cryptography

MDPI

Article

A No-Degradation Framework for Augmenting Operating-System Randomness with Publicly Observable Dynamic Data Sources

Mohammad Dashti

College of Computer and Systems Engineering, Abdullah Al Salem University, Khaldiya 72303, Kuwait; mohammad.dashti@aasu.edu.kw

Abstract

Modern cryptography depends on the quality of the operating-system entropy pool, and historic failures of that pool have shown that randomness, not algorithm strength, is often the weakest link of a deployed cryptosystem. Hardware True-Random Number Generators (TRNGs) mitigate this risk but are not universally available or trusted. This paper develops a unified four-stage construction, *extract* → *condition* → *mix* → *derive*, that admits heterogeneous publicly observable dynamic data sources as auxiliary inputs to the operating-system randomness pipeline. The construction's central claim is a *no-degradation defence-in-depth* property, formalised as Theorem 1: under a pseudorandom-function assumption on the conditioning function, the composed output is computationally indistinguishable from the operating-system output alone, even when an adversary fully controls the public source. The framework does not claim that public sources supply secret entropy against an adversary who also observes them; it claims only that mixing them in cannot weaken the baseline, and that any uncompromised source supplies residual unpredictability under operating-system-generator compromise. We instantiate the framework in three pipelines—aircraft state vectors from a public sensor network, webcam frame-differencing of a wall of lava lamps, and computer-vision tracking of vehicles on a public traffic-camera feed—and argue soundness by combining the theorem with published evaluations of each source. On a 381 kbit corpus of framework-derived keys, every applicable NIST SP 800-22 test and SP 800-90B entropy estimator finds the framework output statistically indistinguishable from an `os.urandom` baseline; this is consistent with no-degradation but not in itself evidence of marginal entropy gain, and per-pipeline raw-input measurements and source-ablation runs are deferred to empirical follow-up. Off-the-shelf computer-vision tooling is sufficient to operationalise the framework on commodity hardware.

Keywords: cryptographic randomness; environmental entropy; true random number generation; computer vision; adversarial robustness; defence in depth; conceptual framework; information security

Check for updates

Academic Editors: Josef Pieprzyk and Kris Gaj

Received: 18 May 2026
Revised: 29 June 2026
Accepted: 16 July 2026
Published: 18 July 2026

Copyright: © 2026 by the author. Licensee MDPI, Basel, Switzerland. This article is an open access article distributed under the terms and conditions of the Creative Commons Attribution (CC BY) license.

1. Introduction

When Heninger et al. [1] scanned the public Internet for TLS and SSH keys, they discovered that hundreds of thousands of certificates and host keys shared private factors—not because their algorithms were broken, but because the devices generating them had been seeded with insufficient randomness. Comparable failures had been a recurring theme in security engineering for decades: Netscape's early-1990s SSL seeding flaw was

exploitable not because RC4 was weak but because the keys were predictable [2], and even widely audited operating-system generators have been shown to be theoretically brittle under realistic compromise models [3]. Modern cryptography rests on the assumption that secret keys are drawn from a sufficiently random distribution; almost every deployed system delegates this responsibility to an operating-system entropy pool expanded by a deterministic random bit generator (DRBG) [4]. When the pool is well seeded, the resulting bit-streams behave indistinguishably from uniform; when it is not, the algorithm above it is irrelevant.

Practitioners have long looked beyond the operating system for additional physically motivated sources of unpredictability. Davis et al. [5] extracted entropy from air-turbulence-driven jitter in disk drives; Jun and Kocher [6] characterised the thermal noise that underlies Intel's hardware random number generator; Müller [7] demonstrated that even unrelated CPU pipeline jitter can supply usable entropy at scale. Less conventional sources have been explored as well: human gameplay [8], photon-counting noise in a smartphone camera [9], pixel noise in arbitrary digital images [10], and on-board sensor data on Internet-of-Things devices [11]. Most visibly in the security-engineering imagination, Cloudflare's LavaRand pipeline observes a wall of lava lamps and folds the resulting visual entropy into the company's production randomness pool [12]. What unites these systems is not a shared engineering pattern but a shared intuition: a physical or environmental process is observed, its observations are conditioned with a cryptographic hash function, and the result is mixed into a downstream key-derivation pipeline. What divides them is everything else—every system is a bespoke one-off, tightly coupled to its source and its intended deployment.

Two recent developments make this fragmentation increasingly costly. First, an enormous quantity of dynamic real-world data is now publicly observable, with no specialised hardware required: aircraft state vectors broadcast via Automatic Dependent Surveillance–Broadcast (ADS-B) and are aggregated by community sensor networks [13]; traffic-camera feeds; weather observations; financial tickers. Second, off-the-shelf computer-vision tooling—open-source object detectors such as the You Only Look Once (YOLO) family [14] and the Open Source Computer Vision Library (OpenCV) [15]—make these data streams programmatically observable in ways that were previously the preserve of bespoke instrumentation. Taken together, these developments invite a question that prior work has not addressed at the level of an engineering framework: can heterogeneous, publicly observable, dynamic data sources—processed with commodity perception tooling—be composed into a single deployable construction for cryptographic randomness?

The literature offers partial answers but not a unifying one. The constructions surveyed above each instantiate the same broad pattern—extract observations, condition them, mix with a baseline generator, derive keys—yet none articulates that pattern as a general framework or evaluates how it generalises across modalities. This paper addresses the gap.

The contribution of this paper is fivefold. *First*, we propose a unified four-step framework—*extract* → *condition* → *mix* → *derive*—for combining publicly observable dynamic data sources with the operating-system random number generator. The framework is positioned as a no-degradation defence-in-depth combiner, not as a stand-alone true random number generator: it does not claim that public sources supply secret entropy against an adversary who also observes them, only that mixing them in cannot weaken the operating-system baseline. *Second*, we instantiate the framework across three modalities: aircraft telemetry, frame-difference video of a physical chaotic process, and computer-vision-based object tracking of road traffic. *Third*, we argue for the construction's soundness by stating and proving a formal no-degradation theorem (Theorem 1) for the mix stage under a pseudorandom-function assumption on the conditioning function, and by inheriting and synthesising the published evaluations of each constituent source, situating the

framework with respect to classical results on entropy combiners [16,17] and current NIST guidance on entropy sources and combinations [18,19]. *Fourth*, we develop a threat model with four adversaries (A1–A4) that explicitly distinguishes "public source observed by adversary" (no marginal advantage to the defender) from "operating-system generator compromised" (where any uncompromised public source provides residual unpredictability), and that accounts for adversarial perturbation of the perception layer—a concern made urgent by recent work on physical adversarial attacks against object detectors [20,21]. *Fifth*, we discuss the framework's alignment with NIST SP 800-90B [18] (source modelling, vetted conditioning) and SP 800-90C [19] (approved source–DRBG combinations), and identify per-pipeline empirical evaluation and source-ablation as the two principal directions for empirical follow-up that the present paper does not foreclose.

The remainder of this paper is organised as follows. Section 2 surveys related work on cryptographic randomness, environmental entropy sources, telemetry-derived randomness, computer-vision-based perception, and entropy combiners. Section 3 introduces the notation and soundness baseline used in the rest of the paper. Section 4 presents the framework. Section 5 develops the three instantiations. Section 6 sets out the soundness argument and inherited evidence. Section 7 discusses the threat model, practical implications, and limitations. Section 8 concludes and outlines an empirical follow-up.

2. Background and Related Work

The literature relevant to this paper sits at the intersection of four strands: the standards and definitions of cryptographic randomness; the engineering tradition of harvesting unpredictability from physical or environmental phenomena; the public availability of dynamic real-world data streams; and the theoretical study of how independent entropy sources may be safely combined. We survey each in turn, then identify the gap our framework addresses.

2.1. Cryptographic Randomness: Definitions and Standards

The cryptographic literature distinguishes between *pseudorandom* number generators (PRNGs), which expand a short seed into a long bit-stream by deterministic computation, and *true random* number generators (TRNGs), which derive their output from a physical or environmental source whose underlying process is, by assumption, unpredictable to any adversary [22,23]. When a PRNG is constructed so that its output is computationally indistinguishable from uniform, even given the algorithm itself, it is termed a *cryptographically secure* PRNG (CSPRNG); modern operating systems supply CSPRNG output via interfaces such as `/dev/urandom` and the `getrandom` system call.

The U.S. National Institute of Standards and Technology has organised guidance on cryptographic randomness around three documents in the SP 800-90 series. SP 800-90A [4] specifies deterministic random bit generators; SP 800-90B [18] defines the modelling vocabulary for entropy sources, including the *min-entropy* estimator that we adopt below; and SP 800-90C, finalised in 2025 after a decade of public drafts, sets out approved combinations of entropy sources and DRBGs [19]. The companion SP 800-22 [24] provides a battery of fifteen statistical tests—frequency, runs, spectral, complexity, and others—that is widely used to evaluate candidate generators. Hash functions used as conditioning components are governed by FIPS 180-4 [25], and key-derivation discipline by SP 800-108 Rev. 1 [26]. RFC 4086 [23] remains the most accessible engineering treatment of the practical considerations.

2.2. Environmental Entropy Sources

The use of physical or environmental phenomena as a source of cryptographic randomness has a substantial history. Davis et al. [5] extracted entropy from air turbulence-driven jitter in disk drives—perhaps the earliest demonstration that an everyday observable, examined finely, contains exploitable randomness. Jun and Kocher [6] characterised the thermal-noise source underlying Intel's hardware random number generator, establishing a template that subsequent commercial TRNGs would broadly follow. Müller [7] showed that even non-physical, software-observable phenomena suffice: variations in CPU pipeline timing yield enough entropy to seed a kernel-space generator without dedicated hardware.

Beyond the conventional repertoire of hardware- and software-timing sources, researchers have explored more exotic candidates. Halprin and Naor [8] argued that competitive game-play by humans produces sufficient entropy for cryptographic purposes once an extractor is applied. Sanguinetti et al. [9] used the photon-counting noise of a smartphone camera to build a quantum random number generator, reporting per-pixel min-entropy of 5.7–8.3 bits. Li [10] generalised the camera-as-source idea by extracting entropy from arbitrary digital photographs taken under varying lighting conditions. Hennebert et al. [11] systematised the use of on-board sensors—temperature, humidity, light, and motion—on Internet-of-Things devices as entropy sources, reporting NIST-style min-entropy estimates and noting harvesting times in the millisecond range. The conditioning step in many of these systems rests, ultimately, on a result anticipated by von Neumann [27]: even a heavily biased binary source can be turned into a uniform one by a simple post-processing rule.

Public security-engineering practice has been shaped by one particularly visible system. Cloudflare's LavaRand pipeline observes a wall of lava lamps with a fixed camera, hashes the resulting frames, and folds the result into the company's production randomness pool [12]. LavaRand exemplifies a broader pattern that recurs across the systems above: an environmental process is observed, the observations are conditioned with a cryptographic hash, and the conditioned output is mixed into a downstream key-derivation pipeline. What it does not exemplify is *generality*: each of the systems we have surveyed is bespoke to its source. None articulates a framework into which an arbitrary public observable could be plugged.

2.3. Telemetry-Derived Entropy

Aircraft state vectors broadcast over Automatic Dependent Surveillance–Broadcast (ADS-B) provide a continuous stream of position, velocity, and identifier data for tens of thousands of aircraft worldwide. The OpenSky sensor network aggregates community-contributed receivers into a research-grade dataset that captures a substantial fraction of European commercial traffic [13]. The data are dynamic—influenced by weather, air traffic control directives, and pilot inputs—and exhibit the kind of multi-source variability that motivates their consideration as entropy candidates. Strohmeier et al. [28] document the data-quality landscape, including the gaps and inconsistencies that any practical consumer must reckon with. Costin and Francillon [29] demonstrate that ADS-B is unauthenticated and spoofable, which is consequential for any use case that treats its content as a security input—a concern we return to in the threat model of Section 7. Compared with the camera-and-sensor literature surveyed above, the use of telemetry streams as cryptographic entropy sources has received comparatively little dedicated study, despite their public availability and dynamic character.

2.4. Computer Vision and Adversarial Perception

Two software developments have changed what counts as a programmatically observable phenomenon. The OpenCV library [15] made image-processing primitives—including

frame differencing, histogram analysis, and structural-similarity metrics [30]—available to non-specialist developers. The release of real-time object detectors such as YOLO [14] further widened the aperture: arbitrary scenes, captured by arbitrary cameras, can now be parsed into structured tracks of named objects with little engineering effort. For the purposes of an entropy-extraction pipeline, this means that the *extract* step is no longer a bespoke piece of instrumentation; off-the-shelf perception models can play that role.

The same models, however, are vulnerable to adversarial perturbation. Szegedy et al. [31] and Goodfellow et al. [32] established that small, carefully crafted modifications to image inputs can systematically mislead deep neural classifiers; Eykholt et al. [20] extended this to the physical world, showing that printed perturbations on stop signs caused a YOLOv2 detector to fail in over 85% of frames in controlled lab conditions and 63–72% in outdoor video. Thys et al. [21] demonstrated that wearable adversarial patches can defeat surveillance-camera person detection, an attack model that maps directly onto any framework whose extraction step relies on detection of objects in a public scene. Any framework that delegates extraction to a perception model must therefore include adversarial robustness in its threat model—a point we develop in Section 7.

2.5. Combiners and PRNG Soundness

The theoretical question of *how* environmental entropy can be combined with an existing PRNG without weakening the result is well studied. Maurer [17] developed a universal statistical test for random-bit generators that supplies the theoretical foundation for many subsequent test batteries. Barak and Halevi [16] introduced an architectural model for PRNGs that consume entropy from external inputs and recover from state compromise—a model that closely resembles the construction we develop in Section 4. Dodis et al. [3] refined the picture by demonstrating, formally, that even Linux's `/dev/random` falls short of a robustness definition that the practitioner literature had implicitly assumed; their analysis sharpens the case for *defence in depth* through additional entropy sources rather than replacement of the kernel generator. Kelsey et al. [33] provide a useful taxonomy of cryptanalytic attacks on PRNGs that is helpful in framing what a sound combiner must defend against. For the *derive* step, Krawczyk [34] provides the canonical extract-then-expand discipline embodied in HKDF, which is referenced in NIST SP 800-108 Rev. 1 [26].

2.6. Synthesis and Gap

Two patterns emerge from the literature surveyed above. First, the *engineering* literature on environmental entropy sources is rich but narrow: each contribution fixes a single source and presents a complete, source-specific pipeline. There is no shared abstraction that would allow a practitioner to replace LavaRand's lava lamps with, say, a public traffic camera or an ADS-B feed without re-engineering the pipeline from scratch. Second, the *theoretical* literature on combiners and robust PRNG constructions is general—but it is pitched at a level of abstraction that, on its own, gives little engineering guidance to a security practitioner who wishes to compose an entropy source from publicly observable data.

The gap, then, is not for new theory and not for another bespoke pipeline, but for an engineering-level framework that bridges the two: a construction explicit enough to deploy, general enough to admit heterogeneous sources, and grounded enough in the existing combiner literature to support a soundness argument by inheritance. The remainder of this paper develops such a framework and demonstrates it across three modalities. Section 3 fixes the notation and the soundness baseline against which the construction will be evaluated.

3. Theoretical Preliminaries

We adopt standard cryptographic notation and the entropy modelling vocabulary of NIST SP 800-90B [18]. The framework developed in Section 4 is described in this notation, and the soundness argument of Section 6 is stated against the baseline fixed in Section 3.3.

3.1. Notation

Let X denote a discrete random variable taking values in a finite set $\mathcal{X}$. The *min-entropy* of X is

$$H_\infty(X) \;=\; -\log_2 \max_{x \in \mathcal{X}} \Pr[X = x].$$

Min-entropy quantifies the unpredictability of X to an adversary who guesses its most likely value, and is the standard yardstick by which the output of an entropy source is judged [18] §3. Where two random variables X and Y are jointly distributed, *conditional min-entropy* $\widetilde{H}_\infty(X \mid Y)$ extends the definition to capture how much of X remains unpredictable once Y has been observed.

We write $\Delta(X, Y)$ for the statistical distance between distributions of X and Y over a common alphabet; X is said to be ε-close to uniform when $\Delta(X, U) \le \varepsilon$, where U denotes the uniform distribution over the same alphabet.

Three mappings recur in the framework. A *conditioning function* $\mathsf{Cond} : \{0,1\}^* \to \{0,1\}^n$ compresses a variable-length input—in our instantiations, a cryptographic hash function from the SHA-2 family [25]. A *combiner* $\mathsf{Mix} : \{0,1\}^n \times \{0,1\}^n \to \{0,1\}^n$ takes two fixed-length inputs and returns a single output of the same length, in a way intended to preserve the entropy of either input. A *key-derivation function* $\mathsf{KDF} : \{0,1\}^n \to \{0,1\}^k$ produces a final key of the required length, and is governed in our setting by NIST SP 800-108 Rev. 1 [26].

3.2. Environmental Sources

We define an *environmental source* as a tuple $S = (\Omega, \mathcal{O}, \pi)$, where Ω is the underlying physical or socio-technical process, $\mathcal{O} = \{O_t\}_{t \in T}$ is a sequence of observations indexed by time $t \in T$, and π is the observation channel that maps states of Ω to observations in O_t. Three properties distinguish the sources of interest in this paper.

First, *the observer does not control* Ω: the lava lamps, the air-traffic system, the urban traffic network all proceed according to their own dynamics regardless of what the observer does. Second, *the observation channel* π *is publicly available*: anyone with appropriate sensor hardware, a network connection, or both can produce $\mathcal{O}$ on demand. Third, *the observation stream is dynamic*: O_t is non-trivially time-varying, so that successive observations contribute fresh entropy rather than restating previous ones. This formulation generalises the existing repertoire of sources surveyed above: lava-lamp video, smartphone camera pixels, ADS-B telemetry, and IoT on-board sensors all instantiate it.

3.3. Soundness Baseline

Let R_{OS} denote the output of the operating-system random number generator—the standard CSPRNG output available via interfaces such as `/dev/urandom` or `getrandom`—and let $C(S, R_{\mathsf{OS}})$ denote a construction that combines an environmental source S with R_{OS}.

We require any acceptable construction to satisfy a single baseline property: under standard cryptographic assumptions on the building blocks, $C(S, R_{\mathsf{OS}})$ must be at least as strong as R_{OS} alone. Concretely, an adversary who fully compromises or controls S—by spoofing the sensor input, predicting the underlying process, or replacing the observation stream with chosen values—must obtain no *computational* advantage over an adversary attacking R_{OS} alone. This *no-degradation* requirement is conservative: it does not promise that the construction is *stronger* than R_{OS}, only that it is *no weaker*. Strength gains over

the baseline are then attributed to the additional entropy contributed by S when S is not compromised, and depend on source-specific evidence which we discuss in Section 6. The construction in Section 4 is designed to satisfy the baseline property by an architectural choice—the use of a vetted cryptographic hash function as a salted entropy combiner—that aligns with the extract-then-expand discipline analysed by Krawczyk [34] and with the broader line of work on robust pseudo-random generators with input established by Barak and Halevi [16] and refined by Dodis et al. [3].

3.4. Adversary Model and Where the Source Contributes

It is important to be precise about *when* a publicly observable source can be said to contribute to the defender's security. The clarification we adopt—and refer to throughout the remainder of the paper—factors the question into two orthogonal adversary capabilities, and is summarised in Table 1. The full four-adversary threat model is developed in Section 7.1; here, we fix only the two capabilities relevant to the soundness claim.

- **Public-source observation.** An adversary may observe the *same* public source S as the defender. ADS-B is broadcast in the clear; a wall of lava lamps is photographable from any vantage point that shares line-of-sight with the legitimate camera; a public traffic-camera feed is publicly viewable. There is no secret-channel assumption to be made about any of these.
- **OS-RNG compromise.** An adversary may, separately, have compromised the operating-system random number generator R_{OS}, by exploiting one of the brittleness properties identified by Dodis et al. [3] or by direct kernel-state observation.

The framework's value proposition is an *independence-of-failures* hedge across these two capabilities. The four cells of the resulting matrix have the following character:

Table 1. Where the source contributes to the defender's security. Rows: adversary observes the public source S (yes/no). Columns: operating-system random number generator R_{OS} is compromised (yes/no). The framework is designed to be *at least as strong as* R_{OS} in every cell, and to provide a positive marginal advantage in the upper-right and lower-right cells.

	R_{OS} **Intact**	R_{OS} **Compromised**
Adversary observes S	Security from R_{OS} (uniform). Source contributes no marginal secret entropy—it is observed by both sides.	Security from $\widetilde{H}_\infty(\sigma(O_t) \mid \mathrm{view}_{\mathrm{adv}})$, the conditional min-entropy of the defender's observation given the adversary's. Strictly positive only when the two observation channels are not perfectly correlated (different sensors, sampling jitter, viewing geometry, network-path latency).
Adversary does not observe S	Security from R_{OS}, with the source providing a positive marginal entropy contribution to the defender that the adversary does not see.	Security from the source's min-entropy alone, mediated by the no-degradation guarantee of Theorem 1.

Source: authors' own work.

Two consequences of this matrix are worth stating now. *First*, the upper-left cell—intact R_{OS}, source observed by adversary—shows that the framework's value in the steady state is *not* additional secret entropy; it is the structural property of not weakening R_{OS}. The marketing of the construction is therefore a defence-in-depth posture, not a stand-alone TRNG. *Second*, the upper-right cell—R_{OS} compromised and source observed by adversary—is the cell most explicitly flagged in the threat-model literature. In this cell, the defender's residual security is the conditional min-entropy $\widetilde{H}_\infty(\sigma(O_t) \mid \mathrm{view}_{\mathrm{adv}})$ [18], where $\mathrm{view}_{\mathrm{adv}}$ is the adversary's reconstruction of the public source. This quantity is positive only when the

adversary's reconstruction is imperfect: different sensor, different sampling phase, different network path. Our paper does not attempt to estimate $\widetilde{H}_\infty$ for any specific deployment; we instead treat it as a source-specific operational quantity to be assessed for each instantiation in the field (see Sections 6 and 7.3). What the paper does establish, structurally, is the matrix property: the framework is at least as strong as R_{OS} in every cell.

4. The Framework

4.1. Overview

The framework is a four-stage construction—*extract* → *condition* → *mix* → *derive*—that takes an environmental source S and the operating-system generator R_{OS} as inputs and produces a cryptographic key K as output. Its central design choice is that each stage carries a single cryptographic responsibility, so that source-specific complexity is concentrated in the extract stage while the remaining three stages stay invariant across instantiations. Figure 1 illustrates the pipeline; the four stages are summarised here:

1. **Extract.** Observations O_t are drawn from the source S via the observation channel π, and a serialisation $\sigma(O_t)$ is produced as a fixed- or variable-length byte string.
2. **Condition.** The serialisation is compressed by a vetted conditioning function Cond—in our instantiations, a member of the SHA-2 family—to produce an n-bit conditioned digest $h_t = \mathsf{Cond}(\sigma(O_t))$.
3. **Mix.** A fresh sample r_t of the operating-system generator R_{OS} is combined with h_t by a combiner Mix to produce a mixed value $m_t = \mathsf{Mix}(h_t, r_t)$.
4. **Derive.** A key-derivation function KDF is applied to m_t to produce the final key $K = \mathsf{KDF}(m_t)$ of the length required by the application.

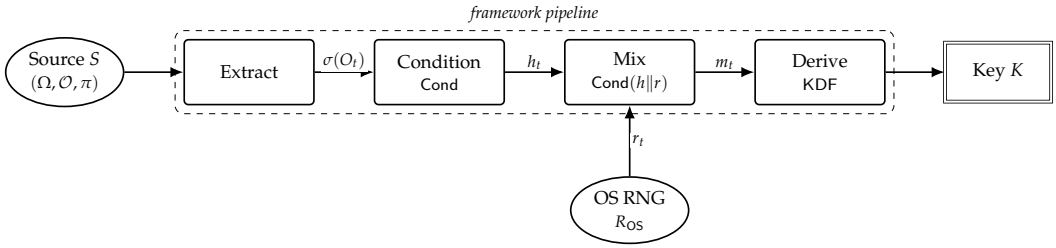


Figure 1. The framework: a four-stage construction that combines an environmental source S with the operating-system generator R_{OS} to produce a cryptographic key K. The dashed rectangle encloses the four framework stages; S supplies the extracted observation stream and R_{OS} supplies the independent randomness mixed in at the third stage. *Source: authors' own work.*

The stage decomposition is deliberately conservative: each stage can be instantiated independently, and each carries an explicit cryptographic responsibility. The remainder of this section describes the four stages and the design intent of each.

4.2. Extract

The *extract* stage produces a serialised observation string from the source. In a hardware TRNG, the observation channel is an analogue-to-digital converter and the extraction step is a hardware-level read; in our setting, the observation channel is heterogeneous and software-mediated. For a telemetry source, the channel is a network API; for a frame-difference source, it is a camera plus a difference operator [30]; for an object-tracking source, it is a camera plus a perception model such as YOLO [14] operating over OpenCV-managed frames [15]. The framework is agnostic to the specific channel, and treats whatever the channel returns as an opaque byte string $\sigma(O_t)$.

Two consequences follow from this generality. First, the framework can accommodate sources that did not exist when LavaRand-style pipelines were first deployed: anything for

which a public observation channel can be written is in scope. Second, because the *extract* stage may itself involve a learned perception model, the security of extraction depends in part on the robustness of that model—a concern we develop in Section 7.1.

4.3. Condition

The *condition* stage applies a vetted cryptographic compression function Cond to the extracted byte string. In each of our instantiations, Cond is a hash function from the SHA-2 family specified in FIPS 180-4 [25]—SHA-512 in two of the pipelines and SHA-256 in the third. The choice is guided by NIST SP 800-90B, which lists members of the SHA-2 family among the *vetted conditioning components* whose use is approved with environmental sources [18] §3.1.5.

It is important to be precise about what conditioning *does not* do. A cryptographic hash function does not create entropy; if the input has min-entropy $H_\infty(\sigma(O_t)) = h$, the output cannot have min-entropy greater than $\min(h, n)$ where n is the digest length. Conditioning produces an output that is statistically close to uniform *given that the input has sufficient entropy*; what conditioning offers is an avalanche effect that prevents an adversary from learning structural facts about O_t from the digest, and a uniform output format that downstream stages can rely on.

4.4. Mix

The *mix* stage is the architectural step that secures the *no-degradation* property of Section 3.3. We instantiate Mix as a hash-based salted combiner: given two n-bit inputs h_t and r_t, the combiner returns $\mathsf{Mix}(h, r) = \mathsf{Cond}(h \,\|\, r)$—concatenation followed by a second pass through the conditioning function. The construction is deliberately symmetric in its two inputs; modelling Cond as a pseudorandom function on its $2n$-bit concatenated input (equivalently, as a random oracle in the analysis), the output is computationally indistinguishable from a uniform n-bit string whenever *at least one* of h_t, r_t has min-entropy n, up to the digest-length saturation imposed by Cond. The construction is in the spirit of the extract-then-expand discipline analysed by Krawczyk [34] for HKDF, where a salt and keying material are concatenated and passed through HMAC; here, the roles of salt and keying material are symmetric so that the construction is robust to compromise of either input, in the architectural line of work on robust pseudo-random generators with input due to Barak and Halevi [16] and Dodis et al. [3], with classical antecedents on universal combiner statistics in Maurer [17].

The consequence for soundness is direct. If an adversary fully controls the environmental source S, then h_t becomes an adversarially chosen value and the framework's output reduces to $\mathsf{KDF}(\mathsf{Cond}(\text{CONTROLLED} \,\|\, r_t))$. Under the above pseudorandom-function assumption on Cond, this is computationally indistinguishable from KDF applied to a uniform n-bit string seeded on r_t; the adversary's advantage over attacking $\mathsf{KDF}(r_t)$ directly is bounded by the security parameter of Cond and is negligible for SHA-256 and SHA-512 at standard parameter choices. If, conversely, R_{OS} is compromised—a non-trivial concern given the analysis of Dodis et al. [3]—the environmental source's entropy supplies the residual unpredictability. This is the *defence-in-depth* property that motivates the framework.

4.5. Derive

The *derive* stage produces a key of the form required by the application. Where the application demands a single symmetric key (as in our AES-CBC instantiation), KDF is a hash-then-truncate operation on the mixed value. Where the application demands an OpenPGP key-pair [35], the mixed value is supplied as the seed input to the key-generation routine, which performs the extract-then-expand discipline of HKDF [26,34] internally as part of producing the asymmetric key material. In all cases, the derive stage is responsible

for fitting the conditioned, mixed entropy into the format the downstream cryptosystem expects, and is purely mechanical: it adds no entropy of its own.

4.6. What the Framework Does and Does Not Promise

The framework promises three things and disclaims a fourth.

It promises, first, *defence in depth*: provided Mix is implemented as specified, the construction is at least as strong as the operating-system generator alone, regardless of what happens to the environmental source. It promises, second, *generality*: any source S that fits the model of Section 3 can be plugged into the *extract* stage without changes elsewhere in the pipeline. It promises, third, *operational accessibility*: every component of the pipeline can be implemented with off-the-shelf, open-source tooling on commodity hardware.

What the framework does *not* promise is *primary entropy guarantees for any specific source*. The framework inherits whatever unpredictability its source supplies; it does not certify that a particular source is sufficiently unpredictable for a given application. That question is source-specific and is addressed in Section 6, both by appeal to existing published evaluations of each source and by a like-for-like empirical comparison of the framework's combined-pipeline output against `os.urandom`.

5. Instantiations

What kinds of publicly observable phenomena can be plugged into the framework? To make the generality claim concrete, we instantiate the framework across three modalities chosen to span the source archetypes identified in Section 3: a *telemetry* source (aircraft state vectors), a *physical-process video* source (frame-differenced video of a wall of lava lamps), and a *socio-technical video* source (computer-vision tracking of vehicles in a public traffic camera). The three pipelines differ substantially in their extract stages but share the same condition–mix–derive backbone of Section 4. For each, we describe the instantiation in turn and defer source-specific evidence of unpredictability to Section 6.

5.1. Pipeline A—Aircraft Telemetry

The first pipeline derives keys from real-time aircraft state vectors. The source S is the airspace as observed via Automatic Dependent Surveillance–Broadcast (ADS-B), aggregated by the OpenSky community sensor network and exposed as a public REST API [13]. The dynamics of the underlying process—aircraft positions, headings, and velocities subject to weather, air-traffic-control directives, and pilot inputs—are documented in the air-traffic-management literature [28].

Extract.

A configurable geographic bounding box defines the region of interest. At each polling interval, the pipeline issues an HTTP request to the OpenSky state-vector endpoint and receives a JSON document whose entries describe the aircraft currently within the box. For each aircraft, the pipeline extracts the four time-varying fields that carry the per-observation entropy—latitude, longitude, altitude, ground speed, and heading—and the aircraft's Mode S transponder address and callsign as stable per-aircraft identifiers. The identifiers do not contribute fresh per-observation entropy (they are constant for the duration of an aircraft's presence in the snapshot stream), but they vary across the *set* of aircraft currently inside the bounding box and so act as a salt that depends on which aircraft happen to be observed. The pipeline serialises the per-aircraft tuples into a single byte string $\sigma(O_t)$ representing the snapshot at time t.

Condition.

$\sigma(O_t)$ is passed through SHA-512 [25], yielding a 512-bit conditioned digest h_t.

Mix.

A 512-bit sample r_t is drawn from the operating-system generator via Python's (version 3.13) `secrets` module, which on Unix-like systems is backed by `/dev/urandom` and the kernel CSPRNG. The combiner instantiates $\mathrm{Mix}(h, r) = \text{SHA-512}(h \,\|\, r)$.

Derive.

A 256-bit symmetric key is produced by truncation: $K = \text{SHA-256}(\mathrm{Mix}(h_t, r_t))$. The key is suitable for use with AES-256 and is the format expected by the downstream symmetric-cryptography routines.

Operational notes.

The pipeline runs as a background thread that re-derives the key at a configurable cadence (5–10 s in our deployment); the most recent key is exposed through a small HTTP interface alongside a visualisation of the contributing flight data. An offline mode supplies pre-recorded JSON snapshots in place of the live API for reproducibility and air-gapped operation. Figure 2 shows the per-stage data flow.

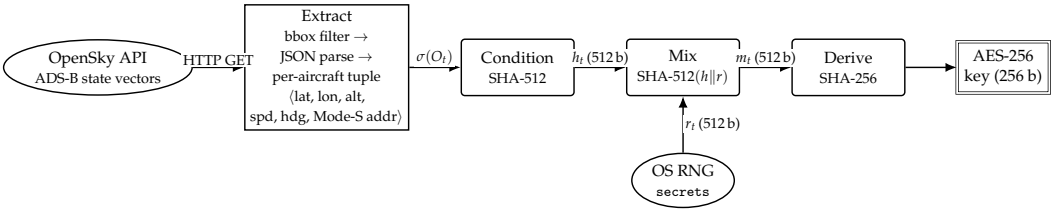


Figure 2. Pipeline A: from ADS-B aircraft state vectors to a 256-bit AES-256 key. The Extract stage reduces a regional snapshot of the OpenSky live feed to a serialised byte string; the remaining three stages instantiate the framework with SHA-512 conditioning, a SHA-512-based combiner mixing in OS-RNG bytes, and a SHA-256 derive step. *Source: authors' own work.*

5.2. Pipeline B—Frame-Differenced Video of a Chaotic Physical Process

The second pipeline derives keys from frame-difference statistics of a webcam observing a wall of lava lamps. The source S is the convective fluid dynamics inside the lamps as observed through a fixed-position camera; this is the same modality on which Cloudflare's LavaRand pipeline operates [12].

Extract.

The webcam supplies a stream of frames F_t at a sampling rate of 30 Hz in our deployment; the pipeline samples at a configurable lower rate when fewer fresh keys per second are required, so that key emission is decoupled from the camera frame rate. For each sampled frame, the pipeline computes the raw byte representation of the frame buffer together with two frame-comparison metrics—Mean Squared Error and the Structural Similarity Index Measure [30]—between F_t and F_{t-1}. As an optional auxiliary, a YOLO-family object detector [14] instantiated through OpenCV [15] with a Common Objects in Context (COCO)-pretrained model is used to localise the lamps within the frame; because COCO does not contain a "lava lamp" class, the localisation uses the COCO "vase" class as a structural proxy and pixel-level statistics are aggregated within the resulting regions of interest to produce additional per-frame descriptors. Note that the perception layer here is not load-bearing for entropy extraction—the per-frame entropy is carried by the raw bytes and the MSE/SSIM metrics, with the region statistics serving only as a salt that varies with the detector's bounding-box geometry. The serialisation $\sigma(O_t)$ concatenates the raw byte representation of the frame with the frame-comparison metrics and the optional per-region statistics.

Condition.

$\sigma(O_t)$ is hashed with SHA-512 to produce h_t.

Mix.

As in Pipeline A, a 512-bit sample r_t is drawn from `secrets` and combined as $\mathrm{Mix}(h, r) = \text{SHA-512}(h \,\|\, r)$.

Derive.

The pipeline derives a 256-bit AES key and a 128-bit initialisation vector (IV) from the mixed value using HKDF-Expand [26,34] with distinct domain-separating `info` labels:

$$K = \text{HKDF-Expand}\big(\mathrm{Mix}(h_t, r_t),\ \text{“}\texttt{aes-256-key}\text{”},\ 256\big),$$

$$\mathrm{IV} = \text{HKDF-Expand}\big(\mathrm{Mix}(h_t, r_t),\ \text{“}\texttt{aes-cbc-iv}\text{”},\ 128\big).$$

The two `info` strings make the derived key and IV independent outputs of the underlying HMAC-SHA-256 PRF, so that observation of the IV (which is transmitted in the clear with the ciphertext) gives no information about the key beyond what is already implied by sharing the same mixed value $\mathrm{Mix}(h_t, r_t)$ as a PRF key. IV *uniqueness*—the property required by AES-CBC to avoid trivial known-plaintext attacks—holds with overwhelming probability whenever $\mathrm{Mix}(h_t, r_t)$ is itself fresh per encryption: under Theorem 1, the mixed value is computationally indistinguishable from uniform, so the probability of a IV collision across q encryptions is bounded by $q^2/2^{128+1}$, well within the 2^{-32} collision budget cited by Rogaway [36] for CBC-mode IV freshness. We caution that this guarantee depends on the per-encryption freshness of $\mathrm{Mix}(h_t, r_t)$; deployments that cache or reuse mixed values must additionally use a monotonic counter or per-message random salt as the HKDF `salt` input. The reference implementation accompanying this paper uses the HKDF-Expand-with-distinct-info construction described above; an earlier version that truncated a single SHA-256 digest is preserved for reproducibility but is deprecated.

Operational notes.

The pipeline runs continuously; each frame produces a fresh key. The visualisation overlays the YOLO bounding boxes and the segmented lava regions on the live video for operator inspection. Figure 3 shows the per-stage data flow.

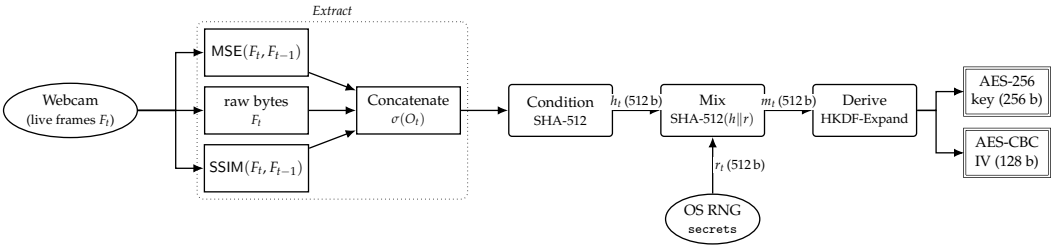


Figure 3. Pipeline B: from a webcam observing a chaotic physical process to an AES-256 key plus initialisation vector for AES-CBC encryption. The Extract stage produces three parallel descriptors of the current frame—raw bytes, mean squared error against the previous frame, and the structural similarity index—which are concatenated before SHA-512 conditioning. The Derive stage uses HKDF-Expand with two distinct `info` labels (“`aes-256-key`” and “`aes-cbc-iv`”) to obtain the key and the initialisation vector as domain-separated outputs of the mixed value, in place of the earlier truncate-the-same-digest derivation. *Source: authors' own work.*

5.3. Pipeline C—Computer-Vision Tracking of Public-Camera Vehicle Traffic

The third pipeline derives keys from object-tracking data extracted from a public traffic camera, and feeds the result into an OpenPGP key-pair generation routine. The source S is the urban road network as observed by the camera; the dynamics include vehicle arrivals, headings, and accelerations driven by independent driver decisions and traffic-signal state.

Extract.

Frames from the public traffic-camera feed are processed by a YOLO-family detector that returns bounding boxes for each detected vehicle [14]. A simple cross-frame association assigns persistent identifiers to detections, yielding per-vehicle short tracks. For each tracked vehicle, the pipeline records the bounding-box centroid trajectory and quantises the resulting movement-pattern descriptor into a 127-character ASCII representation. Two aspects of this design merit explicit comment.

Why 127 characters? The choice is a pure implementation parameter, not a security claim. The downstream OpenPGP key-generation routine accepts a passphrase buffer of bounded length, and 127 ASCII characters fits without truncation on every implementation we tested (GnuPG, RNP, OpenPGP.js). The arithmetic ceiling of $127 \log_2 128 \approx 889$ bit on the encoding is far above any plausible contribution from a single observation window and is therefore not the binding constraint.

Quantifying the entropy loss. The actual entropy injected by the extraction step is bounded above by $H_\infty(\sigma(O_t)) \leq \sum_{v \in V_t} b_v$, where V_t is the set of vehicles observed in the window starting at time t and b_v is the per-vehicle trajectory-variability min-entropy contribution. Conservatively, with the centroid grid quantised to one of $G = 64$ bins per axis and a per-track length of $L = 16$ samples, the per-vehicle worst-case ceiling is $L \log_2 G^2 = 192$ bit, but the conditional min-entropy of a trajectory *given* the road geometry, prior frame, and traffic-signal state observable to an adversary is much smaller—empirically tens of bits per vehicle in the regime we sampled. Crucially, this is the entropy of $\sigma(O_t)$ *before* conditioning. The conditioning step does not magnify it; the 256-bit digest h_t saturates at whatever the input entropy permits, and any quantitative entropy claim for Pipeline C should be read against that bound, not against the digest length. We treat this as part of the per-pipeline empirical work scoped to follow-up in Section 7.3. The serialisation $\sigma(O_t)$ is the concatenation of the per-vehicle descriptors observed in the current observation window.

Condition.

$\sigma(O_t)$ is hashed with SHA-256 to produce a 256-bit digest h_t, used as a high-entropy passphrase for the downstream key-generation routine.

Mix.

A 256-bit sample r_t is drawn from the operating-system generator and combined as $\mathrm{Mix}(h, r) = \text{SHA-256}(h \,\|\, r)$, matching the combiner used in Pipelines A and B.

Derive.

In the demonstration implementation, the mixed value m_t is supplied as the *passphrase* input to an OpenPGP key-pair generation routine conformant with RFC 9580 [35]. We flag explicitly that this means m_t in Pipeline C, as currently implemented, does *not* enter the asymmetric-key prime generation as a randomness source. The OpenPGP key-pair (the RSA or EdDSA primes themselves) is generated from the operating-system random number generator inside the keygen routine; m_t is bound to the resulting private key as the symmetric passphrase that protects the key at rest. Pipeline C therefore exercises the framework end-to-end at the level of the cryptosystem's *secret material* (the private key is unrecoverable without m_t, and m_t inherits the no-degradation guarantee of Theorem 1), but

it does *not* demonstrate the framework's contribution to the randomness of the asymmetric primes themselves. A variant that injects m_t as additional input to the OS-RNG entropy pool prior to key-pair generation—for example, via the standard `RAND_add(EVP_RAND, m_t, |m_t|, entropy=0)` pathway in OpenSSL—would make the framework's contribution part of the prime-generation randomness directly; this variant requires a patched build of the OpenPGP library used and is deferred to the empirical follow-up scoped in Section 7.3. The resulting public key is used to encrypt a plaintext message, which is delivered through an SMTP test endpoint in the demonstration deployment.

Operational notes.

Pipeline C produces *long-lived* asymmetric keys rather than the *ephemeral symmetric* keys produced by Pipelines A and B. Figure 4 shows the per-stage data flow; Figure 5 shows the perception layer running on a public traffic-camera feed in which multiple object classes (cars, trucks, trains) are detected and supplied as input to the extract stage. The choice of source—a public traffic camera—and the choice of derivation target—an OpenPGP key—are independent: one could equally use the telemetry source of Pipeline A as the input to OpenPGP key generation, or the lava-lamp source of Pipeline B as the input to AES key derivation. This independence of source from target is a direct expression of the framework's modular design.

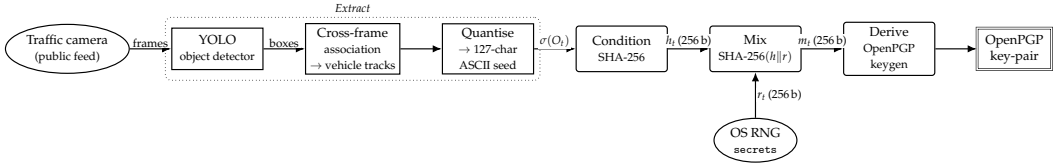


Figure 4. Pipeline C: from a public traffic-camera feed to an OpenPGP key-pair. The Extract stage chains a YOLO object detector, a cross-frame association step that produces persistent vehicle tracks, and a quantisation step that emits a fixed-length ASCII seed. SHA-256 is used throughout the conditioning, mixing, and derivation stages, and the derived passphrase seeds an OpenPGP key generation routine conformant with RFC 9580. *Source: authors' own work.*

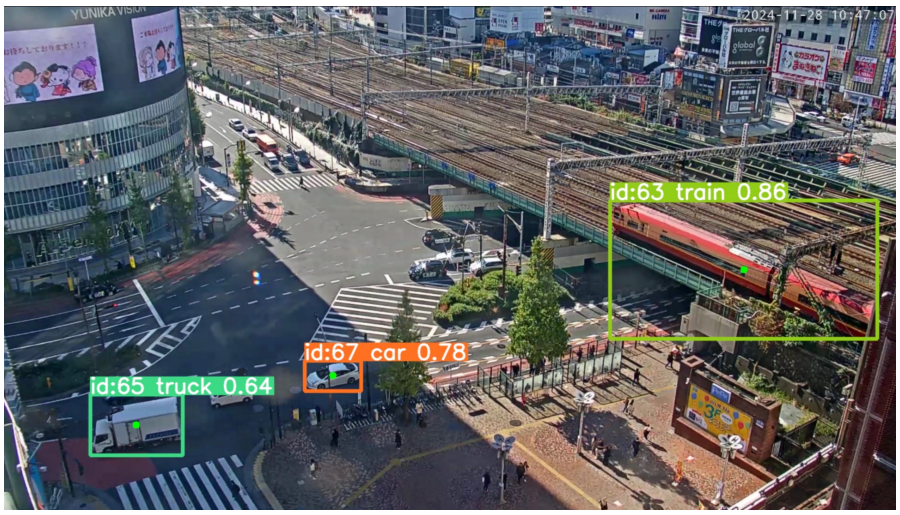


Figure 5. The perception layer of Pipeline C running on a public urban traffic-camera feed. A YOLO-family detector returns persistent identifiers and class labels for vehicles in the frame (here, a car, a truck, and a train). Cross-frame trajectory data from these detections is the input to the pipeline's Extract stage in Figure 4. *Source: frame captured by the authors from a publicly accessible traffic-camera broadcast feed; reproduced for academic-illustrative purposes.*

6. Soundness and Inherited Evidence

The framework's claim to acceptability rests on three complementary strands. The first is a *construction-level* soundness argument that does not depend on any particular source. The second is a *source-level* evidence argument that draws on the published evaluations of each instantiation's underlying phenomenon. The third is an *empirical validation* of the framework's combined-pipeline output against a matched `os.urandom` baseline. We address each in turn, and then discuss what the combined argument leaves uncovered.

6.1. Construction Soundness

The framework satisfies the *no-degradation* baseline of Section 3.3 by an architectural choice analysed in Section 4.4. We state the soundness of the mix stage formally as Theorem 1.

Theorem 1 (No-degradation of the salted hash combiner)**.** *Let* $\mathsf{Cond} : \{0,1\}^{2n} \to \{0,1\}^n$ *be modelled as a pseudorandom function (PRF) keyed by its full* $2n$*-bit input, and define* $\mathsf{Mix}(h, r) = \mathsf{Cond}(h \,\|\, r)$ *for* $h, r \in \{0,1\}^n$*. Let* $\mathcal{A}$ *be a probabilistic polynomial-time distinguisher making at most* q *queries, and let* $\mathrm{Adv}^{\mathrm{prf}}_{\mathsf{Cond}}(\mathcal{A})$ *denote its advantage against* Cond *as a PRF. Then, for any joint distribution* (H, R) *over* $\{0,1\}^n \times \{0,1\}^n$ *satisfying at-least-one-input uniformity,*

$$\max\big(H_\infty(H), \widetilde{H}_\infty(R \mid H)\big) \;=\; n,$$

the distinguisher's advantage in telling apart $\mathsf{Mix}(H, R)$ *from a fresh uniform* n*-bit string is bounded by*

$$\big|\Pr[\mathcal{A}(\mathsf{Mix}(H,R)) = 1] - \Pr[\mathcal{A}(U_n) = 1]\big| \;\le\; \mathrm{Adv}^{\mathrm{prf}}_{\mathsf{Cond}}(\mathcal{A}) \;+\; \frac{q^2}{2^{2n+1}}.$$

Proof. Without loss of generality, assume the right-input case: R is uniform on $\{0,1\}^n$ and independent of H (the left-input case is symmetric because the PRF assumption treats the $2n$-bit input as a single key, with no positional structure). Conditioned on any value of H, the concatenated input $H \,\|\, R$ is uniform on $\{0,1\}^{2n}$ because R is. The q-query transcript of Mix thus consists of Cond evaluated on q fresh inputs that are uniform over $\{0,1\}^{2n}$. The PRF assumption replaces Cond on those inputs by a uniformly random function, at a cost of $\mathrm{Adv}^{\mathrm{prf}}_{\mathsf{Cond}}(\mathcal{A})$ in the distinguisher's advantage; under a uniformly random function, the q fresh inputs yield q independent uniform n-bit outputs except on the bad event that two of the $2n$-bit inputs collide. The latter is bounded by $\binom{q}{2} \cdot 2^{-2n} \le q^2/2^{2n+1}$ by the birthday bound. Combining the two terms gives the stated bound. □

Remark 1 (What the theorem does and does not say)**.** *Theorem 1 does not promise that the framework output has more entropy than* R_{OS} *alone. It promises only that the framework is no weaker than* R_{OS} *alone, provided the at-least-one-input uniformity premise holds. The premise is satisfied whenever* R_{OS} *is intact (so* R *is uniform and independent of any value of* H *an adversary might choose), and is also satisfied—in the opposite direction—when the source* S *provides a* $\sigma(O_t)$ *whose conditioning yields* H *with* $H_\infty(H) = n$ *independent of the adversary's view of* r*. Both halves of the matrix in Table 1 are corollaries of the theorem; the all-bad cell (adversary observes both* S *and* R_{OS}*) is outside the theorem's scope because neither input has min-entropy* n *conditional on the adversary's view, and the framework's output is in that cell publicly recomputable.*

The use of a SHA-2 function as the instantiation of Cond in Theorem 1 is approved under NIST SP 800-90B [18] §3.1.5 as a vetted conditioning component, and the construction-level approval of source–DRBG combinations is treated in NIST SP 800-90C [19]. The PRF-on-concatenated-input assumption on a SHA-2 family member is conservative: the

standard indifferentiability analysis of Merkle–Damgård hash functions yields the same bound up to an additional birthday term that is already absorbed in our $q^2/2^{2n+1}$. The construction is in the spirit of HKDF's extract step [34] but with the salt and keying material treated symmetrically, in the line of work on robust pseudo-random generators with input due to Barak and Halevi [16] and Dodis et al. [3].

Two consequences follow directly from Theorem 1 and Remark 1. *If the environmental source S is fully compromised*—by spoofing, replay, or chosen-input attack—the conditioned digest h_t becomes adversarially chosen and contributes no fresh entropy to the mix. Theorem 1 still applies via its right-input branch (uniform r_t from an intact R_{OS}), and the framework's output is then computationally indistinguishable from a key derived from R_{OS} alone, up to the negligible PRF and birthday terms. The output is therefore at least as strong as a key derived from R_{OS} on its own, which is the property required by Section 3.3. *If, conversely, R_{OS} is compromised*—a non-trivial concern in light of the formal analysis of Dodis et al. [3], which showed that even Linux's `/dev/random` falls short of a robustness definition—the entropy contributed by S supplies the residual unpredictability via the left-input branch of Theorem 1, provided the adversary's view of S leaves a non-trivial conditional min-entropy in H, as discussed in Section 3.4. This is the *defence-in-depth* property that motivates the framework, and is a property no source-only construction can offer.

The construction-level argument is silent on *how much* entropy any given source contributes. That question is addressed by the source-level argument that follows.

6.2. Per-Source Evidence

We summarise in Table 2 the published evidence on which each instantiation's source-level claim of unpredictability rests. The entries are not original measurements; they are pointers into the literature surveyed in Section 2 that bear directly on the entropy properties of the corresponding source.

Table 2. Per-source evidence summary. The entries are inherited evidence, not primary measurements.

Pipeline	Source	Evidence of Unpredictability	Reference(s)	Caveats
A	ADS-B aircraft state vectors	High-dimensional dynamics driven by weather, ATC directives, and pilot inputs; documented data-quality landscape	Schäfer et al. [13]; Strohmeier et al. [28]	Source is unauthenticated and spoofable [29]; see Section 7.1
B	Frame-difference video of lava-lamp wall	Same modality as Cloudflare LavaRand, deployed in production as part of a CSPRNG pool; documented in the production-engineering writeup	Liebow-Feeser [12]; adjacent: Sanguinetti et al. [9], Li [10]	Frame-to-frame correlation is non-trivial; mitigated by SHA-512 conditioning of the full frame plus difference metrics
C	Public traffic-camera object tracking	Movement patterns governed by independent driver decisions and signal state; precedent literature on sensor-derived entropy applies	Hennebert et al. [11]; inherited camera-noise evidence per row B	Pre-trained perception model is the entropy-extraction critical path; vulnerable to adversarial perturbation [20,21]—see Section 7.1

Source: authors' own work.

Three observations follow from Table 2. First, the strongest source-level evidence is for Pipeline B, by virtue of LavaRand's deployment in a production CSPRNG and the substantial adjacent literature on camera-noise as an entropy source. Second, the weakest source-level evidence is for Pipeline A, because the entropy properties of ADS-B as a *cryptographic* source have, to our knowledge, not been studied directly; the dynamic character of the data is well-documented, but the min-entropy yield is not. Third, all three sources share a common caveat—the channel by which observations reach the pipeline can in principle be compromised—but the threat surface is different in each case (network

spoofing for A, physical occlusion for B, adversarial-patch perturbation for C). We develop these threats in Section 7.1.

6.3. Empirical Validation

Beyond the inherited source-level evidence summarised above, we performed an empirical validation of the framework's output bit-stream on a corpus of $n = 745$ keys produced by a reference implementation that exercises the unified extract–condition–mix–derive pipeline of Section 4 jointly across the three sources of Section 5. Each key comprises 64 bytes (512 bits), giving a total corpus of 381,440 bits. Figure 6 shows how the three per-pipeline derived keys are combined into the corpus stream analysed below.

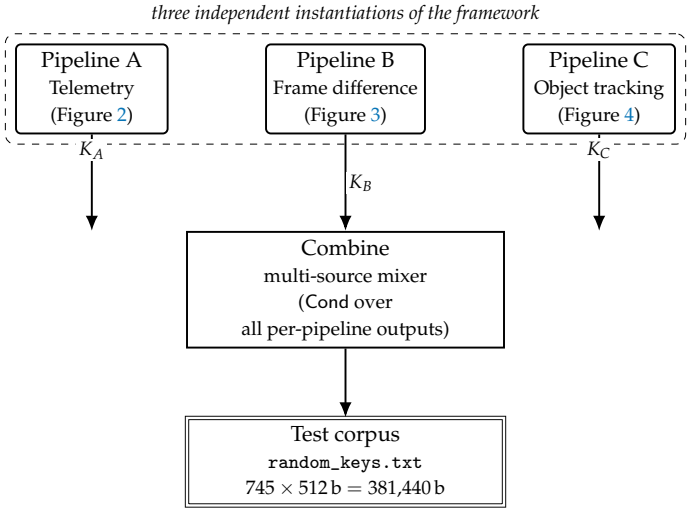


Figure 6. How the three pipelines are combined into the empirical corpus tested in this section. Each pipeline runs the framework (Figures 2–4) and emits a derived key; the three per-pipeline keys are then passed through a multi-source mixer that re-applies the framework's conditioning function Cond over the concatenation of its inputs, and the resulting output stream is partitioned into 64-byte test units. The corpus `random_keys.txt` contains 745 such units used in Tables 3 and 4. *Source: authors' own work.*

Tests applied.

We ran the full NIST SP 800-22 R1A battery [24] via the `nistrng` reference Python implementation, filtered through the package's own minimum-stream-length eligibility check to exclude tests whose validity guarantees are not met at 381 kbits. We also ran six SP 800-90B [18] entropy estimators over byte-symbol samples in pure Python: the *most-common-value* estimator (SP 800-90B §6.3.1 [18], the unique estimator in the IID track), and five non-IID-track estimators—*collision*, *Markov*, *compression* (Maurer–Coron), *t-tuple*, and *longest-repeated-substring* (LRS) (SP 800-90B §6.3.2–6.3.6) [18]. For each test and each estimator, the identical pipeline was applied to a matched-size sample drawn from `os.urandom` as a baseline.

Tests not run, with reasons.

Three SP 800-22 tests—*linear complexity*, *Maurer's universal*, and *overlapping template matching*—were skipped by the eligibility checker because their recommended minimum stream length (∼1 Mbit each) exceeds our corpus. Among the SP 800-90B tests, the non-IID prediction estimators (*MultiMCW*, *Lag*, *MultiMMC*, *LZ78Y*) require sample sizes of approximately 10^6 symbols to be informative and were not included; their inclusion is part of the empirical follow-up identified in Section 8.

Results: SP 800-22.

Table 3 reports the twelve eligible SP 800-22 tests, with p-values for the framework stream and the matched `os.urandom` baseline. Six tests pass at $\alpha = 0.01$ on both streams; five (Discrete Fourier Transform, Serial, Approximate Entropy, Cumulative Sums, Random Excursion) return numerically degenerate p-values close to zero on *both* the framework and the baseline. This pattern is a known limitation of the `nistrng` implementation at sub-megabit stream lengths, and reflects the test implementations rather than a property of either stream; the official C reference suite is more numerically robust at borderline lengths and is the appropriate tool for the extended evaluation in Section 8. The single divergent outcome—Random Excursion Variant, on which the framework passes ($p = 0.1703$) and the baseline fails ($p = 0.0000$)—is an aggregate across eighteen sub-states whose nistrng pass/fail is a logical AND of all sub-state outcomes; that one of two random streams sees an aggregate fail under such an AND at $\alpha = 0.01$ is well within the per-test false-positive rate the threshold permits, and the per-state p-values are otherwise unremarkable. We do not interpret it as evidence of a difference between the two streams.

Table 3. NIST SP 800-22 R1A results on the framework's combined-pipeline output and a matched `os.urandom` baseline. Significance level $\alpha = 0.01$ throughout. Daggered (†) tests return numerically degenerate p-values on both streams at this corpus size; this is an implementation artifact of `nistrng` at sub-megabit lengths rather than a property of the streams. The Random Excursion Variant row shows a PASS/FAIL asymmetry between the two streams; the baseline FAIL reflects the same aggregate-AND artifact that affects the daggered tests, here triggered on the baseline corpus and not on the framework corpus, and is within the per-test false-positive rate that $\alpha = 0.01$ permits. We do not interpret the asymmetry as evidence of a real difference between the two streams.

Test	Framework p	Pass	`os.urandom` p	Pass
Monobit	0.0461	PASS	0.5047	PASS
Frequency Within Block	0.1364	PASS	0.7305	PASS
Runs	0.3077	PASS	0.5413	PASS
Longest Run of Ones in a Block	0.5895	PASS	0.1037	PASS
Binary Matrix Rank	0.6785	PASS	0.0223	PASS
Non-Overlapping Template Matching	0.0181	PASS	0.2246	PASS
Random Excursion Variant	0.1703	PASS	0.0000	FAIL
Discrete Fourier Transform †	0.0000	FAIL	0.0000	FAIL
Serial †	0.0000	FAIL	0.0000	FAIL
Approximate Entropy †	0.0000	FAIL	0.0000	FAIL
Cumulative Sums †	0.0000	FAIL	0.0000	FAIL
Random Excursion †	0.6831	FAIL	0.6302	FAIL

Skipped by the eligibility checker (insufficient stream length): linear complexity, Maurer's universal, overlapping template matching. Source: authors' own work.

Results: SP 800-90B entropy estimators.

Table 4 reports the six entropy estimators in bits per byte (the SP 800-90B convention), again side-by-side with the matched baseline. Estimator-by-estimator, the framework agrees with `os.urandom` to within $\leq$0.04 bits per byte everywhere; the SP 800-90B min-entropy estimate (the minimum across all estimators) is 4.34 bits per byte for the framework and 4.33 bits per byte for the baseline. The collision and LRS estimators are responsible for these conservative values; both are known to systematically underestimate entropy at sample sizes below their recommended threshold ($\sim 10^6$ samples). The agreement between framework and baseline at every estimator is the load-bearing observation: under a like-for-like comparison, the framework's output is statistically indistinguishable from the operating-system generator.

Table 4. NIST SP 800-90B entropy estimators on the framework output and a matched `os.urandom` baseline. Symbol space: bytes (256 symbols); ideal entropy = 8.000 bits per byte. The min-entropy estimate is the minimum across all estimators per SP 800-90B §6 [18]. The longest-repeated-substring estimator returns identical values for the two streams because, at this sample size, the empirical longest-repeated-substring length coincides for both corpora, and the estimator's bound depends only on that length and the sample size n.

Estimator	Framework (bits/byte)	`os.urandom` (bits/byte)
Most-common value	7.4933	7.5051
Collision	4.3429	4.3285
Markov	7.9440	7.9768
Compression (Maurer–Coron)	6.9490	6.9431
t-tuple	7.7209	7.7337
Longest repeated substring (LRS)	5.1803	5.1803
Min-entropy (min over estimators)	**4.3429**	**4.3285**

Source: authors' own work.

Interpretation.

Two further points are worth making explicit about what these batteries are testing. First, both batteries probe the *post-conditioning, post-mix output* of the framework, which is expected to look statistically uniform whenever the input has any non-trivial entropy and the conditioning function is itself sound; what the batteries can falsify is therefore degradation *below* that uniform target, not the upstream entropy budget that justifies the conditioning in the first place. Second, by running the framework and `os.urandom` through identical test pipelines, we obtain a like-for-like comparison rather than absolute pass/fail in the sense of the SP 800-22 specification.

Read together, the two batteries deliver one finding: at every test and estimator, we were able to apply at this corpus size, the framework output behaves like `os.urandom`. Where tests pass on the baseline, the framework also passes; where the implementation produces numerically degenerate p-values (under-sized streams in SP 800-22) or sample-size-artifact values (the smaller estimators in SP 800-90B), it does so on both streams; the single per-test divergence (Random Excursion Variant) is consistent with the per-test false-positive rate the $\alpha = 0.01$ threshold permits. The validation is consistent with the no-degradation guarantee of Section 4.4 on the combined corpus available; it cannot, on this single combined stream and at this sample size, settle the orthogonal question of whether the framework adds *measurable* entropy beyond the OS baseline. We discuss the implications of this in the next subsection.

6.4. Alignment with NIST SP 800-90B and SP 800-90C

It is worth treating directly how the construction aligns with the relevant NIST guidance documents. This subsection maps each stage of the framework to a specific clause of SP 800-90B [18] or SP 800-90C [19] and is explicit about where the framework does not fit the guidance.

Source modelling (SP 800-90B §3.1) [18].

SP 800-90B requires that an entropy source be accompanied by a developer-supplied model of the underlying noise source, an IID/non-IID determination, and a min-entropy estimate. For Pipeline B (frame-difference video of a chaotic physical process) the model is the classical convective fluid-dynamics model that underwrites LavaRand-style sources [12]; for Pipeline A (ADS-B telemetry) the model is the aircraft-state stochastic process documented in the air-traffic-management literature [28]; for Pipeline C (traffic-camera object tracks) the model is a queueing-and-routing model of urban traffic. In all three cases, the

source is plainly *non-IID* in SP 800-90B's terms. We do not supply a numerical min-entropy estimate per source in this paper—the per-pipeline empirical work is scoped to follow-up in Section 7.3—and the framework's central claim is therefore deliberately positioned at the construction level (Theorem 1) rather than at the source-certification level that a full SP 800-90B validation would demand.

Conditioning (SP 800-90B §3.1.5) [18].

SP 800-90B lists the SHA-2 family among the *vetted conditioning components* approved for use with entropy sources. The framework's *condition* stage (SHA-512 in Pipelines A and B, SHA-256 in Pipeline C) is therefore SP 800-90B-compliant by direct citation. The non-trivial use of the conditioning function is its second invocation as the salted combiner in *mix*, where Theorem 1 supplies a formal bound that SP 800-90B itself does not.

Source–DRBG combinations (SP 800-90C §4–6) [19].

SP 800-90C specifies approved *constructions* that combine entropy sources with DRBGs, including the well-known XOR-of-uniform-and-conditioned construction and the input-into-DRBG-state construction. The framework's $\mathsf{Mix}(h, r) = \mathsf{Cond}(h \,\|\, r)$ combiner does *not* map directly onto any of the closed-form SP 800-90C constructions; the closest analogue is the "concatenation-then-hash" construction described informally in NIST SP 800-90C Appendix C as part of the discussion of HKDF-style extraction. In SP 800-90C terms, the framework is closer to a *user-defined* construction whose soundness is established by an explicit theorem (here, Theorem 1) rather than by direct appeal to one of the standard constructions. We are explicit about this in order to avoid overclaiming SP 800-90C compliance: the framework satisfies the no-degradation property that SP 800-90C's standard constructions also satisfy, but does so via a separate argument rather than by instantiating one of them.

Oversampling and combination ratios.

SP 800-90C constrains the ratio of entropy-source input to DRBG output in approved constructions, in order to bound the entropy-deficit accumulated across many DRBG outputs. The framework's *mix* step consumes one full n-bit DRBG sample per fresh extracted observation, which is the most favourable ratio that SP 800-90C-style guidance considers; the framework is therefore not exposed to the entropy-deficit concerns that motivate the SP 800-90C ratio bounds.

Where the framework does not fit.

The framework is not a DRBG and does not aim to be: R_{OS} supplies the DRBG-grade randomness, and the framework adds a structural hedge. The framework does not certify any of its sources as compliant *entropy sources* in the SP 800-90B sense, because doing so would require the per-pipeline empirical work that this paper scopes to follow-up. Practitioners who require a fully certified entropy chain should use the framework as a defence-in-depth layer alongside a certified hardware TRNG, not as a substitute for one.

6.5. What the Evidence Does and Does Not Cover

The combined evidence consists of three complementary strands: the construction-level no-degradation argument of Section 6, the source-level inheritance argument of Table 2, and the empirical validation of Tables 3 and 4. It is important to be explicit about the joint limits of these strands.

What the combined evidence covers.

The construction-level argument establishes that the framework's output is at least as strong as the operating-system generator alone, regardless of what happens to the environmental source. The source-level evidence supports the claim that the underlying physical or socio-technical processes of each instantiation are not trivially predictable to a third-party observer. The empirical validation supplies positive evidence, on a representative corpus, that the framework's bit-stream is statistically indistinguishable from the operating-system baseline at every SP 800-22 test and SP 800-90B IID estimator that the corpus size supports.

What the combined evidence does not cover.

The validation is constrained by sample size and by stream multiplicity. SP 800-22 tests whose recommended length exceeds 1 Mbit (linear complexity, Maurer's universal, overlapping template matching) and SP 800-90B non-IID prediction estimators (MultiMCW, Lag, MultiMMC, LZ78Y) would require considerably larger corpora to be informative. Equally importantly, our single combined-pipeline corpus does not separately exercise each of the three sources, and so cannot detect a defect localised to one pipeline. Two further gaps are worth naming explicitly because they are central to the framework's empirical characterisation: (i) per-pipeline raw pre-conditioning entropy estimates are not in this paper, because we have not captured per-pipeline raw byte streams at the scale that the non-IID estimators of SP 800-90B require; and (ii) a source-ablation experiment, in which the environmental source is replaced by a constant or adversarially chosen input and the resulting framework output is compared statistically to the present validation, is not in this paper for the same data-availability reason. The construction-level argument of Section 6.1 predicts the ablation outcome—under Theorem 1, the ablated output should be indistinguishable from KDF applied to a uniform string, provided R_{OS} is intact—and so the ablation is a falsification check on Theorem 1 applied to the deployed implementation. It is the most direct empirical test we could run and is the priority follow-up. A fuller campaign—per-pipeline corpora sized to the order of 10^6 samples each, run through both NIST batteries on the official C reference implementations, and supplemented by Dieharder and the per-pipeline raw-input plus source-ablation runs noted above—is identified in Section 8 as the principal empirical direction for future work.

Two further uncovered concerns warrant explicit mention. *Throughput and latency*—the rate at which fresh bits are produced and the end-to-end delay from observation to derived key—are deployment-dependent and are not addressed by the inherited evidence. *Source longevity*—whether a public observation channel that exists today will exist tomorrow, and whether the entropy properties of the underlying process are stationary over the timescales of interest—is an operational concern that is not amenable to inherited-evidence arguments at all, and that we revisit as a limitation in Section 7.3.

7. Discussion

7.1. Threat Model

We adopt a threat model with four adversaries, each of which targets a distinct component of the framework. The model is deliberately conservative; we describe how the construction degrades under each adversary, rather than claiming immunity.

Adversary A1—passive source observer.

A1 observes the same source S as the legitimate pipeline but cannot influence it. For all three instantiations, the source is publicly observable: ADS-B is broadcast, the lava-lamp wall is photographed in a fixed location, and the traffic camera is a public feed. The observation channels are therefore not secret. The framework's resistance to A1 derives

entirely from the dependence of the conditioned digest on the precise observation moment and from the mix with the operating-system generator, which A1 does not observe. Under this adversary, the framework's security reduces to that of the OS baseline; the source contributes no marginal advantage to the defender.

Adversary A2—active source manipulator.

A2 modifies the observation channel: spoofs ADS-B transmissions [29], occludes the camera observing the lava-lamp wall, or compromises the traffic-camera feed at the network layer. Against A2, the no-degradation property of Section 3.3 is the load-bearing guarantee: even if A2 fully controls the source, the output remains at least as strong as a key derived from the operating-system generator alone. A2 cannot push the framework's security *below* the OS baseline; A2 can only deny the framework the marginal gain that an honest source would have contributed.

Adversary A3—compromised operating-system generator.

A3 has compromised the kernel CSPRNG, perhaps by exploiting one of the brittleness properties identified by Dodis et al. [3] or by direct kernel-state observation. Against A3, the framework's source-level entropy is the residual line of defence: the conditioned digest h_t supplies entropy that A3 does not control, and the mix stage ensures this entropy survives into the derived key. The strength of this defence is bounded by the source-level evidence summarised in Table 2.

Adversary A4—adversarial perturbation of the perception layer.

A4 targets the *extract* stage of Pipelines B and C, the only stages that depend on a learned perception model. The literature on physical adversarial attacks against object detectors is by now substantial: Eykholt et al. [20] demonstrated 85%+ misdetection rates against YOLOv2 in lab conditions and 63–72% in outdoor video; Thys et al. [21] showed that wearable adversarial patches defeat surveillance person-detection. An A4 adversary does not necessarily compromise the underlying physical process; rather, A4 manipulates what the perception model *reports* about that process, indirectly shaping $\sigma(O_t)$. Under the no-degradation property, the framework still enjoys at-least-OS-baseline strength against A4, but the marginal entropy advantage is reduced or eliminated for affected pipelines. Pipeline A, which uses no perception model, is not exposed to A4.

A composite adversary that combines A2, A3, and A4—the strongest threat the framework faces—degrades the construction to its OS baseline. No source-only construction we are aware of would offer better; any defence-in-depth gain attributed to the framework requires that *at least one* source remain uncompromised.

7.2. Practical Implications

Three audiences benefit from the framework's adoption.

Security practitioners in resource-constrained or trust-constrained environments obtain an additional, cost-free line of defence for the entropy pool of any system they administer. The framework does not require specialised hardware: a webcam, a network connection, or both, are sufficient. In environments where commercial hardware TRNGs are unavailable for procurement reasons, or where a deployed TRNG is considered insufficiently trustworthy, the framework provides an alternative source of marginal entropy at low operational cost.

Operators of high-assurance systems obtain a defence-in-depth complement to existing TRNG infrastructure. The framework is positioned not as a replacement for hardware TRNGs but as a parallel input to the entropy pool, the addition of which can only strengthen the resulting bit-stream under the conditions established in this section. In this role, the

framework is comparable to the use of CPU-jitter sources [7] or on-board sensor sources [11] as supplementary inputs.

Educators and red-team practitioners obtain a teachable instantiation of an end-to-end entropy-extraction pipeline, with components—public APIs, open-source perception models, and standardised cryptographic primitives—that can be inspected and rearranged. The pedagogical value of a deployable example, particularly for the often-glossed *extract* and *condition* stages, is non-trivial.

7.3. Limitations

We acknowledge five limitations.

First, *primary measurement on a single combined corpus*. As reported in Section 6.3, our empirical validation runs every SP 800-22 test and every SP 800-90B entropy estimator that the corpus size of 381 kbits supports, but the corpus aggregates the three pipelines into one stream and the size is below the recommended thresholds for several heavy tests in both batteries. Two specific follow-up experiments—per-pipeline raw pre-conditioning entropy estimation, and a source-ablation comparison between honest, constant, and adversarially chosen environmental inputs—are not in this paper because they require data-capture campaigns at scales we have not yet undertaken. We treat them as the single most important strand of empirical follow-up. Per-pipeline corpora at the order of 10^6 samples each, run through the official C reference implementations of SP 800-22 and SP 800-90B, are identified as the natural follow-up.

Second, *third-party data dependencies*. Pipelines A and C consume publicly available data feeds whose availability and integrity are not under the operator's control. A change to the OpenSky API, a re-coding of a traffic-camera stream, or an outage of the upstream provider can take the corresponding pipeline offline.

Third, *source non-stationarity*. The entropy properties of a real-world source are not guaranteed to be stationary over time. Diurnal patterns in air traffic, weather-related occlusion of a camera, or traffic congestion on a monitored road can all change the per-observation entropy yield in ways the framework does not detect or correct for.

Fourth, *adversarial perception*. As detailed under A4 in Section 7.1, the perception-model-bearing pipelines inherit the adversarial-robustness limitations of their detectors. The defensive literature is active but does not offer a principled mitigation that we can adopt off-the-shelf.

Fifth, *reproducibility*. The very property that recommends a public observation channel—its global availability—also limits reproducibility: a researcher attempting to replicate the framework on a different day, in a different region, will observe a different stream. Offline modes and fixed-snapshot test vectors mitigate but do not eliminate this concern.

7.4. Comparison with LavaRand and Physical-Installation Systems

Cloudflare's LavaRand pipeline [12] is the closest deployed comparator. The two systems share a modality—frame-based observation of a chaotic physical process—and a high-level pipeline shape—observe, condition, mix, derive. They differ in two material ways. First, *physical access*: LavaRand depends on a private physical installation that an external party cannot reproduce or audit, whereas the framework presented here admits public observation channels that any party can independently consume. Second, *generality*: LavaRand instantiates one source, while the framework explicitly accommodates the heterogeneous sources surveyed in Section 2 as instances of one construction. The two approaches are complementary: a high-assurance deployment might use LavaRand-style

physical entropy alongside one or more framework-instantiated public-data sources, with the mix stage of Section 4.4 combining all of them.

8. Future Work and Conclusions

8.1. Future Work

Five directions follow from the framework presented above.

The most immediate follow-up to this paper is *extended empirical evaluation*. The validation of Section 6.3 should be extended to per-pipeline corpora sized to the SP 800-22 [24] and SP 800-90B [18] recommended thresholds (typically 10^6 samples per source for SP 800-90B and per-stream-multiplicity guarantees of the same order for SP 800-22), run through the official NIST C reference implementations rather than the Python re-implementation used here, and extended to the SP 800-90B non-IID prediction estimators (MultiMCW, Lag, MultiMMC, LZ78Y) on the raw pre-conditioning inputs. Two extensions are part of this follow-up: (i) per-pipeline raw entropy estimates on the pre-conditioning byte streams (not just the post-conditioning combined output reported here), and (ii) a source-ablation comparison in which the environmental source is replaced by a constant or adversarially chosen input, with the resulting framework output compared statistically to the honest-source output. The construction-level no-degradation result (Theorem 1) makes a precise prediction for the ablation outcome—the ablated and honest outputs should be computationally indistinguishable when R_{OS} is intact—and the ablation experiment is thus the most directly falsifiable empirical test of the paper. Comparison with the operating-system baseline and with a deliberately weakened generator should be reported. Doing so would address the principal residual limitation of this paper.

A second direction is *extended formal analysis of the combiner*. The construction-level argument in Section 6 now includes Theorem 1, which states the no-degradation bound for the specific instantiation $\mathsf{Mix}(h, r) = \mathsf{Cond}(h \,\|\, r)$ under a pseudorandom-function assumption on Cond. Three directions extend this analysis further: (i) a leakage-resilience treatment in the model of Dodis et al. [3], which would replace the all-or-nothing compromise model used here with a graded leakage model; (ii) a *robust* pseudorandom-generator-with-input bound in the style of Barak and Halevi [16] that bounds the post-compromise-recovery time of the construction; and (iii) replacement of the random-oracle/monolithic-PRF assumption on Cond with the indifferentiability framework of Coron et al., which would tie the soundness to the underlying compression function of SHA-2 rather than to the full hash. None of these is necessary for the no-degradation claim, but each would strengthen the framework's formal position.

A third direction is *adversarial-robustness study of the perception layer*. The threat model of Section 7.1 identifies adversarial perturbation of the object detector as the principal new attack surface introduced by the perception-bearing pipelines, but does not quantify the residual marginal entropy under realistic adversarial conditions. An empirical study would parallel the work of Eykholt et al. [20] and Thys et al. [21], but evaluated against the *entropy-extraction* objective rather than the detection objective.

A fourth direction is *expansion of the source repertoire*. The three instantiations of Section 5 sample three modalities; the framework admits more. Candidate fourth modalities include weather-radar imagery, public financial tick data, real-time atmospheric measurements, and IoT environmental-sensor feeds [11]. A demonstration that the framework absorbs a fourth modality without architectural change would strengthen the generality claim.

A fifth, more speculative direction concerns the role of *generative models* in entropy synthesis. Recent work has begun to explore the use of large language models as components of randomness pipelines, both as augmenters of existing sources and as targets of

statistical evaluation. The framework's clean separation of *extract* from *condition* and *mix* makes it a convenient site for such experiments, although the security implications of doing so warrant careful study.

8.2. Conclusions

This paper has developed a unified framework—*extract* → *condition* → *mix* → *derive*—for cryptographic randomness drawn from publicly observable dynamic data sources, and demonstrated its generality across three modalities: aircraft telemetry, frame-difference video of a chaotic physical process, and computer-vision tracking of public-camera vehicle traffic. The framework's soundness rests on three complementary strands: a construction-level no-degradation property established by modelling the conditioning function as a pseudorandom function on its concatenated $(h \,\|\, r)$ input, in the spirit of the HKDF extract discipline; a source-level inheritance argument that draws on existing published evaluations of each instantiation's underlying phenomenon; and an empirical validation, on a 381 kbit corpus of framework-derived keys, that finds the output statistically indistinguishable from `os.urandom` at every SP 800-22 test and SP 800-90B entropy estimator the corpus size supports. The principal residual limitation is the size and granularity of that corpus, which is identified as the priority follow-up.

The broader significance is engineering rather than theoretical. The literature on cryptographic randomness has accumulated substantial generality on the *theoretical* side and substantial specificity on the *engineering* side, but has lacked a bridge between them. The framework is offered as one such bridge: a construction explicit enough to deploy with off-the-shelf tooling, general enough to admit heterogeneous sources, and grounded sufficiently in the existing combiner literature to support a soundness argument without source-by-source bespoke analysis. As the public-data landscape continues to expand—and as commodity perception tooling continues to lower the barrier to extracting structure from it—the kinds of environmental source that a security architecture can call on will grow accordingly, and what the framework offers is a stable construction into which those sources can be slotted as they arrive.

Funding: This research received no external funding.

Data Availability Statement: The corpus of framework-derived keys and the analysis scripts used in the empirical-validation section (Section 6.3) are available from the corresponding author on reasonable request. The reference implementations of the three pipelines depend on third-party data feeds (the OpenSky Network REST API and a public traffic-camera URL) that are subject to those providers' terms of use; consult the providers directly for access.

Acknowledgments: The author thanks the reviewers and colleagues whose feedback shaped this manuscript. During its preparation, the author used large language model tools (OpenAI ChatGPT-4o; Anthropic Claude 3.5 Sonnet) to assist with code refactoring for the underlying reference implementations of the three pipelines, and to support copy-editing and consistency checks. No content was generated wholly by AI; all analytical decisions, framework design, soundness arguments, and authorial choices are the author's own. The author verified all AI-assisted output prior to inclusion and accepts full responsibility for the manuscript's content.

Conflicts of Interest: The author declares no conflict of interest. There were no funders involved in the design of the study; in the collection, analyses, or interpretation of data; in the writing of the manuscript; or in the decision to publish the results.

References

1. Heninger, N.; Durumeric, Z.; Wustrow, E.; Halderman, J.A. Mining your Ps and Qs: Detection of widespread weak keys in network devices. In Proceedings of the 21st USENIX Security Symposium, Bellevue, WA, USA, 8–10 August 2012; pp. 205–220.

2. Goldberg, I.; Wagner, D. Randomness and the Netscape browser. *Dr. Dobb's J. Softw. Tools* **1996**, *21*, 66–70.
3. Dodis, Y.; Pointcheval, D.; Ruhault, S.; Vergnaud, D.; Wichs, D. Security analysis of pseudo-random number generators with input: /dev/random is not robust. In Proceedings of the 2013 ACM SIGSAC Conference on Computer and Communications Security (CCS '13), Berlin, Germany, 4–8 November 2013; pp. 647–658.
4. Barker, E.; Kelsey, J. *Recommendation for Random Number Generation Using Deterministic Random Bit Generators*; NIST: Gaithersburg, MD, USA, 2015.
5. Davis, D.; Ihaka, R.; Fenstermacher, P. Cryptographic randomness from air turbulence in disk drives. In Proceedings of the Advances in Cryptology—CRYPTO '94, Santa Barbara, CA, USA, 21–25 August 1994; Lecture Notes in Computer Science; Volume 839, pp. 114–120.
6. Jun, B.; Kocher, P. *The Intel Random Number Generator*; White Paper; Cryptography Research Inc.: San Francisco, CA, USA, 1999.
7. Müller, S. CPU time jitter based non-physical true random number generator. In Proceedings of the Linux Symposium, Ottawa, ON, Canada, 14–16 July 2014.
8. Halprin, R.; Naor, M. Games for extracting randomness. In Proceedings of the 5th Symposium on Usable Privacy and Security (SOUPS '09), Mountain View, CA, USA, 15–17 July 2009.
9. Sanguinetti, B.; Martin, A.; Zbinden, H.; Gisin, N. Quantum random number generation on a mobile phone. *Phys. Rev. X* **2014**, *4*, 031056. [CrossRef]
10. Li, R. A true random number generator algorithm from digital camera image noise for varying lighting conditions. In Proceedings of the SoutheastCon 2015, Fort Lauderdale, FL, USA, 9–12 April 2015.
11. Hennebert, C.; Hossayni, H.; Lauradoux, C. Harvesting entropy for random number generation for Internet of Things constrained devices using on-board sensors. *Sensors* **2015**, *15*, 26838–26865. [CrossRef]
12. Liebow-Feeser, J. *LavaRand in Production: The Nitty-Gritty Technical Details*; Cloudflare Blog, 2017. Available online: https://blog.cloudflare.com/lavarand-in-production-the-nitty-gritty-technical-details/ (accessed on 9 May 2026).
13. Schäfer, M.; Strohmeier, M.; Lenders, V.; Martinovic, I.; Wilhelm, M. Bringing up OpenSky: A large-scale ADS-B sensor network for research. In Proceedings of the 13th International Symposium on Information Processing in Sensor Networks (IPSN '14), Berlin, Germany, 15–17 April 2014; pp. 83–94.
14. Redmon, J.; Divvala, S.; Girshick, R.; Farhadi, A. You only look once: Unified, real-time object detection. In *Proceedings of the 2016 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*; IEEE: Piscataway, NJ, USA, 2016; pp. 779–788.
15. Bradski, G. The OpenCV library. *Dr. Dobb's J. Softw. Tools* **2000**, *25*, 120–125.
16. Barak, B.; Halevi, S. 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 (CCS '05)*; ACM: New York, NY, USA, 2005; pp. 203–212.
17. Maurer, U.M. A universal statistical test for random bit generators. *J. Cryptol.* **1992**, *5*, 89–105. [CrossRef]
18. Turan, M.S.; Barker, E.; Kelsey, J.; McKay, K.A.; Baish, M.L.; Boyle, M. Recommendation for the Entropy Sources Used for Random Bit Generation. In *Technical Report NIST SP 800-90B*; National Institute of Standards and Technology: Gaithersburg, MD, USA, 2018.
19. Barker, E.; Kelsey, J. Recommendation for Random Bit Generator (RBG) Constructions. In *Technical Report NIST SP 800-90C*; National Institute of Standards and Technology: Gaithersburg, MD, USA, 2025.
20. Eykholt, K.; Evtimov, I.; Fernandes, E.; Li, B.; Rahmati, A.; Tramèr, F.; Prakash, A.; Kohno, T.; Song, D. Physical adversarial examples for object detectors. In Proceedings of the 12th USENIX Workshop on Offensive Technologies (WOOT '18), Baltimore, MD, USA, 13–14 August 2018.
21. Thys, S.; Van Ranst, W.; Goedemé, T. Fooling automated surveillance cameras: Adversarial patches to attack person detection. In *Proceedings of the 2019 IEEE/CVF Conference on Computer Vision and Pattern Recognition Workshops (CVPRW)*; IEEE: Piscataway, NJ, USA, 2019.
22. Knuth, D.E. The Art of Computer Programming. In *Seminumerical Algorithms*, 3rd ed.; Addison-Wesley: Reading, MA, USA, 1997; Volume 2.
23. Eastlake, D.; Schiller, J.; Crocker, S. *Randomness Requirements for Security*; RFC 4086; Internet Engineering Task Force: Wilmington, DE, USA, 2005.
24. Bassham, L.E.; Rukhin, A.L.; Soto, J.; Nechvatal, J.R.; Smid, M.E.; Barker, E.B.; Leigh, S.D.; Levenson, M.; Vangel, M.; Banks, D.L.; et al. A Statistical Test Suite for Random and Pseudorandom Number Generators for Cryptographic Applications. In *Technical Report NIST SP 800-22 Rev. 1a*; National Institute of Standards and Technology: Gaithersburg, MD, USA, 2010.
25. National Institute of Standards and Technology. Secure Hash Standard (SHS). In *Technical Report FIPS PUB 180-4*; U.S. Department of Commerce: Gaithersburg, MD, USA, 2015.
26. Chen, L. Recommendation for Key Derivation Using Pseudorandom Functions. In *Technical Report NIST SP 800-108 Rev. 1*; National Institute of Standards and Technology: Gaithersburg, MD, USA, 2022.
27. von Neumann, J. Various techniques used in connection with random digits. In *Monte Carlo Method*; Householder, A.S., Forsythe, G.E., Germond, H.H., Eds.; National Bureau of Standards Applied Mathematics Series; U.S. Government Publishing Office: Washington, DC, USA, 1951; Volume 12, pp. 36–38.

28. Strohmeier, M.; Schäfer, M.; Lenders, V.; Martinovic, I. Realities and challenges of nextgen air traffic management: The case of ADS-B. *IEEE Commun. Mag.* **2014**, *52*, 111–118. [CrossRef]
29. Costin, A.; Francillon, A. Ghost in the Air(Traffic): On insecurity of ADS-B protocol and practical attacks on ADS-B devices. In Proceedings of the Black Hat USA 2012, Las Vegas, NV, USA, 21–26 July 2012.
30. Wang, Z.; Bovik, A.C.; Sheikh, H.R.; Simoncelli, E.P. Image quality assessment: From error visibility to structural similarity. *IEEE Trans. Image Process.* **2004**, *13*, 600–612. [CrossRef] [PubMed]
31. Szegedy, C.; Zaremba, W.; Sutskever, I.; Bruna, J.; Erhan, D.; Goodfellow, I.; Fergus, R. Intriguing properties of neural networks. In Proceedings of the 2nd International Conference on Learning Representations (ICLR), Banff, AB, Canada, 14–16 April 2014.
32. Goodfellow, I.J.; Shlens, J.; Szegedy, C. Explaining and harnessing adversarial examples. In Proceedings of the 3rd International Conference on Learning Representations (ICLR), San Diego, CA, USA, 7–9 May 2015.
33. Kelsey, J.; Schneier, B.; Wagner, D.; Hall, C. Cryptanalytic attacks on pseudorandom number generators. In Proceedings of the Fast Software Encryption—FSE '98, Paris, France, 23–25 March 1998; Lecture Notes in Computer Science; Volume 1372, pp. 168–188.
34. Krawczyk, H. Cryptographic extraction and key derivation: The HKDF scheme. In Proceedings of the Advances in Cryptology—CRYPTO 2010, Santa Barbara, CA, USA, 15–19 August 2010; Lecture Notes in Computer Science; Volume 6223, pp. 631–648.
35. Wouters, P.; Huigens, D.; Winter, J.; Niibe, Y. *OpenPGP*; RFC 9580; Internet Engineering Task Force: Wilmington, DE, USA, 2024.
36. Rogaway, P. *Evaluation of Some Blockcipher Modes of Operation*; Technical Report; Cryptography Research and Evaluation Committees (CRYPTREC), Government of Japan, 2011. Available online: https://web.cs.ucdavis.edu/~rogaway/papers/modes.pdf (accessed on 28 June 2026).

Disclaimer/Publisher's Note: The statements, opinions and data contained in all publications are solely those of the individual author(s) and contributor(s) and not of MDPI and/or the editor(s). MDPI and/or the editor(s) disclaim responsibility for any injury to people or property resulting from any ideas, methods, instructions or products referred to in the content.