JEnt v2.2.0 LFSR Conditioning Analysis

Document Version: TRD14
Date: August 17, 2026

By:
Joshua E. Hill
KeyPair Consulting Inc.
987 Osos Street
San Luis Obispo, CA 93401
United States

Yvonne Cliff
Teron Labs
Level 2, 14 Moore St
Canberra, ACT 2601
Australia

Abstract

In this paper we analyze the non-vetted LFSR-based conditioning component included in JEnt v2.2.0. We present an explicit linear model of this conditioning component, characterize its diffusion properties, and develop a heuristic analysis of its conditioned output min entropy which yields an estimate of 42.3 bits of min entropy per 64-bit conditioned output when the osr is set using an analysis of post-filtered data; the linear conditioning analysis does not apply to more modern versions of JEnt. For JEnt v2.2.0 – v3.6.3 we also describe the hardware-dependent impact of feeding the conditioning function a stuck-free filtered data stream and bound the resulting entropy loss under an IID assumption. We then describe the interaction between the conditioning analysis and the SP 800-90B non-vetted conditioning requirements, including the observed min entropy assessment distribution for the conditioned output statistical testing that SP 800-90B requires for non-vetted conditioning components. We conclude with guidance for selecting alternate parameters that maximize either the per block output min entropy or min entropy per unit time.

Keywords

Jitter Entropy Library (JEnt), LFSR, linear model, entropy of filtered data, entropy source, SP 800-90B, non-vetted conditioning, min entropy estimate, Entropy Source Validation (ESV).

Table of Contents

1 Introduction

The CPU Jitter Random Number Generator (JEnt) library v2.2.0 ([JEnt v2.2.0], released September 2019) has been widely integrated into many cryptographic modules and it forms the basis of several entropy sources with ESV certificates (e.g., #E8, #E19, #E20, #E37, #E47, #E48, #E50, #E54, #E59, #E60, #E61, #E62, #E90, #E99, #E117, #E151, #E174, #E175, #E226, #E235, #E315, #E325, #E328, #E337). JEnt v2.2.0 was the first version that intended compliance to NIST SP 800-90B, but a substantial amount of development has occurred in the years since JEnt v2.2.0's release and there are many more recent JEnt versions that are more amenable to validation against the SP 800-90B requirements. There are several challenges when validating JEnt v2.2.0 against the requirements of SP 800-90B, [Hill 2023] but it is still useful to discuss the SP 800-90B (sometimes non-)compliance of this codebase, as it is widely integrated into settings that make replacement difficult (e.g., many Linux kernel versions include a JEnt version that is based on JEnt v2.2.0).

One change that was made to modern versions of JEnt was the inclusion of a vetted conditioning function to replace the Linear Feedback Shift Register (LFSR) used as a conditioning component in JEnt v2.2.0. This new vetted conditioner makes validation to the conditioning requirements of SP 800-90B much more straightforward. In the instance where SP 800-90B validation of an entropy source using JEnt v2.2.0 is desired, the [JEnt Design] document provides substantial validation guidance in many areas, but the included conditioning analysis is largely limited to an application of the [SP 800-90B §3.1.5.1.2] random map analysis, which is only sufficient for analysis of vetted conditioning functions[1]. As such, the vendor and test lab are left with the task of creating a technical assessment of the impact of the non-vetted LFSR-based conditioner used within the entropy source.

The levels of conditioned output min entropy documented in the published JEnt v2.2.0 ESV certificates have been inconsistent over time and are chiefly in a range that suggests that the documented values were constrained by the required [SP 800-90B §3.1.5.2] statistical entropy assessment of a conditioned sequential dataset[2].

In this paper, we present a theoretical basis for modeling this non-vetted LFSR-based conditioning, provide verification that this model is equivalent to the conditioning implemented within JEnt v2.2.0, bound the impact of conditioning only stuck-filtered data (which applies to JEnt v2.2.0 – v3.6.3) under an IID assumption, and present one possible approach to using the described linear model to find a defensible estimate for entropy of 42.3 bits of min entropy per conditioned 64-bit output block under the assumption that $\mathtt{osr}$ is set reasonably using post-filtered data[3].

We also provide the distribution of statistical entropy assessment (h') results for essentially any non-vetted conditioning function whose output is suitably pseudorandom; this distribution applies to most non-vetted conditioning components. We finally describe some optional changes to JEnt v2.2.0 that would allow for higher conditioned output block min entropy claims (56.05 to 59.75 bits of min entropy per 64-bit block of conditioned

[1] Note that the included discussion in [JEnt Design §7.2.6] presumes that $\mathtt{osr} = 1$ and misinterprets the meaning of h' described in [SP 800-90B §3.1.5.2], consequently omitting any impact from the required statistical min entropy assessment of a conditioned sequential dataset described in [SP 800-90B §3.1.1]. The argument in [JEnt Design §7.2.26] is the statement "An LFSR with a primitive and irreducible polynomial is considered to not diminish the entropy", which is true of *some* LFSR-based designs, but is certainly not true with *this* conditioning component used as it is, as the LFSR input data rate is always greater than the LFSR output data rate.

[2] Most (i.e., all but #E315 and #E328) JEnt v2.2.0 ESV certs seem to credit entropy values that could credibly be derived from only statistical testing of the conditioned output data (results in the range 56.22 to 60 bits of min entropy per 64-bit block of conditioned output). Conditioned min entropy claims less than the expected statistical entropy assessment range may indicate the entropy claim was reduced as a consequence of using the LFSR-specific mathematical evidence referenced in [SP 800-90B §3.2.3, Requirement 5] and [IG D.K, Resolution 5]. [NIST SHALL ID #52] and [NIST SHALL IDs #106-#107] are marked with contradictory requirement statuses, which suggests that such mathematical evidence may not have been a requirement early in the ESV program, but the current status of these requirements is not clear.

[3] The appropriate value for osr is dependent on the underlying hardware configuration and JEnt configuration, and it is set based on the entropy assessment process; it is not a free parameter.

output for roughly 95% of such validations)[4], though this increased claim comes at the cost of decreasing the entropy output from the entropy source per unit time.

1.1 Related Work

Linear post-processing of biased physical noise sources has been studied extensively. Some of the structural implications of LFSR-based conditioning (or more broadly, linear conditioning in general) were explored in [BL 2008, §II.D] and the initial development here is broadly similar.

[Lacharme 2008] and [Lacharme 2009] follow on the development in [Dichtl 2007] and analyze the use of both linear and non-linear extractors, including LFSR-based constructions for noise sources that produce output that is both identically distributed and independent on a per-bit basis. These papers give constructions with provable output-bias based on the minimum distance of the associated linear code. Our setting differs in the input model (as noted in Assumption A2); in JEnt the bits within a raw symbol (both bit-to-bit within a single raw symbol and symbol-to-symbol) are neither identically distributed nor independent. Most high-order bits are fixed and zero and the per-symbol variation is concentrated in a hardware-dependent subset of low-order bits. As such, the per-input-bit bounded-bias models underlying those results do not apply directly.

More broadly, [BST 2003] sets out an extractor-theoretic framework and argues for constructions that remain sound under a changing environment. The LFSR analyzed here is a fixed linear map with no public parameter and falls outside that framework. Sunar, Martin and Stinson [SMS 2007] give a related example of using linear codes to extract entropy and to obtain provable guarantees under an active-attacker model.

On the validation side, the German AIS 20/31 methodology [PS 2024] requires a stochastic model of the noise source and its post-processing, which is analogous to what the analysis in Section 3.3 constructs for the SP 800-90B setting. This style of analysis is nominally required by SP 800-90B (e.g., [SP 800-90B §3.2.3, Requirement 5], [IG D.K, Resolution 5]), though the enforcement of these requirements has been inconsistent over the history of the ESV program.

There has been limited exploration of the behavior of the SP 800-90B entropy estimators on pseudorandom data; this first occurred for some specialized distributions in [KMS 2015] (which focuses on predictor-based entropy estimation) and [Johnston 2017], and later for pseudorandom and specialized distributions in [HJ 2019].

1.2 Notation In Use

This paper uses the notation presented in this section. Bolded terms represent vector or matrix quantities, and all matrices are 64×64 binary matrices.

Symbol	Meaning	First Introduced
Algebraic Objects		
$\mathbb{F}_2$	The finite field with two elements.	Section 2
$\mathbb{F}_2^{64}$	The 64-dimensional vector space over $\mathbb{F}_2$.	Section 2
$\mathbb{F}_{2^{64}}$	A finite field with 2^{64} elements, isomorphic to $\mathbb{F}_2[z]/\langle p(z)\rangle$.	Section 2
$\mathbb{F}_2[z]$	The polynomial ring in one variable over $\mathbb{F}_2$.	Section 2
$\mathbb{F}_2[z]/\langle p(z)\rangle$	A quotient field realization of $\mathbb{F}_{2^{64}}$.	Section 2

[4] The top values in these ranges are high-probability-mass atoms; see Section 4.2 for details.

Symbol	Meaning	First Introduced
$\mathbb{F}_{2^{64}}^{\times}$	The multiplicative group of $\mathbb{F}_{2^{64}}$, of order $2^{64}-1$.	Section 2
$p(z)$	The primitive characteristic polynomial of the LFSR; the reciprocal of $p^{\star}(z)$.	Section 2
$p^{\star}(z)$	Feedback polynomial of the LFSR.	Section 2
$\boldsymbol{M}^{\mathrm{T}}$	The transpose of the matrix $\boldsymbol{M}$.	Section 2.1
$\boldsymbol{M}^{-1}$	The multiplicative inverse of the matrix $\boldsymbol{M}$.	Section 2.1
$\boldsymbol{I}$	The 64×64 identity matrix.	Section 2.1
$\mathbf{0}$	The zero column vector in $\mathbb{F}_2^{64}$.	Section 2.1
$\langle \boldsymbol{x}, \boldsymbol{y} \rangle_{\boldsymbol{M}}$	The bilinear form $\boldsymbol{x}^{\mathrm{T}}\, \boldsymbol{M}\boldsymbol{y}$.	Section 2.2
Conditioner Model		
$\boldsymbol{s}$, $\boldsymbol{s}_j$	The LFSR internal state and the jth LFSR internal state, both as 64-bit column vectors.	Section 2.1
$\boldsymbol{x}$, $\boldsymbol{x}_j$	The current raw symbol and the jth raw symbol supplied to the conditioning component (respectively), both as 64-bit column vectors.	Section 2.1
$\boldsymbol{f}(\boldsymbol{s}, \boldsymbol{x})$, $\boldsymbol{f}^{(j)}(\boldsymbol{s}, \boldsymbol{x})$	The conditioning function and the jth bit of this function (respectively).	Section 2.1
$\boldsymbol{A}_{\text{single}}$	The matrix representing a single LFSR step.	Section 2.1
$\boldsymbol{A}$	The matrix representing the 64 LFSR steps used in a single round of conditioning; $\boldsymbol{A} = \boldsymbol{A}_{\text{single}}^{64}$.	Section 2.1
$\boldsymbol{B}$	The matrix used for loading a raw symbol into the internal state from an initial state of $\mathbf{0}$; symmetric and invertible.	Section 2.1
$\boldsymbol{g}_j(\boldsymbol{x})$	The per-input processing, $\boldsymbol{A}^j \boldsymbol{B}\, \boldsymbol{x}$**;** the transform applied to a symbol that entered j rounds earlier.	Equation (3)
$\boldsymbol{o}_j(\boldsymbol{s}, \boldsymbol{x}_1, \boldsymbol{x}_2, \ldots \boldsymbol{x}_j)$	The LFSR state (and hence the conditioner output) after j raw symbols ($\boldsymbol{x}_1, \ldots \boldsymbol{x}_j$) have been input to the conditioner after starting with the state $\boldsymbol{s}$.	Equation (4)
Raw Data and Filtering		
Δ_j	The jth raw value (i.e., prior to applying the stuck filtering); the difference between the current timestamp and the prior round's timestamp.	Section 3.3.2
m	Filter probability mass: the probability that the "stuck" test discards a raw sample.	Section 3.3.2
`osr`	The JEnt oversampling rate.	Section 2
H	Per-symbol min entropy of the raw data stream prior to stuck filtering.	Section 3.3.2
H'	Per-symbol min entropy of the data supplied to the conditioner after stuck filtering.	Equation (5)

Symbol	Meaning	First Introduced
$h_{\text{submitter}}$	The submitter's claimed per-symbol min entropy, $1/\texttt{osr}$.	Section 3.3.3
Heuristic Conditioning Analysis		
$p_{\max}$, $p'_{\max}$	Probability of the most likely raw symbol and filtered symbol (respectively).	Section 3.3.2
ε, ε_i	Bias of a biased bit, written $p = \frac{1+\varepsilon}{2}$ with $0 \leq \varepsilon \leq 1$, and the ith such bias (respectively).	Section 3.3.3
w	The number of raw symbols input to the conditioner between output blocks.	Section 3.3.3
$h_{\text{heuristic}}$	The heuristic conditioned-output min entropy estimate developed in this paper.	Section 3.3
$h_{\text{heuristic}}^{\text{uniform}}(H', w)$	The closed form of the $h_{\text{heuristic}}$estimate under the analysis assumptions.	Equation (8)
δ_j	Marginal increase in $h_{\text{heuristic}}$ caused by the introduction of the jth raw symbol input into the conditioner.	Section 4.2.3.2
SP 800-90B Conditioning Parameters (Per-Conditioning Stage)		
n	The raw data symbol width in bits (64 in this application).	Table 3
n_{in}	Input bits to the conditioning function ($w \times n$).	Equation (9)
n_{out}	Number of bits output from the conditioning function (64).	Equation (9)
nw	The narrowest width of the conditioning function (64).	Table 3
h_{in}	Total input entropy per conditioned output ($w \times H$).	Equation (9)
h_{out}	The assessed conditioned output entropy.	Equation (9)
h'	Bitstring min entropy assessment of the conditioned dataset.	Equation (9)
Output_Entropy$(\cdot)$	The SP 800-90B random-map entropy accumulation function.	Equation (9)
α	Probability that h' is in the described bounds.	Table 6
k	The number of raw symbols input into the stuck filter prior to output from the stuck filter.	Section 4.1
t-Tuple Estimator		
t	Length of the longest substring that repeats at least 35 times in the sample.	Section 4.2.2.2
$Q[i]$	The number of occurrences of the most common i-length substring in the data.	Section 4.2.2.2
$P[i]$	The observed probability of this i-length substring.	Section 4.2.2.2
$P_{\max}[i]$	A normalized per-symbol probability associated with this most common i-length substring.	Section 4.2.2.2

Symbol	Meaning	First Introduced
$\hat{p}_{\max}$	The maximum observed value of $P_{\max}[i]$ $(1 \leq i \leq t)$.	Section 4.2.2.2
Selection of the w Parameter		
$\bar{h}_{\text{out}}$	The expected value of h_{out}.	Section 4.2.3.3
$\mathcal{S}$	Scaling factor for w, with $w = \lceil \mathcal{S} \times 64 \rceil \times osr$.	Section 4.2.3.1
E	The efficiency with respect to the number of input symbols ($\frac{\bar{h}_{\text{out}}}{w}$).	Equation (10)
E_{original}	The efficiency of the original behavior.	Equation (10)
$E_{\text{normalized}}$	The ratio of E to $\mathrm{E}_{\text{original}}$.	Equation (10)

2 LFSR Modeling

The conditioning component used within JEnt v2.2.0 is a 64-bit LFSR used in a multiplicative scrambler mode, where the raw symbol is fed into the LFSR a bit at a time, clocking the LFSR after each bit is fed in, until all 64 bits have been integrated into the LFSR. This is repeated until $64 \times$ osr "non-stuck" raw symbols are fed in, at which point the current 64-bit state of the LFSR is used as the output of the LFSR conditioning component.

The feedback polynomial for the LFSR in use is $p^{\star}(z) = z^{64} + z^{61} + z^{56} + z^{31} + z^{28} + z^{23} + 1$, which corresponds to the characteristic polynomial $p(z) = z^{64} \times p^{\star}\left(\frac{1}{z}\right) = z^{64} + z^{41} + z^{36} + z^{33} + z^{8} + z^{3} + 1$, which is primitive. [Živković 1994] In other words, $p(z)$ is a primitive (and thus irreducible) polynomial over $\mathbb{F}_2$, so the quotient ring $\mathbb{F}_2[z]/\langle p(z)\rangle$ is isomorphic to the finite field $\mathbb{F}_{2^{64}}$ and has a root that generates the multiplicative group of the field. More precisely, the quotient ring element $z + \langle p(z)\rangle$ generates the full multiplicative group associated with that field (that is, the set $\{(z + \langle p(z)\rangle)^j\}_{j=1}^{2^{64}-1}$ is a group under multiplication of order $2^{64} - 1$ and is (group) isomorphic to $\mathbb{F}_{2^{64}}^{\times}$).

2.1 LFSR Conditioning as Linear Transforms

We can connect this base LFSR operation to the LFSR in multiplicative scrambler mode. The processing of an LFSR in this multiplicative scrambler mode can be expressed as a function of the current state ($\boldsymbol{s}$) and the input ($\boldsymbol{x}$); we denote this function as $\boldsymbol{f}(\boldsymbol{s}, \boldsymbol{x})$, where these bolded terms represent column vector quantities from the vector space $\mathbb{F}_2^{64}$. Each bit of the function's output is simply an XOR of specific (feedback polynomial-determined) fixed bit places of the initial state and the input data bits. Using the commutativity of XOR and the non-effect of XORing zero bits[5], we then see that we can represent the jth bit as a sum of the effect due to the initial state and the effect of the input value, that is

$$f^{(j)}(\boldsymbol{s}, \boldsymbol{x}) = f^{(j)}(\boldsymbol{s}, \boldsymbol{0}) + f^{(j)}(\boldsymbol{0}, \boldsymbol{x}).$$

This applies for all bit places, so the vector-valued function can be similarly decomposed

$$\boldsymbol{f}(\boldsymbol{s}, \boldsymbol{x}) = \boldsymbol{f}(\boldsymbol{s}, \boldsymbol{0}) + \boldsymbol{f}(\boldsymbol{0}, \boldsymbol{x}). \quad (1)$$

[5] This is a direct consequence of the field and vector space axioms for the underlying field and vector space ($\mathbb{F}_2$ and $\mathbb{F}_2^{64}$, respectively).

Though the same feedback polynomial is used for both the current state and the shifted in data, the fact that the current state is fully present initially and the data is shifted in a bit at a time makes the functions for the initial state and the input distinct.

The effect of an LFSR on each parameter is linear, that is functions $\boldsymbol{f}(\boldsymbol{s}, \boldsymbol{0})$ and $\boldsymbol{f}(\boldsymbol{0}, \boldsymbol{x})$ are linear in their respective variables so we can express their respective actions using matrix multiplication. We can then represent the function as a sum (i.e., XOR) of distinct linear operators, namely:

$$\boldsymbol{f}(\boldsymbol{s}, \boldsymbol{x}) = \boldsymbol{A}\boldsymbol{s} + \boldsymbol{B}\boldsymbol{x}, \tag{2}$$

where $\boldsymbol{A}$ and $\boldsymbol{B}$ are both 64×64 binary matrices that encode the impact of the (characteristic-polynomial-dependent) LFSR feedback on the corresponding parameter. The matrix $\boldsymbol{B}$ is the matrix that loads the input vector (raw symbol) into the internal LFSR state under the assumption that the initial state was $\boldsymbol{0}$, and the matrix $\boldsymbol{A}$ is the matrix that performs 64 LFSR operations on a specific LFSR internal state.

These matrices are easy to calculate by supplying the standard basis to each parameter separately, which completely specifies the behavior of these linear maps for all inputs. See [Code] for code that generates and tests these matrices under a "little endian" convention, where the least significant bit (lsb) of the LFSR state corresponds to the first value in the column vector, and using the standard basis to produce the matrix. This code also performs stochastic testing that compares the results of this matrix-based representation with those of the LFSR implementation from JEnt v2.2.0 to help convince readers who are less comfortable with the guarantees provided by the underlying linear algebra structure.

Both $\boldsymbol{A}$ and $\boldsymbol{B}$ are invertible, and do not commute with each other. The matrix $\boldsymbol{B}$ is symmetric (i.e., it is equal to its transpose); because $\boldsymbol{B}$ is symmetric and invertible, its inverse $\boldsymbol{B}^{-1}$ is also symmetric.

The operation represented by the matrix $\boldsymbol{A}$ is 64 iterated LFSR operations. We can also calculate the corresponding single-round version by using a subset of the conditioning code that performs only a single LFSR step; we call the matrix representation of the action of this single-step LFSR conditioning $\boldsymbol{A}_{\text{single}}$.

$\boldsymbol{A}_{\text{single}}$ is similar (i.e., conjugate) to the companion matrix of the primitive polynomial $p(z)$ and each iteration of the LFSR (i.e., multiplication by $\boldsymbol{A}_{\text{single}}$) corresponds to multiplication by a generator of $\mathbb{F}_{2^{64}}^{\times}$ under a specific change of basis. The primitive nature of the characteristic polynomial $p(z)$ shows us that if we start at any non-zero state and iteratively apply the LFSR, then we cycle through the full multiplicative group $\mathbb{F}_{2^{64}}^{\times}$ in a fixed (feedback polynomial-dependent) pattern, thus we are guaranteed that the corresponding LFSR will have the maximal period of $2^{64} - 1$ for any non-zero starting state.

We have directly verified that the characteristic polynomial for $\boldsymbol{A}_{\text{single}}$ is $p(z)$, as expected. We also verified the order of the underlying LFSR by noting that $\boldsymbol{A}_{\text{single}}^{2^{64}-1} = \boldsymbol{I}$, and verifying that this is not true for any proper divisor of $2^{64} - 1$ and that

$$\boldsymbol{A} = \boldsymbol{A}_{\text{single}}^{64}.$$

Because $\gcd(2^{64} - 1, 64) = 1$, we see that it is also the case that $\boldsymbol{A}$ has order $2^{64} - 1$, which we similarly verified.

We can visualize these binary matrices as in the following diagrams (where a white value is 0 and a 1 entry is depicted in color). The matrix $\boldsymbol{A}_{\text{single}}$ is of the expected form; note that the feedback polynomial can be directly inferred from the first row with this basis selection.

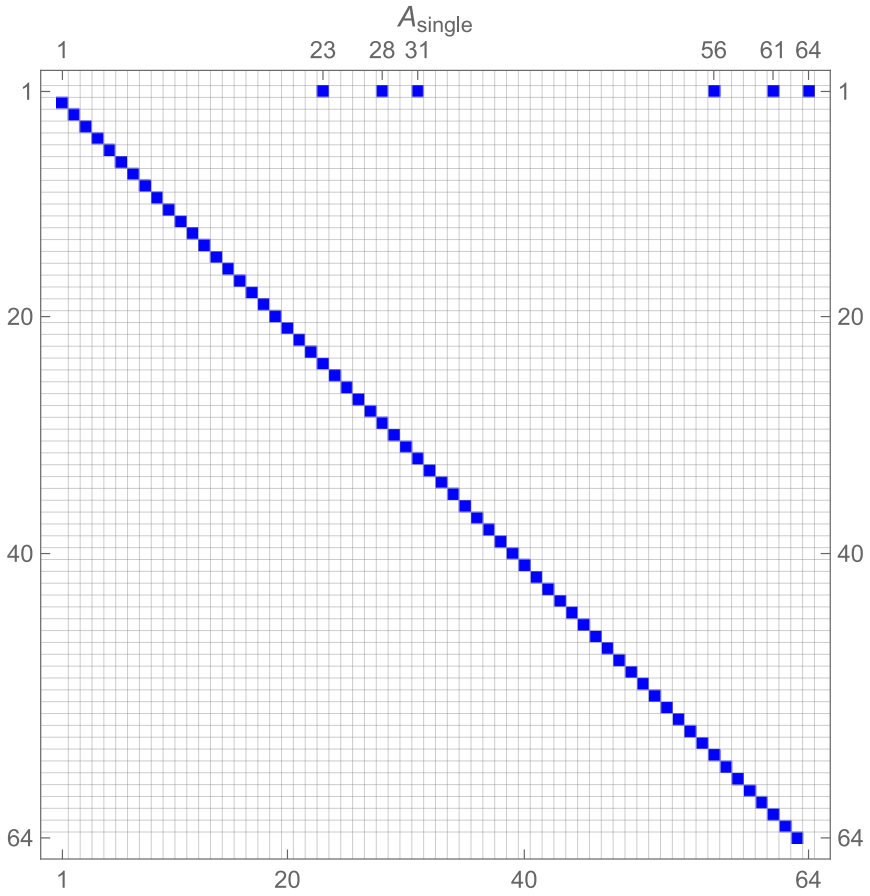

Figure 1. A_{single} in Matrix Form

For the other matrices it is not quite as easy to directly read out design information, but one can get some sense for the action of the repeated LFSR operation ($\boldsymbol{A}$) and the load-in transform ($\boldsymbol{B}$).

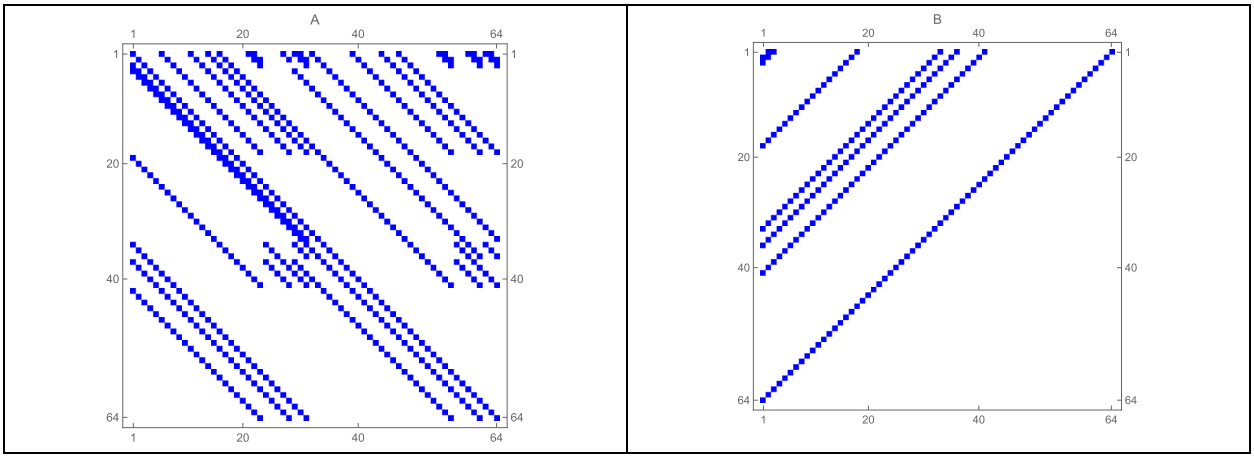

Figure 2. The Matrices A (Left) and B (Right)

2.2 The Conditioning in Closed Form

To simplify later notation, we write the per-input processing as

$$\boldsymbol{g}_j(\boldsymbol{x}) = \boldsymbol{A}^j \boldsymbol{B}\, \boldsymbol{x}. \tag{3}$$

We note that the LFSR state for the current step is wholly established by the LFSR state after the prior step has completed, which is in turn dependent on the initial LFSR state and all prior raw symbols input. This gives the following recurrence relation:

$$\begin{aligned}\boldsymbol{o}_j(\boldsymbol{s}_j, \boldsymbol{x}_j) &= \boldsymbol{f}(\boldsymbol{s}_j, \boldsymbol{0}) + \boldsymbol{f}(\boldsymbol{0}, \boldsymbol{x}_j) \\ &= \boldsymbol{f}(\boldsymbol{o}_{j-1}(\boldsymbol{s}_{j-1}, \boldsymbol{x}_{j-1}), \boldsymbol{0}) + \boldsymbol{f}(\boldsymbol{0}, \boldsymbol{x}_j).\end{aligned}$$

Expanding the first few terms of this recurrence relation gives us

$$\begin{aligned}\boldsymbol{o}_1(\boldsymbol{s_1}, \boldsymbol{x_1}) &= \boldsymbol{f}(\boldsymbol{s}_1, \boldsymbol{0}) + \boldsymbol{f}(\boldsymbol{0}, \boldsymbol{x}_1) \\ &= \boldsymbol{A}\boldsymbol{s}_1 + \boldsymbol{B}\boldsymbol{x}_1 \\ &= \boldsymbol{A}\boldsymbol{s}_1 + \boldsymbol{g}_0(\boldsymbol{x}_1).\end{aligned}$$

As $\boldsymbol{s}_2 = \boldsymbol{o}_1(\boldsymbol{s}_1, \boldsymbol{x}_1)$, we have

$$\begin{aligned}\boldsymbol{o}_2(\boldsymbol{s}_2, \boldsymbol{x}_2) &= \boldsymbol{f}(\boldsymbol{o}_1(\boldsymbol{s}_1, \boldsymbol{x}_1), \boldsymbol{0}) + \boldsymbol{f}(\boldsymbol{0}, \boldsymbol{x}_2) \\ &= \boldsymbol{f}(\boldsymbol{A}\boldsymbol{s}_1 + \boldsymbol{B}\boldsymbol{x}_1, \boldsymbol{0}) + \boldsymbol{f}(\boldsymbol{0}, \boldsymbol{x}_2) \\ &= \boldsymbol{A^2}\boldsymbol{s}_1 + \boldsymbol{A}\boldsymbol{B}\boldsymbol{x}_1 + \boldsymbol{B}\boldsymbol{x}_2 \\ &= \boldsymbol{A^2}\boldsymbol{s}_1 + \boldsymbol{g}_1(\boldsymbol{x}_1) + \boldsymbol{g}_0(\boldsymbol{x}_2).\end{aligned}$$

Similarly, we have $\boldsymbol{s}_3 = \boldsymbol{o}_2(\boldsymbol{s}_2, \boldsymbol{x}_2)$, so

$$\begin{aligned}\boldsymbol{o}_3(\boldsymbol{s}_3, \boldsymbol{x}_3) &= \boldsymbol{f}(\boldsymbol{o}_2(\boldsymbol{s}_2, \boldsymbol{x}_2), \boldsymbol{0}) + \boldsymbol{f}(\boldsymbol{0}, \boldsymbol{x}_3) \\ &= \boldsymbol{f}(\boldsymbol{A^2}\boldsymbol{s}_1 + \boldsymbol{A}\boldsymbol{B}\boldsymbol{x}_1 + \boldsymbol{B}\boldsymbol{x}_2, \boldsymbol{0}) + \boldsymbol{f}(\boldsymbol{0}, \boldsymbol{x}_3) \\ &= \boldsymbol{A^3}\boldsymbol{s}_1 + \boldsymbol{A^2}\boldsymbol{B}\boldsymbol{x}_1 + \boldsymbol{A}\boldsymbol{B}\boldsymbol{x}_2 + \boldsymbol{B}\boldsymbol{x}_3 \\ &= \boldsymbol{A^3}\boldsymbol{s}_1 + \boldsymbol{g}_2(\boldsymbol{x}_1) + \boldsymbol{g}_1(\boldsymbol{x}_2) + \boldsymbol{g}_0(\boldsymbol{x}_3).\end{aligned}$$

From the evident pattern we can expand to the general case:

$$\boldsymbol{o}_j(\boldsymbol{s}_1, \boldsymbol{x}_1, \boldsymbol{x}_2, \ldots \boldsymbol{x}_j) = \boldsymbol{A}^j \boldsymbol{s}_1 + \sum_{k=1}^{j} \boldsymbol{g}_{j-k}(\boldsymbol{x}_k)\,. \tag{4}$$

The initial state of the LFSR is $\boldsymbol{0}$, so if we are tracking the evolution of the LFSR state from the beginning, the first summand has no impact (as $\boldsymbol{A}^j\boldsymbol{0} = \boldsymbol{0}$).

This general form suggests that we should be interested in the matrices of the form $\boldsymbol{A}^j\boldsymbol{B}$, as these completely define all processing on the inputs prior to the final XOR.

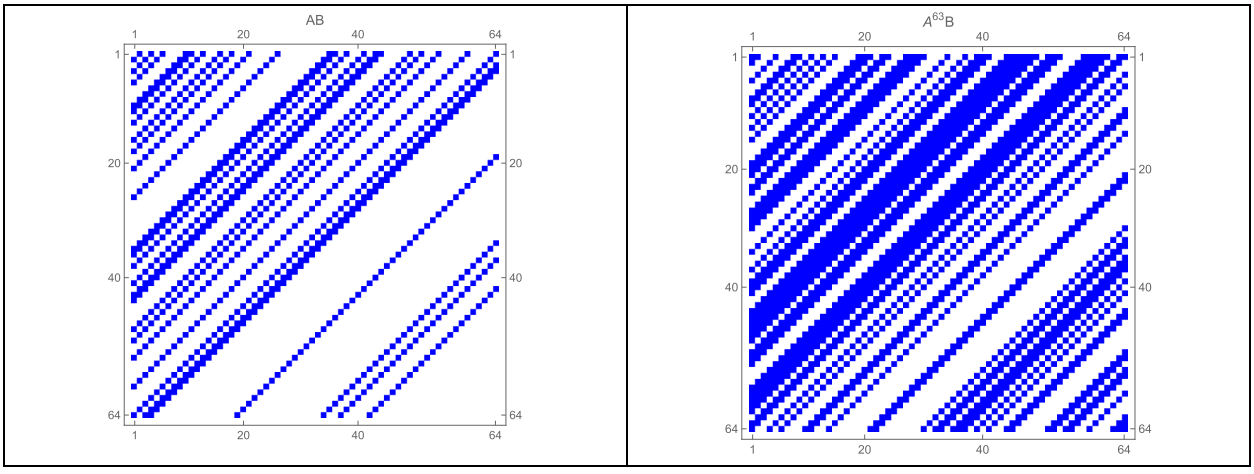

Figure 3. Some Example Matrices of the Form $\boldsymbol{A}^j\boldsymbol{B}$

Matrices of the form $\boldsymbol{A}_{\text{single}}^{j}\boldsymbol{B}$ and $\boldsymbol{A}^{j}\boldsymbol{B}$ are symmetric (that is $\left(\boldsymbol{A}_{\text{single}}^{j}\boldsymbol{B}\right)^{\text{T}} = \boldsymbol{A}_{\text{single}}^{j}\boldsymbol{B}$) for all non-negative integers j. This is because **B** is non-singular and symmetric (so $\boldsymbol{B}^{\text{T}} = \boldsymbol{B}$) and direct calculation[6] shows that

$$\boldsymbol{B}\boldsymbol{A}_{\text{single}}^{\text{T}} = \boldsymbol{A}_{\text{single}}\boldsymbol{B},$$

and thus

$$\boldsymbol{A}_{\text{single}}^{\text{T}} = \boldsymbol{B}^{-1}\boldsymbol{A}_{\text{single}}\boldsymbol{B}.$$

Examining

$$\left(\boldsymbol{A}_{\text{single}}^{j}\right)^{\text{T}} = \left(\boldsymbol{A}_{\text{single}}^{\text{T}}\right)^{j} = \left(\boldsymbol{B}^{-1}\boldsymbol{A}_{\text{single}}\boldsymbol{B}\right)^{j} = \boldsymbol{B}^{-1}\boldsymbol{A}_{\text{single}}^{j}\boldsymbol{B},$$

we then see

$$\begin{aligned}\left(\boldsymbol{A}_{\text{single}}^{j}\boldsymbol{B}\right)^{\text{T}} &= \boldsymbol{B}^{\text{T}}\left(\boldsymbol{A}_{\text{single}}^{\text{T}}\right)^{j} \\ &= \boldsymbol{B}\left(\boldsymbol{B}^{-1}\boldsymbol{A}_{\text{single}}\boldsymbol{B}\right)^{j} \\ &= \boldsymbol{B}\boldsymbol{B}^{-1}\boldsymbol{A}_{\text{single}}^{j}\boldsymbol{B} \\ &= \boldsymbol{A}_{\text{single}}^{j}\boldsymbol{B}.\end{aligned}$$

This is true for all non-negative j, so in particular, it is true for multiples of 64, thus similarly we have

$$(\boldsymbol{A}^{j}\boldsymbol{B})^{\text{T}} = \boldsymbol{A}^{j}\boldsymbol{B}.$$

The $\boldsymbol{B}\boldsymbol{A}_{\text{single}}^{\text{T}} = \boldsymbol{A}_{\text{single}}\boldsymbol{B}$ observation also tells us that $\boldsymbol{A}_{\text{single}}$ is self-adjoint with respect to the symmetric non-degenerate bilinear form

$$\langle \boldsymbol{x}, \boldsymbol{y}\rangle_{\boldsymbol{B}^{-1}} = \boldsymbol{x}^{\text{T}}\,\boldsymbol{B}^{-1}\boldsymbol{y},$$

as

$$\begin{aligned}\left\langle \boldsymbol{A}_{single}\boldsymbol{x}, \boldsymbol{y}\right\rangle_{\boldsymbol{B}^{-1}} &= \boldsymbol{x}^{\text{T}}\boldsymbol{A}_{\text{single}}^{\text{T}}\boldsymbol{B}^{-1}\boldsymbol{y} \\ &= \boldsymbol{x}^{\text{T}}\boldsymbol{B}^{-1}\boldsymbol{A}_{\text{single}}\boldsymbol{B}\boldsymbol{B}^{-1}\boldsymbol{y} \\ &= \boldsymbol{x}^{\text{T}}\boldsymbol{B}^{-1}\boldsymbol{A}_{\text{single}}\boldsymbol{y} \\ &= \left\langle \boldsymbol{x}, \boldsymbol{A}_{single}\boldsymbol{y}\right\rangle_{\boldsymbol{B}^{-1}}.\end{aligned}$$

Similarly, $\boldsymbol{A}_{\text{single}}^{\text{T}}$ is self-adjoint with respect to $\langle \boldsymbol{x}, \boldsymbol{y}\rangle_{\boldsymbol{B}}$.

[6] To verify, use the $\boldsymbol{A}_{\text{single}}$ and $\boldsymbol{B}$ matrices produced by [Code], perform the two specified matrix operations and manually verify equality.

3 The Impact of LFSR Processing on Min Entropy

3.1 Min Entropy of Each Summand

We have found that the LFSR conditioning function's output is the sum (XOR) of the iteratively LFSR-processed initial internal state and the $64 \times \mathrm{osr}$ LFSR processed symbols that were input.

This presentation of the per-raw symbol processing highlights the importance of the choice of feedback polynomial: each output is essentially the sum (XOR) of increasingly LFSR-processed versions of the input values.

The function $\boldsymbol{g}_j(\boldsymbol{x})$ is bijective: in this processing, an input is loaded into the LFSR using the matrix $\boldsymbol{B}$, which is invertible (and can thus be reversed by multiplying by the matrix inverse $\boldsymbol{B}^{-1}$), and repeated LFSR processing is equivalent to iterative left multiplication by the matrix $\boldsymbol{A}$. The matrix $\boldsymbol{A}$ is invertible, so all these matrix multiplications can be inverted by left multiplying the result with $\boldsymbol{A}^{-1}$ the same number of times. Formally we have

$$\boldsymbol{g}_j^{-1}(\boldsymbol{y}) = \boldsymbol{B}^{-1}\boldsymbol{A}^{-j}\boldsymbol{y}.$$

A quick verification shows that $\boldsymbol{g}_j^{-1}(\boldsymbol{y})$ is the (two-sided) inverse function of $\boldsymbol{g}_j(\boldsymbol{x})$: [7]

$$\begin{aligned}\boldsymbol{g}_j^{-1}\left(\boldsymbol{g}_j(\boldsymbol{x})\right) &= \boldsymbol{B}^{-1}\boldsymbol{A}^{-j}(\boldsymbol{A}^j\boldsymbol{B}\boldsymbol{x})\\ &= \boldsymbol{B}^{-1}(\boldsymbol{A}^{-j}\boldsymbol{A}^j)\boldsymbol{B}\boldsymbol{x}\\ &= (\boldsymbol{B}^{-1}\boldsymbol{B})\boldsymbol{x}\\ &= \boldsymbol{x}\end{aligned} \qquad \begin{aligned}\boldsymbol{g}_j\left(\boldsymbol{g}_j^{-1}(\boldsymbol{y})\right) &= \boldsymbol{A}^j\boldsymbol{B}(\boldsymbol{B}^{-1}\boldsymbol{A}^{-j}\boldsymbol{y})\\ &= \boldsymbol{A}^j(\boldsymbol{B}\boldsymbol{B}^{-1})\boldsymbol{A}^{-j}\boldsymbol{y}\\ &= (\boldsymbol{A}^j\boldsymbol{A}^{-j})\boldsymbol{y}\\ &= \boldsymbol{y}\end{aligned}$$

As this per-summand input processing is bijective, the entropy present in each summand is necessarily unchanged by this portion of the conditioning. This statement should *not* be taken to mean that the overall conditioning is bijective; we will see that it is not for any parameter selections used within JEnt v2.2.0.

3.2 Statistics of the Evolution of Individual Summands

This presentation of the processing performed by the LFSR conditioning component makes it clear that, as more raw symbols are input to the conditioning component, the state resulting from each previously input raw symbol ($\boldsymbol{B}\boldsymbol{x}_i$) is additionally LFSR processed. The internal state after each input is the sum (XOR) of all these terms, so it is useful to consider the statistical properties of streams produced by each term of the sum $(\boldsymbol{A}^j\boldsymbol{B}\,\boldsymbol{x}_i)_{j=0}^{n-1}$, as this gives us some intuition of the impact of the LFSR processing as more raw symbols are integrated into the internal state. Such streams are called truncated m-sequences.

LFSRs have a long history of use as random number generators (see [Golomb 2017, Chapter 3], [PTVF 2011], and [Knuth 1998] for general treatments), particularly in hardware applications. LFSR output has some known statistical flaws, and LFSR output streams are generally distinguishable from the output stream of a true random number generator. This manifests in a number of ways:

- LFSR outputs have a fixed algebraic relationship which can be detected through a linear complexity test.
- When the entirety of the state is output, there is substantial serial correlation due to the per-invocation shift being equivalent to a "multiply by 2". This results in an obvious relationship between many adjacent outputs which is evident when adjacent outputs are plotted as x/y values. In this application a single conditioning step results in 64 LFSR operations, which completely cycles the internal state, so the output internal state essentially contains only those bits that would have been shifted out in a bit outputting LFSR.

[7] Evidencing an explicit inverse function is sufficient to show that this transform is *in some sense* a bijection, but to explicitly satisfy the requirement in [IG D.K, Resolution 3], note that the kernel of the linear operator $\boldsymbol{g}_j(\boldsymbol{x})$ is trivial so this map is injective and the dimensions of the co-domain and domain are finite and equal, which (with injectivity) tells us that the map is also surjective. As this map is both injective and surjective, it is bijective.

- Output bits display better auto-correlation results than expected for a random bitstream.
- There can be no repeated output values until the entire multiplicative group has cycled through; this is detectable in large datasets.
- More generally, there are frequency domain anomalies in LFSR states, particularly a notable bias towards low frequencies.

Output distinguishability concerns are not directly relevant for this application, but we will later care if the data streams are distinguishable using the SP 800-90B tests. For each possible initial raw symbol, $\boldsymbol{x}_i$ (in some example raw data distribution), we iteratively produced the output $(\boldsymbol{A}^j \boldsymbol{B}\, \boldsymbol{x}_i)_{j=0}^{124999}$, resulting in 10^6 bytes of output data per initial symbol. We then produced a bitwise assessment for this data sample (as done with conditioned outputs in [SP 800-90B §3.1.5.2]). Raw data distributions consisting of very few possible input values necessarily produce "bursty" assessment result histograms which prevent comparison with our reference distribution. As such, we characterize all possible initial values for a system consistent with a raw data (wider, Intel-TSC-style sub-) distribution with 256 possible values for this testing.

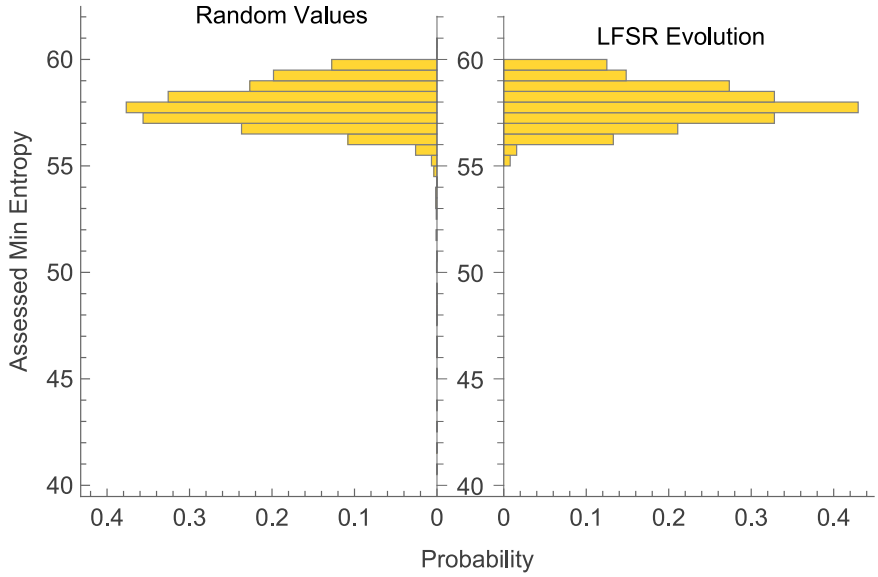

Figure 4. Comparison of Assessments for Random Data vs. LFSR Conditioned Output

Qualitatively, the results seem very similar to those coming from the testing of random data (see Section 4.2.2 for details). More formally, the Kolmogorov-Smirnov two-sample test performed at the 95% confidence level for these two distributions produces the KS test statistic $D = 0.032499$ and a p-value of approximately 0.95, which fails to reject the null hypothesis that the two datasets are drawn from the same distribution; it is important to note that with this size dataset (particularly, with the 256 sample values from the LFSR evolution dataset), the KS test is not very sensitive, and would allow a distance of up to

$$D_{\text{crit}} = 1.358\sqrt{\frac{10^6 + 256}{256 \times 10^6}} \approx 0.085$$

(or an 8.5% deviation between the CDFs) prior to rejecting the hypothesis at the 95% confidence level.

This suggests that the identified statistical LFSR issues do not significantly impact the results of this SP 800-90B assessment and support our expectation that this LFSR produces pseudorandom streams with fairly good statistical characteristics.

3.3 Heuristic Analysis of the Conditioned Output Min Entropy

If the initial state is known and a conditioned value were output after inputting only a single raw value, then all processing of that value would be bijective, there would be no entropy accumulation (so the output entropy would be the same as the input entropy), and the input could be re-constructed from the output. This is not the way that this conditioning function is used, as $64 \times$ osr raw symbols are input between each 64-bit output. As such, even though some components of this conditioning function are bijections, *the overall conditioning function is not bijective*.

To complete our understanding of this conditioning function, we need to consider the impact of the XOR of these terms.

We saw in Section 3.2 that the truncated m-sequences associated with all tested starting points (fixing the input term and iteratively applying the LFSR processing) result in pseudorandom outputs and our testing in that section is consistent with this expectation. Substantial entropy cancelation would only occur in the final sum (XOR) step when there was a very specific algebraic relationship between the inputs. This noise source is not expected to output data with any particular algebraic relationship between the outputs so substantial reduction of the entropy (beyond that due to the final XOR of many 64-bit quantities) is not expected.[8] Similarly, the prior state (which is itself an XOR of many such values) is not expected to have any particular algebraic relationship with future inputs so we do not expect substantial cancelation between the LFSR-processed initial LFSR state and LFSR-processed inputs.

The starting internal LFSR state is either identically 0 (if we are tracking the conditioning from the start) or is some already output value which cannot be credited as contributing additional entropy. As such, this term does not contribute to the min entropy estimate.

3.3.1 Assumptions

We will need some central assumptions to proceed:

A1. The min entropy of each "non-stuck" 64-bit raw symbol *added to the conditioner* (i.e., post stuck filtering) is greater than or equal to $\frac{1}{\text{osr}}$, conditioned on all prior output.

A2. The individual bits within the raw symbols are not identically distributed or independent of each other, but we do assume that bits containing the (joint conditional) min entropy can be combined using the "piling up" XOR step on a per-bit basis. In practice, most of the high-order bits are fixed to zero, and the variation guaranteed by A1 is generally only present in some of the low-order bits. The exact distribution for each bit is highly hardware- and configuration-dependent.

A3. Iteratively applying the LFSR operation spreads the influence of each bit of $\boldsymbol{x}_k$ across the 64-bit full LFSR state. Each bit of the raw symbol $\boldsymbol{x}_k$ is expected to influence approximately half of the bits in the 64-bit $\boldsymbol{A}^j\boldsymbol{B}\boldsymbol{x}_k$ expression.

[8] An active attacker who knew when specific values were shifted in could cancel out these values in perpetuity by selecting an input so that $\boldsymbol{B}\boldsymbol{x}_j$ was equal to the sum of the relevant iteratively-LFSR-scrambled summands, but that requires that an attacker can both know prior inputs to the LFSR and completely specify future inputs; this style of attacker would have already completely compromised the entropy source, so this is not an interesting attack scenario. More subtle types of manipulation by an attacker on the same device are likely to spread out the timing distribution, but without foreknowledge of which symbol the attacker is perturbing, such interference is unlikely to introduce linear structure into the raw data. This LFSR-based conditioning function provides no backwards security both because the LFSR can freely be run backwards, and because the output of this function is the entirety of the conditioning function's current state.

A4. The diffusion described in A3 causes the joint min entropy of the entire 64-bit output symbol to be equal to 64 times the per-bit min entropy. We take this assumption as a modeling assumption; A4 does not directly follow from A1 and A3 because positive bitwise correlation between symbols could cause a reduction of the min entropy. The conditional per-symbol min entropy assumption of A1 and the uniformizing assumption of A3 are consistent with this assumption, but not sufficient to justify this assumption.

A5. When the `osr` value is established using the raw data (rather than the post-stuck-filter data), this analysis requires that the underlying noise source is IID. This is essentially never strictly true for the full raw data samples, though some sources may be essentially IID for some truncations of the raw data samples; we treat the IID reasoning as a modeling assumption rather than as a claim about the source. Even if the underlying source is non-IID, this analysis may provide some useful insight even in the non-IID case (similar to how SP 800-90B approved health tests and estimators typically operate under an implicit assumption that the source is IID even for non-IID sources). Also note that for sources that are IID or essentially IID, the stuck filter induces correlation in the post-stuck filtered data stream (see Section 3.3.2 for details). This assumption is not needed when `osr` is set using an entropy analysis of the post-stuck-filtered data stream, which is the approach recommended in Section 3.3.3.

3.3.2 Raw Data vs. Filtered Data

Traditionally, the values used in JEnt entropy validation are the raw values (Δ_j), which are the difference between the current timestamp and the last round's timestamp[9]. In JEnt versions v2.2.0 – v3.6.3, only "non-stuck" raw values are provided to the conditioner[10], and this filtering step has implications for the min entropy. In JEnt, "non-stuck" values necessarily meet the following conditions:

1) Non-zero ($\Delta_j \neq 0$; the first difference of the timer is non-zero). On most hardware, the raw data distribution does not include 0, so this should not have any practical impact for hardware that supports JEnt use. Additionally, JEnt's initialization testing produces an error if it encounters any raw values that equal 0.

2) Not the same as the prior raw data sample ($\Delta_j \neq \Delta_{j-1}$; the second difference of the timer is non-zero). This precludes integrating runs of the same symbol into the conditioner.

3) The change between the raw values cannot be fixed ($\Delta_j - \Delta_{j-1} \neq \Delta_{j-1} - \Delta_{j-2}$; the third difference of the timer is non-zero). This precludes changes in the raw data by fixed amounts.

The "non-stuck" condition 1 is unlikely to have any impact for any supported hardware (it would suggest that the timer resolution is much too slow to observe changes in system behavior). The "non-stuck" conditions 2-3 have the practical impact of excluding exactly 1 or 2 values for each raw symbol (namely, $\Delta_j \neq \Delta_{j-1}$ and $\Delta_j \neq 2\Delta_{j-1} - \Delta_{j-2}$). Note, this "stuck" test precludes exactly 1 value if and only if the prior raw symbol was itself a repeat (i.e., if $\Delta_{j-1} = \Delta_{j-2}$); conditions 2-3 otherwise preclude exactly 2 distinct symbols. As a consequence of this, runs of fixed raw values trigger both condition 2 and condition 3.

[9] The source and resolution of this timestamp are hardware- and operating system-dependent.

[10] The changes integrated after the release of JEnt v3.6.3 and before the release of JEnt v3.7.0 include a change in the `jent_hash_insert()` function (analogous to part of the `jent_hash_time()` function in v3.0.0 – v3.6.3 and part of the `jent_lfsr_time()` function in v2.2.0) so that it always inserts the raw data sample into the conditioner irrespective of the "stuck" setting. This "only condition non-stuck data" feature was introduced in the v2.1.2 to v2.2.0 transition so this filtered data finding applies to JEnt versions v2.2.0 – v3.6.3.

If we know the filter probability mass of the precluded symbols (m) (i.e., probability that the stuck test discards data), then we can provide an entropy reduction bound *for IID sources*, namely the filtered dataset has an updated $p'_{\max} \leq \frac{p_{\max}}{1-m}$ and so

$$H' \geq \max(H + \log_2(1-m), 0). \tag{5}$$

For IID sources, these bounds provide a worst-case estimate that is only sharp in the instance where one of the distribution's modes survives filtering. This reduction directly applies to an IID source[11], but using such reasoning to gain insight into the non-IID source case is very common within the SP 800-90B document (e.g., the analysis of the approved health tests in [SP 800-90B §4.4] and most of the entropy estimators in [SP 800-90B §6.2]).

When testing the current raw sample Δ_j, the filter probability mass depends on Δ_{j-1} and Δ_{j-2}, whose values are stochastic. Symbols with a higher probability mass are (by definition) more likely to be output, and thus more likely to be filtered out by condition 2 of the stuck filter. We can bound the impact of such values, but such bounds are so broad that they are not generally useful. We can get a better (hardware-dependent) empirical estimate for the average value of m through analysis of a large sample of raw data produced on the hardware being used as part of the assessed entropy source. The [Theseus] `u32-jent-nonstuck` tool can be used to produce post-stuck-filter data streams for testing and to estimate this filter probability mass, m.

To help give some anecdotal sense of the anticipated range of the filter probability mass, we tested a variety of available hardware (including various Intel, AMD, and ARM hardware platforms) using raw datasets of 100 million raw samples.

Table 1. Observed Average Filter Probability Mass (m)

Architecture	JEnt Version[12]	Raw Data 90B Tool Estimate	Average m[13]	Observed False Positive Rate[14]	Filtered Data 90B Tool Estimate
ARM Cortex-A53 on a networking SoC	2.2.0	0.110118	0.44510479	$2^{-1.17}$	0.131213
ARM Cortex-A72 on a networking SoC	2.2.0	0.126359	0.26391090	$2^{-1.92}$	0.0907289
Intel Atom Denverton	2.2.0	0.614900	0.04028809	$2^{-4.63}$	0.698771
Intel Xeon Broadwell	2.2.0	5.53007	0.00249539	$2^{-8.65}$	5.52947
Intel Xeon Cascade Lake	2.2.0	0.134489	0.06620185	$2^{-3.92}$	0.229460
	3.6.3	7.50837	0.00096938	$2^{-10.01}$	7.52045
AMD EPYC Milan	2.2.0	1.17812	0.26561537	$2^{-1.91}$	1.19899
	3.6.3	4.36032	0.01453134	$2^{-6.10}$	4.38333
Apple M4 E-core	2.2.0	0.930320	0.29066697	$2^{-1.78}$	0.372956
	3.6.3	4.73669	0.01417988	$2^{-6.14}$	4.92845

[11] Even if the source is IID, this filtering process would induce a (Markov order 2) dependence with prior raw data samples so the post-filter data stream would *not* be IID. This dependence is not directly assessable using only the filtered data stream.

[12] The tested v2.2.0 version was modified for ease of validation to the SP 800-90B requirements: a corrected version of the APT health test was backported from v3.6.3; the TSC was used as the underlying timer for the Intel and AMD architectures, and other architectures used a nanosecond-resolution clock (with a less idiomatic encoding than is default); pseudorandom variation in the raw data caused by `jent_shuffle()` was removed; and an interface to access the raw data was added. The tested v3.6.3 version used a 128MB memory region. This code is available at the [Code] repository.

[13] Note that this is the average value of the filter masses; this value should be considered as a *lower* bound for the per-symbol filter mass, which is the value used in Equations (5)-(7).

[14] Note that [SP 800-90B §4.3, Footnote 8] suggests that a reduction in min entropy is possible when the false positive rate is greater than 2^{-20}.

Architecture	JEnt Version[12]	Raw Data 90B Tool Estimate	Average m[13]	Observed False Positive Rate[14]	Filtered Data 90B Tool Estimate
Apple M4 P-core	2.2.0	0.586240	0.38881142	$2^{-1.36}$	0.286127
	3.6.3	2.60697	0.06235075	$2^{-4.00}$	2.64165

This table shows that the filtering can evidently raise or lower the 90B tool's assessed min entropy. We also see that direct assessment of the filtered data stream can support the use of configurations that would be precluded by Equation (5); this is the case for the ARM Cortex-A53, ARM Cortex-A72, and Apple M4 P-core using JEnt v2.2.0. Conversely, the Apple M4 E-core filter entropy reduction is greater than would be suggested by using the average filter mass as m with an IID source. This confirms that Equation (5) does not bound the entropy reduction due to filtering when using an average value for m with a non-IID source. Together, these emphasize the notion that direct analysis of the post-filtered data stream is an important step in any such analysis.

For IID sources, the relationship expressed in Equation (5) gives us a bound for the $\mathtt{osr}$ as it relates to m in order to make a positive claim on the entropy input to the conditioner, namely when $m > 0$,

$$\mathtt{osr} < \left\lceil -\frac{1}{\log_2(1-m)} \right\rceil, \tag{6}$$

and

$$m < 1 - 2^{-\frac{1}{\mathtt{osr}}}. \tag{7}$$

When $m = 0$, this analysis does not impose an upper bound for the setting of $\mathtt{osr}$.

This provides the restrictions in Table 2.

Table 2. Maximum Filter Probability Mass (m) Supported for Raw-Data-Based osr Selections in IID Sources

osr	m Upper Bound	osr	m Upper Bound
1	0.5	11	0.0610691
2	0.292893	12	0.0561257
3	0.206299	13	0.0519225
4	0.159104	14	0.0483048
5	0.129449	15	0.0451584
6	0.109101	16	0.0423967
7	0.0942763	17	0.0399533
8	0.082996	18	0.0377762
9	0.0741253	19	0.035824
10	0.066967	20	0.0340637

The JEnt source is generally not IID, so the min entropy estimate is not based on the probability of the most likely symbol and is instead reduced based on the correlation between outputs. On certain hardware / JEnt version combinations, some of the low-order bits do appear to possess essentially IID behavior, and this observation is the basis for one of the public heuristic entropy arguments. [JEnt Heuristic]

Performing this data-dependent filtering clearly has the capacity to diminish the Most Common Value Estimate (which uses the same most-probable-symbol probability estimator that applies to an IID source), but the actual impact of this filtering is distribution- (thus hardware-) dependent and could push the assessment in either direction. As such, it is preferable to directly test the post-filtered data and establish the value of $1/\mathtt{osr}$ based on these results (which is consistent with the assumption A1). In this case, the setting $m = 0$ could be used for the bounds provided in this paper.

For example, examine an IID source with an alphabet of 256 symbols {0x100 … 0x1FF} where the most probable symbol is 0x180, whose probability is $p_{\max} = \frac{1}{2}$ (thus $H = -\log_2 \frac{1}{2} = 1$), and the rest of the symbols are equally likely ($p_{\text{other}} = \frac{1}{2\times 255}$). For this source, when the prior two raw symbols were both 0x180 the stuck filter excludes only a single symbol, and the new min entropy for the next symbol output from the filter is now $H' = -\log_2 \frac{1}{255} \approx 7.99$ bits of min entropy.

Continuing with this example, if the two prior outputs were 0x100, then the only symbol excluded by the stuck filter is 0x100, and the next symbol output will have a min entropy of $H' = -\log_2 \frac{(1/2)}{(1-1/2\times 255)} = 0.997$, which demonstrates one of the circumstances where applying the stuck filter results in an entropy reduction.

In all cases, the A1 assumption requires a per-symbol bound that must apply for all possible histories, not just the best case.

JEnt v3.7.0 and later unconditionally inject all samples (including stuck samples) into the conditioning function, so in this case $m = 0$ is the appropriate filter probability mass to use.

3.3.3 LFSR Processing of the Filtered Data Stream

As noted in Section 3.1, these LFSR conditioning steps are bijective so the min entropy of the $\boldsymbol{A}^j\boldsymbol{B}\boldsymbol{x}_k$ summand is equal to the min entropy of $\boldsymbol{x}_k$, though the distribution of the min entropy throughout the 64 bits is changed as the LFSR processing is iterated. The min entropy in $\boldsymbol{x}_k$ resides mainly in the low-order bits of the raw delta values. Due to the uniformizing assumption (A3), we model the entropy content of each bit of the $\boldsymbol{A}^j\boldsymbol{B}\boldsymbol{x}_k$ summand as containing 1/64$^{\text{th}}$ of the min entropy of $\boldsymbol{x}_k$.

The goal of this simplifying assumption is that, by focusing on the (post-uniformized) LFSR-processed per-bit min entropy, we can avoid a detailed analysis which both tracks each bit of each raw symbol throughout the LFSR processing and explicitly establishes the min entropy distribution within the raw symbols.

To support the assumption that the bits of each of the inputs are well distributed, we can examine the count of the 1s of the rows of the matrix of $\boldsymbol{A}^j\boldsymbol{B}$; this tells us how many input bits contribute to a single output bit in the expression $\boldsymbol{A}^j\boldsymbol{B}\boldsymbol{x}_k$. In Figure 5 we provide a quantile fan for the row weights of $\boldsymbol{A}^j\boldsymbol{B}$ for *j* in the range 0 to 100 (i.e., the number of input bits that contribute to a single output bit) that shows the median, quartiles, 10-90% and min/max bands. The behavior does not significantly vary for larger values of j. In Figure 6, we show two pooled histograms of these row weights, one for $0 \le j \le 2$, and one for $3 \le j \le 100$. The $3 \le j \le 100$ row weight histogram closely resembles the $\text{Binomial}(n = 64, p = 1/2)$ distribution (also shown on this graph). This suggests that for $j \ge 3$, the $\boldsymbol{A}^j\boldsymbol{B}$ have row weights similar to random bit matrices, and that the row weights of $\boldsymbol{A}^j\boldsymbol{B}$ for $0 \le j \le 2$ are substantially different than the behavior for larger values of j.

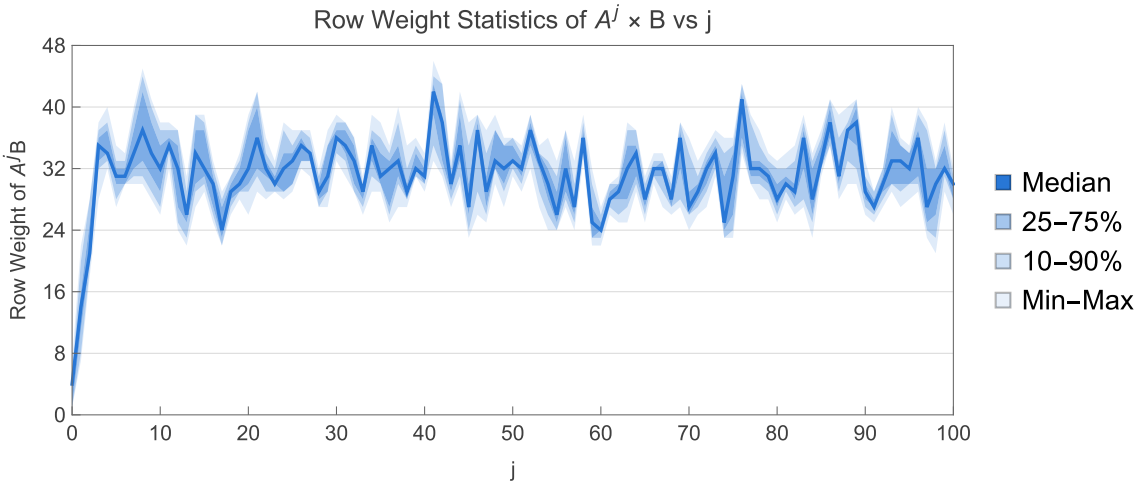

Figure 5. Row Weight Quantile Fan for A^jB with $0 \leq j \leq 100$

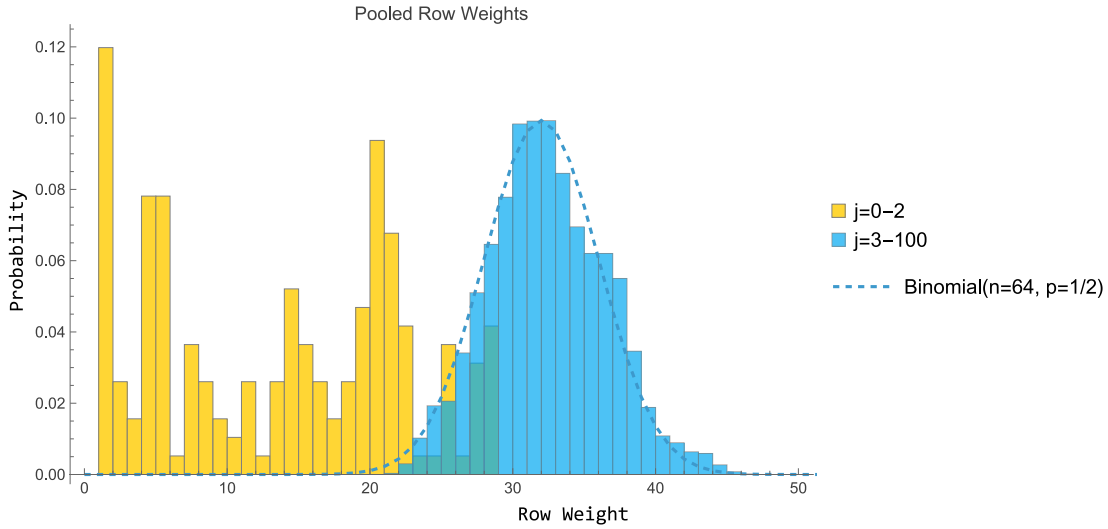

Figure 6. Histogram of Pooled Row Weights

Because the matrix $\boldsymbol{A}^j\boldsymbol{B}$ is symmetric for all non-negative integer values of j, these same graphs serve as the statistics of the count of the number of 1s in each column, so we can also use this graph to understand how many output bits are influenced by each input bit.

We see that generally the last few symbols input into the conditioning function (roughly the last three, corresponding with j=0, j=1 and j=2) impact notably fewer output bits, but the raw symbols otherwise have a fairly consistent impact on all the bits of the output. The underrepresentation of the last three raw symbols is a case where the general uniformity assumption is not quite correct, but it is important to note that these inputs still have an impact. The information for the last few inputs is not well represented across all the bits of the final LFSR state, but those transforms are still invertible, so the information is present (just not yet well distributed).

For JEnt v2.2.0, the final 64-bit sum (XOR) that integrates the entropy contribution from each of the LFSR-processed raw inputs has exactly $64 \times \mathtt{osr}$ summands, each of the form $\boldsymbol{A}^j\boldsymbol{B}\boldsymbol{x}_k$. Each bit place of the output is established only by the corresponding bit place of each of the $\boldsymbol{A}^j\boldsymbol{B}\boldsymbol{x}_k$ summands.

Within a JEnt entropy analysis $\mathtt{osr}$ has traditionally been selected to allow the claim that each symbol in the raw stream (Δ_k) has at least $\frac{1}{\mathtt{osr}}$ bits of min entropy in the raw stream. We saw above that this approach disregards the impact of the stuck filtering present in JEnt v2.2.0 – v3.6.3 which can lead to overestimating the min entropy provided to the conditioning function. For IID sources, we can further bound the min entropy loss due to stuck filtering using Equation 5, but a more reasonable approach is to base the assessment on the post-stuck-filtered data stream.

Under our simplifying assumptions A3-A4, we further presume that each bit of $\boldsymbol{A}^j\boldsymbol{B}\boldsymbol{x}_k$ contains at least $\frac{H'}{64}$ bits of independent min entropy, so the joint entropy of the entire 64-bit symbol can be calculated by multiplying the per-bit modeled min entropy by 64.

Given two independent bits with known min entropy m_1 and m_2, this corresponds to the most likely symbol probabilities $p_1 = 2^{-m_1}$ and $p_2 = 2^{-m_2}$. Because we are looking at the most likely symbol for each of these cases, it has probability $p_i \geq \frac{1}{2}$, so these probabilities can be written as $p_1 = \frac{1+\varepsilon_1}{2}$ and $p_2 = \frac{1+\varepsilon_2}{2}$ for some $0 \leq \varepsilon_i \leq 1$.

In each of the four possible cases (the most likely symbols for the two bits are both 0, both 1, or mixed 0 and 1), the probability of the most likely symbol of the XOR of these two bits is $\frac{1+\varepsilon_1\varepsilon_2}{2}$, so the resulting min entropy is a direct result of this probability. Similarly, if we XOR n bits with the same bias, then the probability of the most likely symbol is $\frac{1+\varepsilon^n}{2}$.

Under assumption A2, this statement gives us an estimate for the entropy of the LFSR output on a per-bit basis.

Putting this together, we find that the resulting conditioned min entropy assessment after inputting w symbols each with entropy H' into the LFSR under this uniform assumption is then:

$$h^{\text{uniform}}_{\text{heuristic}}(H', w) = 64\left(1 - \log_2\left(\left(2^{1-H'/64} - 1\right)^w + 1\right)\right). \tag{8}$$

So long as we have selected $\mathtt{osr}$ based on the assessed entropy of the post-stuck-filter data stream, the behavior of JEnt v2.2.0 is thus described by $h^{\text{uniform}}_{\text{heuristic}}\left(\frac{1}{\mathtt{osr}}, 64 \times \mathtt{osr}\right)$. If we instead consider a more conservative approach where we do not account for the last three underrepresented symbols input into the conditioning function, then we get $h^{\text{uniform}}_{\text{heuristic}}\left(\frac{1}{\mathtt{osr}}, 64 \times \mathtt{osr} - 3\right)$. Setting w to values less than 64 never occurs in practice, but one can note that $h^{\text{uniform}}_{\text{heuristic}}(1, 1) = 1$, which is not that surprising as the conditioning function is bijective when 64 bits or fewer are input.

This JEnt v2.2.0 behavior is graphically depicted in the relevant range in Figure 7.

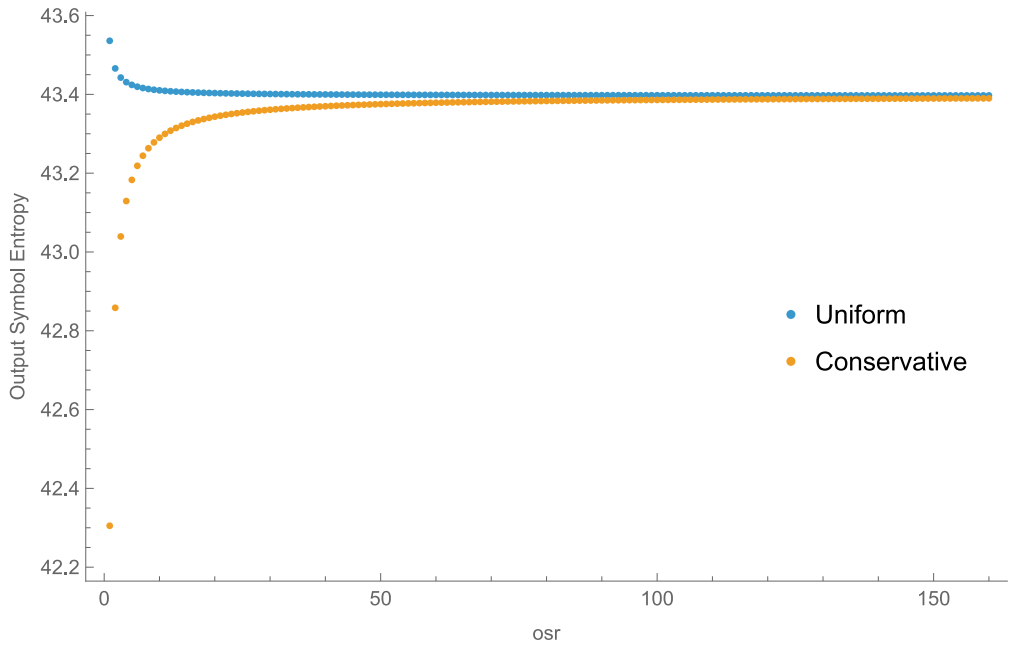

Figure 7. LFSR Output Min Entropy vs. osr

In limit, we see that

$$\lim_{\mathtt{osr}\to\infty} h^{\text{uniform}}_{\text{heuristic}}(1/\mathtt{osr}, 64\times\mathtt{osr}) = \lim_{\mathtt{osr}\to\infty} h^{\text{uniform}}_{\text{heuristic}}(1/\mathtt{osr}, 64\times\mathtt{osr}-3) = 64\log_2\left(\frac{8}{5}\right) \approx 43.3966.$$

The osr=1 case of the conservative accounting approach provides the worst case in this analysis (42.3), which shows that for the JEnt v2.2.0 behavior, we can expect $h_{\text{heuristic}} \geq 42.3$ bits of min entropy per 64-bit conditioned output block for any osr setting so long as this osr setting supports a claim of $h_{\text{submitter}} = \frac{1}{\mathtt{osr}}$ bits of min entropy based on the post-stuck-filtered data stream[15]. If we instead fix osr to some positive integer value and take the limit in w, we then find that

$$\lim_{w\to\infty} h^{\text{uniform}}_{\text{heuristic}}(1/\mathtt{osr}, w) = 64,$$

So, given enough data, $h_{\text{heuristic}}$ can be made as close to 64 as desired.

Now that we have identified the worst-case setting for osr, we can expand our understanding of the impact of the stuck filter probability mass on an IID system.

[15] The result here is a truncation of the calculated value of 42.3052, where truncation is used to provide a conservative estimate.

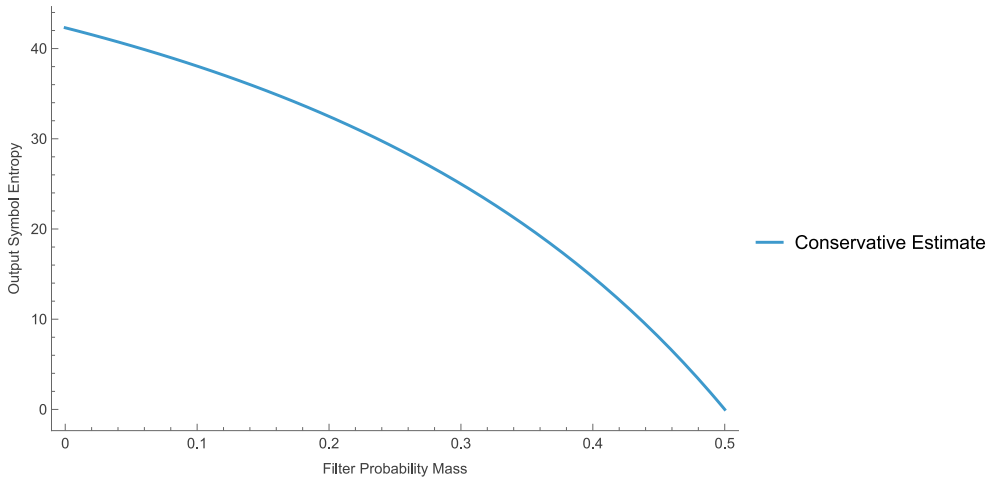

Figure 8. IID JEnt LFSR Output Min Entropy Bound vs. Filter Probability Mass (`osr` $= 1$)

With certain architectures (e.g. some embedded ARM architectures) it is common to both need an `osr` setting much larger than 1 and to have a concentrated output symbol distribution. Condition 2 of the stuck filter excludes $\Delta_j = \Delta_{j-1}$, and the probability of this occurrence is generally much higher for concentrated distributions, thus such distributions are more likely to produce high *m* values. This behavior can effectively preclude crediting a non-zero per-symbol min entropy when the `osr` is set based on testing of the raw data stream. In such cases, validation can proceed by performing the assessment using post-stuck-filtered datasets so that `osr` can be set based on these results (and then using $m = 0$).

4 SP 800-90B Non-Vetted Conditioning Analysis

The entropy at each stage of the conditioning chain must be assessed. All conditioning performed within the entropy source is non-vetted conditioning, so the analysis at each stage is broadly similar.

An assessment approach for non-vetted conditioning components is outlined in [SP 800-90B §3.1.5.2]; this is slightly updated by [IG D.K, Resolution 5], which further requires that the claimed h_{out} cannot be greater than the entropy output level justified with mathematical evidence, which we call $h_{\text{heuristic}}$. Together, we have:

$$h_{\text{out}} = \min(\text{Output\_Entropy}(n_{\text{in}}, n_{\text{out}}, nw, h_{\text{in}}), 0.999 \times n_{\text{out}}, h' \times n_{\text{out}}, h_{\text{heuristic}}). \quad (9)$$

The Output_Entropy(·) function described in [SP 800-90B §3.1.5.1.2] is a method for modeling entropy accumulation within a random map. For non-vetted conditioning (like the LFSR used here), this entropy may be further reduced based on the result of a bitstring-oriented statistical test of the output of the conditioning function (h', See [SP 800-90B §3.1.5.2] for details).

As a practical matter, the $0.999 \times n_{\text{out}}$ is never the minimum of these expressions, as we necessarily have $h' < 0.999$ for any reasonable size of conditioned sequential dataset.[16]

4.1 Stuck Filter Conditioning

In SP 800-90B terms, health testing is conducted using the raw data stream, so the JEnt raw data stream is the SP 800-90B "raw data". Consequently, the processing performed within the stuck filter must be classified as the first non-vetted conditioning function in the conditioning chain.

The amount of data consumed by this conditioning function per output is not fixed (it is data-dependent), so some of the conditioning parameters for this step can only be conservatively bounded.

The raw data has significant structure and much of this structure survives the stuck filter, so this conditioning function is not well modeled using the Output_Entropy(·) function (which only provides useful insight into the action of random maps). This paper discusses the use of the Output_Entropy(·) function here only because its use is required within the SP 800-90B validation process.

The SP 800-90B Output_Entropy(·) parameters for the stuck filtering are $n = n_{\text{out}} = nw = 64$. As the number of symbols input per filter output (k) increases, $n_{\text{in}} = 64 \times k$ and $h_{\text{in}} \geq H \times k$ increase and $\text{Output\_Entropy}(64 \times k, 64, 64, H \times k)$ increases. We can then bound

$$\text{Output\_Entropy}(64 \times k, 64, 64, H \times k) \geq \text{Output\_Entropy}(64, 64, 64, H) \approx H.$$

The bitwise statistical estimation of the post-filtered data, h', will need to be calculated using a post-stuck-filter dataset (such a dataset for analysis can be generated using the [Theseus] `u32-jent-nonstuck` tool to filter a raw dataset). There is no reason to expect that this h' assessment will look anything like the assessments described in Section 4.2.2; we expect it to be substantially lower. The h_{out}value for this conditioning stage is then bounded above by $h' \times 64$.

It is important to note that an additional complete entropy assessment is necessary at this stage to estimate an $h_{\text{heuristic}} \leq H'$ for the output from the stuck filter conditioning.

Finally, we note that at this conditioning stage $h_{\text{out}} \leq H$ because h_{out} is bounded above by $\text{Output\_Entropy}(64, 64, 64, H) \approx H$. Most of our analysis has attempted to bound the possible entropy *reduction* due to the stuck filtering, but this same analysis shows that in some circumstances H' may increase due to this filtering, and it is interesting to note that within the SP 800-90B conditioning analysis any such increase cannot be

[16] The largest h' assessment encountered in the testing described in Section 4.2.2 (on pseudorandom data) was $h' = 0.951516$, which is well below 0.999.

credited. To satisfy assumption A1, we need to select osr so that $H' \geq 1/\mathtt{osr}$; due to the SP 800-90B conditioning analysis requirements, we have $h_{\text{out}} \leq h_{\text{heuristic}} \leq H'$, so when we select osr so that $h_{\text{out}} \geq 1/\mathtt{osr}$, we automatically satisfy assumption A1.

4.2 LFSR Conditioning

The input to the LFSR conditioning is the output from the stuck filter conditioning, so we proceed forward using the above bounded values.

4.2.1 The Output_Entropy(·) Function

The LFSR has significant algebraic structure between the inputs and outputs, so LFSRs are not well modeled using this function (which only makes sense for random maps). This paper discusses the use of the Output_Entropy(·) function only because its use is required within the SP 800-90B validation process.

The parameters used within the SP 800-90B Output_Entropy(·) function are summarized in Table 3.

Table 3. SP 800-90B Output_Entropy(·) Parameter Summary

Parameter	Value
Symbol Width (n)	64
n_{out}	64
nw ($\leq n_{\text{in}}$)	64
$n_{\text{in}}(w \times n)$	$w \times 64$
H	$1/\mathtt{osr}$
$h_{\text{in}} = w \times H$	$\geq w/\mathtt{osr}$

In the unmodified JEnt v2.2.0, $w = 64 \times \mathtt{osr}$ so $h_{\text{in}} = 64 \times \mathtt{osr} \times \frac{1}{\mathtt{osr}} = 64$. We will later vary this w parameter.

The only parameter that may require explanation is the narrowest width, *nw*. The internal state is 64 bits at every step and the state update map is a bijection on that state, so no narrowing occurs anywhere. This can be made more explicit when viewed in terms of the matrix representation of the LFSR action: the internal state is a 64-bit vector, and all input data is integrated into this result, so *nw* is clearly not larger than 64 bits.

In order to show that the narrowest width is not less than 64, we note that if we select bit ℓ in the jth state

$$\boldsymbol{o}_j(\boldsymbol{s}_1, \boldsymbol{x}_1, \boldsymbol{x}_2, \ldots \boldsymbol{x}_j) = \boldsymbol{A}^j \boldsymbol{s}_1 + \sum_{k=1}^{j} \boldsymbol{A}^{j-k} \boldsymbol{B} \boldsymbol{x}_k$$

we can determine exactly which bits in each of the j inputs (as k runs 1 to j) influence this bit by looking at the ℓth row of the matrix used to transform the kth input:

$$\boldsymbol{A}^{j-k} \boldsymbol{B}.$$

The matrices $\boldsymbol{A}$ and $\boldsymbol{B}$ are invertible, so $\boldsymbol{A}^{j-k}\boldsymbol{B}$ is also invertible. An invertible matrix cannot have a row with all 0s or a column with all 0s (as this would cause the determinant to be 0) so both this ℓth row and ℓth column cannot be identically 0 for any selection of k or j. This internal state is the output value, so every bit of this state is influenced by at least one input bit, and similarly every input bit influences at least one output bit. As such, the narrowest width cannot be less than 64, and it cannot be greater than 64, so it is exactly equal to 64.

The numerical results of the Output_Entropy(·) function are clearly dependent on the values w (the number of symbols fed into the conditioner between outputs) and osr, but the general relationships are consistent across these parameter values. Figure 9 shows the relationship between the LFSR entropy accumulation described in this analysis and the Output_Entropy(·) function output (both for osr=1) as the number of symbols input (w) varies.

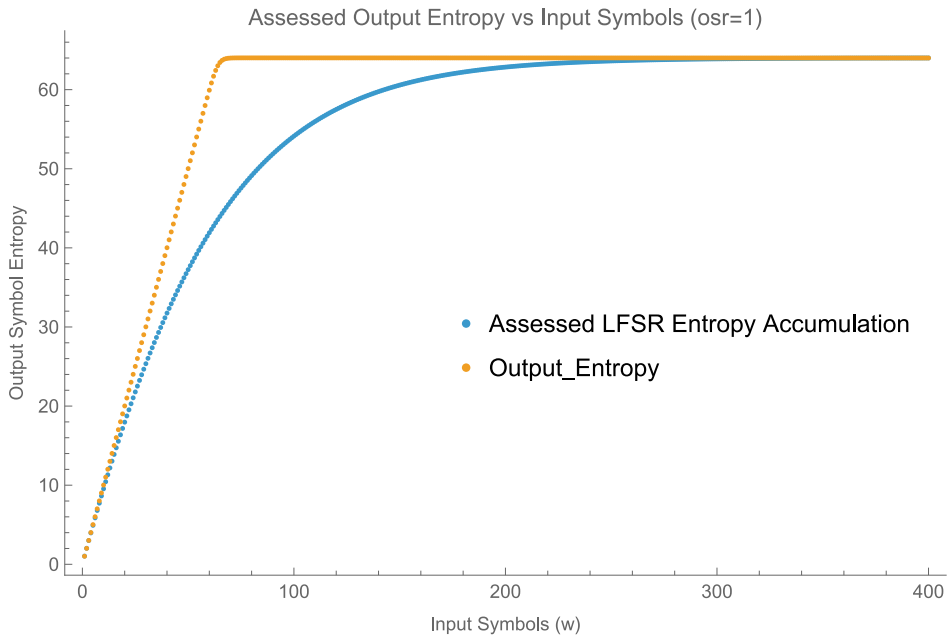

Figure 9. Output Min Entropy vs. Raw Symbols Input ($\mathbf{osr} = \mathbf{1}, \boldsymbol{m} = \mathbf{0}$)

It is clear from this graph that under this analysis the LFSR accumulates entropy more slowly than modeled by the Output_Entropy(·) function.[17] This continues to be true for larger values of $\mathtt{osr}$. As such, the minimum value that establishes h_{out} is not the Output_Entropy(·) result.

4.2.2 The Distribution of h' For Random Data

As seen in Section 3.2, this LFSR has reasonable statistical properties (even absent entropy being fed into it). The output of the XOR of many such pseudorandom strings will itself be pseudorandom so the LFSR-conditioned output of JEnt is expected to be indistinguishable from random data using the SP 800-90B test suite. Similarly, there is a broad class of non-vetted conditioning components that also produce pseudorandom outputs irrespective of the quality of the data supplied to them. Most non-vetted conditioning components evaluated within the ESV process are of this class.

4.2.2.1 Empirical Estimation of the h' Distribution

As part of the non-vetted conditioning procedure described in [SP 800-90B §3.1.5.2], the tester is required to statistically evaluate a conditioned sequential dataset output as a bitstring. The NIST ESV testing infrastructure presently only allows samples of 10^6 bytes to be submitted for testing.[18] For the conditioned output test, this means that the sample is treated as a single bitstring of length 8×10^6 bits.

When the NIST entropy assessment tool performs a bitstring assessment on a sample of 8×10^6 bits of pseudorandom data, the statistical testing results are expected to conform to a specific min entropy estimate distribution. This min entropy estimate distribution conveys an effective "best case" for this step of the SP 800-90B assessment procedure, as conditioning components that produce detectably flawed (i.e., evidently non-pseudorandom) output will generally assess lower than those in this min entropy estimate distribution.

[17] A more detailed analysis where the min entropy distribution within the raw input symbol is established and the contribution of each input bit is separately credited may yield a higher $h_{\text{heuristic}}$.

[18] Testing larger pseudorandom data samples would tend to narrow the assessment distribution and shift this entire distribution higher. In all cases, this testing is fundamentally stochastic in nature, so there is some element of chance in this portion of the assessment.

To get a better sense of how the SP 800-90B bitstring min entropy estimator results are distributed for such datasets we generated 10^6 independent (and non-overlapping) datasets, each with 10^6 random bytes (8×10^6 bits) and tested each using the SP 800-90B bitstring tests, just as is done with samples of non-vetted conditioned output within the ESV process. Random generation and min entropy assessment was done within the [Theseus] package.[19]

The random number generator used to produce the reference datasets is a combination of two architecturally distinct PRNGs, both with excellent statistical properties; each random data sample is the XOR of the outputs of the xoshiro256** PRNG [BV 2021] (seeded using /dev/urandom) and the RdRand instruction (rapidly reseeded CTR_DRBG with no derivation function using one Generate call per 128-bit output, seeded using a built-in hardware metastable latch entropy source). Combining these two generators is intended to remove any linear structure from the output of xoshiro256**.

The result distribution for these random data SP 800-90B bitstring tests is presented in Figure 10.[20]

This figure also shows which of the SP 800-90B min entropy estimators produced the reported value; the reported value is the lowest estimator result. This shows us that the compression estimate is generally (89% of the time) the estimator that provides the lowest min entropy estimate in such pseudorandom bitstring assessments. Note that the key in Figure 10 is sorted by the number of times that estimator's result was used as the final assessment (from most to least common).

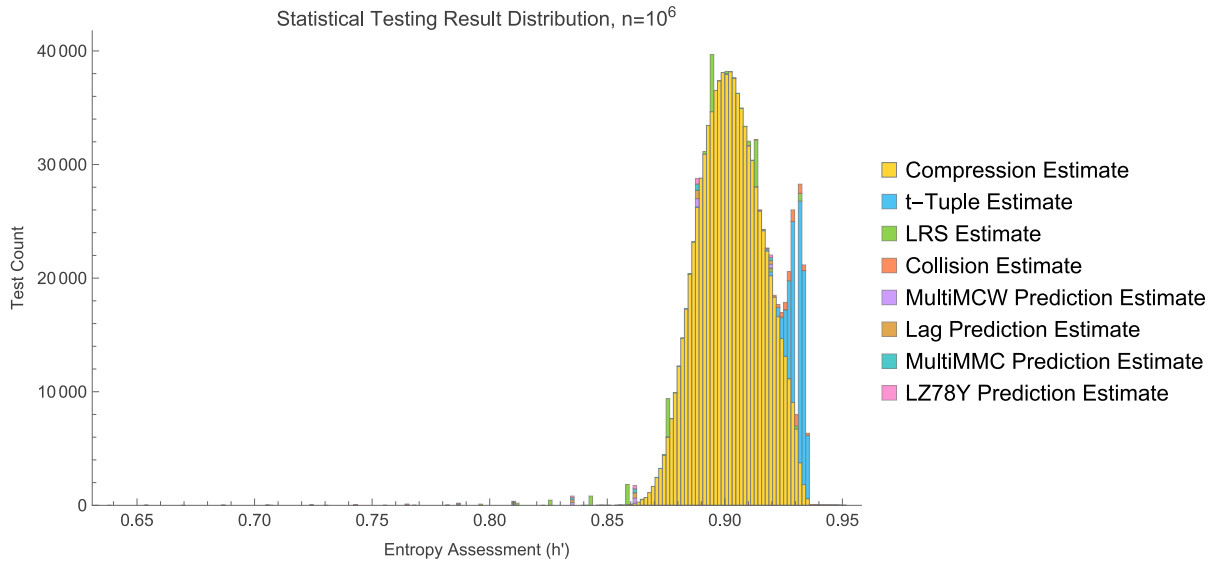

Figure 10. Histogram of the h' Assessments of 10^6-Byte Pseudorandom Samples

We summarize the high-level statistical properties of this distribution in Table 4.

Table 4. Estimate Statistics

Statistic	h'
Minimum	0.638278
1st Quartile	0.893352
Median	0.903675
3rd Quartile	0.915660
Maximum	0.951516

[19] To verify consistency, we compared the full set of estimator results for the [Theseus] and [NIST Tool] for 10000 pseudorandom datasets, each containing 1 million bytes. The maximum observed estimator absolute difference in the result was less than 9.7511×10^{-13}.

[20] The raw results of this testing are available on [hPrimeResults].

Statistic	h'
Mean	0.904343
Standard Deviation	0.016253
Distinct Estimate Count	891105

4.2.2.2 The t-Tuple Estimator Assessment Spikes

This distribution depicted in Figure 10 appears to have several estimate spikes, but the histogram does not convey how narrow these spikes are. In particular, the t-Tuple estimator was the source of 79726 of the final min entropy estimates, but these estimates are all one of only 43 distinct values.

Adopting the notation from the specification of the t-Tuple Estimator [SP 800-90B §6.3.5], in the t-Tuple estimator we first find the longest substring that repeats at least 35 times; the length of this substring is t. $Q[i]$ is the number of occurrences of the most common i-length substring in the data, $P[i]$ is the observed probability of this i-length substring, and $P_{\max}[i]$ is a normalized per-symbol probability associated with this most common i-length substring. This estimator calculates $P_{\max}[i]$ for all i from 1 to t, and calls the maximum observed value $\hat{p}_{\max}$. This $\hat{p}_{\max}$ value is then used as the center of a confidence interval whose upper bound is used to estimate the min entropy.

The observed repeated entropy estimate seems to occur because of the construction of this estimator. In pseudorandom strings, the counts of such "longest repeated substrings of length i" is somewhat stable, and by estimator construction the number of times the longest repeated substring is observed ($Q[t]$) is likely to be close to 35. In the cases where the t-Tuple produces the lowest min entropy estimate, the substring length producing the maximum $P_{\max}[i]$ is quite likely to be the longest observed substring of length t; all of the observed t-Tuple assessment spikes reflect this occurrence.

As such, this estimator tends to take a limited set of possible discrete parameters and thus tends to produce a small number of possible estimates.

Borrowing from measure theory, such repeated estimates form larger "atoms" (that is, values that cannot be subdivided by distinct quantiles because they are equal) which can make any statistical result based on the rank of data values near such atoms harder to interpret. Table 5 presents the eight most common t-Tuple estimates that were the source of overall assessments.

Table 5. Observed t-Tuple High Probability Mass Atoms

Entropy Estimate (Data Sample Atom)	Observed Mass (%)	t	i where $\hat{p}_{\max} = P_{\max[i]}$	$Q[i]$
0.93570634778060169	0.5561	19	19	35
0.93356924430377808	1.8837	19	19	36
0.93149069917309046	2.3024	19	19	37
0.92946758870669888	1.5923	19	19	38
0.92749703268343042	0.8625	19	19	39
0.92557636968160262	0.4103	19	19	40
0.92370313546342575	0.1840	19	19	41
0.92187504396449971	0.0794	19	19	42

Here, we see that these eight specific estimates (associated with the $t = i = 19$, and Q[i] values between 35 and 42) comprise almost 8% of the entire h' dataset and 98.7% of the h' estimates arising from the t-Tuple estimator.

4.2.2.3 Identifying Unusual Test Results

We can determine a sequence of nested two-sided bounds that are expected to contain the result for this testing with some specific probability. For each stated α, we construct a two-sided interval by calculating BCa 99% confidence intervals for the $\frac{1-\alpha}{2}$ and $\frac{1+\alpha}{2}$ percentile values, and take the lower confidence interval bound for the $\frac{1-\alpha}{2}$ confidence interval and the upper confidence interval bound for the $\frac{1+\alpha}{2}$ confidence interval.

Table 6. Expected Result Bounds

Probability the Result is in the Bound (α)	**h' BCa 99% Confidence Interval**	**$h' \times n_{\text{out}}$ BCa 99% Confidence Interval**
95	[0.875830, 0.933569]	[56.0532, 59.7484]
99	[0.858875, 0.935706]	[54.9680, 59.8852]
99.9	[0.786601, 0.935706]	[50.3425, 59.8852]

Each upper bound in this table coincides with one of the high-mass t-Tuple atoms presented in Table 5 (those associated with the $Q[i] = 36$ and $Q[i] = 35$ values) so the empirical quantile function is flat there. Bootstrap confidence intervals for a quantile require a positive density at that quantile [BF 1981], and BCa bootstraps additionally assume the existence of a monotone normalizing transformation, essentially a further smoothness requirement. [Efron 1987]. Neither of these conditions holds at an atom so the confidence interval does not have the intended significance. This is a property of the t-Tuple estimator, not an artifact of this dataset.

Results for h' above the 0.935706 atom are uncommon for this assessment approach; only 262 such assessments occurred across 10^6 rounds of testing (0.0262%), so such results should occur less than once every 3500 validations. If such values are observed in testing (particularly if this occurs multiple times) it may be reasonable to investigate why the conditioned dataset selection procedure in use produces such unlikely results.

4.2.2.4 Post LFSR Results

In Section 3.2, we presented some results showing that the individual summand terms making up the LFSR conditioning function behaved in an evidently pseudorandom way. In Section 4.2.2, we presented the h' distribution for pseudorandom data. Here, we describe the results of the post-XOR combination of these summands (i.e., the output of the full conditioning function) to show that the results are consistent with the h' distribution presented in Section 4.2.2. These results should not be taken to suggest that the noted value is in any sense a meaningful depiction of the entropy present in the string, rather we are showing that the post-conditioned bitstrings appear to be pseudorandom, as expected.

We created JEnt v2.2.0 instances (all using $\mathtt{osr} = 1$) on the Cascade Lake, AMD EPYC, Apple M4 P-core, and Apple M4 E-core architectures and extracted 100 1-million-byte outputs from each of the JEnt instances. For the Cascade Lake platform, we also produced a similar $\mathrm{osr} = 5$ dataset for comparison, to demonstrate that increasing the osr does not substantially alter the LFSR output assessment distribution.

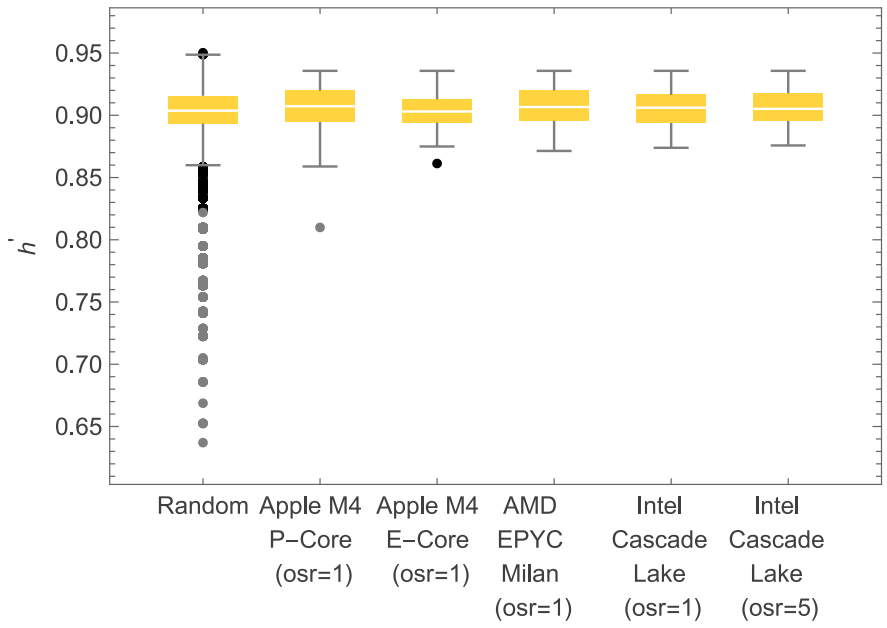

Figure 11. Random Data Assessments ($n = 10^6$) Compared to LFSR Conditioned Data Assessments (Each $n = 100$)[21]

We then pool the resulting 500 results and compare them to the reference bounds provided in Table 6.

Table 7. Observed LFSR h' Results as Compared to Expected Result Bounds

Probability the Result is in the Bound (α)	h' BCa 99% Confidence Interval	Expected Outside ($n = 500$)	Observed Outside	Binomial p-Value
95	[0.875830, 0.933569]	25	18	0.086
99	[0.858875, 0.935706]	5	1	0.039
99.9	[0.786601, 0.935706]	0.5	0	0.606

These results are reasonably close to the expected rate of exclusion for each band, as shown using binomial p-values for $n = 500$ and $p = 0.05$, $p = 0.01$, $p = 0.001$ (respectively). This suggests that the assessed values are, if anything, more central to the h' distribution, as the post-LFSR assessment results produce fewer extreme values than the reference distribution predicts. Figure 11 clearly indicates that the reference distribution has a longer lower tail than is present in any of the evaluated platforms.

As an additional confirmation, the Kolmogorov-Smirnov two-sample test performed at the 95% confidence level for the pseudorandom distribution depicted in Figure 10 and the pooled data for all the platforms produces the KS test statistic $D = 0.0563490$ and a p-value of approximately 0.084, which fails to reject the null hypothesis that the two datasets are drawn from the same distribution at the 95% confidence level. Note that the observed statistic is 93% of the critical value, so this is a marginal result rather than a strong demonstration of agreement, and it is important to note that with this size dataset (particularly, with the 500 sample values from the LFSR output analysis), the KS test is not very sensitive, and would allow a distance of up to

$$D_{\text{crit}} = 1.358\sqrt{\frac{10^6 + 500}{500 \times 10^6}} \approx 0.061$$

(or a 6.1% deviation between the CDFs) prior to rejecting the hypothesis at the 95% confidence level.

[21] For each of the platforms, the highest observed h' assessment was 0.93570634778060169, the $Q[i] = 35$ t-Tuple atom described in Section 4.2.2.2.

Further, comparing the Cascade Lake ($\texttt{osr} = 1$) and the Cascade Lake ($\texttt{osr} = 5$) distributions using the Kolmogorov-Smirnov two-sample test performed at the 95% confidence level produces the KS test statistic $D = 0.09$ and a p-value of approximately 0.813, which fails to reject the null hypothesis that the two datasets are drawn from the same distribution at the 95% confidence level. With the size of these datasets, the KS test is not very sensitive, and would allow a distance of up to

$$D_{\text{crit}} = 1.358\sqrt{\frac{100+100}{100\times 100}} \approx 0.19205$$

(or a 19.2% deviation between the CDFs) prior to rejecting the hypothesis at the 95% confidence level. Between this KS two-sample test and the fact that the medians of these two distributions are close to each other (0.906121 for the $\texttt{osr} = 1$ case and 0.905170 for the $\texttt{osr} = 5$ case), we observe that these distributions are closely related to each other.

4.2.3 Alternate Selections for the *w* Parameter

JEnt v2.2.0 normally outputs a block of conditioned output after $w = 64 \times osr$ raw symbols have been sent to the conditioning component, resulting in the $h_{\text{heuristic}}$ value provided in Section 3.3. If this behavior were to be altered, how should w be set?

Section 3.3 shows that $h_{\text{heuristic}}$ can get asymptotically close to the maximal claim of 64 bits of min entropy per output by sending more data to the conditioner between outputs. Theoretically, sending more raw data into the conditioner always results in a higher $h_{\text{heuristic}}$ value, which is in some sense "better". When does it make sense to stop?

Two distinct (and conflicting) goals both seem reasonable. The first is making the largest possible claim for h_{out}, and the second is getting as much entropy as possible from the entropy source in some fixed amount of time.

4.2.3.1 Largest Possible $\boldsymbol{h}_{out}$ Assessment Goal

If we want to make the largest possible claim for h_{out}, then we need to control the chance that the $h_{\text{heuristic}}$ estimate is less than $h' \times n_{\text{out}}$ and thus reduces h_{out}. We have seen that for this conditioner the value that ultimately establishes h_{out} is either $h' \times n_{\text{out}}$ or $h_{\text{heuristic}}$. The LFSR output is expected to have excellent statistical properties given any reasonable amount of input, so the value we find for h' is essentially stochastic in nature and expected to be distributed as seen in Figure 10. We can use this expected result distribution to better understand how large the $h_{\text{heuristic}}$ claim should be to avoid the case where $h_{\text{heuristic}}$ reduces the final h_{out} assessment.

If we wanted to accumulate entropy until there was at least a 50% chance that the $h_{\text{heuristic}}$ value was higher than the statistical test result, then we would accumulate entropy to at least the median of the $h' \times n_{\text{out}}$ min entropy estimates. Similarly, if we wanted there to be a high (99%) chance that the $h_{\text{heuristic}}$ value was higher than the statistical test result, then we would accumulate entropy to at least the 99$^{\text{th}}$ percentile of the $h' \times n_{\text{out}}$ min entropy estimates. Finally, if we wanted there to be an essentially 100% chance that the $h_{\text{heuristic}}$ value was higher than the statistical test result, then we would accumulate entropy to at least the maximum observed $h' \times n_{\text{out}}$ min entropy estimate.

We can determine how many symbols should be input into the LFSR conditioner so that $h_{\text{heuristic}}$ is at least these reference entropy levels under this assessment approach. This information is summarized in Figure 12.

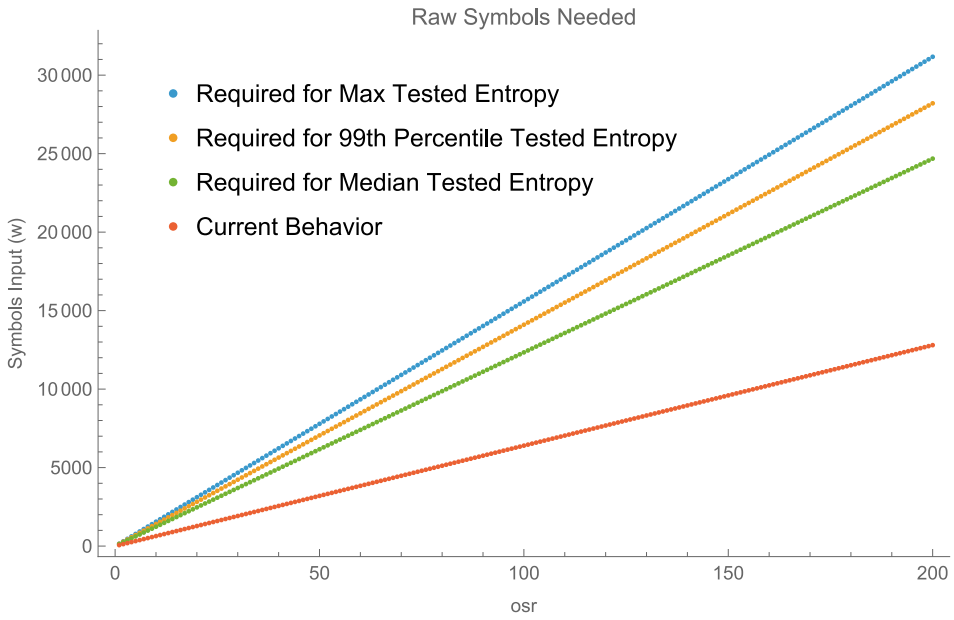

Figure 12. Number of Raw Symbols Required to Prevent Likely Assessment Reduction

The depicted relationship is essentially linear, so we can simply scale w by the scaling factor ($\mathcal{S}$) as needed to meet our goal. These scaling factors are presented in Table 8.

4.2.3.2 Largest Output Entropy Per Input Symbol Goal

Each raw symbol takes approximately the same time to produce, so if we are interested in getting as much entropy as possible from the system in some fixed time, then we need to maximize the impact of the min entropy present in each raw symbol on the conditioned output assessment. To gauge the impact of each additional symbol on $h_{\text{heuristic}}$, we examine the marginal increase in $h_{\text{heuristic}}$ per additional raw symbol input. More explicitly, the marginal increase in $h_{\text{heuristic}}$ for the jth symbol is

$$\delta_j = h_{\text{heuristic}}^{\text{uniform}}(H', j) - h_{\text{heuristic}}^{\text{uniform}}(H', j-1),$$

that is the change in $h_{\text{heuristic}}$ after inputting the jth symbol. A graph showing the marginal increase in $h_{\text{heuristic}}$ for $\texttt{osr} = 1$ is depicted in Figure 13; higher values of `osr` yield less steep declines, but the general trend is the same for larger settings of `osr`.

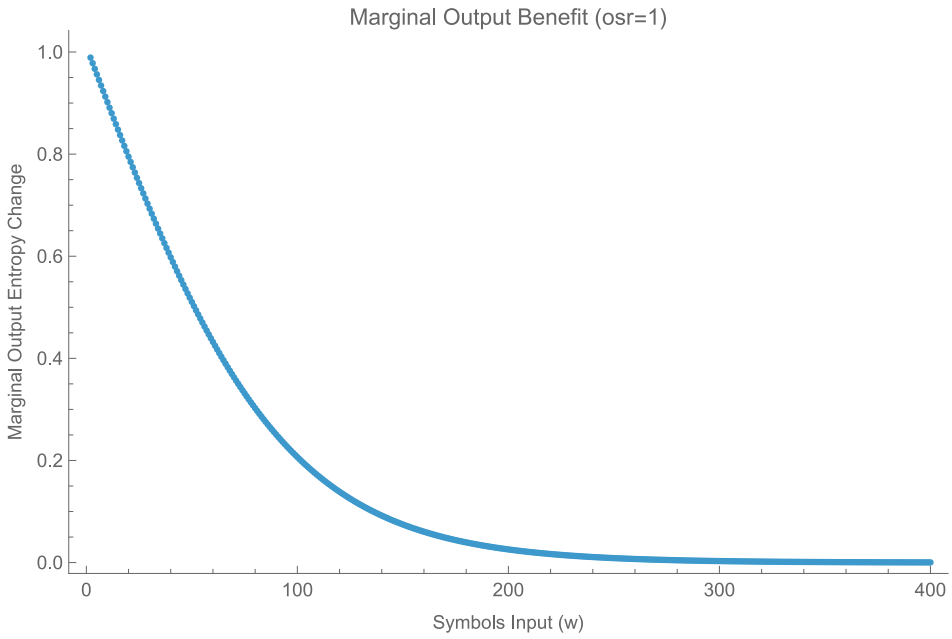

Figure 13. Marginal $h_{\text{heuristic}}$ Increase Per Raw Symbol Input (osr=1)

This shows that within this heuristic we get the largest increase of min entropy output for early symbols, and the benefit of adding marginal inputs decreases as more symbols are added. Because each raw symbol takes approximately the same amount of time to produce, this suggests that the most entropy output per unit time is achieved when *w* is kept as small as possible.

*4.2.3.3 Selection of **w** Parameter Result Summary*

To characterize overall efficiency, we are interested in getting some notion of how much benefit we get by increasing w. A reasonable raw efficiency (expressed as *average* assessed conditioned min entropy per raw symbol input) is

$$E = \frac{\bar{h}_{\text{out}}}{w} = \frac{\bar{h}_{\text{out}}}{\lceil \mathcal{S} \times 64 \rceil \times \texttt{osr}}.$$

This has a dependency on the selection of `osr`, so we instead normalize the efficiency with respect to the original behavior, namely

$$E_{\text{original}} = \frac{42.3052}{64 \times \texttt{osr}}.$$

The normalized efficiencies are then

$$\begin{aligned} E_{\text{normalized}} &= \frac{E}{E_{\text{original}}} \\ &= \frac{h_{\text{out}}}{\lceil \mathcal{S} \times 64 \rceil \times \texttt{osr}} \times \frac{64 \times \texttt{osr}}{42.3052} \\ &= \frac{\bar{h}_{\text{out}}}{42.3052} \times \frac{64}{\lceil \mathcal{S} \times 64 \rceil}. \end{aligned} \tag{10}$$

This tells us how efficient each of these options is as compared to the efficiency of the original implementation. Smaller normalized efficiency values denote less efficiency. The modeled selections for w are summarized in Table 8.

Table 8. Strategies for the Selection of w

Approach	Chance of $h_{\text{heuristic}}$ Reducing h_{out}	Scaling Factor (S)	$w = \lceil S \times 64 \rceil \times osr$	Mean h_{out}	Normalized Efficiency ($E_{\text{normalized}}$)
Original Behavior	$\approx 100\%$	1	$w = 64 \times \texttt{osr}$	42.3052	1
$h_{\text{heuristic}}$ is likely above $h' \times n_{\text{out}}$	$\leq 50\%$	1.96875	$w = 126 \times \texttt{osr}$	57.4610	0.689904
$h_{\text{heuristic}}$ is very likely above $h' \times n_{\text{out}}$	$\leq 1\%$	2.25000	$w = 144 \times \texttt{osr}$	57.8775	0.608041
$h_{\text{heuristic}}$ is always above $h' \times n_{\text{out}}$	$\approx 0\%$	2.48438	$w = 159 \times \texttt{osr}$	57.8780	0.550683

The "Mean h_{out}" value in this table is the observed mean (across all observed h' values noted in Section 4.2.2) of the function used to establish input entropy,

$$h_{\text{out}} = \min\left(\text{Output\_Entropy}(n_{\text{in}}, n_{\text{out}}, nw, h_{\text{in}}), 0.999 \times n_{\text{out}}, h' \times n_{\text{out}}, h_{\text{heuristic}}^{\text{uniform}}(1, w-3)\right).$$

As we have seen, here this is the same as $\min\left(h' \times n_{\text{out}}, h_{\text{heuristic}}^{\text{uniform}}(1, w-3)\right)$. The value $\texttt{osr} = 1$ was chosen for this mean h_{out} column because this represents the lowest heuristic estimate. The value h' is stochastic in nature and is expected to vary between validations. (See Figure 10 for the expected distribution of h'.)

This analysis shows that inputting more than $159 \times \texttt{osr}$ raw symbols between LFSR outputs is not expected to provide any SP 800-90B assessment benefit. We also see that the original behavior is the most efficient of the analyzed selections for w.

5 Conclusion

We presented a theoretical basis for modeling JEnt v2.2.0's LFSR non-vetted conditioning component. This model is demonstrated to be equivalent to the LFSR-based conditioning component present in JEnt v2.2.0. Various characteristics of this LFSR conditioner were verified using this model. We then used this model to develop a heuristic analysis of the conditioned output min entropy, finding a defensible estimate for an entropy claim of 42.3 bits of min entropy per conditioned 64-bit output block (under the assumption that `osr` is set reasonably using post-filtered data).

We analyzed the distribution of the SP 800-90B bitstring assessments of sample sets of 8×10^6 bits which applies to any non-vetted conditioning function that produces output that cannot be distinguished from pseudorandom output using the SP 800-90B tests.

We identified an entropy reduction required for IID sources when setting the osr value based on the raw data stream. In such sources, this reduction may preclude using a raw-data-stream-only assessment approach for some architectures where a limited set of high-probability symbols are present. In most cases it is still possible to move forward by using post-filtered data streams as a basis for the selection of `osr`. We also found that the use of average filter mass may understate the degree of entropy loss, particularly for non-IID sources.

We also described a change to JEnt v2.2.0 that would allow for higher conditioned output block min entropy claims, though this increased claim comes at the cost of decreasing the entropy output from the entropy source per unit time.

This analysis provides a sound framework for modeling the entropy claims required for a SP 800-90B conditioning assessment and is a significant step forward in providing a solid justification for the level of entropy output by JEnt v2.2.0.

5.1 Future Work

A more detailed analysis could establish an extremal min entropy distribution within the raw symbol and separately credit the contribution of each input bit, rather than relying on the uniformizing assumption A3.

An alternate approach would replace the heuristic of Section 3.3 with a rigorous bound. Because $(\boldsymbol{A}^{w-k}\boldsymbol{B})^{\mathrm{T}} = \boldsymbol{A}^{w-k}\boldsymbol{B}$ for all k (Section 2.2) and $\boldsymbol{A} = \boldsymbol{A}_{\text{single}}^{64}$ advances the LFSR state by exactly one 64-bit block, each term of the LFSR sum forms a contiguous $64 \times w$ -bit window of the m-sequence generated by A_{single}. This observation allows the application of [Lacharme 2008] so long as the per-bit input bias can be bounded.

Some additional work would involve characterizing the filter probability mass m introduced in Section 3.3.2. The bound of Equation 5 is a worst case derived under an IID assumption, and the observation that the filter preferentially removes the most probable symbol suggests that the ultimate effect may be smaller and may in some cases increase the min entropy. Ideally there could be some way of producing m as a function of the raw distribution. Additionally, some progress toward giving a rigorous treatment for non-IID sources that accounts for the Markov dependence the filter induces would provide a framework for better understanding the meaning of this parameter.

6 References

[BST 2003] Boaz Barak, Ronen Shaltiel, and Eran Tromer. *True Random Number Generators Secure in a Changing Environment*. CHES 2003. https://doi.org/10.1007/978-3-540-45238-6_14.

[BF 1981] Peter J. Bickel and David A. Freedman. *Some Asymptotic Theory for the Bootstrap*. The Annals of Statistics. 1981, Vol. 9, No. 6, pp. 1196–1217. https://doi.org/10.1214/aos/1176345637.

[BV 2021] David Blackman and Sebastiano Vigna. *Scrambled Linear Pseudorandom Number Generators*. ACM Transactions on Mathematical Software, Volume 47, Issue 4, Article No.36, pp. 1–32. December 2021. https://doi.org/10.1145/3460772.

[BL 2008] Marco Bucci and Raimondo Luzzi. *Fully Digital Random Bit Generators for Cryptographic Applications*. IEEE Transactions on Circuits and Systems I: Regular Papers, Vol. 55, No. 3, pp. 861–875, April 2008. https://doi.org/10.1109/TCSI.2008.916446.

[JEnt Heuristic] CMUF Entropy Working Group. *Jitter Entropy Library v3.4.0–v3.6.2 Entropy Source Opportunistically Essentially-IID Heuristic Assessment Procedure*. March 31, 2025.

[IG] CMVP. *Implementation Guidance for FIPS 140-3 and the Cryptographic Module Validation Program*. April 16, 2026. https://csrc.nist.gov/csrc/media/Projects/cryptographic-module-validation-program/documents/fips%20140-3/FIPS%20140-3%20IG.pdf.

[Dichtl 2007] Markus Dichtl. *Bad and Good Ways of Post-processing Biased Physical Random Numbers*. FSE 2007. https://doi.org/10.1007/978-3-540-74619-5_9.

[Efron 1987] Bradley Efron. *Better Bootstrap Confidence Intervals*. Journal of the American Statistical Association. Vol. 82, No. 397, pp 171–185. https://doi.org/10.2307/2289144.

[Golomb 2017] Solomon W. Golomb. *Shift Register Sequences*. World Scientific, 2017.

[HJ 2019] Joshua E. Hill and Ben Jackson. NIST Special Publication 800-90B Comments. V1.9, 20191213. https://www.untruth.org/~josh/sp80090b/UL%20SP800-90B-final%20comments%20v1.9%2020191212.pdf.

[Hill 2023] Joshua E Hill. *What to Expect When You're Expecting: (Part II: … to Assess the CPU Jitter Random Number Generator v2.2.0 Against SP 800-90B)*. Presentation Version 20230711. CMUF Entropy Working Group. https://www.untruth.org/~josh/sp80090b/JEnt%20v2.2.0%20Validation%20Notes%2020230711-2.pdf.

[Theseus] Joshua E. Hill. *Theseus*. Version 20260626. https://github.com/KeyPair-Consulting/Theseus/archive/refs/tags/20260626.tar.gz.

[Code] Joshua E. Hill. Code used within the paper *JEnt v2.2.0 LFSR Conditioning Analysis*. Tagged 20260817-2. https://github.com/joshuaehill/JEntLFSRModel/releases/tag/20260817-2.

[hPrimeResults] Joshua E. Hill. h' Results. 20260802. https://untruth.org/~josh/sp80090b/jent-lfsr/hprime-results.tar.xz.

[Johnston 2017] David Johnston. STS-2.1.2 and SP800-90B Assessment Suite Anomalous Results. 20170523. https://github.com/dj-on-github/90B_check.

[KMS 2015] John Kelsey, Kerry McKay, Meltem Sönmez Turan. Predictive Models for Min-entropy Estimation. CHES 2015. https://doi.org/10.1007/978-3-662-48324-4_19.

[Knuth 1998] Donald E. Knuth. *The Art of Computer Programming*. Vol. 2, 3rd ed. Addison-Wesley, 1998. pp. 29–32.

[Lacharme 2008] Patrick Lacharme. *Post-Processing Functions for a Biased Physical Random Number Generator*. FSE 2008. https://www.iacr.org/workshops/fse2008/docs/papers/day_2_sess_4/21_lacharmeFSE.pdf.

[Lacharme 2009] Patrick Lacharme. *Analysis and Construction of Correctors*. IEEE Transactions on Information Theory, Volume 55, Issue 10, pp. 4742–4748. https://doi.org/10.1109/TIT.2009.2027483.

[JEnt v2.2.0] Stephan Müller. *Jitterentropy Library v2.2.0*. September 2019. https://www.chronox.de/jent/releases/historic/jitterentropy-library-2.2.0.tar.xz.

[JEnt Design] Stephan Müller. *CPU Time Jitter Based Non-Physical True Random Number Generator*. October 24, 2022. https://www.chronox.de/jent/CPU-Jitter-NPTRNG-v2.2.0.pdf.

[NIST Tool] NIST. SP 800-90B Entropy Assessment Tool, v1.1.8. https://github.com/usnistgov/SP800-90B_EntropyAssessment.

[NIST SHALL] NIST CMVP. *90B-Shall-Statements*. August 9, 2021. https://csrc.nist.gov/CSRC/media/Projects/cryptographic-module-validation-program/documents/esv/90B%20Shall%20Statements.xlsx.

[PS 2024] Matthias Peter and Werner Schindler (BSI). *A Proposal for: Functionality Classes for Random Number Generators*. V3.0. September 10, 2024. (AIS 20/AIS 31.)

[PTVF 2011] William H. Press, Saul A. Teukolsky, William T. Vetterling, and Brian P. Flannery. *Numerical Recipes: The Art of Scientific Computing*. Third Edition. Cambridge University Press, 2011. pp. 380–386.

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

[SMS 2007] B. Sunar, W. J. Martin and D. R. Stinson. *A Provably Secure True Random Number Generator with Built-In Tolerance to Active Attacks*. IEEE Transactions on Computers, Vol. 56, No. 1, pp. 109–119, Jan. 2007. https://doi.org/10.1109/TC.2007.250627.

[Živković 1994] Miodrag Živković. *A Table of Primitive Binary Polynomials*. Mathematics of Computation. Vol. 62, No. 205 (1994), pp. 385–386. https://poincare.matf.bg.ac.rs/~ezivkovm/publications/primpol1.pdf.