implications.

7.1 Negative Results

We begin with some negative results, showing that the ISK-RNG achieves neither forward, nor backwards security; this immediately rules out robustness, too.

Theorem 5 (ISK-RNG lacks forward security). *There exists an adversary A making one* $\mathsf{next\text{-}ror}$ *query and one* $\mathsf{get\text{-}state}$ *query such that for any entropy source $\mathcal{D}$, $\mathbf{Adv}^{\mathrm{fwd}}_{\mathsf{ISK},\mathcal{D}}(A) = 1 - 2^{-128}$.*

Theorem 6 (ISK-RNG lacks backward security). *There exists an adversary A making one* $\mathsf{next\text{-}ror}$ *query and one* $\mathsf{get\text{-}state}$ *query such that for any entropy source $\mathcal{D}$, $\mathbf{Adv}^{\mathrm{bwd}}_{\mathsf{ISK},\mathcal{D}}(A) = 1 - 2^{-128}$.*

In the case of backwards security, the adversary assigns $S \leftarrow \mathsf{get\text{-}state}()$, makes a sequence of $\mathcal{D}$-$\mathsf{refresh}$ calls to clear the $\mathsf{corrupt}$ flag (which, by our previously state assumptions, will happen as soon as the CE buffer becomes available), and finally assigns $X \leftarrow \mathsf{next\text{-}ror}(\texttt{RDRAND})$. The adversary then checks if $X = S.\mathsf{out}_1$, and outputs 0 if this is the case and 1 otherwise. For forward security, the adversary assigns $X \leftarrow \mathsf{next\text{-}ror}(\texttt{RDSEED})$ followed by $S \leftarrow \mathsf{get\text{-}state}()$. If $X = \mathsf{AES}_{\mathbf{0}}^{-1}(S.\mathsf{CE}_0) \parallel \mathsf{AES}_{\mathbf{0}}^{-1}(S.\mathsf{CE}_1)$, the adversary outputs 0; otherwise, the it outputs 1. (Here, $\mathbf{0} = 0^{128}$.)

However, these results are very conservative. In the case of forward security, the hardware will very quickly recondition the CE buffer, effectively erasing all state that could be used to compute previous outputs. In the case of backward security, a large but finite number future outputs can be compromised by examining the buffer; however, here the situation is slightly more complicated because future DRBG keys are leaked via the ESSR, OSTE, and CE buffers.

7.2 Positive results

We now turn our attention to restricted, but still conservative, classes of adversary in order to produce positive results.

Additional assumptions. We further assume that in the forward-security game, adversaries do not make their get-state query until they have allowed the output buffers to be refilled. This assumption is motivated by the speed with which the hardware will automatically accomplish this: at the reported `RDRAND` throughput of 500 MB/s, all eight 64-bit buffers can be refilled around 8 million times per second. Formally:

Definition 6 (Delayed adversaries). *An adversary A attacking ISK-RNG in the forward-security game is* delayed *if after making its last* get-next *and* next-ror *queries, A calls $\mathcal{D}$-*refresh *until the CE buffer is available, then calls* wait *nine times before making its* get-state *query.* □

This will trigger a reseed and then refill any of the eight output buffers.

Moreover, we will assume there is some positive probability β such that each 256-bit block of bits from the entropy source is healthy with probability at least β. Formally (recall that $\mathcal{H} \subseteq \{0,1\}^{256}$ is the set of strings deemed healthy by ISK-RNG's online health tests):

Definition 7 (β-healthy). *Fix $\beta > 0$. For $i = 1, 2, 3, \ldots$ define $(\sigma_i, b_i, \gamma_i, z_i) = \mathcal{D}(\sigma_{i-1})$ (where $\sigma_0 = \varepsilon$), and for $j = 0, 1, 2, \ldots$, define $B_j = b_{256j} \parallel b_{256j+1} \parallel \cdots \parallel b_{256j+255}$. Let $H_j = 1$ if $B_j \in \mathcal{H}$, and set $H_j = 0$, otherwise. Then $\mathcal{D}$ is β-*healthy *if for all such j and all $H \in \{0,1\}^{j-1}$,* $\Pr[\, B_j \in \mathcal{H} \mid (H_\ell)_{\ell<j} = H \,] \geq \beta$. □

It follows that for any positive integers ℓ and L_m, we can bound the probability that the sequence $B_\ell, B_{\ell+1}, \ldots, B_{\ell+(L_m-1)}$ contains fewer than m healthy values using:

$$\Pr[\, |\{j \,:\, B_j \in \mathcal{H}, \ell \leq j < \ell + L_m\}| < m \,] \leq \sum_{i=0}^{m-1} \binom{L_m}{i} \beta^i (1-\beta)^{L_m - i}.$$

With these assumptions, we are ready to continue on to our positive results. Our first step is to define an appropriate masking function that describes an "ideal" state, and then to prove that setup creates such a state. This will later proofs to simply assume we begin in an idealized state (see Theorem 3).

ISK-RNG masking function. Fix the masking function $M : \{0,1\}^n \to \{0,1\}^n$ that on input S, overwrites $S.\mathsf{CE}$, $S.K$, $S.\mathsf{IV}$, and $S.\mathsf{out}_{1,\ldots,8}$ with independent, uniformly random strings of the appropriate lengths, leaves all other portions of the state untouched, and returns the result. This is the ISK-RNG masking function.

The following lemma says that if AES is a secure PRP (against adversaries making three queries) and each healthy sample from the entropy source has sufficiently large min-entropy, then the ISK-RNG masking function is honest. That is, that the ISK-RNG setup procedure successfully places the hardware in a state where (we will show) it can begin producing pseudorandom outputs.

Recall the results of Theorem 2. For convenience, we define

$$\epsilon(L_m) = \mathcal{O}(L_m + 1)/2^{k/2} \quad \text{and} \quad \hat{\epsilon}(L_m) = \sum_{i=0}^{m-1} \binom{L_m}{i} \beta^i (1-\beta)^{L_m - i},$$

where the former expression comes from Thoerem 2 and the latter from the above bound on the probability of obtaining fewer than m healthy samples from L_m trials. Here β is the assumed probability that the online health test will mark a given sequence of $2k = 256$ bits from the entropy source as "healthy", m is the number of healthy strings that need to be "conditioned" before the CE buffer is available, and $k = 128$ is the block size used by CBCMAC.

Lemma 1 (ISK-RNG masking function is honest). *Fix positive integers k and m, and fix $0 < \beta \leq 1$. Let L_m be a positive integer. Let M be the ISK-RNG masking function, defined above. Let $\mathcal{D}$ be a β-healthy entropy source. Then for any adversary A, there exists an adversary B running in the same time and making three queries such that $\mathbf{Adv}^{\mathrm{init}}_{\mathsf{ISK},\mathcal{D},M}(A) \leq 2^{(k-m\gamma)/2+2} + 4\epsilon(L_m) + 8\hat{\epsilon}(L_m) + 5\left(\mathbf{Adv}^{\mathrm{prp}}_{\mathsf{AES}}(B) + \frac{3}{2^k}\right)$.*

The following proof refers to a large body of pseudocode, found in Figure 3. We are largely free to choose L_m, and will ultimately set it to an optimal value (increasing L_m increases $\epsilon(L_m)$ but decreases $\hat{\epsilon}(L_m)$). Using reasonable estimates for the big-$\mathcal{O}$ constant and γ (discussed in Section 7.3) provides us with an upper bound of roughly 2^{-60} for the first three terms of the security bound for both $m = 2, 3$.

Proof. Recall that $\mathbf{Adv}^{\mathrm{init}}_{\mathsf{ISK},\mathcal{D}}(A) = \Pr[\,A(S,\pi,z) \Rightarrow 1\,] - \Pr[\,A(M(S),\pi,z) \Rightarrow 1\,]$, where S is the state produced by the setup procedure and z contains any side-channel information leaked to the adversary during setup. The probabilities are over the coins of $\mathsf{initialize}()$, $\mathcal{D}$, and A. During setup, ISK-RNG reconditions the CE buffer four times, and reseeds immediately afterwards each time. Let $C = (C_0, C_1, C_2, C_3)$ be the four CE buffer values used to perform these reseeding operations. Note that $X = (S, \pi, z)$ is a deterministic function of C, π, and $J = (S.\mathsf{OSTE}_1, S.\mathsf{OSTE}_2, S.\mathsf{ESSR}, z)$; we make this explicit by writing $X = f(C, \pi, J)$. Let $R = (R_0, R_1, R_2, R_3)$ be sampled from $\{0,1\}^{4k}$. We have

$$\begin{aligned}\mathbf{Adv}^{\mathrm{init}}_{\mathsf{ISK},\mathcal{D}}(A) &= \Pr[\,A(f(C,\pi,J)) \Rightarrow 1\,] - \Pr[\,A(f(R,\pi,J)) \Rightarrow 1\,] \\ &\quad + \Pr[\,A(f(R,\pi,J)) \Rightarrow 1\,] - \Pr[\,A(M(S),\pi,z) \Rightarrow 1\,] \\ &\leq \Delta(\,(\pi,C)|_J\,,(\pi,R)|_J) + (\Pr[\,A(f(R,\pi,J)) \Rightarrow 1\,] - \Pr[\,A(M(S),\pi,z) \Rightarrow 1\,])\,.\end{aligned}$$

For $j \in [0..3]$, $i = 0, 1$, let I^i_j be defined as in Figure 3; that is, I^i_j is the string of bits from the entropy source that gets used to update $S.\mathsf{CE}_i$ between the jth time and the $(j+1)$st time that buffer is available. So, for example, $C_j = C^0_j \,\|\, C^1_j$, where $C^i_j = \mathsf{CBCMAC}(C^i_{j-1} I^i_j)$ and $C^0_0 = C^1_0 = 0^{128}$. For each i, j, let $I^i_j = B^i_{j,1} B^i_{j,2} \cdots B^i_{j,\ell(j)}$, with each $\left|B^i_{j,j'}\right| = 2k$. Let $\mathcal{E}$ be the event that for each such (i,j), $\left|\left\{j' \,:\, B^i_{j,j'} \in \mathcal{H}, j' < L_m\right\}\right| \geq m$. There are eight such (i,j) pairs, so $\Pr[\,\neg\mathcal{E}\,] \leq 8\hat{\epsilon}(L_m)$. (In cases where $\ell(i) < L_m$, I_i necessarily contains m "healthy blocks" because ISK-RNG will only stop accumulating entropy if this condition holds.) Since π is independent of both $\mathcal{E}$ and J, Theorem 2 tells us that

$$\begin{aligned}\Delta(\,(\pi,C)|_J\,,(\pi,R)|_J) &\leq \Delta((\pi, C|_{J,\mathcal{E}}),(\pi,R)) + \Pr[\,\neg\mathcal{E}\,] \\ &\leq 8\left(\frac{1}{2}\sqrt{2^{k-m\gamma} + \epsilon(L_m)^2}\right) + 8\hat{\epsilon}(L_m) \\ &\leq 2^{(k-m\gamma)/2+2} + 4\epsilon(L_m) + 8\hat{\epsilon}(L_m).\end{aligned}$$

We claim that there exists some adversary B making three queries and running in time t such that

$$\Pr\left[\, A(f(R,\pi,J)) \Rightarrow 1 \,\right] - \Pr\left[\, A(M(S),\pi,z) \Rightarrow 1 \,\right] \le 5\left(\mathbf{Adv}^{\mathrm{prp}}_{\mathsf{AES}}(B) + \frac{3}{2^k}\right).$$

Consider the experiment $A(f(R,\pi,J))$. During the third reseed, ISK-RNG updates $S.K \parallel S.\mathsf{IV}$ by encrypting $S.\mathsf{CE}$ in counter-mode. But at this point in time, $S.\mathsf{CE} = R_3$ is uniform random bits, and thus so are the new key and IV (R_3 acts as a one-time pad on the counter-mode keystream). Let K_0 and V_0 be the values of $S.K$ and $S.\mathsf{IV}$ immediately following this operation. The next reseed operation creates a new DRGB key and IV, K_1 and V_1, by computing $K_1 \parallel V_1 \leftarrow \mathsf{CTR}^{V_0}_{K_0}(R_4)$. Finally, for $i \in [0..3]$, the output buffers are filled by computing $S.\mathsf{out}_{2i} \parallel S.\mathsf{out}_{2i+1} \parallel K_{i+2} \parallel V_{i+2} \leftarrow \mathsf{CTR}^{V_{i+1}}_{K_{i+1}}(0^{3\cdot 128})$. The proof of our claim concludes with a standard hybrid argument where the outputs of these five CTR computations are replaced one-by-one with random bits; the $3/2^k$ term falls out of the PRF-PRP switching lemma. The final CE buffer in the $A(f(R,\pi,J))$ experiment is already uniformly random (R_4), and the final step in the hybrid argument completes the masking function's task of replacing $S.\mathsf{out}$, $S.K$, and $S.\mathsf{IV}$ with uniform random bits. □

Remark. The PRP term may be problematic if one takes the view that `RDSEED` should offer information-theoretic security. That is, Lemma 2 says that the ISK-RNG initialization procedure yields state — which includes the CE buffers — that is only computationally indistinguishable from "ideal". However, we observe that if one adjusts the masking function to leave the output buffers unchanged, and demands a post-setup reconditioning (which the hardware endeavors to provide, anyway), one could indeed use the result to prove information-theoretic `RDSEED` security. However, this would be at the expense of *not* being able to prove security of the `RDRAND` interface, a task which necessarily requires computational assumptions.

Forward security. Our exploration of forward security proceeds in two steps. To begin, we introduce a new game, M-`RDRAND`, which differs from M-FWD in that the $\mathsf{next\text{-}ror}$ oracle always returns the "real" value R_0 when queried on the $m =$ `RDSEED` interface, but behaves normally during queries to the `RDRAND` interface. Define $\mathbf{Adv}^{\mathrm{fwd-RDRAND}/M}_{\mathcal{G},\mathcal{D}}(A) = 2\Pr\left[\, M\text{-RDRAND}(A) \Rightarrow 1 \,\right] - 1$. Proving the security of this game is not only a useful intermediate step in proving the security of M-FWD, but also can be interpreted as measuring the strength of `RDRAND` return values when an adversary also has access to the `RDSEED` instruction (which can be used to learn information about the ISK-RNG state, but that we do not require to return pseudorandom values). This distinction is valuable, because the concrete bounds on the M-FWD experiment are not as strong as one would hope.

Theorem 7 **(M-`RDRAND`).** *Let A be a delayed adversary making q queries to* `RDRAND` *and running in time t. Then there exists an adversary B making three queries and running in time $\mathcal{O}(t)$ such that* $\mathbf{Adv}^{\mathrm{fwd-RDRAND}/M}_{\mathsf{ISK},\mathcal{D}}(A) \le 2(q+4)\left(\mathbf{Adv}^{\mathrm{prp}}_{\mathsf{AES}}(B) + \frac{3}{2^k}\right)$.

Proof. During an execution of M-`RDRAND`(A), the DRBG key $S.K$ is changed during reseeding or after a $\mathsf{get\text{-}next}$ call. Let M-`RDRAND`$^{\nu}(A)$ be the same as M-`RDRAND`, but where $\mathsf{AES}_{S.K}$ is replaced with a random function the first ν times this happens. Then $\Pr\left[\, M\text{-RDRAND}^{\nu}(A) \Rightarrow 1 \,\right] - \Pr\left[\, M\text{-RDRAND}^{\nu+1}(A) \Rightarrow 1 \,\right] \le \mathbf{Adv}^{\mathrm{prp}}_{\mathsf{AES}}(B_\nu) + \frac{3}{2^k}$, for some adversary B making three queries and running in the same time as A. (B_ν simulates M-`RDRAND`$^{\nu}$ for A using random functions for the first ν keys, uses its oracle for key $(\nu+1)$, and then uses AES for the remainder of the experiment; B_ν returns whatever value A does.)

Let $d_\nu = 2\Pr[\,M\text{-}\mathtt{RDRAND}^\nu(A) \Rightarrow 1\,] - 1$. Observe that in Game M-$\mathtt{RDRAND}^\nu$, we can defer assigning b a value until after query ν; until that point, all the coins of the experiment are independent Then $d_{q+4} = 0$ because the final state, revealed by $\mathsf{get\text{-}state}()$, is likewise independent of b. Further, $d_0 = \mathbf{Adv}^{\mathrm{fwd-}\mathtt{RDRAND}/M}_{\mathsf{ISK},\mathcal{D}}(A)$. It follows that $d_0 \le 2\sum_{\nu=1}^{q+4}\left(\mathbf{Adv}^{\mathrm{prp}}_{\mathsf{AES}}(B_\nu) + \frac{3}{2^k}\right)$. Taking B to be the B_ν with maximal advantage completes the proof. □

Barring an efficient attack on AES (that only uses three queries!) this bound is quite strong. If q were to grow quite large, say on the order of $q \approx 2^{80}$, then the bound might begin to approach 2^{-40}, which seems a reasonable safety margin. However, even at the reported rate of around 500 MB/s, ISK-RNG would take over 70 years to reach this point. Moreover, the hybrid factor of q is likely a conservative artifact of the proof.

Note, however, that this bound applies to ISK-RNG when starting in an "ideal" masked state; one needs to add in the bound from Lemma 1 to account for initialization. As we mentioned earlier, reasonable estimates for the big-O constant and γ (see Section 7.3) place this term at roughly 2^{-60}.

We now proceed to the "full" forward-security result, where both the RDRAND and the RDSEED interfaces are required to produce indistinguishable-from-random outputs. Since RDSEED reads directly from the CE buffer, this bound relies more heavily on the entropy source and CBCMAC extractor (and less on the computational security of AES).

Theorem 8. *Fix a positive integers k and m, and fix $0 < \beta \le 1$. Let L_m be a positive integer. Let A be a delayed adversary making a combined q queries to* get-next *and* next-ror*. Then if $\mathcal{D}$ is β-healthy, there exists some adversary B making three queries and running in the same time as A such that* $\mathbf{Adv}^{\mathrm{fwd}/M}_{\mathsf{ISK},\mathcal{D}}(A) \le (q+1)\left(2^{(k-m\gamma)/2} + \epsilon(L_m) + 2\hat\epsilon(L_m)\right) + 2(q+4)\left(\mathbf{Adv}^{\mathrm{prp}}_{\mathsf{AES}}(B) + \frac{3}{2^k}\right)$.

Proof. For $j \in [1..q+1]$, let $C_j = C^0_j \,\|\, C^1_j$ be the sequence of available conditioned entropy buffers produced over the course of the experiment M-FWD, as defined as in Figure 3. Note that because the outputs of $\mathcal{D}$ do not depend on the behavior of A, we are free to fix them, and hence the C_j, before the experiment M-FWD begins.

Let $C = (C_1, \ldots, C_{q+1})$. We write $\mathbf{Adv}^{\mathrm{fwd}/M}_{\mathsf{ISK},\mathcal{D}}(A)_C$ to denote running the experiment using these values. Let $R = (R_1, \ldots, R_{q+1})$ be independent uniformly random values. Then for any adversary A,

$$\mathbf{Adv}^{\mathrm{fwd}/M}_{\mathsf{ISK},\mathcal{D}}(A)_C - \mathbf{Adv}^{\mathrm{fwd}/M}_{\mathsf{ISK},\mathcal{D}}(A)_R \le 2\Delta(C,R)$$

As before, for $j \in [1..q+1]$, $i = 1,2$, let I^i_j be defined as in Figure 3 (so, for example, $C_j = C^0_j \,\|\, C^1_j$, where $C^i_j = \mathsf{CBCMAC}(C^i_{j-1} I^i_j)$ and $C^i_0 = 0^{128}$).

For each i, j, let $I^i_j = B^i_{j,1} B^i_{j,2} \cdots B^i_{j,\ell(j)}$, with each $\left|B^i_{j,j'}\right| = 2k$. Let $\mathcal{E}$ be the event that for each such (i,j), $\left|\left\{j' \,:\, B^i_{j,j'} \in \mathcal{H}, j' < L_m\right\}\right| \ge m$. There are $2(q+1)$ such (i,j) pairs, so $\Pr[\,\neg\mathcal{E}\,] \le 2(q+1)\hat\epsilon(L_m)$. (In cases where $\ell(i) < L_m$, I_i necessarily contains m "healthy blocks" because ISK-RNG will only stop accumulating entropy if this condition holds.) By Theorem 2,

$$\Delta(C,R) \le \Delta(C|_{\mathcal{E}}, R) + \Pr[\,\neg\mathcal{E}\,] \le 2(q+1)\left(\frac{1}{2}\sqrt{2^{k-m\gamma} + \epsilon(L_m)^2}\right) + 2(q+1)\hat\epsilon(L_m).$$

At this point we apply the argument of Theorem 7 to show

$$\mathbf{Adv}^{\mathrm{fwd}/M}_{\mathsf{ISK},\mathcal{D}}(A)_R \le 2(q+4)\left(\mathbf{Adv}^{\mathrm{prp}}_{\mathsf{AES}}(B) + \frac{3}{2^k}\right),$$

this time allowing the RDSEED interface to return random bits when $b = 1$. This change, however, has no bearing on the argument, as the "real" values are now uniformly random anyway.

The only subtly here is that the last value returned by RDSEED may not be independent of the final contents of the CE buffer. This is why we assume that the adversary invokes $\mathcal{D}$-refresh until the CE buffer is available: the bound accounts for the fact that the new CE buffer, R_j, must be statistically close to uniform and independent of $(R_i)_{i<j}$. Since the buffer can be refreshed at most once for each get-next and next-ror query, $j \leq q + 1$. □

Corollary 1. *Let A be a delayed adversary making a combined q queries to its* get-next *and* next-ror *oracles. If $\mathcal{D}$ is β-healthy, then there exists and adversary B making three queries and running in the same time as A such that $\mathbf{Adv}^{\mathrm{fwd}}_{\mathsf{ISK},\mathcal{D}}(A) \leq (q+5)\left(2^{(k-m\gamma)/2} + \epsilon(L_m) + 2\hat{\epsilon}(L_m)\right) + (2q+13)\left(\mathbf{Adv}^{\mathrm{prp}}_{\mathsf{AES}}(B) + \frac{3}{2^k}\right)$, where the remaining quantities are defined as in Theorem 8.*

The corollary follows from applying Theorem 3 to Theorem 8, and using Lemma 1. We defer our discussion of this bound to Section 7.3. First, we briefly turn our attention to the questions of backwards security and robustness.

Backwards security and Robustness. The issue with obtaining backwards security (and hence robustness) is that future outputs can linger in the output buffers indefinitely: the hardware will shutdown the entropy source after all the buffers are full and the CE buffer is available. Moreover, even if RDSEED instructions cause the ES to produce more random bits, those bits will not be used to reseed the DRBG as long as the buffers remain full. Hence, state remains compromised until fresh entropy filters through the $\mathsf{ESSR} \to \mathsf{OSTE}_1 \to \mathsf{OSTE}_2 \to \mathsf{CE}$ buffers and is used to reseed the DRBG, without first being siphoned off by RDSEED.

Consider the worst-case scenario for Ivy Bridge chips, where only the RDRAND interface is available. Following a state compromise, the next 8 outputs are revealed in the output buffers, the next 511 may be generated using the compromised DRBG seed, the next 511 may be generated using a DRBG seed determined by the compromised CE buffer, and the next 511 may be generated using a DRBG key determined by the compromised OSTE and ESSR buffers. This amounts to slightly more than 12KB worth of outputs that an adversary could potentially predict.

However, we show in Appendix C that if one restricts the model to "read-only" adversaries (by denying adversaries access to set-state but permitting access to get-state) *and* one discounts wins based on the above attacks (by denying adversaries access to next-ror until after the "corrupted" values have already been replaced) then ISK-RNG is secure. The concrete bounds we obtain are essentially identical to those provided by Theorems 7 and 8, depending on whether or not one requires the RDSEED interface to be secure. See the appendix for further discussion of how these restrictions can be interpretted and a formal theorem statement and proof.

7.3 Discussion of results

Let us examine the bound of Corollary 1 in detail. We specialize to the parameters used by Intel: $k = 128$ (a consequence of using AES), $m = 2$ for Ivy Bridge chips, and $m = 3$ for Broadwell chips.

To estimate γ, we turn to the CRI report [7]. Hamburg, Kocher, and Marson subjected raw entropy source bits (using data provided by Intel) to a battery of statistical tests. Using a Markov model with 12 bits of state, they estimate the entropy source produces approximately 0.65 bits of min-entropy per bit of output. However, this was an average (some states of the Markov model resulted in more predictable bits), and a 12-bit state, though perhaps necessary to collect enough samples for a meaningful empirical analysis, is not enough for our purposes. Therefore let us suppose a more conservative rate of 0.5, leading to $\gamma = 128$.

This sets the $(q+5)2^{(k-m\gamma)/2}$ term of our bound to $(q+5)2^{-64}$ for Ivy Bridge (where $m = 2$) and $(q+5)2^{-128}$ for Broadwell (where $m = 3$). The latter bound is quite strong, but, given how quickly q can grow, the former may be worrisome if one wishes to maintain strong security guarantees (e.g., one wishes to cap an adversary's advantage at 2^{-40}). Unfortunately, this is not the dominate term in the security bound.

We next consider the term $(q + 5)(\epsilon(L_m) + 2\hat{\epsilon}(L_m))$. If we set the big-O constant[2] of ϵ to c (so $\epsilon(L_m) = cL/2^{64}$) then we can choose L_m to optimize this expression. Taking $\beta = 1/2$, $c = \sqrt{10}$, which we believe to be conservative, gives an upper bound of $(q+5)2^{-56}$; a more generous $\beta = 0.99$, $c = 1$ improves the upper bound to about $(q + 5)2^{-60}$. (These bounds are accurate for both $m = 2$ and $m = 3$, although the corresponding values for L_m differ considerably.)

At this point, limiting an adversary's advantage to 2^{-40} is difficult — an adversarial process gathering random bits at the benchmarked rate of 500 MB/s could issue the maximum allowable number of queries in under one millisecond. Or at least, this is the case if we demand that `RDSEED` produces uniform random outputs. On the other hand, if one only needs `RDRAND` to be secure, then Theorem 7 suggests that limiting an adversary's advantage to 2^{-40} is entirely reasonable; in this setting, we only pick up a single $4(\epsilon(L_m) + 2\hat{\epsilon}(L_m))$ term even after moving to the unmasked forward-security setting, with no troublesome multiplicative factor of q.

The remaining term, $(2q + 13)(\mathbf{Adv}^{\mathrm{prp}}_{\mathsf{AES}}(B) + 3/2^{128})$, is likely to be negligible (recall that B is permitted only three queries).

Our analysis does not point to any obvious, practical attacks (aside from the trivial ones that exploit the output buffers, though it seems a stretch to deem those practical). However, it exposes the CBCMAC extraction process as the likely weakest link, and quantifies the extent of that weakness. An actual attack would need to exploit how the specific output distribution of the entropy source interacts with CBCMAC under the fixed key K'. Our goal, though, has been to demand and seek out positive evidence of security.

8 Acknowledgements

The authors wish to thank DJ Johnston and Jesse Walker of Intel for answering our questions regarding the design and implementation details of ISK-RNG. Both Terashima and Shrimpton were supported by NSF grants CNS-0845610 and CNS-1319061.

References

1. Boaz Barak and Shai Halevi. A model and architecture for pseudo-random generation with applications to `/dev/random`. In *Proceedings of the 12th ACM conference on Computer and communications security*, pages 203–212. ACM, 2005.
2. Mihir Bellare and Phillip Rogaway. The security of triple encryption and a framework for code-based game-playing proofs. In *Advances in Cryptology—EUROCRYPT 2006*, pages 409–426. Springer, 2006.
3. Olivier Chevassut, Pierre-Alain Fouque, Pierrick Gaudry, and David Pointcheval. The twist-AUgmented technique for key exchange. In *Public Key Cryptography*, pages 410–426. Springer, 2006.

[2] This constant is not specified in [4], but at least one author assures us that it is "small", e.g., less than 10.

4. Yevgeniy Dodis, Rosario Gennaro, Johan Håstad, Hugo Krawczyk, and Tal Rabin. Randomness extraction and key derivation using the CBC, cascade and HMAC modes. In *Advances in Cryptology—CRYPTO 2004*, pages 494–510. Springer, 2004.
5. Yevgeniy Dodis, David Pointcheval, Sylvain Ruhault, Damien Vergniaud, and Daniel Wichs. Security analysis of pseudo-random number generators with input: `/dev/random` is not robust. In *Proceedings of the 2013 ACM SIGSAC conference on Computer & communications security*, pages 647–658. ACM, 2013.
6. Zvi Gutterman, Benny Pinkas, and Tzachy Reinman. Analysis of the Linux random number generator. In *Security and Privacy, 2006 IEEE Symposium on*, pages 15–pp. IEEE, 2006.
7. Mike Hamburg, Paul Kocher, and Mark E Marson. Analysis of Intel's Ivy Bridge digital random number generator. *Online: http://www. cryptography. com/public/pdf/Intel_TRN G_Report_20120312. pdf*, 2012.
8. Gael Hofemeier. Intel Digital Random Number Generator (DRNG) software implementation guide. https://software.intel.com/en-us/articles/intel-digital-random-number-generator-drng-software-implementation-guide, August 2012. Accessed May 2014.
9. Gael Hofemeier and Robert Chesebrough. Introduction to Intel AES-NI and Intel Secure Key instructions. https://software.intel.com/en-us/articles/introduction-to-intel-aes-ni-and-intel-secure-key-instructions, July 2012. Accessed May 2014.
10. JD Johnston (Intel). Personal communication, May 2014.
11. Patrick Lacharme, Andrea Röck, Vincent Strubel, and Marion Videau. The Linux pseudorandom number generator revisited. *IACR Cryptology ePrint Archive*, 2012:251, 2012.
12. John Mechalas. The difference between RDRAND and RDSEED. https://software.intel.com/en-us/blogs/2012/11/17/the-difference-between-rdrand-and-rdseed, November 2012. Accessed April 2014.
13. Jesse Walker. Conceptual foundations of the Ivy Bridge random number generator. http://www.ists.dartmouth.edu/docs/walker_ivy-bridge.pdf, November 2011.

A PWIs with non-pseudo-uniformly random state

We contend the security definitions of [5] are flawed. Recall that in [5], the experiments defining FWD, BWD, RES, and ROB security begin by sampling a uniform random state. The advantage of an adversary in, for example, the FWD security definition is equivalent to our definition for $\mathbf{Adv}^{\$-\mathrm{fwd}}$, where $\$(\cdot)$ returns a fresh random string on each invocation.

Consider following PWI $\mathcal{G} = (\mathsf{setup}, \mathsf{refresh}, \mathsf{next})$ with $2n$ bits of state. Define setup to return 0^{2n}, $\mathsf{refresh}(S, I) = S$, and $\mathsf{next}(S_0 \parallel S_1) = (S_0' \parallel S_1, R)$, where $|S_0| = |S_1| = |S_0'| = |R| = n$, $S_0' \parallel R = f_{S_0}(0^m) \parallel f_{S_0}(1^m)$, and f is a PRF. It isn't hard to show that for any A making q queries, there exists some B making $2q$ queries such that $\mathbf{Adv}^{\mathrm{fwd}/\$}_{\mathcal{G}}(A) \leq \mathbf{Adv}^{\mathrm{prf}}_{f}(B)$. However, $\mathcal{G}$ cannot reasonably be said to have forward security because using $\$(\cdot)$ to create the initial state does not reflect anything approximating the behavior of setup.

One could argue that this is simply an artifact of $\mathcal{G}$'s lack of *backward* security. After all, it is the adversary's knowledge concerning the initial state that is problematic here. But this flaw should not be surfaced in the definition of *forward* security. Granted, we do need forward security to start from a "healthy" state. Using an explicit setup procedure allows us to neatly avoid dictating what that state should be without requiring the security definitions themselves to handle the issue on an *ad-hoc* basis.

Further consider the (admittedly pathological) example $\mathcal{G}' = (\mathsf{setup}', \mathsf{refresh}, \mathsf{next}')$, where

$$\mathsf{next}'(S_0 \parallel S_1) = \begin{cases} \mathsf{next}(S_0, S_1) & \text{If } S_1 = 0^n, \\ (f_{S_0}(0^m) \parallel S_0, f_{S_0}(1^m)) & \text{otherwise,} \end{cases}$$

and $\mathsf{setup}'(\mathsf{seed}, \mathcal{D})$ uses an entropy extractor (e.g., the one described in [5]) to produce a random value $R \in \{0,1\}^n$, then outputs $R \parallel 0^n$. Now the state leaks the PRF key used to generate the most recent next value — *unless* the right n-bits of state are 0^n. Here, starting with a uniform random state can artificially "break" a PWI that would otherwise have forward security.

B On proving robustness

When Dodis et al. introduced their PWI model [5], they presented a theorem showing that any PWI possessing two relatively weak security properties, *preserving* and *recovering* security, is also robust. However, both of these security definitions assume that state should ideally be uniformly random. The theorem therefore isn't useful for PWIs like ISK-RNG, for which this assumption is false (see Appendix A). In this section we present analogous definitions that instead use masking functions, and prove a corresponding theorem. Although this result cannot be applied to ISK-RNG, which lacks even backwards security, we offer it as a contribution towards developing a theory for PWIs.

First we define idempotent masking functions. This property, which we believe most interesting masking functions will possess, will be a mathematical convenience in the coming proof. It allows an adversary in a reduction argument to blindly apply a masking function to a state that may or may not already be masked without worrying that doing so will cause the simulated environment and the "expected" environment to diverge.

Definition 8 (Idempotent masking functions). *A masking function $M : \{0,1\}^n \to \{0,1\}^n$ is* idempotent *if for any state $S \in \{0,1\}^n$, $M(S)$ and $M(M(S))$ are identically distributed random variables.* □

The security experiments for Recovering security and Preserving security are shown in Figure 5. We refer to the two games as Recover and Preserve, respectively. Our definitions are equivalent to those of [5] if one specializes to a M that returns a string sampled from $\{0,1\}^n$ and considers only non-blocking PWIs.

Roughly, a PWI has recovering security if even when starting in a compromised state, it will, after harvesting sufficient entropy, eventually return to a pseudo-random state (as described by the masking function) and begin producing pseudo-random outputs. It has preserving security if it can continue to maintain a pseudo-random state and produce pseudo-random outputs even if given arbitrary, adversarially controlled inputs from the entropy source. Intuitively, these properties should suffice to ensure robustness, and indeed, this is the case.

Definition 9 ((Witnessed) Recovering-Security). *A PWI $\mathcal{G} = (\mathsf{setup}, \mathsf{refresh}, \mathsf{next}, \mathsf{tick})$ has $(t, q_{\mathcal{D}}, \gamma^*, \epsilon)$-recovering security, witnessed by the masking function M, if, for any attacker A and legitimate sampler $\mathcal{D}$, both running in time t, the recovering advantage*

$$\mathbf{Adv}^{\mathrm{rec}}_{\mathcal{G};M}(A) = 2\Pr\left[\,\mathsf{Recover}_{\mathcal{G}}(A, \mathcal{D}, M) = 1\,\right] - 1$$

is at most ϵ. □

Definition 10 ((Witnessed) Preserving Security). *A PWI $\mathcal{G}$ has (t, ϵ)*-preserving security *witnessed by the (possibly randomized) function $M : \{0,1\}^n \to \{0,1\}^n$ if for any attacker A running in time t, the preserving advantage $\mathbf{Adv}^{\mathrm{pres}}_{\mathcal{G};M}(A) = 2\Pr\left[\,\mathsf{Preserve}(\mathcal{G}, A, M) = 1\,\right] - 1$ is at most ϵ.* □

With this setup, we can state our PWI robustness security result. With the above definitions in place, the proof itself is essentially identical to that of Theorem 1 of DPRVW; our exposition closely follows theirs, and we include it for the sake of completeness. The only two subtleties are the need for the idempotence property of masking functions and the change to Preserve that places the initial (unmasked) state under the adversary's control.

Experiment $\mathsf{Recover}(\mathcal{G}, A, \mathcal{D}, M)$:

```
$(\mathsf{setup}, \mathsf{refresh}, \mathsf{next}) \leftarrow \mathcal{G}$
$\mathsf{seed} \xleftarrow{\$} \mathcal{S}$
$b \xleftarrow{\$} \{0,1\}$; $\sigma_0 \leftarrow 0$; $\mu \leftarrow 0$
for $k = 1, \ldots, q_{\mathcal{D}}$ do
    $(\sigma_k, I_k, \gamma_k, z_k) \leftarrow \mathcal{D}(\sigma_{k-1})$
$(S_0, d, \sigma') \xleftarrow{\$} A^{\mathsf{get\text{-}refresh()}}(\mathsf{seed}, \gamma_1, \ldots, \gamma_{q_{\mathcal{D}}}, z_1, \ldots, z_{q_{\mathcal{D}}})$
if $\mu + d > q_{\mathcal{D}}$ or $\sum_{j=\mu+1}^{\mu+d} \gamma_j < \gamma^*$ then
    return $0$
for $j = 1, \ldots, d$ do
    $S_j \leftarrow \mathsf{refresh}(S_{j-1}, I_{\mu+j}, \mathsf{seed})$
$(S_0^*, R_0^*) \leftarrow \mathsf{next}(S_d)$
$S_1^* \xleftarrow{\$} M(S_0^*)$
if $R_0^* = \perp$ then
    $R_1^* \leftarrow \perp$
else
    $R_1^* \xleftarrow{\$} \{0,1\}^{\ell}$
$b^* \xleftarrow{\$} A(\sigma', S_b^*, R_b, I_{\mu+d+1}, \ldots, I_{q_{\mathcal{D}}})$
if $b^* = b$ then
    return $1$
else
    return $0$
```

Proc $\mathsf{get\text{-}refresh()}$:

```
$\mu \leftarrow \mu + 1$
return $I_\mu$
```

Experiment $\mathsf{Preserve}(\mathcal{G}, A, M)$:

```
$(\mathsf{setup}, \mathsf{refresh}, \mathsf{next}) \leftarrow \mathcal{G}$
$\mathsf{seed} \xleftarrow{\$} \mathcal{S}$; $b \xleftarrow{\$} \{0,1\}$
$(S_0', I_1, \ldots, I_d, \sigma') \xleftarrow{\$} A(\mathsf{seed})$
$S_0 \xleftarrow{\$} M(S_0')$
for $j = 1, \ldots, d$ do
    $S_j \leftarrow \mathsf{refresh}(S_{j-1}, I_j, \mathsf{seed})$
$(S_0^*, R_0^*) \xleftarrow{\$} \mathsf{next}(S_d)$
$S_1^* \xleftarrow{\$} M(S_0^*)$
if $R_0^* = \perp$ then
    $R_1^* \leftarrow \perp$
else
    $R_1^* \xleftarrow{\$} \{0,1\}^{\ell}$
$b^* \xleftarrow{\$} A(\sigma', S_b^*, R_b^*)$
if $b^* = b$ then
    return $1$
else
    return $0$
```

Fig. 5. Security experiments for recovering and preserving security, with respect to a witness masking function M.

Theorem 9. *[Formal statement of Thm. 4] Let q_N be a positive integer. If there is an idempotent masking function $M : \{0,1\}^n \to \{0,1\}^n$ that witnesses the (t, ϵ_r)-recovering and (t, ϵ_p)-preserving security of a PWI $\mathcal{G}$, and M is $(\mathcal{G}, \mathcal{D}, t, \epsilon_h)$-honest, then $\mathcal{G}$ is $(t', \epsilon_h + q_N(\epsilon_r + \epsilon_p))$-robust.*

Proof. We refer to queries to either the get-next or next-ror oracles as next queries. A next query is *uncompromised* if it is made when corrupt = false, and *compromised* otherwise. If at any point in the experiment corrupt is true, the next *uncompromised* next query is a *recovering* query. The remaining uncompromised next queries are *preserving* queries.

When an adversary makes a recovering query, we associate with it a *most recent entropy drain* (MRED) query: namely, the most recent query to get-state, set-state, or get-next. (We assume without loss of generality that the adversary never makes a query to next-ror when corrupt = true.) Note that between any recovering query and its associated MRED query (which set $c \leftarrow 0$), the adversary must have made a series of queries to $\mathcal{D}$-refresh, generating a sequence of *recovering samples* $I = (I_i, I_{i+1}, \ldots, I_{i+d})$ with the property that the corresponding entropy estimates $(\gamma_i, \gamma_{i+1}, \gamma_{i+d})$ sum to at least γ^*.

Let M-ROB be the robustness experiment were the post-setup state S is overwritten with $M(S)$. Define game G_i to be the same as $M\text{-ROB}_{\mathcal{G},\mathcal{D}}$ except that for the first i next queries:

- If the query is to next-ror, the challenger sets $S \leftarrow M(S)$ after Line 9, and always returns R_1 (not R_b).
- If the query is to get-next, the challenger sets $S \leftarrow M(S)$ after Line 15, and then if $R \neq \bot$, overwrites $R \xleftarrow{\$} \{0,1\}^\ell$.

So for the first i next queries, the state gets overwritten with a mask of the state, and the adversary receives random bits.

Further define $G_{i+1/2}$, which behaves like G_i when the $(i+1)$st next query is preserving, and like G_{i+1} otherwise. Note that $\Pr[G_0(A) = 1] = \Pr[M\text{-ROB}_{\mathcal{G},\mathcal{D}}(A) = 1] + \epsilon_h$ and that $\Pr[G_{q_N}(A) = 1] = 1/2$. (The latter equality holds because in Game G_{q_N}, all of the oracle outputs are independent of b). Therefore:

$$\Pr[M\text{-ROB}_{\mathcal{G},\mathcal{D}}(A) = 1] \leq \sum_{i=0}^{q_N - 1} \Big(\big|\Pr[G_i(A,\mathcal{D}) = 1] - \Pr\big[G_{i+1/2}(A,\mathcal{D}) = 1\big]\big| + \big|\Pr\big[G_{i+1/2}(A,\mathcal{D}) = 1\big] - \Pr[G_{i+1}(A,\mathcal{D}) = 1]\big| \Big).$$

We show in two following lemmata that the former absolute value is upper bounded by ϵ_p, while the latter is upper bounded by ϵ_r. Since $\big|\Pr[\text{ROB}_{\mathcal{G},\mathcal{D}}(A) = 1] - \Pr\big[M\text{-ROB}_{\mathcal{G},\mathcal{D}(A)} = 1\big]\big| \leq \epsilon_h$, this completes the proof. □

Lemma 2. *Let Game G_i be defined as above, with respect to some PWI $\mathcal{G}$. Then if $\mathcal{G}$ is (t', ϵ_p)-preserving secure, $\big|G_i(A,\mathcal{D}) - G_{i+1/2}(A,\mathcal{D})\big| \leq \epsilon_p$ for any adversary A running in time $t \approx t'$.*

Proof. Given A, we will construct an adversary A' for the recovery game. Note that we may assume without loss of generality that the $(i+1)$st next query will be a preserving query, because otherwise G_i and $G_{i+1/2}$ are identical. A' obtains a seed from the challenger, and uses it along with the code for $\mathcal{D}$ to simulate G_i for A until the $(i+1)$st next query.

Let S_0 be the state that A' uses to simulate G_i for A immediately following the ith next query. When A makes its $(i+1)$st next query, A' gives the challenger S_0 along with the entropy inputs $I_1, \ldots, I_d$ it generated between A's ith and $(i+1)$st next queries. The challenger replies with (S_b^*, R_b^*).

Next, A' flips its own coin $b' \xleftarrow{\$} \{0,1\}$. If $b' = 0$, A' replies to A's $(i+1)$st next query with R^{**} (if this is a get-next query) or with (S^{**}, R^{**}) (if it's a next-ror query), where if $b' = 0$ then $(S^{**}, R^{**}) = (S^*, R^*)$, and if $b' = 1$ then $S^{**} = M(S^*)$ and

$$R^{**} = \begin{cases} \perp \text{ if } R^* = \perp \\ R \text{ otherwise, for } R \xleftarrow{\$} \{0,1\}^{\ell}. \end{cases}$$

Finally, A' uses resumes simulating the environment for A, starting from state S^{**}, and following the specification of G_{i+1}. When A outputs b^*, A' outputs $b^{*'}$, which is 1 if $b^* = b'$, and 0 otherwise.

Now, if the original challenge bit was $b = 0$, then A' has exactly simulated G_i for A: the first i next queries followed the specification of G_i, while the $(i+1)$st next query returned the "real" state and output bits if $b' = 0$, and returned uniformly random bits and a mask of the state if $b' = 1$. Further, because M is idempotent, the fact that the challenger applies a mask to S_0 does not change its distribution — G_i calls for the state to be masked after every next query, and so S_0 is "already" masked.

On the other hand, if the challenge bit was $b = 1$, then A' has exactly simulated $G_{i+1/2}$ for A. It returned uniformly random bits and a mask of the state on the $(i+1)$st query. Our idempotence assumption again comes into play: when $b = 1$, A' receives a state $S^* = M(S)$ that has already been masked, and returns $S^{**} = M(M(S))$ to A; however, these random variables are identically distributed.

Consequently,

$$\begin{aligned} \left|\Pr\left[\, G_i(A, \mathcal{D}) = 1 \,\right] - \Pr\left[\, G_{i+1/2}(A, \mathcal{D}) = 1 \,\right]\right| &= \left|\Pr\left[\, b' = b^* \mid b' = 0 \,\right] - \Pr\left[\, b' = b^* \mid b' = 1 \,\right]\right| \\ &= \left|2 \Pr\left[\, b^{*'} = b \,\right] - 1\right| \le \epsilon_p, \end{aligned}$$

which is what we wanted. □

Lemma 3. *Let Game G_i be defined as above, with respect to some PWI $\mathcal{G}$. Then if $\mathcal{G}$ is (t', ϵ_r)-recovery secure, $\left|G_{i+1/2}(A, \mathcal{D}) - G_{i+1}(A, \mathcal{D})\right| \le \epsilon_r$ for any adversary A running in time $t \approx t'$.*

Proof. Given A, we will construct an adversary A' for the recovery game. Note that we may assume without loss of generality that the $(i+1)$st next query will be a recovering query, because otherwise $G_{i+1/2}$ and G_i are identical. A' obtains a seed and information $(\gamma_j, z_j)_{j=1}^{q_{\mathcal{D}}}$ from the challenger, chooses a challenge bit $b' \xleftarrow{\$} \{0,1\}$, and generates an initial state $S' \xleftarrow{\$}$ setup. Then A' simulates G_i for A, providing A with the seed and using the leaked information it obtained from the challenger and its get-refresh oracle to simulate the $\mathcal{D}$-refresh oracle for A'.

After the ith next query, however, A' no longer immediately uses its get-refresh oracle to update its state in response to a $\mathcal{D}$-refresh query from A. Instead, it simply returns the appropriate (γ, z) pair. The on the $(i+1)$st next query, A' invokes its get-refresh oracle to update its state to the point immediately following the corresponding MRED. Call this state S_0. Next, A' counts the number d of $\mathcal{D}$-refresh calls A made after the MRED, and submits (S_0, d) to the challenger. A' receives the challenge (S^*, R^*) in reply, along with the entropy strings $I_\nu, I_{\nu+1}, \ldots, I_{q_{\mathcal{D}}}$ it will later use to resume simulating an environment for A.

If $b' = 0$, A' replies to A's $(i+1)$st next query with R^{**} (if this is a get-next query) or with (S^{**}, R^{**}) (if it's a next-ror query), where if $b' = 0$ then $(S^{**}, R^{**}) = (S^*, R^*)$, and if $b' = 1$ then $S^{**} = M(S^*)$ and

$$R^{**} = \begin{cases} \perp \text{ if } R^* = \perp \\ R \text{ otherwise, for } R \xleftarrow{\$} \{0,1\}^\ell \end{cases}$$

Finally, A' uses $I_{\nu+1}, \ldots, I_{q_\mathcal{D}}$ to resume simulating the environment for A, starting from state S^{**}, and following the specification of G_{i+1}. When A outputs b^*, A' outputs $b^{*'}$, which is 1 if $b^* = b'$, and 0 otherwise.

Now, if the original challenge bit was $b = 0$, then A' has exactly simulated $G_{i+1/2}$ for A: the first i next queries followed the specification of G_i, while the $(i+1)$st next query returned the "real" state and output bits if $b' = 0$, and returned uniformly random bits and a mask of the state if $b' = 1$.

On the other hand, if the challenge bit was $b = 1$, then A' has exactly simulated G_{i+1} for A. It returned uniformly random bits and a mask of the state on the $(i + 1)$st query. This is where our idempotence assumption comes into play: when $b = 1$, A' receives a state $M(S)$ that has already been masked, and returns $M(M(S))$ to A; however, these random variables are identically distributed.

Consequently,

$$\begin{aligned} \left|\Pr\left[\, G_{i+1/2}(A,\mathcal{D}) = 1 \,\right] - \Pr\left[\, G_{i+1}(A,\mathcal{D}) = 1 \,\right]\right| &= \left|\Pr\left[\, b' = b^* \mid b' = 0 \,\right] - \Pr\left[\, b' = b^* \mid b' = 1 \,\right]\right| \\ &= \left|2\Pr\left[\, b^{*'} = b \,\right] - 1\right| \leq \epsilon_r, \end{aligned}$$

which was what we wanted. □

C On the read-only robustness of ISK-RNG

This appendix contains our analysis of the robustness of ISK-RNG against a restricted adversary, in a somewhat restricted model. We discussed the necessity of limiting the adversary on pg. 22; in short, future output values persist in output buffers even if the ES produces large amounts of entropy following a state compromise. Therefore these outputs (and in some cases many more, as we discussed earlier) will necessarily be compromised. The best we can hope for is that ISK-RNG will eventually recover, where now "eventually" means after it prodcues a certain number of (unsafe) outputs, rather than after it accumulates γ^* bits of entropy.

A second change to the ROB experiment we need to make is that we will not permit the adversary to make set-state queries. He may, however, make get-state queries without restriction. That is, we consider adversaries who can learn information about the PWI state, but not adversaries that can tamper with it. We feel this is a reasonable assumption to make because, in addition to the practical difficulties of tampering with hardware not accessible to software, an adversary possessing both the required technical expertise and opportunity to tamper with hardware will have numerous other attack vectors at his disposal, anyway; attempting to defend against him would be futile.

The distinction between "read-only" adversaries and those with write capabilities may seem rather fine, perhaps even artificial, in the context of hardware RNGs. Our intent here is not to speculate on the relative difficulty of such attacks, but rather to determine how damaging they might be. This being said, we acknowledge that the distinction is likely more important in software RNGs (for example, the recent Heartbleed vulnerability in OpenSSL permitted attackers to read, but not modify, the target's memory); hence, a read-only robustness notion may prove useful in that domain, as well.

Returning to ISK-RNG, the reason for a change to "read-only" adversaries is that a set-state query would permit the adversary to replace otherwise entropic bits with seed-dependent values, preventing us from leveraging results from the entropy-extraction literature, specifically the Left-over Hash Lemma and Theorem 1. ISK-RNG would propagate this dependency through all subsequent reconditionings. We were able to overcome these obstacles in the proof of Theorem 2 by bootstrapping off of the near-uniform randomness of each conditioner output to argue for the randomness of the next, but this remedy is unavailable here.

We call this new game (arbitrary get-state queries, no set-state queries) ro-ROB, for read-only robustness.

We call an adversary A *slow* if after making a get-state or set-state call, A does not make a next-ror call until it performs the following sequence of steps:

1. Invokes $\mathcal{D}$-refresh until the CE buffer becomes available.
2. Empties the CE buffer (by using the `RDSEED` interface, or by using `RDRAND` and then allowing ISK-RNG to reseed) and makes it available again. At this point, the "compromised" OSTE and ESSR buffers have been conditioned into the CE buffer, which therefore remains compromised.
3. Empties the CE buffer and makes it available again. One can now hope that its contents are now information theoretically random and independent of any compromised value.
4. Causes the DRBG to reseed (by using the `RDRAND` interface, which empties one of the output buffers, and then invoking wait). If the previous step successfully produced a fresh, random value for the CE buffer, then this value will act as a one-time pad when generating a new DRBG key and IV.
5. Flushes the remaining output buffers by invoking `RDRAND` and invokes $\mathcal{D}$-refresh until the CE buffer becomes available. (This final step prevents A from invoking get-state to learn the "fresh" entropy used to reseed the DRBG.)
6. Invokes wait() until the eight output buffers are repopulated.

We note that this seemingly-elaborate sequene of steps will naturally occur once processes running on the machine dispatch a sufficient number of `RDRAND` or `RDSEED` instructions.

Theorem 10 (ISK-RNG is read-only robust against slow adversaries). *Let A be a slow adversary running in time t, making q_{gs} queries to its* get-state *oracle, and a combined q queries to its* get-next *and* next-ror *oracles. Let L_m be a positive integer. Suppose $\mathcal{D}$ is β-healthy for some $\beta > 0$. Then there exists an adversasry B making three queries and running in time $\mathcal{O}(t)$ such that*

$$\mathbf{Adv}^{\mathrm{ro-rob}/M}_{\mathsf{ISK},\mathcal{D}}(A) \leq (q+3q_{\mathsf{gs}})\left(2^{(k-m\gamma)/2} + \epsilon(L_m) + 2\hat{\epsilon}(L_m)\right) + 2(q+8q_{\mathsf{gs}})\left(\mathbf{Adv}^{\mathrm{prp}}_{\mathsf{AES}}(B) + \frac{3}{2^k}\right),$$

where $\epsilon(L_m) = \mathcal{O}(L_m+1)/2^{k/2}$ and $\hat{\epsilon}(L_m) = \sum_{i=0}^{m-1}\binom{L_m}{i}\beta^i(1-\beta)^{L_m-i}$.

Proof. For $i \in [0..q_{\mathsf{gs}}]$, define G_i to be the game that behaves identically to $\mathsf{ro\text{-}ROB}_{\mathsf{ISK},\mathcal{D}}$ with challenge bit $b = 0$ (i.e., the next-ror oracle returns random values) until after i queries to get-state, and then behaves as though $b = 1$. So

$$\mathbf{Adv}^{\mathrm{ro-rob}/M}_{\mathsf{ISK},\mathcal{D}}(A) = \Pr\left[\, G_{q_{\mathsf{gs}}+1}(A) \Rightarrow 1 \,\right] - \Pr\left[\, G_0(A) \Rightarrow 1 \,\right].$$

Let S_i be the state following the ith get-state query in G_i (so S_i is not masked). Let S_i^j be that same state after the slow adversary A completes Step j above. Then $S_i^2.\mathsf{CE}_0$ is a permutation as a function of $S_{i-1}.\mathsf{CE}_0$ for any fixed values of $S_{i-1}.\mathsf{ESSR}$, $S_{i-1}.\mathsf{OSTE}$, and intervening $\mathcal{D}$ outputs, and similarly for $S_i^2.\mathsf{CE}_1$ with respect to $S_{i-1}.\mathsf{CE}_1$. Therefore $S_i^2.\mathsf{CE}$ is uniformly distributed, and likewise independent of π and any entropy source outputs.

Define the game G_i' to behave identically to G_i, except that after the CE buffer becomes available during Steps 3 and 5 above, its contents are immediately rewritten with uniform random values. It follows (cf. the proof of Lemma 1) that

$$\Pr\left[\, G_i'(A) \Rightarrow 1 \,\right] - \Pr\left[\, G_i(A) \Rightarrow 1 \,\right] \leq 2\left(2^{(k-m\gamma^*)/2} + \epsilon(L_m) + 2\hat{\epsilon}(L_m)\right).$$

Define the game G_i'' to behave identically to G_i', except that the output of each AES invocation during Step 6 is replaced with a uniform random string. Follow the logic of Lemma 1, we have that there is some adversary B making three queries and running in time t such that

$$\Pr\left[\, G_i''(A) \Rightarrow 1 \,\right] - \Pr\left[\, G_i'(A) \Rightarrow 1 \,\right] \leq 8\left(\mathbf{Adv}^{\mathrm{prp}}_{\mathsf{AES}}(B) + \frac{3}{2^k}\right).$$

Finally, let G_i''' be the game that behaves identically to G_i'', except that immediately after A completes Step 6 after making its ith $\mathsf{get\text{-}state}$ query, G_i''' overwrites the state S with $M(S)$. But since S is at this point identically distributed to $M(S)$,

$$\Pr\left[\, G_i'''(A) \Rightarrow 1 \,\right] - \Pr\left[\, G_i''(A) \Rightarrow 1 \,\right] = 0.$$

Let q_i be the combined number of queries that A makes to $\mathsf{get\text{-}next}$ and $\mathsf{next\text{-}ror}$ between its ith and $(i+1)$st $\mathsf{get\text{-}state}$ query. It follows that there is some adversary B_i making (q_i+1) queries and running in time t such that

$$\Pr\left[\, G_{i+1}(A) \Rightarrow 1 \,\right] - \Pr\left[\, G_i'''(A) \Rightarrow 1 \,\right] \leq \mathbf{Adv}^{\mathrm{fwd}/M}_{\mathsf{ISK},\mathcal{D}}(B_i).$$

(There is a subtle issue here: the M-FWD experiment starts by masking some S generated by the setup proceedure, which is not the case in this reduction; however, the proof of Theorem 8 follows through cleanly starting from $M(S)$ for arbitrary S.)

Putting everything together,

$$\begin{aligned}
\mathbf{Adv}^{\mathrm{ro-rob}/M}_{\mathsf{ISK},\mathcal{D}}(A) &\leq \sum_{i=0}^{q_{\mathrm{gs}}} \left(\Pr\left[\, G_{i+1}(A) \Rightarrow 1 \,\right] - \Pr\left[\, G_i(A) \Rightarrow 1 \,\right]\right) \\
&\leq \sum_{i=0}^{q_{\mathrm{gs}}} \Bigg[2\left(2^{(k-m\gamma^*)/2} + \epsilon(L_m) + 2\hat{\epsilon}(L_m)\right) \\
&\qquad\qquad + 8\left(\mathbf{Adv}^{\mathrm{prp}}_{\mathsf{AES}}(B) + \frac{3}{2^k}\right) + \mathbf{Adv}^{\mathrm{fwd}/M}_{\mathsf{ISK},\mathcal{D}}(B_i)\Bigg] \\
&\leq \sum_{i=0}^{q_{\mathrm{gs}}} \left[(q_i+3)\left(2^{(k-m\gamma^*)/2} + \epsilon(L_m) + 2\hat{\epsilon}(L_m)\right) + 2(q_i+8)\left(\mathbf{Adv}^{\mathrm{prp}}_{\mathsf{AES}}(B) + \frac{3}{2^k}\right)\right] \\
&\leq (q+3q_{\mathsf{gs}})\left(2^{(k-m\gamma)/2} + \epsilon(L_m) + 2\hat{\epsilon}(L_m)\right) + 2(q+8q_{\mathsf{gs}})\left(\mathbf{Adv}^{\mathrm{prp}}_{\mathsf{AES}}(B) + \frac{3}{2^k}\right).
\end{aligned}$$

This completes the proof.

Discussion. This bound is very similar to the one provided by Theorem 8. Additionally, we note that if one only requires `RDRAND` to be secure (as opposed to both `RDRAND` and `RDSEED`), then one could instead obtain a bound similar to that of Theorem 7 by replacing the $\mathbf{Adv}^{\mathrm{fwd}/M}_{\mathsf{ISK},\mathcal{D}}(B_i)$ term in the proof with a $\mathbf{Adv}^{\mathrm{fwd-\texttt{RDRAND}}/M}_{\mathsf{ISK},\mathcal{D}}(B_i)$ term, upperbounding it with the value provided by that theorem.

Consequently, the discussion of Theorems 8 and 7 applies more-or-less intact to Theorem 10 and its `RDRAND`-only variant. The discussion can be found in Section 7.3 on pg. 22.

D Towards a notion of PWI availability

If a PWI can block, that immediately raises the question of availability. In general, however, any attempt to estimate the entropy contained in a sequence of input strings will necessarily be heuristic and subject to failure; imagine, for example, an entropy source that internally generates a random 128-bit AES key, and then begins outputting a CTR mode key stream. As long as AES is secure (as a PRP and hence a PRF, up until a large number of outputs have been produced), then no efficient test will be able to distinguish these outputs from random. Yet after the first couple outputs, the rest have essentially *zero* conditional min-entropy.

Still, entropy estimation and blocking may be reasonable ways to address specific, foreseeable failures with the entropy source (perhaps including certain types of hardware failure). Therefore we propose the following definition as a step toward capturing availability.

A PWI $\mathcal{P}$ is $(\mathcal{D}, t, q, q^*, \gamma^*, \epsilon)$-*available* if for all adversaries A making q queries and running in time t, the probability of A winning the above game (which is parameterized by q^* and γ^*) is at most ϵ. (Note that a definition that simply requires the PWI to not block would not be well-suited for PWIs that endeavor to provide truly random, rather than cryptographically random, bits; this is the case with the `RDSEED` interface, and, ostensibly, the `/dev/random` device in Linux.)

The basic idea is that the PWI should be available as long as it has gathered at least γ^* bits of entropy since q^* queries ago. However, we do not count any entropy gathered prior to the adversary tampering with state. (We do allow the adversary to *view* the state without penalty.) By making the entropy source $\mathcal{D}$ non-adversarial, we leave room to prove availability with specific (classes of) entropy sources.

The question of whether, say, ISK-RNG meets this definition of availability would require assumptions on $\mathcal{D}$ beyond simply that it provide high min-entropy See the discussion immediately following the proof of Theorem 2.

```
Oracle $\mathcal{D}$-refresh:
 $(\sigma, I, \gamma, z) \xleftarrow{\$} \mathcal{D}(\sigma)$
 $\gamma_t \leftarrow \gamma_t + \gamma$
 $S \leftarrow \mathsf{refresh}(S, I)$
 return $(\gamma, z)$

Oracle get-next:
 $(S, R) \leftarrow \mathsf{next}(S)$
 if $R = \perp$ then
    if $\gamma_\mu + \gamma_{\mu+1} + \cdots + \gamma_t > \gamma^*$ then
       $b \leftarrow 1$
 $t \leftarrow t + 1$
 if $\mu < t - q^*$ then
    $\mu \leftarrow t - q^*$
 return $R$

Oracle get-state:
 return $S$

Oracle set-state($S^*$):
 $\mu \leftarrow t$
 $\gamma_t \leftarrow 0$
 $S \leftarrow S^*$

Proc initialize:
 $\mathsf{seed} \leftarrow \mathsf{setup}$
 $\sigma \leftarrow 0$; $S \xleftarrow{\$} \{0,1\}^n$
 $\mu \leftarrow 0$; $t \leftarrow 0$
 $\gamma_0 \leftarrow \gamma^*$; $b \leftarrow 0$

Proc finalize:
 return $b$
```