Seedless Fruit is the Sweetest: Random Number Generation, Revisited

Sandro Coretti*[1], Yevgeniy Dodis†[2], Harish Karthikeyan‡[2], and Stefano Tessaro§[3]

[1] IOHK, Zurich, Switzerland
corettis@gmail.com
[2] New York University, New York, USA
{dodis,karthik}@cs.nyu.edu
[3] University of Washington, Washington, USA
tessaro@cs.washington.edu

Abstract. The need for high-quality randomness in cryptography makes random-number generation one of its most fundamental tasks.

A recent important line of work (initiated by Dodis et al., CCS '13) focuses on the notion of *robustness* for *pseudorandom number generators (PRNGs) with inputs*. These are primitives that use various sources to accumulate sufficient entropy into a state, from which pseudorandom bits are extracted. Robustness ensures that PRNGs remain secure even under state compromise and adversarial control of entropy sources. However, the achievability of robustness inherently depends on a *seed*, or, alternatively, on an ideal primitive (e.g., a random oracle), independent of the source of entropy. Both assumptions are problematic: seed generation requires randomness to start with, and it is arguable whether the seed or the ideal primitive can be kept independent of the source.

This paper resolves this dilemma by putting forward new notions of robustness which enable both (1) *seedless* PRNGs and (2) *primitive-dependent* adversarial sources of entropy. To bypass obvious impossibility results, we make a realistic compromise by requiring that the source produce sufficient entropy *even* given its evaluations of the underlying primitive. We also provide natural, practical, and provably secure constructions based on hash-function designs from compression functions, block ciphers, and permutations. Our constructions can be instantiated with minimal changes to industry-standard hash functions SHA-2 and SHA-3, or key derivation function HKDF, and can be downgraded to *(online) seedless randomness extractors*, which are of independent interest.

On the way we consider both a *computational* variant of robustness, where attackers only make a bounded number of queries to the ideal primitive, as well as a new *information-theoretic* variant, which dispenses with this assumption to a certain extent, at the price of requiring a high

*Work done while at NYU. Supported by NSF grants 1314568 and 1619158.

†Partially supported by gifts from VMware Labs, Facebook and Google, and NSF grants 1314568, 1619158, 1815546.

‡Supported by NSF grant 1619158.

§Partially supported by NSF grants CNS-1553758 (CAREER), CNS-1719146, CNS-1528178, and IIS-1528041, and by a Sloan Research Fellowship.

rate of injected weak randomness (as it is, e.g., plausible on Intel's on-chip RNG). The latter notion enables applications such as everlasting security. Finally, we show that the CBC extractor, used by Intel's on-chip RNG, is provably insecure in our model.

Keywords: provable security, pseudorandom number generation, provable security, symmetric cryptography

1 Introduction

Good random number generation is essential for cryptography and beyond. In practice, this difficult task is solved by a primitive called *pseudorandom number generator with input (PRNG)*, whose aim is to quickly accumulate entropy from various physical sources in the environment (such as keyboard presses, timing of interrupts, etc.) into the state of the PRNG and then convert this high-entropy state into (pseudo) random bits. In particular, entropy accumulation should never stop since one may need to recover from occasional compromises of the PRNG state. PRNGs are ubiquitous and have extensive applications. For example, virtually all operating systems come equipped with a PRNG; e.g., `/dev/random` [48] for Linux, Yarrow [34] for MacOs/iOS/FreeBSD, and Fortuna [24] for Windows [23], where the latter two make use of standard cryptographic primitives as part of their design. Still, as we will argue below in a much broader context, even these widely used PRNGs lack adequate theoretical understanding and analysis, which are critical if such PRNGs or their future tweaks continue to be used ubiquitously.

The situation is not better in terms of standardization efforts, where existing PRNG standards [32,35,22,5] are less mature than those for most other cryptographic primitives. For starters, there has not been any rigorous competition soliciting PNRG designs, and big parts of the existing standards concentrate on the difficult (ad-hoc and non-cryptographic) problem of entropy estimation rather than entropy accumulation and extraction. More importantly, standardized cryptographic PRNG constructions are rather ad-hoc, have no clear security definition/model, often have confusing syntax, and sometimes have been broken by subsequent analyses of the cryptographic community. The most widely known example is the DualEC PRNG, which appeared in the first version of the NIST SP 800-90A standard [5] in 2005 and remained there for years—despite early warnings by [42,44] and allowing potential exploitation [12]—until Snowden's revelations finally led to its deprecation. Recent work [49] identified a lot of gaps and imprecision (sometimes leading to attacks or security concerns) in the existing analyses and deployment for the other 3 PRNGs from the NIST SP 800-90A standard. In a similar vein, [43] found several gaps and misconceptions in previous analyses and security justifications for the popular Intel Secure Key Hardware PRNG introduced in 2011.

One of the main goals of this work is to reverse this poor state of affairs and to design a rigorous, theoretically sound model of PRNGs. This model should be general enough to incorporate practical entropy sources available in the real

world, as well as to formally prove security of "good," widely used PRNGs against realistic attackers.

1.1 Previous Theoretical Models for PRNGs: Seeds

In view of their practical importance, we are certainly not the first to formally study PRNGs through a theoretical lens. Indeed, several theoretical models and analyses of PRNGs have appeared in the literature [1,19,43,21,26,28]. While differing in various details, these important works share two key principles:

(a) The PRNG should work even against *adversarial* entropy sources, as long as such sources eventually provide enough entropy (such sources are called "legitimate" [19]);

(b) assuming more structure beyond entropy is undesirable and brittle,[4] as this requires a rather detailed understanding of one's entropy sources.

However, while such extremely minimalist assumptions make these PRNG models applicable to a wide variety of entropy sources, they also come with a subtle, but very important caveat: the randomness extraction module *cannot* be deterministic, as deterministic extraction from *general* entropy sources is impossible [15]. As a result, the PRNGs studied by these works are *seeded* (with the seed somehow chosen at initialization), but the entropy sources are assumed to be *independent of the seed*. This modeling is inherited from the underlying problem of randomness extraction, where *seeded extractors* [40] indeed overcome the impossibility of deterministic (or "seedless") extraction from general entropy sources.

While natural and sufficient for some applications of extractors, we argue that the need for a seed seems rather problematic in the deployment of PRNGs. First, if the reason for random number generation is the lack of access to high-quality random bits, then we may not have any way to generate the seed. More importantly, even if we *can* generate a uniformly random seed, it is crucial for the analysis that (potentially adversarial) entropy sources remain *independent* of the seed, for otherwise the extractor guarantees are lost. For example, if physical entropy sources inside the computer are used, these sources may be affected by the internal computations of the PRNG itself, and thus there may be correlations between the seed and the sources. Moreover, for many seeded PRNGs, the attacker could obtain information about the seed by either directly reading it from memory, or indirectly when the recently compromised or rebooted RNG is called on "low-entropy" inputs (so the output is no longer random and leaks information about the seed; this is called "premature next" attack by [21]).

This means that it is certainly an issue if the seed is just generated once and for all (perhaps using an expensive source of randomness) and hard-coded within implementations to be used for all future randomness extractions. Moreover, if multiple entropy sources are used, it is natural that some of these sources are adversarial and could depend on the seed (which is hard to protect against with a dedicated attacker). Somewhat paradoxically (considering the common belief that "more entropy cannot hurt"), the mixing of such seed-dependent sources

[4] We do, however, later discuss an interesting approach suggested by [3].

once again invalidates all the provable guarantees of seeded PRNGs, even if all the entropy is obtained from other, seed-independent sources.

We thus face a dilemma:

> *We want to support general entropy sources, for which seedless extraction is impossible, and seeded extraction is only possible under very dangerous and hard-to-ensure independence assumptions, which we would rather avoid.*

The goal of this work is to provide a meaningful solution to this dilemma, by keeping the PRNG design *seedless* while respecting properties (a) and (b) mentioned above.

1.2 *Seedless* PRNGs and Extractors from Cryptographic Hashing

We will achieve this goal by using popular *cryptographic hash functions (CHF)* as our technical tool, and by carefully defining the notion of entropy in the setting when certain components of these CHFs are assumed idealized.

WHY CRYPTOGRAPHIC HASHING? Before describing our solution in more detail, we explain why using CHFs appears essential for the design of *seedless*[5] PRNGs. For starters, all general-purpose software PRNGs used today, as well as all recommendations in existing PRNG standards, are based on CHFs. Hence, this setting must definitely be understood in order to provide results useful in the real world.

However, there is a more glaring theoretical reason as well. The key component of any PRNG is the shrinking function $\mathsf{refresh}$ which takes the current PRNG state S as well as a new entropic input X and produces a new state $S' \leftarrow \mathsf{refresh}(S, X)$. The goal of this function is to absorb the potential entropy of X into the PRNG state S, in which case the entropy of S' should be higher than the original entropy of S. In the extraction literature, this property is called *condensing*. If one uses a seed, building such condensers is easy to accomplish information-theoretically. For example, in the PRNG design of [19], the refresh function is linear: $S' = aS + X$, where a is a seed independent of X.

In the seedless/seed-dependent setting, it is not hard to see [20] that condensers must be built cryptographically, as they require at least some form of preimage- and collision-resistance.[6] For example, when used in iteration, the simple $aS + X$ condenser function above—which yields (together with other building blocks) a provably secure seeded PRNG construction [19]—can be broken *in a catastrophic way* if the distribution of the input blocks $X_1, X_2, \ldots$ could depend on the constant a: it is not hard to see that an attacker knowing a can rather easily produce *high-entropy inputs* such that if the condenser is applied to it, the resulting would have no entropy at all. In practice, one cannot imagine a PRNG system which would risk such a catastrophic failure by critically depending on

[5] Or, in the non-uniform setting, "seed-dependent"

[6] For example, the ability to compute a random preimage of a given element, which is known to imply one-way functions [31], allows the attacker to produce entropic inputs whose entropy is completely lost by the refresh procedure.

the fact that the constant a must remain hidden for the lifetime of the PRNG. Therefore, not surprisingly, all real-world PRNG designs—including those used by Windows, MacOS, and FreeBSD—critically rely on CHFs, despite lacking adequate theoretical justification.

Cryptographically secure condensers, which at an absolute minimum seedless PRNGs have to be, can be built using a (very strong form of) collision-resistance [20]. However, the types of condensers needed for applications, called average-case seedless condensers, seem to require non-standard cryptographic assumptions. For example, a relatively weak form of such average-case condensers (called "condensers for leaky sources") are already sufficient for instantiating the Fiat-Shamir heuristic for public-coin proof systems [20]—and it is a major open problem to provide such an instantiation under standard cryptographic assumptions.

To put it differently, even ignoring the fact that we want our PRNGs to be full-blown *seedless extractors*—a problem we will address next—just achieving provably secure entropy accumulation appears to require the use of CHFs as well as either (1) non-standard cryptographic assumptions (making the results appear somewhat tautologous) or (2) some supporting justification argument in an idealized model of computation, which is the approach taken by this work.

Our approach: new min-entropy notion. To describe our approach, it is instructive to recall the basic impossibility of seedless extraction for general entropy sources. Given any candidate (seedless) extractor G, an adversary can perform a so-called *extractor-fixing attack* by sampling a random input X several times until the first bit of $G(X)$ is 0. The resulting distribution X has very high entropy, but $G(X)$ is clearly not uniformly random. Observe that with a strong enough CHF G, one might be able to formally argue that the extractor-fixing attack is the "most damaging" attack possible; for example by showing, that $G(X)$ has almost full entropy (i.e., is a good condenser) for any efficiently samplable source X, as was done by [20]. In other words, using CHFs will protect against the completely devastating attacks possible with information-theoretic extractors.

However, our goal is to have a meaningful model where real randomness *extraction* is possible, so that we can later extend it all the way to the full PRNG system. Our solution will be to define a elegant and practically motivated refinement of general min-entropy *in settings where CHFs exist*, so that:

(a) somewhat artificial sources resulting from intentionally performing extractor-fixing will not have much entropy according to our notion (meaning they are no longer "legitimate"); in fact, seedless extraction will become *possible* for our notion of min-entropy;

(b) most natural entropy sources, including those used by major operating systems, will likely have good entropy according to our new measure.

While our final constructions and interpretation of our security analyses will apply to real-world CHFs, such as those derived from SHA-2, SHA-3, HMAC or HKDF, at present the only rigorous way we know how to achieve our ambitious goals (a) and (b) will be by going to the idealized models of computation,

such as the random oracle, the ideal cipher or the random permutation model. This is quite standard for many areas of symmetric-key cryptography, and we already indicated that doing provably secure (non-tautologous) seedless PRNG constructions in the "standard model" appears beyond our current capabilities, even for much simpler building blocks, such as (average-case) seedless condensers.

1.3 Toy Case: Monolithic Seedless Extraction from Oracle-Dependent Sources

We start by presenting our new entropy notion for the simpler problem of "monolithic randomness extraction," where the entropy source X is assumed to come in one piece (rather than slowly accumulated using a fixed-length PRNG state), and a monolithic CHF G—modeled as a monolithic random oracle—is used to output the value $R = G(X)$ (so that we temporarily ignore any find-grained structure inside G, such as Merkle-Damgård or Sponge [8] iteration).

At first, it appears that we solved our problem in a totally trivial (and uninteresting) way, even without refining standard min-entropy. Namely, in the random oracle model, the following folklore proof (see [18]) appears to show that a (seedless) random oracle G is a good extractor: For any min-entropy γ^* source X, the probability the distinguisher D can distinguish $G(X)$ is upper bounded by the probability D queries G on X, which is at most $q \cdot 2^{-\gamma^*}$, where q is the number of random oracle queries allowed to $\mathcal{A}$.[7]

Implicit in this simple proof, however, is the key assumption that the distribution X is independent of the random oracle G, meaning that our (potentially adversarial) sampler producing X is not allowed to call the random oracle G. Thus, modeling G as a random oracle is but a fancy way of introducing an exponentially long seed that is independent of the source, making extraction trivial.[8] Indeed, to capture PRNG sources X arising in the real world, we must allow the source X to depend on the ideal primitive G. For example, if the timing of computer interrupts is used as our entropy source X—which is the most common source of randomness in software PRNGs—it seems unreasonable to assume that none of these interrupts could be affected by frequent hash function computations done inside and outside the operating system.

ORACLE-DEPENDENT SOURCES. To fix this problem, in Section 3 we will explicitly model our source as part of the attacker $\mathcal{A}$, so that $\mathcal{A}^G = (\mathcal{A}_1^G, \mathcal{A}_2^G)$, where $\mathcal{A}_1^G$ outputs the *oracle-dependent* source X and passes state Σ to the second state attacker $\mathcal{A}_2^G(\Sigma)$, whose goal is to distinguish $R = G(X)$ from uniform. Of course, for this definition to make sense, we must require that X is "legitimate," meaning it has entropy at least γ^* given the state information Σ (for some parameter γ^*).

[7] In fact, if the length of $G(X)$ is slightly less than γ^*, we can even let $\mathcal{A}$ query all of G and use leftover-hash lemma [30] to get information-theoretic security.

[8] Prior to our work, the above modeling of sources as being independent of the ideal primitive, was the only way to overcome extractor-fixing attacks. Examples of this approach include [18,36,49] and many others. While these results are non-trivial due to the "non-monolithic" structure of their extractors G, none of these works model the setting where *the source could depend on the ideal primitive*.

In the standard model, this could be formalized by requiring $\mathrm{H}_\infty(X|\Sigma) \geq \gamma^*$ (see Section 2). But this is too weak, as this still allows for extractor-fixing attacks, by sampling a long random X and remembering a few bits of $G(X)$ in the leakage Σ. In fact, this extractor-fixing attack still works even if we condition on the entire random oracle G (i.e., require $\mathrm{H}_\infty(X|\Sigma, G) \geq \gamma^*$). This leads to a central question of this work:

What is the "right" notion of entropy for oracle-dependent sources X?

The key insight of our work comes from the fact that while it is reasonable to assume that the source X could *depend* on the random oracle G, the natural sources of entropy we want to extract from do not natively evaluate cryptographic hash functions, but somehow add extra entropy *in addition* to all hash function evaluations around them. For example, it is unreasonable to assume that the timing of interrupts could not depend, even slightly, on various hash function evaluations inside the computer. However, it seems that the real entropy of interrupt timings comes from the fact that the attacker cannot perfectly predict the exact lower order bits of the timing measurements, *even if the attacker knew all the hash function evaluations*. Indeed, instead of only requiring that $\mathrm{H}_\infty(X|\Sigma, G) \geq \gamma^*$, our approach will make a stronger requirement that

$$\mathrm{H}_\infty(X|(\Sigma, \mathcal{L})) \geq \gamma^* \ , \tag{1}$$

where $\mathcal{L}$ *is the* input-output *list of random oracle queries made by the sampler $\mathcal{A}_1$ to the random oracle*. Another, equivalent way to interpret this legitimacy condition is to mandate that $\mathcal{A}_1$ cannot "forget" any of its random oracle queries when passing its state Σ to $\mathcal{A}_2$, but must forget some other useful information about X, to ensure that X has entropy conditioned on Σ *and* $\mathcal{L}$.

Notice, our solution places a more stringent requirement than conditioning on the entire G, as $\mathcal{A}_1$ did not touch anything outside $\mathcal{L}$, so these un-queried values do not reduce entropy of X beyond what is done by $\mathcal{L}$. Also, when the number of queries q is not too large, the extractor-fixing is no longer a legitimate attack, since X will not have much entropy when conditioned on $\mathcal{L}$ (which contains the pair $(X, G(X))$). In fact, we can easily show *full extraction* (see Theorems 1 and 2), along the lines of the folklore proof for oracle-independent sources mentioned above. The basic intuition comes from the fact that our conditioning on the list $\mathcal{L}$ ensures that with overwhelming probability the sampler $\mathcal{A}_1$ did not himself evaluate $G(X)$, which is essential for the extractor-fixing attack to succeed.

Did we go too far? Of course, the main question is whether the legitimacy requirement $\mathrm{H}_\infty(X|(\Sigma, \mathcal{L})) \geq \gamma^*$ does not overly limit the class of high-entropy sources from which we want to extract. We believe the answer is negative. First, in the restrictive "folklore case" when X is independent of G (meaning $\mathcal{L} = \emptyset$), we get the best-possible min-entropy condition $\mathrm{H}_\infty(X|\Sigma) \geq \gamma^*$ we had in the standard (non-random-oracle) model. Namely, our notion of min-entropy relative to G includes all general min-entropy sources.[9]

[9] Of course, when we instantiate G with a real-world hash function, this is no longer the case, as we discuss below.

Second, while we certainly allow the source X to substantially depend on G, we ensure that non-trivial bulk of entropy must come from *outside* of the actual oracle evaluation queries. In other words, while "nature," who outputs X, could conceivably be influenced by a couple of hash function evaluations, it should generate some intrinsic entropy *in addition to* (but possibly dependent on!) these evaluations. We feel that all practically used physical sources (timing of interrupts, temperature, keystroke dynamics, etc.) have very little to do with hash functions, and should easily satisfy this requirement.

Thus, we believe that our technical restriction on the legitimacy for extraction using CHFs—by conditioning min-entropy on the list of hash function evaluations—strikes the right balance between allowing for seedless extraction, and yet keeping the family of high-entropy sources large and realistic for applications.

1.4 Our Results

While the above toy example (analyzed in Section 3.2) illustrated the key technical insight behind our approach, in practice it is uncommon to assume access to a monolithic random oracle G. Instead, practical hash functions are usually built from (public) compression functions, ciphers, or permutations. These underlying primitives P have limited input length and will therefore not be able to process inputs of arbitrary length m. Therefore, extractors and PRNGs should be designed in such a way that they can process short m-bit input blocks (e.g., $m = 256, 512, 1600$) and accumulate their entropy in the internal state.

Online extractors and insecurity of CBC. Thus, in Section 3.3 we formalize the more realistic notion of *online (seedless) extractors*, which slowly accumulate their long input into a fixed-length state (using access to a P), and then finalize their output once the whole input is processed. We also define both computational and information-theoretic (IT) notions of online extractor security, where in the latter notion the attacker is allowed to read the entire ideal primitive P after it finished generating the oracle-dependent source X.

Turning to natural and widely used examples of such online extractors, we show that the popular CBC mode of operation is insecure as a seedless extractor in our framework. The details of our attack are given in Section 3.4, but the result is a somewhat unexpected, since CBC is used as the extractor underlying the `CTR_DRBG` construction in the NIST PRNG standard NIST SP 800-90A Rev. 1 [4], and also as the extractor for Intel's on-chip RNG [38]. Moreover, its security was formally shown by Dodis et al. [18], but in the setting where the entropy source X was *independent* of the random permutation π. In contrast, we show that once the latter assumption is relaxed to our oracle-dependent sources, the CBC extractor is no longer secure (unless one generalizes it to the Sponge construction in Section 5.3, where the input is only XORed to *part* of the state). Of course, our attack is somewhat theoretical, and does not directly translate to attacking the Intel on-chip RNG, for example. However, coupled with our positive results, we feel our attack suggests using a different online extractor, if possible.

On a positive side, in the full version [17], we show several other (both computational and information-theoretic) online extractors based on popular modes

of operations used inside hash functions SHA-2 and SHA-3, which are provably secure in our framework: from Merkle-Damgård with a random compression function, from Merkle-Damgård with the Davies-Meyer compression function, and from Sponges. Hence, for the first time practitioners can use seedless extractors which are both practical and have firm theoretical foundation. The security of these natural online extractors follows as special cases of more general PRNG security results, which we describe next.

Full-scale seedless PRNGs. Finally, we take all our ideas together to solve our main problem: defining and building practical, yet provably secure seedless PRNGs. In Section 4 we introduce a novel security definition for PRNGs that differs from previous notions [19,1,26] in several crucial ways. The detailed comparison appears in the full version, but we present the highlights here.

First and foremost, our design is seedless. This is accomplished by carefully defining the legitimacy condition (relative to the fixed-length ideal primitive P), by conditioning our entropy notion on the list $\mathcal{L}$ of the queries to P made by the attacker. Second, our seedless design allows us to merge the "distribution sampler" and the distinguisher used by [19,26] into a single attacker $\mathcal{A}$,[10] making our notion much simpler to describe. Third, the works of [19,26] used a much weaker notion of worst-case min-entropy; moreover, the final entropy of the the source X was defined as sum of individual worst-case min-entropies of the individual blocks of X conditioned on all the other blocks (before and after). In contrast, we use a much better notion of *average-case* min-entropy, and only look at the global average-case min-entropy of the entire (long) vector X. Thus, our notion of entropy is much less conservative: realistic entropy sources are likely to have *much higher* entropy according to our definition, even when conditioning on the list $\mathcal{L}$. Fourth, the notion of [19,26] had explicit "entropy estimates" that the attacker had to provide. Our notion gets rid of these estimates. Finally, and somewhat surprisingly, we still managed to define our notion of legitimacy of the entropy source in a manner which is *construction-independent*. This means that one can potentially study the entropy properties of the source in a manner independent of the PRNG used on this source.

We also define both computational and information-theoretic (IT) notions of PRNG security. As with on-line extractors, for IT-PRNGs the attacker is allowed to read the entire ideal primitive P after it finished generating the last block of it's oracle-dependent source X used for extraction. Such a notion is important for applications where privacy must hold well after the PRNG is finished its operations, or where information-theoretic security is important.

Our PRNG constructions. In Section 5 we then present three main PRNGs which are provably secure in our framework: based on Merkle-Damgård with a random compression function (see Figure 2), based on Merkle-Damgård with the Davies-Meyer compression function (see Figure 3), and based on Sponges (see Figure 4). All these constructions are extremely natural and practical, as Merkle-

[10] Since we no longer need to hide the seed from the distribution sampler, forcing us to separate it from the attacker.

Damgård-based functions abstract SHA-2, while Sponges abstract SHA-3—two most widely used cryptographic hash functions. Thus, our work (including new notion of oracle-dependent entropy) could be used as theoretical justifications why these popular hash functions yield good *seedless* PRNGs (as well as online randomness extractors) even for a wide class of oracle-dependent entropy sources.

Moreover, for Merkle-Damgård based variants we also proved the security for the information-theoretic variant (the Sponge case is open, although we defined the variant which we conjecture is IT-secure). Our three computational proofs heavily use the "coefficient-H" technique [41,13], while our two information-theoretic proofs extend the framework of so-called "graph-counting" proofs [18,7,25] to bound the collision probability of iterated hash constructions. One novel challenge we had to solve here comes from the fact that the input source could depend on the list $\mathcal{L}$ of the ideal primitive queries, which breaks the "source-primitive" independence assumption crucially used in these already subtle proofs.

We also showed numeric examples of how we propose to use our constructions. Overall, we believe all of them are deployment ready, and we hope this work will start influencing future PRNG deployments, and will be incorporated into next RNG standards.

Implications to standard model. To overcome the impossibility of seedless extraction, our entropy notion is defined relative to the ideal primitive P. As we argued in detail in Section 1.2, working in the idealized model seems somewhat inherent to our approach, provided we wish to avoid highly non-standard, and likely tautological, cryptographic assumptions about the CHF we are using in the standard model. Still, it is good to ask what one might expect from our extractor and PRNG constructions with real-world CHFs, such as those based on SHA-2, SHA-3, HKDF, etc. As we already mentioned, we believe these constructions are secure for real-world entropy sources, because our idealized notion of entropy informally corresponds to sources which have fresh entropy, even given all the hash function evaluations happening around the source. To state the counter-positive, we believe that any real-world attack against our constructions with existing hash functions will either require a highly artificial entropy source, or will find a surprising weakness in the corresponding CHF.

1.5 Other Related Work

We mention some important categories of related works, in particular with respect to seedless extraction, PRNGs, and their security.

Seeded extractors and PRNGs. We already mentioned the extensive work on seeded extractors started by the seminal paper of Nisan and Zuckerman [39], and why they are problematic in our context. In the context of PRNGs, the first seeded PRNG notion was defined and constructed by Dodis et al. [19], who extended the prior "monolithic PRNG" definition of Barak and Halevi [1] (which did not explicitly talk about the seed, assuming the extraction module is "good enough" for the class of distributions produced by the entropy source). This line of work was extended in various ways by [21,26,29], where the latter two works were also analyzed in the random permutation model (in addition to the seed).

However, none of these works considered a seedless setting for general entropy sources.

Extractors and PRNGs in ideal models. Extractors and PRNGs were also studied in the ideal models by several works [18,9,43,49]. While not having explicit seeds, these works nevertheless modeled the *entropy source as being independent of the ideal primitive*. As we argued above, such oracle-independent modeling seems to be too restrictive for many realistic scenarios. Also, from a theory point of view, it effectively allows an exponentially long seed (the randomness used to sample the corresponding ideal primitive), making the positive results less interesting theoretically than the above-mentioned work on seeded extractors and PRNGs.

Indeed, the main motivation of all these papers was not to design theoretically optimal extractors and PRNGs, but to analyze the heuristic use of various cryptographic hash functions and popular modes of operations (such as CBC, HMAC, etc.) for randomness generation and extraction—a task these objects were not natively designed for. From this perspective, and given their widespread use, analyzing their extraction properties was an important first step in understanding their security, even under the restrictive oracle-independence assumption. Our work could be viewed as making a critical leap forward, by dropping—for the first time—the oracle-independence assumption, but instead carefully modeling what constitutes entropy in the much more realistic, oracle-dependent setting.

Restricting the class of entropy sources. This line of work has primarily focused on the question of extraction, by assuming that the source X has more structure beyond entropy. Early examples [47,16,10,37,14] include various bit-fixing and limited dependence sources, culminating with the question of extracting from several independent sources [2,11]. While mathematically very elegant, the types of sources studied by these works appear "too structured" to be realistic in the PRNGs scenario.

A different kind of restriction on the entropy source was studied by Barak et al. [3]. Rather than restrict sources by some property of their distribution, the work of [3] allows for arbitrary min-entropy sources, but assumes they come from an a-priori bounded number of distributions. While potentially promising for the setting of PRNGs, there are two disadvantages of the work of [3] as compared to this work. First, the work of [3] concentrated on the "monolithic" extraction setting, and did not address the question of entropy accumulation, where the entropy in X might come slowly from a large number of samples, and has to be accumulated into bounded state. In particular, it is unclear how to extend their constructions to address entropy accumulation with a fixed-length state. Second, the particular solutions offers by [3] used so called t-wise independent hash functions for a large values of t (at least as large as the overall source length). These functions are quite inefficient, and might not be fast enough for general purpose PRNGs.

We note that our work could also be viewed as overcoming impossibility of extraction by restricting the type of the source. However, we feel that our

modeling is more natural for (and, thus, applicable to) the existing entropy sources, as used by the current PRNGs.

Low-Complexity Samplers. Introduced by Trevisan and Vadhan [46] and later extended by [33], here one assumes that the entropy source producing input X is unable to run the extractor/PRNG even once, thus making it impossible to do extractor-fixing. While this might be useful for situations where the entropy source is extremely simple, it is too restrictive for most applications, such as general purpose PRNG design studied in this work. In contrast, in this work the entropy source can easily run the extractor, but the legitimacy condition is defined in a way that doing the trivial extractor-fixing attack—by running the extractor—will result in a low-entropy, "illegitimate" source.

Randomness condensers. This approach, formalized by Dodis, Ristenpart and Vadhan [20], relaxes the security guarantees of the randomness extractor to only ensure that the output of the (seedless or "source-dependent-on-seed") condenser is almost full entropy, despite not being perfectly uniform. Indeed, this weaker security turns out to be sufficient for several applications, such as key derivation schemes for signature schemes. Unfortunately, if we want an extractor rather than a condenser—which is essential for general purpose PRNGs—this approach is not sufficient.

UCEs and public-seed pseudorandomness. The notion of universal computational extractors (UCEs) [6], and its generalizations [45], study a complementary problem to the one studies here: how to extract from any entropy source which is only *computationally*-hard-to-predict, so it only has "computational entropy". On a positive, and similar to this work, when instantiated with constructions from an ideal primitive P, a UCE hash function yields a good extractor even if the inputs to it (the actual source) can be sampled depending on the ideal primitive. The issue, however, is that the current UCE notion inherently *requires a seed*, making in inapplicable for the PRNG scenario. An interesting direction for future research could be to extend our work to deal with computational entropy, by defining and constructing *seedless* UCEs in idealized models, and possibly extending them to full-blown seedless PRNGs for computational entropy.

2 Preliminaries

2.1 Statistical Distance and Min-Entropy

The *statistical distance* of two random variables X and Y is $\mathsf{SD}(X,Y) = \frac{1}{2}\sum_x |\mathsf{P}[X=x] - \mathsf{P}[Y=x]|$. The *prediction probability* of a random variable X is $\mathsf{Pred}(X) := \max_x \mathsf{P}[X=x]$, and we also denote $\mathsf{Pred}(X|y) := \max_x \mathsf{P}[X=x|Y=y]$. The conditional version of prediction probability is defined as

$$\mathsf{Pred}(X|Y) \;:=\; \mathbf{E}_{y\leftarrow Y}\big[\mathsf{Pred}(X|y)\big]\ .$$

The *(average-case) conditional min-entropy* is $\mathrm{H}_\infty(X|Y) = -\log(\mathsf{Pred}(X|Y))$.

2.2 Security Games

All of the security properties considered in this paper are captured by considering a game between a challenger and an attacker $\mathcal{A}$, both of which may have access to an ideal primitive P. The goal of the attacker is to guess a random bit b chosen by the challenger, who offers a set of oracles to the attacker to aid with this task. The *advantage* of $\mathcal{A}$ is defined as

$$2 \cdot \left| \, \mathsf{P}[\mathcal{A} \text{ wins}] - 1/2 \, \right| ,$$

where the probability is over the randomness of $\mathcal{A}$, of the challenger, and of the ideal primitive. The cases where $b = 0$ and $b = 1$ are referred to as the *real world* and the *ideal world*, respectively. One may equivalently consider $\mathcal{A}$'s advantage at telling these two worlds apart, i.e.,

$$\left| \, \mathsf{P}[\mathcal{A} = 1 | b = 0] - \mathsf{P}[\mathcal{A} = 1 | b = 1] \, \right| .$$

3 Seedless Extraction

As a warm-up for full-fledged seedless PRNGs, this section considers the simpler property of *extraction*, i.e., producing uniformly random bits from weak high-entropy sources. Extraction can be seen as corresponding to the post-compromise security of PRNGs, and as such it will be implied by PRNG *robustness* (as defined in Section 4.2).

The definition of extraction security in Section 3.1 considers the entropy of the attacker's input to the extractor conditioned on the attackers *state* and the *queries* made to an ideal primitive P. A definition is provided for *computational* or *information-theoretic* security. IT extractors differ from computational ones in that the output of the extractor remains random even if the attacker, *after providing the input*, is given the entire function table of the underlying ideal primitive. That is, IT extractors achieve so-called *everlasting security* (cf. works in the hybrid bounded-storage model by Harnik and Naor [27]).

Section 3.2 considers extracting with a *monolithic random oracle*. The corresponding security proofs (for the computational and IT cases) are instructive for understanding the actual PRNG constructions provided in Section 5. Since considering a monolithic oracle is not motivated by any hash function used in practice, Section 3.3 introduces the concept of *online* extraction. An online extractor accumulates the entropy of its inputs in an internal state, from which uniform randomness can be produced. Finally, in order to illustrate the non-triviality of online extraction, Section 3.4 shows that extractors based on the popular CBC mode are not suitable for extraction.

3.1 Definition

In a model with idealized primitive P (chosen from some set $\mathcal{P}$), seedless extractors are algorithms $\mathsf{ext}^P : \mathcal{X} \to \mathcal{Y}$ with oracle access to P. The security definition for

such extractors considers a two-stage attacker $\mathcal{A} = (\mathcal{A}_1, \mathcal{A}_2)$, where both parts have access to P. The first stage $\mathcal{A}_1$ outputs a value x and some state information σ for $\mathcal{A}_2$. The second stage takes an input $y \in \mathcal{Y}$ and outputs a single bit (i.e., it acts as a distinguisher).

For an attacker $\mathcal{A}$, denote by $\mathcal{L}_1$ and $\mathcal{L}_2$ the (random variables corresponding to) the lists of the P-queries made by $\mathcal{A}_1$ and $\mathcal{A}_2$, respectively.

Definition 1. *An attacker $\mathcal{A} = (\mathcal{A}_1, \mathcal{A}_2)$ is called a* q-attacker *if $|\mathcal{L}_1 \cup \mathcal{L}_2| \leq q$ always; it is called a* q-IT-attacker *if $|\mathcal{L}_1| \leq q$ always.*

That is, for IT-attackers the second stage $\mathcal{A}_2$ may make an arbitrary number of queries to P. Equivalently, $\mathcal{A}_2$ can be thought of as being given the entire function table of P.

The security game for seedless extractors in the P-model roughly requires that if the extractor is given a high-entropy input by $\mathcal{A}_1$, then $\mathcal{A}_2$ cannot tell the extractor output apart from a random value in $\mathcal{Y}$, even given the state information σ and access to P. Formally, it proceeds as follows:

1. The challenger chooses $b \leftarrow \{0,1\}$ and $P \leftarrow \mathcal{P}$ uniformly at random.
2. $\mathcal{A}_1$ gets access to P and produces $(\sigma, x) \leftarrow \mathcal{A}_1^P$.
3. The output of the extractor is computed as $y_0 \leftarrow \mathsf{ext}^P(x)$. Moreover, the challenger picks a value $y_1 \leftarrow \mathcal{Y}$ uniformly at random.
4. The second-stage attacker $\mathcal{A}_2$ is given σ and y_b and outputs a decision bit $b' \leftarrow \mathcal{A}_2^P(\sigma, y_b)$. The attacker wins if and only if $b' = b$.

The advantage of $\mathcal{A}$ in this extraction game is denoted by $\mathrm{Adv}_{\mathsf{ext}}^{\mathrm{ext},P}(\mathcal{A})$.

An attacker has to satisfy a legitimacy condition. Intuitively, this condition requires that the output X of $\mathcal{A}_1$ have high min-entropy even conditioned on the state information Σ and the list of queries $\mathcal{L}_1$.[11]

Definition 2. *An attacker $\mathcal{A} = (\mathcal{A}_1, \mathcal{A}_2)$ is said to be* γ^*-legitimate *if, in the extraction game above,*

$$\mathrm{H}_\infty(X|\Sigma\mathcal{L}_1) \;\geq\; \gamma^* \;.$$

The above finally leads to the following definition of seedless extractor in the P-model:

Definition 3. *An algorithm $\mathsf{ext}^P : \mathcal{X} \to \mathcal{Y}$ with oracle access to P is an* $(\gamma^*, q, \varepsilon)$-(IT-)extractor in the P-model *if for every γ^*-legitimate q-(IT-)attacker $\mathcal{A}$,*

$$\mathrm{Adv}_{\mathsf{ext}}^{\mathrm{ext},P}(\mathcal{A}) \;\leq\; \varepsilon \;.$$

[11] Note, in the extraction game the definition of $\mathcal{L}_1$ is the same in the real and the ideal worlds. For our future definitions of PRNGs, however, it will be important that the notion of legitimacy is defined in the ideal world (i.e., conditioned on $b = 1$).

3.2 Seedless Extraction with a Monolithic Random Oracle

For instructive purposes it is useful to consider monolithic extraction, i.e., the case where the ideal primitive P itself is used as an extractor. To exemplify this, assume P is a random oracle, i.e., a function $G : \{0,1\}^m \to \{0,1\}^n$ chosen uniformly at random. Then, the monolithic extractor is defined as follows:

Construction 1 (Monolithic extractor). *The monolithic seedless extractor* $\mathsf{mono}^G : \{0,1\}^m \to \{0,1\}^n$ *using a random oracle* $G : \{0,1\}^m \to \{0,1\}^n$ *is defined by*

$$\mathsf{mono}^G(x) := G(x) \ .$$

Theorem 1 (Monolithic seedless extraction). *Construction* mono *is a* $(\gamma^*, q, \varepsilon)$*-extractor in the* G*-model for*

$$\varepsilon \ \leq \ \frac{q}{2^{\gamma^*}} \ .$$

The proof of Theorem 1 is a straight-forward application of the H-coefficient technique. The idea is to first show that unless $\mathcal{A}_1$ or $\mathcal{A}_2$ queries the input x provided by $\mathcal{A}_1$, the real and ideal worlds (i.e., the cases where $b = 0$ and $b = 1$, respectively) are indistinguishable. That is, the corresponding ratio of transcript probabilities is 1. Transcripts where x is in the query list are defined to be *bad* transcripts, and the second part of the proof shows that bad transcripts are unlikely to occur due to the legitimacy of $\mathcal{A}$. The latter proof crucially relies on the fact that the H-coefficient technique enables performing the bad-event analysis in the *ideal* world. The proof of the following theorem is deferred and can be found in the full version.

Theorem 2 (Monolithic seedless IT-extraction). *Construction* mono *is a* $(\gamma^*, q, \varepsilon)$-IT-extractor in the G-model *for*

$$\varepsilon \ \leq \ \frac{1}{2}\sqrt{\frac{2^{-(\gamma^*-n)}}{1-\rho}} + \rho \ ,$$

where $\rho = q/2^{\gamma^*}$.

The proof of Theorem 2 proceeds by bounding the statistical distance of $\mathcal{A}_2$'s views in the real and ideal experiments via the corresponding collision probabilities (as done in the proof of the left-over hash lemma). In the proofs of the actual PRNG constructions in the following sections, bounding said collision probabilities constitutes the bulk of the proof and is quite involved. The formal proof is deferred and can be found in the full version.

Parameter choices. In terms of concrete parameters, observe the following for the constructions towards monolithic seedless extraction from above:

– *Computational:* If we let $n = 512$ and $q = 2^{80}$. We would need $\gamma^* \approx 160$ to get 80 bits of security.

– *Information Theoretic:* We let $n = 512$. We also approximate $1/(1-\rho) \leq 2$, very generously Then, if we set for example $q = 2^{80}$. We would need the entropy loss, i.e, $\gamma^* = 160$ for 80 bits of security.

3.3 Online Extraction

An "accumulating" extractor ext satisfies the same security Definition 3, but its syntax can be thought of as two algorithms $\mathsf{ext} = (\mathsf{refresh}, \mathsf{finalize})$, where $\mathsf{refresh}$ accumulates entropy in an internal state and $\mathsf{finalize}$ produces the extractor output from the current state.

Definition 4. *An* online extractor construction *consists of two algorithms* $\mathsf{ext} = (\mathsf{refresh}, \mathsf{finalize})$*, where*

– $\mathsf{refresh}$ *takes a state* s *and an input* $x \in \{0,1\}^m$ *and produces a new state* $s' \leftarrow \mathsf{refresh}^P(s, x)$*, and*
– $\mathsf{finalize}$ *takes a state* s *and produces an output* $y \in \{0,1\}^r$*, i.e.,* $y \leftarrow \mathsf{finalize}^P(s)$*.*

An online extractor processing m*-bit inputs and producing* r*-bit output is called a* (m, r)-online extractor*.*

The security definition for online extractors additionally considers the number ℓ of times $\mathsf{refresh}$ is called by the attacker, i.e., it considers (q, ℓ)-attackers.

Definition 5. *An algorithm* $\mathsf{ext}^P : \mathcal{X} \to \mathcal{Y}$ *defined by two algorithms* $\mathsf{ext} = (\mathsf{refresh}, \mathsf{finalize})$ *with oracle access to* P *is an* $(\gamma^*, q, \ell, \varepsilon)$-(IT-)online extractor in the P-model *if for every* γ^**-legitimate* (q, ℓ)*-(IT-)attacker* $\mathcal{A}$*,*

$$\mathrm{Adv}^{\mathsf{ext},P}_{\mathsf{ext}}(\mathcal{A}) \;\leq\; \varepsilon \;.$$

Online extractors can be built just like the PRNG constructions in Section 5, and, in fact, the corresponding security results follow as a special case of PRNG security. Correspondingly, their treatment is deferred until Section 5, where such online extractors (and, in fact, full-fledged PRNGs) can be obtained from Merkle-Damgård with a random compression function, from Merkle-Damgård with the Davies-Meyer compression function, and from Sponges. For the reader's convenience, the full version of this paper [17] contains the online extractor constructions along with the security bounds—for applications where extraction is sufficient.

In contrast to Merkle-Damgård and Sponges, as shown in the next section, using the CBC paradigm (which can be thought of as an "extreme sponge") will not lead to a secure online extractor.

3.4 CBC-Based Extractors Are Insecure

A natural candidate for an online seedless extractor is using a permutation in CBC mode. A CBC-based extractor construction uses a permutation $\pi : \{0,1\}^n \to \{0,1\}^n$ to absorb n-bit inputs. Its refresh function is defined as

$$\mathsf{refresh}^{\pi}(s, x) \;=\; \pi(s \oplus x) \;.$$

However, it turns out that this approach does not lead to a secure extractor. This section presents a simple attack against CBC-based extractors. The attack works irrespective of how the finalization function is defined.

Theorem 3 (Attack against CBC Extractors). *Let* refresh *as defined above. There exists an ℓ-legitimate q-attacker $\mathcal{A}$ with black-box access to a function* finalize*, such that for all* $\mathsf{CBC} = (\mathsf{refresh}, \mathsf{finalize})$

$$\mathrm{Adv}^{\mathrm{ext},\pi}_{\mathsf{CBC}}(\mathcal{A}) \;=\; 1 - 2^{-(r-1)} \;,$$

where r is the output length of the extractor, $q = 2\ell + 2\alpha$, and α is the query complexity of finalize.

The idea of the attack is to have the attacker create the i^{th} input block as either $\pi^i(0^n) \oplus \pi^i(1^n)$ or 0, each with probability 1/2.[12] After ℓ such steps, the attacker will have provided ℓ bits of entropy (even conditioned on its π-queries), but only a single bit will have accumulated in the state, which will be $\pi^i(0^n)$ or $\pi^i(1^n)$, each with probability 1/2.

The proof can be found in the full version of this paper [17].

4 Pseudorandom Number Generators with Input

A *pseudorandom number generator with input (PRNG)* is a stateful cryptographic primitive. It gradually accumulates entropy in its state by absorbing inputs and can be used to output pseudorandom bits once the entropy of the state is sufficiently high. Moreover, it is both forward and backward secure, i.e., past outputs remain random upon future state compromise, and, by absorbing sufficient amounts of entropy, a PRNG can recover from state compromise.

This section introduces a novel security definition for PRNGs that differs from previous notions in several crucial ways. Specifically, a comparison to the original *robustness* notion by Dodis *et al.* [19], based on work by Barak and Halevi [1], as well as to an adaptation of it by Gazi and Tessaro [26] for idealized models is provided in the full version which is available on ePrint.

This paper considers two notions of PRNGs: computational PRNGs and information-theoretically secure (IT) PRNGs. IT PRNGs differ from computational PRNGs in that once the attacker stops interacting with the PRNG, the output of the PRNG remains random even if the attacker is given the entire function table of the underlying ideal primitive. That is, IT PRNGs achieve so-called *everlasting security* (cf. works in the hybrid bounded-storage model by Harnik and Naor [27]). This distinction is analogous to that between seedless extractors and IT seedless extractors (cf. Section 3).

4.1 Syntax

A PRNG consists of two algorithms: one for absorbing new inputs and one for producing pseudorandom outputs. Formally, it is defined as follows:

[12] Here, π^i denotes the i-fold application of π.

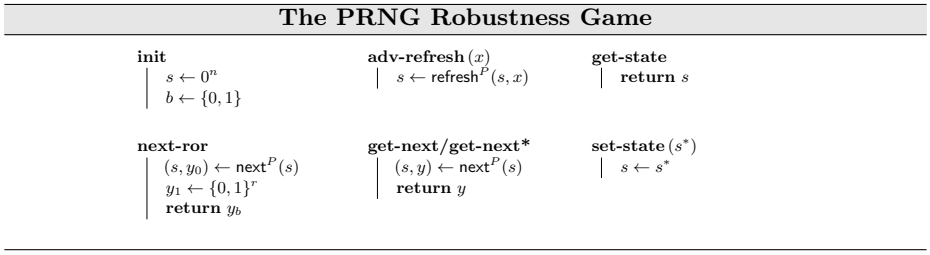

Fig. 1. Oracles for the PRNG robustness game.

Definition 6 (Syntax of PRNGs). *A* pseudorandom number generator with input (PRNG) *is a pair of algorithms* $\mathsf{PRNG} = (\mathsf{refresh}, \mathsf{next})$ *having access to an ideal primitive* P *and sharing an* n*-bit state* s*, where*

- $\mathsf{refresh}$ *takes a state* s *and an input* $x \in \{0,1\}^m$ *and produces a new state* $s' \leftarrow \mathsf{refresh}^P(s,x)$*, and*
- next *takes a state* s *and produces a new state and an output* $y \in \{0,1\}^r$*, i.e,* $(s', y) \leftarrow \mathsf{next}^P(s)$*.*

A PRNG processing m*-bit inputs and producing* r*-bit output is called a* (m, r)-PRNG*.*

4.2 Security Game

Robustness game. PRNGs are expected to satisfy the so-called *robustness* property, which captures the properties discussed at the beginning of Section 4. The corresponding security game is depicted in Figure 1. The game initially chooses a random bit b and initializes the state of the PRNG to 0^n. Subsequently, it offers the following oracles to $\mathcal{A}$:

- **adv-refresh**(x) calls the $\mathsf{refresh}$ procedure to absorb $x \in \{0,1\}^n$ into the internal state of the PRNG;
- **get-next** and **get-next*** allow the attacker to get pseudorandom outputs by calling the next procedure on the current state and returning the output y. The difference between the two oracles is that **get-next** is supposed to be called only when the state has high entropy, whereas **get-next*** can be called *prematurely*, i.e., before the state has absorbed enough randomness for the next function to output pseudorandom values (cf. definition of legitimate attackers below).
- **next-ror** works like the **get-next**-oracle, except that it creates a challenge, i.e., if $b = 1$, it outputs a uniform random value $y_1 \in \{0,1\}^r$ instead of the PRNG output y_0.
- **get-state** and **set-state** model state compromises by letting the attacker learn the current state or set it to an arbitrary value, respectively.

The advantage of $\mathcal{A}$ in the robustness game is denoted by $\mathrm{Adv}^{\mathrm{rob},P}_{\mathsf{PRNG}}(\mathcal{A})$.

Canonical attackers. It will be useful to define to following notion of canonical attackers: Consider the interaction of an attacker $\mathcal{A}$ with the robustness game. The following events are called *entropy drains*:

- the beginning of the game,
- calls to **get-state** or **set-state**, and
- calls to **get-next***.

In other words, entropy drains are the events that cause the PRNG state to lose its entropy, which includes premature calls to next. An attacker $\mathcal{A}$ is said to be *canonical* if it does not make **get-next*** queries nor the following query pattern: an entropy drain followed by one or more **adv-refresh** queries, followed by a **get-state** query.

Considering canonical attackers only is without loss of generality. This is because the above sequence of queries can be simulated by the attacker by making a **get-state** query right away and computing the output of **get-state** or **get-next*** itself. In particular, for every attacker $\mathcal{A}$, there exists a canonical attacker $\mathcal{A}$ with the same advantage. All attackers in the remainder of this work are therefore assumed to be canonical.

Legitimate attackers. In order to obtain a sensible definition devoid of trivial attacks, attackers must satisfy a "legitimacy" condition. The condition roughly requires that an attacker only ask for challenges when it has sufficient amount of uncertainty about the PRNG's internal state.

Towards formalizing the legitimacy condition, consider the interaction of $\mathcal{A}$ with a *variant* of the robustness game defined as follows: Whenever oracles **next-ror** or **get-next** are called, instead of evaluating next, the game simply uses two uniformly random and independent values (s, y) as the output of next.

Observe that this variant of the robustness game, called the *legitimacy game* corresponds to an interaction between $\mathcal{A}$ and an *ideal* PRNG, which produces perfect randomness. Moreover, the legitimacy game is *construction-independent.*

In the legitimacy game, define now the following random variables immediately before $\mathcal{A}$ makes the i^{th} call to oracle **get-next** or **next-ror**:

- $\mathcal{L}_i$: the list of P-queries by $\mathcal{A}$ and the corresponding answers;
- Σ_i: the state of $\mathcal{A}$;
- $\overline{X}_i$: vector of inputs provided by $\mathcal{A}$ since the *the most recent entropy drain (MRED)*; and
- S_i: the state of the PRNG immediately after the MRED.

The legitimacy condition requires that $\mathcal{A}$ provide inputs that have high min-entropy even conditioned on its current state, the queries so far, and the state of the PRNG after the MRED.

Definition 7 (Legitimate attackers). *An attacker $\mathcal{A}$ is said to be* γ^*-legitimate *if for all i,*

$$\mathrm{H}_\infty(\overline{X}_i | \Sigma_i \mathcal{L}_i S_i) \;\geq\; \gamma^* \;,$$

where MREDs are defined as above.

In order to capture IT-legitimate attackers (against IT PRNGs), the set of entropy drains is extended to include

– calls to **get-next** and **next-ror**.

With this definition of MRED and notation analogous to that in the previous definition, IT-legitimate attackers are defined as follows:

Definition 8 (Legitimate IT attackers). *An attacker $\mathcal{A}$ is said to be* γ^*-IT-legitimate *if for all i,*

$$\mathrm{H}_\infty(\overline{X}_i|\Sigma_i\mathcal{L}_i S_i) \;\geq\; \gamma^* \;,$$

w.r.t. the extended definition of MRED.

Robust PRNGs. We are now ready to quantify the efficiency of attacker $\mathcal{A}$, and to define our final notion of PRNG robustness.

Definition 9 (Attacker efficiency). *An attacker is called a* (q, t, ℓ)-attacker *if*

– *q is the maximum number of P-queries it makes,*

– *ℓ is the maximum number of* **adv-refresh** *calls between any entropy drain and successive call to either* **next-ror** *or* **get-next**, *and*

– *t is the maximum total number of calls to any oracle in the robustness game other than* **adv-refresh**.

An attacker is called a (q, t, ℓ)-IT-attacker *if it satisfies the above conditions but makes an arbitrary number of queries to P* after *the interaction with the challenger ends.*

Definition 10 (Robustness of PRNGs). *A PRNG construction* $\mathsf{PRNG} = (\mathsf{refresh}, \mathsf{next})$ *with oracle access an ideal primitive P is* $(\gamma^*, q, t, \ell, \varepsilon)$-(IT-)robust *in the P-model if for every γ^*-(IT-)legitimate (q, t, ℓ)-(IT-)attacker,*

$$\mathrm{Adv}^{\mathrm{rob},P}_{\mathsf{PRNG}}(\mathcal{A}) \;\leq\; \varepsilon \;.$$

Observe that online extractors (cf. Definition 4) are a special case of robust PRNGs. In terms of construction, the PRNG next algorithm can be replaced by $\mathsf{finalize}$, which simply discards the state output by next. If then the PRNG robustness game is relaxed such that the only queries the attacker can make are (a) arbitrarily many queries to **adv-refresh** followed by (b) $t = 1$ query to **next-ror**, one obtains a notion equivalent to Definiton 3.

5 Constructions of PRNGs

This section presents three simple, intuitive, and—most importantly—practical PRNG constructions:

– a construction based on the *Merkle-Damgård paradigm* using a public *fixed-length compression function*;
– a construction based on the *Merkle-Damgård paradigm* using the *Davies-Meyer compression function* (as in SHA-2), which is built from any public block cipher; and
– a construction based on the *Sponge paradigm* (as in SHA-3), which uses a public permutation.

For each paradigm, there are in fact two constructions: one achieving normal, computational PRNG security and one achieving information-theoretic (IT) security. The security analyses of these constructions can be found in the full version of this paper, available online.

5.1 PRNGs from Merkle-Damgård

A PRNG can be obtained from a compression function F as follows (cf. Figure 2):[13]

Construction 2 (PRNG from Merkle-Damgård). *The (m, r)-PRNG construction $\mathsf{MD} = (\mathsf{refresh}, \mathsf{next})$ based on Merkle-Damgård with a compression function $F : \{0,1\}^n \times \{0,1\}^m \to \{0,1\}^n$ is defined as follows:*[14]

– $\mathsf{refresh}^F(s, x) = F(s, x)$, *and*
– $\mathsf{next}^F(s) = (F(s, 0), F(s, 1) \| \cdots \| F(s, r/n))$.

The security of Construction 2 is proved in the F-model, where F is a uniformly random function.

Theorem 4 (Robustness of Merkle-Damgård PRNGs). *Construction 2 is a $(\gamma^*, q, t, \ell, \varepsilon_{\text{rob}})$-robust PRNG in the F-model for*

$$\varepsilon_{\text{rob}} \;\leq\; 2t \cdot \left(\frac{\tilde{q}^2 + \tilde{q}\ell + \ell^2}{2^n} + \frac{\tilde{q}}{2^{\gamma^*}} \right),$$

where $\tilde{q} = q + r/n + 1$.

An IT-robust PRNG based on Merkle-Damgård can be obtained if the next function simply outputs the truncated state (and outputs 0^n as the new state):

Construction 3 (IT-PRNG from Merkle-Damgård). *The (m, r)-PRNG construction $\mathsf{MD}_r = (\mathsf{refresh}, \mathsf{next})$ based on Merkle-Damgård with a compression function $F : \{0,1\}^n \times \{0,1\}^m \to \{0,1\}^n$ is defined as follows:*

[13] To reduce notational clutter, the algorithms refresh and next of the PRNG constructions are not "branded" with the design name. There will be no ambiguity as to which construction is meant in any place in this paper.

[14] The integer arguments to the compression function are to be naturally mapped to $\{0,1\}^n$.

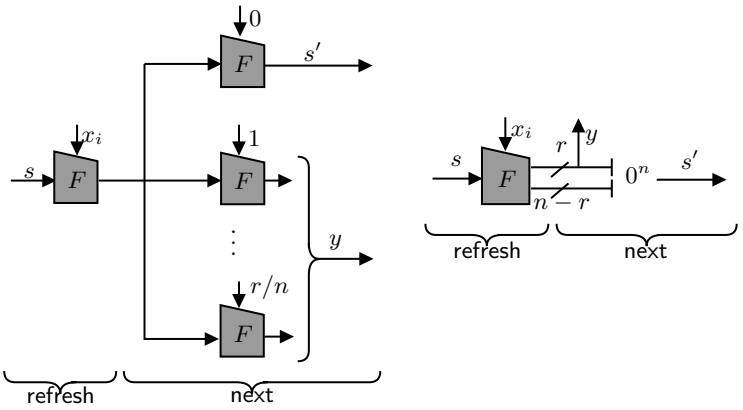

Fig. 2. Procedures refresh (processing a single-block input x_i) and next of Merkle-Damgård PRNG constructions with compression function F. Left: Computationally secure Construction 2; right: IT secure Construction 3.

- $\mathsf{refresh}^F(s, x) = F(s, x)$, *and*
- $\mathsf{next}^F(s) = (0^n, s[1..r])$.

The security of Construction 3 is proved in the F-model, where F is a uniformly random function. To state the theorem for the IT construction, for an integer ℓ, let

$$d'(\ell) = \max_{\ell' \in \{1,\ldots,\ell\}} |\{d \in \mathbb{N} \ : \ d|\ell'\}| \, .$$

Observe that, asymptotically, $d'(\ell)$ grows very slowly, i.e., as $\ell^{o(1)}$. Furthermore, let F be a random compression function.

Theorem 5 (IT-Robustness of Merkle-Damgård PRNGs). *Construction 3 is a $(\gamma^*, q, t, \ell, \varepsilon_{\text{rob}})$-IT-robust PRNG in the F-model, where*

$$\varepsilon_{\text{rob-it}} \ \leq \ \frac{t}{2}\sqrt{\frac{2^{r-\gamma^*}}{(1-\rho)} + \ell \cdot d'(\ell) \cdot \frac{2^r}{2^n} + 64\ell^4 \cdot \frac{2^r}{2^{2n}} + 16\ell^2 \cdot \frac{\tilde{q}^2 2^r}{2^{2n}}} \ + \ t\rho \ ,$$

for $\rho = \frac{\tilde{q}^2}{2^r}$ where $\tilde{q} = q + t\ell$.

Parameter choices. In terms of concrete parameters, observe the following for the Merkle-Damgård constructions above:

- *Computational PRNG:* If one were to use SHA-512 as compression function with $n = 512$, and, moreover, choose $r = n$. We let $t = 1, q = 2^{80}$ and let $\gamma^* = \ell$. This assumes that we get at least one bit of entropy from each block. We would need $\gamma^* \approx 160$ to get 80 bits of security.

– *IT PRNG:* For example, assume SHA-512's compression function is used, i.e., $n = 512$. If we let $r = 256$, then we get (we also approximate $1/(1-\rho) \leq 2$, very generously)

$$\varepsilon_{\text{rob-it}} \;\leq\; \frac{t}{2}\sqrt{2^{257-\gamma^*} + \frac{\ell \cdot d'(\ell)}{2^{256}}} \;+\; t\frac{q^2}{2^{256}} \;,$$

We let $\ell = \gamma^*$. Then, if we set for example $q = 2^{80}$. We would need the entropy loss, i.e, $\gamma^* - r = 162$ for 80 bits of security.

5.2 PRNGs from Merkle-Damgård with Davies-Meyer

The Davies-Meyer compression function maps two inputs $a \in \{0,1\}^m$ and $b \in \{0,1\}^n$ to an n-bit string

$$E(b,a) \oplus a \;,$$

where E is an arbitrary block cipher (where b is the key and a the input).[15] Correspondingly, a PRNG can be obtained from E as follows (cf. Figure 3):

Construction 4 (PRNG from MD-DM). *The (n,r)-PRNG construction $\mathsf{DM} = (\mathsf{refresh}, \mathsf{next})$ based on Merkle-Damgård with Davies-Meyer (MD-DM) uses a cipher $E : \{0,1\}^k \times \{0,1\}^n \to \{0,1\}^n$ and is defined as follows:*[16]

– $\mathsf{refresh}^E(s,x) = E(x,s) \oplus s$, *and*

– $\mathsf{next}^E(s) = (E(0,s) \oplus s, E(1,s) \oplus s \| \cdots \| E(r/n,s) \oplus s)$.

The security of Construction 4 is proved in the E-model, where E is a cipher chosen uniformly at random from the set of all ciphers and can be queried in both the forward and backward direction.

Theorem 6 (Robustness of MD-DM PRNGs). *Construction 4 is a $(\gamma^*, q, t, \ell, \varepsilon_{\text{rob}})$-robust PRNG in the E-model for*

$$\varepsilon_{\text{rob}} \;\leq\; 4t \cdot \left(\frac{\tilde{q}^2 + \tilde{q}\ell + \ell^2}{2^n} + \frac{\tilde{q}}{2^{\gamma^*}}\right),$$

where $\tilde{q} = q + r/n + 1$.

In the IT-secure variant of the MD-DM construction, $\mathsf{refresh}$ remains the same, but next will truncate the input state to r bits, which it outputs, and then zero out the state.

Construction 5 (IT-PRNG from MD-DM). *The (n,r)-PRNG construction $\mathsf{DM}_r = (\mathsf{refresh}, \mathsf{next})$ using Merkle-Damgård with Davies-Meyer (MD-DM) uses a block cipher $E : \{0,1\}^k \times \{0,1\}^n \to \{0,1\}^n$ and is defined as follows:*

[15] A (block) cipher is an efficiently computable and invertible permutation $E(k,\cdot) : \{0,1\}^n \to \{0,1\}^n$ for every key $k \in \{0,1\}^n$.

[16] The integer arguments to the cipher are to be naturally mapped to $\{0,1\}^n$.

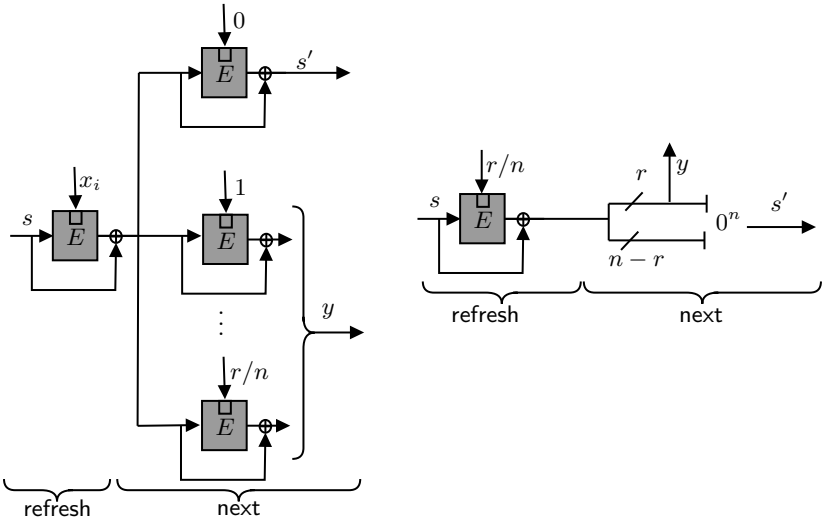

Fig. 3. Procedures refresh (processing a single-block input x_i) and next of Merkle-Damgård PRNG constructions with the Davies-Meyer compression function based on a block cipher E. Left: Computationally secure Construction 4; right: IT secure Construction 5.

– $\mathsf{refresh}^E(s, x) = E(x, s) \oplus s$, *and*
– $\mathsf{next}^E(s) = (0^n, s[1..r])$.

The security of Construction 5 is proved in the E-model, where E is a cipher chosen uniformly at random from the set of all ciphers and can be queried in both the forward and backward direction. Let $d'(\ell)$ be defined as in Section 5.1.

Theorem 7 (IT-Robustness of MD-DM PRNGs). *Construction 5 is a* $(\gamma^*, q, t, \ell, \varepsilon_{\text{rob}})$*-IT-robust PRNG in the E-model, where*

$$\varepsilon_{\text{rob-it}} \leq \frac{t}{2}\sqrt{\frac{2^{r-\gamma^*}}{(1-\rho)} + \ell \cdot d'(\ell)\frac{2^r}{2^{n-1}} + 64\ell^4 \cdot \frac{2^r}{2^{2n-2}} + 16\ell^2\tilde{q}^2 \cdot \frac{2^r}{2^{2n-2}}} + t\rho \ ,$$

for $\rho = \frac{\tilde{q}^2}{2^r}$ *where* $\tilde{q} = q + t\ell$

Parameter choices. In terms of concrete parameters, observe the following for the PRNG constructions from Merkle-Damgård with Davies-Meyer above:

– *Computational PRNG:* SHA-512 is a 512-bit block cipher algorithm that encrypts 512 bit hash value using the input as key. Therefore, we let $n = 512$ and set $r = n$. We let $t = 1, q = 2^{80}$ and let $\ell = \gamma^*$. This assumes that we get at least one bit of entropy from each block. We would need $\gamma^* \approx 163$ to get 80 bits of security.

– *IT PRNG:* We again let $n = 512$. If we let $r = 256$, then we get (we also approximate $1/(1-\rho) \leq 2$, very generously)

$$\varepsilon_{\text{rob-it}} \;\leq\; \frac{t}{2}\sqrt{2^{129-\gamma^*} + \frac{\ell \cdot d'(\ell)}{2^{127}}} \;+\; t\frac{q^2}{2^{128}}\;,$$

We let $\ell = \gamma^*$. Then, if we set for example $q = 2^{80}$. We would need the entropy loss, i.e, $\gamma^* - r = 162$ for 80 bits of security.

5.3 PRNGs from Sponges

Let $n \in \mathbb{N}$ and $n = r + c$. In the following, for an n-bit string s, let $s = s^{(\mathsf{r})} \| s^{(\mathsf{c})}$ be decomposition of s into an r-bit and c-bit string. A PRNG using the Sponge paradigm can be obtained from a permutation π as follows (cf. Figure 4):

Construction 6 (PRNG from Sponges). *The Sponge-based PRNG construction* $\mathsf{Spg} = (\mathsf{refresh}, \mathsf{next})$ *uses a permutation* $\pi : \{0,1\}^n \to \{0,1\}^n$ *to absorb and produce* r*-bit inputs and outputs, respectively, and is defined as follows:*

– $\mathsf{refresh}^\pi(s, x) = \pi(s \oplus x \| 0^c)$*, and*

– $\mathsf{next}^\pi(s) = (\pi(s) \oplus 0^r \| s^{(\mathsf{c})}, s^{(\mathsf{r})})$.

The next function design is due to Hutchinson [28], who simplified a proposal by Gazi and Tessaro [26]. Recall that the Merkle-Damgård constructions have a "parallel" next function in order to produce r/n blocks of random output with $r/n + 1$ calls to the ideal primitive, where the additional call is used to produce a new state. Were it not for this optimization, on order to obtain r bits of output, one would have to apply the next function r/n times in a row, which would results in twice the number of ideal-primitive calls.

The next function for Sponges, on the other hand, only makes a single call to the ideal primitive to produce both a new state and the random output. Therefore, no parallel next function is provided for the Sponge-based PRNG.

The security of Construction 6 is proved in the π-model, where π is a uniformly random permutation, which can be queried in both the forward and backward direction.

Theorem 8 (Robustness of Sponge PRNGs). *Construction 6 is a* $(\gamma^*, q, t, \ell, \varepsilon_{\text{rob}})$*-robust PRNG in the* π*-model for*

$$\varepsilon_{\text{rob}} \;\leq\; 4t \cdot \left(\frac{\tilde{q}^2 + \tilde{q}\ell + \ell^2}{2^n} + \frac{\tilde{q}}{2^{\gamma^*}} + \frac{\tilde{q}^2}{2^c}\right),$$

where $\tilde{q} = q + r/n + 1$.

Observe that the bound in Theorem 8 is only reasonable when c is large enough, which matches the fact that CBC-based PRNGs—which correspond to the case $c = 0$, are not secure.

In the IT variant of the Sponge construction, refresh remains the same, but next will truncate the input state to r bits, which it outputs, and then zero out the state.

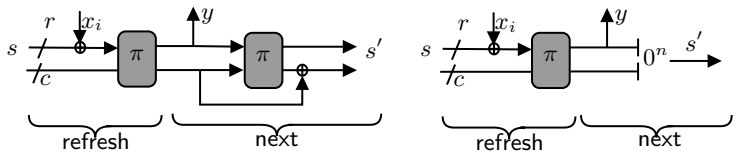

Fig. 4. Procedures refresh (processing a single-block input x_i) and next of Merkle-Damgård PRNG constructions with compression function F. Left: Computationally secure Construction 2; right: IT candidate Construction 3.

Construction 7 (IT-PRNG from Sponges). *The Sponge-based PRNG construction* $\mathsf{Spg}_r = (\mathsf{refresh}, \mathsf{next})$ *uses a permutation* $\pi : \{0,1\}^n \to \{0,1\}^n$ *to absorb and produce* r*-bit inputs and outputs, respectively, and is defined as follows:*

- $\mathsf{refresh}^\pi(s, x) = \pi(s \oplus x \| 0^c)$*, and*
- $\mathsf{next}^\pi(s) = (0^n, s[1..r])$.

Theorem 9 (IT-Robustness of Sponge PRNGs). *Construction 7 is a* $(\gamma^*, q, t, \ell, \varepsilon_{\text{rob}})$*-IT-robust PRNG in the* π*-model for*

$$\varepsilon_{\text{rob-it}} \;\leq\; \frac{t}{2}\sqrt{\frac{2^{r-\gamma^*}}{(1-\rho)} + \frac{\ell \cdot (\ell + \tilde{q})}{2^{c-1}}} + t\rho \;,$$

for $\rho = \frac{\tilde{q}^2}{2^c}$ *where* $\tilde{q} = q + t\ell$

Parameter choices. In terms of concrete parameters, observe the following for the PRNG constructions from Sponges above: above:

- *Computational PRNG:* SHA-3 like parameters have $n = 1600$ and $c = 1024$. We let $t = 1, q = 2^{80}$ and let $\ell = \gamma^*$. This assumes that we get at least one bit of entropy from each block. We would need $\gamma^* \approx 163$ to get 80 bits of security.
- *IT PRNG:* We let $n = 1600$ and $c = 1024$. In addition, we let $t = 1$ and $q = 2^{80}$. We also let $\ell = \gamma^*$. Therefore, we incur an entropy loss of 160 bits to get 80 bits of security.

6 Overview of Our Techniques

Due to paucity of space we defer the proofs of the various constructions to the appendix. Due to paucity of space, the proofs have been deferred to the full version of the paper which is now available on ePrint [17]. The proofs appear in separate sections for the computational PRNG constructions and the IT constructions. In this section we give a brief overview of our techniques.

Computational PRNGs Proving Techniques. The main technique we use in all the proofs is the "H-Coefficient" technique. In addition, it is instructive to view the robustness game through the lens of simpler intermediate security notions. We define two properties - *recovering* and *preserving*. The former requires that the PRNG, after accumulating enough entropy after a drain, has the output of the next function looking random. The latter defines the property that when the start state is random, even after absorbing adversarially controlled inputs, the output of next is still random. A formal proof showing how they generically imply robustness can be found in the full version.

Further, we define the ideas of *extraction security*, *maintaining security* and *next security*. The first of the three requires that the *state* of the PRNG is indistinguishable from random when sufficient entropy has been absorbed. Maintaining security requires that the PRNG *state* is indistinguishable from random even in the face of adversarially chosen inputs, provided the initial state itself was random. Next security requires that the output of next is indistinguishable from random if the input itself was random. It is easy to see how these ideas would imply the larger properties of recovering and preserving.

IT PRNGs Proving Techniques. The crux of our proofs is the idea of reducing the robustness game to online extraction. We then employ a graph counting argument to bound the collision probability. The bound for the collision probability is then used to compute an upper bound for the statistical distance of our distribution from uniform. To this end, we use three propositions to achieve the final bound. Indeed, similar to the intermediate security notion for robustness of computational PRNGs, we define a notion of *recovering security*. This requires that, after an entropy drain, the IT-PRNG can accumulate enough entropy thereby making the output of next indistinguishable from $(0^n, U_r)$. It is easy to see that this constraint is a relaxation of the requirement posed by its computational counterpart.

References

1. Boaz Barak and Shai Halevi. A model and architecture for pseudo-random generation with applications to /dev/random. In Vijayalakshmi Atluri, Catherine Meadows, and Ari Juels, editors, *ACM CCS 05*, pages 203–212, Alexandria, Virginia, USA, November 7–11, 2005. ACM Press.
2. Boaz Barak, Russell Impagliazzo, and Avi Wigderson. Extracting randomness using few independent sources. In *45th FOCS*, pages 384–393, Rome, Italy, October 17–19, 2004. IEEE Computer Society Press.
3. Boaz Barak, Ronen Shaltiel, and Eran Tromer. True random number generators secure in a changing environment. In Colin D. Walter, Çetin Kaya Koç, and Christof Paar, editors, *CHES 2003*, volume 2779 of *LNCS*, pages 166–180, Cologne, Germany, September 8–10, 2003. Springer, Heidelberg, Germany.
4. Elaine Barker and John Kelsey. NIST Special Publication 800-90A (A revision of SP 800-90) Recommendation for random number generation using deterministic random bit generators. https://csrc.nist.gov/publications/detail/sp/800-90a/rev-1/final, 2012.

5. Elaine Barker and John Kelsey. Recommendation for random number generation using deterministic random bit generators. NIST Special Publication 800-90A, 2012.
6. Mihir Bellare, Viet Tung Hoang, and Sriram Keelveedhi. Instantiating random oracles via UCEs. In Ran Canetti and Juan A. Garay, editors, *CRYPTO 2013, Part II*, volume 8043 of *LNCS*, pages 398–415, Santa Barbara, CA, USA, August 18–22, 2013. Springer, Heidelberg, Germany.
7. Mihir Bellare, Krzysztof Pietrzak, and Phillip Rogaway. Improved security analyses for CBC MACs. In Victor Shoup, editor, *CRYPTO 2005*, volume 3621 of *LNCS*, pages 527–545, Santa Barbara, CA, USA, August 14–18, 2005. Springer, Heidelberg, Germany.
8. Guido Bertoni, Joan Daemen, Michael Peeters, and Gilles Van Assche. On the indifferentiability of the sponge construction. In Nigel P. Smart, editor, *EUROCRYPT 2008*, volume 4965 of *LNCS*, pages 181–197, Istanbul, Turkey, April 13–17, 2008. Springer, Heidelberg, Germany.
9. Guido Bertoni, Joan Daemen, Michael Peeters, and Gilles Van Assche. Sponge-based pseudo-random number generators. In Stefan Mangard and François-Xavier Standaert, editors, *CHES 2010*, volume 6225 of *LNCS*, pages 33–47, Santa Barbara, CA, USA, August 17–20, 2010. Springer, Heidelberg, Germany.
10. Manuel Blum. Independent unbiased coin flips from a correlated biased source-a finite stae markov chain. *Combinatorica*, 6(2):97–108, 1986.
11. Eshan Chattopadhyay and David Zuckerman. Explicit two-source extractors and resilient functions. In Daniel Wichs and Yishay Mansour, editors, *48th ACM STOC*, pages 670–683, Cambridge, MA, USA, June 18–21, 2016. ACM Press.
12. Stephen Checkoway, Ruben Niederhagen, Adam Everspaugh, Matthew Green, Tanja Lange, Thomas Ristenpart, Daniel J. Bernstein, Jake Maskiewicz, Hovav Shacham, and Matthew Fredrikson. On the practical exploitability of dual EC in TLS implementations. In *Proceedings of the 23rd USENIX Security Symposium, San Diego, CA, USA, August 20-22, 2014.*, pages 319–335, 2014.
13. Shan Chen and John P. Steinberger. Tight security bounds for key-alternating ciphers. In Phong Q. Nguyen and Elisabeth Oswald, editors, *EUROCRYPT 2014*, volume 8441 of *LNCS*, pages 327–350, Copenhagen, Denmark, May 11–15, 2014. Springer, Heidelberg, Germany.
14. Benny Chor and Oded Goldreich. Unbiased bits from sources of weak randomness and probabilistic communication complexity (extended abstract). In *26th FOCS*, pages 429–442, Portland, Oregon, October 21–23, 1985. IEEE Computer Society Press.
15. Benny Chor and Oded Goldreich. Unbiased bits from sources of weak randomness and probabilistic communication complexity. *SIAM J. Comput.*, 17(2):230–261, 1988.
16. Benny Chor, Oded Goldreich, Johan Håstad, Joel Friedman, Steven Rudich, and Roman Smolensky. The bit extraction problem of t-resilient functions (preliminary version). In *26th FOCS*, pages 396–407, Portland, Oregon, October 21–23, 1985. IEEE Computer Society Press.
17. Sandro Coretti, Yevgeniy Dodis, Harish Karthikeyan, and Stefano Tessaro. Seedless fruit is the sweetest: Random number generation, revisited. Cryptology ePrint Archive, Report 2019/198, 2019. https://eprint.iacr.org/2019/198.
18. Yevgeniy Dodis, Rosario Gennaro, Johan Håstad, Hugo Krawczyk, and Tal Rabin. Randomness extraction and key derivation using the CBC, cascade and HMAC modes. In Matthew Franklin, editor, *CRYPTO 2004*, volume 3152 of *LNCS*, pages

494–510, Santa Barbara, CA, USA, August 15–19, 2004. Springer, Heidelberg, Germany.
19. Yevgeniy Dodis, David Pointcheval, Sylvain Ruhault, Damien Vergnaud, and Daniel Wichs. Security analysis of pseudo-random number generators with input: /dev/random is not robust. In Ahmad-Reza Sadeghi, Virgil D. Gligor, and Moti Yung, editors, *ACM CCS 13*, pages 647–658, Berlin, Germany, November 4–8, 2013. ACM Press.
20. Yevgeniy Dodis, Thomas Ristenpart, and Salil P. Vadhan. Randomness condensers for efficiently samplable, seed-dependent sources. In Ronald Cramer, editor, *TCC 2012*, volume 7194 of *LNCS*, pages 618–635, Taormina, Sicily, Italy, March 19–21, 2012. Springer, Heidelberg, Germany.
21. Yevgeniy Dodis, Adi Shamir, Noah Stephens-Davidowitz, and Daniel Wichs. How to eat your entropy and have it too - optimal recovery strategies for compromised RNGs. In Juan A. Garay and Rosario Gennaro, editors, *CRYPTO 2014, Part II*, volume 8617 of *LNCS*, pages 37–54, Santa Barbara, CA, USA, August 17–21, 2014. Springer, Heidelberg, Germany.
22. D. Eastlake, J. Schiller, and S. Crocker. *RFC 4086 - Randomness Requirements for Security*, June 2005.
23. Niels Ferguson. Private communication, 2013.
24. Niels Ferguson and Bruce Schneier. *Practical Cryptography*. John Wiley & Sons, Inc., New York, NY, USA, 1 edition, 2003.
25. Peter Gaži, Krzysztof Pietrzak, and Michal Rybár. The exact PRF-security of NMAC and HMAC. In Juan A. Garay and Rosario Gennaro, editors, *CRYPTO 2014, Part I*, volume 8616 of *LNCS*, pages 113–130, Santa Barbara, CA, USA, August 17–21, 2014. Springer, Heidelberg, Germany.
26. Peter Gazi and Stefano Tessaro. Provably robust sponge-based PRNGs and KDFs. In Marc Fischlin and Jean-Sébastien Coron, editors, *EUROCRYPT 2016, Part I*, volume 9665 of *LNCS*, pages 87–116, Vienna, Austria, May 8–12, 2016. Springer, Heidelberg, Germany.
27. Danny Harnik and Moni Naor. On everlasting security in the hybrid bounded storage model. In Michele Bugliesi, Bart Preneel, Vladimiro Sassone, and Ingo Wegener, editors, *ICALP 2006, Part II*, volume 4052 of *LNCS*, pages 192–203, Venice, Italy, July 10–14, 2006. Springer, Heidelberg, Germany.
28. Daniel Hutchinson. A robust and sponge-like PRNG with improved efficiency. In Roberto Avanzi and Howard M. Heys, editors, *SAC 2016*, volume 10532 of *LNCS*, pages 381–398, St. John's, NL, Canada, August 10–12, 2016. Springer, Heidelberg, Germany.
29. Daniel Hutchinson. A robust and sponge-like PRNG with improved efficiency. Cryptology ePrint Archive, Report 2016/886, 2016. http://eprint.iacr.org/2016/886.
30. Russell Impagliazzo, Leonid A. Levin, and Michael Luby. Pseudo-random generation from one-way functions (extended abstracts). In *21st ACM STOC*, pages 12–24, Seattle, WA, USA, May 15–17, 1989. ACM Press.
31. Russell Impagliazzo and Michael Luby. One-way functions are essential for complexity based cryptography (extended abstract). In *30th FOCS*, pages 230–235, Research Triangle Park, North Carolina, October 30 – November 1, 1989. IEEE Computer Society Press.
32. Information technology - Security techniques - Random bit generation. ISO/IEC18031:2011, 2011.
33. Jesse Kamp, Anup Rao, Salil P. Vadhan, and David Zuckerman. Deterministic extractors for small-space sources. *J. Comput. Syst. Sci.*, 77(1):191–220, 2011.

34. John Kelsey, Bruce Schneier, and Niels Ferguson. Yarrow-160: Notes on the design and analysis of the yarrow cryptographic pseudorandom number generator. In *In Sixth Annual Workshop on Selected Areas in Cryptography*, pages 13–33. Springer, 1999.
35. Killmann, W. and Schindler, W. A proposal for: Functionality classes for random number generators. AIS 20 / AIS31, 2011.
36. Hugo Krawczyk. Cryptographic extraction and key derivation: The HKDF scheme. In Tal Rabin, editor, *CRYPTO 2010*, volume 6223 of *LNCS*, pages 631–648, Santa Barbara, CA, USA, August 15–19, 2010. Springer, Heidelberg, Germany.
37. David Lichtenstein, Nathan Linial, and Michael E. Saks. Some extremal problems arising form discrete control processes. *Combinatorica*, 9(3):269–287, 1989.
38. John M. 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, 2014.
39. Noam Nisan and David Zuckerman. More deterministic simulation in logspace. In *Proceedings of the Twenty-Fifth Annual ACM Symposium on Theory of Computing, May 16-18, 1993, San Diego, CA, USA*, pages 235–244, 1993.
40. Noam Nisan and David Zuckerman. Randomness is linear in space. *J. Comput. Syst. Sci.*, 52(1):43–52, 1996.
41. Jacques Patarin. The "coefficients H" technique (invited talk). In Roberto Maria Avanzi, Liam Keliher, and Francesco Sica, editors, *SAC 2008*, volume 5381 of *LNCS*, pages 328–345, Sackville, New Brunswick, Canada, August 14–15, 2009. Springer, Heidelberg, Germany.
42. Berry Schoenmakers and Andrey Sidorenko. Cryptanalysis of the dual elliptic curve pseudorandom generator. Cryptology ePrint Archive, Report 2006/190, 2006. http://eprint.iacr.org/2006/190.
43. Thomas Shrimpton and R. Seth Terashima. A provable-security analysis of Intel's secure key RNG. In Elisabeth Oswald and Marc Fischlin, editors, *EUROCRYPT 2015, Part I*, volume 9056 of *LNCS*, pages 77–100, Sofia, Bulgaria, April 26–30, 2015. Springer, Heidelberg, Germany.
44. Dan Shumow and Niels Ferguson. On the possibility of a back door in the nist sp800-90 dual ec prng. *CRYPTO Rump Session*, 2007.
45. Pratik Soni and Stefano Tessaro. Public-seed pseudorandom permutations. In Jean-Sébastien Coron and Jesper Buus Nielsen, editors, *EUROCRYPT 2017, Part II*, volume 10211 of *LNCS*, pages 412–441, Paris, France, April 30 – May 4, 2017. Springer, Heidelberg, Germany.
46. Luca Trevisan and Salil P. Vadhan. Extracting randomness from samplable distributions. In *41st FOCS*, pages 32–42, Redondo Beach, CA, USA, November 12–14, 2000. IEEE Computer Society Press.
47. John von Neumann. Various techniques used in connection with random digits. In A.S. Householder, G.E. Forsythe, and H.H. Germond, editors, *Monte Carlo Method*, pages 36–38. National Bureau of Standards Applied Mathematics Series, 12, Washington, D.C.: U.S. Government Printing Office, 1951.
48. Wikipedia. /dev/random. http://en.wikipedia.org/wiki//dev/random, 2004. [Online; accessed 09-February-2014].
49. Joanne Woodage and Dan Shumow. An analysis of the NIST SP 800-90A standard. In *Advances in Cryptology - EUROCRYPT 2019 - 38th Annual International Conference on the Theory and Applications of Cryptographic Techniques, Darmstadt, Germany, May 19 - May 23*, 2019.