# JEnt v2.2.0 LFSR Conditioning

*Joshua E. Hill, PhD (KeyPair Consulting)*
*and*
*Yvonne Cliff, PhD (Teron Labs)*

*CMUF EWG*
*20251209*

# Meta Summary

- This is a presentation of “JEnt v2.2.0 LFSR Conditioning Analysis” Technical Review Draft 9.
- This paper is joint work with Yvonne Cliff (Teron Labs).
- This paper is still in development, but the parts presented today seem stable.
    - We are still working toward an analysis approach using the Leftover Hash Lemma

# *Dramatis Personae*

- JEnt v2.2.0 (released September 2019) has been widely integrated into many entropy sources.
  - e.g., #E8, #E19, #E20, #E37, #E47, #E48, #E50, #E54, #E59, #E60, #E61, #E62, #E90, #E99, #E117, #E151, #E174, #E175, #E226, #E235.
  - Many Linux kernel versions include a JEnt version that is based on JEnt v2.2.0.
- JEnt v2.2.0 uses an LFSR for conditioning.
- There is no public analysis of the LFSR conditioning to support ESV testing.
  - Testing requires mathematical evidence for various properties (see [SP 800-90B §3.2.3, Requirement 5] and [IG D.K, Resolution 5]; note that [NIST SHALL ID #52] is marked “Optional”, but [NIST SHALL IDs #106-#107] are marked as “Required”.)

TERON LABS

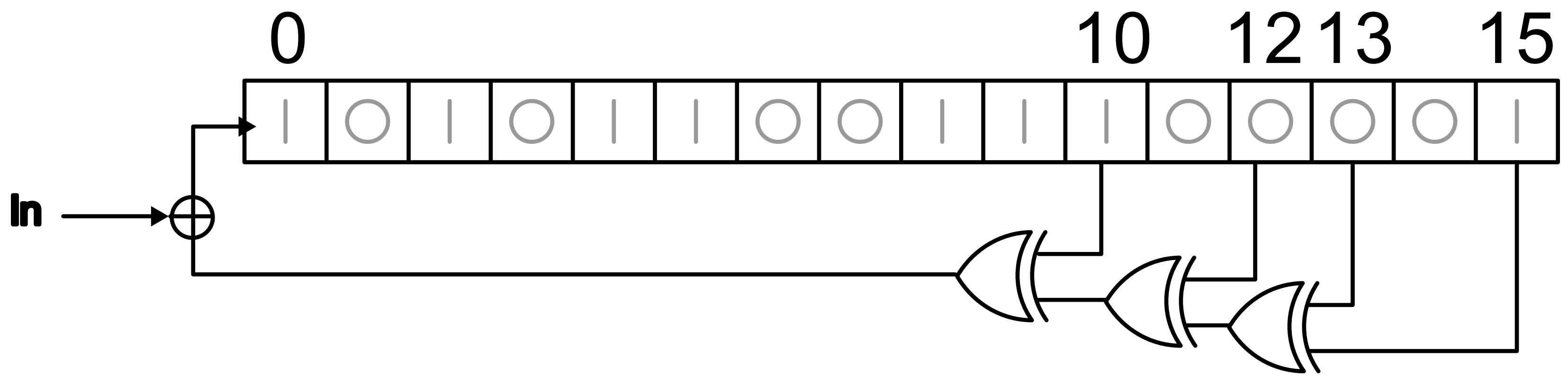
# In Summary, We…

- Present a linear model for this conditioning (Part 1)
    - Use this model to verify various characteristics of the conditioning.
    - Verify that this model is equivalent to the implemented conditioning.
- Present a heuristic analysis using this model obtain a conditioned output min entropy estimate. (Part 2)
- Provide the distribution of statistical entropy assessment results for most non-vetted conditioning functions. (Part 3.1)
- Describe changes to JEnt v2.2.0 that would allow for higher conditioned output block min entropy claims. (Part 3.2)

TERON LABS

# Part 0: LFSRs

- An LFSR is a shift register with linear (i.e., XOR-based) feedback.
- The JEnt v2.2.0 conditioning function uses a Fibonacci LFSR, e.g. something like:




Adapted from
[KCAuXy4p on Wikipedia]

# Part 1: The JEnt v2.2.0 LFSR

- The LFSR is 64 bits wide and is used in multiplicative scrambler mode (i.e., takes an external input that is XORed into bit 0 of the LFSR)
- A single bit is fed in at a time until all 64 bits of the raw symbol are integrated into the LFSR.
- After sufficient ($64 \times$ osr) raw symbols are thus integrated, the entire internal state is output as conditioned data.
- The "taps" used in this LFSR are based on a primitive polynomial:
$$p(z) = z^{64} + z^{61} + z^{56} + z^{31} + z^{28} + z^{23} + 1$$
- The LFSR has "good" structure, and loops through either 1 state (for the state value 0) or $2^{64} - 1$ states (for all other states).

TERON LABS

# Part 1: The JEnt v2.2.0 LFSR

LFSRs are **linear** so the impact of the initial state and the input values can be broken apart by additivity:

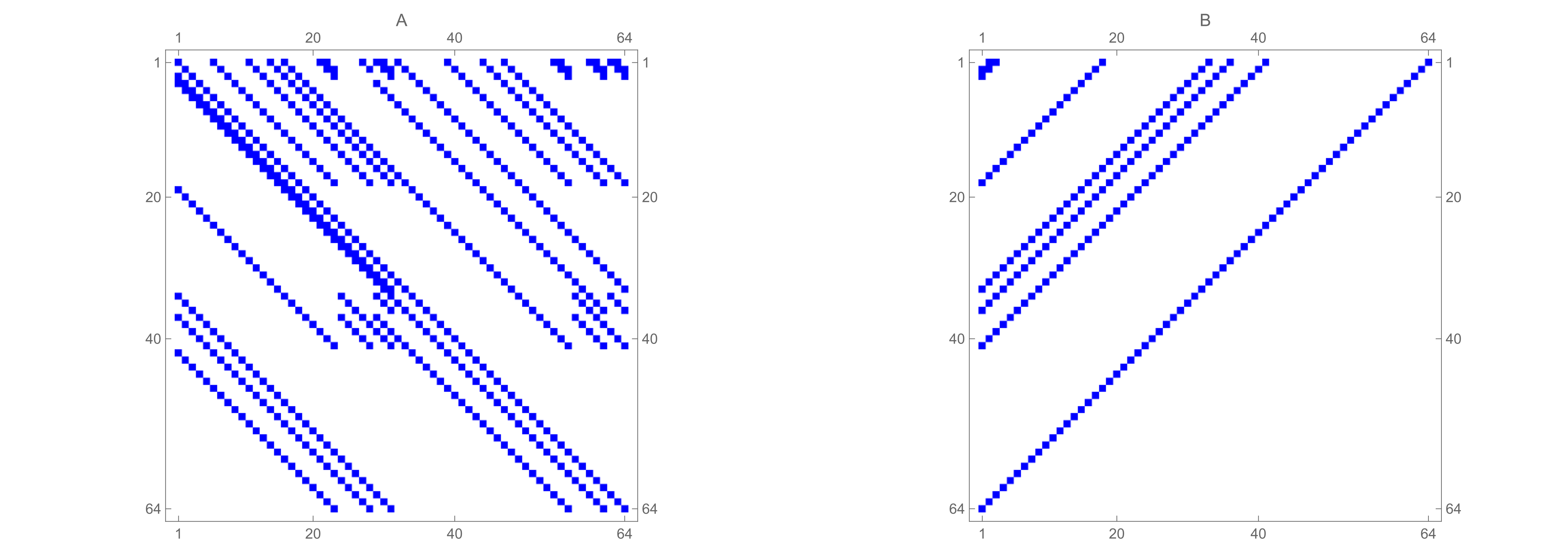
$$\boldsymbol{f(s,x) = f(s,0) + f(0,x)}$$

and each of these transforms can be represented using matrix multiplication:

$$\boldsymbol{f(s,x) = As + Bx}.$$

Further, we can find the explicit values for the matrices $\boldsymbol{A}$ and $\boldsymbol{B}$.

# Part 1: The JEnt v2.2.0 LFSR



# Part 1: The JEnt v2.2.0 LFSR

- So now we can implement JEnt v2.2.0 conditioning… really slowly?
- We can directly verify some LFSR characteristics (e.g., the order of the LFSR)
- We can also… <handwave>reason about the conditioning</handwave>!

TERON LABS

# Part 1: The JEnt v2.2.0 LFSR

We can make a recurrence relation:

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

Thus…

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

TERON LABS

# Part 1: The JEnt v2.2.0 LFSR

- This makes it clear what our matrices are accomplishing.
  - $\boldsymbol{A}$ performs 64 LFSR operations on the current internal state.
  - $\boldsymbol{B}$ deals with the loading of input data.
- We also see that every input is iteratively LFSR processed by a fixed function:

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

- Both the matrices $\boldsymbol{A}$ and $\boldsymbol{B}$ are invertible (+ some linear algebra) so $\boldsymbol{g}_j(\boldsymbol{x})$ is a bijection.

# Part 1: Record Scratch

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

is a bijection, but repeated XOR is **NOT bijective**, so

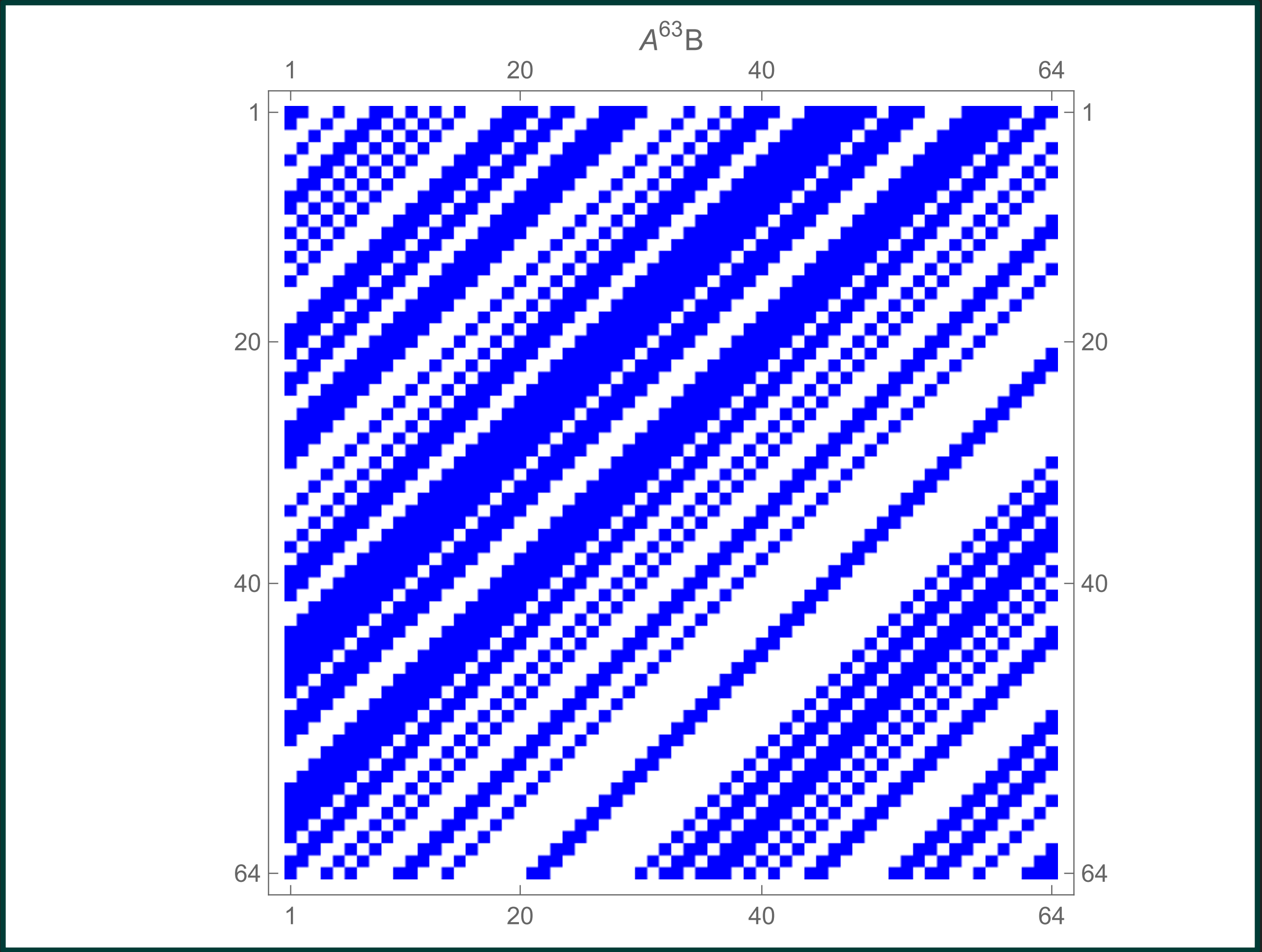
$$\boldsymbol{o}_j(\boldsymbol{s}_1, \boldsymbol{x}_1, \boldsymbol{x}_2, \ldots \boldsymbol{x}_j) = \boldsymbol{A}^j \boldsymbol{s}_1 + \sum_{k=1}^{j} \boldsymbol{g}_{j-k}(\boldsymbol{x}_k)$$

is not bijective.

(This is good, as otherwise we could not accumulate entropy!)

# Part 1: The JEnt v2.2.0 LFSR

- We use matrices of the form $A^j B$ so we are interested in their form, e.g.



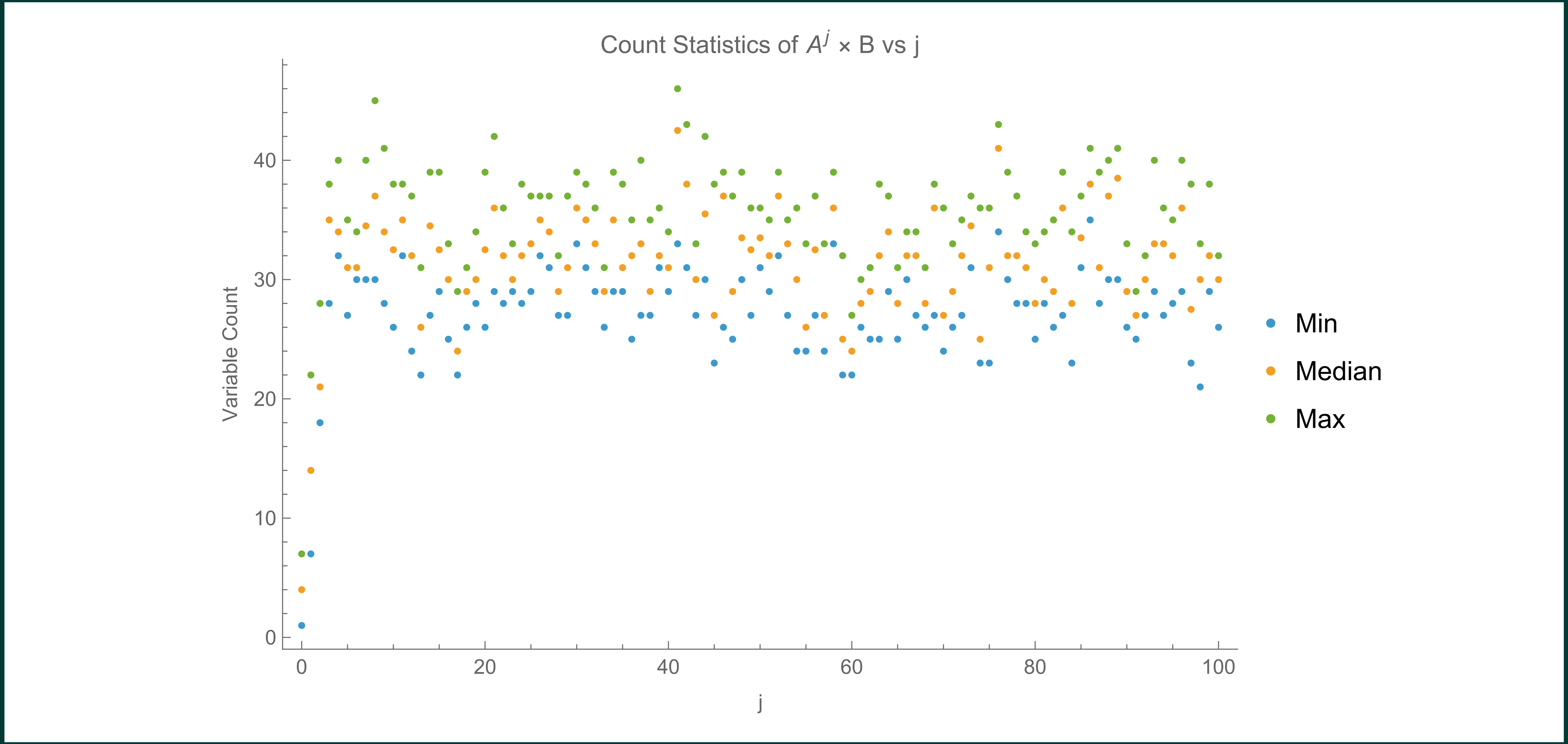
# Part 1: The JEnt v2.2.0 LFSR

- For all the values we care about (to cover conditioning up to osr=200) (for this basis convention) these are Hankel matrices and symmetric.
- These properties are likely true more broadly…

# Part 2: Heuristic Conditioning Min Entropy Analysis

- The function $g_j(x)$ is a bijection, so the entropy for each term in the sum is fixed.
- Iterative application of additional LFSR processing spreads out the impact of each input bit.

# Part 2: Heuristic Conditioning Min Entropy Analysis




TERON LABS

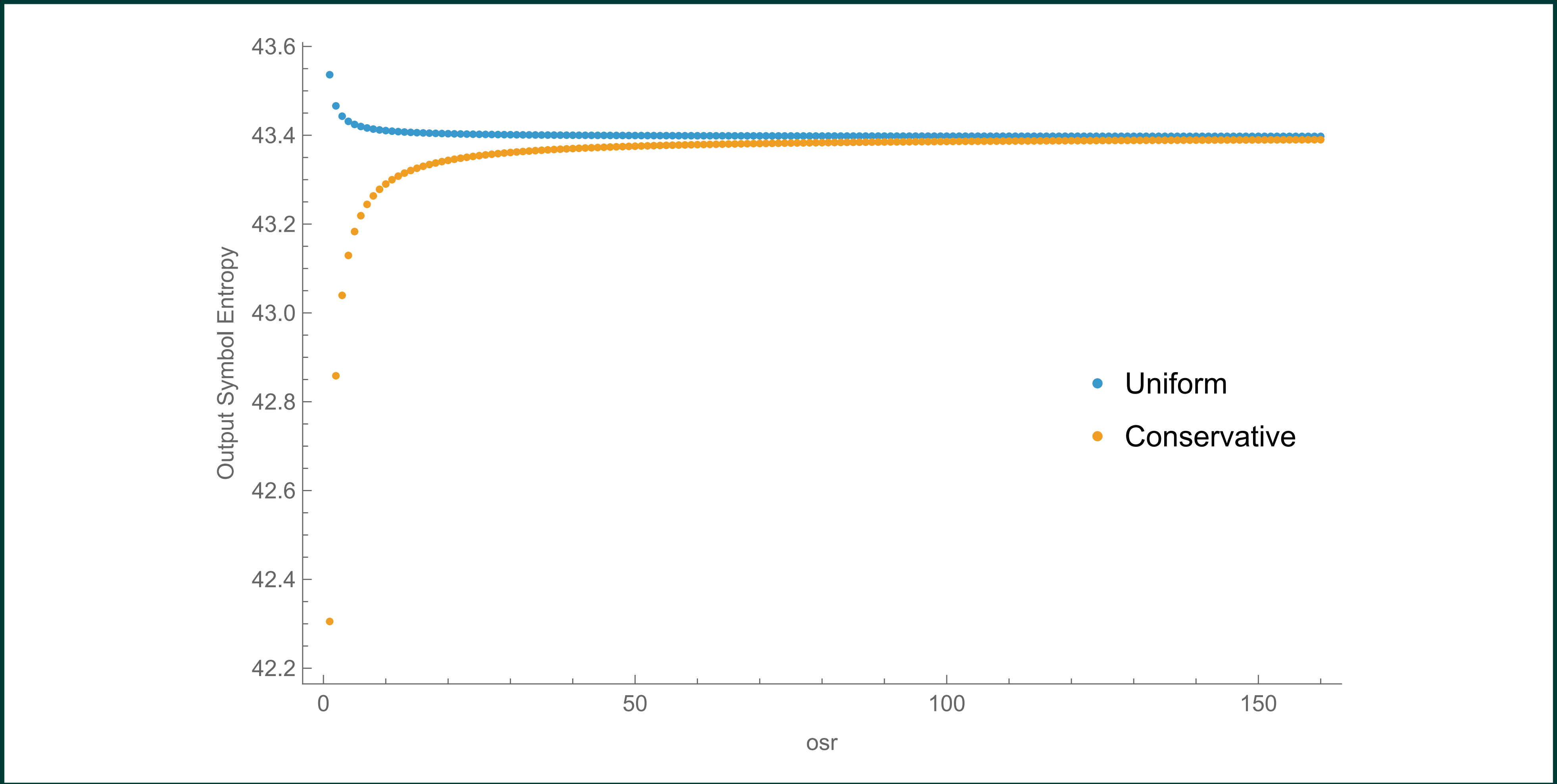
# Part 2: Heuristic Conditioning Min Entropy Analysis

- By hypothesis, the min entropy for each raw symbol is $\geq \frac{1}{osr}$.
- After a few iterations ($\geq 3$), the entropy has been spread throughout the state.
- As such, we can model iteratively XORing $w$ together bits, each with at least $\frac{1}{64 \times \mathrm{osr}}$ bits of min entropy, and then scale to the full 64 bits.

$$h_{\text{heuristic}}^{\text{uniform}}(\mathrm{osr}, w) = 64\left(1 - \log_2\left(\left(2^{1-1/(64\times\mathrm{osr})} - 1\right)^w + 1\right)\right)$$

TERON LABS

# Part 2: Heuristic Conditioning Min Entropy Analysis




TERON LABS

# Part 2: Heuristic Conditioning Min Entropy Analysis

The `osr`=1 case (discarding the entropy of the last three symbols) is the worst case:

$$h_{\text{heuristic}} \geq 42.3052$$

per 64-bit conditioned output block (assuming $h_{\text{submitter}} = \frac{1}{\texttt{osr}}$)

# Part 3: SP 800-90B Non-Vetted Conditioning Analysis

- Combining [SP 800-90B §3.1.5.2] and [IG D.K, Resolution 5], we get a formula like:

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

- The smallest of these terms dictates $h_{\text{out}}$.

# Part 3: SP 800-90B Non-Vetted Conditioning Analysis

- Some of these are not likely to be the minimum here.
  - $0.999 \times n_{\text{out}}$ is essentially never the minimum of these expressions, as we necessarily have $h' < 0.999$ for any reasonable size of conditioned sequential dataset.

TERON LABS

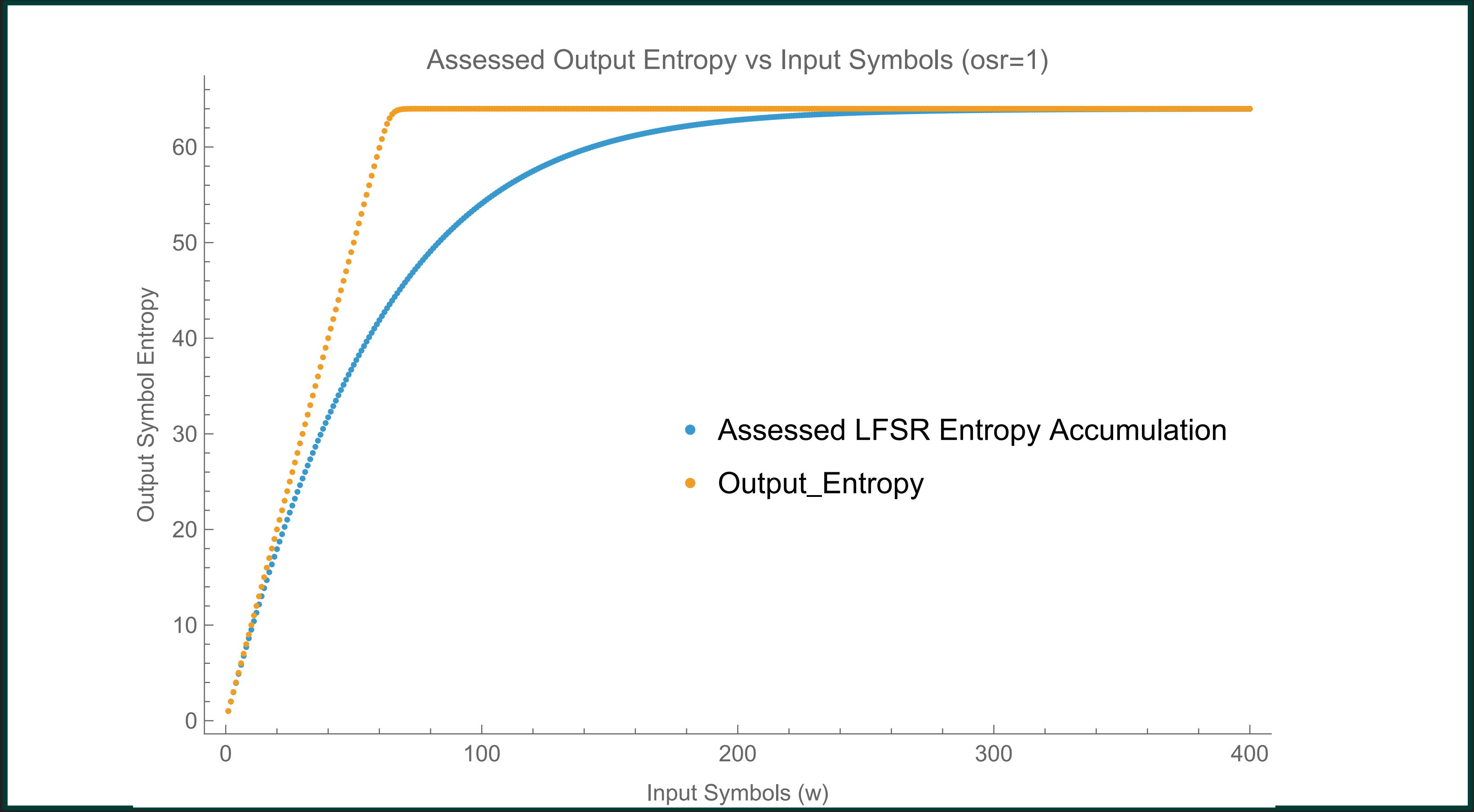
# Part 3: SP 800-90B Non-Vetted Conditioning Analysis

- Output_Entropy(·) Parameters:

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

# Part 3: SP 800-90B Non-Vetted Conditioning Analysis

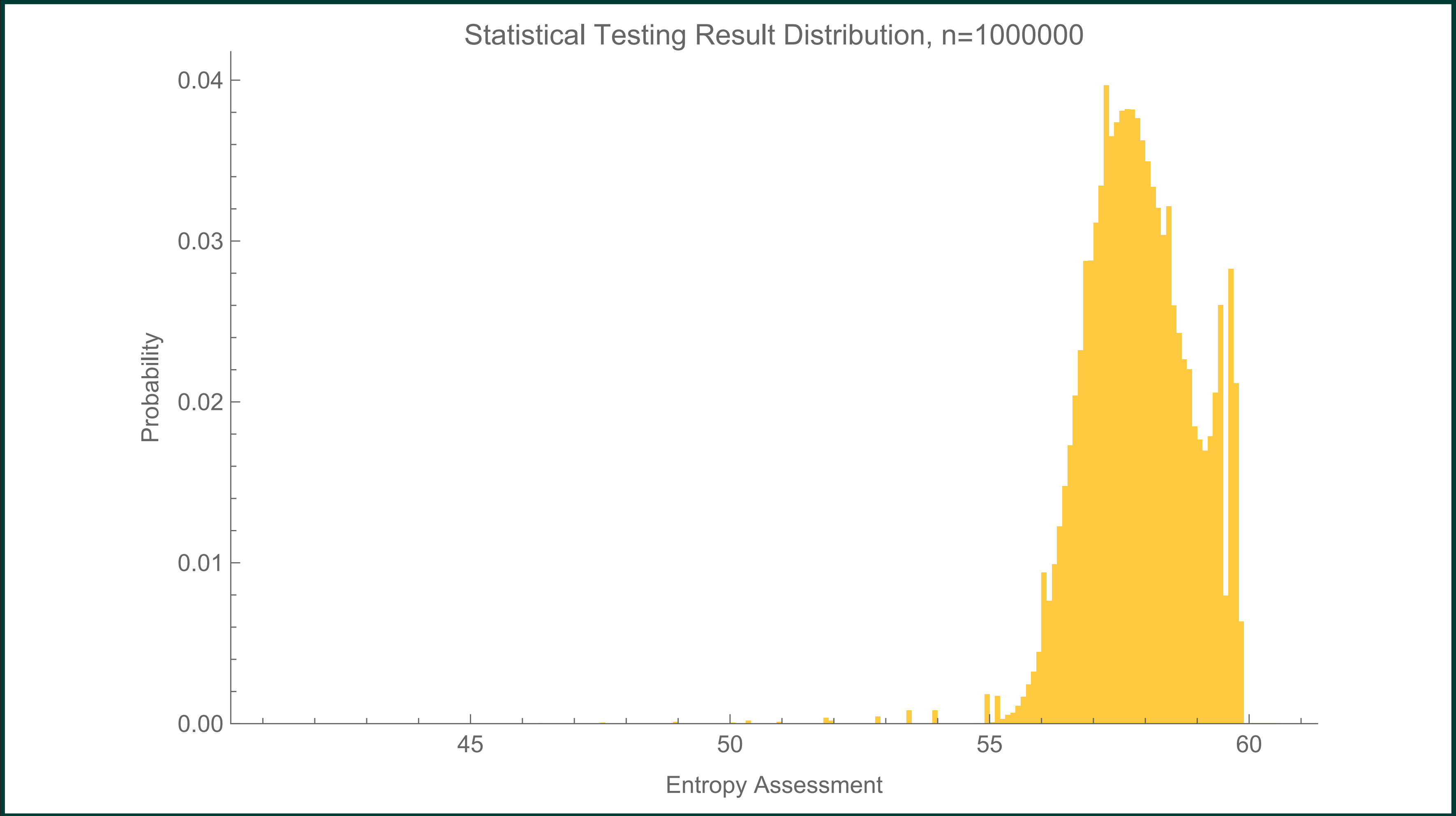
- Output_Entropy(·) Results:



# Part 3.1: SP 800-90B Non-Vetted Conditioning Analysis

- The LFSR’s output looks pseudorandom.
- Evaluation of pseudorandom data provides a practical “best case” for the $h' \times n_{\text{out}}$ term.
- This “best case” is essentially attained by any conditioner whose output is pseudorandom (which is most conditioners).
- The ESV program allows for submission of up to 1 million-byte samples (thus 8 million bits).
- With this size and type of data, this estimation approach yields a consistent distribution.

# Part 3.1: $h' \times n_{\text{out}}$ for 64-Bit Blocks of Random Data




TERON LABS

# Part 3.1: $h' \times n_{\text{out}}$ for 64-Bit Blocks of Random Data

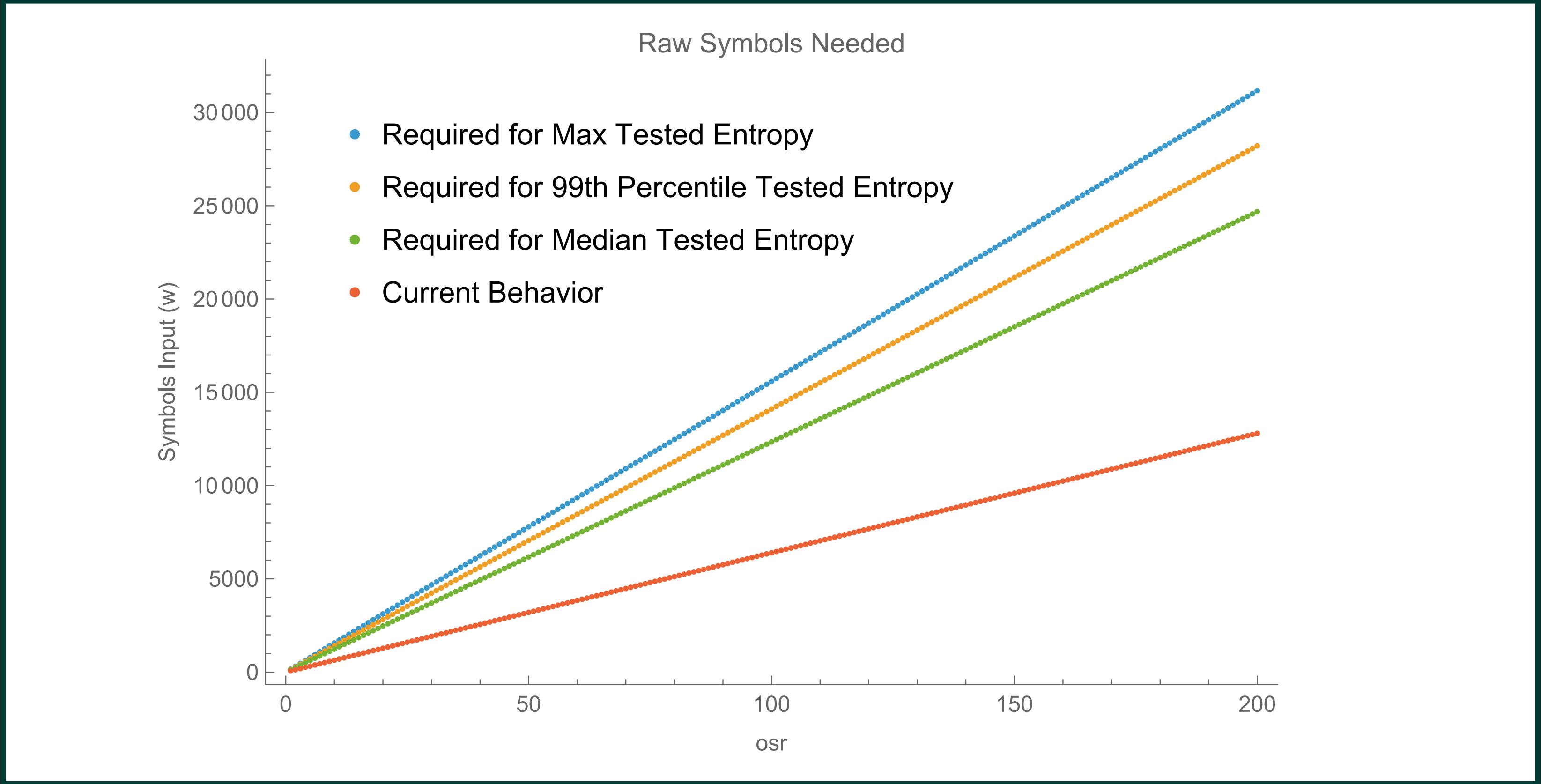
| Percentile (%) | Value |
|---|---|
| $\approx 0$ (Min) | 40.8494 |
| 50 (Median) | 57.8352 |
| 99 | 59.7484 |
| $\approx 100$ (Max) | 60.8971 |

| Probability the Result is in the Bound (%) | Result Interval |
|---|---|
| 95 | [56.0599, 59.7484] |
| 99 | [54.9680, 59.8852] |
| 99.9 | [50.3425, 59.8852] |

TERON LABS

# Part 3.2: Alternate values for $w$

- Now that we understand the distribution for $h' \times n_{\text{out}}$, we can set $w$ to alter the chance that $h_{\text{out}}$ is established by the heuristic estimate (which only happens when $h' \times n_{\text{out}} > h_{\text{heuristic}}$).



# Part 3.2: SP 800-90B Non-Vetted Conditioning Analysis

| Approach | Chance of $h_{\text{heuristic}}$ Reducing $h_{\text{out}}$ | Scaling Factor ($\mathcal{S}$) | $w = \lceil \mathcal{S} \times 64 \rceil \times osr$ | Average $h_{\text{out}}$ | Normalized Efficiency ($E_{\text{normalized}}$) |
|---|---|---|---|---|---|
| Original Behavior | $\approx 100\%$ | 1 | $w = 64 \times \text{osr}$ | 42.3053 | 1 |
| $h_{\text{heuristic}}$ is likely above $h' \times n_{\text{out}}$ | $\leq 50\%$ | 1.96875 | $w = 126 \times \text{osr}$ | 57.4610 | 0.689904 |
| $h_{\text{heuristic}}$ is very likely above $h' \times n_{\text{out}}$ | $\leq 1\%$ | 2.25000 | $w = 144 \times \text{osr}$ | 57.8775 | 0.608041 |
| $h_{\text{heuristic}}$ is always above $h' \times n_{\text{out}}$ | $\approx 0\%$ | 2.48438 | $w = 159 \times \text{osr}$ | 57.8780 | 0.550683 |

TERON LABS

# Part 3.2: Comments on Efficiency

- The best rate of entropy per unit time is attained by setting $w$ as small as possible.
- Higher claims (min entropy per 64 bit block of conditioned data) are possible, but they decrease the amount of entropy per unit time.

TERON LABS

# Wait… What Were We **Just** Talking About?

We…

- Presented a linear model for this conditioning.
  - This was just a mathy way of conceptualizing the conditioning.
- Presented a heuristic analysis using this model
- Provided a broadly-applicable distribution of statistical entropy assessment results that applies to most non-vetted conditioning functions.
- Described changes to JEnt v2.2.0 that allow for higher conditioned output block min entropy claims (but at WHAT COST? AT WHAT COST?!?)

TERON LABS

# References


- IG] CMVP. *Implementation Guidance for FIPS 140-3 and the Cryptographic Module Validation Program*. NIST, September 2, 2025. https://csrc.nist.gov/csrc/media/Projects/cryptographic-module-validation-program/documents/fips%20140-3/FIPS%20140-3%20IG.pdf.
- [NIST SHALL] NIST CAVP. *90B-Shall-Statements*. NIST, August 9, 2021. https://csrc.nist.gov/CSRC/media/Projects/cryptographic-module-validation-program/documents/esv/90B%20Shall%20Statements.xlsx.
- [HC 2025] Joshua E. Hill and Yvonne Cliff. *JEnt v2.2.0 LFSR Conditioning Analysis*. TRD9.
- [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*. NIST, January 2018. https://doi.org/10.6028/NIST.SP.800-90B.


TERON LABS